



IUL School of Technology and Architecture
Department of Information Science and Technology

Light Field Processor: A Lytro Illum imaging application

Agostinho Silva

A Dissertation presented in partial fulfilment of the Requirements for the Degree of
Master in Computer Science Engineering, Multimedia Specialisation

Supervisor:
PhD Paulo Nunes, Assistant Professor
ISCTE-IUL

November, 2016

Acknowledgments

I would like to thank my dear friend
Paulo Tavares de Almeida
for his continuous support, without which this work would not have been possible.

I would also like to thank the expert comments and tips given by
Maria Henriqueta Tavares de Almeida
João Luis Pimentel Nunes
Marco Gaspar

who contributed to the enrichment of this work.

Finally, a special thanks to
Professor Paulo Nunes
for his humble and simple nature, which made me feel comfortable when
interacting with him and certainly made a lot of things, a lot easier.

Dedicated to the memory of my father
Augusto José da Silva Rêgo
who abandoned this world in July 2015,
just when I was about to start the development stage of this work.

Light Field Processor was partially developed under a scholarship
of the 3D-SERVICIS project granted by
Instituto de Telecomunicações (IT) – Lisbon, Portugal

The author acknowledges the support of
Fundação para a Ciência e Tecnologia,
under the project UID/EEA/50008/2013

Resumo

A tecnologia de imagem de campo de luz está na intersecção de três grandes áreas de investigação: gráficos por computador, fotografia computacional e visão por computador. Esta tecnologia tem o potencial de possibilitar funcionalidades que eram anteriormente impraticáveis, senão mesmo impossíveis, tais como refocar imagens fotográficas após a captura ou movimentar-se numa cena de RV produzida por um motor de jogos em tempo real, com 6 graus de liberdade.

A fotografia tradicional produz uma única saída sempre que um utilizador prime o botão de disparo. A fotografia campo de luz pode ter várias saídas diferentes porque junta muito mais dados acerca da cena. Logo ela requer pós-processamento por forma a extrair qualquer informação útil, como imagens 2D, e essa é uma funcionalidade caracterizante desta tecnologia que a torna substancialmente diferente de todas as outras no ramo da produção de imagem.

Pós-processamento significa usar uma aplicação especializada e, uma vez que esta tecnologia ainda está na sua infância, essas aplicações são escassas. Este contexto proporcionou uma boa oportunidade para tal desenvolvimento.

Light Field Processor é o principal resultado deste trabalho. É uma aplicação para computador capaz de abrir e decodificar imagens de cameras Lytro Illum campo de luz, que podem então ser armazenadas como um novo formato de ficheiro (Campo de Luz Descodificado), proposto nesta dissertação, para uso posterior. É capaz de extrair pontos de vista 2D, mapas 2D de pontos de vista ou conjunto de microlentes, vídeos mostrando a paralaxe intrínseca do campo de luz e metadados, assim como fazer algum processamento de imagem básico.

Palavras-chave: Campo de Luz, Processamento de Imagem, Fotografia Computacional, Lytro Illum.

Abstract

Light field imaging technology is at the intersection of three main research areas: computer graphics, computational photography and computer vision. This technology has the potential to allow functionalities that were previously impracticable, if not impossible, like refocusing photographic images after the capture or moving around in a VR scene produced by a real-time game engine, with 6DoF.

Traditional photography produces one single output whenever a user presses the shot button. Light field photography may have several different outputs because it collects much more data about a scene. Thus it requires post-processing in order to extract any piece of useful information, like 2D images, and that is a characteristic feature that makes this technology substantially different from all others in the field of image making.

Post processing means using a specialised application and, since this technology is still in its infancy, those applications are scarce. This context presented a good opportunity for such a development.

Light Field Processor is the main outcome of this work. It is a computer application able to open and decode images from Lytro Illum light field cameras, which it may then store as a new file format (Decoded Light Field), proposed in this dissertation, for later use. It is able to extract 2D viewpoints, 2D maps of viewpoints or the microlens array, videos showing the intrinsic parallax of the light field and metadata, as well as do some basic image processing.

Keywords: Light Field, Image Processing, Computational Photography, Lytro Illum.

DISSERTATION

CONTENTS

1. INTRODUCTION	1
1.1. Context and Motivation.....	1
1.2. Goals	3
1.2.1. Application requirements and features.....	3
1.2.2. Target users	3
1.3. Methodologies.....	4
1.3.1. Application development	4
1.3.2. Dissertation.....	5
1.3.3. This document's readability	6
1.4. Main Contributions of This Work	7
1.4.1. Application features perspective.....	7
1.4.2. Application development perspective.....	7
1.5. Quick Overview of Contents	8
2. HOW PHOTOGRAPHY WAS BORN	9
2.1. The Birth of a New Technology	10
2.1.1. The term	10
2.1.2. How photography came into existence.....	11
2.2. The First Photocopy Office	12
2.3. Adding Colour to Images	13

2.3.1. Maxwell: Validation of the trichromatic theory	13
2.3.2. Hauron: First implementation of the three-colour method.....	15
2.3.3. Lippmann: A different approach to colour	16
2.3.4. Ives: Avoiding printed paper	16
2.3.5. And the winners are: the Leopold musicians!	17
2.4. The Modern Era	18
2.4.1. Developments leading to light field photography	19
2.4.2. Current light field cameras	19
3. LIGHT FIELD IMAGING CONCEPTS	21
3.1. Human Vision	22
3.1.1. Binocular vision	22
3.1.2. Stereoscopy	22
3.1.3. Binocular disparity and stereopsis.....	22
3.1.4. Sensing light.....	23
3.1.5. Interpreting light	23
3.2. Light Field Theory Overview	24
3.2.1. The plenoptic function	24
3.2.2. Radiance.....	25
3.2.3. Light field	25
3.3. Light Field Image Formation	25
3.3.1. Brief overview of a light field camera.....	25
3.3.2. Light field image	27
3.4. Light Field Image Processing	28
3.4.1. Decoding	29
3.4.2. Viewpoint.....	29
3.4.3. White image	30
3.4.4. Depth of field	30
3.4.5. Depth of focus.....	31
3.4.6. All-in-focus.....	31
3.4.7. Refocusing.....	31
3.4.8. Vignetting.....	31

4. LIGHT FIELD PROCESSOR	33
4.1. What is Light Field Processor?	34
4.1.1. LFR related outputs	35
4.1.2. DLF related outputs	35
4.2. LF Processor: Usability.....	36
4.2.1. Obtaining help inside the application	36
4.2.2. Application folders.....	37
4.2.3. Automatic naming scheme for exported files	38
4.2.4. Validation of user inputs	39
4.2.5. Running LF Processor for the first time	39
4.3. LF Processor: Graphical Interface	41
4.3.1. Main screen	41
4.3.2. Main workspace.....	42
4.3.3. LFR workspace.....	43
4.3.4. DLF workspace	43
4.4. LF Processor: Features	45
4.4.1. File menu	45
4.4.2. Process menu.....	46
4.4.3. Process menu: Decode.....	47
4.4.4. Process menu: Image (processing).....	48
4.4.5. Process menu: Depth Filters (engine).....	50
4.4.6. Export menu	54
4.4.7. View menu	56
4.4.8. Other operations.....	56
5. DEVELOPMENT OF LF PROCESSOR	57
5.1. Software Development Methodology	58
5.1.1. Requirements.....	58
5.1.2. Design, Implementation and Testing	60
5.1.3. Deployment	60
5.2. Application Architecture.....	61
5.2.1. External architecture.....	61

5.2.2. Internal architecture	62
5.3. Programming	67
5.3.1. DLF data structure	67
5.3.2. Paradigm adopted.....	68
5.3.3. How to save the DLF file's exact size on disk, within itself	69
5.4. Development Tools and Contents.....	70
5.4.1. Matlab: programming language and editor.....	70
5.4.2. Application dependencies: Light Field Toolbox	70
5.4.3. Photoshop: Image editing	70
5.4.4. Light field image datasets used	70
5.5. Extra Work That Was Not Used	71
6. CONCLUSIONS	73
6.1. Contributions of This Work	73
6.1.1. LF Processor	74
6.1.2. XSL template for automatic bibliographic sources	75
6.2. Future Work	76
BIBLIOGRAPHY	77
A. APPENDIX A: APPLICATION DEPLOYMENT	85
A.1. Application Versions	85
A.1.1. Dependent application	85
A.1.2. Standalone application	86
A.2. Installing LF Processor	86
A.2.1. As a dependent application	86
A.2.2. As a standalone application	87
A.2.3. Application Configuration	88

LIST OF

FIGURES

Figure 1 – Overview of the software development methodology used: Iterative mini-Waterfalls.	5
Figure 2 – Hercule Florence, around the age of 70 [25].....	10
Figure 3 – The oldest surviving photograph taken around 1826 by J. Niépce [30].	11
Figure 4 – Photocopy of a Diploma made by H. Florence around 1839 [25].	12
Figure 5 - Left: Colour triangle shown on Maxwell’s paper [34]. Right: A modern depiction of the colour triangle and colour disc envisioned by Maxwell using real colours [37].	14
Figure 6 – Depiction of three colour filtered plates that allowed the reproduction of a colour image for the first time, using the trichromatic process (adapted from [38]).....	14
Figure 7 – An 1877 photograph using Hauron’s method [39].....	15
Figure 8 – Colour photograph from the 1890s using the interference method [44].	16
Figure 9 – Vase of Flowers: A Kromogram of 1897 [46].....	17
Figure 10 – A Kodachrome picture by Chalmers Butterfield [45].	18
Figure 11 – Effects of horizontal disparity on binocular vision (original creation from parts of [4]).....	22
Figure 12 – Light rays reflected off an apple [4].....	23
Figure 13 – Parameterization of the 5D Plenoptic function [62].	24
Figure 14 – A light field represents the amount of energy across a tubular area [62].....	25
Figure 15 – Left: Conventional 2D camera (lens focuses directly onto image sensor). Right: Light field camera (lens generates intermediate image, which is then focused by the Microlens Array onto the image sensor) (adapted from [64] and [5]).....	26

Figure 16 – A light field camera captures angular information (adapted from [4]).	27
Figure 17 – Parameterization of a light field using two planes (adapted from [66]).	27
Figure 18 – An amplified slice of a raw light field image.	28
Figure 19 – A cut on a Views Map emphasising an amplified Viewpoint.	29
Figure 20 – Amplified sample of a White Image shown by LF Toolbox while decoding [13].	30
Figure 21 – Processing flow of the Light Field Processor application, emphasising its main functional purpose and outputs.	34
Figure 22 – Initial Configuration procedure - Step 1: Application Data Folders.	37
Figure 23 – Button Tooltips.	37
Figure 24 – Application folders structure showing the exported contents of the “Flowers” light field image.	38
Figure 25 – Crucial information before configuring LF Processor.	40
Figure 26 – Light Field Processor main screen, just after launch.	41
Figure 27 – Presentation of the Toolbar.	41
Figure 28 – Main screen and workspace after opening an LFR file.	42
Figure 29 – A Data Cursor tip.	43
Figure 30 – Main screen and workspace after opening a DLF file.	44
Figure 31 – The Viewpoint Navigator.	44
Figure 32 – Potentially problematic data shown in red.	45
Figure 33 – <i>Open Light Field</i> dialogue box and the <i>File</i> menu.	45
Figure 34 – Preferences: Image Processing options screen.	48
Figure 35 – Comparison of image processing tasks. Left: original DLF without any effects; Centre: applied CC (R=1.00, G=1.00, B=1.00); Right: applied CC+GC+HE+IB (all with default values).	49
Figure 36 – Example of the Improve Borders (IB) feature.	50
Figure 37 – <i>Depth Filters</i> submenu options.	51
Figure 38 – Preferences: Depth Filters options screen; (each operation’s initialism is highlighted with a red border).	51
Figure 39 – Depth Filters engine: running the <i>Shift Sum</i> filter (the front image is defocused as a direct consequence of the application of the filter).	52
Figure 40 – Depth Filters engine: generated folder structure and file naming scheme.	53
Figure 41 – Results of applying the <i>Shift Sum</i> filter to DLF file “Friends_1”, with a growth of +0.50 per iteration of the slope parameter. Left: [03] SS(slope=-0.50); Centre: [04] SS(slope=0.00); Right: [05] SS(slope=0.50).	54
Figure 42 – Preferences: File Input/Output options screen.	55
Figure 43 – Detailed software development methodology used: Iterative mini-Waterfalls.	58
Figure 44 – External architecture of LF Processor, using a simplified approach.	61
Figure 45 – Internal architecture of LF Processor.	62

Figure 46 – Folder structure of the project's files.	62
Figure 47 – Example of code from the Action Class, used mainly as a powerful data structure.	63
Figure 48 – Diagram of the application Configuration data structures (numbers are respective to the tab used in the Configuration screen).	64
Figure 49 – Application Configuration: Cameras Management screen.	64
Figure 50 – Diagram of the application Preferences data structure (numbers are respective to the tab used in the Preferences screen).	65
Figure 51 – Using GUIDE to visually design the DLF data panel.	66
Figure 52 – Workspace panel (left): Top level of the internal data structure of a DLF file. Command Window (right): contents of some DLF fields.	67
Figure 53 – Matlab's console showing a DLF file <i>ipTasks</i> variable and its contents.	68
Figure 54 – List of the 30 files used for Class definitions.	69
Figure 55 – Step 3 of Application Configuration: White Images Database update..	88

LIST OF

ACRONYMS

2D	Two-dimensional; or regarding two dimensions
3D	Three-dimensional; or regarding three dimensions
4D	Four-dimensional; or regarding four dimensions
5D	Five-dimensional; or regarding five dimensions
6D	Six-dimensional; or regarding six dimensions
6DoF	Six Degrees of Freedom (in the context of Virtual Reality)
CC	Colour Correction
CCM	Colour Correction Matrix
DLF	Decoded Light Field image
.DLF/.dlf	DLF file name extension; or regarding the data contained in such file
DoF	Depth of Field
GC	Gamma Correction
GUI	Graphical User Interface
HE	Histogram Equalisation (also <i>Histogram Equalization</i>)
HVS	Human Visual System
IB	Improve Borders
IDE	Integrated Development Environment
IT	Information Technology
LF	Light Field
LFR	Light Field Raw image: standard output file of a Lytro Illum camera
.LFR/.lfr	LFR file name extension; or regarding the data contained in such file
MLA	Microlens Array (also <i>Micro Lens Array</i>)
OOP	Object Oriented Programming

VR	Virtual Reality
WID	White Images Database

NOTE: The abbreviation **LF Processor** is used extensively throughout this work to refer to the *Light Field Processor* software application. We do not use the acronym *LFP* simply because it could be confused with the file type *LFP (Light Field Picture)* that Lytro defined as the raw light field image file type for its first generation cameras (F1) and also as a container file type for the Lytro Illum camera, which is the main focus of this work.

CHAPTER 1

INTRODUCTION

Writing an introduction is like filling a bottle of water through a funnel.

No matter how much content you have or how good it may be, it will always have to be small enough to fit in the bottle and flexible enough to pass through the funnel.

While funnelling my thoughts...

1.1. Context and Motivation

The trend towards automated systems (e.g. auto-driven vehicles) has led to the increased demand for ways to automatically interpret images and extract information like positioning and target detection, amongst other things.

3D reconstruction from 2D images has long been a well-known problem in the scientific community and new and better ways to achieve it have been developed. One of the main inputs required is the knowledge about the depth of objects in the scene. In traditional photography, a single (mono) image may be sufficient to infer depth but it requires more complex algorithms than the ones required by stereo methods using a pair of images [1].

However, this apparently simple requirement of extraction of depth information from the visual scene poses a problem when capturing scenes with camera or object motion, because all images must be taken at the exact same time, so that only the angle of view is changed and not the positioning of objects on the scene. Moreover, having more than two perspectives may provide even more reliable results and simpler algorithms but requires further data to process and more complex capture devices. Still, that seemed like the right path to go and some systems were created that allow the capture of such still images.

Stanford University, in the USA, has been pioneering the development of light field image capture systems. Two examples are the Multi-Camera Array [2] and the Spherical Gantry [3]. Unfortunately, those systems are not portable, so they cannot be easily carried around to capture nature scenes, for instance. Besides that, setting those systems up requires a lot of expertise and resources, which also makes them unaffordable for a common user, contrary to any conventional photographic camera [4].

A silent revolution began when a new approach was used to create an array, not of cameras, but of tiny lenses that could allow the capture of hundreds of tiny images at the same time, from the same scene, with slight changes of perspective. This is the architecture of current portable light field cameras from Raytrix [5] and Lytro [6], [7] which is setting the standards for this technology.

Light field imaging technology is at the intersection of three main research areas:

- Computer Graphics;
- Computational Photography;
- Computer Vision.

Implicitly, this also includes knowledge on a variety of other fields of study such as human vision, software development and geometry, to name just a few. Therefore, this is clearly a multi-disciplinary technology and that fact makes it difficult to understand at first. However, it has the potential to allow functionalities that would otherwise not be possible (or at least would be much more complicated to build), like refocusing photographic images after the capture (*post focusing*) [8], [7]; or seeing around 360° in virtual scenes using systems like “Moon”, a 6DoF VR system developed by Lytro [9] which promises a revolution in the field.

Perhaps the one single characteristic of this technology that makes it harder to absorb by a casual user is the fact that it requires post-processing in order to extract useful information like 2D images, and that means having to use a specialised software application. Since this technology is still in its infancy, these applications are scarce.

We interpreted this context as a good opportunity to develop such an application and this work is the outcome of such effort.

1.2. Goals

1.2.1. Application requirements and features

Since Lytro was the first company to launch a Light Field camera geared towards the consumer market, a few free software packages to process its images soon appeared. However, the launch of its second generation on July 2014, named Lytro Illum [10], improved and changed several aspects of how the technology was used.

The change in file format and its bit packing scheme (12-bit for version 1 and 10-bit for version 2) made the output from both cameras incompatible and, with it, all the software developed for the first generation. The image resolution and aspect ratio are also different. Such major changes made developers cautious about investing more resources on future developments which lead to most of that software [11] not being updated to the latest format. Some of the few free packages that were updated included *Lytro Desktop*, which is the proprietary software package that Lytro developed to process the light field images from its cameras [12] and a library of functions written for Matlab, called Light Field Toolbox, developed by Donald Dansereau [13]. The latter, although having some very useful functions, requires the user to have some technical expertise and programming skills. It also requires that resources and workflow be managed manually by the user because it is not an application and it does not have a Graphical User Interface (GUI).

Therefore, we identified an opportunity and momentum to:

- Develop a free application with a GUI that could automate some tasks in the light field image processing workflow (e.g. cameras management);
- Integrate features that could be useful to both technicians and end users;
- Move the focus towards the scientific side of the technology, which is more appealing to researchers and students of this technology.

Those tasks became the main goals of this work and that led to the design and implementation of the Light Field Processor (LF Processor) software application.

1.2.2. Target users

The target group of users for the LF Processor software application is primarily constituted by researchers and students interested in light field photography and related technologies, but it may also include any user of a Lytro Illum light field camera. The dissertation may be of interest to even more user profiles, such as project managers and software architects or developers, due to *Chapter 5* which is more oriented towards the IT field.

In practice this may encompass people from a wide range of areas of expertise, which represented a huge challenge in the development of this work and required special caution in the way to interact with and convey information to this target

audience, through the application and the dissertation respectively, as will be explained throughout this document whenever needed.

1.3. Methodologies

1.3.1. Application development

Considering that some of the target audience of this work may not have had formal training in software engineering and its theory, a few technical remarks are due for the sake of clarity.

It is important to notice that a software application should not be misinterpreted as an algorithm or a process; usually, at least in terms of scope, an application is much more than that. It ought to be understood as a whole functional unit, which may be constituted by many modules or components, which are themselves built with algorithms used to process data towards achieving predetermined goals.

An *application* is driven by the user, whilst a *process* is driven by the developer who programmed it, which implies that only the latter may be expressed in terms of a work flow diagram, simply because user actions, such as deciding which process to execute through menu options, are made in application running time, not in development time when any diagram is built.

The author of this work follows the definition of **Software Architecture** presented by Bass, Clements and Kazman in [14] and hereby reproduced: “*The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. Architecture is concerned with the public side of interfaces; private details of elements—details having to do solely with internal implementation—are not architectural.*”

This definition and others may also be found on the online *Microsoft Application Architecture Guide* [15] along with other useful information regarding this matter.

This project was developed within the context of a Masters dissertation and thus individually, in clear contrast with a full-blown corporate application. Therefore, due to its nature, most of the techniques for enterprise development were not adequate (e.g. group collaboration), so we opted for a downscaled and simplified model of *Iterative and Incremental Development* ([16], [17]) that best fits the modest requirements of this work. Actually, our model could be even better described as an *Iterative mini-Waterfalls* [16] as shown in Figure 1.

Preparation for this work started with learning the fundamentals about light field image processing and computational photography (for a brief overview on the matter, see *Chapter 3*). It was followed by a brief study of the current existing tools and applications for light field processing of Lytro cameras. This enabled the

definition of *Requirements*, which included what type of application and which main features to develop.

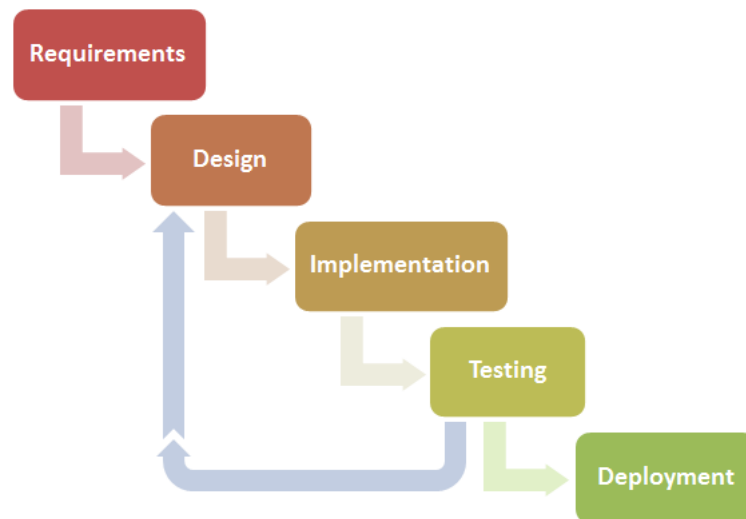


Figure 1 – Overview of the software development methodology used: Iterative mini-Waterfalls.

We then initiated the iterative process of *Design*, *Implementation* and *Testing*, which started with a simple prototype developed under a Rapid Development method using the standard scripting language of Matlab and its default Graphical Interface architecture. This initial first stage was very important because it immediately underpinned some limitations of this development system.

Consequently a few critical decisions, such as changing to an Object Oriented Paradigm (OOP), were made sooner than if we had used the more “classic” non-iterative version of the *Waterfalls* model, which saved time, in the long run.

For details regarding the iterative part of the development process, see *Chapter 5*.

After finishing the development, final testing and deployment were performed on Windows 7, 8 and 10, using the two latest versions of Matlab (R2015b and R2016a).

1.3.2. Dissertation

There are many ways to tell a story or explain an idea. The method used to present the Light Field Processor application, through this very document, is based on my own formal training on pedagogical techniques and professional experience as both a trainer and a teacher for many years.

The ***Association principle*** is a neuroscience theory that explains how the human brain learns through the association of ideas and lived experiences [18], [19]. Practice confirms that people learn more easily when they associate new knowledge with previous knowledge or with other experiences of the senses, which happen naturally when practicing the new knowledge.

Considering the wide diversity of expertise of our targeted group of users (as mentioned in *Section 1.2.2 above*) a *Top-Down* approach will be used, starting with less technical subjects, to reach a wider audience (*Top*), and then gradually progressing to the more technologically challenging information on the later chapters, oriented towards a more specialised group of people (*Down*).

By merging the Top-Down approach and the Association principle, it makes sense to start presenting the Light Field Processor application by its main features and how they can be reached through the graphical user interface; first in a broader way, then in practical terms.

As this process of learning unfolds, a user, whether a light field photographer, a multimedia student, a researcher on light field technologies or a software developer specialising in digital image processing, will gradually and effortlessly become aware of what a light field image is and how its data may be manipulated by experimenting the various outputs provided by the application.

If a reader of this work has some technical background on software engineering, or pursues it, and wants to get a more technically oriented perspective regarding the development of the application, then it will be easier to understand the diagrams shown, as well as how and why some technical decisions were made and what was their purpose, when reading the subsequent chapters (in particular, *Chapter 5*).

1.3.3. This document's readability

The elaboration of this document followed the rules of:

- British English spelling;
- The IEEE Editorial Style Manual for bibliographic sources, available at [20].

This document was produced with Microsoft Word 2010, which unfortunately does not include an updated template for the style rules imposed by IEEE in [20]. In order to comply with these rules and still have an automated mechanism of managing bibliographic sources within Microsoft Word, a new XSL template had to be developed. For that task we used BibWord [21] as a model. In the process we also took the opportunity to add some rules regarding online sources (that were not defined by IEEE) and create a consistent and hopefully more visually appealing formatting scheme.

Since this document will only be publicly distributed in static text formats (printed on paper or digital PDF file), there is no need to go into details on how to install the bibliography template.

1.4. Main Contributions of This Work

On a broader perspective, the main contribution of this work is obviously the development of the Light Field Processor computer application. However, such a project comprises different modules and features which require varied types of knowledge and skills. Therefore, dividing this global contribution into its constituent parts can certainly help in better understand its merits.

Nevertheless, in this introductory chapter we will do just an overview of some of those contributions. For details about them, see *Chapter 4* and *Chapter 5*.

1.4.1. Application features perspective

Light Field Processor is a software application built to run on a Windows PC. It has a Graphical User Interface that aims to integrate its features into a whole functional unit, that:

- Automates repetitive tasks;
- Performs validation of data and inputs up to a certain level;
- Provides interactive information;
- Allows visualisation of a light field image in innovative ways (e.g. Viewpoint Navigator);
- Does basic image processing (e.g. Colour Correction);
- Adds an innovative way of systematically running and testing algorithms (e.g. Depth Filters engine);
- Proposes the creation of a new file format (DLF) for storing decoded light field data;
- Keeps user preferences and configuration options between different sessions.

1.4.2. Application development perspective

Matlab's original purpose was essentially that of a powerful, practical and scalable calculator machine. Although currently it is much more than that, trying to develop a full graphical computer application with such a tool was certainly a very risky decision. Even more so when we consider that it was also developed using the new dot notation architecture (which only became available in Matlab R2014b) and was still unknown by most of its users during this work's development. This notation is perhaps the most visible part of an attempt to construct a true Object Oriented Programming (OOP) paradigm, with classes and objects, in order to update Matlab's development engine to modern techniques, standards and industry requirements.

It now seems obvious, after the development is finished, that building this application was only possible thanks to the use of OOP techniques which allowed the scalability and flexibility that such an application development required, especially in such a relatively short period of time.

As is often the case in scenarios like this, innovative solutions had to be developed to face the many yet unsolved problems faced with this new implementation. However, ultimately it was Matlab's graphical interface engine that proved to be the most challenging for the developer. Details about this and other software developments will be discussed in *Chapter 5*.

1.5. Quick Overview of Contents

This dissertation is structured around three main parts (as explained in the methodology section):

1. Introduction to the work and its technologies (chapters 1-3);
2. Detailed presentation of the application (chapters 4-5);
3. Conclusions about the work: main contributions and suggestions for future work (chapter 6).

Part one is divided in three chapters: the first, and current chapter, does the formal introduction to this work; the second chapter goes through a brief history of photography because, amongst other reasons, knowing the past helps in predicting the future; the third chapter explains the most relevant concepts required to understand the remaining work.

The second part presents the software application from two different perspectives: (1) the user perspective or light field image researcher perspective, which encompasses the interface and all the application's features; (2) the software developer perspective, which comprises aspects such as development methodologies used, application architecture and data structures implemented.

Finally, the third part closes this work, including some of its major contributions and suggestions for future work.

CHAPTER 2

HOW PHOTOGRAPHY WAS BORN

"And you, Divine sun, lend me your rays."

[Originally written in French, on a glass surface prepared with silver nitrate, this phrase actually shone when exposed to the sun!]

Hercule Florence, 1833

Humankind has gone a long way since the first forms of rock art, some dating back around 30000 years. Since then there has been a trend to make depictions ever more realistic.

On ancient civilisations, like the Greeks and the Egyptians, engravings in pottery and home walls reflected daily life on flat one-sided views.

Many centuries later, in the Western culture, the importance of images grew throughout the medieval period. First as a means to celebrate and "imprint" historic events in people's minds (like the "Bayeux Tapestry" did for the conquest of England by the Normans [22]). Secondly, with the spread of religion, as a solution for the need to convey spiritual messages of things that people would otherwise only be able to imagine (e.g. demons). On a time when about 90% of the population in Europe was illiterate, images were much more efficient than words

to convey those ideas. This trend led to the wonderful works of art of the Renaissance and the importance of ever more realistic depictions.

2.1. The Birth of a New Technology

A word of caution is required before we start this brief journey into history. The evolution of photography had many contributors and sometimes simultaneous discoveries, some of them only recently became known. Therefore it is not easy to establish a definitive and clear chronological timeline of all great achievements. This brief study was conducted using several different sources, which are all shown. Although it may seem excessive, it is only a means to provide the tools so that a more inspective reader may confirm our findings. Having that into consideration we start by the beginning: how the term itself appeared.

2.1.1. The term

The word **photograph** had more than one independent author because it was a natural choice for anyone who knew Greek and had the knowledge about the technology. It is formed by the combination of the Greek words *photo* (light [23]) and *grapho* (to write, to draw up in writing [23], [24]) and it literally means “light drawing” or “drawing with light”.



Figure 2 – Hercule Florence, around the age of 70 [25].

Only relatively recently (1976) the world learned that the term was first used by a French-Brazilian painter and inventor, called **Antoine Hercule Romuald Florence** (Figure 2), who went to Brazil in 1824, with just 20 years of age. There he developed a technique to capture images which he used for the first time in 1832. We know that because he recorded it, in 1834, as “*photographie*” (French word for photography) in the set of notebooks he kept about his experiments (covered in *Section 2.2*) [26], [25], [27], [28], [29, p. 27].

John Herschel, an Englishman considered to be the first photo chemist [29, p. 79], made several contributions to the early development of photography and he used the terms *photograph* and *photography* in England, a few years later, in 1839, to describe both the product and the process [29, p. 27]. Curiously, because he was already a reputed scientist in the field, he is usually credited as being the one who coined the terms. Although he certainly did a lot to the establishment of those terms, the records of Antoine Florence provide proof that he, in fact, was the one who first used that term and should get proper credit for it.

2.1.2. How photography came into existence

By the XIX century, the optics required to produce photographic cameras already existed, but it was being used only for the production of telescopes because a method to capture a permanent image on a solid surface was not yet discovered.

Around 1816, **Joseph Nicéphore Niépce**, a Frenchman who had been involved with etching and lithography, discovered a method to temporarily fix negative images on paper. Not satisfied with the results though, he continued experimenting with other materials and by 1822 he was able to permanently capture an image of an engraving of Pope Pius VII, in what is thought to be the first ever photographic image [30].

He called his technique ***heliography*** (Greek for “sun drawing”) and the plates in which the images were captured permanently he called *heliographs*. This process would serve as the conceptual model for the photoengraving industry. Using a *camera obscura* and applying his method, between 1826 and 1827 he produced the oldest surviving photograph of nature entitled “View from the Window at Le Gras”. A 1952 reproduction of that photograph is shown in Figure 3.

From the window of Niépce’s working room in Nice, photography would become a window to the World. For the discovery of this method Joseph Nicéphore Niépce is usually considered the inventor of photography [30], [29, pp. 27-28].



Figure 3 – The oldest surviving photograph taken around 1826 by J. Niépce [30].

In 1826, **Louis Daguerre**, another inventor who was also pursuing a method to fix images obtained from light, contacted Niépce to propose a partnership. The two inventors started working towards a common goal in 1829, by improving on the heliograph, but Niépce died in 1833 leaving his method unpublished. His son, Isadore Niépce, would continue the work with Daguerre. Two years later, in 1835, Daguerre is credited to have discovered a method that shortened the exposure from hours to just around 20 minutes and still allowed an image to be stabilized, in what he called a *daguerreotype* [29, p. 28].

In September 1839 the French government announced the process to the world after agreeing to offer both inventors a pension for the technology behind it. The Daguerreotype became an instant success and led to the birth of what we know today as the photographic industry [29, p. 28].

It is important to mention that there were other inventors who made important contributions to the development of photographic techniques. **William Talbot**, an Englishman, discovered a method to create engravings, although with an inferior quality to the Daguerreotype. However, in 1841 he developed a process, called *calotype*, which yielded good quality and required only a fraction of the time needed to produce a printed image on paper, but it was technically demanding and the license he imposed was very expensive, which confined its appeal only to the educated upper class, so it never became popular [29, pp. 28-30].

2.2. The First Photocopy Office

Another important pioneer in the development of photography was **Hercule Florence**, as mentioned earlier. The recorded details of his technique were confirmed to be valid, in 1976, by the Rochester Institute of Technology [31, p. 18].



Figure 4 – Photocopy of a Diploma made by H. Florence around 1839 [25].

His technique would yield superior results to the one developed later by Daguerre based on Niépce's work, which would become the model worldwide. However, being isolated from the centre of technological development in Europe, his

discoveries went unknown outside of Brazil for more than a century [26], [25], [27], [28], [29, p. 27].

As an illustrator, painter and calligrapher, he was initially motivated by the need to reproduce his own documents as easily and as faithfully as possible. For that he created his own printing office, in the city of Campinas, Brazil, where he immediately started using his copying techniques commercially, as is shown by a photocopy of a diploma made by him around 1839 (Figure 4).

Since no other discovery of this kind is known earlier than this, it seems therefore fair to say that Florence was the first person to create a photocopy of a document and to use such technique commercially, which would constitute an industry in itself, in parallel with that of photography.

2.3. Adding Colour to Images

2.3.1. Maxwell: Validation of the trichromatic theory

The appearance of the theory of trichromatic colour vision [32, pp. 207-211] by **Thomas Young** in 1801 [33], which was further developed by **Hermann von Helmholtz** in 1850, motivated **James Clerk Maxwell** to prove it by using the branch of mathematics called Linear Algebra that had been recently developed. In his classic paper of 1855 [34], concerning experiments on colour, he demonstrated that a monochromatic light affecting three receptors should be able to be equally stimulated by a set of three different monochromatic lights.

He had a colour disc made especially to allow a visual demonstration of the results and conceived a colour triangle using his three primary colours (red, green and blue) which he claimed would allow anyone to determine, with mathematical precision, which amount of any of the primary colours was needed to synthesize a specific colour. It is important to notice though that colour triangles were already widely known after **Tobias Mayer**, a German astronomer and mapmaker [35], created one in 1758 to visually show the different gradients of colour that humans were able to distinguish.

Maxwell didn't actually create a colour triangle, contrary to what some sources say [36]. He did however create a new method to use one with mathematical accuracy which was very important at a time when most theories derived from direct observation or experimentation and not by means of mathematical calculation. His principles would later be used to create colour spaces and laid the foundations for a new branch of science that researches the quantitative description of colour mixture and is used to quantify and describe human colour perception, nowadays known as *colourimetry*.

Figure 5 shows the original triangle envisioned by Maxwell and its modern implementation with real colours, including some example intensity values of the

three basic colours used to prepare the colour disc depicted to show that specific colour.

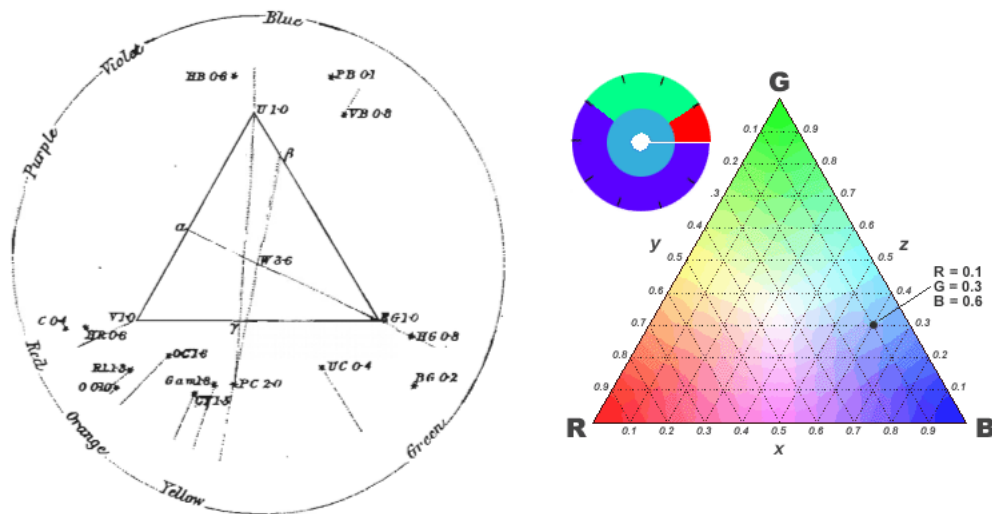


Figure 5 - Left: Colour triangle shown on Maxwell's paper [34]. Right: A modern depiction of the colour triangle and colour disc envisioned by Maxwell using real colours [37].

In 1861, James Maxwell was invited to the Royal Institution in London to give a lecture on colour vision. By that time **Thomas Sutton** was perhaps the most prominent inventor in the field of photography in England, having created the first single lens reflex camera that very same year. It was therefore no surprise that Maxwell asked Sutton for help. The latter created three negatives of the same picture (a ribbon) but through three colour filters: red, green and blue. These were then used to create lantern slides which Maxwell used to make a projection of the images, as shown in Figure 6.

When the first slide was projected the image was seen on red, but when all three monochromatic images were projected overlapped at the same spot, a colour image of the ribbon was seen. Although by modern standards the virtual image certainly lacked in the spectrum of “reds” and “greens” (we now know that the materials used were not sensitive to red light and were only marginally sensitive to green), the projection was convincing enough for a XIX century audience unprepared to evaluate something that had never been attempted before.

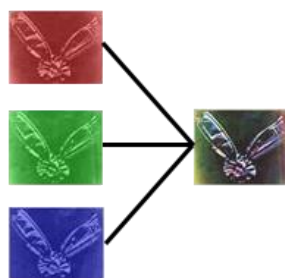


Figure 6 – Depiction of three colour filtered plates that allowed the reproduction of a colour image for the first time, using the trichromatic process (adapted from [38]).

Some sources, like Wikipedia [39] and BBC (in both news [40] and video documentary [36]), affirm that this is the first photograph ever taken. Perhaps some caution before making such a bold statement would have helped understand that what was captured were three independent monochromatic photographs of the same subject (a colour ribbon) that were later projected overlapped at the same spot on a wall or blackboard. No single plate captured all the colours, and so no permanent colour image was captured with this process, and therefore such a statement cannot be considered to be true. Moreover, if some sort of image projection were to be considered a photograph then one might as well consider any projection of a camera obscura, invented around 1000 years ago, to be a colour photograph too.

Camera Obscura

In 1020 AD, Ibn al-Haytham, an Arab physicist, published a Book of Optics in which he described a pinhole camera and a camera obscura.

2.3.2. Hauron: First implementation of the three-colour method

In 1868 **Louis Ducos du Hauron** patented a process that allowed three-colour prints on paper using the *subtractive method*, which instead of adding intensities of red, green and blue colours to black (additive colour system), subtracts their respective opposite colours: cyan, magenta and yellow, from white. For a detailed explanation of colour systems please consult [41] at *The American WideScreen Museum* Web site.

The “negatives” he developed used materials that were a bit more sensitive to the red and green spectrums of light than the ones used by Maxwell and Sutton seven years before, but were still incapable of capturing a satisfyingly range of intensities of these colours and it also required long exposure times.

Although this technology was far from yielding the best results, to the best of my knowledge, it was the first to produce a true colour printed photograph (not just a projected combination of three monochromatic images, as was the case of the Maxwell/Sutton solution). In Figure 7 the three coloured layers are clearly visible.



Figure 7 – An 1877 photograph using Hauron’s method [39].

2.3.3. Lippmann: A different approach to colour

Gabriel Lippmann was a physicist and inventor who played a fundamental role in the development of light field photography. But before that, he would become worldwide known for proposing a method to solve the problem of colour photography that was completely different from the previous ones.



Figure 8 – Colour photograph from the 1890s using the interference method [44].

Around 1886 he started searching for a method of fixing colour on a photographic plate and by 1891 he announced a solution by using a phenomenon called *interference* which physics explains as the outcome of the collision and resulting interference of two waves and applies to almost any kind of wave, be it radio, light or surface water.

In 1892 he produced the first photographs using his method: one is shown in Figure 8. This invention would later earn him the Nobel Prize in Physics for 1908. His Nobel lecture is available online and explains the details of the process [42], [43], [44].

However the search for the ideal colour method did not end here because the interference process was too complex, had problems with reflections of waves and required the use of mercury which is highly toxic, so it was never adopted by the industry [45], [39].

2.3.4. Ives: Avoiding printed paper

Although Maxwell proved the trichromatic theory to be valid and useful for the purpose of showing colour images, the photographic emulsions were only sensitive to ultraviolet, violet and blue wavelengths, which only allowed what were essentially monochromatic images. Therefore, further development in colour photography required a development in chemistry and that took many years of new experiments and improvements.

Meanwhile, a way to avoid this problem was to not use paper to print the images. **Frederic Eugene Ives**, a North American photography pioneer, did just that with his “Kromskop” system which was commercially available in England in late 1897 [45].

Using the previous work of Maxwell, he created a system in which three black and white “positive” images were taken through special red, green and blue filters. Then, instead of printing those images on paper, they were printed on transparencies (called *Kromograms* — an example is shown in Figure 9) which would then be viewed on a specific device with special colour filters: the ***Kromskop Viewer***. Such a device might nowadays be recognised as a fusion

between a microscope and a slide projector. There were monocular and stereoscopic versions, the latter being the most popular for allowing visualisations in full colour and 3D, through its pair of lenses, which at the time certainly must have stunned viewers.



Figure 9 – Vase of Flowers: A Kromogram of 1897 [46].

Although it delivered very good results, the fact that it required special equipment and three pictures to be made of each object with extreme accuracy made it unattractive for most people and it was not a commercial success being discontinued a decade later [45], [46].

2.3.5. And the winners are: the Leopold musicians!

In 1903, in France, the famous **Lumière brothers** patented the Autochrome which would become the major colour photographic process after it was commercially available in 1907. It was an additive colour process which used a glass plate coated with tiny grains of potato starch of red-orange, green and blue-violet colours that served as filters [47].

Nevertheless, the true hero of the colour processes arrived in the 1920s, through the unexpected discoveries of two American musicians: **Leopold Godowsky Jr.** and **Leopold Mannes**. They started experimenting with colour in 1917 after watching a movie that was supposed to be in colour but failed their expectations. They refined their process during the 1920s with the backing of an investor.

In 1930 the system impressed Eastman Kodak so much that both men were hired to work in the Eastman Kodak laboratories. The outcome was launched in 1935. Named Kodachrome, being commercially viable and backed by the marketing power of a company like Kodak, it would become the base for all further developments in this field.

As seen in Figure 10, the Kodachrome yielded very good quality colours thanks to a subtractive colour film coated with three layers each of them only sensitive to one third of the visible colour spectrum, usually represented by the colours: red, green and blue. Other chemical processes were required to extract the final slides, but Kodak made it simple for their target home cinema users, by simply allowing them to send in their negatives and Kodak would send them back the slides already processed. This greatly helped the system become a success commercially.



Figure 10 – A Kodachrome picture by Chalmers Butterfield [45].

Further developments followed. An important one was done the next year by Agfa, which found a way to develop the three colour layers at the same time and thus greatly simplify the process.

In 1942 Kodak launched the Kodacolor which was a negative film used specifically for still photography and allowed colour prints on paper. Together, these processes were adopted by the cinema and photography industries as a standard until the appearance of digital cameras in the 1990s [39], [45], [48], [49].

2.4. The Modern Era

Knowing about the past helps anyone to better preview the future, which is something important for fast-paced technological areas like digital photography and imaging. We think that task is already accomplished with what was mentioned so far in this chapter, and it is out of the scope of this work to proceed on further historical details.

However, some technologies that relate more closely to the ones used in this work, although much more recent, still need a few more words.

2.4.1. Developments leading to light field photography

Although not important for printed photography, stereoscopy played an important role in photography and image in the XIX century. The stereoscope was invented in 1838 by Charles Wheatstone and remained hugely popular throughout the century.

In 1846, Michael Faraday, a brilliant English experimentalist, was the first to propose that light should be interpreted as a field, just like magnetic fields on which he had been working for many years.

In 1908, Gabriel Lippmann introduced what he then called “*integral photography*”, to describe a plane array of closely spaced small lenses that is used to capture a scene, from several slightly different perspectives (horizontal and vertical). This principle was the basis to develop what is now known as light field photography, the key technology upon which this work was built.

2.4.2. Current light field cameras

Light field photography (sometimes also known as *plenoptic photography*) only became a reality thanks to the digital revolution and the integration of many of the previous major inventions such as colour, stereoscopy and autofocus. This was a principle that guided this work as well: *integration*.

It even expands technologies from *stereoscopy* (two views) to *multiscopy* (multiple views) and *pre-single-focus*, meaning one single focus chosen at the time of capture, to *post-multi-focus*, meaning selection of one single focus with control over depth of field or *all-in-focus*, after capture. See *Chapter 3* for definitions on some of these concepts.

Light field photography is not new as a concept, as mentioned above, but only relatively recently has been made a reality by companies like Raytrix (Germany) and Lytro (USA).

On July 2010 Raytrix launched its first models R5 and R11 geared towards the industrial and scientific markets [50], with effective resolutions of up to 1 and 3 megapixels respectively.

On October 2011 Lytro launched its first generation light field camera called F1, oriented towards the consumer market, with an effective resolution of roughly 1.1 megapixels [51].

In 2014, Lytro upgraded its cameras range to the second generation with the **Lytro Illum**, which served as reference to this work [10]. The camera offers a 40-megaray light field sensor, 8x optical zoom range and a constant f/2.0 aperture. For more technical details about this camera, see [52].

CHAPTER 3

LIGHT FIELD IMAGING CONCEPTS

*If we present a painting, we speak of Art.
If we present technology, we speak of Science.
Why should we then say "State of the Art" to present
the current development status of Technology or
Science?*

My enquiring mind

The target audience of this work is constituted mainly by students and researchers interested in light field photography and related technologies, which may encompass a wide range of areas of expertise, as stated before. So it seemed wiser to use a language that would be understandable by a wider audience and not just by scientists of the field. Besides that, some concepts require previous knowledge of other concepts.

In order not to make this a step by step mechanical reading experience, the approach used in this work is to first introduce the concepts in a more general way, and then, gradually, go into more technical details, as needed.

The concepts explained here are not supposed to be an exhaustive review of the technology but rather a practical and oriented overview geared towards providing a better understanding of the computer application developed within this work. To those who want to deepen their knowledge, several sources are provided.

3.1. Human Vision

3.1.1. Binocular vision

Some animals, in particular humans, are equipped with a pair of frontal eyes which are aligned horizontally. That configuration allows the eyes to capture images from two slightly different viewing angles simultaneously. The great advantage of having two such images is allowing ***stereopsis***, which is a very powerful process of depth perception. These are the defining features of binocular vision [32, pp. 44-51], [53].

3.1.2. Stereoscopy

Stereoscopy is a technique used to simulate the effects of binocular vision, especially depth perception, in man made products.

A ***stereoscopic*** method or system requires that two images, with slightly different horizontal perspectives, be presented to the eyes simultaneously (or almost) in a convincingly manner so that the Human Visual System (HVS) uses them, instead of the real world, in the stereopsis process from which depth is perceived [54].

3.1.3. Binocular disparity and stereopsis

Binocular Disparity explores the differences in geometry obtained from binocular vision. Since human eyes are levelled on the same horizontal plane they only provide horizontal disparity. Therefore, in the context of human vision, ***Disparity*** usually relates only to horizontal distances, although the concept itself can be equally applied to both vertical and horizontal distances, so it is a good practice to explicitly mention the direction of disparity [55], [56], [53].

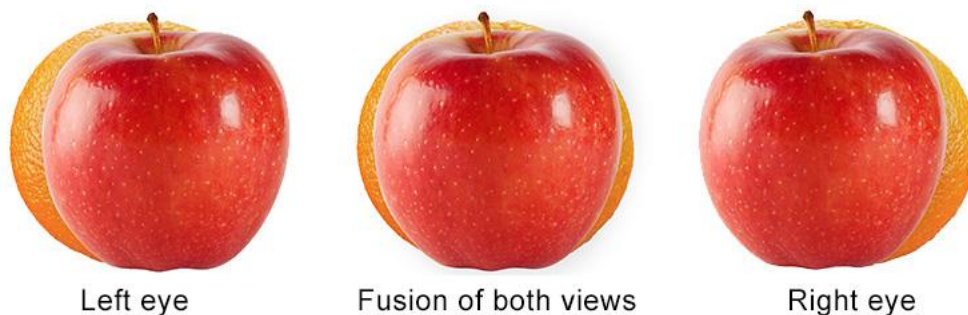


Figure 11 – Effects of horizontal disparity on binocular vision (original creation from parts of [4]).

In Figure 11 we can see the effects of horizontal disparity in binocular vision. Notice the slight shift of the apple, relative to the orange behind it, in the three depictions. This happens because the apple is nearer than the orange. Since depth is inversely proportional to disparity (i.e. the closer an object is to the camera or the eye, the greater the disparity is) [57] it may be used to determine the depth of objects in a scene [8], [58]. This is the underlying principle of ***stereopsis***.

In the context of photography, **Disparity** may be understood as the distance between a specific point of the world, represented in two images of the same scene but taken from different perspectives.

3.1.4. Sensing light

Each eye behaves much like a visual sensor by collecting rays of light from one single direction, which is controlled by the movement of the head and the eyeball. More detailed data is collected on the retina of the eye coming from a specific spot of the world where the image is focused with the help of the lens.

Although the dynamic range of wavelengths of light is, in practice, virtually unlimited, the resources of any biological system, such as the HVS, are not. So, in order to make the process of interpreting light more “manageable” and possible to occur in real time, the HVS processes only a narrow band of the full light spectrum, between about 400nm to 700nm, which is commonly known as the spectrum of visible light [32, pp. 44-51], [59, p. 5].

3.1.5. Interpreting light

We live in a world of light. The sun is our major source of natural light during daytime and in its absence we may use artificial light, like that produced by a lamp.

One fact that may not appear immediately obvious though, is that most of the light rays processed by our eyes do not come directly from its source (be it natural or artificial); instead, they come from some sort of reflection on other objects and structures of the world around us. This constant process of bouncing off other surfaces makes the light rays lose energy until they become so faint that we cannot perceive them.

To interpret the apple of Figure 12, the eyes capture all the rays that are reflected

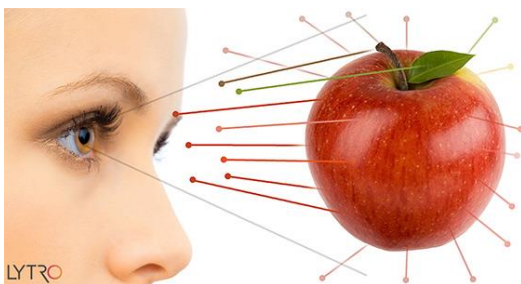


Figure 12 – Light rays reflected off an apple [4].

from the apple in their direction and process those with enough energy to be detected by their retinas. Since we have binocular vision, our brain gets two images of the apple from two slightly different perspectives (as shown in Figure 11) and, mainly through stereopsis, it is able to infer depth and

therefore we are able to perceive the apple in 3D space with its appearance [4], [8, pp. 6-8], [32, pp. 44-51].

3.2. Light Field Theory Overview

3.2.1. The plenoptic function

An image is formed from the light rays that hit the capturing device sensor. If we were to fully reconstruct, in space-time, the scene represented by that image from those light rays, we would need to store seven parameters for each light ray, relating to time, frequency (or wavelength), space and direction.

These parameters were formally described by Adelson and Bergen, in 1991, through what they called the **Plenoptic function** [60], which was an improvement on the work done by the Russian physicist Andrey A. Gershun, in 1936, regarding light fields [61].

The Plenoptic function has been used as a modelling function in computer graphics related fields to express the image of a scene, viewed from any possible position and angle, at any moment in time; or, paraphrasing Carl Sagan, as a description to the omnipresent image of everything that ever existed or will ever exist, on a specific scene. It serves as the basis for much of the light field imaging theory and for that reason a more detailed explanation of its parameters follows.

3.2.1.1. Time

In a metaphorical language, one could say that a photograph is like a moment frozen in time. The literal meaning of this sentence is not far from reality though, because each picture is truly unique! Even if we retake that picture, it will be done in a different moment in time and therefore it will use different light rays, even though their properties might be very similar to the ones captured originally.

Nevertheless, since time does not vary in a photograph it is not important for our present work and will consequently be ignored, for the sake of simplicity.

3.2.1.2. Wavelength

In the process of image formation, this is the value that will later be translated into the colour that we see on every picture element.

Wavelength (λ) is measured in nanometres (*nm*). Since *Frequency* (f) is inversely proportional to wavelength, it may also be used to describe this property of light.

3.2.1.3. Space and Direction

To complete the representation of a light ray we need to know a point in space where the ray passes, as well as its direction. The former requires three parameters, usually x , y and z , whilst the latter needs two angles, commonly represented by the Greek letters θ and ϕ . If we restrict the Plenoptic

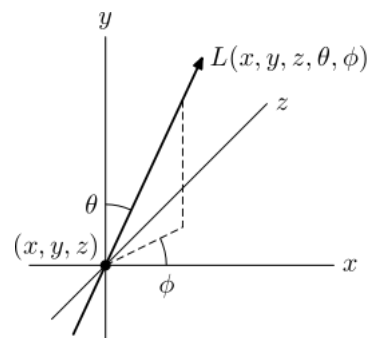


Figure 13 – Parameterization of the 5D Plenoptic function [62].

function to these five geometric parameters, it then becomes a 5D function that expresses the flow of light in 3D space and can be represented as seen in *Figure 13* [58], [62], [60].

3.2.2. Radiance

For this work, perhaps the most important aspect to comprehend about **Radiance** (L), along a light ray, is that it may be understood as the amount of energy given by light particles (*photons*) flowing through all possible straight lines in a constrained tubular area.

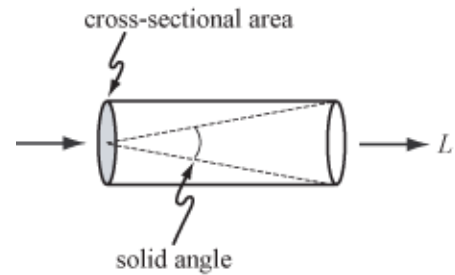


Figure 14 – A light field represents the amount of energy across a tubular area [62].

This work does not require any formal measurement of radiance because that is done by the optics and electronics systems of any light field camera. Yet, this is an important concept to grasp since it is the final “piece of the puzzle” in the process of understanding what a light field really is, and for the sake of completeness it should be noticed that *Radiance* is measured in *watts* (W) per *steradian* (sr) per *meter squared* (m^2). The two variables that define the actual tubular area in which light is measured are the steradian, which measures a solid angle, and the meter squared, which measures the cross-sectional area, as Figure 14 visually expresses [62], [58].

3.2.3. Light field

Considering the information conveyed in the previous sections and using a more formal approach, it is now possible to affirm that a **Light Field** represents the total amount of energy (*radiance*) moving in all directions (to which we called *light rays*), throughout all points in space, in a given area [58], [62], [4], [63].

3.3. Light Field Image Formation

3.3.1. Brief overview of a light field camera

It is out of the scope of this work to discuss technical details about the optics or the electronics systems of a light field camera. However, learning a few things about those may help in better understanding certain concepts that are important for this work, so a brief overview follows.

3.3.1.1. Microlens Array

A conventional 2D camera captures light with the lens and sends it directly to the sensor where the image is formed, as shown on the left side of Figure 15. However, a light field camera has an intermediary between both: the *Microlens Array* (also written as Micro Lens Array and abbreviated as MLA).

The **Micro Lens Array** is a collection of hundreds of tiny lenses that sits before the sensor and acts as a camera array that focuses the image captured by the main lens onto the sensor (right side of Figure 15).

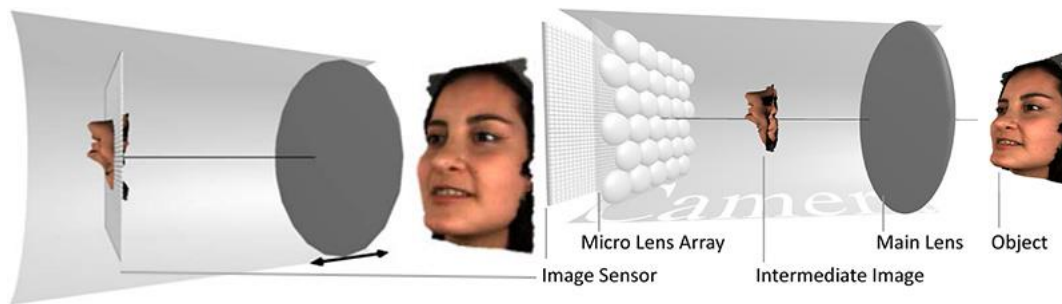


Figure 15 – Left: Conventional 2D camera (lens focuses directly onto image sensor). Right: Light field camera (lens generates intermediate image, which is then focused by the Micro Lens Array onto the image sensor) (adapted from [64] and [5]).

This architecture allows the capture of hundreds of tiny images called **micro-images** that contain small “pieces” of the scene in various different perspectives, according to their physical position in the MLA. Those “pieces” may then be used to form actual 2D images or reconstruct 3D space up to a certain level.

Since there are many micro-images with slightly different perspectives, there is some redundancy in the image data that is captured from the scene by each micro-image and that fact may be explored to infer the depth of objects in the scene with great accuracy.

The disparity and/or depth information that can be gathered by a light field camera, with precision and reliability, constitute one of the major improvements over traditional photography [65], [5], [64], [4].

3.3.1.2. The source of light rays

In conventional photography the camera’s main lens captures the colour and the brightness of rays, but considers all the rays that hit the same point in the sensor as a single unit. It does not gather data regarding the number of rays that contributed to that unit nor their directions (*angular information*) or, in other words, a traditional camera does not capture information regarding the source of its data. Consequently, a traditional 2D image, by itself, does not contain enough information to reliably and accurately allow the reconstruction of the scene it represents from its original data alone.

Although it is possible to generate or estimate the missing data (e.g. depth) from one single image, it is a very demanding process that is seldom (if ever) as reliable and accurate as the capture of the original missing data would be.

This is a fundamental limitation of conventional 2D photography that light field photography is able to partially overcome.

Light field photography does gather data about the source of its rays, such as radiance and angle, which in turn allows the rays to be traced back to their original position in 3D space, and consequently supply the required data for reconstruction of scenes in both 2D and 3D space, from its captured images [4], [58].

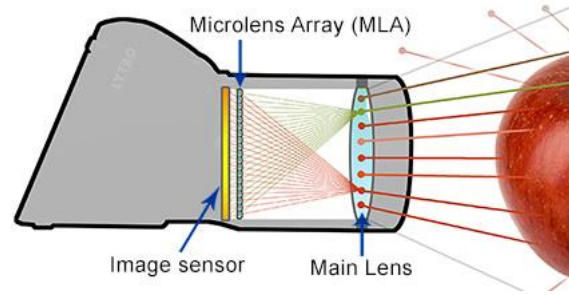


Figure 16 – A light field camera captures angular information (adapted from [4]).

The Light Field Processor application, developed in the context of this dissertation, explores this feature of light field photography, up to a certain degree (for details see *Chapter 4*).

3.3.2. Light field image

3.3.2.1. Conceptually

Using simple terms, one may say that a **light field image**, as implemented by current light field photographic cameras (i.e. captured through a MLA), is a single-shot collection of 2D images with slightly varying perspectives, usually stored in a two-dimensional grid format [58], [62]. Considering this multi-perspective feature, it can also be defined as a **multiscopic image** since it has multiple views of the same scene.

3.3.2.2. Dimensionally

The **spatial domain** encompasses all data that represents the physical space occupied by an image. In digital imaging, the *Picture Element* (or **pixel**, as it is commonly known) is the smallest piece of construction of space, in an image. As a curious remark, it may be added that due to the phonetics of the English language, the term *pixel* is actually an attempt of merging the words *picture* and *element* through their initial letters (*“pict”* and *“el”* respectively).

Besides the traditional two spatial dimensions in 2D images, a light field image needs two more dimensions to represent the angles of light rays as they were captured. These two extra dimensions constitute the **angular domain**.

Such a 4D light field may be parameterized in different ways. One common method is the two planes parameterization, shown in Figure 17. A possible way to interpret it is to consider the s, t parameters as the equivalent x, y coordinates in a traditional 2D image, and the u, v parameters as expressing the angular domain, relating to the θ and φ angles from the plenoptic function. For a detailed explanation of parameterizations, see [58], [62], [66].

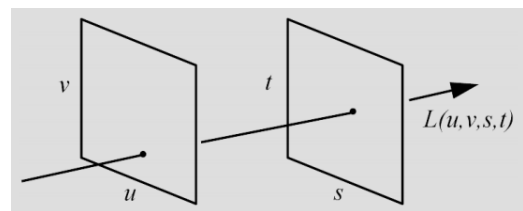


Figure 17 – Parameterization of a light field using two planes (adapted from [66]).

3.3.2.3. Structurally

A light field image may be interpreted as a 4D structure, composed of hundreds of tiny 2D micro-images, each captured by an individual microlens and stored as a 2D grid (hence the four dimensions). Figure 18 shows a piece of an amplified raw light field image from a Lytro Illum camera in which those tiny micro-images are perceptible through the hexagon shaped grid of microlenses [67], [13].



Figure 18 – An amplified slice of a raw light field image.

The horizontal and vertical dimensions of an extracted 2D image relate directly to the number of micro-images in those same orientations, because each one contributes with an equal number of pixels to the final 2D image (unless we want a distorted image, horizontally or vertically).

The area size of a decoded light field image, in pixels, is the product of the spatial dimension of each micro-image by the total number of micro-images in that dimension. For instance, if a micro-image has a size of 15 x 15 pixels and there are 20 micro-images horizontally and 10 vertically, the raw light field image should have a dimension of 300 x 150 pixels (15 times 20 x 15 times 10).

Some light field camera models (like those from Raytrix [5]) may also contain some explicit 3D data about a scene, like depth, but generally that information is computed (in post-processing) from the disparity of the many micro-images available. So, strictly speaking, a light field image is not a 3D data format, because it does not contain 3D space data ([4], [5], [58]), like, for instance, a *Point Cloud* data structure (for a Point Cloud definition see [68], [69]), although it does allow for accurate 3D space reconstruction, up to a certain degree, in post-processing [64].

For this work, it is also important to note that the Lytro Illum camera produces its own light field file format, with extension **.LFR**, that stands for *Light Field Raw* [70]. It contains 4D data that must be unpacked in order to extract the actual raw light field image (sometimes also informally known as “*honeycomb*”), which in this work we call **Sensor Raw Image**, for which an example is shown in Figure 18.

3.4. Light Field Image Processing

The main idea behind light field image processing is that it performs transformations on data from light field images, so any valid definition of *Image Processing* such as [71] or [72] may serve as the basis to understand this concept. However, it is not enough. There are procedures, like *decoding*, and terms, like

white images, that are related specifically to *light field image processing* and require a contextualised explanation; hence the need for this section.

3.4.1. Decoding

What *decoding* actually means varies depending on its context or what it refers to. For instance, in the context of data compression, decoding simply means decompressing the data back to its original state.

However, in the context of Lytro imagery, which this work shares, ***decoding*** refers to the process of converting the raw light field image data, contained in LFR files, into a 4D data structure, that is more manageable and allows the extraction of useful information from it, such as, for instance, 2D images (viewpoints) and metadata.

To learn how it is used in LF Processor and technical details of its implementation see *Section 4.4.3*.

3.4.2. Viewpoint

A ***Viewpoint*** (sometimes simply referred as *View*, in this work) is actually a 2D image reconstructed from the light field image data, with a unique perspective. It therefore provides some horizontal and vertical parallax information [8] which, in conjunction with other Viewpoints, may be used to extract depth information about objects in the scene.

Due to the practical nature of this work, it became convenient to classify Viewpoints according to three types. For a detailed explanation of those, why they may appear in light field images and how they are used in LF Processor, see *Section 4.4.6.1*.

3.4.2.1. Views Map

A ***Views Map*** is a regular 2D image that assembles all the individual viewpoints (views) of the 4D light field image after the decoding process has occurred. By showing the viewpoints respective positions, it provides an easy way to get a global idea of what a light field image is and how it may be organised. These images are usually very large, so only a small portion of one is shown in Figure 19.

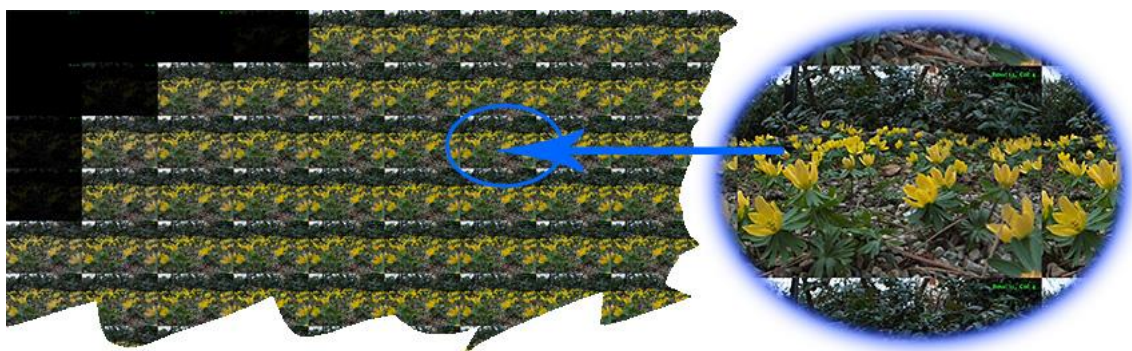


Figure 19 – A cut on a Views Map emphasising an amplified Viewpoint.

3.4.2.2. Microlenses Map

A **Microlenses Map** is similar to the Views Map, with two differences: its grid orientation respects that of the MLA (hence its name) and it contains the micro-images obtained by each microlens.

3.4.3. White image

A **White Image** is an image taken through a white diffuser or of a white scene. Since the scene is supposed to be all white, its pixels should represent absolute white, which in an 8-bits RGB scheme would be: R=255, B=255, G=255 [13].

Since only the centres of the microlenses are able to capture all the light, due to reflectance, *White Images* are used to estimate the centre of microlenses and also to do “**Devignetting**”, which consists in countering the undesirable effect of reduction of light at the edges of images, caused by the curved shape of the lens, known as *Vignetting* (for a definition, see *Section 3.4.8* below). For more information on how this might affect light field images, see *Section 4.4.6.1*.

White Images may also be used to detect defective pixels or anomalies in the MLA. Points which are either outside or far from the border of any microlens, but are still not totally white in a White Image, may be considered as defective and thus benefit from special interpretation (processing).

Figure 20 shows an amplified piece of a White Image extracted during the decoding process of the Light Field Toolbox, where it estimates the microlenses centres. The slightly oval shaped microlenses may be distinguished through its shades of grey, with a red dot indicating its estimated centre.

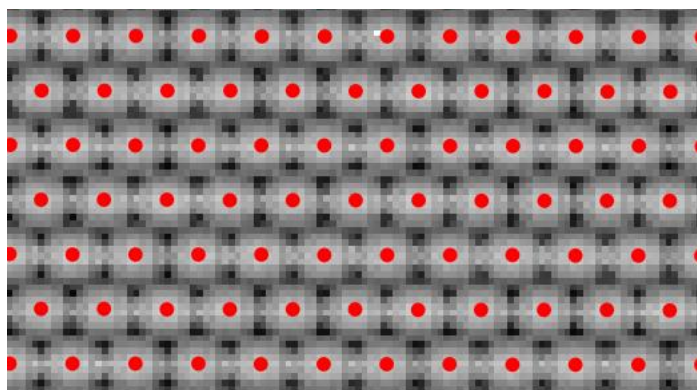


Figure 20 – Amplified sample of a White Image shown by LF Toolbox while decoding [13].

3.4.4. Depth of field

In optics and photography, **Depth of Field** (DoF) is the distance between the nearest and farthest objects in a scene that appear properly focused, or by other words, that are acceptably sharp.

Since the defocus (lack of sharpness) is gradual, determining this distance is not an exact science since it depends on visual acuity, viewing conditions and image scale.

Nevertheless, it may be said that it always lies somewhere between the boundaries of what is and what is not considered to be sharp enough [8], [73].

3.4.5. Depth of focus

It is important to notice that this term is not the same as the previous (DoF) although their expressions are almost identical.

Depth of focus is a concept used in optics to measure the tolerance of placement of the image plane in relation to the lens [74], or as summarised by Ren Ng in [8], “*the very narrow range of sensor depths inside the camera for which a desired plane will be rendered crisply*”.

3.4.6. All-in-focus

For conventional photography this is considered a *technique* that requires expert skills [75], but for light field photography this is simply a *feature* that can be easily explored with a tool like LF Toolbox or an application such as LF Processor.

As the name implies, this concept refers to the fact that everything in the scene of a photograph is properly focused. Thanks to this useful feature, in a light field image there are never out of focus background objects, unless the user wants it.

All viewpoint images extracted with LF Processor are automatically all-in-focus.

3.4.7. Refocusing

Refocusing is a new concept that applies only to light field photography and means exactly what it says: the ability to focus a picture after the fact (of being shot).

Although it is a simple concept to understand from the user point of view, it is a somewhat complex feature to implement computationally.

3.4.8. Vignetting

Vignetting is the result of the physical limitation of capturing light near the edges of lenses (or microlenses), due to the angle at which light hits them. Thus, the light gathered near the edges of lenses is usually weaker than the light gathered at their centres because of the lenses intrinsic curvature, which results in images that are darker at the corners reflecting the usually oval shape of the lenses [76], [77].

CHAPTER 4

LIGHT FIELD PROCESSOR

*"In the beginning there was raw light. Then,
LF Processor decoded it, and after a minute or so,
there came new light: a Decoded Light Field".*

Paraphrasing the "LF Bible"

This chapter presents the Light Field Processor software application and its main goals and features, especially from an image processing perspective.

We assume that the typical user of this application possesses some knowledge in the field of image processing but has little or no knowledge in light field image technology. We also expect any reader of this chapter to have already been through the previous chapter regarding basic light field concepts which are required to understand some of the features of this application.

LF Processor was developed especially to those interested in learning more about light field image processing and its potential, from **a practical perspective**, such as students, researchers, or simply a curious mind. Moreover, since it may also automate most of its operations (such as decoding, image processing operations, saving DLF files, extracting outputs, amongst others) it may also be useful to people doing repetitive tasks (e.g. comparing results of algorithms of depth filters).

4.1. What is Light Field Processor?

Light Field Processor (LF Processor) is a software application for the Windows operating system, developed with Matlab. Its main purpose is to extract, present and store useful information from light field images produced by any Lytro Illum camera (for this device's technical specifications, see [52]).

Appendix A contains detailed instructions for application deployment, which includes information regarding what versions exist, which one best suits a user's needs and how to install it.

LF Processor was built around the idea that, in order to be useful, a light field image must first be *processed*. This actually means several operations such as: unpack the light field image data from the raw file (.lfr); convert it into practical data structures which will allow it to be interpreted to produce new information from that data; present it in useful ways and export the processed data in appropriate file formats as outputs, according to user preferences. Figure 21 outlines the most important functional aspects of the application which will be discussed throughout this chapter.

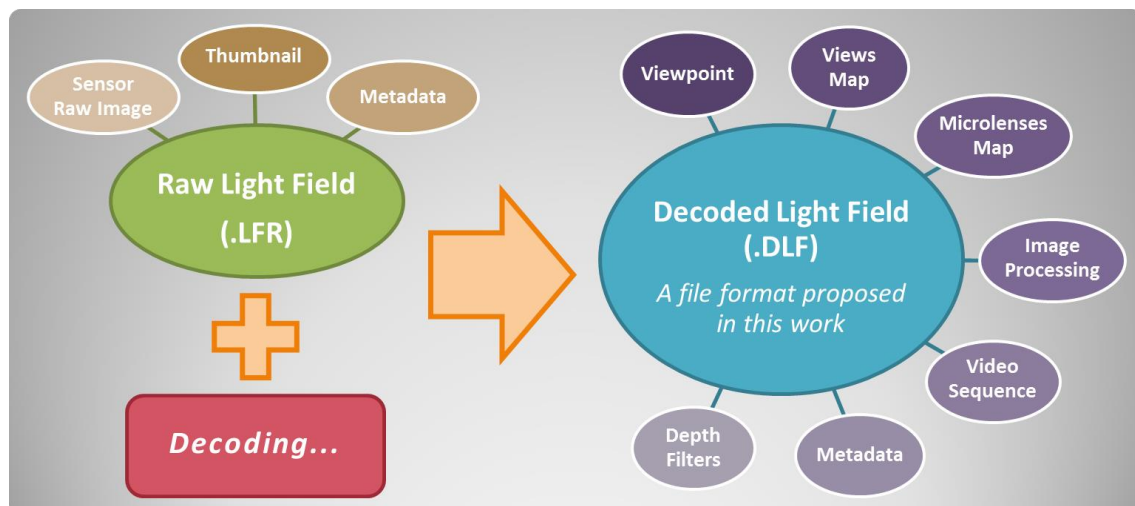


Figure 21 – Processing flow of the Light Field Processor application, emphasising its main functional purpose and outputs.

LF Processor accepts two types of input (for details on these file types, see *Section 3.3.2.3* and *Sections 4.1.2, 4.4.1.3*)

- *Light Field Raw* image file (with extension .lfr/.LFR);
- *Decoded Light Field* image file (with extension .dlf/.DLF).

The first is the proprietary file format created by Lytro for its Illum cameras, for which this application was specifically developed.

The second is the main original output of this application, a file type containing a complete 4D light field image data structure and its metadata.

In order to pass from the first to the second file type, the LFR data has to be *decoded* (Section 4.4.3), which might take around 1m15s on a PC with an Intel i7 processor @ 2.6 MHz (16 MB RAM, 500GB SSD disk), but may be more or less, depending on the computer used. After that operation is finished, the new data may then be saved as a *Decoded Light Field* file (.dlf). Regardless of saving, or not, that data to a file, it will then be possible to execute all the DLF related operations.

4.1.1. LFR related outputs

A raw light field file is used mainly as the starting point for decoding. However, it also delivers outputs that do not require any processing:

1. **Thumbnail:** A jpeg file with the central viewpoint of the light field image that may or may not exist embedded in an LFR file.
2. **Sensor Raw Image:** The original and “*undecoded*” light field image:
 - a) The *Black and White (B/W)* version is exactly as extracted from the camera sensor (i.e. a monochrome image with a bit depth of 10 bits).
 - b) The *Colour* version went through a *debayering* process to acquire colour. For details about this process, see [78], [79].
3. **Metadata:** A plain text file with metadata regarding the light field image itself. Includes data such as Zoom and Focus settings that were used to capture it.

4.1.2. DLF related outputs

A decoded light field file is the ideal starting point for almost all processing tasks. Although it is always possible to start with an LFR file and then decode it to get a decoded light field image, why repeat this time-consuming process every time? The DLF file format was created exactly to avoid this repetitive process.

We present now a brief overview of the main operations that can either process the data of, or extract data from, a DLF file, or an LFR file after being decoded (for details about how they work in LF Processor, see Section 4.4):

- **Viewpoint, Views Map and Microlenses Map:** These are regular 2D images built from data extracted from light field images (see Section 3.4.2 for details);
- **Image Processing:** The operations in this category are:
 - ◆ *Colour Correction;*
 - ◆ *Gamma Correction;*
 - ◆ *Histogram Equalisation;*
 - ◆ *Improve Borders:* Special operation to fix a problem that sometimes occurs with the outermost pixels of a light field image, after decoding it with LF Toolbox, as we do (for details, see Section 4.4.4).
- **Video Sequence:** This is a video file (in .avi format) that contains frames from the various Viewpoints of the light field, organised in different ways to:
 - ◆ Emphasise the parallax in specific directions (vertical, horizontal and diagonal);
 - ◆ Present viewpoints in a dynamic way, from *Bottom-Up* or from *Top-Down*.

- **Metadata:** A plain text file with metadata regarding the decoded light field image itself.
- **Depth Filters engine:** This is perhaps one of the most interesting features of LF Processor, especially to become acquainted with the effects of Depth Filters on a light field image (for details about its use, see *Section 4.4.5*).
 - ◆ It is a custom-made engine that provides an easy, yet powerful, interface to the automatic execution of one of five depth filters available, useful to run systematic tests and comparisons;
 - ◆ It also allows continuing a previous session from the point where it stopped in case the results are not decisive.

4.2. LF Processor: Usability

This chapter was not meant to be just as a step-by-step “User Guide”. No user should rely exclusively on it to clarify all doubts about the usability of this application. Some details about how to proceed are explained through the graphical interface or accessible through help buttons, which should be used to clarify certain technical procedures. Besides that, this software application, like any other, requires hands-on practice to get to be known.

This section explains how to access some of those pieces of crucial information and guidance that may be found within the application itself.

4.2.1. Obtaining help inside the application

4.2.1.1. During configuration

Some features of the GUI in the Configuration procedure may not be immediately noticed by the user, but they are very important, so they will now be emphasised.

- Each of the three steps of the initial Configuration has its own tab although it is only shown as required; so, initially only the first tab is visible (Figure 22). The intention was to not overwhelm or confuse the user with different types of information that may or may not be yet required.
- All tabs have a “How to...” or “Why this?” button in the top right corner (which should be self-explanatory), providing critical information that should be read. In order to “persuade” the user to do so, these buttons start with green colour and will only change into regular colour after being clicked at least once. When that happens, a window opens up with **specific instructions** and **information** about that particular tab contents. They are not mandatory, though.
- A different colour changing scheme is used for critical *Action Buttons* (the ones the user is enforced to click) like the “*Select Folder*” button shown in Figure 22. It starts red coloured to inform the user that it is a mandatory button and will change to regular colour after the user selects a valid folder.

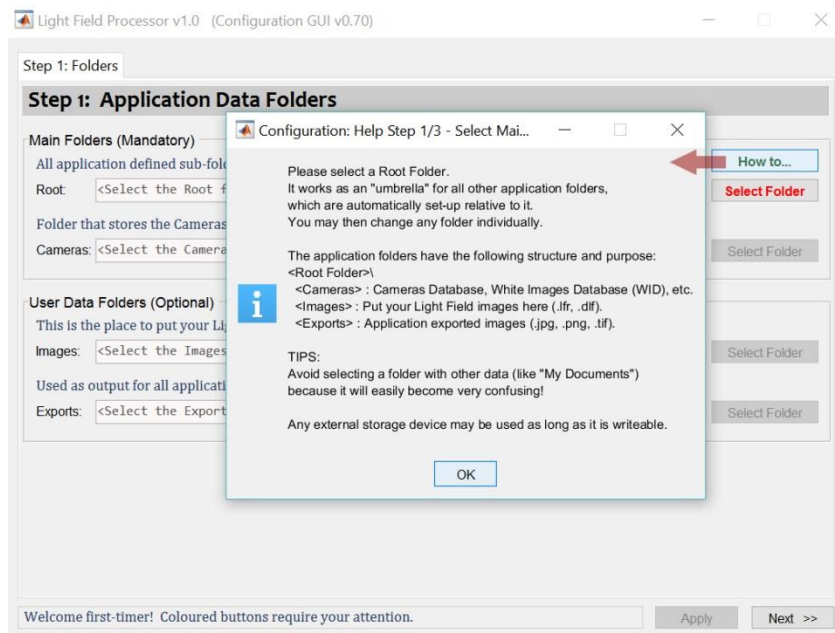


Figure 22 – Initial Configuration procedure - Step 1: Application Data Folders.

4.2.1.2. Globally

Here we present some features that are common throughout the whole GUI.

- Tooltips are shown after stopping the mouse for two seconds over a button (Figure 23).
- Recently inputted user data, is shown in Iron Orange (Configuration and User Preferences screens) until it is validated using either the *OK* or the *Apply* button of the respective screen.
- Using colours is not merely aesthetic but is instead informative: the Iron Orange colour means that the data was not yet saved on the internal data file, and therefore if a user cancels the pending operation, that data will be discarded.

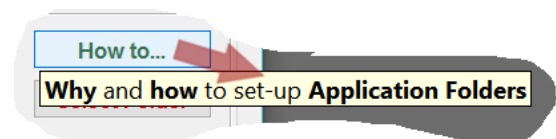


Figure 23 – Button Tooltips.

Only after pressing *Ok* or *Apply* buttons is the data on screen actually saved on the application internal data file (this is relevant in Configuration and user Preferences screens).

4.2.2. Application folders

Due to the somewhat complex, diversified and especially repetitive nature of the outputs of this application, an automatic management scheme for file saving was adopted; so, the application needs to know beforehand the path to certain key folders.

There are four *Application Data Folders* required by LF Processor before it can run for the first time, hence their importance, although only one must be explicitly selected by the user. The *Application Configuration* is where that selection of folders may occur (Figure 22).

A brief explanation of those folders and their purpose follows:

1. Root Folder:

- a) As the name implies, it serves as reference for all other folders' path;
- b) No file is exported to this folder by default; only to its subfolders;
- c) It will be used as the main deposit for all user files (files copied or created under user supervision).
- d) The selection of this folder is mandatory and it should be the first one to be selected;
- e) After this folder is select, all other folders are automatically set based on it. However, the user may then change the remaining folders individually.

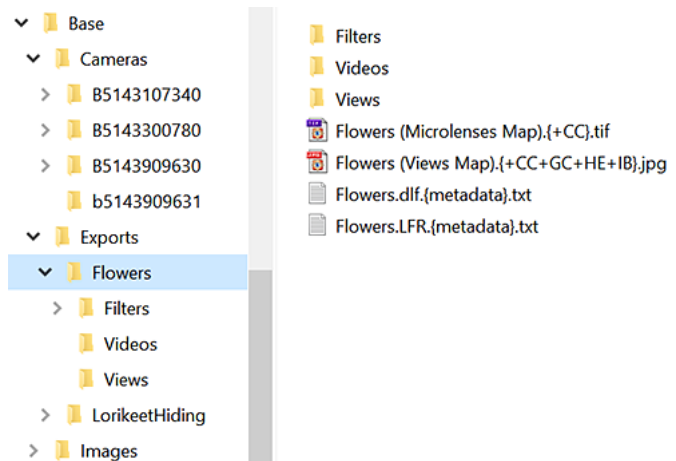


Figure 24 – Application folders structure showing the exported contents of the “Flowers” light field image.

2. Cameras Folder:

- a) Stores files relating to the Cameras Management System;
- b) Examples include: Cameras List, White Images Database (WID) and calibration data regarding to the cameras installed.

3. Images Folder:

- a) This is the default folder to open LFR and DLF files.
- b) Nevertheless, a user may choose any other folder to open a file.

4. Exports Folder:

- a) Used as the base path for all application generated files;
- b) All exported files from a light field are saved inside a folder bearing its name, and hereby called “*light field folder*” (Figure 24);
- c) For instance, for a file named “Flowers.lfr” or “Flowers.dlf” the exports subfolder will be: “Flowers”.
- d) Only certain files are directly saved in the *light field folder*, such as Maps (Views and Microlenses) and Metadata (LFR and DLF);
- e) All the remaining files are saved inside a subfolder of the light field folder, named after its contents; here are some examples:
 - Filters: folder to save the outcome of depth filters’ operations;
 - Videos: folder for saved video sequences;
 - Views: folder for saving 2D viewpoints.

4.2.3. Automatic naming scheme for exported files

Exported files are named automatically according to two properties: (1) the original LFR or DLF file name; (2) the processing tasks that were already performed on the file or, in the specific case of exports from Depth Filters, the parameters used.

Image processing tasks use the initialisms of their operations, preceded by a plus sign (+) to mean inclusion of operation, and then put inside curly braces just before the file name extension. Initialisms of image processing tasks are:

- Colour Correction: **CC**
- Gamma Correction: **GC**
- Histogram Equalisation: **HE**
- Improve Borders: **IB**

For instance, “myLF.{+CC}.dlf” informs that the light field image “myLF” is already decoded (.dlf) and that its data went through a *Colour Correction* (+CC) procedure. More examples of file names using this structure may be seen in Figure 24.

When exporting files using the Depth Filters engine, the parameters have a similar structure but vary according to the specific filter algorithm used.

4.2.4. Validation of user inputs

In most cases, LF Processor validates user inputs automatically through the use of GUI components such as lists, scrolling bars, radio buttons and others, in particular when the range of inputs is predetermined and relatively short, therefore freeing the user from such responsibility. An example may be found in Figure 22.

However, it is not always possible to validate data properly through the use of graphical components alone, especially when the range of input values is large or the precision required is greater than the one available by a graphical component that is usually handled by a computer mouse. This is particularly evident in cases that require direct textual or numerical user input, like the parameters for image processing operations (e.g. Colour Correction).

Any user is always entirely responsible for the inputs he/she supplies to a software application. Since the emphasis of this work was not about producing a commercial software package, **in the cases in which the user is required to directly introduce numerical or textual values** (mainly image processing tasks and the preferences settings of Depth Filters), we used only basic validation (e.g. validation of parameter boundaries), which still works well for most situations, but may sometimes allow user mistakes.

Therefore, if something does not occur as expected, we recommend the user to pay special attention to the parameters that were introduced. As usual, a small, but unnoticed, typing error may be enough to produce wrong results.

4.2.5. Running LF Processor for the first time

Certain aspects of LF processor (e.g. Application Folders) must be configured before the application may be run for the first time. This short section serves as a guide through that stage and also to explain the importance of some configuring decisions made by the end user through the application’s Configuration option.

An LFR file, by itself, is not an image but instead a collection of raw image data that requires processing.

Light field image processing allows the extraction of 2D images from LFR files, but it requires both metadata, containing the settings (e.g. zoom, focus) used to capture the picture, and calibration data, regarding the camera used. That is why a *Configuration* is required beforehand.

The first time the application runs, two things should happen: (1) A browser window should open up (Figure 25) with crucial information on how and why to make this initial configuration procedure; (2) An application window should open to perform the Configuration steps, as shown in Figure 22.

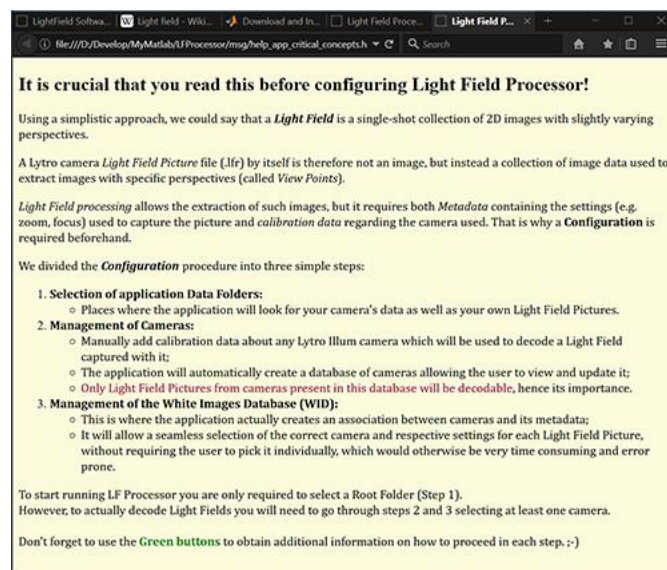


Figure 25 – Crucial information before configuring LF Processor.

The Configuration procedure was divided into three independent steps. A user needs only to follow the on-screen instructions for each step, presented below:

1. Selection of *Application Data Folders* (Figure 22)
 - a) As explained in *Section 4.2.2* above;
2. Management of Cameras
 - a) Manually add calibration data about any Lytro Illum camera which will be used to decode a Light Field captured with it;
 - b) The application will automatically create a list of cameras allowing the user to view and update it;
 - c) Only light field images from cameras present in this database will be decodable, hence its importance;
3. Management of the *White Images Database (WID)*
 - a) This is where the application actually creates an association between the cameras and their metadata;
 - b) It will allow a seamless selection of the correct camera and respective settings for each light field image without requiring the user to select it

manually (by passing its full path as a parameter to a function), which would otherwise be time-consuming and error prone.

NOTE: To start using LF Processor right away and open LFR files, the user is only required to select a Root Folder (Step 1). However, to actually decode light field images the user will need to go through steps 2 and 3, selecting at least one camera (the one with which the light field pictures that need processing were taken).

4.3. LF Processor: Graphical Interface

The main file types (LFR and DLF) used by this application store different data structures and allow different operations too. Therefore, they are presented with a different graphical interface. This section explains the major data elements visible on screen whenever one of that file types is opened.

4.3.1. Main screen

The application starts by showing the main menu, the toolbar and the status bar at the bottom. The workspace remains empty until a file is opened (Figure 26).



Figure 26 – Light Field Processor main screen, just after launch.

4.3.1.1. Toolbar

Initially, only the following buttons are available on the toolbar: Open light field file, Configuration and Preferences.

Figure 27 shows that the toolbar is composed by eleven buttons that encompass some of the most frequently

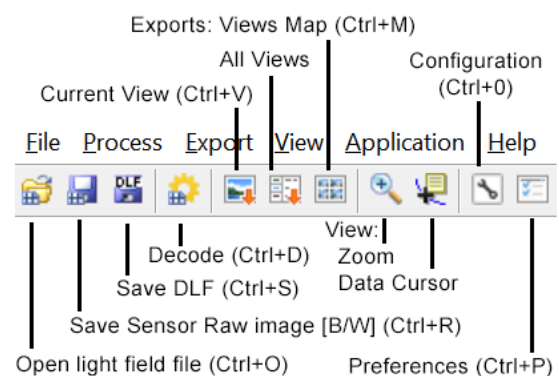


Figure 27 – Presentation of the Toolbar.

used features. The respective shortcut keys are presented in brackets near the description of each button and on the application itself through tooltips.

4.3.1.2. Status Bar

The Status Bar is a graphical interface component that is always present at the bottom of the respective window.

It gives feedback to the user concerning the current state of the application on things such as the stage, success or time taken, regarding the current or last operation, or simply the name and path of the last exported or opened file (the case shown in Figure 28).

4.3.2. Main workspace

When a light field image file is loaded, the workspace is split in two main areas, as may be seen in Figure 28: the *image panel* (left) and the *data panel* (right).

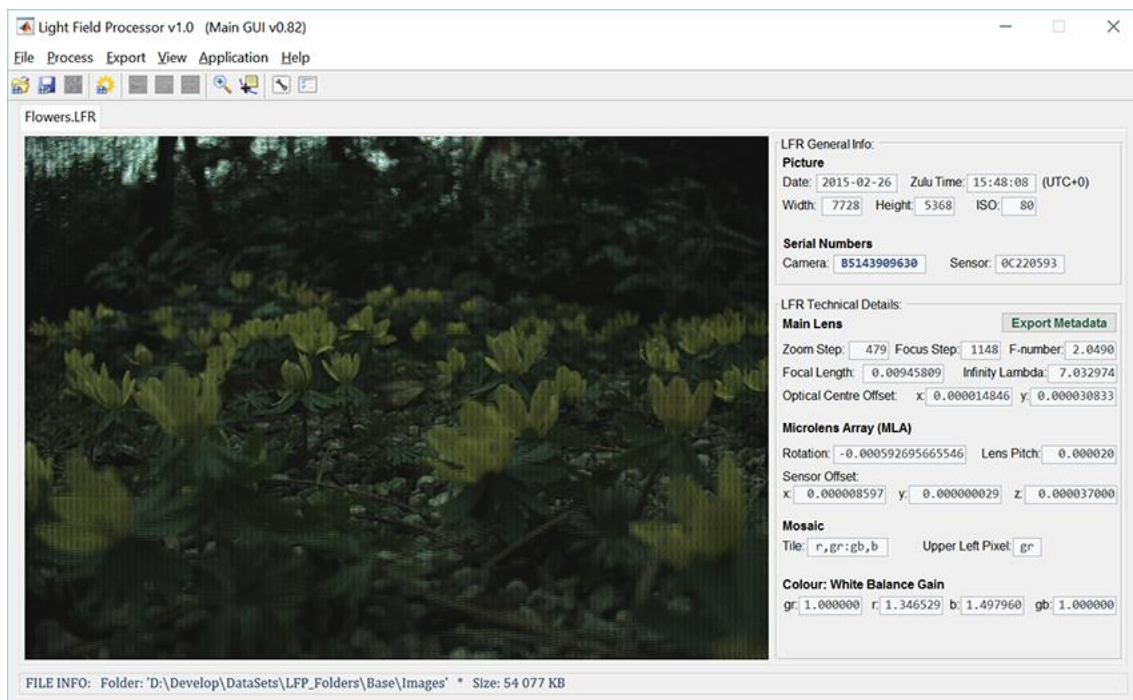


Figure 28 – Main screen and workspace after opening an LFR file.

4.3.2.1. Image panel (left)

The left panel shows the main raw light field image currently loaded, hence its designation. The name of the loaded file appears on the *tab* (top left) of its respective panel (see Figure 28 and Figure 30).

4.3.2.2. Data panel (right)

The right panel shows the metadata concerning the image loaded on the left panel and also navigational components, in the case of DLF files.

All data fields are grouped according to their contents and are identified with labels that should be self-explanatory. The data presented in this panel is extracted

from the loaded image file. In the case of LFR files, that data was created by the camera itself, at the moment of shooting the light field picture.

A substantial part of the metadata of DLF files comes from the original LFR; the remaining is generated when the file is saved. This means that when a user decodes a light field image and then works with it for a while, there will be a gap in time between the moment in which it was decoded and the moment when it is saved. It is the latter that is stamped in the *time* field of the DLF file.

This panel also includes the button “*Export Metadata*” (in green) that serves as a convenient shortcut to export all the metadata from that file (there are many more fields than the ones shown in Figure 28), instead of using the *Export* menu.

It is out of the scope of this work to go through a detailed explanation of the many dozens of data fields included in LFR or DLF metadata.

4.3.2.3. View options

The *View* options work with an *On/Off* model. Therefore, after clicking the toolbar button or menu option once, it stays *On* until the user clicks it again to switch it *Off*.

It is possible to zoom in (left click) or zoom out (Shift + left click) any loaded image by using the respective toolbar button (*View > Zoom*) and then clicking an area of that image.

The Data Cursor feature is available through the respective toolbar button or menu option (*View > Data Cursor*) and allows the user to see the actual RGB values and the coordinates of a pixel of the image by clicking on it with the mouse’s left button (Figure 29). Pressing the *Delete* key while the Data Cursor mode is *On* removes the current Data Cursor tip.

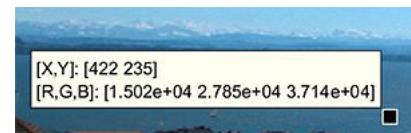


Figure 29 – A Data Cursor tip.

4.3.3. LFR workspace

When an LFR file is opened (Figure 28), a copy of its raw image (in B/W) is created and silently demosaiced, to acquire colour. It is this last version that is presented to the user, although both versions may be exported through the *File* menu.

4.3.4. DLF workspace

Figure 30 shows the workspace for a DLF file. It has a unique graphical component: the *Viewpoint Navigator*, which only makes sense using with a DLF file. There is also a specific tab name scheme that adds information to the file name.

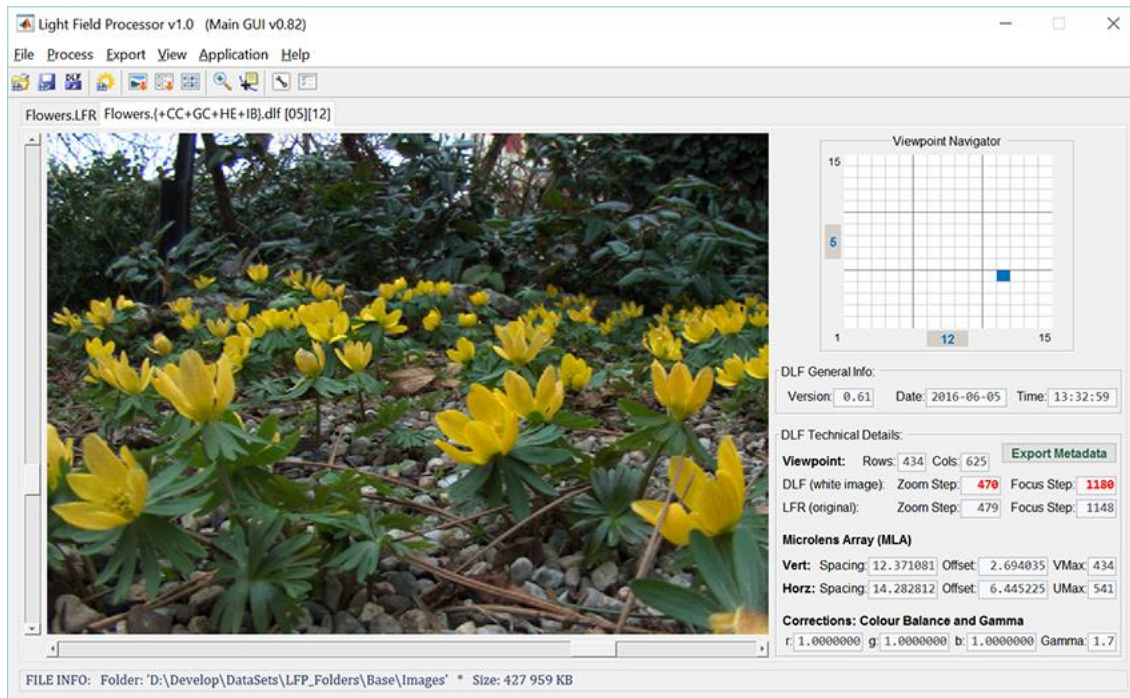


Figure 30 – Main screen and workspace after opening a DLF file.

4.3.4.1. The Viewpoint Navigator

The *Viewpoint Navigator* is a custom-made graphical interface developed specifically to better serve the purpose of navigating through the available viewpoints of a decoded light field image file.

It has several advantages over common *scroll bars*:

- Direct access in 2D space, with one single click through the left mouse button;
- Relative and step by step navigation through keyboard cursor (arrows) keys;
- Relative (keys) navigation allows immediate perception of parallax motion effect between viewpoints, in any direction;
- Shows the current viewpoint coordinates in a clear and intuitive fashion, in real-time;
- The coordinates shown also inform the user about the positioning of the current viewpoint in the light field image structure, thus helping the user to better understand what a decoded light field image actually contains and how it is organised.

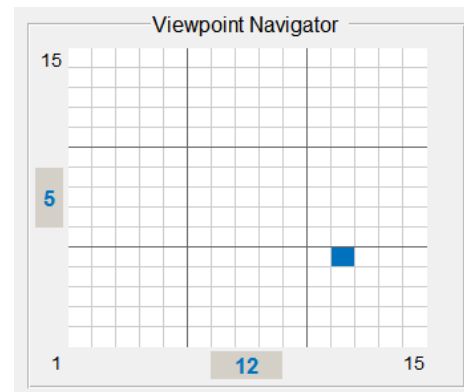


Figure 31 – The Viewpoint Navigator.

Figure 31 informs that the currently visualised viewpoint (on the left panel that is shown) is at coordinates Y=5 and X=12 in the light field image spatial dimensions (which in this case relate to the viewpoints in the 4D DLF data structure).

4.3.4.2. DLF tab name scheme

As mentioned above, the tab of a DLF contains extra information:

- **File Name:** The *base name* of the light field, extracted from the LFR file name;
- **Initialism:** Informs what image processing tasks were already performed to this data (already explained in detail in *Section 4.2.3*);
- **[Y][X] Coordinates:** Reinforces the information about the active viewpoint.

The DLF tab in Figure 30 is named as: “Flowers.{+CC+GC+HE+IB}.dlf [05][12]”.

4.3.4.3. DLF metadata vs. LFR metadata

Figure 32 shows the zoom and focus settings in red colour. This means that the values used to decode the light field image are different from the ones with which the original shot was taken by the camera.

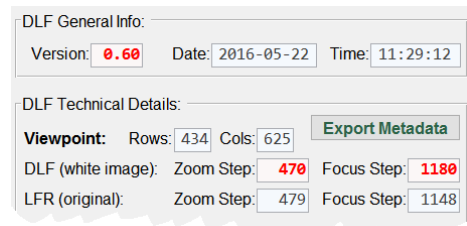


Figure 32 – Potentially problematic data shown in red.

During the decoding process, the original zoom and focus settings are looked up in the white images database and, if the exact values are not found, the closest ones are used. Since only a few tens are supported by the manufacturer of the camera, it is common that these values are not the same. However, the closer they are to the originals, the better the decoded image quality will be, so it may be useful to know these values.

4.4. LF Processor: Features

4.4.1. File menu

The **File** menu contains the usual file operations such as: *Open*, *Save* and *Close All* files. It also contains options to save the *Sensor Raw Image* as well as *Exit* the application.

4.4.1.1. Open light field

In LF Processor any operation must start by opening a file; two LF image types are accepted: LFR and DLF as shown in Figure 33. For more information about these file types see *Section 3.3.2.3* and *Sections 4.1.2, 4.4.1.3, 5.3.1*, respectively.

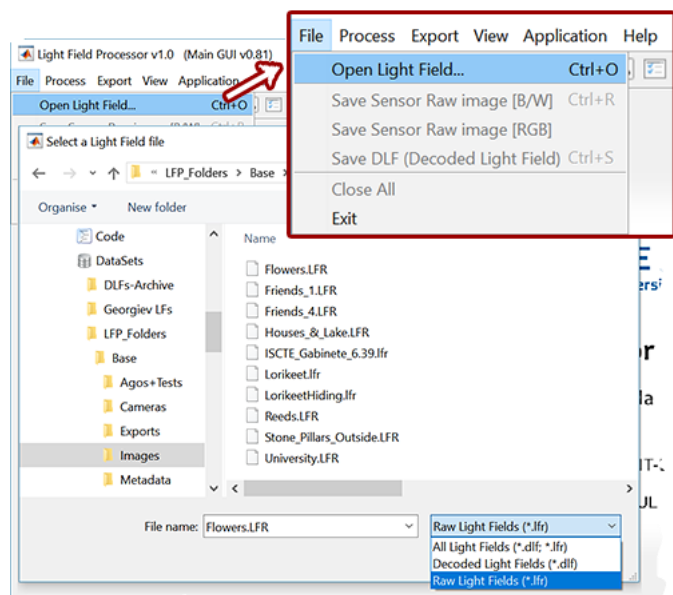


Figure 33 – *Open Light Field* dialogue box and the *File* menu.

4.4.1.2. Saving the sensor raw image

In LF Processor a **Sensor Raw Image** refers to the raw light field image captured by the sensor of a Lytro Illum light field camera, and contained in an LFR file, prior to decoding. Informally, this image is also sometimes known as “*honeycomb*”, due to the fact that it is made up of tiny micro-images that are organised hexagonally reflecting the shape of the physical organisation of microlenses in the Microlens Array.

The original raw image is monochromatic, with a bit depth of 10 bits per sample. However since this is a non-standard value for bit depth, the image is saved in either an 8-bit or 16-bit depth, in which cases the sample values are automatically scaled in proportion, by LF Processor. Internally, samples are scaled to 16 bits.

The output format of images, along with other parameters, may be changed in the application Preferences (to learn how to do it, see *Section 4.4.6.2* and Figure 42).

4.4.1.3. Saving and updating DLF files

If the workspace has a DLF tab opened, then it means that DLF data is in memory and may therefore be saved as a DLF file. If that file already exists, it will be updated, if it does not yet exist, it will be created. It may take around 4 to 20 seconds to store about 400 MB of data, depending on the disk’s speed. The file is automatically compressed by Matlab so it is not possible to know its size beforehand, although its true size on disk is always saved within the file, which may seem like a paradox. The programming “trick” that allowed that to happen is explained in *Section 5.3.3*.

The DLF format went through several changes during the development stages of this work, so it became important to define a *version* field that could be associated with the specific data and organisation contained within those files, independently of the application’s version. It would help in doing a better management of the current DLF files and also in identifying what data those files contained.

The version number of a DLF file may be seen in the data panel (right), just below the Viewpoint Navigator. In Figure 30 that field is visible in normal colour (dark grey) because it represents the most up-to-date version of the format (0.61). However, in Figure 32 the version is shown in red colour, meaning that this DLF was saved in an older format and requires an update to its information, which can be done automatically by the application just by resaving the file.

4.4.2. Process menu

The *Process* menu includes all the operations that require some sort of image processing. Since it includes three main groups, in which two of them have several options that require detailed explanation, we opted to split all three main groups in its own section, for clarity.

Most of these operations require input parameters. The default parameters were tested and chosen as the “safest” to start with; but what is “safe”, of course, may depend on each computer system (e.g. brightness of monitor). However, all input parameters may be changed before running each of these operations.

In case a user systematically uses a specific set of values as parameters, instead of having to constantly change them, LF Processor allows them to be set by default, in Preferences. Each section will supply instructions on how to do this for its operations.

4.4.3. Process menu: Decode

A conceptual explanation of this process already exists in *Section 3.4.1*. Here we explain **how the Decode process is used and integrated into LF Processor**.

LF Processor uses some of the functions developed by Donald Dansereau in the Light Field Toolbox v0.4 [13], to perform this operation. A detailed and technical explanation of how it executes the decoding of LFR files may be found in [67].

Decoding is the core operation of LF Processor. It unpacks and converts the original raw light field image data into several data structures, including a 4D image structure, metadata, thumbnails (if existing in the source) and other data, that is then organised into what we call a *Decoded Light Field* image, which may then be saved (manually or automatically, controlled through user Preferences) as a new file format (proposed in this work) and reused for further processing.

LF Processor automates some aspects of this procedure that LF Toolbox does not. For instance, the camera used to decode the image is automatically selected by the application, as are the folders where all the calibration data for that camera may be found and the WID. In LF Toolbox, that information must be explicitly passed as function parameters (with full paths for file names outside the expected paths) which is time-consuming and error prone.

NOTE: It is important to stress that **LF Processor is not just a GUI for the LF Toolbox**. Although some of LF Processor’s features use LF Toolbox functions, **most do not**. Besides that, the features based on LF toolbox functions are integrated into a whole new functional unit (software application), which automates repetitive tasks, performs validation of data and inputs up to a certain level, provides interactive information, allows visualisation of a light field image in innovative ways (e.g. Viewpoint Navigator), improves some algorithms (e.g. Colour Correction) and adds completely new functionality (e.g. Depth Filters engine).

The functions of LF Toolbox send feedback messages to the Matlab’s console, because they are not integrated into a GUI. LF Processor does have an integrated GUI and, in order to still supply feedback to the user while the operation is running, the code of LF Toolbox, where those messages were sent, **had to be**

changed to redirect them from the Matlab's console into the Status Bar of LF Processor. No other behaviour of LF Toolbox was changed though, and this has no impact on the time taken to process data. **This is the underlying reason why LF Processor uses a “custom-made” version of LF Toolbox.**

4.4.4. Process menu: Image (processing)

Since most of these operations require input parameters to execute, in order to simplify its usage, which may be repetitive, LF Processor allows the user to control two aspects of these operations: (1) If they are automatically run after the decoding process; and (2) which parameters should then be used automatically, or by default, in case of manual operation.

Figure 34 presents the Preferences screen, where the options mentioned above may be changed (default values are shown).

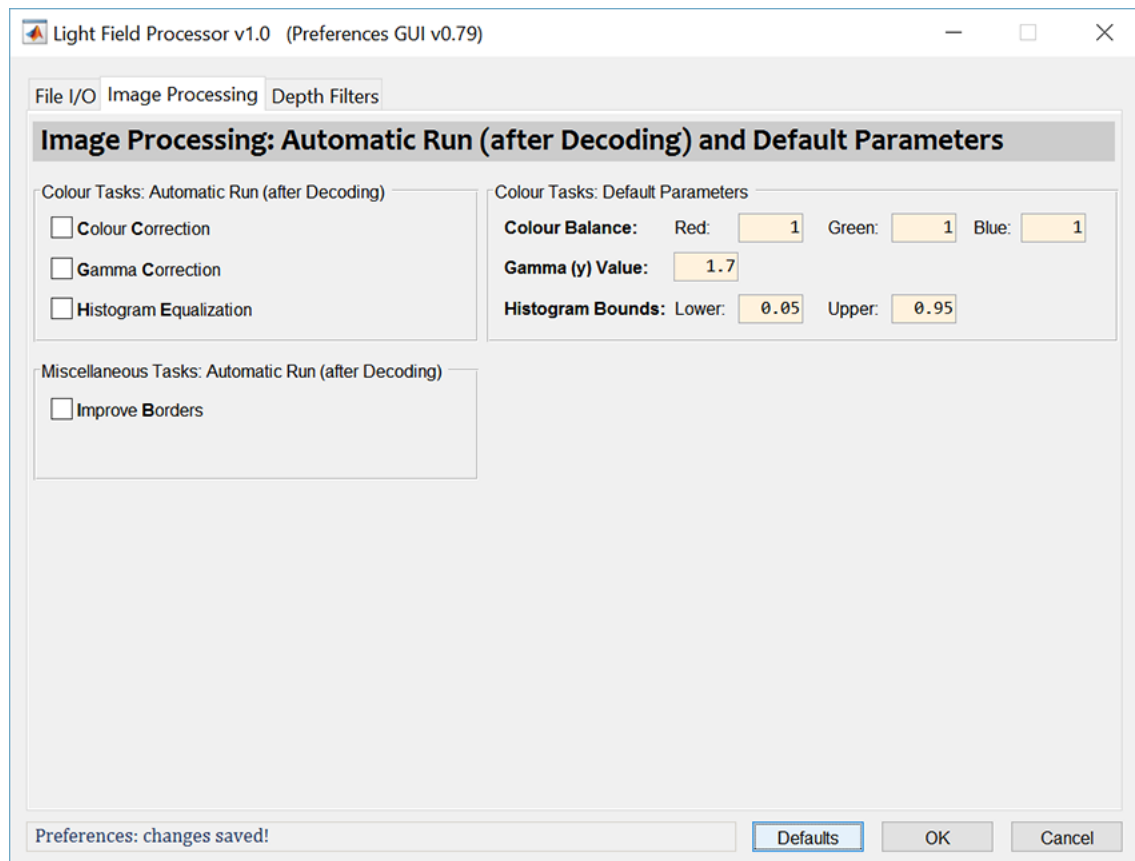


Figure 34 – Preferences: Image Processing options screen.

4.4.4.1. Colour Correction (CC)

A custom-made algorithm was done for this feature (for details see *Chapter 5*).

The *Colour Correction Matrix* (CCM) used as input is automatically embedded in all LFR file's metadata by the camera manufacturer.

Some empirical tests were made with several different parameters in a set of five light field images. From these tests, we concluded that our chosen CCM produces slightly more realistic colours than the CCM used by LF Toolbox.

Since the main focus of this work is not on this type of features, we will not present thorough results in the tests we conducted to reach this conclusion. Also, considering that the difference between the two CCMs was very dim, we saw no need to implement any selection mechanism between the two, on the application.

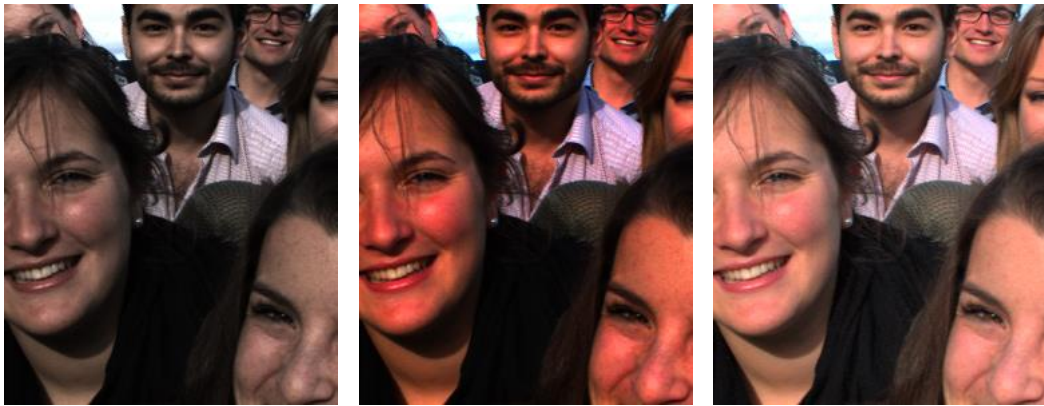


Figure 35 – Comparison of image processing tasks. Left: original DLF without any effects; Centre: applied CC ($R=1.00$, $G=1.00$, $B=1.00$); Right: applied CC+GC+HE+IB (all with default values).

This feature may be used independently from the Gamma Correction, which is something not possible to do using LF Toolbox since both are used in the same compact function [13].

For colour matrices theory and its applications, see [80], [81], [82]. For colour temperature theory, see [83].

4.4.4.2. Gamma Correction (GC)

Gamma Correction is a well-known operation used in electronics, to improve the visual response of screens, and image processing, as a method to manipulate brightness in an image [84].

In LF Processor the default (and recommended) input value for *gamma* (γ) is: **1.7**.

4.4.4.3. Histogram Equalisation (HE)

Histogram equalisation is another well-known image processing method to adjust contrast of an image using that image's histogram [85]. LF Processor uses the LF Toolbox function without any major changes, except for GUI related issues.

4.4.4.4. Improve Borders (IB)

As mentioned earlier, LF Processor uses the Light Field Toolbox to decode light field images (Section 3.4.1). During that process, one of the first things that LF Toolbox does is aligning the MLA micro-images into a rectangular shape (because in the Lytro Illum camera they are laid in a hexagonal grid). That is required for better handling of its data structures, but above all, to allow the individual storage

and processing of each micro-image through “rectangular” data structures, such as a 4D array (for details see [67]).

In such a geometrical conversion, problems will arise at the borders of the image, which in Lytro Illum imagery is especially evident at the vertical borders due to the physical organisation of the MLA, where some points will have only half a pixel or none at all. Those images may sometimes appear odd or “unfinished”.

To solve this problem, the solution implemented by Lytro Desktop is simply to remove the outermost lines of an image, which although being practical and visually efficient, has the disadvantage of removing pixels from the original image.

LF Toolbox does not attempt to solve this problem.

The *Improve Borders* option was developed to counter that displeasing “effect”. The method used here is based on well-known techniques described in [86]. It consists in copying the penultimate line (if counting from the centre of the image) in each direction, to the outermost line in that same direction. Figure 36 shows an amplified piece of the right border of a light field image; before (left side) and after (right side) applying this feature.



Figure 36 – Example of the Improve Borders (IB) feature.

4.4.5. Process menu: Depth Filters (engine)

Perhaps one of the most disliked activities of a researcher in image processing is to repetitively run the same algorithms in order to extract outputs in a systematic way, so that their results can be compared accurately. This is usually an arduous task that requires time, patience, accuracy and manual labour in taking notice of what was done and what still needs to be done.

4.4.5.1. Design goals

The *Depth Filters engine* was conceived to address the problem described above by minimizing its costs in terms of time and effort. For flexibility and scalability, the engine was designed so that it could be extended programmatically, in order to allow more filters and parameters to be included, as needed, for a particular task.

In LF Processor, this engine was developed to run systematic tests on algorithms that apply depth filters to a light field image, each with its own set of parameters.

Since our goal was not to implement the algorithms themselves, but rather to build an engine that could run them systematically, whilst still controlled by a user, we chose to use five depth filters already developed and known in the field, provided by the LF Toolbox.

Needless to say that **in order to use this engine, a user must also understand how the depth filters actually work**, including their goals and input parameters. For technical details about these algorithms see [87], [88], [13].

4.4.5.2. Options

The *Depth Filters* options include running each of the five filters alone and also one option that may run any of the previous filters cumulatively (one after the other) and therefore merge their outcomes visually (selected option in Figure 37).

The *Depth Filters* submenu options are:

- Shift Sum (Planar Focus)
- 2D Frequency Vertical
- 2D Frequency Horizontal
- 4D Frequency Plane
- 4D Frequency Hyperfan
- Run Cumulatively

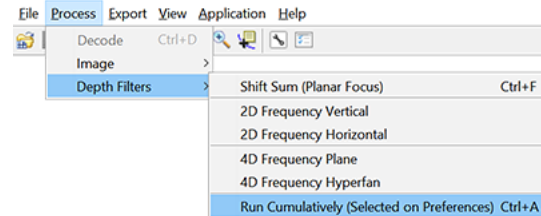


Figure 37 – *Depth Filters* submenu options.

4.4.5.3. Usage

As mentioned above, each of the depth filters' algorithms has its own set of input parameters, to which the engine adds a few more of its own, to control the automatic running of the algorithms (in *Series*).

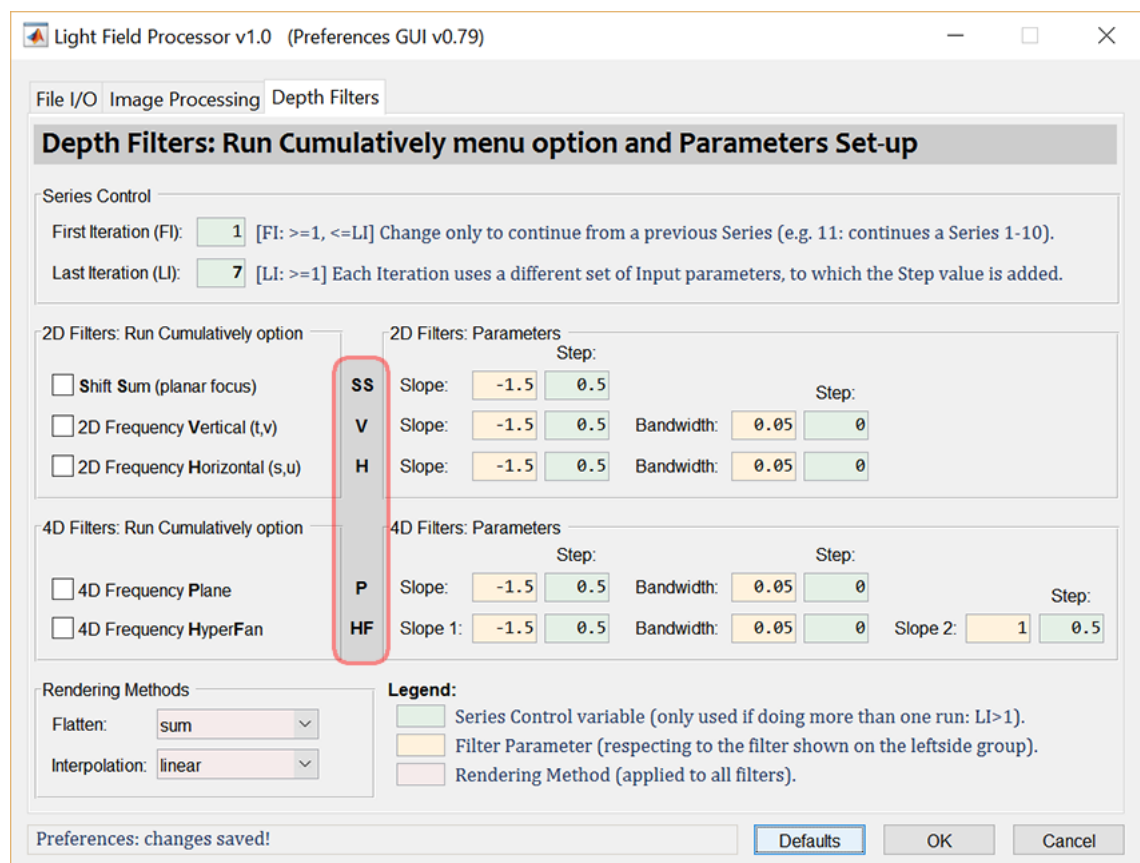


Figure 38 – Preferences: *Depth Filters* options screen; (each operation's initialism is highlighted with a red border).

Therefore, it would be impracticable to constantly require a user to input all those values before each execution. We solved that problem (perhaps in a somewhat innovative way, considering application user interfaces) by creating a screen in the

application Preferences so that the user may manage all those parameters in a compact and convenient way, as shown in Figure 38.

The parameters for each filter are then passed automatically, by the application, to the required algorithms, thus avoiding the need for the user to constantly input or even validate them. Since all the user preferences data is stored in an internal application file, those parameters are permanently available even after closing the application, until the user changes them again.

Figure 39 shows the Depth Filters engine in action, running a series of the *Shift Sum* filter. During execution a pop-up window appears to show the last processed image and also its input data (above) that will also be used for the file name.

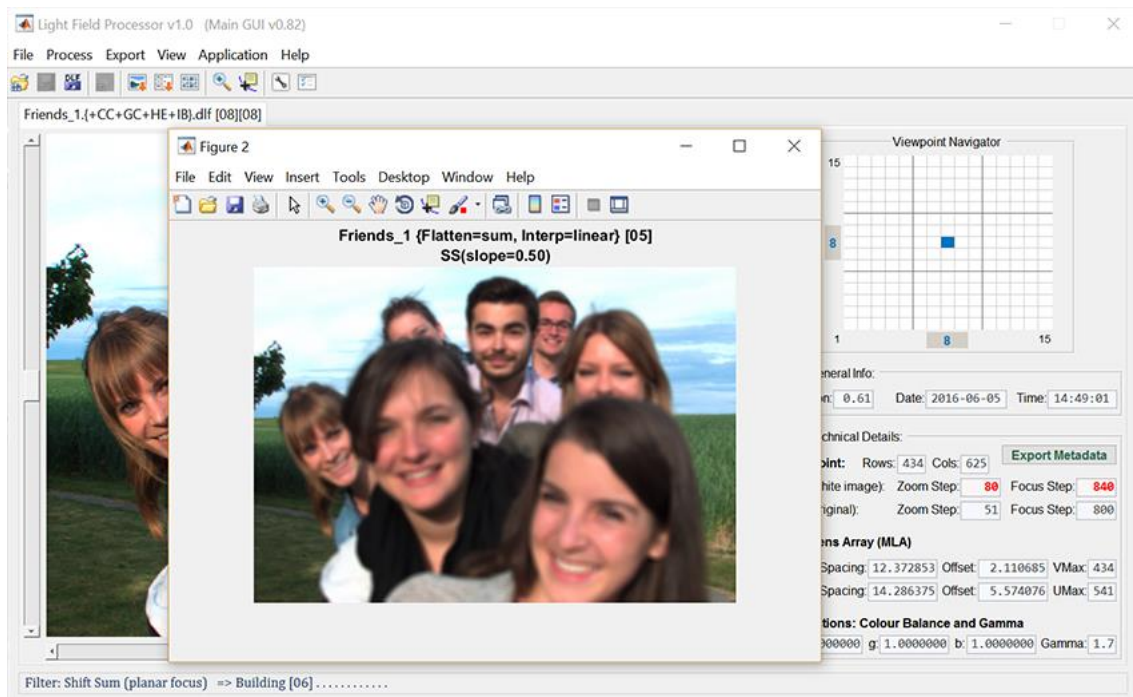


Figure 39 – Depth Filters engine: running the *Shift Sum* filter
(the front image is defocused as a direct consequence of the application of the filter).

4.4.5.4. Automatic naming scheme of output files

Since the output of each run of these algorithms is one image file, it is convenient that they are named in an orderly and self-explanatory fashion. So, another useful aspect of this engine is that it addresses the requirement mentioned above by automatizing the file name generation of those output files.

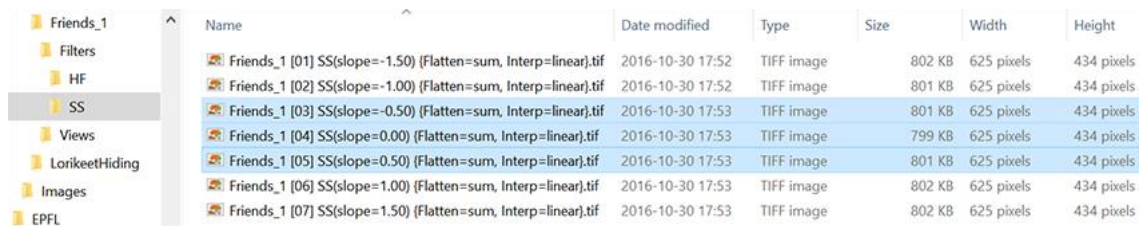
The logic used for this naming scheme is similar to that mentioned in *Section 4.2.3* and *Section 4.3.4.2*. However, since the data structures are different, the schemes had also to be different.

Initialisms were assigned to Depth Filters so that they could be used to shorten the file names, which would otherwise be too long (Figure 38 highlights them in red).

The following is the list of all Depth Filters initialisms used:

- Shift Sum (Planar Focus): **SS**
- 2D Frequency Vertical: **V**
- 2D Frequency Horizontal: **H**
- 4D Frequency Plane: **P**
- 4D Frequency Hyperfan: **HF**

In Figure 40 it is possible to verify, in the folders column (left side), that the DLF filename is “Friends_1” and that inside the “Filters” folder there exists another folder for each of the Depth Filters algorithms already executed at least once, in this case: **HF** (*Horizontal Frequency*) and **SS** (*Shift Sum*).



Name	Date modified	Type	Size	Width	Height
Friends_1 [01] SS(slope=-1.50) {Flatten=sum, Interp=linear}.tif	2016-10-30 17:52	TIFF image	802 KB	625 pixels	434 pixels
Friends_1 [02] SS(slope=-1.00) {Flatten=sum, Interp=linear}.tif	2016-10-30 17:52	TIFF image	801 KB	625 pixels	434 pixels
Friends_1 [03] SS(slope=-0.50) {Flatten=sum, Interp=linear}.tif	2016-10-30 17:53	TIFF image	801 KB	625 pixels	434 pixels
Friends_1 [04] SS(slope=0.00) {Flatten=sum, Interp=linear}.tif	2016-10-30 17:53	TIFF image	799 KB	625 pixels	434 pixels
Friends_1 [05] SS(slope=0.50) {Flatten=sum, Interp=linear}.tif	2016-10-30 17:53	TIFF image	801 KB	625 pixels	434 pixels
Friends_1 [06] SS(slope=1.00) {Flatten=sum, Interp=linear}.tif	2016-10-30 17:53	TIFF image	802 KB	625 pixels	434 pixels
Friends_1 [07] SS(slope=1.50) {Flatten=sum, Interp=linear}.tif	2016-10-30 17:53	TIFF image	802 KB	625 pixels	434 pixels

Figure 40 – Depth Filters engine: generated folder structure and file naming scheme.

The complete file names of the output images are also visible in the files column (right side) of Figure 40. Those names are automatically generated by the application in a systematic and ordered fashion to make the files easily identifiable, including not only their order of execution but also the parameters used to generate them.

An example for a complete file name of an output image may be the following: **“Friends_1 [03] SS (slope=-0.50) {Flatten=sum, Interp=linear}.tif”**.

This file name informs that the output light field image is “Friends_1” and it is the third iteration in this *Series*; the filter applied was the *Shift Sum*. The numeric input parameter was Slope=-0.50. The qualitative parameters inputted were: “Flatten method” and “Interpolation”, with the values “sum” and “linear”, respectively. Finally, this image was saved using the TIFF standard format.

More examples of file names using this structure may be seen in Figure 40.

A piece of each of the three selected files (in cyan) in the files panel (right side) of Figure 40, are shown in Figure 41 as an example of applying the *Shift Sum* depth filter to a decoded light field image. This filter changes the plane of focus.

The series shown in Figure 41 makes it easier to perceive how the *slope* parameter affects the image as it grows. In a series, each *Iteration* represents a single run of the filter using a specific set of parameters.

As may be seen in Figure 40, this series had seven iterations. It was run with a +0.50 growth of the *slope* parameter, per each iteration. The results show that, in

this case, the focus starts on a closer plane (left) and then it shifts to planes further away from the camera (right).

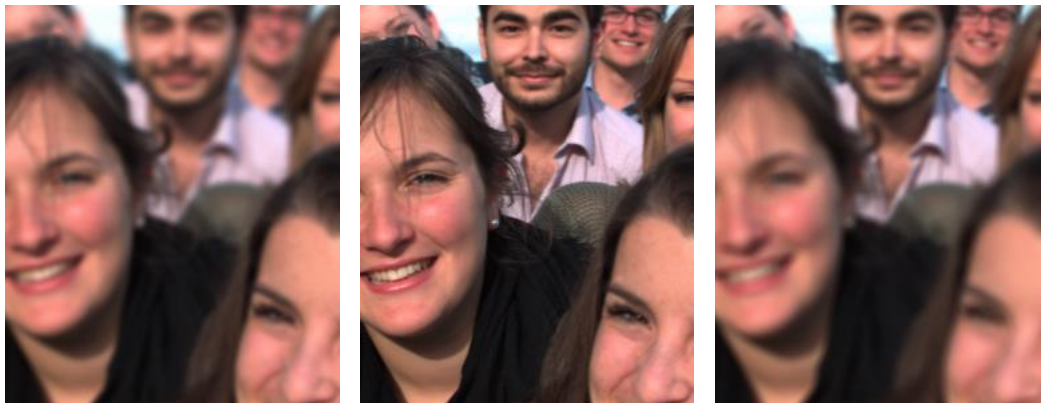


Figure 41 – Results of applying the *Shift Sum* filter to DLF file “Friends_1”, with a growth of +0.50 per iteration of the slope parameter.

Left: [03] SS(slope=-0.50); Centre: [04] SS(slope=0.00); Right: [05] SS(slope=0.50).

4.4.6. Export menu

The *Export* menu encompasses all the outputs of LF Processor that are the direct consequence of data extraction or processing from a light field image file, and may assume the form of an image, video or text file.

4.4.6.1. Types of Views

In LF Processor there are three types of Views (or Viewpoints): *Empty*, *Very Dark* and *Normal*.

Empty views contain no actual valid data (only zeros) and were created artificially, during the decoding process, for geometrical alignment and data structure storage compatibility reasons.

Very Dark views, as the name implies, although do contain valid data, their samples may be considered as too dark, and therefore not suitable for some applications (e.g. video frames in *Video Sequence* options). This happens due to the effect of *Vignetting* (for a definition, see *Section 3.4.8*; to learn more about some of its effects in light field imaging, see *Section 3.4.3*).

Since the corner viewpoints (2D images) of a light field image are built from the samples gathered from the edges of microlenses, they “inherit” those same dark properties.

By exclusion of parts, **Normal views** are considered to be all the other views, which are neither *Empty* nor *Very Dark*.

Since it may be undesirable to use *Empty* or *Very Dark* viewpoints, some control options (explained on screen) were implemented in the application Preferences screen, especially made to control File Input/Output operations, as shown in Figure 42.

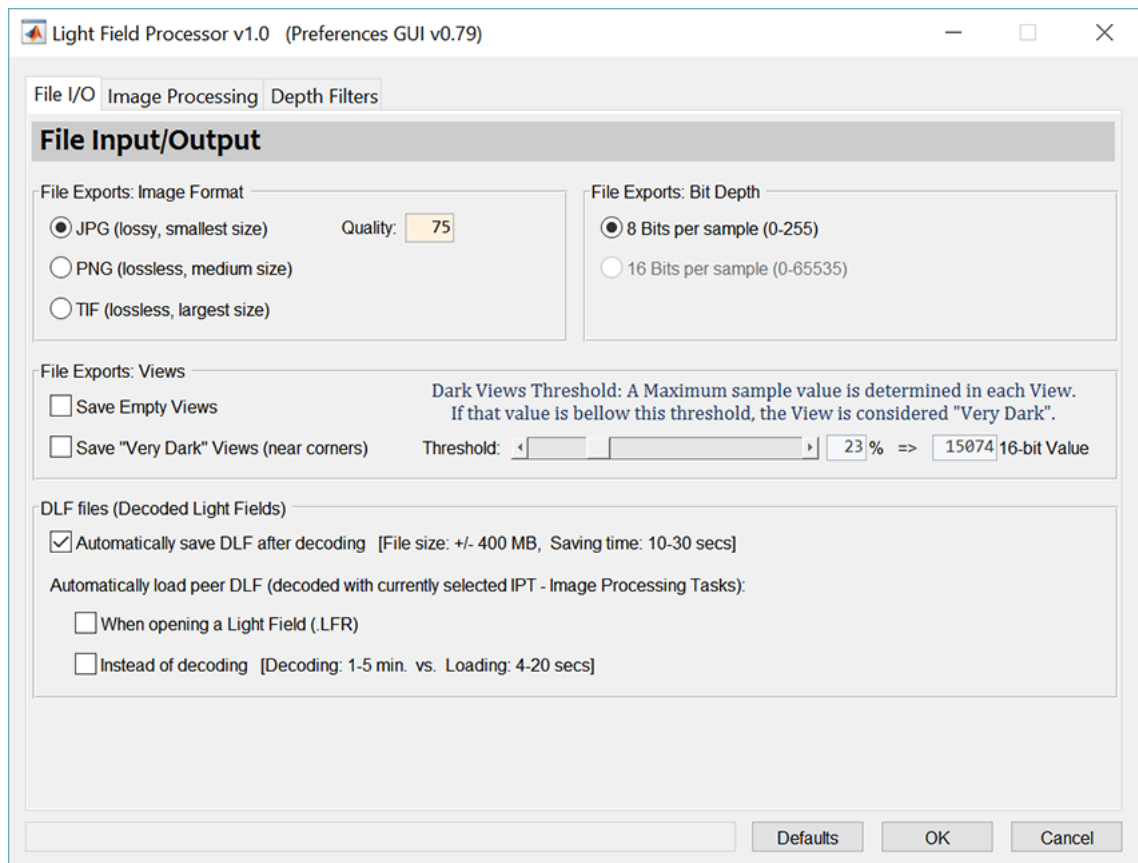


Figure 42 – Preferences: File Input/Output options screen.

4.4.6.2. Viewpoint, Views Map and Microlenses Map

What these operations refer to, as well as their outputs, was already explained previously in *Section 3.4.2*.

LF Processor allows Viewpoints (or any other 2D image file) to be saved in three different image file formats, controlled by the user in Preferences: *jpg* (with adjustable compression rate), *png* and *tif*, in 8 or 16 bits of depth, as shown in Figure 42.

The spatial dimensions of a Viewpoint are usually formed by **434** rows and **625** columns. In some rare cases there may be only 433 rows (only found in one case, amongst around 25 LFR files used, from three different cameras), presumably because it was not possible to clearly recover or identify one row from the MLA (top or bottom) during the decoding stage. When the dimensions of a DLF are not the usual, they are shown in red colour, in their respective field in the data panel.

4.4.6.3. All Viewpoints

Exporting *All Viewpoints* is an operation that saves to disk all the available viewpoints from the active DLF image, according to the user Preferences options, as explained above.

Those files are saved in a subfolder called “*AllViews*”, inside the currently selected light field image folder, for a better identification and organisation.

4.4.6.4. Thumbnails and Metadata

These operations are already explained in *Section 4.1.1* and *Section 4.1.2*.

4.4.6.5. Video Sequence

This is a curious set of operations developed to generate and save a single video file, in the AVI format, usually with 15 fps (to be in synch with the maximum number of different views in a light field image, in any direction) and the dimensions of that DLF file's Viewpoint (usually 434 x 625, as mentioned in *Section 4.4.6.2* above).

It encompasses a custom-made algorithm to generate a path, within micro-lens dimensions (15x15), and then “follow” that path, by showing the currently selected viewpoint on screen, in real-time, and then adding it, as a frame, to the video that is being generated in the background.

The path that each operation follows is indicated by its sub-menu option label, which should be self-explanatory. By following those paths, the intrinsic parallax of a light field image, in that direction, may immediately become perceptible.

The video file is saved automatically, upon finishing the operation, in a subfolder called “*Videos*”, inside the currently selected light field image folder, for a better identification and organisation. Its name will contain the name of the light field image, the type of path followed and, when it applies, in which column or row.

4.4.7. View menu

These are not exactly operations, but instead modes of visualisation of the information on screen. Their usage was already explained in *Section 4.3.2.3*.

4.4.8. Other operations

4.4.8.1. Preferences and Configuration

The Preferences screen has three tabs, which contents and options were already explained in the sections regarding their data, so there is no need to repeat that information here.

The same applies to the Configuration screen. In case a user wants to know more about how to install the application, please see *Appendix A*.

4.4.8.2. Auto-check if application folders are missing

Upon start, LF Processor automatically checks if any of the application folders is missing. If the root folder is missing, then the Configuration procedure is automatically launched in the same way as if it was the first time the application is run. If only a sub-folder is missing, than it is recreated in the operating system using the root folder as base, transparently for the user.

CHAPTER 5

DEVELOPMENT OF LF PROCESSOR

"There are 10 kinds of people in this world: those who understand binary, and those who don't." ☺

A joke extracted from [89]

Using or watching an application being executed, is just one side of the “coin”. The other usually remains “unseen” by most: its development. This chapter is focused on providing the usually invisible side of software development. From the perspective of the project manager, to that of the software developer, which in a small scale project like this, is often the same person.

Needless to say, the perspective presented here is geared towards computer science and software engineering. Therefore, some understanding of software development theory will be required to better understand discussions regarding tools, methods and strategies used. Moreover, to understand the sections concerning actual code, some knowledge of Matlab (or any C++ language derivative) and Object Oriented Programming theory, will also be important.

Here may be found details about the application development such as: its stages of development, the underlying architecture of its major components, explanations of some of the challenges found and how they were solved in terms of programming techniques and the tools used.

Since the outcome of this work was not focused on computer science itself, but rather in using it to produce an application with features capable of performing image processing with light field images, not much time was left to detail the huge amount of work that was done, and why, to develop it.

Nevertheless, some of the most important decisions and developments made in this work are mentioned superficially in this chapter, relating exclusively to the realm of software development, although we will not go into actual code details or their direct interactions.

5.1. Software Development Methodology

As mentioned before in chapter one, the decision to use the iterative *mini-Waterfalls* model proved right since the beginning due to its flexibility in a dynamic scenario, full of “unknowns”, just like the one in which this work was made. Figure 43 presents a diagram of this model containing details of the work done in each stage of the process.

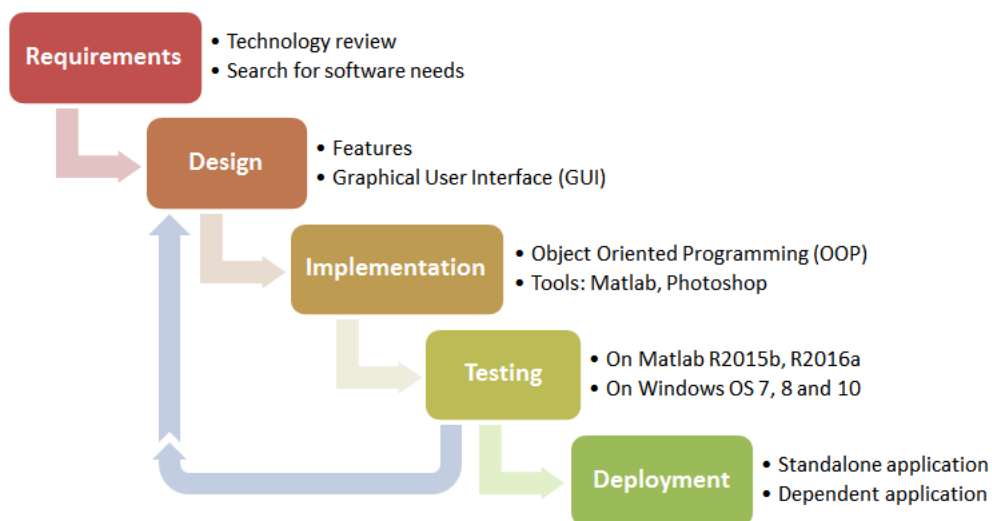


Figure 43 – Detailed software development methodology used: Iterative mini-Waterfalls.

5.1.1. Requirements

In a real or professional software development environment, this stage is used in “*determining the needs or conditions to meet for a new or altered product or project*”, as mentioned in [90]. So, strictly speaking, in a project of this scale and nature, this would mean, essentially, naming the tools, products or devices that are needed to run the application, because nothing else applies.

Therefore it seemed to make sense to also include the “requirements” for the developer himself (e.g. having to learn the higher-level knowledge to develop the application’s features, which is presumed already existent in a real-world scenario, but not in this particular case), as well as other aspects more concerned with the development stages and how they were conducted, rather than the application itself.

Since this is a relatively new field of study for which the developer of this work has had no formal training before, it was only natural that the very first thing to do was investigating about light field image theory and related technologies, especially within the academic community literature.

At ISCTE-IUL we have at our disposal both a Lytro F1 (first generation) and a Lytro Illum Lytro (second generation) light field cameras. So, the most obvious thing to do was to take advantage of that, which is why the Lytro Illum became the choice of camera around which this work was based. The excellent work done by Donald Dansereau, with his Light Field Toolbox, also contributed to that decision (that will be the subject of another section in this chapter).

Knowing the basic theory about light field imaging, having a target camera and a reference software tool with which to work, it was then time to search for other software tools or frameworks that could be used to accelerate the development of such an application in the limited time available in a project of this nature.

Several tools were reviewed, but unfortunately most were either outdated, for other camera systems, for the Lytro F1 (which was incompatible with Lytro Illum, our choice) or simply incomplete isolated pieces of software. Some of those examples may be found in [11], [91], [92]. Although certainly useful by their own, these tools were not useful for this work. Apart from those type of tools, there were the professional solutions such as the Lytro Desktop [12] which could only be used as reference for certain operations, because it was done with a totally different propose than our work.

By this stage there were not many choices available, in terms of tools (as detailed in *Section 5.4*), but since the Light Field Toolbox was able to decode raw light field images from Lytro Illum cameras, it became obvious that by using it, this work would at least benefit from not having to worry about that “low-level” interface and could instead concentrate on higher level features, such as extracting 2D images and then showing and building new information around them.

Considering what was learned so far, it was defined that the application should target mainly the academic community. Since Matlab is a popular tool in the engineering fields of study, it should also be familiar to most of the target users.

It was also decided to use a Graphical User Interface in order to make it more appealing to those who might be interested in the technology, but may not have the time or skills required to go through a more technical approach, like the one used by the Light Field Toolbox, which requires some programming skills.

A few higher-level features were initially defined, only to serve as reference: read and present the contents of a raw light field image and save those images in standard formats.

As the project grew, more features were introduced; some as requirements to fulfil other requests (e.g. the cameras management system was a requirement to allow “transparently” decoding any LFR file), some to fulfil operational needs (e.g. DLF format) and others came as unexpected but interesting features that should be added (e.g. Depth Filters engine).

5.1.2. Design, Implementation and Testing

After designing and developing the first prototype versions, it became increasingly obvious that it would be very difficult to manage the code using the “standard scripting model” of programming that Matlab uses.

For instance, the dreadfully famous “*spaghetti code*” suddenly was everywhere because Matlab’s scripting model does not allow calling another function on an external file (.m), unless that function has the same name as the called file, which in practice means that only one function per file may be reused. That restriction makes managing code for a project that requires constant changes, and therefore flexibility and scalability, nothing short of a nightmare.

Fortunately, the company responsible for Matlab development (MathWorks) has been implementing and updating an Object Oriented Programming (OOP) model since version R2014b. So the decision to shift the programming paradigm became obvious and inevitable.

In spite of having very little documented support and being very awkward in some technical matters, Matlab’s OOP model behaved fairly consistently throughout the remaining development and proved to be a very good choice. That decision implied the redesign of the application and some refactoring, time during which the application graphical interface engine was developed.

Testing was done permanently after each development and more thoroughly before each presentation with the advisor, who also played the “role” of the “client” in this scenario.

As the application acquired more features, increasing care had to be taken with the interface and the access to those features. To make development more manageable and keep everything synchronised (one of the downsides of iterative methods is that they make this task more laborious), a custom-made engine was developed to automatically manage and keep menus, toolbars, shortcut keys and classes in synch, which involved the creation of a few more classes (see Figure 47).

New icons for the toolbar and images were created as needed, using Photoshop.

5.1.3. Deployment

There were a few distinct landmarks in the deployment stage throughout this development: before and after the OOP paradigm was fully implemented, and also before and after the compilation methodology was used. The latter allowed the

creation of two different versions of the application: the *Dependent* (running only within Matlab) and the *Standalone* (compiled and running directly under the operating system). These concepts are explained in more detail in *Appendix A*.

The first serious deployment tests were successfully performed on Windows 8 and 7, as part of the testing for the development of the *Standalone* version of the application (for Standalone version details, see *Section A.1.2*).

From then on, the project development got back to the iterative part until the very end, when the final version of the application was prepared for distribution with this work.

5.2. Application Architecture

To better convey an idea of the major building blocks of LF Processor, it will be presented from two different views.

First, an “*external*” view, mainly concerned with the operational side of the application, which is that of a regular user, or of someone who sees the application being used or demonstrated (such as a reader of this work). This is done to provide a “bridge” between the image processing perspective (more user oriented), presented in the previous chapter, and this one.

Second, an “*internal*” view, regarding the building blocks of the application itself, used by the developer to make this a software project.

5.2.1. External architecture

LF Processor requires either an LFR or a DLF file as inputs to do most of its operations. But, in case the input is a raw light field image, it also needs the calibration data of the camera with which that picture was taken in order to be able to decode it, and consequently, transform it into a DLF data structure.

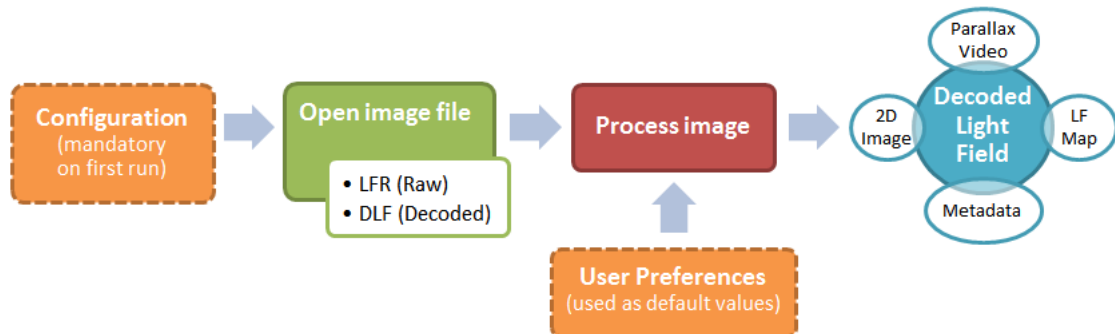


Figure 44 – External architecture of LF Processor, using a simplified approach.

This requirement exists because, as mentioned before (*Section 4.2.5*), in order to interpret the data contained inside a raw light field image, the decoding engine must know certain camera parameters, such as the zoom and focus used to take the picture. As a consequence of this “low-level” requirement in order to actually

read the source data of a light field image, a mechanism was needed to keep a list of cameras and their calibration data, ready to be used when needed (Section 6.1.1.5). For *decoding* and its integration into LF Processor, see Section 4.4.3.

Figure 44 shows the *external* architecture of the LF Processor application. Some aspects of it are simplified, such as outputs and the processing that may be done, but includes the use of “external” entities, such as Configuration and Preferences, that although not being part of the focus of the application itself (image processing) are still critical to support that purpose.

5.2.2. Internal architecture

The internal architecture of LF Processor is composed of five distinct groups of data, as may be seen Figure 45.

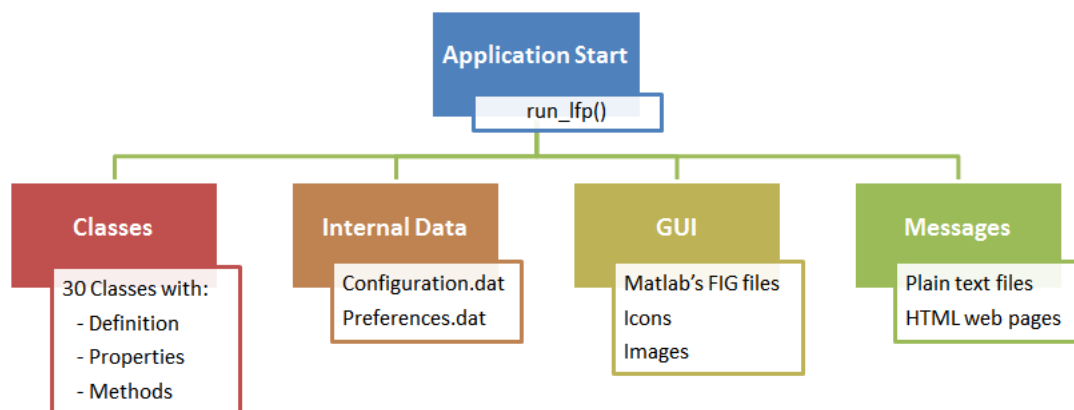


Figure 45 – Internal architecture of LF Processor.

The architecture of the project is reflected in the organisation of the structure of folders used by it to save the different types of files used, as shown in Figure 46.

As may be understood by seeing Figure 45, which diagrams the *internal* architecture of LF Processor, the development paradigm was totally object oriented, for the exception of the start file, which in Matlab must have the classical functional orientation.

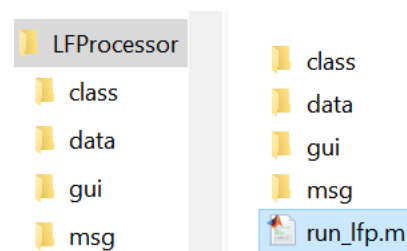


Figure 46 – Folder structure of the project's files.

5.2.2.1. Classes

Each class definition is included in its own file (.m) inside a folder named “class”, as required by Matlab [93], in order for it to interpret these files as part of a class, and not just a standard script file.

There are 30 different *Classes* in the project, each with its own *Properties* and *Methods*.

Some of those classes are just a simpler and more practical custom implementation of *Enumerations*, which in Matlab's OOP model use a huge amount of data considering what their purpose is, which makes them use much more memory and also to be slower to process than is usual in a programming language (explaining this would require low-level coding and some practice, so it is out of the scope of this work to enter into details about it).

Some examples of classes that are used as enumerations are: Colour, FileExt and FileType, which purpose should be self-explanatory.

Some other classes are used as more powerful data structures. A good example of that is the Class Action, which is used as the basis for the *Menus Engine*, that automatically synchronises menus (and their options), toolbars and their calling methods. Figure 47 shows a glimpse on some of the code from the Action Class.

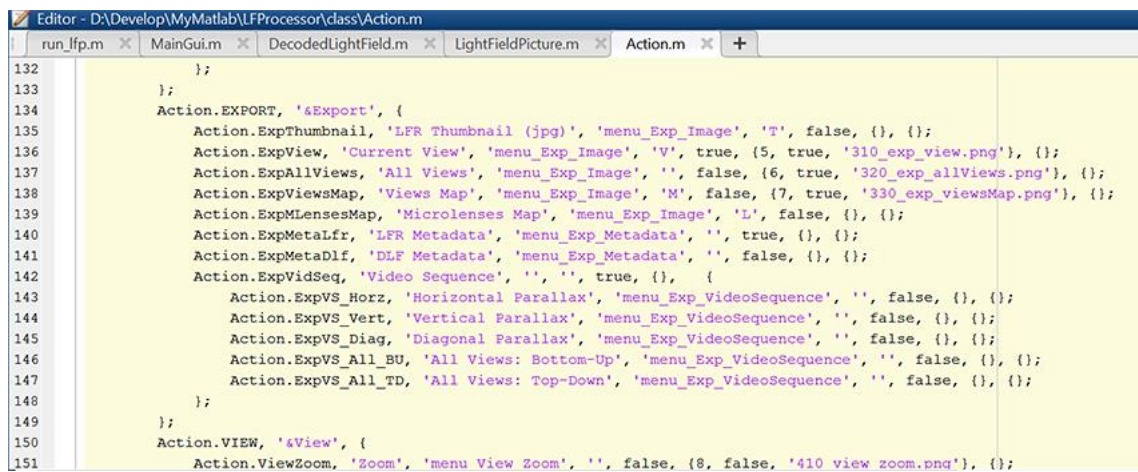


Figure 47 – Example of code from the Action Class, used mainly as a powerful data structure.

5.2.2.2. Internal data files

There are two files that support the operations allowed through the application Configuration and Preferences menu options, already discussed throughout *Chapter 4*. For obvious reasons they were named “Configuration.dat” and “Preferences.dat”, and are both kept inside the folder “data”.

One important aspect of the management of these files, by the application, is that if any of them is found missing during the application launch phase, some action occurs to regenerate the file, depending on what file is missing, as was already explained in *Section 4.4.8.2*.

5.2.2.3. Configuration data file

The Configuration file was a requirement to implement a system that automatically identified and selected the right camera calibration data and supplied it to the decoding engine, so that LFR files could be decoded just by user selection (which something not possible with LF Toolbox), and thus making this process truly user friendly.

Although this seems quite a natural thing to ask from an application, due to the nature of LFR files and how they are decoded, it was actually a very arduous and relatively complex development to be made. At least we did not have to create and manage the White Images Database, since that was already being done by LF Toolbox. That is the reason why that file is shown apart from the actual configuration file, in Figure 48.

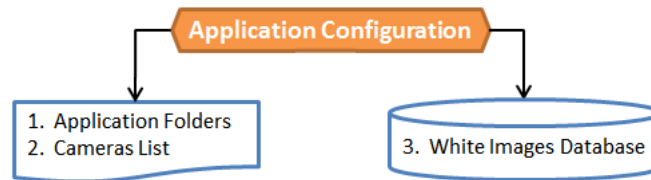


Figure 48 – Diagram of the application Configuration data structures (numbers are respective to the tab used in the Configuration screen).

The two items that are actually stored in this file, and are directly managed by application, are the *Application Folders* and the *Cameras List*, which data may be inputted by the user through the screens shown in Figure 22 (Section 4.2.1.1) and Figure 49, respectively.

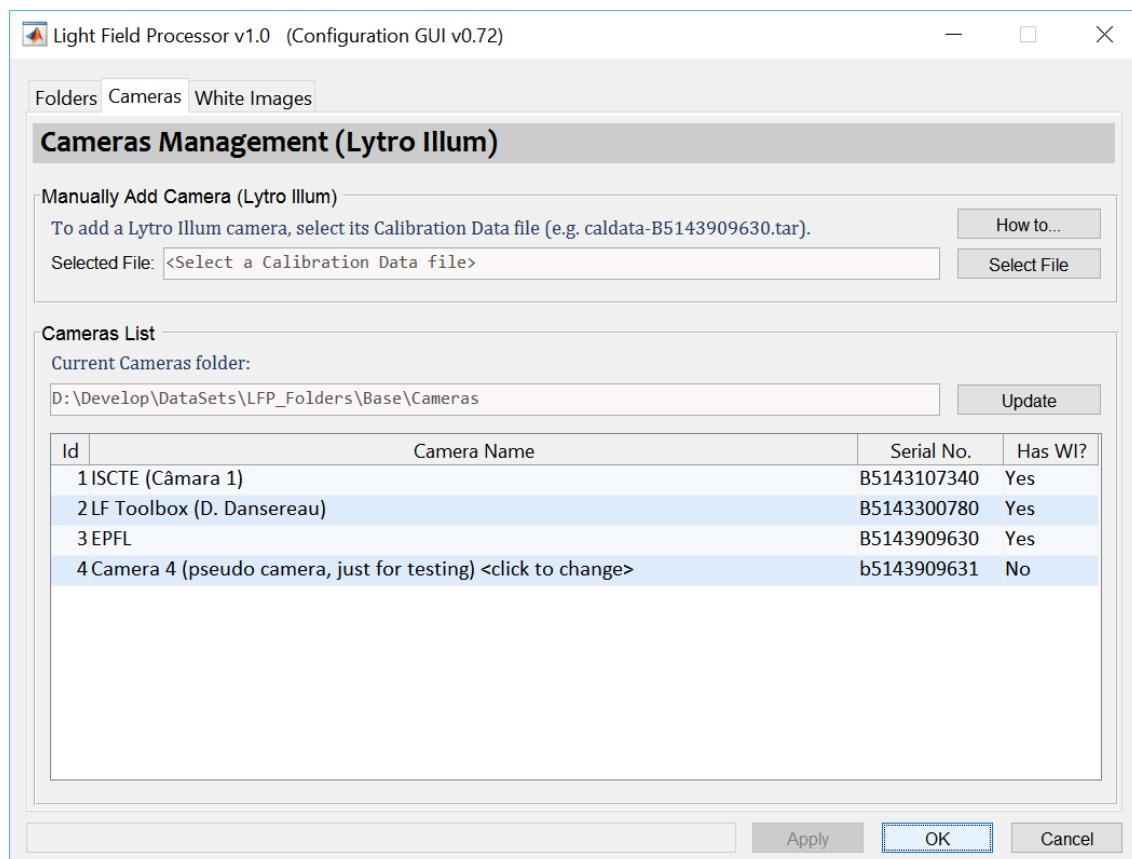


Figure 49 – Application Configuration: Cameras Management screen.

If this file is corrupt or missing during the application launch phase, it will be automatically and transparently regenerated with default values, without the user even noticing that. This may happen for numerous reasons, such as when a user

moves the installed application from one computer to another; renames its folder; or simply removes it by accident.

5.2.2.4. Preferences data file

Due to the repetitive nature of many of the tasks that the application may perform, it was decided early on its development two things: (1) that all those tasks should be identified; (2) a means to simplify their use and automate their selection should be provided to the user. Such system involved creating the Preferences.dat file, which is stored inside the “data” folder.

The information stored here is used by most of the application’s operations and is divided in three main areas:

- *File Input/Output* data (Figure 42, Section 4.4.6):
for instance, for all export file formats;
- *Image* processing parameters (Figure 34, Section 4.4.4):
it also stores options (On/Off) to decide if they are to be executed automatically after decoding;
- *Depth Filters* parameters (Figure 38, Section 4.4.5.3):
also saves data relating to the series management.

An overview of this file’s structure is shown in Figure 50.

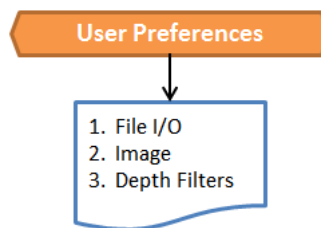


Figure 50 – Diagram of the application Preferences data structure (numbers are respective to the tab used in the Preferences screen).

5.2.2.5. Graphical User Interface

Developing the GUI for LF Processor ended up being perhaps the most technically challenging aspect of the development. Matlab is a good tool to develop relatively short scripts that do not require much integration. However this work required a greater level of integration and scope.

Matlab uses GUIDE as its Visual Designer for graphical components, a tool that generates “figure” or simply “fig” files, due to their extension (.fig). These misleadingly named files are actually used to store screen layouts. In spite of its limitations, this tool was certainly helpful in designing the graphical interface.

Another innovative system that was developed was the **mechanism for hiding and showing tabs** in a Matlab screen. It was used to dynamically add new tabs as the user progressed through the steps of Configuration (Figure 22, Section 4.2.1.1).

In terms of pure Graphical User Interface, the most innovative development was the **Viewpoint Navigator**, described in *Section 4.3.4.1*.

Neither GUIDE nor “Fig” files were used in its development. It was completely programmed as a custom graphical component. The movement of the mouse and the real-time synchronisation between the position in the grid (blue dot) and the image shown on the left (image panel) was developed in this project.

At the time of development, nothing similar to that interface was known to exist to perform the same purpose.

Matlab system was not intended to be used programmatically and incorporated into Classes, as we did. Finding a way to use the “best of both worlds” proved to be hard, but a **custom solution was developed to allow merging the OOP paradigm with the GUIDE exported “Fig” files**. That system proved to be a very good investment of development time since it was widely used in the application and saved many hours in what would otherwise have been a laborious “manual” work, prone to error, which would have required constant copying and maintenance of huge “chunks” of code.

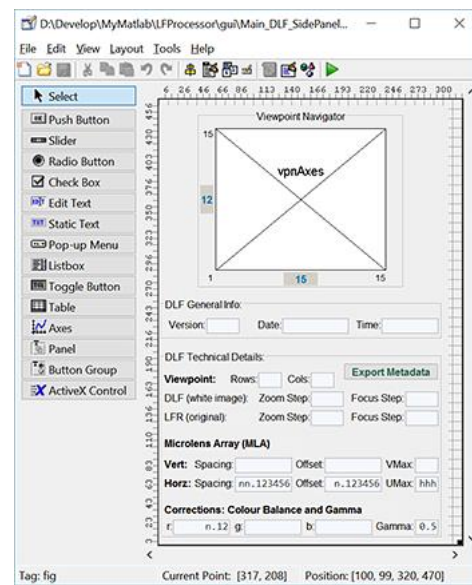


Figure 51 – Using GUIDE to visually design the DLF data panel.

The **system to allow the creation of screens with graphical components from more than one “Fig” file (used in side by side panels shown in the LFR and DLF screens)**, was also custom made, because Matlab’s GUIDE system does not allow embedding or splitting the layout of a fig file into two separate files, which is what was done to build those screens. Figure 51 shows GUIDE in editing mode, with the DLF data panel opened. It is also noticeable that there is no image loaded for the Viewpoint grid. That space is reserved here so that later it is completely drawn in real-time, programmatically.

Unfortunately due to space and time constraints it is not possible to detail much more of the solutions developed.

5.2.2.6. Messages

Two systems for sending messages were developed: one for internal usage (the Status Bar), and another one which used text coming from plain text files. The latter was used in the *About* window of LF Processor and also during the Configuration pop-up boxes.

The former was a complete custom system for managing messages which included their properties, such as:

- **Type:** Error, Success or Warning;
- **Sound file** associated with this type of message: will be automatically played upon showing that message;
- **Output Channel:** allows this system to transparently send messages to either the Matlab console or any graphical component. In fact, it sends messages to different Status Bars, since each window has its own Status Bar (examples are: Main window, Preferences and Configuration screens).

5.3. Programming

5.3.1. DLF data structure

The DLF file usefulness and advantages was already discussed throughout this work. Here, we do a short description of its internal data structure. Figure 52 shows the usually “invisible” side of a DLF file, as it really is, if viewed inside the Matlab Integrated Development Environment (IDE).

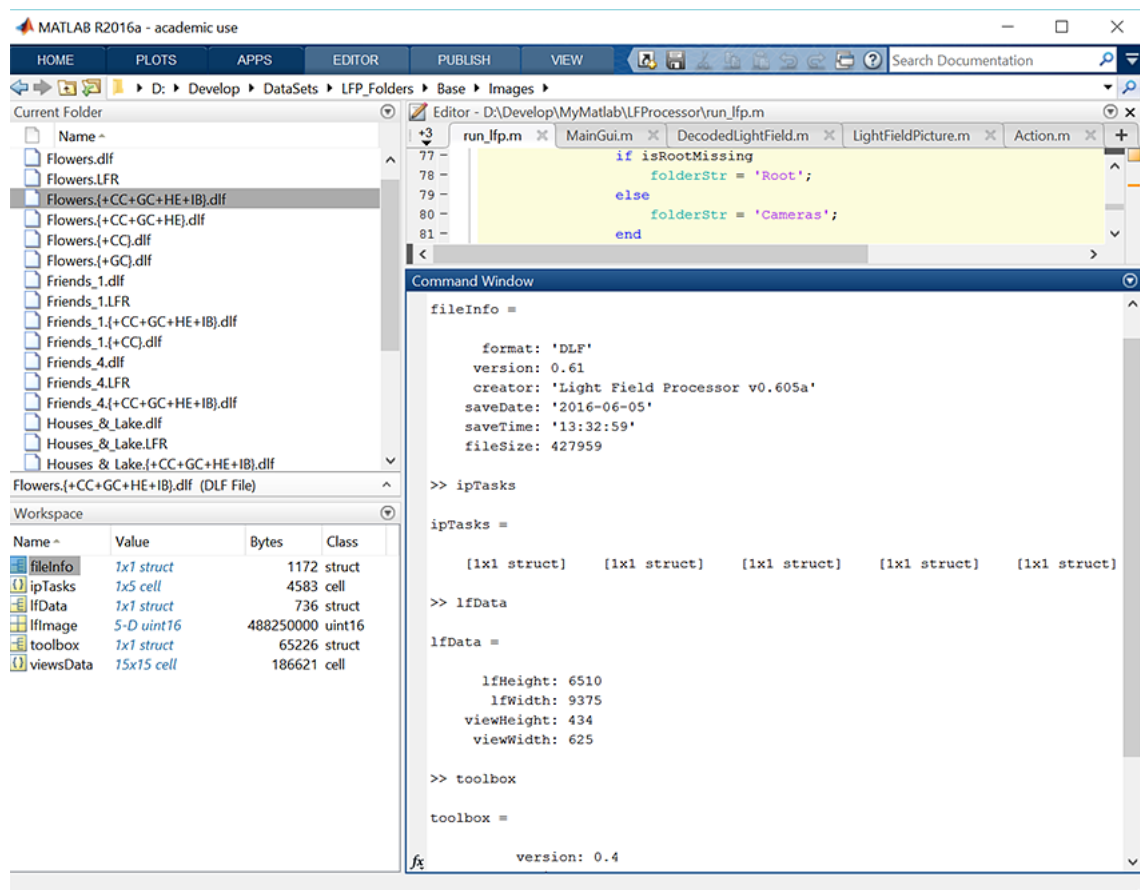


Figure 52 – Workspace panel (left): Top level of the internal data structure of a DLF file. Command Window (right): contents of some DLF fields.

There are six top level variables saved in this structure, as may be seen on the left panel of the Matlab IDE, in Figure 52.

The variable containing the actual light field image is: *lflImage*. This structure is the same one inherited from Light Field Toolbox. Although some of its fields were improved in DLF and others are not used, we opted to maintain it in its original form for compatibility with LF Toolbox, which allows saving the data of a light field structure, after being decoded, into a standard Matlab data file (.mat).

This means that, technically, a .mat file directly generated by LF Toolbox v0.4 (the one used in this work) and containing a light field image, may be imported and transformed into a DLF file. However that feature was not yet developed, but is instead suggested as future work.

```
ipTasks =
    [1x1 struct]    [1x1 struct]    [1x1 struct]    [1x1 struct]    [1x1 struct]
>> ipTasks{1}
ans =
    isOn: 1
    name: 'Colour Correction'
    initialism: '+CC'
    ccm: [3x3 double]
    cb: [1 1 1]
    cct: 0
>> ipTasks{1}.ccm
ans =
    2.482781    -0.367615    -0.18722
   -1.101808     1.666777    -0.733172
   -0.380973    -0.299162     1.920393
```

Figure 53 – Matlab’s console showing a DLF file *ipTasks* variable and its contents.

As shown in Figure 53, when the property *isOn* = 1, it means that this operation (identified by the property *name*) was applied to this DLF file. The *ccm* property contains the actual CCM used to apply the Colour Correction filter.

5.3.2. Paradigm adopted

The fact of using an Object Oriented Programming paradigm surely minimised certain difficulties and made the application much more flexible and scalable, besides becoming much more readable, for a programmer.

Still, this application grew in size much more than it was initially foreseen, as can be understood by looking at the 30 Classes that were developed (files shown in Figure 54).

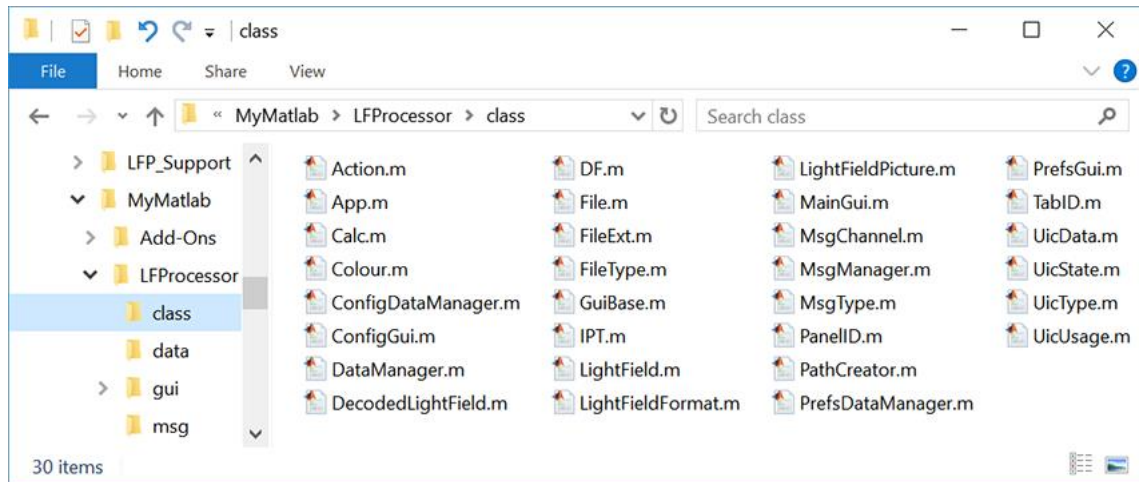


Figure 54 – List of the 30 files used for Class definitions.

5.3.3. How to save the DLF file's exact size on disk, within itself

This brief explanation serves as an example of how a tricky problem was identified and solved, within the context of this work.

Since the DLF file is automatically compressed by Matlab, we don't have any control over that process, so it is not possible to estimate the exact size of the file on disk before actually saving it, even though we know exactly what it may contain and its size in memory.

Putting it in simple and common language, this problem seemed a paradox similar to that of: *"Which came first: the egg or the chicken?"*

After a careful analysis of the available data, what could, and what could not, be controlled by the programmer, as well as the typical architecture of operating systems, all that was needed was a bit of what the Americans usually call "hacking" (in the original sense of the word).

The method implemented works like this:

1. Add a temporary ("dummy") value to the *Size* field, but having the same scale as the final value, so that the actual number of bytes occupied by both values is the same. This simple aspect is key;
2. Save the file;
3. Check the file size on disk;
4. Update the *Size* field in the DLF data structure;
5. Update just the *Size* field within the file, with the new value.

This is the method used to update the size of a DLF file whenever it is saved or updated. The second update is actually very fast, because most of the file data should already be on the disk cache, so this process is fast and transparent to the user.

5.4. Development Tools and Contents

5.4.1. Matlab: programming language and editor

Initial development of LF Processor was done using Matlab R2015b, but it was later migrated to Matlab R2016a, which is the recommended version to run LF Processor as a *dependent* application. For the concept of *dependent* application or instructions on how to install LF Processor, see *Appendix A*, regarding deployment.

The development of a LF image processing application would definitely benefit from the usage of the Light Field Toolbox, which already had functions to unpack and decode Lytro Illum's LFR files. However, this required the mandatory use of either Matlab or Octave which supports some of its code. Unfortunately Octave does not yet support graphical user interfaces. Consequently, if Light Field Toolbox were to be used, the only choice left was Matlab, which ended up being the software package used for the development of Light Field Processor.

5.4.2. Application dependencies: Light Field Toolbox

LF Processor is dependent on the Light Field Toolbox (LFT) [13] for some of its operations, such as: decoding, creation of the White Images Database, histogram equalisation and application of Depth Filters on light field images. However, it is not just a GUI for that toolbox, quite the contrary. Besides having its own GUI, LF Processor improves some aspects of LFT (e.g. colour correction) and adds new functionalities (e.g. microlenses maps) that Light Field Toolbox does not provide, amongst other features that were already mentioned throughout this work.

5.4.3. Photoshop: Image editing

Photoshop was used to design and edit the toolbar icons as well as the background image of the application. It was also used to capture and edit the images of this document.

The two icons used for the *View* operations were directly extracted from Matlab's resource icons, without any special editing (View Zoom and Data Cursor).

All the remaining icons were created as original work although usually inspired by similar Windows operating system icons. All of them required heavy editing though, because producing a readable 16x16 pixels version of an icon that must be discernible in the application screen, is no easy task.

5.4.4. Light field image datasets used

For testing purposes we used three packs of light field images:

- Light-Field Image Dataset from the *Ecole Polytechnique Fédérale de Lausanne* (EPFL), available for download at [94];
- The set that comes with Light Field Toolbox [13];
- Miscellaneous pictures taken with our own Lytro Illum camera.

5.5. Extra Work That Was Not Used

Some extra work was developed and properly tested for light field image rendering, but was not incorporated into LF Processor due to time constraints. Since that work may still be useful, and is actually proposed as future work, we think it is worth mentioning it here.

One additional algorithm for 2D image rendering from light field images was developed based in the work of Todor Georgiev and Andrew Lumsdaine [65]. Although the algorithm they proposed deals with type of input only. It is a custom format consisting of raw light field image data in 2D format, very similar to that of a *Sensor Raw Image* in LF Processor, but with a very different geometric structure (for instance, each micro-image is inverted in both directions and has a size of 74x74 pixels). More details of this input format may be found in [95].

The algorithm developed within the context of this work, hereby called *LF Render*, presents flexibility, since it is able to process the inputs used in the reference work (available to download at [96]), and also 4D light field images in the DLF format (proposed by this work). The time taken to process both formats is identical due to the programming techniques used that optimise the access to each of the different data structures.

Such an algorithm may include loops with thousands of iterations (to process pixels in the image). Because of that, each of the operations that run inside those loops should be fast to execute by a computer processor. That implied making them as simple as possible. One strategy to achieve that is to avoid complex computational operations, which are slower, such as potentiation or root extraction. That fact, not only made the algorithm simpler and clearer to understand, but also faster.

CHAPTER 6

CONCLUSIONS

"No matter what predictions we may do, the odds are that Present will always be a reflection of the Past and, the Future, a hostage of both."

Português (Portuguese):

"Independentemente das previsões que possamos fazer, as probabilidades são de que o Presente será sempre um reflexo do Passado e, o Futuro, um refém de ambos."

While considering the future ...

6.1. Contributions of This Work

The LF Processor application, developed within the context of this work, is oriented towards one large (wide) field of study: light field image processing. However, the application itself does not intend to solve one single "large" problem: instead it shows how to solve several problems that might eventually be "simpler", in some cases, but are all related to this one single field of study.

Nevertheless, in order to achieve that, many challenges were faced that had to be solved. Some of those challenges were in the field of digital imaging, but others were in the software engineering field. Although most of the visible part of this project is related to image processing with light field images, a considerable

amount of work had to be done, sometimes in innovative ways, regarding software development analysis and techniques. Therefore it seems justifiable to also mention them, especially since this is a work done as a requirement for a degree in computer science engineering.

This document's structure reflects the dual perspective by which this work may be viewed. However, that may also make it more challenging to fully understand the scope of its efforts, precisely because it may also be valued differently according to what each one values the most, or considers more important for his/her own personal work.

In this section we point out some of the major contributions of this work, considering both views as equally respectable and important.

6.1.1. LF Processor

LF Processor is clearly the main contribution of this work. It involved many challenges regarding the development tools used and the knowledge required to develop most of its features.

However, if we wanted to emphasise some of its specific developments, we would consider the following to be the major contributions of this work:

- Graphical Interface: **Viewpoint Navigator** (described in *Section 4.3.4.1*);
- Data Structure: **DLF file format** (described in *Sections 4.1.2, 4.4.1.3, 5.3.1*);
- Engine: **Depth Filters** (described in *Section 4.4.5*);
- Algorithm: **Colour Correction** (described in *Section 4.4.4.1*).
- Integration: **Automatic Cameras Management** (described in *Sections 4.2.5, 4.3.4.3, 5.2.1*).

6.1.1.1. Viewpoint Navigator

The Viewpoint Navigator (see Figure 31) may be considered as an innovative development, in the category of Graphical User Interfaces components, for displaying and controlling viewpoints in a light field image. No such graphical component was found that aimed to produce the same results, amongst the light field imaging software reviewed.

6.1.1.2. DLF file format

In the category of Data Structures, the DLF format proved to be very useful for any user of the LF Processor application, even if the user is only experimenting with the application, the parameters used for image processing tasks are stored with the file, making it useful for both present and future.

The format is flexible, optimised in terms of size (it is compressed), and scalable (expanding it with new data structures is a relatively easy task), so this may be considered as an innovative development as well.

6.1.1.3. Depth Filters engine

Amongst all engines developed here (e.g. Video Sequence), the Depth Filters engine was clearly the most laborious.

Although simple, it may also be considered innovative in concept. It should also be the most useful engine for work in real scenarios of image processing. Since the principle may be applied to most other algorithms in image processing, it could be interesting to expand it to use other algorithms.

6.1.1.4. Colour Correction

This algorithm uses a different method for calculating the Colour Balance than the Light Field Toolbox and it also uses a different CCM.

The results produced are therefore different and slightly better than the ones produced by Light Field Toolbox.

6.1.1.5. Automatic Cameras Management

The *Automatic Cameras Management* is a “background system” that was developed to support automatic selection of metadata (e.g. zoom and focus) associated with each LFR file so that it can be decoded without the user having to explicitly select its source camera, which would otherwise be required.

Its usefulness is only realised when using the application and especially when using pictures taken with different cameras. Light Field Toolbox is oriented towards a single camera usage. **Light Field Processor**, by the contrary, **is oriented towards a multi-camera environment**, more typical of a collaborative and worldwide academic community.

This system also serves as a typical example of what an application may achieve in terms of integration.

6.1.2. XSL template for automatic bibliographic sources

Despite being a work outside the scope of this dissertation’s goals, the development of the Microsoft Word XSL template for bibliographic sources may be worthy of mention, at least for its usefulness: it allows the automatic management of sources within the most used word processor in the world and it is compliant with the latest IEEE style rules, as of 2016.

It has a large target audience amongst students and scholars in the engineering fields around the world.

6.2. Future Work

In order to further improve this work, we leave a few suggestions:

- Incorporate into the LF Processor user interface, the algorithms developed for light field rendering, based on the work of Todor Georgiev and Andrew Lumsdaine;
- Expand the Depth Filters engine to include more image processing algorithms;
- Expand the DLF format to include more data;
- Convert the code to a standard development language like C++ (if targeting Linux and Windows) or C# (if targeting only Windows).

BIBLIOGRAPHY

- [1] A. Saxena, S. H. Chung, and A. Y. Ng, "Learning Depth from Single Monocular Images," in *Neural Information Processing Systems*, 2005. Available: <http://www.cs.cornell.edu/~asaxena/learningdepth/>.
- [2] M. Levoy, "The Stanford Multi-Camera Array," *Stanford University*, 2011. [Online]. [Accessed 15 Oct. 2016]. Available: <http://graphics.stanford.edu/projects/array/>.
- [3] M. Levoy, "Stanford Spherical Gantry," *Stanford University*, 12 Feb. 2012. [Online]. [Accessed 15 Oct. 2016]. Available: <http://graphics.stanford.edu/projects/gantry/>.
- [4] Lytro, "What is Light Field?," *Lytro Blog*, 05 Nov. 2015. [Online]. [Accessed 16 Nov. 2016]. Available: <http://blog.lytro.com/what-is-light-field>.
- [5] Raytrix, "Light Field Camera Technology," *Raytrix*, 2016. [Online]. [Accessed 15 Oct. 2016]. Available: <https://www.raytrix.de/technologie/>.
- [6] Lytro, "How does the Lytro Light Field Camera work?," *Lytro*, 01 Apr. 2016. [Online]. [Accessed 15 Oct. 2016]. Available: <https://support.lytro.com/hc/en-us/articles/200863430-How-does-the-Lytro-Light-Field-Camera-work->.
- [7] M. Archambault, "The Science Behind Lytro's Light Field Technology and Megaray Sensors," *PetaPixel*, 22 June 2015. [Online]. [Accessed 15 Oct. 2016]. Available: <http://petapixel.com/2015/06/22/the-science-behind-lytros-light-field-technology-and-megaray-sensors/>.
- [8] R. Ng, "Digital Light Field Photography," Ph.D. thesis, Stanford University, 2006.
- [9] Lytro, "A Seminal Day in VR: Announcing 'Moon'," *Lytro Blog*, 29 Aug. 2016. [Online]. [Accessed 16 Nov. 2016]. Available: <http://blog.lytro.com/vr-announcing-moon>.
- [10] Lytro, "Lytro Unveils World's First Light Field Camera Designed for Creative Pioneers," 22 Apr. 2014. [Web]. [Accessed 15 July 2016]. Available: <https://www.lytro.com/press/releases/lytro-unveils-worlds-first-light-field-camera-designed-for-creative-pioneers>.
- [11] Light Field Forum, "LightField Software to Process your DIY LightField pictures," *Light Field Forum*, 28 Jan. 2013. [Online]. [Accessed 15 July 2016]. Available: <http://>

- lightfield-forum.com/2013/01/lightfield-software-to-process-your-diy-lightfield-pictures-part-1/.
- [12] Lytro, "Lytro Desktop software". [Digital]. [Accessed 17 July 2016]. Available: <https://illum.lytro.com/desktop>.
- [13] D. Dansereau, *Light Field Toolbox v0.4*, 12 Feb. 2015. [Matlab Code]. [Accessed 01 Mar. 2015]. Available: <http://www.mathworks.com/matlabcentral/fileexchange/49683-light-field-toolbox-v0-4>.
- [14] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley Professional, 2003. Available: <https://www.pearsonhighered.com/product/Bass-Software-Architecture-in-Practice-2nd-Edition/9780321154958.html>.
- [15] Microsoft Corporation, "Chapter 1: What is Software Architecture?," *Microsoft Application Architecture Guide*, 2009. [Online]. [Accessed 20 Oct. 2016]. Available: <https://msdn.microsoft.com/en-us/library/ee658098.aspx>.
- [16] Wikipedia, "Software development process," *Wikipedia*, 18 Oct. 2016. [Online]. [Accessed 20 Oct. 2016]. Available: https://en.wikipedia.org/wiki/Software_development_process.
- [17] Wikipedia, "Iterative and incremental development," *Wikipedia*, 21 Aug. 2016. [Online]. [Accessed 20 Oct. 2016]. Available: https://en.wikipedia.org/wiki/Iterative_and_incremental_development.
- [18] Changing Minds, "Association principle," *Changing Minds*. [Online]. [Accessed 13 Oct. 2016]. Available: <http://changingminds.org/principles/association.htm>.
- [19] B. Thomas, "How Brain Organizes Reality: Neuroscience Of Meaning & Association," *The Huffington Post*, 27 Dec. 2012. [Online]. [Accessed 13 Oct. 2016]. Available: http://www.huffingtonpost.com/2012/12/26/how-the-brain-organizes-reality_n_2366843.html.
- [20] IEEE, "IEEE eXpress Conference Publishing," *IEEE*, 2016. [Online]. [Accessed 20 Oct. 2016]. Available: https://www.ieee.org/conferences_events/conferences/publishing/index.html.
- [21] Y. Dhondt, "BibWord: Home," *BibWord*, 2013. [Online]. [Accessed 20 Oct. 2016]. Available: <http://bibword.codeplex.com/>.
- [22] Wikipedia, "Bayeux Tapestry". [Online]. [Accessed 02 July 2016]. Available: https://en.wikipedia.org/wiki/Bayeux_Tapestry.
- [23] The Free Dictionary, *The Free Dictionary*. [Online]. [Accessed 27 July 2016]. Available: <http://www.thefreedictionary.com>.
- [24] Thayer's and Smith's Bible Dictionary, "New Testament Greek Lexicon - New American Standard," *Bible Study Tools*, 2014. [Online]. [Accessed 27 July 2016]. Available: <http://www.biblestudytools.com/lexicons/greek/nas/>.
- [25] Globo, Media Company (Brazil), "Descoberta da fotografia no Brasil faz 180 anos despercebida por brasileiros," *Globo.com*, 15 Aug. 2012. [Online]. [Accessed 27 Sep. 2016]. Available: <http://g1.globo.com/fotos/noticia/2012/08/descoberta-da-fotografia-no-brasil-faz-180-anos-despercebida-por-brasileiros.html>.
- [26] B. Kossoy, "Boris Kossoy: Selected Writings," *Boris Kossoy*. [Online]. [Accessed 27

- Sep. 2016]. Available: <http://boriskossoy.com/en/selected-writings/>.
- [27] M. Rezende, "Original Creators: Hércules Florence, The Forgotten Father Of Photography," *The Creators Project*, 19 Sep. 2011. [Online]. [Accessed 25 July 2016]. Available: <http://thecreatorsproject.vice.com/blog/original-creators-h%C3%A9rcules-florence-the-forgotten-father-of-photography>.
- [28] Wikipedia, "Hércules Florence". [Online]. [Accessed 25 July 2016]. Available: https://en.wikipedia.org/wiki/H%C3%A9rcules_Florence.
- [29] M. R. Peres, Ed., *The Focal Encyclopedia of Photography*, 4th ed. USA: Focal Press, 2007. ISBN: 978-0240807409.
- [30] Harry Ransom Center, "Joseph Nicéphore Niépce - Process," *Harry Ransom Center, The University of Texas at Austin*. [Online]. [Accessed 27 July 2016]. Available: <http://www.hrc.utexas.edu/exhibitions/permanent/firstphotograph/>.
- [31] B. Kossoy, *Hercule Florence, Pioneer of Photography in Brazil*, 1976. [PDF]. [Accessed 28 Sep. 2016]. Available: http://boriskossoy.com/wp-content/uploads/2014/11/hercule_florence_image.pdf.
- [32] E. B. Goldstein, *Sensation and Perception*, 8th ed. Belmont, USA: Wadsworth, Cengage Learning, 2010. ISBN-13: 978-0-495-60149-4. [eBook]. Available: <http://www.cengage.com>.
- [33] T. Young, "The Bakerian Lecture: On the Theory of Light and Colours," *Phil. Trans. R. Soc. Lond.*, no. 92, pp. 12-48, Jan. 1802. doi: [10.1098/rstl.1802.0004](https://doi.org/10.1098/rstl.1802.0004).
- [34] J. C. Maxwell, "XVIII.—Experiments on Colour, as perceived by the Eye, with Remarks on Colour-Blindness," *Earth and Environmental Science Transactions of the Royal Society of Edinburgh*, vol. 21, no. 02, pp. 275-298, 1857. doi: [10.1017/S0080456800032117](https://doi.org/10.1017/S0080456800032117).
- [35] S. Lowengard, "The Creation of Color in Eighteenth-Century Europe," *Gutenberg-e.com*, 2006. [Online]. [Accessed 28 July 2016]. Available: http://www.gutenberg-e.org/lowengard/A_Chap03.html#txt14.
- [36] BBC Scotland, "Scotland's Einstein: James Clerk Maxwell - The Man Who Changed the World," in *BBC*, I. Stewart, Ed., 02 Dec. 2015. [Video]. [Accessed 27 July 2016]. Available: <http://www.bbc.co.uk/programmes/b06rd56j>.
- [37] B. MacEvoy, "the color top," *Handprint: color theory*, 20 Nov. 2002. [Online]. [Accessed 28 July 2016]. Available: <http://www.handprint.com/HP/WCL/colortop.html>.
- [38] A. Brown, "Trichromatic Color," *Homo discens*. [Online]. [Accessed 28 July 2016]. Available: http://www.homodiscens.com/home/ways/perspicax/color_vision_sub/trichromatic/index.htm.
- [39] Wikipedia, "Color photography," *Wikipedia*. [Online]. [Accessed 28 July 2016]. Available: https://en.wikipedia.org/wiki/Color_photography.
- [40] Phil Coomes, "Pioneering colour photography," *BBC*, 16 May 2011. [Online]. [Accessed 28 July 2016]. Available: <http://www.bbc.com/news/13411083>.
- [41] M. Hart, "Additive and Subtractive Color Systems Explained," *American WideScreen Museum*, 1998. [Online]. [Accessed 07 Oct. 2016]. Available: <http://www.widescreenmuseum.com/oldcolor/additive-subtractive.htm>.

- [42] G. Lippmann, *The Nobel Prize in Physics 1908 - Nobel Lecture*, 14 Dec. December 14, 1908. [Online]. [Accessed 28 Sep. 2016]. Available: https://www.nobelprize.org/nobel_prizes/physics/laureates/1908/lippmann-lecture.html.
- [43] Wikipedia, "Lippmann plate," *Wikipedia*. [Online]. [Accessed 28 Sep. 2016]. Available: http://www.wikiwand.com/en/Lippmann_plate.
- [44] Wikipedia, "Gabriel Lippmann," *Wikipedia*. [Online]. [Accessed 27 Sep. 2016]. Available: https://en.wikipedia.org/wiki/Gabriel_Lippmann.
- [45] M. Archambault, "A Brief History of Color Photography, From Dream to Reality," *PetaPixel*, 11 Oct. 2015. [Online]. [Accessed 28 Sep. 2016]. Available: <http://petapixel.com/2015/10/11/a-brief-history-of-color-photography-from-dream-to-reality/>.
- [46] Wikipedia, "Frederic Eugene Ives," *Wikipedia*. [Online]. [Accessed 29 Sep. 2016]. Available: http://www.wikiwand.com/en/Frederic_Eugene_Ives.
- [47] Wikipedia, "Autochrome Lumière," *Wikipedia*. [Online]. [Accessed 28 Sep. 2016]. Available: http://www.wikiwand.com/en/Autochrome_Lumi%C3%A8re.
- [48] Wikipedia, "Leopold Godowsky, Jr.," *Wikipedia*. [Online]. [Accessed 29 Sep. 2016]. Available: https://en.wikipedia.org/wiki/Leopold_Godowsky,_Jr.
- [49] Wikipedia, "Kodacolor (still photography)," *Wikipedia*. [Online]. [Accessed 29 Sep. 2016]. Available: [https://en.wikipedia.org/wiki/Kodacolor_\(still_photography\)](https://en.wikipedia.org/wiki/Kodacolor_(still_photography)).
- [50] M. Zhang, *The First Plenoptic Camera on the Market*, 23 Sep. 2010. [Web]. [Accessed 17 July 2016]. Available: <http://petapixel.com/2010/09/23/the-first-plenoptic-camera-on-the-market/>.
- [51] Lytro, "What are the specs on the First Generation Lytro Light Field Camera?," 27 Jan. 2016. [Online]. [Accessed 17 July 2016]. Available: <https://support.lytro.com/hc/en-us/articles/200863400-What-are-the-specs-on-the-First-Generation-Lytro-Light-Field-Camera->.
- [52] Lytro, "Lytro Illum Technical Specifications," *Lytro*. [Online]. [Accessed 20 June 2016]. Available: <https://illum.lytro.com/illum/specs>.
- [53] Wikipedia, "Stereopsis," *Wikipedia*. [Online]. [Accessed 15 Oct. 2016]. Available: <https://en.wikipedia.org/wiki/Stereopsis>.
- [54] Wikipedia, "Stereoscopy," *Wikipedia*, 16 Aug. 2016. [Online]. [Accessed 15 Oct. 2016]. Available: <http://en.wikipedia.org/wiki/Stereoscopy>.
- [55] C. W. Tyler, "The Horopter and Binocular Fusion," in *Vision and Visual Disorders*, Vol. 9, Binocular Vision, D. Regan, Ed. New York: MacMillan, 1991, ch. 2, pp. 19-37. Available: http://www.ski.org/CWTyler_lab/CWTyler/TylerPDFs/Tyler_HoropterCh1991.PDF.
- [56] N. Qian, "Binocular Disparity and the Perception of Depth," *Neuron*, vol. 18, no. 3, pp. 359-368, Mar. 1997. doi: 10.1016/S0896-6273(00)81238-6.
- [57] Y. Morvan, "Acquisition, Compression and Rendering of Depth and Texture for Multi-View Video," Ph.D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2009. ISBN: 978-90-386-1682-7. Available: <http://www.epixea.com/research/publications-multi-view-video-coding.html>.
- [58] M. Levoy, "Light Fields and Computational Imaging," *Computer*, vol. 39, no. 8, pp. 46-

- 55, August 2006. doi: [10.1109/MC.2006.270](https://doi.org/10.1109/MC.2006.270).
- [59] ISO, "Technical report of the joint ad hoc group for digital representations of light/sound fields for immersive media applications," ISO/IEC JTC1/SC29/WG1(JPEG) & WG11(MPEG), Geneva, Switzerland, June 2016. 16352.
- [60] E. H. Adelson and J. R. Bergen, "The Plenoptic Function and the Elements of Early Vision," in *Computational Models of Visual Processing*, M. Landy and J. A. Movshon, Eds. MIT Press, 1991, ch. 1, pp. 3-20.
- [61] A. A. Gershun, "The Light Field," *Translated by P. Moon and G. Timoshenko in Journal of Mathematics and Physics*, vol. XVIII, pp. 51–151, 1939 (originally published in Russian in 1936).
- [62] Wikipedia, "Light Field," *Wikipedia*. [Online]. [Accessed 24 July 2016]. Available: https://en.wikipedia.org/wiki/Light_field.
- [63] I. Wigmore, "light field," *WhatIs.com - Multimedia and graphics glossary*, June 2012. [Online]. [Accessed 18 Oct. 2016]. Available: <http://whatis.techtarget.com/definition/light-field>.
- [64] C. Perwass, "Light Field Camera Presentation," *Light Field Cameras for metric 3D measurements*, 02 June 2015. [PDF]. [Accessed 20 Oct. 2016]. Available: <https://www.raytrix.de/?ddownload=1252>.
- [65] T. Georgiev and A. Lumsdaine, "Focused plenoptic camera and rendering," *Journal of Electronic Imaging*, vol. 19, no. 2, p. 021106, June 2010. doi: [10.1117/1.3442712](https://doi.org/10.1117/1.3442712). Available: <http://www.tgeorgiev.net/>.
- [66] T. Georgiev, "100 Years Light-Field," *History of the Lightfield approach*, 2008. [Powerpoint presentation]. [Accessed 25 Oct. 2016]. Available: <http://www.tgeorgiev.net/Lippmann/index.html>.
- [67] 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)*, Portland, OR, 2013, pp. 1027-1034. doi: [10.1109/CVPR.2013.137](https://doi.org/10.1109/CVPR.2013.137).
- [68] Wikipedia, "Point cloud," *Wikipedia*, 04 June 2016. [Online]. [Accessed 15 Oct. 2016]. Available: https://en.wikipedia.org/wiki/Point_cloud.
- [69] MathWorks, "3-D Point Cloud Processing," *MathWorks Documentation*, 2016. [Online]. [Accessed 15 Oct. 2016]. Available: <https://www.mathworks.com/help/vision/3-d-point-cloud-processing.html>.
- [70] Lytro, "Desktop 4 - Exporting," *Lytro Support*, 10 Apr. 2016. [Online]. [Accessed 25 Oct. 2016]. Available: <https://support.lytro.com/hc/en-us/articles/200865460-Desktop-4-Exporting>.
- [71] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, NJ 07458, USA: Prentice-Hall, Inc., 2006. 0-13-168728-x; 978-0-13-168728-8.
- [72] Wikipedia, "Image processing," *Wikipedia*, 12 Oct. 2016. [Online]. [Accessed 25 Oct. 2016]. Available: https://en.wikipedia.org/wiki/Image_processing.
- [73] J. Charles S. Johnson, *Science for the Curious Photographer*. Natick, Massachusetts, U.S.A.: A K Peters, Lda, 2010. ISBN: 978-1-56881-581-7.
- [74] Wikipedia, "Depth of focus," *Wikipedia*, 17 Oct. 2016. [Online]. [Accessed 25 Oct.

- 2016]. Available: https://en.wikipedia.org/wiki/Depth_of_focus.
- [75] D. Peterson, "How Do They Do That? Getting Everything In Focus.," *Digital Photo Secrets*, 2016. [Online]. [Accessed 25 Oct. 2016]. Available: <http://www.digital-photo-secrets.com/tip/614/how-do-they-do-that-getting-everything-in-focus/>.
- [76] N. Mansurov, "What is Vignetting?," *Photography Life*, 2016. [Online]. [Accessed 31 Oct. 2016]. Available: <https://photographylife.com/what-is-vignetting>.
- [77] Wikipedia, "Vignetting," *Wikipedia*, 01 Aug. 2016. [Online]. [Accessed 31 Oct. 2016]. Available: <https://en.wikipedia.org/wiki/Vignetting>.
- [78] HDHead, "Technical Info: Demosaicing or Debayering," *HDHead*. [Online]. [Accessed 27 Oct. 2016]. Available: <http://www.hdhead.com/?p=154>.
- [79] Wikipedia, "Demosaicing," *Wikipedia*, 23 Sep. 2016. [Online]. [Accessed 27 Oct. 2016]. Available: <https://en.wikipedia.org/wiki/Demosaicing>.
- [80] Matrix Vision, "Color Correction Matrix," *Products: Cameras*, 28 May 2012. [Online]. [Accessed 30 Oct. 2016]. Available: <https://www.matrix-vision.com/cougar-x-advanced-features-details/color-correction-matrix.html>.
- [81] D. Lang, "C# - ColorMatrix v1.5," *Dennis Lang's Source Code and Performance Metrics*, 30 Apr. 2010. [Online]. [Accessed 30 Oct. 2016]. Available: <http://landenlabs.com/cs-colormatrix/colormatrix.html>.
- [82] Imatest, "Color correction matrix," *Documentation*, 2016. [Online]. [Accessed 30 Oct. 2016]. Available: <http://www.imatest.com/docs/colormatrix/>.
- [83] Rensselaer Polytechnic Institute, "What is correlated color temperature?," *Lighting Research Center*, Oct. 2004. [Online]. [Accessed 30 Oct. 2016]. Available: <http://www.lrc.rpi.edu/programs/nlpip/lightinganswers/lightsources/whatisCCT.asp>.
- [84] Wikipedia, "Gamma correction," *Wikipedia*, 18 Oct. 2016. [Online]. [Accessed 30 Oct. 2016]. Available: https://en.wikipedia.org/wiki/Gamma_correction.
- [85] Wikipedia, "Histogram equalization," *Wikipedia*, 22 Oct. 2016. [Online]. [Accessed 31 Oct. 2016]. Available: [Histogram equalization](#).
- [86] W. Burger and M. J. Burge, "Filters," in *Principles of Digital Image Processing: Core Algorithms*. Springer Publishing Company, Incorporated, 2009, ch. 5, pp. 125-126. 978-1-84800-191-6. doi: 10.1007/978-1-84800-191-6. [eBook]. Available: <http://www.springer.com/us/book/9781848001909>.
- [87] D. G. Dansereau and L. T. Bruton, "A 4D frequency-planar IIR filter and its application to light field processing," in *Proceedings of the Intl. Symposium on Circuits and Systems*, May 2003, vol. 4, pp. 476-479. Available: <http://www-personal.acfr.usyd.edu.au/ddan1654/dansereau20034d.pdf>.
- [88] D. G. Dansereau, O. Pizarro, and S. B. Williams, "Linear volumetric focus for light field cameras," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 2, Feb. 2015. Available: <https://dl.acm.org/authorize.cfm?key=N97974>.
- [89] J. Urborg, "10 Jokes Only Data Scientists Will Understand," *Apixio*, 2016. [Online]. [Accessed 12 Oct. 2016]. Available: <http://www.apixio.com/10-jokes-only-data-scientists-will-understand/>.
- [90] Wikipedia, "Requirements analysis," *Wikipedia*, 01 Nov. 2016. [Online]. [Accessed 01 Nov. 2016]. Available: https://en.wikipedia.org/wiki/Requirements_analysis.

- [91] N. R. Patel, "LF Splitter," *LF Tools*, 2012. [Online]. [Accessed 30 Oct. 2016]. Available: <http://lightfield-forum.com/2013/01/lfpsplitter-update-for-images-processed-with-lytro-desktop-2-x/#more-7258>.
- [92] L. Jirkovsky, "Lyli," *Bitbucket*, 2016. [Online]. [Accessed 30 Oct. 2016]. Available: <https://bitbucket.org/stativ/lyli/overview>.
- [93] MathWorks, "Class Files and Folders," *MathWorks: Documentation*, 2016. [Online]. [Accessed 30 Jan. 2016]. Available: http://www.mathworks.com/help/matlab/matlab_oop/class-files-and-folders.html.
- [94] EPFL, "Light-Field Image Dataset," *EPFL: Multimedia Signal Processing Group (MMSPG)*, 2016. [Online]. [Accessed 30 Oct. 2016]. Available: <http://mmspg.epfl.ch/EPFL-light-field-image-dataset>.
- [95] T. Georgiev and A. Lumsdaine, "Focused plenoptic camera and rendering," *Journal of Electronic Imaging*, vol. 19, no. 2, p. 021106, Apr-Jun 2010. doi: [10.1117/1.3442712](https://doi.org/10.1117/1.3442712).
- [96] T. Georgiev, "Personal Web page". [Online]. [Accessed 30 Oct. 2016]. Available: <http://www.tgeorgiev.net/>.
- [97] Q. Fang, *JSONlab: a toolbox to encode/decode JSON files in MATLAB/Octave*, 2015. [Software]. [Accessed 06 Oct. 2016]. Available: <https://github.com/fangq/jsonlab>.
- [98] MathWorks, *Download and Install the MATLAB Runtime*, 2016. [Online]. [Accessed 05 Oct. 2016]. Available: <https://www.mathworks.com/help/mps/qs/download-and-install-the-matlab-compiler-runtime-mcr.html>.
- [99] Wikipedia, "Photography," *Wikipedia*. [Online]. [Accessed 27 July 2016]. Available: <https://en.wikipedia.org/wiki/Photography>.
- [100] J. P. S. Boguslav Cyganek, *An Introduction to 3D Computer Vision Techniques and Algorithms*. Chichester, United Kingdom: Wiley, 2009. ISBN: 978-0-470-01704-3.
- [101] M. Bass, Ed., *Handbook of Optics*, 3rd ed., Vol. III. USA: McGraw-Hill / Optical Society of America, 2010. ISBN: 9780071753425. [eBook]. Available: <http://www.mhprofessional.com/product.php?isbn=0071753427>.
- [102] Murdoch University, "University Library: IEEE Style," *Murdoch University*, 2016. [Online]. [Accessed 23 Oct. 2016]. Available: <http://libguides.murdoch.edu.au/IEEE>.

APPENDICES

APPENDIX A: APPLICATION DEPLOYMENT

Once upon a time, when we all used diskettes, I once had a trainee who told me he had trouble installing a certain software package. He easily installed the first diskette. The second was harder. But the third, try as he might, he never managed to install it. Not even by kicking it into an empty slot in the PC...

— A “story” based on real facts! ☺

In this appendix we go through the task of installing the Light Field Processor software application on a computer running the Windows operating system. We also took the opportunity to present some technical concepts regarding the deployment that are used in this appendix.

A.1. Application Versions

There are two types of versions that may be installed, which have very different requirements and so must be treated separately.

A.1.1. Dependent application

In this work, the term **Dependent** (also known as *Interpreted*) application refers to an application that runs by being interpreted in real-time within its original development tool or framework, as opposed to a *Standalone* application (see

Section A.1.2). A Web page with Javascript code that is interpreted and runs in real-time by a Web browser is a common example of such a model.

The standard development model of Matlab is intrinsically geared towards the development of *dependent* (or *interpreted*) autonomous scripts. It has two main disadvantages for an end user, when compared to the *standalone* model: (1) it is usually slower to execute and (2) requires a license of the software tool originally used to develop it, in this case: Matlab, which is likely to be too expensive and unfeasible for the casual user.

A.1.2. Standalone application

A ***standalone application*** is compiled and then executed directly by the operating system without requiring the presence and/or license of the software and tools originally used to develop it, such as a text editor, a C++ compiler or an Integrated Development Environment (IDE) like Microsoft Visual Studio. Matlab's interface is essentially that of an IDE.

In the Windows operating system a compiled file greater than 64KB in size is attributed the well-known extension “.exe”, which the operating system automatically interprets as executable (meaning that it contains machine code that must be executed), hence the name. As a technical remark: if the file is smaller than 64KB, a “.com” file is generated, although, from the user perspective, it shares exactly the same properties as an “.exe”

Professional software packages are usually compiled and distributed as a standalone package; that is why it makes sense to provide such a version.

However, certain development tools require extra *runtime software libraries* (sometimes simply called *runtime*) in order to run their compiled code; that is the case of Matlab. Fortunately, the *runtime* is usually free of charge and available to download from the tool developer Web site.

A.2. Installing LF Processor

A.2.1. As a dependent application

To run LF Processor as a dependent application, the recommended version of Matlab is R2016a.

To install LF Processor as a dependent application, proceed as following:

1. Find the application script files (with extension **.m**) in your package. They should be inside a folder named **“LFProcessor”**.
2. Add that folder and all its contents to the Matlab *Search Path*;
3. If there is an “Add-ons” folder inside the previous folder, all required Add-ons were already installed;

4. If you don't have the "Add-ons" you will need to download and install them manually. LF Processor requires the following Add-ons:
 - a) **Light Field Toolbox**: A set of tools for working with light field (a.k.a. plenoptic) imagery in Matlab. Downloadable from: [13].
Important: See grey box below before installing!
 - b) **JSONlab**: A toolbox to encode/decode JSON files in MATLAB/Octave.
Downloadable from: [97]
5. Restart Matlab.
This is recommended because some of its state data is only loaded once, upon start.

IMPORTANT: LF Processor uses a custom version of the Light Field Toolbox
LF Processor requires a custom version of Light Field Toolbox so that it can be integrated with the application's Graphical User Interface.

The changes made are properly commented in the code and only redirect the output of some messages (e.g. while decoding) to the window of LF Processor. They do not interfere with any other behaviour of the Toolbox; all its credits are maintained. This custom version of the Light Field Toolbox should be found inside its own folder when distributed with LF Processor.

However, if another version of Light Field Toolbox is already installed in the Matlab Path, they will conflict, because only one of the files will actually be run; which one will be used, depends on the actual path and the order by which Matlab finds files (usually by full path alphabetical order). To avoid any undesired side effects, please remove any other version of Light Field Toolbox from the Matlab Path before executing Light Field Processor.

To start LF Processor from within Matlab's workspace, just execute the following:

```
run_lfp()
```

A.2.2. As a standalone application

The standalone version of LF Processor does not require a previous installation of Matlab, nor a license. It does not need any of the third-party libraries required in the dependent version either.

To install LF Processor as a standalone application, proceed as following:

1. According to experience, in order to work correctly under Windows 8 and Windows 10, all these procedures must be executed under *Administrator* privileges.
 - a) If you don't know what that is, please contact the person responsible for installing the operating system in your computer.
2. **If Matlab R2016a is already installed on the operating system, you may skip this step.**

- a) Download the Matlab R2016a runtime freely from the MathWorks Web site at [98];
- b) Install the Matlab R2016a runtime.
- c) It occupies around 1.4 GB of disk space.
3. Find the standalone version of LF Processor in your package
 - a) The file name should be: **lfp_win.exe**
 - b) Install it as any regular windows application, by double clicking the file.
4. Go to the folder where you installed the application and double click the only executable file you find there.

A.2.3. Application Configuration

Figure 55 shows the last step of the application Configuration, which consists in updating the White Images Database.

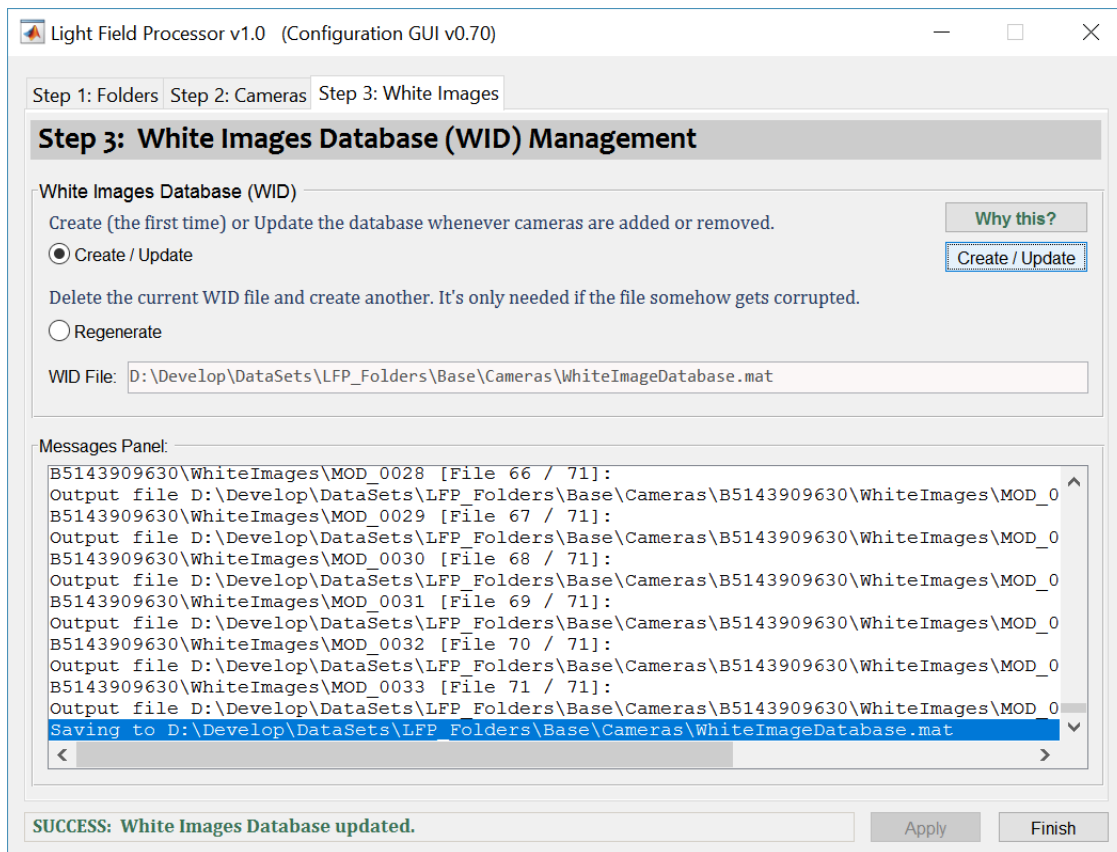


Figure 55 – Step 3 of Application Configuration: White Images Database update.