COMMUNICATION SYSTEMS
SEMESTER PROJECT
Spring 2017

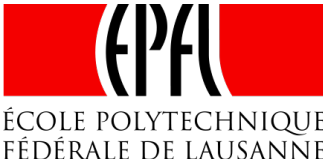
Tracking User Interaction For Light Field Compression

Student: Tanguy ALBRICI

Supervised by: Irene VIOLA
Prof. Dr. Touradj EBRAHIMI

JUNE 9, 2017

MULTIMEDIA SIGNAL PROCESSING GROUP
EPFL



Abstract

Light Field (LF) represents the intensity of light flowing in every direction for every point in space. This new way of representing content allows interactions such as changing the point of view or the focal point after content has been captured. Several compression schemes have been presented to efficiently code LF contents. However, the level of impairment caused by lossy compression schemes should be assessed in a more reliable way. Moreover, the quality of experience of users interacting with LF contents should be evaluated.

The main goal of this project is to implement a Graphical User Interface (GUI) on desktop devices, enabling the rendering of light field content and the manipulation of this content, such as refocus and change of view point. Additionally, the GUI will track interaction of the user with the LF content. The implemented interface has to be subjective test methodology-independent. In this way, different subjective tests, using various assessment methodologies, would be possible thanks to this GUI.

Table of contents

1. Introduction

In the past few years, great advance were made in light field imaging, notably with the commercialization of the first hand-held plenoptic cameras. As capturing devices become more widely available, the need for more efficient ways to process and transmit light field data grows in importance. For this purpose, compression methods suitable for light field images need to be proposed and evaluated to determine their efficiency. With this in mind, the main goal of this project is to create a desktop app for the subjective assessment of light field images, so that different compression techniques can be compared based on the human perception of image quality.

2. Theory

2.1. Light field

2.1.1. Introduction to light field

The light field (LF) is a representation of the amount of light flowing in every direction and at any point in space. This light intensity in a given direction is usually expressed in terms of radiance along a ray, which can be seen as the amount of light along all possible lines through a tube of given cross-sectional area and solid angle, as shown in Figure 2.1. The 7D plenoptic function L describes the light field using this concept of radiance, and is represented as follows:

$$L = L(\theta, \phi, Vx, Vy, Vz, \lambda, t)$$

It represents the intensity of light rays in any direction (θ, ϕ), through any point in 3D space (Vx, Vy, Vz), for any wavelength λ, and at any point in time t. Because this function is very complex it is more frequent to represent the light field in a simplified way by reducing the number of dimensions. A common simplification is to discard the time t and the wavelength λ. This 5D function can be thought of as a snapshot of the light field at a single time instance where we neglect the difference between wavelengths. It has to be mentioned that neglecting the wavelength λ does not mean that no information about the color of light can be represented. Indeed we can represent the color of light using only the three frequencies corresponding to red, green and blue (RGB).

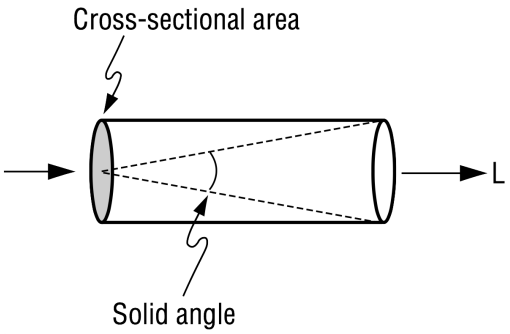


Figure 2.1: Radiance L along a ray.

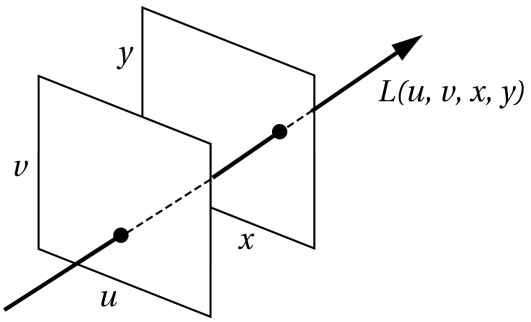


Figure 2.2: Parametrization of a light ray using two planes .

It can be observed that the radiance at two points along a same ray remains constant if no object blocks the light. Therefore if we assume a 3D space free of any occlusions, the 5D function

described can be simplified into a 4D light field function by eliminating this redundancy. The light field function L represents the radiance along all rays in empty space.

$$L = L(u, v, x, y)$$

Where each ray is parametrized by its intersection with the two respective planes uv (aperture plane), and xy (image plane), as shown in Figure 2.2. As a consequence, the light field function can be thought of as a collection of perspective image of the plane xy, taken from different positions on the uv plane. Even though this parametrization cannot represent all rays (e.g., rays parallel to the planes), it is very convenient because of its similarity with the way light field photography works, as we will see in the next chapter.

There are multiple methods available when capturing a light field. One of them uses a moving camera. For a static scene, the light field can be captured by moving the camera around the scene and taking pictures from multiple perspectives. The camera can be operated either by a motorised gantry, or by hand if the camera's position and direction can be precisely estimated. Another way is to use an array of cameras. This allows to capture the light field of a dynamic scene by synchronizing their shutter speed. The third possible method is to capture the light field via an array of lenses. This is the method used by plenoptic cameras, which is the subject of the next chapter. Compared to the first two methods, the size of the capturing device can be significantly reduced by using micro lenses. But the obvious downside is that the range of viewpoints is also reduced.

2.1.2. Plenoptic cameras

As explained in the previous chapter, plenoptic cameras capture the light field through the use of an array of micro lenses. The most simple model is therefore to place a standard image sensor directly behind the array of micro lenses. In this particular layout, the sensor will capture multiple images of the scene from different positions in the uv plane, each corresponding to a different micro lens. This results in a light field whose angular resolution (or uv resolution) depends on the number of micro lenses, and whose spatial resolution (or xy resolution) depends on the number of pixels behind each micro lens.

While this arrangement can produce very compact and thin cameras, the resolution of the views computed remains limited. For this reason, when the thickness of the camera is not concern, a more common arrangement — and the one used to capture all images in the project — consist of adding a single main lens in front of the array of lenses, as shown in Figure 2.3. In this new layout, each micro lens captures the rays of light coming from a single point on the focal plane, but in different directions. Compared to the first arrangement described, the light field is transposed; its angular resolution (or uv resolution) depends on the number of pixels behind each micro lens, and its spatial resolution (or xy resolution) depends on the number of micro lenses. In the past few years, commercially available plenoptic cameras of this type have been introduced (e.g., Lytro, Raytrix).

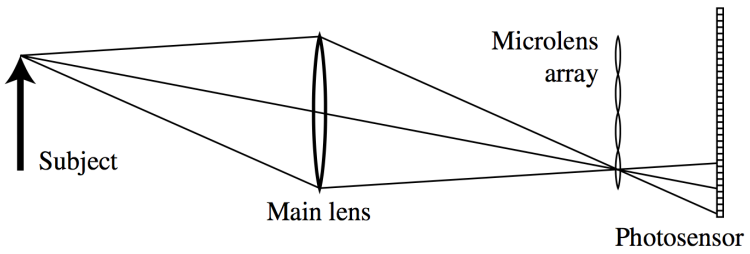


Figure 2.3: Optical design of a plenoptic camera with a main lens and a micro lens array.

2.1.3. Visualizing light field data

As we now understand the basic inner workings of a plenoptic camera, we can now develop on the different possible ways to visualise the data collected from a light field.

Raw Light Field Photograph

The most straightforward visualisation of the light field is obtained by the raw image as read from the sensor underneath the micro lens array. At a glance this raw image looks similar to an image from a conventional camera, as seen in Figure 2.4, but in reality it is constituted of an array of disks, where each disk is formed by the light rays going through a single micro lens. Consequently, the raw light field photograph can be thought of as a grid of images where each image capture the light coming from a point (x,y) on the focal plane, and where each pixel of an image correspond to a different ray of light going through the point (u, v) in the aperture plane.

Sub-Aperture Images

Light field data is also commonly represented as an array of images, each corresponding to an image of the scene, taken from a different point (u, v) in the aperture plane (Figure 2.5). A sub-aperture image is computed by combining the pixels that lie at a same point (u, v) under each micro lens in the raw light field photograph. Since the image under any micro lens has a circular shape, the grid of sub-aperture image will also be circle-shaped. We will see in chapter 3 that the two apps developed during this project use sub-aperture images as a way to display the light field to the user.

Epipolar Images

The third and most abstract way to visualise the light field is through an array of epipolar images. Each epipolar image represent a 2D slice of the light field, and is obtained by fixing y and v, while letting x and u vary respectively horizontally and vertically. Figure 2.6 shows an array of epipolar image disposed such that v increases to the right, while y increases up the image. One interesting aspect of this representation is that the depth of the objects captured in the scene can be estimated from the slope of lines in the epipolar images. Indeed, a negative slope implies that the object lies behind the focal plane, while a positive slope indicates that the object lies in front of the focal plane. To show this more explicitly, Figure 2.6 contains two closeups z1 and z2 of the array of epipolar images, showing each 5 epipolar images stacked vertically. The spatial coordinates (x, y) corresponding to these two closeups are shown in Figure 2.7. As we can see all lines in z1 are negative, and indeed the corresponding pixels in Figure 2.7 are in the background. In contrast, some lines in z2 are vertical because they correspond to the girl's nose, which is in focus.

2.1.4. Light field compression

As seen in previous chapters, capturing the light field with a plenoptic camera results in large amounts of data. Thus, efficient compression methods are needed to transmit this data in acceptable times. LF compression methods can be classified into two main categories depending on the stage of the processing chain at which compression is performed. The first approach is to compress directly the raw sensor data (also called lenslet image) after minimal processing, such as demosaicing and devignetting. The 4D LF (in its sub-aperture representation) can then be obtained through decompression and extensive post-processing, where the perspective images are extracted. Since this post-processing step is strongly dependent of the acquisition device used, camera metadata needs to be sent along with the compressed image and is used for calibration. The second approach is to compress the 4D LF that is extracted from the raw data. This time, no metadata is needed in order to visualise the compressed LF.

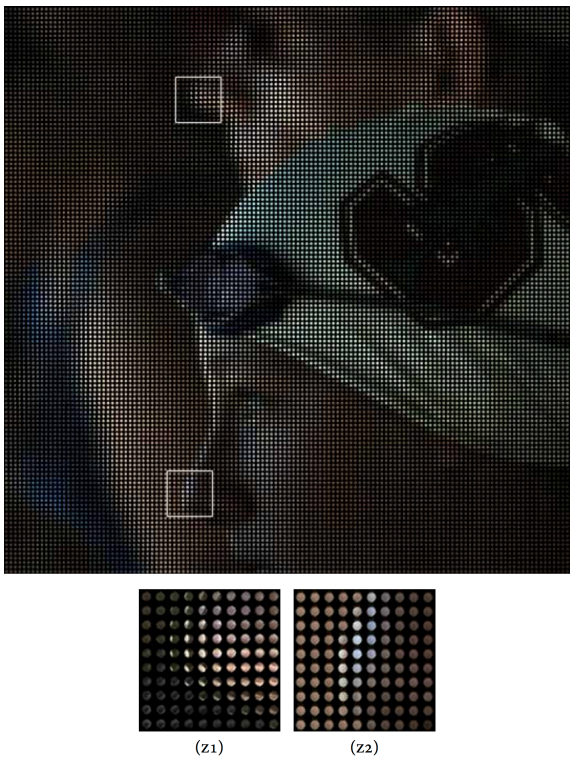


Figure 2.4: Raw light photograph (top), with two closeups z1 and z2 below.

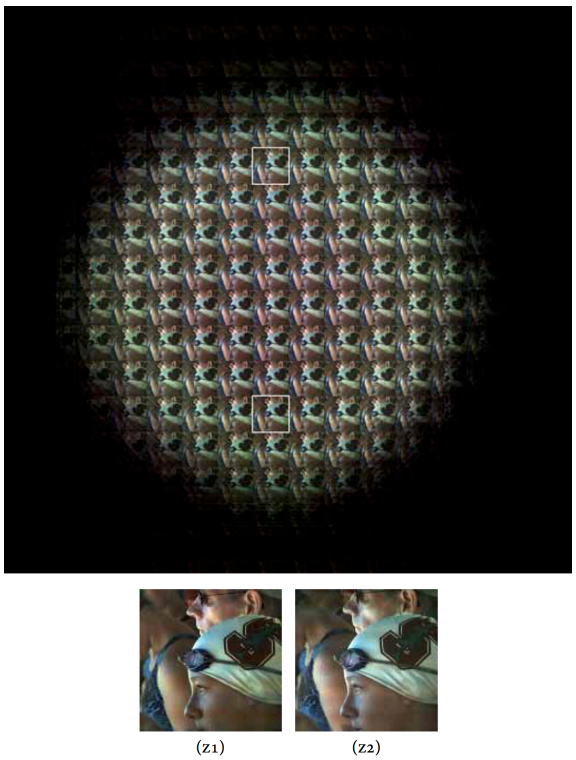


Figure 2.5: Sub-aperture images (top), with two closeups z1 and z2 below.

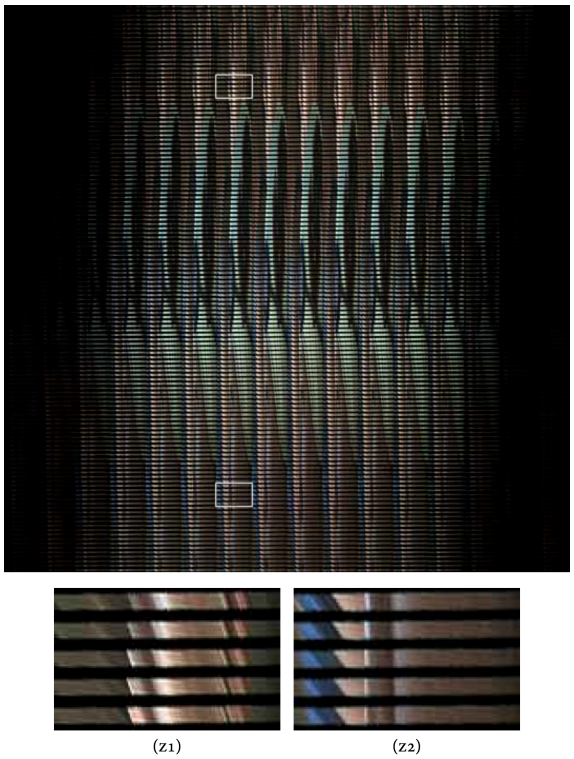


Figure 2.6: Epipolar Images (top), with two closeups z1 and z2 below.

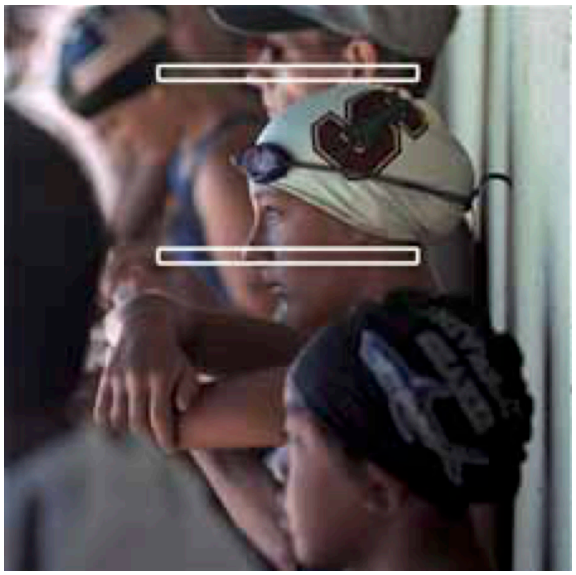


Figure 2.7: Areas of the image corresponding to the two closeups z1 and z2 in Figure 2.6.

For most applications, performing the compression on the 4D LF is preferable, as this approach has a few advantages over lenslet compression. First of all, it can be observed that this approach consistently leads to a better visual quality. Second, coding the 4D LF doesn't require any metadata, which reduces the bitrate, and allows for the process to be applied on data acquired from different devices. Third, converting the lenslet image to 4D LF is a computationally expensive task that is not suitable for low-memory devices. Finally, when coding the 4D LF, additional bitrate can be saved by discarding the most angled perspective views, which are usually very distorted. However, having the raw lenslet image and camera metadata allows for more flexibility during processing, as all the data captured is available. Thus, performing compression on the lenslet image might be a better option for professionals, but the lesser image quality implied makes it a non optimal solution.

2.1.5. Using the Light Field Toolbox for Matlab

Light Field Toolbox is a set of tools to work with light field images in Matlab. During this project this toolbox was used to process images captured with a Lytro Illum B01 camera, and more specifically to compute the different perspective views and refocused images needed for the app. Before beginning to use the toolbox, one needs to have the images coming straight from the camera (LFR files), as well as camera metadata containing white images. The 4D LF can then be decoded from the raw data using the command `LFUtilDecodeLytroFolder`. From there, a wide variety of filters can be applied before the subsequent visualization of the light field. The most useful one for this project is the shift-and-sum refocusing filter (`LFFiltShiftSum`). As its name suggests, it sums a subset of the sub-aperture images after having shifted them by a distance relative to the image's angular position (u,v) and a slope that is given as parameter. This slope can be related to the one described by epipolar images in chapter 2.1.3. Indeed, a slope parameter of zero will refocus the image on the base focus plane that was chosen when capturing the image, whereas a negative slope will refocus behind that plane, and a positive slope will refocus in front of that plane. By shifting the sub-aperture images before summing them, the points which lie on the focus plane corresponding to the slope are at the same position on each shifted sub-aperture image and are thus sharp, whereas the other points are blurred, because they have different positions among the shifted sub-aperture images.

2.2. Image quality assessment methods

Image quality assessment (IQA) plays a fundamental part when evaluating the performance of different compression methods. As it will be developed in the next chapters, this assessment can be done either objectively or subjectively.

2.2.1. Objective image quality assessment

Objective IQA is meant to mimic the quality assessment of an average human observer, but in a fully automated way. Objective IQA methods are classified in three categories, based on the availability of a perfect quality reference image.

No-reference image quality assessment (NR-IQA)

When no distortion-free reference image is available, the quality assessment must be based on the test image alone. While this type of assessment can be done relatively efficiently by humans because they already have a sense of what a good quality image should look like, implementing an equivalent automated method can prove to be a difficult task.

Reduced-reference image quality assessment (RR-IQA)

When the reference image is not fully available, a number of features can be extracted from it. These features will be used to assess the image quality and should therefore satisfy the following criteria. They should provide a solid summary of the reference image and be sensitive to a variety of distortion types, while keeping a good perceptual relevance. The framework of an RR-IQA system is shown in Figure 2.8. The transmitter sends the features extracted from the reference image through an auxiliary channel, while sending the full reference image through a distortion channel. The receiver extracts the features from the distorted test image and uses them as well as the features from the auxiliary channel in order to assess the test image quality.

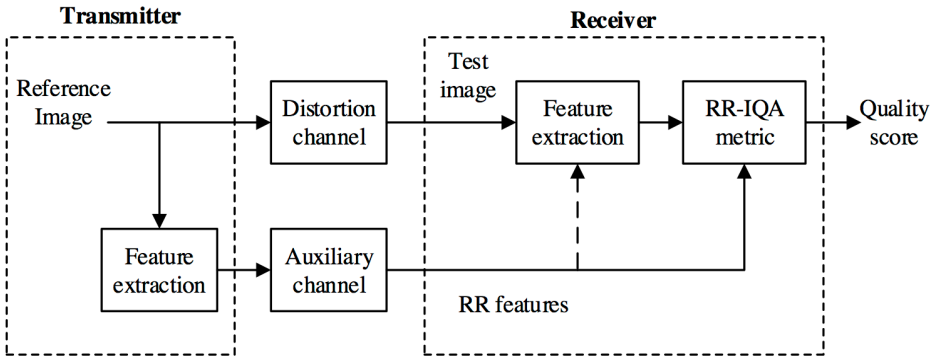


Figure 2.8: Framework of a RR-IQA system.

Full-reference image quality assessment (FR-IQA)

In this last and most simple case, the reference image is fully available, which is the case for the images used in this project. There are many different methods for FR-IQA, and we will go through a few of them.

2.2.1.1. Mean squared error

The mean square error (MSE) measures the power of the distortion, i.e., the average of the squares of the error between the reference and the test image. It is computed as follows:

$$\text{MSE} = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (I(i,j) - R(i,j))^2$$

Where M and N are the dimensions of the image, and $I(i,j)$ is the value of the test image at position (i,j), whereas $R(i,j)$ is the value of the reference image at position (i,j).

MSE is also often converted in the form of peak signal-to-noise ratio (PSNR), which is the ratio between the maximum power of the signal (in our case the image) and the power of the noise distorting the signal, and is measured in decibels (dB). It is given by:

$$\text{PSNR} = 10 \cdot log_{10}\left(\frac{MAX_I^2}{\text{MSE}}\right)$$

Where MAX_I is the largest possible value pixel value of the image (e.g., 255 for an 8 bits image).

Because of its simplicity and its low computational cost, MSE is used extensively in a wide range of signal processing applications. However, MSE is not a particularly efficient metric when trying to mimic the human perception of image quality. For light field images, since large amounts of data are implied, the MSE is usually preferred over a more computationally expensive metric like SSIM, which will be covered in what follows.

2.2.1.2. Structural similarity index

The aim of the structural similarity index (SSIM) metric is to be more consistent with human visual perception. Unlike MSE, SSIM is a model that considers distortion as a perceived change in the structural information of the image. This is based on the observation that pixels close to each other display strong dependencies, and these dependencies carry information about the structure of the scene. More specifically, as shown in Figure 2.9, the SSIM algorithm extracts structural information by separating the effect of illumination and contrast. The resulting similarity is then measured by combining the results of the luminance comparison, contrast comparison and structure comparison. This approach has the advantage to take into account both luminance masking and contrast masking when evaluating the similarity of two images. Luminance masking is the phenomenon by which distortion is less perceptible to the human eye in bright area, while contrast masking is the phenomenon by which distortion is less visible in areas that already displays a high level of detail.

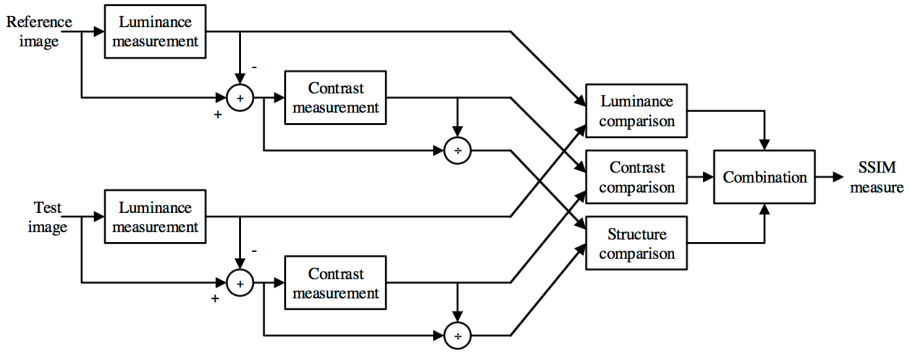


Figure 2.9: Block diagram of the SSIM algorithm.

2.2.2. Subjective image quality assessment

Since the aim of IQA is to evaluate the quality of images as it is perceived by humans, the most reliable way to do so is through subjective testing, where a group of people are asked to rate the quality of some test images. They are many different ways to conduct subjective assessments, but there exists some widely accepted standard methodologies that guarantee reliable results. In what follows, we will outline some of the recommendations described by the standard BT.500-13, produced by the International Telecommunication Union. We will first detail some general recommendations valid for any subjective IQA method, before focusing on the specifics of the double-stimulus impairment scale (DSIS) method, which is the one used in this project.

2.2.2.1. General recommendations

Before the test session

In order to produce reliable data, a minimum of 15 observers are needed. It is important to note that this does not include any potential outliers that should be removed by following the procedure explained later, thus in practice it is better to have more than 15 assessors. These observers should all be screened for correct visual acuity with the Snellen or Landolt charts, and for colour vision (e.g., using Ishihara charts). Before starting a test session, assessors should be thoroughly introduced to the method of assessment, the grading scale, and the types of impairment that may occur.

Test session

At the beginning of the test session, about five “dummy images” should be added in order to give the observer some time to stabilize his opinion. The ratings given to these images should thus not be taken into account in the test results. The images to be assessed should be ordered randomly, such that the effect of tiredness over the grading is balanced out between observers. Moreover, a test session should last no longer than thirty minutes.

Presentation of the results

Once enough data is collected, it must be processed using statistical techniques, in order to better summarize the performance of the tested system. For each image, the mean score and a 95% confidence interval should be given. Furthermore, to prevent an observer whose scores are anomalous from corrupting the test results, outliers should be detected and discarded. Assuming the distribution of scores to be normal, this process can be done by comparing the score of observer i with the interval $[P,Q] = [\mu - 2\sigma, \mu + 2\sigma]$, where μ and σ are respectively the mean and the standard deviation computed from all the scores given to this image. For each image, if the observer's score is less than P, a counter p_i is incremented, whereas if the score is greater then Q, a counter q_i is incremented. A recommended procedure is then to remove the observer if the following conditions are satisfied:

$$\frac{P+Q}{S} > 0.05 \qquad \text{and} \qquad \left|\frac{P-Q}{P+Q}\right| < 0.3$$

where S is the total number of scores

For the sake of transparency and reproducibility, it is good practice to also provide details about the test configuration and materials, number of assessors, type of picture source and type of display monitors.

2.2.2.2. Double-stimulus impairment scale (DSIS) method

During a test session implementing the DSIS method, the assessor is presented a reference unimpaired image, and the corresponding impaired image. He is then asked to rate the impairment of the latter compared to the former. This method uses the five-grade impairment scale as described in Figure 2.10. The test images should be chosen such that most observers will use every possible grade, and an overall mean score of 3 should be aimed at.

5	imperceptible
4	perceptible, but not annoying
3	slightly annoying
2	annoying
1	very annoying

Figure 2.10: Impairment scale for the DSIS method.

3. Tools developed

3.1. Desktop app

The desktop app was developed in python and allows to collect data for subjective tests using both an interactive and a passive methodology. The app was made with the DSIS method in mind, but it can be adapted to any double or single stimulus method effortlessly by changing the input parameters. In addition to storing the quality grades given by the user, the app also tracks user interaction by storing which images were displayed and for how long.

3.1.1. Required packages

Before running the app, it is important to make sure that the following python packages are installed:

- tkinter
- PIL
- threading
- datetime
- os

3.1.2. General code structure and parameters

Here we will explain briefly the role of each python file that constitutes the app. For more detailed information, please refer to the documentation in the code itself.

app.py

This is the file containing the main code which needs to be run in order to launch the app. This is where the images to be assessed are specified and the parameters most likely to be changed, such as the question to be displayed, the possible answers, their description and the assessment method (either double-stimulus or single-stimulus) are set. We can also choose whether to show the preview and whether to preload the images. Moreover, if the double-stimulus method was chosen, on can specify the side on which the test image should be displayed. The constructor of the TestSession class is then called with these parameters, which starts the test session.

TestSession.py

This file contains the TestSession class, which represents a test session for the assessment of light field images. This class controls which LF image is displayed, handles user input by transmitting it to the current LF image if necessary, and writes the grades given to the output file, among other things.

LFImage.py

This file contains the LFImage class, which represents a light field image. This class controls which viewpoint or refocused image is displayed based on user input, and write data to the tracking file. The parameters less likely to be changed have a default value set in the LFImage constructor, so that they don't necessarily have to be chosen in the main code.

SubapertureImage.py

This file contains the SubapertureImage class, which represents a sub-aperture image using its (u, v) coordinates and focus depth. For the non-refocused images, the depth is set to "None".

helper.py

This file contains some helper functions and variables which are used in multiple classes. Parameters such as the background color, the paths of the img and output folders, the image format, and the number of images to preload can be chosen.

3.1.3. GUI

The main screen of the app displays the LF image to be tested, as well as a slider for refocusing, and on button for each possible grade. If the double-stimulus method was chosen both the test image and the reference are displayed (Figure 3.2), whereas only the test image is shown for the single-stimulus method (Figure 3.3).

Loading the images

Before displaying any image on the screen, it must first be loaded in memory. This loading process can take a non-negligible amount of time, based on the size of the image and performance of the computer. If the aim is to display a single static image this is not a problem, but in our case the user should be able to navigate through the perspective images as smoothly as possible. Thus for a better user experience, each image should be preloaded before being displayed. The solution implemented in the app is to load some fixed number of images in advance, and in an asynchronous fashion. Hence, the user will most likely never have to wait for the images to be loaded, but in case he is, a loading message is presented to him. The memory allocated to previous images is cleared in order to avoid running out of memory.

Preview animation

Before letting the user interact with the LF image, a short preview animation which corresponds to the passive test methodology is shown. This preview consists of a sequence showing different viewpoints and refocused views. The viewpoints are displayed in the order given by Figure 3.1, at a rate of 10 images per seconds. As it can be seen on the figure, only a subset of the 15 x 15 viewpoints are used, due to the distortions inherent to the lenslet structure. Then the refocused view are displayed going from foreground to background and back to foreground, at a rate of 4 image per seconds.

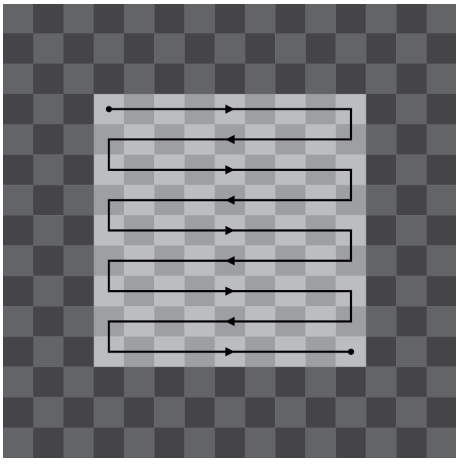


Figure 3.1: Ordering of the viewpoints for the preview animation.

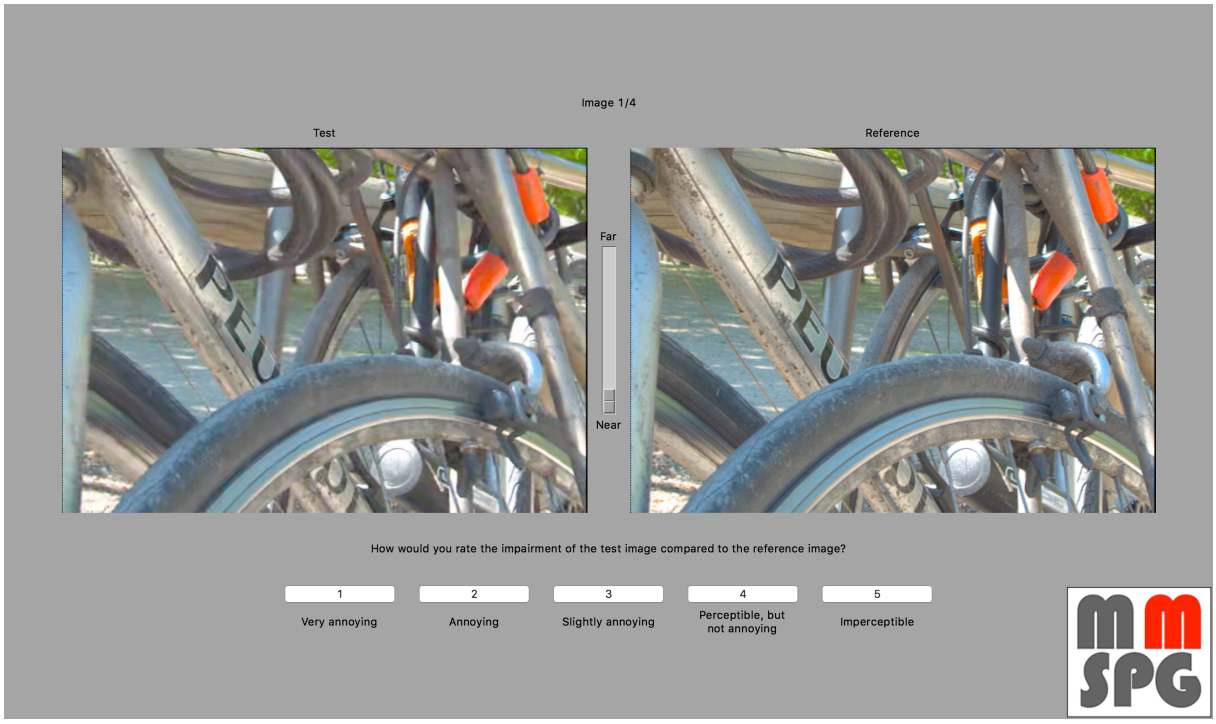


Figure 3.2: Main screen of the desktop app when using the DSIS method.

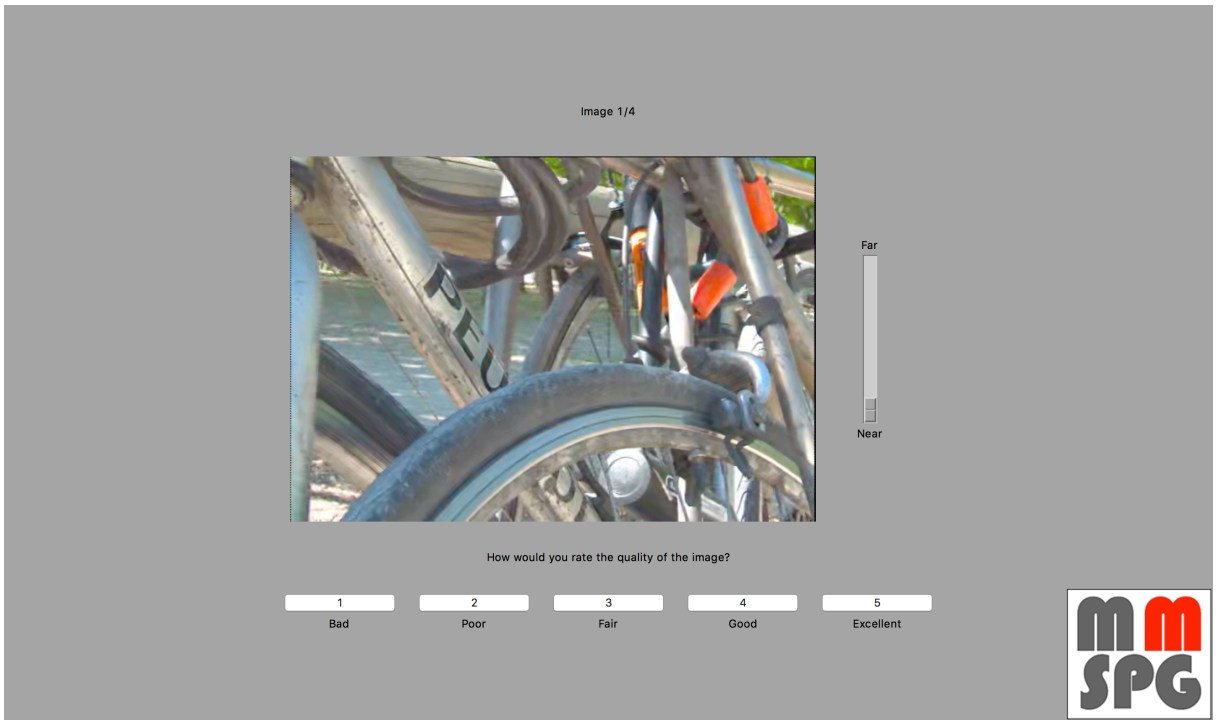


Figure 3.3: Main screen of the desktop app when using a single-stimulus method.

User interaction

The user can go through the perspective images by clicking on any of the two images and moving the mouse while keeping it pressed down. Additionally, there are two ways to display the refocused images. The first and most intuitive is to double click on any of the two image at the point where one wants the focus to be. By using the depth map of the LF image, the corresponding refocused image is then displayed. The second way is to use the slider to directly select any refocused image. This alternative way was added to ensure that the user can access every refocused image, which is not necessarily the case with the depth map-based refocusing. In order to limit the size of the image set, only the refocused images for the center viewpoint are used, thus when choosing a focus point, the image displayed goes back to the centre viewpoint regardless of the previous viewpoint position.

Grading

The user can rate the impairment of the test image by clicking on one of the buttons designed for this purpose. This directly displays the next image or the end screen if all images were graded.

3.1.4. Input images

All the images must be 8-bit PNG files stored in an *img* folder, for which the structure is shown in Figure 3.4. In the *img* folder, the different viewpoints and refocused images belonging to LF image should be grouped in a folder whose name should consist of the name of the LF image, followed by an uppercase "R" followed by an index corresponding to the compression used. For the uncompressed reference image, this index should be zero. An example could be *I01R1*, which corresponds to the LF image *I01*, with the compression 1. The sub-aperture images of each LF image should be named according to their (u, v) coordinates, with the (0,0) point being the image taken from the top-left corner. The refocused images should also have a third coordinate corresponding to the depth at which the focus is, with zero being the closest focus. The refocused images only need to be available for the center viewpoint.

The depth map of every LF image should also be available in a separate "depth_map" folder. Theses depth maps should have the name of their corresponding reference image, for example *I01R0*. These depth maps should be 8-bit grayscale PNG images. For more information about how to create them, refer to chapter 3.4.

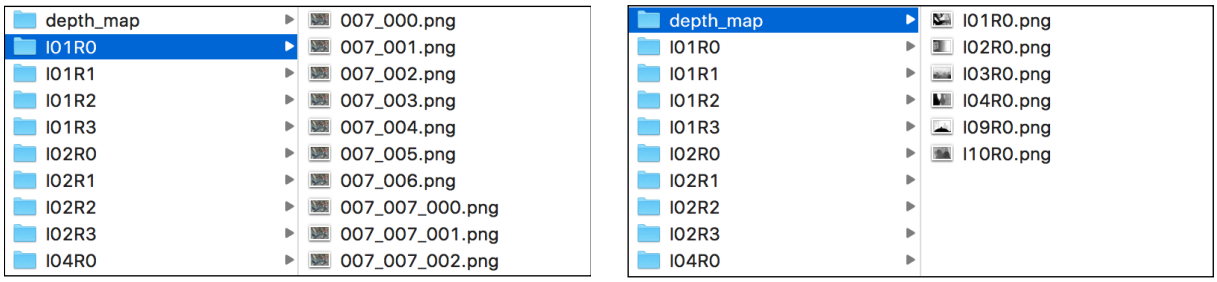


Figure 3.4: Contents of the *img* folder, with an example of a LF image folder (left) and the *depth_map* folder (right).

3.1.5. Output files

Two text files are written in the *output* folder. Both of them have a timestamp in prefix of their filename, so that multiple test session can be done before retrieving the output files.

The first one, *answers.txt* stores the grades given by the user. Each grade is on a separate line, and each line consists of the name of the image, extended or trimmed to 30 characters, followed by the corresponding grade. An example is shown in Figure 3.5.

The second one, *tracking.txt* contains information about which images were displayed and for how long. Each image displayed by the user in chronological order is detailed on a single line. When switching to the next LF image, a blank line is added. A line contains the name of the image, the time at which it was first displayed (start), the time at which it was replaced (end), and the duration over which it was visible (on-screen). These 4 pieces of information are delimited by two spaces. The format of a line is shown in Figure 3.6.

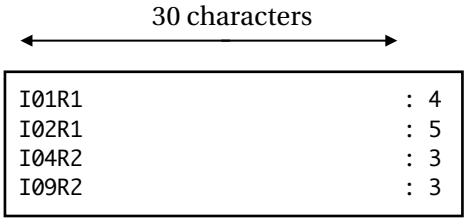
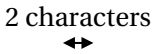


Figure 3.5: Example of a possible *answers.txt* output file.




```
I01R1/007_007.png  start: 14:11:03.459519  end: 14:11:13.169439  on-screen: 0:00:09.709920
I01R1/008_007.png  start: 14:11:13.169439  end: 14:11:14.139553  on-screen: 0:00:00.970114
I01R1/008_008.png  start: 14:11:14.139553  end: 14:11:15.083176  on-screen: 0:00:00.943623
I01R1/007_008.png  start: 14:11:15.083176  end: 14:11:18.089184  on-screen: 0:00:03.006008

I02R1/007_007.png  start: 14:11:18.108976  end: 14:11:21.827534  on-screen: 0:00:03.718558
I02R1/004_004_007.png  start: 14:11:21.827534  end: 14:11:23.075084  on-screen: 0:00:01.247550
I02R1/004_004_006.png  start: 14:11:23.075084  end: 14:11:23.540867  on-screen: 0:00:00.465783
I02R1/004_004.png  start: 14:11:23.540867  end: 14:11:25.068436  on-screen: 0:00:01.527569
I02R1/004_005.png  start: 14:11:25.068436  end: 14:11:25.332414  on-screen: 0:00:00.263978
I02R1/005_005.png  start: 14:11:25.332414  end: 14:11:26.925067  on-screen: 0:00:01.592653

I04R2/007_007.png  start: 14:11:26.944821  end: 14:11:31.525070  on-screen: 0:00:04.580249
I04R2/004_004_004.png  start: 14:11:31.525070  end: 14:11:33.953296  on-screen: 0:00:02.428226
I04R2/004_004_005.png  start: 14:11:33.953296  end: 14:11:36.258538  on-screen: 0:00:02.305242

I09R2/007_007.png  start: 14:11:36.277950  end: 14:11:39.858093  on-screen: 0:00:03.580143
I09R2/006_007.png  start: 14:11:39.858093  end: 14:11:40.154226  on-screen: 0:00:00.296133
I09R2/006_008.png  start: 14:11:40.154226  end: 14:11:40.562137  on-screen: 0:00:00.407911
I09R2/007_008.png  start: 14:11:40.562137  end: 14:11:43.842554  on-screen: 0:00:03.280417
```

Figure 3.6: Example of a possible *tracking.txt* output file.

3.2. Mobile app

The mobile app was developed for iPad using Swift and is very similar to the desktop app, except the refocusing is only done via double-tapping the image. The app also stores the quality grades given by the user, and tracks user interaction by storing which images were displayed and for how long.

3.2.1. General code structure and parameters

Here we will explain briefly the role of each swift file that constitutes the app. For more detailed information, please refer to the documentation in the code itself.

ViewController.swift

This file contains the ViewController of the main scene, which represents a test session for the assessment of light field images. This class controls which LF image is displayed and writes the grades given to the output file, among other things. This is where the name and order of the images to be assessed are specified.

LFImageStack.swift

This file contains the LFImageStack class, which is a UIStackView representing the two side-by-side image. This class controls which viewpoint or refocused image is displayed based on user input, and write data to the tracking file. The parameters less likely to be changed such as the size of the image on the screen, the angular resolution and the depth resolution are set at the very top of the class.

AnswerButtonsStack.swift

This file contains the AnswerButtonsStack class, which is a UIStackView representing the buttons used by the user to give his answers. The possible grades and their description are defined there. The buttons are all created programatically in this class, based on these parameters. As a side note, the question displayed to the user can be modified directly in the Main.storyboard file, by double-clicking on the corresponding label.

SubapertureImage.swift

This file contains the SubapertureImage class, which represents a sub-aperture image using its (u, v) coordinates and focus depth. For the non-refocused images, the depth is “nil”.

TouchDownGestureRecognizer.swift

This file contains the custom TouchDownGestureRecognizer class, which is a UIGestureRecognizer that is triggered when the user first touches the display.

Figure 3.7: Main screen of the mobile app.

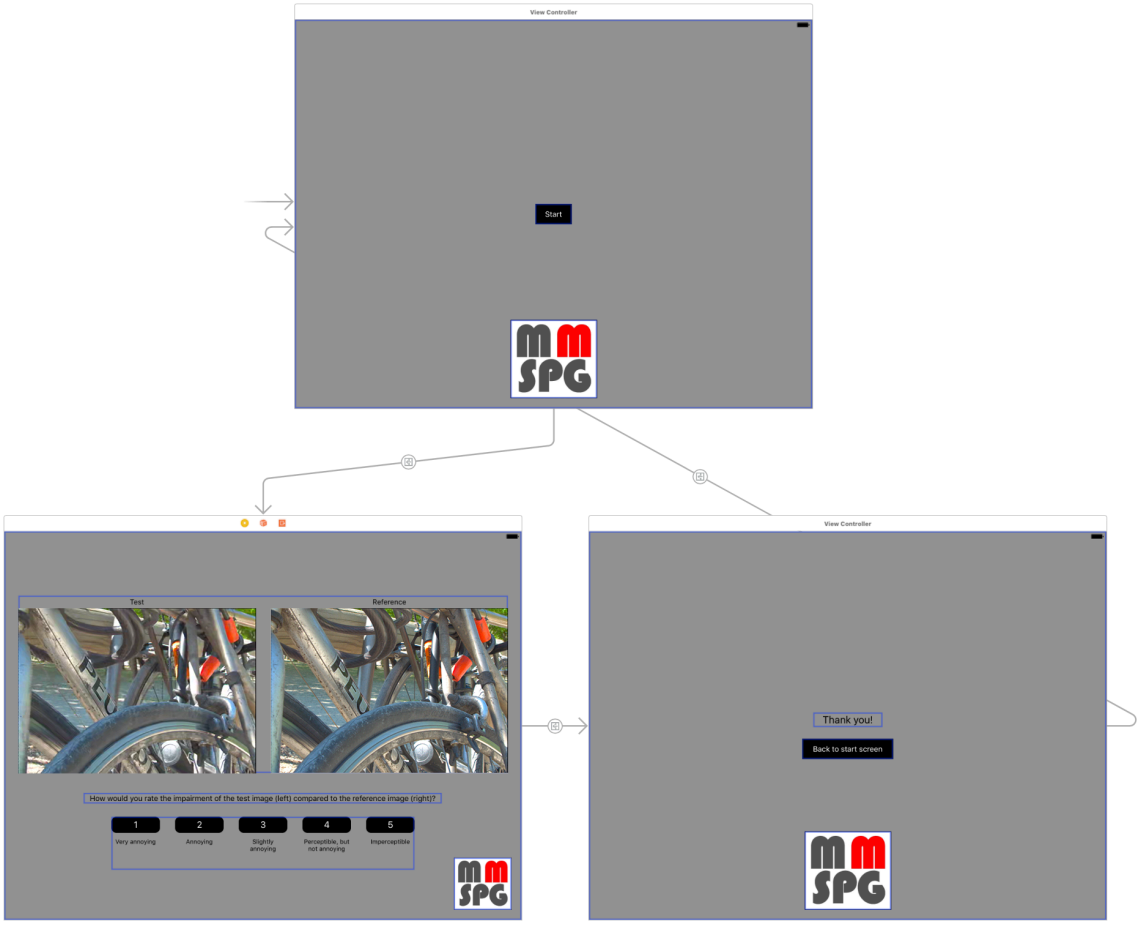


Figure 3.8: Storyboard of the mobile app, with the start screen (top) and end screen (right).

3.2.2. GUI

The GUI is very similar to the desktop app, as we can see in Figure 3.7. But since the mobile app was only a bonus for this project, the desktop app was prioritized and not every feature of the desktop app was implemented in the mobile app. The main differences is that there is no slider to refocus the image, and there is no preview of the images.

As we can see in the in the storyboard shown in Figure 3.8, when launching the app the start screen consists of a single button to start the test session. Once the test session started, the main screen with the test and reference images is displayed. The user can the go through the different viewpoints by dragging over the image, and select a refocused image by double-tapping at the desired focus point. The images can be graded using the dedicated buttons, and then the next image is displayed.

3.2.3. Input images

The folder structure and name of the images should follow the same guidelines as described for the desktop app in 3.1.4, except all images should be added to the app's assets (Assets.xcassets) instead of the *img* folder. For each image folder, it is also important to make sure that "Provide Namespace" is checked in the attribute inspector. That way, an image is identified by e.g. *I0R1/007_007* instead of just *007_007*.

3.2.4. Output files

Similarly to the desktop app, two text files are written as output, and they also have a timestamp in prefix of their filename, so that a new test session doesn't overwrite old output files. The output files are written in the so-called container of the app. The container can be downloaded using Xcode by following the steps below:

- Plug the iPad via usb to a computer
- In Xcode, got to *Window* > *Devices*
- Select the iPad under *Devices*
- Select the LFTracking app under *Installed Apps*
- Click on the gear wheel icon below and select *Download Container…*
- The container is saved as a xcappdata file. To access its contents, right-click and select *Show Package Contents*. The two output files can then be found in *AppData* / *Documents*.
- (Optional): To remove output files you don't need anymore on the iPad, delete them in the downloaded container. Then in Xcode select *Replace Container…* under the gear wheel and select the container.

The format of both output files is exactly the same as the one described for the desktop app in chapter 3.1.3, except the name of the images in the tracking files is specified without its extension (e.g., ".png"). Figure 3.9 below shows an example of a single line from the tracking file.

```
I01R1/007_007  start: 14:11:03.459519  end: 14:11:13.169439  on-screen: 0:00:09.709920
```

Figure 3.9: Example of a possible *tracking.txt* output file.

3.3. Creation of the depth maps

In order to be able to refocus the image by double-clicking on it, a depth map is needed. The depth map of a LF image captured with a Lytro camera can be obtained easily using the Lytro Desktop app. Once the LFR file is opened and selected in the Lytro Desktop app, go to *File* > *Export*, and save the *Editable Depth Map*.

Before using the depth maps, some modifications need to be made. First, they need to be converted to 8-bit PNG (the depth maps exported from the Lytro Desktop app are 16-bit PNG). Second, the depth map should be adjusted to the refocused images. Because in order to determine the refocused image corresponding to a given point on the depth map, the value of the pixel at that point is divided by the number of refocused images. Thus, all pixels of the depth map that are closer than the focal plane on the first (i.e. the closest) refocused image should be black, whereas the all pixels that are further than the focal plane on the last refocused image should be white.

In practice, the easiest way to do this adjustment is to open in Photoshop the depth map as well as the first and last refocused images. Then, apply a *Levels* adjustment layer and use the black point and white point tools to adjust the *Levels* according to respectively the first and the last refocused image, as shown in Figure 3.10.

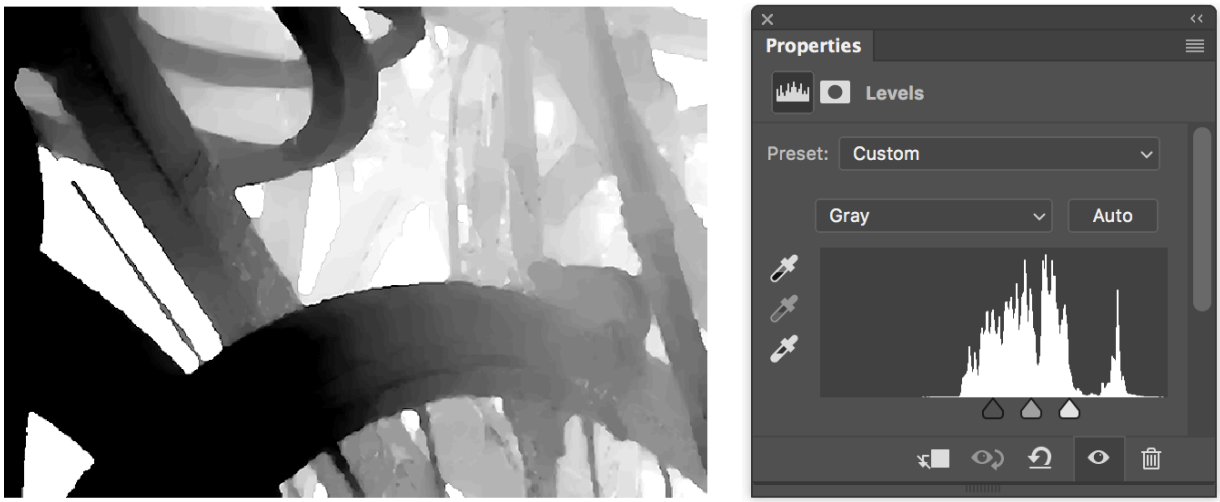


Figure 3.10: Adjustment of the depth map in Photoshop, using a *Levels* adjustment filter to set the black point and white point.

3.4. Results of the dry run

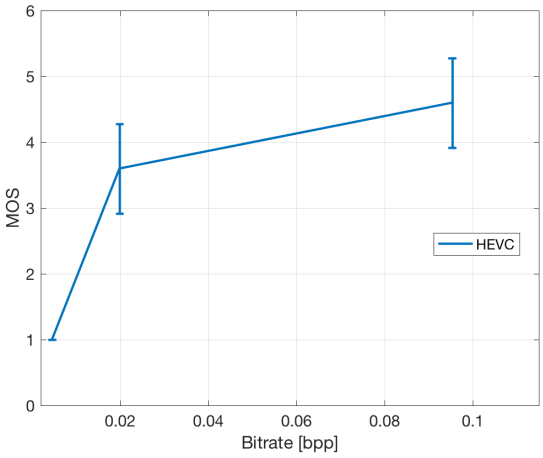
For the dry run, the five images shown in Figure 3.11 were tested on five subjects. For each LF image, its sub-aperture and refocused images were all included in a short video file that was compressed using the High Efficiency Video Coding (HEVC) standard, at three different compression rates. The order in which the images were displayed was randomized.

Outlier detection was then performed on the data collected as described in chapter 2.2.2.1, but none were found, which was to be expected for such a small sample size. The plots of the mean opinion score (MOS) compared to the actual bitrate obtained (in bits per pixel) are shown in Figure 3.12 for each image. For each point in the plots, a 95% confidence interval is shown. The actual bitrate of the compression is obtained by dividing the size of the compressed video in bits by the cumulative number of pixels among all sub-aperture and refocused images included in the video.

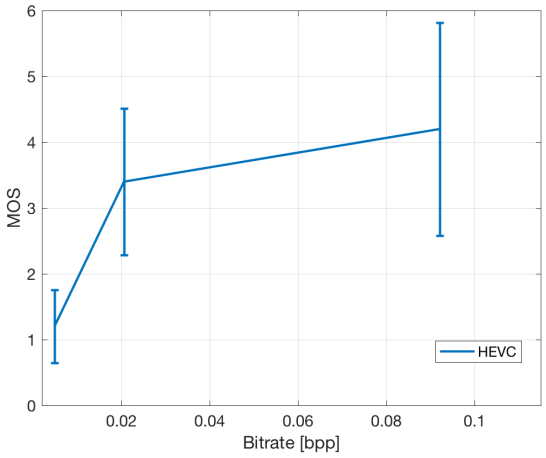
I01 I02 I04

I09 I10

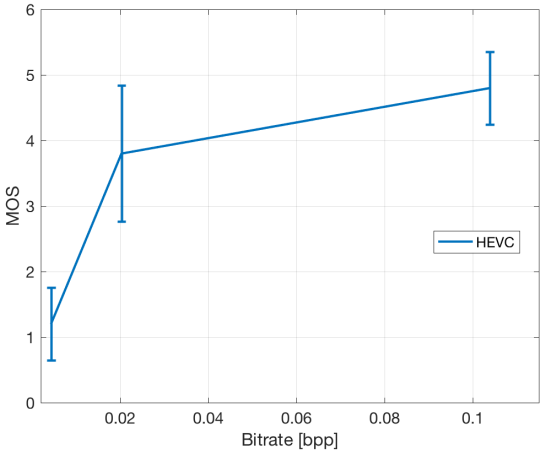
Figure 3.11: Images used for the dry run.



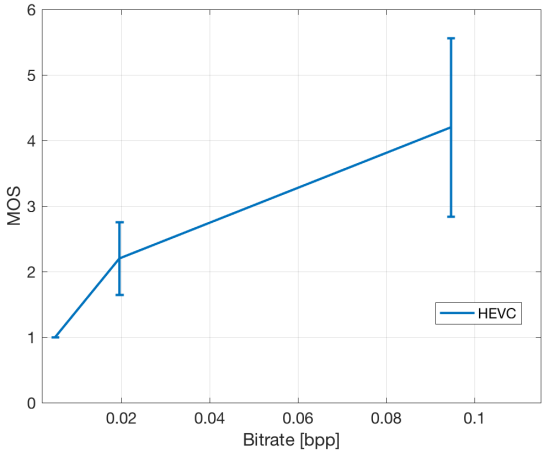
I01



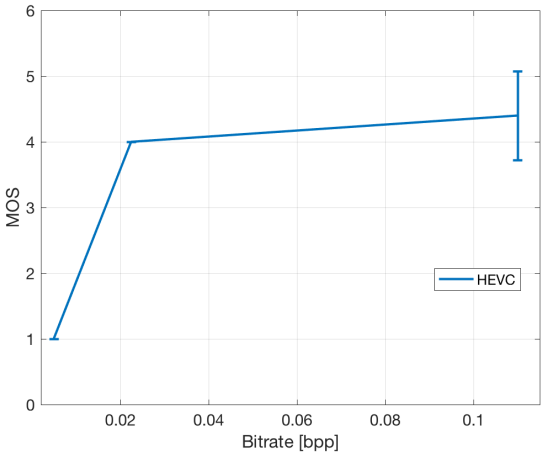
I02



I04



I09



I10

Figure 3.12: Plots of the MOS compared to the actual bitrate in bits per pixel (bpp).

4. Conclusion

During this project, I really enjoyed learning about two subjects I had no or very little experience with: light field imaging and image quality assessment. Even though no knowledge of these subjects were required to create the two apps, this research step was essential to this project, because it allowed me to get a better understanding of why and how the app will be used. Moreover, being a photography enthusiast, I was able to satisfy the curiosity I already had for light field imaging.

Besides, I also significantly improved my python programming skills by creating this app, which was of a much larger scale than anything I had previously done with this programming language. In addition, I had to learn iOS development from scratch in order to create the mobile app. This is something that I had been wanting to do for some time, and I am very glad that I could do it during this project.

In conclusion, I think that this project was very successful. I managed to create a desktop and a mobile app that serve their purpose very well, and while doing so I learned many new things about light field imaging, image quality assessment, as well as python and iOS programming.

References

M. Levoy, “Light Fields and Computational Imaging”, *IEEE Computer*, vol. 39, no 8, pp. 46-55, 2006

R. Ng, “Digital Light Field Photography”, PhD thesis, Stanford University, 2006

R. Ng, “Light Field Photography with a Hand-held Plenoptic Camera ”, *Stanford Tech Report CTSR 2005-02*, 2005

I. Viola, M. Rerabek and T. Ebrahimi, “Evaluation of light field image coding approaches”, *IEEE Journal of Selected Topics in Signal Processing*

P. Mohammadi, A. Ebrahimi-Moghadam , S. Shirani, “Subjective and Objective Quality Assessment of Image: A Survey”, 2014

ITU-R BT.500-13, “Methodology for the subjective assessment of the quality of television pictures”, International Telecommunication Union, 2012

D. G. Danserau, “Light Field Toolbox for Matlab”, software manual, 2015

Figures sources

Fig. 2.1-2.2: M. Levoy, “Light Fields and Computational Imaging”, *IEEE Computer*, vol. 39, no 8, pp. 47, 2006

Fig. 2.3: R. Ng, “Light Field Photography with a Hand-held Plenoptic Camera ”, *Stanford Tech Report CTSR 2005-02*, 2005

Fig. 2.4-2.7: R. Ng, “Digital Light Field Photography”, PhD thesis, Stanford University, 2006

Fig. 2.8-2.9: ITU-R BT.500-13, “Methodology for the subjective assessment of the quality of television pictures”, International Telecommunication Union, 2012