UTRECHT UNIVERSITY

ICA-3836266

Automotive Camera Simulation for Testing ADAS and Active Safety Systems

by

Dick Booisma

Supervisors:
dr. ing. J. Bikker (Utrecht University)
ir. M. Wantenaar (Tass International)
MSc. M. Phillips (Tass International)

Second examiner:
dr. A. Vaxman (Utrecht University)

October 2017

Utrecht University

tass international

Abstract

In recent years the automotive industry has increasingly used virtual camera sensor simulation to test advanced driver assistance systems and active safety systems. These simulations need to be as close to the real world as possible to be representative for real world scenarios. An accurate light representation in the render engine is of paramount importance to the simulation's realism. Also weather effects have a large impact on the performance of vision based controlling systems. In this thesis we investigate both subjects. We construct a custom color space based on the sensitivity curves of an automotive camera to better approximate the light in a render engine that is only able to process trichromatic color vectors. Our results show however that using single wavelength primaries is not the solution for doing this. Furthermore, we show how water drops on the windscreen can be simulated with both our ray tracer implementation and a direct mapping method. We demonstrate that internal reflections become more important for the end result when the water drops have a large contact angle or when they are at an inclination angle with respect to the lens of the camera. The direct mapping method is not able to simulate these reflections.

Contents

List of Figures

Chapter 1

Introduction

Until a few years ago, camera sensors were only used for extending the drivers rear view to allow for easier parking. However, besides radar and LIDAR, the camera sensor is gaining interest as a viable option to serve as input for Advanced Driver Assistance Systems (ADAS). Applications that are hard to build with radar or LIDAR like traffic sign recognition, lane keeping or robust pedestrian detection are much easier with a camera sensor in combination with computer vision algorithms.

A lot of testing is involved in the process of developing controlling systems for ADAS. This testing can be done by taking finished control system prototypes to the road but it would be much more convenient to test earlier on in the process. With the simulation of a camera sensor, the controlling system can be tested with total control over the virtual environment. Also dangerous situations can be avoided, hundreds of tests of varying scenarios can automatically be performed sequentially and easy resetting of tests allows for accurate repeatability.

However, a problem may arise. When the modeled virtual domain is not a good representation of the real world domain, the tests are not a good representation of how well the system performs in the real world. Also when artificial intelligence algorithms train on an inaccurate virtual domain, the algorithm will never work well in a real world application. Therefore, the main research question that the automotive industry wants to answer with respect to sensor simulation is:

How can we render images of traffic related situations that match the output of automotive cameras as close as possible taking also the weather into account?

The addition of weather effects is important because it could potentially reduce the vision severely with large consequences for the performance of any vision based algorithms. Testing in adverse weather conditions and making systems robust in situations like heavy rainfall or fog can increase the overall safety of the system.

The research that we perform is in cooperation with Tass International (a Siemens Business). Tass International specializes in testing passive car safety, active car safety

and ADAS. Passive safety systems like seat belts, airbags or child seats try to keep the passengers of the car save during a crash whereas active safety systems, which is a subset of ADAS, try to prevent the crash from happening in the first place.

Tass International currently has a software tool called PreScan [Pre] that is already able to simulate sensors like LIDAR, radar and also the camera [RMBvMvdV15] to test ADAS and active safety systems. Figure 1.1 shows a schematic overview of their camera simulation model. Tass International continuously improves the tool which gave us the opportunity to work together and make a small contribution towards better simulations.

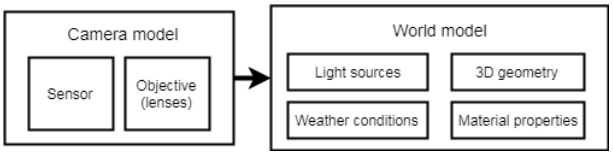

FIGURE 1.1: The camera simulation model

Answering the main research question exhaustively exceeds the scope of a master thesis. Therefore, we focus on two subtopics. Tass International asked us to look into color representations because it is a fundamental part of every realistic simulation. The way light is perceived by a camera is mainly dependant on the sensor characteristics. However, when light in the scene is not modeled correctly the virtual camera is also not able to perceive it correctly.

Besides color representations, they also asked us to look into weather effect simulation using a ray tracer. Of this, we chose water drop simulation on the windscreen because we think this is the most limiting factor in vision based controllers. We captured these subjects with the following sub research questions:

- How can color be represented in a render engine to achieve physics based simulation of the color characteristics of an automotive camera?
- How can stationary water drops on the windscreen in front of an automotive camera be simulated?

Both subjects are connected to the main research question, but are different on their own. Therefore, we divided the thesis in two parts. The first part will discuss the different kinds of representing color and the second part tries to find an answer to the second question about stationary water drops on the windscreen.

Part I

Color

Chapter 2

Previous work

Visible light is part of the electromagnetic spectrum as shown in Figure 2.1 and can be represented in multiple ways within the context of rendering. It depends on the application what method should be chosen. This section will discuss the different ways to represent a light spectrum and also how to use it in a render engine. First we give a little background on the different kinds of spectral curves.

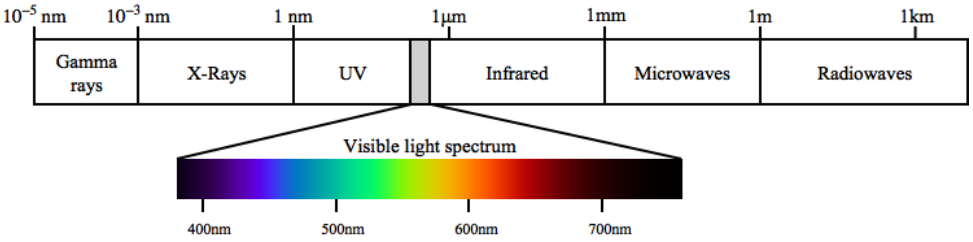

FIGURE 2.1: The place of the visible light in the electromagnetic spectrum.

A *spectral power distribution* (SPD) describes the amount of energy emitted by a light source as a function of the wavelength. Figure 2.2 shows examples of an incandescent and a fluorescent SPD. The smoothness of the two spectra differ to a high degree. This can result in difficulties when they need to be represented in an efficient format.

The *spectral reflectance curve* describes per wavelength a percentage of light energy that is absorbed by a material when light hits the surface. Similar to the reflectance curve, the *transmissivity curve* describes the amount of light that is absorbed when light is traveling through a medium. The curves are usually measured in the range of 380 – 780nm because this is the range of the spectrum that is visible for the human eye [Gla95]. Consumer cameras normally have similar sensitivity characteristics in order to mimic the eyes response as close as possible. However, for automotive cameras the sensitivity can be extended into the near infrared region of the spectrum which is towards the 950nm. Especially for night vision applications this can be beneficial.

When it is not relevant which type of spectral curve is used, we simply use the term *spectrum*. Otherwise we will mention whether it is a *spectral power distribution*, *reflectance distribution* or *transmissivity distribution*.

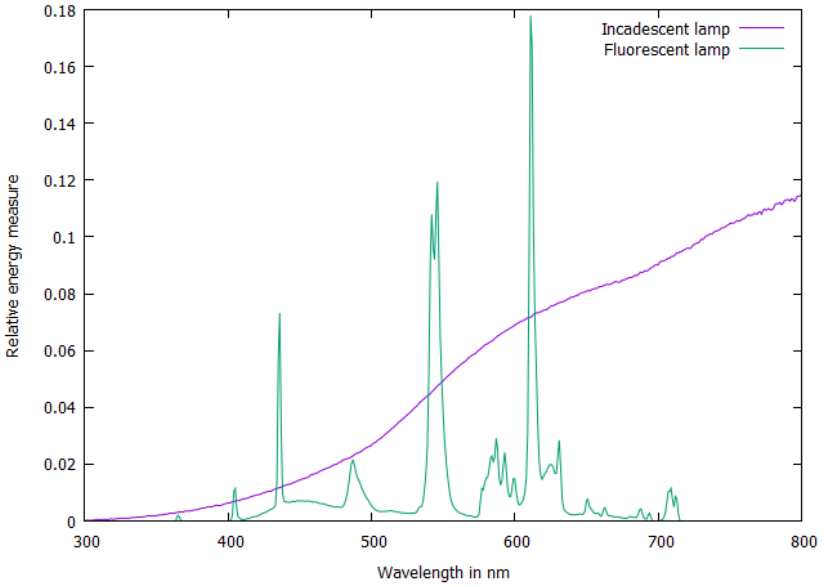

FIGURE 2.2: An example of a smooth incandescent lamp SPD and a spiky fluorescent lamp SPD. Taken from the Lamp Spectral Power Distribution Database [LSP14]

2.1 Trichromacy

Representing color using a trichromatic approach is based on the anatomy of the human eye. The eye contains cones in the retina that are sensitive to three different spectral ranges [HDvM+14]. The brain takes the response of the partly overlapping sensitivity curves in order to perceive color. Besides the cones, the retina also contains rods. Rods are sensitive to a single large range of the radio wave spectrum and are the most active under low light conditions. The cones are more sensitive under normal to bright light conditions. This is the reason why humans cannot distinguish color that well under low light conditions. The cones are the most important for determining the first trichromatic color representation.

Using a technique called *color matching* it is possible to investigate how the eyes and brain perceive color. Color matching basically is the mapping process of one single wavelength from the spectrum to a composite of multiple combined wavelengths. The brain is not able to see the difference when two colors match while their spectra can differ a lot. Experiments into this subject are performed by Wright [Wri28] and Guild [Gui32].

In their method they project two different light spots to a white surface. The first light spot, called the *target light spot*, is created by a monochromator that can emit light

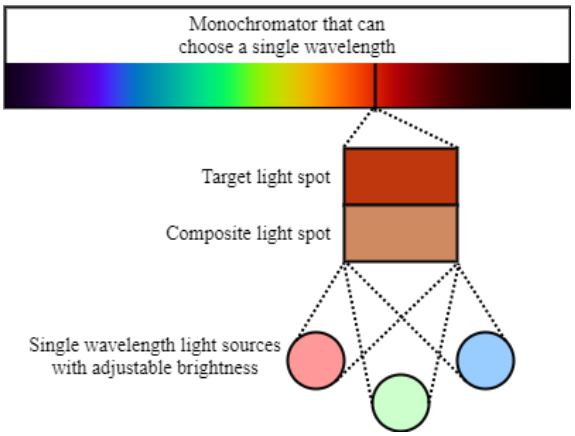

FIGURE 2.3: The experimental setup that Wright and Guild used in their experiments.

with a spectrum consisting of a single wavelength over the range of the visible spectrum. The second light spot is a composite of three adjustable monochromatic light sources, also called *primaries*, that are projected on the same spot. By adjusting the intensities of the primaries the test person is able to match the the composite light spot to the target light spot. The two light spots match when no difference can be perceived. This experiment was repeated with ten persons in Wright's work and with seven in Guilds work in order to get mean red, green and blue matching curves. A schematic overview of the experimental setup is given in Figure 2.3.

Wright [Wri28] used 460nm, 530nm and 650nm as wavelengths for the three primaries. This choice was based on the fact that their equipment could only produce those wavelengths. Later in the experiment of Guild [Gui32] the wavelengths 436nm, 546nm and 700nm were used as primaries because these were chosen as standard primaries by the National Physical Laboratory [Gui32].

The results of the experiments of Guild [Gui32] are shown in Figure 2.4. One thing that stands out is that the red curve also contains negative values, while light cannot be negative. The reason for the negative values is that they encountered single wavelengths that could not be matched to any combination of the primaries. In order to solve this problem they added the red primary to the target light spot instead of the composite light spot. By doing this, the target light spot is transformed and could be matched to the blue and green primaries of the composite light spot. The intensities of the red primary added to the target spot are considered negative values in the graph.

Negative light colors are not possible in the real world. Also doing calculations with negative values was not that convenient in the thirties where most calculations were done by hand [FBH97] [Sha03]. Therefore, the RGB matching curves of previous research are not directly suitable to use as the basis of a color space. The Commission Internationale de l'Eclairage (CIE) came up with a linear transformation that transforms the matching curves so they only contain positive values. The result of this linear transformation

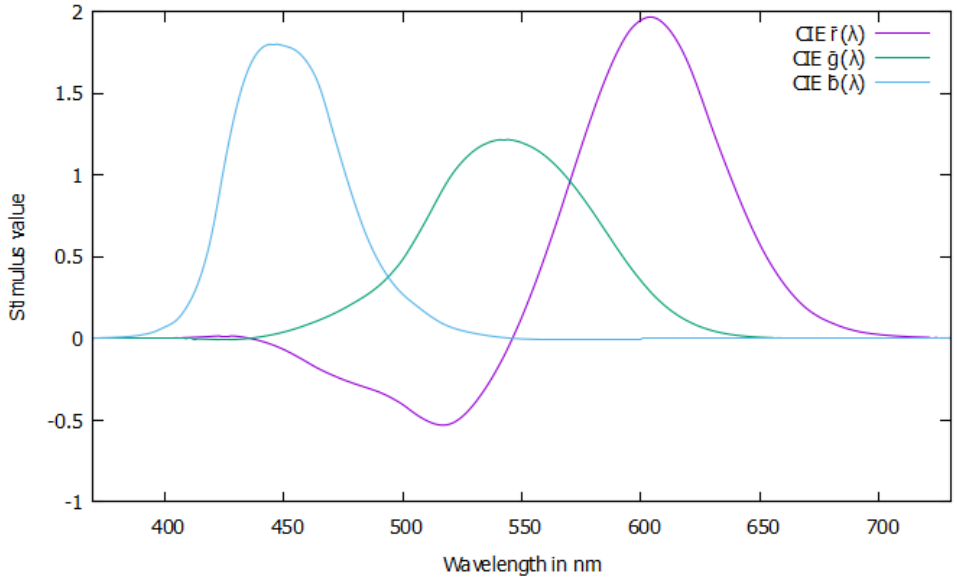

FIGURE 2.4: The normalized RGB matching curves derived from the Guild data.

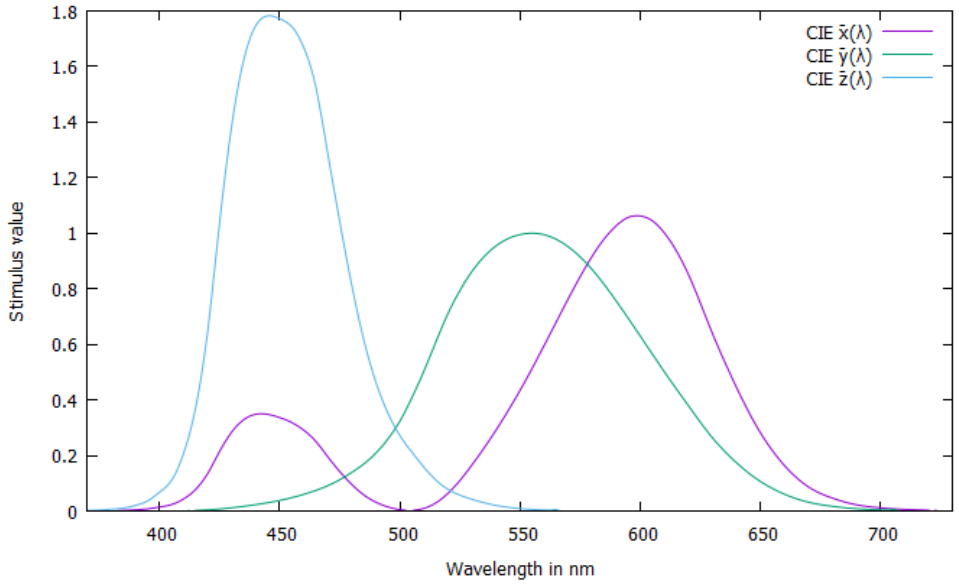

FIGURE 2.5: The CIE $\bar{x}\bar{y}\bar{z}$ matching curves standardized in 1931

became a new standard known as the CIE1931 XYZ color space of which the $\bar{x}(\lambda)$, $\bar{y}(\lambda)$ and $\bar{z}(\lambda)$ matching curves are displayed in Figure 2.5. We refer the reader to the work of Fairman et al. [FBH97] for a detailed explanation of the considerations and working steps of the linear transformation from RGB matching curves to the XYZ matching curves.

The XYZ color space can be made independent of brightness by considering the X, Y and Z tristimulus values as 3D coordinates and projecting them to the $X+Y+Z=1$ plane using normalization as shown in Figure 2.6. This normalized color space is denoted with the lowercase xyz. The Y value in the CIE XYZ color space is designed to hold

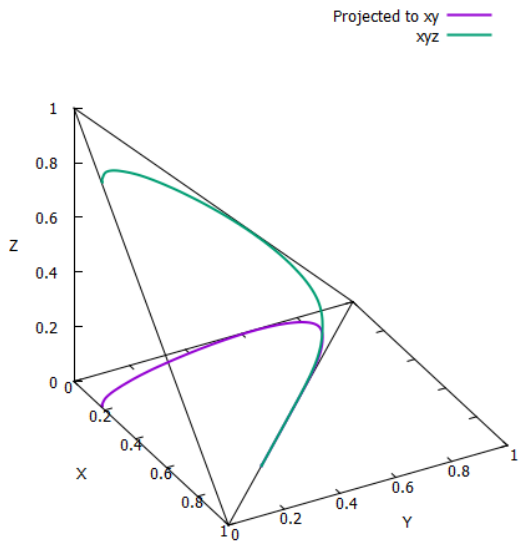

FIGURE 2.6: The $\bar{x}(\lambda)$, $\bar{y}(\lambda)$ and $\bar{z}(\lambda)$ matching curves projected to the $X + Y + Z = 1$ plane. The coordinates in the xy plane represent the chromaticity of the color independent from brightness.

the brightness information of a light source and with the relation $z = 1 - x - y$ the z value can be determined by just the x and y values. Using these two rules a color can be represented with xyY values which separates the chromaticity from the brightness of the light source.

The lines of the graph in Figure 2.6 are called the *spectral locus* and every single wavelength spectrum maps to a point on this border. All points inside the convex hull of the shape represent the visible color spectra as a mixture of the three hypothetical primaries. In order to get the XYZ coordinate of a spectrum, the $\bar{x}(\lambda)$, $\bar{y}(\lambda)$ and $\bar{z}(\lambda)$ matching function values are multiplied with the SPD intensities $S(\lambda)$ per wavelength and integrated over the visible part of the spectrum R_v as follows:

$$X = \int_{\lambda \in R_v} S(\lambda)\bar{x}(\lambda)d\lambda \tag{2.1}$$

$$Y = \int_{\lambda \in R_v} S(\lambda)\bar{y}(\lambda)d\lambda \tag{2.2}$$

$$Z = \int_{\lambda \in R_v} S(\lambda)\bar{z}(\lambda)d\lambda \tag{2.3}$$

Every visible spectrum can be mapped to a single tristimulus coordinate using these three functions and when two different spectra map to the same XYZ tristimulus value they are indistinguishable from each other for a human observer. This is called *metamerism*.

The standard is specifically designed to be a device independent and universal way to describe colors. A consequence of the linear transformation however is that the primaries have become hypothetical primaries which cannot be represented by real world light sources anymore. The XYZ color space is a way to talk about and compare colors but in order to visualize an XYZ color it first needs to be converted into a device dependent color space. XYZ color space represents the perception of color for an average person which are the physiological and psychological light properties and does not represent the physical properties of light. For rendering it is necessary to have a trichromatic color space that is based on real world primaries.

There are many trichromatic standards based on real world primaries like Adobe RGB, ProPhoto or Apple RGB but the current dominant standard is sRGB. It is widely used in game engines like Unity3D [Uni05] and Unreal engine [Unr98] and also in other applications like displaying images. The standard was proposed in 1996 by the companies Microsoft and Hewlett-Packard [SACM96] in order to have an efficient and universal way of displaying colors. Up until then every hardware manufacturer and software developer handled color management in their own way resulting in differing colors between devices and software programs.

The sRGB standard is based on the phosphor primaries of Cathode Ray Tube (CRT) monitors from the nineties which were already standardized in the Rec. 709 standard [Rec15b] from 1990. It uses the CIE D65 standard daylight illuminant as its reference illuminant. This means that when the sRGB triple (R,G,B) = (1, 1, 1) is converted to CIE XYZ space, it will be the same point as the point where the spectrum of D65 will be mapped to in CIE XYZ space. It defines what perfect white light is for a human observer. Furthermore, there are two types of sRGB, namely: linear and gamma corrected sRGB color space. The linear variant can be used for manipulation and rendering. A linear transformation matrix is used to convert from CIE XYZ space to linear sRGB space and vise versa. The gamma corrected sRGB variant can be used to display the end result on the screen. Gamma correction is necessary because screens do not always have a linear response curve.

Figure 2.7 shows the sRGB, Adobe RGB and Rec.2020 [Rec15a] color space expressed in a CIE xy chromaticity diagram. Another word that sometimes is used for color space is *gamut*. The primaries of the color space determine the corners of the triangle and every color inside the triangle can be expressed with a linear combination of these three primaries. It is important to note that visible colors outside of the triangle cannot be represented, or in other words, the color is out of gamut. This means that sRGB is a limited subset of the set of all visible colors. Adobe RGB is able to distinguish more spectra uniquely because it spans a larger area.

Render engines are optimized to work with RGB triplets. Calculating a color is nothing more than adding, subtracting or multiplying 3D vectors and can be done

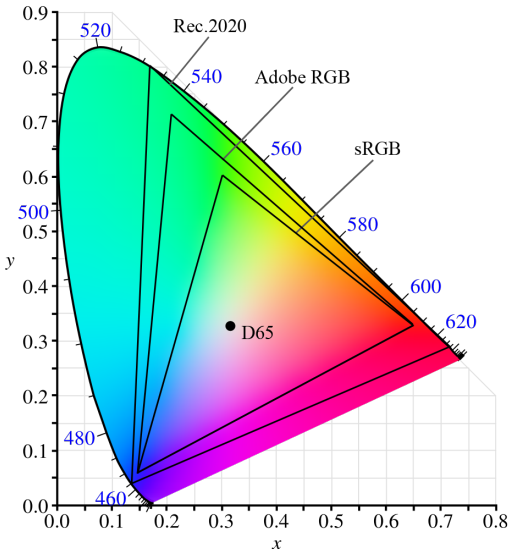

FIGURE 2.7: The CIE xy chromaticity diagram with sRGB, Adobe RGB and Rec2020 plotted inside.

efficiently on GPUs.

2.2 Binning

Binning yields a better representation of the physical color than trichromatic color representations do. Hall and Greenberg [HG83] show that when a trichromatic color space is used during the rendering process it leads to erroneous results as depicted in Figure 2.8. They rendered an image of two overlapping semi transparent objects that filter the light. A control image was made using a full spectrum representation with steps of 1nm in the range of 360 – 830nm. After that they rendered the same image again using CIE XYZ space, RGB space and a binning method using nine bins. There were large errors in the trichromatic color space renders while the binning method was pretty close to the full spectrum render. They state that the nine bin method only costs 2% more computation time compared to trichromatic renders.

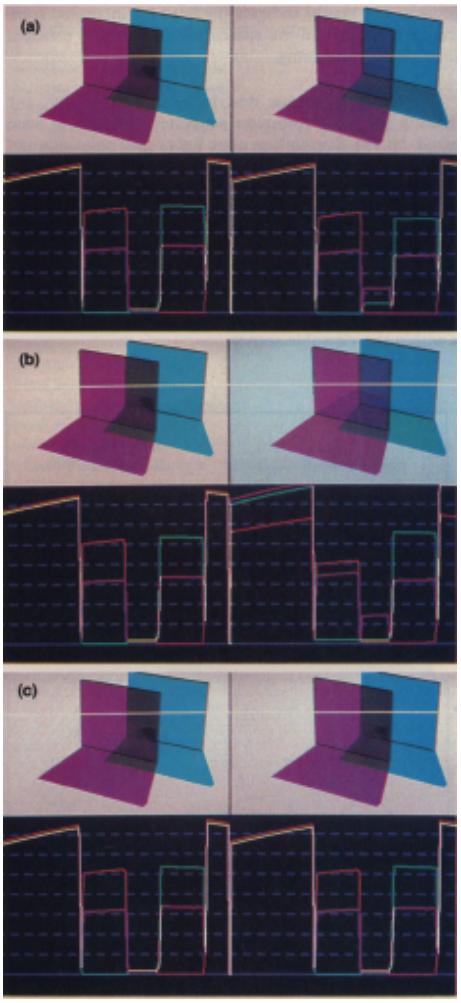

FIGURE 2.8: Taken from the work of Hall and Greenberg [HG83]. The left column shows the control image and the right column shows each individual render method, a) rendered in CIE XYZ space, b) rendered in the RGB color space of the monitor they use, c) rendered with nine bins. The graphs below each pair of images show the RGB intensities for each pixel in the highlighted horizontal pixel row in the images.

The main idea of binning is that a spectrum is divided into a fixed number of equally sized wavelength bins. For each bin the average value is calculated from the original spectrum. This needs to be done before the rendering process can start. Additionally every binned spectrum needs to have the same amount of bins. During the rendering process the SPD of a light source is multiplied with the reflectance curve of a material in a per bin fashion to calculate the reflected light energy. From an implementation perspective it does not require significant changes compared to trichromatic rendering. Arrays with a predefined number of bins need to be multiplied instead RGB triples. What can be a problem however is obtaining texture maps that have binned spectra as pixels.

Determining the optimal number of bins to correctly represent the whole spectrum is not that trivial. Hall and Greenberg [HG83] state that nine bins should be enough, Pharr et al. [PJH16] state that 30 bins is enough and Evans and McCool [EM99] state that 80 bins is enough. In the end, this depends on the goal of the rendered image. When an image should only be visually appealing, a low number of bins could be used. A large amount of bins would be preferable for color verification purposes like in applications such as architectural design. The final error will depend on the amount of bins in combination with the shape of the spectral curves that are used.

Some spectra have narrow spikes which needs small bin sizes to properly represent the whole spectrum. Other spectra have smooth curves which can be faithfully approximated by a small amount of broad bins. The smaller the bin size the more coefficients need to be stored in memory. In a scene with a large amount of varying spectra this can become a problem. When spiky spectral curves are used in combination with smooth curves, the smooth curves need to have the same amount of small bins like the spiky curves. The amount of bins is dependent on the spectrum with the largest amount of irregular values. In the end a trade-off needs to be made between on the one hand memory costs and on the other hand the accuracy of the end result. Figure 2.9 shows an example of a spiky spectrum that is represented with ten bins which does not capture the spectrum correctly.

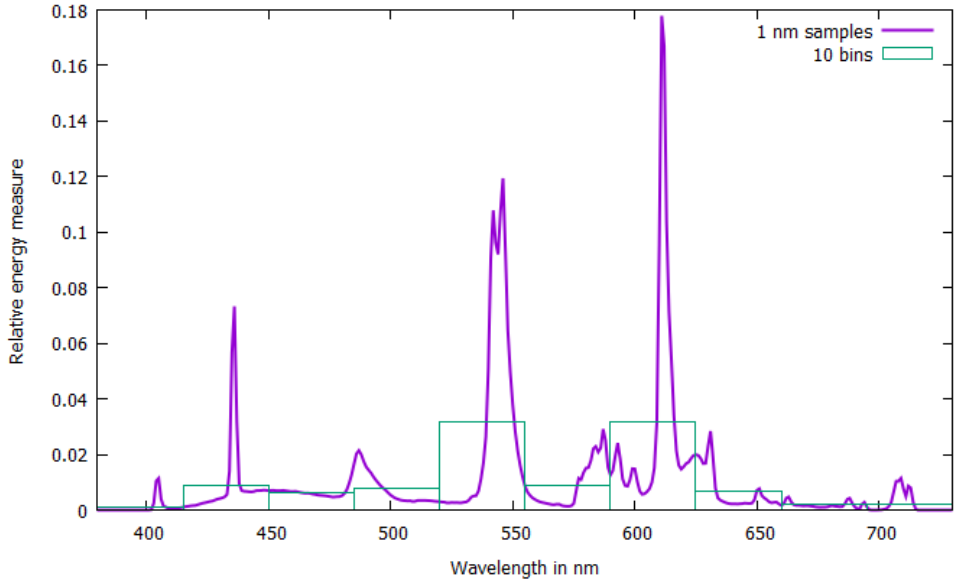

FIGURE 2.9: A spiky fluorescent spectrum binned in 10 bins.

2.3 Spectrum functions

Another way to represent spectra is with a set of functions. The reason why this could be beneficial is that the amount of coefficients that need to be stored could be a lot less while the spectrum is still approximated in a good way. Compared to binning this could work but the gain in memory sometimes comes at the price of performance. This is because the evaluation of the set of functions can be expensive. Moreover, to approximate irregular spectra, the amount of functions that are needed can still cost as much memory as the binning method or even more. On top of that it can be difficult to properly fit the functions to an irregular spectrum.

Raso and Fournier [RF91] try to approximate the spectrum with piece-wise polynomial functions. The spectrum is split into individual parts and to each part a polynomial function is fitted. They state that their approach could lead to discontinuities between the consecutive functions. Therefore, they assume that most of the spectra are smooth, making it a problem of less importance. In practice most of the curves are smooth and when irregular spectra need to be represented it can lead to errors.

Instead of using polynomial functions, linear functions can also work which is what Peercy [Pee93] has proposed. An advantage of this method is that the interaction of light and materials can be solved by a simple matrix multiplication. In order to do this, every spectrum need to be represented by the same amount of functions and the functions also need to be linearly separable. This means that the functions do not overlap each other.

Another spectrum representation that uses functions is developed by Sun et al. [Sun00] [SFDC01]. They state that previous methods were not good enough. The method of Peercy [Pee93] is for example hard to store because in addition to storing

the function coefficients, the functions themselves need to be stored too. Besides that, the cost for multiplying two spectra is $O(N^2)$, where N is the number of functions that are used to describe a spectrum. It does not scale well when there are many different spectra in the scene that are all represented with a high number of functions.

Sun et al. [SFDC01] propose a method that uses Fourier coefficients to represent a spectrum. This way the function does not need to be stored together with the coefficients. To represent spikes in the spectrum they store the location and the intensity of the spike separately. Multiplying two spectra can be done in linear time by linearly interpolating between consecutive samples of both spectra.

The method of representing a spectrum of Sun et al. [SFDC01] is later used by Dong [Don06] in his implementation. Dong observed that not all spectra are always available for every light or material in the scene. He also states that spectral rendering still is more computationally expensive than trichromatic rendering. So he proposes a method that renders some objects with full spectrum calculations and other objects use simple RGB color values. Light sources for which the spectra are not available can be approximated by their corresponding RGB representation. During the rendering process, a spectrum based light source will be converted into RGB format when the material hit by the light is RGB based. When the material is spectrum based and the light is represented with RGB, the RGB values of the light source are converted into a spectrum.

An advantage of this method is that it is flexible with respect to representations. Spectra and trichromatic vectors can be used interchangeably. A disadvantage is that the mix of the lesser quality perception based representation with the higher quality physics based spectrum representations leads to erroneous results. Colors in RGB space cannot be converted uniquely into a spectrum because of metamerism.

2.4 Monte Carlo Sampling

Besides multiplying complete spectra with one another, it is also possible to use Monte Carlo sampling. This gives accurate results but it also requires a lot of computations which makes it hard to run it in real-time. Sampling the spectrum is just another integral added to the full render equation in the case of path tracing. It is a good alternative to full spectrum multiplications as long as there exists an efficient way to sample from a spectrum. Effects like refraction or fluorescence can be better modeled with sampling because individual samples are independent from each other.

The process of Monte Carlo sampling does not solve the memory consumption problem directly because the used spectrum behind the sampling function can still be any representation discussed before. It is however possible to choose a spectrum representation with a small memory footprint. Therefore, Monte Carlo sampling allows for solving

the memory consumption problem separately from the rendering process. This is a great advantage of the method.

Different spectrum representations can be used interchangeably as long as a sampling method exists for each of them. The book of Pharr et al. [PJH16] describes how they use measured spectra and that not every measured spectrum has the same coefficient density. This is a problem when two spectra need to be multiplied at a per coefficient basis. However, because Monte Carlo sampling takes a single sample, linear interpolation between measured samples can be used. This allows for flexibility.

Evans and McCool [EM99] describe how to make the process of random sampling a spectrum more efficient. They encountered that when random spectrum sampling is naively added to a path tracer engine, the color variance increases a lot compared to a path tracer that uses trichromatic color representations. They state that the reason for this variance is that the amount of information transported along the path is much smaller with spectrum sampling than it is with trichromatic rendering. The computational costs for one path stays the same and while a single path carries less information, more paths are needed to get to the same end result with low variance.

Therefore, Evans and McCool [EM99] proposed a method that, instead of transporting one sample along the path, transports a cluster of samples. This way the amount of information transported per path is increased. The total available spectral range is divided into strata after which a random sample position is taken from each stratum. For one path, this cluster of sample positions is used to sample the spectra at each path bounce. They gave three strategies to cope with materials that refract the light at which point a cluster of wavelength samples needs to be split into multiple individual samples.

The first strategy is to pick a random sample position and throw the other samples of the cluster away. This is a simple solution but it throws away previous computations. The second strategy splits the samples of the cluster into individual paths. When refraction occurs, both the first and the second strategy reduce the cluster approach to the naive single sample approach. The first strategy wastes computations while the second adds extra computations to the problem.

The third strategy defers the calculation of the clustered samples to a later moment. First a single sample is taken and a complete path is calculated. Whenever the path has not encountered any refractive materials, other samples are taken according to the stratification scheme and processed for every encountered material. This last strategy does not waste any computations, but it needs to store the whole path which gives some overhead. It depends on the scene characteristics what strategy should be used.

Radziszewski et al. [RBA09] build upon the method of Evans and McCool [EM99]. They propose a sampling scheme that uses importance sampling to lower the variance of the end result. Instead of throwing away color samples upon encountering a refractive

surface, they continue with the complete cluster and scale the spectral samples individually by the probability dependent on glossy refraction characteristics. The new direction of the refracted ray is based on a randomly chosen sample from the cluster.

Continuing with the whole cluster after a refraction is not possible in the method of Evans and McCool [EM99] because they use a perfect refraction model. Radziszewski et al. [RBA09] use glossy Phong-based refraction which does make continuing possible. In a perfect refraction model, the probability of sampling a direction other than the perfect refraction direction would be zero. To also account for perfect refraction, they introduced multiple importance sampling. For glossy refraction this lowers the variance even more. However, when wavelength dependent effects of perfect refraction are modeled, the decrease of variance will be less apparent.

Wilkie et al. [WND+14] have made the multiple importance sampling part of the method of Radziszewski et al. [RBA09] more explicit, making it easier to implement it in conjunction with other variance reduction techniques. This is an improvement over the previous method. However, they also indicate that their method, like prior methods, only uses a single wavelength when perfect refraction must be modeled. Thus reducing the impact of multiple importance sampling.

Besides that, they handle generating the cluster and determining the wavelength that determines the paths refraction differently than Evans and McCool [EM99] and Radziszewski et al. [RBA09]. For this they randomly pick a wavelength sample, which they call the *hero wavelength*, and use this wavelength for the whole path. Radziszewski et al. [RBA09] picked a random wavelength from the cluster at every bounce. All refractions along the path are based on this hero wavelength.

To generate the cluster of other wavelengths they do not use random samples in different strata like Evans and McCool [EM99] did. Instead they place samples at fixed equal distances from the hero wavelength using a wrapping function to ensure that the samples are nicely distributed over the whole spectrum. This saves a few random number generations per sample.

Chapter 3

Color experiment

Color representations can be divided in two distinct groups. On the one hand there are representations that are based on the human perception of light and on the other hand there are representations which are based on the physics principles of light. Modern game engines use sRGB, which is a limited color space based on CRT monitor primaries and is mainly suitable for rendering and displaying visually pleasing images.

Our idea is to construct a custom color space based on the sensitivity curves of the camera to better approximate the light in a render engine that is only capable of processing trichromatic color vectors. This custom color space can be used to render a virtual world in the same way as sRGB color space is used in most render engines. We expect that the red component will capture additional infrared information of the incoming light.

The methodology we use is inspired by the experiments done by Wright [Wri28] and Guild [Gui32] which were specifically designed for human observers. We try to apply the same principles of color matching to camera sensors.

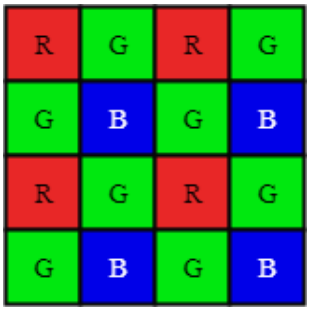

FIGURE 3.1: An RGBG (Red, Green, Blue, Green) Bayer pattern.

Regular color cameras have a Color Filter Array (CFA) which is an arrangement of light filters on top of the sensor pixels. A CFA is also called a *Bayer filter*. Figure 3.1 shows an example of an RGBG bayer pattern commonly used in color cameras. The sensitivity curve of the sensor in combination with the transmissivity curves of the filters

result in combined sensitivity curves that describe how sensitive the camera is to each wavelength.

For experimentation purposes we used the combined sensitivity curves from an arbitrary automotive RGB camera shown in Figure 3.2.

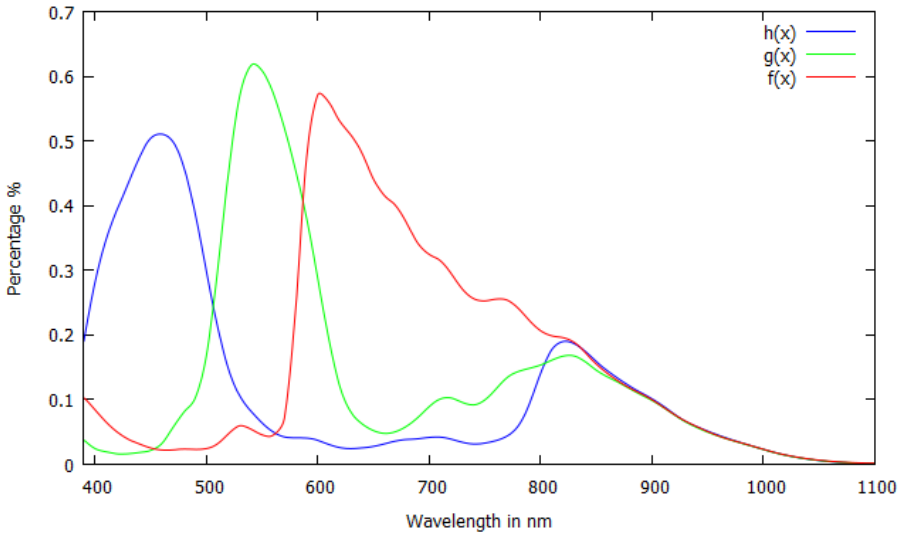

FIGURE 3.2: Sensitivity curves from an arbitrary camera.

3.1 Methodology

To create a tristimulus color space, we need three partially overlapping sensitivity curves that cover the red, green and blue parts of the spectrum which we denote by the functions $f(\lambda)$, $g(\lambda)$ and $h(\lambda)$. For every wavelength λ the functions return a percentage of how much photon energy can be converted by the camera sensor into an electric signal. Moreover, the measured sensitivity curves have to have a wavelength interval smaller than or equal to 5nm. Otherwise the curves are not detailed enough.

The next step is to choose three wavelengths λ_{prim1}, λ_{prim2} and λ_{prim3} as primaries for the new color space. The primaries must be chosen in such a way that they are able to span the full visible spectrum and that they are linear independent [Sha03]. This last condition means that one primary cannot be created by adding the two other primaries together in any way. Another condition that should be taken into account is that for the position of the primary one sensitivity curve is high and the other two are low. Taking the peak wavelength of one of the sensitivity curves usually is a good choice. However, when the other curves also have high values at the peak, another wavelength should be chosen.

Similar to the experiments of Wright [Wri28] and Guild [Gui32] we want to construct matching curves that map a single wavelength to a linear combination of the three chosen

primaries. With A, B and C being the variable coefficients that determine the intensity of the primaries, we get the following set of linear equations that needs to be satisfied:

$$A * f(\lambda_{prim1}) + B * f(\lambda_{prim2}) + C * f(\lambda_{prim3}) = f(\lambda) \tag{3.1}$$

$$A * g(\lambda_{prim1}) + B * g(\lambda_{prim2}) + C * g(\lambda_{prim3}) = g(\lambda) \tag{3.2}$$

$$A * h(\lambda_{prim1}) + B * h(\lambda_{prim2}) + C * h(\lambda_{prim3}) = h(\lambda) \tag{3.3}$$

Rewriting the set of linear equations to matrix form we get:

$$Kx = b \tag{3.4}$$

$$\begin{bmatrix} f(\lambda_{prim1}) & f(\lambda_{prim2}) & f(\lambda_{prim3}) \\ g(\lambda_{prim1}) & g(\lambda_{prim2}) & g(\lambda_{prim3}) \\ h(\lambda_{prim1}) & h(\lambda_{prim2}) & h(\lambda_{prim3}) \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} f(\lambda) \\ g(\lambda) \\ h(\lambda) \end{bmatrix} \tag{3.5}$$

The three chosen primaries are fixed so matrix K can be precalculated after which the inverse can be determined. For every wavelength we are now able to calculate the values A, B and C with:

$$x = K^{-1}b \tag{3.6}$$

This allows us to define the matching functions $\bar{a}(\lambda)$, $\bar{b}(\lambda)$ and $\bar{c}(\lambda)$ that return the A, B or C values respectively for any given wavelength. The $\bar{a}\bar{b}\bar{c}$ matching functions are similar to the CIE $\bar{r}\bar{g}\bar{b}$ matching functions. The CIE transformed the RGB matching curves into the device independent XYZ space with hypothetical primaries. If we were to do that, our new color space would not be usable for rendering anymore.

In order to get the different primary possibilities in the same scale we normalize the curves which is basically a projection to the a + b + c = 1 plane. This can be done with the following functions:

$$\bar{a}(\lambda) = \frac{\bar{a}(\lambda)}{\bar{a}(\lambda) + \bar{b}(\lambda) + \bar{c}(\lambda)} \tag{3.7}$$

$$\bar{b}(\lambda) = \frac{\bar{b}(\lambda)}{\bar{a}(\lambda) + \bar{b}(\lambda) + \bar{c}(\lambda)} \tag{3.8}$$

$$\bar{c}(\lambda) = \frac{\bar{c}(\lambda)}{\bar{a}(\lambda) + \bar{b}(\lambda) + \bar{c}(\lambda)} \tag{3.9}$$

Every spectral power distribution $S(\lambda)$ can now be represented with three coordinates using the matching functions:

$$A = \int_r S(\lambda)\bar{a}(\lambda)dx \tag{3.10}$$

$$B = \int_r S(\lambda)\bar{b}(\lambda)dx \tag{3.11}$$

$$C = \int_r S(\lambda)\bar{c}(\lambda)dx \tag{3.12}$$

To approximate the coordinates we can use the summation instead of the integral so then the coordinates can be calculated with:

$$A = \sum S(\lambda)\bar{a}(\lambda) \tag{3.13}$$

$$B = \sum S(\lambda)\bar{b}(\lambda) \tag{3.14}$$

$$C = \sum S(\lambda)\bar{c}(\lambda) \tag{3.15}$$

3.2 Results

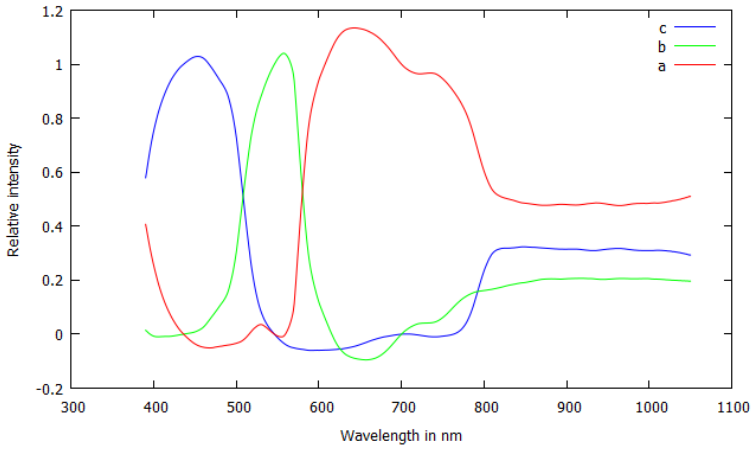

FIGURE 3.3: The relative normalized values of the ABC matching curves with 436nm, 546nm and 700nm being the single wavelength primaries. These primaries are the same as the primaries that Guild [Gui32] used in his experiment to retrieve the average human color matching curves. The spectrum range we used for this is from 390 to 1050nm.

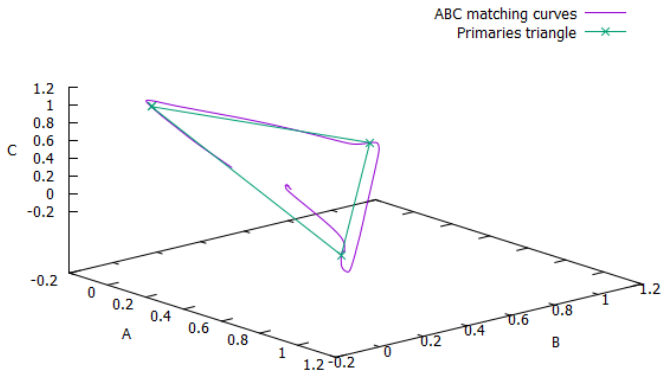

FIGURE 3.4: The relative normalized values shown in a 3D plot with primaries of 436nm, 546nm and 700nm for the wavelength range of 390 to 1050nm

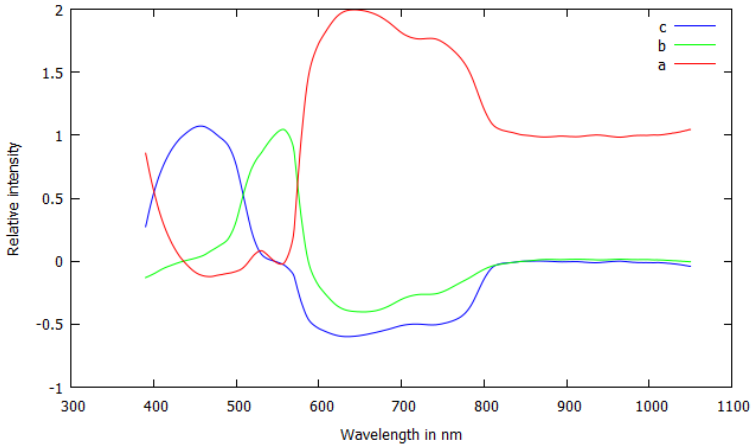

FIGURE 3.5: The relative normalized values with primaries of 436nm, 546nm and 850nm for the wavelength range of 390 to 1050nm

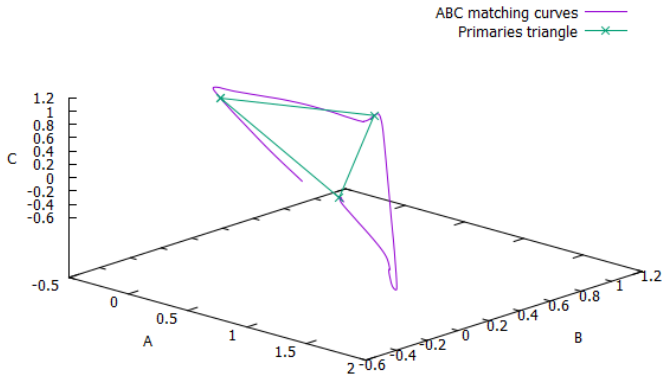

FIGURE 3.6: The relative normalized values shown in a 3D plot with primaries of 436nm, 546nm and 850nm for the wavelength range of 390 to 1050nm

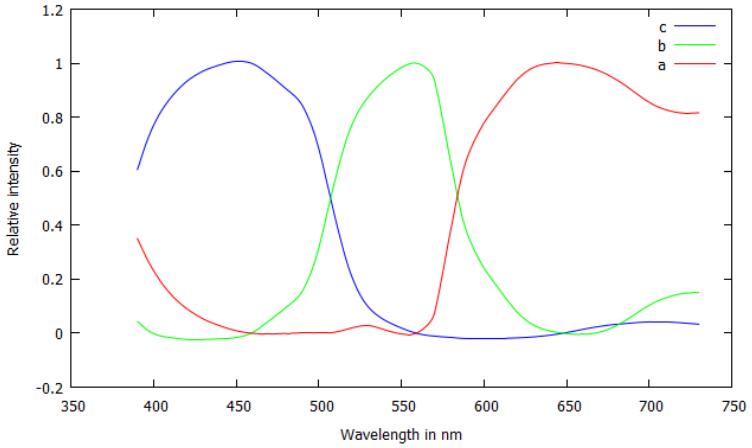

FIGURE 3.7: The relative normalized values with primaries of 459nm, 559nm and 648nm for the wavelength range of 390 to 730nm

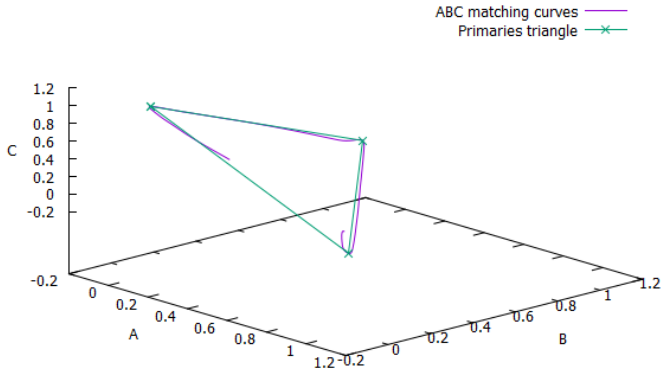

FIGURE 3.8: The relative normalized values shown in a 3D plot with primaries of 459nm, 559nm and 648nm for the wavelength range of 390 to 730nm

3.3 Discussion

We created a custom color space that is based on the sensitivity curves of an automotive camera. The results we include show the relative tristimulus values of three single wavelength primaries that are chosen by us.

Figure 3.3 and 3.4 show our first results for which we used the same primaries that were also used in the thirties by Guild [Gui32]. The graph of Figure 3.3 shows many negative values, which means that those trichromatic colors can not be created by physical light. This is also visible in Figure 3.4 in which the green triangle spans the color space that can be created with a linear combination of the three primaries. The values inside the purple line and outside of the triangle cannot be represented with positive values.

This is even more visible in our experiment that has a third primary with a wavelength of 850nm which is in the infrared part of the spectrum. By creating a custom color space we hoped to capture more infrared data in red component of an RGB representation of light, but from our results shown in Figure 3.5 and 3.6 we can see that with a primary in the infrared part of the spectrum there are more colors that cannot be represented because of negative values. The primaries can describe less colors compared to the previous experiment. The main cause of this is that the values of the three sensitivity curves that we used for our experiments (Figure 3.2) are lining up after 800nm which means that there is no discriminative information anymore.

In the set of results displayed in Figure 3.7 and 3.8 we show our attempt to find the optimal wavelengths for the primaries. There are as little negative values as possible. The primaries cover the range of ∼460nm to ∼650nm which is just a small part compared to the total range that the camera is capable of sensing. The method we proposed to approximate the raw signal of a camera does not give satisfactory results. Using only single wavelengths as primaries is too limited because it always requires negative light values to match the complete spectrum visible to the camera.

In practice, automotive RGB cameras have an additional infrared filter that cuts off the infrared part of the incoming light. The images would otherwise become blurred because after around 750nm the sensitivity curves start to line up and the same light intensity is registered for each pixel in the Bayer pattern. This is common in silicon based CMOS cameras. We first thought that our custom color space would give a better result because it represents extra information from the infrared part but since this part of the spectrum is cut off, the custom camera color space does not have that much of an advantage anymore.

Another point that we also learned later in the research is that cameras with RCCC color filter arrays are regularly used in the automotive industry too. This is in order to increase the sensor's sensitivity to light. The R stands for red and the C stands for

clear which means that there is no filter in front of the pixel. These cameras cannot be approximated with a trichromatic color space because there are just two sensitivity curves, namely: one for the red filter and one for absence of a filter.

One option to make the trichromatic rendering more accurate is to use a color space that has a larger gamut like Adobe RGB or the color space of Rec. 2020 [Rec15a]. More of the visible spectrum can be uniquely distinguished, because the gamut is larger. This is because the positive linear combination of the primaries covers a larger area in the CIE XYZ color space than sRGB does. After the rendering is done a linear transformation matrix can be used to convert the image with the large gamut into CIE XYZ color space. From there it can be converted into the native camera color space which can be obtained using least squares fitting of known reflectance patches. For example, Finlayson and Drew [FD97] discuss a method that uses a combination of constrained regression and least squares fitting to match the RGB values of the camera to the known XYZ values of the Macbeth color checker.

3.4 Conclusion

Using single wavelength primaries is not sufficient to approximate the raw response of an automotive RGB camera. For accurately approximating the response it is better to use a method that represents a full spectrum and not use a trichromatic color space. The problem of metamerism will always be present which means that different spectra will be mapped to the same 3D color vector. Especially spectra with spikes are not well represented in a trichromatic color space.

If however color errors are not that big of an issue or when the render engine is only limited to trichromatic representations one could use a color space with a large gamut. This will lower the small errors because there is more space to which spectra can be mapped uniquely. A color space based on the primaries of Rec. 2020 would be a good choice since it covers a large part of the chromaticity diagram. For color verification purposes it is best to use sampled spectra because this allows for accurate color calculations.

3.5 Future work

For learning purposes the virtual domain needs to be as close as possible to the real world domain. This means that light in the virtual environment and reflections of light on surfaces need to be physically realistic. However, for testing a prototype of a detection algorithm we do not think that color is that important. It can be argued that when a car is red or blue, it stays a car and it needs to be classified as such regardless of the color.

When traffic signs or traffic lights need to be classified, color might be more important because different colors can have different meanings. However, even for this, the relative position of the traffic light or the shape and size of the traffic signs can already tell a lot about the meaning. Future work must prove what the influence of color errors are on the different types of controlling algorithms that are used in the automotive industry. Especially whether it is important that the shade of color is an exact match or that small differences in color are allowed.

Another interesting topic is about whether specific parts of the spectrum are more important for computer vision algorithms than others. The red part of the spectrum might be for example more important to represent correctly than the green part of the spectrum.

Part II

Water drops

Chapter 4

Rendering methods

This section provides preliminary background information about the possible render methods that can be used for the simulation of an automotive camera. First we discuss rasterization which is suitable for Human-In-The-Loop and Hardware-In-the-Loop testing because of its fast render time characteristics. The results of rasterization based rendering can be visually pleasing but whether this is good enough for testing car systems or training neural networks remains to be seen. Rasterization lacks physics based light transport.

After rasterization we give some information about three types of ray tracing, namely: Whitted-style ray tracing, distributed ray tracing and path tracing. Ray tracing in general takes more time to output an image than rasterization but the results are closer to the real world because it supports global illumination. Ray tracing is because of its slower render characteristics a suitable method for Software-In-the-Loop testing purposes.

4.1 Rasterization

Rasterization is the process of mapping three dimensional polygons to a two dimensional raster of pixels [HDvM+14] and is part of the graphics pipeline used in games or other real time graphical simulations. A simplified overview of the graphics pipeline is shown in Figure 4.1.

Rasterization itself boils down to matrix vector multiplications that convert points from one space to the other and does not include the shading of the pixels. Shading is the process of adding texture colors and lighting to the pixels of the final render and is done in the fragment shader. An optional post processing step can be performed which is normally used for adding special effects like motion blur, bloom or color corrections.

A rasterization based graphics pipeline loops over all the polygons in the scene and projects them to the screen. The overlapping pixels are determined for every polygon projection and only the information of the closest polygon according to the z-buffer is

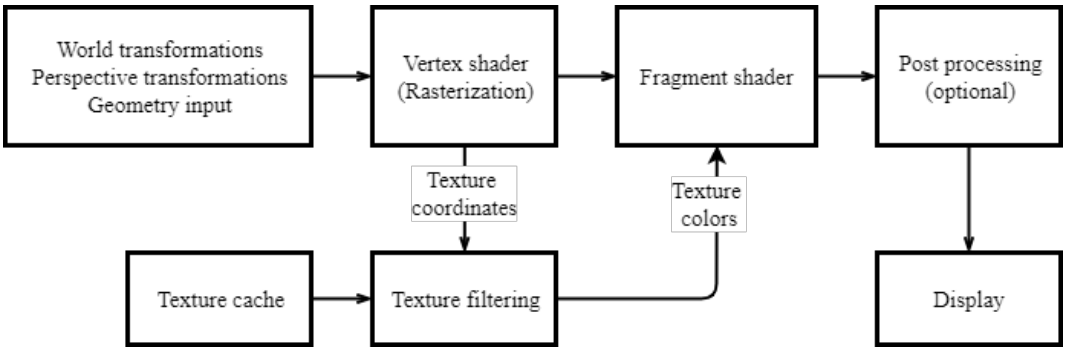

FIGURE 4.1: A rasterization based graphics pipeline.

stored in a buffer. Every polygon is basically being checked on whether it is visible from the viewer or not. After this, the polygon information buffer is used in the fragment shader in combination with the texture and lighting information of the scene.

An advantage of a rasterizer is that it is highly parallelizable because individual polygons can be processed independently from each other. This is why dedicated graphics hardware, which is specifically designed for this purpose, can be used to achieve real time applications. A limitation of rasterization is that global illumination can not be simulated well. Global information of other surfaces or occlusion of lights is not available when the polygons are handled one by one.

4.2 Ray tracing

Like rasterization, ray tracing can also be classified as a visibility algorithm. It is first introduced by Arthur Appel in 1968 [App68] who uses it for adding shading to line drawings of industrial designs in order to make them more readable. Unlike the natural behavior of light, which travels from a light source to the viewer, Appel traces rays from the viewer into the scene to determine the amount of light that is reflected by the hit surface from the light sources. This approach is much more efficient because tracing rays from a light source would yield wasted computations on light rays that never reach the viewer.

Later in 1980 Turner Whitted introduced an improved shading model that we now know as Whitted-style ray tracing [T.80]. This new model includes visibility checking of light sources like the model from Appel but also has diffuse shading, pure specular reflection and transmission of light through a medium. Algorithm 1 shows the pseudo code for a whitted-style ray tracer.

A ray is cast into the scene from the eye position through every pixel center and the nearest hit primitive is returned. These rays are called *primary rays*. Then a new ray is traced from the intersection point to every light source in the scene. When this ray does not encounter any other objects along its way, the light is added to the

Algorithm 1 Whitted-style ray tracer

```
1: function TRACE(Ray)
2:     for every pixel on the screen do
3:         Determine Ray_primary from eye position through pixel center
4:         Find closest hit in the scene
5:         if no hit then return background color
6:         if material type diffuse then
7:             return material color * illumination of visible light sources
8:         if material type is pure specular then
9:             Determine Ray_refl
10:            return material color * Trace(Ray_refl)
11:        if material type is transmissive then
12:            Determine Ray_refl
13:            Use Snell's law to determine Ray_refr
14:            Use Fresnel to determine the energy weights w_refl and w_refr
15:            return w_refl * Trace(Ray_refl) + w_refr * Trace(Ray_refr)
```

total contribution. This new ray checks whether the point is in the shadow or not and hence is called a *shadow ray*. Rays are defined as a line segment with an origin and a direction. To calculate the final color of a sampled pixel, the light contribution of every unoccluded light in the scene is summed and scaled by the reflectance color of the material description.

Shadow rays are only calculated for materials that are not completely specular. For a pure specular material the primary ray direction is reflected using the surface normal after which a new ray is constructed using the reflected direction and the intersection point as its origin. The algorithm goes into recursion with the reflected ray. This new ray is called a *secondary ray* or *extension ray*.

When a ray intersects a transmissive object there are two extension rays created. The light energy returned by the reflected and refracted rays are weighted according to the Fresnel equations. As these equations use a lot of cosines and square roots, Schlick's approximation [C.94] of the Fresnel equations is usually used because it is less computationally expensive. The direction of the refracted ray can be determined with Snell's law.

$$R(\theta) = R_0 + (1 - R_0)(1 - cos\theta)^5 \tag{4.1}$$

$$R_0 = (\frac{n_1 - n_2}{n_1 + n_2})^5 \tag{4.2}$$

$$\frac{sin\theta_1}{sin\theta_2} = \frac{n_2}{n_1} \tag{4.3}$$

Ray tracing is a point sampling process where the sampling is limited by the frequency of the pixel grid. Details are missed when the spatial frequency in the scene is higher than the sample frequency of the ray tracer. Small objects or objects that are far away will not be visible and edges become jagged due to this aliasing effect as can be seen in Figure 4.2.

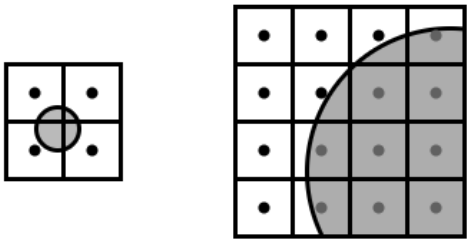

FIGURE 4.2: Left: missed detail when object is between sampling points. Right: aliasing when sampling frequency is to low.

Another consequence of sampling points is that the renderer is only able to simulate hard shadows. This is because light sources are represented by dimensionless points and a sampled point, for which a shadow ray is traced, can either be inside or outside the shadow. Soft shadows can only be modeled by an area light that is partially occluded. It is however possible to approximate soft shadows by having many point lights but this is not practical and does not give high quality results when a small amount of point lights are used.

4.3 Distributed ray tracing

To overcome the limitations of point sampling in the Whitted-style ray tracer, Cook et al. introduced distributed ray tracing [CPC84]. Instead of casting one ray to collect a single sample, multiple rays are cast over an interval to collect multiple samples. The results of the samples are summed and averaged by the amount of samples that are taken. Using this method it is possible to model soft phenomenons like soft shadows, anti-aliasing of edges, depth of field, glossy reflections or even motion blur.

When for example the visibility of an area light needs to be evaluated for a point on an object, the visibility function needs to be checked for every infinitesimally small point on the surface of the light source. But because other objects occluding the light do not necessarily have simple shapes, the task of evaluating the integral analytically becomes too complex. However, instead of evaluating the integral analytically, the integral of the function can be approximated by using Monte Carlo integration.

Monte Carlo integration is the process of evaluating a continuous function with random samples over a given interval after which the results are summed and divided

by the number of samples that are taken as shown in equation 4.4. When the number of samples goes to infinity, the outcome of this procedure will converge to the expected value of the function due to the law of large numbers.

$$E(x) = \int_{\alpha}^{\beta} f(x)dx \approx \frac{\beta - \alpha}{N} \sum_{i=1}^{N} f(\bar{x}) \tag{4.4}$$

As long as there is a way to evaluate the function for one sample, Monte Carlo integration is easy to implement.

4.4 Path tracing

All render methods discussed so far have in common that they all try to model how light is transported in a virtual scene. It was in 1986 that Kajiya introduced the rendering equation [Kaj86] in which all these render methods could be expressed. The notion of how light transport works was not new but up until then this was not formalized in the context of computer graphics yet. A common way to write Kajiya's render equation is as follows:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i, \omega_o) L_i(x, \omega_i) \cos\theta_i d\omega_i \tag{4.5}$$

In this equation is $L_o(x, \omega_o)$ the total spectral radiance that leaves point x along the outgoing direction ω_o. $L_e(x, \omega_o)$ Describes the spectral radiance that is emitted by point x along ω_o. This is the direct light. The integral over the hemisphere Ω returns the indirect light. Every material reflects light in its own way which is captured with $f_r(x, \omega_i, \omega_o)$. This is also called the *Bidirectional Reflectance Distribution Function* (BRDF) and describes how much energy is reflected. $L_i(x, \omega_i)$ is the term that describes how much radiance goes towards x along the incoming direction ω_i. Multiplying with $\cos\theta_i$ gives the irradiance where θ_i is the angle between the incoming direction ω_i and the surface normal at x.

Another way to write the rendering equation is to integrate over all surfaces in the scene instead of integrating over the hemisphere:

$$L(s \leftarrow x) = L_E(s \leftarrow x) + \int_{M} f_r(s \leftarrow x \leftarrow x') L(x \leftarrow x') G(x \leftrightarrow x') dM(x') \tag{4.6}$$

In this equation $L(s \leftarrow x)$ is the amount of light that is transported from point x to point s. $L_E(s \leftarrow x)$ gives the amount of light energy that goes directly from point x on an emissive surface to point s. Then the integral over the union of all surfaces M gives the light intensity that is reflected at point x from all surfaces toward point s. With $f_r(s \leftarrow x \leftarrow x')$ being the BRDF and $L(x \leftarrow x)$ being the amount of indirect light

energy that travels from point x' to point x. $G(x \leftrightarrow x')$ is the geometry term which describes how the light fall off behaves and whether point x and x' are mutually visible. It will be 0 if x' is occluded and $\frac{\cos\theta_i}{r^2}$ when it is visible. The $\cos\theta_i$ accounts for the conversion from radiance to irradiance the same way as in equation 4.5.

Expressing rasterization in terms of the render equation gives an equation that is not recursive because indirect lighting is not taken into account. Also the geometry term does not account for visibility anymore and the integral has become a sum of the energy of all point light sources in the scene:

$$L(s \leftarrow x) = L_E(s \leftarrow x) + \sum_l f_r(s \leftarrow x \leftarrow x')L(x \leftarrow x')G(x \leftrightarrow x') \tag{4.7}$$

Ray tracing basically has the same version of the render equation as rasterization with the exception of the way the geometry term works. A basic rasterizer has no global information about occlusion so it cannot directly determine whether a light is occluded or not. Light occlusion can however be implemented in a rasterizer but this would need additional work. With ray tracing this global information about occlusion is available so visibility can be checked and the geometry term can return zero for occluded points directly.

Rasterization and ray tracing do not including the indirect lighting term which results in an underestimation of the actual light energy that should be perceived by the viewer. This problem is solved when Kajiya [Kaj86] introduced path tracing which approximates the full render equation.

The idea of path tracing is to use a Monte Carlo Markov chain and converge to the expected value of the integral function after taking many random samples. The following slightly modified pseudo code from Kajiya [Kaj86] shows his method:

Algorithm 2 Basic Path Tracer

```
1: for every pixel on the screen do
2:     Determine primaryRay and find closest hit point x in the scene
3:     for the length of a Markov path do do
4:         Select the point x' and calculate the geometrical factor G(x ↔ x')
5:         Calculate the reflectance function f_r(s ← x ← x')
6:         Multiply the reflectance by L(x ← x')
7:         Add this contribution to the pixel intensity
```

Whenever a light is encountered, the light intensity is propagated along the path from the light source to the viewer and part of the light is absorbed at every bounce according to the material characteristics of the surfaces. The path might be very long before it encounters a light source or it may even not encounter a light at all. In the former case the total light contribution of the path is small and in the latter case there is no light contribution. Both cases add to the sampling noise which will reduce over

time, but this could possibly take many random paths before the noise level is below an acceptable threshold.

There are however smart ways to lower the amount of sampling noise like (multiple) importance sampling, next event estimation or stratified sampling. Also variations of path tracing like bidirectional path tracing [LW93] or metropolis light transport [VG97] will decrease the convergence time a lot in certain cases. These methods are however not necessary to understand the remainder of this thesis.

Chapter 5

Previous work

The automotive industry requires simulations to be as close as possible to the real world. With that in mind we evaluate to what extend the previous work is capable of modeling the different effects that may appear with water drops in front of a camera. This chapter provides an overview of the previous work with respect to water drop simulation and will describe how the different render techniques that we discussed in the previous chapter are used for this purpose.

First the works that specifically model water drops on the windscreen for automotive simulations will be discussed, which is directly connected to our second sub research question. After this we discuss the methods that use direct mapping to simulate water drops. These methods are applicable to more generic simulations where drops run down vertical glass planes. Lastly, we discuss what geometric drop models are used in the past and how realistic they are.

5.1 Windscreen simulation

Kaneda et al. [KKY93] are the first to propose animation of water droplets on the glass specifically for the application of driving simulations. They model how gravity acts on drops at an incline, how separate drops merge into one larger drop and also how the shape of drops can be approximated with a sphere. Their rendering process is based on a ray tracer implementation in combination with an environment map as depicted in Figure 5.1.

Their method can be split into two steps: The first step updates the locations of every drop. Every drop is placed in a cell of a grid and has a mass and velocity. In the update step they move the drops to other grid cells, but only when the next cell is empty or when the mass of the drop exceeds a threshold value. Drops with a small mass are stationary. These are the drops that are left by large water flows according to the material's affinity for water. Drops will only run in a downslope direction but can meander.

The second step of their method renders the drops to the screen. First they render the environment to a cube map. The forward face of this cube map has a higher resolution than the other faces because, as they state, more rays will hit that face. After that they cast rays from the viewer to the virtual screen and refract them using the normals on the glass plane and the drop surfaces. They do not trace both the reflection and the refraction ray but instead they only trace the ray that will return the largest light intensity of the two. The color which is sampled is determined by intersecting the ray with the cube that is surrounding the viewer.

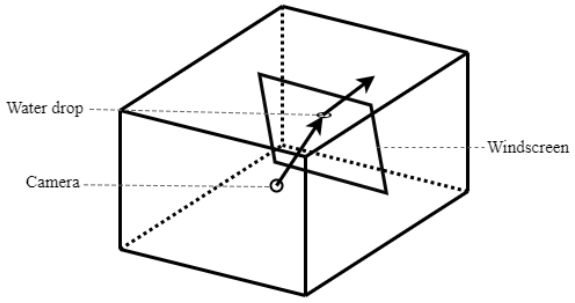

FIGURE 5.1: A cube map around the camera and windscreen setup.

Later, Kaneda et al. [KZY96] extend the method of Kaneda et al. [KKY93] to allow drops to run down from curved surfaces based on Bézier functions. They also include a second option based on metaballs for representing water drops which increases realism. This is instead of the capped sphere model which they previously used. Kaneda et al. [KIY99] further extend the method by adding interaction of objects with the water drops like the wind wipers of a car and also defocused the drops with a uniform weighted average filter.

The method described by Sato et al. [SDY03] also uses the same drop location update scheme as Kaneda et al. [KKY93], however, the rendering method is different. Instead of tracing rays through the drops they use a mapping technique supported by graphics hardware. They achieved real time results on systems from that time which was not the case for Kaneda et al. [KKY93], [KZY96], [KIY99].

First they partition the windscreen in equal grid cells and place a hemisphere representing a static water drop at every corner of the grid. Then they render an environment texture to the surface of every hemisphere using the outer normals for the refraction calculation. The environment background texture is a photograph from inside a car that drives on the road. After this pre-process step the actual rendering of dynamic drops can begin. In order to do this they place a rectangular polygon at the position of every water drop and determine in which grid cell the drop resides. By using the four pre-rendered static drops on the corners of the grid cells they are able to create a composite texture on the rectangular polygon and render this to a separate drops texture. Lastly, they

apply a simple blurring filter to the texture based on weighted 3x3 convolution kernels. By applying two different strengths of blur to the texture and choosing between the two depending on the distance to the viewer they achieve depth dependent blurring.

Another paper that describes how drops can be simulated in the context of automotive applications is from Halimeh and Roser [HR09]. They give a model based on the photometric properties of water drops on a glass plane and give a method that is able to predict whether a water drop is located at a given part of a still image. They call this method *Raindrop Intelligent Geometric Scanner and Environment Constructor* (RIGSEC). For this to work, the method predicts which sample points have to be taken from the background in order to construct a virtual spherical drop on the windscreen. This constructed virtual drop is matched with the real drop based on intensity correlation which is a measure of how likely the selected image region is indeed a drop.

In an experimental setup they test their method in two different environments with a camera behind a tilted glass plane that has water drops on it. An environment is in their case a flat still image of a traffic situation or a checker pattern located in front of the camera. The intensity correlation of the drops in the traffic environment was relatively high. The correlation however of simulated drops with real drops in front of the checker pattern was not that high. They state that this is because they did not model the blurring effect which is usually the case for drops that are close to the camera lens.

Roser and Geiger [RG09] improve the proposed model of Halimeh en Roser [HR09] by adding the missing blur filter which they also made depth dependent. The diameter of the blurring disc is determined with the following equation:

$$\epsilon = \frac{\Delta g f^2}{O(g - \Delta g)(g - f)} \tag{5.1}$$

where

$$O = \frac{f}{D} \tag{5.2}$$

This is based on the geometric relations between the ratios between $\epsilon : D$ and $\Delta b : b + \Delta b$ shown in Figure 5.2. The out-of-focus point is imaged as a disc on the camera sensor which is called the *circle of confusion*. Additionally, they give an application of the RIGSEC method, namely: removing water drops from a pre-recorded video which was captured through the windscreen of a traveling car.

Nakata et al. [NK12] specifically focus on the effects of hydrophobic surfaces on water drops and give a method to simulate how drops move over the windscreen under the influence of gravity, wind drag, contact angle hysteresis drag and viscous dissipation. Furthermore, they assume that drops on surfaces that are treated with a hydrophobic

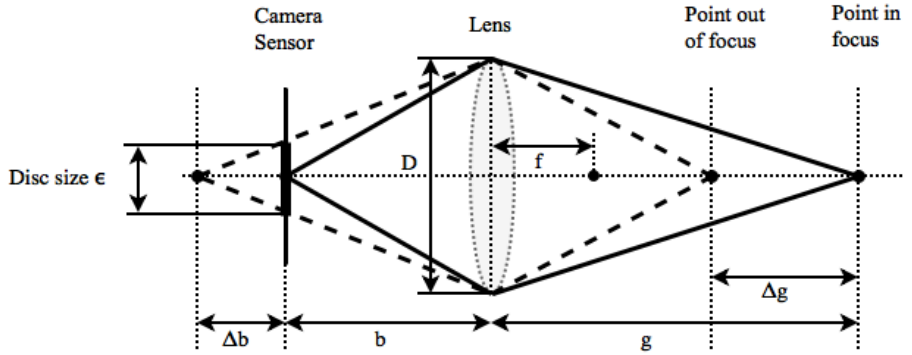

FIGURE 5.2: A schematic overview of the geometric properties of a defocused point. ϵ is the disc size, D is the aperture size, f is the focal length, b is the point of an image point which is in focus, Δb is the distance between the focused image point and the defocused image point, g is the distance from the lens to the object point and lastly Δg is the distance between a focused object point and a defocused object point.

coating can be modeled with hemispherical shapes because they state that the contact angle of those drops normally is between 90 and 100 degrees.

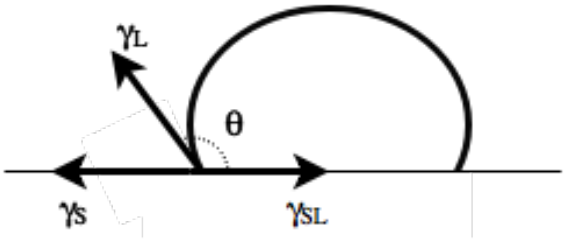

FIGURE 5.3: The contact angle of the drop surface and the underlying surface

The contact angle θ of a drop, as shown in Figure 5.3, is the angle between the surface of the drop and the underlying surface on the point where air, water and the underlying surface touch. The Young equation 5.3 states that this angle is dependent on the surface tension of the water drop γ_L, the surface tension of the underlying surface γ_S and the tension between the drop and the underlying surface γ_{SL}.

$$\gamma_L \cos\theta = \gamma_S - \gamma_{SL} \tag{5.3}$$

They distinguish three types of drops, namely: small stationary drops, large stationary drops and large moving drops. The first type is modeled with a single normal map that is placed on the windscreen. The second type is modeled with flat disc meshes that have modified normals as though the disc is a hemisphere. The last drop type is similar to the second type but has an elongated appearance. Its shape is not a disc but more like two connected discs that make up a flat capsule shape. All three types

of drops are rendered in the same way with a refraction shader that uses the modified normals to map the background to the drop surfaces.

Their simulation first places drops on the windscreen. Then in each time step, the locations of the dynamic drops are updates according to the forces acting on them. After that they check for collisions and merge drops that collided with each other. To simulate the clear trails that the large drops leave behind, they project the large drops to an alpha map that is used to remove small droplets from the single normal map. The final step is rendering it to the screen using a refraction shader. All of this is implemented in the Unity3D game engine and works in real time.

5.2 Direct mapping methods

There are also methods that do not specifically apply the simulation of water drops on glass to driving simulations. The papers we will now discuss give more generic applications of drops on vertical glass surfaces.

Takenaka et al. [TMT08] use a texture mapping technique that maps a background texture to a drop quad which is readily available in OpenGL [DSWND04]. A moving drop is represented with multiple discs of different sizes that are partly overlapping. Their way of rendering is similar to the method of Nakata et al. [NK12] however they make a specific assumption: the same render of a single drop can be used for drops that are close to each other because of their similarity. They divide the screen up in five regions, four corners and one center region, in which the same billboard render is used for every drop in that region. This will speed up the render time per frame a lot but it decreases the realism of the simulation.

Another research that uses direct mapping is the work from Stuppacher and Supan [SS07]. They state that they do not want to create a physically correct simulation but are only interested in a visually pleasing simulation that runs in real time. To achieve this they use a heightmap texture to represent the water drops. A heightmap gives a grayscale value per pixel that describes what the height of the water is at that coordinate. Every timestep the heightmap is updated to let the water move across the texture in the direction of the gravity vector. After updating the heightmap, a weak blur filter is applied to make the heightmap a bit smoother and from this a normal map is created. Rendering is done with an OpenGL refraction shader and an environment map.

The works from Chen et al. [CCW12] [CCW13] are similar to that from Stuppacher and Supan [SS07]. They also create a normal map from a heightmap and use environmental mapping to render it to the screen. The difference however is that they give more attention to how residual drops form behind a flow of water. Residual drops are formed with a given probability which can be set by a user. A new particle will be split

off and be left behind. By applying an erosion operation to the heightmap, which is a basic image processing technique, the connected flow is broken up into separate drops.

In the work of Chen et al. from 2013 [CCW13] they also give a clear explanation of how their method merges two adjacent drops. This is done using an ID map that states for every pixel to what drop particle it belongs. Whenever two drops collide, one or more pixels in the ID map will have two IDs in the update step which is a sign that a merge needs to be performed. During the merge, the old particles are deleted and a new particle is created with a new mass and velocity based on the deleted drop particles.

5.3 Drop models

The previously discussed works all use a hemispherical shape to represent the water drops. Real drops are not perfectly symmetric or round. We will briefly discuss a few of the ways to represent drops with other shapes.

Yu et al. [YJJC98] create drops using parametric metaballs with additional control points. A metaball is defined as a set of one or more implicit functions that describe how the shape is formed in 3D. An implicit function returns whether a point in 3D is inside, on the border or outside the metaball object. By adding three components, namely: gravity, the angle of the slope of the surface on which the drop rests and the friction of the surface, they are able to adjust the standard parametric function of hemispherical shapes to create more realistic looking water drops.

Although the previous method gives credible looking drops, it is not physics based. The method of Fournier et al. [FHP98] looks more into the physics behind the shape. They model a drop with a mass spring system and use four constraints that need to be satisfied for a stable simulation. The first constraint enforces the drops to have a constant volume. The second constraint makes sure that the surface contact of the drop with the air is as small as possible. The third constraint is about making the contact between the drop and the surface on which the drop is resided as large as possible. The last constraint states that the drop can be deformed by other forces acting upon the drop. Their simulation is able to move drops with physically plausible shapes across a triangle mesh but has difficulties with modeling the merging and splitting process.

Wang et al. [WMT05] also describe a physics based method but is a lot more advanced. It accurately models fluid dynamics including external forces, viscous momentum diffusion and velocity advection. All of this is limited to a finite 3D grid in which the water is able to flow from cell to cell and where the amount of flow is stored at the cell faces. Their method specifically looks into how contact angles of water drops can be modeled and also how water drops curvatures change when the surface hydrophobicity is changed. After a simulation is ran, they use the surface information of the drops to construct a triangle mesh for each drop.

Lastly, the work from Roser et al. [RKG10] models drops with Bézier curves. First they take side view images of water drops with varying sizes on a glass plate with different incline angles. Then they use RANdom SAmple Consensus (RANSAC) line fitting to find the curved lines that best describe the shape of the water drop. This is a process which randomly selects a possible fit based on a smaller portion of the drop pixels. Then RANSAC looks whether the rest of the pixels agree with the fit. After finding the fit they use least squares fitting to find the parameters of the Bézier curves which results in the final model of the water drop.

Using two orthogonal Bézier curves is a good way to represent single stationary drops, but it is too limited to model moving or multiple interacting drops. However, this was not the focus of their research. They simply want to show that using something other than a hemisphere shape gives a better fit when a single water drop on a tilted surface with a relatively large volume needs to be simulated.

Chapter 6

Raindrops on windscreen simulation

From the previous work we have seen that there are two main techniques to render water drops, namely: ray tracing every light bounce inside the drop and directly mapping an incoming direction to a background texture coordinate based on the surface normal. What we missed however is an explanation of the importance of reflections inside the drop and what the consequences are when it is not modeled properly.

In this chapter we show that total internal reflection contributes significantly to the appearance of water drops close to a camera. Both the drop curvature and the inclination angle of the glass influence the image inside the drop. Ray tracer based methods are perfectly capable of simulating these reflections however direct mapping techniques cannot model this. Realism is a high priority when simulating automotive cameras. Therefore it is important to know what the limitations are of the different methods.

First we will explain our methodology. After that we will provide observations from the real world that will show what effects are necessary to realistically simulate water drops. Then we provide our experiment results and also compare the two render techniques with each other.

6.1 Methodology

This section describes the various implementation decisions we made during our research. It describes how we set up the scene and how the rendering process of the ray tracing model and direct mapping model are implemented.

6.1.1 Scene

In our implementation one world unit represents one meter. Normally, the origin of a reflected ray is set to have a small offset from the original intersection point in the direction of reflection. This is known as the epsilon offset in a ray tracer and we choose it to be 0.00001 meter. With this value we achieve results without large visible artifacts. It has however the following consequences:

Firstly, it means that errors can be expected when the thickness of geometry in the scene is less than 10 micrometer. When the refracted ray is positioned at the epsilon offset it will be placed on the other side of the object instead of inside the object. This is also the case when rays are almost parallel to the surface of the geometry and hit a corner. Figure 6.1 a and b show examples of this error.

Secondly, primitives in the far distance can have the problem of self occlusion when the epsilon offset is too small. The artifacts occur when the origin of the shadow ray is placed on the wrong side of the primitive due to floating point precision error. The shadow ray encounters the same primitive again and therefore returns no light intensity. Figure 6.1 c shows the intuition behind this error.

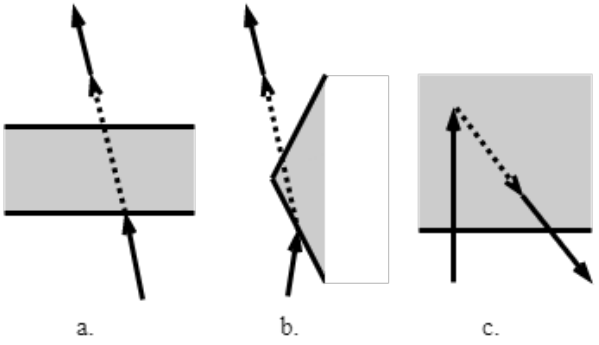

FIGURE 6.1: Examples of cases where errors can occur when an incorrect epsilon offset value is chosen. a) epsilon offset is larger than the thickness of the object. b) epsilon offset is larger than the thickness of the corner. c) the hit position is in the object due to imprecision. The epsilon offset is not large enough to compensate for this.

The tests we perform with the ray tracer consist of scenes with only water drops. This way only the first consequence is relevant for us while we can ignore the second consequence. Otherwise we would need to implement an adaptive epsilon offset because the relative scales in our scene are extremely different. Another advantage is that it also saves implementation time. We do not need to create a full path tracer implementation with correct light transport and construct a scene with geometric objects and widely varying realistic materials.

We use a 360 degrees environment map texture instead to simulate the virtual world. This can be either a photograph or a render from a third party render engine. Simulating the virtual world is now reduced to a simple and fast texture lookup. Therefore, our

render times are not dependent on the scene but only on the way the water drops are rendered which allows us to get our test results faster. Our implementation becomes a post process effect and can be added to arbitrary environment maps.

FIGURE 6.2: Example of an equirectangular environment map. [hei09]

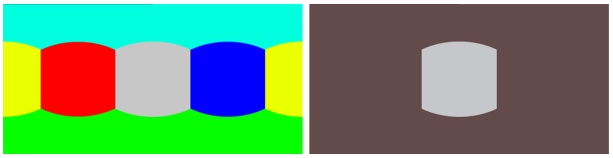

FIGURE 6.3: Two equirectangular environment maps that we used.

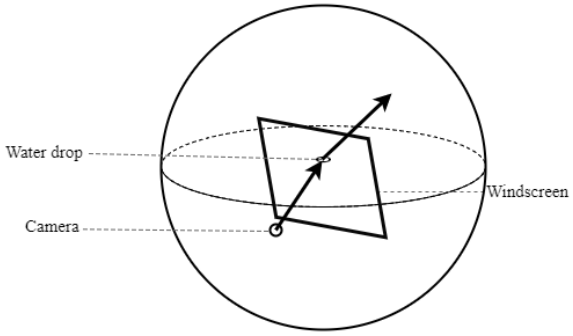

FIGURE 6.4: Environment map centered around the origin position of the ray leaving the water drop.

The idea behind environment mapping is based on the assumption that everything lies at infinity. This way we can assume that the environment map is centered at the origin of a sampling ray, like in Figure 6.4, and only the direction of the ray is used to sample the texture. In our implementation we use two equirectangular projection images with a 2:1 ratio shown in Figure 6.3. Figure 6.2shows an example of an equirectangular photographed image.

To convert from a normalized xyz direction in polar coordinate space to a uv texture coordinate with u and v in the range of [0,1] we use the following functions taken from the work of Reinhard et al. [RHD+10]:

$$u = \frac{1 + (atan2(direction.x, direction.z)/\pi))}{2} \tag{6.1}$$

$$v = acos(direction.y)/\pi \tag{6.2}$$

One way to get a texture sample is with nearest neighbor sampling. The uv texture coordinates are multiplied with the texture width and height respectively which results in the xy coordinate of the sample. Although the method is simple and fast, it results in visible pixels when a water drop magnifies an environment map with a relatively low resolution. So instead of nearest neighbor sampling we use bilinear sampling. This sampling method takes four pixel samples around the uv texture coordinate and uses linear interpolation to get a weighted result. It decreases the minimum necessary resolution of the environment map which in turn lowers memory costs.

We choose to ignore the refraction of light due to the glass of the windscreen because this saves a great amount of calculations. Whenever a ray is trapped inside the glass of the windscreen as shown in Figure 6.5, it would require too many bounces, if not close to infinite bounces, in order to get a sample result. It is also not known how the light behaves inside a windscreen because most windscreens are made of two glass layers with a plastic layer in between. We are aware that without the simulation of glass the setup does not model the behavior of light correctly for this specific case, however this does not affect the correctness of reflection and refraction inside the water drops.

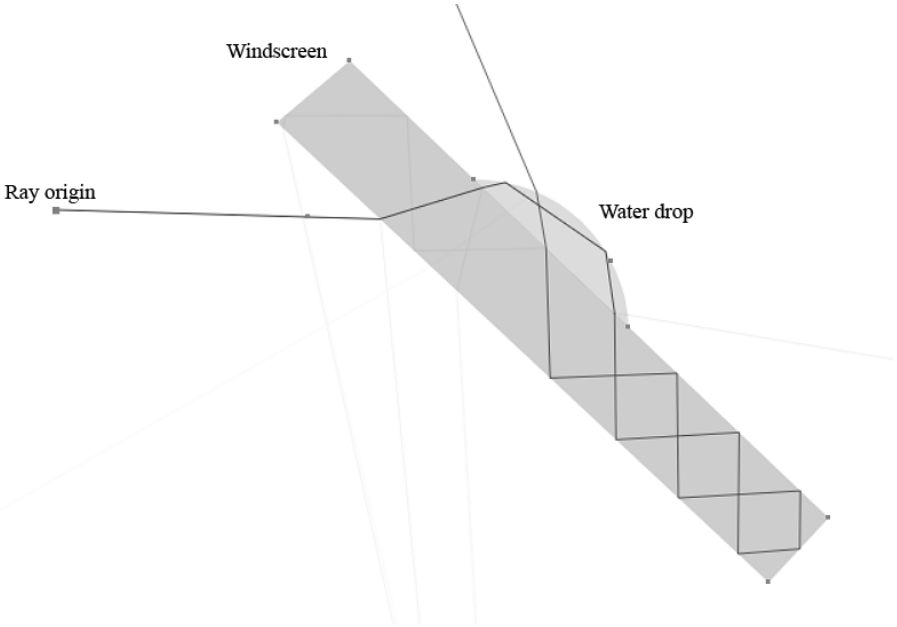

FIGURE 6.5: A trapped ray inside the windscreen causing the ray to bounce fifteen times before it enters the scene.

Most automotive cameras are mounted to the windscreen in a black casing as shown

# Bounces	Color
1	Red
2	Green
3	Blue
4	Yellow
5	White
6	Light Blue

TABLE 6.1: Color coding for the number of bounces in the refraction sample plot

in Figure 6.6. The black casing is designed in such a way that the interior reflects as little light as possible in order to reduce glare. When a ray in a ray tracer implementation would hit this surface, the returned light energy is small and could be neglected. As a result of that black areas are visible inside the drop. We show what parts of a water drop will be black under varying inclination angles and with different curvatures.

FIGURE 6.6: Example of a camera casing.

In order to make this even more clear we create a plot of where the rays sample the equirectangular skydome texture. The plotting process is the same as the regular ray tracing method which we will describe shortly. The only difference is that instead of taking a color from the texture map, a color is added to the hit pixel of the texture map in a nearest neighbor fashion. We base the color on the number of bounces that the ray made before hitting the environment map. The mapping from color to number of bounces is given by table 6.1.

The number of bounces can be more than six but for clarity of the illustration we omit these. Therefore we limited the color assignment to six and also give plots of only white assignments that go up to 20 bounces. The brightness of the pixels is controlled by both the number of rays hitting the same pixel and the amount of light energy according to the Fresnell equation that is transported along the ray. With the plots we are able to

show what parts of the 360 degrees environment maps are sampled the most for different drops shapes and positions.

6.1.2 Drop model

We are focusing on the render process and not on whether the geometric representation of the water drops is physically correct. Therefore we used a spherical cap to model water drops which was also done in previous work ([KKY93] [SS07], [HR09], [RG09], [NK12], [CCW13]). This allows us to have direct control over the curvature and contact angle of the drop without the difficulties of handling the forces acting on a drop inside a physics engine. The control over the curvature is of course limited to a symmetric dome shape.

We use four watertight drop models created in 3ds Max with 120 horizontal rows of edges which gives a high enough polygon count that does not show any signs of discretization anymore. Figure 6.7 displays the four models we use to show the effects of different curvatures. Every drop is capped and then scaled to the same radius. In our simulation we use a standard diameter of three millimeter which is an average drop size.

FIGURE 6.7: Drop models created with a spherical cap at 0.5, 0.6, 0.7 and 0.8 of a unit sphere.

A geometric representation of water drops is necessary for ray tracing, but for direct mapping methods we only need a normal map which is described in section 6.1.4. In the work from Chen et al. [CCW13] a generated heightmap is used to create a normal map. We want to show what errors can be expected when using a bitmap based heightmap of a hemispherical drop model. In order to do this we construct a 3D mesh from a heightmap using a variation of the marching squares algorithm.

We take four pixel values in a square and identify which case we are dealing with. The different cases are given in Figure 6.8 in which a circle corner is a pixel value of zero, a filled disc in the corner is a pixel value larger than zero and the edges determine what triangles need to be added to the 3D mesh. Whenever a triangle is added for the top, we also insert a triangle on the zero plane. The result is a watertight 3D model grid that has different heights.

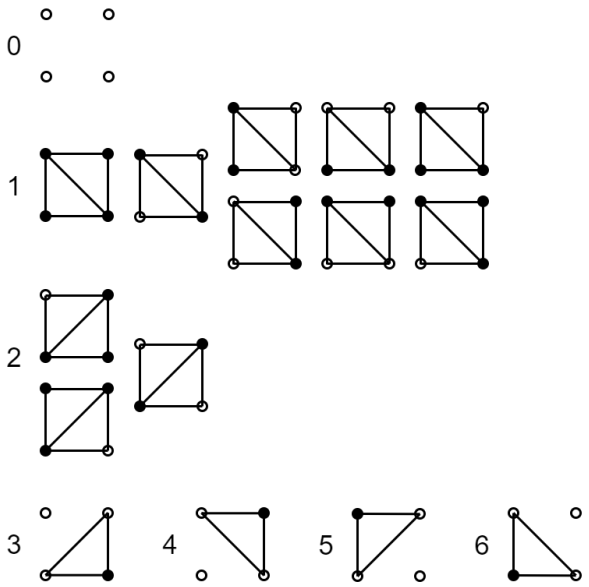

FIGURE 6.8: Marching squares cases.

6.1.3 Ray tracing water drops

We use a ray tracer implementation because it models reflection and refraction of light in a physics based way. A basic whitted-style ray tracer would already be enough to model these effects but this does not alleviate the effects of point sampling. Therefore we also included subpixel sampling to counter aliasing effects. Instead of casting a ray through the center of the pixel, we sample the area of the pixel stochastically. This is schematically shown in Figure 6.9. The contribution of these random samples are averaged which makes our implementation a distributed ray tracer.

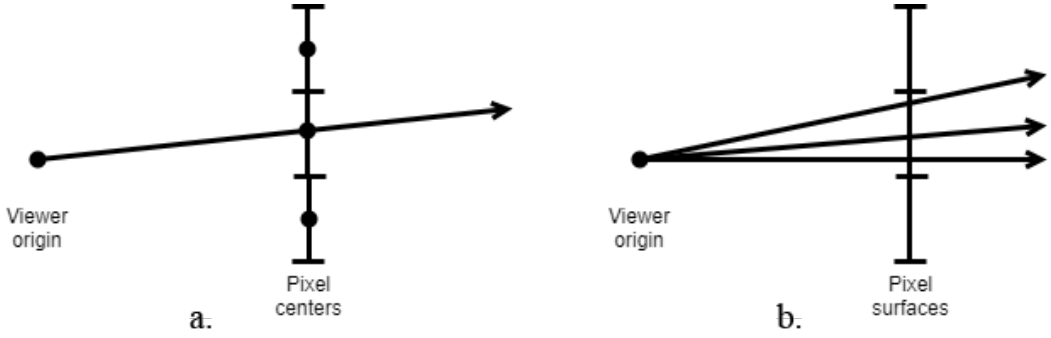

FIGURE 6.9: Left: sampling the scene through pixel centers. Right: sampling the scene through random points on the surface of the pixel.

A basic Whitted-style ray tracer uses a pinhole camera which casts a ray from a single point through every pixel center of the virtual image as shown in Figure 6.10 a. In the real world however, cameras have a limited depth of field which means that drops on the windscreen will rarely be in focus. In order to add the depth of field effect we use a thin lens approximation as described in the PBRT book of Pharr et al. [PJH16].

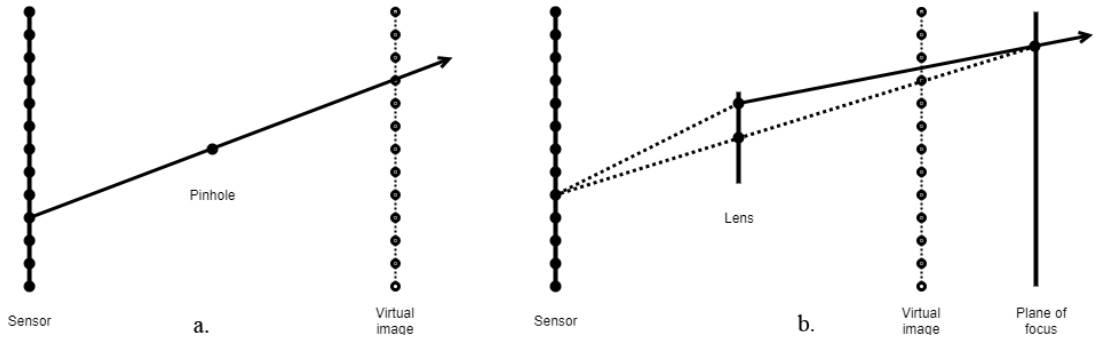

FIGURE 6.10: a) Pinhole camera model. b) Thin lens approximation model.

First a random point on the surface of the lens is chosen. After that a ray is traced from the center of the lens through a point on the virtual image the same way as in the pinhole camera model. Then the intersection of this ray with the plane of focus is calculated. The distance from the center of the lens to the plane of focus is given as input by a user. The normalized vector between the sampled point on the surface of the lens and the intersection point of the pinhole camera ray with the plane of focus is the new direction for the ray that is cast into the scene. Figure 6.10 b shows a schematic overview of this approach.

Points far from the plane of focus will be distributed over multiple pixels in the shape of the lens which in our case is a perfect disc and points that are sampled close to the plane of focus for which the *circle of confusion* is smaller than the size of a pixel will be in focus. Points exactly at the plane of focus will of course also be in focus.

The framework we use for our ray tracer implementation is Embree [Emb11] which efficiently handles ray/scene intersections. It is a C++ library consisting of ray tracing kernels and is optimized for Intel CPUs. Embree makes use of the SIMD capabilities of the CPU to allow for parallel intersection instructions of single rays or ray packets with multiple primitives. It also implements a *bounding volume hierarchy* (BVH) acceleration structure with surface area heuristic [MB90] and spatial splits [SFD09].

A BVH acceleration structure is particularly well suited for scenes with vastly differing scales of geometry. It allows for correctly handling small water drops but also large geometric objects like cars, roads and houses in the same scene. All of this is hidden behind an API of Embree that is simple to use which allows us to focus on the rendering itself instead of the low level details of the acceleration structure.

6.1.4 Drop approximation with normal mapping

Normal mapping is the process of mapping normals from a 2D texture onto a surface of 3D geometry. It is a common technique used in game engines to lower the amount of polygons needed while still maintaining a high level of detail in the shading [SA09].

We use normal mapping to approximate the 3D geometry of water drops. First the normals of a 3D water drop are determined (Figure 6.11 a), then the normals are projected to the plane on which the water drop rests (Figure 6.11 b and c). After that the normals are rendered to a texture so they can be used for refracting sampling rays in a later stage. During the rendering process we invert the normals so they can be directly used (Figure 6.11 d).

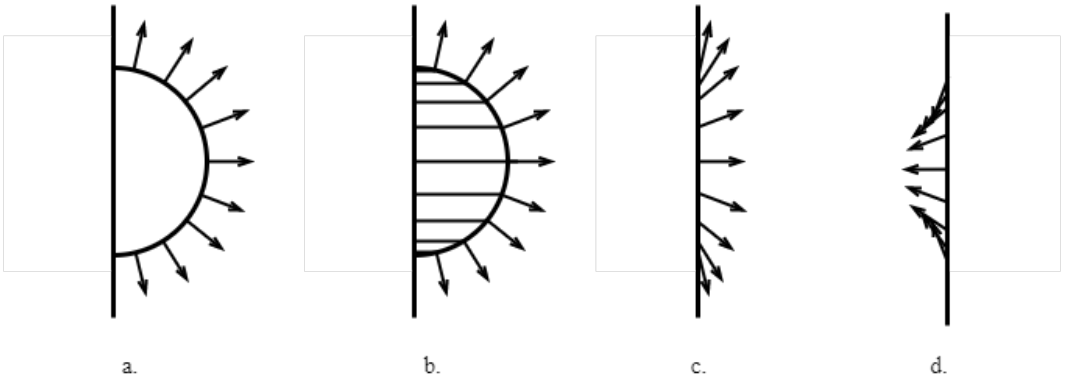

FIGURE 6.11: Normal mapping of a hemisphere.

Algorithm 3 shows the steps for the rendering process of the water drops using normal mapping. Line 1, 2 and 3 are the same steps we take for the ray tracer implementation and can also be implemented with a rasterizer engine. The rest of the method can be seen as a post process effect to the rendering process and is isolated from the previous steps. We use a simple linear alpha blending $(1-\alpha) * backgroundTexturePixelColor + \alpha * dropTexturePixelColor$ is used to add the blurred drops texture to the background texture.

Algorithm 3 Direct mapping method

1: Load (or render) a 360 degrees environment map
2: Calculate primary rays with current camera settings
3: *backgroundTexture* ← Render background using *primaryRays* and *environmentMap*
4: Load water drop normal map
5: Refract the *primaryRays* using the *normalMap*
6: *dropTexture* ← Render water drops
7: Apply blur filter to *dropTexture*
8: Blend *backgroundTexture* with *dropTexture* using alpha blending

A point that is not in focus will be imaged on the sensor in the shape of the camera's aperture or in our case the shape of the lens. We assume that the aperture has the shape of a disc. Because of that we use disc convolution as blur filter on line 7. Although this is not the fastest way to blur an image, it is the closest way to how a camera forms a defocused image. With disc convolution a disc kernel is placed on every pixel of the image and the color of the center pixel is distributed over the pixels that overlap the disc. Other faster options for blurring are for example box blur or Gaussian blur, but

these filters do not uniformly distribute the intensity over a disc resulting in a wrong outcome.

The discretized disc kernels are precalculated offline to improve the online rendering time. Whether a pixel is part of the kernel can be determined by equation 6.3 with x and y being the coordinate of a pixel center with respect to the center of the kernel image. r Is the radius of the disc. Pixel centers that are inside or on the border of the disc are part of the kernel otherwise the pixel is masked out. Figure 6.12 shows an example of a kernel with a diameter of 6 and one with a diameter of 7.

$$x^2 + y^2 - r^2 <= 0 \tag{6.3}$$

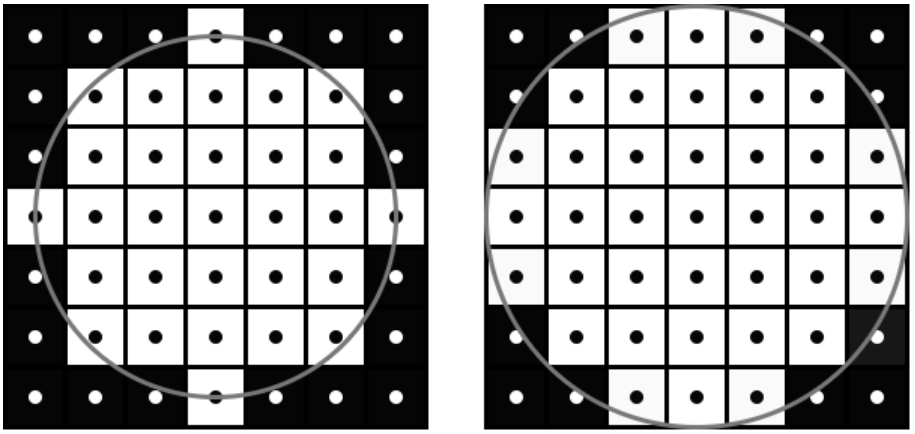

FIGURE 6.12: Examples of a kernel with a diameter of 6 pixels and one with a diameter of 7. Even kernel sizes result in a single pixel at the horizontal and vertical extremes. Odd kernel sizes do not have this.

A windscreen has a large angle with respect to camera sensor. Therefore drops in the lower part of the image will be less defocused than drops in the upper part of the image because the distance to the camera is different. This means that the kernel size of the blur filter needs to be a function of the distance to the camera in order to approximate the defocusing effect correctly. To determine the disc size we use the function from Roser and Geiger [RG09] as discussed in section 5.1.

6.2 Observations from the real world

In order to show what effects can be expected with water drops on the windscreen we took photographs of varying situations. The photographs were all taken with an IPhone 5 front camera.

6.2.1 Focused and defocused drops on a glass surface

Automotive cameras are always focused in the distance and are able to see sharp from a few meters to infinity. This means that the water drops are never in focus because they are to close to the camera. In the photographs of Figure 6.13 we show how focused and defocused water drops look like on a waxed windscreen. We can observe that the bright part of the water drop is more visible when water drops are in front of a dark background. When water drops are however in front of a bright background we can observe that the dark part of the water drop is more visible.

FIGURE 6.13: Left: stationary water drops which are in focus. Right: stationary water drops which are not in focus. The drops appear as bright blurry spots.

This is even more apparent in Figure 6.14 where we captured water drops during dawn with low light conditions. The dark part of the water drops will completely blend with the dark background and only the bright spots inside the water drop are visible.

FIGURE 6.14: Stationary drops on a glass window in the evening. Left: drops in focus. Right: drops not in focus.

6.2.2 Optical effects of water drops on a glass surface

Water drops function as small lenses that refract the light and change the plane of focus. Therefore, it is possible to have the effect of Figure 6.15 where the background and the drop itself are not in focus but the image inside the drop is. This example is not common in automotive applications but it does make us aware of the focus altering characteristics that can be expected.

FIGURE 6.15: A water drop changing the plane of focus of the camera which leads to a focused image inside the drop.

In Figure 6.16 and 6.17 we shined with an LED through water drops on a vertical glass surface. We observe in the focused drop photo that there appears a tiny bright

border around the water drop. This is probably due to dirt on the glass surface causing the drop to have micro irregularities in its border area that refract the light differently. The windows of cars are bound to have dirt on them which can also lead to irregular water drop shapes.

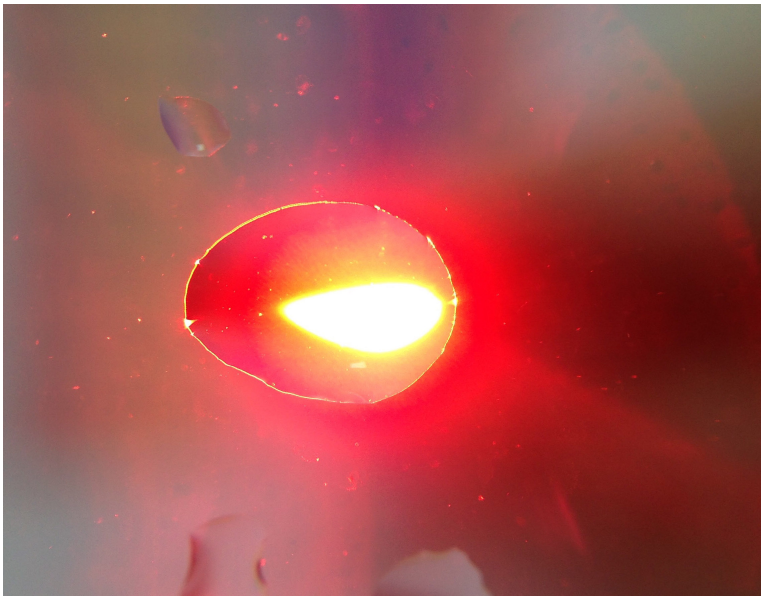

FIGURE 6.16: An LED shining on a water drop which is situated on a vertical glass surface. A tiny bright border appears around the water drop.

These irregularities are also visible when the drops are not in focus as Figure 6.17 shows. The upper imaged LED refraction with a black circle around it shows a serrated top edge while the lower border of this imaged refraction is smooth. Also the imaged refractions that are not circled have caustic patterns in them. These examples show that the imaged refraction is not always a perfect round disc and that the shape of the water drop is important for the defocused shape of the imaged refraction.

Another observation we can make in Figure 6.17 is that wave optical patterns appear in the imaged refractions that are circled. These patterns are caused by interference of the waves. When two wave peaks are in sync they reinforce each other which increases the amplitude, however, the waves are cancelled out when a peak lines up with a valley.

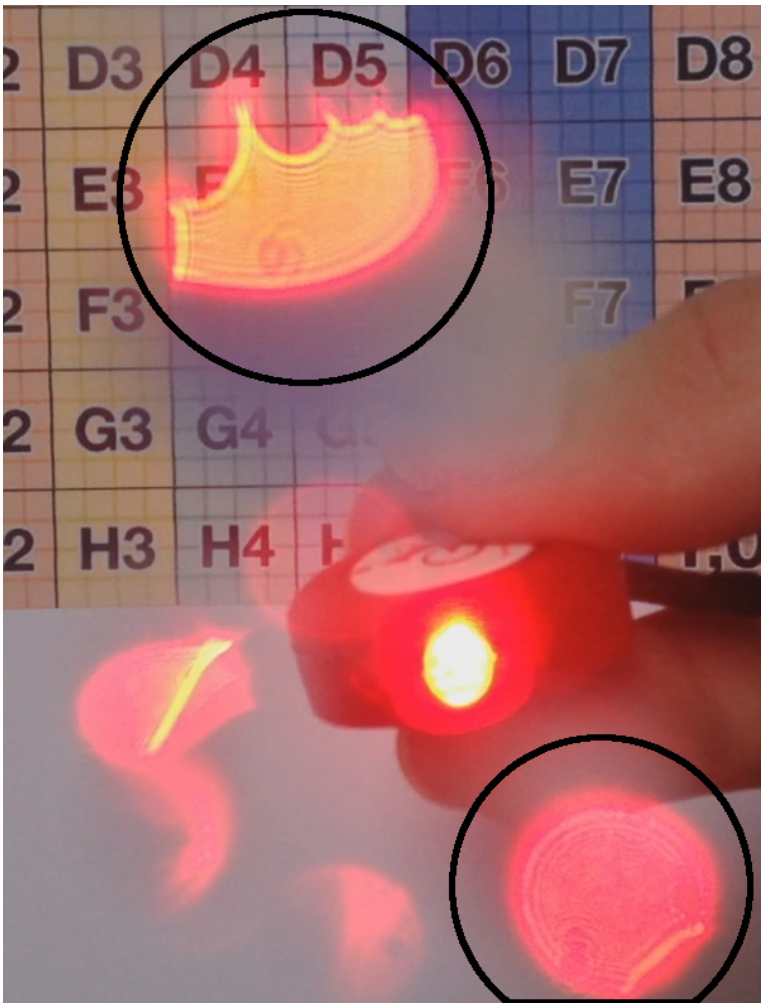

FIGURE 6.17: An LED shining on water drops on a vertical glass surface. The camera is focused on the background, therefore the drops are not in focus. There are interference fringes visible inside the imaged refraction shapes. Also the bokeh shapes are not always completely round.

6.2.3 Reflections inside the drop

Lastly, we want to display the presence of reflections inside the drops. Figure 6.18 shows a photo that is taken from behind a tilted glass surface. The surface has received a wax treatment common for car windscreens which makes the contact angle of the drops larger. The Figure shows two close ups, one of a group of small drops and one close up of a large drop. The visible difference is that the dark top border of the drop is more visible in the large drop than it is in the small drops. Also, there is a green line visible inside the large drop which is the reflection of light bouncing inside the glass plate.

To show more clearly what part of the room around the water drop corresponds to the visible dark border we took an LED and shined it upon the drops from different directions. Figure 6.19 shows a photo where we place the LED directly below the water drops on the camera side of the glass. A bright reflection of the light appears inside the

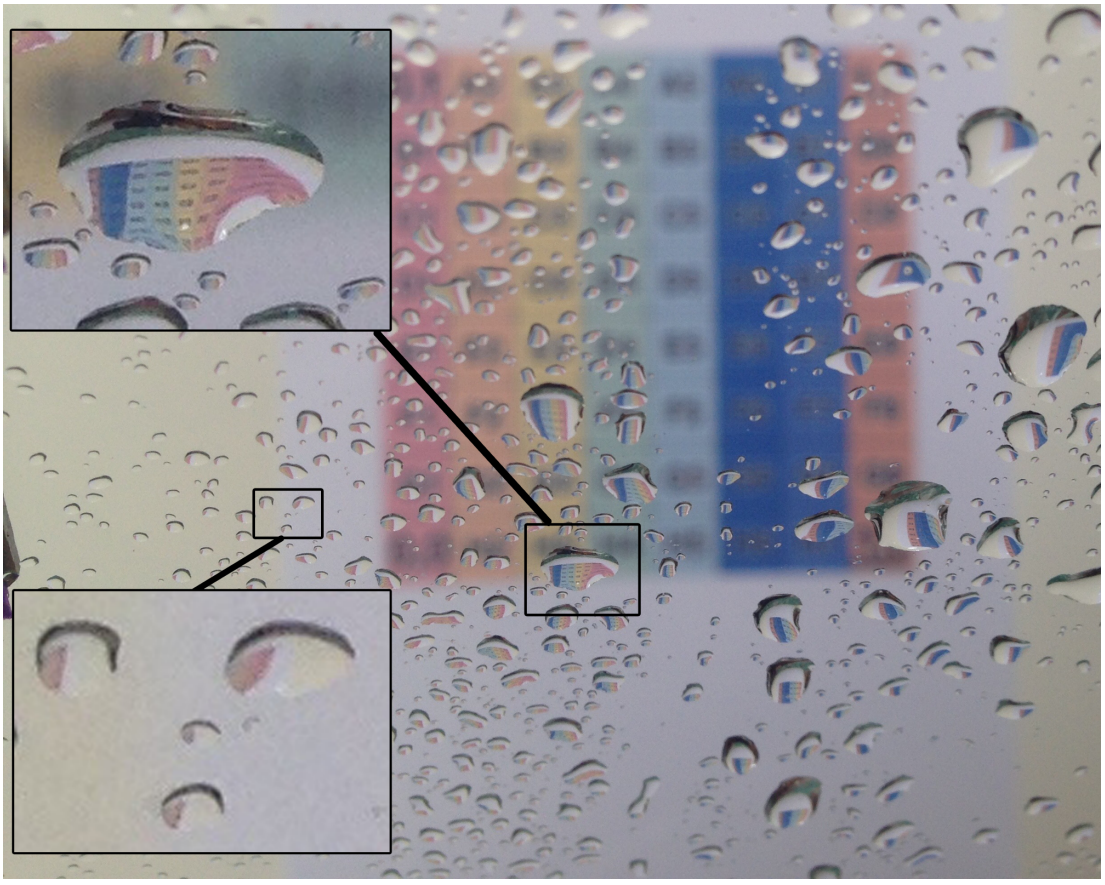

FIGURE 6.18: Water drops on a waxed glass surface under an incline. Large drops show clearer signs of internal reflection than small drops.

dark border which means that a small part of the image inside water drops on a tilted surface is actually coming from below instead of from directly behind the drops.

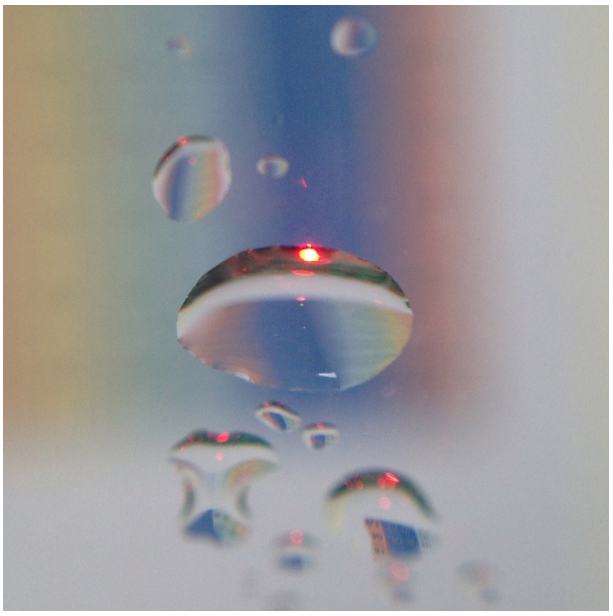

FIGURE 6.19: Water drops on a waxed glass surface at an incline with an LED shining from below the center drop. A bright spot appears in the upper ring of the drop.

When the surface on which the water drops rest is parallel to the camera's lens, the dark border is not present. This is shown in Figure 6.20 which contains water drops on a vertical surface.

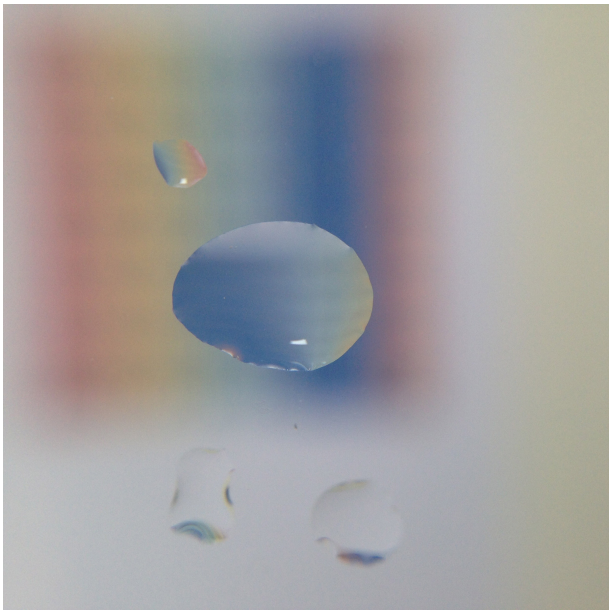

FIGURE 6.20: Water drops on a vertical glass surface that do not show signs of internal reflections

6.3 Simulation results

The following simulation results are all generated with our own implementation. In order to have a good comparison with the real world observations we partly recreated the experimental setup of the previous subsection in the virtual world.

6.3.1 Internal reflections simulation

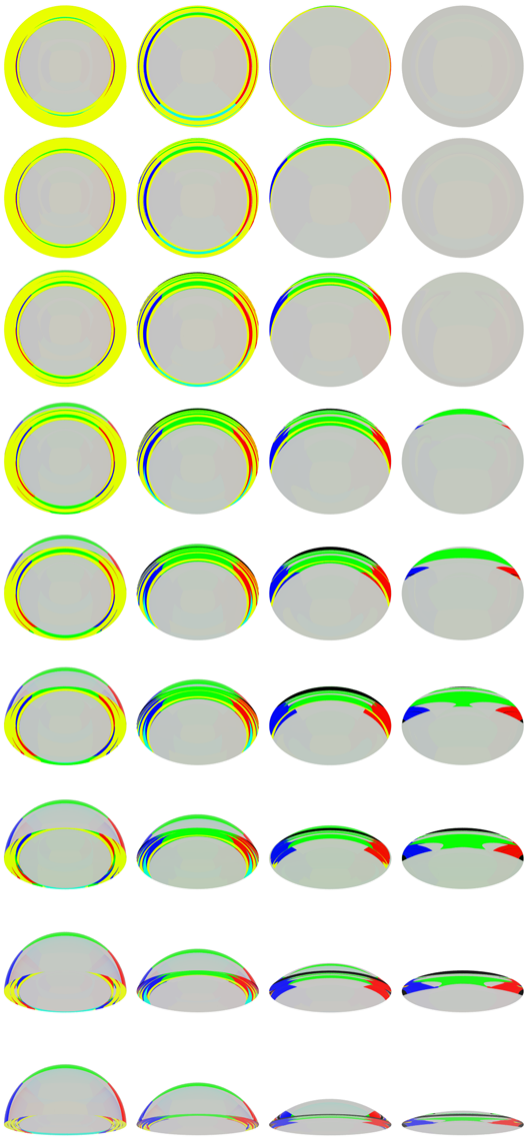

FIGURE 6.21: An overview of different drop shapes with changing angle with respect to the camera. From left to right it is 0.5, 0.6, 0.7 and 0.8. From top to bottom the angle is changed in steps of 10 degrees from 0 to 80 degrees. It is rendered with 50 spp, subpixel sampling is on and the maximum number of ray bounces is 20.

From the real world observations we see that the internal reflections are visible when the glass surface is at an inclination with respect to the camera. Therefore, we created

test results with our ray tracer method that show this effect for different drop contact angles and for different inclination angles which is shown in Figure 6.21. In order to make it clear what part of the environment map is imaged inside the water drop we use a color coded environment map of inside a cube. The left image of Figure 6.3 shows this environment map. We can see that the gray front face is visible in every drop. Also the yellow back face and the green bottom face of the cube are present in almost every rendered water drop.

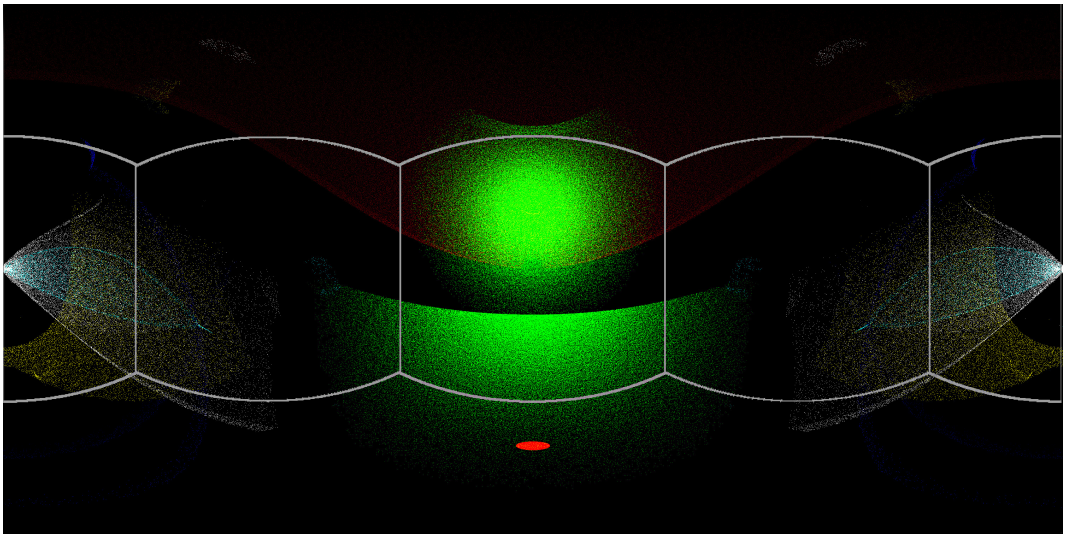

FIGURE 6.22: A sampling plot on the 360 degrees environment map of a 0.5 capped sphere under an inclination of 30 degrees with respect to the horizon in the center of the view of the camera.

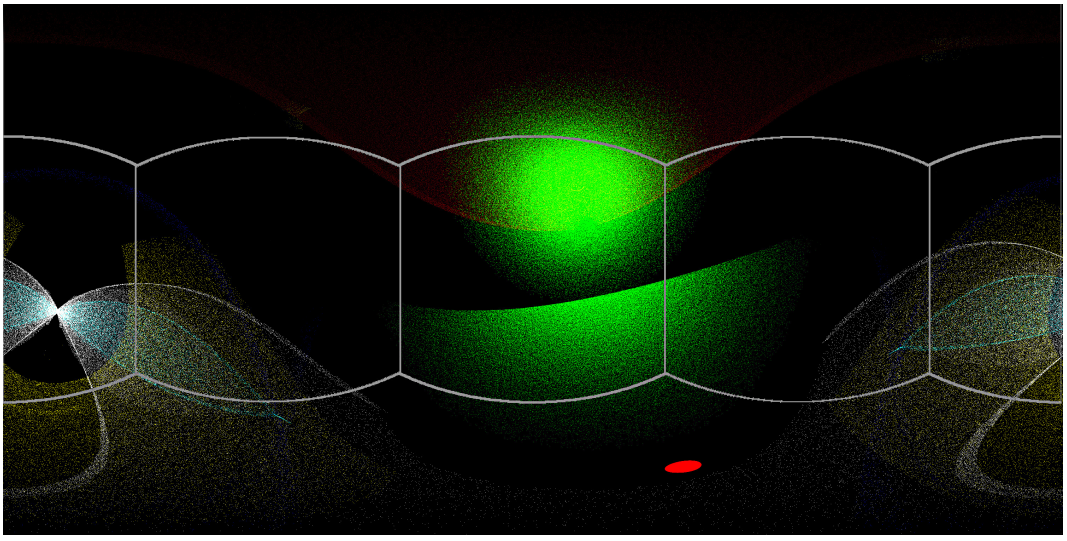

FIGURE 6.23: A sampling plot on the 360 degrees environment map of a 0.5 capped sphere under an inclination of 30 degrees with respect to the horizon in the right top corner of the view of the camera.

Another way to visualize the refractions is to plot the samples that are taken from the environment map. Figure 6.22 and 6.23 show the sampling plot of a hemisphere as

geometric drop model. The former has a centered drop and the latter image shows the same drop in the top right corner of the view of the camera. We did the same in Figure 6.24 and 6.25 where we used a sphere capped at 0.7.

The color coding of table 6.1 states that red pixels are single bounce paths so a ray hit the drop surface once and bounced off into the environment map. Green pixels are samples that are taken after two bounces which means that a ray entered the drop at the first bounce and left the drop at the second bounce. Blue, yellow, white and light blue denote the paths that bounced respectively one, two, three or four times inside the drop. The gray lines indicate where the faces of the projected cube are.

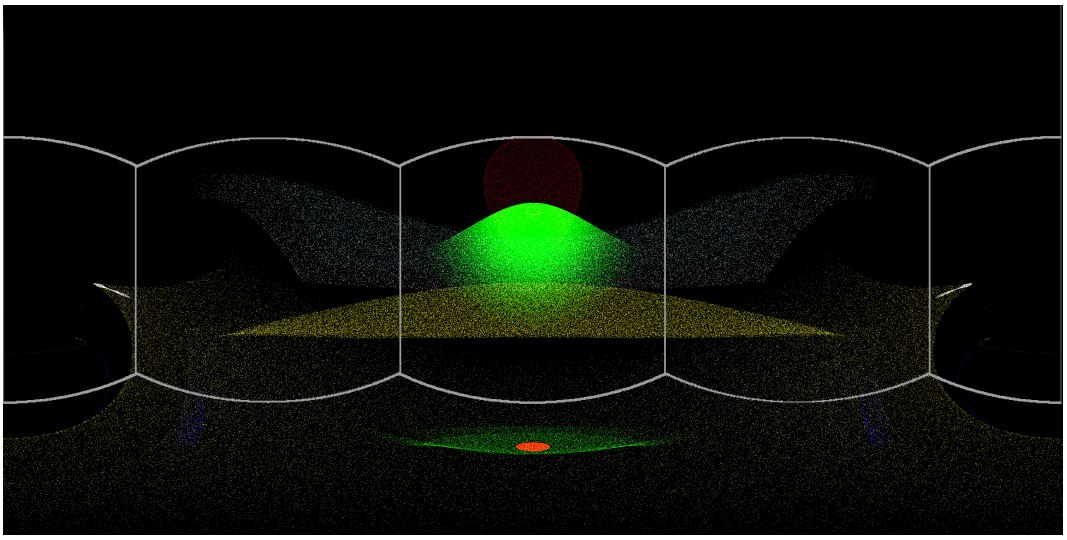

FIGURE 6.24: A sampling plot on the 360 degrees environment map of a 0.7 capped sphere under an inclination of 30 degrees with respect to the horizon in the center of the view of the camera.

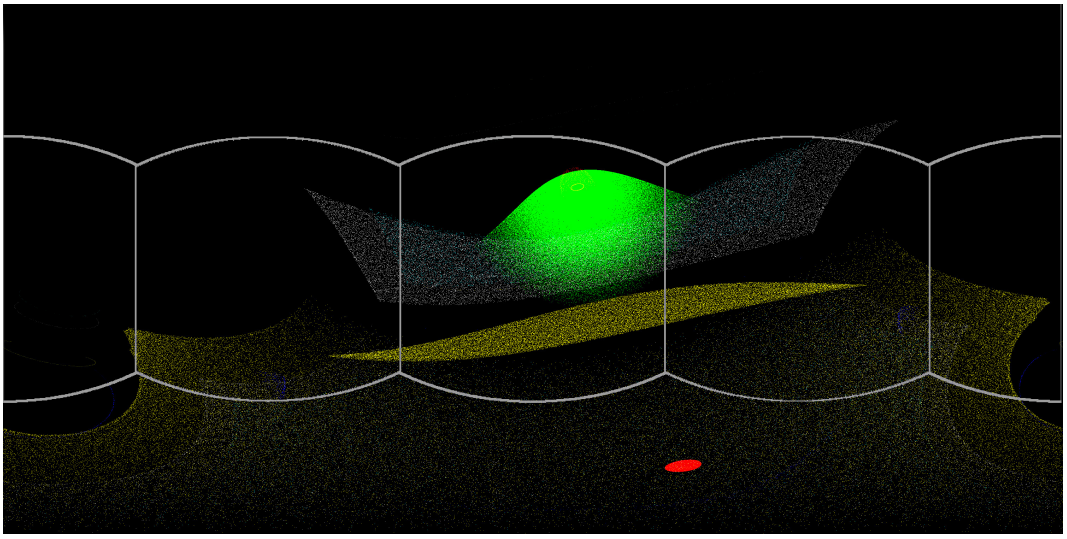

FIGURE 6.25: A sampling plot on the 360 degrees environment map of a 0.7 capped sphere under an inclination of 30 degrees with respect to the horizon in the right top corner of the view of the camera.

We also show in Figure 6.26 what happens when a normal map representation of a drop is tilted compared to a ray traced drop. Total internal reflection is not defined at a flat surface. Therefore, we colored those parts black to emphasize where it happens.

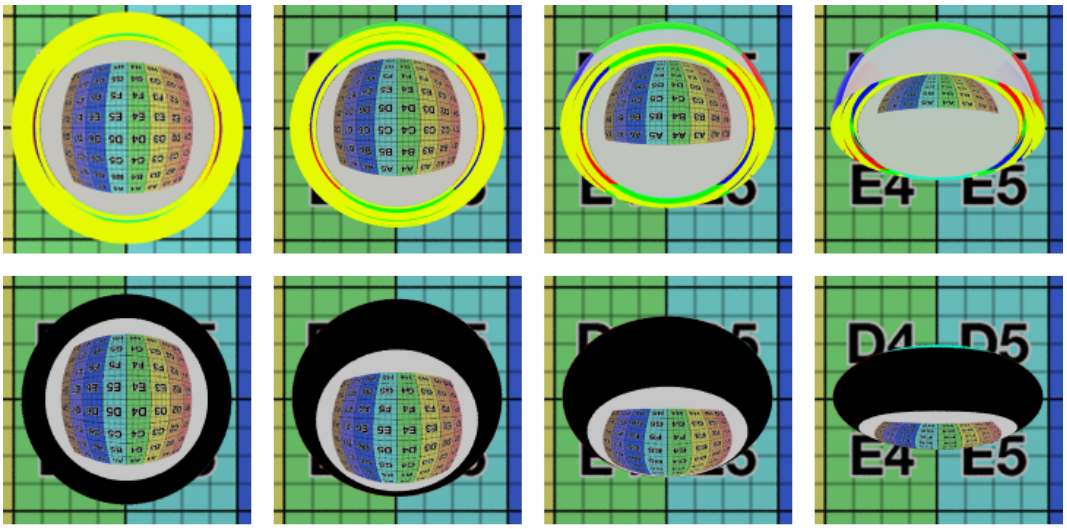

FIGURE 6.26: The top row displays a ray traced hemisphere and the bottom row shows a direct mapping approximation of the hemisphere both under different inclination angles.

6.3.2 Camera effects

We are able to render camera effects that are also observed in the real world. Figure 6.27 shows a simulation of the focused image inside the water drop which is rendered with our ray tracer implementation.

FIGURE 6.27: A simulation of a defocused water drop and background. The image inside the drop is in focus.

Another effect we tried to recreate is the defocused LED image inside the water drop. For this we placed a bright point light source in front of a disc at around the same distance from the lens as in our real world experimental setup. The result of this

is shown in Figure 6.28 where we display four different amounts of samples per pixel. After 2000 samples per pixel the sampling noise is still visible in the bright spot while it is not visible anymore in the defocused drop. There are no wave optical effects visible.

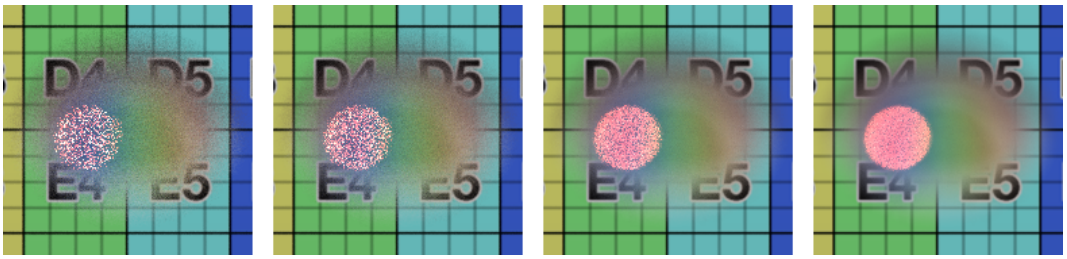

FIGURE 6.28: A defocused drop with a bright light spot. From left to right it is 50spp, 100spp, 500spp and 2000spp.

In order to show the differences between our ray tracing and direct mapping implementation we rendered both the geometric representation and the normal map representation of a sphere capped at 0.7. The ray traced drop us blurred with the thin lens approximation and the direct mapping drop is blurred with disc convolution. From these results we created a difference image as shown in Figure 6.29. We minimize the difference by reflecting the ray in the parts of the normal map where total internal reflection occurs.

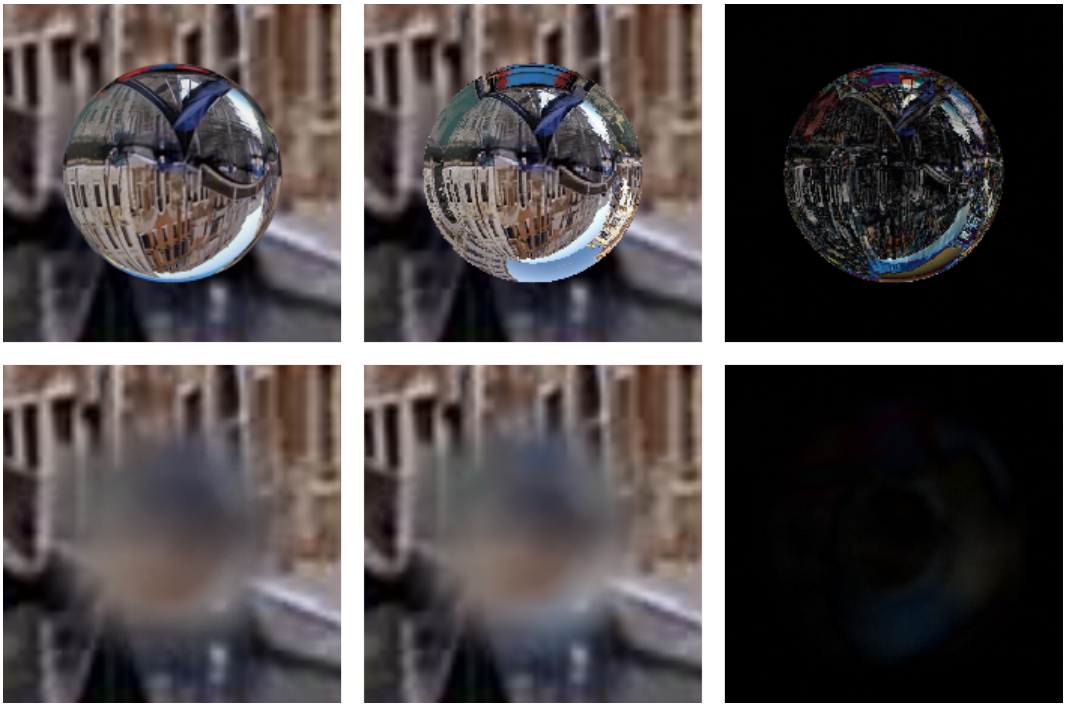

FIGURE 6.29: Comparison of a 0.7 capped sphere model. The left is rendered with a ray tracer with 2000 samples per pixel, the center is rendered with direct mapping and the right is a difference of the previous two. The drop is at 6 cm from the lens and has a diameter of 3mm. The defocusing of the drop is done with an aperture diameter of 2mm and focusing at infinity. This leads to a disc kernel size of approximately 49 pixels.

6.3.3 Drop geometry

Lastly, we looked into a geometric water drop representation based on a heightmap. Figure 6.30 shows a close-up of a two water drops created with heightmaps that have different bit depths. The drop geometry based on the lower bit depth heightmap has less levels and therefore has more visible patterns than the high bit depth heightmap. Figure 6.31 shows the artifacts that occur when using a heightmap in general.

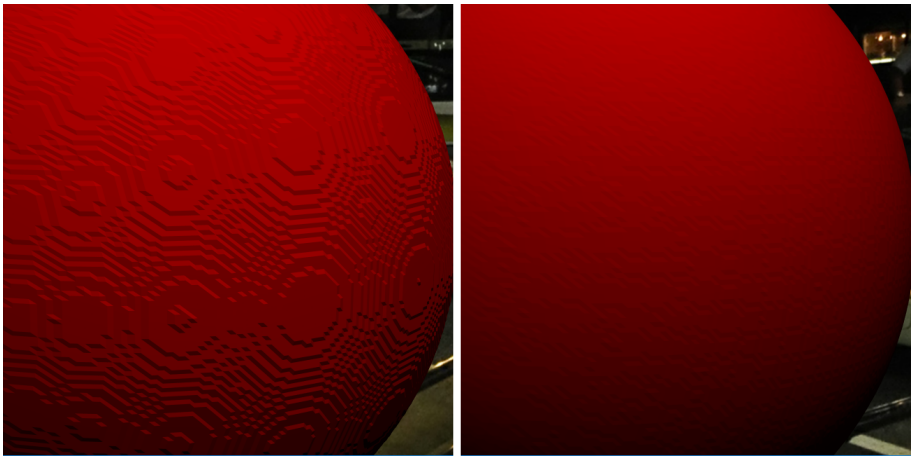

FIGURE 6.30: Left: A drop mesh based on a 24-bit heightmap. Right: A drop mesh based on an 96-bit openEXR heightmap.

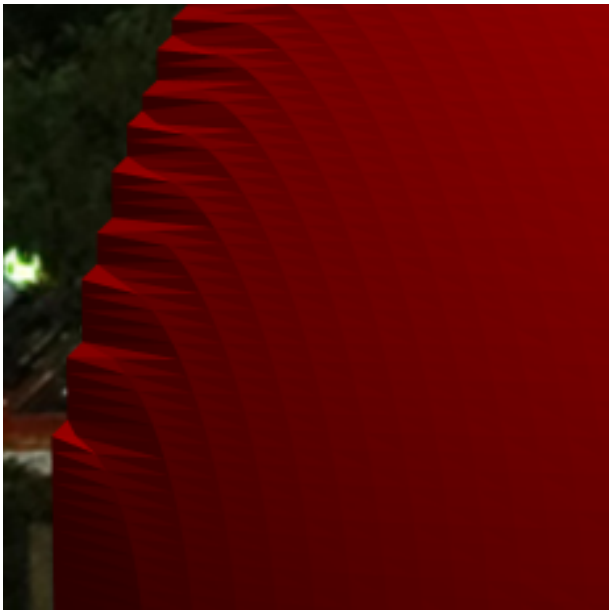

FIGURE 6.31: The border of a hemispherical 3D mesh based on a heightmap.

6.4 Discussion

6.4.1 Internal reflections

From Figure 6.21 we can see that both the contact angle and the angle of the drop itself with respect to the camera are important. The amount of internal reflections becomes larger when a drop has a smaller contact angle with the surface it rests on or when the drop is more perpendicular to the lens objective. It is also possible that there are no internal reflections when the contact angle is small like the three top right drops show. Another thing we can state about this image is that the larger the contact angle, the more area of the drop represents a reflection from the back. There is more yellow area in the first column than there is in the other columns which is the square behind the camera. These results are similar to what we observed in Figure 6.20, 6.19 and 6.18. It is however hard to make a specific statement about with what angles this occurs or in what situations real world drops show internal reflections because their shape, orientation and position with respect to the camera is not clearly defined.

Internal reflections are not defined in the direct mapping method. This means that a direct mapping method is only able to approximate drops that do not have internal reflections. The previously discussed works from Takenaka et al. [TMT08], Nakata et al. [NK12], Sato et al. [SDY03], Halimeh and Roser [HR09], Chen et al. [CCW12] [CCW13] and Stuppacher and Supan [SS07] that use direct mapping do not discuss this issue explicitly. Only in the work of Nakata et al. [NK12] the result of total internal reflection is visible as a hard border between a round center image and a ring around that center, but they do not state how they approximate it. We show with our results that it is important to include internal reflections in order to make the outcome more physically correct.

When the direct mapping technique is used to approximate drops on a tilted surface the error becomes even larger. The inverted image of the background inside the drop moves in a different way. Figure 6.26 shows that in a ray traced drop the refracted image moves to the top which is not the case for the direct mapped water drop approximation.

Lastly, we show with Figure 6.22, 6.23, 6.24 and 6.25 where the samples are taken from the 360 degrees environment map for a single drop. Red pixels show the single bounce reflections of which there are more in the hemisphere plots than there are in the capped sphere plots. This is because the camera is able to see more of the curved surface of the hemisphere drop than it sees of the curved surface of the capped sphere drop model.

Furthermore, the green pixels show the samples that are taken after two bounces. We can see that both in the hemisphere and the capped sphere plots the green pixels stay within the forward facing 180 degrees of the environment map. Kaneda et al. [KKY93] state that only the front face of the cube map needs to have the highest resolution

which is partly confirmed by our plots. Most of the green pixels are within the front face denoted by the gray lines but for the hemisphere plot there are also green pixels outside the front face region. Drops that are modeled with a hemisphere need 180 degrees of the environment map in order to have the green part covered. Flatter drops do not need such a large angle and while most of the time drops have a flatter curvature than a hemisphere we can state that a smaller environment map can be used.

The capped sphere plots have more yellow and white pixels in the central and bottom region of the environment map than the hemisphere plots have. In the hemisphere plots the colored pixels that represent more than one internal bounce are in the back of the environment map. Samples that are from back reflections or down reflections are not to be taken into account because there is a camera casing that blocks those samples. The plots show that for the capped sphere drops multiple internal bounces are important to simulate because they are mapped towards the front. For drops with a larger contact angle like the hemisphere drop there are less internal reflections that map towards the front so in that case the internal reflections are less important to simulate. From the real world observations we can see that the images inside the drop are closer to the capped sphere than to the hemisphere shape.

6.4.2 Blurring characteristics

To show how the direct mapping method with disc convolution blurring compares to a ray traced image we rendered a similar setup with both methods. The result is shown in Figure 6.29. What we can see is that although a focused drop shows a clear difference, the defocused drop does not. The difference error is spread out over a larger area because of the blurring and is less noticeable. Upon visually inspecting the blurred drops we can say that they are similar to a large extent for this simple case. When both drops are tilted however, larger differences can be expected as shown in Figure 6.26.

With the thin lens approximation we are able to correctly models the optical effects that can occur with defocused water drops on a glass plate. Even a special case like when the background and the water drop itself are not in focus but the image inside the water drop is. A real world example of this effect is given in Figure 6.15 and Figure 6.27 depicts a simulation. Trying to simulate this case with the direct mapping method is not possible because the complete drop will be blurred.

It takes however a lot of samples before the ray tracer has a result without noticeable noise. This is extra visible in night time simulations where distant bright lights are present. Figure 6.28 shows that after 2000 samples per pixel a bright spot is still noisy while the rest of the blurred drop does not show noise anymore. Our method based on a distributed ray tracer is therefore not sufficient to simulate night time scenarios with large differences in light intensities. It would take too much time to get a satisfactory

final result. In daytime scenarios the differences in light intensities are not that large. Therefore fewer samples are necessary. The direct mapping technique does not suffer from sampling noise and can deliver results a lot faster because the process is less computationally expensive.

Figure 6.17 shows what happens when an LED is behind defocused drops. Due to the irregular shape of the drop it is possible that irregular bokeh shapes appear instead of perfect round discs. For a ray tracer this does not pose any real problems but the direct mapping technique is not able to cope with this optical effect. It will blur all the pixels with a round convolution kernel which does not capture the caustic effects.

Another observation we can make is that there are interference fringes visible inside the bokeh shapes. This can be explained by considering light as waves instead of particles. When two waves of light have different path lengths the peaks can cancel each other out. This effect can only be simulated when wave optics are taken into account inside a ray tracer implementation. It is impossible to add this to a direct mapping method.

6.4.3 Drop geometry

In previous work the drops are approximated with a normal map which is created from a heightmap. This made us wonder whether this would be a viable option for ray traced drop geometry too. Therefore we created a 3D mesh from a heightmap to show what kind of errors can be expected. Figure 6.30 shows that when using a bitmap for the heightmap the bit depth is important to take into account. When the bit depth is too small there are visible patterns on the shape that even cannot be solved by interpolating the normals. A heightmap based on an EXR image file which has floating point values does not show the signs of patterns that clearly.

Another thing to take into account is that a heightmap cannot model the border of the water drop that well. Figure 6.31 shows artifacts that can be expected when the contact angle of the water drop is too large. The height increases too rapidly which causes the shape to be not smooth. The direct mapping method uses a normal map which has the same problem as a heightmap.

6.5 Conclusion

Our direct mapping method is able to approximate drops that are relatively flat and also parallel to the lens. The error after blurring the water drops is in such cases not that large. However, when the drop has a large contact angle or when the drop is at an inclination angle with respect to the lens of the camera, it is better to render the drop with our ray tracer method because internal reflections become more important for the end result. Previous work based on the direct mapping method does not mention

internal reflections at all. We show that large parts of the image inside the drop is actually part of back or down reflections and should not be omitted.

Furthermore, our distributed ray tracer based method is capable of simulating the optical effects of internal reflections that can be observed in photos. The addition of the thin lens approximation allows for creating basic depth of field that blurs the water drops. However, additional work needs to be done in order to simulate interference effects because wave optics need to be taken into account for these advanced simulations. Also accurate distortions of the camera objective can only be modeled with a complete simulation of the lenses of the camera.

6.6 Future work

A direction for future work is creating a realistic drop model to replace the capped sphere model that we used. The new drop model should also include animations of the ray traced drops which can be a challenge. It requires the BVH to be updated after every time step, but it is worth the effort in order to further increase the realism and the capabilities of the simulation. Also looking into optimizing the ray tracing to speed up the rendering process can be beneficial. Our implementation is not able to render a ray traced drop in real time which is necessary for hardware in the loop or camera in the loop testing.

Chapter 7

Final conclusion

In this work we contributed to answering the question of how automotive cameras can be simulated with also taking the weather into account. We looked both into color representations which are part of the internal camera characteristics and water drop simulation on the windscreen which is part of the external simulation aspects of virtual world modeling.

With respect to color representations we can conclude that a color space of single wavelength primaries does not sufficiently approximate the raw response of an automotive camera. We hoped to get more information from the infrared part of the spectrum in the red component of the trichromatic color vector, but since the three sensitivity curves usually line up in the infrared part in RGB cameras there is no additional information to be gained. Therefore, it is best to use a wide gamut color space and transform this into the native camera color space during post processing. To get the best result without the limitations of metamerism it is recommended to use binning or Monte Carlo sampling of the full spectrum.

With respect to water drop simulation on the windscreen we show the importance of simulating the internal reflections of light inside the drop. This is done with our ray tracer method. We also show that the direct mapping method is not capable of realistically simulating drops with a large contact angle or when the contact surface of the drop is not aligned with the camera objective. Previous work that uses the direct mapping method does not address this limitation.

Acknowledgements

First of all I would like to thank Jacco Bikker for his supervision throughout the project. His positive mindset and constructive feedback were very helpful to me. I also want to thank Marcel Wantenaar who supervised me during the start of the project and guided me towards a research subject. I would like to offer my special thanks to Martin Kielhorn who repeatedly gave his professional opinion on subjects connected to optics and also to Edwin Rijpkema who helped me progress in the subject of color. Furthermore, I am grateful for Michael Phillips who took over the task of Marcel towards the end of my project. He provided me with valuable feedback on the research structure and thesis draft which helped me a lot. I also want to thank the company Tass International for giving me the opportunity to work with them on interesting topics in the field of computer graphics and also for accommodating a nice workplace. The contact that I had with colleagues was very positive and I immediately felt at ease within the working environment for which I want to thank them. Last but not least I want to thank my bride-to-be Kjersti and other family and friends for their moral support and patience during stressful moments.

Bibliography

[App68] A. Appel. Some techniques for shading machine renderings of solids. *AFIPS '68*, pages 37–45, 1968.

[C.94] Schlick C. An inexpensive brdf model for physically-based rendering. *in Proceedings of Eurographics '94, Computer Graphics Forum*, 13(3):233–246, 1994.

[CCW12] K.-C. Chen, P.-S. Chen, and S.-K. Wong. A hybrid method for water droplet simulation. *VRCAI2012 Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*, pages 341–344, 2012.

[CCW13] K.-C. Chen, P.-S. Chen, and S.-K. Wong. A heuristic approach to the simulation of water drops and flows on glass panes. *Computers and Graphics*, 37:963–973, 2013.

[CPC84] R.L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *SIGGRAPH '84 Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 18:137–145, 1984.

[Don06] W. Dong. *Rendering Optical Effects Based on Spectra Representation in Complex Scenes*, pages 719–726. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[DSWND04] D. D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide 4th edition*. Addison Wesley, 2004.

[EM99] G.F. Evans and M.D. McCool. Stratified wavelength clusters for efficient spectral monte carlo rendering. *Proceedings of the 1999 conference on Graphics interface '99*, pages 42–49, 1999.

[Emb11] Embree, a collection of high-performance ray tracing kernels developed at intel, https://embree.github.io/, 2011. Accessed: 2017-10-01.

[FBH97] H.S. Fairman, M.H. Brill, and H. Hemmendinger. How the cie 1931 color-matching functions were derived from wright-guild data. *Color Research & Application*, 22(1):11–23, 1997.

[FD97] G.D. Finlayson and M.S. Drew. Constrained least-squares regression in color spaces. *Journal of Electronic Imaging*, 6:484–493, 1997.

[FHP98] P. Fournier, A. Habibi, and P. Poulin. Simulating the flow of liquid droplets. *Proceedings of Graphics Interface*, pages 133–142, 1998.

[Gla95] A.S. Glassner. *Principles of Digital Image Synthesis*. Kaufmann, 1995.

[Gui32] J. Guild. The colorimetric properties of the spectrum. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 230(681-693):149–187, 1932.

[HDvM+14] J.F. Hughes, A. Dam van, M. MCGuire, D.F. Sklar, J.D. Foley, S.K. Feiner, and K. Akeley. *Computer Graphcis Principles and Practice, third edition*. Addison-Wesley, 2014.

[hei09] A 360 degrees panoramic view of the biggest camera store in japan, this file is licensed under the creative commons attribution 3.0 generic license https://creativecommons.org/licenses/by/3.0/ by flickr user heiwa4126, 2009. Accessed: 2017-09-07.

[HG83] R.A. Hall and D.P. Greenberg. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications*, 3:10–20, 1983.

[HR09] J. C. Halimeh and M. Roser. Raindrop detection on car windshields using geometric-photometric environment construction and intensity-based correlation. *2009 IEEE Intelligent Vehicles Symposium*, pages 610–615, 2009.

[Kaj86] J.T. Kajiya. The rendering equation. *SIGGRAPH '86 Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 20:143–150, 1986.

[KIY99] K. Kaneda, S. Ikeda, and H. Yamashita. Animation of water droplet moving down a surface. *The Journal of Visualization and Computer Animation*, 10:15–26, 1999.

[KKY93] K. Kaneda, T. Kagawa, and H. Yamashita. *Animation of Water Droplets on a Glass Plate*, pages 177–189. Springer Japan, Tokyo, 1993.

[KZY96] K. Kaneda, Y. Zuyama, and H. Yamashita. Animation of water droplet flow on curved surfaces. *Proceedings of Pacific Graphics*, pages 50–65, 1996.

[LSP14] Lamp spectral power distribution database, http://galileo.graphycs.cegepsherbrooke.qc.ca/app/en/home, spectra are licensed under the creative commons attribution-noncommercial-no derivative 2.5 canada https://creativecommons.org/licenses/by-nc-nd/2.5/ca/deed.fr, 2014. Accessed: 2017-09-25.

[LW93] E.P. Lafortune and Y.D. Willems. Bi-directional path tracing. *Proceedings of third international conference on computational graphics and visualization techniques, COMPUGRAPHICS'93*, pages 145–153, 1993.

[MB90] D.J. MacDonald and K.S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer: International Journal of Computer Graphics archive*, 6:153–166, 1990.

[NK12] N. Nakata and Kakimoto. Animation of water droplets on a hydrophobic windshield. *WSCG2012*, 2012.

[Pee93] M.S. Peercy. Linear color representations for full spectral rendering. *SIGGRAPH '93 Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 191–198, 1993.

[PJH16] M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering, Third Edition: From Theory to Implementation.* Morgan Kaufmann, 2016.

[Pre] Prescan, simulation tool of adas and active safety systems, https://www.tassinternational.com/prescan. Accessed: 2017-10-01.

[RBA09] M. Radziszewski, K. Boryczko, and W. Alda. An improved technique for full spectral rendering. *Journal of WSCG*, 17:9–16, 2009.

[Rec15a] Recommendation itu-r bt.2020-2, parameter values for ultra-high definition television systems for production and international programme exchange, 2015.

[Rec15b] Recommendation itu-r bt.709-6, parameter values for the hdtv standards for production and international programme exchange, 2015.

[RF91] M.G. Raso and A. Fournier. A piecewise polynomial approach to shading using spectral distributions. *Proceedings of Graphics Interface '91*, pages 40–46, 1991.

[RG09] M. Roser and A. Geiger. *Video-based raindrop detection for improved image registration.* IEEE, 2009.

[RHD+10] E. Reinhard, W. Heidrich, P. Debevec, S. Pattanaik, G. Ward, and K. Myszkowski. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting, 2nd Edition.* Morgan Kaufmann, 2010.

[RKG10] M. Roser, J. Kurz, and A. Geiger. Realistic modeling of water droplets for monocular adherent raindrop recognition using bzier curves. *ACCV'10 Proceedings of the 2010 international conference on Computer visio*, pages 235–244, 2010.

[RMBvMvdV15] R. R. Molenaar, A. Bilsen van, R. Made van der, and R. Vries. Full spectrum camera simulation for reliable virtual development and validation of adas and automated driving applications. *Intelligent Vehicles Symposium (IV), IEEE*, pages 47–52, 2015.

[SA09] P. Shirley and M. Ashikhmin. *Fundamentals of Computer Graphics, third edition.* Taylor & Francis Inc, 2009.

[SACM96] M. Stokes, M. Anderson, S. Chandrasekar, and R. Motta. A standard default color space for the internet - sRGB, https://www.w3.org/Graphics/Color/sRGB.html, 1996. Accessed: 2017-09-26.

[SDY03] T. Sato, Y. Dobashi, and T. Yamamoto. A method for real-time rendering of water droplets taking into account interactive depth of field effects. *Entertainment Computing. IFIP*, 112:125–132, 2003.

[SFD09] M. Stich, H. Friedrich, and A. Dietrich. *Spatial splits in bounding volume hierarchies.* ACM New York, NY, USA, 2009.

[SFDC01] Y. Sun, F.D. Fracchia, M.S Drew, and T.W. Calvert. A spectrally based framework for realistic image synthesis. *The Visual Computer - Springer Journals*, 17:429–444, 2001.

[Sha03] G. Sharma. *Digital Color Imaging Handbook.* CRC Press LLC, 2003.

[SS07] I. Stuppacher and P. Supan. Rendering of water drops in real-time. *In Central European Seminar on Computer Graphics for students*, 2007.

[Sun00] Y. Sun. *A spectrum-based framework for realistic image synthesis*. PhD thesis, Simon Fraser university, 2000.

[T.80] Whitted T. An improved illumination model for shaded display. *Communications of the ACM*, 23:343–349, 1980.

[TMT08] S. Takenaka, Y. Mizukami, and K. Tadamura. A fast rendering method for water droplets on glass surfaces. *The 23rd International Technical Conference on Circuits/Systems, Computers and Communications*, pages 13–16, 2008.

[Uni05] Unity3D game engine, https://unity3d.com/, 2005. Accessed: 2017-09-26.

[Unr98] Unreal game engine, https://www.unrealengine.com/, 1998. Accessed: 2017-09-26.

[VG97] E. Veach and L.J. Guibas. Metropolis light transport. *SIGGRAPH 97 Proceedings*, pages 65–76, 1997.

[WMT05] H. Wang, P.J. Mucha, and G. Turk. Water drops on surfaces. *ACM SIGGRAPH*, pages 921–929, 2005.

[WND+14] A. Wilkie, S. Nawaz, M. Droske, A. Weidlich, and J. Hanika. Hero wavelength spectral sampling. *Computer Graphics Forum*, 33(4):123–131, 2014.

[Wri28] W.D Wright. A re-determination of the mixture curves of the spectrum. *Transactions of the Optical Society*, 30(4):141–164, 1928.

[YJJC98] Yu Y.-J., H.-Y. Jung, and H.-G. Cho. A new rendering technique for water droplet using metaball in the gravitation force. *WSCG1998*, 1998.