# A paint-based approach for optimal lighting design in real scenes

N.F.A. van der Veen
4220560

Computer Graphics and Visualisation

Embedded Systems

EEMCS, TU Delft

23 November 2018

# Contents

# 1 Preface

Well performed lighting design is an important aspect of human well-being. While well considered lighting can improve the mood and performance of an individual, the opposite can lead to poor concentration, fatigue or can even be dangerous in some situations. Proper lighting can induce emotions, provide safety, create harmony and enhance cohesion. Because of all these important aspects of lighting and the differences it can make in various situations, proper lighting design is an art that should not be underestimated.

## 1.1 A short history of lighting

In Baltimore, 1816, gas light was introduced. Although only available in public space, important buildings such as shops and theaters and rich peoples homes, it was a big improvement over candles and other lighting fueled by animal fats. A gas lamp could produce light equal to roughly fifteen candles and was safer since it was firmly connected to gas piping and hence could not be accidentally knocked over. The prices were high and the product was not perfected until around thirty five years later, when gas lighting was officially inaugurated in the city of Chicago. Because of this increase in artificial lighting quantity and quality, the people of Chicago were more safe when traveling at night. Not only was this improved public lighting system protecting them from criminals and muggers, the light also aided them in safely maneuvering on unpaved streets and dangerous intersections. The constant need for more light in the city streets, factories and offices kept the search for better lighting alive. Placing more and more gas burners was not an option anymore because of the abundance of heat and risk of fires and explosions, so an alternative had to be found. [1]

In 1878, John P. Barrett started experimenting with arc lamps that were invented by the American inventor Charles Brush. Despite this new form of lighting was a great success, its flaws resulted in yet another overhaul a couple of years later. A thin filament was placed inside a glass bulb, heated by an electric current to the point of light emission. The first light bulb was born. [1]

Years later, huge improvements in electrical power infrastructure and numerous inventions in the lighting area resulted in more and more use of lighting for all kinds of purposes. Not only was artificial lighting a source of safety and comfort, it was also widely used for commercial purposes and advertising. With the break-through of the LED in 1962, electric lighting became much more energy efficient and compact and hence could be used on a larger scale and in different shapes and sizes. Nowadays, the LED is indispensable in modern public lighting installations and lighting designers have more freedom than ever in designing our light in the darkness.

Lighting design is nowadays mainly carried out with the help of Computer Aided Design (CAD). CAD-software enables the designer to create a virtual situation and place virtual sources while being able to observe the result. It is possible to place certain objects like chairs and tables as well. All objects and also the materials in the environment influence the way that light distributes. The quality of the objects and the materials defined in the simulation are of great impact to the quality of the final lighting design. One of those CAD-tools that can be used to design lighting in the described way is DIALux [2]. With this CAD-program the user can model a scene, place light sources from an extensive catalogue and simulate the behaviour of the light in this scene. The results are presented in both a graphical (visualisation) and numerical (isolux plot) manner. These results are often calculated with the help of ray-tracing: the use of physics and mathematics that requires significant computer resources to get a sufficiently accurate result (as we will shortly see).

This work, in collaboration with Lichtvormgevers B.V. dives into a way of simulating light in a particular scene, but without substantial use of these physical and mathematical calculations, complex 3D-models and accurate material definitions. Thereby aiming to be a lot faster without sacrificing too much on accuracy. Also, the design process is extended by allowing the user to directly paint an indication of the wanted light distribution, which will be used to steer a light placement process, instead of having to place light sources manually in a trial-and-error fashion in order to achieve the wanted light ditribution. Hereby not only the process becomes more intuitive, but it also gives the opportunity to increase the design efficiency.

The research question formulated for this thesis is:

*Is it possible to generate an (optimized) lighting design for real-world applications with the use of image based lighting and a, goal-driven, painted lighting design?*

To my grandfather

F.J. Beerens
(26/11/1926 - 24/2/2018)

Niels van der Veen, 2 September 2018

## 2 Introduction

In order to make a lighting design for a certain room or environment, the interactions of light with this environment need to be known. To obtain this behaviour of light in a real scene, we could model this scene precisely in a CAD program and place virtual light sources to simulate light at different locations in this scene. Then, to design the lighting for this particular scene, we could place the virtual light sources in a trial-and-error like approach until we are satisfied with our design. This is however quite computationally expensive for complex objects and the process of converting the real scene into the virtual counterpart is a time consuming process if the result needs to be realistic. Furthermore, the cycle of having to move light sources and checking the result is time consuming and inefficient. Therefore, in this work, a different approach is used that relies on image-based lighting techniques together with a painting interface to achieve an intuitive interaction.

## 3 Previous work

In this work we capture the process of simulating artificial light with information from the real scene, thereby omitting the need of performing complex calculations. Moreover, by using information from the real scene, complex modelling of objects and materials is not needed.

As stated in the work of Debevec [3], Image-Based Lighting is "the process of illuminating scenes and objects with images of light from the real world". In this work, we use structured images of real light to arbitrarily illuminate real objects afterwards. The term "structured" refers to the way the real light images are taken, as we will see shortly. The method of obtaining the real light information in this work is an extension of the work of Salamon et. al. [4].

In [4], a light bulb attached to a stick was used to illuminate certain areas of a room in a sweeping pattern. This process was recorded using a video camera whereafter the lighting information from that room was extracted. With this information, so called "light paintings" could be created. These light paintings are originally created by moving a light source through a scene during a long exposure photograph, but could now be created afterwards. However, this approach was not directly suitable for lighting design because of the impossibility to simulate real world lighting fixtures.

The same is true for [5]. In this work, real-time relighting of scenes is accomplished by using a precomputed unstructured light cloud. After the outgoing radiance for the lights in the cloud is calculated, a new light can be inserted at any location in the cloud. The resulting exitant radiance of this light can then be calculated. In important feature of author's method, is that it enables full global illumination relighting in real-time. The system was not designed to simulate lighting fixtures and is therefore not directly suitable for (architectural) lighting design, although the authors state that spotlights could be incorporated in future works.

Using image based lighting together with a paint-like approach for lighting design is described in [6]. Here, controllable light sources are placed in a fixed pattern around a certain object. Then, the contribution of the light from each light source is independently captured. The lighting designer then paints the desired illumination on a photograph of the object, and using the acquired lighting information, the intensity and color values of the light sources are calculated in order to reproduce the design in real-life. Building a stage to place the light sources at a fixed pattern is quite time consuming, especially for larger scenes. On top of that, while the lighting design was reproduced almost instantly, no design implementation can be realised other than with omnidirectional light sources. Hence, no realistic real-world lighting design can be created with the use of this system.

Painting with light in a real-time and real-world situation is discussed in [7]. In this work, the authors built an augmented reality user interface which they called Lighty. The system consist of a number of actuated lights, a camera and a computer. For each light, the angle and brightness could be controlled individually by the computer. The user can observe a top-down picture of the scene through a tablet while also having the opportunity to design the illumination distribution of the scene by painting on the picture. An optimization module then calculates the parameters of the lights and actuates them, recreating the designed lighting distribution. While this is concrete example of lighting design by painting and could be used for evaluational purposes, it is not a realistic system. This is because only white lights with a fixed lighting distribution are used and lighting can be designed from a top-down view only.

Designing lighting with the use of a painting interface is also explored in [8]. This work aims at enabling lighters in the field of computer cinematography to paint the desired light in a purely virtual scene. Afterwards, the location and direction of virtual light sources is calculated as well as the intensity and color of the emitting light. While this work looks promising in the way it uses a "natural medium" for lighting design, the rendering is cinematic rather than realistic and requires complex modelling of the scene of interest.

Lighting design with the use of a "goal-based" approach is discussed in [9]. In this work, the lighting of a virtual 3D scene can be designed by specifying inverse luminaires which are fictitious luminaires used as a design goal. These inverse luminaires can be positive (wanted light) or negative (unwanted light). After the geometry and materials of the virtual scene are specified and the inverse luminaires are placed, the system optimizes for so called desired luminares that determine the final design result. While this method uses real-world fixtures for the lighting design, the geometry and materials of the scene need to be specified beforehand. On top of that, the design method of placing inverse luminaires is not a very intuitive one.

In [10], lighting design system for exterior scenes is discussed. While not using image-based lighting techniques, a more or less paint-based approach is used for defining lighting goals, constraints and the locations where light sources are allowed. The system makes use of real world fixtures and a stochastically based solver to automatically generate a lighting solution. However, because of not having the benefits from image-based lighting, this system relies heavily on 3D objects and textures which should be carefully constructed when creating real world lighting designs.

# 4 Method overview

The behaviour of light in a real scene in this work is captured by a real light source. To accomplish this, an omnidirectional light source is mounted to a stick and positioned at different locations in a scene. At each location, a sample is taken which contains all the lighting information needed such as shadows, texture, reflections and indirect light. It would be quite tiresome to place the light source at every point in the scene and take a picture afterwards, so instead of doing this, the light source is swept a certain path across the scene and this process is filmed. This way, each frame contains a sample of light emitting from a certain point in the scene. If the sweep is performed slowly (or the camera records at a high frame rate) the sample density can be high and hence more information is acquired.

The light probe is a term used in this thesis and is referred to the following:

***Light probe:*** *a movable omnidirectional light source that closely matches the light distribution of a point light that is used to acquire lighting behaviour of a scene.*

Another term that is used frequently in conjunction with the term light probe is a scan, which is referred to as the following:

***Scan:*** *sweeping the light probe across a certain path through the scene while filming this process.*

For this light probe to provide the best results, certain parameters have to be chosen. One of these parameters is the light intensity or luminous flux. The light probe needs to emit a certain intensity of light but this should not be too high (strongly lighting the probe handle and person carrying it) or too low (resulting in a lot of noise). Although this parameter strongly depends on the settings and sensitivity of the camera that is used to capture the footage, it is generally better to have a slightly too bright light source than a light source that is too dim. This has to do with the fact that for the recording one can easily compensate for the bright source by choosing a smaller aperture or a faster frame rate. If a too dim light source was used, this could be compensated by either choosing a higher sensitivity (increasing noise), choosing a larger aperture (shallower depth of field) or lowering the frame rate (introducing more motion blur). For most of the tests, a 60 watt incandescence bulb was used. The light probe used in the real world tests that are presented in this thesis, was a simple piece of equipment consisting of a PVC tube approximately a meter long with the light bulb on the end.

A second important parameter is the size of the light source. In general, a light source as small as possible is beneficial for the accuracy and realism of the final image. However, a smaller light source generally needs a more detailed scan. The size of the light source depends on the distance of the camera and the focal distance of the lens. It is found that, in a general situation and with a maximum depth of around three meters, the following formula gives acceptable results:

$$\frac{\text{Frame width [pixels]}}{\text{Light source width [pixels]}} \approx 25$$

Of course, this value will change when the probe sweep from front to back will be larger.

A third parameter that is important is the so called CRI (Color Rendering Index) of the used light source. This value, on a theoretical scale from 0 to 100, gives information about how accurate the light that is emitted from the light source can "display" colors. Since, by definition of a black body radiator, an incandescence light source has the highest possible CRI of 100, it is clear that this type of light source should be used for the best possible color capturing. However, recent developments in LED lighting have resulted in LED lights sources with a CRI exceeding 98, the low power consumption of LED light sources can be a valid argument supporting the use of these light sources instead.

The details of acquiring lighting information with a light probe can be found in Section 5.

If the lighting information for a particular scene is acquired, this information needs to be used in such a way that real world lighting fixtures can be simulated. For details about the processing of this information, please refer to Section 6.

The ultimate goal for lighting designers, is that they can express their ideas and designs by means of a natural medium. Generally, a lighting designer is not interested in placing fixtures in a trial-and-error approach; choosing fixtures, placing fixtures, tuning/changing fixtures and maybe even changing the initial design based on the implementation problems that arise later. These changes and tweaks, and most importantly, the selection of light sources can be very troublesome and difficult to carry out effectively. The designer is interested in designing the illumination of a certain scene or object and typically wants the final implementation to be as close to his or her design as possible. To automate the trial-and-error process and to optimize the final implementation to be as close to the initial design under certain constraints, a system is developed. This system uses the lighting simulation and visualisation framework that is build in this work, and expands its functionality to autonomously find and optimize a solution to a given situation. On top of that, this system allows the designer to paint areas of light and hence use a very intuitive way of lighting design that can increase the productivity and creativity of the designer. A graphical user interface has been built to improve usability and provide the designer with lighting visualisations that directly correspond to the design choices the designer makes. An in-depth discussion of the graphical user interface can be found in Section 7.

The system described above is dubbed the *SIMOLIDE* system, acronym for **S**mart **im**plementation and **o**ptimization of painted **li**ghting **de**sign. It is a mixed system that uses both a genetic algorithm and a linear system in combination with the image based lighting technique exploited in this work to find an implementation solution for a certain (painted) lighting design. This implementation solution is in the form of fixture parameters and a lighting visualisation. These fixture parameters can be used to give an indication for the required fixtures, positions, angles, luminance and color in the real world situation. In-depth information about this system is provided in Section 8.

# 5 Probe detection

## 5.1 The $x$ and $y$ coordinates

After a certain scene is scanned with the light-probe, the resulting footage is fed into the probe detection algorithm. This algorithm makes use of the MATLAB function *imfindcircles* [11]. This function detects circles using the circular Hough transform and is very suitable for this application. To improve the accuracy of the probe detection, effort is taken to make the algorithm look for circles in a specific radius range that is proportional to the detected probe size history. The pseudocode of this algorithm is shown below.

```
FIND PROBE

initialisation:
range_min = SIZEMIN
range_max = SIZEMAX

if imfindcircles( probe_image, [(range_min − MARGIN) (range_max + MARGIN)]) fails
then

        range_min = range_min − PERCENTAGE up to ABSOLUTEMIN
        range_max = range_max + PERCENTAGE up to ABSOLUTEMAX

else


        nothing changes
end
```

What this does is reducing the rate that a (small) circular reflection is, erroneously detected as a circle while preserving the ability to detect circles of variable size (since the probe moves in a 3D dimensional space and the third dimension is determined by the size of the probe (proximity to the camera)). For the scanfile *Opname 2.mp4* (Christmas tree, shown in Figure 1) the following results are obtained.

| Type | Detected frames | Errors | % error |
|---|---|---|---|
| Plain version | 226 | 10 | 0.044 |
| PERCENTAGE = 10% | 61 | 0 | 0 |
| PERCENTAGE = 20% | 81 | 0 | 0 |
| PERCENTAGE = 30% | 97 | 0 | 0 |
| PERCENTAGE = 10% with margin | 188 | 8 | 0.0518 |
| PERCENTAGE = 20% with margin | 189 | 8 | 0.0423 |
| PERCENTAGE = 30% with margin | 193 | 10 | 0.0426 |

As shown, the new algorithm reduces the amount of erroneous probe detection, but also significantly decreases the total amount of detected frames. Since the error is not problematic in most cases, this decrease is unacceptable, hence the plain circle detection algorithm is used and no further effort has been made in improving the probe detection algorithm.

*Figure 1: The scene of Opname 2.mp4*

After the probe location is tracked in every frame, the $[x, y]$ coordinates of the probe are known and we have the position of the probe in a two dimensional space. Since the three dimensional spatial coordinates of the probe are needed for light reproduction in a three dimensional space, information about the $z$ coordinate of the probe is still needed.

## 5.2 The missing $z$ coordinate

The camera that is used to capture the scanning process, nor the probe are equipped with a usable depth sensor of any kind, hence another way has to be found to gather this information. A very usable method is to, instead of tracking only the position of the probe, also track to size of the light source sphere. This metric can be used to estimate the distance between the probe and the camera and gives a quite accurate representation of the relative probe distance.

## 5.3 Triangulation

After the probe tracking process, for every frame, the $[x, y, z]$ coordinates of the light probe are known. Now the next step is to perform a three dimensional triangulation (Figure 2), which essentially is the construction of tetrahedra, to enable the use of the light information of the probe for simulation of arbitrarily placed point lights later on by interpolating the contributions of the light sources. This triangulation makes sure that every virtual point light source $V_{x,y,z}$ is surrounded by four physical point lights[1] $P_{x,y,z}$. This holds for any point $V$ inside the triangulation (Figure 3).

For the simulation of a point light at the arbitrary location $V_L = [x_L, y_L, z_L]$ the first step is to find the three dimensional tetrahedron

$$T = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}, P_i = [x_i, y_i, z_i]$$

that point $V_L$ fully encloses. The vertexes of the tetrahedron correspond to real probe locations, hence by assigning a weight to vertex $P_i$ that are inversely proportional to the distance from $V_L$ to vertex $P_i$, the contribution of each probe location to the virtual point light can be calculated. Hence:

$$\lambda_i = \frac{1}{d(V_L, P_i)}$$

[1] A physical point light in this context is just (the lighting information of) a captured probe location

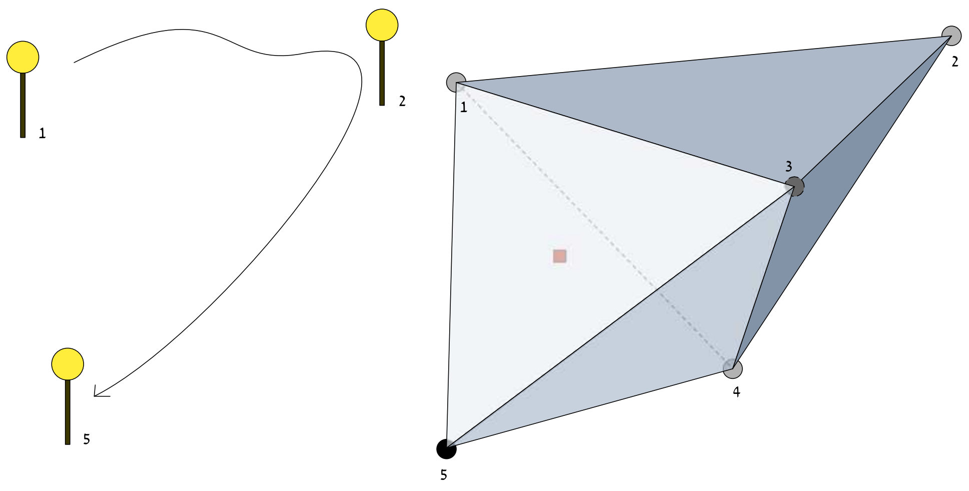


*Figure 2: The tracked probe locations are triangulized*

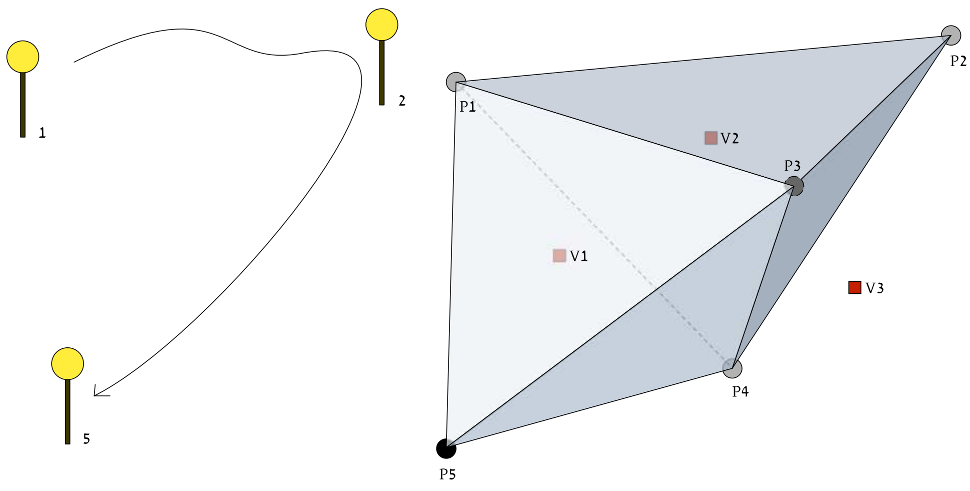


*Figure 3: The triangulation with the virtual and physical point lights. Note that $V_3$ is placed outside the triangulation*

where $d(V_L, P_i)$ is the distance between points $V_L$ and $P_i$, defined by

$$d(V_L, P_i) = \sqrt{(P_{i,x} - V_{L,x})^2 + (P_{i,y} - V_{L,y})^2 + (P_{i,z} - V_{L,z})^2}$$

$\lambda_i$ is the weight of contribution of $P_i$ (see Figure 4). AS can be seen, the weight is inversely proportional to the distance between the physical point light ($P$) and the requested virtual point light ($V$).

Now that the contribution of probe frames to the simulation of an arbitrary point light $V_L$ is known, this can be presented in the form:

$$A_{V_L} = \begin{bmatrix} \lambda_1 & P_1 \\ \lambda_2 & P_2 \\ \lambda_3 & P_3 \\ \lambda_4 & P_4 \end{bmatrix}$$

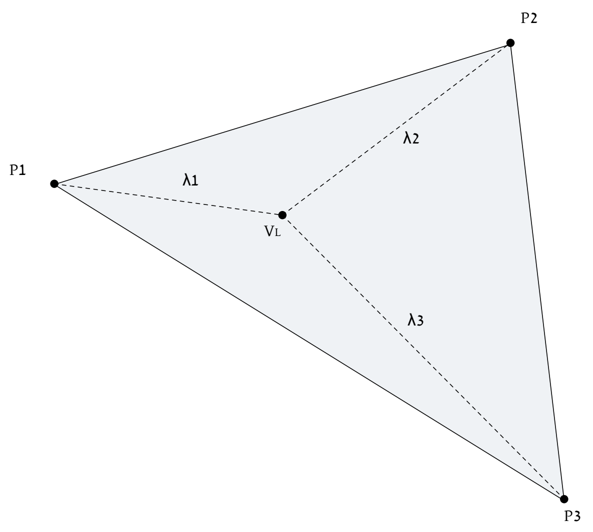


*Figure 4: Part of tetrahedron projected in two dimensions, showing the weights of the vertices.*

The frames that correspond to the vertices $P_1 \ldots P_2$ cannot be used straight away and need an extra processing step before they can be used. This is discussed in Section 5.4.

## 5.4 Removing the light probe image

After the scene is scanned one is left with a video file of the scan and hence, at each frame, lighting information of different locations in the scene. These frames are "raw", meaning that everything is visible including the entire light probe and most importantly the light source. If these raw images were used without further processing, they were of less use since the final image would also be contaminated with images of the scanning probe. Because of all this, an extra processing step is necessary.

A lot of research is already carried out in the field of image manipulation and the seamless replacement of parts of images. Use texture synthesis to replace parts of images [12]. Their work is mainly focussed at removing flaws from images. Using exemplar-based inpainting to replicate both texture and structure[13]. By scanning real world examples, inpainted regions of different images can be synthesized.

The method of inpainting used in this work, is a method based on poisson integration [14]. This method is used to replace the actual probe image, with a "neutral" image from the scene. This will effectively remove the probe image.

### 5.4.1 Poisson integration

The Poisson image editing technique is based on the conservation of the image gradient when a certain part of image $A$ is replaced by (a certain part of) image $B$.

In general we have two images; image $S$ and image $D$. Image $D$ (destination) contains the part that needs to be replaced by image $S$ (source) resulting in image $P$ (result). This is visualized in Figure 9.

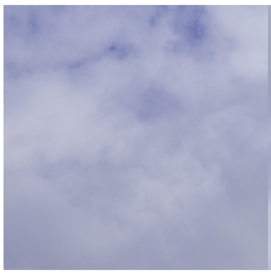


*Figure 5: S*

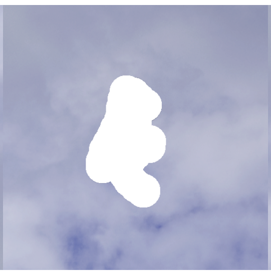


*Figure 6: D*

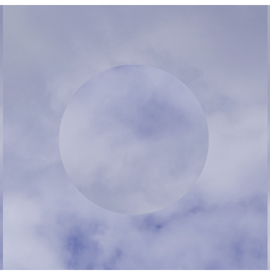


*Figure 7: R*

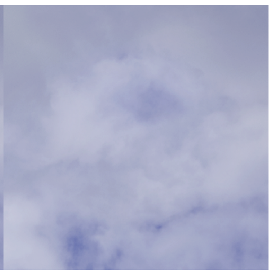


*Figure 8: P*

*Figure 9: Example images illustrating the Poisson integration. A region from D is replaced by S resulting in R. Poisson integration is executed over R, resulting in the final image P*

When placing $\Omega \in S$ in image $D$ without a special technique using just "copy and paste" tactics, the result will not look very convincing (figure $R$). In the resulting image it will be clearly visible that $\Omega$ does not belong in $D$. To improve the image replacement, we have to make sure that no sharp transition is visible between $\Omega$ and $D$ in the resulting image $P$. This can be accomplished by making the gradient $\nabla P$ inside $\Omega$ to look as close as possible to the gradient $\nabla S$ inside $\Omega$ [15]. Hence,

$$\min_{P(x,y)\in\Omega} \iint_{\Omega} |\nabla P(x,y) - \nabla S(x,y)|^2 dxdy$$

On top of that, we want to make sure that on the boundary $\delta\Omega$, the resulting image is the same as the destination image. Hence,

$$P(x,y) = D(x,y) \qquad \forall P(x,y) \in \delta\Omega$$

If this technique is applied, the region $S(x,y) \in \Omega$ is smoothly blended in $D$ resulting in a much more convincing result $P$. The reason for applying an inpainting technique in this application, is to remove the actual probe image.

**Actual probe image removal** In order to remove the probe image, one needs to replace the image region containing the probe image by another image. This image needs to be chosen carefully in order to achieve a result that is as convincing as possible. The image that is chosen for this task is the trimmed mean of all frames from the original scanning footage $\bar{\mu}$. Salamon et al [4] achieve very good results by using the trimmed mean with the brightest and darkest 10% removed. This is the reason the same technique is adopted in this work. An example of such $\bar{\mu}$ and the resulting image after probe removal is shown in Figure 10 and 11.

In this example and in the actual program, the above mentioned Poisson integration technique was used to replace the region described by the probe position and a circle with certain diameter $r$. In which $r$ depends on the size of the probe light source in the image (e.g. $r$ increases as the probe is moved closer to the camera). The Matlab code for the actual Poisson integration is the work of [16].

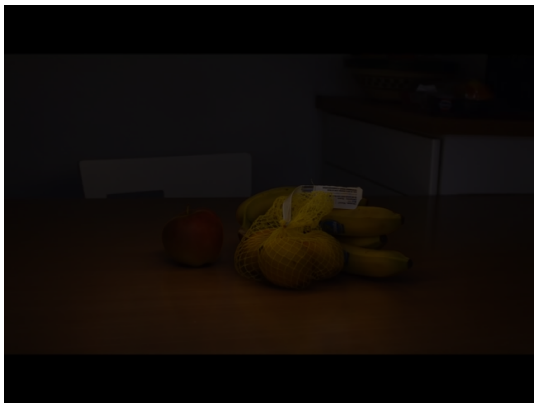
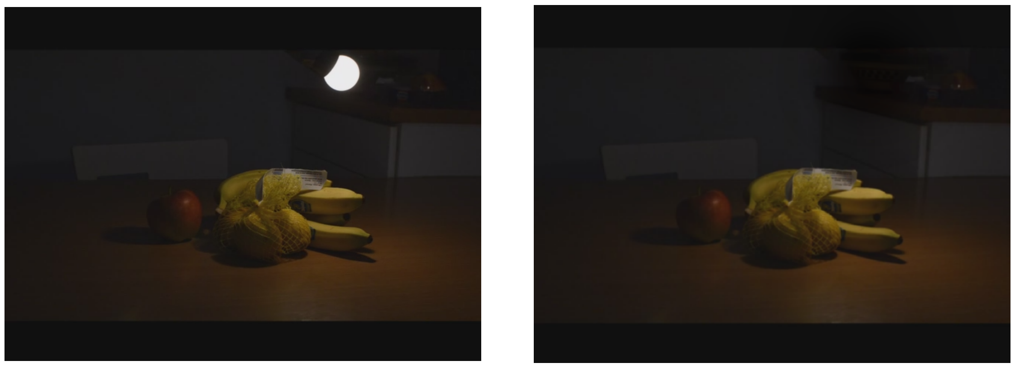


*Figure 10: The trimmed mean $\bar{\mu}$.*



*(a) Probe image not removed* *(b) Probe image removed*

*Figure 11: Removing the light probe image.*

## 6 Fixture simulation

### 6.1 Point light simulation

To simulate a fixture, a good starting point would be to simulate a point light. This can be carried out after the scanning process has been completed and the footage is processed (tracking probe location and removing probe image). As the light probe itself was an approximation of a point light, this can be done fairly accurately. To simulate a point light at location $A = (x, y, z)$, the user first defines the coordinates of this point followed by lighting properties of this point. These properties can include the color of the emitted light and the luminous flux $\Phi$. After the coordinates of the point light are defined, the frames that are necessary to carry out simulation need to be found. This is done via the triangulation process discussed in Section 5.3. In Figure 12, $P_1$, $P_2$ and $P_3$ then refer for example to frame no. 254, 165 and 311, respectively.

Now that the frames and the frame weights are found, the simulation is completed by summing these weighted frames. The resulting image is multiplied by a certain color vector $\hat{C}$ = [Red Green Blue], which represents the color of the light. Because light is additive [17], multiple point lights at different locations and of different colors can be individually simulated, whereafter a summation of the different point light images result in an image that represents the resulting effect of using all these different point lights together. An example of an obtained result is shown in Figure 13.

For the architectural lighting simulation it is important to be able to simulate existing real fixtures. Either custom made or off the shelf products. To be able to accomplish this goal,

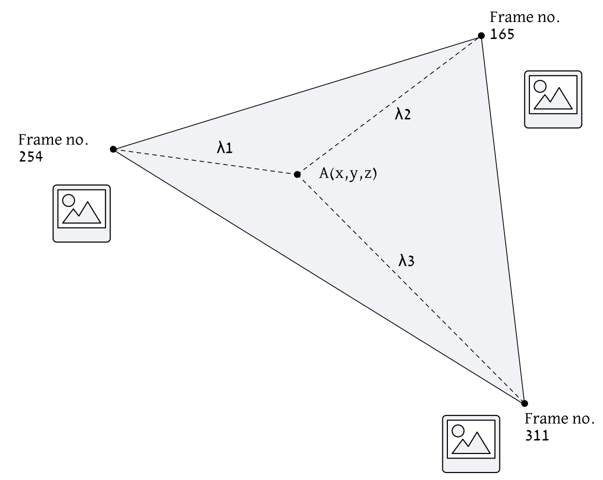


*Figure 12: The triangulation results in frame numbers with corresponding weigths.*

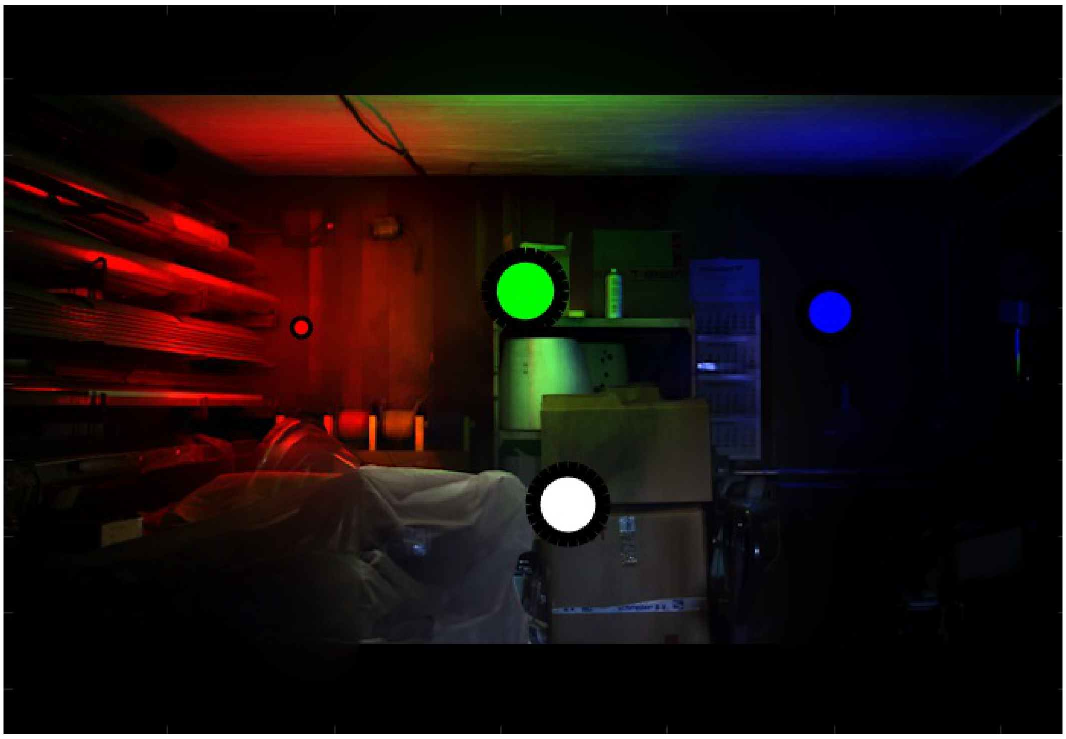


*Figure 13: The result of the point light simulation for four light sources.*

information from these fixtures need to be obtained. Fortunately file standards exists that includes all necessary information. One of these standards is the EULUMDAT file (.ldt) [18]. In this file, all relevant information from the fixture such as luminous flux, power consumption, type of light source, but also the fixtures light distribution, is provided. The latter is very important when simulating real world fixtures.

## 6.2 Calculating the light projection

Since the light intensity is defined at certain angles from the "aiming vector" of a fixture, this information can be used to calculate the light intensity at different locations in the virtual scene as if emitted by a virtual fixture. To perform this, the angle between the lighting fixtures "aiming vector" $\mathbf{A}$ and the vector starting at the location of the virtual fixture $\mathbf{F}$ and a pixel $\mathbf{P}$ ($\hat{\mathbf{A}}$) is calculated. $\mathbf{A}$ is defined by $\phi$ and $\theta$ as being the angle with respect to the $x$ and $y$ axis, respectively. This can be seen in figure 14.

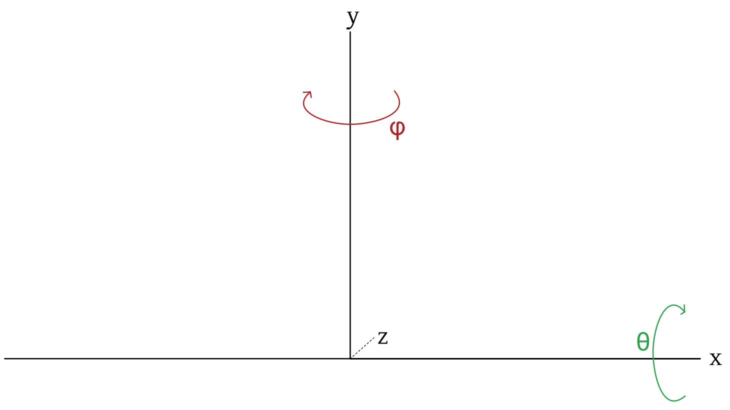


*Figure 14: The angles $\phi$ and $\theta$ in the used coordinate system*

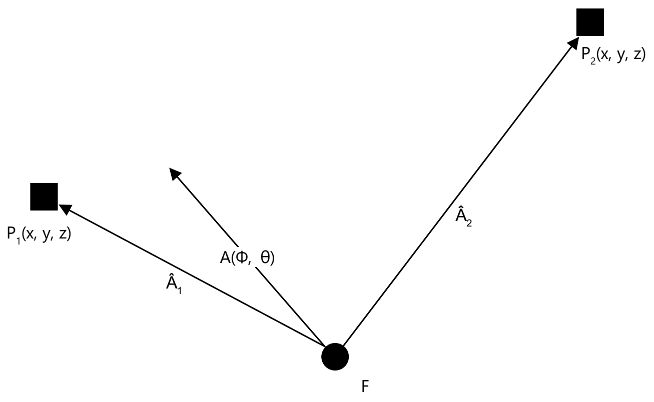


*Figure 15: The calculation of the pixel-fixture angle for two pixels $P_1$ and $P_2$*

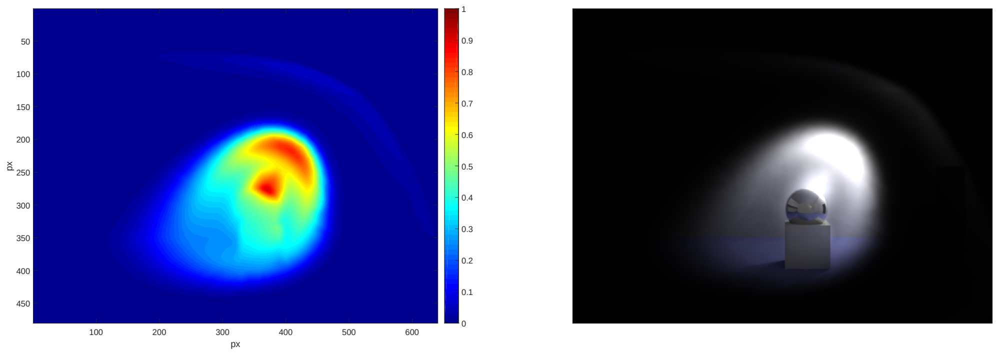
The location of every pixel in the scene is described by coordinate $(x, y, z)$. $x$ and $y$ are obtained by the location on the frame and the $z$ coordinate is obtained by a depth map (Z-map) of the scene, which in essence is an image storing in each pixel the distance to the underlying object. A detailed explanation of the calculation of this depth per pixel is discussed in Section 6.4. When the angle is calculated (figure 15), the intensity for that angle ($I_\Gamma$) is looked up from the data in the .ldt file. This intensity is then divided by the length of $\hat{\mathbf{A}}$ squared to simulate realistic light intensity falloff by distance:

$$I_\Gamma^* = \frac{I_\Gamma}{|\hat{\mathbf{A}}|^2}$$

This process is repeated for every pixel and an intensity map is generated.

This intensity map (an example is shown in Figure 25) is then used as an intensity mask which is then multiplied by the probe image as obtained at the location of the virtual fixture.

The result is the correct lighting and correct light distribution at the location of the virtual fixture.



(a) *Intensity map* (b) *The result of using the intensity map as a mask*

*Figure 16: Using the intensity map.*

## 6.3 Reading the .ldt file

A MATLAB function is written that reads an EULUMDAT file according to its file specifications line by line and extracts (a portion of) the light distribution from it. This light distribution is then used to calculate the light projection of this fixture when shone on a surface.

## 6.4 Obtaining the $Z$-map

The $Z$-map, which is essentially a matrix containing the distance of a physical point described by a pixel to the camera, is needed for an accurate visualisation of a lighting fixture. In this work, three methods have been explored for obtaining the depth map. These three methods are the following:

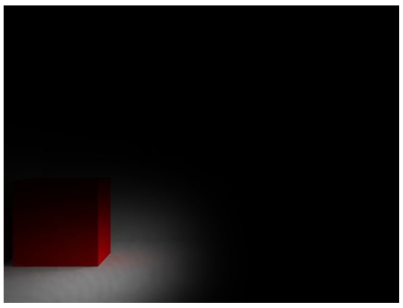
- Exploiting stereo vision.
- Obtaining depth information with the use of a Structure Sensor [19].
- Using 3D-modelled content.

To illustrate of the importance of the use of $Z$-maps for the visualisation of lighting fixtures, Figure 17 is used.

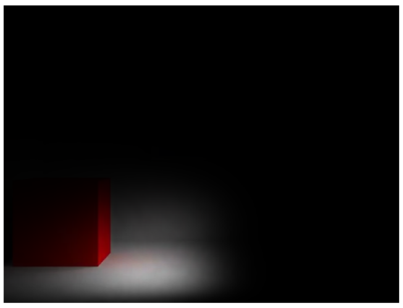
### 6.4.1 Exploiting stereo vision

Stereo vision is the technique of using two points of view to create the illusion of depth in order to simulate the human binocular vision [20]. It uses two images with a different vanishing point of a certain scene to extract depth information about that scene. While this technique requires calibration to achieve accurate results, it can be used for complex scenes and has a decent range (depending on the resolution of the camera). In this work, two different tests were carried out to assess the applicability in the described system. The first test was a simple test with a single Nikon D5100 camera that was used to take two pictures of a scene while moving the camera approximately 10 cm to the left after taking the first picture. A result of this setup is shown in Figure 39 in Section 11 [2].

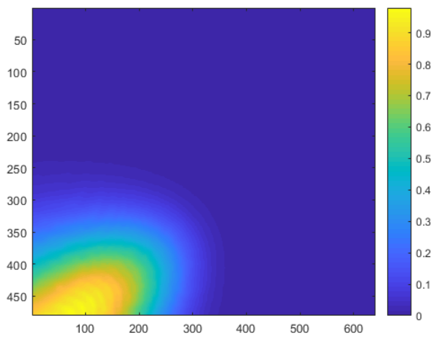
[2]The results are obtained with the help of the free software from http://3dstereophoto.blogspot.com/p/software.html



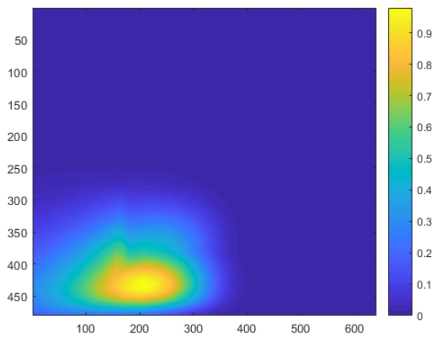
*(a) The calculated fixture projection without the use of depth information.*



*(b) The calculated fixture projection with the use of depth information.*



*(c) The calculated fixture projection mask without the use of depth information.*



*(d) The calculated fixture projection with the use of depth information.*

*Figure 17: The importance of the use of Z-maps for the visualisation of lighting fixtures.*

Shown in Figure 39 *(b)*, the two images are calibrated to ensure the resulting images are in the same plane (hence, they have an offset only in the $x$-direction). This calibration is performed automatically using the *estimateFundamentalMatrix*, *estimateUncalibratedRectification* and *detectSURFFeatures* functions in MATLAB [21].

After the images have been calibrated, the $Z$-map can be calculated. The MATLAB function *disparity* [22] is used for this purpose. The result is then an image visible in Figure 39 *(c)*.

The second test setup consists of two small identical web-cams. These webcams captured $640x480$ pixels footage and were not very high quality, but since the two were identical, they could be fixed on a rig reducing the need for calibration. This test was carried out for the latter reason. The results of this test are shown in Figure 40. As can be seen, the stereo vision method has troubles working with solid colored objects or objects with less contrast.

### 6.4.2 Obtaining depth information with the use of a Structure Sensor

To overcome the solid colored surface problem, additional test were carried out with structured light. The sensor that is used is the Structure Sensor from Occipital, Inc [19]. This sensor was at hand and is designed for scanning 3D objects and scenes, converting them to a 3D model that can for instance be viewed and edited in CAD-tools. The Structure Sensor uses the structured light method to perceive depth [23]. By projecting a certain pattern onto a three dimensional shape, depth can be measured by sensing the change in the projected pattern by a camera. This is clarified in Figure 18.

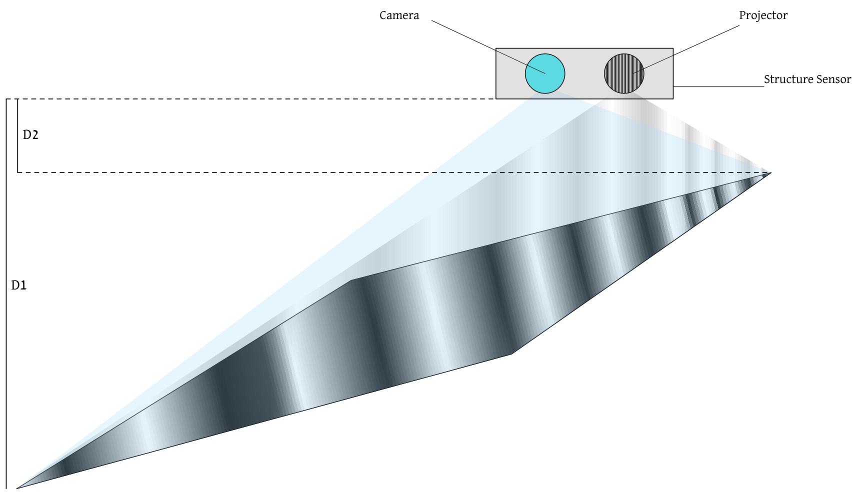


*Figure 18: The principle of the Structure Sensor depth measuring capability. For long distance $D_1$ the projected features are separated by a larger distance while for short distances $D_2$ this separation is smaller.*

Since this sensor in essence creates its own texture, it has no problem finding depth information for solid colored surfaces. A test that was carried out with the Structure Sensor contained a badly made bed with pleated solid white sheets laying on top of it. The depth map that was calculated by the Structure Sensor setup is shown in Figure 41.

The depth map that is generated by the Structure Sensor needs to be processed to reduce noise and holes[3]. This processing step is performed by a custom MATLAB script. This script evaluates the gradient of the $Z$-map image to identify the holes. In general, a stronger gradient is present around a (visible) hole in an image. Once these holes are identified, the original $Z$-map image is processed in such a way that at the location of the holes, a small region is filled using inward interpolation. The later is done using the function *regionfill* [24].

### 6.4.3 Using 3D-modelled content

The third method for achieving depth information is a quite counter-intuitive one; instead of using the depth map to find the 3D-model of an object, a simple 3D-model of an object is made to find its depth map. For a simple scene, for instance the kitchen scene shown in Figure 42, the simple 3D-model takes approximately five minutes of work to create in Blender[25].

After the 3D-model is created, the normalized depth map is exported and can be used.

## 6.5 Additional effects

To be able to simulate real life fixtures as accurate as possible and to give the use the possibility to add an extra layer of visualisation to his or her design, several (simplified) lighting effects are implemented. These lighting effects are the following and are discussed in their corresponding sections:

[3]A hole in this case is a small area in the $Z$-map where the $Z$ values are not defined. This originates from the fact that shadows are being created by geometry that blocks the structured light pattern.

- Indirect lighting (Section 6.5.1)
- Glare (Section 6.5.2)
- CRI (Section 6.5.3)

### 6.5.1 Indirect lighting

To achieve realistic lighting results, indirect lighting is a very important aspect. The point light simulation that is performed with the "light probing technique", already incorporates indirect lighting by nature. The light from the probe that is bounced off an object, is captured when recording the scan. However, for fixture simulation, this information is meanly removed by the mask that is used to model the fixture (see Figure 19).

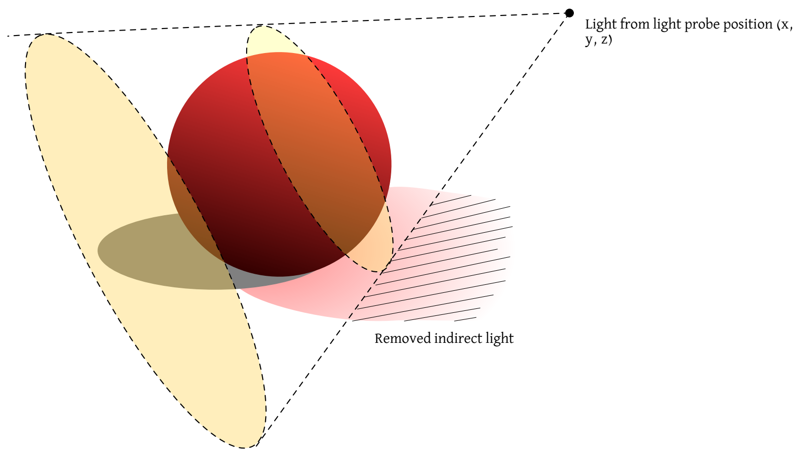


*Figure 19: A section of the point light distribution is cut out to simulate a fixture. This inevitably results in a (possible) loss of indirect lighting information.*

The simulation of this discarded indirect light is now necessary. This is done by analysing the projection of the fixture as placed into the virtual scene. The two brightest spots in this projection are found and their locations are averaged to get the point $V$.

$$V_x = \frac{\Gamma_{1,x} + \Gamma_{2,x}}{2}$$

$$V_y = \frac{\Gamma_{1,y} + \Gamma_{2,y}}{2}$$

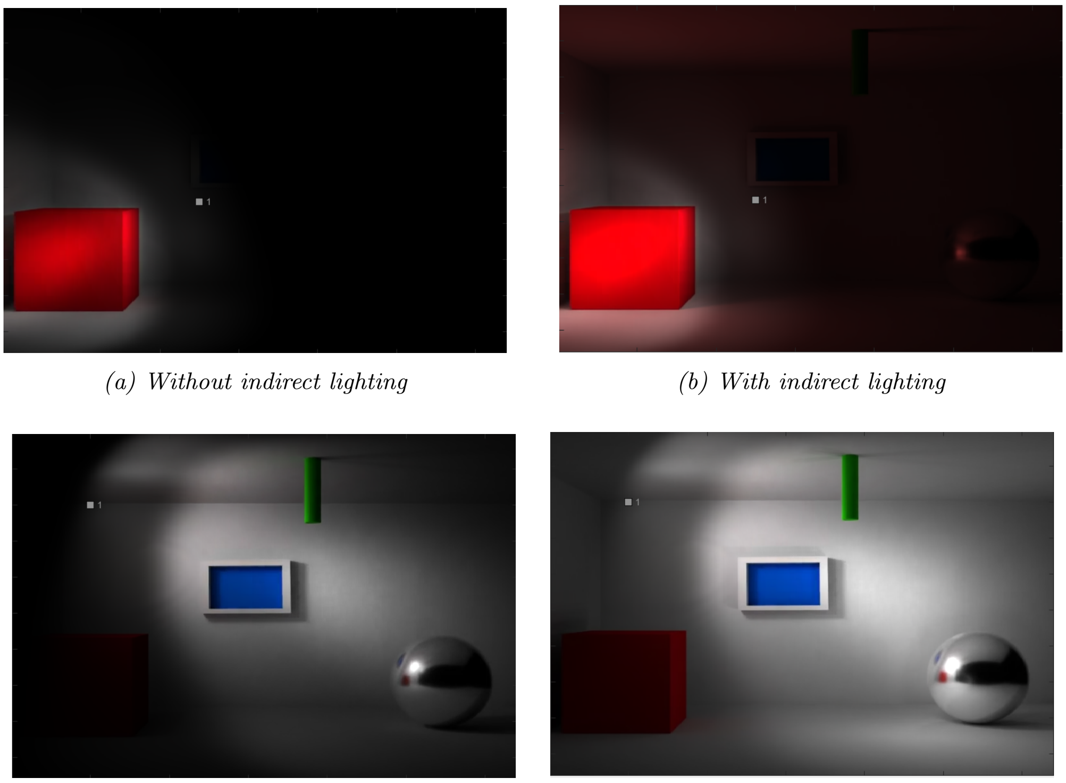
In which $\Gamma_1$ is the brightest spot in the image, and $\Gamma_2$ the brightest spot that has 75% the intensity of $\Gamma_1$. Now, the color information of the indirect light needs to be known and is calculated as follows:

$$Col_c = \overline{I_c > \tau}$$

In which $Col_c$ is the $c$'th color channel, $I_c$ is the intensity matrix of color channel $c$ and $\tau$ a certain threshold value. To visualize the indirect lighting after the fixture is placed, a point light $P(V_x, V_y, V_z, Col)$ is placed at the location of the brightest spot with the corresponding color and intensity.

### 6.5.2 Glare simulation

Another important aspect of light simulation is glare that is, or can be caused by lighting fixtures. In lighting design, both visual comfort and safety are important factors and therefore the simulation of glare is something that should be taken into account. However, since



*(a) Without indirect lighting* *(b) With indirect lighting*

*(c) Without indirect lighting* *(d) With indirect lighting*

*Figure 20: Indirect lighting simulation*

glare simulation is beyond the scope of this thesis, only a simple method of simulating glare is briefly discussed.

**Background information**
Glare is caused when there is a sufficient difference between the luminance of an observed object and the luminance of a fixture in the direction of the eye. Trivially, the latter should be higher to induce glare. To evaluate the discomfort of glare, the Unified Glare Rating (UGR)[26] is used:

$$UGR = 8log(\frac{0.25}{L_b}\sum\frac{L^2\omega}{p^2})$$

In this formula, $L_b$ is the background luminance, $L$ is the luminance of each fixture in the direction of the eye, $\omega$ is the solid angle of each fixture at the eye and $p$ is the Guth position index of each fixture. High UGR values indicate high discomfort, low UGR values indicate low discomfort, while a value smaller than ten indicates no discomfort.

**The implementation**
To simulate glare, the intensity of the the light emitted by the virtual fixture that hits a virtual eye, is calculated. We call this $F_{eye}$. The ratio between this value and the average light intensity of the scene, determines the amount of glare that is added to the particular fixture. Hence,

$$g = \frac{F_{eye}}{F_{scene,avg}} \cdot \gamma$$

In which $\gamma$ is an empiric value and $g$ is the glare intensity value. The glare is rendered by constructing a glare matrix $G$ which is an $n \times m$ matrix destribed by:

$$G = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & g & \vdots \\ 0 & \cdots & 0 \end{bmatrix}, n = \lceil \frac{RESx}{g} \rceil, m = \lceil \frac{RESy}{g} \rceil$$

The constructed matrix $G$ is convolved by a glare image like the one that is shown in figure 21.

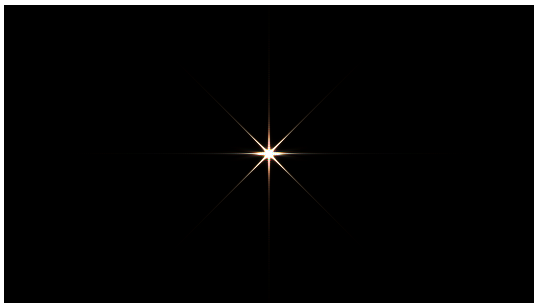


*Figure 21: The glare image used to generate the output result (the same glare image as used in [27]).*

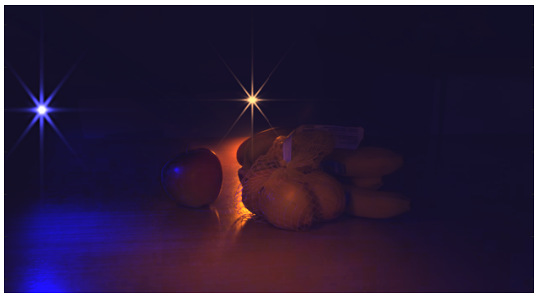


*Figure 22: Glare rendered using the described method.*

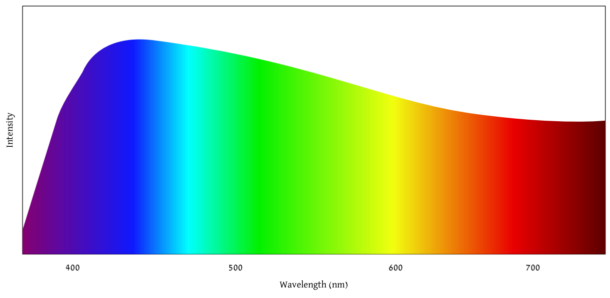
### 6.5.3 CRI

Instead of focussing on the spatial behaviour of the fixtures in terms of light projection, induction of glare etcetera, lighting designers are also interested in the quality of the light emitted by those fixtures. This light quality determines largely how, for example, colors are perceived when lit by the fixture. The metric that can be used to determine the quality of the light when regarding color perception, is CRI. CRI stands for **C**olor **R**endering **I**ndex. 100 is the maximum value for CRI and is achieved by a black body radiator by definition. Determining the CRI of a given light source $L_t$ is performed by illuminating sample colors by light source $L_t$ and a reference light source $L_r$ [28]. This reference is a black body radiator. By measuring the light reflected from the sample colors, and after applying a von Kries transform to chromatically adapt each sample, the Euclidean distance is calculated. This Euclidean distance $\Delta E_i$, is the difference between the light from $L_t$ and $L_r$. The $R$ values are then found using:

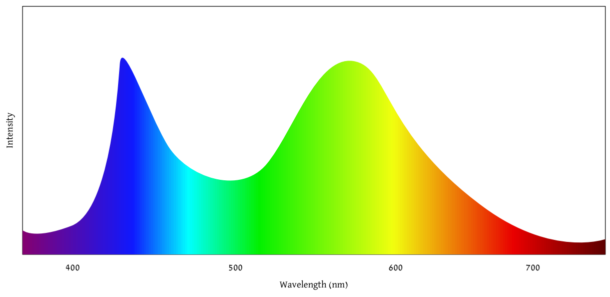
$$R_i = 100 - 4.6\Delta E_i$$

For each sample color $i$. In the end the CRI is found by:

$$CRI = \frac{1}{N}\sum_{i=1}^{N} R_i$$



*(a) Daylight spectrum*



*(b) Typical (cool) white LED spectrum[29]*

*Figure 23: Spectra of two light sources*

**Visualising the CRI**
The CRI can be roughly visualised by allowing more or less color saturation for specific colors in the image, according to the light spectrum, to visualise the deficiency of certain colors in the spectrum of a light source. When observing the spectra in Figure 23a and 23b, one can immediately see that Figure 23b incorporates some ”gaps” at certain wavelengths (400nm, 500nm and 700nm). These ”gaps” is what result in a $<$ 100 CRI. If one would have used the light of this LED and shone it on the sample colors, the wavelengths at the ”gaps” are reflected less than the sample colors that are illuminated by daylight. Hence, if the spectrum of a to be simulated light source is known, the shortcomings in its spectrum can be visualised by decreasing the saturations of colors with certain wavelengths. These shortcomings can be calculated by subtracting the to be simulated spectrum $S$ from the daylight reference spectrum $S_r$. Hence,

$$S_e = S - S_r$$

The resulting spectrum $S_e$ is the error spectrum showing the difference between the two spectra. This is visualised in Figure 24.

$S_e$ can be used to find the wavelengths and hence colors of the light that is not accurately perceived when lighting an object with spectrum $S$. These colors can be artificially dulled in the RGB lighting design image.

**CRI approximization**
Using this technique may give the user an impression of how colors can be perceived when using light of a certain light source. However, since the camera used to capture the scan footage in the first place is not near a spectrometer, accurate measurement of color are not possible. On top of that, image files are stored as RGB pixel values and not wavelengths. This all means that any changes to these RGB values with the purpose of filtering certain wavelengths is not possible. However, this is still a fair approximation that can be used for better perception.

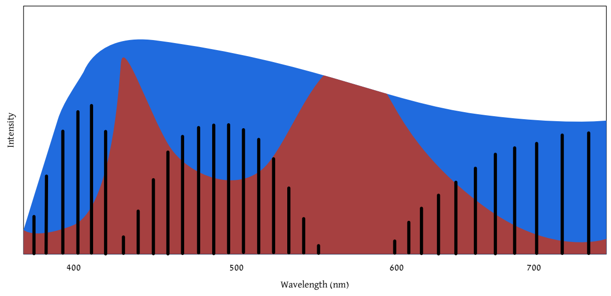


*Figure 24: Difference between the spectra.*

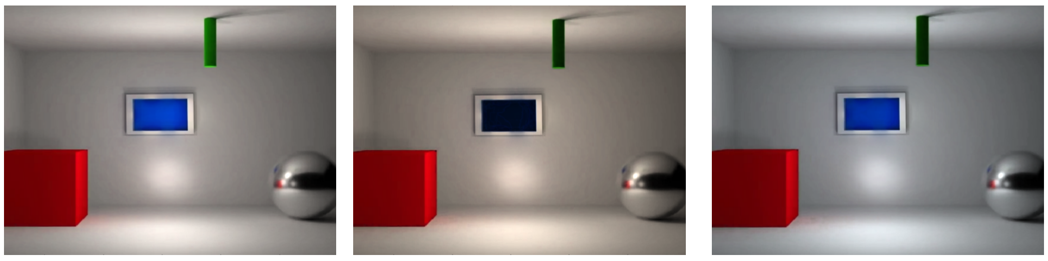


*Figure 25: Visualizing the CRI with the use of an approximated light source spectrum, shows the effect that the light has on certain colors. In figure b) for example, the blue pane on the wall is not accurately perceived because of "imperfections" in the spectrum of the particular light source. Additionally, the color temperature of the light source can be visualised from its spectrum information.*

# 7 The Graphical User Interface

The software that is used to put this work into practice, is entirely written in MATLAB. MATLAB GUIDE [30] is uses for creating the Graphical User Interface (or GUI). This GUI consists of an image viewer, buttons for various settings and functionalities, a table containing information about the virtual fixtures, a dedicated file system and a console for debugging and general information. Moreover, the GUI allowed for the implementation of the final goal of this thesis: the paint-based approach for lighting design.

## 7.1 Functionality

The software written for this work has the following functionalities:

- Loading scan footage
- Processing scans including file management
- Saving of processed scans
- Simulation of point lights
- Simulation of .LDT defined fixtures
- Simulation of simple user defined fixtures
- CRI simulation
- Simulation of indirect light
- Simulation of glare
- Simple light level report generation
- Smart implementation and optimization of painted lighting design

**Loading scan footage** When a scan with a light probe is made, the resulting video file (.mp4) can be loaded.

**Processing scans including file management** The software autonomously processes the loaded footage. It creates a file system containing the following key files at a user defined destination folder:

- AVGF.png contains the trimmed mean of all scanned frames
- Probeloc.dat contains the $(x, y, z)$ coordinates of the probe for every frame
- RPF#.png contains the processed frames. # holds the number of the frame

Figure 26 shows the above described file system.

**Saving of processed scans** The above file system can be saved for later use. When loaded, the footage is ready to use and does not need to be processed again.

**Simulation of point lights** Simple point lights can be simulated as described in the corresponding section.

**Simulation of .LDT defined fixtures** Real world fixtures with an .LDT description can be simulated as described in the corresponding section.

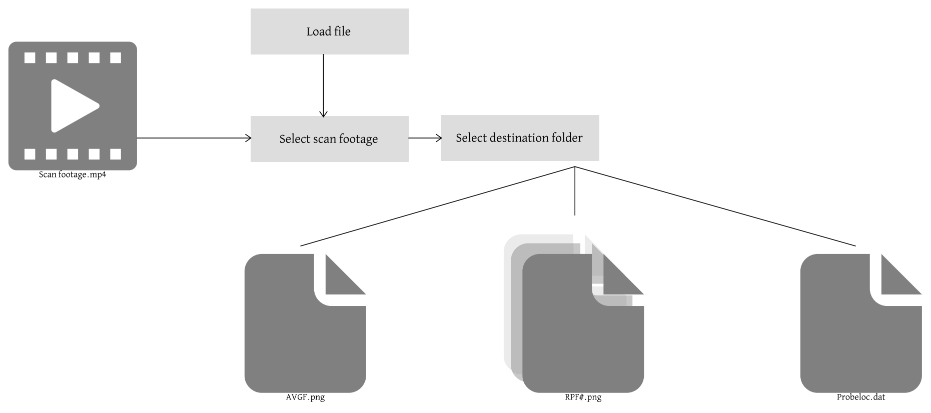


*Figure 26: The file system.*

**Simulation of simple user defined fixtures** Simple user defined fixtures can be simulated. A user can define a fixture by defining its angle.

**CRI simulation** The CRI can be simulated using a spectrum comparison technique. Since it is not possible to accurately simulate the CRI of camera based footage, this functionality is for visualisation only.

**Simulation of indirect light** Indirect light simulation can be turned on and off.

**Simulation of glare** Glare simulation can be turned on and off.

**Simple light level report generation** A simple light level report can be generated containing isolux plots of the scene. The light intensity information is calculated based on the fixture properties and the global scene geometry.

**Smart implementation and optimization of painted light- ing design** The software tries to find the optimal real-world solution to a user painted lighting design.

## 7.2 Overview of the graphical user interface

Figure 27 gives an overview of the designed user interface. The various aspects described in the previous section are shown in detail in Figures 28 to 33. Figure 28 shows the controls for opening and processing scan footage. In the current state of development, only the buttons *Load file* and *Load z-map* are used. When loading a video file, the program asks for a save location at which it saves the processed files automatically. The *Load z-map* button is used for loading a depth map used for the simulation of fixtures (see Section 6).

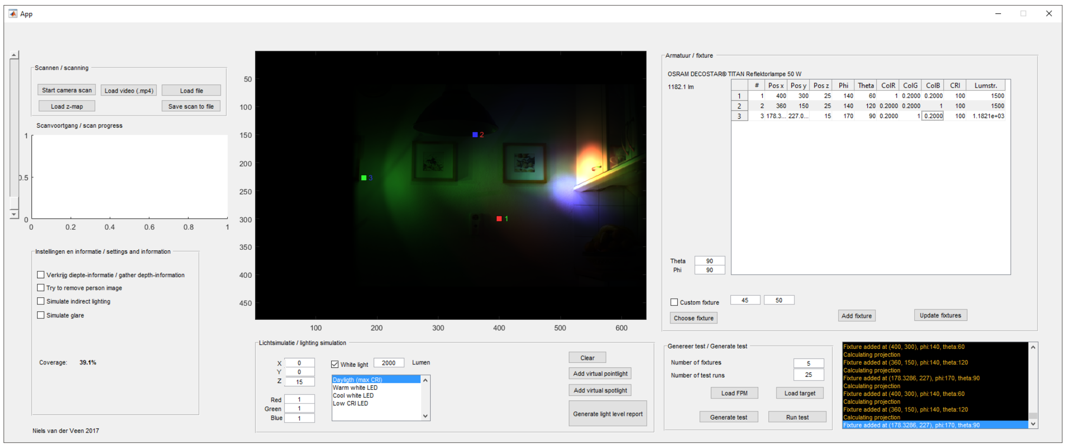
Figure 29 shows the viewer area, in which the user can view his or her design. In this viewer window, the light is simulated. The colored square markers show the position of the fixtures labelled by an identifier number. The color of the marker is the color of the light that is emitted from the corresponding fixture whereas the color of the identifier text is the complement color. This is done to ensure a good readability when fixtures point straight in the $z$-direction and the identifier is surrounded by the colored light its fixture is emitting.

The point light settings (which became more or less obsolete when the fixture simulator was implemented) shows various settings for the simulation of simple point lights and is shown in Figure 30. Also, the CRI simulator is located in this section of the GUI.

Figure 31 displays the section of the GUI that contains the settings for the smart reproduction and optimization of painted lighting designs. *Number of fixtures* defines the number of fixtures that is tested and simulated, whereas *Number of test runs* defines the number of children used per generation in the genetic algorithm. *Load FPM* enables the user to load a Fixture Placement Map and the *Load target* button let the user load a painted design target. *Generate test* asks the user for a database.txt file containing all the fixtures that the program may use in its design. Finally, the test can be started by clicking the *Run test* button. Additionally, not shown in Figure 31, in a later stage of development another button was added named "Initialize". This button initialized the kickstarter that was used in the SIMOLIDE system in a certain development stage (see Section 8.4).

To aid the development process, to inform the user in case of error and to display the action history a simple console was implemented. This console is shown in Figure 32. Clear English sentences are used to inform the user of various actions.

Finally Figure 33 shows the section of the GUI that gives control over the fixtures that are simulated. A user can place a new fixture by first pressing *Choose fixture*. The user is prompted to select a fixtures .ldt definition. Then after pressing the *Add fixture* button, a crosshair appears in the viewer window at the location of the mouse cursor enabling the user to place the selected fixture. All placed fixtures are listed in the fixture table to give a clear overview of the used fixtures and its parameters. Changes to certain parameters can be made directly in the table after which the fixtures can be updated by pressing the *Update fixtures* button. The user can also place a custom fixture by ticking the *Custom fixture* checkbox and entering the [*angle*] parameter.



☐ Verkrijg diepte-informatie / gather depth-information
☐ Try to remove person image
☐ Simulate indirect lighting
☐ Simulate glare

Coverage: 39.1%

X 0
Y 0
Z 15
Red 1
Green 1
Blue 1
☑ White light 2000 Lumen
Daylight (max CRI)
Warm white LED
Cool white LED
Low CRI LED

OSRAM DECOSTAR® TITAN Reflektorlampe 50 W
1182.1 lm

| | # | Pos x | Pos y | Pos z | Phi | Theta | ColR | ColG | ColB | CRI | Lumstr. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 400 | 300 | 25 | 140 | 60 | 1 | 0.2000 | 0.2000 | 100 | 1500 |
| 2 | 2 | 360 | 150 | 25 | 140 | 120 | 0.2000 | 0.2000 | 1 | 100 | 1500 |
| 3 | 3 | 178.3... | 227.0... | 15 | 170 | 90 | 0.2000 | 1 | 0.2000 | 100 | 1.1821e+03 |

Theta 90
Phi 90
☐ Custom fixture 45 50

Number of fixtures 5
Number of test runs 25

```
Fixture added at (400, 300), phi:140, theta:60
Calculating projection
Fixture added at (360, 150), phi:140, theta:120
Calculating projection
Fixture added at (178.3286, 227), phi:170, theta:90
Calculating projection
Fixture added at (400, 300), phi:140, theta:60
Calculating projection
Fixture added at (360, 150), phi:140, theta:120
Calculating projection
Fixture added at (178.3286, 227), phi:170, theta:90
```

*Figure 27: A screenshot displaying the GUI in action.*

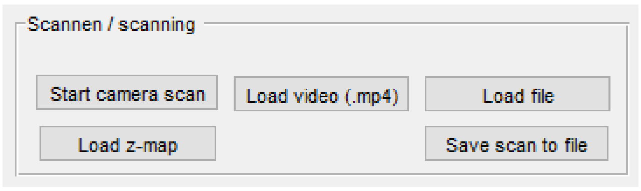


*Figure 28: Loading and saving a scan.*

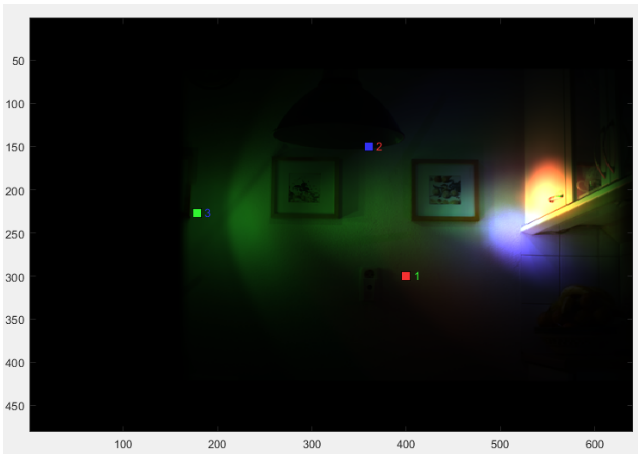
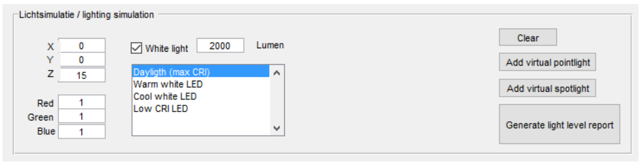


*Figure 29: The viewer displaying the simulated lighting design.*



Lichtsimulatie / lighting simulation

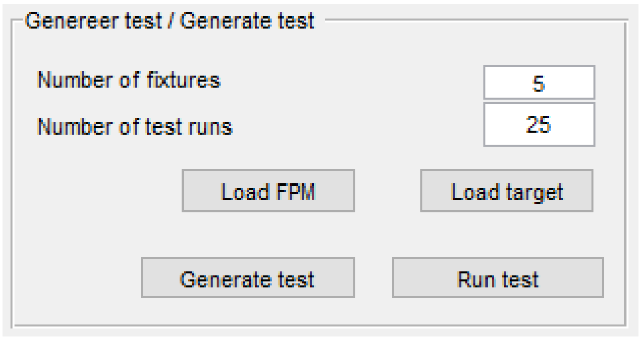
X 0
Y 0
Z 15

Red 1
Green 1
Blue 1

[✓] White light 2000 Lumen

Daylitgh (max CRI)
Warm white LED
Cool white LED
Low CRI LED

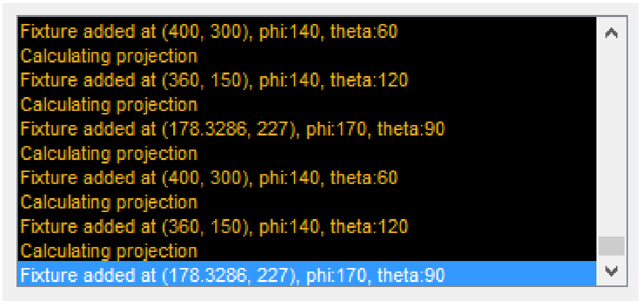
*Figure 30: Settings for simple point light sources.*



Genereer test / Generate test

Number of fixtures 5
Number of test runs 25

*Figure 31: Controls for smart reproduction and optimization of painted lighting design.*

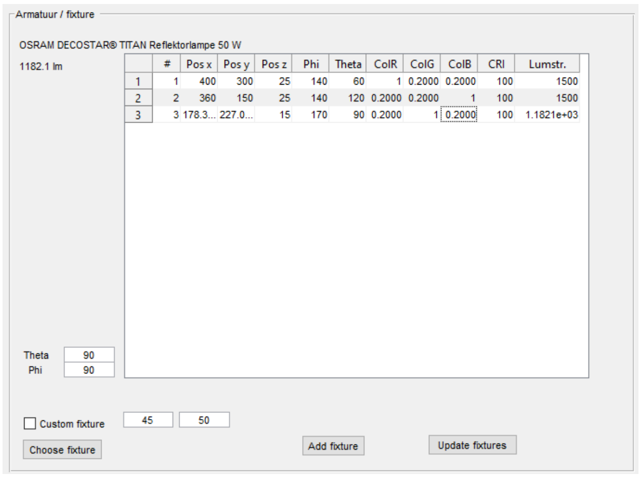


```
Fixture added at (400, 300), phi:140, theta:60
Calculating projection
Fixture added at (360, 150), phi:140, theta:120
Calculating projection
Fixture added at (178.3286, 227), phi:170, theta:90
Calculating projection
Fixture added at (400, 300), phi:140, theta:60
Calculating projection
Fixture added at (360, 150), phi:140, theta:120
Calculating projection
Fixture added at (178.3286, 227), phi:170, theta:90
```

*Figure 32: Debugging console.*

# 8 Smart implementation and optimization of painted lighting design

The ultimate goal in this work is to provide the designer with an intuitive painting tool where he can paint areas of light whereafter the software autonomously finds a solution in



Armatuur / fixture

OSRAM DECOSTAR® TITAN Reflektorlampe 50 W

1182.1 lm

|  | # | Pos x | Pos y | Pos z | Phi | Theta | ColR | ColG | ColB | CRI | Lumstr. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 400 | 300 | 25 | 140 | 60 | 1 | 0.2000 | 0.2000 | 100 | 1500 |
| 2 | 2 | 360 | 150 | 25 | 140 | 120 | 0.2000 | 0.2000 | 1 | 100 | 1500 |
| 3 | 3 | 178.3... | 227.0... | 15 | 170 | 90 | 0.2000 | 1 | 0.2000 | 100 | 1.1821e+03 |

Theta 90
Phi 90

Custom fixture: unchecked; 45; 50

*Figure 33: The fixture settings with fixture table.*

terms of fixture types and positions. In this section the reader is guided through design process of this tool, beginning with a naive approach (Section 8.2) that had some major drawbacks, working up to the final mixed system (Section 8.5).

## 8.1 The problem

In this work, each fixture can be described by its spatial coordinates, two angular values describing its pointing direction, the color and intensity of the emitted light and the emitted lighting distribution. To accomplish the goal that is stated at the beginning of this report, a mathematical problem needs to be solved, which distils in finding the ”correct” fixtures and their parameters for a certain painted lighting design. To clarify some terms that are used throughout this section, we introduce them here:

***Target:*** *the painted lighting design which the user provides as an image file.*

***FPM:*** *Fixture Placement Map, a binary image that shows in black (0) and white (1) where fixtures are allowed to be positioned. This is also provided as an image file.*

***Fitness function:*** *A mathematical formula that calculates a value that indicates the extent to which the lighting visualisation resulting from the parameter set of the fixtures in the design correspond to a target design.*

***Body:*** *a set of variables $\mathcal{S}$ that results in a lighting visualisation that can be observed.*

The problem can be defined as finding the optimal set of variables

$$\mathcal{S} = [s_1, s_2, ..., s_N]$$

in which

$$s_i = [x_i, y_i, z_i, \phi_i, \theta_i, C_i, \Phi_i, P_i]$$

is the parameter set for fixture $i$, $C_i$ is the color vector $[R_i, G_i, B_i]$ and $N$ is the number of fixtures.

To find out what is optimal, a fitness function $\mathcal{F}$ is used, which calculates the fitness of a certain parameter set. Hence the problem can be defined as finding the parameter set $\mathcal{S}$ that maximizes the fitness:

$$\max_{x_i, y_i, z_i, \phi_i, \theta_i, C_i, \Phi_i, P_i \in \mathbb{Z}} \mathcal{F}_{\mathcal{S}}$$

Since the search domain for $\mathcal{S}$ is very large (especially for a large number of fixtures), the problem is rather complex and finding the optimal parameters is certainly not a trivial task. Moreover, the parameters couple in a complex way and optimal values for $x$, $y$, $\phi$ and $\theta$ for example can change dramatically when a different fixture profile is used. Because of the complex and in some ways unpredictable nature of this problem, an algorithm is used that combines a linear solver with evolutionary programming. More or less the same problem is faced in [10], here also a genetic approach was used. However, this problem is different in the sense that no image based lighting techniques were used and that the constraints were of different nature.

In this work, the solver has to work with the image based lighting technique discussed in previous sections. Therefore, the problem comes down to finding a combination of fixtures with different parameters that result to an image that is as close as possible to the target.

## 8.2 Naive approach

The first approach in solving this problem was a plain genetic algorithm. An overview of this genetic algorithm is shown in Figure 34. This initial system uses a simple genetic algorithm to generate random bodies that are automatically tweaked until a valid design is found. In this figure, *Target* contains the painted lighting design. *FPM* is the fixture placement map. *.LDT files* is the fixture database and *number of fixtures* defines the number of fixtures the solver has to work with.

This works as follows. For the first generation, the system generates a (large) population of bodies. These bodies have random parameters for a fixed number of fixtures and uses random fixture profiles from a certain fixture database (see Section 9.1). For each of these bodies, a fitness function calculates the fitness score corresponding to that body. After the first generation, the two highest scoring bodies are used to create offspring. The offspring is again scored by the fitness function and this repeats until a valid implementation is found.

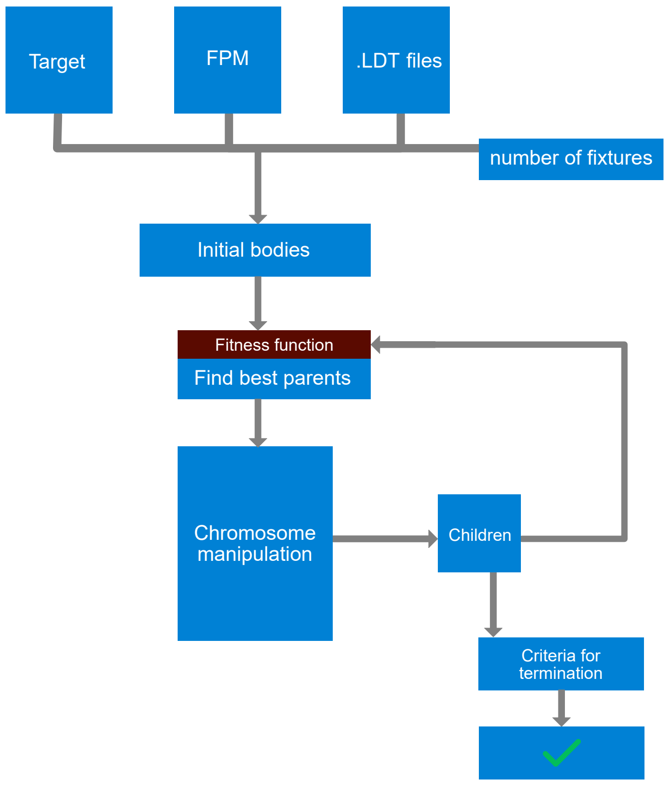


*Figure 34: An overview of the initial system*

### 8.2.1 Chromosomes

***Chromosome:*** *the parameter sets $s_i$ of a fixture $i$.*

Each body exists of at least one chromosome, but can be more when more fixtures are used in the design. The number of fixtures $N$ is equal to the number of chromosomes in a design. Offspring generation is performed by altering the genes encoded on the parental chromosomes, hence changing the values of one or more parameters of a fixture. This chromosome manipulation is done by means of mutation and crossover operations. See section A for more information.

### 8.2.2 Fitness function

As described earlier, the fitness function is used to score each individual body and is hence a quite important aspect of the system. Since initially, a lot of problems arose when using this system, also a lot of different fitness functions were tested in order to improve the performance of the algorithm. As we will see shortly, the problems were not resulting from a seemingly wrong fitness function. Fitness functions that were tested include the cross correlation, the MSE (Mean Squared Error), the SSIM (Structural Similarity index) and the mean absolute error. In the end, the fitness function that is used in the system is defined as:

$$\mathcal{F} = \frac{1}{MSE + 1}$$

### 8.2.3 Limitations of the naive approach

The results obtained with the initial naive approach were very unpredictable; for certain cases the results were reasonable, for others it was a complete disaster. On top of that, the finding of a reasonable solution in a decent amount of time did in no way guarantee that a reasonable solution was found later on. Something was fundamentally wrong about this system and the initial guess was the fitness function, since sometimes a very illogically and unexpectedly high score was given to a solution that was seemingly way off. To test this theory, a lot of different fitness functions where used and compared. Unfortunately, this did not result in a significant improvement in the system performance. As will become clear later, the problem was in the stochastic nature of the genetic algorithm combined with the complexity of the problem. For a run with a population size of $N$ bodies, and a fixture count $F_n$, the size of the search space ($\Sigma$) becomes:

$$\Sigma = (\mathbb{N} \cap [1, 639], \mathbb{N} \cap [1, 479], \mathbb{N}, \mathbb{N} \cap [0, 180], \mathbb{N} \cap [0, 180], \mathbb{R}^3, \mathbb{N}, \mathbb{N} \cap [0, F_{cat}])^{F_n}$$

In which $F_{cat}$ is the number of fixtures in the fixture database and a working resolution of 640px by 480px. The ranges $\mathbb{N} \cap [1639]$, $\mathbb{N} \cap [1479]$, $\mathbb{N} \cap [0180]$, $\mathbb{N}$, $\mathbb{N} \cap [0180]$ come from the spatial and angular degrees of freedom a virtual fixture has in addition to its color ($\mathbb{R}^3$) and intensity ($\mathbb{N}$). This search space is highly dimensional and very large, hence the probability of finding the optimal parameter set by randomly changing the parameters is close to zero.

## 8.3 The improved approach

Since the naive approach worked well in some cases but could not converge to a solution in a realistic amount of time, something had to be improved. The difficulties with converging to a solution lie in the huge search space the system has to go through, especially for problems using more than two or three fixtures. A solution to this problem is carried out in the form of a "kickstarter" for the genetic algorithm. This kickstarter provides the genetic algorithm with a better starting position in order to decrease the time until the algorithm converges. On top of that it should also make the average solution error smaller. When the genetic algorithm starts with a fixture setup that already more or less resembles the desired design, only some minor tuning has to be carried out by the system for perfecting the result.

## 8.4 The kickstarter

The kickstarter is build around a linear system also used in face-recognition systems; it tries to find elements that are similar to certain parts of an image. For face-recognition systems, this results in finding the linear combination of images from a training set of face images [31]. While the system in this work does not incorporate faces, the idea of finding a linear combination of images that result in the final goal can be perfectly applied to this system. The reason for this resides in the very nature of the discussed technique of lighting simulation; a lighting design is in fact a combination of different "light samples" in a room or scene. As discussed in the beginning of this thesis, a probe scan results in different probe images that provide a sample of light at a certain location in the scene. A linear combination of these samples can be found after which back-traced location and color information can provide the genetic algorithm with a decent starting position. Now that we know the idea after the kickstarter, we are going to dive into its implementation.

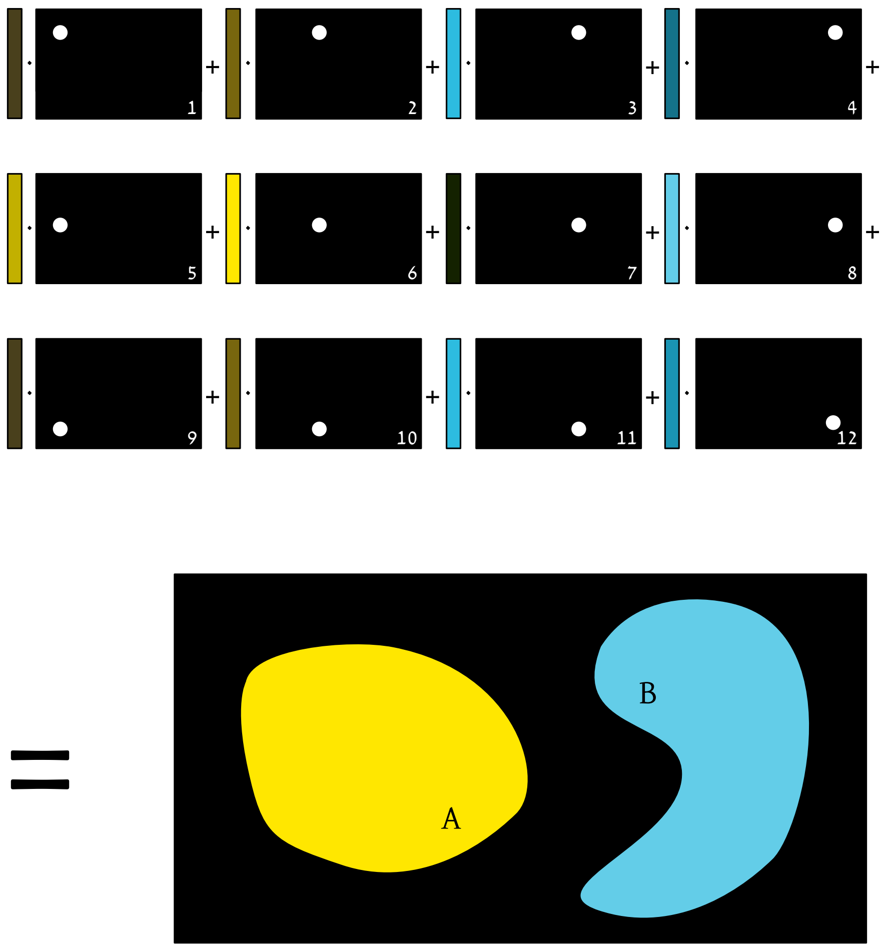


*Figure 35: The idea behind the kickstarter. On the right, the target is shown. On the left, the probe images are shown together with a color weight.*

### 8.4.1 The implementation of the kickstarter

The kickstarter works by selecting initial fixture positions and colors in order to give the genetic algorithm a starting set to work with. This is done by analysing the frames that are captured by the camera in the scanning process and selecting the frames that could be of use in the design. In Figure 35, this is illustrated. On the bottom (after the "=" sign), the target is shown. It consists of two parts. Part one, a yellow circular illuminated area, and part two, a blue elliptical area leaning more to the right. On the upper side of the image, the captured frames of the light probe are shown. Note that in this simplified figure, the actual light emitted from the light probe interacting with the scene, is not shown. In order to construct the lighting design shown on the bottom side with the lighting information on the upper side, each frame can be multiplied with a certain color vector $[r, g, b]$, here shown as a colored rectangle. For example, in this situation the illumination from the probe in images 5 and 6 of Figure 35 corresponds to part $A$ of the design, while images 3, 4, 8, 11 and 12 correspond to part $B$. The correspondence result in a color vector that describes the color of the light at those positions. These color vectors is what we need in order to kickstart the algorithm and can be found by solving a linear system.

Let $\alpha_i$ be the color vector associated with frame $i$ and let $T$ be the target (painted) lighting design). $\omega_i$ is the pixel matrix of frame $i$. Now,

$$\sum_{i=1}^{N} \omega_i \cdot \alpha_i = T$$

in which $N$ is the total number of frames. When all frames $\omega_1 ... \omega_N$ are concatenated into a big matrix

$$\Omega = \begin{bmatrix} \bar{\omega_1} & ... & \bar{\omega_N} \end{bmatrix}$$

In which $\bar{\omega_i}$ are all the pixels of $\omega_i$ color channel wise concatenated in a vertical vector. Hence:

$$\bar{\omega_i} = \begin{bmatrix} \omega_{i,Red} & \omega_{i,Green} & \omega_{i,Blue} \end{bmatrix}$$

Then,

$$\Omega \cdot \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix} = T$$

In which $\alpha_i$ is the color triplet $\begin{bmatrix} \alpha_{i,Red} & \alpha_{i,Green} & \alpha_{i,Blue} \end{bmatrix}$

This can be written as:

$$\Omega \cdot \alpha = T$$

and now $\alpha$ can easily be found by:

$$\alpha = T\Omega^{-1}$$

The vector $\alpha$ now describes for every frame in $\Omega$ the color value that should be multiplied by that frame in order that the sum of those multiplications result in target $T$. It is perfectly possible that this computation results in a vector $\alpha$ that is very dense. In other words, almost all frames contribute to the final design with some contributions being very small and almost negligible. To solve this problem and to achieve a more sparse matrix, multiple regularization algorithms are used to solve for $\alpha$. These algorithms are referred to as:

- LASSO
- l1_ls_nonneg
- lsqnonneg

**LASSO** *LASSO* is an acronym for Least Absolute Shrinkage and Selection Operator[32]. It is an $L_1$ regularization algorithm that performs regularization and feature selection in order to reduce the number of nonzero coefficients in the calculated $\alpha$ matrix. This increases accuracy and can be very useful for system where little observational data is available[33]. Because it reduces the number of nonzero coefficients, it also makes the calculated $\alpha$ matrix a lot sparser.

Formally, LASSO solves the following problem:

$$\min_{\alpha_0,\alpha}\left(\frac{1}{2N}\sum_{i=1}^{N}(y_i - \alpha_0 - x_i^T\alpha)^2 + \lambda\sum_{j=1}^{p}|\alpha_j|\right)$$

Here, the most important variables are $\alpha$, the problem solution and $\lambda$. The way LASSO ($L_1$ regularization) works, is to force the sum of the absolute values of $\alpha$ to become less. $\lambda$ is a parameter that controls the $L_1$ regularization. As this value increases, the number of nonzero components in $\beta$ decreases. Hence the result becomes more sparse. The MATLAB implementation of the LASSO library function calculates the largest value for $\lambda$ that results in a nonnull model [34]. While the *LASSO* algorithm results that are smaller than zero, but the model does not, the $\alpha$ matrix is processed afterwards to eliminate values smaller than zero.

**l1_ls_nonneg** [35] As the practical use of the *alpha* matrix does not allow negative values, *LASSO* might not be the most suitable algorithm for this purpose. For that reason, another regularization algorithm was tested. The *l1_ls_nonneg* algorithm is a MATLAB implementation of a $L_1$ regularization algorithm with the constraint that the result matrix $\alpha$ cannot have entries that are negative. As this is the case, no further processing that eliminates negative values is necessary anymore.

**lsqnonneg** This algorithm solves the linear least-squares problem with the constraint that the result cannot be negative. The algorithm is not designed to give a sparse output and is therefore probably not very useful for the purpose of this system. The inclusion of this algorithm is more for comparative reasons.

As described, three initialization algorithms have been implemented. Although there is probably a correlation between the scene and the algorithm, it is thought that there is one algorithm that performs best for all cases. This algorithm is probably the *l1_ls_nonneg* algorithm since it is based on $L_1$ regularization and has the additional constraint that ensures the result is positive. To validate this hypothesis, a series of three tests have been carried out which are described in Section 11.2.

## 8.5 A mixed system

While using a linear system as an initialization step for the genetic algorithm improves the performance significantly, the performance can potentially be further improved. By using a combination of the genetic algorithm and a linear system, the best of both worlds can be combined. On top of that, using more competitors helps the exploration of the solution space as can be seen in Figure 36. The solution space can be visualized as a landscape with crests and valleys. In this case a crest correspond with low fitness and a valley with high fitness. Essentially, the ultimate goal is to find the deepest valley in an unknown landscape. Both the naive approach and the improved approach generated a random population after which the best was chosen and processed by the genetic algorithm to improve the fitness even more. In this scenario it is possible to find a local minimum, but the probability of finding the global minimum in a large solution space is close to zero (Figure 36a). The mixed system uses multiple competing bodies, this way exploring more parts of the solution space and hence increasing the probability of finding the global minimum (Figure 36b).

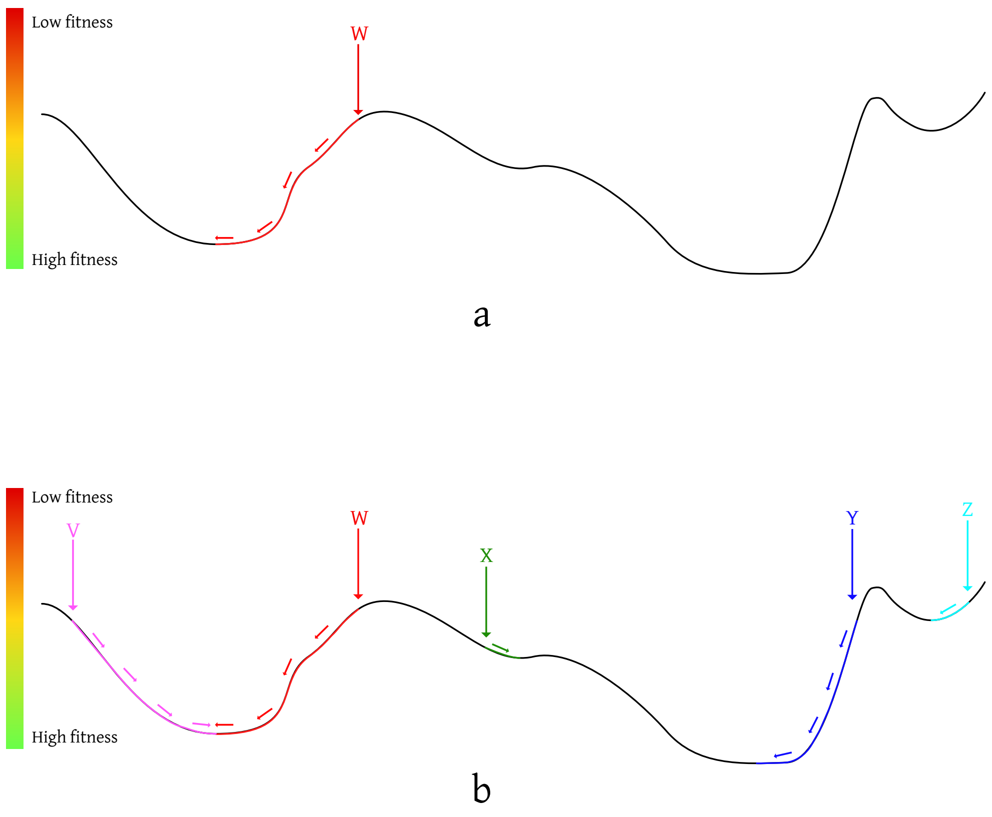


*Figure 36: Part a of the figure shows the exploration procedure used in the naive approach. Part b shows the exploration procedure used in the mixed approach. The naive approach only finds the local minimum by exploration (solution W) while the mixed approach also finds the global minimum (solution Y).*

The idea is as follows: the genetic algorithm generates random $n$-fixture bodies (in which $n$ is a user provided parameter for the solver). The best $k$ bodies are chosen (based on fitness) and are stored. These stored bodies are separated into $n$ 1-fixture bodies. These 1-fixture bodies are then linearly combined using the linear solver, using the *l1_ls_nonneg* solver described in the previous section, which determines the color and intensity values $\alpha$ for each fixture in the body.

$$\alpha = T\Omega^{-1}$$

with

$$\Omega = \begin{bmatrix} \bar{\omega_1} & \dots & \bar{\omega_n} \end{bmatrix}$$

and $\bar{\omega_i}$ the separated 1-fixture images in an $n$-fixture body. Then, these values are used together with the, already known, spatial and angular parameters of a fixture to generated a new population. Hereafter the process is repeated. See Figure 37 for a visualisation.

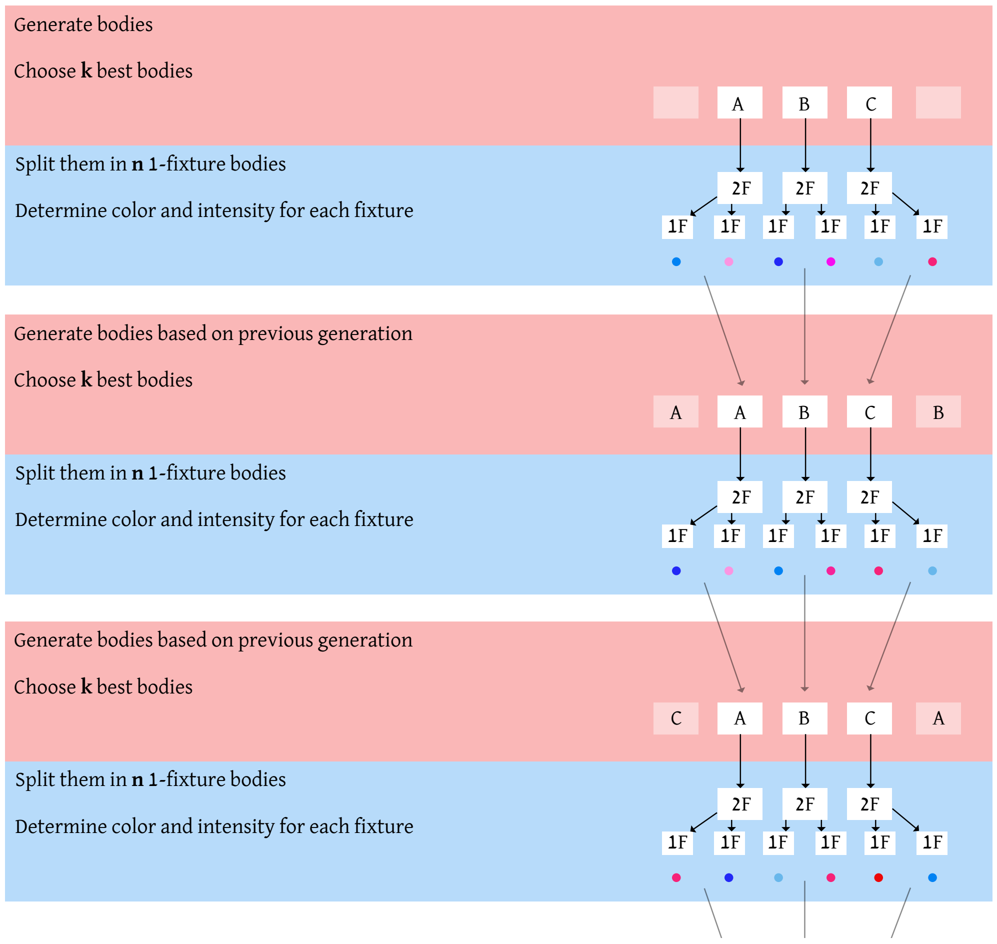

Generate bodies
Choose **k** best bodies
Split them in **n** 1-fixture bodies
Determine color and intensity for each fixture
Generate bodies based on previous generation
Choose **k** best bodies
Split them in **n** 1-fixture bodies
Determine color and intensity for each fixture
Generate bodies based on previous generation
Choose **k** best bodies
Split them in **n** 1-fixture bodies
Determine color and intensity for each fixture
A
B
C
2F
1F

*Figure 37: The mixed approach. The red part is genetic, the blue part is solved with a linear system. This example is for a problem with 2 fixtures ($n = 2$).*

Using this approach ensures that each part of the process (namely genetic and linear) does what it can do best; the genetic part generates and modifies spatial and angular parameters and calculates the fitness, while the linear part calculates the intensity and color values for these parameters. The added feature of having the $k$ best bodies competing against each other helps exploring the solution space. Every body from 1 to $k$ has a unique identifier that is maintained throughout the iterations (visualized in Figure 37 as $A$, $B$, and $C$). This way, the variety of the solutions is not destroyed when the algorithm chooses a number of bodies that are all but a small variation to an original "winner".

The mixed approach on the other hand selects the best $k$ bodies and genetically modifies these to a new generation in which also the best $k$ bodies are chosen. This way, on multiple points (arrows in the figure) "seeds are planted" that can evolve. Still an emphasis is put on keeping the best "seeds" leading to some seeds being eliminated while others are passed to the next round.

## 9 Testing the SIMOLIDE system

The system as described in this thesis is comprehensively tested to validate its functionality and to achieve results that can be used for comparison between the different algorithms. The tests can roughly be divided in three sections:

- **Simple and straightforward** Fixtures can be placed everywhere and the design is simple. Using monochromatic or two-tonal designs that describe clearly delimited areas
- **Complex** Fixtures can be placed everywhere but the design is more detailed and can include more than two colors. Color boundaries can be adjacent to each other.
- **Realistic** Placement of fixtures are constraint to a certain position and cannot be placed at any arbitrary location. The designs are detailed and color boundaries can be adjacent to each other.

Each test is carried out by defining a target design, specify the fixture placement map and let the system solve for fixture parameters. The Mean Squared Error (MSE) of the system is recorded per generation and is a metric for the performance of the system. For each testing section as shown above, two or more sets of target designs and constraints were defined. These are shown in Table 1.

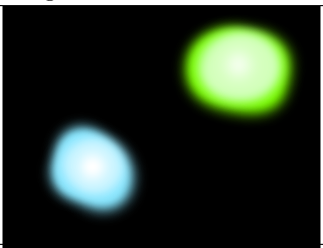
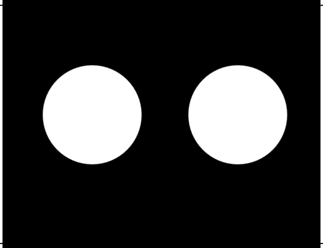
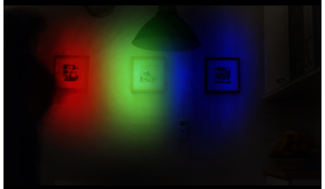
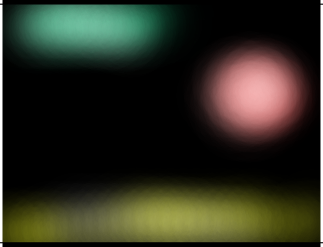
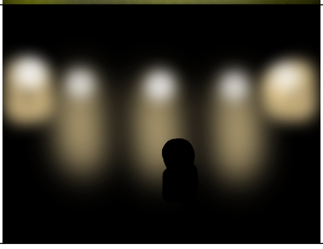
| ID | Test section | Target | Description |
|---|---|---|---|
| 1 | Simple and straightforward |  | Two distinct lighting locations that are clearly separated from each other. |
| 2 | Simple and straightforward |  | Two distinct hardly bounded lighting locations that are located closer to each other. White light without color information. |
| 3 | Complex |  | An RGB projection that shows vertical light strokes that are adjacent to each other. |
| 4 | Complex |  | Three distinct lighting locations of which two illuminate a horizontal surface (taking into account perspective). |
| 5 | Realistic |  | A realistic scene in which the walls of the room are washed by warm white light from top to bottom. |

*Table 1: The different test cases that were used to validate the functionality of the system*

**Note: unless stated otherwise, in all tests, CG scanning footage was generated in order to test solely the light painting system. This scanning footage is a rendered probe sweep in an artificial room created in Blender.**

A more detailed explanation of the different test cases is written below.

The test cases in Table 1 were designed to each test a different aspect of the system and are also increasingly more complex.

Test case 1 test the system's ability to "think out of the box" by placing fixtures at an angle and illuminating different parts of the scene with a different color. Trivial solutions were prevented by choosing an appropriate FPM.

Test case 2 test the ability to create straightforward projections without color and in relatively close proximity to each other.

Test case 3 is aimed at close proximity and testes the ability to maintain hard edges while instructed to do so.

Test case 4 is all about perspective and challenges the system to correctly find the aiming vectors of the fixtures in order to get correct projections on floor, ceiling and back wall.

Finally, test case 5 is a more realistic case and tests a combination of the above and verifies whether the system can come up with a realistic solution.

## 9.1 Fixture database

Since the genetic algorithm is based on random mutations of certain parameters, also the fixture that is used should in some way be randomly varied. A completely randomized fixture or random mutations to the light distribution for instance is unwanted, because this could result in unrealistic fixtures which misses the entire point of this kind of lighting design. Therefore, a fixture database is used in which several fixtures are stored with a certain identifier. Now the identifier can be mutated providing a way to use the fixture parameter in the genetic algorithm while maintaining realism and feasibility.

The fixture database contains seven fixtures and these are tabulated in Table 2. The eulumdat files of the fixtures have been downloaded from *https://lumsearch.com/en#0* and the light distribution plots in Table 2 are created using *LDT Editor*. Both are provided by *Dial*[2].

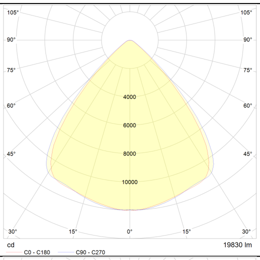
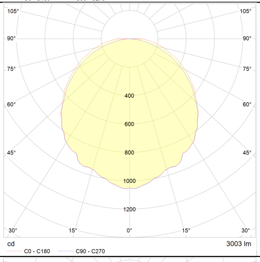
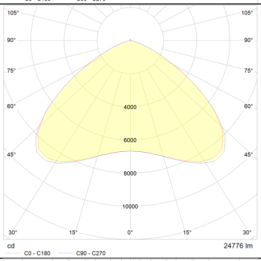
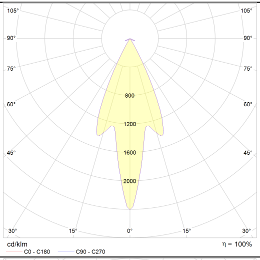
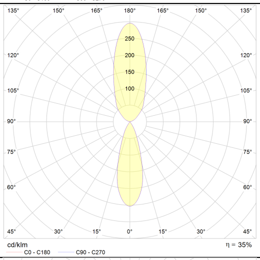
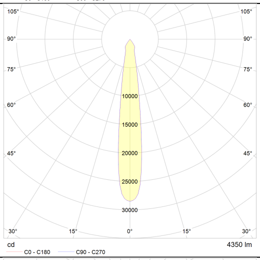
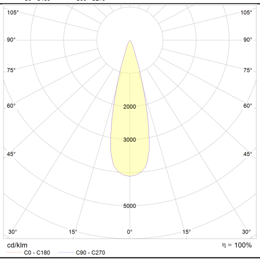
| ID | Name | Light distribution | Description |
|---|---|---|---|
| 1 | Noxion LED High-bay Concord 150W 4000K CRI80 |  | Wide light cone fixture for equal lighting of large surfaces. |
| 2 | Tanek LED |  | A very wide light cone fixture with a circular shaped and uneven light distribution. |
| 3 | Noxion LED High-bay Pro-Clean (HACCP) 200W 4000K CRI ¿80 |  | Very wide fixture with a weaker center projection. |
| 4 | HALOSPOT 111, Alu-Refektor 75 W 12 V |  | Narrow fixture with a brighter center projection. |
| 5 | KAP SURFACE WALL BLUE QT-14 2X33W ES |  | A narrow fixture emitting light on both ends. |
| 6 | Tarek S |  | A very narrow and high intensity spot fixture. |
| 7 | CONE 204.27.35.2/HBT |  | A spotlight fixture. |

*Table 2: The fixtures in the fixture database.*

# 10 System overview

The methods described in the previous sections are combined to create the system described in this section. A general overview of the system is shown in Figure 38.

When looking at the system overview, the biggest element that can be recognized is the *visualization engine*. This part can be seen as the most important element and is responsible for using the data that is gathered or calculated to generate the simulated lighting design image. As can be seen, this process can be broken up in four sub-processes:

- Triangulation
- Combination
- Masking
- Extras (being glare visualisation, indirect lighting and the CRI simulator)

These four sub-processes are discussed earlier in this thesis and hence are referred to for more information. The visualisation engine takes its information form the *file system builder*, *light distribution calculation* and the *fixture parameters*. The file system is more in-depth discussed in Section 7.1. The light distribution calculation is taking spatial information in the form of a depth map, together with the photometric information of a certain fixture (in the form of an EULUMDAT file) and uses them to calculate the light distribution of a certain fixture in any place in the scene. For more information about the light distribution calculation, see Section 6.2. The fixture parameters are, as the name suggests, the parameters of the virtual fixtures that are placed in the scene and describe the fixture in such way that it can be simulated by the visualisation engine. These fixture parameters are in the form of:

| ID | $x$ | $y$ | $z$ | $\phi$ | $\theta$ | R | G | B | CRI | $\Phi$ | $P$ |
|---|---|---|---|---|---|---|---|---|---|---|---|

In which ID is the identifier of the fixture, $x$, $y$ and $z$ are the spatial coordinates of the fixture, $\phi$ and $\theta$ are the pointing angles of the fixture, R, G and B represent the color value, $\Phi$ is the luminous flux of the light the fixture emits and $P$ is the profile i.e. the lighting distribution of the fixture. The CRI parameter is not used at the time of writing but could be used to provide the CRI simulator with information for simulating the CRI of individual fixtures.

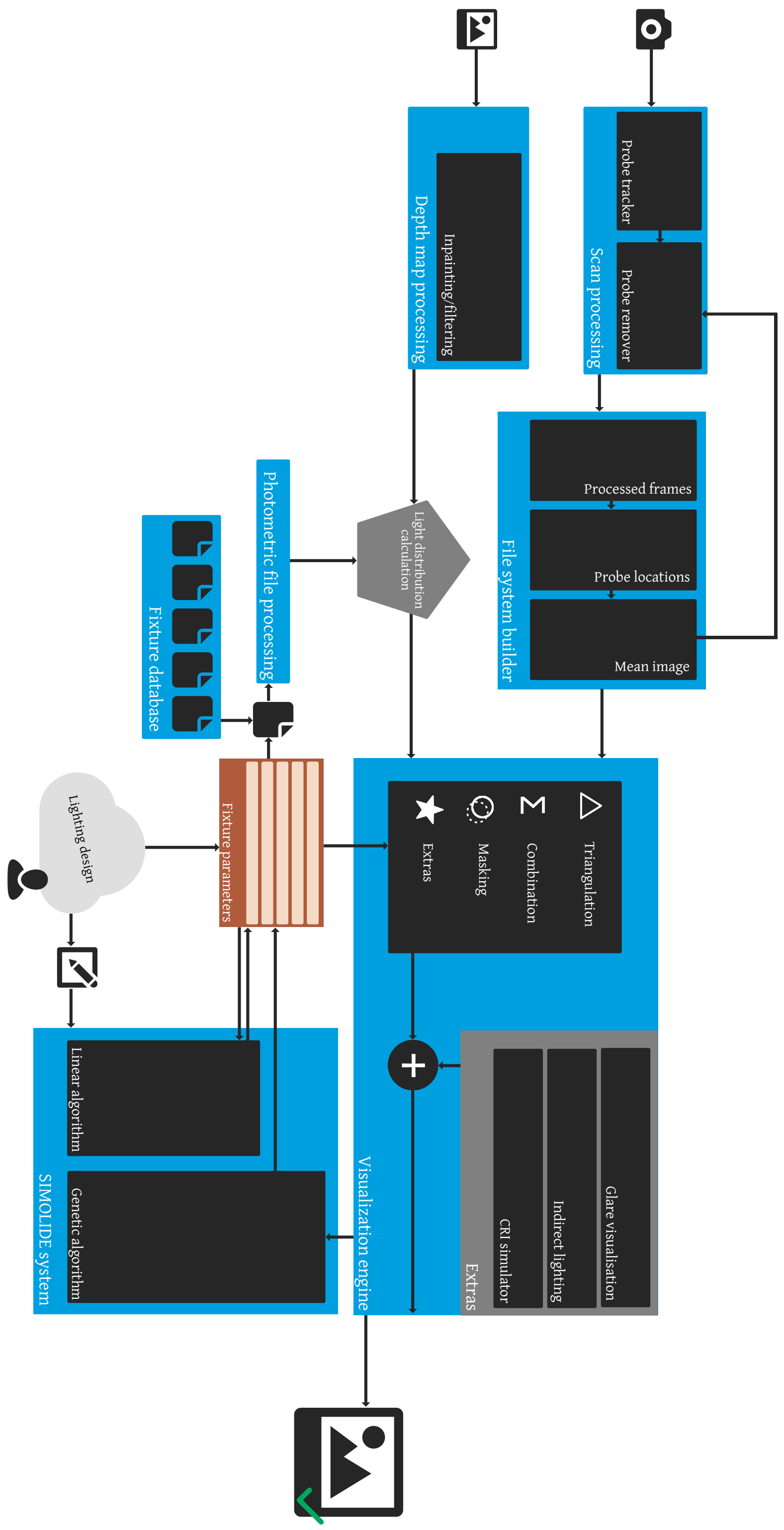


Figure 38: An overview of the of the entire lighting simulation system

# 11 Results

## 11.1 Obtaining the $Z$-map

### 11.1.1 Exploiting stereo vision

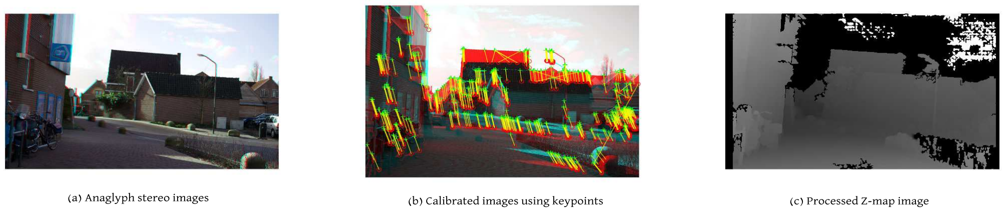
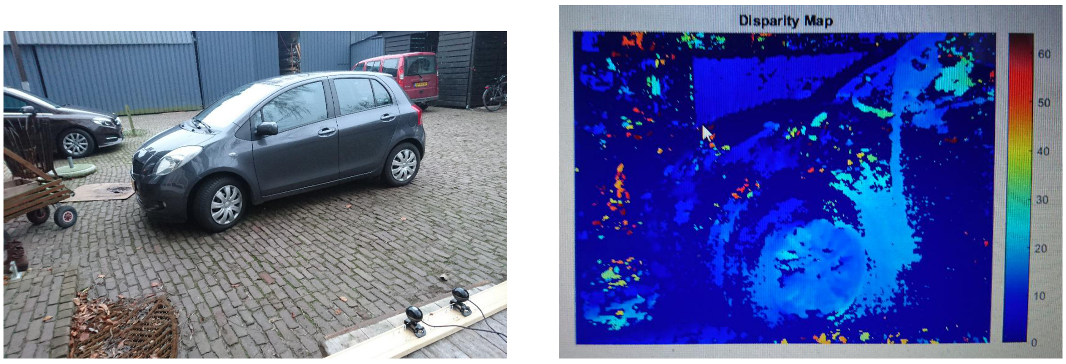

(a) Anaglyph stereo images
(b) Calibrated images using keypoints
(c) Processed Z-map image

*Figure 39: The result obtained by using a Nikon D5100 to create a Z-map.*



*(a) The two-camera setup.* *(b) A Z-map obtained by this setup.*

*Figure 40: The two camera setup with the obtained result.*

As can be seen in Figures 39 and 40, the resulting depth map image is quite accurate for textured walls. For solid colored surfaces, or surfaces with less contrast or reflections, this approach is problematic. This can for instance be seen on the relatively solid colored sky and bands above the supermarket logo and the car paint. The reason for this behaviour results from the used technique, which relies on matching features between two images and measure the distance between those feature. For solid colored surfaces, this is not very accurate since there are (almost) no features that can be matched. For reflections, the features that can be detected are false because they are not features of the surface itself.

### 11.1.2 Obtaining depth information with the use of a Structure Sensor

Using the stereo vision approach, a disparity map could not be calculated because of the lack of key points (features). The surface did not have a texture that could be used to match certain key points from the left eye camera to the right eye camera. However, when the Structure Sensor was used, it found a good estimation of the depth of this scene without major issues.

### 11.1.3 Using 3D-modelled content

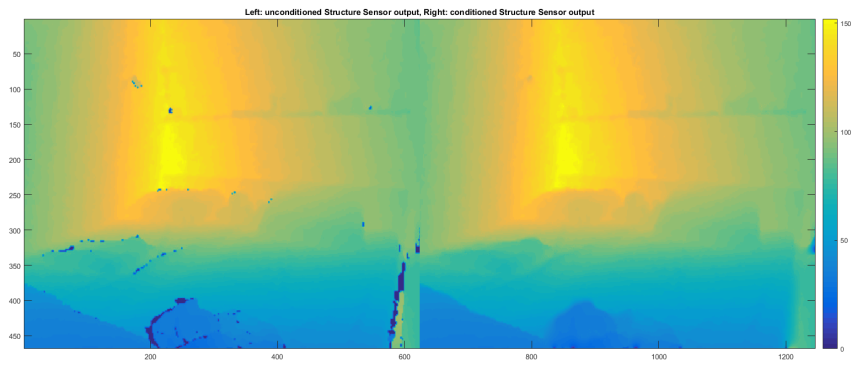
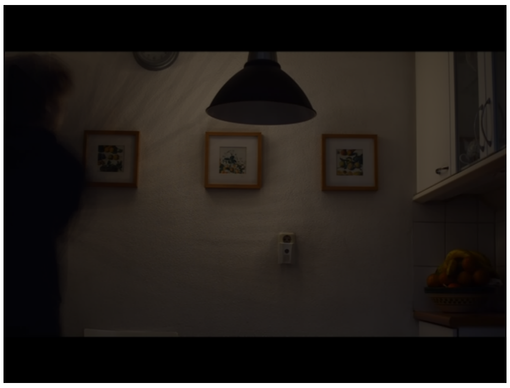
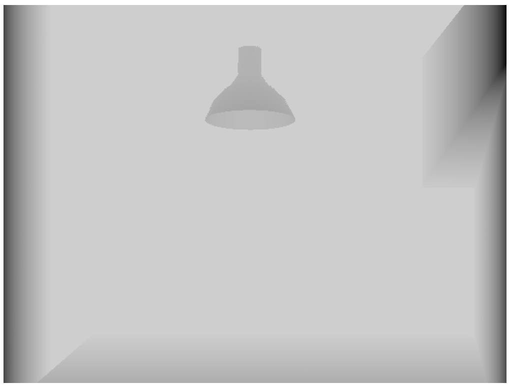


*Figure 41: The depth map generated by the Structure Sensor. The scene depicts a bed with solid white heavily pleated sheets. The left image shows the raw depth map and the right image is processed to reduce noise and holes.*



*(a) The kitchen scene*



*(b) The generated Z-map.*

*Figure 42: It takes approximately 5 minutes of work to create the Z-map for this kitchen scene.*

## 11.2 Comparing the initialization algorithms

Three tests have been carried out concerning test cases 1, 3 and 4 (Table 1. Cases 3 and 4 are unconstrained and case 1 is constrained. The tests have been performed with a generation size of 200 children to reduce the effect of chance. In each test, the system in conjunction with different initialization algorithms was tested together with a test showing the performance of the system without the initialization step. Before this test, a the initial fitness (maximum fitness of the first generation) is measured. This is done, for cases 1, 4 and 5. To purely compare the initial setup of the fixtures by the system (without involving the genetic algorithm), no placement constraints were active. For each algorithm, ten initial generations with 50 children each, were tested, of which the mean was taken to achieve the fitness.

In Figure 43, the first generation fitness is shown for the different algorithms.

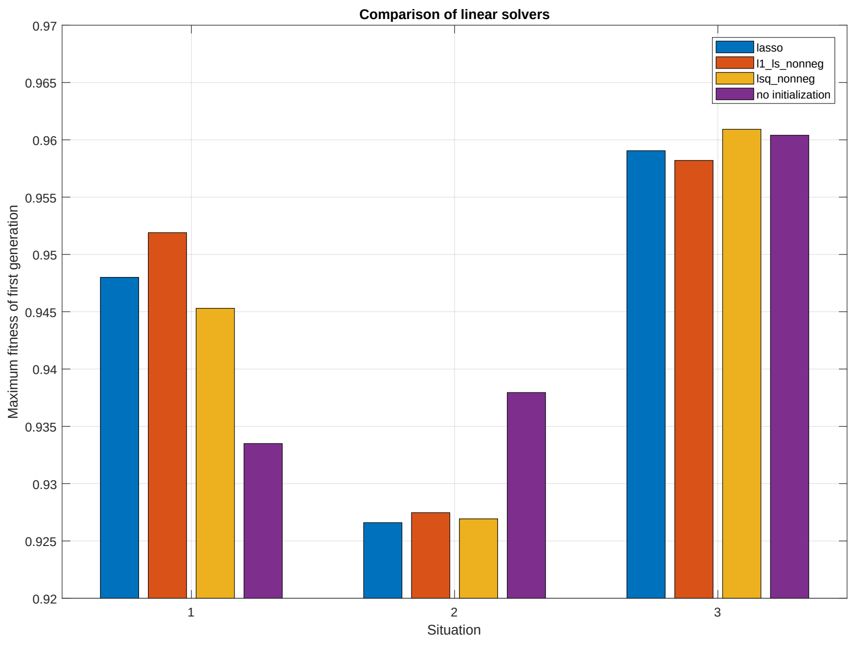


*Figure 43: The performance of the first generation for the different algorithms, averaged over ten runs. Situation 1 corresponds to case 1, situation 2 corresponds to case 5 and situation 3 corresponds to case 4.*

The results are discussed in Section 12.3

## 11.3 The SIMOLIDE system

The final functionality of the system is validated by running a series of tests (as shown in Table 1 with the mixed implementation. The results of this validation step are shown in this section. For all the test cases the solver is run with no extras, like glare, indirect light or CRI simulations. For every case it is shown what the scene was, what the painted lighting design was, whether any spatial constraints were put on the placement of the fixtures[4] and the values of the simulation parameters. Also, the positions of the fixtures in the result is showed in a both graphical and tabulated form. The latter provides in-depth details about the result.

### 11.3.1 Test case 1

Test case 1 was designed to validate the solving capabilities of the system. Two colored light spots have been drawn and spatial constraints were put on the fixtures to discourage the trivial solution.

[4]A white image for the fixture placement map means that fixtures can be placed anywhere.

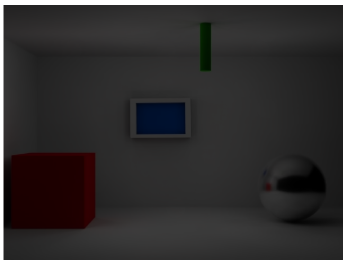


*Figure 44: Scene*

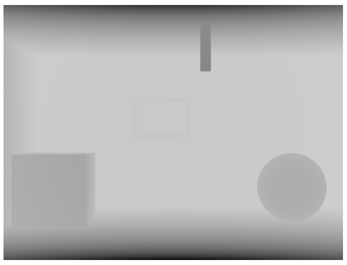


*Figure 45: Z-map*

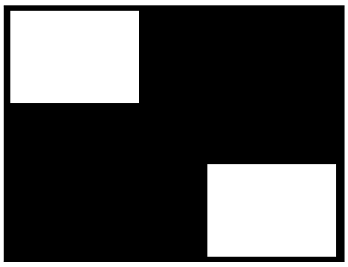


*Figure 46: Constraints*

*Figure 47: Test case 1*

**Simulation parameters**

| Parameter | Value |
|---|---|
| Number of fixtures | 2 |
| Number of iterations | 45 |
| Population size | 100 |

*Table 3: The simulation parameters used in test case 1*

**Target**

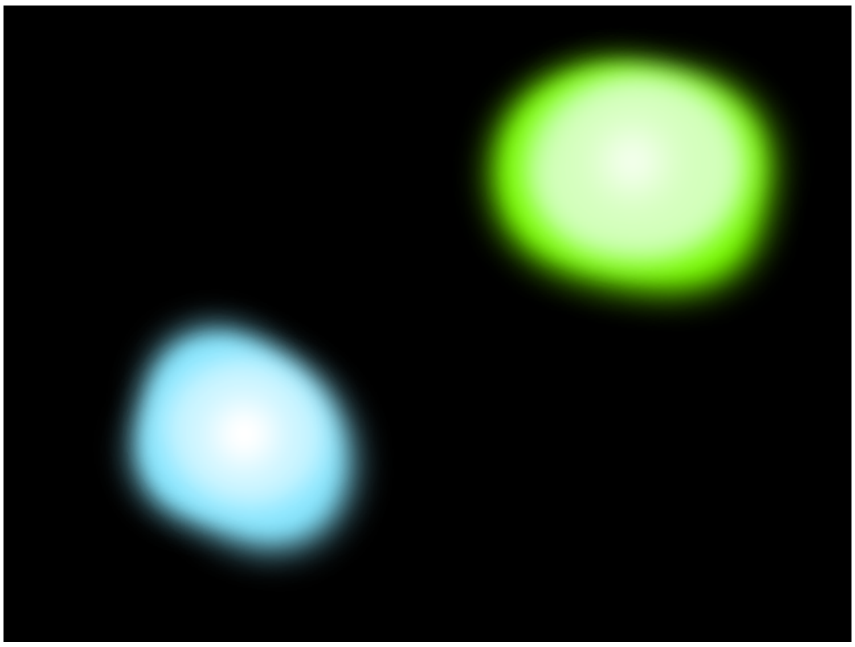


*Figure 48: The target of test case 1*

### Image result

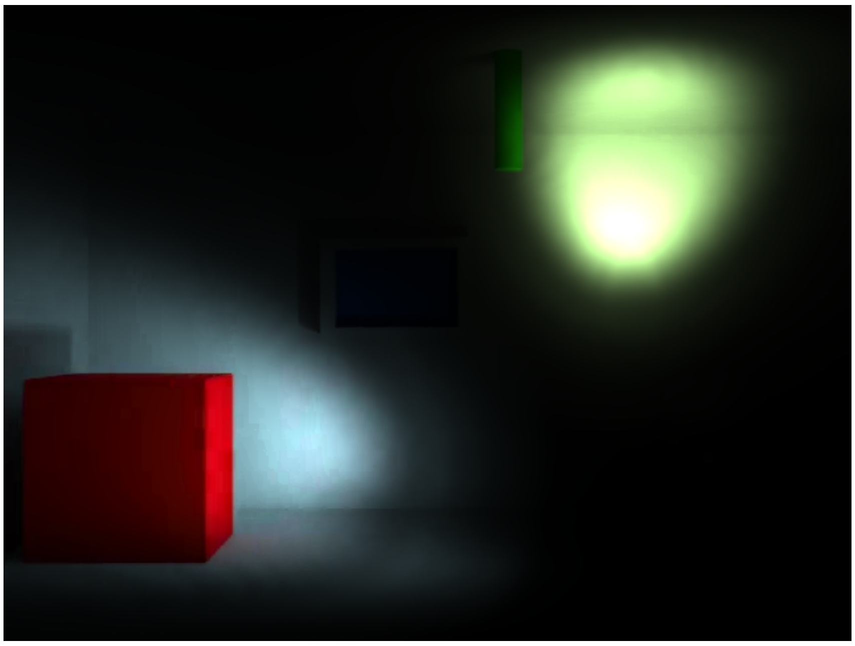


*Figure 49: The calculated result of test case 1*

### Fixture placement

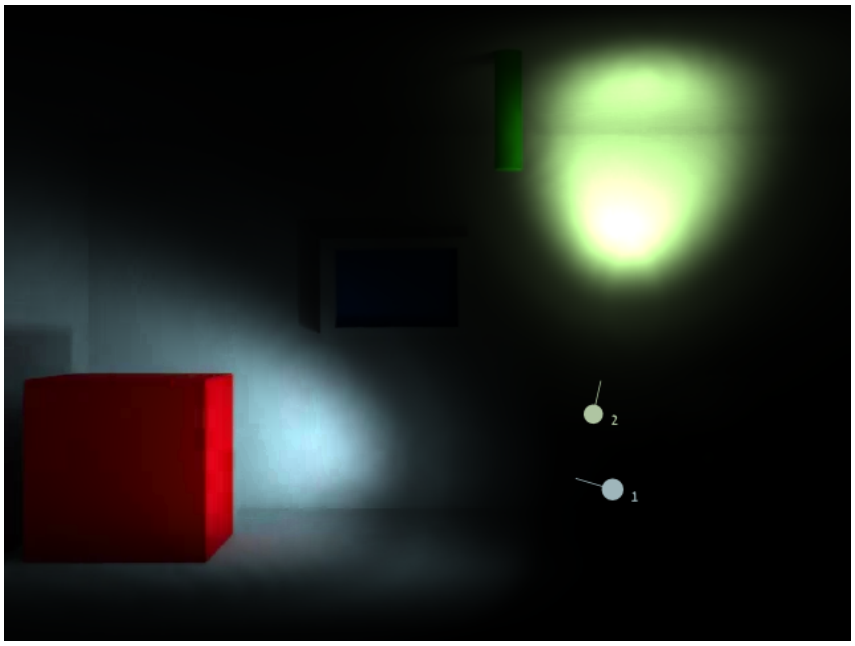


*Figure 50: The placement of the fixtures by the algorithm for test case 1*

### Parameters result

| ID | $x$ | $y$ | $z$ | $\phi$ | $\theta$ | R | G | B | $L$ | Profile |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 450 | 374 | 13 | 22 | 81 | 2.8536 | 3.8677 | 4.1173 | 1000 | 7 |
| 2 | 446 | 299 | 17 | 100 | 11 | 2.1994 | 2.8619 | 1.7574 | 737 | 7 |

*Table 4: The fixture parameters of the found solution for test case 1*

**Discussion**
The target and the fixture placement map of test case 1 were set up on purpose to force the solver to come up with a solution different from the trivial solution, namely two fixtures pointing in the $z$-direction. The solver correctly found that placing two fixtures with a steep angle results in the requested design. The choice of the used fixtures was a ”normal” spotlight fixture with an angle of approximately 30°. This fixture was chosen for both the fixtures, which is as can be expected. As can be seen in Table 4, fixture 1 is the blue spotlight illuminating the lower left corner of the image. The shape of the light projection is resembling the shape in the painted target, except for the light falloff on the left. This, however is negligible. Fixture 2, the green spot, points upwards to illuminate a part of the ceiling to create the requested projection.

### 11.3.2 Test case 2

This case was designed to find whether the solver can come up with a trivial solution. Without constraints and with a fairly simple and structured target, the solver does not have to be very ”creative”.

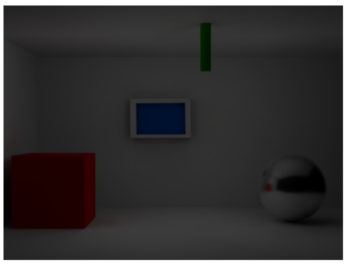


*Figure 51: Scene*

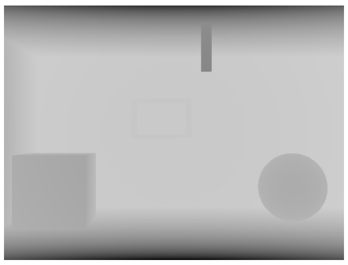


*Figure 52: Z-map*

*Figure 53: Constrains*

*Figure 54: Test case 2*

### Simulation parameters

| Parameter | Value |
|---|---|
| Number of fixtures | 2 |
| Number of iterations | 34 |
| Population size | 100 |

*Table 5: The simulation parameters used in test case 2*

**Target**

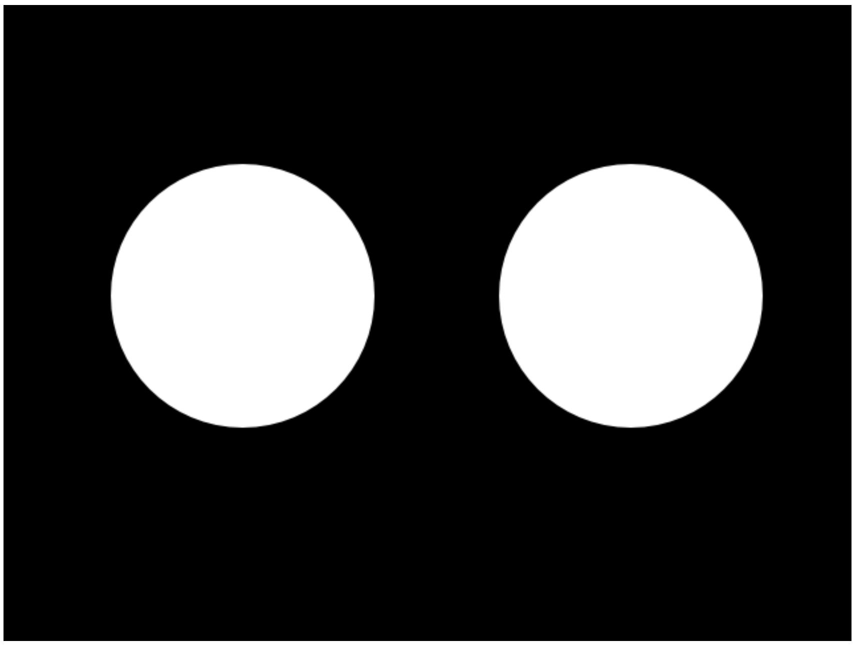


*Figure 55: The target of test case 2*

**Image result**

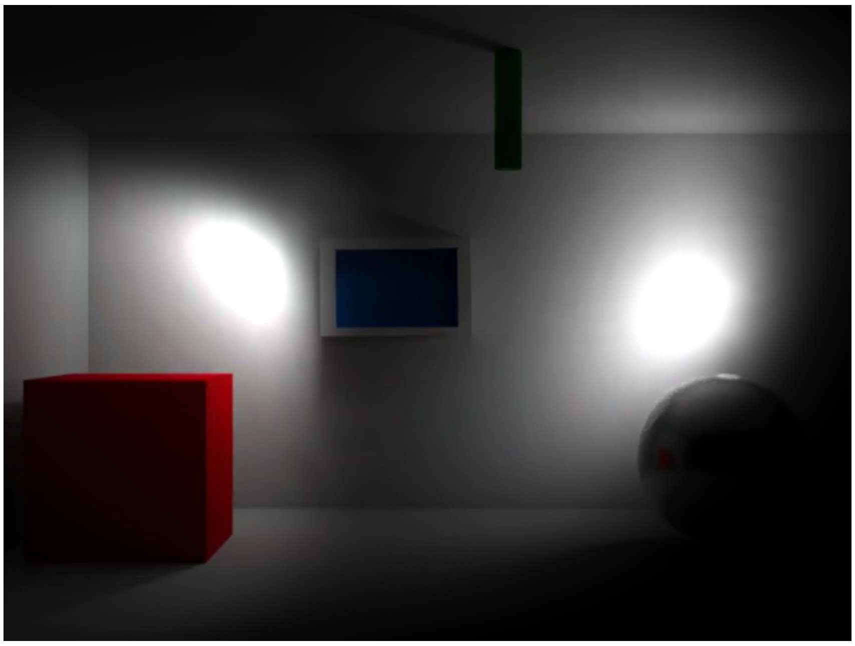


*Figure 56: The calculated result of test case 2*

**Fixture placement**

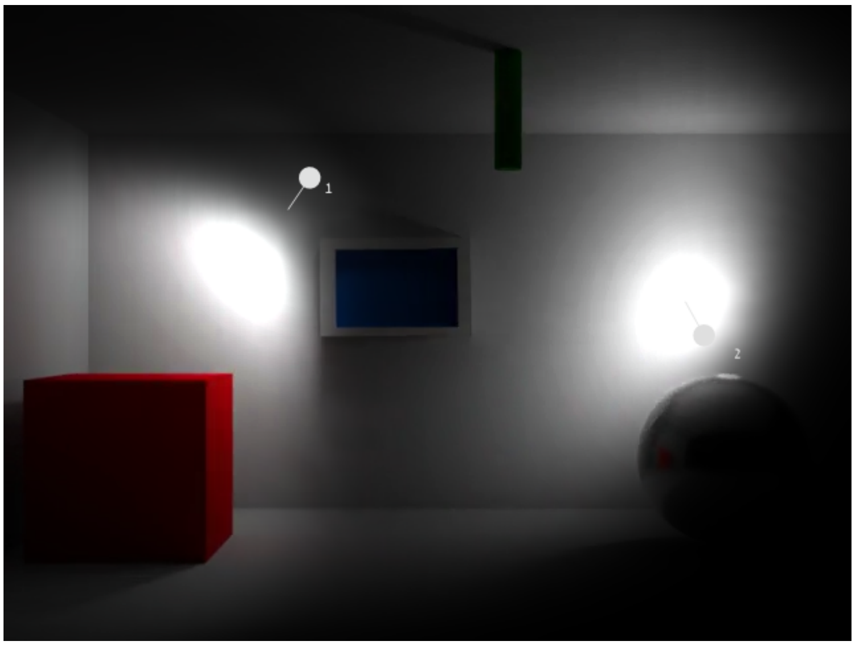


*Figure 57: The placement of the fixtures by the algorithm for test case 2*

**Parameters result**

| ID | $x$ | $y$ | $z$ | $\phi$ | $\theta$ | R | G | B | $L$ | Profile |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 239 | 143 | 11 | 29 | 140 | 2.4285 | 2.4527 | 2.4554 | 887 | 1 |
| 2 | 529 | 250 | 18 | 40 | 65 | 1.4934 | 1.498 | 1.5012 | 1000 | 3 |

*Table 6: The fixture parameters of the found solution for test case 2*

**Discussion**
In this test case, the solver was free to place the fixtures at any location. To create two equal white cicles on the back wall, intuitively the solver would place two equal fixtures shining perpendicular to the wall. However, two slightly different fixtures which were both placed at an angle. This suboptimal solution is probably due to the complex stochastic nature of the solver and the position of the fixtures and the optimal angle. It is expected that, when running the solver for more iterations, it will approach to the optimal solution as written above. As shown in Table 6, the fixtures are not pointing in the pure $z$-direction, as would be expected.

### 11.3.3 Test case 3

Case 3 is a real world test with a recorded probe scan. Here we want to test the capabilities of the solver in real world situations.

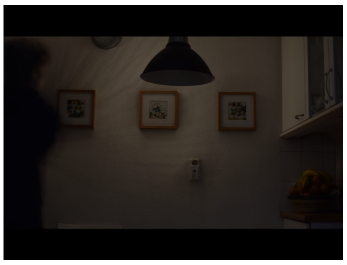


*Figure 58: Scene*

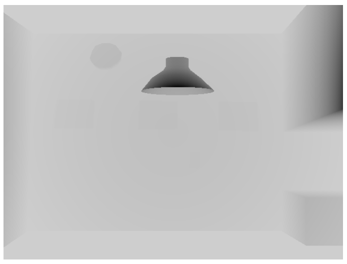


*Figure 59: Z-map*

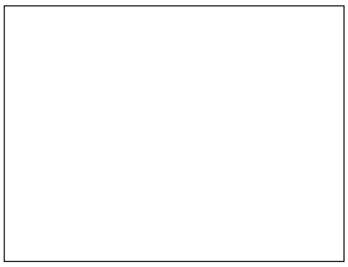


*Figure 60: Constrains*

*Figure 61: Test case 3*

**Simulation parameters**

| Parameter | Value |
|---|---|
| Number of fixtures | 3 |
| Number of iterations | 45 |
| Population size | 100 |

*Table 7: The simulation parameters used in test case 3*

**Target**

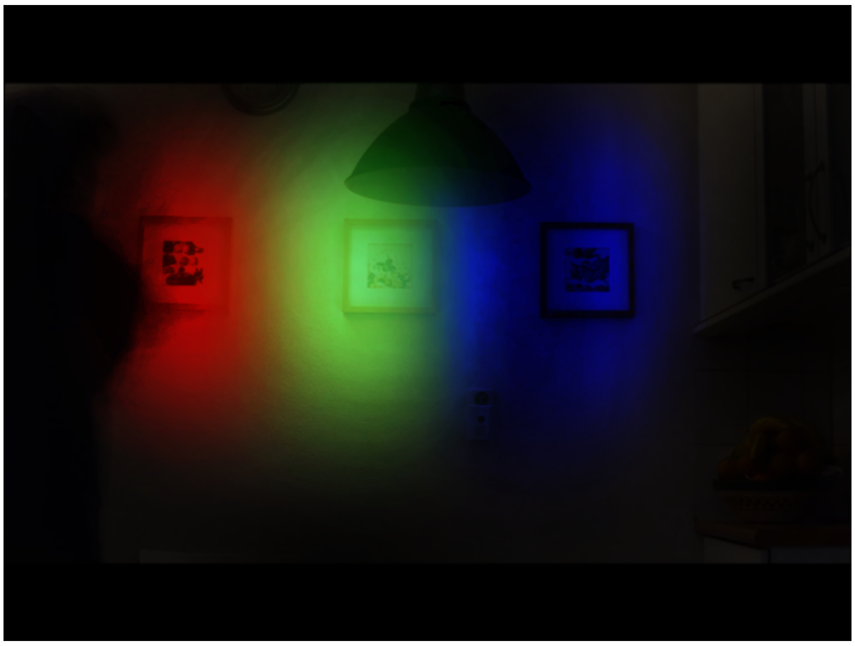


*Figure 62: The target of test case 3*

### Image result

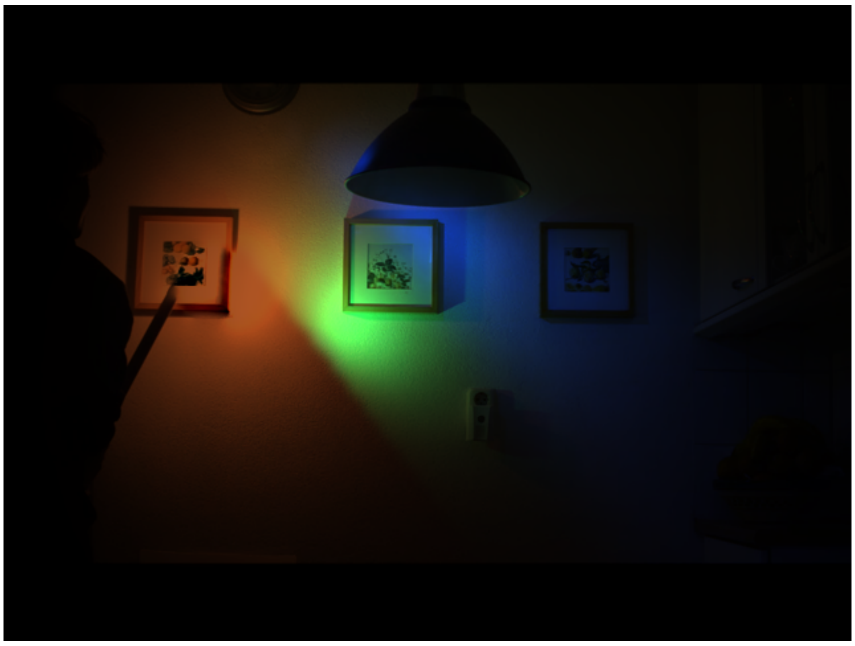


*Figure 63: The calculated result of test case 3*

### Fixture placement

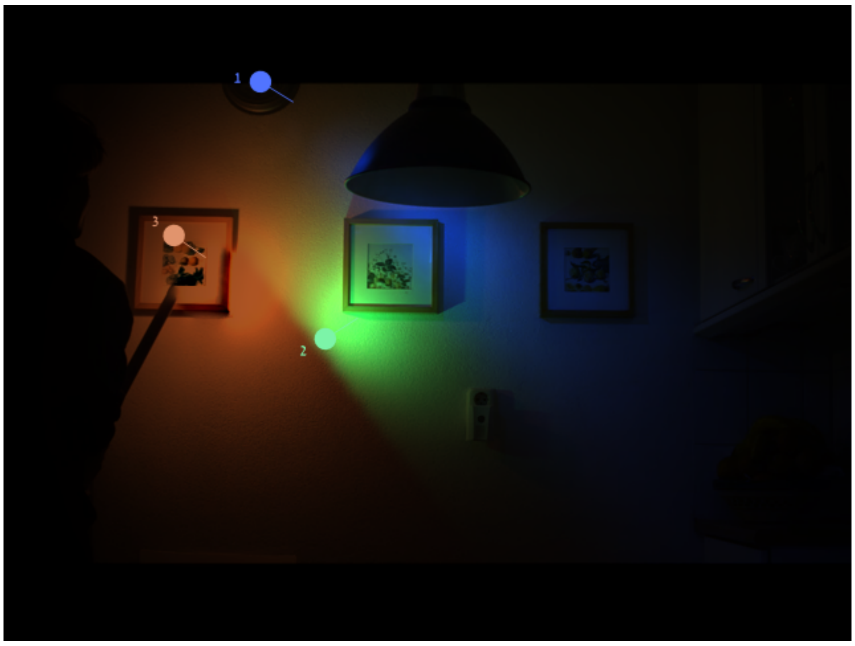


*Figure 64: The placement of the fixtures by the algorithm for test case 3*

**Parameters result**

| ID | $x$ | $y$ | $z$ | $\phi$ | $\theta$ | R | G | B | $L$ | Profile |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 204 | 80 | 9 | 180 | 137 | 0.00053407 | 0.38661 | 2.0059 | 1000 | 7 |
| 2 | 234 | 254 | 12 | 180 | 39 | 0.14257 | 1.2235 | 0.52832 | 959 | 2 |
| 3 | 121 | 188 | 39 | 149 | 110 | 0.58944 | 0.28583 | 0.11978 | 955 | 2 |

*Table 8: The fixture parameters of the found solution for test case 3*

**Discussion**
This test incorporates the use of real world footage. E.i. a real world location that is scanned with a light probe and recorded with a camera. Here, the solver was also free to place the fixtures at any location. The target showed three very distinct color bands, which the solver quite accurately reproduced. The way this is realised seems to be suboptimal. However, given the limited set of provided fixtures to choose from and the fact that no constraints were put, this result is acceptable since it clearly incorporates the three coloured zones at the correct locations while keeping the other areas dark. Shown in Table 8, three differently colored fixtures of which the blue fixture (fixture 1) has a different profile than the other two. Fixture 1 uses a spot light distribution while fixture 2 and 3 use a more wide light distribution. This is because fixture 1 tries to mimic the blue projection as can be seen in the target image, from afar. For this, it cannot interfere with the green fixture 2. This would have happened if fixture 1 had a wide light distribution.

### 11.3.4 Test case 4

Case 4 is a quite complex situation in which the solver has to be inventive in the sense that here the ceiling and the floor has to be lit. We want to test if the solver is capable of placing the fixtures in such a manner, that this is achieved.

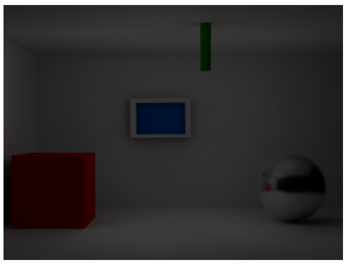


*Figure 65: Scene*

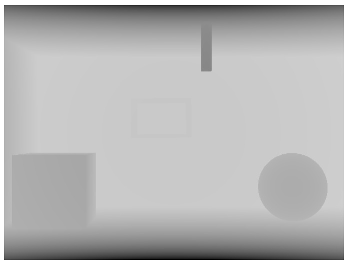


*Figure 66: Z-map*

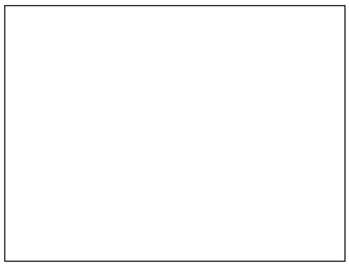


*Figure 67: Constrains*

*Figure 68: Test case 4*

**Simulation parameters**

| Parameter | Value |
|---|---|
| Number of fixtures | 3 |
| Number of iterations | 45 |
| Population size | 100 |

*Table 9: The simulation parameters used in test case 4*

**Target**

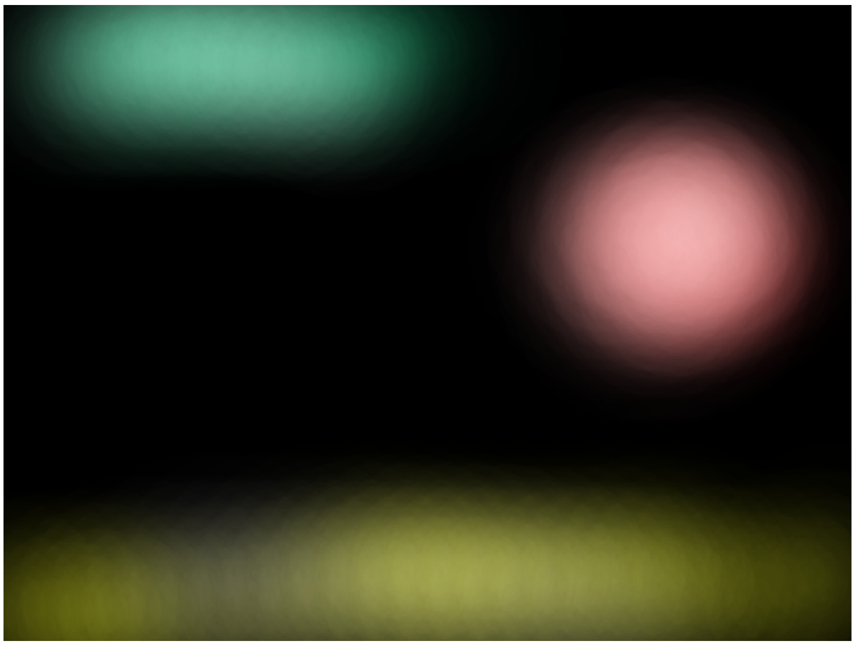


*Figure 69: The target of test case 4*

**Image result**

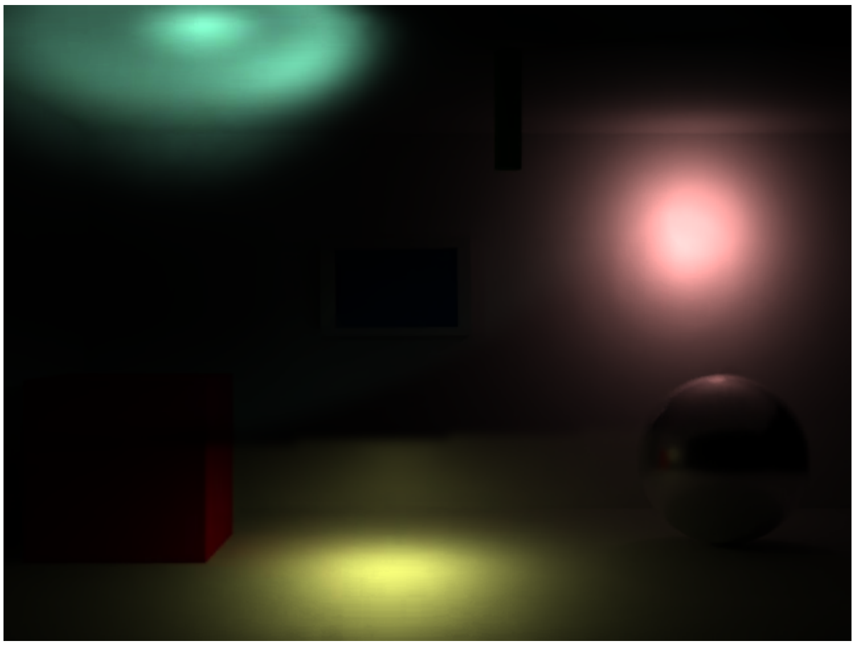


*Figure 70: The calculated result of test case 4*

**Fixture placement**

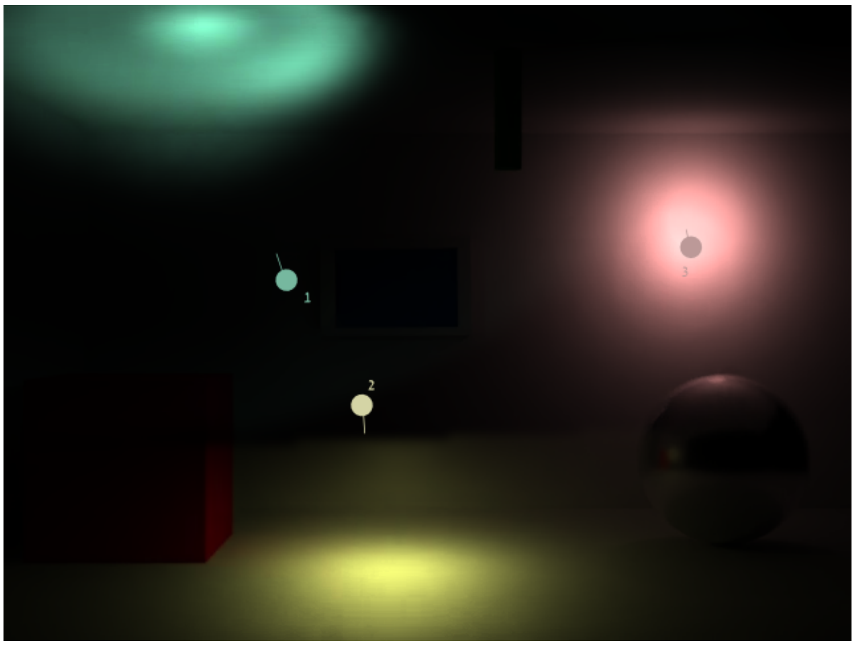


*Figure 71: The placement of the fixtures by the algorithm for test case 4*

**Parameters result**

| ID | $x$ | $y$ | $z$ | $\phi$ | $\theta$ | R | G | B | $L$ | Profile |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 223 | 215 | 25 | 68 | 1 | 1.4173 | 2.7249 | 2.2096 | 1000 | 4 |
| 2 | 286 | 328 | 12 | 92 | 180 | 1.853 | 1.9059 | 0.9131 | 1000 | 2 |
| 3 | 521 | 168 | 9 | 83 | 106 | 1.2974 | 0.84587 | 0.83656 | 1000 | 2 |

*Table 10: The fixture parameters of the found solution for test case 4*

**Discussion**
Again, no constraints were put. The target consists of three distinct color projections of which one is a circular projection on the back wall, another on the ceiling and one on the floor. The solver does a great job at placing the fixtures at the right positions and angles. As shown in Table 10, especially fixture 3 is facing the wall almost perpendicularly, creating a round projection. A remark could be made for fixture 1, which should ideally be placed slightly more to the left and pointing straight up. However, the choice to place this fixture at a small angle and more towards the right is probably due to the red cube obstructing this position.

### 11.3.5 Test case 5

This case incorporates a complex and realistic lighting design in which strong spatial constraints were imposed. Here we push the system to the limit and find what it is capable of. Also, for this case, two test runs were performed with a different population size to monitor the differences in the final result.

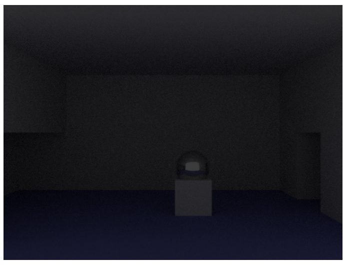


*Figure 72: Scene*

*Figure 73: Z-map*

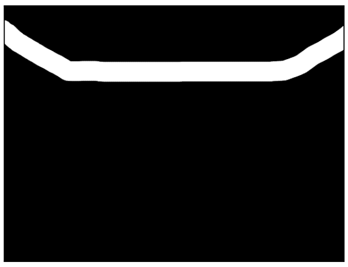


*Figure 74: Constrains*

*Figure 75: Test case 5*

**Simulation parameters**

| Parameter | Value |
|---|---|
| Number of fixtures | 5 |
| Number of iterations | 45 |
| Population size | 50 |

*Table 11: The simulation parameters used in test case 5*

**Target**

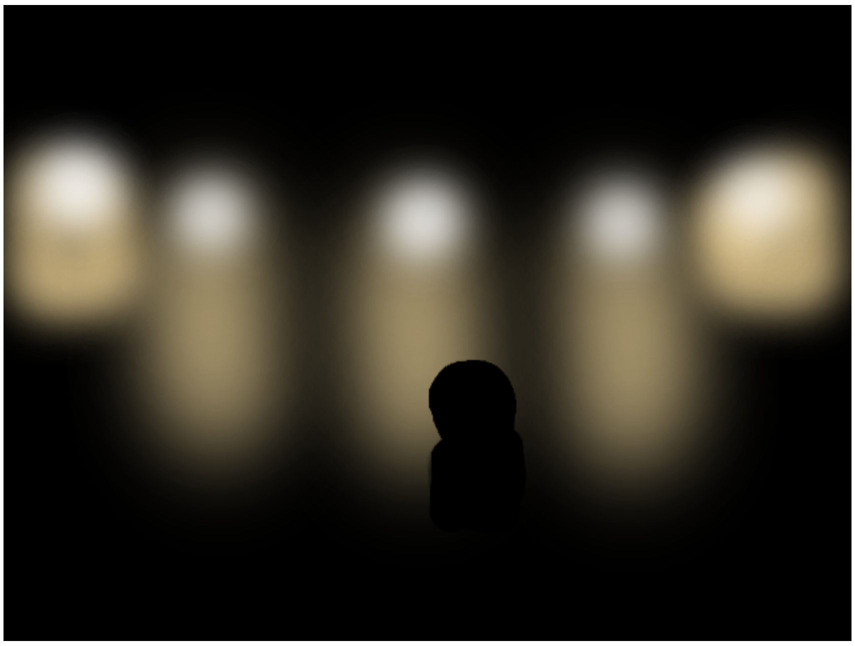


*Figure 76: The target of test case 5*

### Image result

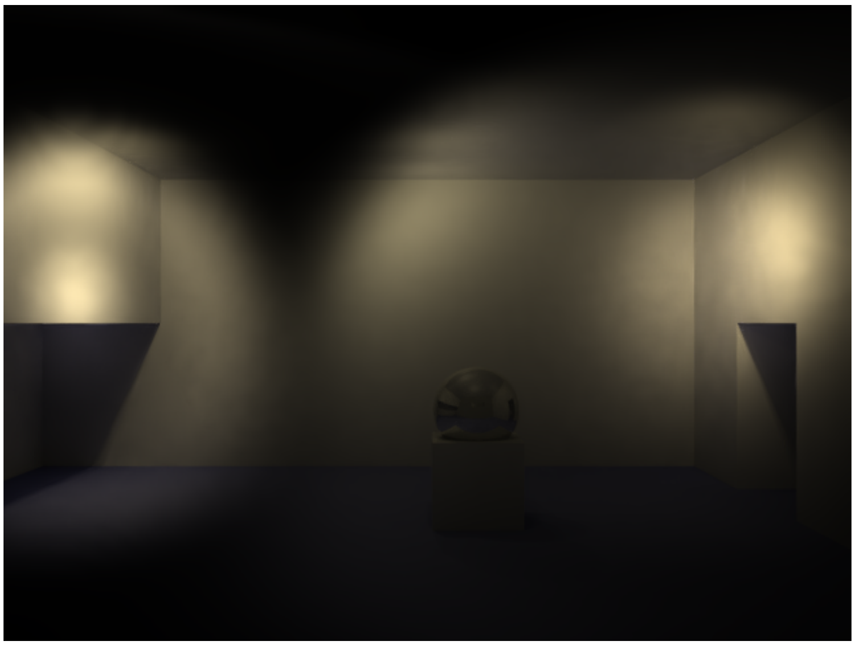


*Figure 77: The calculated result of test case 5*

### Fixture placement

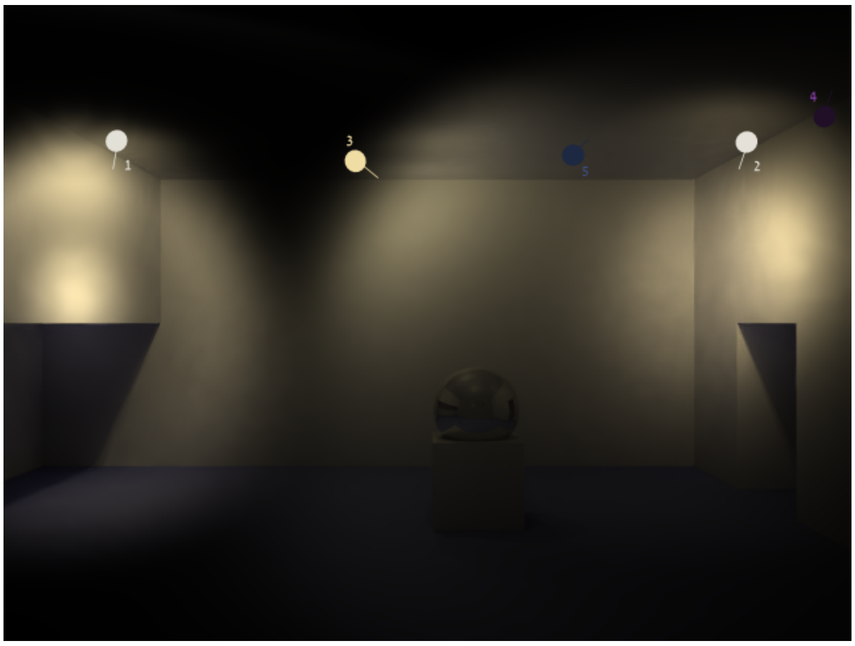


*Figure 78: The placement of the fixtures by the algorithm for test case 5*

**Parameters result**

| ID | $x$ | $y$ | $z$ | $\phi$ | $\theta$ | R | G | B | $L$ | Profile |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 81 | 114 | 22 | 84 | 115 | 1.1656 | 1.0675 | 0.81206 | 970 | 4 |
| 2 | 577 | 106 | 18 | 81 | 115 | 1.1336 | 1.0238 | 0.75794 | 1000 | 5 |
| 3 | 275 | 111 | 10 | 151 | 161 | 0.23909 | 0.22129 | 0.16426 | 1000 | 3 |
| 4 | 620 | 66 | 8 | 116 | 3 | 0.0003231 | 0.00014822 | 0.00039879 | 876 | 3 |
| 5 | 426 | 115 | 6 | 158 | 114 | 0.00010028 | 0.00013512 | 0.00022895 | 713 | 2 |

*Table 12: The fixture parameters of the found solution for test case 5*

**Discussion**
This is a quite complex scene with strict constraints on the position of the fixtures. The solution the algorithm comes up with, resembles the painted lighting design but it would not be a real world solution. The algorithm now has found a three fixture solution, effectively, since fixtures 4 and 5 both have a light output of almost zero (Table 12). This is the optimization the solver has made with the current parameters. Since the solver is based on stochastic principles, running the solver with a population size of, for instance, 200 instead of 50, would increase the chance of finding a more realistic and more optimal solution. To validate this hypothesis, the solver is configured to work with a population size of 200. Since the time consumption is higher when working with bigger populations, the choice has been made to work with a size of 200 and not 500 for example. After a number of iterations, the results are observed and compared against the version with a population size of 50. The results are shown below.

**Testing with a higher population size**
Test 5 was repeated with a higher population size to determine the differences this makes to the resulting output image. The scene, $Z$-map and constrains remain the same as shown in Figure 75 as well as the target image in Figure 76. The result of this test with a higher population is shown in Figure 79 and the fixture parameters are shown in Table 14.

**Simulation parameters**

| Parameter | Value |
|---|---|
| Number of fixtures | 5 |
| Number of iterations | 29 |
| Population size | 200 |

*Table 13: The simulation parameters used in test case 5, with a higher population size.*

## Image result

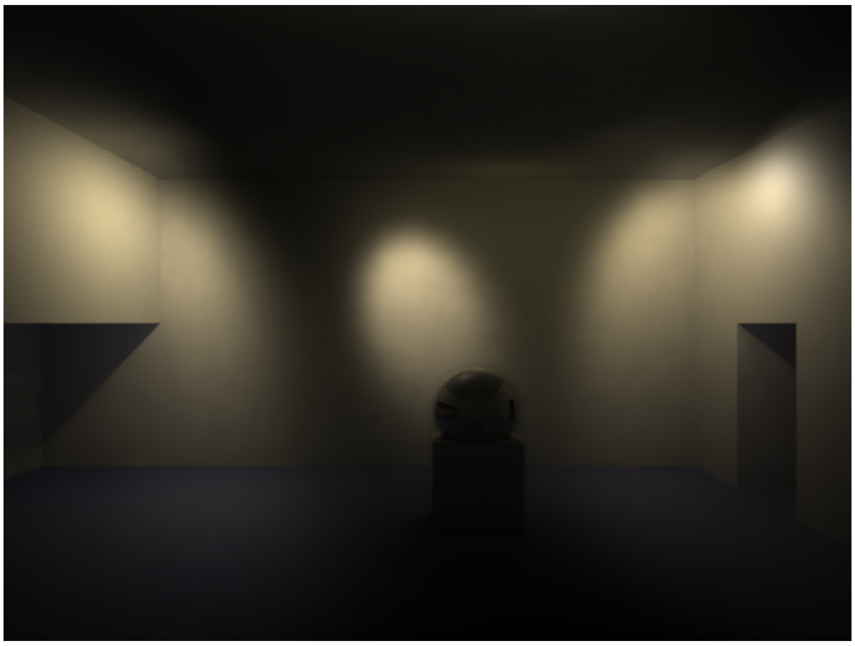


*Figure 79: The calculated result of test case 5, with a higher population size.*

## Fixture placement

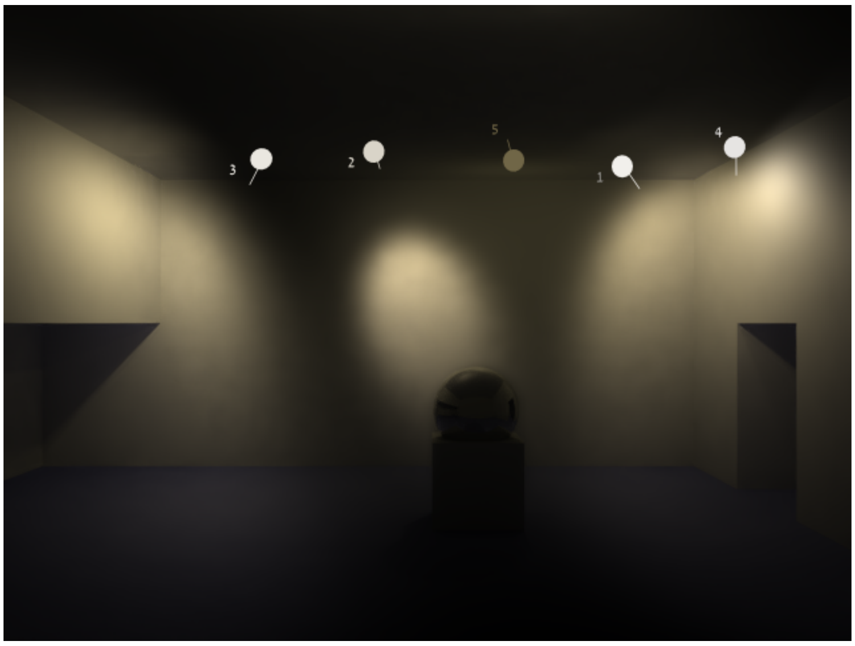


*Figure 80: The placement of the fixtures by the algorithm for test case 5, with a higher population size.*

### Parameters result

| ID | $x$ | $y$ | $z$ | $\phi$ | $\theta$ | R | G | B | $L$ | Profile |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 462 | 118 | 8 | 137 | 165 | 0.97914 | 0.88427 | 0.66007 | 927 | 1 |
| 2 | 271 | 112 | 9 | 111 | 164 | 0.95383 | 0.86292 | 0.65731 | 1000 | 7 |
| 3 | 189 | 121 | 8 | 1 | 158 | 0.93839 | 0.85725 | 0.64387 | 986 | 3 |
| 4 | 558 | 113 | 4 | 180 | 180 | 0.4496 | 0.42331 | 0.37976 | 1000 | 2 |
| 5 | 379 | 138 | 27 | 75 | 73 | 0.43104 | 0.39949 | 0.27308 | 1000 | 3 |

*Table 14: The fixture parameters of the found solution for test case 5, with a higher population size.*

### Discussion

Here we immediately see the impact of the bigger population size. Not only is the result more realistic, it also is objectively better (a higher fitness score/ a lower $MSE$, Figure 81) and it took less iterations for this fitness to be achieved. Also, now all five fixtures are in use (see Table 14). Almost all of them pointing in a logic direction except for fixture 5 which is pointing upward. Doing this, it is illuminating a small portion on the ceiling and back wall, slightly contributing to a higher fitness value. Despite the "better looking" result that results from the second test, the result in Figure 77 also presents a possible solution. By having smaller candidates to choose from (a smaller population size), the algorithm naturally uses the least number of fixtures because the other fixtures do not get the time "to evolve and develop" properly. However, for practical uses this can be beneficial because using less fixtures generally mean smaller costs.

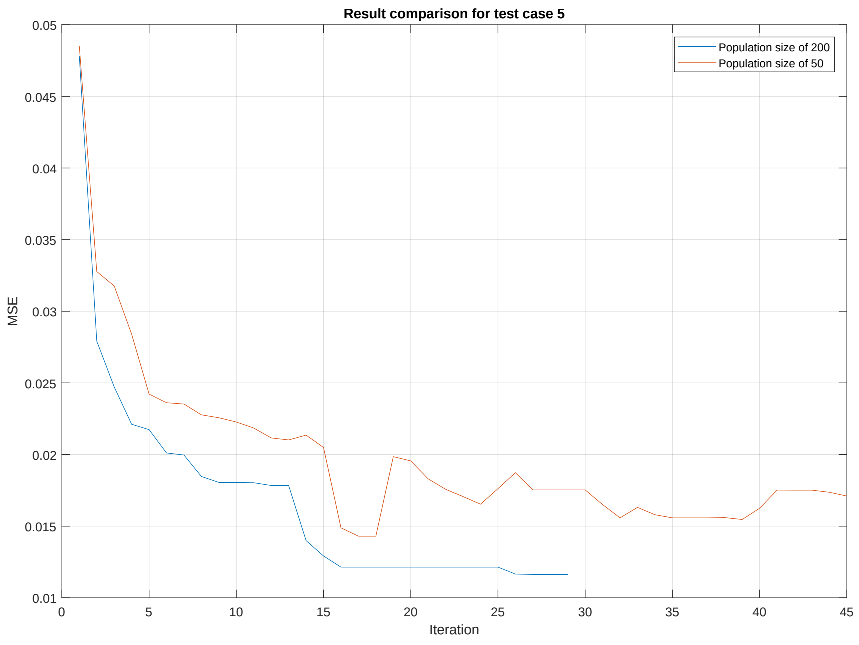
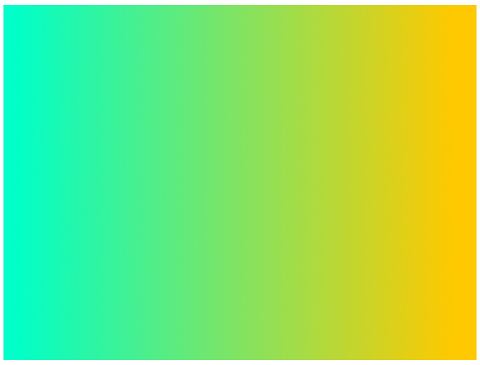
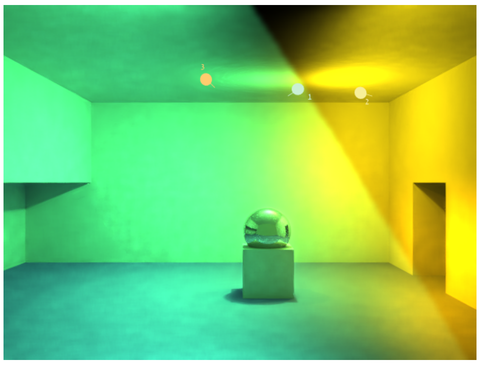


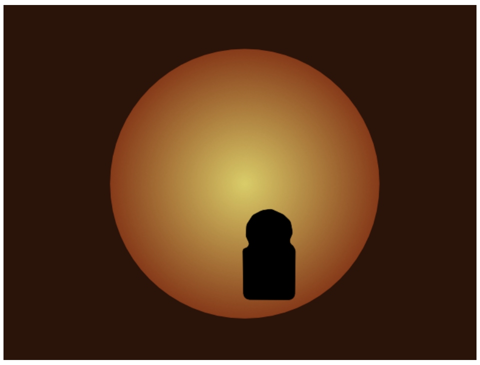
*Figure 81: The solver progression for two different population sizes. The fitness is defined as $\mathcal{F} = \frac{1}{MSE+1}$*

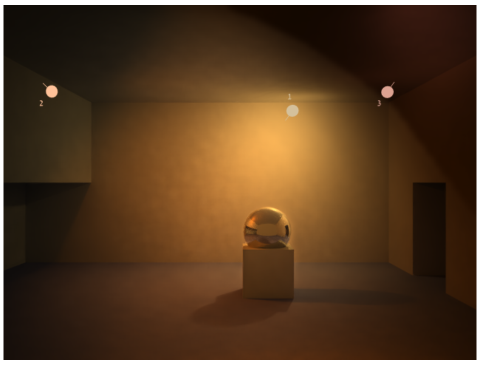
### 11.3.6 Other examples

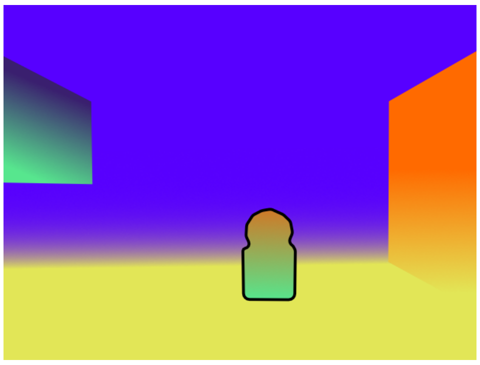
In order to explore creativity and the possibilities and capabilities of the system, some unrealistic example tests have been carried out in addition to the structured tests that were performed in the previous section. These are shown in figure 82.

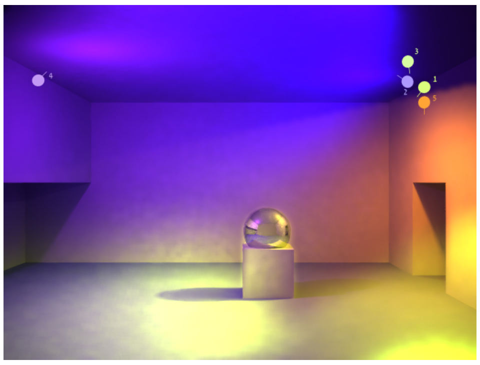

*(a) Example 1. Target image.*

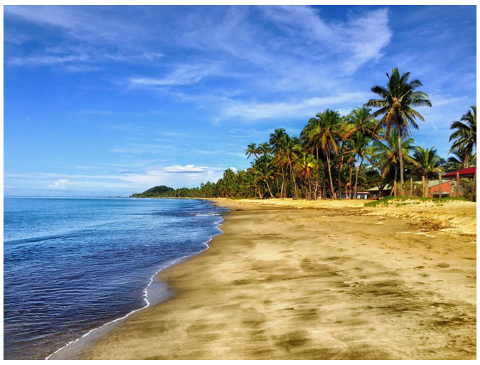

*(b) Example 1. Result. (3, 17, 200)*


*(c) Example 2. Target image.*


*(d) Example 2. Result. (3, 20, 200)*


*(e) Example 3. Target image.*


*(f) Example 3. Result. (5, 23, 200)*


*(g) Example 4. Target image.*

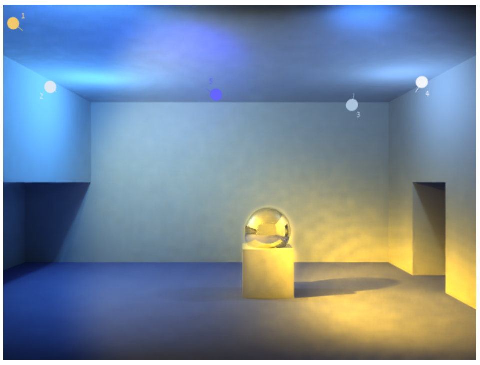

*(h) Example 4. Result. (5, 20, 200)*

*Figure 82: Several "toy" examples. The solver parameters are noted in the caption of each result image in the following format: (number of fixtures, number of iterations, population size). Source of beach photograph: https://pixabay.com/nl/fiji-strand-zand-palmbomen-tropen-293826/*

# 12 Discussion

## 12.1 Overall project

Using image based lighting as a basis for (architectural) lighting design was already proven to be both feasible and useful. Using this technique in combination with fixture models to simulate real world light fixtures in a virtual scene, provided a solid foundation for the lighting design tool developed in this work. The system, dubbed SIMOLIDE, accepts a lighting design painted by a designer and uses this to find (optimal) fixture positions and other parameters to come as close to the initial design as possible. The generated images in the previous section validate the functionality of the SIMOLIDE system. Although in some cases the algorithm comes up with a solution that is a little questionable, it finds a solution that is both feasible and has a close coherence to the painted lighting design. Especially when strong spatial constraints are enforced on the placement of the fixtures, the algorithm proves to be able to find a solution. As stated in the previous chapter, multiple versions of the system have been implemented and it is clear that the latter is the most promising version. The idea of using a linear solver as a kickstarter to aid in the initialization step of the genetic solver, inspired for a better use of the linear part in the mixed system. By using the flexibility and exploring nature of the genetic algorithm combined with the straightforward and calculative power of the linear system, the performance of the SIMOLIDE system was greatly improved.

In general, allowing more iterations improves the result of the system. However, this effect does not carry on indefinitely since most of the improvements happen in the first few iterations. This is because those large changes in the fixture parameters lead to the most profound improvements and the more subtle changes later on only contribute for a small amount to the final result. Using a larger population size also improves the final result (Section 11.3.5, testing with a higher population size) because this increases the chance of having a high-fitness body generated in that population. For most of the results shown in this work, the solver was manually aborted when there were no major improvements to the lighting design in recent iterations and the system was observed to be converted to a valid solution.

## 12.2 Obtaining the $Z$-map

As it was easier and faster to acquire depth information with the two camera setup (there was no immediate need for calibration), the result was not necessarily better. This is probably because of the lower resolution of the cameras. These results very likely improve when two identical high quality cameras are used (this is also confirmed by other scientific sources). However, although better quality cameras will result in better quality $Z$-maps, this will still not entirely cure the problematic behaviour when there is dealt with solid colored surfaces. Since, in the scope of this project, it is very likely that there will be scenes that has some sort of solid colored surface (like walls) present. The stereo vision approach is almost certainly not the way to go.

The backside of depth perception using structured light is its limited range and difficulties it has with non-reflecting surfaces like black cloth. The latter is not a big problem. Both because architectural scenes (which probably is the main application field of this technology) generally do not incorporate a lot of these non-reflecting surfaces and *if* small areas are non-reflecting, this can be overcome by the post process step. The limited range, however, is a serious limitation. The Structure Sensor has a range of $+3.5m$, according to its website [36]. This is generally too low. Especially when designing lighting for a larger room or building.

The method of using 3D-modelled content, although it can be rather time consuming, is free of noise, holes, problems due to non-reflectance or short range. However, care should be taken to make the 3D-model as accurate as possible to achieve accurate results. Though, for general architectural scenes, only the main features (walls, large objects etc.) need to be modelled and hence the time impact is low.

## 12.3 Comparing the initialization algorithms

In most of the times, the initialization step resulted in an increased initial fitness. However, for situation 2, this is not the case; the initialization step failed to find a decent solution. This is probably due to the complexity of the design of situation 2. Apart from all this, there is one thing to note. A high first generation fitness does not guarantee nor suggest a high performance solution. This is because the system can mistakenly rank a certain initial solution high in fitness rating while this solution does not have a solid basis for future generations. Following an example to illustrate this case:

A painted design is composed of three colors. Red, orange and blue (Figure 83 and 84). Now it is perfectly possible that the solution shown in Figure 83 has a higher fitness score than the solution in Figure 84 although it is clear that the latter has a more solid placement of fixtures; it needs less alteration of parameters to come close to the painted lighting design. This, however, is inherited to working with genetic algorithms and a fitness function that is image-based. The final mixed system is less sensitive to this effect since it uses a competition between multiple bodies.

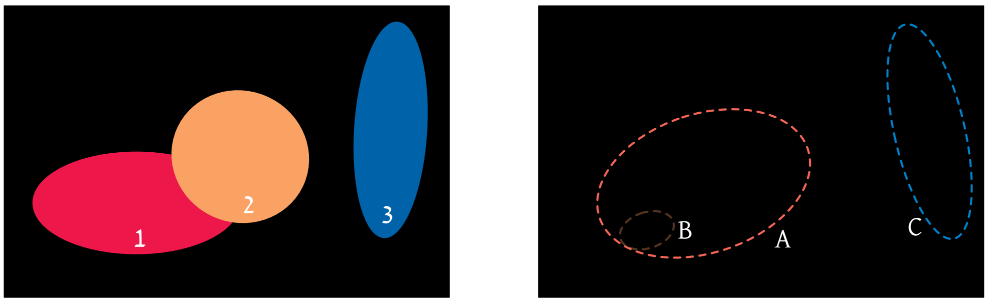


*Figure 83: A poor placement of fixtures result in a good first generation fitness. Fixture B is positioned inside A and has a low intensity. The fitness of this body is high because the size and shape of A corresponds to* 1 *and* 2*. However, B will probably not soon develop into* 2.

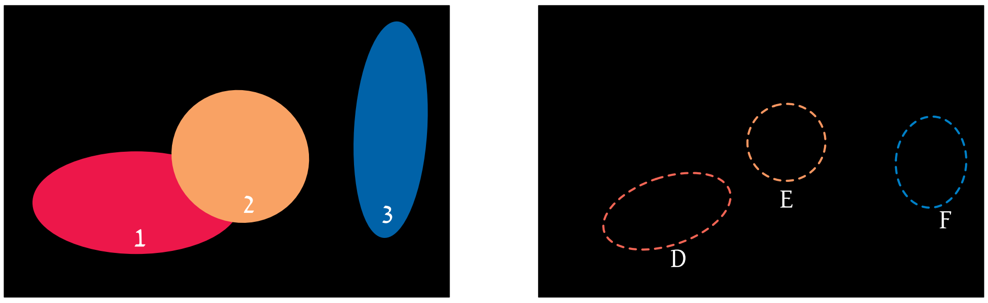


*Figure 84: A solid placement of fixtures result in a suboptimal first generation fitness. All fixtures are too small but in the right location. The fitness of this body is rather low, but the fixtures can quickly develop to the correct size, angle, position and color.*

The importance of this test is that it shows that the use of a suitable initialization algorithm can potentially decrease the exploration time of the genetic algorithm. This can be concluded since it provides the genetic algorithm with a better (fitter) starting point. Nevertheless, while this initialization step is not incorporated in the final mixed solution, it was part of the inspiration that led to the final solution.

# 13 Conclusion

A lot of time was invested in the development of the software foundation, which was built from scratch. First, a scan had to be processed in a certain way that virtual fixtures could be placed in the scene, while simulating its light output. When this was complete, research was put into different aspects of lighting simulation such as indirect light, glare and the effects of a luminaires CRI. Basic implementations of these effects were added to the system. This foundation opened the door for a optimization platform. A framework where an algorithm explores the possibilities of a certain scene, which was later used for the SIMOLIDE system. Three versions of this system were implemented with an increasing performance figure, while solving smaller problems and improving certain algorithms. Although effort has been made to test and validate the final system as thoroughly as possible, not every aspect and variation could be tested. Mostly synthetic scenes were used for validation. This is because it reduces the complexity of the system as a whole and rules out flaws that are the result of imperfections in another section of the system. For instance, the probe detection and removal process is not perfect and can introduce artefacts that are not only visible to the naked eye, but can also negatively interfere with the SIMOLIDE system. This interference could lead to a suboptimal or unwanted solution to a painted lighting design. Also, in real world scans, the person image is in some circumstances still present. Therefore, using a synthetic scene that incorporates all aspects of a real world scan except its flaws, helped validating the performance of the SIMOLIDE system.

# 14 Future work

For now, the system designed and implemented in this master thesis is focussed on (automatic) lighting design with the use of novel simulation techniques based on image based lighting. Of course there could be different usages for (parts) of this system. For example, instead of only finding optimal fixture parameters for a certain (painted) lighting design, an extension to the system could allow it to also maintain the designed lighting. For this, a camera could be used that monitors the light output of the fixture set and compares this with the actual design. When something changes, for example the environmental lighting, the color and intensity of certain fixtures can be tuned in order to maintain the same lighting perception. Another possibility would be to use spherical images (like the ones used in Google streetview) together with a spherical depth map. Now the lighting can be designed for an entire room instead of only one viewport. Of course this introduces a lot of different problems and difficulties and is beyond the scope of this project. On top of that, a GPU version can be developed to increase speed.

## 14.1 The hardware

While the scanning process is now carried out with the use of a DSLR camera and a simple light source on a stick, the 3D map is created afterwards with the help of 3D modelling software. In the future this could be combined in a special device that uses a high quality, high dynamic range camera to record the footage and two smaller cameras to acquire the spatial information. Since the scene is preferably scanned in complete darkness, the two smaller cameras should be able to get the depth information in darkness or at least in low light situations. This can be provided by using IR-sensitive cameras and a infra-red light source. If the three cameras are aligned properly, no human intervention should be necessary and the system should be able to use the footage to get the lighting information together with the spatial information. The camera rig only has to be set up properly, and the scanning process can begin. A concept drawing and render of a possible lighting rig solution is shown in figure 85 and 86 respectively.

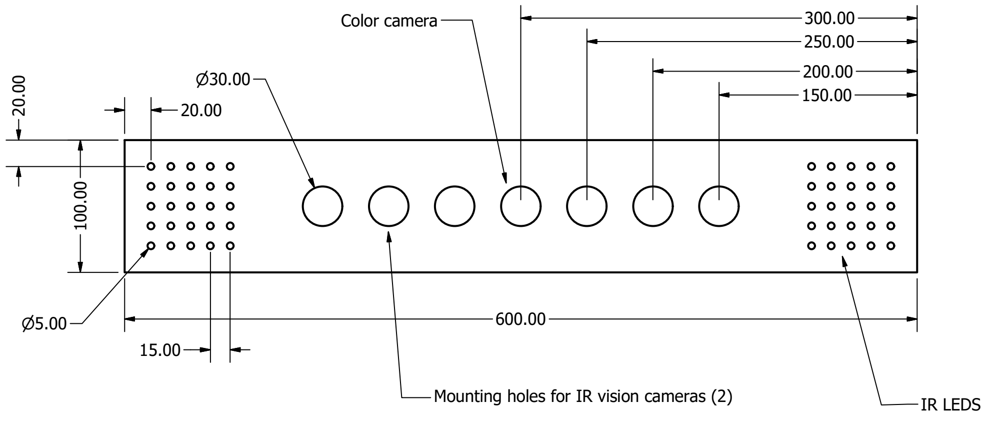


*Figure 85: A concept drawing of the camera rig.*

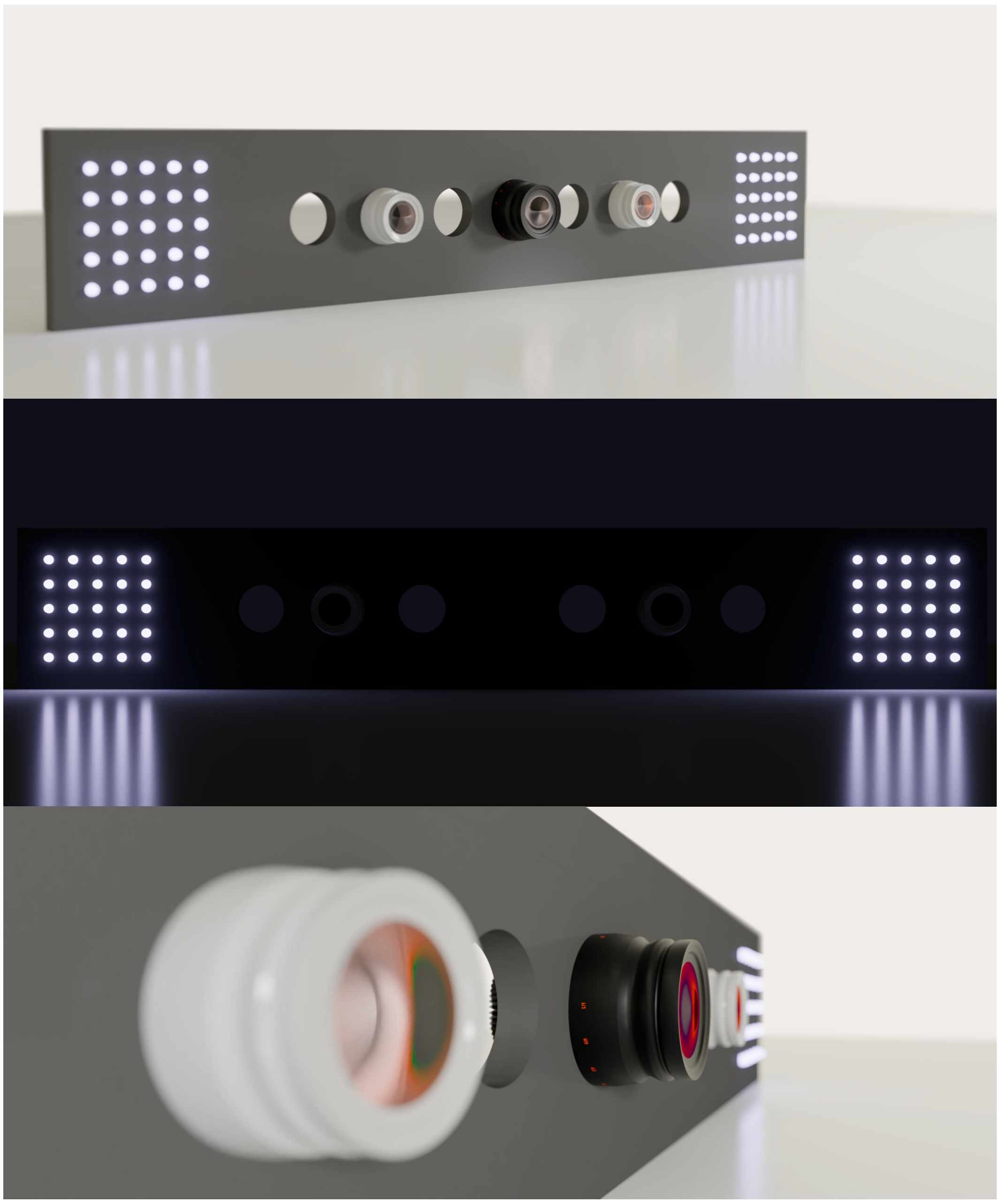


*Figure 86: A concept render of the camera rig.*

The light probe, now being the light source on a stick, could also be improved. The light probe should be a portable, ergonomical, handheld device, operating on replaceable batteries. Since only the omnidirectional light source should be present on scene when scanning, the handle of the device should be made as thin and invisible as possible. The intensity of the light source should be manually adjustable, or autonomously. The camera could observe the scene, and while communicating with the light source, could automatically adjust the light intensity together with camera ISO, aperture size and shutter speed to get the best illumination and the highest recording quality possible without motion blur and with low noise. Apart from being light intensity adjustable, the emitted light should also be of a as high as possible quality. This means low ripple, a natural CCT and a high CRI [37].

# References

[1] H. L. Platt, *The Electric City: Energy and the Growth of the Chicago Area, 1880-1930.* University of Chicago Press, 1991.

[2] D. GmbH, “Better architecture through light and automation,” March 2018. [Online]. Available: https://www.dial.de/en/home/

[3] P. Debevec, “Image-based lighting,” in *ACM SIGGRAPH 2006 Courses.* ACM, 2006, p. 4.

[4] N. Salamon, M. Lancelle, and E. Eisemann, “Computational light painting using a virtual exposure,” in *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 36, Eurographics. John Wiley & Sons, 2017. [Online]. Available: http://graphics.tudelft.nl/Publications-new/2017/SLE17

[5] A. W. Kristensen, T. Akenine-Möller, and H. W. Jensen, “Precomputed local radiance transfer for real-time lighting design,” in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 1208–1215.

[6] F. Anrys and P. Dutré, “Image-based lighting design,” 2004.

[7] S.-t. Noh, S. Hashimoto, D. Yamanaka, Y. Kamiyama, M. Inami, and T. Igarashi, “Lighty: A painting interface for room illumination by robotic light array,” in *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on.* IEEE, 2012, pp. 305–306.

[8] F. Pellacini, F. Battaglia, R. K. Morley, and A. Finkelstein, “Lighting with paint,” *ACM Transactions on Graphics (TOG)*, vol. 26, no. 2, p. 9, 2007.

[9] A. C. Costa, A. A. Sousa, and F. N. Ferreira, “Lighting design: A goal based approach using optimisation,” in *Rendering Techniques’ 99.* Springer, 1999, pp. 317–328.

[10] M. Schwarz and P. Wonka, “Procedural design of exterior lighting for buildings with complex constraints,” vol. 33, no. 5. ACM, 2014, p. 166.

[11] Mathworks, “Find circles using circular hough transform - matlab imfindcircles - mathworks benelux,” March 2018. [Online]. Available: https://nl.mathworks.com/help/images/ref/imfindcircles.html

[12] H. Igehy and L. Pereira, “Image replacement through texture synthesis,” in *Image Processing, 1997. Proceedings., International Conference on*, vol. 3. IEEE, 1997, pp. 186–189.

[13] A. Criminisi, P. Perez, and K. Toyama, “Object removal by exemplar-based inpainting,” in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 2. IEEE, 2003, pp. II–II.

[14] P. Pérez, M. Gangnet, and A. Blake, “Poisson image editing,” *ACM Transactions on graphics (TOG)*, vol. 22, no. 3, pp. 313–318, 2003.

[15] R. Radke, “Dip lecture 22: Image blending,” March 2018. [Online]. Available: https://www.youtube.com/watch?v=UcTJDamstdk

[16] M. Tanaka, “Matlab community profile of masayuki tanaka,” August 2018. [Online]. Available: https://www.mathworks.com/matlabcentral/profile/authors/3502213-masayuki-tanaka

[17] H. Von Helmholtz, *Handbuch der physiologischen Optik.* Voss, 1867, vol. 9.

[18] byHeart Consultants Limited, “Eulumdat file format specification,” March 2018. [Online]. Available: http://www.helios32.com/Eulumdat.htm

[19] I. Occipital, “Structure sensor - 3d scanning, augmented reality, and more for mobile devices,” April 2018. [Online]. Available: https://structure.io/

[20] A. Ortis, F. Rundo, G. Di Giore, and S. Battiato, "Adaptive compression of stereoscopic images," in *International Conference on Image Analysis and Processing.* Springer, 2013, pp. 391–399.

[21] MathWorks, "Uncalibrated stereo image rectification - matlab & simulink - mathworks benelux," April 2018. [Online]. Available: https://nl.mathworks.com/help/vision/examples/uncalibrated-stereo-image-rectification.html

[22] ——, "Disparity map between stereo images - matlab disparity - mathworks benelux," April 2018. [Online]. Available: https://nl.mathworks.com/help/vision/ref/disparity.html?s_tid=gn_loc_drop

[23] L. Zhang, B. Curless, and S. M. Seitz, "Rapid shape acquisition using color structured light and multi-pass dynamic programming," in *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on.* IEEE, 2002, pp. 24–36.

[24] Mathworks, "regionfill," September 2018. [Online]. Available: https://www.mathworks.com/help/images/ref/regionfill.html?s_tid=doc_ta

[25] B. Foundation, "blender.org - home of the blender project - free and open 3d creation software," April 2018. [Online]. Available: https://www.blender.org/

[26] Y. I. Tyukhova, "Discomfort glare from small, high huminance light sources in outdoor nighttime environments," 2015.

[27] H. Joo, S. Kwon, S. Lee, E. Eisemann, and S. Lee, "Efficient ray tracing through aspheric lenses and imperfect bokeh synthesis," in *Computer Graphics Forum*, vol. 35, no. 4. Wiley Online Library, 2016, pp. 99–105.

[28] J.-S. Botero, F.-E. López, and J.-F. Vargas, "Classification of artificial light sources and estimation of color rendering index using rgb sensors, k nearest neighbor and radial basis function." *International Journal on Smart Sensing & Intelligent Systems*, vol. 8, no. 3, 2015.

[29] C. Inc., "Product family datasheet. cree® xlamp® xm-l leds," 2017.

[30] MathWorks, "Create a simple app using guide," April 2018. [Online]. Available: https://nl.mathworks.com/help/matlab/creating_guis/about-the-simple-guide-gui-example.html

[31] A. Y. Yang, Z. Zhou, A. G. Balasubramanian, S. S. Sastry, and Y. Ma, "Fast \ell_1-minimization algorithms for robust face recognition," *IEEE Transactions on Image Processing*, vol. 22, no. 8, pp. 3234–3246, 2013.

[32] M. Schmidt, "Least squares optimization with l1-norm regularization," *CS542B Project Report*, pp. 14–18, 2005.

[33] V. Fonti and E. Belitser, "Feature selection using lasso," *VU Amsterdam Research Paper in Business Analytics*, 2017.

[34] MathWorks, "Lasso or elastic net regularization for linear models - matlab lasso," June 2018. [Online]. Available: https://www.mathworks.com/help/stats/lasso.html

[35] K. e. a. Koh, "Simple matlab solver for l1-regularized least squares problems," May 2018. [Online]. Available: https://web.stanford.du/~boyd/l1_ls/

[36] I. Occipital, "Structure sensor support center — what are the structure sensor's technical specifications?" April 2018. [Online]. Available: https://structure.io/support/what-are-the-structure-sensors-technical-specifications

[37] Yujileds, “Yj-bc-400h-g01 high cri led,” June 2018. [Online]. Available: https://cdn.shopify.com/s/files/1/0344/6401/files/YJWJ006-1.1_YJ-BC-400H-G01.pdf?8015580825686102054

[38] A. S. Glassner, “An introduction to ray tracing,” 1989.

[39] M. Mitchel, “An introduction to genetic algorithms. a bradford book,” 1996.

[40] Z. Xu, H. Zhang, Y. Wang, X. Chang, and Y. Liang, “L 1/2 regularization,” *Science China Information Sciences*, vol. 53, no. 6, pp. 1159–1169, 2010.

# Appendices

## A Background information

The simulation and visualisation of light behaviour is a very important topic in the field of computer graphics. For most applications, the concept is based on ray-tracing, which is briefly discussed below. In this work, we make use of image-based lighting to perform the task of visualizing artificial light. This method is thoroughly discussed in this thesis.

A strong method for finding an optimal solution in a big search space is evolutionary computation. Genetic Algorithms (or GA) use natures own strategy to optimize very complex problems. In this work, GA together with regression analysis is used, therefore a brief description of the basics of GA and regression analysis is written below.

Because of the use of evolutionary computation for finding an optimal design together with the strong comparison between traditional lighting visualisation strategies and the one described in this work, a moment has been taken to introduce the reader in these topics.

### A.1 Ray-tracing

Ray-tracing is a technique for creating a 2D picture out of a three dimensional scene. In this section we will briefly discuss the principle of ray-tracing [38].

At its basis, ray-tracing is determining the color of a pixel in a synthesized two dimensional picture $P$. If we model $P$ as an image plane in a three dimensional world $W$ we can obtain the color of each pixel in $P$ by combining the colors of the light rays (photons) that strike that pixel.

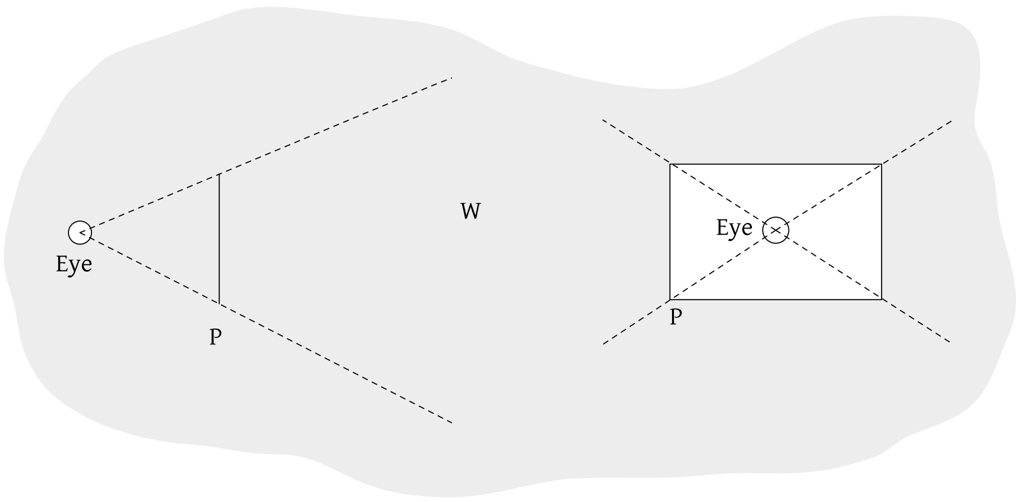


*Figure 87: The image plane $P$ as viewed from the observer (eye) in the three dimensional world $W$.*

There are roughly two ways to determine which photons will strike the pixel and hence contribute to the final color of that pixel:

- Forward ray-tracing
- Backward ray-tracing

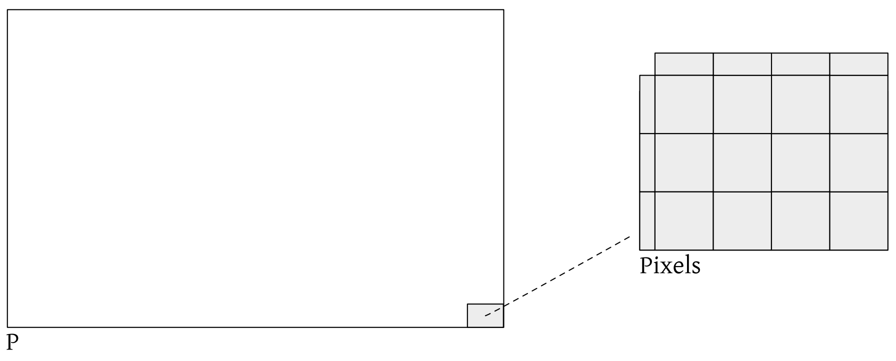


*Figure 88: The image plane $P$ consists of pixels.*

### A.1.1 Forward ray-tracing

Forward ray-tracing is the method that feels most natural as it follows all photons that are emitted from a virtual light source $S$. Some of them reach $P$, but others do not. The photons that reach a pixel in $P$ in one way or the other can be combined to find the color of that pixel.

### A.1.2 Backward ray-tracing

Since a light source emits a lot of photons and only a relatively small part of those photons will eventually reach $P$, following the pixels from $S$ to $P$ will be very inefficient. To overcome this problem, backward ray-tracing is applied. Backward ray-tracing follows the photons from $P$ to $S$ and focusses hereby only on the photons that contribute to the final image. Because of the increased efficiency of backward ray-tracing, it has become almost a synonym for ray-tracing.

It is clear that, when 3D objects become more complex and more virtual light sources are present, the computation time rapidly increases since more photons travel more complex paths.

## A.2 Genetic Algorithms

Finding the best solution amongst other (sub-optimal) solutions is not an easy task. Especially when the amount of possible solutions is very large. To accomplish this goal, there are certain strategies that one can use. One of them is the use of Genetic Algorithms. Genetic algorithms use the same strategy that nature uses to let creatures "adapt" to certain environmental factors and that hence result in creatures that are "perfectly" fitted. I.e. nature has found the best solution. There are different definitions of genetic algorithms but most share the following elements [39]:

- Chromosomes
- Selection based on fitness
- New offspring generation with:
  - Crossover

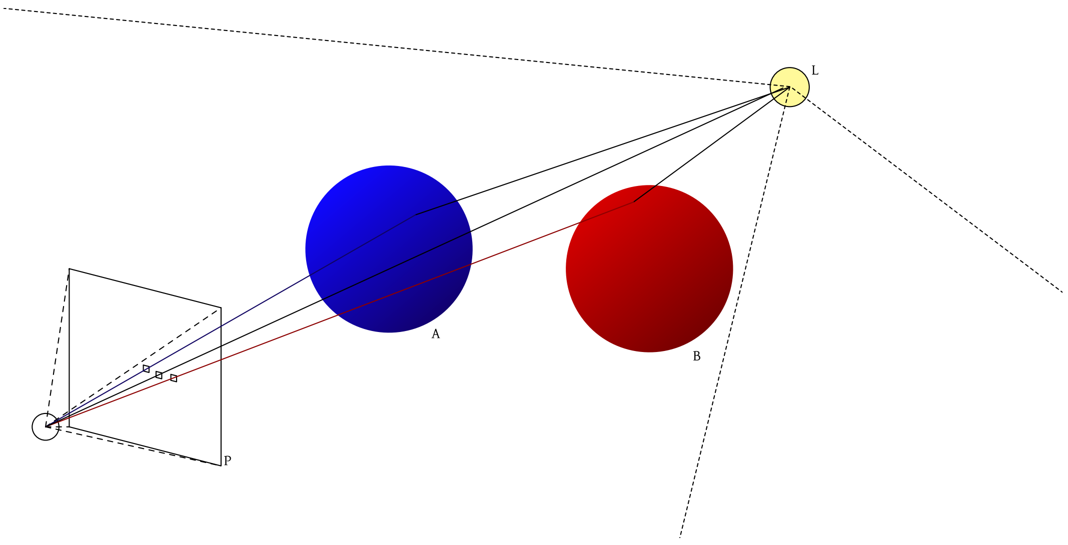


*Figure 89: A lot of photons emitted by light source $L$ do not reach $P$. From the photons that reach $P$, some of them interacted with objects $A$ or $B$.*

  – Mutation

### A.2.1 Chromosomes

Like in nature, chromosomes encode (genetic) information. In the field of GA this is in the form of bit strings, as can be seen in figure 90.

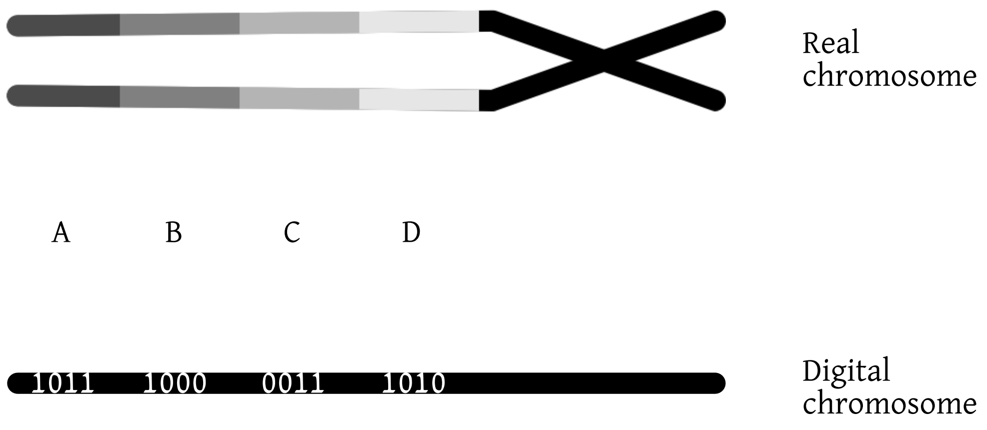


*Figure 90: A real chromosome versus a digital chromosome. The digital chromosome encodes the genes $A, B, C, D$ in the form of bit strings*

These chromosomes encode parameters that are used in a certain problem that needs to be solved. Initially, when the GA is active for the first time, it generates a number of random chromosomes. This is the initial population.

### A.2.2 Selection based on fitness

Each chromosome in the initial population tries to solve the problem with its parameters and a fitness function gives a score (fitness value) to each chromosomes based on how well the problem is solved. From the initial population, some chromosomes were able to solve the problem, while others did not. The chromosomes that were able to solve the problem are compared based on their fitness value.

### A.2.3 New offspring generation

New offspring is generated by ”parent” chromosomes selected based on their fitness value. This offspring generation is generally based on two operators: crossover and mutation.

**Crossover** The crossover operator exchanges the bit strings from the two chromosomes up to a single point called the locus. This means that if two bit strings 11111111 and 00010101 are crossed over after the fourth locus, this will result in the offspring: 11110101 and 00011111.

**Mutation** Mutation is the flipping of a bit in the chromosome at a random location with a certain probability. 00010101 could result in 10010101 when a random bit flip has occurred before the first locus.

Now, the process repeats by letting each new chromosome try to solve the problem and assigning a fitness value.

### A.2.4 Example

To illustrate the principle of genetic algorithms, the following simple example is used: Consider a computer game with a small cannon that can adjust its angle ($\theta$) from 0 to 90 degrees, and firepower ($\Omega$) from 0 to 100 percent to hit a target that is placed a certain distance away from the cannon. In order to find $\theta$ and $\Omega$ that enable the cannon to hit the target in the shortest amount of time, a genetic algorithms is used.

For the cannon example, the chromosome is a concatenation of $\theta$ and $\Omega$. Initially, the algorithm generates $n$ random chromosomes in the form:

$$C_n = [\Omega_b || \theta_b] \quad \Omega_b, \theta_b \in \mathbb{Z}_2^i$$

Hence the chromosomes are binary representations of random values for the firepower and the angle, concatenated.

Now the cannon is set according to the different chromosomes (i.e. the different values for $\Omega$ and $\theta$) after which the cannon takes a shot. Each shot is scored based on a fitness function, which could in this example be like:

$$fitness = \frac{1}{\alpha + |x_{target} - x_{hit}| \cdot TIME}$$

We can see that the fitness score is higher when the cannonball hits the ground closer to the target, or when time until impact is smaller, which are the criteria we want to optimize for. While every chromosome now has a score assigned, the best chromosome(s) can be picked and processed in certain ways to create offspring. Usually with the help of mutation and crossover operators.

Consider the chromosomes $C_8^1 = 11010001_b$ and $C_{45}^1 = 11001001_b$ as the fittest chromosomes, offspring can be generated by mutation (e.g. $C_1^2 = 11110001_b$ (bit three was flipped)) or crossover (e.g. $C_2^2 = 11000001$ (the first half of the chromosome is taken from $C_{45}^1$ and the second half from $C_8^1$)). This way, a new generation (generation 2) is generated with the genetic properties of the first generation.

After this is done, generation 2 is scored with the help of the fitness function and the process repeats. After a certain number of generations, the best possible values for $\Omega$ and $\theta$ are found for which the cannon hits are the most accurate and the fastest.

## A.3 Regression analysis

Relationships among variables are not always clear. Finding these relationships and estimating the mathematical background behind these relationships is performed with the use of regression analysis. A lot of different regression techniques have been developed, like the earliest form of regression called least squares regression. In this thesis, regression is used to find the color and intensity of light emitted by the virtual lighting fixtures (see section 8.5).

### A.3.1 Linear regression

In linear regression, there is an (estimated) linear relationship in the variables that together come to a certain result $y$:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

A method used to make estimations on this type of regression is linear least squares. As can be seen in figure 91, a relationship between the variables $x$ and $y$ has been found. This relationship is estimated by the linear least squares method.

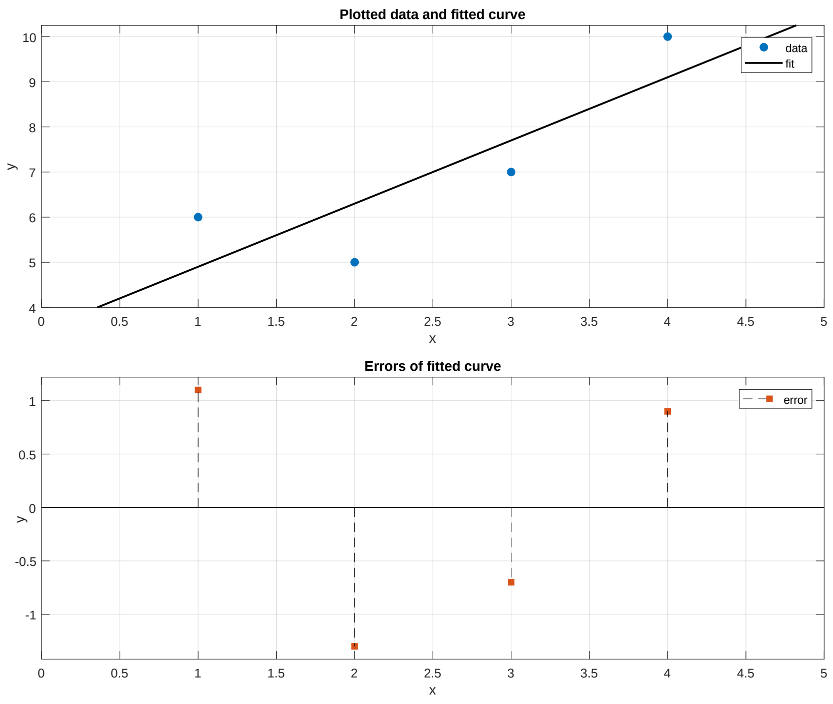


*Figure 91: An example of linear least squares regression. On the top graph, the blue dots are the actual data points wile the solid black line is relation estimated by the regression algorithm. The bottom graph show the error of each data point.*

The errors between the estimated relation and the actual data points (residues) are shown in the bottom graph. The linear least squares method tries to make the sum of the squares of these residues as small as possible:

$$\min \sum (f(x) - \bar{f(x)})^2$$

hence the name.

### A.3.2 Regularization

As estimated relationships become more complex with the use of different regression algorithms, overfitting can occur. In the practical case of finding a relationship between data in the form of pixel matrices, overfitting can result in a solution that is very dense. In other words, when finding a (linear) combination of images that should result in a target image, overfitting results in solutions that incorporate a lot of very small image contributions that obstruct finding a limited set of contributors. To reduce the degree of overfitting, amongst others, regularization is applied. There are different kinds of regularizators (described by $L_n$ in which $n$ is an integer), but the $L_0$, $L_1$ and $L_2$ regularizators are the most common. Regularization discourages a complex model by adding a cost term as complexity increases. This can be seen in the following equation:

$$\min_{\theta} J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^{m} h_\theta(x(i) - y(i))^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

In this equation, the orange part is the regularization term, which adds a penalty corresponding to the sum of the parameters squared. This keeps the model from becoming too complex. $\lambda$ is the regularization parameter. For each type of regularizator, the regularization term is different, leading to differences in behaviour. For instance, $L_0$ regularization leads to the most sparse solution but has trouble finding the most optimal solution while the $L_2$ regularization finds a solution that is not very sparse but is relatively stable[40].