Shader-Based Realtime Simulation of High-Definition Automotive Headlamps

Nico Rüddenklau, Patrick Biemelt, Sven Henning, Sandra Gausemeier and Ansgar Trächtler
Chair of Control Engineering and Mechatronics, Heinz Nixdorf Institute, University of Paderborn
33102 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. This paper presents a first realtime simulation of high-definition systems in virtual environments. The simulation results are evaluated using validated software for simulating static light distributions at night. The performance of the implementation on the test hardware is also discussed. First results show a great potential for future development, which will be shown at the end of the contribution.**

Keywords–high-definition headlamp; realtime headlamp simulation; night driving simulation; dynamic light distribution.

I. INTRODUCTION

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. A further complication is that dynamic headlamps, away from the simulation, can only be proven by time-consuming and cost-intensive night driving tests under non-reproducible test conditions. Furthermore, the construction of a prototype is expensive, therefore, it should ideally be used in late development stages only. These difficulties motivate the simulation-based development of headlamps.

Realtime simulation of headlamps was first dealt with by Kemeny et al. [1]. This initial realization implements per-vertex-lighting. 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 [2]. A comparable concept is presented in [3]. It utilizes a proprietary development for the simulation of night driving with additional functionalities. Based on such simulations, various publications exist for testing dynamic lighting functions. In [4], Kemeny et al. present the simulation of the cornering light function within the established driving simulation software SCANeR. Like Berssenbrügge et al. in [5], they implement predictive cornering light, which calculates the ideal light distribution on the basis of navigation data and vehicle speed. Next to the cornering light function, Berssenbrügge et al. are also testing an advanced leveling light system in their simulation environment [6]. With active safety light Knoll et al. presents the simulative testing of a new light function for highlighting possible escape ways in risky driving situations [7].

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 the orientation angles of the light sources. High-definition (HD) headlamps, however, realize lighting functions in a totally different way and must therefore be modelled and simulated differently. Their outstanding features and their modelling are described in Section II. Section III the shader-based implementation of the light simulation is discussed. This is divided into the determination of the total light distribution (Section III-A) and the illumination of the scene based on that (Section III-B). In Section IV, the simulation results are evaluated using a validated reference simulation tool and their performance on the utilized test hardware is examined. The last section provides a conclusion as well as recommendations 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 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 [8].

For testing the simulation presented here, the HD84-Matrix-LED-Headlamp developed by Hella KG from Lippstadt 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. 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. In total, the HD84-Matrix-LED-Headlamp therefore has 95 light sources.

In order to simulate the light distribution in any situation, the characteristics of the emitted light must be known for each individual light source. For this purpose, so-called luminous intensity distributions are determined in lighting technology, which describe the luminous intensity of the light source depending on the direction of radiation [9]. These can be obtained for an existing headlamp by a goniophotometer-measurement [10] or for a computer model by ray tracing methods [11].

Figure 1 shows the low beam distribution of the left HD84-Matrix-LED-Headlamp. The horizontal angle θ around the vertical axis is plotted on the x-axis (0° corresponds to

 ISBN: 978-1-61208-672-9

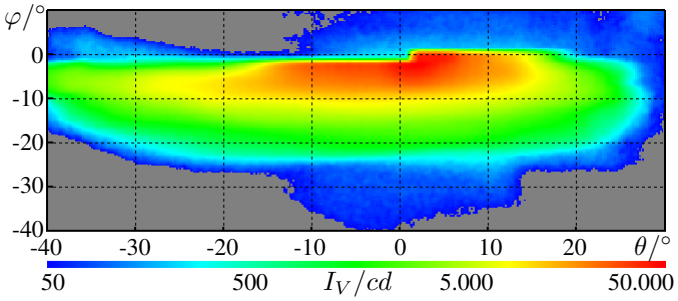

Figure 1. Low-beam distribution of luminous intensity I_V of left HD84-Matrix-LED-Headlamp measured in Candela [cd].

direction of travel). Along the y-axis the vertical angle φ is plotted around the transverse axis. This is varied in the context of headlights in a much smaller range. Within the diagram, the luminous intensity I_V measured in Candela [cd] is color-coded for a given pairing of horizontal and vertical angles.

In order to be able to precisely reference the relevant elements of the light distributions of HD systems in the following sections, these are described more formally. A discretized light distribution can be interpreted as a matrix whose dimensions depend on the considered angular range $[\varphi_0, \varphi_1]$ or $[\theta_0, \theta_1]$ and the corresponding resolution $\Delta\varphi$ or $\Delta\theta$ in horizontal or vertical direction, whereby $M = \frac{\varphi_1 - \varphi_0}{\Delta\varphi}, N = \frac{\theta_1 - \theta_0}{\Delta\theta} = N$ 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 $\varphi_0 + m \cdot \Delta\varphi$ or the vertical angle $\theta_0 + n \cdot \Delta\theta$. 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 $L_k \in \mathbb{R}_{\geq 0}^{M \times N}$.

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 in the discretized direction of the vertical angle φ_m and the horizontal angle θ_n in Candela.

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. 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

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

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

III. Implementation of HD-Headlight-Simulation

The rendering of the dynamic light distributions of HD systems was realized in the development environment Unity (Version 2017.3.1f1 [12]). To ensure realtime capability in terms of a sufficient number of frames per second (fps), the high parallelism of the calculation must be exploited. For this reason, not only the lighting of the scene, but also the determination of the time-dependent total light distribution L is outsourced to the GPU, which can solve such tasks many times faster than the CPU. Consequently, both tasks are implemented as shaders in the language Cg (C for graphics) [13] and can thus be integrated into the programmable rendering pipeline [14]. First of all, it should be mentioned that because of its advantages with many light sources deferred shading is used [15]. In the following section, the generation of the total light distribution from the individual light distributions by the CookieCombiner-Shader is discussed. Afterwards the Headlight-Shader used for lighting the scene is introduced.

A. CookieCombiner-Shader

Textures, whose pixels correspond to the matrix entries, are used to represent the light distributions on digital level. Each pixel carries RGBA color information, with the first three channels determining its color in the RGB color space and the fourth channel (α-channel) determining the transparency of the pixel. In principle, the direction-dependent variation of the light colour would be possible via the RGB channels. In the presented solution, however, these channels are not considered, which is the reason why only monochrome light can be displayed. The directional luminous intensity is coded in the α-channel. Up to now, the texture considers an angle range of $180°$ in both vertical and horizontal direction and resolves in $0.2°$ steps. It therefore has a size of 900x900 pixels. In computer graphics, both the horizontal (u) and vertical (v) coordinates for accessing a texture and the permissible values of individual pixel information are normalized to the interval $[0, 1]$. To map the coded luminous intensity to this value range, all entries of the matrices $L_1, \ldots, L_K$ are divided by their maximum l_{max}.

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 CookieCombiner-Shader. Its task is to combine the cookies of individual light sources for total light distribution according to (1). Its render target is a cookie whose α-values are all initialized with 0. The shader is now repeatedly called on this render target and receives a single light distribution L_k and its weighting factor as parameters. In the fragment program of the shader, the α-values of the respective cookie $\alpha_{uv}(k)$ with $u, v \in [0, 1]$ are multiplied by the weighting factor and then transferred to the render target by additive blending [14]. To ensure that the α-values of the render target do not exceed the permissible value range $[0, 1]$, the weighting factor used is formed as the product of the relative electrical current i_k and the reciprocal of the total number K of all light sources. If the previous considerations are combined, the α-value α_{uv} of the render target at the normalized texture coordinates (u, v) after passing through the shader for all single light distributions is calculated as follows

$$\alpha_{uv} = \sum_{k=1}^{K} \frac{i_k}{K} \cdot \alpha_{uv}(k) = \frac{1}{K} \cdot \sum_{k=1}^{K} i_k \cdot \alpha_{uv}(k). \quad (2)$$

With (2) and $\alpha_{uv}(k), i_k \in [0, 1]$ it follows that $\alpha_{u,v}$ is also within the permissible value range. After blending, the render target contains the normalized luminous intensity of the entire headlamp. The texture coordinates correspond to the spatial directions into which the light radiates according

 ISBN: 978-1-61208-672-9

to the explanations at the beginning of Section II. In order to reconstruct the actual luminous intensity, the normalizations required by the limited range of values must be reversed. Formally, the relationship between the original matrix entry $l(m,n)$ in L and the corresponding texture value $\alpha_{\frac{m}{M}\frac{n}{N}}$ is defined as follows

$$l(m,n) = K \cdot l_{max} \cdot \alpha_{\frac{m}{M}\frac{n}{N}}, \tag{3}$$

which ensures undistorted computational mapping of luminous intensity. The render target of the CookieCombiner-Shader represents the total light distribution of the headlamp in the form of a cookie and serves as input for the Headlight-Shader presented below.

B. Headlight-Shader

The implementation of the Headlight-Shader is much more extensive. Berssenbrügge et al. solved a similar problem in [2] by using a built-in spotlight and mapping the light distribution to its cookie scheme. The present contribution is based on Unity's built-in shader for deferred shading, but differs in the light volume, used to describe the illumination range of the headlamp, and the interpretation of the light cookie. Moreover, no shadows are rendered, as their calculation is very computation-intensive. Also, the driver and headlamps are in approximately the same position, so that shadows are hardly noticed by the driver.

Following the deferred shading procedure, the scene in the initial base path is rendered into the g-buffer (geometry buffer) independent of any lights. A very simple example of a scene is shown in Figure 2. That way, the complex 3D scene is reduced to the 2-dimensional screen output. The information relevant for lighting, such as the surface normals, depth values or color information, is stored in the g-buffer. The Headlight-Shader in the subsequent lighting pass can refer to this information.

In the lighting pass, the illumination range of the light is described by a light volume in the form of a half-sphere, which is illustrated by yellow lines in Figure 2. The light source is located in the center of the half-sphere and is oriented vertically to the plane half-spherical surface in the direction of the curvature. The light volume, approximated by a mesh of 65 vertices, is rendered, which is why the light space l is the current object space.

According to the schema of the graphics pipeline, the vertices of the mesh are first processed by the vertex program. Since the Headlight-Shader implements per-fragment-lighting, the vertex program can be realized by a few lines of code. Essentially, three parameters are generated. First, a vertex ${}_lp'$ of the light volume mesh, passed in homogeneous coordinates, 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 M, 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 2). c has the same origin as v, but distorts the coordinates in perspective for 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 (visible part of the scene in the form of a pyramid frustum, see thick dashed lines in Figure 2) in accordance with the perspective division in the

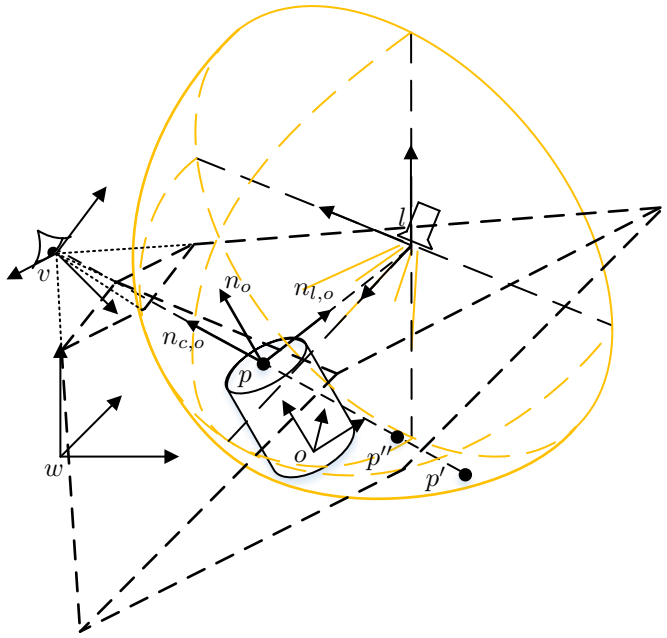

Figure 2. Simple scene to be rendered with coordinate systems for world space (w), view space (v), light space (l) and an exemplarily object with local space o.

fragment program. This information is then used to correctly address the g-buffer and forms 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 M and V (line 4). This value is needed next to ${}_cp'_{uv}$ to reconstruct the 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: ${}_cp' \leftarrow P \cdot V \cdot M \cdot {}_lp'$ // 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: ${}_vp' \leftarrow V \cdot M \cdot {}_lp'$ // 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 interpolated to the fragments according to their distances to the vertices. The fragment shader is called for all fragments covered by the light volume with the respective interpolation results ${}_cp'_{uv}, {}_vp'$.

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: ${}_wl \leftarrow M[1:4,4]$ // 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

 ISBN: 978-1-61208-672-9

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

16: $att \leftarrow \frac{1}{l.range^2} \cdot {}_{w}v_{l,o} \cdot {}_{w}v_{l,o}$ // 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$(c_o, {}_{w}n_o, {}_{w}n_{c,o}, {}_{w}n_{l,o}, l.color)$

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. The vector p' is mapped to the same screen position as p (see dashed line through p, p' and p'' in Figure 2). Therefore ${}_{v}p'$ describes the cameras view direction to p in v. The g-buffer created in the base path of deferred shading can be used to determine the exact position of p on the corresponding line. It encodes the z coordinate 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 ${}_{c}p'_{uv}$ is defined in the vertex shader, whose x- and y-coordinates lie in the interval $[0, {}_{c}p'_{v}.w = {}_{c}p'_{uv}.w]$ 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 ${}_{v}p'$ to the far clipping plane receiving ${}_{v}p''$ (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 ${}_{w}n_{o,c}$ (see Figure 2) 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 ${}_{w}l$ is extracted from the matrix M (line 8). Since the Headlight-Shader only renders the mesh of the light volume into the light buffer, the transformation matrix M 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. Since the light source is in the coordinate origin of l, the translation in M corresponds to the position of the light in w and can be read from the fourth column of M. Now the normalized direction vector from the object point to the light source in world space ${}_{w}n_{l,o}$ (see Figure 2) can be determined by the lines 9 and 10 similar to the previous section.

In the lines 11 to 15, the directional luminous intensity generated as cookie (T_{cookie}) in the CookieCombiner-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 and vertical angles of the incident light beam with respect to the light centre axis must be calculated. For this purpose, the object point belonging to the fragment is transferred to the light space l through multiplying by L. 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 ${}_{l}p$ 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 must be addressed with normalized texture coordinates. Therefore, the angles moving in the range $[-90^{\circ}, +90^{\circ}]$ due to the used light volume are transformed by line 14 to texture coordinates a_{uv}. Thereby, the applicable value can finally be read out by the light distribution (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 ${}_{w}v_{l,o}$ 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}, which is normalized to the interval $[0, 1]$, att is multiplied by this value. The calculated light 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 a Unity-internal local lighting model has been used. This receives the previously determined normals ${}_{w}n_{o,c}, {}_{w}n_{l,o}$, the surface normal ${}_{w}n_{o}$ (see Figure 2) and material data c_o of the object from the g buffer 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 is presented, the results are evaluated in terms of subjective appearance and computation time.

A. Appearance

A strictly objective evaluation based on numerical measurements apart from the legal requirements is difficult in the context of headlighting. Instead, a subjective comparison of the implemented simulation and the LightDriver software from Hella KG (64 Bit Version built on Jul, 2017) is used as the primary evaluation basis. LightDriver is a validated software tool for simulating headlamp light in night driving, which Hella KG uses for its own development work. In contrast to the implementation presented here, the LightDriver is not able to display the high dynamics of HD systems in realtime, since it has been primarily designed for the representation of static light distributions. However, they are sufficient for the plausibility check concerning the appearance of the presented implementation and serve as a reference.

Figure 3 compares the dipped beam distribution of the HD84-Matrix-LED-Headlamp as calculated by

 ISBN: 978-1-61208-672-9

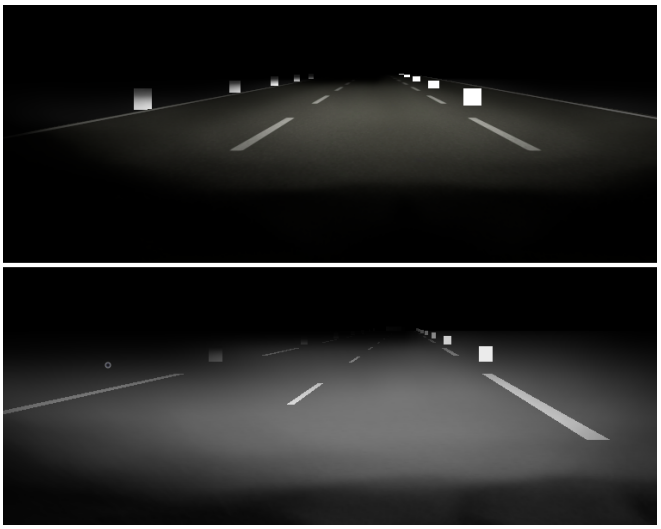

Figure 3. Low-beam light of HD84-Matrix-LED-Headlamp projected on a street with auxiliary surfaces simulated with own implementation (top) and LightDriver (bottom).

CookieCombiner- and Headlight-Shader with the light distribution of the LightDriver as reference. 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 simpler input in the form of a single light distribution, 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 geometric dimensions, textures and colours of scene objects are not exactly the same, but should suffice for a plausibility check.

At first glance you can see that the overall brightness differ between both images. This difference is due to the different textures. On closer inspection, one notices that the LightDriver shows weak light curtains in the vehicle apron, which are not reproduced in the implementation presented. There are also qualitative differences in distance-dependent light attenuation. The reasons for this are on the one hand the too coarse resolution of 8 bit (α-channel of a RGBA texture) used for the luminous intensity and on the other hand the different light models. Essentially, however, there is a strong similarity between the light distributions of both simulations. They have comparable expansions and the same qualitative profile. If one looks at the auxiliary surfaces at the left and right edge of the road, it is noticeable that both light distributions illuminate one's own side of the road much more strongly in the distance than the one next to it. This is a classic feature of low-beam light distributions that ensures that oncoming traffic is not glared.

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.

Although on Figure 4 both measuring walls are each 10 meters away, different distances seem to be present. This impression is deceptive because of different road textures leading to a distorted perception, especially due to the positions

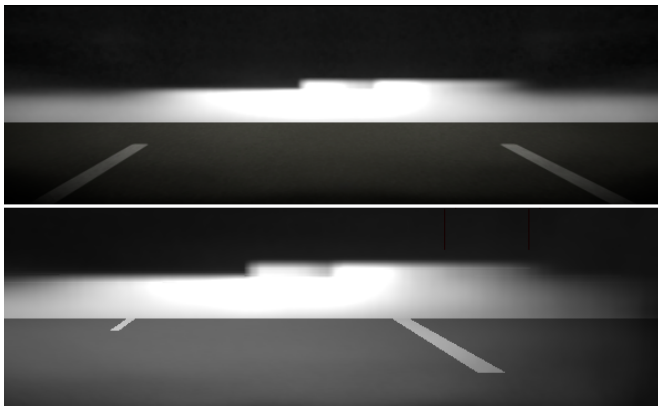

Figure 4. Low-beam light of HD84-Matrix-LED-Headlamp simulated with own implementation (top) and LightDriver (bottom) on a measurement wall.

of the central strips. In addition, the parameters of the camera are not known in the LightDriver and therefore there is a different view frustum in the implementation presented here. The already mentioned disadvantages of the too low resolution of the presented implementation are expressed by a sharper light-dark boundary. Apart from these inequalities the vertical projection shows the similarity of both simulations even clearer than Figure 3. They have a dipped beam typical border to the left in the upper middle area. The high resolution of the HD system allows a sharp cut along the vertical center line and thus ensures optimum long-range illumination of one's own lane without restricting other traffic participants. In addition to the comparison shown, the high beam distribution was also tested with a high degree of consistency.

B. Computation Time

With regard to the realtime requirement, the performance of the implementation has to be taken into account. Our workstation PC, which is representative for current average computers, as test hardware is sufficient for a fundamental evaluation of the realtime capability. This is a 64bit Windows 10 system with an Intel Core i7-6820HQ CPU (2.7 GHz, 8 cores), 16GB RAM and an NVIDIA Quadro M3000M with 4GB VRAM. The runtime of the rendering can be classified as sufficient, if the calculation of the entire scene can be carried out at least with the maximum frame rate of 60 fps of a classical output device.

The independent evaluation of the performance of the Headlight-Shader is only conditionally meaningful due to the many influences, which result from the non realtime capable operating system, the test hardware and the measurement itself. For this reason, the implemented lights are compared with Unity Spotlights, which represent the light sources with the most similar properties. A large, flat surface serves as a test scenario. In the middle of it, the camera is oriented parallel to the ground. The scene can be rendered in 6.11 ms (164 fps). In front of the camera position, 50 Unity Spotlights are placed and parameterized with a range of 60 length units and a spot angle of 179° (max. value). The central axis of the light corresponds to the direction of the camera's view. The calculation time is recorded over 1000 frames. The average value for the calculation of a frame is 9.78 ms (102 fps). In the same scenario, the Unity Spotlights are now replaced by as many lights from the implementation presented here with the same range parameter. In this case, the average calculation time

 ISBN: 978-1-61208-672-9

is 19.60 ms (51 fps). After subtraction of the rendering of the unexposed scene, 0.07 ms per Unity Spotlight and 0.27 ms per headlamp are required. Even if the Headlight-Shader turns out to be slower than a Unity Spotlight, it is suitable for realtime simulation even including external traffic. In particular, the computing time is not negatively affected by more complex scenes, but depends only on the resolution of the output device (here 1354x873 pixels) due to the use of deferred shading.

In addition to the Headlight-Shader, the performance of the CookieCombiner-Shader must be checked. The headlamp ECU (Electronic Control Unit) is clocked at 50 Hz, so that a recalculation in each frame should be assumed for the CookieCombiner-Shader. Due to the difficulties in measuring computing time mentioned in the previous section, an artificial test scenario is also created for the CookieCombiner-Shader. To obtain the most representative value possible, 1,000 total light distributions of the HD84-Matrix-LED-Headlamp are determined by iteration. Since the shader is called once per single light distribution, this corresponds to 95,000 calls of the CookieCombiner-Shader. The average calculation time of a total light distribution is 0.304 ms. Accordingly, CookieCombiner- and Headlight-Shader use a comparable amount of resources, so that a total of 0.6 ms is required for the calculation of a headlamp. Considering complex scenes and the integration of external traffic, further optimization potentials should be researched.

V. Conclusion and Future Work

This contribution presents an approach for realtime simulation of dynamic HD headlamp systems and thus lays the foundation for the simulation-based development of high-resolution dynamic light functions. Taking into account the rather average test hardware and the optimization possibilities discussed below, the shader-based implementation shows potential to simulate night driving with dynamic HD system in reduced scenes.

The rendering of the headlight is divided into two steps. In the first step, the current total light distribution is determined. This 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 with a resolution of 900x900 pixels and superimposed by blending the results of the CookieCombiner-Shader applied to them. The total light distribution initiates the second rendering step as input for the Headlight-Shader. In the lighting pass of deferred shading, this shader determines the final color values of the output device pixels, taking into account the light influences.

The LightDriver software from Hella KG is used to evaluate the results. As a tool for headlamp development that has been established for years, it is an excellent reference. As the Figures 3 and 4 show, the implemented simulation is similar to the LightDriver. The differences can mainly be traced back to the unequal scenes and light models, as well as the differently resolved light intensity. The last two points are starting points for future work.

The performance analysis shows that Headlight- and CookieCombiner-Shader are of comparable complexity and lead to a calculation time of 0.6 ms per headlamp. For driving simulations with the integration of external traffic, these execution times should be further reduced. Therefore two ideas could be persued. On the one hand, the previously considered angular range in the vertical dimension could be reduced from $[-90°, +90°]$ to about $[+10°, -25°]$ without hiding relevant areas of the light distribution. This would decrease the required pixels per cookie significant, reducing computing operations and memory required. A more sophisticated coding of the light distribution including additional color channels could also be considered to reduce memory and increase the resolution of the luminous intensity. On the other hand, it would be possible to segment the calculation of the total light distribution to divide it into several frames. However, this way only the frame rate, not the update rate of the light distribution can be increased.

Acknowledgment

The authors would like to thank Hella KG 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] P. Lecocq, J.-M. Kelada, and A. Kemeny, "Interactive Headlight Simulation," Driving Simulation Conference, 1999.

[2] J. Berssenbrügge, J. Gausemeier, G. M., 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] "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/index.cfm?pid=1&pk=629502&p=1 [retrieved: 8, 2018].

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

[10] 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.

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

[12] "Unity Homepage," URL: https://unity3d.com/ [retrieved: 8, 2018].

[13] 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.

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

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

[16] F. Pedrotti, L. Pedrotti, W. Bausch, and H. Schmidt, Eds., Optik für Ingenieure - Grundlagen, 4. Auflage (Optics for Engineers - Basics, 4th edition). Springer, Berlin, 2007, ISBN: 9783540734710.

 ISBN: 978-1-61208-672-9