Real-Time Lighting of High-Definition Headlamps for Night Driving Simulation

Nico Rüddenklau, Patrick Biemelt, Sven Henning, Sandra Gausemeier and Ansgar Trächtler
Chair of Control Engineering and Mechatronics, Heinz Nixdorf Institute, University of Paderborn
Paderborn, Germany
Email: {nico.rueddenklau, patrick.biemelt, sven.henning, sandra.gausemeier, ansgar.traechtler}@hni.upb.de

***Abstract*—Introducing high-definition headlamp systems in the automotive industry opens up a wide range of possibilities for improving existing and developing new types of dynamic lighting functions. Due to the complexity and subjectivity of light distributions of modern headlamp systems, simulation-based development is indispensable. Strong restrictions regarding the time of day and weather conditions as well as the hardly reproducible traffic situations are further reasons to shift the testing of such systems as much as possible from reality to simulation. This contribution presents a first real-time simulation of high-definition systems in virtual environments. The simulation results are validated by measuring data and evaluated using validated software for simulating static light distributions at night. The performance of the implementation in terms of computational effort is also discussed. As it turns out, the presented implementation is well suited in terms of appearance and computational performance as a basis for a night driving simulation.**

Keywords*–*high-definition headlamp; real-time simulation; night driving simulation; dynamic light function; shader; lighting.

I. Introduction

The first implementation of a real-time capable simulation of high-definition (HD) headlamp systems with dynamically adjustable light intensity distribution was presented in [1]. This contribution extends the original paper by a sharper resolution of luminous intensity, a broader validation and a more detailed discussion of the implementation.

The introduction of glare-free high beam in 2010 gave a major boost to headlamp development and highlighted the benefits of situation-adaptive lighting functions [2]. It allows driving with permanent high beam, with the headlamp control unit masking the areas classified as glare-critical by an environment camera. Figure 1 shows the schematic function using the example of glare protection for oncoming traffic. Such lighting functions considerably increase driving safety and comfort at night. While this new functionality was initially achieved by mechanically swivelling headlamp components, the trend towards digitalization has made its way more and more into the automotive industry in recent years. Even though the idea of a pixel light was first presented at the PAL (today ISAL) conference in 2001, it took many years for it to be implemented due to technical challenges [3]. Since 2013, mechatronic headlamp systems have increasingly been replaced by light-emitting diodes (LED) matrix solutions [4]. These work without mechanically moved components and realize the dynamics of their light distribution through a considerably higher number of independently controllable light sources with sharply separated solid angle areas of their light cones. The light distributions of the individual light sources add up to the total light distribution weighted by their respective continuously selectable electrical currents. The resulting total light distribution can thus be freely selected via the current values within the limits of the available resolution.

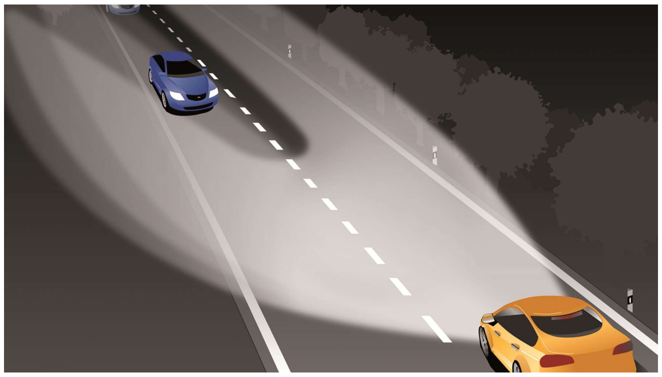
Figure 1. Glare-free highbeam light function cutting off a sharp spatial angle to avoid glaring oncoming traffic. © HELLA

After the matrix headlamps initially equipped with approx. 30 LED segments, a system with 84 LEDs was introduced in 2016 [5]. The increased resolution of the HD84 system allows even more extensive and precise adaptive lighting functions to be realized. The headlamp systems of future vehicles will be equipped with even higher resolutions. A specific example of this is the Liquid Crystal Display (LCD) technology. Only a few LEDs are still used as backlight. The light distribution of these LEDs is then primarily adjusted by a downstream LCD, which filters the light emitted by the LEDs with high local resolution [6]. With this technology, resolutions of several tens of thousand up to a million pixels can be achieved.

Compared to other vehicle components, the development of headlamp systems is characterized by the multidimensional solution space in terms of possible light distributions and the highly subjective factor in their evaluation. These properties require a strongly test-driven development. Because practical testing requires a dark environment, Original Equipment Manufacturers (OEMs) take a huge amount of time and effort to build test tracks that are shielded from ambient light. The light channel of HELLA in Lippstadt, shown in Figure 2, comprises a straight, 140 m long, two-lane road with road markings, which is located in a hall.

Due to the high variability of the light distributions and the situational adaptability of nowadays headlamp systems, static tests such as can be carried out in the light channel are by far not sufficient to cover the available functional spectrum of today's systems. On the other hand, the development and testing of dynamic lighting functions by night driving is no alternative due to safety, time and cost aspects as well as the limited or non-existent reproducibility of environmental influences.

Motivated by this, the idea of a simulation-based virtual night drive was born. In addition to the simulator interface, this requires in a first step the physically precise simulation of

Figure 2. HELLA light channel in Lippstadt. © HELLA

the ego vehicle (vehicle, which defines the driver's view) and in particular the headlight emitted by it, the other road users, the scenery, which includes the road, roadside elements, urban buildings, etc., and the weather conditions in real-time. Based on such a solution, very fast virtual headlamp tests are possible in freely selectable scenarios with complete reproducibility. Even if virtual night drives cannot completely replace the real ones, they allow a considerable reduction of test effort and thus lead not only to a time and cost saving, but also to a safety gain.

The first implementation of a real-time simulation of headlamp systems was presented by Lecocq et al. [7]. This initial realization implements per-vertex-lighting, which means that the lighting model is only evaluated at the vertices of scene objects. Subsequently, the pixel colors are determined by interpolation over all pixels on the basis of their surrounding vertices. The quality of the simulated light distributions therefore depends strongly on the tesselation of the scene objects. Berssenbrügge et al. present an approach based on per-fragment-lighting that decouples tesselation from the resolution of light distribution [8]. In their contribution, the lighting model is evaluated for each pixel of the output device, whereby inaccuracies by interpolations can be completely eliminated. A comparable concept is presented in [9]. It utilizes a proprietary development for the simulation of night driving with additional functionalities. For example, they provide a bird view including a distance grid to assess the light distribution with regard to range, width, homogenity and contrast sensitivity. Based on such simulations, various publications exist for testing dynamic lighting functions. In [10], Kemeny et al. present the simulation of the cornering light function within the established driving simulation software SCANeR. Like Berssenbrügge et al. in [11], they implement predictive cornering light, which calculates the ideal light distribution on the basis of navigation data and vehicle speed. As it turns out, this concept leads to better results than making the tilting of the light cone dependent only on the steering angle. Next to the cornering light function, Berssenbrügge et al. are also testing an advanced leveling light system in their simulation environment [12]. Here, as well, it can be seen that the development and testing of new lighting functions and the test-driven optimization of their design parameters can be considerably accelerated by virtual night driving simulation. With active safety light Knoll et al. present the simulative testing of a new light function for highlighting possible escape ways in risky driving situations [13]. This contribution represents a particularly interesting application case for simulation-based development, as the dangerous driving maneuvers for testing active safety light can only be performed under enormous safety precautions, or in some cases not at all.

All of the implementations mentioned above have in common that the light distributions of the light sources used are static. In concrete terms, this means that they are independent of time. Dynamic functions are mapped exclusively by means of switching light sources on or off and changing their orientation angles. HD headlamps, however, realize lighting functions in a totally different way. In this paper we present the first real-time simulation and its underlying modelling of HD headlamp systems. Their outstanding features and their abstract modelling are described in Section II. In Section III the shader-based implementation of the light simulation is discussed. The first Subsection III-A transforms a mathematical representation of a luminous intensity distribution (LID) into concrete structures of computer graphics. Based on this, the light rendering procedure is divided into the determination of the total light distribution as a weighted sum of the distributions of the individual light sources in the headlamp (Section III-B) and the illumination of the scene based on the overall distribution (Section III-C). In Section IV, the simulation results are validated with measurement data (Section IV-A), evaluated using a validated reference simulation tool (Section IV-B) and their performance on the utilized test hardware is examined (Section IV-C). The last sections provide a conclusion as well as an outlook for future work.

II. HD Headlamp Systems

HD systems are characterized by a great number of independent controllable light sources. Their illumination areas concentrate on sharply defined solid angle intervals with small overlapping areas. The total light distribution of such a headlamp results from the superposition of all the individual light distributions. This architecture enables the generation of highly dynamic overall light distributions by independently controlling the electrical current of each individual light source

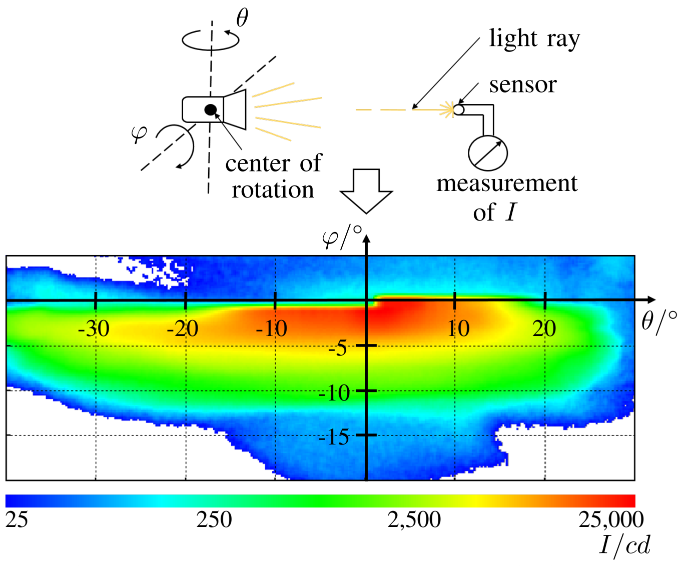

Figure 3. Low-beam distribution of luminous intensity I of left HD84-Matrix-LED headlamp measured in Candela [cd].

in the headlamp. The variety of the representable light distributions is limited only by the resolution of the headlamp, which can range from approx. 100 to several 10,000 pixels, and the permissible values of the electrical current depending on the light technology used [14].

For testing the simulation technique presented in this contribution, the HD84-Matrix-LED headlamp developed by HELLA is used. The actual HD component of this headlamp is realized by a matrix of 84 LEDs, which can be supplied individually with continuously adjustable electrical current. The individual light sources are mounted in three lines, with the lowest line (line 1) illuminating the close range in front of the vehicle, the middle line (line 2) focusing on the effective range of the low beam and the upper line (line 3) dedicated to functions in the high beam range (see upper area of Figure 4). To compensate for the relatively low resolution of 84 pixels, the illumination range of the HD module is limited to the solid angle range with the greatest variability requirements. Accordingly, additional light sources are provided to illuminate the vehicle front area, the sides and to support the high beam. Although the additional light sources are not visualized in Figure 4 for reasons of clarity, they are processed in the same way as the 84 LEDs of the matrix. In total, the HD84-Matrix-LED headlamp therefore contains 95 light sources.

To describe the characteristics of a light source, in lighting technology so-called luminous intensity distributions (see lower area of Figure 3) are used, which describe the luminous intensity depending on the direction of radiation [15]. The luminous intensity I measured in Candela $[cd]$ describes the radiation characteristic of a light source, by resolving the radiant power or luminous flux Φ (unit: Lumen $[lm]$) emitted by it in relation to the through-radiated solid angle Ω (unit: Steradiant $[sr]$) according to

$$I = \frac{\Phi}{\Omega} \quad [cd] = \left[\frac{lm}{sr}\right] \tag{1}$$

for a homogeneous luminous intensity within the considered solid angle. Using spherical coordinates the solid angle Ω is related to the polar angle φ and the azimuth angle θ by

$$\Omega = \int_{\theta_1}^{\theta_2} \int_{\varphi_1}^{\varphi_2} sin(\varphi) d\varphi d\theta, \tag{2}$$

whereby $[\varphi_1, \varphi_2]$ is the considered angle interval around the horizontal axis and $[\theta_1, \theta_2]$ corresponds to the interval around the vertical axis, which forms a square cutout on a spherical surface. [16]

In context of headlamps, the luminous intensity varies greatly with the radiation direction. By shrinking the solid angle to an infinitesimal area, the dependence of the luminous intensity I on the already mentioned angles φ and θ can be found by

$$I(\varphi, \theta) = \frac{d\Phi(\varphi, \theta)}{d\Omega} = \frac{d\Phi(\varphi, \theta)}{sin(\varphi) d\varphi d\theta}. \tag{3}$$

In the lower half of Figure 3, the low-beam distribution of the left HD84 headlamp is shown. The direction of radiation is defined by the polar angle φ and the azimuth angle θ. Their zero-position equals the direction of travel with regard to their mounting position in the vehicle. Even though Figure 3 does not show the entire angle range for better recognition, data for $\varphi \in [-29.9°, +19.9°]$ and $\theta \in [-89.9°, +89.9°]$ is available for all light distributions used in the simulation. The values of luminous intensity are color-coded with a logarithmic scale, as is usual for those diagrams. They vary over five orders of magnitude up to $25.000cd$. For high-beam distributions, the range is even greater. Therefore, this is also referred as high dynamic range information.

These luminous intensity distributions can be obtained for an existing headlamp by a goniophotometer-measurement [17]. In this case, the headlamp is mounted in a goniometer construction, by which it can be swivelled around its vertical and horizontal axis (see upper left area of Figure 3). The luminous intensity is usually measured by a fixed sensor at a distance of 25m (see upper right area of Figure 3). This distance serves in particular to maintain the photometric limit distance, under which the light source to be measured can no longer be approximated as a point source. The rotation angle around the horizontal and vertical axes of the goniometer construction correspond to the polar and azimuth angles φ and θ of the luminous intensity distribution.

Alternatively, the light distribution can be determined on the basis of computer models of the headlamp using ray tracing methods [18] and elaborate offline simulations, which were introduced at first by Neumann and Hogrefe in [19]. The quality of the resulting light distribution depends primarily on the model quality. Even though the characteristics of a real headlamp can never be exactly reproduced, this variant is particularly interesting in early development phases before the construction of a prototype.

In order to simulate the light distribution of a HD headlamp in any situation, the characteristics of the emitted light must be known for each individual light source in it. Therefore, the luminous intensity distribution is measured for each light source, in concrete terms 95 times, of the headlamp and especially for each LED of the HD84-Matrix. In the middle area of Figure 4, it can be seen exemplary intensity distributions of LED 1 (line 1, left), LED 45 (line 2, middle) and LED 84 (line 3, right). From the illustrations it becomes clear that each LED emits exclusively in a certain solid angle. As the lower line is responsible for the close range, the light intensity center is also found at negative polar angles. This regularity can also be applied to the other lines. Similarly, it becomes visible that the horizontal arrangement of the LEDs within the matrix corresponds to the horizontal arrangement of their light intensity centers. In addition, it is noticeable that the light distributions of the lateral light sources extend over larger areas than it is the case for light sources in the middle. Decisive for this are the especially high variability requirements placed on the central area in front of the vehicle. This area must therefore be resolved at a particularly high resolution in relation to the overall light distribution.

The individual light distributions are measured with running the corresponding LEDs at full power. During normal operation, the LEDs can be dimmed independently of each other in the range of 0-100% by specifying their electrical currents. In the lower left area of Figure 4 the low-beam distribution of the left HD84-Matrix headlamp is shown. The outline of the light distribution is also referred to as the light-dark boundary in the context of automotive headlamps. A characteristic feature of a low-beam light-dark boundary is the step in the range of $\theta = 0°$ in the light distribution. For negative values of θ, the luminous intensity in the direction of

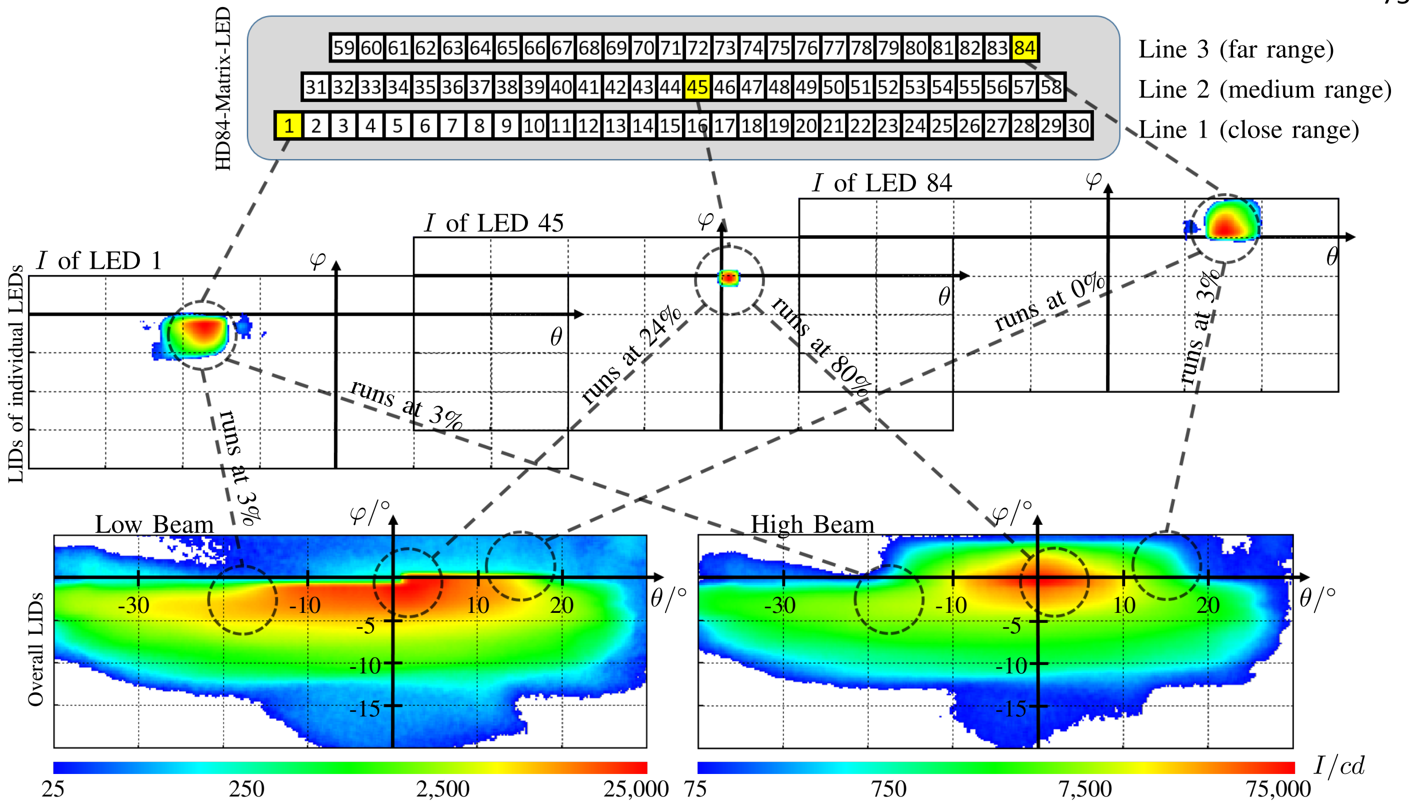

Figure 4. Luminous intensity distributions L_k of individual light sources (in this example for $k = 1, 45, 84$) and their electrical current weighted compositions to overall distributions, in example low-beam distribution (left) and high-beam distribution (right).

positive φ is cut off earlier than is the case for positive θ. In other words, the lane of oncoming traffic is illuminated less than the lane of one's own. This ensures that oncoming traffic is not glared, but simultaneously there is a good view of one's own lane.

In contrast to conventional headlamps, in which the geometry of the reflector shade makes it possible to achieve a low-beam light distribution using a single light source, the overall light distribution of an HD headlamp is obtained by adding all the light distributions of its individual light sources. In order to achieve the desired light-dark boundary of low beam, the headlamp control unit supplies all light sources with suitable dimming values. In the concrete example, all the LEDs in line 3, which is intended for the far range, are completely switched off. Therefore, the light distribution of LED 84 does not influence the overall light distribution of the low beam (see dotted lines between the individual and overall light distributions in Figure 4). In line 2, LEDs are primarily energized in the right half. LED 45, which is located directly at the step of the light-dark boundary, is operated at 24% of its maximum light output, for example. In line 1, all LEDs are energized, whereby the LEDs in the edge areas are dimmed more strongly than those in the middle area. This explains the low value of 3% of LED 1. When driving around bends, the center of the light distribution could be swivelled in the direction of the curve by adjusting the electrical current values to better illuminate the road.

The advantages of generating a light distribution from many small blocks can be easily seen. While the characteristic of the light distribution in conventional headlamps is fixed by the geometry of the reflector, it can be varied over a wide range in HD headlamps. Two factors limit the variety. On the one hand, this is the number of pixels and, on the other hand, the step size of the dimming value discretization, which plays a minor role. In the example of the HD84 module, the dimming values are coded by 6 bits. This results in a step size of about 1.6% relative to the respective maximum light output. The theoretical total number of light distributions is unimaginably high ($(2^6)^{84} \cdot 2^{16}$, 84 pixels in the HD84 module with 2^6 possible dimming values and 16 non-dimmable light sources).

The lower right part of Figure 4 shows the high beam distribution of the headlamp. The step in the light-dark boundary disappears and is replaced by a high beam cone which shifts the center of light intensity to positive φ. In addition, the light distribution is roughly symmetrical, since unlike in the case of low beam, no glare suppression of the opposite lane is desired. In addition, the headlamp shines brighter by a factor of 3, as can be seen from the false color scales at the bottom of Figure 4. Just as in the case of the low beam, the high beam distribution shown is generated exclusively by adjusting the dimming values. Line 3 is now active and the left half of line 2 is also energized. In addition, the light output of all LEDs is increased in order to achieve the overall higher light intensity. For example, the power of LED 45 is raised from 24% at low beam to 80% at high beam mode.

III. Implementation of HD Headlamp Simulation

After discussing the essential properties and functional principles of high-resolution headlamp systems, this section will focus on the real-time rendering of the headlamp light in the virtual scene. The implementation presented here sets the basis for virtual night drives with high-resolution systems and thus creates the possibility to develop and test these

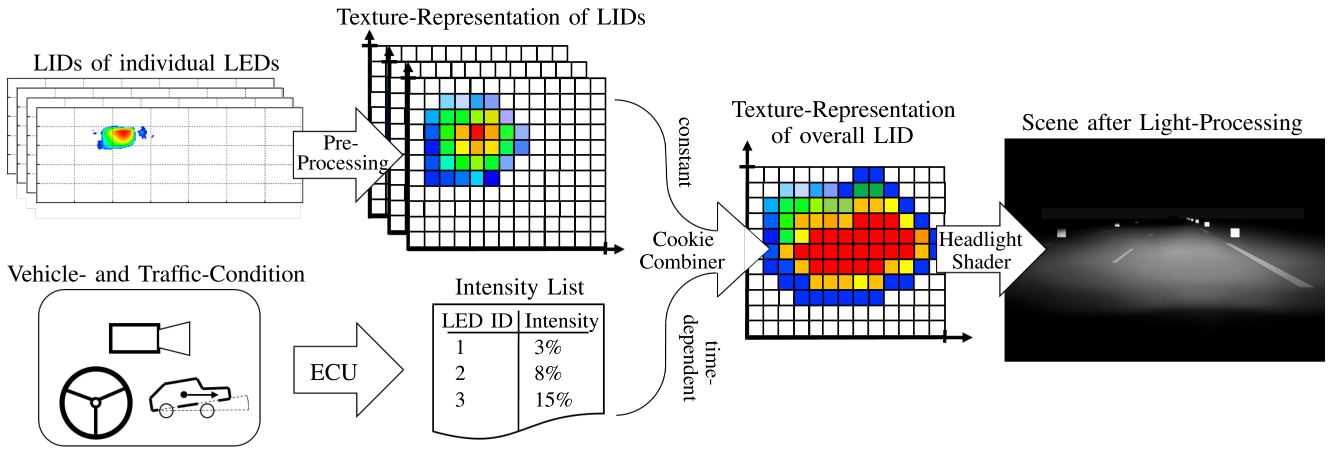

Figure 5. Functional flowchart of HD headlamp rendering.

simulation-based. For implementing the visualization of the night drive, we used the development environment Unity3D (Version 2017.3.1f1 [20]).

Figure 5 shows the logical flow of the presented implementation. The first step is to convert the previously measured luminous intensity distributions of the individual headlamp light sources into a data format that can be efficiently managed under the real-time requirements of driving simulation. Since this information does not vary over time, it is sufficient to perform this conversion as part of preprocessing. In concrete terms, each luminous intensity distribution is converted into a texture, as illustrated in the upper left area of Figure 5. This procedure only has to be carried out once for each headlamp type and is then available for any simulations. Details on this process will follow in Section III-A.

In addition to the individual light distributions, the relative current values of all light sources are necessary to determine the overall light distribution. The entirety of their values at a given point in time is referred to below as the intensity list. These are specified in the real vehicle by the headlamp control unit, depending on the traffic situation detected by the sensors (see lower left area of Figure 5). This updates the current values with a frequency of 50Hz. Within the framework of the implementation, the behaviour of the control unit is not simulated, since it will later be integrated into a hardware-in-the-loop simulation. In this respect, the current values can be assumed to be given, whereby it must be ensured that the overall light distribution can be determined within a reasonable time due to the high update frequency.

Once both, the individual light distributions and the temporary intensity list, are available, the total light distribution can be calculated. This can be achieved by adding the individual light distributions weighted by the relative current values. Due to the time dependence of the current values, the total light distribution also varies with time, so that it must be recalculated with the update rate of the intensity list. To be able to perform this calculation 50 times per second, it is carried out in a highly parallel manner using a shader on the GPU. All shaders presented here are implemented in Cg (C for graphics) [21]. This shader is called Cookie Combiner and is described in more detail in Section III-B. The output type of the Cookie Combiner corresponds to the types of the individual light distributions. It is also a texture that encodes the luminous intensity values with respect to the spatial angles.

Finally, all information is available to render the virtual scene. Here, deferred shading was used [22]. While the light calculations in the more established forward rendering are performed individually for each object of the scene, in deferred rendering all objects are first rendered in the base pass under exclusion of light influences in the so-called G-Buffer (Geometry Buffer). In the subsequent lighting pass, the light calculations are applied to the G-Buffer only once per light source, which generally requires considerably fewer shader executions than it is the case in forward rendering. Besides possible standard light sources of the scene, the lighting pass also activates the Headlight-Shader intended for displaying headlamp light (see last step of Figure 5). The execution of the Headlight-Shader is downstream of the standard shader for deferred shading and is integrated into the Programmable Rendering Pipeline by a Command Buffer [23]. This uses the total light distribution determined by the Cookie Combiner and, using a lighting model, determines the color pixel by pixel, which results from the object and light properties as well as the geometric relationships.

A. Digital Representation of LIDs

In order to be able to precisely reference the relevant elements of the light distributions of HD systems in the following sections, first of all these are described more formally. A discretized light distribution can be interpreted as a matrix whose dimensions depend on the considered angular range $[\varphi_l, \varphi_u]$ or $[\theta_l, \theta_u]$ and the corresponding resolution $\Delta\varphi$ or $\Delta\theta$ in horizontal or vertical direction, whereby $M = \frac{\varphi_u - \varphi_l}{\Delta\varphi}, N = \frac{\theta_u - \theta_l}{\Delta\theta}$ with $M, N \in \mathbb{N}$ must apply. The discrete value φ_m with $0 \leq m \leq M$ or θ_n with $0 \leq n \leq N$ then refers to the horizontal angle $\varphi_l + m \cdot \Delta\varphi$ or the vertical angle $\theta_l + n \cdot \Delta\theta$. The light distribution L_k of the light source k with $k \in \{1, K\}$ of an HD headlamp with a total of K individual light sources then has the form $L_k \in \mathbb{R}_{\geq 0}^{M \times N}$. In the concrete case, the values $\varphi_l = -29.9°, \varphi_u = +19.9°, \theta_l = -89.9°, \theta_u = +89.9°$ apply to the measured angular ranges. Resolving both angles with $\Delta\varphi = \Delta\theta = 0.2°$ results in matrices of $M = 250$ rows and $N = 900$ columns. In addition, the headlamp examined has a total of $K = 95$ light sources.

The entry $l_k(m, n)$ of row m and column n of the L_k matrix now contains the luminous intensity of the light source

k at full power in the discretized direction of the vertical angle φ_m and the horizontal angle θ_n in Candela. For example, the entry $l_{17}(100, 400)$ contains the luminous intensity of the light source with index 17 in the direction of the vertical angle $\varphi_{100} = \varphi_l + 100 \cdot \Delta\varphi = -29.9° + 100 \cdot 0.2° = -9.9°$ and the horizontal angle $\theta_{400} = \theta_l + 400 \cdot \Delta\theta = -89.9° + 400 \cdot 0.2° = -9.9°$.

After defining the light distributions of individual light sources, the aggregation of these to the total light distribution L can be formulated. For this purpose, a system is defined whose input variables represent the relative electrical current values $i_k \in [0, 1]$ of the individual light sources normalized to their maximum value. In contrast to the matrices $L_1, \ldots, L_K$, these values are time-dependent signals coming from the headlight control unit at 50 Hz. The output of the system is the resulting light distribution of the headlamp and thus a weighted composition of all individual light distributions. Formally, the output variable corresponds to a L matrix whose dimensions are identical to the dimensions of the individual distributions $L_1, \ldots, L_K$. The current overall light distribution can be formulated as

$$L(t) = \sum_{k=1}^{K} i_k(t) \cdot L_k \text{ with } L(t) \in \mathbb{R}_{\geq 0}^{M \times N}. \tag{4}$$

The time-dependent matrix L contains all information describing the light emitted by the headlamp at time t, which constitutes the basic information for simulation purposes.

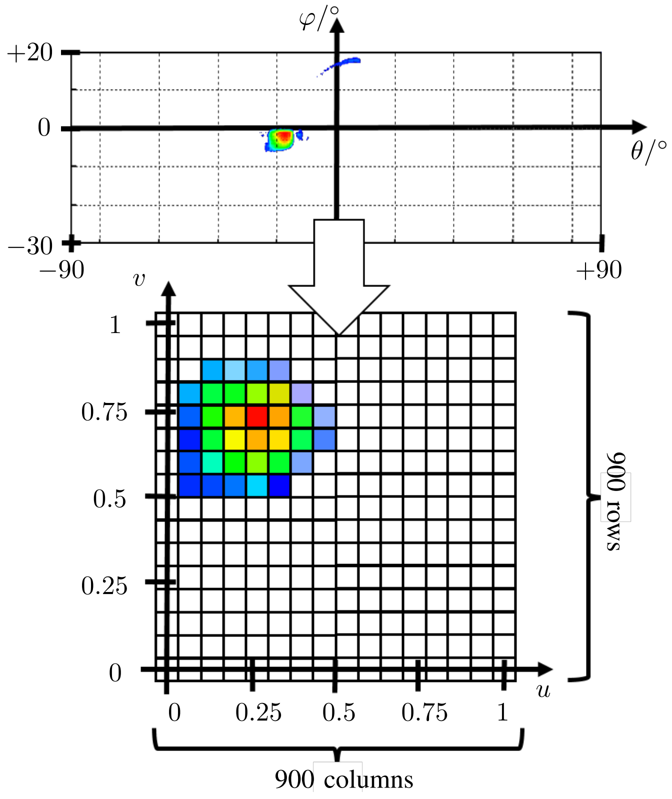

Figure 6. Digital representation of a luminous intensity distribution.

To represent the formally introduced matrices in the computer graphics field, textures are used. A texture is a data format established in computer graphics which was originally intended for the realistic coloring of scene objects. In this role it can be understood as an image which is placed like a sticker on the surface of a three-dimensional scene object. The individual entries of the texture encode the color at the respective place then typically in the 3 RGB channels (red, green, blue) and a 4th transparency channel (α channel). In this case, textures are used to encode light intensity distributions.

Now some technical details of the conversion of luminous intensity distributions to textures shall be described, which are relevant for the later explanation of Cookie Combiner- and Headlight-Shader. To minimize interpolation inaccuracies, we use textures with the same row and column numbers as we have for the discretized luminous intensity distributions (250 rows, 900 columns). Since the textures have to be square for technical reasons, it is necessary to increase the number of rows from 250 to 900. For this purpose, the vertical angle range is artificially increased from [-29.9, +19.9] to [-89.9, +89.9] and the luminous intensity values at the non-measured angles are set to 0. As a consequence, the texture contains 900x900 data points, which are also called texels. Using the same resolution as the measured luminous intensity distributions, each texel corresponds directly to a measuring point of the distribution. In classic color textures the values of the R, G, B and A channels of a texel are encoded with 8 bit fixed-point, whereby each texel contains 32 bit information (RGBA32 texture). A luminous intensity distribution, in contrast, contains only one dimensional information - the luminous intensity. Here the use of a RGBA32 texture would be inefficient, since a precision of 8 bit is on the one hand too low for the high dynamics of a luminous intensity distribution and on the other hand the remaining 3 channels remain unused. Instead, we use an RFloat texture format. This supports only one channel (R-Channel) and uses for this the entire 32 bit available to the corresponding texel. In addition, the value is coded in floating point format, which also benefits the high dynamics of the luminous intensity values.

Another important point is the indexing of textures. Figure 6 can be used for understanding. Texture coordinates or uv coordinates are always used normalized. In other words, a texel is not addressed by its absolute row and column number in the texture, but by the values divided by their total numbers. $u \in [0, 1]$ represents the horizontal axis (corresponds to the column number) and $v \in [0, 1]$ the vertical axis (corresponds to the row number). The texel at address $(u, v) = (0, 0)$ thus contains the luminous intensity in the direction of the vertical angle $\varphi = -89.9°$ and $\theta = -89.9°$ (respectively texel at $(u, v) = (1, 1)$ corresponds to the luminous intensity value at $\varphi = +89.9°$ and $\theta = +89.9°$).

Since u and v can be specified from a continuous value range, u, v pairs can also address places of the texture that do not match exactly to a texel but lie in the intermediate areas. Such accesses are possible and are answered with the bilinear interpolation between the surrounding texel values, which is visualized in Figure 7. In general, the return r of a texture access at the coordinates (u, v) can be traced back to the luminous intensity distribution as follows:

If the uv coordinates correspond exactly to a texel of the texture, its value can be returned unchanged. The condition for this is that $u \cdot M$ and $v \cdot N$ are elements of the natural numbers. In the other case, the neighboring texels of the access coordinates (u, v) must be found first, which are designated by t_{ll}, t_{lr}, t_{ul} and t_{ur} in Figure 7. As also noted in Figure 7, these

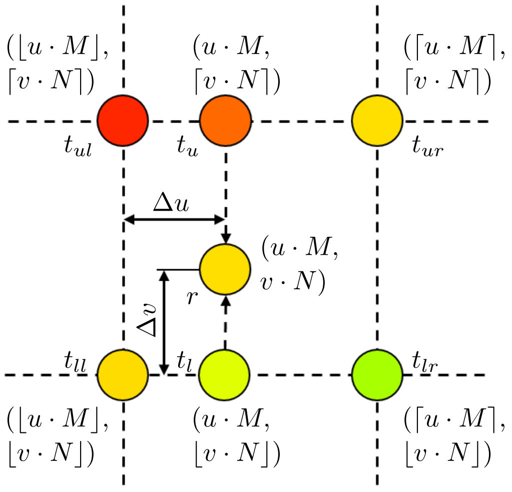
$(\lfloor u\cdot M\rfloor, \lceil v\cdot N\rceil)$
$(u\cdot M, \lceil v\cdot N\rceil)$
$(\lceil u\cdot M\rceil, \lceil v\cdot N\rceil)$
t_{ul}
t_u
t_{ur}
Δu
$(u\cdot M, v\cdot N)$
r
Δv
t_{ll}
t_l
t_{lr}
$(\lfloor u\cdot M\rfloor, \lfloor v\cdot N\rfloor)$
$(u\cdot M, \lfloor v\cdot N\rfloor)$
$(\lceil u\cdot M\rceil, \lfloor v\cdot N\rfloor)$

Figure 7. Bilinear interpolation on texture access at coordinate (u, v).

can be addressed by simply rounding $u \cdot M$ and $v \cdot N$ up and down. Afterwards, one interpolation is performed between the upper neighbor texels and one interpolation between the lower neighbor texels along the u-axis, yielding the values t_l and t_u. Formally these result to

$$\begin{aligned} t_l &= lerp(t_{ll}, t_{lr}, \Delta u) \text{ and} \\ t_u &= lerp(t_{ul}, t_{ur}, \Delta u) \\ &\text{with } \Delta u = u \cdot M - \lfloor u \cdot M \rfloor \\ &lerp(v_1, v_2, c) = v_1 \cdot (1 - c) + v_2 \cdot c, \\ &v_1, v_2 \in \mathbb{R}, c \in [0, 1] \end{aligned} \tag{5}$$

The return value r is now obtained from the bilinear interpolation by interpolating the interpolations along the u coordinate again along the v coordinate:

$$\begin{aligned} r &= lerp(t_l, t_u, \Delta v) \\ &\text{with } \Delta v = v \cdot N - \lfloor v \cdot N \rfloor \end{aligned} \tag{6}$$

Finally, in order to clarify the relationship between textures and the original luminous intensity distributions, the way back from a texture T to luminous intensity l in the direction of vertical angle $\varphi \in [\varphi_l, \varphi_u]$ and horizontal angle $\theta \in [\theta_l, \theta_u]$ is shown:

$$\begin{aligned} l(\varphi, \theta) &= T(u, v) \text{ with} \\ u &= \frac{\varphi - \varphi_l}{\varphi_u - \varphi_l}, \varphi \in [\varphi_l, \varphi_u] \\ v &= \frac{\theta - \theta_l}{\theta_u - \theta_l}, \theta \in [\theta_l, \theta_u] \end{aligned} \tag{7}$$

whereby $T(\cdot, \cdot)$ means a texture lookup at T at the u and v coordinates given in this order as operands.

B. Cookie Combiner-Shader

Once a way has been found to digitally map luminous intensity distributions to a graphics card compatible manner, the next step is to determine the total light distribution from the individual ones. The result can then be used to adjust the light intensity in the lighting pass, more precisely in the Headlight-Shader executed in it (see Section III-C), depending on the direction. Vividly, this realization is comparable to a dynamic transparency film, by which a homogeneous light source is filtered to produce the desired radiation characteristic. Texturing of light sources to vary the luminous intensity in different beam directions is already established. Such light textures are called cookies, explaining the name of the Cookie Combiner-Shader. Its task is to combine the textures of the individual light distributions into a total light distribution texture according to Equation (4). Figure 9 illustrates the data flow of the combining procedure on a mathematical level.

The implementation of this calculation as a shader enables the highly parallel execution on the graphics card, which is necessary to fulfill the real-time requirements. Within the scope of this work, only vertex and fragment shaders are used. Vertex shaders process vertices and the information associated with them of the three-dimensional geometries in the scene. Afterwards the scene is rasterized object by object and transformed into a two-dimensional image. Then fragment shaders work on the fragments of this image and determine the pixel colors. Comprehensive information about shaders can be looked up in [23].

In the case of the Cookie Combiner, the vertex shader is trivial, since only two-dimensional data (luminous intensity distributions) is processed and no vertex operations are performed. Its whole logic is placed in the fragment program. The operations are visualized by the following pseudo code on the one hand and by Figure 8 in a graphical way on the other hand. While in the rendering pipeline a fragment program writes into the screen output and the pixels of the screen slip into the role of the fragments, we define a texture, which is reused in the rendering pipeline in a later step, as the render target for our application. More precisely, the render target of the Cookie Combiner-Shader is a texture T_{comb} with the same type and dimensions as the textures of the individual light distributions - a scalar floating point texture with 900x900 texels.

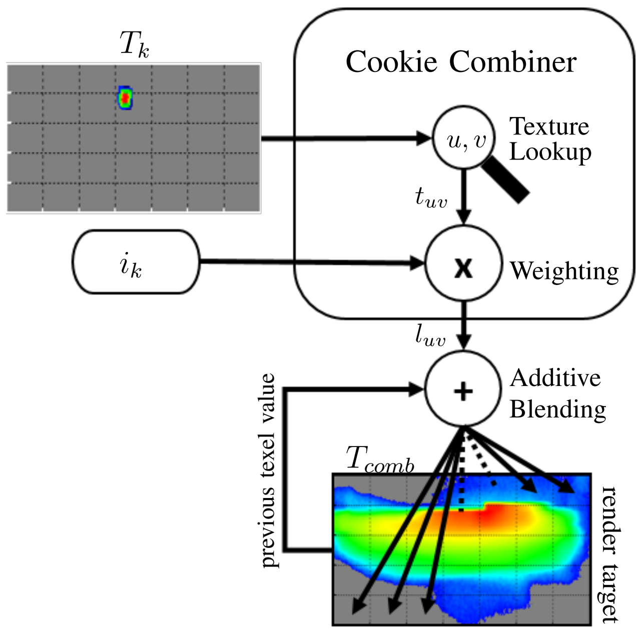

Figure 8. Logic diagram of the technical Cookie Combining Procedure.

Require: $u, v \in [0, 1]$ normalized coordinates of current render target texel, T_k scalar floating point texture corresponding to luminous intensity distribution L_k and relative current value $i_k \in [0, 1]$ of individual light source k

```
1: $t_{uv} = T_k(p_{target}.u, p_{target}.v)$                // texture lookup
```

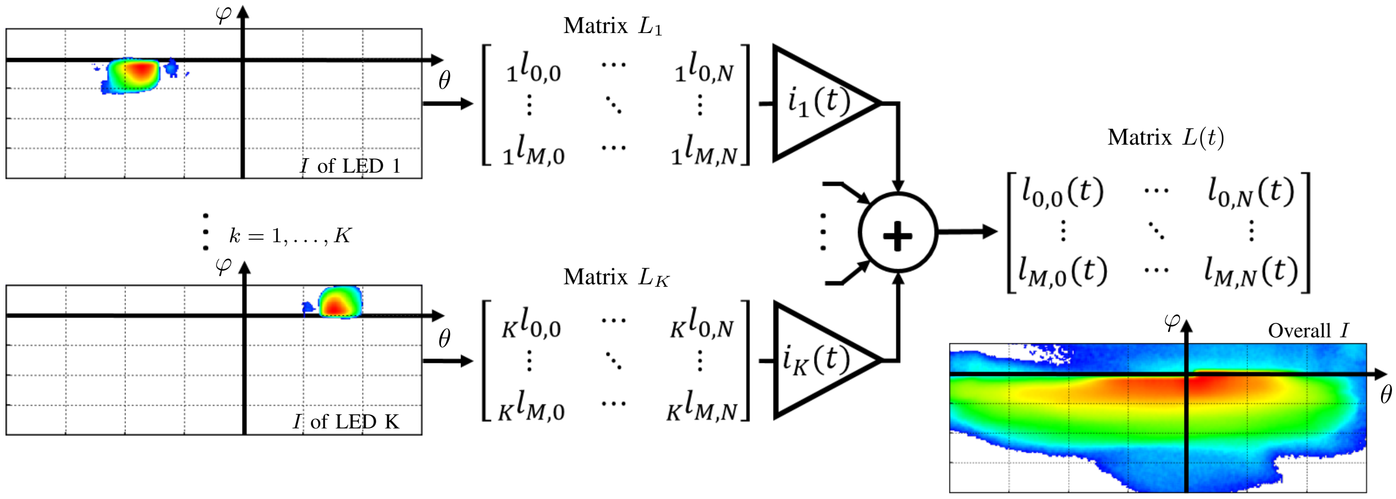
Matrix L_1
$\begin{bmatrix} {}_1l_{0,0} & \cdots & {}_1l_{0,N} \\ \vdots & \ddots & \vdots \\ {}_1l_{M,0} & \cdots & {}_1l_{M,N} \end{bmatrix}$
$i_1(t)$
I of LED 1
$k = 1, \ldots, K$
Matrix L_K
$\begin{bmatrix} {}_Kl_{0,0} & \cdots & {}_Kl_{0,N} \\ \vdots & \ddots & \vdots \\ {}_Kl_{M,0} & \cdots & {}_Kl_{M,N} \end{bmatrix}$
$i_K(t)$
I of LED K
Matrix $L(t)$
$\begin{bmatrix} l_{0,0}(t) & \cdots & l_{0,N}(t) \\ \vdots & \ddots & \vdots \\ l_{M,0}(t) & \cdots & l_{M,N}(t) \end{bmatrix}$
Overall I
φ
θ

Figure 9. Data flow of Cookie Combining during a single frame.

```
2: $l_{uv} \leftarrow i_k \cdot t_{uv}$        // weighting
3: return $l_{uv}$        // additive blending into render target
```

The fragment program of the Cookie Combiner-Shader expects two parameters (see upper left area of Figure 8). One is the texture T_k, which represents the light intensity distribution L_k of the individual light source k. The other parameter represents the relative intensity i_k with which this light source is currently operated. Its code is executed for each texel of the render target T_{comb} in parallel (see lower area of Figure 8), whereby the individual threads can be distinguished by further inherent parameters u and v constituting the normalized coordinates of the thread specific texel of the render target.

In the first step within processing, the shader reads the light intensity value t_{uv} of the given individual light texture T_k at position (u, v) corresponding to the texel to be written to in the render target (line 1). This simple method of addressing is made possible because both the individual light distributions and the target texture encode the light intensity distribution over the same angular ranges, even if their resolutions were different. Considering the intensity with which the light source k is currently operated, the multiplication of the maximum luminous intensity t_{uv} with the relative current value i_k takes place at line 2. Even if the relationship between the electric current and the luminous intensity is not linear, it can be assumed to be linear here by compensating the non-linearity with the real control unit. The product l_{uv} as actual luminous intensity in the direction represented by the u and v coordinates of the current texel according to 7 represents the output of the shader (line 3).

After completion of the shader operations for all texels, the target texture contains the contribution of the light source k within the total light distribution, whereby a single component of the sum in the data flow visualized by Figure 9 is mapped. In order to obtain the complete light distribution of the headlamp, all texels of the render target T_{comb} are first initialized with 0. Then the Cookie Combiner is applied to the target texture repeatedly with iterating through all individual light sources by changing the parameters T_k and i_k. The rendering is done with additive blending (see Figures 8 and 9), so the previous values in the render target are not overwritten by the returns of the shader but added to it [23].

After applying the Cookie Combiner to all individual light sources $k = 1, \ldots, K$, the render target T_{comb} contains the total luminous intensity distribution, in which the texels can still be interpreted as in Section III-A. Due to its dependence on the time-varying relative currents $i_1, \ldots, i_K$, this texture is only valid for the current time step. The K times execution of the Cookie Combiner must therefore take place with the update rate of the headlight control unit for each simulated headlamp, which is 50 Hz. The determined total light distribution represents a central parameter of the Headlight-Shader presented below.

C. Headlight-Shader

The implementation of the Headlight-Shader is much more extensive. Berssenbrügge et al. solved a similar problem for static light distributions in [8] by using a built-in spotlight and mapping the light distribution to its cookie scheme. In this contribution, the lighting is done by a custom shader using the deferred shading pipeline, whose implementation is oriented to Unity's built-in lighting shader for deferred rendering. Moreover, High Dynamic Range (HDR) Rendering is used. This technique allows a more detailed resolution of the color information by using higher memory resources, whereby especially with high-contrast images their details are preserved in the best possible way and come closer to the perception of the human eye [24]. Even if the color information has to be reduced to 255 brightness levels (8 bit per color channel) for output on a monitor, the colors can be dynamically scaled instead of just setting too bright areas to white, as is the case with Standard Dynamic Range (SDR). Especially in the context of headlamp simulation, the use of HDR colors leads to visibly higher quality images.

Figure 10 illustrates a very simplified pipeline run in deferred rendering. The concept of deferred shading is characterized by the strict division of rendering into a first step of projecting the 3-dimensional scene information to a 2-dimensional image (G(eometry)-Buffer) and the subsequent lighting on the basis of this G-Buffer. The main advantage here is that lighting only has to be applied to all pixels of the 2-dimensional G-Buffer instead of every object in the scene, as is the case with conventional forward rendering. Thus, the computational complexity of a scene with m objects and n lights can be reduced from $\mathcal{O}(m \cdot n)$ to $\mathcal{O}(m + n)$.

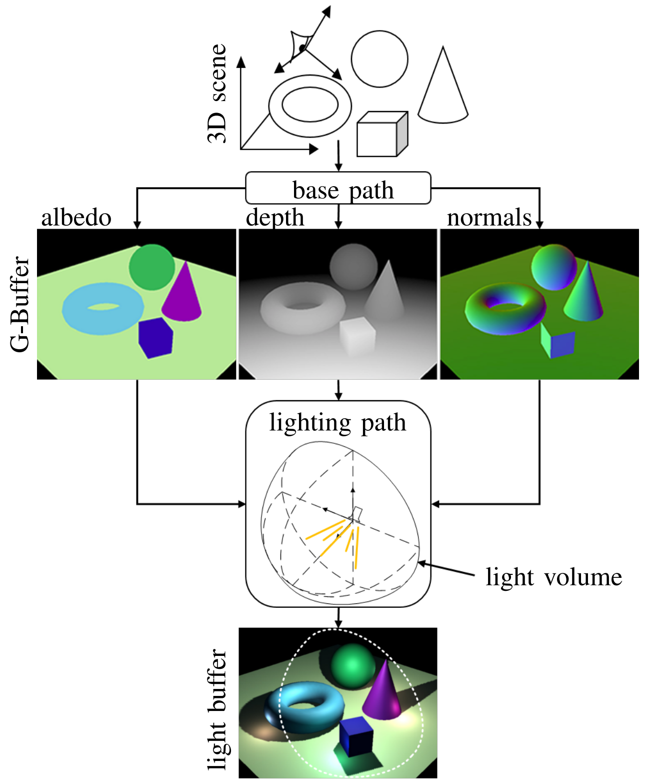

Figure 10. Deferred rendering pipeline.

At first, the 3-dimensional scene objects are rendered in the base path into the G-Buffer, whereby the scene information is reduced to two dimensions. Since the lighting calculations are done in a deferred step, it is not possible to render directly the final pixel colors. Therefore, all relevant information for later lighting calculations has to be contained in the G-Buffer. In detail, this could be color information (often distinguished in colors for ambient, diffuse and specular components, see left image of middle left area of Figure 10), depth values in terms of the distance between camera and object (encoded by grayscale at center area of Figure 10) or surface orientations defined by normal vectors (normal directions color-coded at middle left area of Figure 10).

Given the G-Buffer, the lighting pass can be initiated. This is the pipeline step, where the integration of Headlight-Shader next to Unity's standard shader takes place. Using Unity's Graphics Command Buffer [26], its integration is done subsequently to the 'Lighting'-Block in Figure 11, where standard lighting by the built-in shader takes place. A Command Buffer holds list of rendering commands, which can be injected between all boxes in Figure 11 to the conventional rendering pipeline. Integrating the Headlight-Shader this way instead of modifying the built-in shader, preserves compatibility to all standard light sources in the scene.

Within the lighting pass, there is one shader called per light source. Their common render target is the light buffer, visualized in the lower area of Figure 10. After finishing all light shaders, the light buffer contains the final scene image, ready for displaying on the output device. While the standard shader is called for Unity's built-in lights (directional, point or spot light), the Headlight-Shader is called for each headlamp. In both cases, for each light source there is a predefined light

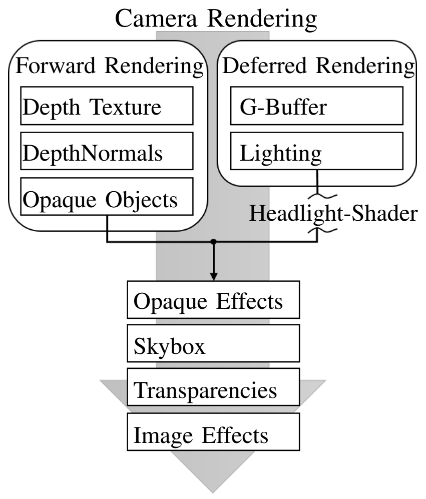

Figure 11. Injection of Graphics Command Buffer in Unity's rendering pipeline (created in accordance with [25]).

volume describing the volume, which is potentially affected by it, relative to the light position and orientation. For this purpose, we choose a half sphere, approximated by a mesh of 65 vertices (see Figure 13), in contrast to a pyramid, used by Unity's spotlight. With reference to the luminous intensity distributions measured at an angular range of 180° in horizontal and vertical directions, a half sphere is much more suitable for this application. The light source is located in the spherical center and is oriented vertically to the plane half-spherical surface in the direction of the curvature (see center area of Figure 10). While processed by the vertex program of the Headlight-Shader, the light volume is rotated and translated to get its correct position in the scene. This ensures that the subsequent rasterization of the light volume can be understood as a marking of the pixels of the G-Buffer that may have been influenced by the light, which are illustrated by the white dotted outline in the light buffer of Figure 10. The following fragment program of the Headlight-Shader operates on all of these marked fragments of the light buffer, calculating their colors by accessing relevant information of the illuminated fragment stored in the G-Buffer. To consider all light sources, whose light volumes cover the current light buffer pixel, the return values of all light shaders are written in the light buffer by additive blending. After passing through all shaders in the lighting pass, the light buffer contains the completely rendered scene image.

At this point, the vertex and fragment program of the Headlight-Shader will be explained in more detail along its pseudo code. For better understanding, Figure 12 shows a very reduced scene and illustrates the essential components and coordinate systems for rendering. The relative positions of all components are described in the world space w. A further important space is the view space v (also common: 'eye space'), describing the scene from camera's point of view. All points in the scene, visible for the camera, can be limited by a pyramid frustum with the tip in the center of v and the floor and top area perpendicular to the viewing direction (see blue thick dashed lines in Figure 12). This geometry is known in computer graphics as view frustum. Its top area with distance n to camera position is called 'near clipping plane', while its floor area with distance f to camera is termed 'far clipping plane'. With the lighting pass, the light source becomes relevant for the first time during the rendering process. Figure 12 shows the headlamp to be simulated with its local light space l. As already mentioned, the illumination range of a headlamp is described by a light volume in the form of a half-sphere, which is illustrated by yellow lines in Figure 12. Furthermore, important for the following considerations are the local spaces of all objects within the scene. As an example, Figure 12 shows a cylindrical object with the local coordinate system o.

According to the schema of lighting pass in deferred rendering, the vertices of the light volume are first processed by the vertex program. In this sense, during the lighting pass, the light volume slips into the role of a scene object in the base pass. Implementing per-fragment-lighting, the vertex program can be realized by a few lines of code, since all light calculations are done in the subsequent fragment program. Essentially, three parameters are generated. First, a vertex ${}_lp'$ of the light volume mesh, passed in homogeneous coordinates [27], is transformed from the light space l into the clip space c of the camera (line 1). The transformation from l to the world space w is done by the matrix L, from w to the view space v by the matrix V and from v to the clip space c by the matrix P (see Figure 12). c has the same origin as v, but distorts the coordinates in perspective for easier mapping to the screen. The vertex coordinates in the clip space ${}_cp'$ form the basis of the rasterization and must be included in the return of the vertex shader. Furthermore, the clip space position in the lines 2 and 3 is transformed so that the x and y coordinates are normalized to $[0,1]$ for vertices in the view frustum in accordance with the perspective division in the fragment program. These values are then used as texture coordinates to correctly address the G-Buffer and form the second output of the vertex shader. Finally, the vector from the camera to the vertex in view space ${}_vp'$ is calculated by multiplying the vertex ${}_lp'$ in l by L and V (line 4). This value is needed next to ${}_cp'_{uv}$ to reconstruct the 3-dimensional position of the fragment in w within the fragment shader and forms the third return of the vertex shader. This way all vertices of the half sphere are processed by the shader.

```
Require: ${}_lp' \in \mathbb{R}^4$ vertex of light volume as homogeneus coordinates in $l$
1: ${}_cp' \leftarrow P \cdot V \cdot L \cdot {}_lp'$                                  // vertex in $c$
2: ${}_cp'_{uv}.x \leftarrow \frac{1}{2} \cdot {}_cp'.x + \frac{1}{2} \cdot {}_cp'.w$   // transform to screen space
3: ${}_cp'_{uv}.y \leftarrow \frac{1}{2} \cdot {}_cp'.y + \frac{1}{2} \cdot {}_cp'.w$   // transform to screen space
4: ${}_vp' \leftarrow V \cdot L \cdot {}_lp'$                                          // vertex in $v$
5: return ${}_cp', {}_cp'_{uv}, {}_vp'$
```

Afterwards the rasterization is effected based on the clip space coordinates ${}_cp'$. The remaining vertex program returns are bilinear interpolated to the fragments according to their distances to the vertices. The fragment shader processes all fragments in the light buffer covered by the light volume with the respective interpolation results ${}_cp'_{uv}, {}_vp'$ as parameters.

```
Require: ${}_cp'_{uv} \in \mathbb{R}^4$ transformed coords in $c$ and ${}_vp' \in \mathbb{R}^3$ coords in $v$ of vertex $p'$ of light volume
1: ${}_vp'' \leftarrow \frac{f}{{}_vp'.z} \cdot {}_vp'$                    // scale to far clipping distance (f)
2: $b_{uv} \leftarrow \frac{1}{{}_cp'_{uv}.w} \cdot {}_cp'_{uv}.xy$         // buffer coords of $p'$ (same for $p$)
3: $z_{uv} \leftarrow G_{depth}(b_{uv})$                                     // norm. depth at screen position $b_{uv}$
4: ${}_vp \leftarrow z_{uv} \cdot {}_vp''$                                   // position of $p$ in $v$
5: ${}_wp \leftarrow V^{-1} \cdot {}_vp$                                     // position of $p$ in $w$
6: ${}_wv_{c,o} \leftarrow {}_wp_c - {}_wp_o$                                // vector p→camera in $w$
7: ${}_wn_{c,o} \leftarrow \frac{{}_wv_{c,o}}{|{}_wv_{c,o}|}$                // direction p→camera in $w$

8: ${}_wl \leftarrow L[1:4,4]$                                               // position of light in $w$
9: ${}_wv_{l,o} \leftarrow {}_wl - {}_wp$                                    // vector p→light in $w$
10: ${}_wn_{l,o} \leftarrow \frac{{}_wv_{l,o}}{|{}_wv_{l,o}|}$               // direction p→light in $w$

11: ${}_lp \leftarrow L^{-1} \cdot {}_wp$                                    // vector light→p in $l$
12: $a_{deg}.x \leftarrow$atan2$({}_lp.x, {}_lp.z)$                          // horiz. and vert. angle be-
13: $a_{deg}.y \leftarrow$atan2$({}_lp.y, {}_lp.z)$                          // tween light axis and ${}_lp$
14: $t_{uv} \leftarrow \frac{a_{deg}+90^\circ}{180^\circ}$                   // Light-Cookie coordinates
15: $l_{uv} \leftarrow T_{cookie}(t_{uv})$                                   // light power in specific direction

16: $att \leftarrow \frac{1}{l.range^2} \cdot {}_wv_{l,o} \cdot {}_wv_{l,o}$  // light attenuation
17: $att \leftarrow att \cdot l_{uv}$                                         // consider light power
18: $l.color \leftarrow att \cdot l.color$                                    // attenuated light color

19: return lightingModel$(c_o, {}_wn_o, {}_wn_{c,o}, {}_wn_{l,o}, l.color)$
```

The shader code can be divided into five logical blocks. The first block (line 1 to 7) reconstructs the three dimensional surface point p of the scene object o visible on the current fragment. This reconstruction is enabled by the additional information provided by the vertex shader. Outgoing from p', which triggers the current fragment program call as it is part of the light volume surface, it can be seen, p' is mapped to the same screen position as p (see dashed line through p, p' and p'' in Figure 12). p in turn represents a surface point of a physical scene object for which lighting has to be performed in the following. According to the figure, ${}_vp'$ describes the cameras view direction to p in v. The G-Buffer created in the base path of deferred rendering can be used to determine the exact position of p on the corresponding line. It encodes the z coordinate (or depth value) in v for each fragment in addition to other data. To read the correct value from the depth buffer, the buffer coordinates must be determined first. Therefore, the vector ${}_cp'_{uv}$ is defined in the vertex shader, whose x- and y-coordinates lie in the interval $[0, {}_cp'_v.w = {}_cp'_{uv}.w]$ for points within view frustum. After the perspective division by the homogeneous component in line 2 the coordinates b_{uv} are in the value range $[0,1]$ (line 3) used for texture/buffer access. The depth buffer encodes depth z_{uv} normalized on distance f to the far clipping plane in the interval $[0,1]$. The coords of p in v result from scaling of ${}_vp'$ to the far clipping plane receiving ${}_vp''$ (line 1) and the subsequent multiplication with the normalized depth z_{uv} (line 4). Multiplication by the inverse of V transfers the object point p from v to w. For the evaluation of the lighting model, the normalized direction vector from the object point to the camera (eye vector) in world space ${}_wn_{o,c}$ (see Figure 12) is needed (line 6 and 7).

In addition to the eye vector, the incidence of light on the object plays a central role in the lighting model, too. The light vector is defined in lines 8 to 10. First, the position of the light source in world space ${}_wl$ is extracted from the matrix L (line 8). Since the Headlight-Shader only renders the mesh of the light volume into the light buffer, the transformation matrix L from current object space l to w is constant across all calls of the vertex program and contains the translation, rotation and scaling of the light volume mesh into w. In general, the world space coordinates ${}_ws$ of a point s can be achieved by multiplying its light space coordinates ${}_ls$ by the homogeneous transformation matrix L from l to w according to

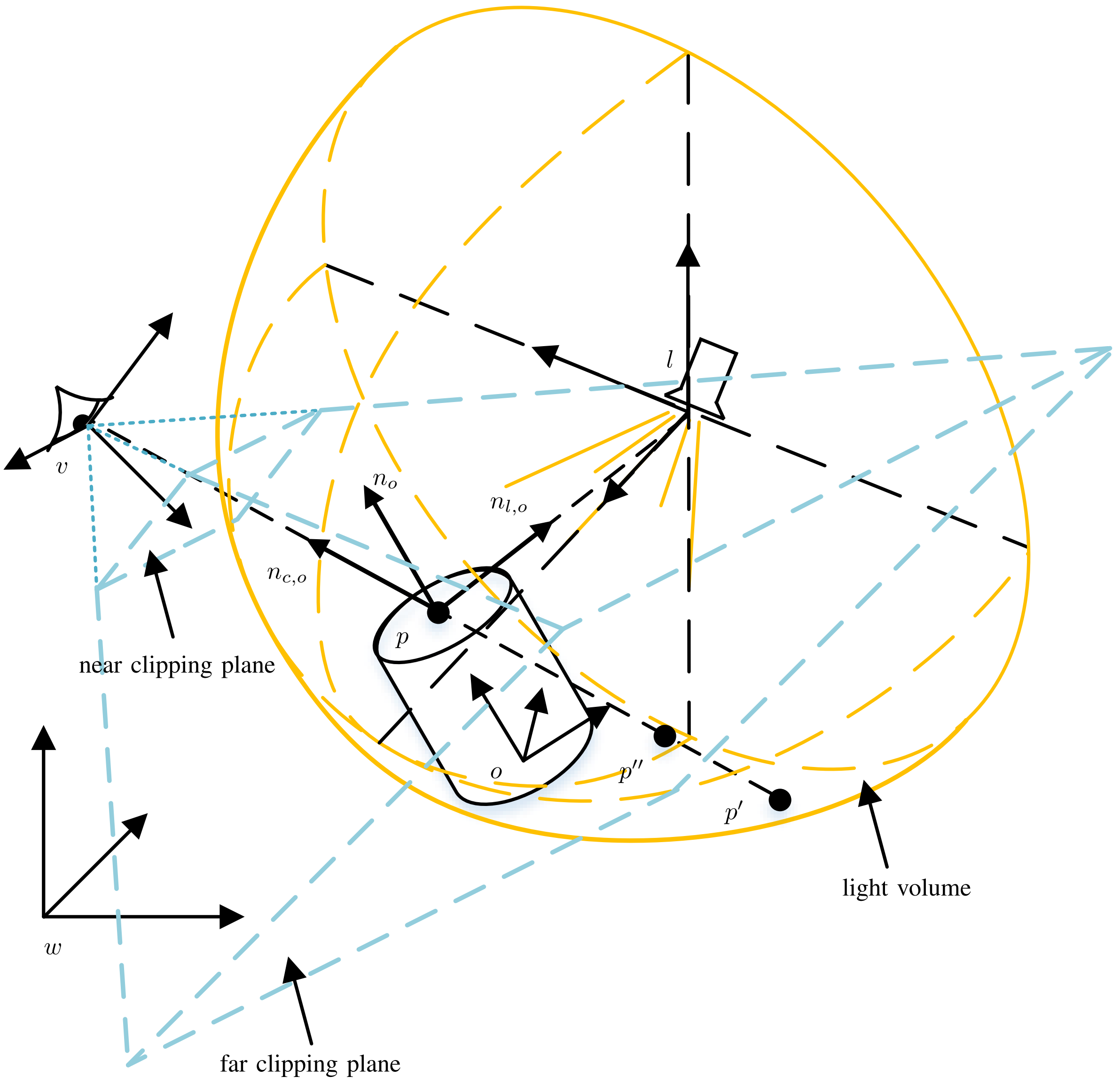

Figure 12. Simple scene to be rendered with coordinate systems for world space (w), view space (v), light space (l) and an exemplarily object with its local space (o).

$$_ws = L \cdot {}_ls \text{ with } L = \begin{bmatrix} l_{11} & \dots & l_{14} \\ \vdots & & \vdots \\ l_{41} & \dots & l_{44} \end{bmatrix}, \qquad (8)$$

whereby l_{14}, l_{24} and l_{34} contain the translation from l to w along the x-, y- and z-directions [23]. Since the light source is in the coordinate origin of l, the translation in L corresponds to the position of the light in w and can be read from the fourth column of L. Now the normalized direction vector from the object point to the light source in world space $_wn_{l,o}$ (see Figure 12) can be determined by the lines 9 and 10 similar to the previous section.

In the lines 11 to 15, the luminous intensity distribution T_{comb} determined by the Cookie Combiner-Shader is taken into account. In order to determine the luminous intensity to be applied to the current fragment from the light distribution, the horizontal (θ) and vertical (φ) angles of the incident light beam with respect to the light center axis must be calculated. For this purpose, the object point belonging to the fragment is transferred to the light space l by multiplying with L^{-1}. The position of this point in l, in whose coordinate origin the light source is located and oriented along the z axis, simultaneously corresponds to the light beam $_lp$ from the source to the object. Its angle to the light center axis or z axis in l can then be determined with the $atan2$ function (line 12 and 13). The cookie T_{comb} must be addressed with normalized texture coordinates. Therefore, the angles moving in the range $[-90°, +90°]$ due to the used light volume are transformed by line 14 to texture coordinates t_{uv} according to Equation (7). Thereby, the applicable value can finally be read from the light distribution texture T_{comb} (line 15).

After the positions of the relevant elements and the luminous intensity of the light are already known, the distance to the light source must be taken into account in the lines 16 to 18. The intensity of light decreases square with the distance to the shined object [16]. This square distance can be formulated most efficiently as a scalar product of the vector from object to light ${}_wv_{l,o}$ with itself. The distance between object and light is referred to a freely selectable positive light parameter $l.range$, which allows the user to adjust the range respectively the intensity of the light (line 16). It applies $att = 1$ if the distance between light source and object corresponds to the parameter $l.range$, and $att \to \infty$ for increasing distances. To take into account the directional luminous intensity l_{uv}, it is multiplied by att to the final light attenuation. The calculated attenuation can finally be mapped by the light color in line 18 through multiplying the color of the light defined by the user by att. Assuming a white light, this multiplication corresponds to a shift on the grayscale.

Finally, the lighting model can be evaluated. At this point, no separate solution has been implemented yet, but an Unity-internal local lighting model has been used. This receives the previously determined normals ${}_wn_{o,c}, {}_wn_{l,o}$, the surface normal ${}_wn_o$ (see Figure 12) and material data c_o of the object from the G-Buffer at buffer coordinates b_{uv} and the light color $l.color$, which already considers attenuation. Based on these data, the lighting model delivers the resulting color for the currently considered fragment and thus generates the finished image of the scene on the output medium.

IV. Results

After the implementation details, the images resulting from the rendering will be presented and discussed on their appearance. By selecting a suitable scene, it is in particular possible to validate the rendering results with respect to underlying measurement data (see Section IV-A). In the second part a comparison with the software LightDriver will be done. The LightDriver is an established tool for headlamp simulation from HELLA that has been in use for many years. It is successfully applied to support the development process and can therefore be regarded as validated. For this reason, it serves to validate the solution presented here using more realistic scenes as in Section IV-A. In addition, Section IV-C contains some remarks on the real-time capability of the simulation. Due to the dependence on countless factors (hardware and software configuration used, complexity of the scene and the depth of detail of the objects in it, number of simulated vehicles and other light sources, etc.), no general statement can be made here about the required calculation time per frame. The basic usability of the implementation for the night driving simulation should nevertheless be proven.

A. Validation

As described in Section II, the radiation characteristics of a light source can be defined by plotting the luminous intensity over the angles φ and θ in the spherical coordinate system. In this way, the luminous intensity can be represented in a plot, in which it is coded by false colors. In order to validate the rendered light based on the actual light distribution of the simulated headlamp, the artificial scene shown in Figure 13 is used.

It contains only three elements - a gray sphere, a left-side HD84 headlamp in the center of it and a camera defining the rendering perspective. Position and orientation of camera and light source are equal. They can be localized at the origin of coordinate system represented by the blue, red and green arrows, whereby the blue arrow defines the viewing direction of the camera as well as the light axis ($\varphi = \theta = 0°$). The field of view of the camera is chosen to 60°, which corresponds to a view frustum tip angle of 120° in the horizontal plane. The already mentioned half-spherical light volume approximated by a mesh of 65 vertices is visualized by the yellow wireframe.

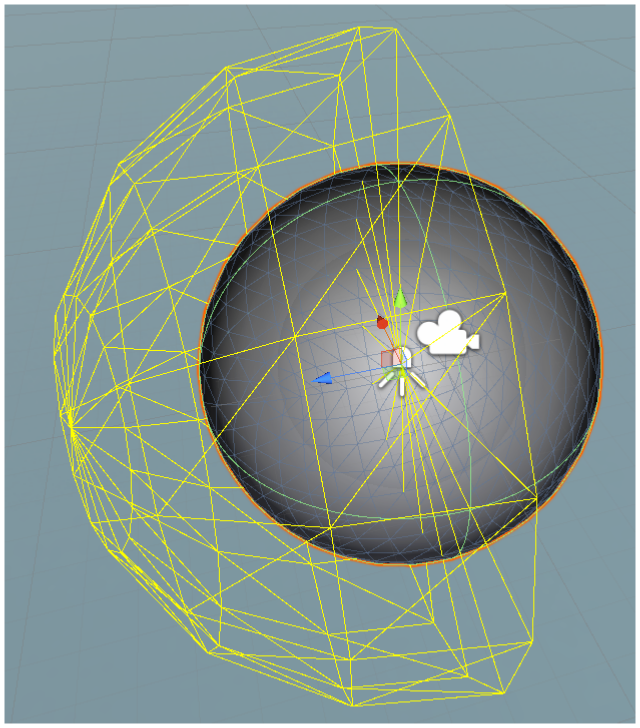
Figure 13. Artificial validation scene to compare the rendered headlight with measurement data.

According to this scene definition, the observer is in the center of the gray sphere and sees the light as it is emitted by the light source onto the inner wall of the sphere according to the implementation presented here. Running the headlamp with electrical current values to generate a low beam distribution, the rendering process produces the image shown in the upper part of Figure 14.

In rendered light, classic properties of a low beam distribution can already be perceived. In particular, the level of the light-dark boundary line along the vertical central axis should be mentioned here. The considerably greater horizontal spread of light compared with vertical spread can also be observed. For the complete validation of the implemented solution, however, the alignment with the measured data is necessary. For this purpose, it is shown in the lower part of Figure 14. In a direct comparison, the similarity of shape between rendering and measurement can be well demonstrated. At the same time, however, it is observed that the limited contrast of the light resolved linearly in gray scales is neither sufficient to perceive intensity differences in the bright area of the light distribution nor to perceive curtains in the edge area.

For this reason, the illuminance of the light on the elements of the scene was rendered logarithmically with another shader and coded in false colors. Luminous intensity I and

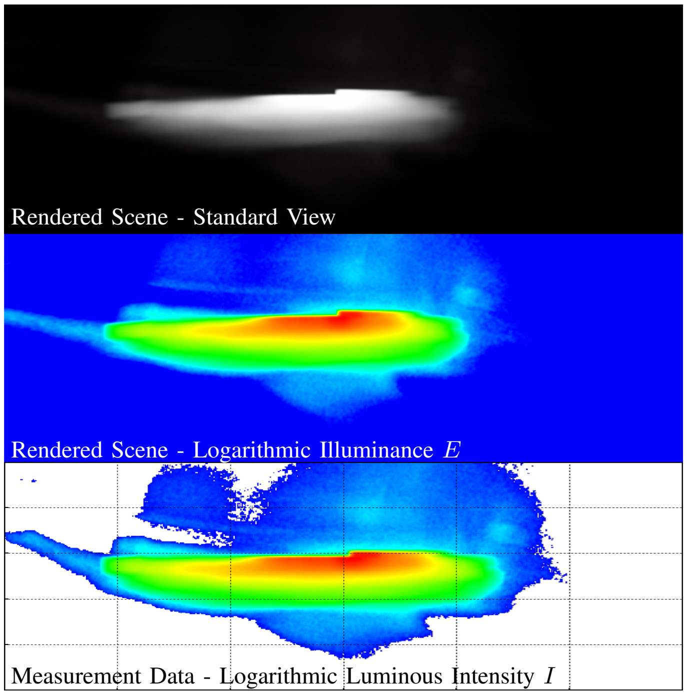

Figure 14. Comparison of rendered light (top) of left HD84 headlamp in low beam mode (false color coded illuminance on the spherical wall for better comparabilimty shown in the center area) and measured data of the same configuration (bottom).

illuminance E are related according to

$$E = \frac{I}{r^2} \cdot \cos(\epsilon), \tag{9}$$

where r is the distance between the light source and the illuminated object and ϵ is the angle between the light beam and the surface normal of the illuminated object [16]. In the specially selected scene, the distance between the light source and the illuminated object is constant at each pixel with the sphere radius. At the same time, each illuminated area is perpendicular to the incident light beam. This means that illuminance and luminous intensity have the same characteristics, as there is a linear relationship between them, making it particularly easy to compare the image produced with the measured data.

As it turns out, the correspondence of the simulated light distribution (center area of Figure 14) with the measured data (lower area of Figure 14) is even more convincing in the logarithmic false color representation. The gradations of intensity in the center of the light cone as well as the light curtains in the edge area can be found in both figures in the same way. These checks were also carried out for other light distributions and led to the same observations, so that the implementation can be considered as validated.

B. LightDriver

Even if the correctness of the implementation in terms of the mathematically correct reproduction of the luminous intensity distribution has already been carried out in Section IV-A, the appearance of the light distribution on the road should also be evaluated. In such an investigation, significantly more factors have an influence. The limitations of the output device (especially with regard to luminance and contrast), the modeling accuracy of the scene (textures, normal maps, reflection properties, ...) and the complexity of the lighting model used (reflections, ray tracing, ...) are just a few examples. Since the approximate reconstruction of a real environment is not an acceptable effort, the HELLA LightDriver (64 Bit Version built on July, 2017) should be used as a reference instead. This is HELLA's own development for night driving simulation, which has been used successfully for several years in the development process of new headlamp systems and lighting functions. In contrast to the implementation presented here, the LightDriver is not able to change the light distribution of a headlamp dynamically, as is necessary for the simulation of an HD system. Nevertheless, this fact does not limit its suitability for validating this implementation. The desired light distributions can be calculated in advance and then loaded into the LightDriver as a static total light distribution.

Figure 15 compares the low beam distribution of the HD84-Matrix-LED headlight (left and right headlamp) as calculated by Cookie Combiner- and Headlight-Shader (top) with the low beam distribution of the LightDriver as reference (bottom) in a simple street scene. In the right area of the figure, the scene is complemented by a white measuring wall with red control lines at a distance of 10m from the vehicle. This is a classic analysis tool for the evaluation of light distributions, as their shapes are more recognizable on this. The two vertical control lines are aligned with the mounting positions of the headlamps. While in the presented implementation only the electrical current values of the individual light sources belonging to this light distribution are specified, the LightDriver requires a complete light distribution as input, which is therefore calculated in advance. The scene for this comparison could not be taken directly from the LightDriver and was therefore recreated. As a consequence the textures and colors of scene objects are not exactly the same, but should suffice for a plausibility check.

If one compares the low beam distributions in the left area of Figure 15, they match well overall. A closer look reveals slight differences. On the one hand, the basic color or brightness on the street appears not quite uniform in both representations. In addition, it seems that the light curtains directly in front of the vehicle are of different brightness. The reasons for the deviations mentioned can be many and varied due to the large number of influencing factors already mentioned. However, despite its successful use in the development process, the LightDriver should not be regarded as an exclusive measure of optimality. The differences in the images result to a large extent from the use of HDR colors in the implementation presented, while the LightDriver uses classic SDR colors. In particular, the more clearly visible light curtains in front of the vehicle in this implementation are caused by this. Further differences may result from slightly different definitions of the lighting model and the scene objects.

In addition to these minimal differences, however, both simulations match, so that the rendering method presented here can also be checked for plausibility in realistic scenes. With the help of the road markings, the qualitative form equality of the illuminated road areas of both implementations can be recognized. In addition, the different light ranges for the left and right lanes are clearly visible in both representations. For this reason, the white control areas at the edges of the road on the right-hand side are illuminated at a greater distance than on the left-hand side. This characteristic is typical for low beam distributions and ensures the best possible illumination of one's own lane without glaring oncoming traffic.

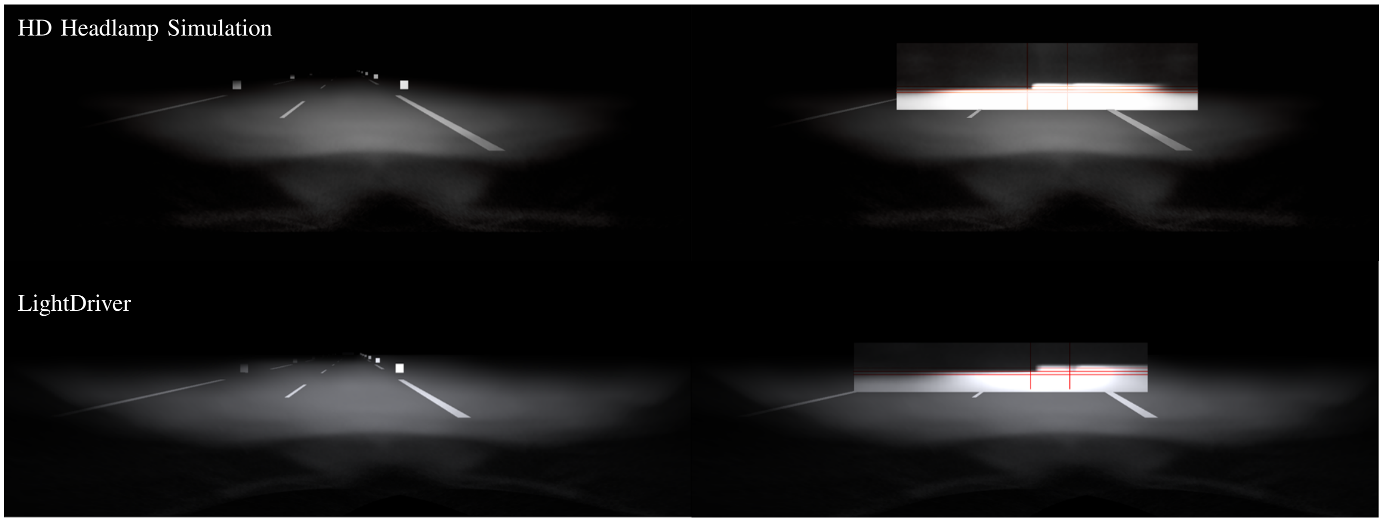

Figure 15. Comparison of the implementation presented (top) and HELLA LightDriver (bottom) simulating a low beam distribution in a simple street scene (left) and with a measuring wall in 10m distance (right).

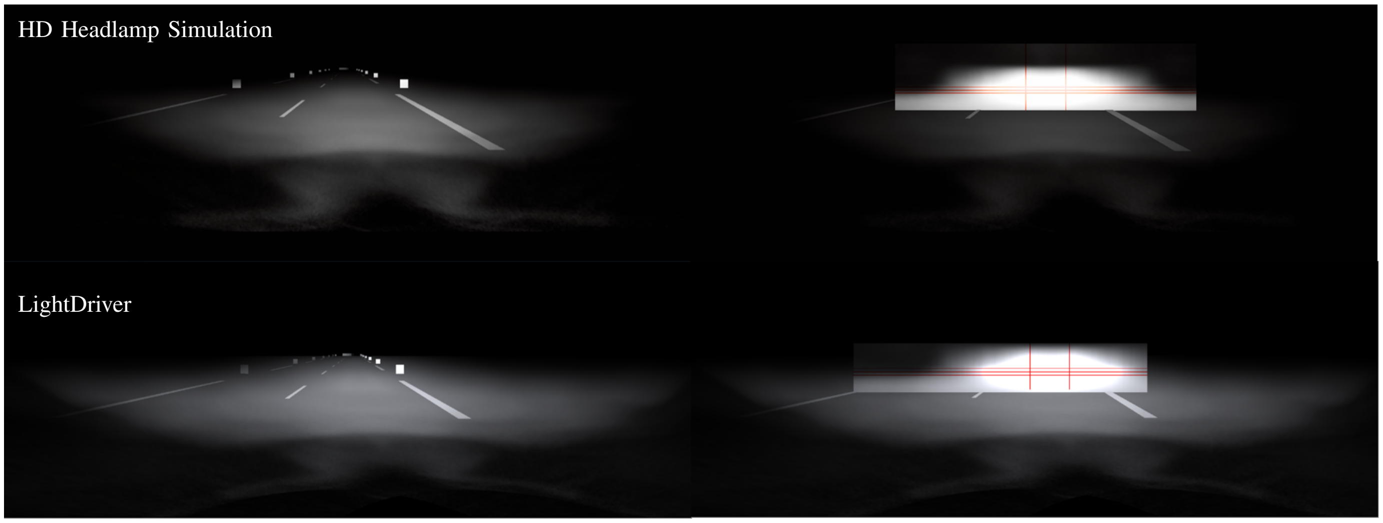

Figure 16. Comparison of the implementation presented (top) and HELLA LightDriver (bottom) simulating a high beam distribution in a simple street scene (left) and with a measuring wall in 10m distance (right).

Even if the light distribution on the road is the central evaluation criterion for the driver, the observation of light distributions on a vertical measuring wall has proven to be useful especially for comparison purposes. The contours of the light distribution, which are called the light-dark boundary in this context, become clear through the close projection of distant areas. The varying illumination distances of the lanes, which could already be observed through the control surfaces, become even clearer on the measuring wall. They can be found in the typical low beam distribution step in the upper middle area of the light-dark boundary. As can be seen from the simulation images, these steps form in the middle of the left and right headlamps, which can be identified by the vertical red control lines.

After comparing the low beam distributions in both simulations, Figure 16 shows a similar comparison for the high beam as a second light distribution of central importance. For the LightDriver the light intensity distribution was precalculated again and loaded as static total light distribution. As in the case of low beam, a good overall agreement can be observed in addition to differences in detail between the images.

As was to be expected, the illumination of the distant areas has increased in comparison with the low beam in Figure 15. The difference is particularly noticeable on the opposite lane and the control surfaces positioned along it. This effect is achieved by a fundamental change in the shape of the light distribution, as can be observed on the measuring walls on the right-hand side of Figure 16. The step in the light-dark boundary of the low beam disappears when high beam is used. Instead, a symmetrical light distribution is generated with a so-called high beam cone, which illuminates the distant areas in front of the vehicle.

C. Computational Effort

With regard to the application of the implementation presented here in an interactive night driving simulation, compliance with the real-time requirements must be considered. In order to give the viewer the impression of a dynamic scene, at least 30 frames per second (fps) should be calculated. The optimal case is 60 fps, which corresponds to the refresh rate of most output devices [28]. A further optimization of the computing time does not lead to any benefit after this limit.

At the same time, however, it should be mentioned that a computer with a Windows operating system is not a real-time system. Due to the uninfluenceable scheduling of all running processes, measurements of calculation times are always rough approximations and can be subject to strong fluctuations.

In addition, the calculation times naturally depend heavily on the hardware and software configuration. The results discussed below were recorded on a mobile PC, whose specification can be found in Table I.

TABLE I. HARD- AND SOFTWARE-CONFIGURATION OF THE MEASURMENT PC

Operating System	Windows 10 Pro 64-bit (10.0, Build 16299)
Used Graphics Engine	Unity3D, Version 2017.3.1f1
Model	Dell Precision 7710
Processor	Intel(R) Core(TM) i7-6820HQ CPU @2.7GHz
Memory	16384 MB RAM
DirectX Version	DirectX 12
Graphic	NVIDIA Quadro M3000M
Video Memory	4062 MB VRAM (+ 8133 MB Shared)

In order to comply with real-time requirements, only those processes are relevant that are constantly being executed at runtime. In this application, these are Cookie Combining and lighting. Both can be examined in isolation from each other. The Profiler available in Unity is used to measure the calculation times. With this profiler, the calculation times of the CPU and GPU can be broken down frame by frame with regard to the operations performed.

First, the performance of the cookie combining is examined. It depends only on the number of single light sources and the resolution of the floating point textures describing their luminous intensity distribution (here: 95 32bit floating point textures with 900x900 texels). To exclude possible influences of the program start and caching, the total light distribution is calculated many times. All relative current values are randomly selected between 0% and 100% for each calculation.

The analysis in the profiler shows that combining the 95 floating point textures on the CPU requires an average of 0.45ms (min: 0.27ms, max: 0.64ms). With cookie combining, however, the CPU acts primarily as the client of the GPU. It instructs the graphics card to execute the Cookie Combiner shader by creating draw calls and defines the relevant context information, such as the render target or the current individual light distribution. So it is not surprising that the GPU has a significantly higher average calculation time of 4.61ms (min: 4.44ms, max: 4.73ms).

In addition to cookie combining, the calculation time of the headlight shader must also be examined. Due to the deferred rendering pipeline used here, this is called only once per light source and frame. The effort of the lighting therefore does not depend on the scene complexity, but only on the resolution of the output and thus of the light buffer, which is selected here as 1430x780 pixels. On the CPU, the lighting of a spotlight simulation requires an average of 0.02ms (min: 0.01ms, max: 0.04ms). The calculation time on the GPU measures approximately 0.097ms (min: 0.095ms, max: 0.114ms). In this respect, the effort of the actual lighting can be neglected compared to the calculation time of the Cookie Combining.

In comparison with the much simpler implementation of cookie combining, this result may seem astonishing. However, the GPU can perform the lighting calculations for all pixels of the light buffer in parallel. The Cookie Combiner shader, on the other hand, must be called repeatedly in one frame for all light sources with changing context information.

As a result, both shaders are performant enough to be used in the night driving simulation of a single vehicle with two headlamps. If the simulation of several vehicles with HD headlamps is intended, Cookie Combining will reach the limits of available computing power even for small vehicle quantities. If dynamic HD headlights are desired on all vehicles, more powerful graphics cards can be used to shift these limitations within certain bounds. On the other hand, the question arises whether dynamic lighting functions are required. If this is not the case, the precalculation of static light distributions (e.g. dipped beam) is a sensible alternative for third-party vehicles. In this way, they retain the basic light characteristics of an HD headlamp without making significant use of resources. The ego vehicle can still be simulated with dynamic lighting functions using cookie combining.

V. CONCLUSION

This contribution presents an approach for real-time simulation of dynamic HD headlamp systems and thus lays the foundation for the simulation-based development of high-resolution dynamic light functions. The implementation reproduces the real light distribution accurately and is also executable on average hardware for today's standards.

This contribution is motivated by the completely missing possibility of real-time simulation of high-resolution headlamp systems, which is indispensable for a systematic and verifiable procedure of development. In addition, there is the need for darkness and suitable weather conditions during real test drives, which cannot be completely eliminated by using a simulation, but can be significantly reduced.

Before the simulation of HD headlamp systems is presented, their technical structure and functionality are described in Section II. During the course, the light intensity distributions established in the context of headlamp measurement will also be introduced. They describe the luminous intensity emitted by a light source depending on the spatial direction. The HD84-Matrix-LED headlamp type under consideration here consists of a matrix with 84 LEDs and 11 further light sources (apron area lights, bend lights, additional high beam lights). The luminous intensity distribution is known for each of these light sources. They can be operated with current control, which results in corresponding scaling of the luminous intensity values. By locally separating their illumination areas, it is possible to build the overall light distribution summing up the individual distributions, whereby their shape variability is limited only by the resolution of the headlamp. An overview of this procedure is illustrated by Figure 4.

The following Section III describes the implementation, the overall scheme of which is shown in Figure 5. First of all, the digital representation of a luminous intensity distribution by a texture is defined in Section III-A. In particular, 32bit floating point textures are used to capture the high light dynamics in a night driving scene in detail. Since the underlying measurement data are available with an angular resolution of $0.2°$ over a range of $180°$ in horizontal and $50°$ in vertical direction, textures with 900x900 texels are selected. The measuring points can thus be mapped unchanged onto the texels (see Figure 6 and Equation (7)). Light intensity values between the measuring points are bilinearly interpolated according to

Figure 7 and Equations (5) and (6). In preprocessing, the measured luminous intensity distributions for all light sources in the headlamp can be converted into floating point textures and thus used for subsequent rendering.

The rendering of the headlight is divided into two steps. In the first step, the current total light distribution is determined as described in Section III-B. It is the weighted sum of the 95 individual light distributions of all adjustable light sources of the HD84-Matrix-LED headlamp. In the technical implementation, the light distributions are represented by textures and superimposed by blending the results of the Cookie Combiner-Shader applied to them as shown in Figure 8. Mathematically formulated, the total light distribution is formed as a linear combination of the matrices or textures of the individual light distributions (see Figure 9). Each individual light distribution is weighted with the present relative current value. In order to provide the optimum overall light distribution in every driving situation, the current values can be adjusted with a frequency of 50 Hz. The output of the Cookie Combiner-Shader is a floating point texture of the overall light distribution, which is structurally no different from that of the individual distributions.

The total light distribution initiates the second rendering step as input for the Headlight-Shader discussed in Section III-C. This is integrated into the deferred rendering pipeline by using Unity's Graphics Command Buffer. As Figure 11 illustrates, it is downstream of the standard operations in the lighting pass. In this way, compatibility with all standard lights in the scene is maintained. In addition, HDR rendering is used to reproduce the high contrasts of a night drive scene with the existing limitations of the output device as detailed as possible. Along the pseudo code the procedure in the headlight shader is discussed, whose final task is the determination of a color value for the respective pixel of the light buffer or the output under consideration of the current light source. The essential spaces, geometric correlations and vector operations can be understood by Figure 12. After finding all relevant information for the determination of the pixel color, this is transferred to a Unity-internal lighting model. This model finally returns the resulting color, which corresponds to the output of the Headlight-Shader.

Section V concludes the contribution by presenting the results. In the first step, the simulated luminous intensity distribution is validated on the basis of the measurement data. For this purpose, an artificial scene is created which serves this purpose exclusively. It contains only a sphere and in its center a HD84-Matrix-LED headlamp to be simulated. The background of this scene definition is the linear relationship between the luminous intensity of the headlamp and the illuminance on the spherical surface, as can be seen from Equation (9). In this way, in Figure 14, the simulated illuminance on the spherical surface and the measured luminous intensity distribution can be compared. The agreement of these data is convincing.

After the proof for the mathematically correct reproduction of the real light intensity distribution has been provided, the light impression in a more realistic scene is evaluated in Section IV-B. In such an evaluation, far more influences come into play than can be controlled within the framework of the implementation presented here. For this reason, the night driving simulation software LightDriver developed by HELLA serves as orientation for evaluating the rendering results. As a tool for headlamp development that has been established for years, it is suitable as reference, even if it is not an incontestable optimum. As Figures 15 (low beam) and 16 (high beam) show, the implemented simulation is very similar to the LightDriver. The differences can mainly be traced back to the unequal scenes and light models, as well as the higher luminous intensity resolution and HDR rendering used here. Consequently, they do not represent a quality defect of the presented implementation.

The performance analysis shows that the major share of the computing effort is attributable to Cookie Combining. Even if the simulation of a vehicle on the mobile PC with which the computing time measurements were carried out does not pose a problem, it should be noted that Cookie Combining quickly reaches the limits of computing power when simulating several vehicles. These can be moved upwards by using better hardware (in particular graphics card). As an alternative, the application of static light distributions for external traffic is proposed. In this case, the Cookie Combining must only be carried out for the ego vehicle.

VI. Future Work

In view of the good rendering results, future work will focus less on the further development of the presented implementation. Nevertheless, the lighting model, for which the Unity standrad BRDF model has been used so far, could be replaced by an own implementation in the future, depending on the resulting improvements. In addition, it should be checked whether Cookie Combining, which represents the bottleneck of the implementation in terms of computing time, provides potential for further performance improvement.

The presented work should serve much more as a basis for the night driving simulation, for the implementation of which various follow-up work is necessary. First of all, the previously manual setting of the current values for the individual light sources must be replaced by the integration of the headlamp control unit. This integration is divided into two steps. On the one hand, the outputs of the control unit must be received by the visualization system and transferred to an intensity list (see Figure 5) that is compatible with the Cookie Combiner. This step could already be performed at the current time with the necessary requirement of 50 Hz. On the other hand, the connection of the control unit only makes sense if it knows about the current traffic situation in order to select the light distribution based on it. For this purpose, the real sensors in the vehicle must be simulated by virtual sensors. The central role is played by the surrounding camera. Various approaches to implementation are currently being investigated, including machine learning methods on the images rendered from the point of view of the surrounding camera.

A third approach for further work is the use of analysis tools to assess the headlight in the scene. For this purpose, false color and iso-line representations have already been implemented. These can be applied to pixel brightness or illuminance. Their scaling to physical quantities and their adaptation to legal or customary standards will still have to take place in the future.

Acknowledgment

The authors would like to thank HELLA for providing the light distribution data of the HD84-Matrix-LED headlamp and the LightDriver simulation software. This paper is part of the EFRE.NRW (European Fonds for Regional

Development, North Rhine-Westphalia)-funded project 'Smart Headlamp Technology (SHT)'.

REFERENCES

[1] N. Rüddenklau, P. Biemelt, S. Henning, S. Gausemeier, and A. Trächtler, “Shader-based realtime simulation of high-definition automotive headlamps,” IARIA SIMUL 2018, The Tenth International Conference on Advances in System Simulation, 2018.

[2] B. Fleury, L. Evrard, J.-P. Ravier, and B. Reiss, “Expanded Functionality of Glare Free High Beam Systems,” ATZ Autotechnology, 2012.

[3] M. Enders, “Pixel light,” Progress in Automobile Lighting (PAL), 2001.

[4] F.-J. Kalze and D. Brunne, “LED im Fahrzeug: Die Rolle der Matrixscheinwerfer und was sie leisten (LED in the Vehicle: The Role of Matrix Headlamps and what they perform),” Elektronik Praxis, 2018.

[5] C. Schmidt, B. Willeke, and B. Fischer, “Laser versus Hochleistungs-LED - Vergleich der Einsatzmöglichkeiten bei hochauflösenden Matrix-Scheinwerfer-Systemen (Laser versus High Power LED - Comparison of Application Possibilities for High-Definition Matrix Headlamp Systems),” VDI-Tagung Optische Technologien in der Fahrzeugtechnik, Karlsruhe, 2016.

[6] J. Roslak and C. Wilks, “Hochauflösende LCD-Scheinwerfer - Herausforderungen für Elektronikarchitekturen (High-Definition LCD Headlights - Challenges for Electronic Architectures),” Automobiltechnische Zeitschrift elektronik (ATZelektronik), 2017.

[7] P. Lecocq, J.-M. Kelada, and A. Kemeny, “Interactive Headlight Simulation,” Driving Simulation Conference, 1999.

[8] J. Berssenbrügge, J. Gausemeier, M. Grafe, C. Matysczok, and K. Pöhland, “Real-Time Representation of Complex Lighting Data in a Nightdrive Simulation,” 7. International Immersive Projection Technologies Workshop, 9. Eurographics Workshop on Virtual Environments, 2003.

[9] J. Löwenau and M. Strobl, “Advanced Lighting Simulation (ALS) for the Evaluation of the BMW System Adaptive Light Control (ALC),” International Body Engineering Conference & Exhibition and Automative & Transportation Technology Conference, 2002.

[10] A. Kemeny et al., “Application of real-time lighting simulation for intellignet front-lighting studies,” Driving Simulation Conference, 2000.

[11] J. Berssenbrügge, J. Bauch, and J. Gausemeier, “A Virtual Reality-based Night Drive Simulator for the Evaluation of a Predictive Advanced Front Lighting System,” Design Engineering Technical Conferences & Computers and Information in Engineering Conference, 2006.

[12] J. Berssenbrügge, S. Kreft, and J. Gausemeier, “Virtual Prototyping of an Advanced Leveling Light System Using a Virtual Reality-Based Night Drive Simulator,” Journal of Computing and Information Science in Engineering, 2010.

[13] A. Knoll et al., “Evaluation of an Active Safety Light using Virtual Test Drive within Vehicle In The Loop,” IEEE International Conference on Industrial Technology, 2010.

[14] “AutomobilIndustrie: Adaptives LCD-Licht mit 30.000 Pixeln (Automotive Industry: Adaptive LCD-Light with 30,000 Pixels),” 2017, URL: https://www.automobil-industrie.vogel.de/adaptives-lcd-licht-mit-30000-pixeln-a-629502/ [retrieved: 5, 2019].

[15] K. Reif, Ed., Automobilelektronik (Automotive Electronics), p. 301 ff. Vieweg+Teubner, GWV Fachverlage GmbH, Wiesbaden, 2009, ISBN: 978-3-8348-0446-4.

[16] F. Pedrotti, L. Pedrotti, W. Bausch, and H. Schmidt, Eds., Optik für Ingenieure - Grundlagen, 4. Auflage (Optics for Engineers - Basics, 4th edition), p. 13 ff. Springer, Berlin, 2007, ISBN: 978-3-540-22813-6.

[17] H.-H. P. Wu, Y.-P. Lee, and S.-H. Chang, “Fast measurement of automotive headlamps based on high dynamic range imaging,” OSA Applied Optics Vol. 51, 2012.

[18] A. S. Glassner, Ed., An Introduction to Ray Tracing. ACADEMIC PRESS INC., San Diego, CA 92101, 1989, ISBN: 0-12-286160-4.

[19] R. Neumann and H. Hogrefe, “Computer simulation of light distributions for headlamp systems,” SAE Technical Paper, 1991.

[20] “Unity Homepage,” URL: https://unity3d.com/ [retrieved: 5, 2019].

[21] R. Fernando and M. J. Kilgard, Eds., The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics. Addison Wesley Pub Co Inc., Feb. 2003, ISBN: 978-0321194961.

[22] T. Saito and T. Takahashi, “Comprehensible Rendering of 3-D Shapes,” SIGGRAPH '90, Dallas, 1990.

[23] J. F. Hughes et al., Eds., Computer Graphics - Principles and Practice, 3th Edition. Addison-Wesley, Jul. 2013, ISBN: 978-0321399526.

[24] E. Reinhard, W. Heidrich, P. Debevec, S. Pattanaik, G. Ward, and K. Myszkowski, Eds., High Dynamic Range Imaging - 2nd Edition. Morgan Kaufmann, 2010, ISBN: 978-0-12-374914-7.

[25] “Graphics Command Buffer,” 2018, URL: https://docs.unity3d.com/Manual/GraphicsCommandBuffers.html [retrieved: 5, 2019].

[26] “Unity Blog: Extending Unity 5 rendering pipeline: Command Buffers,” 2015, URL: https://blogs.unity3d.com/2015/02/06/extending-unity-5-rendering-pipeline-command-buffers/ [retrieved: 5, 2019].

[27] A. Beutelspacher and U. Rosenbaum, Eds., Projektive Geometrie, 2. Auflage (Projective Geometry - 2nd edition), p. 63 ff. Vieweg, Wiesbaden, 2004, ISBN: 978-3-528-17241-1.

[28] A. Banitalebi-Dehkordi, M. T. Pourazad, and N. Panos, “Effect of High Frame Rates on 3D Video Quality of Experience,” IEEE International Conference on Consumer Electronics (ICCE), 2014.