TU WIEN Informatics

Kido

eine hardwarebeschleunigte Lichtplanungssoftware

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Michael Franz Landauer
Matrikelnummer 11778912

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: David Hahn, PhD
Dipl.-Ing. Lukas Lipp

Wien, 26. September 2022

Michael Franz Landauer	Michael Wimmer

TU WIEN Informatics

Kido

a hardware-accelerated lighting planning software

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Michael Franz Landauer
Registration Number 11778912

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: David Hahn, PhD
Dipl.-Ing. Lukas Lipp

Vienna, 26th September, 2022

Michael Franz Landauer	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Michael Franz Landauer

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 26. September 2022

Michael Franz Landauer

Danksagung

Zu Beginn möchte ich meinen zwei Betreuern, David Hahn und Lukas Lipp danken, welche mich über den ganzen Zeitraum der Arbeit unterstützt haben. Am meisten schätzte ich die Diskussionen und Meetings, bei denen wir unser Wissen teilen konnten. Das Anfertigen dieser Abschlussarbeit war für mich ein einmaliges Erlebnis.

Zusätzlich möchte ich meinen Eltern danken, welche stets an mich geglaubt haben und mich in jedem Aspekt meines Lebens motiviert haben. Ohne sie wäre ich nicht da, wo ich heute bin.

Acknowledgements

First of all I want to thank David Hahn and Lukas Lipp for supporting me throughout the whole process of this thesis. I appreciate all the discussions and meetings we had, where we shared our knowledge on various topics. Working on this thesis was a unique experience for me, which I will miss.

Special thanks goes to my parents, who always believed and motivated me in every aspect of my life. Without them I would not be there were I am today.

Kurzfassung

Das visuelle System des menschlichen Körpers erlaubt es uns, unsere Umgebung wahrzunehmen, indem es einkommendes Licht verarbeitet. Dieses visuelle System ermöglicht es uns, Entscheidungen zu treffen, basierend auf den Informationen oder Objekten, die wir sehen. Abhängig von einer Aufgabe kann eine gut beleuchtete Umgebung von Vorteil sein. Um eine gut ausgeleuchtete Umgebung zu planen, werden Computerprogramme verwendet, welche eine Lichtsimulation ausführen und die Ergebnisse anhand vordefinierten Bedingungen prüfen. Der momentane Ablauf von State of the Art Software ist sequenziell, wo ein User zuerst eine Szene plant und danach die Lichtsimulation startet. Das Problem bei dieser Herangehensweise ist, dass der User keinen Einblick in die resultierenden Lichtverhältnisse hat, bis die Simulation abgeschlossen ist. Lichtplanung ist ein iterativer Prozess, bei dem die Szene öfters geändert wird, bis der Plan den Anforderungen gerecht wird. Das Verkürzen der Zeit pro Iteration reduziert die ganze Planungszeit und steigert somit die Produktivität des Users. Diese Arbeit präsentiert einen neuen Planungsablauf, indem die Software zwischenzeitliche Ergebnisse der Lichtsimulation dem User zur Verfügung stellt. Durch unseren Ansatz erhoffen wir uns, dass die Planung intuitiver ist und somit die Planungszeit insgesamt reduziert werden kann.

Abstract

The human visual system allows us to perceive our environment by processing incoming light. This visual system enables us to make decisions based on the information or objects we see. Many tasks in our daily life require us to see our surroundings. Depending on the task, a well-illuminated environment will be beneficial to accomplish the given task. To properly design lighting environments, computer applications are used to simulate the distribution of light and test if the results meet the predefined requirements. The current workflow of state-of-the-art applications is sequential, where the user first plans a scene and then triggers the lighting simulation. The problem with this approach is that the user will have no insight into the resulting lighting conditions until the simulation has finished. Planning a lighting environment is an iterative process, where the user iteratively changes the scene till the plan meets the requirements. Reducing the time per iteration will also reduce the overall planning time and increase the planner's productivity. This thesis presents a novel workflow by providing the user with intermediate results of the computed lighting conditions in the planning phase. We believe that this approach will help the user to plan lighting environments faster and more intuitively.

Contents

CHAPTER 1

Introduction

Visual perception plays an essential role in the daily life of humans, and much information can only be gathered visually. Reading literature and perceiving visual art like architecture or paintings are examples where the ability to see is required, but also interpersonal communication, where we perceive the mood through the facial expression of our communication partner. The human visual system enables us to see our environment and based on the visually gathered information, we are able to include this information in our decision-making process. To see our surroundings, good lighting conditions are crucial so that the human visual system is able to process the incoming light.

The sun is a natural emitter and is our most used and most important light source. Most of the work done by humans is scheduled during the daytime to make use of the sun's light exposure. It is a reliable and free light source but it is not available at night. This unavailability results in the problem that artificial light sources are required to maintain good lighting conditions for various tasks like reading, crafting, or painting. Different tasks require different lighting conditions. Reading a book will not require as much light as crafting jewelry in a workshop. To achieve optimal task-specific lighting conditions, we have to plan a lighting environment that fits to the performed task. Artificial light sources can illuminate the environment independent of the current time and position of the sun. Multiple computer applications are available on the market, which assist the user to plan indoor luminaire setups. The workflow in current state-of-the-art software is that the user creates the scene in the first step. After the scene is designed, the user will start the simulation and has to wait till the results are available. This sequential approach has the disadvantage that the user does not get any insight into the changing lighting conditions when he or she edits the environment. Only when the simulation is triggered and finished, the user will get insight how the light is distributed. In this thesis, we will redefine the previously mentioned workflow in a way where the simulation is triggered automatically. As the user plans the scene, the software will visualize the current results of the simulation, and therefore the user will get faster insight into the current lighting

situation. This automatically triggered approach eliminates the computation phase of current state-of-the-art software, where the light planner has to wait for the results. To achieve this automatically triggered approach, we have to speed up the simulation to provide fast and robust results to the user.

We implemented our approach on top of Tamashii [Lip21] which is a scientific rendering framework. Compared to previous scientific work, we use path tracing to compute the illumination, which gives accurate results while still being interactive due to the use of the graphics processing unit (GPU). Utilizing the GPU to evaluate the illuminance on measuring surfaces is the main contribution of our work. Our simulation kernel is based on the rendering equation formulated by Kajiya [Kaj86]. We believe that our approach will revolutionize light planning software and enable the user to plan environments faster and more intuitively.

This thesis is structured in a way that we first guide the reader through the main concepts of lighting simulations. We begin with a brief introduction about light and the most important units related to it. Afterwards we explain how light affects us humans and why lighting planning is important. Then we give a brief introduction on existing lighting regulations and explain which units are used to determine if a room is well illuminated according to the existing regulations. In Chapter 3 we summarize related scientific publications and present current state-of-the-art software. After that, we describe our method in detail, how we represent luminaires and convert the photometric units provided by the luminaire manufacturers to radiometric units. This conversion is important to also simulate indirect lighting in the scene. We then describe how we calculate the illuminance on measuring surfaces and store them conveniently in a texture. This texture is further mapped using a color map to visualize the illuminance on the measuring surfaces in our renderer. The conversion of the illuminance values using a color map helps the lighting planner to get an intuition of the current light distribution in the scene. In Chapter 5 we describe how we implemented our method so that the application uses hardware-accelerated ray tracing to speed up the lighting simulation. Then we go into detail about how we implemented measuring surfaces and how we perform our lighting simulation on the new ray tracing hardware. Afterwards we benchmark our implementation against other state-of-the-art software in terms of computation time and correctness of the computed illuminance values. There we also show that with existing consumer hardware, we were able to present computed illuminance values faster compared to other applications. Furthermore, we created a simple real-world measuring test scene to validate our simulation and also compare it to other state-of-the-art software. In the end we conclude our insights and share ideas for open topics related to this work.

CHAPTER 2

Background

This chapter summarizes the basics of light and its behavior, which is needed for understanding the following methods and literature. Furthermore, we discuss the effects of light on the human body and introduce the reader to lighting-related norms and regulations.

2.1 Light

Visible radiation, also known under the term light, is a small section of the electromagnetic spectrum. The human visual system can perceive electromagnetic radiation that has a wavelength λ between 380 and 760 nanometers [Tov08]. Electromagnetic radiation in this spectrum is registered by photoreceptors (rods and cones) that convert this radiation into electrical activity and is further processed by the human visual system [Sch10]. The visual system is able to adapt between seeing in well-illuminated light environments and environments with dim light conditions and is therefore able to perceive an extensive range of light levels. The adaptation is generally grouped into three modes: photopic vision for well-illuminated conditions, scotopic vision for dim light conditions, and mesopic vision for environments between photopic and scotopic. The fovea is a region with a highly developed visual acuity, which enables us to see sharply and covers about two degrees [Tov08] of the whole visual field of the human eye [Sch10]. Cones enable humans to see different color and are mainly located at the fovea. With the lens, the human eye can gather the incoming light so that it is projected correctly focused onto the fovea. The eye adapts to scotopic vision in environments with poor light conditions, where the more sensitive rods are activated to receive the light. These rods are located outside the fovea and are distributed on the retina and provide monochromatic perception. Approximately the human eye has 6 million cones and 120 million rods [Sch10].

To distinguish between different electromagnetic spectra, the human eye consists of three different types of cones and therefore has a trichromatic system. These receptors are

L-cones (long), M-cones (medium) and S-cones (short). The L-cones mainly contribute to perceiving radiation in the spectrum from about 500 nm to 650 nm. The M-cones perceive radiation from about 450 nm to 600 nm and S-cones perceive radiation from about 400 nm to 500 nm. The incoming radiometric spectrum gets weighted by the three different cone types and is then further processed by the visual system and defines the color which the human sees. Due to this composition of the visual system, it is possible that two different spectra can result in the same perceived color and this phenomenon is called metamerism. The responsiveness of the photoreceptors is also different between individuals and can not be described by a formula but can be standardized by a mean observer. The estimated responsiveness per cone type can be seen in Figure 2.1, proposed by Stockman et al. [SS00].

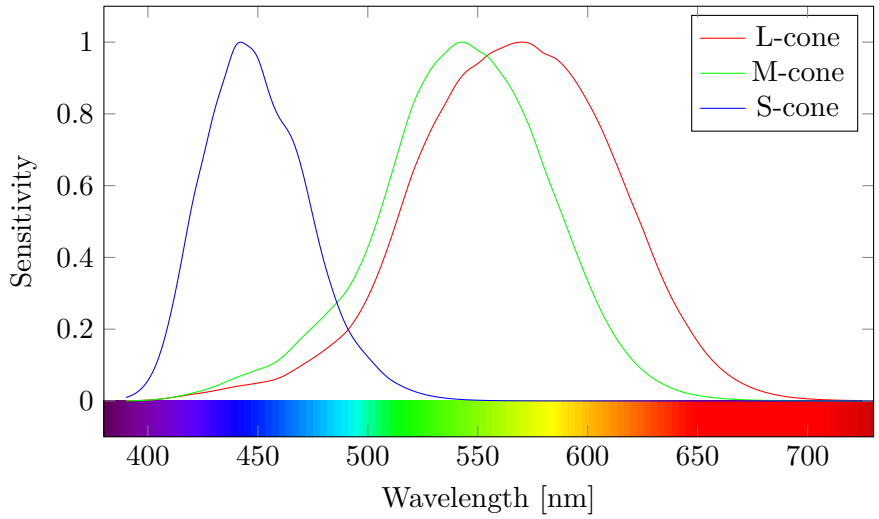


Figure 2.1: The diagram shows the normalized sensitivity per cone type from a two-degree observer according to Stockman et al. [SS00]. When capturing the sensitivity, a two-degree observer refers to an observer, where only the cones located on the fovea are considered in the experiment.

2.1.1 Radiometry and photometry

Radiometry covers various units to measure electromagnetic radiation over the whole spectrum. One unit is radiant flux Φ_e that defines the energy per unit time, which is emitted, transferred or received in form of electromagnetic radiation [BVSoA95]. On the other hand, photometry provides standardized units that cover the perceived brightness according to the human eye defined by a standard observer. The photopic luminosity function $V(\lambda)$, shown in Figure 2.2 defines the brightness for an ideal standard observer defined by the Commission Internationale de L'Eclairage (CIE). This luminosity function is normalized at 555 nm and is used as a weighting function for the spectral radiant flux $\Phi_e(\lambda)$. As scaling, the standard defined that a monochromatic light at 555 nm has a luminous flux of 683 lumens per watt. Many radiometric units have a photometric

counterpart, for example radiant flux Φ_e and luminous flux Φ_v as seen in Table 2.1. The luminous flux can be computed as follows [BVSoA95]:

$$\Phi_v = 683 \int_0^\infty \Phi_e(\lambda) V(\lambda) d\lambda, \tag{2.1}$$

where the whole spectrum gets weighted with the luminosity function and scaled with 683 lumens per watt.

When converting from radiometric units to photometric units, information about the spectrum is lost because the spectrum is weighted with $V(\lambda)$. This loss of information leads to the problem that different spectra (e.g. blue and red light) can lead to identical photometric values. However, the human visual system recognizes them as different colors. Therefore it is crucial that light simulations use radiometric units, because different materials reflect electromagnetic radiation differently depending on the wavelength. Interaction between light and an object must be evaluated in radiometric quantities to give correct results. If the spectrum of a light emitter is not known, a spectrum must be assumed.

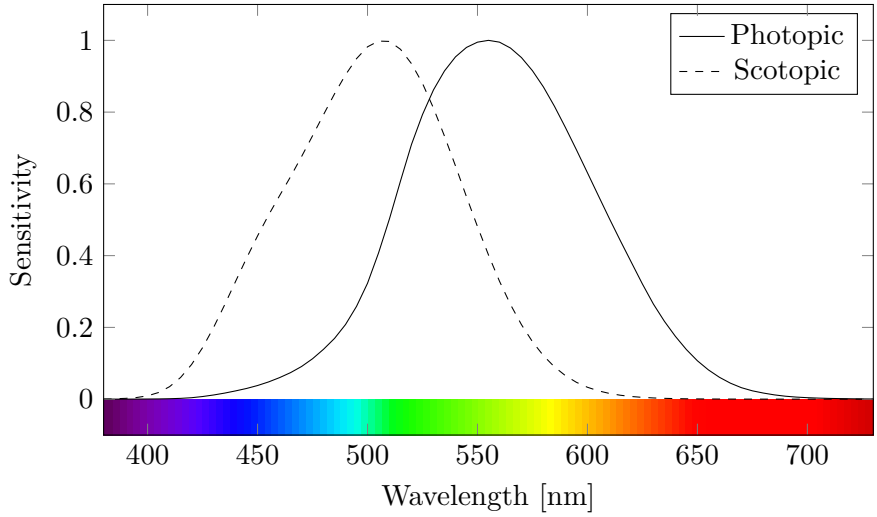


Figure 2.2: The diagram shows two luminous efficiency curves published by the CIE in 1924 (photopic) and 1951 (scotopic). It shows that radiant energy around 555 nm is perceived more bright by the human visual system than from other frequencies. For scotopic vision, this peak is around 505 nm. The curves represent a standard observer and the data is taken from the Colour and Vision Research Laboratory [Uni22].

2.1.2 Radiometric and photometric units

For describing electromagnetic radiation, many units were defined. In this chapter, we will focus on four units that can be found in Table 2.1 which are relevant for understanding light calculations in this thesis. The photometric counterpart of the radiometric unit can

always be calculated by weighting the radiometric quantity with the luminous efficiency function $V(\lambda)$ and multiplied with 683, like done in Equation 2.1.

Radiometric	Symbol	Unit	Photometric	Symbol	Unit
Radiant flux	Φ_e	W	Luminous flux	Φ_v	lm
Radiant intensity	I_e	W/sr	Luminous intensity	I_v	$cd = lm/sr$
Irradiance	E_e	W/m^2	Illuminance	E_v	$lx = lm/m^2$
Radiance	L_e	$W/(m^2 sr)$	Luminance	L_v	cd/m^2

Units: W watt, sr steradiant, m meter, lm lumen, cd candela and lx lux

Table 2.1: The table shows the four radio- and photometric units. To distinguish radiometric from photometric units, it is commonly used to index the symbols with e for energetic and v for visual.

Radiant flux is the radiant energy emitted per unit time. It is suited to describe the overall spectral output of an emitter but does not describe any spatial distribution of this emitted spectrum.

The radiant intensity is suitable to describe how much light is emitted in a specific direction by an object. It is defined as radiant flux passing through a solid angle ω from an infinitesimal emitter and therefore is not suited to describe an area emitter. The solid angle is defined as the area on a unit sphere that is covered from a point p, as seen in Figure 2.3. Luminous intensity is often used by many luminaire definition formats to describe the spatial light distribution of an emitter. This allows manufacturers to adequately describe the luminous output of a luminaire with a small amount of values, but only in a sense of brightness and not in terms of spectral distribution.

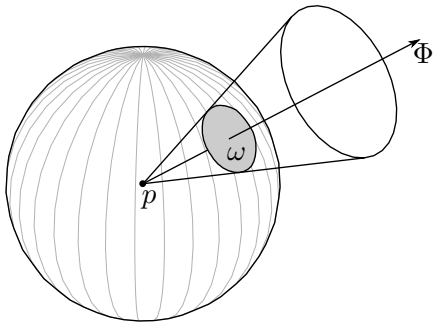


Figure 2.3: Radiant intensity describes the outgoing radiant flux Φ_e for a solid angle ω from an infinitesimal emitter at point a p.

Another significant quantity is irradiance, which describes incident radiant flux received on a surface and it is defined as radiant flux per area. Figure 2.4 visualizes the irradiance. If the area is infinitesimal dA, incoming radiant flux can only be received from a direction that is covered by a hemisphere, assuming that the tangent space is well-defined. Incident light coming from a shallower angle contributes less and light that is more perpendicular

to the surface normal n contributes more. Light that is received over the hemisphere gets projected on the flat surface dA and depends on the dot product of the surface normal n and the direction v of the incoming light. The photometric unit illuminance is suitable to describe how much visible radiation is received on a surface, but does not describe how much light is received by the human eye, because it does not describe how much radiation is reflected from a point of an object to the eye.

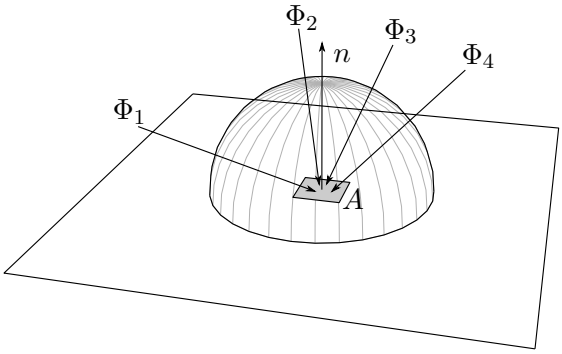


Figure 2.4: Irradiance describes the incoming radiant flux Φ_e on an area A.

Radiance is the last unit that is missing to perform light calculations. In a lighting simulation we have to describe how much light is going from a point on a surface to another point in the scene. Radiant intensity is not suitable for describing the distribution of light in a scene because it is not considering an area where the light is received or emitted. Radiance considers that spatial context and is defined as radiant flux Φ_e, emitted per solid angle ω and per area A, that is projected on the hemisphere, as shown in Figure 2.5.

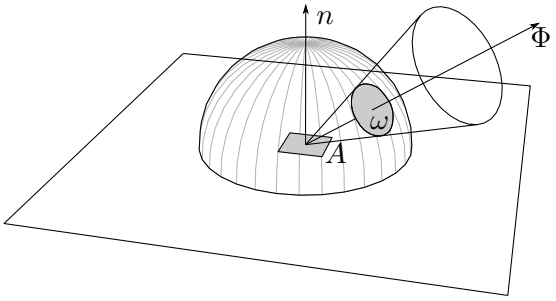


Figure 2.5: Radiance describes the emitted Φ_e per solid angle ω per area A. The area is projected on the hemisphere with the cos factor between the surface normal n and the direction. This cos factor is the dot product of the surface normal and the direction vector.

2.2 Effect of light on the human

The human visual system does not only provide sight. It also regulates the human body and, therefore light also affects our organism. In specialized literature, these two aspects are separated into visual responses and non-visual responses [HE21]. Visual responses account for visual performance and comfort, where glare or too strong light causes discomfort or damage to the visual system. On the other hand, bad lighting conditions can cause non-visual secondary long-term effects like stress, circadian phase shift and sleep quality [HE21]. These non-visual responses are controlled mainly by the photosensitive retinal ganglion cells [Ber02] [Hat02]. Chang et al. showed that light exposure has a direct effect on the human circadian system [CSC11] and therefore good lighting design in indoor environments can have positive effects on the well-being of people. The combined planning process in terms of visual aspect and medical aspect is known under the term human-centric lighting.

For various tasks, different light levels are needed. Precise working on a component in an industrial workshop needs more light than reading a book. Different standards defined light levels for specific tasks, where a minimum required illuminance must be fulfilled in the working environment. The Comité Européen de Normalisation (CEN) defined a norm, the EN 12464-1:2021, that indoor workplaces must fulfill certain light conditions depending on their specific working task. Non-visual responses are not covered in a standard but are defined in a technical rule CEN/TR 16791:2017. Due to costs and efficiency, many lighting installations in working environments fulfill the standard but do not keep in mind biological aspects and therefore play a secondary role in the light planning process. Studies indicate that daylight in classrooms positively affects the performance of students [HWO02] and short-wavelength light can improve the concentration of elementary school children [SMG+12]. Informing stakeholders like employers about the impact of light on the human body, may motivate them to invest in good lighting design for employees to improve their well-being and productivity for the company.

2.3 Regulation

In the European Union there are three standardization organizations, CEN, Comité Européen de Normalisation Électrotechnique (CENELEC) and European Telecommunications Standards Institute (ETSI). Norms and regulation defined by these organizations are created together with a broad range of stakeholders and are a foundation of the shared European market. European norms (EN), which are appointed, must be transposed into a national standard by all members of the European Union. For Austria, this is the Austrian Standard International (ASI) and in Germany, the Deutsches Institut für Normung (DIN).

The CEN defines two norms for lighting in working places, one for indoor EN 12464-1 and one for outdoor EN 12464-2. This work focuses on indoor environments, and therefore it focuses more on the EN 12464-1 standard. On 25/08/2021, a new EN 12464-1:2021

became available and had to be implemented into national standards till 28/02/2022. DIN EN 12464-1:2021 [DIN21] is Germany's latest version of the EN 12464-1:2021 and is used in this thesis as reference.

The EN 12464-1:2021 uses defined grids, where the illuminance will be calculated to determine if a room is well illuminated. The illuminance values must exceed the minimum requirements declared by the norm so that the workspace is compliant. The grids have either a rectangular or cylindric shape, where the illuminance is calculated on discrete points. The rectangular shape is used to measure the incident light on the working spaces, for example, the area on a desk. Additionally, the rectangular shape is also used for measuring ceilings and walls. On the other hand, the cylindric shape is used as an indicator to determine how uniform a room is illuminated. Equal light distribution is essential when persons want to communicate together. Without proper illumination, persons may have difficulties perceiving the facial expression of their conversational partners. Figure 2.6 illustrates the impact of equal light distribution.

Figure 2.6: The figure shows how important equally distributed light is by comparing the same head model [McG17] with two different lighting environments. On the left side, the head is illuminated equally by four lights, whereas on the right side, the head is illuminated by only two lights. Therefore, cylindrical illuminance is an excellent indicator of whether the room is uniformly illuminated.

The norm defines eight values as requirements that need to be fulfilled for indoor workplaces. These values are:

$\bar{\mathbf{E}}_{\mathbf{m,r}}[\mathbf{lux}]$ is the required maintained illuminance, where the illuminance on each point on the working space must not be lower than the given lux value.

$\bar{\mathbf{E}}_{\mathbf{m,m}}[\mathbf{lux}]$ is the modified maintained illuminance, that is required under special context like sparse daylight and working in low-contrast environments.

$\mathbf{U_o}$ is the minimum illuminance uniformity, that has to be fulfilled on the working space.

$\mathbf{R_a}$ is the minimum color rendering index on the measured point on the working space.

$\mathbf{R_{UGL}}$ is the maximum Unified Glare Rating, for an observer.

$\mathbf{\bar{E}_z}$[**lux**] is the minimum cylindrical illuminance, measured on the height of human faces.

$\mathbf{\bar{E}_{m,wall}}$[**lux**] minimum average illuminance on walls.

$\mathbf{\bar{E}_{m,ceiling}}$[**lux**] minimum average illuminance on ceilings.

Table 2.2 shows an extract of the lighting requirements listed in EN 12464-1:2021. For different tasks, different requirements were defined within the norm. For corridors, a moderate illumination of 100 lux is sufficient. Unlike floors or corridors, office rooms need more light to perform office-related work like writing and reading. Precise work, like crafting need even more light and more uniform light distribution.

Ref. no	Type of task	$\bar{E}_{m,r}$	$\bar{E}_{m,m}$	U_o	U_z	$\bar{E}_{m,\text{wall}}$	$\bar{E}_{m,\text{ceiling}}$
34.2	Office - General tasks	500	1000	0.6	150	150	100
9.1	Corridors and circulation areas	100	150	0.4	50	50	30
19.6	Electronic workshops, testing, adjusting	1500	2000	0.7	150	150	100

Table 2.2: Subset of the lighting requirements listed in the EN 12464-1:2021.

2.4 Luminaire definition

To perform a lighting simulation, light sources have to be properly defined. A light emitter can be described by how much radiant flux it emits in a particular direction. For area emitters, this emitted radiant flux must be defined for each point on the surface and therefore results in many data points for the definition of a light source. Luminaire manufacturers prefer defining light sources more compactly by simplifying the light source to a point light source, which has an infinitesimal area. This simplification has the advantage that a point light source can be described by luminous intensity. Common used data formats are IES [PE19] and ELUMDAT [Sto98]. Both formats describe light sources by luminous intensity. IES is the more widely used format and for this thesis, we decided to support IES files. The IES file format stores the luminous intensity in horizontal planes, where each plane defines the luminous intensity for certain vertical angles.

Figure 2.7 shows the common structure of an IES file. The IES file begins with the definition of the standard in the first line, followed by several headers providing general information about the luminaire, like the manufacturer name, luminaire name and many more. After the headers, thirteen values are listed, containing various information, like the number of horizontal and vertical angles. After these thirteen values, the increments in vertical angles are listed followed by the increments of horizontal angles, where the luminous intensity was captured. At the end, all measured data is listed, where each line represents the measured luminous intensity for a specific horizontal angle.

```
1  IESNA:LM-63-1995
2  [TEST] INFINITY LIGHTING PHOTOMETRIC REPORT NO. W00006
3  [LUMINAIRE] ICEAL8-2X150-FL-CG1
4  TILT=NONE
5  2 11000 1 19 5 1 1 1.375 1.04167 0
6  1 1 314
7  0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
8  0 22.5 45 67.5 90
9  9085.62 9033.58 8923.24 8687.43 8145.53 7454.18 6336.9 5050.9 3180.16 2162.21
       1292.02 677.24 243.87 48.44 16.1 6.48 6.25 6.99 3.31
10 9085.62 8902.2 8825.21 8578.38 8462.81 7787.37 6902.89 5633.64 3822.33 2423.91
       1614.81 927.01 382.27 72.18 18.92 8.91 6.41 5.19 2.16
11 9085.62 9029.2 9168.27 8989.93 8818.57 8434.18 7791.45 6806.71 5489.39 3939.15
       2518.39 1573.92 770.93 308.04 81.18 28.92 14.71 8.88 4.75
12 9085.62 9404.94 9244.69 9202.21 9146.45 8779.2 8121.15 7346.47 6513.38 5407.45
       3976.98 2453.4 1106.06 428.6 117.57 35.96 12.34 7.19 1.55
13 9085.62 9315.27 9317.59 9459.68 9433.84 9128.13 8380.85 7752.85 6655.4 5439.5
       3990.21 2472.54 1140.85 377.1 65.79 17.41 9 4.71 2.77
```

Figure 2.7: The content of the comet.ies file [Leo]. Line one specifies the version of the IES file. Lines two and three list the test number and the name of the luminaire. The fourth line marks that the luminous output stays constant when the luminaire gets tilted. Line five till six list various properties, like the size of the luminaire or the amount of horizontal and vertical angles. Line seven lists the vertical increments for each point where the luminous intensity was measured. Line eight lists the horizontal increments for each point where the luminous intensity was measured. From line nine to the end of the file, each line represents the measured luminous intensity per horizontal plane in candela.

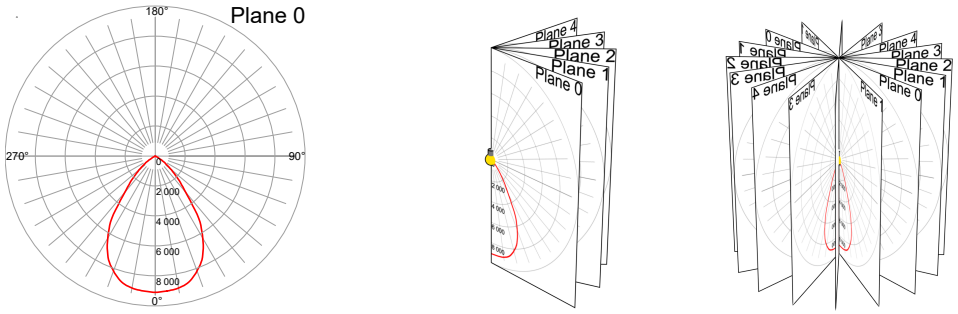


Figure 2.8: On the left side, the diagram shows the luminous intensity distribution curve of the horizontal plane at zero degrees from the comet.ies file [Leo], that is the data on line nine shown in Figure 2.7. The diagram was plotted using IESviewer [And] by Andrey Legotin. The illustration in the middle shows the captured data from the comet.ies file from vertical angle zero to ninety degrees and the five horizontal planes emitted by the light bulb shown in the middle. By definition, the comet.ies file only covers one horizontal quadrant, and the data gets mirrored and repeated to cover the full 360 degrees shown on the right illustration.

The luminous intensity distribution curve is a way to visualize the provided intensity data to understand the emittance of a luminaire. It is often provided by the luminaire manufacturers and shows the luminous intensity emittance per vertical angle. A luminous intensity distribution curve shows the luminous intensity for the full 360 degrees for one horizontal plane and is visualized in Figure 2.8. The luminaire has to be described so that a luminous intensity is defined for each direction. Many light sources have an axisymmetric light emittance and in this case it is possible only to provide data covering 90 degrees in the IES file to fully describe the light source. Missing data gets repeated and mirrored, as shown in Figure 2.8 on the right illustration, so it covers the full 360 degrees.

This chapter summarized the basics of lighting which helps the reader to better understand the upcoming chapters. In the next chapter we will discuss related work to this thesis and current state-of-the-art software for planning indoor lighting environments.

CHAPTER 3

Related Work

Creating lighting simulations is a complex task that requires knowledge of different disciplines, including physics, informatics, and the human anatomy of the visual system. Developing such software requires know-how in the previously mentioned disciplines and writing an application is time-consuming. Interfaces of the program need to be intuitive, and the calculations have to be correct and validated against test cases to prove their correctness. The CIE released 32 test cases [MFA06] that help developers and lighting engineers to identify errors in their application and validate their software against a set of results.

3.1 Scientific

In recent years many different approaches targeting lighting design have been published. One major publication is LiteVis [SOL+16], where the authors published a novel decision-making workflow. The approach minimizes the iteration cycles and enables the lighting designer to explore different lighting setups in a short time period compared to state-of-the-art software. The approach also introduces a new simulation ranking view, where the lighting designer is able to prioritize parameters like uniformity or cost. Depending on these parameters, LiteVis proposes different scenes with different parameters as solutions to the user. The simulation kernel of LiteVis is based on a virtual polygon lights approach proposed by Luksch et al. [LTH+13] which enables fast computation times and provides the user with fast results. Compared to our approach, we use path tracing to compute the illuminance.

To determine if a room is well illuminated out of a rendered two-dimensional image is difficult. Providing a virtual environment where the user can explore the scene interactivly will help the user decide if the lighting situation fits the scene. Natephra et al. [NMFY17] proposed an approach where they use virtual reality technology to explore virtual scenes to give the viewer a realistic feel about the current lighting situation. Users

can change the scene interactively and plan different lighting setups within the virtual reality environment.

Lighting design can be seen as an inverse problem. In our case we want to achieve an equal light distribution on a given surface and compute the position and direction of the luminaires to achieve this given solution. Finding the solution can be seen as an optimization problem and is known under the term inverse rendering. Several papers have been published over the last years to compute scene parameters from a set of requirements. Gkaravelis and Papaioannou [GP18] proposed a solution to automatically find an optimized luminaire installation based on local contrast optimization on the target object. One approach for generating optimal lighting layouts for indoor scenes was proposed by Jin and Lee [JL19]. Based on specific objective terms, including the target illuminance on a given surface, the approach is able to optimize the position, direction and light intensity of multiple luminaires within a scene.

To simulate light and more generally electromagnetic radiation, we have to approximate all phenomena that influence the distribution of radiation in space. A subset of these phenomena are reflectance, transmission, absorption, fluorescence, polarization, subsurface scattering, etc. In general, the radiative transport equation describes how particles propagate through a medium [LRK17] covering different phenomena. To cover all these different effects will result in an immense computational workload and is not practical for computer computation. To still perform a lighting simulation in a reasonable time, James T. Kajiya proposed the rendering equation [Kaj86], which describes the propagation of light in a convenient way that is well suited for computer graphics. This rendering equation describes the outgoing radiance L_o at a specific direction ω_o on a specific point p and is described as follows:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{S^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \Theta_i| d\omega_i. \tag{3.1}$$

Equation 3.1 is a slightly modified version of the original rendering equation [PJH16]. The notation L_e defines the emitted radiance and the integral over the hemisphere S^2 describes the outgoing radiance from all incoming radiance L_i. The function f is the bidirectional reflectance distribution function (BRDF) that defines how much of the incoming radiance is reflected towards a specific direction, it encodes the material's properties at point p. To solve the equation for a point p, we must consider all possible incident light paths. Finding an analytical solution for this equation is impossible due to its recursive nature. Imagine a light ray that can bounce infinite times, which causes infinite calculations. However, with each bounce, the light ray's transported energy will decrease and converge to zero and therefore only considering the first few bounces will be sufficient for our light simulation. One way to solve the equation is to use Monte Carlo Integration to estimate the integral over the hemisphere S^2. The concept of Monte Carlo Integration is to take a finite amount of samples to estimate the real solution of the integral. Estimating an integral using Monte Carlo Integration can be achieved as follows:

$$\int_a^b f(x)dx \approx \frac{1}{N}\sum_{n=1}^{N}\frac{f(X_n)}{p(X_n)}, \tag{3.2}$$

where N corresponds to the number of samples, X_i the random variable and the $p(X_i)$ the probability distribution function, how the samples were chosen. We do not go into detail and refer the reader to Veach's PhD thesis [Vea98] and Physically Based Rendering book [PJH16], which discuss Monte Carlo Integration in detail.

As described in Section 2.1 light can be described as a spectrum and is used in the lighting simulation to describe the light source. Unfortunately many Luminaire manufacturers often do not provide detailed information about their luminaire. In many cases, they only provide the luminous intensity and the color temperature, but detailed information about the emitted spectrum is not included. The emitted spectrum by a luminaire can be very different depending on the luminaire type (LED, incandescent or fluorescent lamps). With unknown spectral distribution, it is not possible to simulate the interaction between the light and the surface of an object. For example, when a red object gets illuminated by an equally distributed white light source, only radiation with a longer wavelength in the visible spectrum gets reflected from the object and this reflected light is perceived by the human eye as the color red. So it is crucial to perform the light simulation in radiometric units to get plausible results and consider indirect lighting.

In our approach we use the correlated color temperature (CCT) [Buk19], which is often listed by the manufacturers to assume the emitted spectrum of the luminaire. The CCT corresponds to the spectrum that is emitted by an ideal black-body radiator at a specific temperature. Figure 3.1 shows five spectral distributions, emitted by an ideal black-body radiator. Planck's law describes this behavior.

Standard renderers in computer graphics use three discrete values to represent this spectrum. These three values describe the color of the light and are mostly represent the colors red, green and blue. Representing colors using three values is fast to compute and is, in most cases, enough to generate realistic images to approximate real-world phenomena. Unfortunately, not all real-world phenomena can be computed using three values. One example is the dispersion of light. Dispersion refers to the phenomenon where the light gets refracted differently depending on its frequency. To simulate dispersion, the renderer has to perform the calculation on the whole electromagnetic spectrum. Renderers that perform the calculations based on a spectrum are called spectral renderers. Due to the complexity, spectral renderers are slower but more accurate than traditional tristimulus renderers. Another downside of spectral renderers is that the reflectance of materials in the scene needs to be defined in spectral domain. Most materials used in computer graphics are given in the sRGB color space and this color space needs to be converted into a spectral domain, one way to tackle this was presented by Jakob and Hanika [JH19].

In the upcoming section we present state-of-the-art software which is currently used by lighting designer and describe how the designers use these applications.

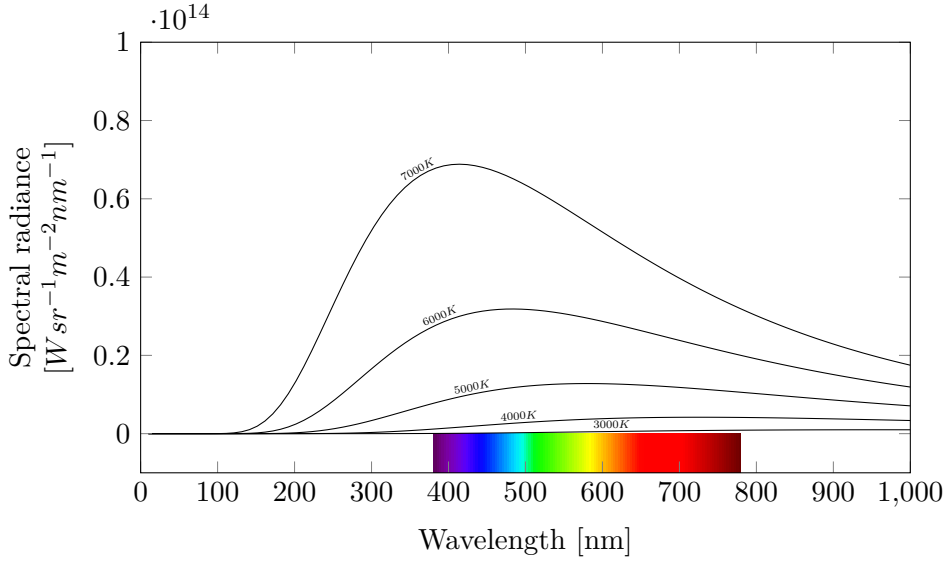


Figure 3.1: The diagram shows the five different spectra emitted by an ideal black-body radiator with temperatures 3000K, 4000K, 5000K, 6000K and 7000K. At lower temperatures, the emitted spectrum is mostly distributed in the section where the light appears red to the human visual system. With higher temperatures, the spectral distribution shifts towards higher frequencies, which appears more blue to the human visual system.

3.2 Industry

As described earlier, building a lighting simulation software requires a lot of work and knowledge and therefore not many applications are available on the market. There are mainly three applications that are widely used by planners to simulate their lighting plans against different norms and requirements.

AGi32 [Lig] by Lighting Analysts is a fee-based software other than DIALux [DIA] and Relux [Rel], which are free and funded mainly by paid courses or luminaire manufacturers. All three applications support planning indoor, outdoor and street lighting environments. In our work, we use Relux and DIALux for comparison to our application, since both are developed in Europe and therefore mainly support European standards, whereas AGi32 is more focused on the US market. Both Relux and DIALux can test the calculation results against the European norms, namely EN 12464-1, EN 12464-2 and EN 13201.

The target audience are light designers and therefore, the interface is designed to assist them in planning and validating the lighting system. Due to the close relationship between the creator of the software and luminaire companies, adding specific luminaires from a manufacturer can be quickly done by selecting them via an online catalog. It is also possible to add luminaire data by providing them in an IES file [PE19] or LDT file [Sto98]. The regular procedure is to define the room dimensions first, followed by adding

windows, doors, and furniture. Then measuring surfaces are added to the scene, which define the surface where the illuminance will be computed. Relux and DIALux support various shapes for measuring surfaces including rectangular and cylindrical surfaces. The results on the measuring surfaces depend on the defined light sources of the scene. When the scene description is finished, the user can start the lighting simulation and needs to wait for the results. This procedure is an iterative process, shown in Figure 3.2, where the user has to wait before he/she knows if the result is compliant. Furthermore, while the calculation is running, no other work can be done and the user does not know if the light is nearly compliant with the norms or completely wrong. Replacing the manually triggered simulation with a continuous simulation, where the ongoing results are visualized will give the user faster insight into the calculation of the resulting illuminance values. The faster insight into the data will help the user because it reduces the iteration time and the user will see how the values change when he or she moves a luminaire. When changing scene data, like moving a luminaire, the simulation must be triggered automatically to restart the whole calculation process.

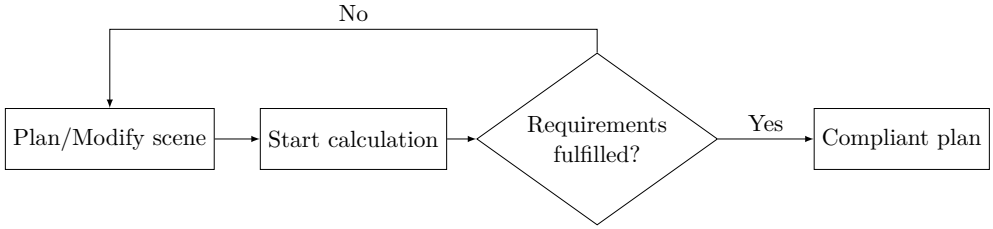


Figure 3.2: The flowchart shows the workflow in Relux and DIALux. First, the user plans and creates the scene, including the luminaires information. Then the calculation is triggered by the user. The calculation time of the simulation heavily depends on the scene complexity. While the simulation is running, the user is not able to change the scene. Depending on the result, the scene is compliant with the norm, or the user has to modify the scene so that the scene becomes compliant.

To achieve automatically triggered simulations while interactively changing the scene, fast computation is crucial to provide the user with early insights and decide if the placement of the luminaires satisfies the norms. With the announcement of Nvidia's Turing architecture at SIGGRAPH 2018 [Cau18], this new architecture of GPU opens up the ability of hardware-accelerated ray tracing to speed up the calculation process in a way that real-time ray tracing becomes possible. Two years later, AMD announced the RX 6000 series [Har20], which is the first generation of AMD's GPU architecture that supports hardware-accelerated ray tracing. The increasing support of ray tracing among GPU manufacturers will affect many ray-trace-based applications by decreasing calculation times and the workflow between the user and the application.

To our knowledge, Relux and DIALux perform their lighting simulation on the central processing unit (CPU), which is not well suited for ray-trace calculations and therefore increases the calculation time but allows the software to run on many different devices. With the increasing amount of hardware that supports hardware-accelerated ray-tracing,

both companies may likely implement their algorithms to be executed on the GPU. Implementing this execution on the GPU in the current version will affect a major part of the simulation codebase. Refactoring the codebase requires a lot of time and domain knowledge, but will improve calculations times to increase the productivity of the lighting planner.

In this chapter we gave an overview about related lighting planning contributions and in the upcoming chapter we will introduce our simulation approach.

CHAPTER 4

Method

In this chapter, we will introduce the reader to the main concepts of this work. As described in Chapter 2, performing lighting simulation is a complex task and there are many ways to calculate light distribution in space. To compute the illuminance on a specific point, we have to consider all incident light paths. Calculating all these infinite paths with a computer will result in infinite calculation time. One way to solve the problem with considering infinite paths is to approximate the result by taking a finite number of samples. The result will come closer to the real solution of considering all paths with increasing sample count. One famous approach for this is Monte Carlo Sampling, which was covered in Section 3.1.

This thesis began with the question, is there a way or factor how to determine if a room is well illuminated. Finding an answer to this question is not an easy task because it heavily depends on the person's activity. For instance, working on a device where a person has to assemble tiny components requires a brightly illuminated environment to perform the task properly. On the other hand, having a conversation with another person does not require as much light as in the example explained before. However, it requires a uniform light distribution, as visualized in Figure 2.6. Current state-of-the-art software like DIALux [DIA] and Relux [Rel] use different norms to clarify if the room has a suitable lighting environment. For indoor working places, the European Union defined the EN 12464-1:2021 norm. We use this norm in our work as guidance to determine if a room is well illuminated.

As mentioned in Section 3.2 we want to redefine the workflow of light planning software with utilizing hardware-accelerated ray tracing to achieve faster computation. Figure 4.1 shows the redefined workflow, where the manually triggered start of the simulation is removed. The simulation is running in parallel and is restarted if necessary. A restart is necessary when the user for example changes the position of a luminaire. We anticipate that this approach will help the user to design and plan lighting environments faster than with the previous workflow.

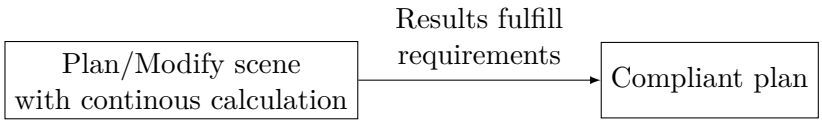


Figure 4.1: Compared to the workflow visualized Figure 3.2, the user plans and modifies the scene. Parallel to the planning phase, the simulation is running parallel and the ongoing results are presented to the user. Displaying current results gives the user faster insight into the computed values and the user can see if the results fulfill the predefined requirements.

Our work covers all values from the EN 12464-1:2021 mentioned in Section 2.3, except for the R_{UGL} and R_a, because for the R_{UGL}, concrete information of luminaire is required. To calculate the R_{UGL} for an observer, we have to calculate the solid angle, which the luminaire covers for the given observer. We only consider luminaire as point light sources and therefore do not consider the Unified Glare Rating in our work. For the R_a value, the emitted radiometric spectrum must be given, to calculate R_a correctly, which is not always provided by the luminaire manufacturers.

4.1 Measuring surfaces

Measuring surfaces are an essential part of our light simulation and need to be integrated and placed into the scene. These measuring surfaces consist of a finite amount of measuring points where the illuminance will be calculated. The calculated illuminance by the simulation needs to fulfill the minimum requirements for the given type of task, defined by the norm.

The basic shapes that are used in the simulation have either rectangular or cylindric shapes. In computer graphics, there are different ways to define three-dimensional shapes. The most common representation is the polygon mesh representation, where a finite set of faces defines the boundary of an object. Faces are defined by grouping a finite set of vertices. The triangle shape is most commonly used to represent faces in computer graphics. Another way to define the shape of an object is to use a mathematical expression or use point clouds as the representation of an object. Point clouds are using only vertices to represent the shape of an object.

For our approach, we decided to use polygon meshes to define the shape of measuring surfaces. This representation has the advantage that many three-dimensional content-creation programs work on mesh-based data and therefore adding measuring surfaces to existing scenes is straightforward. To store the calculated illuminance on a point on the measuring surface, there are two approaches.

One approach which came to our mind is to use vertices that define a mesh can also be used as measuring points, where the illuminance will be calculated. Storing the illuminance in the vertex data has the advantage that it works for arbitrary shapes, but

the number of measuring points is coupled with the number of vertices. When more measuring points are needed, the mesh needs to be subdivided to increase the number of vertices, resulting in higher shape complexity, which will increase the rendering time.

The other approach is to use a texture to store the values and use the texel's sample point as the measuring point's location. Texture mapping is a widely used method in computer graphics to enhance the surface with details by using a two-dimensional texture that gets mapped onto the surface of an object. Using a texture for storing the illuminance comes with the advantage that the amount of measuring points is independent of the number of vertices and is only defined by the size of the texture. The problem with the texture method is that the measuring points are now defined in the two-dimensional texture space, but we need a point in a three-dimensional space for our measuring points. One restriction is that every sample point of the texture has a unique position on the shape of the object. Informally, no texel of the texture is located twice on the surface. Texture mapping uses texture coordinates, that are stored in the vertex data to define the mapping from the three-dimensional shape to the two-dimensional texture space. For the texture approach, we need to define a method to transfer the measuring points from texture space to the three-dimensional space. One naive way is to select a sample point of the texture and test it against all surface primitives to find the primitive on which the point lies. This approach will become impractical for complex meshes with many face primitives because we have to test every surface primitive. Fortunately, the defined measuring surfaces can be expressed by a formula that maps points from texture space to the three-dimensional space. Therefore we have chosen the texture method for our approach because it allows the user to vary the amount of measuring points by simply increasing or decreasing the texture size.

Every measuring surface in our simulation holds a two-dimensional texture where each texel should hold the calculated illuminance value for the corresponding measuring point. Figure 4.2 shows a two-dimensional four by four texture where each data point $d_{i,j}$ is indexed with i and j. In general, the texture data gets uniformly mapped to texture space so that all sample points lie between zero and one, and each data point represents a particular area, called texel. The location of the sample point $\vec{v} \in \mathbb{R}^2$ for the texel is located in the center of the texel as visualized in Figure 4.2 and is defined as:

$$\vec{v}_{i,j} = (\frac{i + 0.5}{s_1}, \frac{j + 0.5}{s_2}), \tag{4.1}$$

where $\vec{s} = (s_1, s_2) \in \mathbb{N}_0^2$ is holding the amount of sample points for both dimensions.

When the texture gets mapped onto a three-dimensional object, the location of the sample point is usually defined by its uv-coordinates, stored in the vertex data. For the two cases of rectangular and cylindric surfaces, the texture can be mapped uniformly to fill up the whole texture space, from zero to one and the position of the sample point can be expressed in closed form as formulated in Equation 4.2 and 4.3.

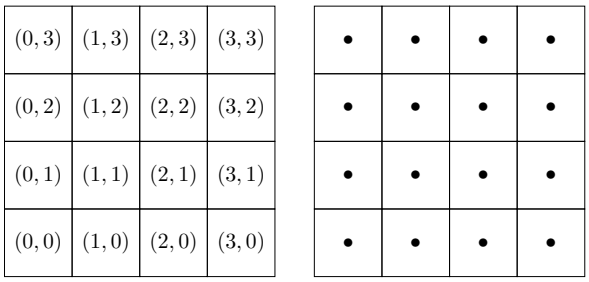


Figure 4.2: A four-by-four two-dimensional texture, which consists of sixteen data values $d_{i,j}$ and its indices on the left. The sample points $\vec{v}$ where the illuminance is calculated are uniformly distributed and are visualized by a dot on the right side.

To calculate the position of each measuring point in three-dimensional space, we have to formulate a mathematical expression from the given mesh data in the scene. With the mathematical definition of the shape, it is easier to transform all the sample points into the three-dimensional space where the illuminance is calculated. Commonly, a rectangle can be defined by two triangles and is visualized in Figure 4.3 on the left side. Our approach is to define a rectangle analytically. We do this by defining a point in space and using two direction vectors to span the surface of the rectangle. Based on three vectors, the position of all measuring points $m_{i,j}$ can be calculated as follows:

$$m_{i,j} = p_{00} + \frac{p_{10} - p_{00}}{s_1} * (i + 0.5) + \frac{p_{01} - p_{00}}{s_2} * (j + 0.5), \tag{4.2}$$

where p_{00} defines the position of the vertex with uv-coordinates $(0, 0)$, p_{10} with uv-coordinates $(1, 0)$ and p_{01} with uv-coordinates $(0, 1)$. The position of these three vertices must be extracted out of the given mesh data of the measuring surface within the scene definition.

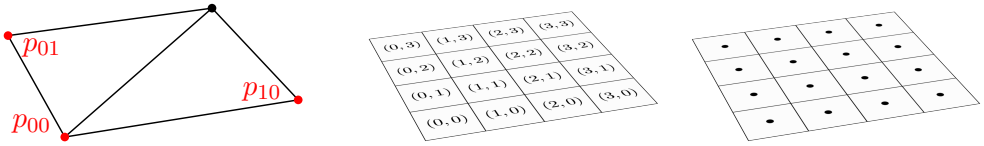


Figure 4.3: All three illustrations show the same rectangle, with different annotations. On the left, the mesh representation is shown, where the four vertices and two faces form the shape of the rectangle. Using the position of the three red vertices, we can compute the location of each measuring surface using Equation 4.2. The illustration in the middle shows the rectangle, divided into sixteen sub-areas, indexed by i and j, where each area shows the size of the texel. The right illustration shows the location of the defined measuring points $m_{i,j}$.

A cylinder consists of more vertices and faces than a rectangle, and it can be closed, where the top and bottom are covered with faces, or open, where no faces cover the top

and bottom of the cylinder. The cylindrical illuminance $\bar{E}_z$[lux] is defined as an open cylinder, and the boundary of the cylinder can be uniformly mapped to a two-dimensional texture that fills up the whole texture space. In general the shape of an open cylinder can be expressed by a mesh where all vertices lie either on the top plane or on the bottom plane. We describe the cylinder analytically, but with a restriction that the mesh of the cylinder has to be aligned so that the line between p_t and p_b as shown in Figure 4.4, lies on the y-axis of the local coordinate system. The point p_t is the mean of all vertices on the top side and p_b the mean of all vertices on the bottom side. With p_t, p_b and previous mentioned restriction the measuring points $m_{i,j}$ can be calculated as follows:

$$m_{i,j} = \begin{pmatrix} r\sin(\frac{2\pi(i+0.5)}{s_1}) \\ p_t \\ r\cos(\frac{2\pi(i+0.5)}{s_1}) \end{pmatrix} + \frac{p_b - p_t}{s_2} * (j + 0.5), \tag{4.3}$$

where r is the radius and can be calculated by computing the distance between a top vertex and p_t.

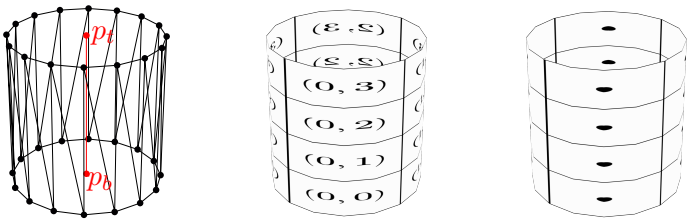


Figure 4.4: All three illustrations show the same open cylinder with different annotations. On the left, the mesh representation showed, where mesh vertices are marked as black dots and the calculated vertices p_t and p_b are marked as red dots. The illustration in the middle shows the cylinder, divided into sixteen texel areas indexed by $d_{i,j}$, where each area shows the size of the texel that is mapped onto the surface of the cylinder. The right illustration shows all the defined measuring points $m_{i,j}$, that can be calculated by formula 4.3.

The calculated measuring points $m_{i,j}$ in the equations 4.2 and 4.3 are in local space of the object. To get the world space position $m_{i,j}$, we have to transform the points from local space to world space by multiplying the vertices with the model matrix. With the defined and calculated measuring points in world space, we know where to compute the illuminance for our simulation.

4.2 Luminaires

As mentioned in Section 2.4, we use the IES file format [PE19] to describe luminaires in our simulation. By having luminous intensity information for all directions, we need a convenient way to query the data and interpolate between data points. Our approach to

conveniently query and interpolate the luminous intensity information is to store the data points into a two-dimensional texture, where the texels hold the luminous intensity. To get the luminous intensity we convert the direction vector we want to query into spherical coordinates and use the azimuthal angle θ and polar angle ϕ to query the texture. With this representation we cover the whole sphere by two parameter $\theta \in [0, 2\pi]$ and $\phi \in [0, \pi]$. By normalizing the range of both angles, we can use both values as texture coordinates u for ϕ and v for θ and these texture coordinates can be computed as follows:

$$(u, v) = (\frac{\phi}{\pi} + \frac{0.5}{s_1}, \frac{\theta}{2\pi} + \frac{0.5}{s_2}). \tag{4.4}$$

Values along the u-axis cover the vertical angles of the provided luminous data and the v-axis all horizontal angles. The advantage of storing the information in a texture is that we can easily query the data and also use hardware-accelerated bilinear interpolation of the GPU.

Figure 4.5 visualizes the concept of querying the luminous intensity data using a texture. When sampling a light direction in world space, the direction needs to be converted in the local coordinate system of the light, which is illustrated in Figure 4.5 on the left. On the right side of Figure 4.5 the luminous intensity is visualized as red color and we can see that most of the light is emitted in the z-direction of the local coordinate system of the luminaire.

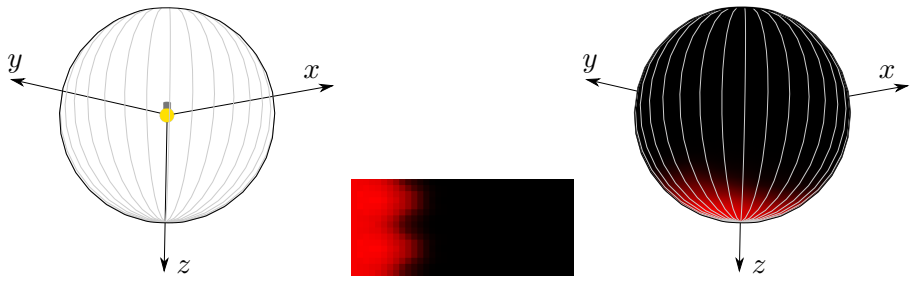


Figure 4.5: The left illustration shows the comet.ies luminaire visualized as a yellow light bulb and its local coordinate system. In the middle, the luminous data is shown as texture, where the texture covers all possible directions of the sphere. The texture is visualized so that all luminous data is normalized and rendered as red color, meaning that a more reddish color indicates higher luminous intensity in the given direction. The mapping from the two-dimensional texture to the three-dimensional sphere is shown on the right side.

4.3 Photometric conversion

As stated in Section 3.1 we use for our approach the CCT to assume the spectral distribution of the light source. It is crucial to convert the given photometric units

provided by the luminaire manufacturers, because otherwise considering indirect lighting would not be possible. To simulate indirect lighting, the interaction of the light with the material must be considered depending on the given light spectrum and the reflectance of the material.

We decided to perform the simulation using tristimulus colors. The reason for this is that the computation is faster and therefore enables faster insight into the computation results. When a lighting planner is designing a room, it is more important that the planner gets fast insight into how the light is distributed compared to truly correct results.

The luminous intensity provided by the IES files is given as $I_v = d\Phi_v/d\Omega$. With a given spectrum, the luminous intensity can be computed as follows [BVSoA95]:

$$I_v = 683 \int_0^\infty V(\lambda) I_e(\lambda) d\lambda. \tag{4.5}$$

For a monochromatic light this equation simplifies to $I_v = 683 V(\lambda) I_e(\lambda)$. Our approach is to convert the luminous intensity to radiant intensity to sample Planck's law at specific wavelengths and weight these sample points accordingly in way that the calculated radiant intensity results in the same luminous intensity. Each sample point represents a monochromatic peak that defines the radiant intensity at a specific wavelength. We use the Dirac delta distribution $\delta(\lambda)$ to sample Planck's law $f(\lambda, T)$ [KM09]:

$$\int_{-\infty}^{\infty} f(\lambda, T)\delta(\lambda - a)d\lambda = \int_{-\infty}^{\infty} \frac{2hc^2}{\lambda^5} \frac{1}{e^{\frac{hc}{\lambda kT}} - 1} \delta(\lambda - a)d\lambda = f(a, T), \tag{4.6}$$

where T represents the absolute temperature in Kelvin, k the Boltzmann constant, h the Planck constant, c the speed of light and a the wavelength where the function is sampled. The sampled points from $f(a, T)$ are stored in $w \in \mathbb{R}^n$, where n is the number of sample points, we take. To use the sample points as weights we have to normalize the sample points as follows:

$$\hat{w} = \frac{w}{\sum_{i=1}^{n} w_i}, \tag{4.7}$$

where $\hat{w} \in \mathbb{R}^n$ is the vector holding the normalized sample point values and expresses the distribution of the monochromatic lights. In our approach we use monochromatic light, therefore we can rewrite Equation 4.5 in following way:

$$I_v = 683 \sum_{i=1}^{n} V(\lambda_n) I_e(\lambda_n), \tag{4.8}$$

where λ_n is the specific wavelength of a monochromatic light. The sum can be rewritten in a dot product:

$$I_v = \langle c, I_e \rangle, \tag{4.9}$$

where $c \in \mathbb{R}^n$ is $683\lambda_n$ for each monochromatic light. The radiant intensity I_e can be computed as follows:

$$I_e = k\hat{w}, \tag{4.10}$$

where k is a scalar. Combining Equation 4.9 and 4.10 we can compute k as follows:

$$k = \frac{I_v}{\langle c, \hat{w} \rangle}. \tag{4.11}$$

With known factor k we can compute the radiant intensity for each wavelength using Equation 4.10. Now that we can convert the luminous intensity to radiant intensity, we can simulate the distribution of light in a scene and take into account that light is reflected differently depending on the material of an object. One missing part is which wavelengths should be chosen to represent the light.

Most materials and lights used in computer graphics are represented using three values. With our defined Equation 4.6, we need to define three wavelengths, which correspond to the RGB values used by our renderer. To find the corresponding wavelengths, we use the CIE chromaticity diagram and calculate the three dominant wavelengths of the sRGB gamut. The gamut of a color space defines all the possible colors that can be represented in this color space. All colors on the boundary of the chromaticity diagram as visualized in Figure 4.6, excluding the colors lying on the straight light on the bottom, are called pure colors. The straight line on the bottom is called the purple boundary and these colors can only be expressed by a mixture of two pure colors. All pure colors can be represented by monochromatic light at a specific wavelength. For a specific color, the dominant wavelength is the pure color, which gets intersected by a line which intersects the specific color, the white point and the pure color. The gamut of the sRGB color space [Int99] is visualized in Figure 4.6.

The dominant wavelengths of the sRGB color space are 611 nm (λ_R), 549.0 nm (λ_G) and 464.0 nm (λ_B). Using these three wavelengths, we can transform the luminous intensity into three monochromatic peaks as follows:

$$\begin{pmatrix} I_{e,R} \\ I_{e,G} \\ I_{e,B} \end{pmatrix} = \begin{pmatrix} k\hat{w}_1 \\ k\hat{w}_2 \\ k\hat{w}_3 \end{pmatrix}. \tag{4.12}$$

With these three intensity values, we have all the information that is needed to perform our lighting simulation. In the next section, we will introduce the reader how we simulate light in a three-dimensional scene and estimate the illuminance using a discrete amount of samples per measuring point.

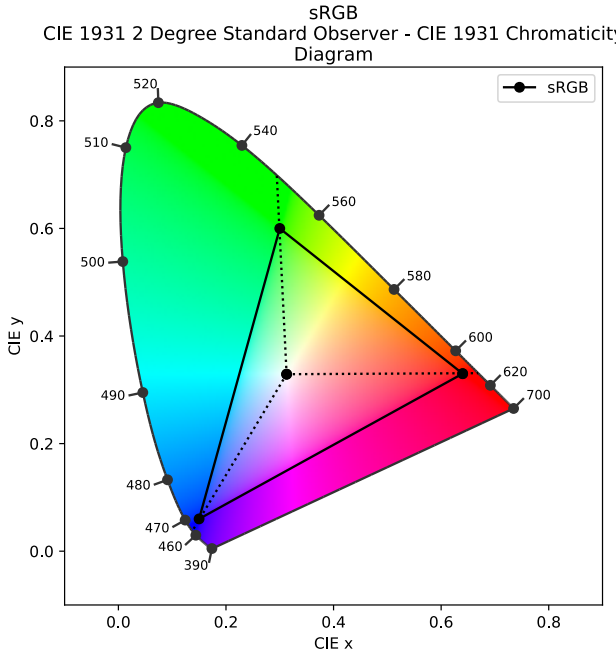


Figure 4.6: The diagram shows the CIE chromaticity diagram, including the gamut of the sRGB color space, which is formed by three different colors. At the diagram's boundary, the dominant wavelengths are crossed by one of the three dotted curves formed by one of the three colors. The point in the middle of the triangle represents the white point. The plot was created using the Colour python library [MMP+22].

4.4 Simulation

In this section we present our lighting simulation based on the rendering equation described in Section 3.1, i.e., how we calculate the illuminance on the given measuring surfaces. In the real world, light is emitted by a light source and is received by our eyes or generally speaking by a sensor. Simulating light this way is known as forward tracing. It is also possible to trace light from the sensor back to the light source and this approach is known under the term backward tracing. For our approach, we use backward tracing because we want to compute the illuminance on an infinitesimal measuring point, which is our sensor. Backward tracing is better suited when starting from a sensor compared to forward tracing, because when using forward tracing it is unlikely that the light rays hit the measuring points. To calculate the irradiance E_e, we approximate the integral over the whole hemisphere S^2 of all possible incident directions:

$$E_e = \int_{S^2} L_{i,e}(m_{i,j}, \omega_i) \cos\theta dS^2. \tag{4.13}$$

To solve the integral using Monte Carlo Integration we rewrite Equation 4.13 using Equation 3.2 which results in:

$$E_e \approx \frac{1}{N} \sum_{n=1}^{N} \frac{L_{i,e}(m_{i,j}, \omega_{i,X_n}) \cos\theta}{p(X_n)}, \tag{4.14}$$

where ω_{i,X_n} is a randomly selected direction from point $m_{i,j}$ and the $p(X_n)$ the probability density of selecting this direction. The incoming radiance $L_{i,e}(m_{i,j}, \omega_{i,X_n})$ is the light coming from a direction ω_{i,X_n} to a point $m_{i,j}$ and can be expressed by the rendering equation 3.1. The recursive substitution of the rendering equation results in infinite bounces, but with each bounce, the light will contribute less to the resulting irradiance. So after some bounces, we can stop this recursion. Another problem is that point light sources will never be hit by a ray because by definition they are infinitesimally small. To tackle this, we have to sample the light directly at each bounce, where we evaluate the contribution of this bounce on the corresponding measuring point. Figure 4.7 visualizes the procedure of sampling a point light directly in a two-dimensional space.

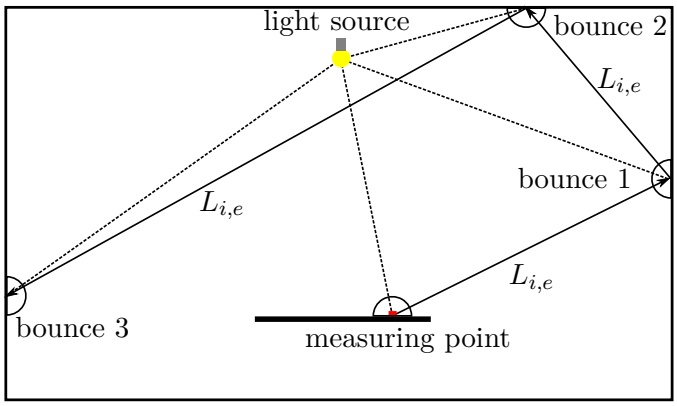


Figure 4.7: The red area visualizes the area where we want to calculate the illuminance. Outgoing from this measuring point we trace multiple randomly selected directions. For simplicity, the figure shows one random direction outgoing from the measuring point. Then from this first bounce, we again select a random direction, hitting a point on the ceiling. With each bounce, the impact on the illuminance on the measuring point will be more negligible. We sample the light directly on each hit point, which is visualized by the dashed lines.

Algorithm 4.1 describes how we evaluate the irradiance on a measuring point. For simplicity, we assumed that the BRDF for all surfaces is π^{-1} for all directions, which is also known as the Lambertian model. It is possible to replace the BRDF with a more complex one that approximates materials better, but will increase the computation time. To calculate the illuminance on a point $m_{i,j}$, we need its position, the normal vector at

$m_{i,j}$, the scene data, the number of bounces we want to compute and the number of samples. Algorithm 4.1 uses the function `sampleLights`, described in Algorithm 4.2 to evaluate how much light from this specific bounce contributes to the measuring point. Before we explain the `sampleLights` function, we introduce the concept of throughput to the reader. Tracing a ray in the scene that bounces multiple times on surfaces, we have to evaluate the outgoing radiance $L_{o,e}$ from the incoming radiance $L_{i,e}$. Evaluating the outgoing radiance on each bounce results in a recursion, where we call the same function multiple times. Recursions come with the problem that with each call, the program has to create a new stack when entering a function. The problem with that is that it slows down the process and another issue and more important is that recursion are not supported by many GPUs. Luckily, the recursion is a tail-recursion that can be transformed into a simple loop. To convert the tail-recursion, we have to track how much light is reflected to the starting point on each bounce. With each bounce, the received light that contributes from this bounce will be less and this contribution is influenced by the BRDF, color of the surface and the direction of the ray. Storing this information is also known under the term throughput.

The function `sampleLights` takes as input the position where the irradiance is evaluated, the normal at this position, the BRDF and the throughput. To evaluate E_e we loop over all light sources l_i and check if l_i is visible from $m_{i,j}$. If so, we query the luminous intensity I_v from the texture where we store the data points from the IES file and convert it accordingly as described in Section 4.3. The I_e is then used to evaluate the irradiance E_e on the given measuring point.

Line one of Algorithm 4.1 evaluates the direct light contribution, which can be evaluated without using Monte Carlo Integration. After evaluating the direct light contribution, we use Monte Carlo Integration to evaluate the indirect light contribution using N samples. The starting point of our ray is the measuring point $m_{i,j}$. On $m_{i,j}$, the light can come from all directions which are covered by a hemisphere, so we have to select a random direction that points to a location on the surface of the hemisphere. We use a uniform distribution to select a random direction and the probability density of selecting this direction is $\frac{1}{2\pi}$. The throughput factor gets initialized with the dot product of the surface normal of $m_{i,j}$ and the randomly selected direction of the ray, divided by the probability density of taking this direction. After that we trace a ray using a loop, the iteration count of the loop is bounded to the number of bounces D we want to compute.

We first trace the ray in the scene using the `traceRay` function, which returns the information t about the intersection between the ray and an object. If the ray hits an object, we calculate how much light is received at that point. This received light consists of the direct light that is received on that point and the indirect light from other surfaces. To calculate the direct light, we again use the `sampleLights` function. For the indirect contribution, we again choose a random direction and trace a ray, where the starting point is the hit point of the current bounce. Finally, we update the throughput accordingly by multiplying it with the previous throughput value.

Summing up all E_e from all samples N and dividing it by the number of samples N,

will give the approximated indirect irradiance $E_{e,indirect}$. In the last step we convert the computed irradiance to illuminance using the predefined monochromatic wavelengths we defined in Section 4.3. The `radiometricToPhotometric` function is listed in Algorithm 4.2.

Algorithm 4.1: Path tracing function

Data: Position of the measuring point vec3 $m_{i,j}$ and the normal vec3 $n_{i,j}$.
Number of samples N to compute per $m_{i,j}$ and number of bounces D.
Scene data (texture, geometry, light sources, ...)

Result: Illuminance on $m_{i,j}$

1 vec3 $E_{e,direct} \leftarrow$ `sampleLights(`$m_{i,j}$, $n_{i,j}$, 1, 1`);`
2 vec3 $E_{e,indirect} \leftarrow (0,0,0)$;
3 **for** $n \leftarrow 1$ **to** N **by** 1 **do**
4 vec3 $E_e \leftarrow (0,0,0)$;
5 vec3 $pos \leftarrow m_{i,j}$;
6 vec3 $dir \leftarrow$ `getRandomDirectionFromHemisphere` $(n_{i,j})$;
7 $probability \leftarrow \frac{1}{2\pi}$;
8 vec3 $throughput \leftarrow$ (1,1,1) $\frac{\langle n_{i,j}, dir\rangle}{probability}$;
9 **for** $d \leftarrow 1$ **to** D **by** 1 **do**
10 $t \leftarrow$ `traceRay` (pos, dir);
11 **if** t *has an intersection* **then**
12 $E_e \leftarrow E_e +$ `sampleLights(`$t.hitPosition$, $t.hitNormal$, $\frac{t.albedo}{\pi}$, $throughput$`);`
 `// Calculate next ray direction`
13 $pos \leftarrow t.hitPosition$;
14 $dir \leftarrow$ `getRandomDirectionFromHemisphere(`$t.hitNormal$`);`
15 $throughput \leftarrow throughput \frac{t.albedo}{\pi} \frac{\langle t.hitNormal, dir\rangle}{probability}$
16 **else**
17 **break**;
18 **end**
19 **end**
20 $E_{e,indirect} \leftarrow E_{e,indirect} + E_e$;
21 **end**
22 **return** `radiometricToPhotometric(`$(\lambda_r, \lambda_g, \lambda_r)$, $E_{e,direct} + \frac{E_{e_indirect}}{N}$`);`

Performing the simulation for all measuring points on multiple measuring surfaces will give us the estimated illuminance. If the number of measuring points is low, we could render the resulting illuminance using numbers directly on the measuring surfaces. However, with an increasing number of measuring points on a surface, the resolution of our texture increases, and we have to visualize the illuminance so that the user gets insight into the data. Otherwise, the screen would be cluttered by numbers. We cover this in the

Algorithm 4.2: Sampling light and radiometric helper functions

Function `radiometricToPhotometric(`*vec3 λ, vec3* L_e`)`**:**
  $r \leftarrow 683 L_{e,r} V(\lambda_r)$;
  $g \leftarrow 683 L_{e,g} V(\lambda_g)$;
  $b \leftarrow 683 L_{e,b} V(\lambda_b)$;
  return $r + g + b$;

Function `sampleLights(`*vec3 position, vec3 normal, vec3 brdf, vec3 troughput*`)`**:**
  vec3 $E_e \leftarrow (0, 0, 0)$;
  foreach $l_i \in$ *Lights* **do**
    if l_i *is visible from position* **then**
      vec3 $lightDirection \leftarrow \frac{position - l_i.center}{\|position - l_i.center\|}$;
      vec2 $uv \leftarrow$ `directionToUVCoordinates(`$lightDirection$`)`;
      $I_v \leftarrow$ `sampleIESTexture(`$l_i.lightIndex$, uv`)`;
      vec3 $\hat{w} \leftarrow l_i.weights$;
      $k \leftarrow \frac{I_v}{683(V(\lambda_r)\hat{w}_r + V(\lambda_g)\hat{w}_g + V(\lambda_b)\hat{w}_b)}$;
      vec3 $I_e \leftarrow \hat{w}k$;
      $E_e \leftarrow E_e + throughput\ brdf\ I_e \frac{\langle normal, -lightDirection \rangle}{\|position - l_i.center\|^2}$;
    end
  end
  return E_e;

upcoming Section 4.5, how we present the resulting illuminance to the user.

4.5 Visualization

To provide the user with insight, we have to visualize the data, in our case, the computed illuminance. The data should be displayed in spatial proximity on the surfaces, where the illuminance was computed. It would be possible to represent the computed illuminance as a number near the measuring point. However, this approach only works with a small amount of measuring points. With an increasing number of measuring points, this will not become feasible since the user is overwhelmed by many numbers, and it is hard to compare the values. One common approach to tackle this is to not show the values as numbers but use the value and convert it to a specific color and visualize it using that specific color. This transformation is also known as colormapping, where predefined colormaps define the mapping between a scalar value, in our case, the illuminance and a color. For more detailed information about colormaps and their usage in visualization, we refer the reader to the survey paper by Zhou et al. [ZH16].

The illuminance is a value that ranges from zero to infinity. To cover this range, we need to define a colormap that maps all possible values to colors to give the user fast

and intuitive insight into the data. In our thesis, we use the defined values described in EN 12464-1:2021 to evaluate if the room is well illuminated. Depending on the given measuring surfaces and requirements, there are two different systems that define the minimum required illuminance. For ceilings, walls and cylindrical surfaces, the norm defines one minimum illuminance value that needs to be fulfilled. The minimum illuminance on workspaces is more complex and the norm defines two minimum values. The required maintained illuminance that is required for all workspaces and the modified maintained illuminance, that is required under special circumstances.

On measuring surfaces with one minimum value, the point of interest for the user is the minimum required illuminance that needs to be fulfilled. The user needs insight how far away the computed illuminance is from the requirement. Naturally, the values are split into two sections, one section below the and one section above the requirement. The section below is bounded by zero to the requirement illuminance and the section above is unbounded from the required illuminance to infinity. As a region of interest, we defined the upper limit of the above section to be the double of the required minimum illuminance to have a symmetric mapping. Values within the range will be gradually mapped to form a color gradient. Illuminance values that are greater than double of the required minimum illuminance are getting mapped to the same color regardless of the illuminance value. We use linear interpolation to map the illuminance values between two colors. Our procedure is visualized in Figure 4.8.

Colormaps are usually defined where the input ranges from zero to one and this range gets mapped to a specific color by linearly interpolating between two defined colors. So in our case, the illuminance values get mapped using the following formula:

$$\hat{E_v} = \frac{E_v}{2\bar{E_z}}. \tag{4.15}$$

The $\bar{E_z}$ in Equation 4.15 can be replaced by $\bar{E}_{m,\text{wall}}$ or $\bar{E}_{m,\text{ceiling}}$ depending on the measuring surface, where the mapping is applied. As upper limit we use the double of the required minimum so that we get a symmetric color mapping. Resulting values that exceed the interval $[0, 2\bar{E_z}]$ get clamped on the borders. We use a diverging colormap to represent the two sections so that the user easily recognizes if the resulting illuminance value is above, below or near the minimum required illuminance value. Values that are far away from the required illuminance are indicated by red or blue and values near the required illuminance are encoded as white. Figure 4.8 visualize the procedure how the illuminance value gets mapped to a specific color.

For work spaces, we use a three section colormap to represent values between zero and $\bar{E}_{m,r}$, between $\bar{E}_{m,r}$ and $\bar{E}_{m,m}$ and between $\bar{E}_{m,m}$ and $2\bar{E}_{m,m}$. To help the user in which sections the illuminance falls, we separate each section with a hard color break, unlike the previous color mapping. This three-section colormap is visualized in Figure 4.9, where we use a red to yellow mapping for the first section, a dark green to light green mapping for the second section and a blue to purple mapping for the third section. The procedure

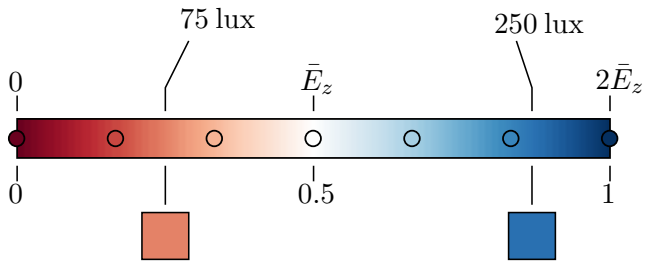


Figure 4.8: The bar shows how the two illuminance values on the top get mapped to a specific color, defined by the colormap in the middle. The black circles in the color bar represent the defined color at a specific position. An illuminance value gets mapped from zero to one and the resulting value is then used to linearly interpolate between two defined colors. For example, 75 lux gets mapped to 0.25 and this value is then used to interpolate between the two colors, so that the resulting color is a 50% mixture between the color of the 2nd circle and the 3rd circle.

is similar to the two-section colormaps except that we split it three times and between the section, we use two different colors that are close together to get a hard break. We decided to use a hard color break so that the user can easily distinguish if the calculated illuminance is above the required maintained illuminance or the modified maintained illuminance.

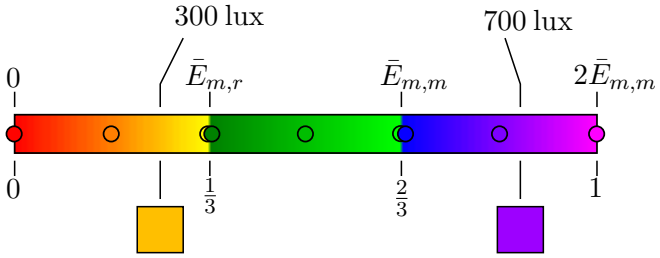


Figure 4.9: The figure shows a three-section colormap. The procedure is the same as shown in Figure 4.8. The difference is that we define three sections and a hard color break between them, which is achieved by using two colors in the colormap, that are closely together, as visualized by the two circles between the two sections.

The EN 12464-1:2021 not only defines the minimum required illuminance but also describes a uniform distribution of the received light on the measuring surface. The minimum illuminance uniformity U_o defines the illuminance uniformity that a measuring surface needs to fulfill to be compliant. The uniformity is evaluated using the following formula:

$$U_o = \frac{E_{v,min}}{E_{v,avg}}, \tag{4.16}$$

where $E_{v,min}$ is the lowest illuminance value on the measuring surface and $E_{v,avg}$ the average of all illuminance values on the measuring surface. To visualize the uniformity, we render a small rectangle near the measuring surface that is filled depending on the uniformity. If the uniformity does not fulfill the requirements, the filling will appear red, otherwise it will be rendered blue as seen in Figure 5.6

This chapter summarized the main parts of our approach, which included the definition of the measuring surfaces, the conversion from photometric units to radiometric units, the simulation itself and the presentation of the calculated illuminance values. In the upcoming chapter we will go in details how we implemented our methods utilizing hardware-accelerated ray tracing.

CHAPTER 5

Implementation

To implement our approach, we used Tamashii [Lip21], a rendering framework that should make writing implementations of scientific renderers easier. Tamashii allows researchers to speed up the implementation of their approach by providing them with a fundamental structure of a working renderer. It is written in C++ and uses the Vulkan application programming interface (API) [Khrb] for rendering and computation tasks. Imgui [Cor] is the natively supported user interface and allows the user to add objects to a scene and change specific parameters. The main focus of Tamashii is to provide an abstraction to the underlying rendering API, so that the implementer can focus on their work and not on topics like memory alignment, shader compilation or scene loading.

Our goal was to implement a renderer that calculates the illuminance on predefined measuring surfaces and provides the user with fast insight into the computed data. Unlike DIALux [DIA] and Relux [Rel], where the user has to wait until the simulation is finished, we want to display the current results of the computed illuminance to the user. To explore and visualize three-dimensional scenes, the software has to render the scene multiple times per second so that the exploration of the scene feels continuous. The calculation time of the simulation heavily depends on the number of samples we calculate for our measuring points and on the hardware where the computation is running. The primary workload is done on the GPU. Before we render the scene, we perform the lighting simulation with a few samples so that the simulation does not take too long. This is important because the renderer has to wait until the simulation is finished, and therefore a long simulation time will increase the rendering time. The samples we computed are accumulated between the frames and with each frame, the simulation results will become more accurate.

The basic workflow of the program is visualized in Figure 5.1. In the first step the renderer is initialized, where all the buffers are created, then the shaders are loaded and the Vulkan context is created. After the initialization process is finished, a scene can be loaded. As supported scene format, we use glTF [Khra], because Tamashii natively supports this file format and it allows us to add custom properties to scene objects. This

is important because we annotate light objects with a custom property where the key is "ies" and the value is the name of the IES file. If a glTF scene is loaded, the importer searches for the ies custom property and parses the IES file and transforms the data to a texture as described in Section 4.2. To distinguish the measuring surfaces from other three-dimensional objects, we use a specific naming convention so that the scene importer knows if the given object is a measuring surface or not. We will cover the naming convention later in this section. In the scene loading process, the program also creates two textures for each measuring surface, one to store the simulation results and one RGB texture for displaying the results on the surfaces.

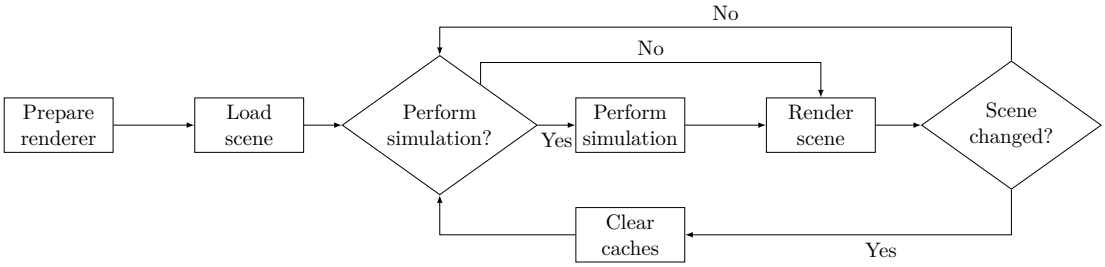


Figure 5.1: The figure shows the basic workflow of Kido.

After the scene is adequately loaded and all data structures are initialized, the simulation is running automatically. The simulation is running as long the sample points are under a certain threshold or stopped by the user. If there are any scene changes that influences the distribution of light in the scene, the program will start the simulation again and clear all previous computed illuminance values. The flow between "Perform simulation" and "Scene changed" in Figure 5.1 is our render loop. As visualized in the flowchart, the simulation occurs before the scene is rendered. When the simulation is started, we call for each measuring surface a ray trace call to perform the calculation on the GPU using the `GL_EXT_ray_tracing` Vulkan extension. This call allows us to speed up the ray trace process by utilizing ray trace kernels on the GPU. We store the calculated illuminance inside a texture and we also store the accumulated irradiance values, which is needed to compute the illuminance across multiple frames. After the computation of the illuminance is done, we convert the illuminance values according to our defined colormaps to a specific color and the resulting color is stored in the albedo map of the measuring surface. The color mapping process is done on the CPU. After the texture is updated, we are able to render the scene, including the color-mapped measuring surfaces. For rendering, we used a simple rasterizer, which is able to display textures and simple light sources. Figure 5.6 shows a screenshot of Kido, including a rendered scene. In the upcoming text, we will explain the most critical sections of the application in detail.

We decided to use the glTF format for our scene description. It is supported by many content-creation programs and allows us to use these programs for creating our scenes. Example scenes that we use in our application were created using Blender [Ble], an open-source three-dimensional content-creation program. In Blender we also annotated

light sources with a custom property that defines the emittance of the light source according to the IES file format. When the parser recognizes that an ies property is set, Kido will load the corresponding ies file and converts the data into a texture, that is then later used in the shader to query the luminous intensity. For our approach, we also need the CCT. Unfortunately, the ies file format does not include a field that holds this value. Therefore we add it manually to an ies file using a custom keyword. We use the keyword `[COLORTEMP]` to store the CCT in Kelvin and use it to convert the luminous intensity to radiant intensity as described in Section 4.3. Kido also has to consider whether an object in the scene is a measuring surface or not. We decided to name the mesh data accordingly so that Kido can distinguish measuring surfaces from other scene objects. The naming convention for measuring surfaces is visualized and described in Figure 5.2.

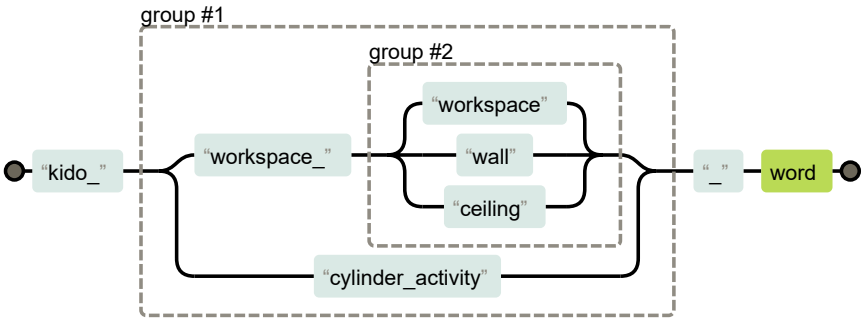


Figure 5.2: The figure shows the naming convention used by Kido to distinguish measuring surfaces from other objects. Each measuring surface begins with the prefix kido, followed by an underline. Kido supports two different measuring surfaces, rectangles and cylinders. The third token describes the surface type if the object is a workspace, wall, ceiling or activity. Activity is used to define the cylindrical illuminance. After the third token, the user can define any character sequence to identify the surface and give it an unique name. The figure was created using Regexper [Ava].

After parsing the name of the measuring surface, Kido knows which surface it is. Depending on the given name, the application calculates the parameters of the surface that are then later used to calculate the measuring points. These parameters are described in Section 4.1. When the scene is adequately loaded, Kido is able to perform the simulation.

To speed up the calculation, Kido uses hardware-accelerated ray tracing using the `vkCmdTraceRaysKHR` function provided by the Vulkan API. The trace ray function takes as input shader tables that describe how a ray is generated, what happens when the ray hits an object and what happens if the ray does not hit any objects. The main logic of the simulation is written, in the ray generation shader. Additionally, the trace ray function takes integer values, describing how often the ray generation shader should be called. For example, when the measuring surface consists of a four-by-four grid, the ray generation shader is called 16 times to calculate the illuminance on 16 different measuring points. Figure 5.3 shows the first section of `main` function, which is called after we start

our simulation process.

```
1  void main() {
2    seed = initRandomSeed(uint(gl_LaunchIDEXT.y * gl_LaunchSizeEXT.x +
       gl_LaunchIDEXT.x), uint(ubo.calc_count/*frameIndex*/));
3
4    // Calculate starting point from measuring surface
5    vec2 deltaUV = 1.0 / mSurface.imageSize;  // texel size in texture space
6    vec4 measuringStartPointWorld;          // world position of the texel center
7    vec3 measuringNormalWorld;            // normal in world space of the texel
8
9    // Rectangle surface
10   if ( mSurface.type == 0 ) {
11     vec4 deltaULocal = mSurface.dirULocal * deltaUV.x;
12     vec4 deltaVLocal = mSurface.dirVLocal * deltaUV.y;
13     vec4 texelOffsetLocal = deltaULocal * 0.5 + deltaVLocal * 0.5;
14     measuringStartPointWorld = mSurface.originLocal + texelOffsetLocal +
       deltaULocal * gl_LaunchIDEXT.x + deltaVLocal * gl_LaunchIDEXT.y;
15     measuringStartPointWorld = mSurface.modelMatrix * vec4(
       measuringStartPointWorld.xyz, 1.0);
16     measuringNormalWorld =  normalize(mat3(mSurface.modelMatrix) * cross(mSurface.
       dirVLocal.xyz, mSurface.dirULocal.xyz));
17   }
18
19   // Cylindric surface
20   if ( mSurface.type == 1 ) {
21     float r = mSurface.radiusLocal;
22     float deltaPhiCircular = 2 * M_PI * deltaUV.x;
23     // Offset M_PI is added because vertex with uv(0,0) is located at the neg. z
       axis
24     float phi = deltaPhiCircular * (gl_LaunchIDEXT.x + 0.5) + M_PI;
25
26     vec3 measuringStartPointULocal = vec3(r * sin(phi), mSurface.topCenterLocal.y,
        r * cos(phi));
27     vec3 stepVLocal = mSurface.dirTopToBotLocal.xyz * deltaUV.y * (gl_LaunchIDEXT.
       y + 0.5);
28
29     measuringStartPointWorld = mSurface.modelMatrix * vec4((
       measuringStartPointULocal + stepVLocal), 1.0);
30     measuringNormalWorld = normalize(mat3(mSurface.modelMatrix) * (
       measuringStartPointULocal.xyz - mSurface.topCenterLocal));
31   }
32
33   ...
34 }
```

Figure 5.3: The figure shows the first part of the ray generation shader, where the position and normal of the measuring point is calculated.

For each measuring point $m_{i,j}$, the `main` function gets called. The predefined variables provided by Vulkan, `gl_LaunchIDEXT.x` and `gl_LaunchIDEXT.y` can be accessed to identify the current execution. We use these two variables as our measuring point indices i and j. The indices are then used together with the precalculated parameters to compute the position and normal in world space of the specific measuring point. The calculation depends on the shape of the surface. The calculation of the normal and position is listed in Figure 5.3 and is based on Equation 4.2 and 4.3.

```
void main() {
  ...

  // Irradiance from direct light
  vec3 direct_irradiance = sampleLights(measuringStartPointWorld.xyz /*x*/,
    measuringNormalWorld.xyz /*n*/, vec3(1.0) /*brdf*/, vec3(1.0));
  // Irradiance from indirect light
  vec3 indirect_irradiance = vec3(0.0);

  // N samples for indirect light
  for(int i = 0; i < ubo.samplesToCompute; i++){
    Ray ray;
    ray.origin = measuringStartPointWorld.xyz;
    ray.direction = normalize(tangentSpaceToWorldSpace(sampleUnitHemisphereUniform
    (vec2(randomFloat(seed), randomFloat(seed))), measuringNormalWorld));
    ray.t_min = RAY_MIN_DISTANCE;
    ray.t_max = RAY_MAX_DISTANCE;

    vec3 irradiance = vec3(0.0);
    float prob = M_INV_2PI;

    vec3 throughput = vec3(dot(measuringNormalWorld, ray.direction) / prob);
    // Go along one path
    for (uint depth = 0; depth < ubo.numBounces; depth++){
      traceRayEXT(tlas, gl_RayFlagsNoneEXT, 0xff, 0, 0, 0, ray.origin, ray.t_min,
    ray.direction, ray.t_max, 0);
      // Miss
      if( rp.instanceID == -1 ) break;

      HitData hd;
      getHitData(hd, ray);

      // Backface check
      if(dot(hd.geo_n_ws_norm, -ray.direction) <= 0.0)  break;

      // Irradiance from point x to v
      irradiance += sampleLights(hd.hit_pos_ws /*x*/, hd.geo_n_ws_norm /*n*/, hd.
    albedo.xyz / M_PI /*brdf*/, throughput);

      // Next ray
      vec3 omega = normalize(tangentSpaceToWorldSpace(sampleUnitHemisphereUniform(
    vec2(randomFloat(seed), randomFloat(seed))), hd.geo_n_ws_norm));
      throughput *= hd.albedo.xyz / M_PI  * dot(hd.geo_n_ws_norm, omega) /
    PDF_UNIT_HEMISPHERE_UNIFORM;
      ray.origin = hd.hit_pos_ws;
      ray.direction = omega;
      ray.t_min = RAY_MIN_DISTANCE;
      ray.t_max = RAY_MAX_DISTANCE;
    }

    indirect_irradiance += irradiance;
  }

  ...
}
```

Figure 5.4: The figure shows the second part of the ray generation shader, where we compute the irradiance for the specific measuring point.

```
void main() {
  ...

  vec4 result = vec4(0.0);
  float illuminance = 0.0;

  if( ubo.accumulate ) {
    if(ubo.calc_count > 0) indirect_irradiance += imageLoad(results_img, ivec2(
    gl_LaunchIDEXT.x, gl_LaunchIDEXT.y)).yzw;
    result = vec4(0.0, vec3(indirect_irradiance));
    illuminance = radiometricToPhotometric(vec3(611e-9, 549e-9, 464e-9),
    direct_irradiance + indirect_irradiance / ubo.numSamples);
  } else {
    illuminance = radiometricToPhotometric(vec3(611e-9, 549e-9, 464e-9),
    direct_irradiance + indirect_irradiance / ubo.samplesToCompute);
  }

  result.x = illuminance;
    imageStore(results_img, ivec2(gl_LaunchIDEXT.x, gl_LaunchIDEXT.y), result);
}
```

Figure 5.5: The figure shows the third part of the ray generation shader, where we convert the irradiance to the illuminance and combine the calculated samples with the previously calculated samples.

Figure 5.4 shows the second part of the `main` function after the position and normal of $m_{i,j}$ is computed. This part contains the computation part, where the irradiance is calculated and it is based on Algorithm 4.1. We sample all ies lights on the measuring point itself and on each bounce using the `sampleLights` function listed in Figure 5.7. Inside the `sampleLights` function the luminous intensity is converted to radiant intensity using our proposed conversion in Section 4.3. The `sampleLights` function is based on the pseudo-code listed in Figure 4.2.

In the last part of the ray generation shader, shown in Figure 5.5, we use an accumulation texture, where we store the previously calculated samples. The buffer is needed because we want to calculate only a few samples for each frame to maintain a stable frame rate. Together with the stored samples and the current computed samples, we convert the irradiance to illuminance. The computed illuminance is then stored in a texture together with the accumulated samples.

After the computation for all measuring surfaces is finished, we have to transfer the data from the GPU memory to the main memory and convert the illuminance values according to our defined colormap. The resulting color for each measuring point is stored in the albedo texture and is later used and displayed by the renderer. We defined the colormaps in Paraview 5.10.0-RC1 [San] and exported it as JSON [Bra17] file. Kido parses this JSON file to its own internal format and uses the defined colormap for converting the illuminance values. After the illuminance values are converted, we can start visualizing the scene to present the current computed illuminance values to the user. The process of calculating the illuminance and rendering the scene is repeated and with time the

Figure 5.6: The figure shows a screenshot of Kido, where an example office scene is loaded. The scene consists of several measuring surfaces, which display the illuminance value as color. In the left upper corner, it is possible to change the parameters of the simulation and the requirements values. Various information about the selected measuring surfaces is displayed on the right bottom corner.

simulation will converge to a stable value.

While the process is running, the user can move objects in the scene. If the position of an object changes, the simulation has to start again to compute correct results. Kido will take care if there are any scene changes and will restart the lighting simulation if needed.

As mentioned before Tamashii uses imgui as underlying user interface, which handles the interaction between the user and the application. The Tamashii framework already provides a set of interactions, like moving and adding objects to the scene, moving the camera and modifying object properties. Kido extends the interface to display the minimum, average and highest illuminance values for the selected measuring surface. It also displays the uniformity of the computed illuminance values. We also included the possibility for the user to change specific parameters of the simulation. These parameters are the number of samples per measuring point and the number of ray bounces. It is also possible to change the requirement values, like the required maintained illuminance and the minimum cylindrical illuminance. With changing the requirements values, the color map will also adapt adequately to the change. Figure 5.6 shows a screenshot of Kido.

In the upcoming section, we take a closer look at how fast Kido can provide robust results to the user.

```
vec3 sampleLights(vec3 x, vec3 n, vec3 brdf, vec3 throughput){
  vec3 irradiance = vec3(0.0);
  for(int i = 0; i < ubo.light_count; i++) {
    if(isIesLight(i)){
      Light_s light = light_buffer[i];

      vec3 dir = light.pos_ws - x;
      Ray shadowRay;
      shadowRay.origin = x;
      shadowRay.direction = normalize(dir);
      shadowRay.t_min = RAY_MIN_DISTANCE;
      shadowRay.t_max = length(dir);

      traceRayEXT(tlas, gl_RayFlagsNoneEXT, 0xff, 0, 0, 0, shadowRay.origin,
    shadowRay.t_min, shadowRay.direction, shadowRay.t_max, 0);
      if ( rp.instanceID == -1 )  {
        if(dot(n, shadowRay.direction) <= 0.0) continue;
        // Light is visible from measuring surface point
        vec2 uv = iesUvCoordinates(i, -shadowRay.direction);
        float intensity = texture(texture_sampler[light.ies_tex_idx], uv).x;

        vec3 weights = radiant_weights[i].xyz;

        float k = intensity / dot(683 * lumEffiCurve(vec3(611e-9, 549e-9, 464e-9))
    , weights);

        vec3 radiant_intensities = vec3(0.0);
        radiant_intensities.x = weights.x * k;
        radiant_intensities.y = weights.y * k;
        radiant_intensities.z = weights.z * k;
        irradiance += throughput * brdf * radiant_intensities * dot(n, shadowRay.
    direction) / (shadowRay.t_max * shadowRay.t_max);
      }
    }
  }
  return irradiance;
}
```

Figure 5.7: The figure shows the sampleLights function in the ray generation shader, where we sample the luminous intensity out of the texture and use it to perform our simulation.

CHAPTER 6

Results

In this chapter, we examine Kido regarding computation time and correctness of the computed illuminance. The upcoming section shows how fast our implementation converges to a specific illuminance value. In Section 6.2 we compare our results with state-of-the-art software and measurings taken from a real-world measurement experiment.

6.1 Benchmark

Our goal was to provide the user with faster insight into the computation results compared to traditional software like DIALux and Relux. Comparing the computation time of Kido with DIALux or Relux is not easily achievable because both applications additionally generate various information when performing the light simulation. These additional information are illuminance tables, multiple pseudo color renderings and documents including all information about the luminaires in the scene and their positions. Due to this behavior, it is hard to measure only the time of the simulation, because it is not possible to only run the simulation without creating this additional information. Nevertheless, we also did a benchmark against a test scene at the end of this section. In the upcoming section, we will compare the calculated illuminance of Relux, DIALux and Kido.

First we only benchmark Kido in terms of computation time and discuss how fast Kido provides plausible computation results. For this we created a test scene that represents a common shared office room. The room has four working desks, three luminaires and has a rectangular shape. We defined a rectangular measuring surface and a cylindrical measuring surface for each working desk. We also included three measuring surfaces on the walls, one measuring surface on the ceiling and one cylindrical measuring surface near the couch. In total the scene consists of thirteen measuring surfaces and each measuring surface has 128 by 128 measuring points.

One prerequisite for benchmarking is that the frame rate should not fall below 30 frames per second. This requirement is crucial, so that the user can change and explore the scene in real-time. As a benchmark environment, we ran the simulation on an Intel i5-6600 CPU and a NVIDIA GeForce RTX 3060 GPU. With our test scene and the computer setup, we could compute ten samples per measuring point under a stable frame rate of 33 frames per second. With our scene, these are 212992 rays we trace every frame with a number of three bounces per ray. To test how fast Kido computes plausible illuminance results, we computed the illuminance multiple times but with different random seeds for a specific measuring point. Figure 6.1 shows the computed illuminance at different runs. It is visible that the variance between the runs is high when the number of samples is low, as visualized in Figure 6.2. After about 500 samples, the variance falls under 25 lux^2 and provides the user with a plausible result. To compute at least 500 samples for each measuring point in our test scene, the simulation will take about 1.5 seconds. After 2500 samples the variance goes down to about 3 lux^2, which takes 7.5 seconds to compute. In our test case with ten different seeds, the illuminance value was in the worst case 3.8 lux away from the reference value. Due to the fast convergence of the result, the user can perceive the current lighting situation and adapt the objects accordingly if they do not meet the defined requirements.

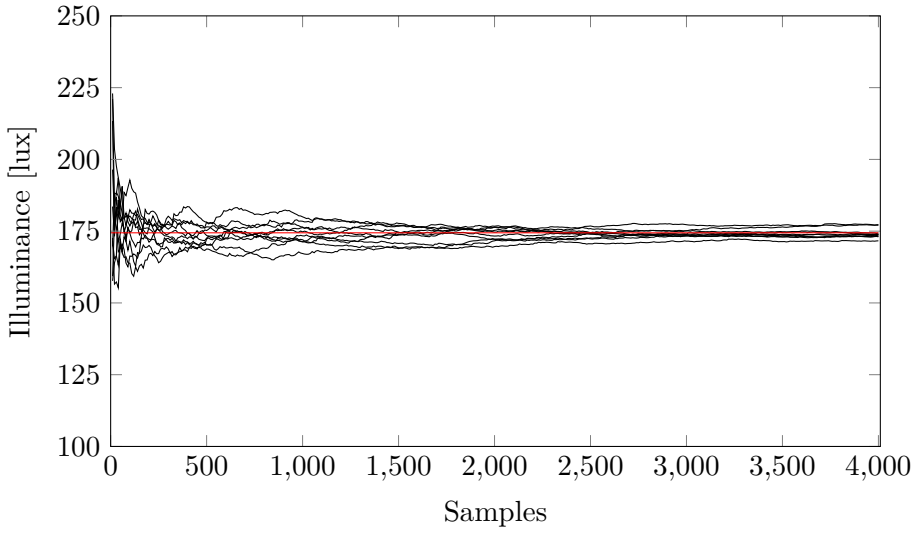


Figure 6.1: The graph shows multiple runs for calculating the illuminance on the same measuring point. Each run is visualized as a black line and has a different initial seed for picking random directions. The straight line is the reference illuminance computed with one million samples, where the result is 174.428 lux.

Figure 6.3 shows how the results change over time with increasing sample count on a measuring surface. Even with a low sample size like ten samples per measuring point, the user will get a good intuition how the illuminance is distributed on the measuring surface.

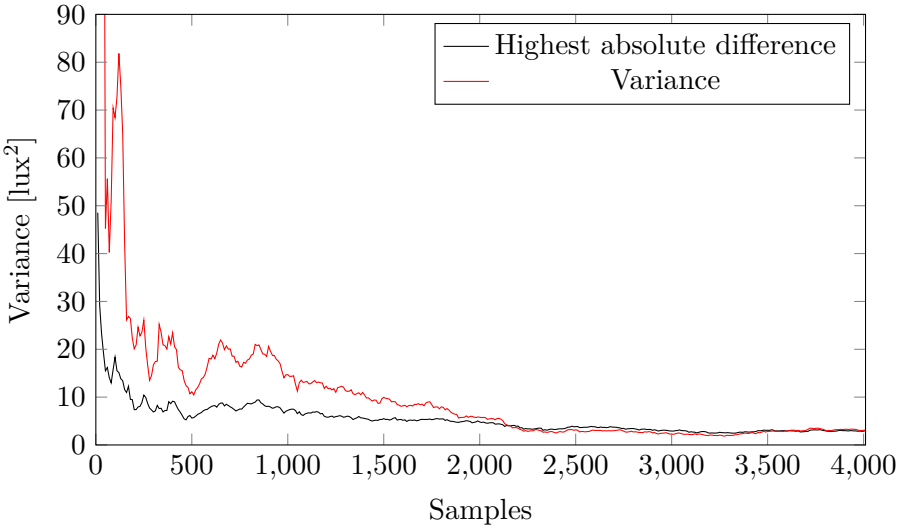
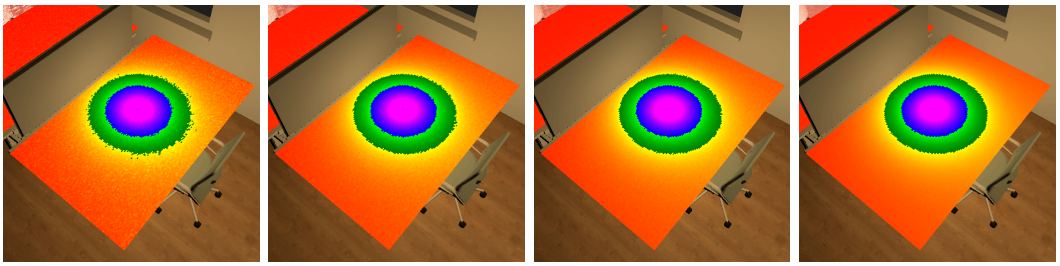


Figure 6.2: The graph shows the variance of the computed illuminance values in red and the highest absolute difference of the reference illuminance value mentioned in Figure 6.1.



(a) 10 Samples (b) 50 Samples (c) 100 Samples (d) 500 Samples

Figure 6.3: The four screenshots show how the visualization of the illuminance values changes with increasing sample count per measuring point.

In addition, we also compared the simulation time between Kido, DIALux version 5.10.0.56785 [DIA] and Relux version 2021.1.1.0 [Rel]. As simulation time we count the time from the start of the simulation till the user sees the result in the application. We used the validation test scene described in the upcoming section to benchmark the three applications. Table 6.1 shows the result of the benchmarks.

Application	Low quality	High quality
Kido	70 ms	130ms
DIALux	approx. 7000 ms	approx. 7000 ms
Relux	approx. 1000 ms	approx. 2000 ms

Table 6.1: Computation times of the validation test scene.

DIALux and Relux support different computation modes, a low quality (faster) and a high quality (slower) mode. For Kido, we used 1500 samples per measuring point for the low quality mode and 3000 samples per measuring point for the high quality mode. As seen in Table 6.1, Kido is significantly faster compared to DIALux and Relux.

The first benchmark of the office scene shows that our application can provide fast and plausible results to the user, which will help the user when planning lighting environments. This fast approximation to the reference value proves that with available consumer hardware, our application meets our requirement in terms of computation time and allows us to redefine the workflow of current state-of-the-art software, as discussed in Chapter 4. Also compared to other state-of-the-art software, Kido is able to outperform DIALux and Relux in terms of calculation time. In the upcoming section, we will take a closer look on the difference of the computed illuminance values by the different applications.

6.2 Validation

To validate our approach in terms of quality we compared it against DIALux version 5.10.0.56785 [DIA] and Relux version 2021.1.1.0 [Rel]. Additionally, we also did a real-world measurement using an illuminance meter. We re-created the real-world scene in all applications to compare the four measurements. For the simulation scene, we defined all surfaces as Lambertian surfaces. As color for the floor we used a light brown color with RGB hex value of 0xD1AF84 and for the walls and ceilings we used light gray color with a value of 0xE5E5E5. The real-world scene and the scene used with Kido can be seen in Figure 6.4. Our test scene contains one luminaire, where we used a Philips LED Lamp [Phi]. The lamp is placed near a corner so that the indirect lighting has a high impact on our measurements. We measure the illuminance on ten specific points, three on each wall and four on the floor. As illuminance meter we used the PeakTech 5086 [Pea]. The exact position of the measurement points and the position of the luminaire are annotated in Figure 6.5.

To calculate the resulting illuminance for our approach, we computed one million samples for every measuring point and set the number of bounces to ten. For DIALux and Relux we used the most accurate computation option to compute the illuminance. Figure 6.6 shows the results when only considering direct light. The direct lighting results produced by the different applications are quite similar except for the measuring point 1 and 10, where DIALux computed 0 lux. For the measuring point 2 and 9, DIALux also computed a much lower value compared to Kido and Relux. Figure 6.7 shows the results of the simulations considering direct and indirect light. For the measuring points on the floor (4, 5, 6, 7) the computed values are quite similar. The measuring points on the wall are more diverse across the computed illuminance values, especially of the computed values by DIALux for point 1 and 10. The measured illuminance using PeakTech 5086 are close to the computed values except for the measuring points 3 and 8. Without knowing the source code it hard to tell why the results are so high in our comparison to the other. Further investigation are needed to find the cause of the differences and

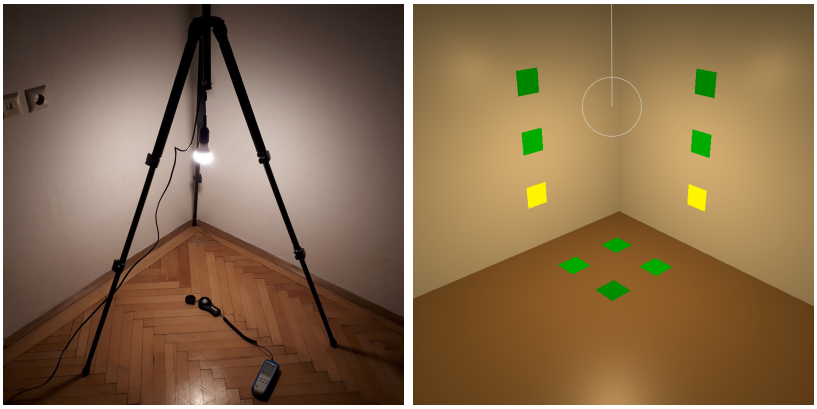


Figure 6.4: The figure shows our real-world measurement setup, including the illuminance meter on the floor. The image on the right shows a screenshot of the test scene in Kido, which includes the ten measuring points as measuring surfaces.

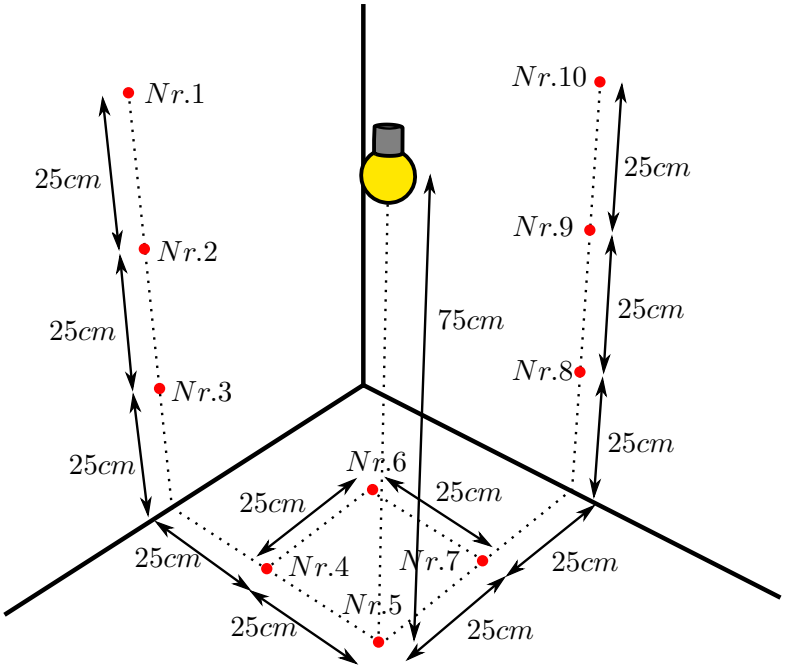


Figure 6.5: The figure shows the location of the measuring points, which are marked as red dots. The location of the luminaire is 75 cm above the measuring point Nr. 5.

making a statement about it. One possible cause why there is so a huge difference at point 3 and 8 can be that the provided IES file is not equal to the real emitted radiation.

The results show that our approach also gives plausible results in terms of correctness of the computed illuminance values compared to DIALux and Relux. Nevertheless, to fully validate our software a more professional test setup would be necessary.

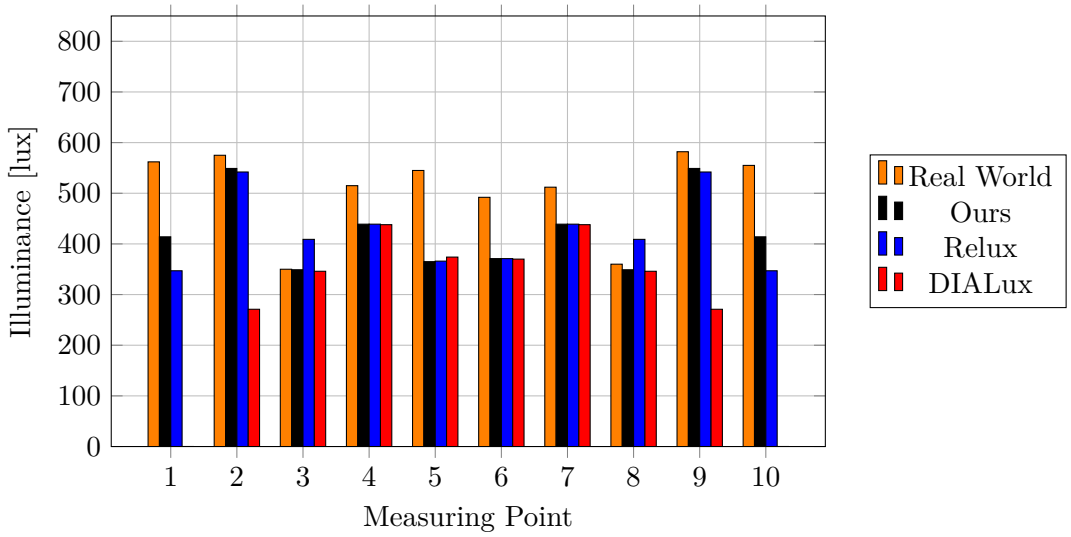


Figure 6.6: Shows the computed and measured illuminance values on the defined measuring points. The simulation results only considered direct light.

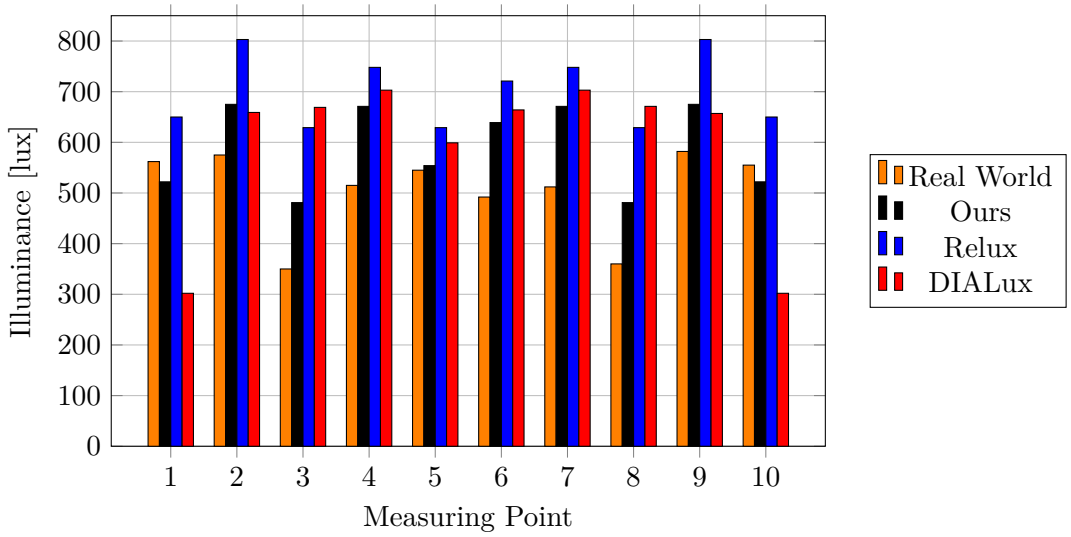


Figure 6.7: Shows the computed and measured illuminance values on the defined measuring points. The simulation results considered direct and indirect light.

CHAPTER 7

Conclusion

In this thesis, we showed that hardware-accelerated ray tracing can reduce the simulation time in lighting planning software. Compared to state-of-the-art software, we were able to present computation results faster to the user. Even with low sample size, the user will get a good intuition how the light is distributed among the scene as mentioned in Section 6.1. By providing early results to the user, he or she can try out different lighting setups in a short time period. The ability to try out different lighting setups faster will result in higher productivity when planning lighting concepts for buildings.

One major topic for future research is to formulate the luminaire setup as inverse problem and utilize hardware-accelerated ray tracing in the computation process. Recent advancements in inverse rendering can help the user to find an optimized solution for a specific scene. Out of a given initial guess provided by the user and defined constraints, inverse rendering could be used to compute the position and direction of the luminaires. Using additional parameters in the inverse rendering process will further improve the solution. These additional parameters could be the cost of operating the luminaire, like in LiteVis [SOL+16], where different solutions are computed and are selectable by the user to explore. Compared to LiteVis, using inverse rendering enables better-optimized solutions, where the whole simulation process is taken into account.

Known limitations of our approach are that we only consider diffuse surfaces and therefore do not consider specular reflections in our simulation process. Replacing the Lambertian BRDF with a more complex BRDF will enhance the overall correctness of the results. Light emitted by the sun has a massive impact on the overall illumination of a scene. In our work, we only considered artificial light sources and did not include incident daylight. Additionally, we only represent the electromagnetic spectrum with three discrete peaks.

Overall our approach proves that the usage of hardware-accelerated ray tracing in the lighting simulation reduces the calculation time and opens up new workflows for lighting planning software.

List of Figures

List of Tables

List of Algorithms

Acronyms

API application programming interface. 35, 37

ASI Austrian Standard International. 8

BRDF bidirectional reflectance distribution function. 14, 28, 29, 49

CCT correlated color temperature. 15, 24, 37

CEN Comité Européen de Normalisation. 8

CENELEC Comité Européen de Normalisation Électrotechnique. 8

CIE Commission Internationale de L'Eclairage. 4, 5, 13, 26, 27

CPU central processing unit. 17, 36, 44

DIN Deutsches Institut für Normung. 8

EN European norms. 8

ETSI European Telecommunications Standards Institute. 8

GPU graphics processing unit. 2, 17, 18, 24, 29, 35, 36, 40, 44

Bibliography

[And] Andrey Legotin. IESviewer. http://photometricviewer.com/. [Online; accessed 13.02.2022].

[Ava] Jeff Avallone. Regexper. https://regexper.com/. [Online; accessed 30.04.2022].

[Ber02] D. M. Berson. Phototransduction by retinal ganglion cells that set the circadian clock. *Science*, 295(5557):1070–1073, February 2002.

[Ble] Blender Foundation. Blender. https://www.blender.org/. [Online; accessed 30.04.2022].

[Bra17] T. Bray. The javascript object notation (json) data interchange format. STD 90, RFC Editor, December 2017.

[Buk19] Michael Bukshtab. *Photometry, Radiometry, and Measurements of Optical Losses*. Springer Singapore, 2019.

[BVSoA95] M. Bass, E.W. Van Stryland, and Optical Society of America. *Handbook of Optics: Devices, measurements, and properties*. Handbook of Optics. McGraw-Hill, 1995.

[Cau18] Brian Caulfield. NVIDIA CEO Jensen Huang Unveils Turing, Reinventing Computer Graphics. https://blogs.nvidia.com/blog/2018/08/13/jensen-huang-siggraph-turing-quadro-rtx/, August 2018. [Online; accessed 28.12.2021].

[Cor] Omar Cornut. Imgui. https://github.com/ocornut/imgui. [Online; accessed 30.04.2022].

[CSC11] Anne-Marie Chang, Frank A. J. L. Scheer, and Charles A. Czeisler. The human circadian system adapts to prior photic history. *The Journal of Physiology*, 589(5):1095–1102, February 2011.

[DIA] DIAL GmbH. DIALux. https://www.dialux.com/en-GB/download. [Online; accessed 27.12.2021].

[DIN21] DIN Deutsches Institut für Normung. Licht und Beleuchtung –Beleuchtung von Arbeitsstätten –Teil 1: Arbeitsstätten in Innenräumen;Deutsche Fassung EN 12464-1:2021, 2021.

[GP18] Anastasios Gkaravelis and Georgios Papaioannou. Light optimization for detail highlighting. *Comput. Graph. Forum*, 37(7):37–44, 2018.

[Har20] Takahiro Harada. Hardware-Accelerated Ray Tracing in AMD Radeon™ ProRender 2.0. `https://gpuopen.com/learn/radeon-prorender-2-0/`, November 2020. [Online; accessed 28.12.2021].

[Hat02] S. Hattar. Melanopsin-containing retinal ganglion cells: Architecture, projections, and intrinsic photosensitivity. *Science*, 295(5557):1065–1070, February 2002.

[HE21] Kevin W. Houser and Tony Esposito. Human-Centric Lighting: Foundational Considerations and a Five-Step Design Process. *Frontiers in Neurology*, 12, January 2021.

[HWO02] Lisa Heschong, Roger L. Wright, and Stacia Okura. Daylighting impacts on human performance in school. *Journal of the Illuminating Engineering Society*, 31(2):101–114, July 2002.

[Int99] International Electrotechnical Commission. Iec 61966-2-1:1999 - multimedia systems and equipment - colour measurement and management - part 2-1: Colour management - default rgb colour space - srgb, 1999.

[JH19] Wenzel Jakob and Johannes Hanika. A low-dimensional function space for efficient spectral upsampling. *Computer Graphics Forum (Proceedings of Eurographics)*, 38(2), March 2019.

[JL19] Sam Jin and Sung-Hee Lee. Lighting layout optimization for 3d indoor scenes. *Computer Graphics Forum*, 38(7):733–743, October 2019.

[Kaj86] James T. Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH '86*. ACM Press, 1986.

[Khra] Khronos Group. gltf. `https://www.khronos.org/gltf/`. [Online; accessed 30.04.2022].

[Khrb] Khronos Group. Vulkan api. `https://www.vulkan.org/`. [Online; accessed 30.04.2022].

[KM09] Gerhard Kramm and Nicole Mölders. Planck's blackbody radiation law: Presentation in different domains and determination of the related dimensional constants. January 2009.

[Leo] LeoMoon Studios. IES Lights Pack. https://leomoon.com/store/shaders/ies-lights-pack/. [Online; accessed 12.02.2022].

[Lig] Lighting Analysts. AGi32. https://lightinganalysts.com/software-products/agi32/overview/. [Online; accessed 27.12.2021].

[Lip21] Lukas Lipp. Tamashii rendering framework, 2021. Private conversation.

[LRK17] André Liemert, Dominik Reitzle, and Alwin Kienle. Analytical solutions of the radiative transport equation for turbid and fluorescent layered media. *Scientific Reports*, 7(1), June 2017.

[LTH+13] Christian Luksch, Robert F. Tobler, Ralf Habel, Michael Schwärzler, and Michael Wimmer. Fast light-map computation with virtual polygon lights. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '13*. ACM Press, 2013.

[McG17] Morgan McGuire. Computer graphics archive. https://casual-effects.com/data, July 2017. [Online; accessed 06.03.2022].

[MFA06] F. Maamari, M. Fontoynont, and N. Adra. Application of the CIE test cases to assess the accuracy of lighting computer programs. *Energy and Buildings*, 38(7):869–877, July 2006.

[MMP+22] Thomas Mansencal, Michael Mauderer, Michael Parsons, Nick Shaw, Kevin Wheatley, Sean Cooper, Jean D. Vandenberg, Luke Canavan, Katherine Crowson, Ofek Lev, Katrin Leinweber, Shriramana Sharma, Troy James Sobotka, Dominik Moritz, Matt Pppp, Chinmay Rane, Pavithra Eswaramoorthy, John Mertic, Ben Pearlstine, Manuel Leonhardt, Olli Niemitalo, Marek Szymanski, Maximilian Schambach, Sianyi Huang, Mike Wei, Nishant Joywardhan, Omar Wagih, Pawel Redman, Joseph Goldstone, Stephen Hill, Jedediah Smith, Frederic Savoir, Geetansh Saxena, Saransh Chopra, Ilia Sibiryakov, Tim Gates, Gajendra Pal, Nicolas Tessore, and Aurélien Pierre. Colour 0.4.1. https://doi.org/10.5281/zenodo.6288658, February 2022. [Online; accessed 05.03.2022].

[NMFY17] Worawan Natephra, Ali Motamedi, Tomohiro Fukuda, and Nobuyoshi Yabuki. Integrating building information modeling and virtual reality development engines for building indoor lighting design. *Visualization in Engineering*, 5(1), October 2017.

[PE19] Jianzhong Jiao Paul Ericson. *ANSI/IES LM-63-19 - IES STANDARD FILE FORMAT FORTHE ELECTRONIC TRANSFER OF PHOTOMETRIC DATAAND RELATED INFORMATION*. Illuminating Engineering Society, 2019.

[Pea] PeakTech. Peaktech p5086. https://www.peaktech.de/PeakTech-P-5086-Lux-Meter-400.000-Counts-0-...-40-400-4000-40000-400000-Lux/P-5086. [Online; accessed 29.05.2022].

[Phi] Philips. Corepro ledbulb nd 13-100w a60 e27 827. https://www.lighting.philips.com/main/prof/led-lamps-and-tubes/led-bulbs/corepro-plastic-ledbulbs/929001234502_EU/product. [Online; accessed 29.05.2022].

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, November 2016.

[Rel] Relux Informatik AG. Relux. https://relux.com/en/relux-desktop.html. [Online; accessed 27.12.2021].

[San] Sandia National Laboratories, Kitware Inc, Los Alamos National Laboratory. Paraview. https://www.paraview.org/. [Online; accessed 30.04.2022].

[Sch10] Steven Schwartz. *Visual perception : a clinical orientation*. McGraw-Hill Medical Pub. Division, New York, 2010.

[SMG+12] PJC Sleegers, NM Moolenaar, M Galetzka, A Pruyn, BE Sarroukh, and B van der Zande. Lighting affects students' concentration positively: Findings from three dutch studies. *Lighting Research & Technology*, 45(2):159–175, June 2012.

[SOL+16] Johannes Sorger, Thomas Ortner, Christian Luksch, Michael Schwärzler, M. Eduard Gröller, and Harald Piringer. Litevis: Integrated visualization for simulation-based decision support in lighting design. *IEEE Trans. Vis. Comput. Graph.*, 22(1):290–299, 2016.

[SS00] Andrew Stockman and Lindsay T. Sharpe. The spectral sensitivities of the middle- and long-wavelength-sensitive cones derived from measurements in observers of known genotype. *Vision Research*, 40(13):1711–1737, jun 2000. Data from http://cvrl.ucl.ac.uk/cones.htm. [Online; accessed 27.12.2021].

[Sto98] AW Stockmar. Eulumdat/2-extended version of a well established luminaire data format. In *CIBSE National Lighting Conference*, page 353, 1998.

[Tov08] Martin J. Tovée. *An Introduction to the Visual System*. CAMBRIDGE, July 2008.

[Uni22] University College London - Institute of Ophthalmology. Luminous efficiency. http://www.cvrl.org/lumindex.htm, 2022. [Online; accessed 08.01.2022].

[Vea98] Eric Veach. *Robust Monte Carlo methods for light transport simulation.* Stanford University, 1998.

[ZH16] Liang Zhou and Charles D. Hansen. A survey of colormaps in visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(8):2051–2069, August 2016.