Mathematisch-
Naturwissenschaftliche
Fakultät

Computergrafik

Bachelorarbeit

Building a monitor-based lightstage

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Computergrafik
Hannes Sturm, hannes.sturm@student.uni-tuebingen.de, 2021

Bearbeitungszeitraum: 30.05.21 - 30.09.21

Betreuer/Gutachter: Prof. Dr. Hendrik Lensch, Universität Tübingen

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Hannes Sturm (Matrikelnummer 3923582), September 29, 2021

Abstract

Many current lightstages have the challange of re-lighting objects from sparse sampling, which enforces the usage of neural networks or other complex algorithms to realistically blend together specular highlights from light sources from in between these samples. To increase the sample rate, it might come to mind to use monitors instead of single LED-lights, since they enable nearly continuous illumination due to much higher pixel density. Therefore, I build a monitor-based lightstage in this Bachelor thesis, program the software for it and run some small tests to make the first steps of acquiring high-quality results. Furthermore, I discuss the advantages and disadvantages of monitor-based lightstages based on my experiences in this work.

Acknowledgments

Thanks to Raphael Braun and Andreas Engelhardt for helping me with this work and motivating me, and Prof. Lensch who gave me the opportunity to do this project. I also want to thank my reviewers who took the time to read this work and gave me valuable feedback.

Contents

1 Introduction **11**
1.1 Problem Statement 11

2 Related Work **13**
2.1 Continuous relighting 13
2.2 Continuous illumination 14
2.2.1 Reflective lightstages 14
2.2.2 Monitor-based lightstages 14
2.3 Spherical Harmonics in Computer Graphics 14

3 Construction **17**
3.1 The design process of the lightstage 17
3.1.1 Shape 18
3.1.2 Monitor arrangement 18
3.1.3 Placing the camera 19
3.1.4 The object holder 20
3.1.5 Virtual prototype 20
3.2 Building the actual lightstage 21
3.2.1 Frame 22
3.2.2 Attaching the monitors 22
3.2.3 Object holder 23
3.2.4 Camera holder 24
3.2.5 Cable Management 24
3.2.6 Mobility 24
3.3 Setting up the lightstage 24
3.3.1 The camera 24
3.3.2 The monitors 24

4 Illumination and Scanning **27**
4.1 Spherical Harmonics 27
4.2 Using spherical harmonics for the lightstage 29
4.2.1 Mapping SH on the monitors 29
4.2.2 Displaying SH 29
4.2.3 Normalizing the brightness 30
4.3 Software 30
4.3.1 Server 32

4.3.2 Clients 34

5 Rendering **37**
5.1 Render basics 37
5.1.1 The render equation 37
5.1.2 Lambert's cosine law 37
5.1.3 Lambertian material 38
5.1.4 Phong shading 38
5.2 Acquiring object information 39
5.2.1 Diffuse albedo 39
5.2.2 Specularity 39
5.2.3 Normal map 39
5.3 Rendering and re-illumination 40
5.3.1 Virtual lights 40
5.3.2 Render equation 41
5.4 Software implementation 42
5.4.1 Image processor 42
5.4.2 Renderer 42
5.5 Render results 43
5.5.1 Diffuse map 46
5.5.2 Specular map 46
5.5.3 Noise reduction 46
5.5.4 Normals 47
5.5.5 Dynamic lights 49
5.5.6 Specular and mirror objects 49

6 Conclusion **53**
6.1 Discussion 53
6.2 Future work 55
6.3 Final conclusion 56

1 Introduction

Achieving realistic colors, reflections and shadows in movie scenes is very important and also difficult when compositing multiple shots in post-production and also adding visual effects and other CG-elements. Achieving realistic and consistent lighting conditions for all layers of the final image often gives not as realistic results as the ground-truth ([CK19]). Movie production and visual effects companies like ILM are now starting to use large LED-lightstages which show the digital environment during the filming process to achieve realistic results already during the filming process. [1], [2].
Most of the current lightstages are only capable of illuminating an object from a relatively sparse set of fixed illumination directions. In this work I increase the resolution of previous lightstages further by using monitors as light sources to get continuous lighting patterns in all directions.

1.1 Problem Statement

This thesis aims for the comparison of traditional and monitor-based lightstages and whether there is an advantage of using a monitor-based lightstage. I have the task to design, build and program a monitor-based lightstage. In the end I run small experiments with it by acquiring object information from illumination pattern responses and then relighting it digitally.

[1] https://www.youtube.com/watch?v=gUnxzVOs3rk&ab_channel=ILMVFX [accessed 24 Sep, 2021]
[2] https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-part-one-lessons-from-the-mandalorian/ [accessed 24 Sep, 2021]

2 Related Work

Lightstages are created to acquire all needed informations of an object by scanning it from several directions and capturing its responses to recreate its reflectance field digitally. With this reflectance field it is possible to re-illuminate the object in post-production.

2.1 Continuous relighting

Spherical illumination from a finite number of lights as it is done in the traditional lightstages has always been a challenge when re-lighting the illuminated object. The illumination is done by either using a fixed set of lights distributed evenly around the object ([DWT+02], [WMP+06], [ECJ+06]) or by using a movable lightsource ([DHT+00], [CGS06]). Tunwattanapong et al. [TFG+13] used a rotatable LED light arc for more continuous horizontal illumination. For realistically re-lighting a scene with these kinds of lightstages, a large set of different lighting conditions has to be captured. This makes the scanning process take very long if not using a high speed camera like Wenger et al. [WGT+05] and needs a lot of storage capacity.
There exist multiple solutions to solve the problem of continuous re-lighting from sparse sampling, which are mostly image blending techniques where the re-illuminated image is acquired by interpolating existing responses([FLBS07]). These image blending techniques can be based on simple (linear) blending or they are deep learning-based([SXZ+20], [RDL+15]).
Recently, Meka et al. [MHP+19] developed a technique which only needs two different spherical gradient illumination patterns to relight the human face, enabling real time performance capture at high frame rates with low storage requirements.
Arguably the biggest challenges in relighting objects are the sharp specular highlights and shadow edges since they can not simply be blended into each other which would result in artifacts and weird looking doubled edges or doubled specular points.
Recently Xu et al. [XSHR18] presented a method which is able to relight an object from a sparse set of five sample images.
Furthermore there exist several relighting techniques if only one original image is available ([SKCJ18], [SBT+19], [SHS+17]).

2.2 Continuous illumination

2.2.1 Reflective lightstages

In the past there were approaches of reflective lightstages (Peers et al. [PHD06]) which illuminate the scene by pointing a projector or other lightsources onto a reflective hemisphere which directs the reflected light to its center where the monitored scene is located.

2.2.2 Monitor-based lightstages

There have been a few approaches on monitor-based lightstages as well ([ZWCS99], [PD03]), using monitors to especially determine inter-reflections of translucent materials and environment matting. Environment matting captures reflections and inter-reflections of a scene. Then it can be added to a new scene, realistically reflecting and refracting its light [CZH+00].
In 2021, Sengupta et al. [SCKSS21] introduced a technique using only one monitor which is placed in front of the person together with a simple webcam, playing any video and using a deep network to acquire the information of the viewer's face. While still having very rough results, this enables new possibilities for acquiring human face information without using a traditional lightstage which is expensive and not so easy to access.
Monitors are able to display a continuous lighting pattern at still high framerates.

2.3 Spherical Harmonics in Computer Graphics

Traditional lightstages with only a sparse set of lights were able to illuminate the object by one light at a time while still keeping an appropriate cost of time and storage. As this sample rate increases or even using high-resolution, monitor-based lightstages this approach becomes very impractical, since the costs would be much higher. Also, one single pixel would be way too dim to make a perceptible difference in illumination. Thus, better illumination techniques and patterns had to be discovered such as wavelets and spherical hamonics ([PD03], [PD05], [TFG+13], [SS95]).
Spherical Harmonics are used in computer graphics to create simple and easy-to-compute shaders by approximating the incident lighting. This is often used for videogames and real-time rendering ([SKS02]). Here, environment maps are created with the assumption of distant illumination. These environment maps can then be transformed into and approximated with spherical harmonics to save computing costs and enable faster real-time rendering.
Gosh et al. [GCP+09] presented a method to reliably acquire specular roughness from second order spherical gradients.

Ramamoorthi et al. [RH01] sped up the acquisition of the irradiance environment map.

3 Construction

Before being able to program and acquire object information for renders I first had to build the lightstage. In this chapter I am discussing the construction process of the lightstage and the hardware I used.
After explaining my thoughts during the design process, I am going into the details of the building process. Building the lightstage includes the construction of the frame, attachment of the monitors, placement of the camera and the object holder. Finally I am shortly covering the wire connections to the computer and the setup of the hardware I used.

3.1 The design process of the lightstage

Figure 3.1: Photo of the final lightstage. The door is opened to see the object holder and the camera inside.

3.1.1 Shape

Before I began to build the lightstage, I have been thinking of different shapes for it. The best shape of a lightstage in general, and closest to previous approaches like the Lightstage 6 from [ECJ+06] or the light arc from [TFG+13], is a perfectly shaped sphere. It is the easiest to compute when using spherical coordinates. But since it is very hard to achieve a spherical lightstage using a limited set of flat, rectangular monitors, this was not an option. Of course the easiest shape to form out of six rectangular monitors is a cuboid of the size $w \times h \times h$ where w is the width and h the height of one monitor, assuming every monitor has the same size. But I wanted the lightstage to be even more symmetrical, so I decided to create a lightstage which is a nearly perfecly shaped cube, hoping this would make the later processing and scanning easier, because I would not have to pay attention to every individual monitor's distance to the center and balance it by using different brightness settings for each monitor.

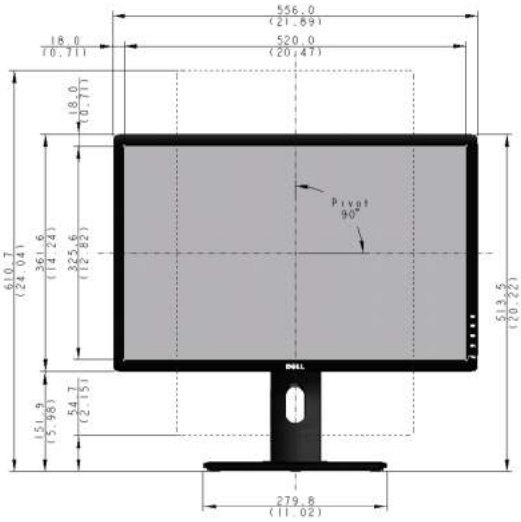
3.1.2 Monitor arrangement

The monitors I had access to were the Dell U2412M-monitors which have a size of $55.6 \times 36.2 \times 6cm$. Each screen is surrounded by a border of the width of $1.8cm$. Thus, the actual screen has a size of $52 \times 32.6cm$ and the cube will have $32.6 \times 32.6cm$ of usable screen surface for each face of the cube. The border is elevated by approximately $5mm$ from the screen surface. This will be useful later when building in the camera. The exact and more detailed information about the monitor's measurements is displayed more clearly in figure 3.2. Their resolution is 1920×1200 pixels and the refresh rate is $60Hz$.

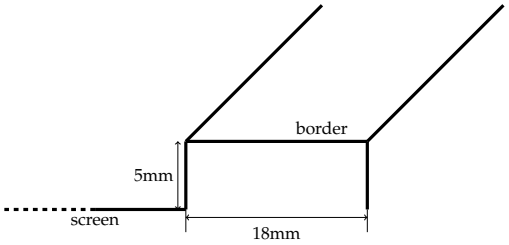
Achieving the cubic shape To achieve a cubic shape with rectangular monitors without creating gaps, I had to place the monitors in a way that the shorter edges create a cube on the inside, while their longer edges are sticking out the lightstage. For example, one monitor is rotated by 90 degrees, so that its longer side is peaking out at the top of the lightstage. This could be used as an additional display to show important information about the lightstage's status.

The door To access the inside of the lightstage to put objects inside to scan, at least one monitor has to be movable. I solved this by turning one monitor on the side into a door, which swings open to a certain degree, enough to easily reach the inside.

Respecting the monitor's borders To get the best results possible, it is desirable to lose as little area as possible of the inner cube to borders, since these areas are "lost pixels", where nothing can be displayed. I managed to hide some of the borders by



(a) [1]



(b)

Figure 3.2: Drawings of a monitor with all relevant measurements.

shifting some monitors in a way such that only one border is located at most of the edges of the cube instead of two edges, as it is displayed in figure 3.3.

3.1.3 Placing the camera

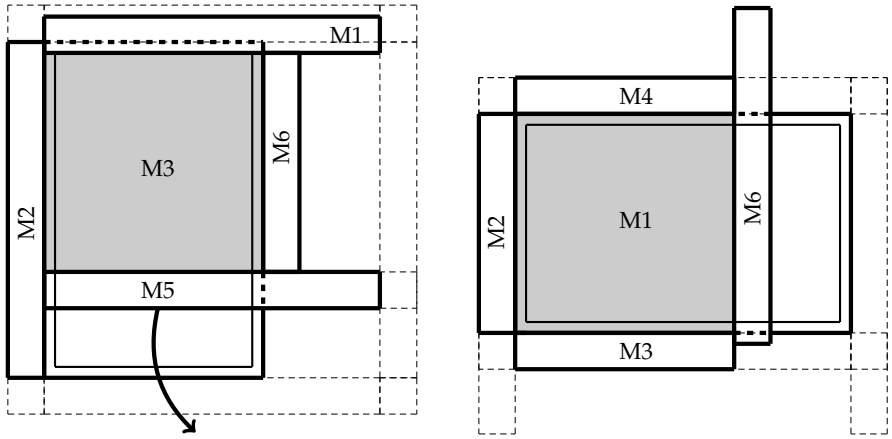
For taking the images in the scanning process a camera has to be placed inside the lightstage, aiming at the center point. I had access to Point Grey's Flea3 FL3-U3-32S2C-CS camera with the following specifications[2]:

Resolution	1920×1080
Max. framerate	60 fps

The camera is chosen due to its small size of $29 \times 29 \times 30 mm$ which makes it easier to

[1] Source: `https://downloads.dell.com/manuals/all-products/esuprt_electronics/esuprt_display/dell-u2412m_user%27s%20guide6_de-de.pdf` [accessed 24 Sep, 2021].

[2] Detailed informations can be found here: https://www.edmundoptics.de/p/flea3-fl3-u3-32s2c-cs-128-color-usb-30-camera/29168/ [accessed 24 Sep, 2021]



(a) Top view. (b) Front view.

Figure 3.3: Design of the lightstage: monitor arrangement. The gray area marks the inner, actual lightstage.

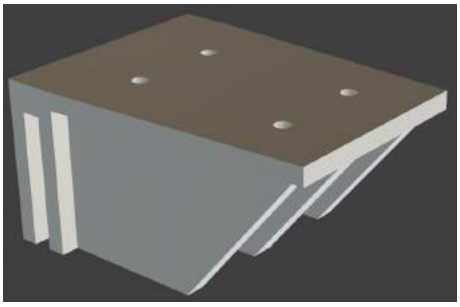
place it inside the lightstage. Another advantage of this camera is that it does not need external cooling which the bigger models need. Nevertheless, it heats up to $70°C$ when it is in use.
The camera is placed in the corner, since the camera covers the least number of pixels in this area from the center point's perspective.
Onto the camera's body I mounted a lense from edmund optics with the variable focal length of $3-10mm$ and an f-number of f/1.6.

3.1.4 The object holder

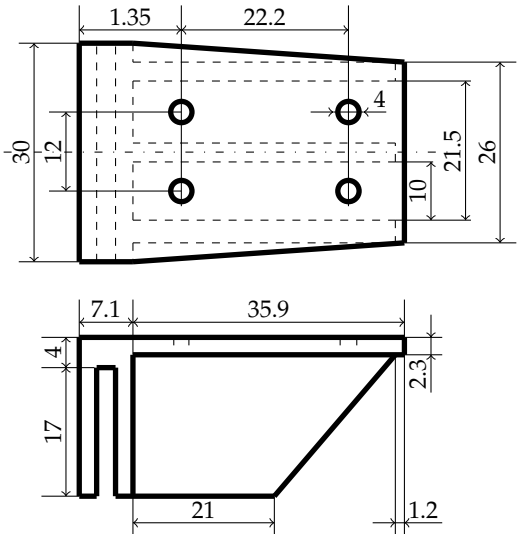
The object holder is designed to keep spherical objects stable, as well as being able to hold other objects. I designed the holder to be round with a lowered center area so that spherical objects can rest stable. Furthermore I planned to create an additional 3D-printed piece which can be put on top of the holder to achieve a flat surface for other than spherical objects. In the end, this was not necessary. It is put and glued onto a long piece of metal. An image of the holder's 3D-model as well as some of the most important measurements and how it is put onto the metal piece can be seen in figure 3.5.

3.1.5 Virtual prototype

Before I started building the lightstage, I first created a digital 3D model of the final lightstage, seen in figure 4.1. Based on this prototype I then decided how to build the lightstage, for example which materials I want to use for different parts.

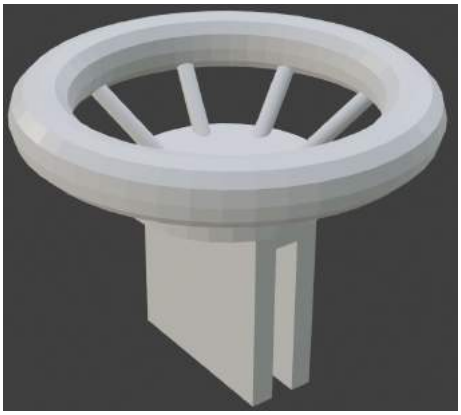


(a) Design of the camera holder.

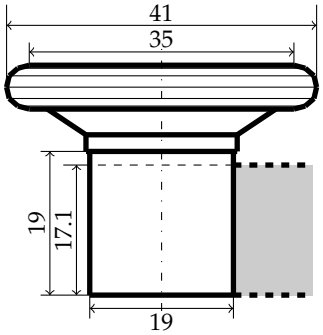


(b) Top-view(top) and side-view(bottom). Lengths in millimeters.

Figure 3.4: The camera holder. The camera is mounted on top of the holder.



(a) Design of the object holder.



(b) Side view with the most important measurements. Lengths in millimeters. The metal piece is marked in gray.

Figure 3.5: Design of the object holder.

3.2 Building the actual lightstage

As seen in figure 3.1 the lightstage is made out of a wooden frame, into which the monitors are built in, facing inwards. I chose wood as the main material, since it is

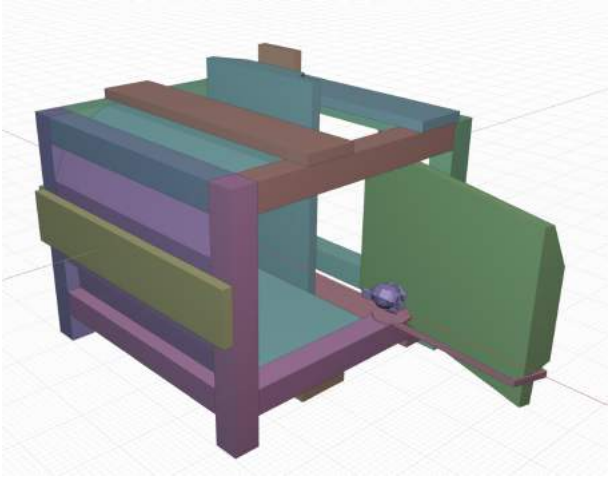


Figure 3.6: Digital prototype of the lightstage.

very cheap, robust and easy to process. On the other hand it is not as precise as metal, since it does change its shape easier. But in the end the advantages were clearly outweighing this argument, since I intended to write a calibration tool as well.
The ligtstage has the outer scale of $67.6 \times 67.6 \times 67.6cm$ (without planks).

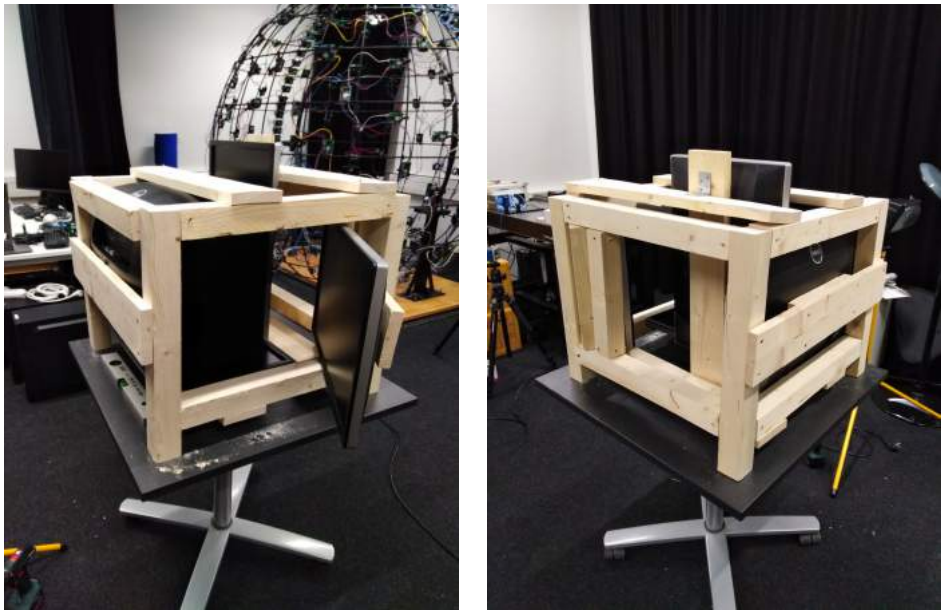
3.2.1 Frame

The first part of the lightstage I built was a robust frame out of wooden beams with the scale of $6 \times 6 \times 55.6cm$ which is exactly the length and thickness of one monitor. Unfortunately, these beams were not exactly $55.6cm$ long but a few millimeters shorter due to sawing inaccuracy, so the whole lightstage had to be widened manually by putting spacers between the beams after it became clear that filing off the inner faces of the beams would take too long.

3.2.2 Attaching the monitors

After the frame was set up, the monitors had to be built in. The monitors are arranged in such a way that their LED surfaces are facing inwards, forming a cube which is as precise as possible. The lightstage was designed in a way which makes it very easy to exchange broken monitors and other parts quickly and meanwhile being as robust as possible.
To achieve this goal, I attached a wooden plank of the length $55.6cm + 2 * 6cm = 67.6cm$ to the back of every monitor, protruding $6cm$ on both sides of the monitor to screw it onto the frame. The big obstacle of doing so was that the planks were very bent and twisted, which bent the entire wooden frame out of its cubic form. This had to be countered by taking apart the frame(but leaving the planks attached to the beams), carefully bending it into its cubic form and screwing the parts together again

without damaging the lightstage or the already attached monitors. This was done until all monitors were in place.
The vertical monitor is inserted from the top into the lightstage, supported by another plank on the top. The plank which is attached to the back of this monitor is longer, to use the protruding part on the bottom as support as well.
For the door I attached a longer plank on the back of the monitor, which again is attached to a shorter beam inside of the wooden frame via a hinge. The door swings open sideways to a degree large enough to reach the inside of the lightstage easily. The monitors are arranged in a way so they form a cube of the inner size of $36.2 \times 36.2 \times 36.2cm$ in one corner of the lightstage. Also the offsets are chosen with attention to lose as few pixels as possible to the borders of the monitors. The offset is achieved by simply screwing one more plank on the back of a monitor to shift it inwards. When buying the planks I chose the planks with a thickness of ~ 2cm which corresponds approximately to the size of the borders to make the shifting as easy as possible. Figure 3.7 shows the lightstage after the monitors have been built in. It also shows how much the defective boards are bending the lightstage.



(a) Front view of the lightstage in front of the traditional lightstage.

(b) Back view.

Figure 3.7: The lightstage during the construction process.

3.2.3 Object holder

The object holder, on which the object can be placed in the middle of the cube, is made with a 3D printer. Its round shape with a lowered center area enables the

placement of spheres. The holder is attached to a metal rod which goes through the space between the door monitor's screen and the monitor next to it. It is bent 45° horizontally on the inside so that it reaches the middle of the lightstage.
Previously it was planned to also 3D-print the metal part, but it had to be as robust as possible for heavier objects like a billard ball, so I decided to use metal instead.

3.2.4 Camera holder

Above this, another metal rod goes through the gap, but directly on the inside of the lightstage it is bent 45° horizontally and 30° vertically downwards. Then the camera is attached onto it using another 3D-printed holder for the camera as it can be seen in figure 3.4.

3.2.5 Cable Management

The cables are bound together and are led around the cube. I used Display Port cables for the monitors, USB 3.1 for the camera. The power cabels are all plugged into one extension cord with a switch to turn the lightstage on.

3.2.6 Mobility

The lightstage is very mobile, since it is built on top of a small, wheeled table. It has also enough space behind the vertical monitor inside the frame to place a small computer there to make it even more independent. It can be moved anywhere and could work as soon as it has electricity.

3.3 Setting up the lightstage

After building the lightstage and connecting it to the PC, it is time to set up the hardware.

3.3.1 The camera

Most of the values remain at their default settings. The white balance of the camera was set to 669(red) and 682(blue).
For the lense I set the focal length to 3.5mm.

3.3.2 The monitors

The brightness settings are set to 100% and the contrast is left on the default value of 75%. Additionally to the six monitors which are directly built into the lightstage,

one additional monitor is connected to the PC. It is used to control and monitor the lightstage.
As visualized in figure 3.9, each GPU is connected with three monitors. Internally, each GPU is assigned to one NVIDIA X-screen. This split into multiple x-screens becomes a problem in the next chapter, because a single application can not access multiple x-screens. At least not without admin rights, which I do not have. Thus I had to write a program for each screen individually.
Later in the project it occured that the monitors had to be re-ordered and inverted internally as seen in figure 3.8. This could also be solved via the lightstage's software, but it seemed to be the easier solution changing the graphics-driver settings.

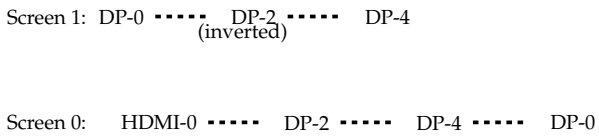


Figure 3.8: The monitors' x-screen settings. The monitors are located next to each other in the shown order. HDMI-0 is connected with the control monitor.

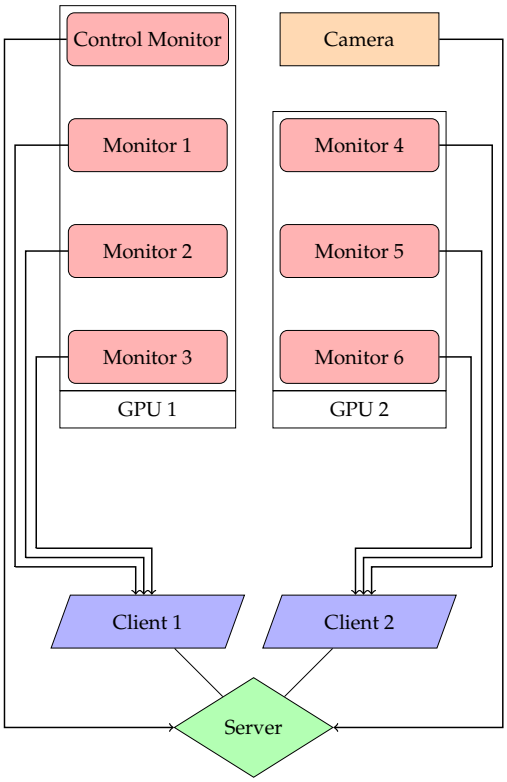


Figure 3.9: Lightstage setup and connection of all graphical units to the PC.

4 Illumination and Scanning

In this section I am presenting a method for scanning the object and getting its illumination responses to acquire important surface information afterwards.
After explaining the theoretical background, I will further explain my approach of implementing this method.

4.1 Spherical Harmonics

In order to get information about the object's surface and material, it has to be illuminated first by a specific pattern which enables further processing of the responses. This can be done in different ways:

Per Pixel Illumination The easiest and brute force approach is to just illuminate the object by every single pixel(or light source in general) individually and taking an image of every response. This has the advantage that complex illumination patterns can be summed up relatively easy in the rendering and re-lighting phase. On the other hand there is a lot of storage needed, since there has to be taken an image for every pixel. Additionally, before a lighting pattern can be formed, there has to be a photo for every pixel available without any intermediate result. Even if the capture process has a speed of 60 images per second - which is unrealistic because the exposure time should be much higher due to the low intensity of one pixel - this would take $\frac{1200 \cdot 1920}{60} = 38400s$, which is more than 10 hours of scanning. So this approach is clearly not the best for an application which has to be as fast as possible.

Spherical Harmonics Another approach, and the one I went for, are lighting patterns from spherical harmonics (SH) as used before by [TFG+13]. But the pixel density of a monitor enables different wavelet patterns as well such [SS95].
SHs are a set of orthogonal functions on the surface of a sphere which form a orthogonal basis. Especially being used in quantum physics, they are also very handy by describing spherical illumination and encoding complex spherical functions as a set of individually scaled and rotated spherical harmonics. They can be seen as Fourier-functions for spherical surfaces. The first spherical harmonic $Y_0^0 = \sqrt{1/4\pi}$ is just the average value of all values on the sphere, just as the first Fourier-function, which is also a constant holding the average value of the signal.

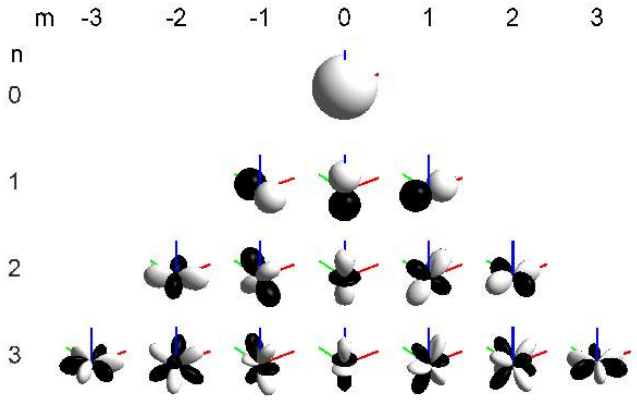


Figure 4.1: Spherical harmonics visualized for degree m and order n. [1]

Just like Fourier-functions, the increasing number of functions increases the level of detail these functions can represent. This gives the opportunity to decide how fast the overall scanning process should be by having a quality - speed tradeoff.
The general formula for a spherical harmonic of order l and degree m is

$$Y_l^m(\theta,\varphi) = \sqrt{\frac{2l+1}{4\pi}\cdot\frac{(l-m)!}{(l+m)!}}\cdot P_l^m(cos(\theta))\cdot cos(e^{im\varphi}) \tag{4.1}$$

where $l \geq 0$, $-l \leq m \leq l$, and θ, φ are the azimuthal and polar angle of a point on the standard sphere's surface. Because only the real part is needed, the equation can be simplified to:

$$Re(Y_l^m(\theta,\varphi)) = \sqrt{\frac{2l+1}{4\pi}\cdot\frac{(l-m)!}{(l+m)!}}\cdot P_l^m(cos(\theta))\cdot cos(m\varphi) \tag{4.2}$$

where $P_l^m(x)$ is the associated Legendre-polynomial:

$$P_l^m(x) = \frac{(-1)^m}{2^l l!}(1-x^2)^{m/2}\frac{d^{l+m}}{dx^{l+m}}(x^2-1)^l. \tag{4.3}$$

Since m is always less than or equal to l, the number of different spherical harmonics dependent on l is $n = 2l+1$.

[1] Source: Producing 3D Audio in Ambisonics - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/First-16-spherical-harmonics-with-order-n-and-degree-m_fig1_281559396 [accessed 24 Sep, 2021]

Like Tunwattanapong et al. [TFG+13], let's assume a reflectance function with a Lambertian diffuse part $D(\omega)$ and specular part $S(\omega)$ with roughness and anisotropy. The sum of these parts can be observed as $f(\omega) = D(\omega) + S(\omega)$.
By defining ω as a vector (θ, φ) yielding $Y_l^m(\omega) = Y_l^m(\theta, \varphi)$ we can now write the response function $f_l^m(\omega)$ as

$$f_l^m(\omega) = \int_\Omega f(\omega) \cdot Y_l^m(\omega) d\omega \tag{4.4}$$

which returns the response of an object illuminated by a specific spherical harmonic $Y_l^m(\omega)$.
Addressing the key observation of Ramamoorthi and Hanrahan [RH01], Tunwattanapong et al. [TFG+13] wrote that "a Lambertian diffuse lobe exhibits the vast majority of its energy in only the 0th, 1st, and 2nd order spherical harmonic bands". Additionally, they concluded that SH coefficients of order $l \geq 3$ only respond to specular reflectance which simplifies the response function to $S(\omega) \approx f(\omega)$ for $l \geq 3$. This means that the coefficients for the diffuse lobe can be computed by first determining the specular coefficients from higher order SH and then computing $D(\omega) = f(\omega) - S(\omega)$ for the lower-order SH responses.

4.2 Using spherical harmonics for the lightstage

4.2.1 Mapping SH on the monitors

Since the spherical harmonics are using the spherical coordinate system, a mapping function is needed which maps the spheres onto the cube formed out of the monitors to be able to display them correctly. To simplify this, it is assumed that the lightstage is shaped as a perfect cube and there are no borders without lights. This could be fixed by calibrating the lightstage.
The mapping function simply transforms the euclidean x, y, z coordinates of every pixel into spherical coordinates:

$$\theta = arctan \frac{\sqrt{x^2 + y^2}}{z} \tag{4.5}$$

$$\varphi = arctan \frac{y}{x} \tag{4.6}$$

4.2.2 Displaying SH

Because the negative values of the SH can not be displayed on a monitor, the positive results are shifted into the interval [0...255] and scaled accordingly as done in [TFG+13]. By inverting the interval, a second pattern is acquired. The difference in the object's responses to both patterns yields the final response f_l^m to the spherical harmonic Y_l^m.

4.2.3 Normalizing the brightness

Because the observed brightness of a pixel changes with its angle and distance, this has to be normalized in order to get the optimal results. For the distance, I assumed a quadratic light falloff, which means the observed brightness is proportional to $\frac{1}{d^2}$ with

$$d = \sqrt{x^2 + y^2 + 1^2}$$

being the distance to the light source. This formula can be calculated by using the Pythagorean theorem and assuming the center of the cube is one unit away from the monitor's center. Now the intensity can be calculated by using

$$I = \frac{1}{d^2} = \frac{1}{x^2 + y^2 + 1}$$

where $-1 \leq x \leq 1$, $-1 \leq y \leq 1$ and assuming that the monitor is one unit away from the center point as seen in Figure 4.3.
To compensate for the angle, a function $f(\delta)$ $(0 \leq f(\delta) \leq 1)$ is applied, resulting in the equation

$$\hat{L}_e = (1 - f(\delta)I) \cdot L_e$$

where $x = y = 0$ at the monitor's center and $\delta = tan^{-1}x^2 + y^2$. The exact value for $f(\delta)$ can be determined experimentally by measuring the light intensity of a monitor from different angles. It is important to do the experiment with the same kind of monitor which is then used inside the lightstage, since the results might depend on the monitor-type and the materials used. I also recommend to investigate the difference in hue and saturation in addition to the brightness difference. In this work, however, $f(\delta)$ is assumed to be a constant factor $0 \leq \alpha \leq 1$ describing only the difference in intensity in the RGB-space, resulting in the equation

$$\hat{L}_e = (1 - \alpha I) \cdot L_e \tag{4.7}$$

To find out which α-value gives the best result, I placed a mirror sphere inside the lightstage and then manually changed the value until the monitors reflected by the mirror sphere appeared to have the same intensity everywhere. The best value for this appeared to be $\alpha = 0.3$. However, this function has to be determined more precisely in the future.

4.3 Software

The lightstage is controlled by a program which is responsible for the scanning and the calibration. It is divided into a server and two client programs.

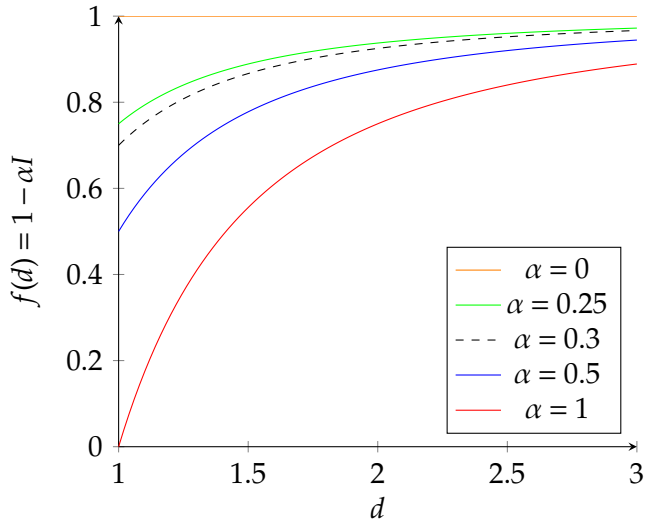


Figure 4.2: Light falloff with different α-values depending on distance d from the center point (note that it is not identical to the center of a monitor but of the entire lightstage).

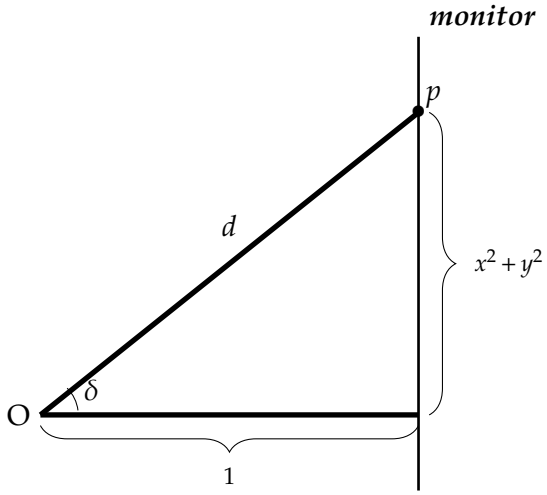


Figure 4.3: Computing the distance d to any point $p = (x, y)$ from the origin O. The origin is assumed to be 1 unit away from the monitor surface, which has a size of 2 x 2 units. Thus, it applies $-45° \leq \delta \leq 45°$.

4.3.1 Server

The server is the heart of the lightstage. It displays information on a GUI, takes commands of the user and controls both client programs. It creates the spherical harmonics and takes the images of the responses as well.

Synchronizing the clients By setting a delay for one or both clients, it is possible to balance out delays, if the hardware connected to one client should be slower than the other client's hardware. It is also useful to slow down the overall scanning process. This was needed when the clients were receiving messages too quickly, yielding in multiple messages combined, unreadable for the client. Of course, this can be solved by marking the end of each message, splitting it and adding all commands to a queue.

Scanner The scanner object has multiple responsibilities. Before starting the scanning process it checks if all necessary spherical harmonics are created for the upcoming scans. If the necessary files do not exist, they are created using the spherical harmonic class.
During the scan process, the scanner object takes the images, once the spherical harmonics are displayed, and stores them as image files.

GUI The server has a simple graphical user interface which enables the user to control the lightstage. The GUI can be seen in Figure 4.4. It displays an overview about the current connection state of both clients as well as the state of the overall lightstage. Using the corresponding buttons, the user can start the calibration and scan process. If any operation fails, this will be displayed on the GUI as well.
On each client's panel there is a setting to set its delay in milliseconds, which will be forwarded to the clients.
When the scan button is being clicked, the lightstage will start the scanning process up to the specified level of detail above it. This value corresponds to the order of the SHs, up to which the object will be scanned with. The progress bar below shows the scanning progress and is updated after every scanned order.

Spherical Harmonics Before calculating the spherical harmonics, a cube of size 1200 x 1200 x 1200 is created virtually and all integer-valued points on its surface are mapped onto a unit sphere. For this I created a *Vector3D* structure with assigned x, y, and z double values as well as a *normalize*-function. This function normalizes the vector and maps all points from the cube's surface, which are all stored as Vector3D-objects, onto the unit sphere.
The *SphericalHarmonics*-class creates the spherical harmonics up to the 3rd order using the formulas from [RH01] and [Slo08] who provided a set of constants and

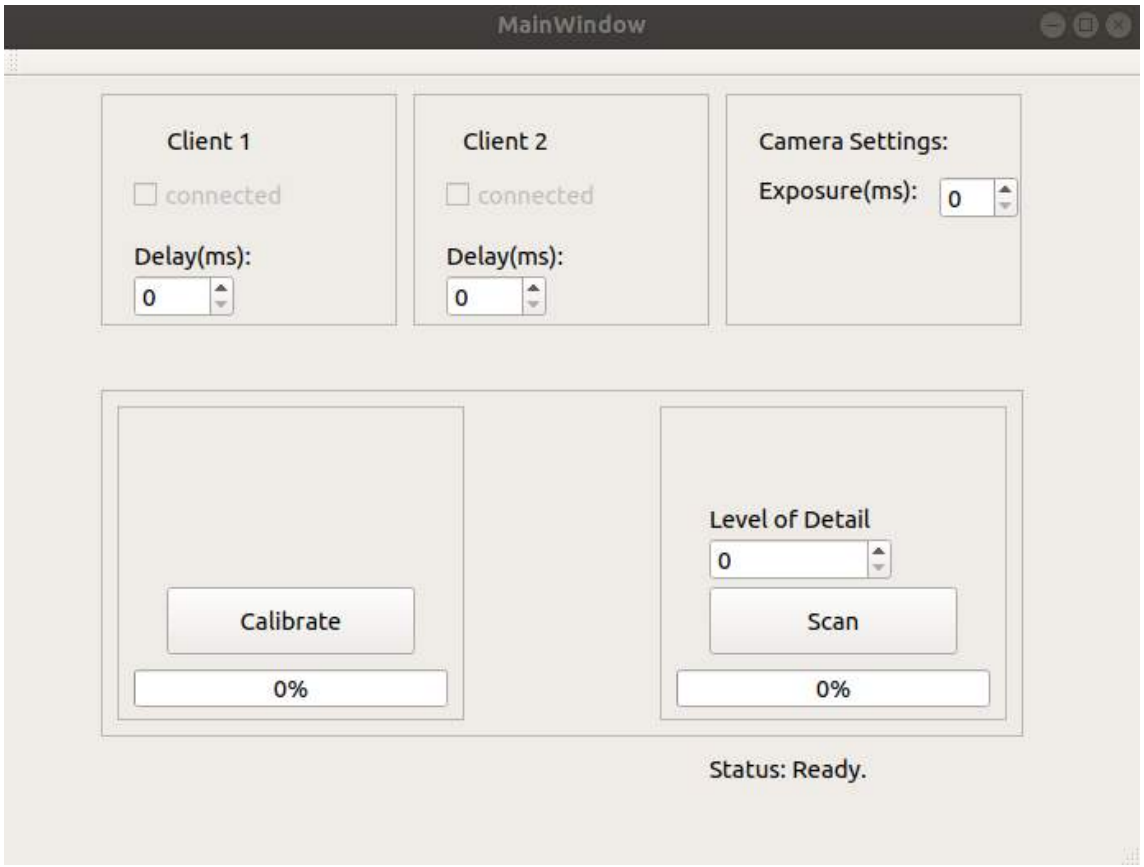
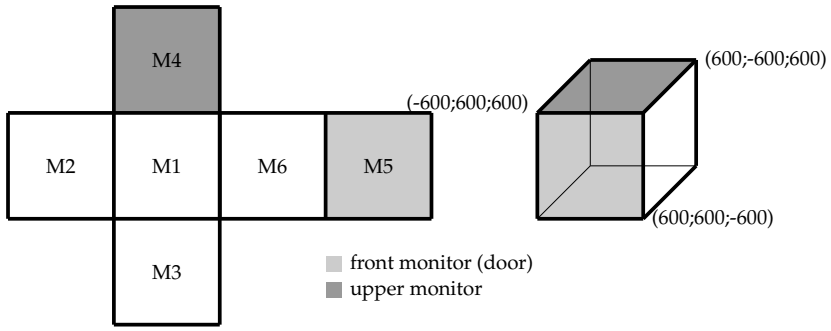


Figure 4.4: The server GUI.



Monitor ID	Origin – opposite point
Monitor 1	(-600;-600;600) – (600;-600;-600)
Monitor 2	(-600;600;600) – (-600;-600;-600)
Monitor 3	(600;600;-600) – (-600;-600;-600)
Monitor 4	(-600;600;600) – (600;-600;600)
Monitor 5	(-600;600;600) – (600;600;-600)
Monitor 6	(600;600;-600) – (600;-600;600)

Figure 4.5: The external positions and orientations of the monitors and their corresponding ID in the server program as well as the pixel's internal coordinates.

formulas to calculate the SH using the (x, y, z) coordinates. This is faster than using the initial formula of spherical harmonics, because only simple multiplications in time $O(1)$ are needed.
Then, the previously presented brightness normalization filter is applied on every square of the cube, to balance out the distance of every pixel to the cube's center point. Finally it stores all spherical harmonics as image files into the *SphericalHarmonics*-folder, divided into six quads with 1200×1200 pixels, each quad standing for one monitor. The images are stored as .bmp files which is a lossless and easy-to-read format.
The mapping function which maps the squared image to the correct monitor was done by trying different orientations of each square, finally yielding the result as seen in 4.5.

4.3.2 Clients

The purpose of the clients is to display the spherical harmonics on the monitors. The clients do not have their own GUI and have to be started after the server by opening the *build*-folder and typing

DISPLAY=:0.x \.Lightstage x

into the command line, where $x \in \{0, 1\}$, which is the assigned ID for the client. It is important to start the server first and then the clients, since the clients only try to

connect to the server once when they are being created. This can be fixed easily by calling a connect-function repeatedly until a connection could be established. When connecting with the server, the first message to the server contains its client ID to "log in" to the server. Then the client displays a default stand-by image while waiting for further input from the server. This standby-image is a simple, black image, yielding an ambient light response which is later explained in detail.
Each client is responsible for three monitors of the lightstage. It creates three windows, stored in an array, which are then transformed to the correct positions by shifting it $1920(1-x)+1920i$ pixels sideways, where i is the window's index in the array. The differentiation between the clients has to be made because the control monitor is assigned to the same screen as client 0, positioned at coordinate (0,0). Thus, all monitors of this screen have to be moved one monitor width to the side. Of course this could have been solved by changing the control monitor's position in the display settings, but since these settings could not be saved due to missing admin-rights, this seemed to be the best option at this point.
Now, when the client receives the message for displaying a specific SH Y_l^m, it loads the corresponding image files into the buffer, displays them on the correct monitor and returns a signal to the server when it is done.

5 Rendering

This chapter object information acquisition and render pipeline which takes the spherical harmonics responses and produces a rendered image out of them, given a set of virtual lights.
After the object is scanned, surface information is acquired by using different sets of the responses of the spherical harmonic's illuminations.
Each response $f_l^m = f \cdot Y_l^m$ of a spherical harmonic Y_l^m can be seen as the sum of the diffuse albedo response and the specular albedo response: $f(\omega) = D(\omega) + S(\omega)$.

5.1 Render basics

5.1.1 The render equation

The render equation computes the final color of a point x by considering viewing vector $-\omega_o$, scene illumination $L_i(x, \omega_i)$ from direction ω_i and surface-specific properties stored in the BRDF function $f_r(\omega_i, x, \omega_o)$. $L_e(x, \omega_o)$ describes the emitted light from surface point x towards the camera.

$$L_r(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(\omega_i, x, \omega_o) L_i(x, \omega_i) cos\theta_i d\omega_i \tag{5.1}$$

The term $cos\theta_i$ derives from Lambert's cosine law which is described in the next subsection.

5.1.2 Lambert's cosine law

Lambert's cosine law says that the radiant intensity of a Lambertian surface and thus ideally diffuse material with constant luminance is directly proportional to the cosine of the angle θ between the surface normal and the outgoing light direction. It follows from his law that the radiance of a Lambertian surface stays constant and has the same luminance for any view direction. Follwing his law, the irradiance E is calculated by

$$E = \int_\Omega L_i(x, \omega_i) \cdot cos\theta_i d\omega_i \tag{5.2}$$

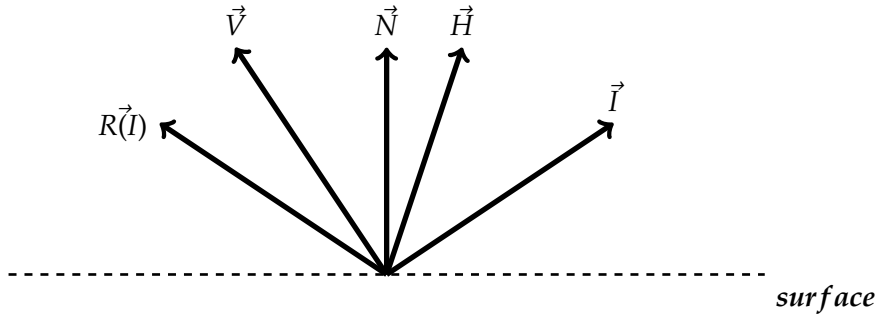


Figure 5.1: Phong illumination model

.

5.1.3 Lambertian material

For Lambertian materials, the BRDF function is constant since Lambertian materials are perfectly diffuse and do not depend on the viewing angle.
Setting the BRDF of the render equation to Lambertian and assuming it does not emit light on its own we get

$$L(x,\omega_o) = \int_\Omega \rho_d L_i(x,\omega_i) cos\theta_i d\omega_i \tag{5.3}$$

$$= \rho_d \int_\Omega L_i(x,\omega_i) cos\theta_i d\omega_i \tag{5.4}$$

where $0 \leq \rho_d \leq 1$ with $\rho_d = \pi \cdot k_d$ and k_d being the material-specific diffuse reflectance coefficient.

5.1.4 Phong shading

The Phong-shader [Pho75] linearly interpolates the normal for each surface point from its surrounding vertex-normals. The same applies to color values. The Phong illumiation model is not energy-preserving since it allows ambient light.
Building a render equation with the Phong illumination model, we get:

$$L_r = k_a L_a + \sum_{l \in Lights} k_d L_l (I_l \cdot N) + k_s L_l (R(I_l) \cdot V)^{k_e} \tag{5.5}$$

where I is the negative light direction, $R(I)$ its reflectance vector, V the negative view vector and H the halfway-vector between I and V. N describes the normal vector, as it is also visualized in figure 5.1. Furthermore k_a, k_d and k_s describe the constant and material-dependent ambient, diffuse and specular coefficients with $0 \leq k_d + k_s \leq 1$ and $0 \leq k_a \leq 1$. k_e describes the size of the specular highlights and width of the specular lobe.

5.2 Acquiring object information

5.2.1 Diffuse albedo

The diffuse albedo map of an object can be seen as the object's plain color without any illumination or shadows. This can easily be acquired by illuminating the object from all directions homogeneously. Luckily, the 0th order spherical harmonic provides this kind of illumination pattern, thus the diffuse map corresponds to the diffuse response D_0^0 of the 0th order SH, divided by π to cancel out the integral of Lambert's lobe function as explained in [TFG+13].

$$p_d = \frac{1}{\pi} D_0^0 / Y_0^0 \tag{5.6}$$

This formula only applies for non-specular and simple, convex-shaped objects. Otherwise, self-reflections and self-inflicted shadows might get baked into the diffuse map.

5.2.2 Specularity

The specularity map stores information about the specularity of an object, meaning how "shiny" it appears under different lighting conditions, and is stored as a value between 0(no specularity) and 1(completely specular). Inspired by [TFG+13] and [BLL96], the specularity is being acquired by the saturation shift for a pixel under different lighting conditions. For this I used the third order SH, since the responses above the second order match the specular lobe closely, as it is mentioned in [TFG+13]. To compute the specular value ρ_s I subtract the maximum saturation of all third order responses from the minimum saturation:

$$\rho_s = \Delta S_{response} \tag{5.7}$$

$$= \arg\max_m \; S(f_3^m Y_3^m) - \arg\min_m \; S(f_3^m Y_3^m) \tag{5.8}$$

5.2.3 Normal map

Normal maps are images which store information about the surface normals of an object by representing the x, y and z components in the RGB values. Of course this technique is limited resolution-wise to 256 steps around every axis($360°/256 steps \approx 1.4°/step$). This could be fixed by using floating point-based color representation instead. Currently it is only possible to store positive normal values, but this can be fixed by scaling and shifting the values to fit into the interval [0...255].

Diffuse normals The object's diffuse surface normals are acquired by using the diffuse responses of the first order spherical harmonics D_1^{-1}, D_1^0, D_1^1. They are acquired

by subtracting the response of the negative spherical harmonic values f_l^{m-} from the response of the positive values f_l^{m+}, resulting in

$$D_l^m = f_l^{m+} - f_l^{m-}$$

Since it is necessary to already know the values of the specular responses S_l^m in order to compute $D = f - S$, and S is very difficult to acquire, it is assumed that $D = f$. For every pixel p, the surface normal $\vec{n}_p$ is computed by first transforming all three spherical harmonic's responses into grayscale images using function g. Then, each resulting grayscale image is assigned to one color channel after normalizing its result:

$$\vec{n}_{diff} = \frac{1}{\sqrt{g(D_1^{-1})^2 + g(D_1^0)^2 + g(D_1^1)^2}} \begin{pmatrix} max(0, g(D_1^{-1})) \\ max(0, g(D_1^0)) \\ max(0, g(D_1^1)) \end{pmatrix} = \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{5.9}$$

Here I already embedded the limitations of RGB values by only taking the positive values.

Specular normals For specular surfaces, the normal acquisition is much more complicated and it would be out of the scope of this thesis to implement, but the rough idea by [TFG+13] is to search for the maximum peak of brightness when rotating the spherical harmonics around the object. This is done by using a hillclimbing algorithm by Sloan [Slo08]. The specular normal then corresponds to the half-way vector of the light direction and the viewing-vector. In this method the third order spherical harmonics are used, since these responses closely resemble the specular aspects of the material.

5.3 Rendering and re-illumination

5.3.1 Virtual lights

For relighting the objects, lights are needed which are placed in a virtual 3D space. For this thesis I limit myself to simple directional light sources. The lights consist of direction, color and intensity, where the direction is a 3-dimensional vector x, y, z, the color a 3-dimensional RGB-vector and the intensity a real positive value.

Figure 5.2: Ambient light on a mirror sphere.

Ambient light One disadvantage of the used monitors was that they are always emitting light even when they should display black pixel values. This results in an ambient light, which is always present in the scene. This should be represented in the render equation as well, extending it with the constant summand L_a. While in theory the ambient light is the same at every point in the scene, this is not the case in reality, where the ambient light is physics-based. Thus, it has to be measured at every point and then changed for every x in the render equation, changing it to $L_a(x)$. Figure 5.2 shows the ambient light illuminating a mirror sphere. Interestingly this light mainly comes from the lightstages' corner areas.

5.3.2 Render equation

The render equation puts all previously acquired surface information together and creates a re-illuminated version L_r, using a given pre-defined set of directional light sources:

$$L_r(\omega_o, x) = L_a(x) + \int_\Omega L(\omega_i) f_{Ward}(\theta_i, \theta_r)\, cos\theta_i\, d\omega \tag{5.10}$$

where $L_a(x)$ is the global ambient light, $L(\omega_i)$ is the incoming radiance from direction ω_i at point x and $f_{Ward}(\theta_i, \theta_r)$ the reflectance function which depends on the incoming and outgoing light direction.

Ward BRDF As reflectance function I used the BRDF model from Ward [War92], since it is easy to compute but still very close to measured data. For simplicity reasons, the material is assumed to be isotropic, so it can be computed as

$$f_{Ward}(\theta_i, \theta_r) = \frac{\rho_d}{\pi} + \rho_s \cdot \frac{1}{\sqrt{cos\theta_i cos\theta_r}} \cdot \frac{exp(-tan^2\delta/\alpha^2)}{4\pi\alpha} \tag{5.11}$$

where α is the standard derivation of the surface slope and δ the angle between normal $\vec{n}$ and half vector $\hat{h}$. ρ_d corresponds to the diffuse albedo and ρ_s to the specular albedo with $\rho_d + \rho_s \leq 1$.
This can be extended to anisotropic materials in future works by separating α into α_x and α_y.

5.4 Software implementation

The render software computes the normal, diffuse and specular map and the rendered image. They are then displayed on a seperate GUI as seen in Figure 5.4.

5.4.1 Image processor

The image processor class has useful functions for image manipulation as well as to compute all necessary input maps for the render function, taking the spherical harmonic respones f_l^m. It computes the normals, diffuse albedo, and specularity maps.

5.4.2 Renderer

The renderer is called every ten seconds. This value is set arbitrarily to set a fixed render frame rate. It can be decreased when it gets clear that the renderer does not need that much time to render. After being called, it loads all current SH responses from the storage, sends them to the image processor and creates a rendered image out of the acquired information. Finally it sends all images as well as the rendering to the GUI.
It uses the preiously presented render equation to calculate the result and stores them as image files. It is also possible to put dynamic lights in the scene which change their position every frame. This is done by storing the light array outside of the function and changing their position on every call of the render function.
Practically, the render equation is implemented by simply iterating over all light sources $l \in L$ for a pixel x on the camera plane:

$$L_r(\theta_r, x) = \sum_{l \in L} L_l(\omega_i) f_{Ward}(-\theta_{li}, \theta_r)\ cos(-\theta_{li}) \tag{5.12}$$

$$L_l(\omega_i) = \begin{cases} 0 & \text{if } \theta_{li} \cdot \vec{n} \geq 0 \\ L_l & \text{else} \end{cases}$$

which is then used on every pixel x within the render area A to create the final rendered image R_A.

$$R_A = \sum_{x \in A} L_r(\theta_r, x) \tag{5.13}$$

where θ_r is calculated by taking the dot product of the acquired normal vector $\vec{n}$ from the normal map and the camera rotation $\vec{C_r}$:

$$\theta_r = \vec{n} \cdot \vec{C_r}$$

and θ_{li} by computing the dot product of the light direction and the surface normal:

$$\theta_{li} = l$$

The camera's position is assumed to be at the origin of the scene, and rotated towards the center of the lightstage which leads to a rotation vector of $\vec{C_r} = \begin{pmatrix} -1 & -1 & -1 \end{pmatrix}^T$. To determine whether a light source is behind or in front of a surface, the dot product $\theta_{li} \cdot \vec{n}$ is calculated. If the value is greater or equal to 0, the light source is behind the surface and does not have to be put into the render equation.

Ambient light The ambient light has to be measured for every scene seperately by taking an image of the scene while the monitors are displaying images with all pixel being black. This is then subtracted from all further acquired images.
Due to the limited time, it was not possible to implement this functionality in this work.

Render Area To speed up the render process I introduced a rectangular render area (x_1, y_1, x_2, y_2) defined by the opposite points (x_1, y_1) and (x_2, y_2) within the image borders, which crops the area to render und thus less pixels have to be computed. This can speed up the rendering by up to 2 seconds as seen in figure 5.3.

GUI The GUI displays all acquired information as well as the final rendering. An image of the GUI can be seen in Figure 5.4. On the left side it displays the final rendering, on the right the acquired surface informations. It is updated automatically after the next rendered image is computed.

5.5 Render results

After acquiring the object data and rendering the corresponding images, it is now time to evaluate the render results. For testing, I used three different objects. One diffuse sphere, one mirror sphere and one object which has diffuse and specular aspects.

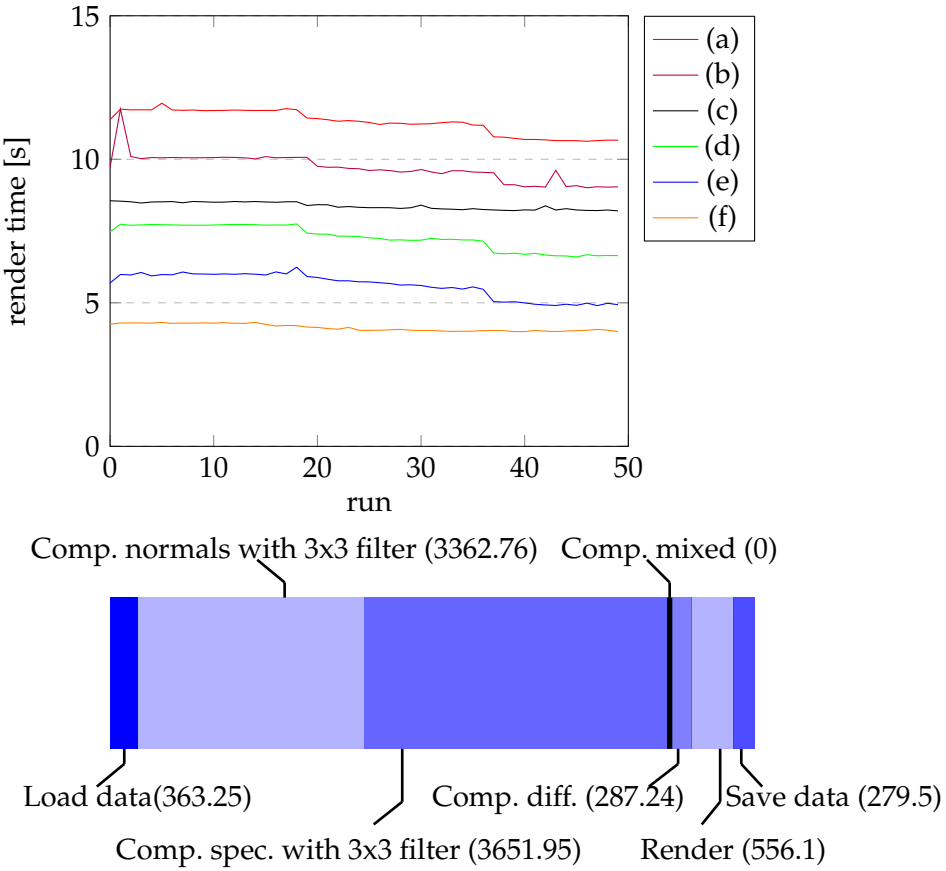


Figure 5.3: Above: Render time comparison from 50 runs. (a) no render Area, 5×5 median filter, (b) no render area, specular and normal map are filtered (both 3x3), (c) 800×800 render area, specular and normal map are filtered (both 3×3), (d) no render area, normals are filtered by a 3×3 median filter, (e) no render area, no filters, (f) 800×800 render area, no filters. Below: Breaking down the overall render time into its aspects, taking the average times of 20 runs. The time is written in brackets behind each aspect (milliseconds).

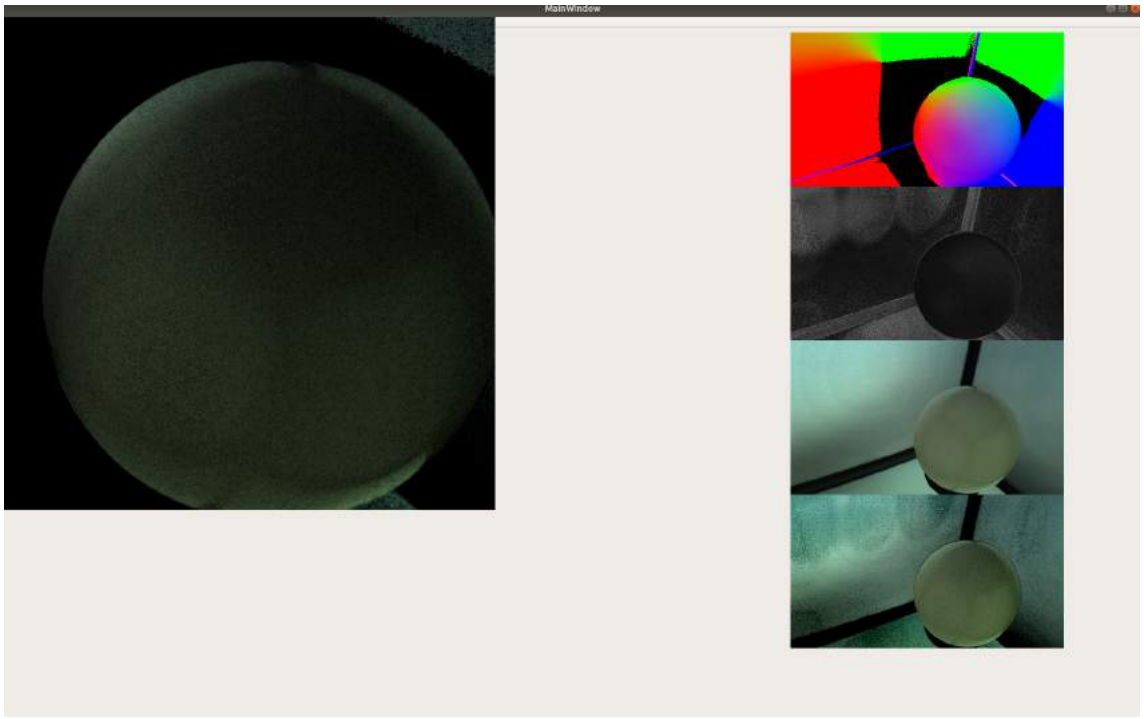


Figure 5.4: GUI of the renderer. Left: Rendered image. Right from top to bottom: Normals, speculars, mixed albedo, diffuse albedo.

(a) (b) (c)

Figure 5.5: Comparing the (a) diffuse, (b) specular and (c) mixed maps of a diffuse sphere.

5.5.1 Diffuse map

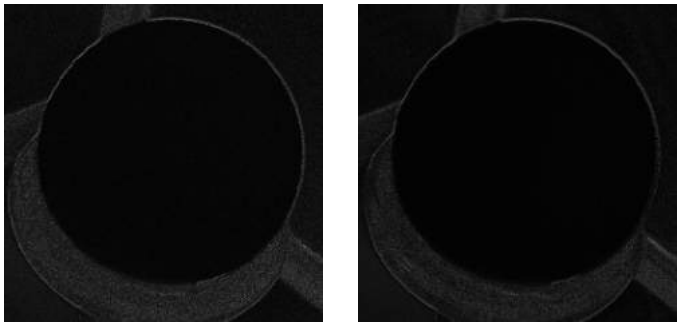
The diffuse and mixed maps recreate the diffuse color of the diffuse sphere very well. A comparison of the recreated diffuse, mixed and ground truth object can be seen in figure 5.5. In the lower part of the sphere, where it comes closer to the object holder, the shadows are baked into the diffuse map and due to errors in the specular map it gets a thin black outline.

5.5.2 Specular map

The specular map is a grayscale image showing the specularity of an object's surface. This means, diffuse objects should appear black, while highly specular surfaces appear brighter. Looking at the result of the diffuse sphere in figure 5.9, this is close to the expected result, ignoring the average noise for a second. Noticable is the white line at the edge of the sphere. I assume this is the result of the Fresnel-effect, showing that the diffuse sphere is not perfectly diffuse but has a small specularity aspect too at high angles.

5.5.3 Noise reduction

In the beginning, the normal map and the specular albedo map were very noisy. The first assumption was that it is due to the camera which is very noisy in dark areas, especially when it is heated up. To test this assumption, I was using the average value of multiple scans for every image. This improved the image quality a bit as it can be seen in Figure 5.6. As this feature is not implemented in the lightstage, this had to be done manually by scanning multiple times and storing each scan's responses into a specific subfolder. In the future this may be directly implemented into the scanning pipeline to calculate the average values on the fly and storing the



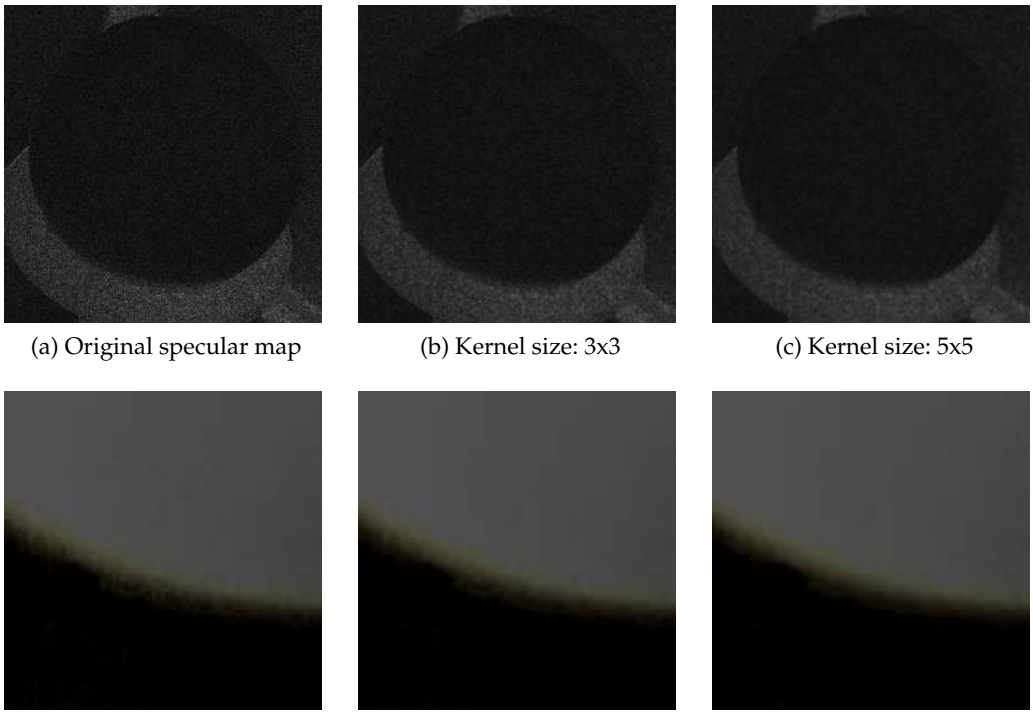
(a) (b)

Figure 5.6: Noise reduction by taking multiple images and using their average values for further processing. (a): noisy image. (b): Average values of five images.

result which decreases the memory costs.
To fix the noise that was still occuring, I applied a simple median-filter on the images to filter out the salt and pepper noise. Trying out different kernel sizes (Figure 5.7) showed that a kernel size of 3x3 pixels yields the best looking result. Bigger kernels did not improve the noise and also increased the render time too much. Although the high cost can be fixed and even be done in $O(1)$ regarding the kernel size ([PH07]), this would not solve the noise problem.
Further improvements to denoise the result include more advanced denoising techniques, namely bilateral filtering. Here the edges are preserved while denoising or smoothing the areas in between. For this, we need to know where the edges are. Luckily, the edges can be seen very clearly on the mixed albedo map. By taking the edges from the mixed albedo, the specular map can be filtered while still keeping these edges. This was not done in this work, but can be easily implemented in the future.

5.5.4 Normals

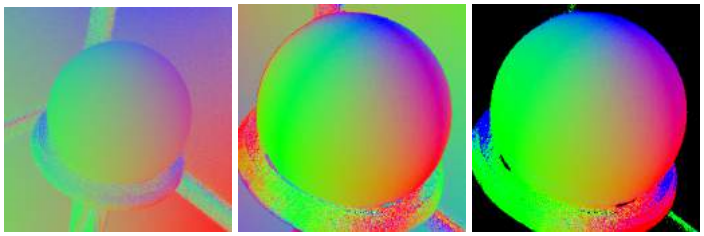
The normal map is acquired by the 1st order spherical harmonics and seems to return good results for diffuse surfaces, as it can be seen on the diffuse sphere example in figure 5.9.
In the beginning I was using the positive SH values only to acqurire the normal vectors. But this solution was not very precise, since it omits half of the information. The improved normal maps are using all spherical harmonics values by first taking the f_1^m SH and then subtracting the $\hat{f}_1^m$ from it, taking the absolute values. This converts all normal vectors to only having positive values which means the results are not correct for surfaces which have negative components in their normal vectors. The final version takes the idea of the previous one, but only takes the positive normal values into its result, omitting all vectors which have only negative parameters. This



(a) Original specular map (b) Kernel size: 3x3 (c) Kernel size: 5x5

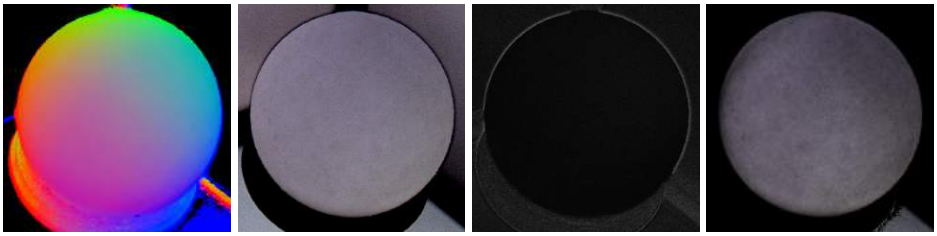
(d) Original specular map (e) Kernel size: 3x3 (f) Kernel size: 5x5

Figure 5.7: Mean filter results. (a)-(c): specular map. (d)-(f): final render



(a) (b) (c)

Figure 5.8: Acquiring normals (a) by only using the response of the scaled and shifted SH values, (b) subtracting the complementary SH response and taking the absolute values, (c) taking only the positive normal values because storing negative values has not been implemented yet.



(a) Normal (b) Diffuse (c) Specular (d) Render

Figure 5.9: Rendering results diffuse sphere.

results in a nice side-effect which creates the black background leading to a strong contrast to the object.
One disdvantage of this method is that all negative normal values are lost and capped at 0. Most of the time this will not affect the result dramatically since the camera has a view direction of $\begin{pmatrix}-1 & -1 & -1\end{pmatrix}^T$, but there are cases where surfaces are visible from the camera even when having negative surface normal values. Applying the above mentioned fix of shifting and scaling to enable negative values as well should solve this problem.

5.5.5 Dynamic lights

Because the scene is re-rendered after a specific time, it is possible to move the lights around between the frames. I tested this by rotating the light source by 5 degrees before the next frame is being rendered, resulting in an image sequence of 180 images which show a light source rotating around the object. Parts of the sequence can be seen in figure 5.13. When looking at the shadows, it can be seen that the sphere is not stored as a perfectly round sphere, since the edge between light and dark parts of the sphere does not develop along the longitudes of the sphere as the angle changes. This indicates incorrect normal values.

5.5.6 Specular and mirror objects

Looking at the results from a mirrored sphere and a more complex object with diffuse and specular aspects (figure 5.10 and figure 5.11), it becomes clear that the current pipeline does not support specular materials. The normal map is only showing diffuse normals, and the specular map does not really register specular surfaces, since they should be much brighter.

Re-illumination from environment maps There is another technique for re-illumination than the one presented before. Instead of using only virtual lights illuminating a structure defined by the normal map and colored by the diffuse

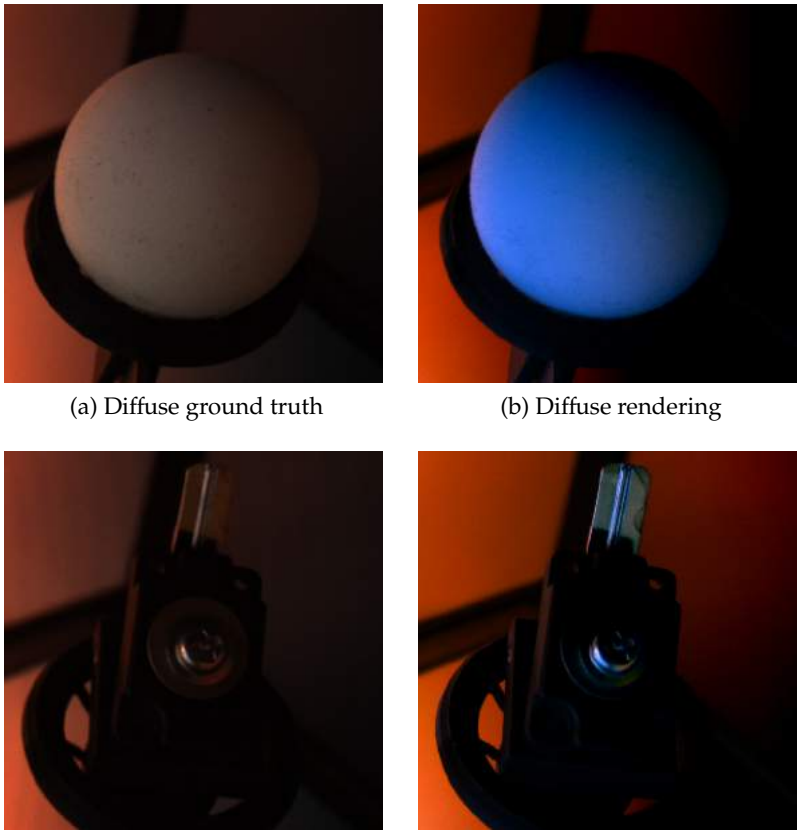
(a) Normal (b) Diffuse (c) Specular (d) Render

Figure 5.10: Rendering results mirror sphere.

(a) Normal (b) Diffuse (c) Specular (d) Render

Figure 5.11: Rendering results of an object containing diffuse and specular elements.

map, it is possible to apply any environment map illumination pattern onto a scene. Spherical harmonics are used to simplify incoming spherical radiance and transforming the incomming radiance into the SH-space. As the maximum SH order increases, the quality and resolution of this transformation gets higher and closer to the original illumination pattern. Transforming a re-illumination pattern into spherical harmonic parameters and summing up all spherical harmonic responses of the object results in the object as it would be illuminated by the new lighting conditions.
I tested my lightstage by using the Grace Cathedral Illumination coefficients provided by Ramamoorthi and Hanrahan [RH01]. The result and ground truth can be seen in figure 5.12. It is easily recognizable that there are differences in color. The rendered image has a more dominant blue color component. My first assumption was that it is due to wrong white balance on the camera, but the problem still appeared after using only grayscaled images for re-illumination. This needs to be investigated in the future.
Note that this result is only calculated by summing up the different spherical harmonic responses weighted by the given coefficients and does not use the normals or any other acquired object information.



(a) Diffuse ground truth (b) Diffuse rendering

(c) Complex ground truth (d) Complex rendering

Figure 5.12: Testing the results by comparing the ground truth response from illumination by the Grace Cathedral illumination pattern with the re-illuminated version. I tested this for a diffuse sphere and an object containing highly specular parts as well.

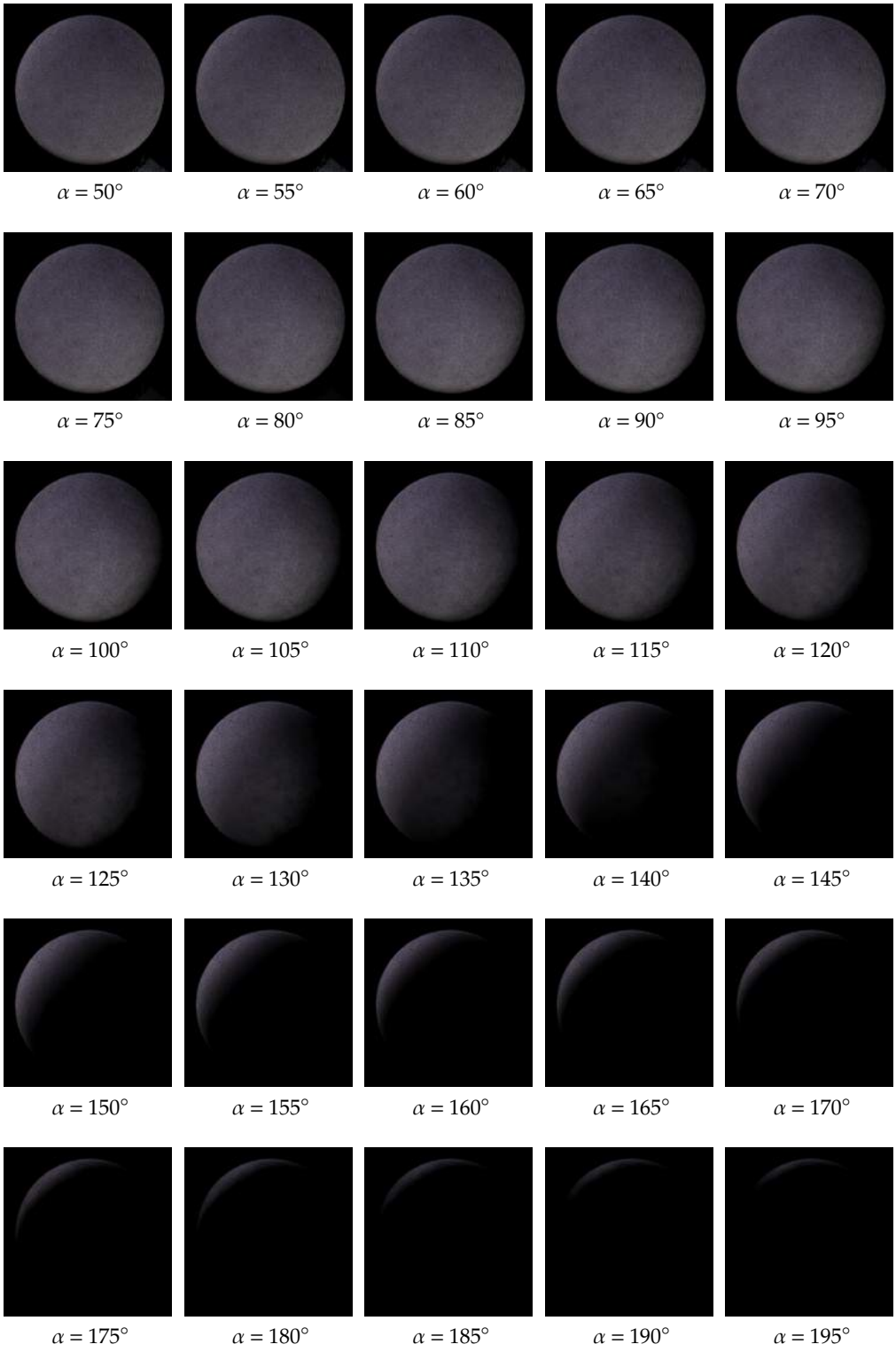


Figure 5.13: Rendering result of a rotating lightsource(around y-Axis)

6 Conclusion

After looking at the render results it is now time to do a conclusion and to check if the monitor-based lightstage is as good as the other methods and what its advantages and disadvantages are.

6.1 Discussion

Reflections One major disadvantage of monitors over a light arc are the inter-reflectancies between monitors. When monitors emit light, this will bounce off the other monitors, creating a low level ambient light and falsifying the responses. This could be improved by using matt monitors which reflect less light. Another, more mathematical approach would be to caclulate the reflected light and subtracting it from the response.

Monitor borders Another disadvantage of using monitors are lost pixels due to borders. This results in uneven distribution of pixels over the lightstage. For better results these borders have to be determined via a calibration algorithm. Alternatively, monitors without borders could be used.

Size In this work the lightstage had a size of $36.2 \times 36.2 \times 36.2cm$. By increasing the size of the lightstage, the brightness of the monitors or the exposure time of the camera has to be increased in order to get responses bright enough for further processing. Changing the lightstage's size could improve the quality in theory because the lightsources are closer to its assumed infinite distance when it is not calibrated. Alternatively it might be a good idea to direct and focus the light from the monitors directly to the center point to simulate a infinite distant lightsource.

Render time The rendering is the state which needs the most time in the pipeline. But it is also the state which can be improved the most. After introducing a render area, this time could already be drastically improved.
As figure 5.3 shows, the computation of the specular map needs the longest of all steps, followed by the normal acquisition. It is advisable to concentrate most the acceleration efforts on these parts of the render pipeline.

Real black values The monitors which were available for this lightstage are not able to show completely black values, so there was always an ambient light which has to be subtracted of every response. While this is not very expensive to do, it is still an aspect which could be improved by using monitors which can display real black values such as OLED displays.

Noise The noise in the rendered image was a big obstacle which already could be improved. The noise can be further reduced by taking more images of the same response during the scan process, only storing the average response as an image file.

Normalizing monitor brightness This step brought improvement to the final result. But it is still not perfect, because the brightness change should be determined via measuring the amount of emitted light for different viewing angles. A spectral analysis of the outgoing light from different angles would help as well, since the colors seem to shift a bit when changing the angle. Also the saturation seems to decrease with larger angles.

Resolution The resolution of the monitor-based lightstage is much higher than in previous approaches using a light arc or a LED sphere. With monitors it is possible to create nearly continuous illumination. The illumination density is larger closer to the monitor's borders, since more pixels are distributed over the same area of the unit sphere. This could be fixed using rounded monitors or building a bigger sphere to minimize this effect. Theoretically it could be also possible to use monitors of higher pixel density in the center.
This resolution enables possiblities for different lighting patterns such as wavelet-based methods.

Real time It is definitely possible to make the scanning and rendering processes faster. Rendering can be improved by parallelizing without losing quality in the results as discussed in the paragraph "Render time".
The scanning process can be accelerated by loading all spherical harmonics into the GPU to prepare the actual scanning process.
It could be also possible to select the render area in the server-GUI before starting any scanning procedure, to only capture the area of interest and thus speeding up the camera's capture process. Additionally it reduces the image file sizes, which speeds up the loading and saving. In the rendering stage, loading the image files was the most expensive part.

6.2 Future work

There are many aspects which may improve the this work by a lot. In the following section I am listing some of the most important improvements for future work.

Specular map In this work the specular map is acquired by a very simplified method to show the general functionality of the lightstage. A better and more state-of-the-art method has to be implemented.

Improving the normal map Enabling the usage of negative normal values is an easy and fast-to-implement improvement. This might improve the quality of the normal map and the rendered image drastically.

Faster rendering One way to speed up the real-time render pipeline is the parallelization of the processes. Each frame could be computed on a different thread, increasing the render speed with the number of threads available.
Second, it is possible to decrease the time the renderer needs to load the images into the memory by loading all images into the GPU-memory directly when taking the pictures. It is furthermore advisable to use the GPU,which is optimized for image processing, for image calculations instead of the CPU.

Changing brightness depending on the monitor angle As clearly seen in figure 5.2 the received light from the monitors depends highly on the angle to the monitors. This has to be compensated by measuring the shift in brightness and color depending on the angle. It might be a good idea to investigate the possible sift in saturation as well.

Multiple scans Since the camera shows a high noise level especially in darker areas it is highly advisable to take multiple scans of each sh-response and only storing the average response for further processing. The number of scans might be based on the desired balance between the quality of the result and the available scanning time.

Better noise filter As mentioned in the previous chapter, it is also possible to use more advanced noise-filters to filter out most of the remaining noise.

Join all programs together The most fundamental change i propose is to join the server program with both clients, as well as joining the lightstage with the renderer. This would make it a lot easier and more efficient to operate. It also solves the

problem of stacked and appended socket messages of multiple scans which appear when the lightstage scans too fast.

Calibration By implementing a calibration the quality of the result could be improved drastically.
In this work it is assumed that the monitor lightsources are infinitely far away to simplify the calculations. However in reality, the monitors are very close to the illuminated object. The distance of the monitors can be determined exactly by calibrating it using a mirrored sphere.

Unused pixels Due to my method of shifting the monitors in the construction phase I managed to lose less pixels to the borders. However, I never used these newly acquired pixels and the displayed images are still quadratic. This could be solved in the future by determining the new image sizes by testing or by calculating it with the given pixel density.

Combine environment maps and rendering Currently, the environment map illumination result and the virtual light illumination render result are two seperately calculated results. It is possible to combine these techniques by interpreting the environment map as infinitely distant illumination. By taking this map into account in the render equation and calculating the reflected light for every point on the object's surface by using the previously acquired normal, specular and diffuse map it should be possible to achieve even more realistic results.

6.3 Final conclusion

By looking at the results, using monitor-based lightstages can be a first step in the direction of continuous illumination. Monitor-based lightstages have the advantage of high frequency sampling, enabling the illumination by different and more precise illumination patterns. However, the improvement is only recognizable when using high-resolution cameras, scanning objects close to the camera or scanning surfaces with high complexity which otherwise might cause aliasing-effects. Interreflection between monitors is also a problem which traditional lightstages do not have to this extent. This should to be solved in the future.
Although monitors can have a refresh rate of $244Hz$ or higher, they can not compete with the high-speed scanning process of Wenger et al.[[WGT+05]]. The advantages of monitor-based lightstages definitely lie with the continuous illumination instead of high-speed scanning. But this might not be necessary for real-time acquisition, since there are methods which only need a few images in order to give good-looking render results([XSHR18], [MHP+19]). My presented lightstage can be improved in

many ways such as applying it to specular surfaces, fixing the normal and specular map and combining the different parts of the pipeline to make the overall process less cumbersome. Overall, I think my work gives a good base for creating better, more complex solutions which might create renderings of much higher quality.

Bibliography

[BLL96] Ruzena Bajcsy, Sang Wook Lee, and Ales Leonardis. Detection of diffuse and specular interface reflections and inter-reflections by color image segmentation. *International Journal of Computer Vision*, 17(3):241–272, 1996.

[CGS06] Tongbo Chen, M. Goesele, and H.-P. Seidel. Mesostructure from specularity. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1825–1832, 2006.

[CK19] Bor-Chun Chen and Andrew Kae. Toward realistic image compositing with adversarial learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8407–8416, 2019.

[CZH+00] Yung-Yu Chuang, Douglas Zongker, Joel Hindorff, Brian Curless, David Salesin, and Richard Szeliski. Environment matting extensions: Towards higher accuracy and real-time capture. *Proc. of SIGGRAPH 2000*, 06 2000.

[DHT+00] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, and Mark Sagar. Acquiring the reflectance field of a human face. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 145–156, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[DWT+02] Paul Debevec, Andreas Wenger, Chris Tchou, Andrew Gardner, Jamie Waese, and Tim Hawkins. A lighting reproduction approach to live-action compositing. *ACM Transactions on Graphics*, 21, 06 2002.

[ECJ+06] Per Einarsson, Charles-Felix Chabert, Andrew Jones, Wan-Chun Ma, Bruce Lamond, Tim Hawkins, Mark Bolas, Sebastian Sylwan, and Paul Debevec. Relighting human locomotion with flowed reflectance fields. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, EGSR '06, page 183–194, Goslar, DEU, 2006. Eurographics Association.

[FLBS07] Martin Fuchs, Hendrik P. A. Lensch, Volker Blanz, and Hans-Peter Seidel. Superresolution reflectance fields: Synthesizing images for intermediate light directions. In Daniel Cohen-Or and Pavel Slavík, editors, *Computer Graphics Forum (Proc. EUROGRAPHICS)*, volume 26(3), pages 447–456, Prague, Czech Republic, September 2007. The European Association for Computer Graphics, Blackwell.

[GCP+09] Abhijeet Ghosh, Tongbo Chen, Pieter Peers, Cyrus Wilson, and Paul Debevec. Estimating specular roughness and anisotropy from second order spherical gradient illumination. 2009.

[MHP+19] Abhimitra Meka, Christian Haene, Rohit Pandey, Michael Zollhoefer, Sean Fanello, Graham Fyffe, Adarsh Kowdle, Xueming Yu, Jay Busch, Jason Dourgarian, Peter Denny, Sofien Bouaziz, Peter Lincoln, Matt Whalen, Geoff Harvey, Jonathan Taylor, Shahram Izadi, Andrea Tagliasacchi, Paul Debevec, Christian Theobalt, Julien Valentin, and Christoph Rhemann. Deep reflectance fields - high-quality facial reflectance field inference from color gradient illumination. volume 38, July 2019.

[PD03] Pieter Peers and Philip Dutré. Wavelet environment matting. In *Proceedings of the 14th Eurographics Workshop on Rendering*, EGRW '03, page 157–166, Goslar, DEU, 2003. Eurographics Association.

[PD05] Pieter Peers and Philip Dutré. Inferring reflectance functions from wavelet noise. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, EGSR '05, page 173–182, Goslar, DEU, 2005. Eurographics Association.

[PH07] Simon Perreault and Patrick Hebert. Median filtering in constant time. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 16:2389–94, 10 2007.

[PHD06] Pieter Peers, Tim Hawkins, and Paul Debevec. A reflective light stage, 2006.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

[RDL+15] Peiran Ren, Yue Dong, Stephen Ching-Feng Lin, Xin Tong, and Baining Guo. Image based relighting using neural networks. *ACM Transactions on Graphics (TOG)*, 34:1 – 12, 2015.

[RH01] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 497–500, New York, NY, USA, 2001. Association for Computing Machinery.

[SBT+19] Tiancheng Sun, Jonathan T. Barron, Yun-Ta Tsai, Zexiang Xu, Xueming Yu, Graham Fyffe, Christoph Rhemann, Jay Busch, Paul Debevec, and Ravi Ramamoorthi. Single image portrait relighting. *ACM Transactions on Graphics*, 38(4):1–12, Jul 2019.

[SCKSS21] Soumyadip Sengupta, Brian Curless, Ira Kemelmacher-Shlizerman, and Steve Seitz. A light stage on every desk, 2021.

[SHS+17] Zhixin Shu, Sunil Hadap, Eli Shechtman, Kalyan Sunkavalli, Sylvain Paris, and Dimitris Samaras. Portrait lighting transfer using a mass transport approach. *ACM Trans. Graph.*, 37(1), October 2017.

[SKCJ18] Soumyadip Sengupta, Angjoo Kanazawa, Carlos D. Castillo, and David W. Jacobs. Sfsnet: Learning shape, reflectance and illuminance of faces 'in the wild'. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[SKS02] Peter-Pike J. Sloan, J. Kautz, and John M. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002.

[Slo08] Peter-Pike Sloan. Stupid spherical harmonics(sh) tricks. 2008.

[SS95] Peter Schröder and Wim Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 161–172, New York, NY, USA, 1995. Association for Computing Machinery.

[SXZ+20] Tiancheng Sun, Zexiang Xu, Xiuming Zhang, Sean Fanello, Christoph Rhemann, Paul Debevec, Yun-Ta Tsai, Jonathan T. Barron, and Ravi Ramamoorthi. Light stage super-resolution: Continuous high-frequency relighting, 2020.

[TFG+13] Borom Tunwattanapong, Graham Fyffe, Paul Graham, Jay Busch, Xueming Yu, Abhijeet Ghosh, and Paul Debevec. Acquiring reflectance and shape from continuous spherical harmonic illumination. *ACM Trans. Graph.*, 32(4), July 2013.

[War92] Gregory J. Ward. Measuring and modeling anisotropic reflection. *SIGGRAPH Comput. Graph.*, 26(2):265–272, July 1992.

[WGT+05] Andreas Wenger, Andrew Gardner, Chris Tchou, Jonas Unger, Tim Hawkins, and Paul Debevec. Performance relighting and reflectance transformation with time-multiplexed illumination. *ACM Trans. Graph.*, 24(3):756–764, July 2005.

[WMP+06] Tim Weyrich, Wojciech Matusik, Hanspeter Pfister, Bernd Bickel, Craig Donner, Chien Tu, Janet McAndless, Jinho Lee, Addy Ngan, Henrik Wann Jensen, and Markus Gross. Analysis of human faces using a measurement-based skin reflectance model. *ACM Trans. Graph.*, 25(3):1013–1024, July 2006.

[XSHR18] Zexiang Xu, Kalyan Sunkavalli, Sunil Hadap, and Ravi Ramamoorthi. Deep image-based relighting from optimal sparse samples. *ACM Trans. Graph.*, 37(4), July 2018.

[ZWCS99] Douglas E. Zongker, Dawn M. Werner, Brian Curless, and David H. Salesin. Environment matting and compositing. In *Proceedings of ACM SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 205–214. ACM Press / ACM SIGGRAPH / Addison Wesley Logman, July 1999.