Band
410

Verlagsschriftenreihe des Heinz Nixdorf Instituts
Prof. Dr.-Ing. Ansgar Trächtler (Hrsg.)
Regelungstechnik und Mechatronik

Nico Rüddenklau

# Hardware-in-the-Loop-Simulation von HD-Scheinwerfer-Steuergeräten zur Entwicklung von Lichtfunktionen in virtuellen Nachtfahrten

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

*Nico Rüddenklau*

# *Hardware-in-the-Loop-Simulation von HD-Scheinwerfer-Steuergeräten zur Entwicklung von Lichtfunktionen in virtuellen Nachtfahrten*

# *Hardware-in-the-Loop Simulation of HD-Headlamp Control Units for Development of Lighting Functions in Virtual Night Drives*

**Bibliografische Information Der Deutschen Bibliothek**
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie. Detaillierte bibliografische Daten sind im Internet über `http://dnb.ddb.de` abrufbar.

Band 410 der Verlagsschriftenreihe des Heinz Nixdorf Instituts



ISSN (Online): 2365-4422
ISBN: 978-3-947647-29-3



Als elektronische Version frei verfügbar über die Digitalen Sammlungen der Universitätsbibliothek Paderborn.

Satz und Gestaltung: Nico Rüddenklau

# Hardware-in-the-Loop-Simulation von HD-Scheinwerfer-Steuergeräten zur Entwicklung von Lichtfunktionen in virtuellen Nachtfahrten

zur Erlangung des akademischen Grades
DOKTOR DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)
der Fakultät Maschinenbau
der Universität Paderborn

genehmigte
DISSERTATION

von
Nico Rüddenklau
aus Liebenau

| | |
|---|---|
| Tag des Kolloquiums: | 14.12.2022 |
| Referent: | Prof. Dr.-Ing. Ansgar Trächtler |
| Korreferent: | Prof. Dr.-Ing. Tran Quoc Khahn |

## Vorwort

Die vorliegende Arbeit „Hardware-in-the-Loop-Simulation von HD-Scheinwerfer-Steuergeräten zur Entwicklung von Lichtfunktionen in virtuellen Nachtfahrten“ entstand während meiner Forschungstätigkeit am Heinz Nixdorf Institut der Universität Paderborn. Sie spiegelt die Erfahrungen und Forschungsergebnisse wider, die ich bei der Bearbeitung des mir aufgetragenen Forschungsprojekts „Smart Headlamp Technology“ (SHT) sammeln konnte [GR17].

Herrn Prof. Dr.-Ing. Ansgar Trächtler danke ich für die Betreuung dieser Arbeit und das stets entgegen gebrachte Vertrauen, welches mir ermöglichte, eigene Ideen und Wege der Problemlösung mit vollem Rückhalt zu verfolgen. Außerdem danke ich ihm für die vielen fachlichen und strategischen Gespräche, die mich nicht nur in Bezug auf diese Arbeit, sondern auch auf die Herausforderungen meines weiteren beruflichen Werdegangs vorbereitet haben.

Bei Herrn Prof. Dr.-Ing. Tran Quoc Khanh möchte ich mich für die Übernahme des Korreferats bedanken. Ebenso danke ich dem Vorsitzenden der Prüfungskommission Prof. Dr.-Ing. Walter Sextro sowie Prof. Dr. Burkard Wördenweber für den Beisitz.

Zudem bedanke ich mich bei allen Kolleginnen und Kollegen, den studentischen Hilfskräften sowie den Bachelor-, Studien- und Masterarbeitern, die mich während meiner Zeit am Heinz Nixdorf Institut begleitet und unterstützt haben. Ein besonderer Dank gilt meiner Teamleiterin Dr.-Ing. Sandra Gausemeier und meinen Kollegen Patrick Biemelt, Sven Mertin, Kevin Malena und Christopher Link für die inhaltlichen Ratschläge, die angenehme kollegiale Atmosphäre und die stetige Unterstützung in kritischen Phasen. Ebenso hervorheben möchte ich die studentischen Hilfskräfte Marcel Weise und Simon Gorissen, die durch ihre hohe Auffassungsgabe sowie schnelle und qualitativ hochwertige Arbeit große Teile zum erfolgreichen Abschluss der vorliegenden Arbeit beigetragen haben. Gleiches gilt für den Studenten Christoph Vlad, welcher mir in meiner Einarbeitungsphase voller Geduld unzählige Fragen aus dem Themenfeld der Computergrafik mit einem faszinierenden Wissensschatz beantworten konnte.

Für das gründliche Korrekturlesen der Arbeit und die Einbringung von Verbesserungsvorschlägen bedanke ich mich ebenfalls bei den bereits genannten Kollegen und studentischen Hilfskräften.

Großen Dank schulde ich meiner Familie, allen voran meinen Eltern für ihr Vertrauen in meine Fähigkeiten und die Möglichkeit, meine Interessen ungehindert zu verfolgen. Zu guter Letzt danke ich meiner Freundin Melanie für ihre moralische Unterstützung und die damit verbundene Balance zwischen meiner wissenschaftlichen Arbeit und dem persönlichen Lebensbereich.

Paderborn, 19.05.2022 Nico Rüddenklau

## Vorveröffentlichungen


[BHR+18] Biemelt, P.; Henning, S.; Rüddenklau, N.; Gausemeier, S.; Trächtler, A.: A Model Predictive Motion Cueing Strategy for a 5-Degree-of-Freedom Driving Simulator with Hybrid Kinematics. *Proceedings of the DSC 2018 Europe VR. Driving Simulation & Virtual Reality Conference & Exhibition* (2018)

[BRM+19] Biemelt, P.; Rüddenklau, N.; Mertin, S.; Gausemeier, S.; Trächtler, A.: Evaluation of a Novel Filter-Based Motion Cueing Algorithm in Comparison to Optimization-Based Control in Interactive Driving Simulation. *International Conference on Advances in System Simulation (SIMUL) IARIA* (2019)

[BRM+20] Biemelt, P.; Rüddenklau, N.; Mertin, S.; Gausemeier, S.; Trächtler, A.: Design and Evaluation of a Novel Filter-Based Motion Cueing Strategy for a Hybrid Kinematics Driving Simulator with 5-Degrees-of-Freedom. *Proceedings of the DSC 2020 Europe VR. Driving Simulation & Virtual Reality Conference & Exhibition* (2020)

[HBR+18] Henning, S.; Biemelt, P.; Rüddenklau, N.; Gausemeier, S.; Trächtler, A.: A Simulation Framework for Testing a Conceptual Hierarchical Autonomous Traffic Management System including an Intelligent External Traffic Simulation. *Proceedings of the DSC 2018 Europe VR. Driving Simulation & Virtual Reality Conference & Exhibition* (2018)

[RBH+18] Rüddenklau, N.; Biemelt, P.; Henning, S.; Gausemeier, S.; Trächtler, A.: Shader-Based Realtime Simulation of High-Definition Automotive Headlamps. *International Conference on Advances in System Simulation (SIMUL) IARIA* (2018)

[RBM+19a] Rüddenklau, N.; Biemelt, P.; Mertin, S.; Gausemeier, S.; Trächtler, A.: Real-Time Lighting of High-Definition Headlamps for Night Driving Simulation. *IARIA SysMea* 12 (2019), S. 72–88

[RBM+19b] Rüddenklau, N.; Biemelt, P.; Mertin, S.; Gausemeier, S.; Trächtler, A.: Simulation-Based Lighting Function Development of High-Definition Headlamps. *ISAL International Symposium of Automotive Lighting* (2019)

[RGT19] Rüddenklau, N.; Gausemeier, S.; Trächtler, A.: Hardware-in-the-Loop Simulation of High-Definition Headlamp Systems. *9. VDI/VDE-Fachtagung AUTOREG* (2019)

[RGT21] Rüddenklau, N.; Gausemeier, S.; Trächtler, A.: Simulative Development of Object-Based Adaptive Front Lighting. *ISAL International Symposium of Automotive Lighting* (2021)

## Zusammenfassung

Die vorliegende Arbeit befasst sich mit dem simulationsbasierten Entwurf hochauflösender Pixel-Scheinwerfersysteme durch virtuellen Nachtfahrten. Nach einer Darstellung der notwendigen theoretischen Grundlagen wird zunächst der derzeitige Stand der Technik beschrieben. Besonderes Augenmerk erhält die simulationsgestützte Entwicklung von Pixel-Scheinwerfersystemen. Die existierenden Lösungen werden vorgestellt und bewertet. Abgeleitet aus den vorhandenen Schwächen derzeitiger Nachtfahrtsimulationen wird ein Anforderungskatalog erarbeitet, den die hier vorgestellte Lösung bestmöglich erfüllen soll. Es folgt die Beschreibung der zu diesem Zweck entwickelten Nachtfahrtsimulation „Hyperion“. Nach einer Darstellung der Gesamtarchitektur, der dazugehörigen Komponenten und ihrer Wechselwirkungen, wird der Forschungskern der vorliegenden Arbeit detailliert betrachtet. Hierzu gehört im ersten Schritt die echtzeitfähige und qualitativ hochwertige Nachbildung des Lichts von Pixel-Scheinwerfern in einer virtuellen Umgebung. Bei der Virtualisierung von Pixel-Systemen werden neben dem ausgesandten Licht beider Scheinwerfer auch das Steuergerät und damit verbundene Sensoren betrachtet. Im zweiten Schritt werden darauf aufbauend Analyse- und Entwurfsverfahren für diese Systeme methodisch eingeführt, prototypisch implementiert und validiert. Hierbei liegt der Fokus nicht auf der optischen Auslegung des Scheinwerfers, sondern auf dem Entwurf von Lichtfunktionen zur situationsadaptiven Steuerung der zahlreichen Lichtquellen eines Pixel-Scheinwerfers. Schließlich wird die entwickelte Lösung anhand des zuvor angefertigten Anforderungskatalogs bewertet. Zum Abschluss werden die zentralen Ergebnisse der Arbeit zusammengefasst. In einem Ausblick werden weitere Potentiale und Ausbaumöglichkeiten der Nachtfahrtsimulation diskutiert.

## Abstract

This thesis deals with the simulation-based design of high definition pixel headlamp systems by virtual night driving. After a presentation of the necessary theoretical basics the current state of the art is presented. Special attention is given to the simulation-based development of pixel headlamp systems. The existing solutions are presented and evaluated. Derived from the prevailing weaknesses of current night driving simulations, a catalog of requirements will be worked out, which the solution presented here should fulfill in the best possible way. The description of the night driving simulation „Hyperion“ developed for this purpose follows. After a presentation of the total architecture, the relevant components and their interactions, the research core of the present work is regarded in detail. This includes in the first step the real-time capable and high-quality reproduction of the light of pixel headlamps in a virtual environment. In the virtualization of pixel systems, the control unit and the sensors connected to it are considered in addition to the emitted light of both headlamps. In a second step, analysis and design procedures for these systems are methodically introduced, prototypically implemented and validated. Here, the focus is not on the optical design of the headlamp, but on the design of light functions for situation-adaptive controlling the numerous light sources of a pixel headlamp. The developed solution is evaluated on the basis of the previously prepared requirements catalog. Finally, the central results of the work are summarized. In an outlook, further potentials and expansion possibilities of the night driving simulation are discussed.

# Hardware-in-the-Loop-Simulation von HD-Scheinwerfer-Steuergeräten zur Entwicklung von Lichtfunktionen in virtuellen Nachtfahrten

## Inhaltsverzeichnis

## Anhang

## Abkürzungsverzeichnis

| | |
|---|---|
| AABB | Axis Aligned Bounding Box |
| ABL | Abblendlicht |
| ADAMS | Automatic Dynamic Analysis of Mechanical Systems |
| AFS | Advanced Frontlighting System |
| API | Application Programming Interface |
| ASM | Automotive Simulation Models |
| ATMOS | Atlas Motion System |
| BMBF | Bundesministerium für Bildung und Forschung |
| BMWi | Bundesministerium für Wirtschaft und Energie |
| BRDF | Bidirektionale Reflektanzverteilungsfunktion |
| BSDRF | Bidirektionale Streuungs- und Reflektanzverteilungsfunktion |
| BSSRDF | Bidirektionale Oberflächenstreuungs-Reflektanzverteilungsfunktion |
| BTDF | Bidirektionale Durchdringungsverteilungsfunktion |
| BVH | Bounding Volume Hierarchy |
| CAD | Computer Aided Construction |
| CAN | Controller Area Network |
| Cg | C for Graphics |
| CGI | Computer Generated Imagery |
| CIE | Internationale Beleuchtungskommission |
| CL | Kurvenlicht |
| CPU | Central Processing Unit |
| CRI | Colour Rendering Index |
| CZL | Baustellenlicht |
| DBL | Dynamic Bending Light |
| DIN | Deutsches Institut für Normumg e.V. |
| DLP | Digital Light Processing |
| DMD | Digital Microscopic Mirror Device |
| EFRE | Europäische Fonds für regionale Entwicklung |
| EN | Europäische Norm |
| FHD | Full HD |
| FL | Fernlicht |
| FoV | Field of View |
| GFHB | Glare Free High Beam |
| GLSL | Open GL Shading Language |
| GPGPU | General Purpose GPU |
| GPU | Graphical Processing Unit |

| | |
|---|---|
| HD | High Definition |
| HDG | Hell-Dunkel-Grenze |
| HDR | High Dynamic Range |
| HFoV | Horizontal FoV |
| HiL | Hardware-in-the-Loop |
| HLSL | High Level Shading Language |
| HNI | Heinz Nixdorf Institut |
| IES | Illuminating Engineering Society |
| IR | Infrarot |
| ISO | Internationale Organisation für Normung |
| JOSM | Java OpenStreetMap Editor |
| JSON | Java Script Object Notation |
| k-DOP | k-Discretely Oriented Polytopes |
| KIT | Karlsruher Institut für Technologie |
| LCD | Liquid Crystal Display |
| LD | Laserdiode |
| LDR | Low Dynamic Range |
| LED | Licht emittierende Diode |
| LiDAR | Light Detection and Ranging |
| MEMS | Micro-elektromechanische Systeme |
| MiL | Model-in-the-Loop |
| MRT | Multiple Render Targets |
| NDC | Normalized Device Coordinates |
| NURBS | Non-uniform rational B-Splines |
| OBB | Oriented Bounding Box |
| ODR | OpenDRIVE |
| OEM | Original Equipment Manufacturer |
| OSM | OpenStreetMap |
| QHD | Quad HD |
| RADAR | Radio Detection and Ranging |
| RoSSHAF | Robuste Sensorik für hochautomatisiertes Fahren |
| SHT | Smart Headlamp Technology |
| SI | Internationales Einheitensystem |
| SiL | Software-in-the-Loop |
| SSL | Solid State Lighting |
| SUMO | Simulation for Urban Mobility |

| | |
|---|---|
| SVBRDF | Räumlich variierende BRDF |
| UDP | User Datagram Protocol |
| UHD | Ultra HD |
| USB | Universal Serial Bus |
| UTM | Universal Transverse Mercator |
| UV | Ultraviolett |
| VFoV | Vertical FoV |
| VR | Virtual Reality |
| XiL | Überbegriff für MiL, SiL und HiL |
| XML | Extensible Markup Language |

## Symbolverzeichnis

| Name | Beschreibung | Einheit |
|---|---|---|
| $A_E$ | Empfängerfläche | $m^2$ |
| $A_K$ | Kugeloberfläche | $m^2$ |
| $A_P$ | Primärvalenz eines beliebigen Farbraums (analog: $B_P$, $C_P$) | – |
| $A_S$ | Senderfläche | $m^2$ |
| $A_{Fzg}$ | Fahrzeugstirnfläche | $m^2$ |
| $A_{TM}$ | Skalierungsparameter der $\gamma$-Kompression | – |
| $A$ | Farbwert bezüglich der Primärvalenz $A_P$ (analog: $B$, $C$) | – |
| $C_a$ | Reflektiertes Licht bei ambienter Reflexion als RGB-Farbvalenz | – |
| $C_d$ | Reflektiertes Licht bei diffuser Reflexion als RGB-Farbvalenz | – |
| $C_l$ | Lichtfarbe als RGB-Farbvalenz | – |
| $C_s$ | Reflektiertes Licht bei spiegelnder Reflexion als RGB-Farbvalenz | – |
| $C_\Sigma$ | Summe aus $C_a$, $C_d$ und $C_s$ | – |
| $C_{a,m}$ | Ambiente Reflexion eines Material bzgl. der RGB-Primärvalenzen | – |
| $C_{d,m}$ | Diffuse Reflexion eines Material bzgl. der RGB-Primärvalenzen | – |
| $C_{s,m}$ | Spiegelnde Reflexion eines Material bzgl. der RGB-Primärvalenzen | – |
| $E_e$ | Bestrahlungsstärke | $\frac{W}{m^2}$ |
| $E_v$ | Beleuchtungsstärke | $\frac{lm}{m^2}$ |
| $E_{\lambda_i}$ | Farbvalenz des schmalbandigen Spektralanteils $\lambda \in [\lambda_i, \lambda_i + \Delta\lambda]$ des energiegleichen Spektrums | – |
| $E_{mn,k}$ | Durch Lichtquelle $k$ im Segment $(m,n)$ realisierte Beleuchtungsstärke | $lx$ |
| $E_{mn,ref}$ | Im Segment $(m,n)$ einzustellende Beleuchtungsstärke | $lx$ |
| $E_{mn}$ | Insgesamt im Segment $(m,n)$ realisierte Beleuchtungsstärke | $lx$ |
| $E_{v,\varnothing}$ | Im Raumwinkel $R_{sel}$ gemittelte Beleuchtungsstärke | $lx$ |
| $E_{v,ref}$ | Im Raumwinkel $R_{sel}$ einzustellende Beleuchtungsstärke | $lx$ |
| $F_L$ | Luftwiderstandskraft | $N$ |
| $F_z$ | Radaufstandskraft | $N$ |
| $F_{s,max}$ | Bei $s_{max}$ übertragene Reifenlängs-/Reifenquerkraft | $N$ |
| $F$ | Allgemeine Farbvalenz eines beliebigen Farbraums ($F \in \mathbb{R}^3$) | – |
| $G$ | Anzahl der Lichtquellen-Gruppen eines Scheinwerfers | – |

| Name | Beschreibung | Einheit |
|---|---|---|
| $I_e$ | Strahlstärke | $\frac{W}{sr}$ |
| $I_v$ | Lichtstärke | $cd$ |
| $I_{v,a}$ | Lichtstärke eines ambienten Lichts (analog: Beleuchtungsstärke $E_{v,a}$) | $cd$ |
| $I_{v,d}$ | Lichtstärke eines gerichteten Lichts (analog: Beleuchtungsstärke $E_{v,d}$) | $cd$ |
| $I_{v,i}(\Omega)$ | Lichtstärke der Lichtquelle $i \in I$ in Richtung $\Omega$ | $cd$ |
| $I_{v,k,\varnothing}$ | Mittlere Lichtstärke der Lichtquelle $k$ im Winkelbereich $R_{l,k}$ | $cd$ |
| $I_{v,k,ref}$ | Beitrag der Lichtquelle $k$ zu $I_{v,ref}$ | $cd$ |
| $I_{v,ref}$ | Im Raumwinkel $R_{sel}$ einzustellende Lichtstärke | $cd$ |
| $I$ | Indexmenge der geometrisch voneinander abgegrenzten Lichtquellen einer Szene | – |
| $K'_m$ | Maximalwert des photometrischen Strahlungsäquivalents für skotopisches Sehen | $\frac{lm}{W}$ |
| $K_m$ | Maximalwert des photometrischen Strahlungsäquivalents für photopische Sehen | $\frac{lm}{W}$ |
| $K_o$ | maximale Anzahl von Lichtquellen, die das selbe Raumwinkelelement bestrahlen | – |
| $K$ | Anzahl der Lichtquellen eines Scheinwerfers bzw. eines HD-Moduls | – |
| $L_A$ | Leuchtdichtebeiwert der Primärvalenz $A_P$ (analog: $L_B$, $L_C$) | $\frac{cd}{m^2}$ |
| $L_F$ | Leuchtdichtebeiwert der Farbvalenz $F$ | $\frac{cd}{m^2}$ |
| $L_R$ | Leuchtdichtebeiwert der Primärvalenz $R_P$ des RGB-Farbraums (analog: $L_G$, $L_B$) | $\frac{cd}{m^2}$ |
| $L_X$ | Leuchtdichtebeiwert der Primärvalenz $X_P$ des XYZ-Farbraums (analog: $L_Y$, $L_Z$) | $\frac{cd}{m^2}$ |
| $L_e$ | Strahldichte | $\frac{W}{sr \cdot m^2}$ |
| $L_g$ | Diskretisierte Gesamtlichtverteilung der Lichtquellen aus der Gruppe $g \in 1, \ldots, G$ | – |
| $L_k$ | Diskretisierte Lichtverteilung der Lichtquelle $k$ | – |
| $L_v(v, \Omega)$ | Leuchtdichte an der Position $v$ in Richtung $\Omega$ | $\frac{cd}{m^2}$ |
| $L_v$ | Leuchtdichte | $\frac{cd}{m^2}$ |
| $L_\Sigma$ | Momentane, diskretisierte Gesamtlichtverteilung des Scheinwerfers bzw. des HD-Moduls | – |
| $L_{v,i}(v, \Omega)$ | Beitrag der Lichtquelle $i \in I$ zu $L_v(v, \Omega)$ | $\frac{cd}{m^2}$ |
| $M_g$ | Zeilenzahl von $L_g$ | – |
| $M_k$ | Zeilenzahl von $L_k$ | – |
| $M_s$ | Zeilenzahl des segmentierten Ausleuchtungsbereichs eines HD-Moduls | – |
| $M_\Sigma$ | Zeilenzahl von $L_\Sigma$ | – |

| Name | Beschreibung | Einheit |
|---|---|---|
| $M$ | Transformationsmatrix von $m$ nach $w$ in $\mathbb{R}^{4\times4}$ | – |
| $N_g$ | Spaltenzahl von $L_g$ | – |
| $N_k$ | Spaltenzahl von $L_k$ | – |
| $N_s$ | Spaltenzahl des segmentierten Ausleuchtungsbereichs eines HD-Moduls | – |
| $N_{MC}$ | Stichprobengröße bei der Monte-Carlo-Integration | – |
| $N_{Ph}$ | Photonenzahl | – |
| $N_\Sigma$ | Spaltenzahl von $L_\Sigma$ | – |
| $P$ | Transformationsmatrix von $e$ nach $c$ in $\mathbb{R}^{4\times4}$ | – |
| $Q_e$ | Strahlungsenergie | $J$ |
| $Q_v$ | Lichtmenge | $lm \cdot s$ |
| $Q_{opt}$ | Matrix des quadratischen Anteils eines quadratischen Optimierungsproblem ($Q \in \mathbb{R}^{K\times K}$) | – |
| $R_P$ | Rote Primärvalenz des RGB-Farbraums (analog: $G_P$, $B_P$) | – |
| $R_h$ | Rotationsmatrix in $\mathbb{R}^{4\times4}$ | – |
| $R_{l,k}$ | Von Lichtquelle $k$ ausgeleuchteter Raumwinkelbereich | – |
| $R_{o,k}$ | Überlappungsbereich von $R_{o,k}$ und $R_{sel}$ | – |
| $R_{rot}$ | Rotationsmatrix in $\mathbb{R}^{3\times3}$ | – |
| $R_{sel}$ | Raumwinkelbereich, in dem Beleuchtungsstärke $E_{v,ref}$ umgesetzt werden soll | – |
| $R$ | Farbwert bezüglich der Primärvalenz $R_P$ (analog: $G$, $B$) | – |
| $S_h$ | Skalierungsmatrix in $\mathbb{R}^{4\times4}$ | – |
| $S$ | Skalierungsmatrix in $\mathbb{R}^{3\times3}$ | – |
| $T_h$ | Translationsmatrix in $\mathbb{R}^{4\times4}$ | – |
| $T_v$ | Zeitkonstante der Dynamik der Fahrzeuggeschwindigkeit | $s$ |
| $T_{SB}$ | Transformationsmatrix von $n$ nach $s$ in $\mathbb{R}^{4\times4}$ | – |
| $T_{dim}$ | Zeitkonstante der Dimmdynamik der Lichtquellen | $s$ |
| $T$ | Allgemeine Transformationsmatrix in $\mathbb{R}^{4\times4}$ | – |
| $V'(\lambda)$ | Hellempfindlichkeitsfunktion für skotopisches Sehen | – |
| $V(\lambda)$ | Hellempfindlichkeitsfunktion für photopisches Sehen | – |
| $V$ | Transformationsmatrix von $w$ nach $e$ in $\mathbb{R}^{4\times4}$ | – |
| $X_P$ | X-Primärvalenz des XYZ-Farbraums (analog: $Y_P$, $Z_P$) | – |
| $X_e$ | Allgemeine radiometrische Größe | $[X_e]$ |
| $X_v$ | Allgemeine photometrische Größe | $[X_v]$ |
| $X_{e,\lambda}$ | Spektralverteilung von $X_e$ | $\frac{[X_e]}{m}$ |
| $X$ | Farbwert bezüglich der Primärvalenz $X_P$ (analog: $Y$, $Z$) | – |
| $\Delta T_{gap}$ | Zeitlücke zwischen $s_p$ und $s_f$ | $s$ |
| $\Delta T$ | Diskretisierungsschrittweite | $s$ |
| $\Delta\Psi_{pf}$ | Gierwinkelunterschied auf $c_{Tr}$ zwischen $s_p$ und $s_f$ | $rad$ |

| Name | Beschreibung | Einheit |
|---|---|---|
| $\Delta\bar{\theta}_{fb}$ | Maximaler vertikaler Abstand zwischen $f_{l,k}$ und $b_{l,k}$ zur Berücksichtigung der Lichtquelle $k$ | $rad$ |
| $\Delta\bar{\varphi}_{fb}$ | Maximaler horizontaler Abstand zwischen $f_{l,k}$ und $b_{l,k}$ zur Berücksichtigung der Lichtquelle $k$ | $rad$ |
| $\Delta\lambda$ | Breite eines Wellenlängenintervalls | $m$ |
| $\Delta\theta_{fb,k}$ | Vertikaler Abstand zwischen $f_{l,k}$ und $b_{l,k}$ | $rad$ |
| $\Delta\theta$ | Diskretisierungsschrittweite des Polarwinkels | $rad$ |
| $\Delta\varphi_{fb,k}$ | Horizontaler Abstand zwischen $f_{l,k}$ und $b_{l,k}$ | $rad$ |
| $\Delta\varphi$ | Diskretisierungsschrittweite des Azimutwinkels | $rad$ |
| $\Delta v$ | Verschiebungsvektor in $\mathbb{R}^3$ | – |
| $\Omega'_n$ | Stichprobenelement aus $\Omega^+$ | $sr$ |
| $\Omega^+$ | Hemisphäre um einen Flächenpunkt | $sr$ |
| $\Omega_E$ | Raumwinkel der sendenen Fläche aus Sicht des Empfängers | $sr$ |
| $\Omega_S$ | Raumwinkel der empfangenden Fläche aus Sicht des Senders | $sr$ |
| $\Omega_k$ | Raumwinkel von $R_{l,k}$ | $sr$ |
| $\Omega$ | Allgemeiner Raumwinkel | $sr$ |
| $\Phi_v$ | Lichtstrom | $lm$ |
| $\Phi_{EE,e,\lambda}$ | Spektralverteilung der Strahlungsleistung des energiegleichen Spektrums | $\frac{W}{m}$ |
| $\Phi_{e,\lambda_i}$ | Strahlungsleistunganteil im Wellenlängebereich $[\lambda_i, \lambda_i + \Delta\lambda]$ | $W$ |
| $\Phi_{mn,k}$ | Lichtstrom der Lichtquelle $k$ im Winkelbereich des Segments $(m, n)$ | $lm$ |
| $\Phi_{v,k}$ | Lichtstrom der Lichtquelle $k$ im Winkelbereich $R_{l,k}$ | $lm$ |
| $\Phi_{v,p}$ | Lichtstrom einer Punktlichtquelle (analog: Lichtstärke $I_{v,p}$, Beleuchtungsstärke $E_{v,p}$) | $lm$ |
| $\Phi_{v,s}$ | Lichtstrom eines Spotlichts (analog: Lichtstärke $I_{v,s}$, Beleuchtungsstärke $E_{v,s}$) | $lm$ |
| $\Psi$ | Gierwinkel eines Fahrzeugs | $rad$ |
| $\alpha_h$ | Horizontales Field of View | $rad$ |
| $\alpha_v$ | Vertikales Field of View | $rad$ |
| $\bar{\lambda}$ | Obere Grenze eines Wellenlängenbereichs | $m$ |
| $\bar{\theta}_k$ | Polarwinkel der oberen Grenze des Ausleuchtungsbereichs der Lichtquelle $k$ | $rad$ |
| $\bar{\theta}$ | Polarwinkel der oberen Grenze des Ausleuchtungsbereichs von $L_\Sigma$ | $rad$ |
| $\bar{\varphi}_k$ | Azimutwinkel der rechten Grenze des Ausleuchtungsbereichs der Lichtquelle $k$ | $rad$ |
| $\bar{\varphi}$ | Azimutwinkel der rechten Grenze des Ausleuchtungsbereichs von $L_\Sigma$ | $rad$ |

| Name | Beschreibung | Einheit |
|---|---|---|
| $\bar{a}(\lambda)$ | Spektralwertfunktion bezüglich der Primärvalenz $A_P$ (analog: $\bar{b}(\lambda)$, $\bar{c}(\lambda)$) | – |
| $\bar{m}_k$ | Größter Zeilenindex in $L_k$ (analog: Spaltenindex $\bar{n}_k$) | – |
| $\bar{m}$ | Größter Zeilenindex in $L_\sum$ (analog: Spaltenindex $\bar{n}$) | – |
| $\bar{r}(\lambda)$ | Spektralwertfunktion bezüglich der Primärvalenz $R_P$ des RGB-Farbraums (analog: $\bar{g}(\lambda)$, $\bar{b}(\lambda)$) | – |
| $\bar{x}(\lambda)$ | Spektralwertfunktion bezüglich der Primärvalenz $X_P$ des XYZ-Farbraums (analog: $\bar{y}(\lambda)$, $\bar{z}(\lambda)$) | – |
| $\bar{I}_{v,k}$ | Maximal auftretende Lichtstärke der Lichtquelle $k$ | $cd$ |
| $\bar{\theta}_{l,k}$ | Obere Grenze von $R_{l,k}$ | $rad$ |
| $\bar{\theta}_{mn}$ | Obere Grenze des Segments $(m,n)$ des Ausleuchtungsbereichs | $rad$ |
| $\bar{\varphi}_{l,k}$ | Rechte Grenze von $R_{l,k}$ | $rad$ |
| $\bar{\varphi}_{mn}$ | Rechte Grenze des Segments $(m,n)$ des Ausleuchtungsbereichs | $rad$ |
| $\delta_i$ | Lenkwinkel des kurveninneren Rads (analog: kurvenaußen $\delta_o$) | $rad$ |
| $\delta_s$ | Öffnungswinkel eines Spotlichts | $rad$ |
| $\delta$ | Abstrahlwinkel bezüglich Mittelachse des Spotlichts | $rad$ |
| $\epsilon_{\Omega' r}$ | Winkel zwischen $\Omega'$ und $r_{spec}$ | $rad$ |
| $\epsilon_{er}$ | Winkel zwischen $e$ und $r_{spec}$ | $rad$ |
| $\frac{dv}{d\Psi}$ | Bremsverhalten des Fahrzeugs abhängig von der Krümmung des Straßenverlaufs | $\frac{m}{s \cdot rad}$ |
| $\gamma_{TM}$ | Exponentieller Parameter der $\gamma$-Kompression | – |
| $\gamma_e$ | Exponentieller Parameter der logarithmischen Wahrnehmung des menschlichen Auges | – |
| $\lambda$ | Wellenlänge | $m$ |
| $\rho_L$ | Luftdichte | $\frac{kg}{m^3}$ |
| $\tau$ | Kollisionswahrscheinlichkeit eines Lichtstrahls im Nebel pro LE (Nebeldichte) | – |
| $\theta_E$ | Winkel zwischen Flächennormale und empfangenem Lichtstrahl (Polarwinkel) | $rad$ |
| $\theta_S$ | Winkel zwischen Flächennormale und abgesandtem Lichtstrahl (Polarwinkel) | $rad$ |
| $\theta$ | Allgemeiner Polarwinkel | $rad$ |
| $\underline{\lambda}$ | Untere Grenze eines Wellenlängenbereichs | $m$ |
| $\underline{\theta}_k$ | Polarwinkel der unteren Grenze des Ausleuchtungsbereichs der Lichtquelle $k$ | $rad$ |
| $\underline{\theta}$ | Polarwinkel der unteren Grenze des Ausleuchtungsbereichs von $L_\sum$ | $rad$ |

| Name | Beschreibung | Einheit |
|---|---|---|
| $\underline{\varphi}_k$ | Azimutwinkel der linken Grenze des Ausleuchtungsbereichs der Lichtquelle $k$ | $rad$ |
| $\underline{\varphi}$ | Azimutwinkel der linken Grenze des Ausleuchtungsbereichs von $L_\Sigma$ | $rad$ |
| $\underline{m}_k$ | Kleinster Zeilenindex in $L_k$ (analog: Spaltenindex $\underline{n}_k$) | – |
| $\underline{m}$ | Kleinster Zeilenindex in $L_\Sigma$ (analog: Spaltenindex $\underline{n}$) | – |
| $\underline{\theta}_{l,k}$ | Untere Grenze von $R_{l,k}$ | $rad$ |
| $\underline{\theta}_{mn}$ | Untere Grenze des Segments $(m,n)$ des Ausleuchtungsbereichs | $rad$ |
| $\underline{\varphi}_{l,k}$ | Linke Grenze von $R_{l,k}$ | $rad$ |
| $\underline{\varphi}_{mn}$ | Linke Grenze des Segments $(m,n)$ des Ausleuchtungsbereichs | $rad$ |
| $\varphi_E$ | Azimutwinkel eines eintreffenden Lichtstrahls | $rad$ |
| $\varphi_S$ | Azimutwinkel eines abgesandten Lichtstrahls | $rad$ |
| $\varphi$ | Allgemeiner Azimutwinkel | $rad$ |
| ${}_T u$ | Allgemeine horizontale Texturkoordinate $\in [0,1]$ | – |
| ${}_T v$ | Allgemeine vertikale Texturkoordinate $\in [0,1]$ | – |
| ${}_c v$ | Vektor $v$ im Clipspace $c$ | – |
| ${}_e v$ | Vektor $v$ im Eye-/Viewspace $e$ | – |
| ${}_m v$ | Vektor $v$ im Model-/Objectspace $m$ | – |
| ${}_n v$ | Vektor $v$ in NDC-Koordinaten $n$ | – |
| ${}_p v$ | Projektion des Vektors $v$ auf die Near Clipping Plane | – |
| ${}_s v$ | Vektor $v$ im Screenspace $s$ | – |
| ${}_w v$ | Vektor $v$ im Worldspace $w$ | – |
| $a_r$ | Augpunkt (entspricht Kameraposition) | – |
| $a_{ref}$ | Reflexionswahrscheinlichkeit eines Lichtstrahls nach Kollision im Nebel | – |
| $a$ | Farbwertanteil bezüglich der Primärvalenz $A_P$ (analog: $b$, $c$) | – |
| $b_{Fzg}$ | Spurbreite des Fahrzeugs | $m$ |
| $b_{l,k}$ | Zu $f_{l,k}$ nächstgelegener Randpunkt auf $R_{sel}$ | – |
| $b_{out}$ | Breite der Ausgabe | $m$ |
| $c_K$ | Kugelzentrum | – |
| $c_j$ | Interpolationsfaktor zwischen $d_{std,k}$ und der Sollvorgabe einer dynamischen Lichtfunktion für die Lichtquelle $j$ | – |
| $c_w$ | Strömungswiderstandskoeffizient | – |
| $c_{HDR}$ | Farbkoordinate im HDR | – |
| $c_{LDR}$ | Farbkoordinate im LDR (normiert) | – |
| $c_{Tr}$ | Trajektorie des Fahrzeugs als bezüglich der Bogenlänge parametrisierte Kurve | – |

| Name | Beschreibung | Einheit |
|---|---|---|
| $c_{opt}$ | Vektor des linearen Anteils eines quadratischen Optimierungsproblem ($c_{opt} \in \mathbb{R}^K$) | – |
| $c_{phy}$ | Physikalische Helligkeit | – |
| $c_{sen}$ | Wahrgenommene Helligkeit | – |
| $c_{spec}(\lambda)$ | Spektralfarbenzug | – |
| $dA'$ | Projiziertes differentielles Flächenelement | $m^2$ |
| $dA_p$ | Projektion von $dA$ auf Kugelfläche | $m^2$ |
| $dA$ | Differentielles Flächenelement | $m^2$ |
| $dE$ | Differentielles Energiepaket | $J$ |
| $d\Omega$ | Differentielles Raumwinkelelement | $sr$ |
| $d\Phi_e$ | an $dA$ empfangene Strahlungsleistung | $W$ |
| $d_k$ | Momentaner, normierter Dimmwert der Lichtquelle $k$ | – |
| $d_r$ | Richtungsvektor eines Rays | – |
| $d_s$ | Mittelachse des Spotlichts | – |
| $d_{std,k}$ | Standarddimmwert der Lichtquelle $k$ | – |
| $e$ | Normierter Richtungsvektor von bestrahlter Fläche zur Kamera | – |
| $f_c$ | Distanz zwischen Kamera und Far Clipping Plane | $m$ |
| $f_r$ | Bidirektionale Reflektanzfunktion | – |
| $f_{TM}$ | Tone Mapping Funktion | – |
| $f_{l,k}$ | Zentrum von $R_{l,k}$ | – |
| $f_{o,k}$ | Zentrum von $R_{o,k}$ | – |
| $f_{opt}$ | Konstanter Anteil eines quadratischen Optimierungsproblems ($f_{opt} \in \mathbb{R}$) | – |
| $f$ | Frequenz | $Hz$ |
| $g_r$ | Geradengleichung eines Rays in Parameterform | – |
| $h_{out}$ | Höhe der Ausgabe | $m$ |
| $h$ | Plancksches Wirkungsquantum | $Js$ |
| $l(m,n)$ | Eintrag in Zeile $m$ und Spalte $n$ aus $L_\Sigma$ | – |
| $l_k(m,n)$ | Eintrag in Zeile $m$ und Spalte $n$ aus $L_k$ | – |
| $l_{Fzg}$ | Radstand des Fahrzeugs | $m$ |
| $l$ | Normierter Richtungsvektor von bestrahlter Fläche zur Lichtquelle | – |
| $n_c$ | Distanz zwischen Kamera und Near Clipping Plane | $m$ |
| $n_r$ | Abschwächung der spiegelnden Reflexion abhängig von $\epsilon_{er}$ | – |
| $n_s$ | Abschwächung des Spotlichts abhängig von $\delta$ | – |
| $n$ | Normalenvektor einer Fläche | – |
| $p(\Omega)$ | Wahrscheinlichkeitsdichtefunktion über $\Omega^+$ | – |

| Name | Beschreibung | Einheit |
|---|---|---|
| $p_{Ph}(\Omega', \Omega)$ | Wahrscheinlichkeit für die Reflexion eines Lichtstrahls aus Rtg. $\Omega'$ in Rtg. $\Omega$ (Phasenfunktion) | – |
| $r_K$ | Kugelradius | $m$ |
| $r_i$ | Abstand der Lichtquelle $i \in I$ vom bestrahlten Punkt | $m$ |
| $r_{mn}$ | Abstand zwischen Lichtquelle und bestrahlter Fläche in Richtung des Segments $(m, n)$ | $m$ |
| $r_{o,k}$ | Abstand zwischen Lichtquelle $k$ und bestrahlter Fläche in Richtung $f_{o,k}$ | $m$ |
| $r_{spec}$ | Normierter Richtungsvektor der idealen spiegelnden Reflexion | – |
| $r$ | Abstand zw. Lichtquelle und bestrahlter Fläche | $m$ |
| $s_f$ | Bogenlängenparameter des Vorausschaupunkts des Fahrzeugs auf $c_{Tr}$ | $m$ |
| $s_p$ | Bogenlängenparameter der momentanen Fahrzeugposition auf $c_{Tr}$ | $m$ |
| $s_\infty$ | Schlupfwert, bei dem sich ein asymptotischer Kraftverlauf einstellt | $\frac{m}{s}$ |
| $s_{max}$ | Schlupfwert mit maximaler Reifenlängs-/Reifenquerkraft | $\frac{m}{s}$ |
| $s$ | Relativgeschwindigkeit zwischen Radnaben- und Radumfangsgeschwindigkeit (Schlupf) | $\frac{m}{s}$ |
| $t_r$ | Ray-Parameter | – |
| $t$ | Zeit | $s$ |
| $v(s)$ | Über Bogenlänge parametrisierter Pfad im Raum | – |
| $v_x$ | $x$-Koordinate des Vektors $v$ (analog: $v_y$, $v_z$, $v_w$) | $m$ |
| $v_{Fzg}$ | Fahrzeuggeschwindigkeit | $\frac{m}{s}$ |
| $v_{alt}$ | Geschwindigkeit des Fahrzeugs im vorhergehenden Zeitschritt | $\frac{m}{s}$ |
| $v_{ist}$ | Momentane Geschwindigkeit des Fahrzeugs | $\frac{m}{s}$ |
| $v_{soll}$ | Sollgeschwindigkeit des Fahrzeugs | $\frac{m}{s}$ |
| $v$ | allgemeiner Vektor | – |
| $w_{mn}$ | Gewichtung des Segments $(m, n)$ in der Zielfunktion eines Optimierungsproblems | – |
| $x_w$ | Basisvektor bezüglich der $x$-Koordinate im Worldspace $w$ (analog: $y_w$, $z_w$) | – |
| $x_{opt}$ | Vektor der Optimierungsvariablen eines quadratischen Optimierungsproblems ($x_{opt} \in \mathbb{R}^K$) | – |

# 1 Einleitung

Die Entwicklung automobiler Scheinwerfer schritt in den letzten Jahren rasant voran. Während die früheren Scheinwerfersysteme ausschließlich über statische Komponenten verfügten, kamen im Laufe der Zeit stetig dynamische Elemente hinzu. Motiviert wird diese Entwicklung durch den Wunsch, das ausgesandte Licht an die jeweilige Umgebungssituation anzupassen. Diese Situationsadaptivität bezieht sich inzwischen nicht mehr ausschließlich auf die bestmögliche Ausleuchtung des Fahrzeugumfelds und der Entblendung anderer Verkehrsteilnehmer, sondern auch auf gänzlich neue Aspekte, wie der Kommunikation mit dem Fahrer und weiteren Verkehrsteilnehmern durch Symbolprojektionen.

Im ersten Schritt wurde diese Adaptivität durch die Einbringung verschiedener Lichtquellen im Scheinwerfer realisiert. Durch unterschiedliche Abstrahlcharakteristika der Leuchtmittel und Reflektor- bzw. Streuscheibengeometrien konnte durch das Um- oder Zuschalten der Lichtquellen zwischen definierten Lichtverteilungen gewechselt werden. Allen voran sind hier die Abblend- und Fernlichtfunktion zu nennen. Darauf folgten mechanische Aktoren im Scheinwerfer, welche die Lage der Lichtquelle oder des Reflektors manipulieren. Durch diese wurde es möglich, die Lichtverteilung horizontal und vertikal zu verschieben, wie es beispielsweise für die dynamische Leuchtweitenregelung oder das Kurvenlicht eingesetzt wird. Zur Umsetzung eines blendfreien Fernlichts wurde in frühen Phasen eine rotierende Walze eingesetzt, die abhängig vom Rotationswinkel unterschiedliche Konturen aufweist und auf diese Weise verschiedene Lichtverteilungen auf der Straße realisiert.

Heutige Scheinwerfersysteme verzichten weitgehend auf mechanische Aktoren. Stattdessen ähnelt das Prinzip zur Gestaltung der Lichtverteilung einem Schwarz-Weiß-Beamer, weshalb der Begriff „Pixellicht“ für diese Systeme etabliert ist. Der insgesamt auszuleuchtende Bereich wird in viele kleine Segmente unterteilt, wobei jedes Segment exklusiv durch eine Lichtquelle bestrahlt wird. Hierzu ist eine Vielzahl einzelner Lichtquellen bzw. Pixellichter notwendig. Jedes Pixellicht kann individuell gedimmt werden. Durch die Gesamtheit der Dimmwerte kann die Lichtverteilung des Scheinwerfers in den Grenzen der Auflösung bausteinartig zusammengesetzt werden. Die Flexibilität dieser Systeme übersteigt die Möglichkeiten der zuvor genannten Entwicklungsstufen um ein Vielfaches.

Zur Realisierung von Pixellicht existieren verschiedene Technologien. Nicht alle verwenden für jedes Segment des Raumwinkelbereichs eine physische Lichtquelle. So wird beispielsweise in DLP (Digital Light Processing)-Scheinwerfern eine leuchtstarke Lichtquelle eingesetzt, deren Licht durch einzeln ansteuerbare Mikrospiegel in die verschiedenen Raumrichtungen abgelenkt werden kann. Das grundsätzliche Prinzip zur Gestaltung der Gesamtlichtverteilung bleibt jedoch erhalten.

Diese Arbeit befasst sich mit der simulationsbasierten Entwicklung derartiger Pixelsysteme. Im nachfolgenden Abschnitt werden die Herausforderungen genannt, die im Zuge der Pixellicht-Technologie auf den Entwicklungsingenieur zukommen. Daraus abgeleitet wird in Abschnitt 1.2 die Zielsetzung der Arbeit, deren Ergebnisse den Ingenieur in bestmöglicher Weise beim Entwurf derartiger Systemen unterstützen sollen. Abschließend zeigt Abschnitt 1.3 die Gliederung der Arbeit auf.

## 1.1 Problemstellung

Im Entwicklungsprozess von Fahrzeugscheinwerfern sind besondere Erschwernisse zu überwinden. Neben den vielfältigen Produktanforderungen seitens der OEM und des Gesetzgebers sowie dem hohen Zeit- und Kostendruck, sind im Kontext von Scheinwerfern die aufwendigen Testvoraussetzungen und der hohe subjektive Anteil innerhalb der Evaluierung zu nennen. Scheinwerfer können unter realen Bedingungen nur in nächtlichen Testfahrten erprobt werden, welche sehr zeit- und kostenintensiv sind. Hinzu kommt, dass die Prototypen üblicherweise auf einem Versuchsträger (Rack) am Fahrzeug installiert werden, wodurch kein Fußgängerschutz gewährleistet ist. Bild 1-1 zeigt ein solches Rack an einem Versuchsfahrzeug. Darüber hinaus ist während der Erprobung im realen Umfeld mit potentiell gefährlichen Fehlfunktionen zu rechnen. Neben den zuvor genannten Problemen ist also auch der Sicherheitsaspekt kritisch zu sehen. Aus diesen Gründen weichen Scheinwerferhersteller soweit möglich auf statische Tests aus. Im Bild 1-2 wird der in Lippstadt befindliche 140m lange Lichtkanal des Scheinwerferherstellers HELLA GmbH & Co. KGaA (nachfolgend: HELLA) gezeigt. Damit entfallen viele der genannten Nachteile. Gleichzeitig ist jedoch die Erprobung dynamischer Fahrfunktionen in einer statischen Verkehrssituation nur sehr begrenzt möglich.

*Bild 1-1*: *Rack zur Erprobung von Scheinwerfer-Prototypen montiert an einem Testfahrzeug der HELLA.*

Bedingt durch die genannten Schwierigkeiten wurde bereits vor dem Aufkommen der Pixel-Technologie auf die simulationsbasierte Erporbung virtueller Prototypen in frühen Phasen der Entwicklung gesetzt. Diese Erprobung geschieht in Form von virtuellen Nachtfahrten. Um eine vollständige Testabdeckung zu erzielen, muss nicht nur das Scheinwerfersystem, sondern auch die gesamte Wechselwirkung mit der Umgebung virtualisiert werden. Konkret sind in diesem Kontext das von den Scheinwerfern ausgesandte Licht, dessen Wechselwirkung mit Objekten in der Umgebung, das Scheinwerfersteuergerät, das Testfahrzeug, relevante Sensoren, andere Verkehrsteilnehmer und verschiedene Witterungsverhältnisse zu nennen.

Mit der Fülle neuer Möglichkeiten, die moderne Pixelsysteme bieten, steigen die Anforderungen an das Virtual Prototyping weiter an. Insbesondere die Virtualisierung des

Lichts gewinnt immens an Komplexität. Bisher konnte ein Scheinwerfer durch wenige virtuelle Lichtquellen mit statischen Lichtverteilungen abgebildet werden. Verschwenkaktoren ließen sich durch die Rotation der Lichtquelle berücksichtigen. Zur Virtualisierung von Pixelscheinwerfern ist dieses Vorgehen ungeeignet. Für einen einzigen Scheinwerfer wären dann hunderte bis hin zu einigen zehntausend virtuelle Lichtquellen erforderlich. Die für eine Fahrsimulation grundlegende Echtzeitfähigkeit wäre unter diesen Voraussetzungen nicht gewährleistet. Folglich müssen zur Virtualisierung von Pixel-Scheinwerfern grundlegend neue Ansätze verfolgt werden.

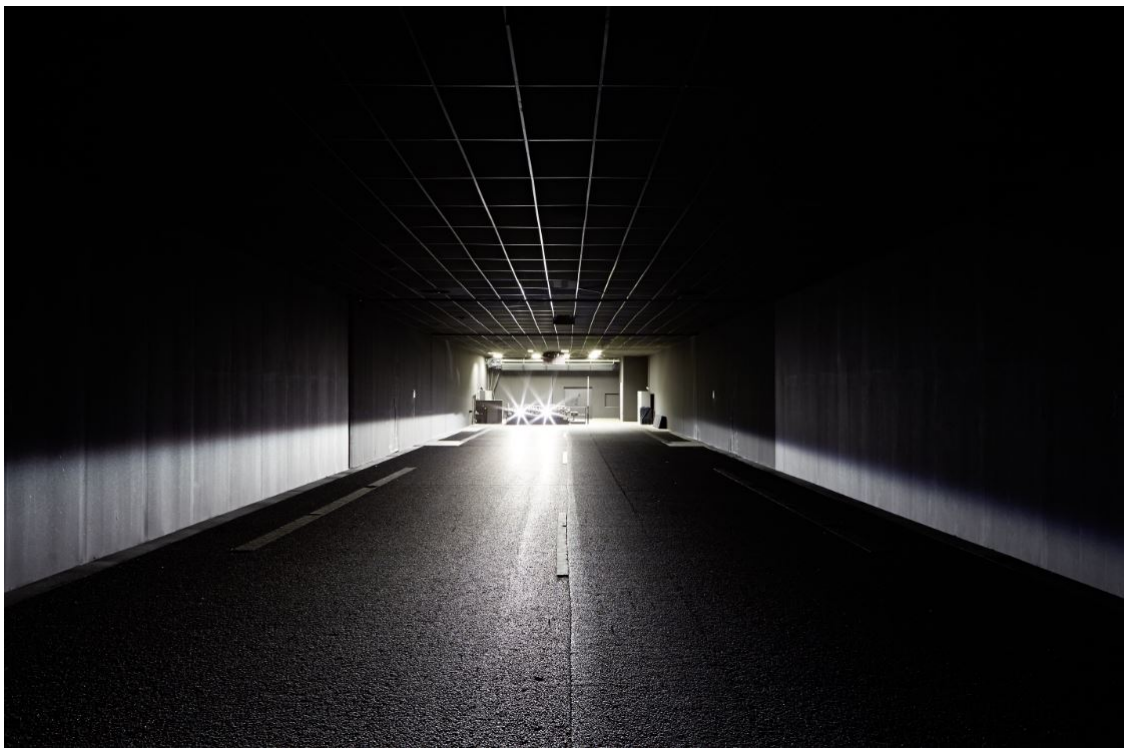

*Bild 1-2*: *Lichtkanal des Scheinwerferherstellers HELLA in Lippstadt mit 140m langer Straße [Quelle: Hella].*

Eine weitere Komponente, deren Komplexitätszuwachs durch die Pixel-Technologie enorm ist, stellt das Steuergerät des Scheinwerfersystems dar. Dessen Aufgabe ist die adäquate Auswahl der Dimmwerte aller Pixellichtlichtquellen zu jedem Fahrzeugzustand und jeder Fahrsituation. Die Wahl der Dimmwerte basiert im Wesentlichen auf den Sensorinformationen und Daten anderer Steuergeräte, die an das Scheinwerfersteuergerät übermittelt werden. Wie genau die Dimmwertvorgabe getroffen wird, hängt von den Lichtfunktionen ab, die auf dem Steuergerät implementiert sind. Aufgrund der hohen Adaptivität von Pixellicht-Systemen lassen sich vielfältigere, exaktere und dynamischere Lichtfunktionen realisieren, als es bei klassischen Systemen der Fall ist. Gleichzeitig steigt die Anzahl der Eingangs- und Ausgangsgrößen einer Lichtfunktion erheblich. So wäre beispielsweise die direkte Vorgabe der Dimmwerte durch den Ingenieur schon für niedrig aufgelöste Systeme nicht mehr praktikabel. Zur Ausschöpfung des Potentials von Pixelsystemen muss die Fülle an Eingangs- und Ausgangsdaten bei der Auslegung von Lichtfunktionen durch geeignete Abstraktionen beherrschbar werden.

## 1.2 Zielsetzung

Motiviert durch die dargelegte Problemstellung ist das Ziel der vorliegenden Arbeit die Entwicklung eines Verfahrens zur echtzeitfähigen und qualitativ hochwertigen Simulation des Lichts von Pixelscheinwerfern im Rahmen von virtuellen Nachtfahrten. Zur Sicherstellung der Praxistauglichkeit soll das Verfahren in eine Nachtfahrtsimulation eingebettet und anhand realer Scheinwerferdatensätze validiert werden. Darauf aufbauend ist ein weiteres Ziel dieser Arbeit die Unterstützung des Lichtingenieurs bei der Auslegung von Lichtfunktionen für Pixelsysteme. Hierzu soll ein Verfahren entwickelt werden, welches die Komplexität der Entwurfsaufgabe drastisch reduziert. Dieses Verfahren soll ebenfalls in die Nachtfahrtsimulation integriert werden. Ein drittes wesentliches Ziel ist die hard- und software-seitige Skalierbarkeit der Nachtfahrtsimulation. Das heißt konkret, dass die Anwendung einerseits kostengünstig am Arbeitsplatz des Lichtingenieurs eingesetzt werden kann und sich andererseits zum Betrieb eines Großsimulators eignet. Außerdem soll eine vollständige Testabdeckung durch MiL-, SiL- und HiL-Tests des Steuergeräts gewährleistet sein.

## 1.3 Vorgehensweise

Das Kapitel 2 legt zunächst die theoretische Basis zum Verständnis der erarbeiteten Verfahren. Hierbei werden vorrangig Inhalte eingeführt, die nicht zwangsläufig zur Wissensbasis im ingenieurwissenschaftlichen Umfeld des Maschinenbaus zählen. Auf oberster Ebene werden die Fachgebiete Optik, Farbmetrik und Computergrafik unterschieden. Bei der Darlegung dieser Disziplinen beschränken sich die Ausführungen auf Inhalte, die im Verlauf dieser Arbeit unmittelbar oder indirekt Anwendung finden.

Im anschließenden Kapitel 3 folgt die Vorstellung des Stands der Technik. Der erste Abschnitt führt in die Scheinwerfertechnik ein und gibt einen groben Überblick über die historische Entwicklung von den ersten einfachen KFZ-Scheinwerfern bis hin zu den modernen Pixelscheinwerfern, die den Betrachtungsgegenstand dieser Arbeit darstellen. In Abschnitt 3.2 folgt eine ausführliche Darstellung der aktuellen Arbeiten zu Lichtfunktionen im Kontext von Pixelsystemen. Abschließend werden die derzeit etablierten Nachtfahrtsimulationen in Abschnitt 3.3 vorgestellt.

Bevor die Lösungen zur Erreichung der in Abschnitt 1.2 dargelegten Ziele präsentiert werden, detailliert Kapitel 4 diese Ziele durch 13 technische Anforderungen. Diese Anforderungen sollten von einer Nachtfahrtsimulation für Pixelsysteme erfüllt werden, damit eine bestmögliche Hilfestellung für den Lichtingenieur erzielt wird. Die in Abschnitt 3.3 vorgestellten Nachtfahrtsimulationen werden hinsichtlich dieser Anforderungen bewertet. Aus den Ergebnissen der Bewertung wird im Abschnitt 4.4 schließlich der Handlungsbedarf abgeleitet.

Mit Kapitel 5 beginnt die Vorstellung der entwickelten Nachtfahrtsimulation, die fortlaufend unter der Bezeichnung „Hyperion“ adressiert wird. Den Leitfaden des Kapitels bildet die Hard- und Softwarearchitektur der Simulation. In den Abschnitten 5.2 bis 5.7 werden Hyperions Komponenten und ihre Wechselwirkungen dargestellt. Den Abschluss des Kapitels 5 bildet die Vorstellung einiger Beispielkonfigurationen in Abschnitt 5.8.

Die detaillierte Betrachtung des Forschungskerns dieser Arbeit erfolgt in Kapitel 6. Dort wird ein Verfahren vorgestellt, welches das Rendering von Pixelsystemen mit einer hohen Anzahl von Lichtquellen im Rahmen einer Nachtfahrtsimulation erlaubt. Die Betrachtung erfolgt zweigeteilt. Zuerst wird in Abschnitt 6.2 die Bestimmung der Gesamtlichtverteilung eines Scheinwerfers basierend auf den momentan vorliegenden Dimmwerten beschrieben. Nachfolgend wird in Abschnitt 6.3 die Implementierung der logischen Ersatzlichtquelle erläutert, welche das Licht des Scheinwerfers in die Szene projiziert. Es schließt sich in Abschnitt 6.4 eine Validierung der Lösung an, wobei aufgrund der besseren Vergleichbarkeit etablierte Simulationssoftware als Referenz herangezogen wird. Außerdem wird in Abschnitt 6.5 die Laufzeit des vorgestellten Verfahrens sowohl komplexitätstheoretisch, als auch anhand von Messungen mit Parametervariationen analysiert. Den Abschluss des Kapitels 6 bildet der Abschnitt 6.6. Dieser stellt die Simulation bestimmter Witterungsverhältnisse und die simulative Nachbildung damit einhergehender Phänomene am Beispiel von Nebel dar.

Aufbauend auf Kapitel 6 führt Kapitel 7 Analyse- und Entwurfsverfahren für Pixelsysteme ein. Die in Abschnitt 7.1 diskutierten Analyseverfahren werden zum Teil von konventionellen Scheinwerfersystemen auf Pixelsysteme adaptiert. Darüber hinaus werden neue Analyseverfahren eingeführt, die speziell zur Analyse des Pixelsystemen zugrunde liegenden Funktionsprinzips ausgelegt sind. Die Designverfahren werden in Abschnitt 7.2 beginnend von der zugrundeliegenden Methodik bis hin zur prototypischen Implementierung vorgestellt und validiert. Beim zuerst vorgestellten Ansatz wird ein besonderes Augenmerk auf Laufzeit gelegt, wodurch sich dieser potentiell zur Integration auf dem Scheinwerfersteuergerät eignet und so eine elegante Implementierung verschiedener Lichtfunktionen ermöglicht. Der zweite Ansatz, vorgestellt in Abschnitt 7.2.4, nutzt globale Optimierungsverfahren und erfordert einen erhöhten Rechenaufwand, liefert aber auch bessere Ergebnisse.

In Kapitel 8 erfolgt die Evaluierung der vorgestellten Nachtfahrtsimulation. Sie wird anhand der Anforderungen vorgenommen, die in Kapitel 4 aufgestellt werden. Eine zusammenfassende Übersicht der Evaluierungsergebnisse erfolgt in Abschnitt 8.2.

Schließlich werden die wesentlichen Ergebnisse der vorliegenden Arbeit in Kapitel 9 zusammengefasst. Außerdem werden Schlussfolgerungen aus den erhaltenen Ergebnissen gezogen, welche in einem Ausblick auf weitere Potentiale und Ausbaumöglichkeiten der vorgestellten Lösung resultieren.

# 2 Grundlagen

Zum Verständnis der nachfolgenden Kapitel sind einige theoretische Grundlagen von Bedeutung, die einem Maschinenbauer nicht zwangsläufig bekannt sind. In den nachfolgenden Abschnitten werden diese Grundlagen unterteilt in die Themengebiete „Optik“, „Farbmetrik“ und „Computergrafik“ vermittelt, sodass eine geeignete Wissensbasis für die folgenden Ausführungen sichergestellt ist.

## 2.1 Optik

Dieser Abschnitt gibt einen Überblick über die optischen Grundlagen, die im Kontext der Aufgabenstellung von Relevanz sind. Dazu werden nach einer allgemeinen physikalischen Einordnung des Lichts Grundgrößen der Strahlungsphysik vorgestellt. Anschließend werden die Wahrnehmung des Lichts durch das menschliche Auge thematisiert und die daraus abgeleiteten Hellempfindlichkeitskurven eingeführt. Durch Bewertung der strahlungsphysikalischen Größen mit diesen Kurven wird schließlich auf die photometrischen Größen übergeleitet, welche die Grundlage für alle nachfolgenden lichttechnischen Betrachtungen darstellen.

Inhaltlich wird in diesem Abschnitt hauptsächlich auf Grundlagenliteratur der Optik Bezug genommen. Konkret sind die Quellen [PPBS05], [Ree62] und [Hen82] in die nachfolgenden Unterabschnitte eingeflossen. Alle vorgestellten Größen und Zusammenhänge werden außerdem in den Teilen 1 bis 3 der DIN 5031 definiert [DIN01].

### 2.1.1 Beschreibung des Lichts

Zur Definition des Lichts bietet es sich an, zunächst einen Blick auf das elektromagnetische Spektrum zu werfen. Licht ist ein kleiner Ausschnitt dieses Spektrums – nämlich elektromagnetische Strahlung im Wellenlängenbereich von 380 bis 780 $nm$ bzw. im Frequenzbereich von 390 und 790 $THz$. Gerade dieses Intervall mit individuellen Abweichungen bezüglich der unteren und oberen Grenze stellt den Teil der elektromagnetischen Strahlung dar, den das menschliche Auge sensieren kann und in Farbreize umsetzt. Bild 2-1 ordnet den sichtbaren Bereich ins elektromagnetische Spektrum ein, welches, wie aus der Literatur bekannt, entlang der Frequenz bzw. der Wellenlänge im Vakuum geordnet ist.

Besonders kurzwelliges Licht nehmen wir als violett wahr, weshalb der Bereich unterhalb der kürzesten sichtbaren Wellenlänge bzw. oberhalb der höchsten sichtbaren Frequenz als ultraviolette (UV) Strahlung bezeichnet wird. Erhöht sich die Wellenlänge, so wird sie durch unser Auge von blau über grün, gelb und orange bis hin zu rot wahrgenommen. Oberhalb der längsten sichtbaren Wellenlänge bzw. unterhalb der niedrigsten sichtbaren Frequenz spricht man von Infrarot (IR)-Strahlung.

Zur Beschreibung von Licht haben sich je nach Anwendungsbereich verschiedene Disziplinen ausgebildet, die das Licht auf verschiedene Arten und in verschiedenen Abstrakti-

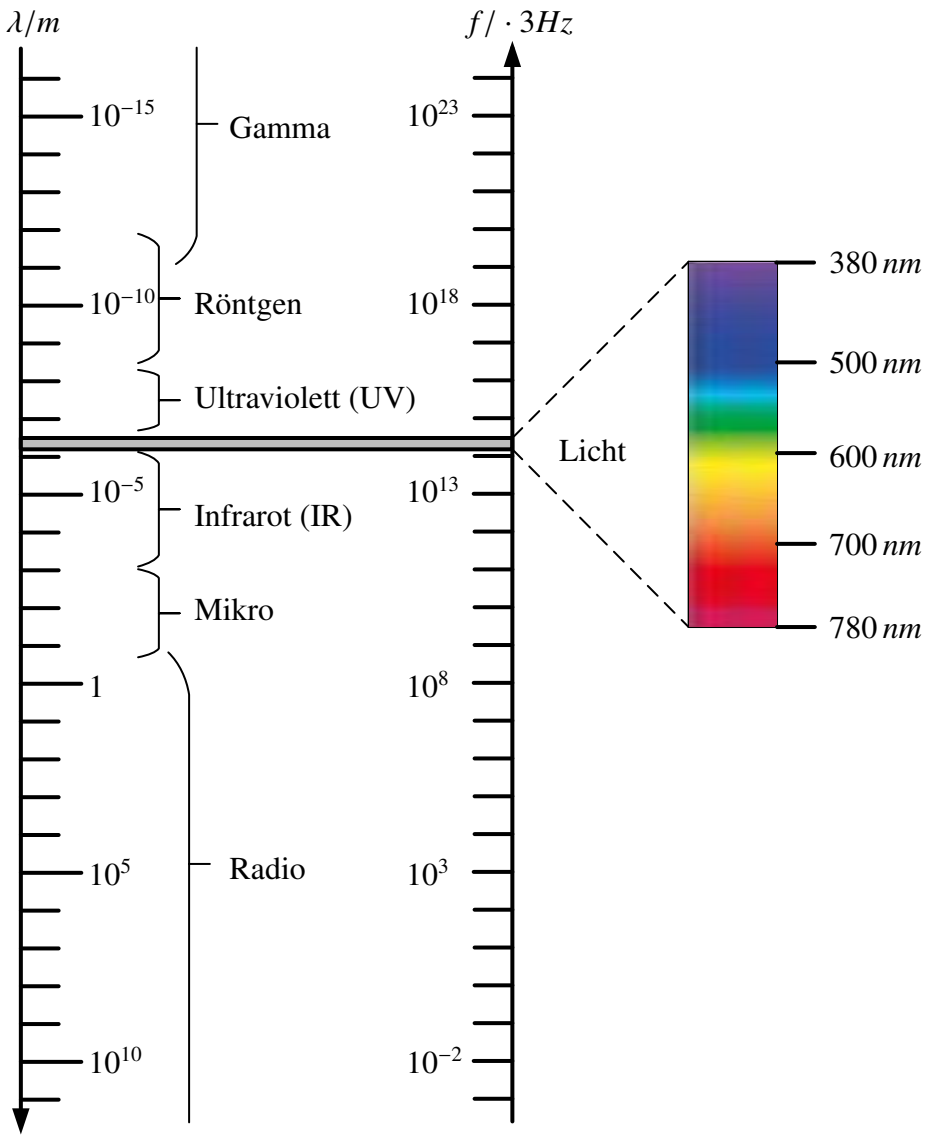


*Bild 2-1*: *Einordnung des Lichts in das elektromagnetische Spektrum.*

onsebenen modellieren. Sie können dabei ganz unterschiedliche Phänomene des Lichts abbilden. Man unterscheidet dabei konkret die Bereiche

- Geometrische Optik,
- Strahlungsoptik bzw. Lichttechnische Optik,
- Wellenoptik
- und Quantenoptik.

Die Quantenoptik erklärt Prozesse der Lichterzeugung und Lichtabsorption. Sie stellt die Lichtstrahlung als Photonenstrom dar, wobei jedes Photon ein von dessen Frequenz abhängiges Energiepaket trägt. In der Wellenoptik befasst man sich mit Lichtausbreitung, Reflexion, Brechung, Interferenz, Beugung und Polarisation. Im Folgenden werden die Wellen- und Quantenoptik ausgeklammert, da die Phänomene, welche in diesen Ansätzen beschrieben werden können, im betrachteten Kontext nicht relevant sind.

In der geometrischen Optik untersucht man die Abbildung von Objekten in Bildern durch Licht unter Einsatz von Linsen, Prismen oder Spiegeln. Licht wird als unendlich dünner Strahl aufgefasst, der Objekt- und Bildpunkte verbindet. Dementsprechend werden Wellenphänomene und Energieaspekte außer Acht gelassen. Die geometrische Optik begegnet dem Leser in dieser Dissertation nur beiläufig. Das Abbildungssystem des Scheinwerfers ist nicht Teil der Betrachtung. Die Bezüge auf die geometrische Optik beschränken sich deshalb lediglich auf die Lichtausbreitung im Raum.

Von hoher Relevanz für die betrachtete Anwendung ist hingegen die Strahlungs- und darauf aufbauend die lichttechnische Optik. In Kurzform befasst sich diese Disziplin mit der Leitung des Lichts in eine gewünschte Richtung durch Lichtquellen und der Wirkung des Lichts auf den Menschen. Aufgrund dieser Zielvorgabe werden folgende Annahmen bei der Modellierung des Lichts zugrunde gelegt:

- geradlinige Ausbreitung in homogenen Medien
- Reflexions- und Brechungsgesetze der Wellenoptik
- Gültigkeit des Energiesatzes
- Amplitude der Lichtstrahlung wird durch Strahlungsleistung oder davon abgeleitete Größen beschrieben
- Beschreibung lichtschwächender, absorbierender und lichtstreuender Eigenschaften von Medien zwischen Lichtquelle und Empfänger durch Koeffizienten
- Beschreibung der Wirkung des Lichts auf Empfänger durch spektrale Empfindlichkeitsfunktionen

Die Bedeutungen und Notwendigkeiten der einzelnen Annahmen werden in den nächsten Abschnitten deutlich. Im Verlauf der Arbeit wird sich zeigen, dass das lichttechnische Modell und dessen zugrunde liegende Annahmen für die betrachtete Anwendung adäquat sind. Deshalb richten sich die Folgeabschnitte diesem Themenkomplex zu und fassen die hier relevanten Aspekte in übersichtlicher Form zusammen.

### 2.1.2 Raumwinkel

Als wichtiges geometrisches Maß wird der Raumwinkel zur Definition grundlegender Größen der Optik herangezogen. Deshalb soll dieser Abschnitt zunächst das Konzept des Raumwinkels und insbesondere dessen Bedeutung im Kontext von Beleuchtungssituationen darstellen, bevor anschließend die in dieser Arbeit relevanten strahlungsphysikalischen bzw. lichttechnischen Größen eingeführt werden. Zum besseren Verständnis setzt Bild 2-2 die zur Definition des Raumwinkels benötigten geometrischen Größen in Bezug. Betrachtet wird ein ebenes Flächenelement mit dem Flächeninhalt $dA$, welches von einer Punktlichtquelle bestrahlt wird. Die Lichteinfallsrichtung wird durch den normierten Vektor $l$ vom Flächenelement zur Lichtquelle beschrieben. $l$ ist für einen hinreichend großen Abstand zwischen Flächenelement und Lichtquelle bzw. ein hinreichend kleines Flächenelement identisch für alle Punkte der Fläche. Die Orientierung der Fläche wird über den Normalenvektor $n$ beschrieben.

Wie man sich leicht verdeutlichen kann, ist die von der Lichtquelle an das Flächenelement abgegebene Strahlungsleistung $d\Phi_e$ nicht unmittelbar von dessen Flächeninhalt $dA$, son-

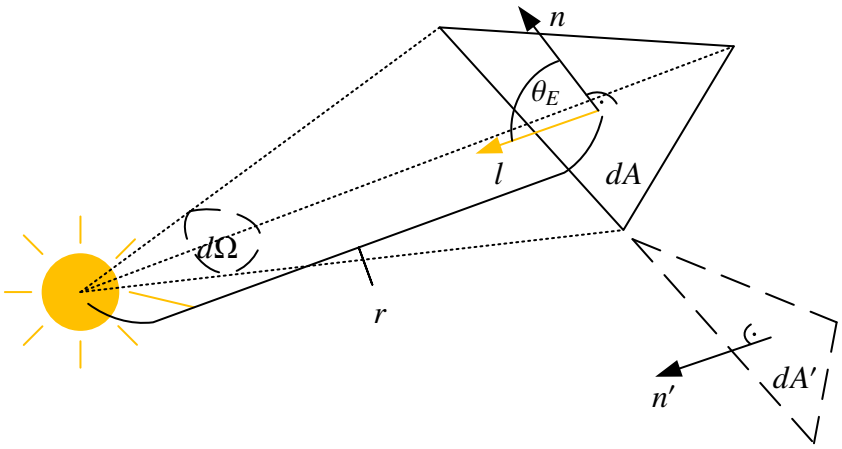


*Bild 2-2*: *Scheinbare Fläche und Raumwinkel eines ebenen Flächenelements.*

dern nur von der in Lichteinfallsrichtung projizierten Fläche $dA'$ abhängig. Diese ergibt sich aus $dA$ unter Berücksichtigung des Winkels $\theta_E$ zwischen $n$ und $l$:

$$dA' = dA \cdot \cos(\theta_E).$$

$dA'$ wird auch als scheinbare Fläche bezeichnet und wird in Bild 2-2 ebenfalls visualisiert. Ihr Normalenvektor $n'$ ist stets parallel zur Lichteinfallsrichtung $l$. Festzuhalten bleibt die Proportionalität zwischen der empfangenden Strahlungsleistung und der scheinbaren Fläche eines Flächenelements:

$$d\Phi_e \sim dA'. \tag{2-1}$$

Neben der scheinbaren Fläche stellt der Abstand $r$ zwischen Lichtquelle und Flächenelement eine weitere Einflussgröße für die vom Flächenelement empfangene Strahlungsleistung dar. Genau wie die Lichteinfallsrichtung ist auch $r$ unter den getroffenen Annahmen für alle Punkte der betrachteten Fläche näherungsweise gleich. Legt man gedanklich eine Kugel um die Lichtquelle, deren Zentrum mit der Punktlichtquelle zusammenfällt, so verteilt sich die Strahlungsleistung auf die einzelnen Flächenelemente der Kugelfläche. Unabhängig vom Radius dieser Kugel bleibt die Summe der Strahlungsleistungen auf allen Teilflächen konstant, da sie als eine Eigenschaft der Lichtquelle unabhängig von der Umgebungsgeometrie ist. Insbesondere lässt sich aus der Formel für die Kugeloberfläche

$$A_K = 4\pi r_K^2$$

ableiten, dass die auf ein Flächenelement fallende Strahlungsleistung umgekehrt proportional mit dem quadratischen Kugelradius $r_K$ abfällt. Wählt man den Kugelradius entsprechend der Distanz $r$ im Bild 2 – 2, wird deutlich, dass die von dem Flächenelement empfangene Strahlungsleistung außerdem folgende Proportionalität aufweist:

$$d\Phi_e \sim \frac{1}{r^2}. \tag{2-2}$$

Insgesamt hängt die empfangene Strahlungsleistung eines Flächenelements nach den Gleichungen (2-1) und (2-2) gemäß

$$d\Phi_e \sim \frac{dA'}{r^2}$$

von der Lage zur Lichtquelle ab. Dieser Quotient wird auch als Raumwinkel $d\Omega$ bezeichnet und nimmt für die folgenden Betrachtungen eine zentrale Rolle ein. Die Einheit des Raumwinkels ist der Steradiant (*sr*). Hierbei handelt es sich um eine dimensionslose Größe (Vgl. *rad*), die deshalb im Allgemeinen und auch in den folgenden Abschnitten dieser Arbeit nicht mitgeführt wird.

Bisher wurde vereinfacht davon ausgegangen, dass der Lichteinfallsvektor $l$ und der Abstand zur Lichtquelle $r$ für alle Punkte des Flächenelements $dA$ konstant sind. Exakt gilt diese Definition jedoch nur, wenn man infinitesimal kleine Flächenelemente betrachtet. Allgemein ergibt sich der Raumwinkel einer gegebenen Fläche $dA$ aus Sicht eines Be-

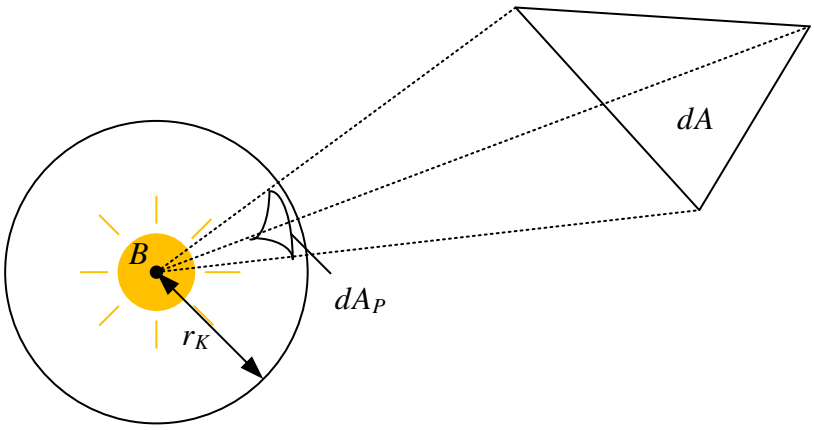


*Bild 2-3*: *Projektion eines Flächenelements auf eine Kugel.*

zugspunkts $B$ aus dem Verhältnis des Flächeninhalts der Projektion $dA_P$ dieser Fläche auf eine Kugel und dem quadratischen Kugelradius $r_K$, wobei $B$ im Kugelzentrum liegt. Das Bild 2-3 veranschaulicht diesen Zusammenhang. Es wird ebenfalls deutlich, dass die Ausdehnung der Kugel den Raumwinkel nicht beeinflusst, da sich die Projektionsfläche $dA_P$ proportional zum Quadrat des Kugelradius $r_K$ verhält. Der gesamte Raum um den Bezugspunkt (in den Bildern 2-2 und 2-3 entspricht dieser der Lichtquelle) umfasst

$$\Omega_{voll} = \frac{A_K}{r_K^2} = \frac{4\pi r_K^2}{r_K^2} = 4\pi sr.$$

Der Bezugspunkt kann anstelle der Punktlichtquelle dieses Beispiels auch als beleuchteter Punkt einer Oberfläche gewählt werden. Bei der Einführung der verschiedenen lichttechnischen Größen im Abschnitt 2.1.3 werden beide Ansätze Anwendung finden.

### 2.1.3 Strahlungsphysikalische Größen

In den folgenden Unterabschnitten werden strahlungsphysikalische Größen eingeführt, aus denen an späterer Stelle die für diese Arbeit relevanten lichttechnischen Größen abgeleitet

werden. Die Strahlungsphysik oder Radiometrie beschäftigt sich mit der Messung elektromagnetischer Strahlung im sichtbaren und nicht sichtbaren Bereich. Zur Unterscheidung von den lichttechnischen Größen, welche später eingeführt werden, erhalten diese Größen den im Kontext geläufigen Index $e$ (energetisch).

### Strahlungsfluss und -energie

Der Strahlungsfluss $\Phi_e$ (auch Strahlungsleistung) ist die differentielle Energiemenge $dE$, die im Zeitraum $dt$ von einer Strahlungsquelle abgegeben wird. Er trägt die Einheit Watt ($W$).

$$\Phi_e = \frac{dE}{dt} \qquad [\Phi_e] = W \tag{2-3}$$

Geht man zunächst von einer monochromatischen Strahlung aus, so besitzen alle elektromagnetischen Wellen die gleiche Wellenlänge $\lambda$ bzw. alle Photonen die gleiche Energiemenge $Q_{Ph} = h \cdot f$, wobei $h$ das Planck'sche Wirkungsquantum und $f$ die Strahlungsfrequenz bezeichnet. Dabei stehen Wellenlänge und Strahlungsfrequenz für ein gegebenes Ausbreitungsmedium über die Lichtgeschwindigkeit $c = \lambda \cdot f$ im direkten Zusammenhang. Der Strahlungsfluss kann in diesem Fall direkt über den Photonenstrom $\frac{dN_{Ph}}{dt}$ angegeben werden:

$$\Phi_e = h \cdot f \cdot \frac{dN_{Ph}}{dt}.$$

Für polychromatische Strahlung gilt grundsätzlich der gleiche Zusammenhang. Allerdings müssen die enthaltenen Frequenzen aufgrund ihrer verschiedenen Energiegehalte differenziert berücksichtigt werden. Dazu wird der Photonenstrom $\frac{dN_{Ph}(f)}{dt}$ frequenzabhängig. Insgesamt ergibt sich für den Strahlungsfluss

$$\Phi_e = h \cdot \int_{f=0}^{\infty} f \cdot \frac{dN_{Ph}(f)}{dt} df. \tag{2-4}$$

Integriert man den Strahlungsfluss über einen Zeitraum $\Delta T$, so erhält man die Strahlungsenergie $Q_e$, die von der Strahlungsquelle im Zeitintervall $\Delta T$ abgegeben wurde. Die Einheit ist Joule ($J$). Aus Gleichung (2-3) folgt für den monochromatischen Fall

$$Q_e = h \cdot f \cdot \int_{t=0}^{\Delta T} \frac{dN_{Ph}}{dt} dt \qquad [Q_e] = J$$

und für den polychromatischen Fall

$$Q_e = h \cdot \int_{t=0}^{\Delta T} \int_{f=0}^{\infty} f \cdot \frac{dN_{Ph}(f)}{dt} df dt.$$

### Strahl- und Bestrahlungsstärke

Ist eine emittierende Fläche einer Strahlungsquelle gegenüber dem Abstand zum Beobachter hinreichend klein, so kann die Strahlungsquelle als sogenannte Punktlichtquelle

approximiert werden, welche keine räumliche Ausdehnung besitzt. Die Quelle aller von ihr ausgesandten Strahlen fällt auf einen Punkt zusammen. Unter diesen Voraussetzungen kann die Strahlstärke $I_e$ eingeführt werden. Diese bezieht den Strahlungsfluss, der von einer Lichtquelle ausgesandt wird, auf den Raumwinkel. Dabei wird der Bezugspunkt des Raumwinkels auf die Position der Punktlichtquelle gesetzt.

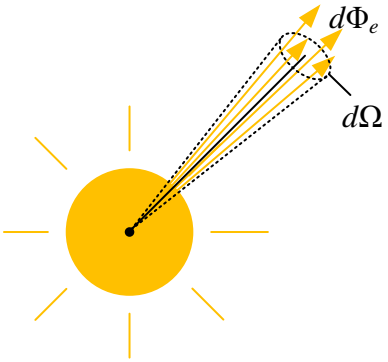


*Bild 2-4*: *Geometrische Zusammenhänge bei der Definition der Strahlstärke.*

$$I_e = \frac{d\Phi_e}{d\Omega} \qquad [I_e] = \frac{W}{sr}$$

Die Strahlstärke beschreibt also die richtungsabhängige Abstrahlcharakteristik der Lichtquelle. Anstelle einer Lichtquelle im engeren Sinne, kann natürlich auch eine extern bestrahlte Fläche, die einen Teil des empfangenen Lichts reflektiert, als solche aufgefasst werden.

Fällt ein Strahlungsfluss $\Phi_e$ auf eine Empfängerfläche $A_E$, so wird die flächenbezogene Dichte des Strahlungsflusses als Bestrahlungsstärke $E_e$ bezeichnet. Im Gegensatz zur Strahlstärke ist hier also nicht die Lichtquelle, sondern die Oberfläche eines bestrahlten Objekts Gegenstand der Betrachtung.

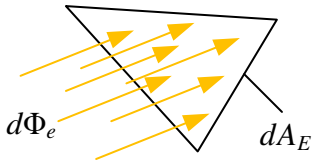


*Bild 2-5*: *Geometrische Zusammenhänge bei der Definition der Bestrahlungsstärke.*

$$E_e = \frac{d\Phi_e}{dA_E} \qquad [E_e] = \frac{W}{m^2} \tag{2-5}$$

Die Bestrahlungsstärke dient als Grundlage zur Berechnung des empfangenen Strahlungsflusses, indem man sie über die Oberfläche des betrachteten Objekts integriert.

### Strahldichte

Im Gegensatz zur Strahlstärke berücksichtigt die Strahldichte $L_e$ die Ausdehnung der Lichtquelle. Sie differenziert nicht nur zwischen den Raumwinkeln bzw. Abstrahlrichtungen $d\Omega$, sondern variiert auch abhängig vom betrachteten Punkt der Lichtquellenoberfläche $A_S$ (Senderfläche):

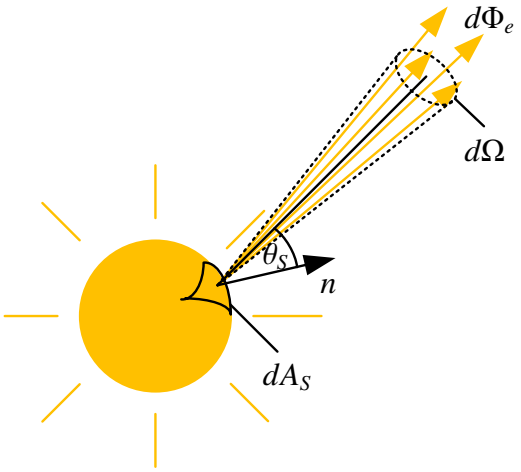


*Bild 2-6*: *Geometrische Zusammenhänge bei der Definition der Strahldichte.*

$$L_e = \frac{dI_e}{dA_S \cos\theta_S} = \frac{d^2\Phi_e}{d\Omega dA_S \cos\theta_S} \qquad [L_e] = \frac{W}{sr \cdot m^2}. \tag{2-6}$$

In Gleichung (2-6) entspricht die Größe $\theta_S$ dem Winkel zwischen der betrachteten Abstrahlrichtung und der Normalen $n$ des Flächenelements $dA_S$, sodass das Produkt $dA_S \cdot \cos\theta_S$ die scheinbare Senderfläche unter Berücksichtigung der Abstrahlrichtung beschreibt. $d^2\Phi_e$ beschreibt letztlich die Strahlungsleistung, die vom Flächenelement $dA_S$ der Strahlungsquelle in das Raumwinkelelement $d\Omega$ abgestrahlt wird.

### Spektralverteilung

Als Spektralverteilung (auch "Frequenzspektrum") einer Strahlung bezeichnet man die Gesamtheit der Frequenzen unterschiedlicher harmonischer Schwingungen, die im Lichtstrahl enthalten sind. Die Amplituden der jeweiligen Frequenzen können beispielsweise durch Fourier-Analyse aus dem zeitlichen Signalverlauf ermittelt werden. Das zu messende Signal $X_e$ entspricht dabei zwangsläufig direkt oder indirekt einer physikalischen Größe. Im Kontext von Strahlung kommen je nach betrachteter Situation meist die Strahlstärke $I_e$, die Bestrahlungsstärke $E_e$, die Strahldichte $L_e$ oder die Strahlungsleistung $\Phi_e$ in Betracht.

Als Spektralverteilung fassen wir nun stets die Dichtefunktion der jeweiligen Signalgröße $X_e$ bezüglich der Wellenlänge $\lambda$ auf und kennzeichnen diese durch den Index $\lambda$ an der Signalgröße:

$$X_{e,\lambda} = \frac{dX_e(\lambda)}{d\lambda} \qquad [X_{e,\lambda}] = \frac{[X_e]}{m}.$$

Die Spektralverteilung der Strahlungsleistung würde gemäß dieses Schemas mit $\Phi_{e,\lambda}$ bezeichnet werden und die Einheit $\frac{W}{m}$ tragen.

### Strahlungsübertragung

Nachdem die für diese Arbeit relevanten strahlungsphysikalischen Größen eingeführt sind, sollen nun ihre Zusammenhänge beleuchtet werden. Dazu wird das in Bild 2-7 skizzierte Szenario angenommen, welches eine Lichtquelle mit der Sendefläche $A_S$ und eine von ihr bestrahlte Empfängerfläche $A_E$ beinhaltet. Es stellt sich nun die Frage, welcher Strahlungsfluss von $A_S$ an $A_E$ übertragen wird und wie genau sich die örtliche Bestrahlungsstärkeverteilung auf $A_E$ darstellt. Zur Beantwortung dieser Frage kann man das

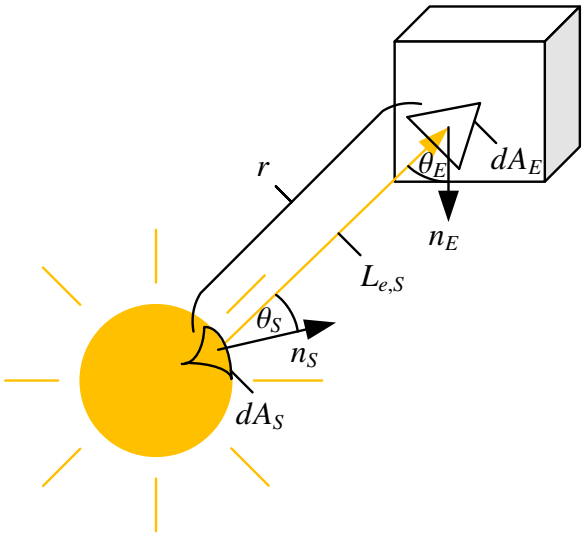


*Bild 2-7*: *Visualisierung der Größen im Grundgesetz der Strahlungsübertragung.*

photometrische Grundgesetz

$$d^2\Phi_{e,S\to E} = L_{e,S} \cdot \frac{dA_S \cos\theta_S \cdot dA_E \cos\theta_E}{r^2} \tag{2-7}$$

heranziehen, welches die von der Teilfläche $dA_S$ des Senders an die Teilfläche $dA_E$ des Empfängers übertragene Strahlungsleistung $d^2\Phi_{e,S\to E}$ beschreibt. Es besagt, dass sich diese proportional zu den scheinbaren Teilflächen verhält, welche von den Winkeln $\theta_S$ bzw. $\theta_E$ zwischen der jeweiligen Flächennormale und der Abstrahl- bzw. Einfallsrichtung abhängen. Weiterhin verhält sich $d^2\Phi_{e,S\to E}$ umgekehrt proportional zum quadratischen Abstand $r$ zwischen Sender und Empfänger, da der Raumwinkel, den die Empfängerfläche aus Sicht des Senders überdeckt, mit dem Quadrat des Abstands kleiner wird. Proportionalitätskonstante ist die Strahldichte $L_{e,S}$ des Flächenelements $dA_S$ in Richtung $dA_E$.

Diese Gesetzmäßigkeit kann zur Berechnung der insgesamt übertragenen Strahlungsleistung auf zweierlei Wegen genutzt werden. Einerseits kann die von $A_S$ auf $A_E$ abgestrahlte Strahlungsleistung bestimmt werden.

**Weg 1: Abstrahlung von $A_S$ auf $A_E$**
Aus Sicht des Senders bedeckt die Teilfläche $dA_E$ den Raumwinkel

$$d\Omega_S = dA_E \cdot \frac{\cos\theta_E}{r^2}. \tag{2-8}$$

Nach Gleichung (2-6) ist die von $dA_S$ nach $dA_E$ übertragene Strahlungsleistung

$$d^2\Phi_e = L_{e,S}\, dA_S\, d\Omega_S \cos\theta_S\,.$$

Um die insgesamt von $dA_S$ an den Empfänger abgegebene Strahlungsleistung $d\Phi_e$ zu erhalten, muss über alle Teilflächen $dA_E$ bzw. die nach Gleichung (2-8) zugehörigen Raumwinkelbereiche $d\Omega$ integriert werden:

$$d\Phi_e = dA_S \int_{d\Omega_S \in \Omega_S} L_{e,S} \cos\theta_S\, d\Omega_S\,.$$

Summiert man nun noch die Anteile der Strahlungsleistung über alle Teilflächen $dA_S$ des Senders zur Gesamtfläche $A_S$ auf, so erhält man die insgesamt vom Sender zum Empfänger übertragene Strahlungsleistung

$$\Phi_e = \int_{dA_S \in A_S} \int_{d\Omega_S \in \Omega_S} L_{e,S} \cos\theta_S\, d\Omega_S\, dA_S\,. \tag{2-9}$$

**Weg 2: Zustrahlung auf $A_E$ von $A_S$**
Alternativ kann das Szenario aus Sicht der bestrahlten Fläche $A_E$ betrachtet werden. Für diese hat die scheinbare Fläche des abstrahlenden Flächenelements $dA_S$ den Flächeninhalt $A_S \cdot \cos\theta_S$ und deckt damit aus Sicht des Empfängers den Raumwinkel

$$d\Omega_E = dA_S \cdot \frac{\cos\theta_S}{r^2} \tag{2-10}$$

ab. Wieder lässt sich mit Gleichung (2-6) die auf $dA_E$ auftreffende von $dA_S$ kommende Strahlungsleistung beschreiben:

$$d^2\Phi = L_{e,S} \cos\theta_E d\Omega_E.$$

Einmaliges Integrieren über die Elemente der Senderfläche bzw. die nach Gleichung (2-10) zugehörigen Raumwinkel $d\Omega$ führt auf die insgesamt von $dA_E$ empfangene Strahlungsleistung

$$d\Phi_e = dA_E \int_{d\Omega_E \in \Omega_E} L_{e,S} \cos\theta_E d\Omega_E.$$

Erneute Integration über die einzelnen Teilflächen des Empfängers führt letztlich auf die insgesamt übertragene Strahlungsleistung

$$\Phi_e = \int_{dA_E \in A_E} \int_{d\Omega_E \in \Omega_E} L_{e,S} \cos\theta_E d\Omega_E dA_E. \tag{2-11}$$

**Grundgesetz der Strahlungsübertragung**
Natürlich führen beide Wege zur gleichen Strahlungsleistung. Die Strukturen der Gleichungen (2-9) und (2-11) sind auch weitgehend identisch. Sie unterscheiden sich nur im

Betrachtungspunkt, der bei Weg 1 auf der Senderfläche und bei Weg 2 auf der Empfängerfläche liegt. Aus diesem Grund werden die Indizes meist weggelassen und es gilt für die von einer Fläche $A_E$ an eine Fläche $A_S$ übertragene Strahlungsleistung:

$$\Phi_e = \int_{dA \in A} \int_{d\Omega \in \Omega} L_{e,S} \cos\theta d\Omega dA. \tag{2-12}$$

Diese Gleichung ist das Grundgesetz der Strahlungsübertragung. Bei seiner Anwendung können die Größen $\theta$, $\Omega$ und $A$ wahlweise, aber konsequent, auf die Sender- oder die Empfängerfläche bezogen werden.

### Photometrisches Entfernungsgesetz

Gemäß der Definition der Bestrahlungsstärke nach Gleichung (2-5) und unter der Berücksichtigung des Grundgesetzes der Strahlungsübertragung (2-12) bzw. (2-11) ergibt sich für eine Empfängerfläche $A_E$ die Bestrahlungsstärke

$$E_e = \frac{d\Phi_e}{dA_E} = \int_{d\Omega_E \in \Omega_E} L_{e,S} \cos\theta_E d\Omega_E,$$

wenn sie durch einen Sender mit der Strahldichte $L_{e,S}$ bestrahlt wird. Für eine Punktlichtquelle kann dieser Term noch weiter reduziert werden. Dazu ersetzt man zunächst den Integranden $d\Omega_E$ gemäß Gleichung (2-10) und erhält

$$E_e = \int_{dA_S \in A_S} L_{e,S} \frac{\cos\theta_E \cos\theta_S}{r^2} dA_S.$$

Weiterhin kann die Strahldichte $L_{e,S}$ durch ihre Definition (2-6) ersetzt werden und es folgt

$$E_e = \int_{dA_S \in A_S} \frac{dI_{e,S}}{dA_S \cos\theta_S} \frac{\cos\theta_E \cos\theta_S}{r^2} dA_S.$$

Kürzen und umformen führt letztlich auf das photometrische Entfernungsgesetz

$$E_e = \frac{\cos\theta_E}{r^2} \frac{\int_{dA_S \in A_S} dI_{e,S} dA_S}{dA_S} = \frac{I_{e,S}}{r^2} \cos\theta_E, \tag{2-13}$$

dessen Zusammenhänge in Bild 2-8 visualisiert werden. Während die Strahlstärke eine Eigenschaft der Lichtquelle und unabhängig von der Distanz zwischen Sender und Empfänger ist, nimmt die Bestrahlungsstärke mit dem Quadrat des Abstands ab. Gleichzeitig geht hervor, dass nur die scheinbare Empfängerfläche zur empfangenen Bestrahlungsstärke beiträgt. Das photometrische Entfernungsgesetz gilt nur für ein differentielles Flächenelement auf der Empfängerfläche, da sowohl die ausgesandte Strahlstärke $I_{e,S}$ als auch der Winkel $\theta_E$ zwischen der Strahleinfallsrichtung und der Normalen der Empfängerfläche variieren können.

Reale Lichtquellen besitzen stets eine räumliche Ausdehnung. Insofern sind Berechnungen auf Basis des photometrischen Entfernungsgesetzes stets fehlerbehaftet. Man bezeichnet deshalb den Abstand zwischen Sender und Empfänger, in dem die Lichtquelle als Punktlichtquelle approximiert werden kann ohne einen nennenswerten Fehler einzubringen, als photometrische Grenzentfernung. Die DIN 13032 zur Vermessung von Lampen und Leuchten fordert in Teil 1 eine Mindestentfernung von dem fünf- bis zehnfachen des größten Durchmessers der leuchtenden Fläche zwischen Leuchte und Messapparatur [DIN02].

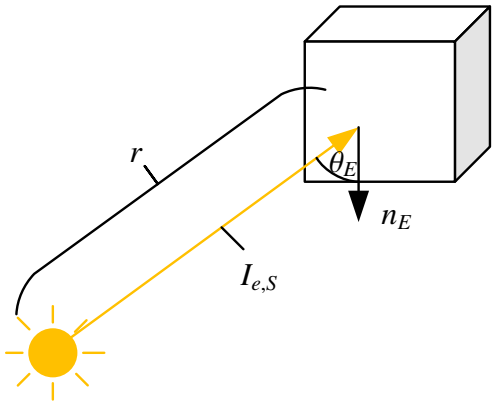


*Bild 2-8*: *Visualisierung der Größen im photometrischen Entfernungsgesetz.*

### 2.1.4 Photometrie

Alle bisher eingeführten Größen fußen auf physikalisch motivierten Basisgrößen, indem sie direkt mit dem Energiegehalt und der Anzahl existierender Photonen in Beziehung stehen. Das menschliche Auge kann diese Größen jedoch nicht unmittelbar messen, wodurch sie nicht als Beschreibung der visuellen Eindrücke durch den Meschen dienen. Die Photometrie (auch "Lichttechnische Optik") schafft hier Abhilfe, indem sie die strahlungsphysikalischen Größen durch eine Bewertungsfunktion in photometrische Größen überführt. Diese wiederum bemessen die Strahlung anhand ihres wahrnehmbaren Anteils – dem Licht.

#### Photometrischer Normalbeobachter

Ein wesentliches Problem bei der Bewertung der strahlungsphysikalischen Größen ist die Subjektivität der menschlichen Wahrnehmung. Krankheiten wie Farbenblindheit, Rot-Grün-Schwäche, Grüner Star oder Nachtblindheit stellen Extrembeispiele für die unterschiedlichen Wahrnehmungen dar. Aber auch unter normal sehenden Personen gibt es Abweichungen in der Empfindung von Helligkeit und Farbe. Folglich muss zur Herleitung einer Bewertungsfunktion zunächst ein möglichst repräsentativer Beobachter definiert werden, der eine Art Mittelwert aus der Menge normal sehender Menschen darstellt.

Die Internationale Beleuchtungskommission (CIE) hat deshalb im Jahre 1924 als Resultat umfangreicher Untersuchungen eine spektrale Hellempfindlichkeitsfunktion für normalsichtige Beobachter festgelegt. Diese als $V(\lambda)$-Funktion notierte wellenlängenabhängige Gewichtung von Strahlungsanteilen definiert den photometrischen Normalbeobachter [CIE19].

#### Hellempfindlichkeitsfunktionen

Die lichttechnischen Größen werden aus den strahlungsphysikalischen Größen durch spektrale Gewichtung gewonnen. Die wellenlängenabhängigen Gewichtungsfaktoren werden

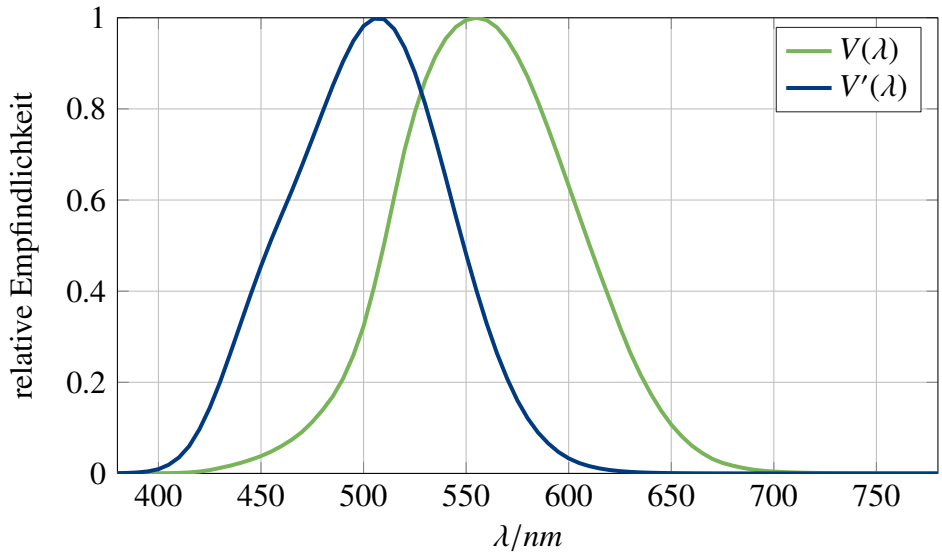


*Bild 2-9*: *Hellempfindlichkeitsfunktionen für Tages ($V(\lambda)$)- und Nachtsehen ($V'(\lambda)$) von 380 nm bis 780 nm in 5 nm Schritten nach DIN 5031 Teil 3. [DIN01]*

dabei durch die sogenannten Hellempfindlichkeitsfunktionen beschrieben. Für eine gegebene Hellempfindlichkeitsfunktion $V(\lambda)$ und eine strahlungsphysikalische Größe $X_e(\lambda)$ kann die zugehörige photometrische Größe $X_v$ (Index $v$ für "visuell") wie folgt ermittelt werden:

$$X_v = K_m \int_{\lambda=\underline{\lambda}}^{\bar{\lambda}} X_e(\lambda) \cdot V(\lambda) d\lambda. \tag{2-14}$$

Dabei entspricht das Intervall $[\underline{\lambda}; \bar{\lambda}]$ dem sichtbaren Wellenlängenbereich des Spektrums, z.B. 380 *nm* bis 780 *nm*. Die Konstante $K_m$ verknüpft das radiometrische und das photometrische Basissystem und trägt die Einheit $[K_m] = \frac{lm}{W}$. Der in Lumen (*lm*) gemessene Lichtstrom als Äquivalent zur Strahlungsleistung wird nachfolgend eingeführt. Durch die Wahl von $K_m$ wird die Hellempfindlichkeitsfunktion $V(\lambda)$ so normiert, dass ihr Wertebereich dem Intervall [0; 1] entspricht.

Ursprünglich wurde die $V(\lambda)$-Funktion durch die CIE von 380 *nm* bis 750 *nm* in 10 *nm*-Schritten vermessen. Die DIN 5031 definiert im Teil 3 die spektrale Empfindlichkeit $V(\lambda)$ durch Inter- und Extrapolation in einem Bereich von 380 *nm* bis 780 *nm* in 1 *nm*-Schritten [DIN01]. Bild 2-9 visualisiert durch die grüne Kurve die dort notierten Datenpunkte. Ihr Maximum liegt bei einer Wellenlänge von 555 *nm*, welches einem monochromatischen Licht mit der Farbe Grün entspricht. Eine Aussage der Hellempfindlichkeitsfunktion ist zum Beispiel, dass grünes Licht (555 *nm*) bei einer nur halb so hohen strahlungsphysikalischen Strahldichte die gleiche photometrische Leuchtdichte bzw. den selben Helligkeitseindruck wie blaues Licht (510 *nm*) erzeugt.

Bedingt durch den Aufbau des menschlichen Auges, welches in dunklen und hellen Umgebungen unterschiedliche Rezeptoren nutzt (Zäpfchen und Stäbchen), gilt die in Bild 2-9 beschriebene $V(\lambda)$-Kurve ausschließlich für das photopische Sehen (reines Tagsehen). Die mit $V'(\lambda)$ bezeichnete Kurve beschreibt die Hellempfindlichkeit des photometrischen Normalbeobachters beim skotopischen Sehen (Nachtsehen), deren Datengrundlage ebenfalls

aus der DIN 5031 Teil 3 stammt [DIN01]. Wie aus Bild 2-9 hervorgeht, verschiebt sich das Maximum für ein dunkeladaptiertes Auge in den kurzwelligen blauen Bereich. Die Werte des Faktors $K_m$ aus Gleichung 2-14 sind nach DIN 5031 Teil 3 für das photopische Sehen $K_m = 683\,lm/W$ und für das skotopische Sehen $K'_m = 1699\,lm/W$.

### Die Candela

Die Candela ($cd$) ist neben Meter ($m$), Kilogramm ($kg$), Sekunde ($s$), Ampere ($A$), Kelvin ($K$) und Mol ($mol$) eine der sieben Basiseinheiten des internationalen Einheitensystems (SI) und international standardisiert [DIN03]. Sie misst die Lichtstärke $I_v$ (Äquivalent zur strahlungsphysikalischen Strahlstärke) und ist so normiert, dass eine monochromatische Lichtquelle der Frequenz $540\,THz$ mit einer Strahlstärke von $\frac{1}{683}\,\frac{W}{sr}$ die Lichtstärke $1\,cd$ besitzt. Eine Strahlung dieser Frequenz hat in Luft eine Wellenlänge von $\lambda_{cd} = 555nm$ und wurde gewählt, da der zugehörige $V(\lambda)$-Wert 1 ist. Deshalb entspricht die Strahlstärke von $\frac{1}{683}\,\frac{W}{sr}$ auch dem Kehrwert des Faktors $K_m$ aus Gleichung 2-14, welcher strahlungsphysikalische und photometrische Größen in Beziehung setzt. Wendet man diese Gleichung auf die beschriebene Situation an, erhält man

$$I_v = K_m \int_{\lambda=\underline{\lambda}}^{\bar{\lambda}} I_e(\lambda) \cdot V(\lambda) d\lambda = K_m \cdot I_e(\lambda_{cd}) \cdot V(\lambda_{cd}) = 683\,\frac{lm}{W} \cdot \frac{1}{683}\,\frac{W}{sr} \cdot 1 = 1\,\frac{lm}{sr} = 1\,cd.$$

Die Wahl der Lichtstärke als photometrische Basisgröße überrascht aus heutiger Sicht, da man den Lichtstrom $\Phi_v$ als fundamentalere Größe ansehen würde (s. Tab. 2-1). In den Anfängen der Photometrie stand jedoch der rein visuelle Vergleich von Lichtquellen im Vordergrund, sodass die Lichtstärke die einzige der Lichtquelle zurechenbare Eigenschaft war, die durch das Auge unter der Voraussetzung eines einheitlichen Abstands wahrgenommen werden konnte. Daher wurde die Lichtstärke $I_v$ als fundamentale photometrische Größe eingeführt [BS75]. Bei der Zerlegung der nachfolgend eingeführten photometrischen Größen in ihre Basiseinheiten, findet sich die Candela stets wieder.

### Photometrische Größen

Im Abschnitt 2.1.3 werden die strahlungsphysikalischen Größen Strahlungsfluss, Strahlungsenergie, Strahlstärke, Bestrahlungsstärke und Strahldichte eingeführt. Außerdem beschreibt Abschnitt 2.1.4 die Hellempfindlichkeit des menschlichen Auges und zeigt, wie mit Gleichung (2-14) strahlungsphysikalische in photometrische Größen überführt werden können. In diesem Abschnitt werden nun alle in dieser Arbeit relevanten photometrischen Größen vorgestellt und mit ihren strahlungsphysikalischen Äquivalenten verknüpft. Die Tabelle 2-1 leistet dieses in übersichtlicher Form.

In den weiteren Ausführungen wird nur noch auf die photometrischen Größen Bezug genommen, da im betrachteten Kontext die Wahrnehmung des Lichts durch den Menschen im Fokus steht. Für sie gelten die bereits vorgestellten Beziehungen, wie das Grundgesetz der Strahlungsübertragung (auch: "Photometrisches Grundgesetz") und das photometrische Entfernungsgesetz, auf gleiche Weise. Es genügt, die Indizes $e$ durch $v$ zu ersetzen. Grundsätzlich ist es möglich, eine beliebige Beleuchtungssituation zuerst auf strahlungsphysikalischer Ebene zu beschreiben und anschließend die Ergebnisse mit Gleichung (2-14) in die Photometrie zu überführen.

*Tabelle 2-1*: *Gegenüberstellung photometrischer und strahlungsphysikalischer Größen.*

| Photometrie | | Radiometrie | |
|---|---|---|---|
| Größe | Einheit | Größe | Einheit |
| Lichtmenge $Q_v$ | $lm \cdot s$ | Strahlungsenergie $Q_e$ | $J$ |
| Lichtstrom $\Phi_v$ | $lm = cd \cdot sr$ | Strahlungsleistung $\Phi_e$ | $W$ |
| Lichtstärke $I_v$ | $cd$ | Strahlstärke $I_e$ | $\frac{W}{sr}$ |
| Beleuchtungsstärke $E_v$ | $lx = \frac{lm}{m^2}$ | Bestrahlungsstärke $E_e$ | $\frac{W}{m^2}$ |
| Leuchtdichte $L_v$ | $\frac{cd}{m^2}$ | Strahldichte $L_e$ | $\frac{W}{sr \cdot m^2}$ |

Von den aufgeführten Größen hat die Leuchtdichte unmittelbaren Bezug zur optischen Sinneswahrnehmung. Der Beobachter nimmt die Leuchtdichten der ihn umgebenden Flächen als deren Flächenhelligkeit wahr. Das menschliche Auge ist ausgesprochen anpassungsfähig und kann Leuchtdichten über viele Größenordnungen sensieren. Die Tabelle 2-2 gliedert die verschiedenen Adaptionsniveaus des Auges bezüglich der vorliegenden Leuchtdichten auf.

*Tabelle 2-2*: *Wahrnehmbarer Leuchtdichtebereich und Zuordnung der Adaptionsbereiche [Wik20].*

| Adaption | Bemerkung | Leuchtdichte $\left[\frac{cd}{m^2}\right]$ |
|---|---|---|
| Sehschwelle | | ca. $3 \cdot 10^{-6}$ |
| skotopisches Sehen | Nachtsehen | $3 \cdot 10^{-6}$ bis $3..30 \cdot 10^{-3}$ |
| mesopisches Sehen | Kombination | $3..30 \cdot 10^{-3}$ bis 3..30 |
| photopisches Sehen | Tagsehen | über 3..30 |
| Zapfensättigung | Blendung | ab $10^5..10^6$ |

Um eine Einschätzung der Größenordnungen zu ermöglichen, zeigt Tabelle 2-3 beispielhaft die Leuchtdichten einiger natürlicher und künstlicher Lichtquellen aus dem alltäglichen Leben.

Aus den angeführten Beispielen geht das hohe Adaptionsvermögen des Auges hervor. Zwischen Sehschwelle und Blendung liegen ca. zwölf Größenordnungen. Gleichzeitig wird deutlich, dass die Wahrnehmung einer Helligkeit extrem kontextabhängig ist. So muss zur immersiven Simulation einer Nachtfahrt die räumliche Umgebung stark abgedunkelt werden. Außerdem geht aus den Leuchtdichteangaben für schwarze und weiße Ausgaben eines LCD-Monitors hervor, dass die heutigen Ausgabegeräte nicht ansatzweise die reale Dynamik einer Nachtfahrt von absoluter Dunkelheit in der Ferne oder im Seitenbereich bis hin zur Blendung durch Gegenverkehr abbilden können. Es kann in der Simulation deshalb nur das Ziel sein, die Relationen der Leuchtdichtewerte möglichst unverändert zu erhalten.

*Tabelle 2-3*: *Leuchtdichten natürlicher und künstlicher Lichtquellen aus dem Alltag [Wik20].*

| Beispiel | Leuchtdichte $\left[\frac{cd}{m^2}\right]$ |
|---|---|
| bewölkter Nachthimmel | $10^{-6}$ bis $10^{-4}$ |
| sternklarer Nachthimmel | $0,001$ |
| Oberfläche des Mondes | 2500 |
| bedeckter Himmel | 2000 |
| klarer Himmel | 8000 |
| Sonnenscheibe am Mittag | $1,6 \cdot 10^9$ |
| Schwarz auf LCD-Monitor | $0,15..0,8$ |
| Weiß auf LCD-Monitor | 150..500 |
| HDR-Monitor | 600..1000 |
| Draht einer Halogenlampe | $20..30 \cdot 10^6$ |
| weiße LED | $50 \cdot 10^6$ |

### 2.1.5 Reflexion

Bisher lag der Fokus auf der direkten Wechselwirkung zwischen Lichtquelle und einer beleuchteten Fläche. Neben dem direkten Licht spielt jedoch auch das indirekte Licht eine wesentliche Rolle für die visuelle Wahrnehmung. Deshalb wird in diesem Abschnitt eine Beschreibung von Reflexionseigenschaften verschiedener Materialien eingeführt und durch das Beispiel des Lambertschen Reflektors angewendet.

#### Bidirektionale Reflektanzverteilungsfunktion

Eine zentrale Rolle bei der Berechnung von Lichteffekten in einer gegebenen Szene spielt die Wechselwirkung der Lichtstrahlen mit den verschiedenen Oberflächen bzw. den Materialien, aus welchen sie bestehen. In einen mathematischen Kontext können diese Wechselwirkungen durch die sogenannten bidirektionalen Streulichtverteilungsfunktionen (BSDF) gesetzt werden, welche 1977 durch Nicodemus et. al. eingeführt wurden [NRH+77]. Diese Verteilungsfunktionen, welche die allgemeinste Variante von BxDF-Funktionen darstellen, berücksichtigen sowohl Reflexion als auch Transmission. Meist werden diese beiden Effekte jedoch durch verschiedene Verteilungsfunktionen beschrieben. Die Beschreibung von Reflexion erfolgt dann durch bidirektionale Reflektanzfunktionen (BRDF), während Transmission durch bidirektionale Transmissionsfunktionen (BTDF) beschrieben wird. Letztere werden hier nicht näher thematisiert, da sie nur für transparente Objekte von besonderer Bedeutung sind. Eine BRDF beschreibt das Reflexionsverhalten von Oberflächen eines Materials unter einem gegebenem Einfallswinkel eines Lichtstrahls. Dazu ordnet sie jedem Paar aus Ein- und Ausfallwinkel das Verhältnis $f_r$ der Strahldichte $L_e$ des austretenden Lichts und der Bestrahlungsstärke $E_e$ des auftreffenden Lichts zu (Vgl. Bild 2-10).

$$f_r(\theta_E, \varphi_E, \theta_S, \varphi_S) = \frac{L_e(\theta_S, \varphi_S)}{E_e(\theta_E, \varphi_E)} \tag{2-15}$$

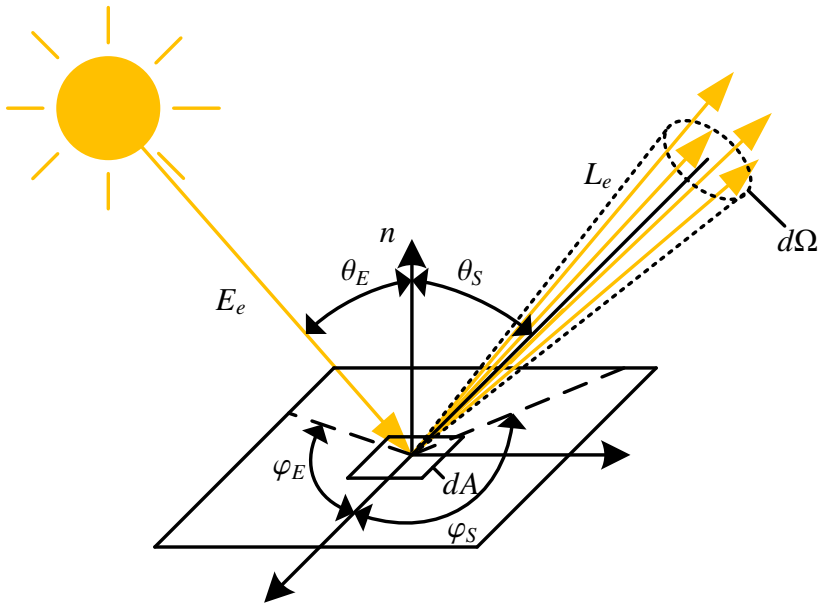


*Bild 2-10*: *Geometrische Zusammenhänge einer bidirektionalen Reflektanzfunktion (BRDF).*

Auch hierbei werden schon einige wesentliche Vereinfachungen getroffen. Neben der dargestellten BRDF gibt es auch allgemeinere Varianten, wie der räumlich variierenden bidirektionalen Reflektanzverteilungsfunktion (SVBRDF). Ihr Definitionsraum wird um zwei Dimensionen erhöht, indem sie die Koordinaten auf der Objektoberfläche berücksichtigt. Dadurch ist sie in der Lage, örtlich variierendes Reflexionsverhalten abzubilden. Eine weitere Variante stellen die bidirektionalen Oberflächenstreuung-Reflektanzverteilungsfunktionen (BSSRDF) dar. In dem Fall ist der Definitionsbereich sogar 8-dimensional, da er zwei Oberflächenpunkte enthält. Nötig ist diese Erweiterung, da Transmissionseffekte berücksichtigt werden, die dafür sorgen, dass der eintreffende Lichtstrahl die Oberfläche des Materials durchdringt, darin reflektiert wird und es an einer anderen Stelle wieder verlässt. Schließlich weist das Reflexionsverhalten verschiedener Materialien auch deutliche Abhängigkeiten von der Wellenlänge auf. Um sie zu berücksichtigen, muss eine weitere Dimension eingeführt werden. Erst so gelingt es, die Farbigkeit von Objektoberflächen geeignet zu modellieren. Alle genannten Erweiterungen sollen hier nicht weiter diskutiert werden, da in der Computergrafik typischerweise stark vereinfachte Modelle Anwendung finden.

Die Funktionswerte einer BRDF können einerseits durch die Speicherung geordneter Messwerte bzw. Simulationsergebnisse oder mittels Approximation durch analytische Funktionen gewonnen werden. Grundsätzlich weist eine BRDF folgende Eigenschaften auf:

- Positivität: $f_r(\theta_E, \varphi_E, \theta_S, \varphi_S) \geq 0 \quad \forall \theta_E, \varphi_E, \theta_S, \varphi_S$
- Helmholtz-Reziprozität: $f_r(\theta_E, \varphi_E, \theta_S, \varphi_S) = f_r(\theta_S, \varphi_S, \theta_E, \varphi_E)$

- Energieerhaltung: $\int_{\Omega_S \in \Omega^+} f_r(\theta_E, \varphi_E, \theta_S, \varphi_S) \cos\theta_S \, \mathrm{d}\Omega_S \leq 1 \forall \Omega_E \in \Omega^+$

Die in der Computergrafik Anwendung findenden BRDF werden typischerweise durch analytische Funktionen abgebildet, sodass verschiedene Materialien durch die Anpassung weniger Parameter modelliert werden können. Außerdem ist die recheneffiziente Lösbarkeit ein wichtiges Kriterium.

### Lambertsches Gesetz

Das Lambertsche Kosinusgesetz bildet den Extremfall des Reflexionsverhaltens von Körpern ab, welche bedingt durch wiederholte mikroskopische Reflexionen innerhalb ihrer Oberflächenstruktur (Streuzentren) das Licht in alle Richtungen streuen und auf diese Weise eine konstante Strahldichte erzeugen. Für die Reflexionscharakteristik spielt es in diesem Fall keine Rolle, aus welcher Richtung die Fläche beleuchtet wird. In der Realität reflektiert kein Material exakt nach dem Lambertschen Gesetz. Es existiert stets eine Abhängigkeit von der Beleuchtungsrichtung. Insbesondere für große Winkel zwischen Flächennormale und Beleuchtungsrichtung werden Abweichungen von der Gesetzmäßigkeit deutlich. Es gibt aber Materialien, die für das menschliche Auge fast unabhängig vom Betrachtungswinkel gleich hell wirken. Mattes Papier oder Milchglas sind Beispiele dafür. Da die vom Auge wahrgenommene Helligkeit der Leuchtdichte entspricht, weist das reflektierte Licht eines Lambertschen Flächenelements $dA$ eine konstante Leuchtdichte $L_v$ bzw. Strahldichte $L_e = const$ auf. Somit gilt nach Gleichung (2-6) für die reflektierte Strahlstärke $dI_e$ dieses Flächenelements

$$dI_e(\theta_S) = dA \cos\theta_S L_e = I_{e,max} \cos\theta_S,$$

wobei $\theta_S$ gemäß Abbildung 2-6 den Winkel zwischen der Flächennormalen und der Reflexionsrichtung beschreibt. Bild 2-11 veranschaulicht grafisch die von einem Lambertschen Flächenelement in verschiedene Raumrichtungen ausgesandte Lichtstärke.

Ein Lambertsches Flächenelement reflektiert die einfallende Strahlung vollständig. Es treten weder Absorption, noch Diffusion auf. Aus diesem Grund steht die reflektierte Strahldichte $L_e$ in einem proportionalen Zusammenhang mit der Bestrahlungsstärke $E_e$. Basierend auf der Energieerhaltung kann man aus den Gleichungen (2-5) und (2-6) folgenden Ansatz gewinnen:

$$\begin{aligned}
\Phi_{e,E} &= \Phi_{e,S} \\
E_e \cdot dA &= \int_{\Omega_S \in \Omega^+} L_e \cos\theta_S \, d\Omega_S \cdot dA \\
E_e &= \int_{\Omega_S \in \Omega^+} L_e \cos\theta_S \, d\Omega_S \\
&= \int_{\varphi_S=0}^{2\pi} \int_{\theta_S=0}^{\frac{\pi}{2}} L_e \cos\theta_S \sin\theta_S \, d\theta_S \, d\theta_S \\
&= \pi L_e.
\end{aligned}$$

Die BRDF eines Lambertschen Flächenelements hat somit gemäß Gleichung (2-15) die besonders einfache Form

$$f_r = \frac{L_e}{E_e} = \frac{1}{\pi}. \tag{2-16}$$

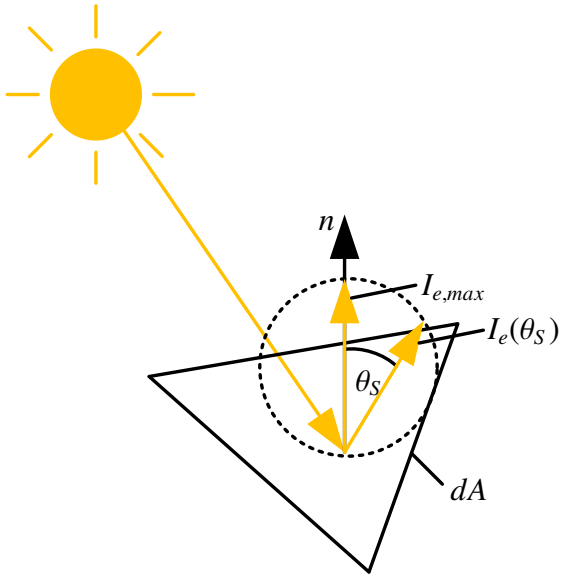


*Bild 2-11*: *Strahlstärkeverteilung der reflektierten Strahlung eines Lambertschen Flächenelements.*

## 2.2 Farbmetrik

Die bisher vorgestellten photometrischen Größen beschreiben die physikalische Strahlung ausschließlich bezüglich ihres Helligkeitseindrucks. Dieser wird durch die Bewertung der Strahlungsenergie mit der spektralen Hellempfindlichkeitsfunktion des menschlichen Auges, genauer des photometrischen Normalbeobachters, quantifiziert. Die Farbmetrik hingegen befasst sich mit der quantitativen Beschreibung von Farbreizen und differenziert die wahrgenommene Strahlung somit noch feiner. Entsprechend der drei Farbrezeptoren im menschlichen Auge, bedarf es drei Koordinaten zur eindeutigen Identifizierung eines Farbreizes. Das Farbsehen wird im Auge durch Zapfen geleistet. Man unterscheidet L (long)-, M (middle)- und S (short)-Zapfen entsprechend der Lage ihrer Empfindlichkeitsmaxima im Frequenzspektrum des sichtbaren Lichts. Die Farbwahrnehmung im menschlichen Auge wird als Farbvalenz bezeichnet. Eine Farbvalenz kann nicht eindeutig auf das Spektrum eines Lichtstrahls zurückgeführt werden. Stattdessen können verschiedene spektrale Zusammensetzungen zur gleichen Farbvalenz führen, solange sie die verschiedenen Rezeptoren jeweils in gleichem Maße anregen.

Die folgenden Abschnitte sollen ein grundlegendes Verständnis der Farbmetrik vermitteln und fokussieren dabei vor allem diejenigen Aspekte, die im weiteren Verlauf von Bedeutung sind. Umfängliche Darstellungen der Farbwissenschaft finden sich in [Ric81] und [WS82], welche auch die Grundlage für die folgenden Unterabschnitte bilden. Die themenbezogenen Normen finden sich in der DIN 5033 [DIN04].

### 2.2.1 Farbräume

Farben lassen sich trotz ihres rein physiologischen Ursprungs formal und mit mathematischer Exaktheit beschreiben. Genauer können die verschiedenen Farbvalenzen als Vektoren in einem Vektorraum, der im vorliegenden Kontext als Farbraum bezeichnet wird, aufgefasst werden. Das Bild 2-12 setzt die mathematischen Begriffe zur Beschreibung eines Vektors mit den korrespondierenden Farbeigenschaften in Beziehung.

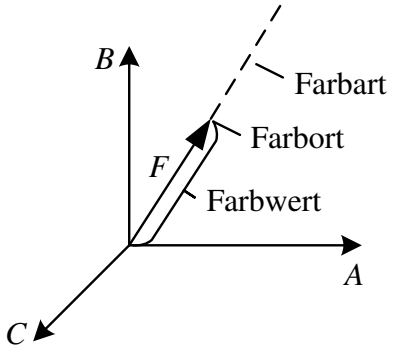


*Bild 2-12*: *Farbvalenz als Vektor in einem Farbraum.*

Die Koordinaten $A$, $B$ und $C$ des Vektors $F$ bezüglich der hier beispielhaft gewählten Basisvektoren $A_P$, $B_P$ und $C_P$ bestimmen den Farbort. Die Länge von $F$ heißt Farbwert. Die Richtung wird als Farbart bezeichnet und die Basisvektoren von Farbräumen werden Primärvalenzen genannt. Linear abhängige Vektoren gehören zur selben Farbart. Sie unterscheiden sich nur bezüglich ihrer Helligkeit. Eine Farbvalenz $F$ ist in diesem Sinne stets eine Linearkombination (auch additive Mischung) der Primärvalenzen $A_P$, $B_P$ und $C_P$:

$$F = A \cdot A_P + B \cdot B_P + C \cdot C_P. \tag{2-17}$$

Dabei sind die Koeffizienten $A$, $B$ und $C$ die benötigten Skalierungen und somit die Farbwerte der Primärvalenzen zur Mischung von $F$. Wie aufgrund der drei Koordinaten zu erwarten ist, beschreibt die Farbvalenz die Reize eines Lichtstrahls im menschlichen Auge genauer als es die Photometrie tut. Aus diesem Grund kann aus der Farbvalenz auf die photometrischen Größen geschlossen werden. Der umgekehrte Weg ist nicht möglich. Um eine photometrische Größe anhand der Farbwerte bestimmen zu können, müssen die verwendeten Primärvalenzen bezüglich dieser Größe vermessen sein. Hier kann eine beliebige der im Abschnitt 2.1.4 vorgestellten photometrischen Größen eingesetzt werden. Beispielhaft gehen wir davon aus, dass die Leuchtdichten $L_A$, $L_B$ und $L_C$ der Primärvalenzen bekannt sind. Diese werden auch als Leuchtdichtebeiwerte bezeichnet. Für die Leuchtdichte $L_F$ der Farbvalenz $F$ gilt dann nach dem Abneyschen Gesetz [Ric81]:

$$L_F = A \cdot L_A + B \cdot L_B + C \cdot L_C. \tag{2-18}$$

### 2.2.2 Farbwertanteile

Farbräume stellen eine vollständige Beschreibung der möglichen Farben und ihrer Abhängigkeiten dar. Da Farbräume als dreidimensionales Gebilde jedoch schlecht visualisiert werden können und zudem in vielen Fällen nur die Farbarten, nicht aber die Helligkeiten der Farbvalenzen von Interesse sind, wird anstelle des dreidimensionalen Farbraums oftmals die zugehörige zweidimensionale Farbtafel betrachtet. Dabei wird auf die Helligkeitsinformation durch Normierung auf die Summe der Farbwerte aller Primärvalenzen verzichtet. Man definiert die normierten Größen $a$, $b$ und $c$ der Farbwerte $A$, $B$ und $C$ gemäß

$$a = \frac{A}{A+B+C}, \quad b = \frac{B}{A+B+C} \quad \text{und} \quad c = \frac{C}{A+B+C} \tag{2-19}$$

und bezeichnet sie sinngemäß als Farbwertanteile. Bedingt durch die Definition der Spektralwerte genügt es, zwei der drei Werte in einer sogenannten Farbtafel aufzutragen, da sich der dritte stets ergibt ($a+b+c=1$). Die Normierung der Farbwerte gemäß Gleichung (2-19) kann als Querschnitt des Farbraums in der Ebene $A+B+C=1$ verstanden werden. Alle Farbvalenzen der gleichen Farbart schneiden die Ebene im selben Punkt. Erstellt man beispielsweise ein $(a,b)$-Diagramm, so erhält man eine zweidimensionale Darstellung aller Farbarten, welche als Farbtafel bezeichnet wird. Das Bild 2-16 zeigt beispielsweise die Farbtafel des später eingeführten Normfarbraums CIE-XYZ. Sie wird bezüglich der Farbwertanteile $x$ und $y$ der Primärvalenzen $X$ und $Y$ aufgetragen.

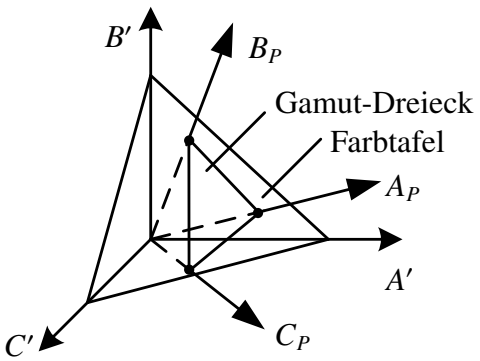


*Bild 2-13*: *Darstellung des Gamut-Dreiecks der Primärvalenzen $A_P$, $B_P$ und $C_P$ in einem anderen Farbraum mit den Primärvalenzen $A'_P$, $B'_P$ und $C'_P$.*

Sind zwei Farbvalenzen bzw. ihre Punkte auf der Farbtafel gegeben, so befinden sich alle additiven Mischungen aus ihnen (Linearkombinationen mit positiven Koeffizienten) auf der direkten Verbindungslinie dieser Punkte innerhalb der Farbtafel. Ausgehend von den drei Primärvalenzen $A_P$, $B_P$ und $C_P$ eines Farbraums befinden sich alle Farben, welche durch ihre additive Mischung entstehen, innerhalb eines Dreiecks, dessen Eckpunkte durch die Farbwertanteile der Primärvalenzen definiert sind. Das in Bild 2-13 durch die Primärvalenzen aufgespannte Dreieck wird auch Gamut-Dreieck genannt. Bild 2-16 zeigt

Gamut-Dreiecke etablierter Farbräume bezogen auf den genormten CIE-XYZ-Farbraum in der Farbtafeldarstellung. Farbvalenzen außerhalb des Dreiecks sind physikalisch nicht unter den gegebenen Primärvalenzen darstellbar, da die Intensität des Lichts nicht negativ werden kann. Mathematisch können sie sehr wohl formuliert werden. Es sind dazu Linearkombinationen mit negativen Koeffizienten notwendig. Entsprechend haben Farbvalenzen außerhalb des Gamut-Dreiecks negative Farbwerte und heißen äußere Mischungen (im Ggs. zu inneren Mischungen). Die negativen Farbwerte sind jedoch keine Eigenschaft der Farbe sondern des Farbraums. Abhängig von den gewählten Primärvalenzen können Farben von inneren zu äußeren Mischungen werden und umgekehrt.

### 2.2.3 Spektralwerte

Nachdem das mathematische Grundgerüst zum Umgang mit Farbvalenzen vorgestellt ist, soll nun die Brücke von der physikalischen Beschreibung eines Lichtstrahls hin zur wahrgenommenen Farbvalenz geschlagen werden. Man betrachtet dazu eine allgemeine polychromatische Strahlung mit der Sepktralverteilung $\Phi_{e,\lambda}$. Die gesamte Strahlungsleistung $\Phi_e$ dieses Lichtstrahls kann in näherungsweise monochromatische Komponenten $\Phi_{e,\lambda_i}$, welche jeweils in einem kleinen Wellenlängenband $\Delta\lambda$ strahlen, zerlegt werden. Die durch diesen Lichtstrahl hervorgerufene Farbvalenz $F$ kann nun ebenso additiv aus den Farbvalenzen der monochromatischen Komponenten gemäß

$$F = \sum_{i=0}^{i_{max}} F_{\lambda_i} \cdot \Delta\lambda \text{ mit } \lambda_i = \underline{\lambda} + i \cdot \Delta\lambda \text{ und } i_{max} = \lfloor \frac{\bar{\lambda} - \underline{\lambda}}{\Delta\lambda} \rfloor \tag{2-20}$$

gebildet werden, wobei $F_{\lambda_i}$ die Farbvalenz des monochromatischen Lichtanteils $\Phi_{e,\lambda_i}$, welches das Wellenlängenband von $\lambda_i$ bis $\lambda_i + \Delta\lambda$ umfasst, bezeichnet.

Außerdem betrachten wir das sogenannte energiegleiche Spektrum $\Phi_{EE,e,\lambda} = const$, welches für alle Wellenlängen mit der gleichen Leistung strahlt ($EE$: Equal Energy). Die Berechnung der Strahlungsleistung nach Gleichung (2-4) würde bei einem energiegleichen Spektrum für jedes beliebige Frequenzband $[\underline{f}, \bar{f}]$ zum gleichen Ergebnis führen, solange die Bandbreite $\bar{f} - \underline{f}$ konstant ist. Wir bezeichnen mit $E_{\lambda_i}$ die Farbvalenz eines monochromatischen Lichtstrahls mit der Wellenlänge $\lambda_i$ aus dem energiegleichen Spektrum. Nach Gleichung (2-17) kann $E_{\lambda_i}$ stets durch die Linearkombination der Primärvalenzen eines Farbraums wiedergegeben werden:

$$E_{\lambda_i} = \bar{a}(\lambda_i) \cdot A_P + \bar{b}(\lambda_i) \cdot B_P + \bar{c}(\lambda_i) \cdot C_P. \tag{2-21}$$

Die Koeffizienten $\bar{a}(\lambda_i)$, $\bar{b}(\lambda_i)$ und $\bar{c}(\lambda_i)$ haben bei der Verwendung des energiegleichen Spektrums eine besondere Bedeutung und werden Spektralwerte genannt. Die durch Variation von $\lambda$ resultierenden Verläufe $\bar{a}(\lambda)$, $\bar{b}(\lambda)$ und $\bar{c}(\lambda)$ heißen Spektralwertfunktionen. Unter ihrer Kenntnis kann die Farbvalenz $F$ der polychromatischen Strahlung $\Phi_{e,\lambda}$ berechnet werden. Man setzt dazu die Gleichungen (2-17) und (2-20) gleich und erhält

$$A \cdot A_P + B \cdot B_P + C \cdot C_P = F = \sum_{i=0}^{i_{max}} F_{\lambda_i} \cdot \Delta\lambda.$$

Die monochromatischen Farbvalenzen $F_{\lambda_i}$ entsprechen den Farbvalenzen $E_{\lambda_i}$ des energiegleichen Spektrums, wenn man diese mit den Werten $\Phi_{e,\lambda_i}$ aus der Spektralverteilung $\Phi_{e,\lambda}$ des Lichtstrahls und einer wellenlängenunabhängigen Konstanten $k$ skaliert. Diese Überlegung führt zu

$$F = \sum_{i=0}^{i_{max}} F_{\lambda_i} \cdot \Delta\lambda = \sum_{i=0}^{i_{max}} E_{\lambda_i} \cdot k\Phi_{e,\lambda_i}\Delta\lambda.$$

Einsetzen von Gleichung (2-21) und Umformung führt schließlich auf

$$\begin{aligned} F = &(k \cdot \sum_{i=0}^{i_{max}} \bar{a}(\lambda_i) \cdot \Phi_{e,\lambda_i}\Delta\lambda) \cdot A_P + \\ &(k \cdot \sum_{i=0}^{i_{max}} \bar{b}(\lambda_i) \cdot \Phi_{e,\lambda_i}\Delta\lambda) \cdot B_P + \\ &(k \cdot \sum_{i=0}^{i_{max}} \bar{c}(\lambda_i) \cdot \Phi_{e,\lambda_i}\Delta\lambda) \cdot C_P, \end{aligned}$$

wobei die Klammerterme gemäß Gleichung (2-17) den gesuchten Farbwerten $A$, $B$ und $C$ der Farbvalenz $F$ unter den gegebenen Primärvalenzen $A_P$, $B_P$ und $C_P$ des Farbraums entsprechen. Beim Übergang zu infinitesimalen Wellenlängenintervallen erhält man die exakten Beziehungen

$$\begin{aligned} A &= k \cdot \int_{\lambda=\underline{\lambda}}^{\bar{\lambda}} \bar{a}(\lambda) \cdot \Phi_{e,\lambda} d\lambda \\ B &= k \cdot \int_{\lambda=\underline{\lambda}}^{\bar{\lambda}} \bar{b}(\lambda) \cdot \Phi_{e,\lambda} d\lambda \\ C &= k \cdot \int_{\lambda=\underline{\lambda}}^{\bar{\lambda}} \bar{c}(\lambda) \cdot \Phi_{e,\lambda} d\lambda. \end{aligned}$$

### 2.2.4 Spektralfarbenzug

Die mit Gleichung (2-21) eingeführten Spektralwertfunktionen $\bar{a}(\lambda)$, $\bar{b}(\lambda)$ und $\bar{c}(\lambda)$ entsprechen für ein festes $\lambda$ der Farbvalenz eines monochromatischen Lichtstrahls mit der Wellenlänge $\lambda$ im ausgewählten Farbraum. Betrachtet man durch Variation von $\lambda$ im Intervall des sichtbaren Lichts $[\underline{\lambda}, \bar{\lambda}]$ alle Farbvalenzen, die durch monochromatisches Licht hervorgerufen werden können, und bildet diese durch die Bestimmung ihrer Farbwertanteile auf die Farbtafel ab, so erhält man den in Bild 2-14 dargestellten Spektralfarbenzug. Auf der Farbtafel ist der Spektralfarbenzug eine $\lambda$-parametrisierte Kurve $c_{spec}(\lambda)$ mit

$$c_{spec}(\lambda) = \begin{pmatrix} a(\lambda) \\ b(\lambda) \end{pmatrix} \quad \text{mit} \quad a(\lambda) = \frac{\bar{a}(\lambda)}{\bar{a}(\lambda) + \bar{b}(\lambda) + \bar{c}(\lambda)} \quad b(\lambda) = \frac{\bar{b}(\lambda)}{\bar{a}(\lambda) + \bar{b}(\lambda) + \bar{c}(\lambda)},$$

wobei die Koordinaten $a$ und $b$ die Farbwertanteile darstellen. Die Terme für $a$ und $b$ folgen aus den Gleichungen (2-19) und (2-21). In Bild 2-16 ist der Spektralfarbenzug

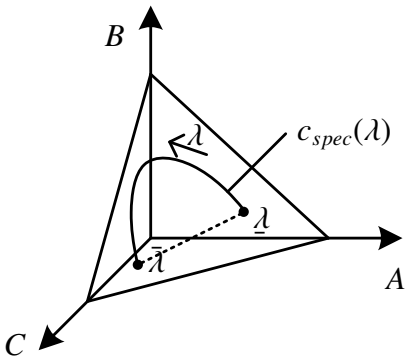


*Bild 2-14*: *Spektralfarbenzug auf der Farbtafel eines Farbraums mit den Primärvalenzen $A_P$, $B_P$ und $C_P$.*

beispielhaft für den CIE-XYZ-Farbraum einsehbar. Er ist unabhängig vom konkreten Farbraum konvex gekrümmt, da sich alle Farbvalenzen aus der additiven Mischung von Spektralfarben ergeben und demzufolge alle geraden Verbindungslinien zwischen zwei beliebigen Spektralfarben innerhalb von $c_{spec}(\lambda)$ verlaufen. Man bezeichnet die Farbarten innerhalb der Fläche, die von $c_{spec}(\lambda)$ und der Verbindungslinie zwischen $c(\underline{\lambda})$ und $c(\bar{\lambda})$ (auch Purpur-Linie, Vgl. Bild 2-16) eingeschlossen werden, als reell. Alle anderen Farbarten werden als imaginär oder virtuell bezeichnet, da Farbvalenzen dieser Farbarten in der Realität nicht auftreten. Wir werden nachfolgend sehen, dass es dennoch von Vorteil sein kann, einen Farbraum mit imaginären Primärvalenzen zu definieren.

### 2.2.5 CIE-RGB-Farbraum

Vergleichbar mit der $V(\lambda)$-Kurve, welche den Energiegehalt einer monochromatischen Strahlung mit dem Helligkeitsempfinden des Menschen in Bezug setzt, hat die CIE 1931 Spektralwertfunktionen auf Grundlage des photometrischen Normalbeobachters festgelegt. Diese beruhen auf Arbeiten von Wright und Guild um 1930 [WS82]. Konkret sollten mehrere Beobachter monochromatische Farbreize durch die additiven Mischungen von rotem, grünem und blauem Licht – den Primärvalenzen – nachstellen. Dabei wurden für jede Wellenlänge der monochromatischen Farbreize die von den Beobachtern eingestellten Anteile der Primärvalenzen aufgezeichnet. Teilweise war es den Beobachtern nicht möglich, die Referenzvalenz durch additive Mischung der Primärvalenzen nachzubilden. In diesem Fall durften sie die Referenz durch Addition einer Primärvalenz verändern, um eine Äquivalenz herstellen zu können. Die notwendige Zugabe zum Referenzlicht wurde dann durch einen negativen Farbwert der jeweiligen Primärvalenz berücksichtigt.

Bevor die CIE die dabei entstandenen Ergebnisse zusammenfasste und 1931 veröffentlichte, wurden die Messungen so umgerechnet, als wären die Primärvalenzen ebenfalls monochromatische Strahlungsquellen. Wie auch aus Bild 2-16 hervorgeht, liegen die durch die Primärvalenzen festgelegten Eckpunkte des Gamut-Dreiecks für den CIE-RGB-

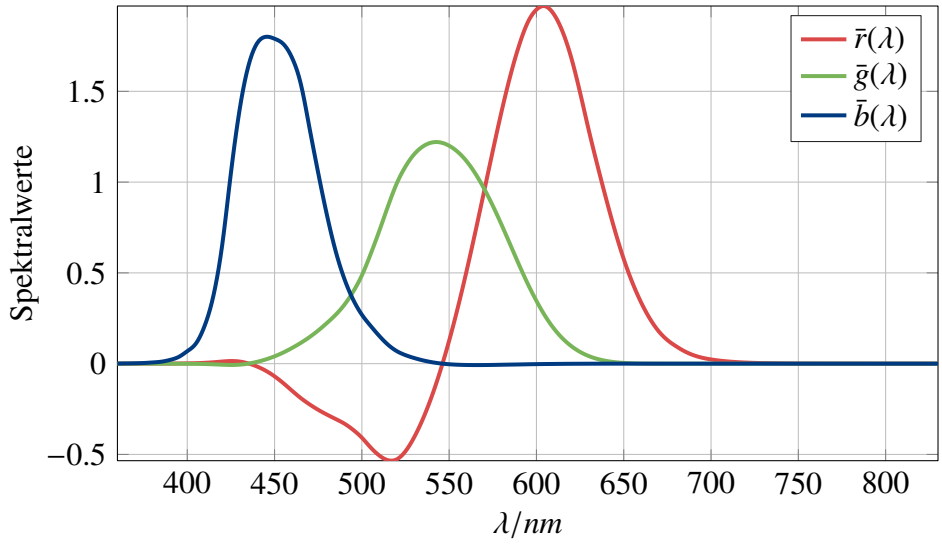


*Bild 2-15*: *Spektralwertfunktionen $\bar{r}(\lambda)$, $\bar{g}(\lambda)$ und $\bar{b}(\lambda)$ bezüglich der roten ($R_P$), grünen ($G_P$) und blauen ($B_P$) Primärvalenz zur addititven Mischung einer monochromatischen Farbvalenz der Wellenlänge $\lambda$.*

Farbraum auf dem Spektralfarbenzug. Konkret wurden aus technischen Gründen die Wellenlängen 700,0 *nm* für Rot ($R_P$), 546,1 *nm* für Grün ($G_P$) und 435,8 *nm* für Blau ($B_P$) gewählt [Sch07]. Das Bild 2-15 zeigt die aus Versuchen gewonnenen Daten in Form der Spektralwertfunktionen.

Die Kurven $\bar{r}(\lambda)$, $\bar{g}(\lambda)$ und $\bar{b}(\lambda)$ stellen also die Anteile der entsprechenden Primärvalenzen dar, die zur Erzeugung der Referenzvalenz nötig sind. Die Helligkeiten der Primärvalenzen wurden dabei so gewählt, dass ihre additive Mischung mit gleichen Farbwerten zur Farbvalenz des energiegleichen Spektrums (Weiß) führt. Die Absolutwerte der Spektralwertfunktionen sind für die Farbart nicht von Bedeutung. Sie ergeben sich aus den Helligkeiten der Primärvalenzen. Deren Leuchtdichtebeiwerte wurden so gewählt, dass zur Erzeugung der Farbvalenz eines energiegleichen Spektrums mit einer Leuchtdichte von $L_W = 1\,\frac{cd}{m^2}$ die Farbwerte $R = G = B = 1$ notwendig sind. Die konkreten Leuchtdichtebeiwerte der Primärvalenzen sind wie folgt:

$$\begin{aligned} L_R &= 0,17697\,\frac{cd}{m^2} \\ L_G &= 0,81240\,\frac{cd}{m^2} \\ L_B &= 0,01063\,\frac{cd}{m^2}. \end{aligned} \tag{2-22}$$

Wie man mit Gleichung (2-18) leicht sieht, besitzt die durch

$$F_W = R \cdot R_P + G \cdot G_P + B \cdot B_P \text{ mit } R = G = B = 1$$

definierte Farbvalenz $F_W$ des energiegleichen Spektrums die gewünschte Leuchtdichte von $1\,\frac{cd}{m^2}$.

### 2.2.6 CIE-XYZ-Farbraum

Neben dem RGB-Farbraum führte die CIE zeitgleich einen weiteren Farbraum ein – den CIE-XYZ-Farbraum. Im Gegensatz zum CIE-RGB-Farbraum hat dieser keinen direkten Bezug zur experimentellen Vermessung. Mehrere andere Vorteile, die im Folgenden erläutert werden, rechtfertigen jedoch die Aufnahme in den Standard.

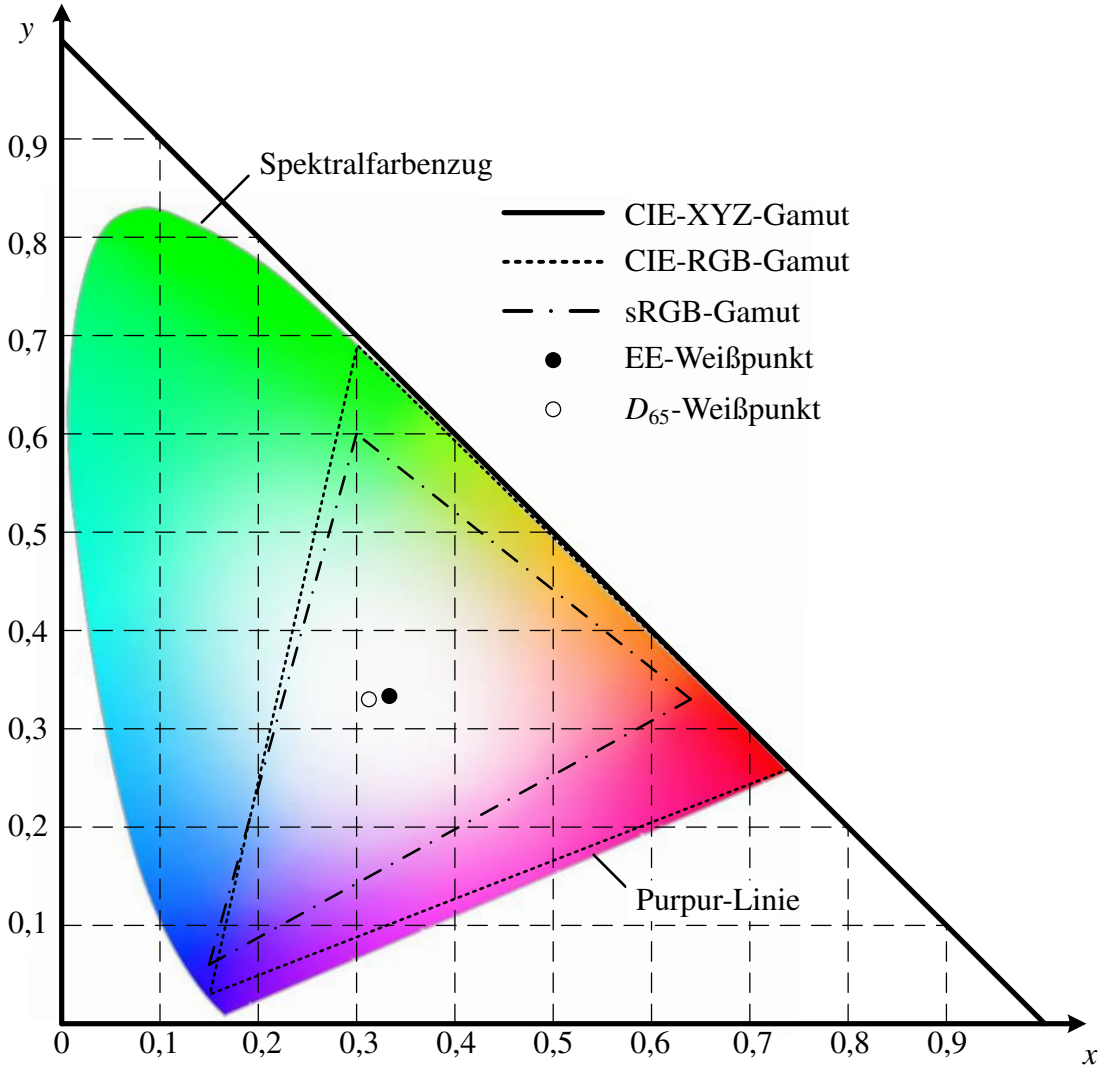


*Bild 2-16*: *Darstellung der Farbarten durch ihre Farbwertanteile x und y in der Normfarbtafel des CIE-XYZ-Farbraums.*

Aus den Spektralwertfunktionen des RGB-Farbraums wird ersichtlich, dass nicht alle Farben durch additive Mischungen der Primärvalenzen erzeugt werden können. Beispielsweise weisen die Spektralfarben im Bereich von ca. 440 bis 550 *nm* negative Farbwerte für die Primärvalenz $R_P$ (rot) auf. Allgemeiner weisen alle Farben negative Koordinaten auf, die außerhalb des Gamut-Dreiecks liegen, welches durch die Primärvalenzen des CIE-RGB-Farbraums aufgespannt wird (Vgl. blaues Gamut-Dreieck in Bild 2-17). Wie aus Bild 2-16 hervorgeht, weist der CIE-RGB-Gamut insbesondere im Bereich blauer und

grüner Farbarten deutliche Defizite auf. Negative Spektralwerte waren für die damalige Handhabung unbequem und sollten im XYZ-System nicht auftreten. Dazu war es nötig, Primärvalenzen zu wählen, deren Gamut-Dreieck den in Bild 2-16 dargestellten Bereich reeller Farben, welcher durch den Spektralfarbenzug und die Purpur-Linie begrenzt wird, vollständig umschließt. So können alle reellen Farben durch innere Mischungen formuliert werden. Das ist nur möglich, wenn die Primärvalenzen selbst imaginär sind und somit nicht physikalisch dargestellt bzw. wahrgenommen werden können. Auf der in Bild 2-16 dargestellten Farbtafel befinden sich die Primärvalenzen $X_P$, $Y_P$ und $Z_P$ per Definition (siehe Gleichung (2-19)) gerade auf den Koordinaten $(1, 0)$, $(0, 1)$ und $(0, 0)$. Ihr Gamut-Dreieck ergibt sich somit aus den Koordinatenachsen und der durchgezogenen schwarzen Linie, die ihre Enden verbindet. Zum besseren räumlichen Verständnis zeigt Bild 2-17 den gleichen Sachverhalt in räumlicher Darstellung. Man erkennt, dass der Spektralfarbenzug (schwarze Kurve) vom roten Gamut-Dreieck des CIE-XYZ-Farbraums vollständig umschlossen wird.

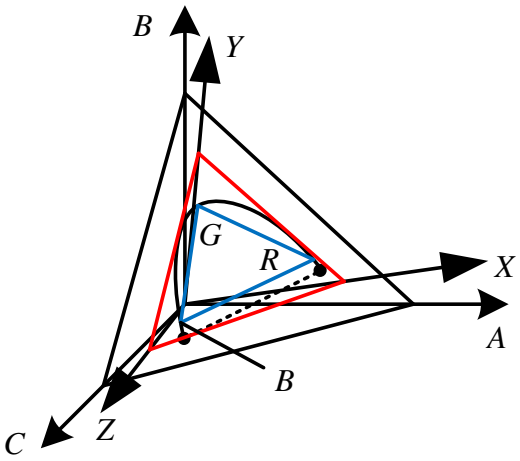


*Bild 2-17*: *Gegenüberstellung des CIE-RGB-(blau) und des CIE-XYZ(rot)-Gamuts auf der Farbtafel.*

Eine weitere hilfreiche Eigenschaft des XYZ-Farbraums ist der unmittelbare Bezug zur Photometrie. Dazu hat man die photometrische Information in der Y-Koordinate isoliert, während die X- und Z-Koordinaten die Farbart beschreiben. Je nach Vorhaben können die relevanten Informationen so unmittelbar aus dem Farbvalenzvektor abgelesen werden. Erreicht wird diese Entkopplung, indem man für die Primärvalenzen $X_P$ und $Z_P$ Vektoren aus der sogenannten Alychne (die Lichtlose) wählt. Hierbei handelt es sich um die Ebene der Farbvalenzen, deren photometrische Größe (meist Leuchtdichte) null ist. Für das Beispiel der Leuchtdichte kann die Alychnenebene mithilfe der Leuchtdichtebeiwerte aus Gleichung (2-22) bezüglich der Koordinaten im RGB-Farbraum formuliert werden:

$$L_R \cdot R + L_G \cdot G + L_B \cdot B = 0.$$

Um mit der Y-Koordinate des CIE-XYZ-Farbraums besonders komfortabel umgehen zu können, wird die Spektralwertfunktion $\bar{y}(\lambda)$ so gewählt, dass sie der Hellempfindlichkeitsfunktion $V(\lambda)$ aus Bild 2-9 entspricht. Diese Wahl setzt voraus, dass $\bar{y}(\lambda)$ aus farbmetrischen Spektralwert-Funktionen (hier: $\bar{r}(\lambda), \bar{g}(\lambda), \bar{b}(\lambda)$) linear kombiniert werden kann. Diese Forderung ist jedoch unter sehr allgemeinen Voraussetzungen erfüllt [WS82]. Durch Gegenüberstellung der Gleichung (2-14) zur Ermittlung einer photometrischen Größe $X_v$ aus einer strahlungsphysikalischen Größe $X_e$ und der Gleichung (2-20), welche die Farbvalenzkoordinaten auf Basis der Spektralverteilung $\Phi_{e,\lambda}$ der strahlungsphysikalischen Größe $X_e$ ausdrückt, ergibt sich folgender Zusammenhang:

$$X_v = K_m \int_{\lambda=\underline{\lambda}}^{\bar{\lambda}} X_e(\lambda) \cdot V(\lambda) d\lambda = k \int_{\lambda=\underline{\lambda}}^{\bar{\lambda}} \Phi_{e,\lambda} \cdot \bar{y}(\lambda) d\lambda = Y.$$

Nun ist offensichtlich, dass die photometrische Größe $X_v$ und die $Y$-Koordinate des XYZ-Farbraums über das Verhältnis der Konstanten $K_m$ und $k$ in einem proportionalen Zusammenhang stehen. Erfährt der Beobachter die Farbvalenz durch die Betrachtung einer Lichtquelle, wählt man $k = K_m$ und damit $Y = X_v$. Für die Betrachtung beleuchteter, nicht selbst leuchtender farbiger Gegenstände, muss darüber hinaus noch das spektrale Reflexionsvermögen des Objekts berücksichtigt werden. Den angestellten Überlegungen zufolge müssen sich die Leuchtdichtebeiwerte

$$\begin{aligned} L_X &= 0 \, \frac{cd}{m^2} \\ L_Y &= 1 \, \frac{cd}{m^2} \quad \text{und} \\ L_Z &= 0 \, \frac{cd}{m^2} \end{aligned}$$

für die Primärvalenzen im CIE-XYZ-Farbraum ergeben, damit gemäß des Abneyschen Gesetzes (Gleichung 2-18) die Gleichheit der Leuchtdichte (für $X_v$) und der Y-Koordinate der Farbvalenz $F$ einer Strahlung gegeben ist:

$$L_F = L_X \cdot X + L_Y \cdot Y + L_Z \cdot Z = Y.$$

Unter den genannten und einigen weiteren Forderungen an den CIE-XYZ-Farbraum ergibt sich schließlich die folgende Transformation von CIE-RGB nach CIE-XYZ [Hun98]:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \begin{pmatrix} 0,49 & 0,31 & 0,20 \\ 0,17697 & 0,81240 & 0,01063 \\ 0,00 & 0,01 & 0,99 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2-23}$$

Die Matrix in Gleichung (2-23) ist die Basiswechselmatrix zwischen beiden Farbräumen. Ihre Werte in Zeile 2 entsprechen den Leuchtdichtebeiwerten der RGB-Primärvalenzen (Vgl. (2-22)). Da außerdem alle Zeilensummen gleich 1 sind, stimmen die Weißpunkte des RGB-Raums und des XYZ-Raums überein ($R = G = B \equiv X = Y = Z$). Aus diesem Grund befindet sich der $EE$-Weißpunkt, der zuvor für den CIE-RGB-Farbraum mit $r = g = b = \frac{1}{3}$ eingeführt wurde, auch auf der CIE-XYZ-Farbtafel in Bild 2-16 an den Koordinaten $x = y = \frac{1}{3}$.

Die Transformationsmatrix kann so modifiziert werden, dass ein Weißpunktwechsel stattfindet. Das Weiß des energiegleichen Spektrums ($R = G = B$ bzw. $r = g = b = \frac{1}{3}$) ist

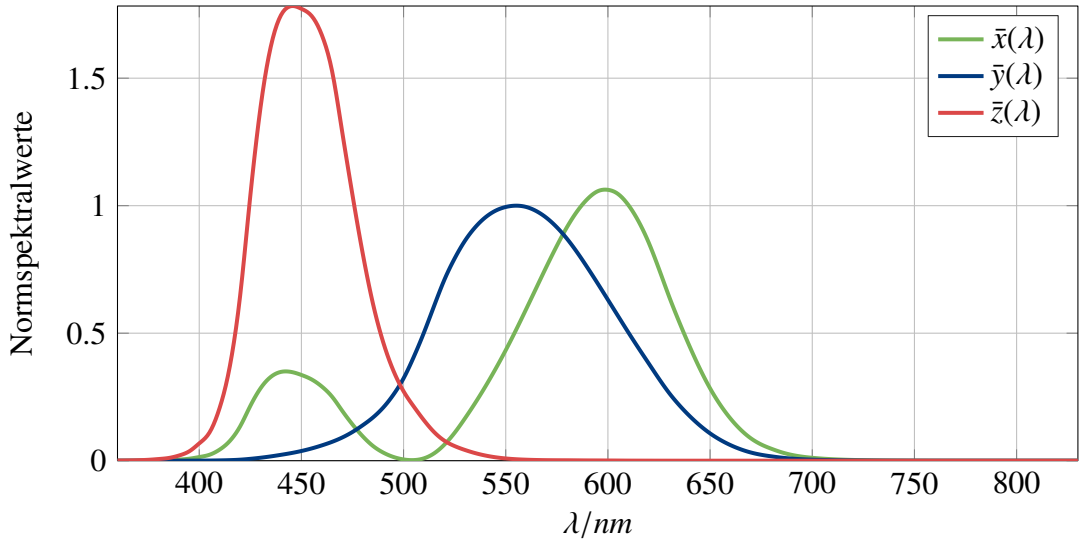


*Bild 2-18*: *Spektralwertfunktionen $\bar{x}$, $\bar{y}$ und $\bar{z}$ des CIE-XYZ-Farbraums von* 360 *nm bis* 830 *nm in* 1 *nm Schritten nach DIN 11664 Teil 1 [DIN05].*

nicht immer die ideale Wahl für den Weißpunkt, da sich das menschliche Auge auf die dominante Beleuchtung einstellt und diese als weiß empfindet, während ähnliche Farbvalenzen als leicht farbstichig empfunden werden. Beispielsweise hat der Weißpunkt der Normlichtart $D_{65}$, welche sich besonders für Beleuchtungssituation bei Tageslicht eignet, auf der Farbtafel des CIE-XYZ-Farbraums die Koordinaten $x = 0,3127$ und $y = 0,329$ (leicht blau). Stellt sich das Auge auf eine solche Beleuchtungssituation ein, rückt der $D_{65}$-Weißpunkt auf die Farbtafel-Position $a = b = \frac{1}{3}$ und das energiegleiche Spektrum erscheint leicht rötlich.

Es ergeben sich die in Bild 2-18 dargestellten Spektralwertfunktionen für den CIE-XYZ-Farbraum. Erwartungsgemäß nehmen die Spektralwertfunktionen keine negativen Werte an. Außerdem entspricht die Spektralwertfunktion $\bar{y}(\lambda)$ der Primärvalenz $Y_P$ der $V(\lambda)$-Kurve aus Bild 2-9.

Der CIE-XYZ-Farbraum wurde nach DIN 5033 als Grundlage zur „Farbbeschreibung von Materialien und Lichtern auf metrischer Grundlage“ erklärt [DIN04]. Demzufolge heißen die Primärvalenzen $X_P$, $Y_P$ und $Z_P$ auch Normvalenzen, die Farbwerte $X$, $Y$ und $Z$ Normfarbwerte, die Farbwertanteile $x$, $y$ und $z$ Normfarbwertanteile, die Spektralwerte $\bar{x}$, $\bar{y}$ und $\bar{z}$ Normspektralwerte und die $(x, y)$-Farbtafel Normfarbtafel.

### 2.2.7 Standard-RGB-Farbraum

Zuletzt soll noch auf den Standard-RGB-Farbraum (sRGB) hingewiesen werden, welcher für Farbschirme eine herausragende Rolle spielt. Zur Erzeugung von Farbvalenzen haben Monitore pro Pixel drei verschiedenfarbige Lichtquellen, welche die Rolle der Primärvalenzen einnehmen. Um einen möglichst großen Anteil der Farben innerhalb des Spektralfarbenzugs darstellen zu können, sollten die Primärvalenzen so gewählt werden, dass sie ein möglichst großes Gamut-Dreieck aufspannen. Gleichzeitig bedeutet dies, dass sie sich nahe der Spektralfarben befinden sollten. Hier entsteht ein Konflikt zwischen der

Vielfalt darstellbarer Farben und der Leuchtkraft sowie der technischen Realisierbarkeit, da die Primärvalenzen nur auf einem sehr schmalen Wellenlängenbereich leuchten dürfen. Aus diesem Grund muss bei der Positionierung der Primärvalenzen ein Kompromiss getroffen werden.

Die Primärvalenzen des sRGB-Farbraums, welche repräsentativ für Bildschirme sind, liegen an den Koordinaten (0,64; 0,33) (Rot), (0,30; 0,60) (Grün) und (0,15; 0,06) (Blau) in der Normfarbtafel. Es spannt sich so das in Bild 2-16 dargestellte Gamut-Dreieck auf. Außerdem wird der Weißpunkt der Normlichtart $D_{65}$ verwendet.

## 2.3 Computergrafik

Der Schlüssel zur echtzeitfähigen Darstellung virtueller Szenen, aber auch zur rasanten Lösung einiger allgemeiner Fragestellungen aus Wissenschaft und Technik geht auf die Computergrafik zurück, die durch raffinierte und hochgradig parallelisierte Prozesse rasante Berechnungen ermöglicht. Die zusätzliche hard- und softwareseitige Unterstützung zur Verwendung der Grafikkarte in grafikfernen Problemstellungen, wie sie beispielsweise durch CUDA (NVIDIA) oder OpenCL (Khronos Group) geleistet wird, erleichtert die Einbindung der Grafikkarte in alle Programme mit parallel berechenbaren Programmabschnitten und schafft so einen erheblichen Performance-Gewinn [Nvi07], [Khr19].

Um dem Leser ohne Hintergrundwissen in Themen der Computergrafik einen schlüssigen Einstieg zu ermöglichen, soll in diesem Abschnitt zunächst die klassische Fixed Function Pipeline mit Fokus auf die Berechnung grafischer Ausgaben vorgestellt werden. Anschließend wird die im Laufe der Jahre zunehmende Anpassbarkeit der Rendering Pipeline thematisiert bis schließlich auf Möglichkeiten eingegangen wird, allgemeine grafikferne Berechnungen auf der GPU (Graphics Processing Unit) auszuführen. Besonderes Augenmerk wird auf die etablierten Methoden der Berechnung von Lichteinflüssen in der virtuellen Szene gelegt, die im Rahmen dieser Arbeit von besonderem Interesse sind.

Die aufgeführten Inhalte und viele darüber hinaus gehende Aspekte der Computergrafik werden in [AFS+13] zugänglich beschrieben. Bezüglich der Darstellung der Rendering-Pipeline kann insbesondere auf die gut dokumentierte OpenGL-Pipeline verwiesen werden [SA06]. Diese Quellen bilden zusammen mit [FK03], welche den Fokus auf Shader (auf der GPU ausführbare Programme) richtet, die Wissensbasis der nachfolgenden Darstellungen.

### 2.3.1 Fixed Function Pipeline

Bis 2001 konnte die auf der Grafikkarte stattfindende Verarbeitungskette durch die in Bild 2-19 dargestellte Fixed Function Pipeline beschrieben werden. Sie trägt ihren Namen, da sie zwar von außen konfigurierbar, aber nicht programmierbar ist. Die Beeinflussung der Verarbeitungsschritte kann nur innerhalb eines vordefinierten Parameterraums erfolgen, während die zugrunde liegenden Abläufe durch die Hardware-Architektur vorgegeben sind.

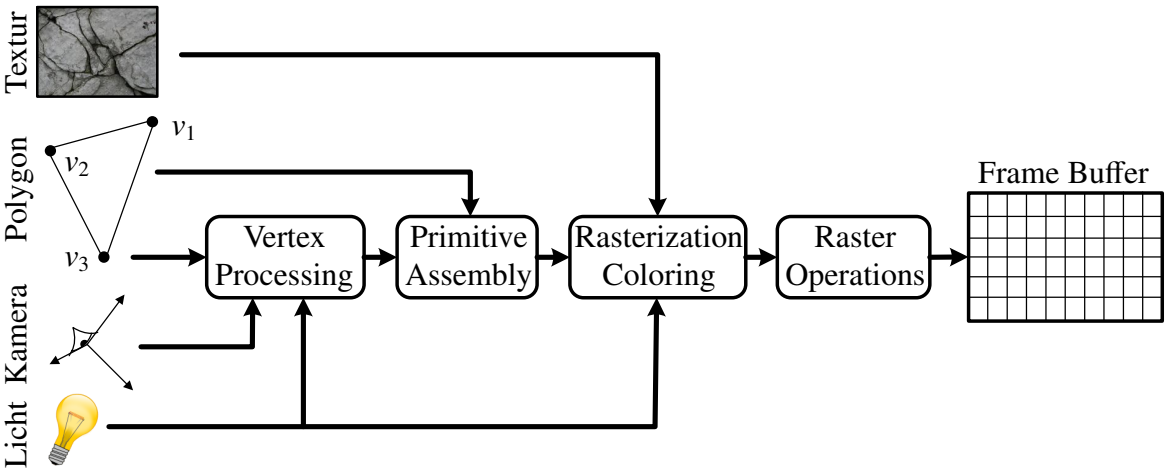


*Bild 2-19*: *Fixed Function Pipeline.*

In den nun folgenden Unterabschnitten sollen die Teilprozesse innerhalb der Fixed Function Pipeline in Kürze dargestellt werden, sodass darauf aufbauend die Unterschiede zur programmierbaren Pipeline deutlich werden.

### Input

Zuallererst soll auf die Herkunft und Gestalt der in die Rendering Pipeline einfließenden Daten eingegangen werden. Ausgangspunkt für ein Rendering ist zunächst eine grafische Anwendung (siehe Bild 2-20: 3D-Anwendung). Sie enthält Daten zur Beschreibung aller Objekte und Oberflächen, die sich in ihr darstellen lassen. Die Objektgeometrie wird dabei meist durch Polyeder beschrieben. Formen, die nicht exakt durch Polyeder dargestellt werden können, werden durch solche approximiert. Hierbei steigt die Genauigkeit mit der Anzahl der Polygone. Datentechnisch werden die Polyeder durch Listen ihrer Eckpunkte (Vertex, pl. Vertices) und der sie verbindenden Kanten (Edge, pl. Edges) beschrieben. Neben der Geometrie bildet das Aussehen der Objektoberflächen einen weiteren wichtigen Bestandteil zur Erzeugung einer realistischen visuellen Wahrnehmung. Neben verschiedenen Parametern, die z.B. die Wechselwirkung der Flächen mit Licht beschreiben, nehmen vor allem Texturen eine wesentliche Rolle ein. Diese kann man sich als Bilder vorstellen, welche auf die Objektgeometrie gelegt werden und so die Einfärbung der Oberflächen definieren. Auf diese Weise können reale Objekte mit verhältnismäßig wenigen Polygonen einigermaßen realistisch visualisiert werden. Am Beispiel einer Straße kann man sich diese als einen einfachen Quader vorstellen, während eine auf der Quaderoberfläche liegende Textur die Seiten- und Mittelstreifen und sogar die Poren des Asphalts visualisiert. Es wird leicht deutlich, welchen Aufwand man betreiben müsste, um ein ähnliches Ergebnis durch das Zusammensetzen einfarbiger Polygone zu erzeugen. Die Positionierung der Texturen auf den Polygonflächen erfolgt dabei über die Verknüpfung der Eckpunkte der Polygone mit definierten Stellen auf der Textur (Texturkoordinaten). Vertices tragen also nicht nur die Information über ihre Position, sondern enthalten darüber hinaus Texturkoordinaten, Flächennormalen und andere Daten, die während des Pipeline-Durchlaufs benötigt werden. Neben Texturen, die zur Einfärbung der Objektflächen dienen, gibt es ähnliche Konstrukte,

mit denen die Wechselwirkungen mit Licht innerhalb der Polygonflächen variiert werden können (z.B. Normal-Maps, Bump-Maps, Displacement-Maps).

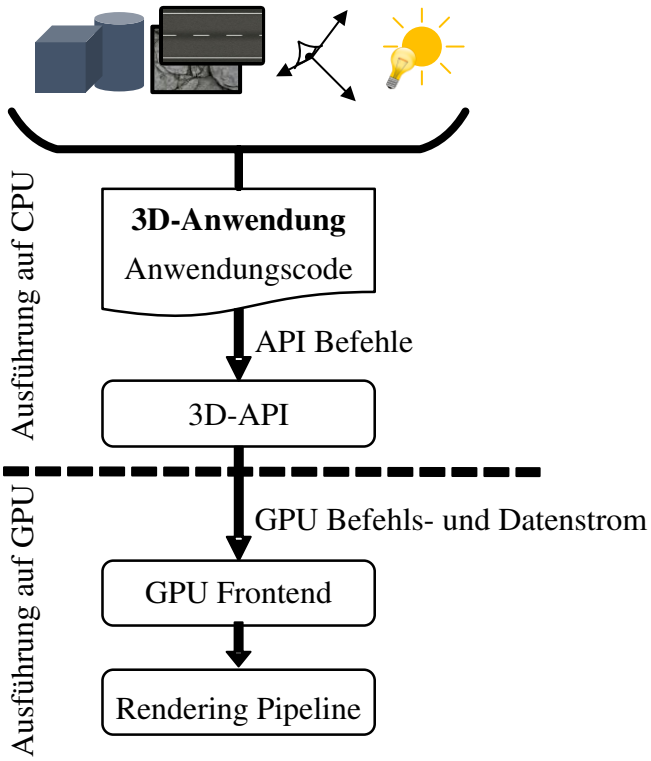


*Bild 2-20*: *Schnittstelle zwischen 3D-Anwendung und Grafikkarte.*

Neben den Objektbeschreibungen müssen weitere Informationen durch die 3D-Anwendung spezifiziert werden. Zentral für die Darstellung der zunächst dreidimensionalen Szene auf einem 2D-Ausgabegerät ist auch der Betrachter. Alle notwendigen Spezifikationen zur Position, Orientierung, Sichtweite und Perspektive des Betrachters werden unter dem Begriff Kameraparameter zusammengefasst. Im Kontext der Computergrafik ist der Betrachter mit der Kamera gleichzusetzen, da der Anwender die virtuelle Szene als Aufnahme einer virtuellen Kamera beobachtet, deren Ausgabe auf dem Bildschirm dargestellt wird. Als letztes wesentliches Element zur Beschreibung einer 3D-Szene sei auf die in ihr befindlichen Lichter hingewiesen. Weitere Details zu allen bisher genannten Aspekten folgen in den nächsten Unterabschnitten.

Die Entwicklung einer interaktiven Anwendung mit dynamischen 3D-Szenen erfordert den Einsatz von darauf zugeschnittenen Grafik-Engines. Diese stellen zahlreiche Funktionen, die im Kontext der grafischen Ausgabe von 3D-Szenen benötigt werden, bereit und unterstützen den Entwickler bei der Verwaltung und Verarbeitung der Szene. Oft mit Grafik-Engines gleich gesetzt, aber genau genommen noch umfangreicher, sind Spiele-Engines. Beispiele hierfür sind Unity3D oder Unreal [Uni20], [Epi20]. Sie ergänzen den Funktionsumfang einer Grafikengine durch zusätzliche spieletypische Funktionen, wie die Modellierung physikalischer Effekte (Schwerkraft, Kollision von Objekten, Trägheit, ...) oder vorbereitete GUI-Elemente (Menüs, Schaltflächen, ...). Zudem wird hier ein größe-

res Augenmerk auf Echtzeitfähigkeit gelegt, da in interaktiven Anwendungen nur einige Millisekunden zur Berechnung eines Bilds zur Verfügung stehen.

Unabhängig davon, ob bei der Entwicklung auf eine Spiele-Engine zurückgegriffen wird oder der Programmierer sämtliche Abläufe selbst implementiert, muss die 3D-Anwendung zur grafischen Ausgabe auf eine 3D-Programmierschnittstelle (API, Application Programming Interface) zurückgreifen. Die bekanntesten Beispiele an dieser Stelle sind DirectX und OpenGL [Mic18], [SA06]. In den letzten Jahren erfreut sich die in 2016 eingeführte API Vulkan vermehrter Beliebtheit [The14]. Diese reduziert die Abstraktionsschicht zwischen Grafikkarte und 3D-Anwendung stärker als es die etablierten Varianten tun und verschiebt damit den Fokus von einer einfach und komfortabel benutzbaren High-Level-Schnittstelle zur Recheneffizienz. Aufgabe der 3D-API ist es, die unterschiedlichen Grafikkarten der verschiedenen Anbieter soweit zu abstrahieren, dass die Anwendung weitgehend unabhängig von der konkreten Hardware des Zielsystems implementiert werden kann. Dazu agiert sie, wie in Bild 2-20 zu sehen, als Dolmetscher und übergibt die allgemeingültigen API Befehle als hardwarespezifische Anweisungen an die Grafikkarte weiter. Konkret beinhalten die übermittelten Informationen zum Beispiel Objektdaten (Vertex- und Edge-Listen, Texturen, ...) oder das Rendering beeinflussende Kontextinformationen (Transformation des Objektmodells in die Szene, Kameralage).

### Vertex Processing

Die in der Anwendung definierte Szene ist zeitveränderlich, weil sich beispielsweise Objektlagen, die Kameralage oder die Beleuchtungssituationen ändern können. Aus diesem Grund muss ihr Abbild (Frame) aus Sicht der Kamera wiederholt erstellt werden. Wie oft das geschieht, hängt von der Komplexität der Szene und des Renderings sowie der Rechenleistung der GPU ab. Basierend auf der menschlichen Wahrnehmung sollten mindestens 30 Frames pro Sekunde (fps) erreicht werden, um ein flüssig wirkendes Video zu erzeugen. Gängige Ausgabegeräte verfügen über eine Bildwiederholfrequenz von 60 fps, sodass dieser Wert als wünschenswert angesehen werden kann [AFKN95]. Im Gaming-Bereich exisitieren inzwischen noch höhere Bildwiederholfrequenzen. Neue Spielkonsolen, wie die XBox Series X oder die Playstation 5 unterstützen bis zu 120 fps [Mic20],[Son20]. High End Gaming Monitore erlauben die Darstellung von bis zu 360 fps [ASU20]. Aufgrund der vielen Berechnungen, die innerhalb kürzester Zeit durchgeführt werden müssen, ist die Abarbeitung der einzelnen Schritte des Renderings zur Durchsatzoptimierung in einer Pipeline organisiert. So kann mit der Ausführung der nächsten Instruktion bereits begonnen werden, nachdem der erste Teilschritt der vorhergehenden Instruktion abgeschlossen ist.

Um die 3D-Szene in einen Frame zu überführen, findet im ersten Teilschritt das Vertex Processing statt. Dieser Begriff fasst alle Operationen zusammen, die pro Vertex abgearbeitet werden. Wichtig hierbei ist, dass die Verarbeitung der einzelnen Vertices unabhängig voneinander geschieht. Diese Forderung bildet eine wichtige Voraussetzung für den enormen Durchsatz der Rendering Pipeline. Würden Abhängigkeiten zwischen den verschiedenen Vertices bestehen, so müsste die Ausführung an vielen Stellen pausiert werden, um auf Ergebnisse anderer Ausführungseinheiten (Threads) zu warten, wodurch die hochgradig parallele Abarbeitung des Datenstroms in mehreren Pipelines nur noch eingeschränkt möglich wäre.

Der wichtigste Schritt innerhalb des Vertex Processing ist die Transformation der Vertices, welche zunächst im lokalen Koordinatensystem des jeweiligen Objekts vorliegen (Model Space *m*), über verschiedene Zwischenschritte in die normalisierten Ausgabekoordinaten (Normalized Device Coordinates, *NDC*), von denen an späterer Stelle auf das Pixelraster des Ausgabegeräts übergegangen wird. Bild 2-21 gibt einen Überblick über die zunächst relevanten Koordinatensysteme.

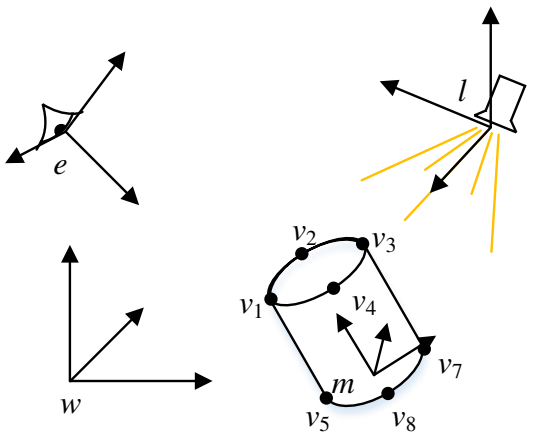


*Bild 2-21*: *Verschiedene Koordinatensysteme bei der Vertex-Tranformation (Model Space m, World Space w, Eye Space e und Light Space l).*

Zu Beginn der Pipeline liegen die Vertices in den Model Spaces der jeweiligen Objekte vor. In Bild 2-21 wird beispielsweise durch die Vertices $v_1, ..., v_8$ die dargestellte Zylinderform im Model Space *m* approximiert. Die Transformationen zwischen den Koordinatensystemen lassen sich in drei atomare Operationen zerlegen – Rotation, Skalierung und Translation. Die Rotation eines dreidimensionalen Vektors *v* kann am Beispiel der Drehung um den Winkel $\alpha$ bezüglich der *x*-Achse durch

$$v' = R_{rot} \cdot v; \text{ Bsp. Rotation um } x\text{-Achse: } R_{rot} = R_{rot,x} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}$$

ausgedrückt werden. Entsprechende Drehungen um andere Achsen ergeben sich analog. Allgemeine Drehungen können durch Konkatenation der Rotationsmatrizen realisiert werden und werden üblicherweise durch Euler- bzw. Kardan-Winkel oder Quaternionen ausgedrückt. Skalierungen bezüglich der verschiedenen Achsen werden gemäß

$$v' = S \cdot v \text{ mit } S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

beschrieben. Die Translation hingegen kann nicht durch die Multiplikation mit einer Matrix ausgedrückt werden, sondern bedarf der Addition

$$v' = v + \Delta v \text{ mit } \Delta v = \begin{pmatrix} \Delta v_x \\ \Delta v_y \\ \Delta v_z \end{pmatrix}.$$

Sie kann deshalb nicht als lineare Abbildung aufgegriffen werden. Zur Transformation eines Objekts in ein anderes Koordinatensystem, müssen auf alle Vertices die gleichen atomaren Operationen angewandt werden. Die Nichtlinearität der Translation ist insofern hinderlich, als dass die Konkatenation linearer Abbildungen zu einer einzigen Matrix zusammengefasst werden kann und deshalb besonders recheneffizient wäre. Aus diesem Grund verwendet man zur Dastellung von Vertices im dreidimensionalen Raum homogene Koordinaten [AFS+13]. Dazu wird der Vertex $v$ künstlich um eine vierte Dimension erweitert, wobei aus dem so gewonnenen Vektor $v_h$ jederzeit der ursprüngliche Vertex durch Normierung auf die vierte, auch als $w$ bezeichnete Komponente rekonstruiert werden kann.

$$v = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \in \mathbb{R}^3 \rightarrow v_h = \begin{pmatrix} w \cdot v_x \\ w \cdot v_y \\ w \cdot v_z \\ w \end{pmatrix} \in \mathbb{R}^4, w \neq 0 \text{ (normalisiert: } w = 1)$$

Vektoren mit unterschiedlichen Werten für $w$, aber gleichen Werten für $v_x$, $v_y$ und $v_z$ sind äquivalent. Sie stellen den gleichen dreidimensionalen Vertex dar. Zur Erweiterung eines Vertex auf die homogene Form wählt man der Einfachheit halber $w = 1$. Handelt es sich nicht um Orts- sondern Richtungsvektoren, muss als vierte Komponente eine 0 ergänzt werden. Unter Verwendung homogener Koordinaten ergeben sich folgende Abwandlungen der vorgestellten atomaren Transformationen:

$$R_{rot} \rightarrow R_h \quad \text{Bsp.: } R_{h,x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S \rightarrow S_h = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$+t \rightarrow T_h = \begin{pmatrix} 1 & 0 & 0 & \Delta v_x \\ 0 & 1 & 0 & \Delta v_y \\ 0 & 0 & 1 & \Delta v_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Wie sich herausstellt, handelt es sich in allen Fällen um lineare Abbildungen. Demzufolge können alle notwendigen Operationen zur Tranformation eines Vertex in ein anderes Koordinatensystem durch eine einzige Transformationsmatrix

$$T = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2-24}$$

beschrieben werden. Dabei entsprechen die Einträge $d$, $h$ und $l$ dem Translationsvektor. Die Einträge $a$, $f$ und $k$ beinhalten die Skalierungsfaktoren. Gleichzeitig wird die obere linke 3x3-Matrix durch die atomaren Rotationen bestimmt.

Auf dieser mathematischen Grundlage können nun für einen Frame der Szene zunächst objektspezifische Transformationsmatrizen $M$ gefunden werden, welche die Vertices ${}_m v$ der Objekte in den World-Space $w$ transformieren. Auf gleiche Weise kann eine framespezifische Matrix $V$ gefunden werden, welche die Vertices ${}_w v$ anschließend in den Eye-Space überführt. Zusammengefasst können die in die Pipeline einfließenden Vertices ${}_m v$ mit nur einer Matrixmultiplikation

$$ {}_e v = V \cdot M \cdot {}_m v $$

in den Eye-Space transformiert werden, wobei die Matrizen $V$ (spezifisch für den Frame) und $M$ (spezifisch für Objekt und Frame) von der 3D-Anwendung vorgegeben sind und ihr Produkt $V \cdot M$ zuvor objektweise gebildet wurde.

Mit der Überführung aller Vertices in das Kamerakoordinatensystem sind die notwendigen Schritte zur Darstellung auf einem Ausgabegerät noch nicht abgeschlossen. Die Szene liegt noch immer in dreidimensionaler Gestalt vor. Sie muss nun unter Berücksichtigung der Perspektive auf eine Ebene abgebildet werden. Bild 2-22 veranschaulicht die vorliegende Situation.

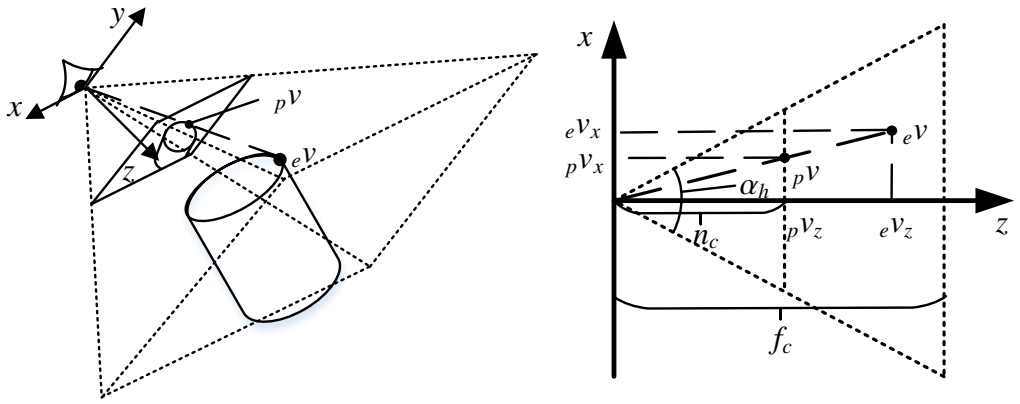


*Bild 2-22*: *Abbildung der Vertices vom View-Space auf die Near Clipping Plane.*

Im linken Bereich von Bild 2-22 wird die Abbildung des Vertex ${}_e v$ auf die Projektionsebene visualisiert. Sein Abbild wird mit ${}_p v$ bezeichnet. Zur besseren Übersicht zeigt der rechte Teil die gleiche Situation auf die $xz$-Ebene reduziert. Die Projektionsebene wird auch als Near Clipping Plane bezeichnet. Ihr Abstand zur $xy$-Ebene des Kamerakoordinatensystems beträgt $n_c$. Vertices, deren $z$-Koordinate kleiner als $n_c$ ist, liegen außerhalb des sichtbaren Volumens der Kamera (View Frustum) und werden nicht dargestellt. Nach hinten wird das View Frustum durch die Far Clipping Plane mit dem Abstand $f_c$ zur $xy$-Ebene begrenzt. Die seitlichen Grenzebenen des View Frustum ergeben sich über den horizontalen ($\alpha_h$) oder vertikalen ($\alpha_v$) Öffnungswinkel (FoV: Field of View) und das Seitenverhältnis $b_{out} : h_{out}$ (aspect ratio).

Unter Zuhilfenahme des Strahlensatzes kann auf Basis der Skizze 2-22 eine Zuordnung zwischen ${}_e v$ und seiner Projektion ${}_p v$ gemäß

$$ \frac{{}_e v_x}{{}_p v_x} = \frac{{}_e v_z}{n_c} \Leftrightarrow {}_p v_x = \frac{n_c}{{}_e v_z} \cdot {}_e v_x; \text{ analog } {}_p v_y = \frac{n_c}{{}_e v_z} {}_e v_y $$

gefunden werden. Weiterhin gilt ${}_pv_z = n_c$. Zusammengefasst kann die Projektion eines Vertex auf die Near Clipping Plane in homogenen Koordinaten ebenfalls linear formuliert werden:

$$ {}_pv' = P' \cdot {}_ev \text{ mit } P' = \begin{pmatrix} n_c & 0 & 0 & 0 \\ 0 & n_c & 0 & 0 \\ 0 & 0 & n_c & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \rightarrow {}_pv' = \begin{pmatrix} n_{ce}v_x \\ n_{ce}v_y \\ n_{ce}v_z \\ {}_ev_z \end{pmatrix} \xrightarrow{norm} {}_pv' = \begin{pmatrix} \frac{n_c}{{}_ev_z} {}_ev_x \\ \frac{n_c}{{}_ev_z} {}_ev_y \\ n_c \\ 1 \end{pmatrix} \quad (2\text{-}25) $$

Die perspektivischen Divisionen im letzten Schritt der Umformung (2-25) sind aus rechentechnischer Sicht kostspielig. Es empfiehlt sich deshalb, diese Operationen nur für Vertices zu vollziehen, die sich innerhalb des View Frustums befinden (Clipping). Die dazu notwendige Überprüfung ist im Eye-Space aufgrund der Pyramidenstumpf-Geometrie rechenintensiv. Ein weiterer Nachteil der vorgestellten Transformation ist, dass die Tiefeninformation des Vertex nach der Normierung verloren geht. Diese wird jedoch an späterer Stelle nochmal benötigt. Um die angeführten Probleme zu umgehen, wählt man anstelle der in Gleichung (2-25) vorgestellten Lösung eine Transformation, welche das pyramidenstumpfförmige View Frustum auf einen im Ursprung zentrierten Würfel mit Kantenlänge 2 abbildet (Clip-Space). Das Bild 2-23 veranschaulicht die Transformation in den Clip-Space, wie sie durch die Matrix $P$ in Gleichung (2-26) realisiert wird. Die Variablen $b_{out}$ und $h_{out}$ bezeichnen dabei die Breite (entlang der $x$-Achse) und Höhe (entlang der $y$-Achse) der Near Clipping Plane, welche sich aus dem FoV $\alpha_h$ und $\alpha_v$ sowie $n_c$ ergeben.

$$ P = \begin{pmatrix} \frac{2}{b_{out}} n_c & 0 & 0 & 0 \\ 0 & \frac{2}{h_{out}} n_c & 0 & 0 \\ 0 & 0 & \frac{f_c+n_c}{f_c-n_c} & -\frac{2n_c f_c}{f_c-n_c} \\ 0 & 0 & 1 & 0 \end{pmatrix} \rightarrow {}_cv = P \cdot {}_ev = \begin{pmatrix} \frac{2}{b_{out}} n_{ce}v_x \\ \frac{2}{h_{out}} n_{ce}v_y \\ \frac{f_c+n_c}{f_c-n_c} {}_ev_z - \frac{2n_c f_c}{f_c-n_c} \\ {}_ev_z \end{pmatrix} \quad (2\text{-}26) $$

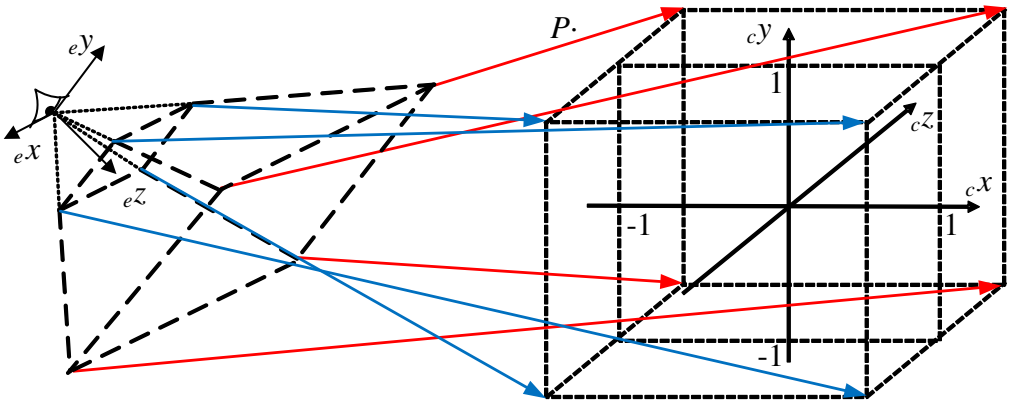


*Bild 2-23*: *Transformation vom Eye-Space in den Clip-Space durch die Matrix P.*

Das Clipping, welches namensgebend für den Clip-Space ist, vereinfacht sich für den transformierten Vektor ${}_cv$ erheblich, da aufgrund des nun würfelförmigen View Frustums

einfache Intervall-Tests der einzelnen Koordinaten genügen. Um die zur Normierung notwendige Division aller Komponenten durch ${}_ez$ auszusparen, vergleicht man die $x$-, $y$- und $z$-Komponenten betragsmäßig gegen die $w$-Komponente des Vektors ${}_cv$ (anstelle von 1):

$$ {}_ev \text{ im View Frustum} \Leftrightarrow -{}_cv_w < {}_cv_x, {}_cv_y, {}_cv_z < {}_cv_w $$

Vertices, die diese Bedingung nicht erfüllen, liegen außerhalb des View Frustums. Abhängig davon, ob die durch sie beschriebenen Polygone ausschließlich oder nur teilweise außerhalb des View Frustums liegen, müssen gegebenenfalls neue Vertices generiert werden, um die sichtbare Teilfläche des Polygons zu erhalten. Bild 2-24 veranschaulicht denkbare Fälle.

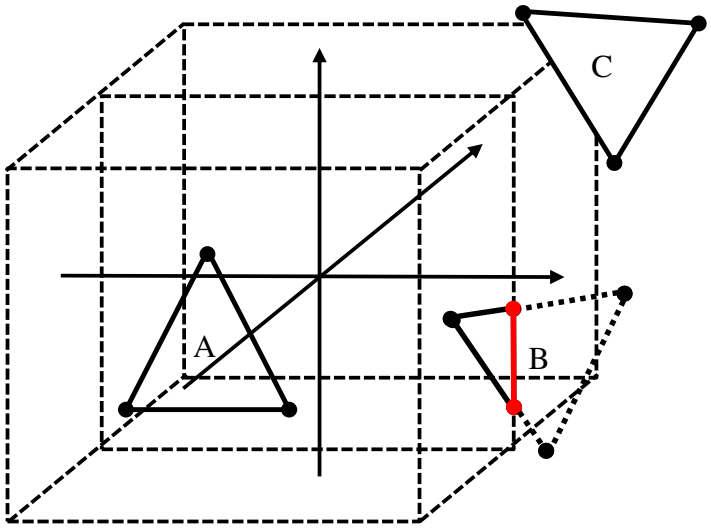


*Bild 2-24*: *Clipping von Polygonen im Clip-Space.*

Im Fall A liegen alle Vertices innerhalb des View Frustum. Sie werden nicht entfernt. Das durch sie und die zugehörigen Edges aufgespannte Polygon bleibt erhalten. Das Dreieck B hingegen enthält Vertices innerhalb und außerhalb des View Frustum. Da die im Sichtbereich liegende Dreiecksspitze für eine korrekte Darstellung nicht verloren gehen darf, müssen anstelle der entfernten Vertices neue im View Frustum liegende Vertices erzeugt werden. Außerdem müssen die betreffenden Edges neu referenziert werden. Sollten zwei Vertices eines Dreiecks innerhalb des View Frustum liegen, so wird das sichtbare Viereck durch zwei Dreiecke substanziiert. Liegen, wie im Fall des Dreiecks C, alle Vertices eines Polygons außerhalb des View Frustum, können diese ersatzlos verworfen werden.

Auf die verbliebenen Vertices wird schließlich die perspektivische Division angewendet, welche in der Regel durch Hardware-Beschleunigung vollzogen wird. Die Koordinaten des normierten Vektors ${}_nv$ werden als Normalized Device Coordinates (NDC) bezeichnet.

$$ {}_cv \quad \xrightarrow{norm} \quad {}_nv = \begin{pmatrix} \frac{2n_c}{b_{out}} \frac{{}_ev_x}{{}_ev_z} \\ \frac{2n_c}{h_{out}} \frac{{}_ev_y}{{}_ev_z} \\ -\frac{2n_c f_c}{f_c - n_c} \frac{1}{{}_ev_z} + \frac{f_c + n_c}{f_c - n_c} \\ 1 \end{pmatrix} $$

Entsprechend des würfelförmigen View Frustums im Eye-Space liegen die $x$-, $y$- und $z$-Komponenten des Vektors ${}_nv$ im Intervall $[-1, 1]$. Während die $x$- und $y$-Komponenten im NDC-System einen linearen Bezug zu den unverzerrten Eye-Space-Koordinaten aufweisen, ist der Zusammenhang zwischen der $z$-Komponente und der tatsächlichen Tiefe des Vertex nichtlinear. Er wird in Bild 2-25 dargestellt.

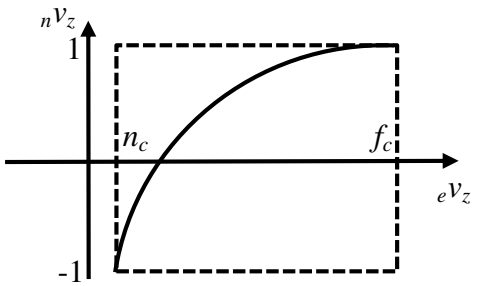


*Bild 2-25*: *Nichtlinearer Zusammenhang zwischen der tatsächlichen Tiefe ${}_ev_z$ eines Vertex in der Szene und der Tiefe ${}_nv_z$ in NDC-Koordinaten.*

Aufgrund des degressiven Verlaufs nimmt die Auflösung der Tiefeninformation bei gegebener Bitmenge mit zunehmendem Abstand zur Kamera ab. Das hat einerseits den Vorteil, dass Objekte im Nahbereich der Kamera auch bei kleinen Tiefenunterschieden noch korrekt dargestellt werden. Andererseits ist nachteilig, dass sich weit entfernte und nah beieinander liegenden Flächen aufgrund einer zu geringen Auflösung nicht mehr korrekt überdecken. Dieser Zusammenhang macht sich häufig als „Flackern“ bemerkbar, da schon bei minimalen Veränderungen der Kameralage mal die eine und mal die andere Fläche als sichtbar dargestellt wird. Man bezeichnet diesen Effekt als „Depth Fighting“.

Im letzten Schritt des Vertex Processing erfolgt der Wechsel vom NDC-System auf die Pixelkoordinaten des Ausgabegeräts (Viewport Transformation). Auch hierzu kann eine geeignete Matrix gefunden werden. Die sogenannte Scale&Bias-Matrix $T_{SB}$ skaliert die NDC-Koordinaten einerseits von den normierten Werten auf die Anzahl der verfügbaren Rasterpunkte und verschiebt außerdem alle Koordinaten in den positiven Bereich. Der entstehende Vektor ${}_sv$ liegt nun im Screen-Space vor. Bild 2-26 zeigt beispielhaft das

Resultat der Viewport Tranformation, wie es aus der in Bild 2-24 dargestellten Situation hervorgehen würde.

$$ {}_s v = T_{SB} \cdot {}_n v \text{ mit } T_{SB} = \begin{pmatrix} \frac{b_{out}}{2} & 0 & 0 & \frac{b_{out}}{2} \\ 0 & \frac{h_{out}}{2} & 0 & \frac{h_{out}}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \rightarrow {}_s v = \begin{pmatrix} {}_s v_x \in [0, b_{out}] \\ {}_s v_y \in [0, h_{out}] \\ {}_s v_z \in [0, 1] \\ {}_s v_w = 1 \end{pmatrix} $$

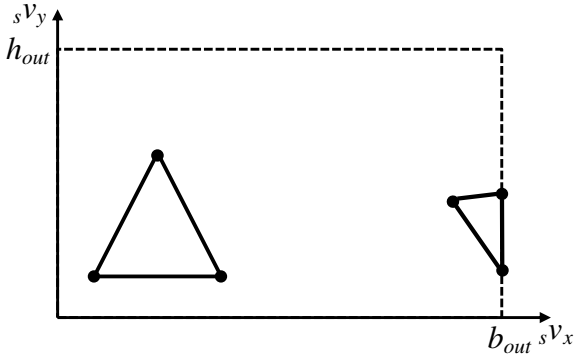


*Bild 2-26*: *Polygone nach der Transformation in den Screen-Space.*

Das Bild 2-27 fasst die im Vertex Processing stattfindenden Transformationen noch einmal zusammen.

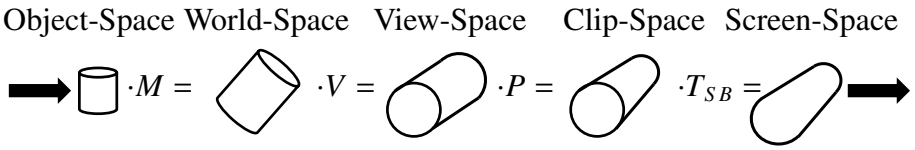


*Bild 2-27*: *Transformationsfolge im Vertex Processing.*

### Primitve Assembly

Innerhalb des Vertex Processing werden die zu verarbeitenden Vertices weitgehend isoliert voneinander betrachtet. Nach der Verarbeitung liegen die Vertices im Screen-Space vor. Ziel ist jedoch nicht die Darstellung der Vertices, sondern der durch sie beschriebenen Polygone. Um diese zu rekonstruieren, werden in der Primitve Assembly-Stufe gemäß Bild 2-28 Vertex-Listen zu geometrischen Primitiven zusammengefasst.

Als Eingabe erwartet diese Pipeline-Stufe zum einen die Vertices, welche zuvor durch das Vertex Processing modifiziert wurden, und zum anderen die Konnektivitätsinformationen zwischen den Vertices. Auf Basis dieser Informationen werden Primitive gebildet, welche die Gestalt eines Punkts, einer Linie oder eines Dreiecks haben können.

Außerdem greift hier ein weiteres Verfahren, um nicht relevante Elemente zu entfernen und so den Vertex- sowie den Pixel-Load (Anzahl der insgesamt zu verarbeitenden Vertices bzw.

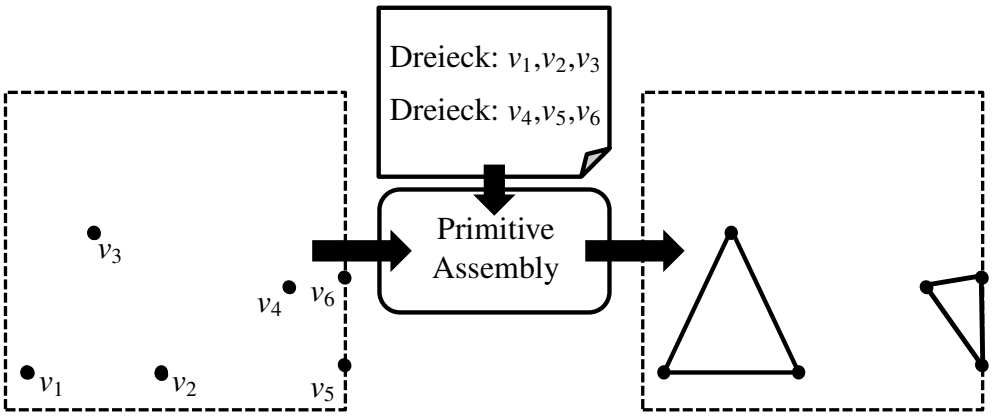


*Bild 2-28*: *Primitve Assembly: Konstruktion geometrischer Primitive aus einzelnen Vertices.*

Pixel) für nachgeschaltete Pipeline-Schritte zu reduzieren. Der als Face Culling bezeichnete Ansatz entfernt die Vertices aller Flächenprimitive, die nicht zur Kamera orientiert sind. Die Entscheidungsgrundlage stellen hierbei die Flächennormalen dar, welche aus zwei Kanten gebildet werden können. Ist das Skalarprodukt aus Flächennormale und einem von der Fläche zur Kamera zeigenden Vektor positiv, so ist die Fläche sichtbar und muss gerendert werden. Im anderen Fall wird das Primitiv verworfen. Stellt man sich beispielsweise einen Würfel vor, so kann dessen Rendering durch Culling auf maximal drei Flächen reduziert werden, da die übrigen Würfelflächen nicht sichtbar sind.

### Rasterization und Coloring

Im nächsten Schritt wird von dem bisher als kontinuierlich angenommenen Screen-Space auf das diskrete Raster technischer Ausgabegeräte übergegangen. Ausgangspunkt stellen die zuvor berechneten Primitive dar.

Zunächst gilt es herauszufinden, welche Pixel in ihrer Einfärbung potentiell durch das aktuell betrachtete Primitiv bestimmt werden könnten. Aufgrund von Überdeckungen und anderen Details ist jedoch zum Zeitpunkt der Rasterung noch nicht abschließend klar, ob und in wie weit die Farbe des entsprechenden Pixels durch das aktuelle Primitiv bestimmt wird. Man spricht deshalb in dieser Pipeline-Stufe noch nicht von Pixeln sondern Fragmenten. Sie können als Kandidaten für Pixel verstanden werden. Wie genau die verschiedenen Primitive in Fragmente zerlegt werden, hängt von dem jeweiligen Primitivtyp ab. Da die grundlegende Idee unverändert bleibt, soll hier beispielhaft die Rasterung eines Dreiecks vorgestellt werden. Bild 2-29 zeigt die dafürnotwendigen Schritte.

Im Schritt 1 werden die Vertices des betrachteten Primitivs auf diskrete Pixelpositionen überführt (z.B. durch Runden). Dabei vererben die Vertices ihre Attribute (Farbe, Tiefe, Normale, Texturkoordinaten, ...) an die entstehenden Fragmente. Als Nächstes werden die Kanten des Dreiecks gerastert (siehe Schritt 2 in Bild 2-29). Hierzu können verschiedene rechenzeitoptimierte Algorithmen eingesetzt werden. Bekannte Beispiele sind Bresenham oder Midpoint [Bre65], [Pit67]. Schließlich müssen noch die Innenbereiche des Dreiecks

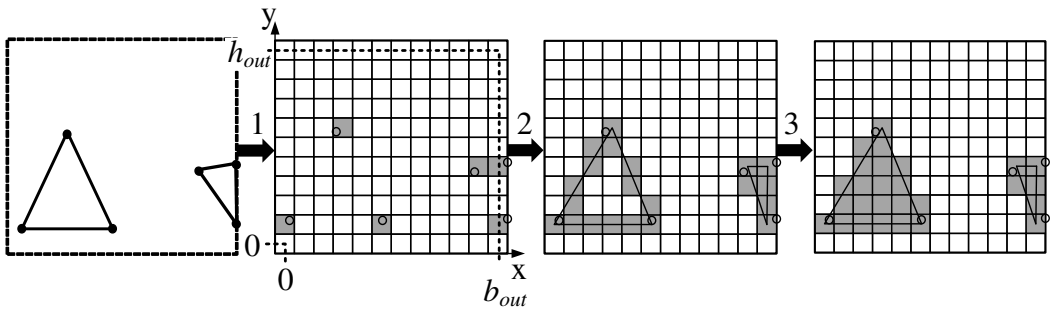


*Bild 2-29*: *Rasterization: Bestimmung der durch ein Primitiv potentiell beeinflussten Pixelpositionen (Fragmente).*

gerastert werden, wofür ebenfalls verschiedene Filling-Ansätze existieren. Während der Rasterung der Dreiecksfläche, werden die Attribute der in Schritt 1 erzeugten Fragmente bezüglich der Distanz auf die übrigen Rasterpunkte interpoliert. Das Bild 2-30 stellt die bilineare Interpolation durch das Scanline-Verfahren dar [Bou70].

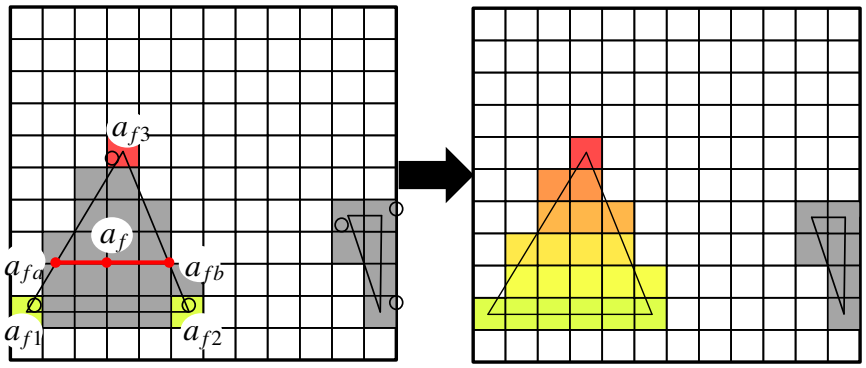


*Bild 2-30*: *Zuweisung der Attribute an die Fragmente eines Primitivs durch bilineare Interpolation der Vertex-Attribute.*

In der vorgestellten Variante erhalten zunächst die Fragmente auf den Kanten ihre Attribute, indem zwischen den zugehörigen Endpunkten linear bezüglich der Distanz interpoliert wird. Betrachtet man die Kante mit den Endpunkten 1 und 3, wobei $a_{f1}$ und $a_{f3}$ ein zu interpolierendes Attribut (z.B. Farbe, Texturkoordinaten) dieser Punkte bezeichnen, so ergibt sich das Attribut $a_{fa}$ eines Rasterpunkts auf dieser Kante gemäß

$$a_{fa} = c_{int} \cdot a_{f1} + (1 - c_{int}) \cdot a_{f3}. \tag{2-27}$$

Dabei ist der Parameter $c_{int}$ der Quotient der Distanzen $\overline{a_{fa}a_{f3}}$ und $\overline{a_{f1}a_{f3}}$. Nachdem über alle Kanten interpoliert wurde, wird im Scanline-Verfahren eine erneute zeilenweise Interpolation angewendet. In Bild 2-30 liegt das zu bestimmende Fragmentattribut $a_f$ beispielsweise auf einer Zeile, die durch die zuvor berechneten Randpunktattribute $a_{fa}$ und $a_{fb}$ bestimmt wird. $a_f$ kann analog zu Gleichung (2-27) bestimmt werden.

In Bild 2-30 wird die Interpolation von Farbwerten über die Fragmente visualisiert. Für detaillierte Visualisierungen von Oberflächen genügt diese Methode jedoch nicht oder

würde eine Vielzahl von Primitiven erfordern, ohne dass sie aus geometrischer Sicht erforderlich wären. Deutlich häufiger werden deshalb Texturen zur Färbung der Fragmente eingesetzt. Anstelle der Farbwerte werden dann die Texturkoordinaten bei der Rasterung interpoliert. Zur besseren Nachvollziehbarkeit wird die Verknüpfung zwischen Textur und Polygon in Bild 2-31 veranschaulicht.

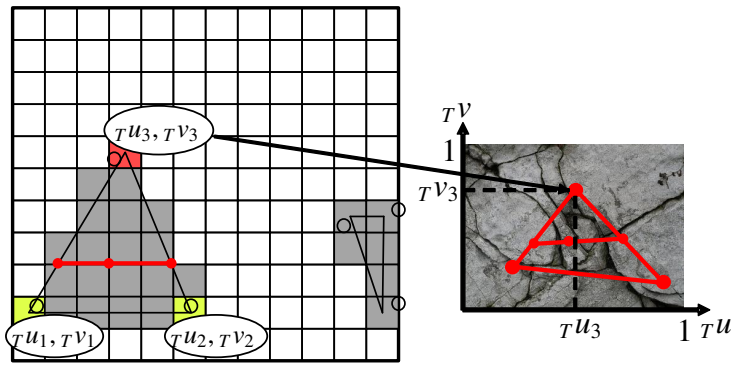


*Bild 2-31*: *Einfärbung der Fragmente durch Texturen unter Berücksichtigung von Texturkoordinaten.*

Im rechten Bereich ist eine Steintextur dargestellt, die beispielsweise zur Texturierung eines Felsen in der virtuellen Szene eingesetzt werden kann. Zur Referenzierung der Textur werden normalisierte Koordinaten (*uv*-Koordinaten) eingesetzt. Jedes Modell einer dreidimensionalen Szene kann mit einer Textur verknüpft werden. Den Vertices des Modells können *uv*-Koordinaten zugewiesen werden, die dann auf Punkte in der Textur referenzieren. In Bild 2-31 wird dieser Zusammenhang für die Texturkoordinaten ($_T u_3$,$_T v_3$) visualisiert. Durch die Rasterung bekommt jedes Fragment eines Primitivs geeignete Texturkoordinaten. Zur Einfärbung des Fragments wird schließlich die Farbe der Textur an der Stelle, die durch die *uv*-Koordinaten des Fragments spezifiziert ist, ausgelesen und als Fragmentfarbe übernommen. Je nach Wahl der *uv*-Koordinaten für die Vertices kann der Texturausschnitt, der schließlich auf dem Modell sichtbar ist, erheblich verzerrt und gedreht sein.

### Raster Operations

Wie im vorhergehenden Unterabschnitt bereits erwähnt, sind Fragmente Kandidaten für die Pixel des Frame Buffers. In den Raster Operations wird entschieden, welche Fragmente auf welche Weise zum finalen Pixel im Frame Buffer beitragen. Der Frame Buffer besteht aus verschiedenen Buffern, die eine einheitliche Größe aufweisen und unterschiedliche Informationen der Pixel verwalten. Der wichtigste Buffer ist der Color Buffer. Er beinhaltet die Pixelfarben und stellt somit den Anteil der Pixelinformationen dar, welcher durch das Ausgabegerät visualisiert wird. Daneben existiert der Depth Buffer. Dieser speichert für die aktuell im Frame Buffer befindlichen Pixel die Tiefen. Auf seiner Grundlage wird entschieden, ob ein neu eingeleitetes Fragment den Pixel an seiner Position überschreibt oder ob es verworfen wird. Bedingung für die Übernahme des Fragments ist, dass sein Tiefenwert geringer als die Tiefe des derzeit im Frame Buffer befindlichen Pixels ist. Damit

ist sichergestellt, dass sich das Fragment näher an der Kamera befindet und den derzeit im Frame Buffer existierenden Wert überdeckt. Diese Überprüfung wird auch als Depth Test oder z-Test bezeichnet. Zur Sicherstellung der korrekten Funktion ist es wichtig, dass die Werte im Depth Buffer vor der Berechnung des nächsten Frames auf das Maximum gesetzt werden. Auf diese Weise führt der Depth Test für das erste Fragment auf einer bestimmten Pixelposition stets zu einem positiven Ergebnis (Fragment wird in Pixel geschrieben). Der Frame Buffer wird zu Beginn mit einer benutzerdefinierten Clear-Color initialisiert. Im Bild 2-32 werden die verschiedenen Raster Operationen und ihre Wechselwirkungen mit dem Frame Buffer dargestellt. Neben dem Depth Test kann eine weitere als Stencil Test

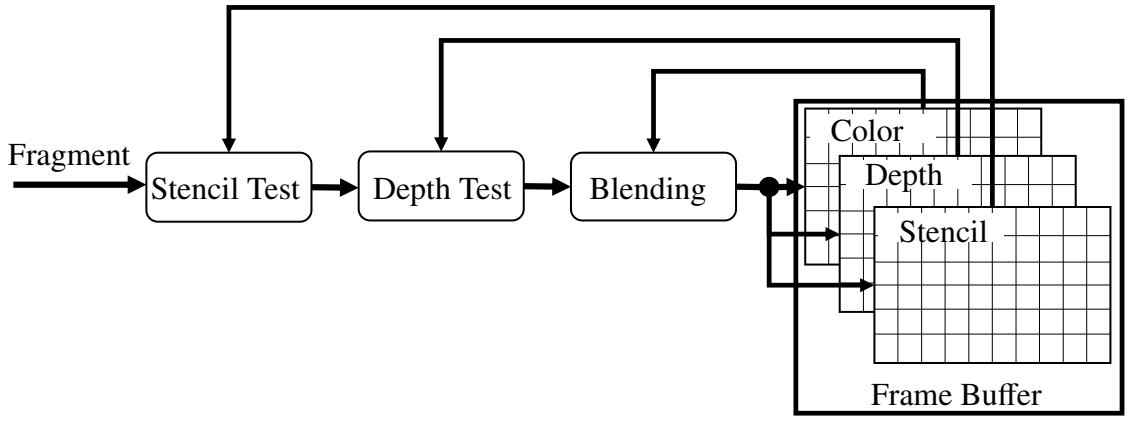


*Bild 2-32*: *Raster Operations: Übernahme der Fragmente für die Pixel des Frame Buffers.*

bezeichnete Prüfung durchgeführt werden. Dazu existiert neben Color- und Depth- ein Stencilbuffer, der pro Pixel einen Stencilwert enthält. Abhängig von der Belegung des Stencil Buffers kann die Übernahme von Fragmenten in den Framebuffer auf bestimmte Eigenschaften der Stencilwerte beschränkt werden. Die konkrete zu prüfende Bedingung ist anders als beim Depth Test manipulierbar und wird als Stencilfunction bezeichnet. Genauso kann die Stenciloperation angepasst werden, welche bei der Verarbeitung eines Fragments darüber entscheidet, wie der Stencilwert angepasst werden soll. Ein einfaches Beispiel für die Anwendung des Stencilbuffer ist die Begrenzung des Rendergebiets. So könnte der statische Fahrzeuginnenraum ausmaskiert werden, sodass nur der durch die Frontscheibe sichtbare Bereich der Szene gerendert wird.

Schließlich gibt es noch die Möglichkeit des Blendings. Im Gegensatz zu den bisher aufgeführten Raster Operationen handelt es sich beim Blending nicht um einen Test. Stattdessen bietet Blending neben dem Verwerfen oder der Übernahme eines Fragments in den Frame Buffer die Möglichkeit, sowohl das zu verarbeitende Fragment als auch den derzeitigen Pixel in die neue Belegung des Color Buffers einfließen zu lassen. Dabei können die Farbwerte additiv, multiplikativ oder auch abhängig vom Transparenzwert (Alpha Blending basierend auf viertem Farbkanal) überlagert werden. Mittels Blending wird beispielsweise das Rendering transparenter Objekte oder Nebel (Fog Blending) umgesetzt.

### 2.3.2 Programmable Pipeline

Die Entwicklung der Rendering-Ansätze verlief Hand in Hand mit den Fortschritten in der Hardware-Entwicklung. Die erste Hardware-Unterstützung für Rasteranzeigen, wie sie bis heute das Funktionsprinzip von Ausgabegeräten darstellen, erschien in den 80er Jahren. Diese waren jedoch ausgesprochen teuer und wurden nur im akademischen und gewerblichen Umfeld eingesetzt. Erst Mitte der 90er Jahre hielten Grafikkarten mit Produkten von IBM, ATI oder Intel Einzug in die Computer privater Anwender. Ihre Architektur bildete über mehrere Zwischenschritte die in Abschnitt 2.3.1 vorgestellte Fixed Function Pipeline ab. Im selben Zeitraum entwickelten sich die APIs OpenGL und Direct3D zu den Standard-Abstraktionsschichten der hersteller- und modellspezifischen Schnittstellen der Grafikhardware und boten den Anwendungsentwicklern auf diese Weise einen intuitiven Zugang zu den neuen Möglichkeiten. Von nun an schritten die Leistungsfähigkeit und Flexibilität der Grafikkarten und APIs, einerseits bedingt durch die erheblichen Fortschritte in der Hardware-Entwicklung und andererseits getrieben durch die ständig wachsenden Anforderungen seitens der Anwendungsentwicklung, rasant voran. Den größten Treiber stellte dabei die Spiele-Industrie dar, die mit jedem neuen Produkt auch durch neue grafische Effekte glänzen wollte.

Der zunehmende Trend der Individualisierung brachte die Fixed Function Pipeline an ihre Grenzen. Wenn sich ihr Verhalten auch an verschiedenen Stellen konfigurieren ließ, war ihre Architektur in Hardware gegossen und ließ deshalb keine Anpassungen zu. Um den steigenden Anforderungen gerecht zu werden, erschienen 2001 gemeinsam mit den API-Standards OpenGL 1.4 und Direct3D 8 Grafikkarten, die an verschiedenen Stellen der Verarbeitungskette alternative Zweige zuließen, die unter Einhaltung der Schnittstellendefinitionen durch eigenen Programmcode spezifiziert werden konnten. Aufgrund der Koexistenz der Fixed Function Pipeline und der programmierbaren Ausführungseinheiten wird diese Generation auch hybride Pipeline genannt. Bezeichnet werden die Programme, die innerhalb der hybriden Pipeline durchlaufen werden können, angelehnt an ihren primären Einsatzzweck zur Bestimmung von Pixelfarben für die grafische Ausgabe, als Shader (to shade: schattieren, abtönen). Tatsächlich ist diese Bezeichnung jedoch irreführend, da sie nicht auf diese Aufgabe begrenzt, sondern vielfältig einsetzbar sind. Großflächigen Einsatz erfuhr die Shader-Technologie mit dem Aufkommen der Hochsprachen HLSL/Cg (Zusammenarbeit von Microsoft und NVIDIA) und GLSL (Konsortium zur OpenGL-Spezfikation), wodurch Shader einerseits wesentlich besser lesbar wurden und andererseits an Plattformunabhängigkeit gewonnen haben [FK03], [KBR17]. Eingesetzt wurden Shader in dieser Einführungsphase typischerweise zur Realisierung spezieller Effekte, während der Großteil der Szene weiterhin durch die Fixed Function Pipeline gerendert wurde.

Erwartungsgemäß wurde der Anteil von Individuallösungen zur Darstellung spezieller Effekte mit den Jahren immer größer. Insbesondere die zunehmende Rechenleistung und der ständig wachsende Funktionsumfang der Grafikkarten-Befehlssätze und APIs führte zu einer so intensiven Nutzung der Shader-Technologie, dass die Fixed Function Pipeline seit 2009 mit den API-Versionen OpenGL 3.2 und Direct3D 10 aus der Architektur entfernt wurde. In der programmierbaren Pipeline existieren für das Vertex- und Fragment-Processing keine in Hardware realisierten Pipeline-Stufen mehr. Um das Rendering von Anwendungen, die für diese Stufen keine eigenen Shader definieren, weiterhin zu unter-

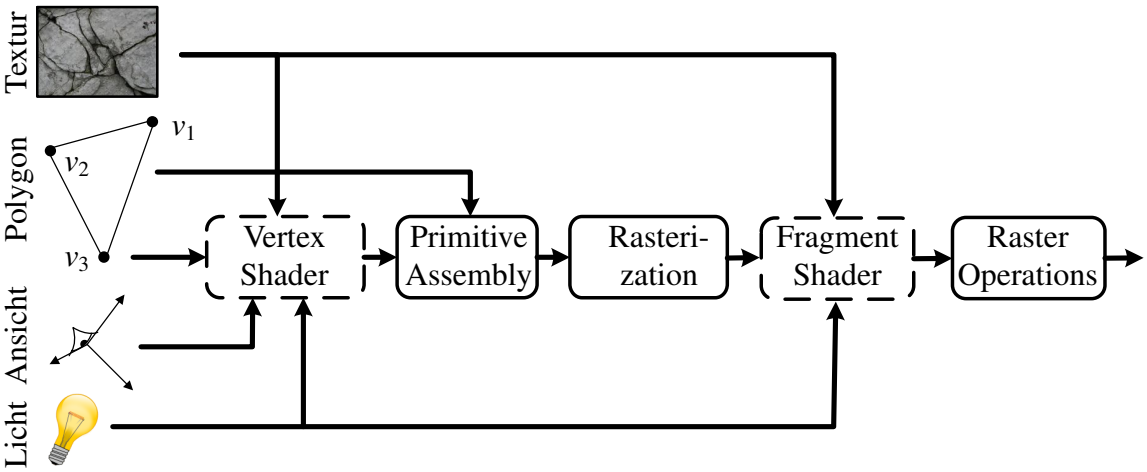


*Bild 2-33*: *Programmierbare Rendering Pipeline.*

stützen, stellen die APIs sogenannte Kompatibilitätsprofile bereit, welche die zuvor durch Hardware realisierten Verarbeitungsschritte in Shadercode abbilden.

Im Folgenden soll auf die Struktur der Vertex und Fragment Shader detaillierter eingegangen werden, da sie im weiteren Verlauf der Arbeit eine wesentliche Rolle spielen werden. An dieser Stelle sei noch erwähnt, dass es weitere Shader gibt, welche in die Pipeline eingreifen können. Ihre Verwendung ist im Gegensatz zu den beiden genannten jedoch optional. Zum einen handelt es sich hierbei um den Geometry Shader. Dieser ist nachgelagert zum Vertex Shader, operiert auf einzelnen Primitiven und kann eine (in Grenzen) beliebige Menge von Primitiven ausgeben, die im weiteren Pipeline-Verlauf der Rasterung zugeführt wird. Zwischen Vertex- und Geomtery Shader kann außerdem ein Tessellation Shader eingebunden werden. Durch diesen können die Primitive in kleinere Primitive zerlegt werden, wodurch beispielsweise eine adaptive Detaillierung von Szenenmodellen implementiert werden kann. Geometry- und Tesselation Shader werden im Bild 2-33 nicht berücksichtigt und im Folgenden nicht weiter diskutiert.

### Vertex Shader

Den Eingang für den Vertex Shader stellt ein einzelner Vertex dar. Dieser umfasst in jedem Fall die Position im Model Space und darüber hinaus meist verschiedene weitere Informationen. Typische Beispiele hierfür sind Farbe, Texturkoordinaten, Normale, aber auch oft individuell wählbare weitere Daten, die für die spezifische Aufgabe des Shaders bzw. seines nachgelagerten Fragment Shaders von Bedeutung sind. Neben diesen vertexspezifischen Daten kann auf sogenannte Uniform-Variablen zugegriffen werden, die für alle Ausführungsstränge des Shaders und somit für alle Vertices des aktuell zu rendernden Objekts gleich sind. Seit dem Shader-Modell 3.0 (DirectX 9.0c) können im Vertex Shader auch Daten aus Texturen ausgelesen werden, die ebenfalls für alle Vertices gleich sind, jedoch typischerweise abhängig vom aktuellen Vertex an verschiedenen Stellen gesampelt werden. Die klassische Aufgabe des Vertex Shaders ist die Tranformation der Vertex Position aus dem Model Space in den Clip Space, wie es im Abschnitt 2.3.1 ausführlich dargestellt wird.

Weitere Berechnungen ergeben sich aus der entsprechenden Aufgabenstellung und müssen nicht zwangsläufig einen Bezug zum klassischen Vertex Processing der Fixed Function Pipeline haben.

Auch wenn die Möglichkeiten eines programmierbaren Vertex Shaders vielfältig sind, unterliegen alle Realisierungen einer wichtigen Restriktion: der exklusiven Berechnung des jeweiligen Vertex. Beispielsweise können im Vertex Shader nicht andere Vertices des selben Polygons referenziert werden (hier schafft der Geometry Shader Abhilfe). Diese Einschränkung ist essentiell für die hohe Parallelisierbarkeit und damit für das echtzeitfähige Rendering komplexer Szenen. Auf diese Weise können große Mengen von Vertices auf verschiedenen Rechenkernen der Grafikkarte parallel abgearbeitet werden. Die GPU nutzt genau diese Eigenschaft von Shader Programmen aus, indem sie über erheblich mehr Rechenkerne als die CPU verfügt (GEFORCE RTX 3090: 10496 CUDA Kerne [Nvi20], AMD Ryzen Threadripper: 32 Kerne [Adv20]). Die Ausführung einer Instanz des Vertex Shaders auf einem einzelnen Kern geschieht dabei nach der in Bild 2-34 dargestellten Logik. Initial werden die Vertex Daten in das Input Register geschrieben, auf welches während der Ausführung des Shadercodes ausschließlich Lesezugriff gewährt wird. Bei der Abarbeitung der Befehlssequenz kann das temporäre Register zum Zwischenspeichern von Ergebnissen verwendet werden. Zugriffe auf Texturen sind im Vertex Shader unüblich, weshalb diese in Bild 2-34 nicht visualisiert werden. Die Rückgabedaten des Shaders werden während des Durchlaufs in das Output Register geschrieben. Auf dieses besteht ausschließlich Schreibzugriff.

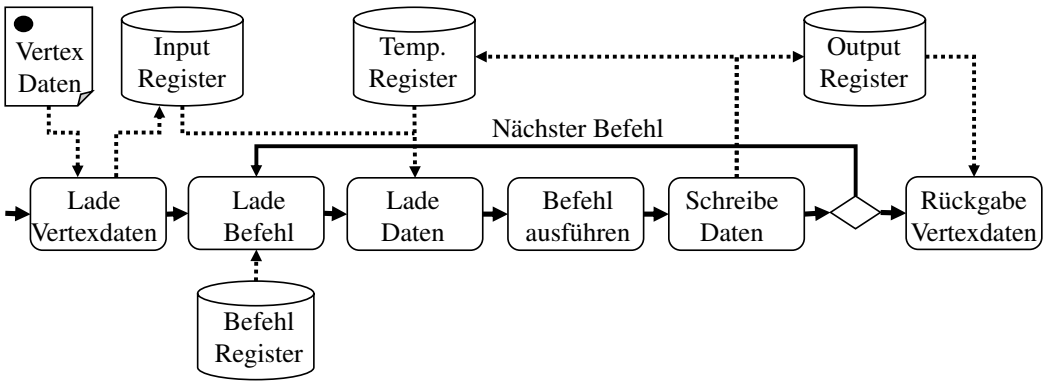


*Bild 2-34*: *Ablaufdiagramm des Vertex Shaders.*

Die mindestens notwendige Rückgabe des Vertex Shaders ist der in den Clip Space transformierte Vertex. Darüber hinaus werden typischerweise die transformierte Normale, Texturkoordinaten oder weitere für den Einsatzzweck dienliche Informationen zurückgegeben. Bezugnehmend auf das in Abschnitt 2.3.1 vorgestellte Vertex Processing der Fixed Function Pipeline fehlen noch einige Arbeitsschritte, bevor die Rasterung durchgeführt werden kann. Konkret sind das Clipping, Culling und die perspektivische Division. Diese sind in der Programmable Pipeline aus Performance-Gründen weiterhin durch Hardware realisiert und werden in der OpenGL-Pipeline unter dem Begriff „Vertex-Postprocessing“ zusammengefasst. In der anschließenden Rasterung werden alle Rückgabewerte des Vertex Shaders auf die verschiedenen Rasterpunkte entsprechend der Gleichung (2-27) interpoliert und dienen so als Eingabe für den nachgelagerten Fragment Shader.

### Fragment Shader

Für jeden Rasterpunkt, der durch das aktuell zu rendernde Objekt überdeckt wird, schließt sich ein Durchlauf des Fragment Shaders an. Im direkten Vergleich zum Vertex Shader wird der Fragment Shader gemäß Bild 2-35 auf ähnliche Weise abgearbeitet. Anstelle der Vertex Daten werden Fragment Daten in das Input Register geladen. Diese beinhalten die Interpolationsergebnisse aller durch den Vertex Shader zurückgegeben Vertex Daten für das im Fokus stehende Fragment. Klassischer Weise finden innerhalb der Befehlssequenz des Fragment Programms auch Texturzugriffe statt. Die auszulesende Stelle der Textur wird durch die interpolierten Texturkoordinaten (siehe Bild 2-31) referenziert. Diese werden typischerweise in einem kontinuierlichen Wertebereich und normiert vorgegeben. Die Pixel der Textur (Texel) haben jedoch diskrete Positionen. Im Allgemeinen wird der Rückgabewert bei einem Texturzugriff also anhand der umliegenden Texel durch Interpolation errechnet. Man spricht in diesem Zusammenhang auch von Filterung oder Sample.

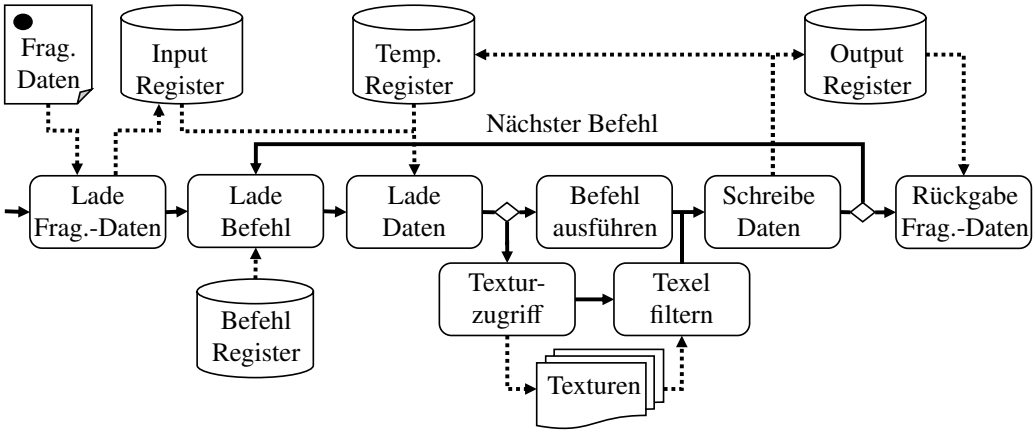


*Bild 2-35*: *Ablaufdiagramm des Fragment Shaders.*

Rückgabe des Fragment Shaders ist die finale Farbe des Fragments und mittelbar die Farbe des an derselben Position befindlichen Pixels des Ausgabegeräts. Nicht zwingend, jedoch häufig, wird zudem der Tiefenwert des Fragments zurückgegeben, sodass im Anschluss eine Überdeckungsprüfung (siehe „Depth Test“ in Abschnitt 2.3.1) erfolgen kann. Nachdem alle Objekte der Szene die Rendering Pipeline durchlaufen haben, enthält der Frame Buffer die darzustellenden Fragmente bzw. Pixel, welche nun auf dem Ausgabegerät angezeigt werden.

## 2.3.3 Forward vs. Deferred Rendering

Das in den vorhergehenden Abschnitten beschriebene Renderingverfahren wird als Forward Rendering bezeichnet. Es ist das meist und bis vor einigen Jahren ausschließlich genutzte Verfahren. Kennzeichnend für das Forward Rendering ist, dass Polygon für Polygon bis zu seiner Darstellung auf dem Frame Buffer verarbeitet wird. Häufig werden neu verarbeitete Polygone bereits gerenderte Polygone verdecken. Die korrekte Überlagerung

wird in letzter Instanz durch den Z-Test für jedes Fragment sichergestellt. Aus rechentechnischer Sicht nachteilig ist hierbei, dass verdeckte Fragmente von Polygonen die gesamte Rendering Pipeline durchlaufen, obwohl sie schlussendlich nicht zur Ausgabe beitragen. Die damit einhergehende Ressourcenverschwendung kann abhängig von der Szenengestaltung, den existierenden Lichtern und den Kameraeinstellungen sehr groß sein. Der größte Rechenaufwand entsteht im Fragment Shader, da die Anzahl der Fragmente eines Polygons je nach Größe und Entfernung zur Kamera erheblich größer als die Anzahl der Vertices ist. Innerhalb des Fragment Shaders besteht die Kernaufgabe in der Anwendung des Beleuchtungsmodells (Per-Fragment-Lighting).

Deferred Rendering greift diese Überlegung auf und nimmt eine Neustrukturierung der Rendering Pipeline vor, welche die beschriebene Ressourcenverschwendung gezielt vermeidet. Zur Diskussionsgrundlage stellt Bild 2-36 die Strukturen der Forward und Deferred Rendering Piplines gegenüber.

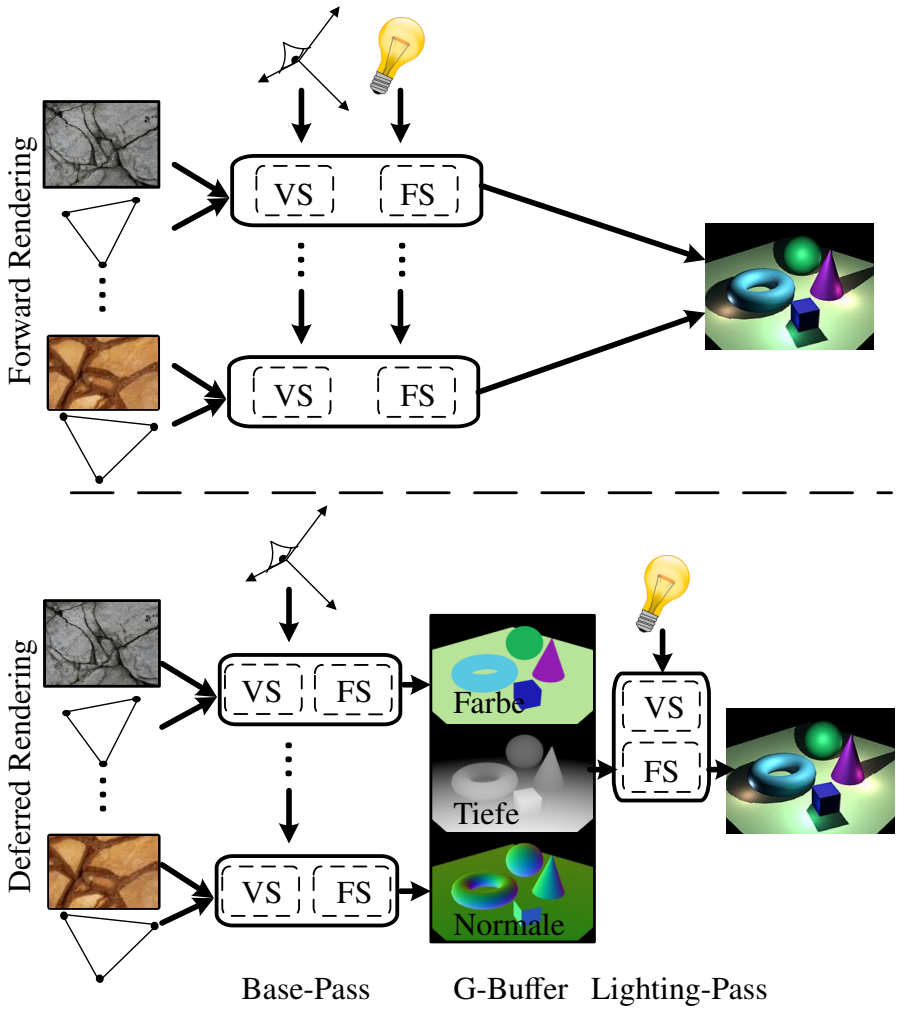


*Bild 2-36*: *Gegenüberstellung der Forward und Deferred Rendering Piplines.*

Im Deferred Rendering wird ein Zwischenschritt eingefügt. Anstatt jede Geometrie isoliert von allen anderen in den Frame Buffer zu rendern, werden die Geometrien im Deferred

Rendering zunächst in den sogenannte G(eometry)-Buffer abgebildet. Bei diesem ersten Schritt, der auch als Base-Pass bezeichnet wird, wird die komplexe Szene bereits auf ihr zweidimensionales Abbild im Screen Space überführt. Genau wie im Forward Rendering verbleiben nur die Fragmente im G-Buffer, welche später tatsächlich als Teil der Ausgabe sichtbar sind. Verdeckte Fragmente werden durch den Z-Test eliminiert. Der namensgebende Unterschied ist, dass Beleuchtungsberechnungen an dieser Stelle noch nicht erfolgt sind, sondern nachgelagert im sogenannten Lighting-Pass vorgenommen werden. Der rechentechnische Aufwand ist deshalb bei der Erstellung des G-Buffers deutlich geringer als im Forward Rendering.

Die Beleuchtung muss in einem zweiten Schritt erfolgen. Vorteilhaft hierbei ist, dass das Beleuchtungsmodell nur noch auf die sichtbaren Fragmente angewendet wird. Der G-Buffer fungiert somit als Filter, welcher schon vor aufwendigen Beleuchtungsberechnungen alle nicht sichtbaren Fragmente verwirft. Da das Beleuchtungsmodell erst im Screen Space angewendet wird und nicht wie üblich in der dreidimensionalen Szene, erfordert das Konzept des Deferred Rendering die Verwendung mehrerer Render Targets. Als Render Target wird eine, meist zweidimensionale, Datenstruktur (Buffer) bezeichnet, in welche das Rendering-Ergebnis geschrieben wird. Neben den Farbinformationen der Fragmente müssen im Deferred Rendering parallel weitere Informationen, wie der Tiefenwert und die Aurichtung der Flächennormale, vorgehalten werden. Seit OpenGL 2.0 und Direct3D 9 unterstützen Grafikkarten das gleichzeitige Rendern verschiedener Informationen in verschiedene Buffer. Bezeichnet wird diese Funktion als „Multiple Render Targets“ (MRT). Der G-Buffer hält alle für das Beleuchtungsmodell relevanten Informationen fragmentspezifisch vor. Im Lighting-Pass werden die Informationen des G-Buffers zusammengeführt, um die finale Ausgabe zu ermitteln. Technisch wird dazu ein fiktiver Volumenkörper, das sogenannte Lichtvolumen, durch einen Lighting Shader gerendert. Dieses Lichtvolumen beinhaltet den Raum der Szene, in dem das Licht potentiell Einfluss nehmen kann. Ziel des Lighting Shaders ist jedoch nicht die Visualisierung des Licht-Volumens. Stattdessen dient es nach der Transformation im Vertex Shader und der Rasterung als Selektionsbereich innerhalb des G-Buffers und markiert die Fragmente, auf welche das Beleuchtungsmodell der betrachteten Lichtquellen angewendet werden muss. Die Beleuchtungsberechnungen erfolgen schließlich im Fragment Shader des Lighting-Pass. Die Abläufe im Lighting-Pass erfolgen für alle Lichtquellen in der Szene und ergeben additiv das finale Szenenbild.

Ein Nachteil des Deferred Rendering ist einerseits, dass es nicht auf jeder Hardware unterstützt wird. Hierbei ist vor allem der Bereich mobiler Endgeräte hervorzuheben. Es ist allerdings davon auszugehen, dass die Durchdringung in den nächsten Jahren auch dieses Marktsegment erreichen wird. Andererseits ist Deferred Rendering durch die zusätzlichen Arbeitsschritte nicht prinzipiell schneller als Forward Rendering. Grundsätzlich kann man jedoch festhalten, dass Deferred Rendering bei der Verwendung vieler dynamischer Lichtquellen eine höhere Performance erreicht. Während sich für das Forward Rendering bei $m$ Objekten und $n$ Lichtquellen innerhalb der Szene ein asymptotischer Aufwand von $O(m \cdot n)$ ergibt, kann dieser durch das Deferred Rendering auf $O(m+n)$ reduziert werden.

### 2.3.4 Beleuchtungsmodelle

Bei den bisherigen Ausführungen zum Rendering wurden die Einflüsse von Lichtquellen ausgeklammert. Aufgrund ihrer besonderen Relevanz im betrachteten Kontext sollen sie in diesem Abschnitt gesondert beschrieben werden. Verfahren, die in der 3D-Computergrafik zur Simulation von Licht eingesetzt werden, bezeichnet man als Beleuchtungsmodelle. Man unterscheidet dabei grundsätzlich zwischen lokalen und globalen Ansätzen. Bevor diese Typen thematisiert werden, wird vorgelagert die Rendergleichung eingeführt. Sie stellt die mathematische Basis aller Beleuchtungsalgorithmen dar. Den Abschluss dieses Abschnitts bildet das High Dynamic Range Rendering, mit welchem der Versuch unternommen wird, die hohe Dynamik realer Beleuchtung auf einem technischen Wiedergabegerät möglichst unverzerrt darzustellen.

#### Rendergleichung

Die Rendergleichung dient in der Computergrafik als Grundlage zur Berechnung globaler Beleuchtung und wurde 1986 zeitgleich von Jim Kajiya und David Immel et al. eingeführt [Kaj86], [ICG86]. Es handelt sich um eine Integralgleichung der Form

$$L_v(v,\Omega) = L_{v,e}(v,\Omega) + \int_{\Omega' \in \Omega^+} f_r(v,\Omega',\Omega) L_v(v,\Omega') \cos\theta_E \mathrm{d}\Omega', \quad v \in \mathbb{R}^3, \Omega \in \Omega^+ \quad \text{(2-28)}$$

und sie beschreibt die Leuchtdichte $L_v(v,\Omega)$ an einem Punkt $v$ in Richtung $\Omega$. Bild 2-37 visualisiert die Größen der Gleichung (2-28). Zur Ermittlung von $L_v(v,\Omega)$ werden die Leuchtdichten der Reflexionen in Richtung $\Omega$ aller Lichtstrahlen, die aus der Umgebung auf $v$ fallen, integriert. Diese erhält man durch die Anwendung des BRDF $f_r$ (siehe Abschnitt 2.1.5) auf die einfallenden Leuchtdichten $L_v(v,\Omega')$, welche je nach Einfallswinkel $\theta_E$ auf die effektive Fläche zu beziehen sind. Das Integral umfasst alle differentiellen Raumwinkelelemente $d\Omega'$ innerhalb der Hemisphäre $\Omega^+$ über dem Flächenelement bei $v$. Befindet sich $v$ selbst auf der Oberfläche einer Lichtquelle, so kann der Term $L_{v,e}(v,\Omega)$ genutzt werden, um die von der Lichtquelle bei $v$ emittierte Strahlung abzubilden.

Um den Einfluss von Farbe berücksichtigen zu können, müsste die Rendergleichung für jedes differentielle Wellenlängenintervall im sichtbaren Bereich ausgewertet werden. Sowohl die Leuchtdichte der bei $v$ einfallenden Lichtstrahlen, die von $v$ emittierte Strahlung als auch das durch $f_r$ abgebildete Reflexionsverhalten bei $v$ variieren wellenlängenabhängig. Die einzelnen Beiträge könnten schließlich addiert werden. Tatsächlich greift man hier jedoch wieder die Idee der Primärvalenzen von Farbräumen auf, mit denen visuelle Farbreize in guter Näherung durch nur drei Wellenlängen approximiert werden können. Man beschreibt Lichtfarben und Reflexionen also nur bezüglich der gegebenen Primärvalenzen (meist RGB) und reduziert das Integral der Rendergleichungen über den sichtbaren Wellenlängenbereich auf die Summe der Auswertungen an den drei Primärvalenzen.

#### Lichtquellen

Bevor im nachfolgenden Abschnitt beschrieben wird, wie Beleuchtungsmodelle zur Berücksichtigung von Lichteinflüssen in der virtuellen Szene eingesetzt werden können, greift

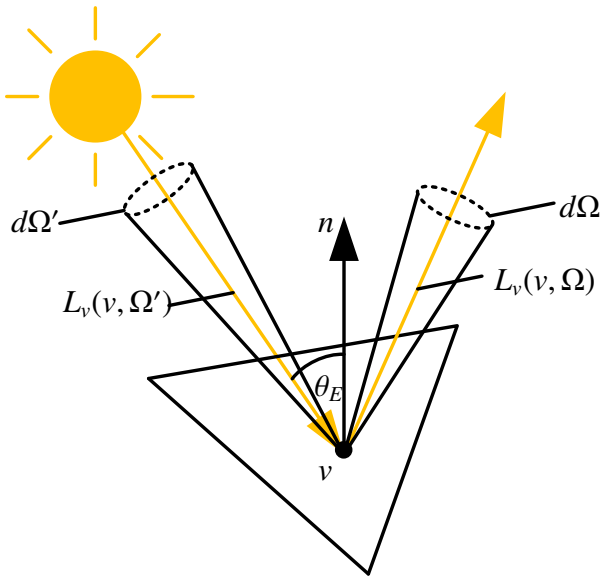


*Bild 2-37*: *Geometrische Zusammenhänge in der Rendergleichung.*

dieser Abschnitt zunächst die Modellierung der Lichtquellen innerhalb der Szene auf. Die etablierten Lichtquellen in der Computergrafik sind starke Vereinfachungen ihrer realen Vorbilder. Man unterscheidet grundsätzlich geometrische und globale Lichtquellen.

Eine bezeichnende Eigenschaft von geometrischen Lichtquellen ist ihre Position im Raum. Sie werden vereinfachend auf einen Punkt reduziert und besitzen keine räumliche Ausdehnung. Lichtquellen dieses Typs weisen außerdem, analog zur Realität, eine distanzabhängige Abschwächung ihrer Beleuchtungsstärke auf. Die klassischen Beispiele für geometrische Lichtquellen sind Punktlichtquellen (Bild 2-38 links) und Spotlichtquellen (Bild 2-38 Mitte). Eine Punktlichtquelle verfügt über eine homogene Lichtstärkeverteilung. Die Lichtstärke beträgt in alle Richtungen $I_{v,p}$. Der von ihr abgegebene Lichtstrom beträgt demnach $\Phi_{v,p} = 4\pi I_{v,p}$. Ein beleuchtetes Flächenelement $dA_E$ (je nach Rendering-Methode erfolgt die Diskretisierung in Vertices oder Fragmente) erfährt durch die Punktlichtquelle die Beleuchtungsstärke $E_v = \frac{d\Phi_{v,p}}{dA_E}$. Es ergibt sich für den auf $dA_E$ fallenden Lichtstrom $d\Phi_v = d\Omega_S I_{v,p}$ und somit für die Beleuchtungsstärke des Flächenelements nach dem photometrischen Entfernungsgesetz (2-13):

$$E_{v,p} = \frac{d\Phi_{v,p}}{dA_E} = \frac{I_{v,p} \cos\theta_E}{r^2}. \tag{2-29}$$

Eine Spotlichtquelle hingegen strahlt in einem Kegel aus, dessen Spitzenwinkel $\delta_s$ durch den Anwender vorgegeben werden kann. Dabei ist es möglich, die Lichtstärke von der Symmetrieachse des Kegels zu seiner Mantelfläche hin abzuschwächen, sodass ein sanfter Übergang von beleuchteten zu dunklen Flächen entsteht. Um diesen Sachverhalt mathematisch abzubilden, wird die in Gleichung (2-29) konstante Lichtstärke durch einen vom

Abstrahlwinkel $\delta$ abhängigen Term ersetzt. Es ergibt sich somit für die Beleuchtungsstärke $E_{v,s}$ unter Verwendung eines Spotlichts der Zusammenhang

$$E_{v,s} = \frac{I_{v,s} \cos\theta_E}{r^2} \text{ mit } I_{v,s} = \begin{cases} I_{v,s0} \cos^{n_s} \delta & |\delta| \leq \delta_s \\ 0 & sonst, \end{cases}$$

wobei $I_{v,s0}$ die Lichtstärke des Spotlichts entlang der Kegelmittelachse $d_s$ beschreibt. Durch den Parameter $n_s \geq 0$ kann beeinflusst werden, ob und wie stark die Lichtstärke im Außenbereich des Kegels abgeschwächt werden soll.

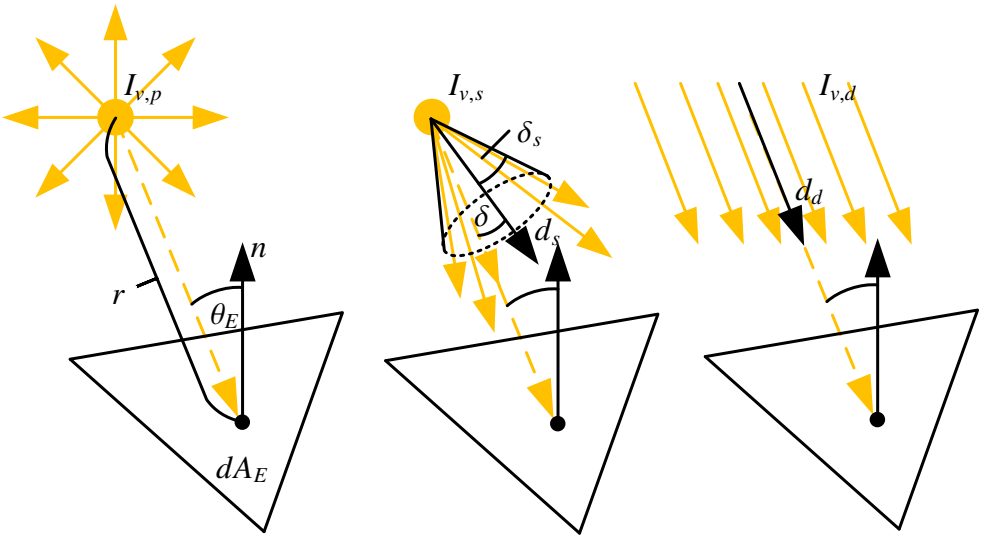


*Bild 2-38*: *Lichtquellen in der Computergrafik (v.l.n.r. Punktlichtquelle (point light), Spotlicht (spot light) und gerichtetes Licht (directional light)).*

Eine weitere Eigenschaft geometrischer Lichter ist die Reichweite. Durch sie wird ein Volumen (Lichtvolumen) definiert, in welchem das jeweilige Licht potentiell Einfluss nehmen kann. Diese Eigenschaft spielt für die Performanz der 3D-Anwendung eine entscheidende Rolle. Durch Schnitttests des Lichtvolumens mit einem Objekt der Szene wird geprüft, ob die entsprechende Lichtquelle beim Rendern dieses Objekts berücksichtigt werden muss. Nur wenn das Lichtvolumen und das Objekt eine Schnittmenge haben, ist das der Fall.

Im Gegensatz zu geometrischen Lichtquellen haben globale Lichter keine räumliche Position und somit auch keinen Volumenkörper, der ihre Einflussnahme begrenzt. Sie wirken gleichermaßen in der gesamten Szene. Reale Lichtquellen haben immer eine Position im Raum. Dennoch ist es sinnvoll, globale Lichtquellen einzuführen. Insbesondere Sonnenlicht wird auf diese Weise modelliert. Grundsätzlich wäre es auch denkbar, die Sonne als Punktlichtquelle abzubilden. In diesem Fall müsste die Punktlichtquelle jedoch sehr weit entfernt von allen übrigen Szenenobjekten positioniert werden und ein sehr großes Lichtvolumen besitzen. Während des Renderings müsste dann für jeden Rasterpunkt der

Objektoberflächen der Lichteinfallsvektor gebildet werden, der sich von Rasterpunkt zu Rasterpunkt kaum unterscheiden würde, da der Abstand zwischen der Sonne und den Objekten erheblich größer ist, als der Abstand der Objekte untereinander. Aus dem gleichen Grund führt die distanzabhängige Lichtabschwächung zu nicht wahrnehmbaren Differenzen in der Beleuchtungsstärke auf den Objekten. Um den Rechenaufwand zu reduzieren, nimmt man deshalb vereinfachend an, dass das Sonnenlicht überall mit einer konstanten Richtung und gleich bleibender Lichtstärke strahlt. Ein derartiges Lichtmodell wird in der Computergrafik als gerichtetes Licht (directional light) bezeichnet und wird im Bild 2-38 auf der rechten Seite dargestellt. Die durch ein gerichtetes Licht hervorgerufene Beleuchtungsstärke $E_{v,d}$ hängt unter den genannten Vereinfachungen ausschließlich vom Einfallswinkel $\theta_E$ ab:

$$E_{v,d} = \frac{d\Phi_{v,d}}{dA_E} \sim I_{v,d} \cos\theta_E.$$

Neben gerichtetem Licht kann ambientes Licht (ambient light) ebenfalls als globale Lichtquelle aufgefasst werden. Hierbei handelt es sich um Licht, das überall und in alle Richtungen mit gleicher Lichtstärke strahlt. Lichtstärke und Beleuchtungsstärke sind in diesem Fall direkt proportional zueinander:

$$E_{v,a} \sim I_{v,a}. \tag{2-30}$$

Ambientes Licht hat kein reales Gegenstück. Die Begründung und Modellierung von ambientem Licht wird im nachfolgenden Abschnitt detaillierter beschrieben.

Die Lichtstärken der bisher vorgestellten Lichtquellen haben keine bzw. eine sehr einfache (Spotlicht) Richtungs- bzw. Ortsabhängigkeit. Im Gegensatz dazu sind diese Eigenschaften bei realen Lichtquellen von großer Bedeutung. Zur Abbildung dieser Effekte existieren in der Computergrafik sogenannte Cookies (Kurzform von "cucoloris"), welche man sich als Transparente vor der Lichtquelle vorstellen kann, die abhängig von der Abstrahlrichtung bzw. dem Abstrahlort verschieden lichtdurchlässig sind. Als Datenstruktur zur Verwaltung dieser Lichtverteilungen dienen Texturen. Diese sind entweder monochrom und bestimmen nur die Helligkeit der Lichtquelle oder farbig, wodurch sie auch die Farbe des Lichts variieren können. Das Bild 2-39 zeigt im rechten Bereich eine einfach Szene, welche durch ein Fenster eintretendes Sonnenlicht darstellen soll. Die Szene selbst beinhaltet jedoch nur den Boden und eine Seitenwand des Raums, sowie die als gerichtetes Licht approximierte Sonne.

Um die Schatten, welche durch den Fensterrahmen entstehen, zu visualisieren, wird das im linken Bereich von Bild 2-39 dargestellte Cookie auf das gerichtete Licht angewendet. Die Farbe schwarz codiert die vollständige Lichtundurchlässigkeit, während die Farbe weiß vollkommene Transparenz repräsentiert. Die Interpretation des Cookies wurde im Beispiel so gewählt, dass dessen Randwerte extrapoliert werden. Im anderen Fall würde neben dem Fenster ebenfalls Licht in den Raum scheinen. Man kann so den Effekt eines durch ein Sprossenfenster eintretendes Licht erzeugen, ohne die zugrunde liegenden Phänomene, wie die Verschattung, modellieren zu müssen.

Bisher wurden die Lichtquellenmodelle der Computergrafik ausschließlich photometrisch diskutiert. Um farbiges Licht und im weiteren Verlauf auch farbabhängige Reflexion modellieren zu können, genügt diese Betrachtung jedoch nicht. Tatsächlich werden Lichtquellen

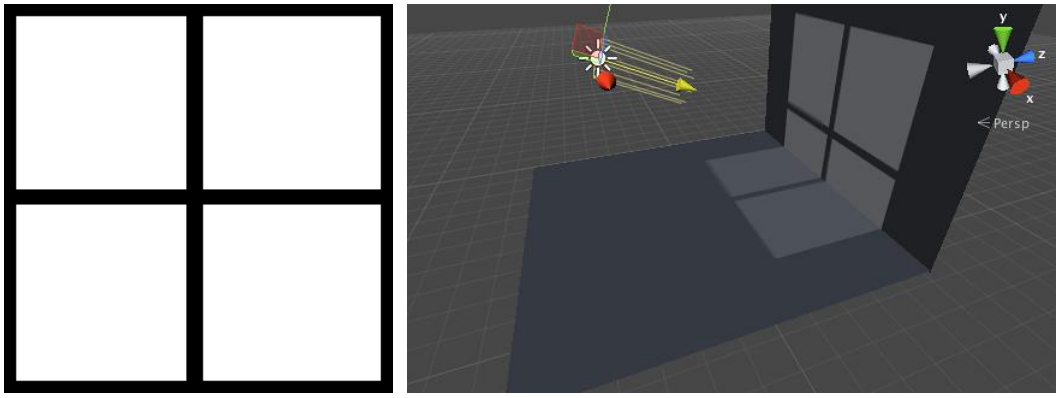


*Bild 2-39*: *Verwendung eines Cookies zur ortsabhängigen Variation der Lichtstärke $I_{v,d}$ eines gerichteten Lichts (links: Cookie-Textur, rechts: Rendering einer einfachen Beispielszene).*

nicht durch ihre skalare Lichtstärke, sondern als Farbe beschrieben. In Abschnitt 2.2.6 wurde gezeigt, wie Licht durch seine Farbkoordinaten nicht nur bezüglich der Helligkeit, sondern auch bezüglich des Farbtons festgelegt werden kann. Grundlage dafür bildete die Bewertung des Lichtspektrums mit den Spektralwertfunktionen. Auf diese Weise konnte der Farbeindruck des Spektrums durch drei skalare Größen – die Farbwerte der Primärvalenzen – beschrieben werden. Als Farbraum wird im Hinblick auf die spätere Ausgabe durch einen Monitor der RGB-Raum verwendet. Man nähert auf diese Weise das kontinuierliche Spektrum des Lichts durch die RGB-Farbanteile an und wertet Reflexionen auf Objekten pro Farbanteil aus. Dieses Verfahren ermöglicht die Modellierung farbiger Lichter und Objekte mit einem vertretbaren rechentechnischen Aufwand und einem übersichtlichen Parameterraum. Mit Ausnahme des High Dynamic Range (HDR) Rendering-Verfahrens (siehe Abschnitt 2.3.4) werden die Farbwerte von Lichtquellen normiert eingesetzt. Da die verwendeten Ausgabegeräte die reale Helligkeit durch hardware-technische Einschränkungen in den allermeisten Fällen nicht wiedergeben können, ist der direkte Bezug zu photometrischen Größen ohnehin nicht von Bedeutung. Entscheidend für die realistische Wahrnehmung ist nur das Verhältnis der Helligkeiten innerhalb der Szene. Die hellste definierbare Lichtquelle ist dementsprechend weiß und hat den Farbvektor $(1, 1, 1)^T$.

### Lokale Beleuchtungsmodelle

Nachdem die Lichtquellenmodelle diskutiert wurden, stellt sich nun die Frage, wie das von ihnen ausgesandte Licht mit den Objekten in der virtuellen Szene wechselwirkt. Grundlage hierfür bilden Beleuchtungsmodelle. Die meist genutzten Beleuchtungsmodelle in der Computergrafik sind lokale Modelle. Sie zeichnen sich dadurch aus, dass die Farbe eines Oberflächenpunkts (je nach Ansatz ein Vertex oder ein Fragment) in der Szene, welche von der Beschaffenheit des Materials und dem Einfluss darauf fallender Lichtstrahlen abhängt, unabhängig von allen anderen Szenenpunkten bestimmt werden kann. Offensichtlich werden hierbei drastische Vereinfachungen getroffen, da ein derartiger Ansatz wesentliche

Effekte, wie reflektiertes Licht oder Transmission, gänzlich ignoriert. Gleichzeitig stellen diese Vereinfachungen jedoch sicher, dass die Farben einzelner Szenenpunkte isoliert voneinander und in einem Schritt berechnet werden können, wodurch erneut die Effizienz der GPU bei der parallelen Berechnung genutzt wird. Aufgrund der Echtzeitanforderungen werden die meisten Anteile in interaktiven Anwendungen, wie Computerspielen oder Simulationen, auf Basis lokaler Beleuchtungsmodelle gerendert.

Anhand des 1975 veröffentlichten Phong-Beleuchtungsmodells, welches bis heute in den Grundzügen unverändert geblieben ist, soll nun der Einsatz von lokalen Beleuchtungsmodellen in der Computergrafik verdeutlicht werden [Pho75]. Aufgabe jedes Beleuchtungsmodells ist die Bestimmung der Objektfarbe aus Sicht des Betrachters. In photometrischen Größen ist die Leuchtdichte für die wahrgenommene Helligkeit entscheidend. Sie ist das Maß für die Wahrnehmung durch das menschliche Auge. Um Farben abbilden zu können, muss diese Größe jedoch separat für die drei Primärvalenzen des RGB-Farbraums ausgewertet werden. Zur Überführung der Bestrahlungsstärke auf einem gegebenen Flächenelement in dessen Leuchtdichte können die in Abschnitt 2.1.5 beschriebenen BRDF eingesetzt werden.

Bei dem Phong-Beleuchtungsmodell handelt es sich um ein empirisches Modell zur Beschreibung von Reflexionsphänomenen an Objektoberflächen. Das Phong-Modell bestimmt die finale Farbe eines Oberflächenpunkts durch die materialabhängige Interpolation verschiedener BRDF. Dazu unterscheidet Phong drei Grenzfälle von Reflexionen, für die jeweils reduzierte BRDF definiert werden. Diese sind ambiente ($a$: ambient), diffuse ($d$: diffuse) und glänzende ($s$: specular) Reflexionen. Für jeden dieser Fälle können den verschiedenen Materialien der Szenenobjekte unterschiedliche farbabhängige Reflexionscharakteristiken ($C_{a,m}$, $C_{d,m}$, $C_{s,m} \in [0,1]^3$) zugewiesen werden. Die Koordinaten der Farbvektoren beschreiben die Reflexionskoeffizienten der Materialien bezüglich der Primärvalenzen des RGB-Farbraums. Insgesamt ergibt sich die beobachtete Farbe eines Objektes aus der RGB-Lichtfarbe $C_l \in [0,1]^3$ und dem Reflexionsverhalten $C_{a,m}$, $C_{d,m}$ und $C_{s,m}$ des Materials für die verschiedenen Reflexionsfälle. Wählt man beispielsweise als Materialfarbe den RGB-Vektor $(1,1,1)^T$, so bleibt die Lichtfarbe (im diffusen Reflexionsfall) nach der Reflexion unverändert. Für $(1,0,0)^T$ wird hingegen nur die rote Komponente des Lichts reflektiert. Das Objekt erscheint deshalb für den Betrachter rot, sofern Lichtquellen mit rotem Farbanteil in der Szene existieren. Im anderen Fall ist das Objekt schwarz. Die für den jeweiligen Reflexionsfall spezifischen BRDF setzen Beleuchtungsstärke und Leuchtdichte somit in Beziehung. Während sich der Zusammenhang im diffusen Fall durch einfache farbspezifische Proportionalitätsfaktoren äußert, werden bei der spiegelnden Reflexion zusätzliche Abhängigkeiten berücksichtigt. Die Details werden im Folgenden diskutiert.

Die ambiente Komponente beschreibt Reflexion durch Licht, das aus allen Richtungen in gleichbleibender Intensität strahlt. Dieses Licht hat keinen physikalischen Ursprung. Es wird der Vernachlässigung von Mehrfachreflexion in lokalen Beleuchtungsmodellen geschuldet. Lokale Beleuchtungsmodelle berücksichtigen abgesehen von der ambienten Komponente ausschließlich Licht, das direkt von einer Lichtquelle auf den betrachteten Oberflächenpunkt geworfen wird, um dessen Farbe aus Sicht des Betrachters zu ermitteln. Tatsächlich fällt Licht jedoch aus allen Richtungen auf einen Oberflächenpunkt, da andere Objekte in der Umgebung Licht reflektieren und auf diese Weise unendlich viele Lichtquellen verschiedener Farbe erzeugen. Da diese Gegebenheit in lokalen Modellen aus

Gründen der Recheneffizienz nicht berücksichtigt wird, dient das Modell des ambienten Lichts zur Approximation der realen Effekte. Aus Gleichung (2-30) ist bekannt, dass sich die Beleuchtungsstärke unabhängig von allen geometrischen Gegebenheiten proportional zur definierten Lichtstärke verhält. Nach Gleichung (2-15) liefert die BRDF somit den direkten Übergang von der definierten Lichtfarbe zur Leuchtdichte. Da die BRDF des ambienten Reflexionsfalls durch farbspezifische Proportionalitätsfaktoren $C_{a,m}$ gebildet wird, ergibt sich letztlich für die Objektfarbe $C_a$ der Term

$$C_a = C_{a,m} \circ C_l. \tag{2-31}$$

Der Operator $\circ$ bezeichnet hierbei das Hadamard-Produkt, sodass $C_a$ ebenfalls ein Element des $[0,1]^3$ ist und die vom Betrachter wahrgenommene Farbe des Szenenpunkts im RGB-Raum definiert.

Als nächstes soll die diffuse Reflexion betrachtet werden. Sie bildet das Reflexionsverhalten von Körpern ab, deren Oberflächen nach dem Lambertschen Gesetz (siehe Abschnitt 2.1.5) reflektieren. Ein Lambertsches Flächenelement weist eine richtungsunabhängige Leuchtdichte auf. Sie hängt nur von der Beleuchtungsstärke ab. Diese ergibt sich abhängig vom Lichtquellentyp gemäß Abschnitt 2.3.4 durch die geometrischen Lagebeziehungen zwischen Lichtquelle und Objekt sowie der definierten Lichtfarbe $C_l$. Unter den Reflexionsfaktoren $C_{d,m}$ des Materials für den diffusen Fall ergibt sich somit die wahrgenommene Objektfarbe $C_d$ gemäß

$$\begin{aligned}
\text{Gerichtetes Licht: } C_d &= C_{d,m} \circ \cos\theta_E \cdot C_l = C_{d,m} \circ (l \cdot n) \cdot C_l \\
\text{Punktlicht: } C_d &= C_{d,m} \circ \frac{\cos\theta_E}{r^2} \cdot C_l = C_{d,m} \circ \frac{l \cdot n}{r^2} \cdot C_l \\
\text{Spotlicht: } C_d &= C_{d,m} \circ \frac{\cos\theta_E}{r^2} \cdot C_l^\star = C_{d,m} \circ \frac{l \cdot n}{r^2} \cdot C_l^\star \\
&\text{mit } C_l^\star = \begin{cases} C_l \cdot \cos^{n_s}\delta & |\delta| \le \delta_s \\ 0 & \text{sonst.} \end{cases}
\end{aligned} \tag{2-32}$$

Da die Koordinaten der Vertices aller Objekte zur Verfügung stehen und Vektoroperationen auf der GPU sehr schnell ausgeführt werden können, erfolgt die Berechnung des Lichteinfallswinkels $\theta_E$ vektorbasiert. Dazu werden die in Bild 2-40 dargestellten Hilfsvektoren genutzt. Der Normalenvektor $n$ steht für jedes Vertex und nach der Rasterung auch für jedes Fragment zur Verfügung. Der Einheitsvektor $l$, welcher ausgehend vom betrachteten Oberflächenpunkt auf die Lichtquelle zeigt, wird berechnet. Auf die gleiche Weise kann der Cosinus von $\delta$ durch das Skalarprodukt $d \cdot (-l)$ ausgedrückt werden.

Die zugrunde liegende BRDF findet sich im diffusen Fall im Vektor $C_{d,m}$. Je nach betrachteter Primärvalenz des RGB-Farbraums entspricht die BRDF $f_r$ der Rendergleichung (2-28) dem ersten (rote Primärvalenz $R_P$), zweiten (grüne Primärvalenz $G_P$) oder dritten (blaue Primärvalenz $B_P$) Eintrag aus $C_{d,m}$:

$$\begin{aligned}
f_{r,d,R} &= c_{d,m,R} \\
f_{r,d,G} &= c_{d,m,G} \\
f_{r,d,B} &= c_{d,m,B} \\
\text{mit } C_{d,m} &= (c_{d,m,R}, c_{d,m,G}, c_{d,m,B})^T.
\end{aligned}$$

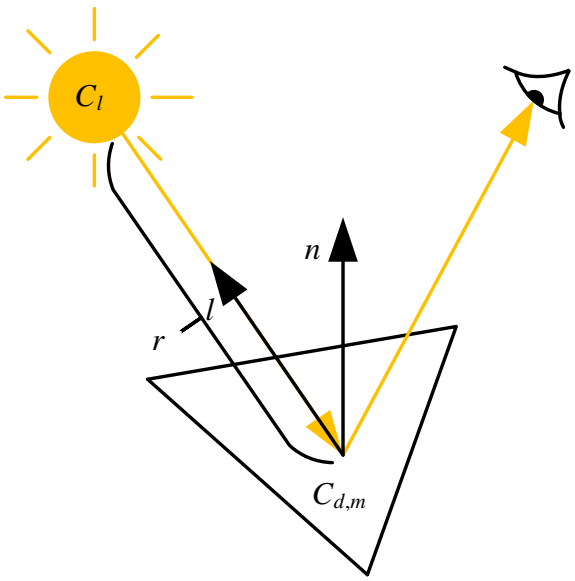


*Bild 2-40*: *Vektorbasierte Berechnung diffuser Reflexion nach Phong.*

Hieran zeigt sich, dass das Phong-Beleuchtungsmodell die Forderung der Energieerhaltung (siehe Abschnitt 2.1.5) an eine physikalisch korrekte Reflexionsdichtefunktion im Allgemeinen nicht erfüllt. Aus den Betrachtungen in Abschnitt 2.1.5 ist hervorgegangen, dass nach Gleichung (2-16) der Wert von $f_{r,L} = \frac{1}{\pi}$ für eine BRDF eines Lambertschen Flächenelements bei vollständiger Reflexion gilt, während für die Einträge des Vektors $C_{d,m}$ Werte im Intervall $[0, 1]$ zulässig sind. Um die Energieerhaltung sicherzustellen, müssten die BRDF also um den Faktor $\frac{1}{\pi}$ ergänzt werden. Da es sich hierbei jedoch nur um einen konstanten Skalierungsfaktor handelt, wird von dieser Normierung abgesehen.

Der dritte Reflexionsfall des Phong-Modells ist die spiegelnde Reflexion. Dieser Fall tritt primär auf glatten Oberflächen, wie geschliffenen Metallen oder nassen Strukturen, auf. Er dient der Darstellung von Glanzeffekten. Bei der spiegelnden Reflexion hängt die Leuchtdichte des betrachteten Szenenpunkts nicht nur von der Beleuchtungsstärke, sondern auch von der Beleuchtungsrichtung und der Beobachtungsrichtung ab. Das Licht wird primär in die ideale Reflexionsrichtung reflektiert. Im Bild 2-41 werden die relevanten Größen zur Berechnung der spiegelnden Reflexion nach Phong visualisiert.

Neben den Größen, die bereits aus dem diffusen Fall bekannt sind, kommen die Vektoren zum Betrachter $e$ und der idealen Reflexionsrichtung $r_{spec}$ hinzu. Entscheidend für die Stärke der Reflexion ist die Übereinstimmung von $e$ und $r_{spec}$. Der Reflexionsvektor $r_{spec}$ kann aus den bekannten Vektoren durch

$$r_{spec} = 2n \cdot (n \cdot l) - l \tag{2-33}$$

ermittelt werden. Da diese Berechnung verhältnismäßig aufwendig ist, hat James F. Blinn 1977 eine erheblich schnellere Approximation eingeführt, die heute typischerweise Anwendung findet [Bli77]. Die optimierte Variante wird als Blinn-Phong-Modell bezeichnet.

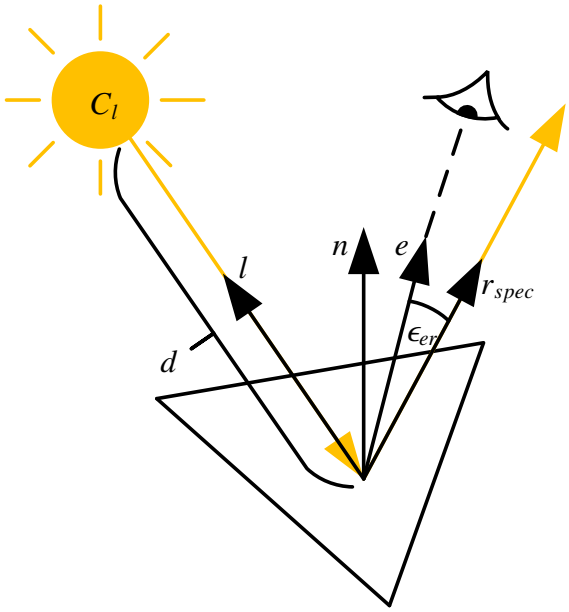


*Bild 2-41*: *Berechnung spiegelnder Reflexion nach Phong.*

Sind $e$ und $r_{spec}$ gleich, nimmt der Beobachter die maximale Helligkeit wahr. Durch den Cosinus des Winkels $\epsilon_{er}$ wird die Übereinstimmung der Vektoren gemessen. Wie bereits aus der Lichtstärke des Spotlichts bekannt, wird darüber hinaus ein Exponent $n_r$ eingesetzt, um die Abschwächung bei zunehmendem Winkel $\epsilon_{er}$ parametrieren zu können. Es handelt sich bei $n_r$ um eine materialspezifische Größe, die für raue Materialien klein ist und mit der Glattheit zunimmt. Für einen idealen Spiegel gilt $n_r \to \infty$. Insgesamt findet sich abhängig von der eingesetzten Lichtquelle folgende Berechnungsvorschrift für die spiegelnde Reflexion:

$$\begin{aligned}
\text{Gerichtetes Licht: } C_s &= C_{s,m} \circ \frac{n_r+1}{2\pi} \cos^{n_r} \epsilon_{er} \cdot C_l = C_{s,m} \circ \frac{n_r+1}{2\pi} (r_{spec} \cdot v)^{n_r} \cdot C_l \\
\text{Punktlicht: } C_s &= C_{s,m} \circ \frac{n_r+1}{2\pi} \frac{\cos^{n_r} \epsilon_{er}}{r^2} \cdot C_l = C_{s,m} \circ \frac{n_r+1}{2\pi} \frac{(r_{spec} \cdot v)^{n_r}}{r^2} \cdot C_l \\
\text{Spotlicht: } C_s &= C_{s,m} \circ \frac{n_r+1}{2\pi} \frac{\cos^{n_r} \epsilon_{er}}{r^2} \cdot C_l^\star = C_{s,m} \circ \frac{n_r+1}{2\pi} \frac{(r_{spec} \cdot v)^{n_r}}{r^2} \cdot C_l^\star .
\end{aligned} \tag{2-34}$$

Die physikalischen Hintergründe der spiegelnden Reflexion sind komplex und werden in dieser Arbeit nicht betrachtet. Bei den Gleichungen (2-34) handelt es sich um rein empirische Approximationen. Der Normierungsfaktor $k_{norm} = \frac{n_r+1}{2\pi}$ dient dazu, den reflektierten Lichtstrom vom Exponenten $n_r$ zu entkoppeln. Würde man auf diese Normierung verzichten, würde der reflektierte Lichtstrom bei konstanter Beleuchtungsstärke mit $n_r$ variieren. Um ihn zu finden, nutzt man die Bedingung der Energieerhaltung einer BRDF aus Abschnitt 2.1.5, indem man fordert, dass die Integration des Reflexionsfaktors eines

Szenenpunkts über seiner gesamten Hemisphäre $\Omega^+$ unabhängig von $n_r$ Eins entsprechen soll:

$$k_{norm} \cdot \int_{\Omega_S \in \Omega^+} \cos^{n_r} \theta_S \, d\Omega_S \stackrel{!}{=} 1.$$

Durch partielle Integration findet man, dass

$$\int_{\Omega_S \in \Omega^+} \cos^{n_r} \theta_S \, d\Omega_S = \int_{\varphi_S=0}^{2\pi} \int_{\theta_S=0}^{\frac{\pi}{2}} \cos^{n_r} \theta_S \, \sin \theta_S \, d\theta_S \, d\varphi_S = \frac{2\pi}{n_r + 1}$$

gilt. Der gesuchte Normierungsfaktor $k_{norm}$ ergibt sich schließlich durch Kehrwertbildung und stellt die Invarianz bezüglich des Exponenten $n_r$ sicher.

Typischerweise ist es so, dass die Lichtquellenfarbe $C_l$ bei der spiegelnden Reflexion im Gegensatz zur diffusen Reflexion meist die ausschlaggebende Komponente darstellt. Es findet sich also im reflektierten Licht $C_s$ primär die Farbe der Lichtquelle, weniger die Farbe des beleuchteten Objekts $C_{s,m}$ wieder. Aus diesem Grund erhalten die Einträge der Farbvektoren für den spiegelnden Fall häufig die gleichen Werte. Glatte Kunststoffe stellen hierfür ein Beispiel dar. Gleichzeitig gilt diese Regel jedoch nicht für alle Metalle (z.B. Gold, Kupfer), deren Eigenfarbe auch an den Glanzpunkten deutlich zu erkennen ist (Vgl. Tabelle 2-4).

Bis hierhin wurden drei Komponenten vorgestellt, welche die Wahrnehmung eines Szenenpunkts durch den Beobachter approximieren. Die meisten Objekte können nicht durch eine einzelne dieser Komponenten beschrieben werden. Stattdessen spannen sie das Feld auf, in dem die Objekte der Szene entsprechend ihres Materials einsortiert werden können. Das Beleuchtungsmodell führt deshalb alle beschriebenen Effekte zusammen und bestimmt so die finale Farbe $C_\sum$ jedes Szenenpunkts durch die Summierung der angesprochenen Reflexionsfälle

$$C_\sum = C_a + C_d + C_s. \tag{2-35}$$

Für einen gegebenen Szenenpunkt gilt es, die Gleichung (2-35) für jede auf ihn einwirkende Lichtquelle auszuwerten, wobei je nach Lichtquellentyp die entsprechende Beziehung aus den Gleichungen (2-31), (2-32) und (2-34) zu wählen ist. Das Bild 2-42 zeigt ein nach dem Phong-Modell beleuchtetes Objekt und isoliert die Einflüsse der verschiedenen Reflexionstypen innerhalb der Gesamterscheinung.

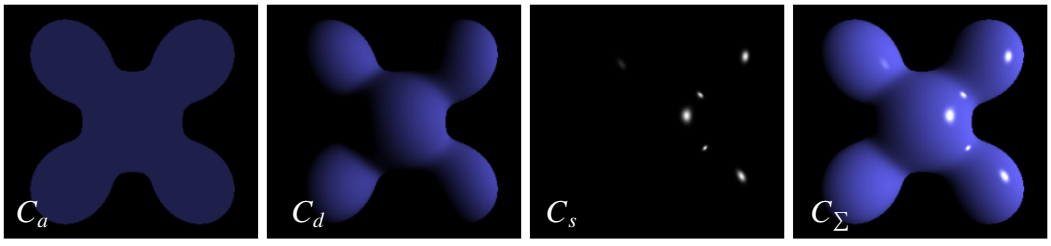


*Bild 2-42*: *Einflüsse der verschiedenen Reflexionstypen innerhalb des Phong-Beleuchtungsmodells [Quelle: Wikipedia].*

Um einen Eindruck geeigneter Parametrierungen des Phong-Beleuchtungsmodells zu erhalten, werden in Tabelle 2-4 abschließend die Werte für verschiedene Materialien aufgelistet.

*Tabelle 2-4*: *Materialparameter verschiedener Materialien für das Phong-Beleuchtungsmodell nach [BB06].*

| Material | $C^T_{a,m}$ | $C^T_{d,m}$ | $C^T_{s,m}$ | $n_r$ |
|---|---|---|---|---|
| Kunststoff | (0.00, 0.00, 0.00) | (0.01, 0.01, 0.01) | (0.50, 0.50, 0.50) | 32 |
| Messing | (0.33, 0.22, 0.03) | (0.78, 0.57, 0.11) | (0.99, 0.94, 0.81) | 28 |
| Bronze | (0.21, 0.13, 0.05) | (0.71, 0.43, 0.18) | (0.39, 0.27, 0.17) | 26 |
| Kupfer | (0.19, 0.07, 0.02) | (0.70, 0.27, 0.08) | (0.26, 0.14, 0.09) | 13 |
| Gold | (0.25, 0.20, 0.07) | (0.75, 0.61, 0.23) | (0.63, 0.56, 0.37) | 51 |
| Silber | (0.19, 0.19, 0.19) | (0.51, 0.51, 0.51) | (0.51, 0.51, 0.51) | 51 |

### Globale Beleuchtungsmodelle

Im Unterschied zu lokalen Beleuchtungsmodellen, die eine isolierte Betrachtung von Objektoberflächen zur Berechnung der wahrgenommenen Farbe ermöglichen, simulieren globale Beleuchtungsmodelle die Ausbreitung von Licht in der Szene. Dabei berücksichtigen sie auch Wechselwirkungen zwischen Objekten der Szene, die keine Lichtquellen sind, aber Licht reflektieren und insofern letztlich selbst als Lichtquellen aufgefasst werden müssen. Globale Ansätze sind erheblich rechenintensiver und nach dem aktuellen Stand der Technik für die allermeisten Echtzeitanwendungen nicht als alleinige Rendering-Verfahren geeignet. Andererseits führen sie aufgrund der deutlich größeren Nähe zur Physik zu realistischeren Ergebnissen. Viele Effekte, die bei der Verwendung von lokalen Ansätzen durch aufwendige Zusatzberechnungen (z.B. Shadow-Maps, Ambient Occlusion, Screen Space Reflection, Depth of Field [AFS+13]) generiert werden müssen und hier nicht thematisiert werden, enstehen beim Einsatz globaler Beleuchtungsmodelle implizit und in überlegener Qualität.

In der Computergrafik finden zwei grundlegend verschiedene Ansätze zur Berechnung globaler Beleuchtung Anwendung. Hierbei handelt es sich um Radiosity und Raytracing. Beide Verfahren bilden die Szene auf unterschiedliche Weise ab und verfügen deshalb über verschiedene Vor- und Nachteile. Da Radiosity annimmt, dass die gesamte Szene aus ideal Lambertschen Flächen besteht (siehe Abschnitt 2.1.5), ist die Beleuchtungssituation unabhängig vom Blickwinkel des Betrachters. Für den Fall statischer Lichtquellen genügt es deshalb, die Beleuchtungssituation einmalig zu ermitteln. Anschließend ist nur noch die Darstellung der Szenenkörper in der richtigen Orientierung und Verdeckung zu leisten. Auf diese Weise ermöglichte das Radiosity-Verfahren schon früh die Berücksichtigung globaler Beleuchtung in Grafik-Anwendungen. Mit der exponentiell steigenden Rechenleistung und neuen algorithmischen Umsetzungen gewann das rechenintensivere Raytracing (dt. Strahlverfolgung) bis heute jedoch immer mehr an Bedeutung und ist dem Radiosity-Verfahren inzwischen in den meisten Fällen überlegen. Es führt insbesondere bei der Existenz spiegelnder Flächen in der Szene zu deutlich besseren Ergebnissen, während glatte Oberflächen im Radiosity-Ansatz in keinster Weise Berücksichtigung finden. Aus diesem Grund soll die globale Beleuchtung im Folgenden am Beispiel des Raytracing-Verfahrens vorgestellt werden.

Wie für die in Abschnitt 2.3.1 beschriebene Rasterisierung in der klassischen Rendering Pipeline, bilden auch bei Verwendung des Raytracing-Verfahrens der Szenegraph und

insbesondere die Lagen und Eigenschaften der Lichter und der Kamera die Grundlage. Bei der Rasterisierung werden Objekt für Objekt, Vertex für Vertex und schließlich Fragment für Fragment weitestgehend unabhängig voneinander abgearbeitet wird. Man nennt diesen Ansatz auch objektorientiert, da die jeweilige Objektgeometrie Kern der Betrachtung ist. Sie bildet den Startpunkt der Algorithmik und aus ihrer Sicht werden Einflüsse von Licht und Betrachtungswinkel bewertet. Das Raytracing-Verfahren hingegen ist bildorientiert. Es arbeitet Pixel für Pixel des aktuellen Frames ab. Bild 2-43 veranschaulicht das zugrunde liegende Prinzip. Obwohl es naheliegend wäre, Lichtstrahlen von den Lichtquellen auf ihrem Weg bis zum Beobachter zu verfolgen, gehen Raytracing-Ansätze in vielen Fällen den umgekehrten Weg. Sie verfolgen den Lichtstrahl ausgehend vom Einfallswinkel am Auge zurück. Das hat den erheblichen Vorteil, dass nur die Strahlen berücksichtigt werden, welche den Betrachter erreichen. Würde man an der Lichtquelle beginnen, so wäre die Chance einen Strahl auszuwählen, der schließlich am Augpunkt (Kameraposition) eintrifft, verschwindend gering. Wie viele Einfallswinkel dabei berücksichtigt werden, hängt von der erwünschten Bildgüte ab. Je mehr Strahlen (Rays) betrachtet werden, desto rechenintensiver wird das Verfahren. Eine untere Schranke für die Menge der zu betrachtenden Rays ist bei einer vollständig durch Raytracing gerenderten Szene durch die Pixel des Frames bzw. die Rasterpunkte der Bildebene gegeben. Im einfachsten Fall schießt man vom Augpunkt durch jeden Pixel einen Ray, wie es in Bild 2-43 beispielhaft für zwei konkrete Pixel dargestellt wird. Diese Rays werden auch als Primärrays bezeichnet, da sie den ersten Schritt der Strahlverfolgung darstellen.

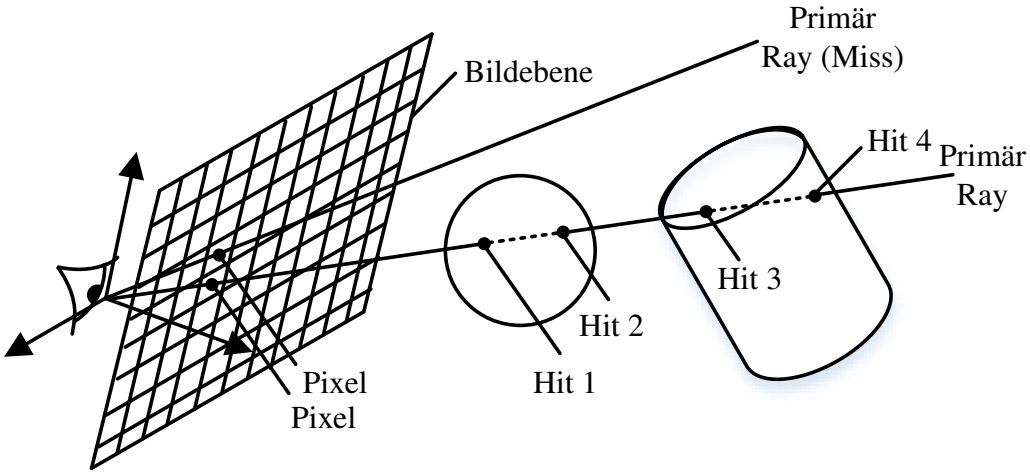


*Bild 2-43*: *Prinzip des bildorientierten Raytracing-Verfahrens.*

Die ursprüngliche Aufgabe des Raytracings ist die Prüfung der Sichtbarkeit von Objekten aus der Perspektive des Augpunkts. Umgesetzt wird diese Überprüfung durch Schnittpunkttests zwischen dem jeweiligen Ray und den Objekten der Szene. Dazu müssen sowohl der Ray als auch die Geometrien der Szenenobjekte mathematisch gefasst werden. Ein Ray kann als Halbgerade $g_r$ im dreidimensionalen Raum durch die Vorschrift

$$g_r : v = t_r \cdot d_r + a_r, t_r \geq 0 \tag{2-36}$$

formuliert werden, wobei $d_r \in \mathbb{R}^3$ der normierte Richtungsvektor und $a_r \in \mathbb{R}^3$ die Position des Augpunkts sind. Der skalare Wert $t_r$ ist der Halbgeradenparameter und $v \in \mathbb{R}^3$ der mit ihm korrespondierende Punkt auf $g_r$. Die Koordinaten von $d_r$ ergeben sich durch den Pixel auf der Bildebene. Die Geometrien der Objekte werden durch Primitive zusammengesetzt. Im Gegensatz zur Rasterisierung bestehen nun jedoch andere Anforderungen an die Primitive. Wesentlich ist, dass die Schnittpunktberechnung effizient erfolgen kann, da diese den größten rechentechnischen Aufwand von Raytracern einnimmt. Trotz dessen beschränkt man sich meist auf Polygone, da Raytracer bis heute größtenteils in hybriden Pipelines, welche einige Aspekte des Renderings durch Rasterisierung und andere durch Raytracing abdecken, zum Einsatz kommen. Wir betrachten beispielhaft eine Kugel, welche durch ihr Zentrum $c_K \in \mathbb{R}^3$ und ihren Radius $r_K \in \mathbb{R}^{\geq 0}$ bestimmt wird.

Die Punkte auf der Kugeloberfläche $s$ erfüllen die Gleichung

$$s : |v - c_K|^2 = r_K^2 \equiv (v - c_K) \cdot (v - c_K) = r_K^2. \tag{2-37}$$

Einsetzen des Rays (2-36) für den Punkt $v$ in der Kugelgleichung (2-37) führt nach Umformung auf die Gleichung

$$(d_r \cdot d_r) \cdot t_r^2 + 2 \cdot (d_r \cdot (a_r - c_K)) \cdot t_r + (a_r - c_K) \cdot (a_r - c_K) - r_K^2 = 0$$

und somit zu der Lösung

$$\begin{aligned} t_r = \quad & \frac{-h_1 \pm \sqrt{h_1^2 - 4d_r^2 h_2}}{2d_r^2} \\ \text{mit } h_1 = \quad & 2d_r \cdot (a_r - c_K) \\ h_2 = \quad & (a_r - c_K)^2 - r_K^2. \end{aligned}$$

Abhängig vom Wert der Diskriminate $D = h_1^2 - 4d_r^2 h_2$ ergeben sich die Fälle

- $D > 0$: Ray durchstößt die Kugel (2 Schnittpunkte)
- $D = 0$: Ray tangiert die Kugel (1 Kontaktpunkt)
- $D < 0$: Ray verfehlt die Kugel,

wobei vorausgesetzt wird, dass der Rayparameter $t_r$ für die entsprechende Lösung mindesten so groß ist, dass der Schnittpunkt aus Sicht des Augpunkts hinter der Bildebene liegt. Im anderen Fall werden die Lösungen verworfen.

Ein Schnittpunkt des Rays mit einem Primitiv der Szene wird als Hit bezeichnet. Im Regelfall ergeben sich für jeden Ray eine Vielzahl von Hits. Von Interesse ist jedoch nur derjenige, welcher den geringsten Abstand zum Augpunkt hat. Transparente Medien stellen eine Ausnahme dar, sollen aber hier nicht näher betrachtet werden. Die Schnittpunktberechnung nimmt den größten Anteil des Rechenaufwands von Raytracing-Verfahren ein. Die Berechnung der Hits mit allen Objekten der Szene ginge mit einer immensen Verschwendung der Rechenleistung einher. Um Raytracer zu beschleunigen, fasst man die Elemente der Szene rekursiv in lokale Gruppen zusammen und sortiert diese in einer geeigneten Datenstruktur. Man spricht bei dieser Konstruktion von einer Hüllkörper-Hierarchie (BVH, Bounding Volume Hierarchy).

Ein Hüllkörper vereinfacht allgemein die Berechnung von Schnittüberprüfungen. Dazu umhüllt er die eigentliche, komplexe Geometrie und weist dabei selbst eine einfache Form auf. Es haben sich verschiedene Formen von Hüllkörpern etabliert. Beispiele sind Axis-Aligned Bounding Boxes (AABB), Oriented Bounding Boxes (OBB), Kugeln oder Bounding Slabs (k-DOP, Discretely Oriented Polytopes). Die Auswahl des Hüllkörpertyps ist ein Kompromiss zwischen einfacher Form bzw. schnellem Schnitttest und der Volumenzunahme gegenüber der eigentlichen Geometrie, welche zu einer größeren Anzahl von Schnittprüfungen führt.

Beispielhaft soll hier die Schnittprüfung mit AABB-Hüllkörpern vorgestellt werden. Bei diesen Hüllkörpern handelt es sich um Quader, deren Kanten parallel zum globalen Koordinatensystem ausgerichtet sind. Die Quader müssen bei dynamischen Objekten deshalb so groß gewählt werden, dass das umhüllte Objekt in allen Rotationen enthalten ist. In Bild 2-44 wird ein Zylinder dargestellt, der von einem AABB-Hüllkörper umschlossen wird.

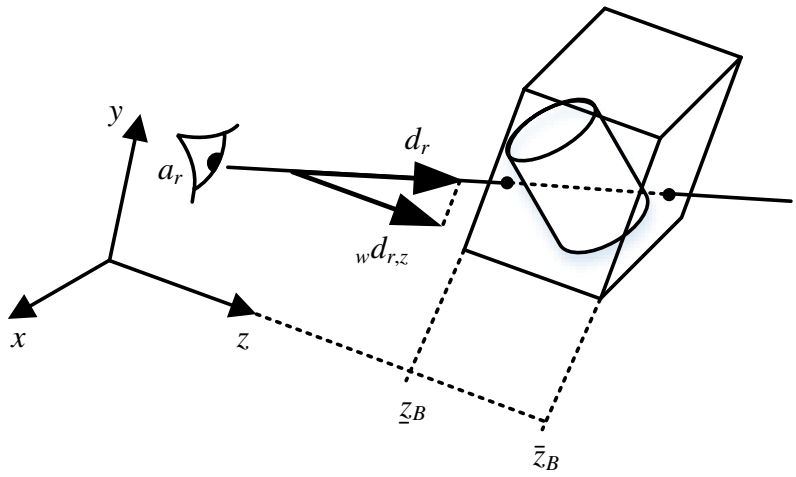


*Bild 2-44*: *Schnitttest mit AABB-Hüllkörper.*

Im Folgenden wird beschrieben, wie die Kollisionsprüfung zwischen einem Ray und dem skizzierten Hüllkörper erfolgt. Im ersten Schritt wird der Ray auf die Achsen des globalen Koordinatensystems projiziert, an welchem der Hüllquader ausgerichtet ist. Dazu werden sowohl der Augpunkt als auch der Richtungsvektor des Rays (siehe Gleichung (2-36)) mit den Basisvektoren $x_w$, $y_w$ und $z_w$ des Weltkoordinatensystems $w$ skalarmultipliziert.

$$\begin{aligned} a_r \cdot x_w &= {}_wa_{r,x}, \quad d_r \cdot x_w = {}_wd_{r,x} \\ a_r \cdot y_w &= {}_wa_{r,y}, \quad d_r \cdot y_w = {}_wd_{r,y} \\ a_r \cdot z_w &= {}_wa_{r,z}, \quad d_r \cdot z_w = {}_wd_{r,z} \end{aligned} \tag{2-38}$$

Der AABB-Hüllkörper kann aufgrund seiner Orientierung im Koordinatensystem $w$ besonders einfach beschrieben werden. Er umfasst alle Punkte $v \in \mathbb{R}^3$, deren Koordinaten in den Intervallen

$$\begin{aligned} {}_wv_x &\in [{}_w\underline{x}_B, {}_w\bar{x}_B] \\ {}_wv_y &\in [{}_w\underline{y}_B, {}_w\bar{y}_B] \\ {}_wv_z &\in [{}_w\underline{z}_B, {}_w\bar{z}_B] \end{aligned} \tag{2-39}$$

enthalten sind. Diese Eigenschaft erlaubt die isolierte Kollisionsprüfung je Koordinatenachse, indem man durch Gleichsetzen der Rayterme (2-38) und der Intervallgrenzen des Hüllkörpers (2-39) für die jeweilige Dimension die Rayparameter erhält:

$$\begin{aligned} \underline{t}_x &= \frac{{}_w\underline{x}_B - {}_wv_x}{{}_wd_{r,x}}, \quad \bar{t}_x = \frac{{}_w\bar{x}_B - {}_wv_x}{{}_wd_{r,x}} \\ \underline{t}_y &= \frac{{}_w\underline{y}_B - {}_wv_y}{{}_wd_{r,y}}, \quad \bar{t}_y = \frac{{}_w\bar{y}_B - {}_wv_y}{{}_wd_{r,y}} \\ \underline{t}_z &= \frac{{}_w\underline{z}_B - {}_wv_z}{{}_wd_{r,z}}, \quad \bar{t}_z = \frac{{}_w\bar{z}_B - {}_wv_z}{{}_wd_{r,z}}. \end{aligned}$$

Aufbauend auf diesen Vorüberlegungen gilt nach Cyrus-Beck die folgende Äquivalenz [CB78]:

$$\text{Ray} \cap AABB = \emptyset \equiv \max\{\underline{t}_x, \underline{t}_y, \underline{t}_z\} > \min\{\bar{t}_x, \bar{t}_y, \bar{t}_z\}$$

Mit dieser einfachen Überprüfung können viele Rays bereits vor einem exakten Schnitttest mit der tatsächlichen Geometrie ausgeschlossen werden. Diejenigen Rays, die den Hüllkörper schneiden, müssen einem erneuten Schnitttest mit der konkreten Geometrie unterzogen werden. Dieser ist im Regelfall erheblich aufwendiger. Bei Polygonmodellen ist die Schnittprüfung mit jeder Polygonfläche erforderlich. Weiterhin ist nun von Interesse, wo die Schnittpunkte auf der Geometrie liegen, um dort die Auswertung des Beleuchtungsmodells vornehmen zu können.

In komplexen Szenen existiert eine Vielzahl von Objekten, sodass die Verwendung von Bounding Volumes die Schnittprüfung bereits erheblich vereinfacht. Ein iteratives Prüfen aller Bounding Volumes in der Szene wäre jedoch noch immer zu rechenintensiv, als dass es die Realisierung eines Echtzeit-Raytracings ermöglichen würde. Zur Reduzierung der nötigen Schnitttests organisiert man die Hüllkörper der einzelnen Objekte in einer Hierarchie, wobei lokale Gruppen von Hüllkörpern wiederum durch Hüllkörper zusammengefasst werden. Auf diese Weise entsteht eine hierarchische, meist baumartige Struktur. Bild 2-45 setzt den Szenegraphen und die korrespondierende BVH in Beziehung.

Die in Bild 2-45 grau dargestellten Szenenobjekte werden jeweils von ihren Hüllkörpern C, D und E umhüllt. D und E werden dabei wiederum durch den Hüllkörper B in einer lokalen Gruppe zusammengefasst. Schließlich umhüllt A alle dargestellten Objekte. Während in Bild 2-45 auf der linken Seite die räumliche Anordnung der Objekte und Hüllkörper in der Szene visualisiert wird, ist auf der rechten Seite die Baumstruktur des BVH dargestellt. Der die gesamte Szene umfassende Hüllkörper A stellt die Wurzel des Baums dar. Die Kinder von A sind B und C. Bei C handelt es sich um ein Blatt des BVH, da es der Hüllkörper einer Geometrie der Szene ist. Im Gegensatz dazu hat B die Kinder D und E und repräsentiert

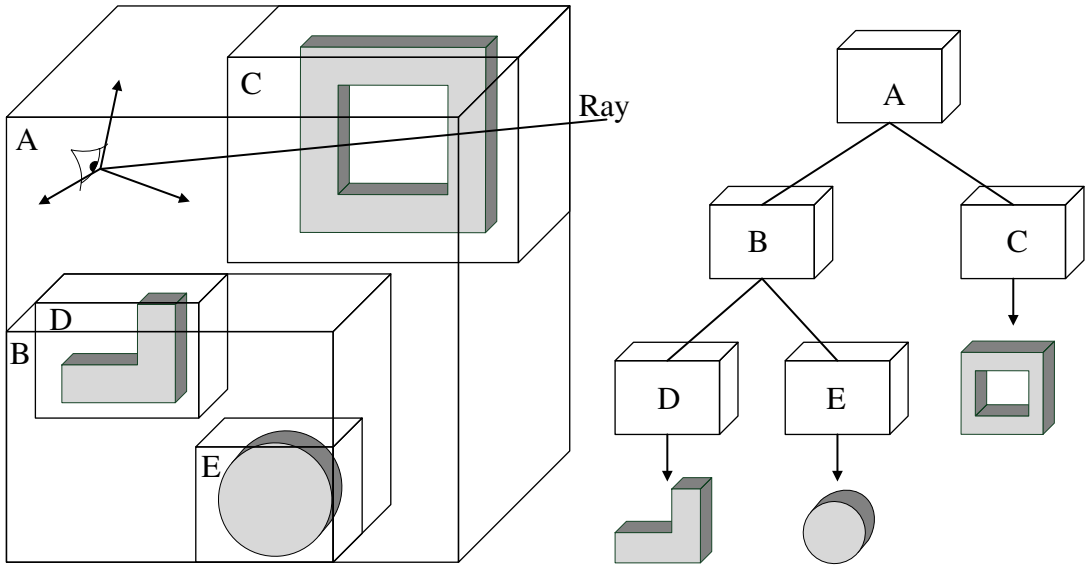


*Bild 2-45*: *Organisation der Szenenobjekte in einer BVH.*

somit kein Objekt der Szene, sondern einen räumlichen Bereich. D und E sind als Blätter schließlich wieder mit Szenenobjekten verknüpft.

Der Nutzen einer BVH offenbart sich, wenn man die Schnittprüfung eines Rays betrachtet, wie er beispielhaft in Bild 2-45 visualisiert wird. Anstelle der iterativen Schnittprüfung zwischen dem Ray und allen Hüllkörpern der Objekte (im Beispiel: C,D,E) wird die Hierarchie traversiert. Beginnend an der Wurzel zeigt sich, dass der Ray den Hüllkörper A schneidet. Dementsprechend werden anschließend dessen Kinder überprüft. Da der Ray den Hüllkörper B nicht schneidet, können alle unterhalb vom Knoten B organisierten Elemente bei der weiteren Traversierung vernachlässigt werden. Zwischen C und A findet sich eine Schnittmenge. Da C ein Blatt ist, wird schließlich die aufwendigere Kollisionsprüfung mit dem eigentlichen Szenenobjekt durchgeführt. Auf diese Weise kann die Anzahl der notwendigen Schnittprüfungen mit zunehmender Anzahl von Szenenobjekten drastisch reduziert werden. Statt des Aufwands $O(n)$ für den iterativen Durchlauf von $n$ Szenenobjekten ergibt sich bei Verwendung einer BVH eine mittlere Anzahl von $O(\log n)$ Prüfungen. Durch eine geschickte BVH-Organisation und Traversierungsreihenfolge kann der Effizienzgewinn in der Praxis noch deutlicher ausfallen [SR17].

Da Szenen typischerweise auch dynamische Objekte enthalten, muss die BVH in regelmäßigen Abständen neu organisiert werden. Ohne diese Updates würden die Hüllkörper der Baumknoten mit der Relativbewegung enthaltener Objekte an Volumen gewinnen. Dadurch erhöht sich die Anzahl der positiven Schnittprüfungen mit diesen Knoten, wodurch die notwendigen Traversierungsschritte zunehmen. Auch für diese Problemstellung gibt es zahlreiche effiziente Algorithmen [YL14].

Bisher wurde beschrieben, wie die Rays im Zusammenhang mit den Pixeln der Ausgabe stehen und wie ihre Verfolgung durch die Szene bis hin zur Kollision mit darin befindlichen Elementen auf effiziente Weise realisiert werden kann. Nun gilt es zu klären, wie die Hits zur finalen Pixelfarbe beitragen. Dazu sei erneut auf das Bild 2-43 verwiesen, in welchem beispielhaft zwei Primärrays visualisiert werden. Mit wenigen Ausnahmen ist nur der

erste Hit eines Rays von Bedeutung. Im Bild 2-43 entspricht das dem Hit 1. Nach der ersten Kollision mit einem Szenenobjekt wird der Ray entweder absorbiert, reflektiert oder abgelenkt. Bei der Absorption endet das Raytracing für den betrachteten Primärstrahl bereits an dieser Stelle. Im zweiten und dritten Fall werden neue Sekundärstrahlen erzeugt, die bei Hit 1 entspringen und in verschiedene, oft zufällige Richtungen ausgesendet werden. Mit zunehmender Anzahl der Sekundärstrahlen (auch: Samples) steigt die Qualität des Renderingergebnisses maßgeblich. Gleichzeitig wird die Rechenzeit jedoch exponentiell erhöht. Mathematisch kann das Absenden mehrerer Rays als die Approximation der Rendergleichung (2-28) durch die Monte-Carlo-Methode aufgefasst werden [HH75]. Sei der Integrand der Rendergleichung mit

$$f(v, \Omega, \Omega') := f_r(v, \Omega', \Omega) L_v(v, \Omega') \cos\theta_E \tag{2-40}$$

bezeichnet, so lässt sich das Integral mit der Monte-Carlo-Integration

$$\int_{\Omega' \in \Omega^+} f(v, \Omega, \Omega') \mathrm{d}\Omega' \approx \frac{1}{N_{MC}} \cdot \sum_{n=1}^{N_{MC}} \frac{f(v, \Omega, \Omega'_n)}{p(\Omega'_n)} \tag{2-41}$$

approximieren. Dabei sind $\Omega'_n$ Stichprobenelemente aus der oberen Hemisphäre $\Omega^+$ des Szenenpunktes $v$. $p(\Omega'_n)$ ist die Wahrscheinlichkeitsdichtefunktion, welche die Verteilung der Stichprobenelemente $\Omega'_n$ beschreibt, und $N_{MC}$ die Größe der Stichprobe. Betrachtet man den Hit eines Primärstrahls mit einem Szenenobjekt, so bezeichnet $v$ den Kollisionspunkt, $\Omega$ die Einfallsrichtung des Primärstrahls und $\Omega'_n$ die Richtung eines der $N_{MC}$ Sekundärstrahlen. Ist der Szenenpunkt $v$ Teil einer diffusen Oberfläche, sind Lichteinflüsse aus allen Richtungen relevant. Schließlich ist die Leuchtdichte eines Lambertschen Flächenelements unabhängig von der Betrachtungsrichtung (siehe Abschnitt 2.3.4). Dementsprechend könnte man die Stichprobenelemente $\Omega'_n$ für diffuse Flächen gleichverteilt wählen. Da das Volumen des Integrationsbereichs (obere Hemisphäre $\Omega^+$) dem Wert $2\pi$ entspricht, ergibt sich bei Gleichverteilung für die Wahrscheinlichkeitsdichtfunktion $p(\Omega') = \frac{1}{2\pi}$ und somit insgesamt das Integral

$$\int_{\Omega' \in \Omega^+} f(v, \Omega, \Omega') \mathrm{d}\Omega' \approx \frac{2\pi}{N_{MC}} \cdot \sum_{n=1}^{N_{MC}} f(v, \Omega, \Omega'_n).$$

Auch wenn diese Variante für ein hinreichend großes $N_{MC}$ zum richtigen Ergebis führt, kann es sinnvoll sein, die Wahrscheinlichkeitsdichtefunktion $p$ gezielt auf die Bereiche mit dem größten Einfluss zu konzentrieren. Wie aus Gleichung (2-11) hervorgeht, ist die übertragene Strahlungsleistung von einer Sender- auf eine Empfängerfläche dann maximal, wenn der Einfallswinkel der Strahlung auf der Empfängerfläche parallel zu ihrem Normalenvektor ist. Für andere Anordnungen nimmt sie mit dem Cosinus des Winkels $\theta_E$ ab. Schlussfolgernd ist es sinnvoll, die Hemisphäre $\Omega^+$ primär im Bereich kleiner Werte von $\theta_E$ abzutasten. Anstelle der Gleichverteilung wählt man für $p$ deshalb geschickter

$$p(\Omega') = \frac{1}{\pi} \cos(\theta_E) \text{ mit } \theta_E \in [0; \frac{\pi}{2}],$$

wobei $\theta_E$ den Winkel zwischen der Flächennormale $n$ und der Samplerichtung $\Omega'$ bezeichnet. Diese effizientere Art des Samplings wird auch als Importance Sampling bezeichnet.

Wie man durch Integration prüfen kann, erfüllt $p$ die Anforderung einer Wahrscheinlichkeitsdichtefunktion, denn es gilt

$$\int_{\Omega' \in \Omega^+} \frac{1}{\pi} \cos(\theta_E) \mathrm{d}\Omega' = \int_{\varphi_E=0}^{2\pi} \int_{\theta_E=0}^{\frac{\pi}{2}} \frac{1}{\pi} \cos\theta_E \sin\theta_E \mathrm{d}\theta_E \mathrm{d}\varphi_E = 1.$$

Auf diese Weise kann die Varianz der Approximation bei gegebener Stichprobengröße $N$ reduziert werden, sodass ein Gesamtbild der Szene im Umkehrschluss durch weniger Samples erzielt werden kann. Insbesondere bei der Berechnung spiegelnder Reflexionen ist die verwendete Verteilungsfunktion von maßgeblicher Bedeutung. Bei Verwendung des Phong-Beleuchtungsmodells empfiehlt sich unter Berücksichtigung der Gleichung (2-34) die Wahrscheinlichkeitsdichtefunktion

$$p(\Omega') = \frac{n_r + 1}{2\pi} \cos^{n_r}(\epsilon_{\Omega' r}),$$

wobei $\epsilon_{\Omega' r}$ den Winkel zwischen dem jeweiligen Sample $\Omega'$ und dem Vektor der perfekten spiegelnden Reflexion im Sinne der Gleichung (2-33) bezeichnet. Der Betrachter nimmt in der Gleichung (2-33) die Rolle der Lichtquelle ein. Das ist aufgrund der Helmholtz-Reziprozität (siehe Abschnitt 2.1.5) zulässig. Auf diese Weise werden hauptsächlich Samples generiert, deren Reflexionen näherungsweise dem Primärstrahl entsprechen. Diese haben für den spiegelnden Fall den größten Einfluss auf die Erscheinung des Szenenpunkts $v$ aus der Perspektive des Betrachters.

Die Monte-Carlo-Integration stellt somit eine Möglichkeit dar, Samples zu generieren. Sie kann sowohl zur Generierung mehrerer Primärstrahlen pro Pixel, als auch zur Erzeugung von Sekundärstrahlen, ausgehend von einem Hit, eingesetzt werden. Die ausgesandten Sekundärstrahlen kollidieren wiederum mit Szenenobjekten. Ihre Hits werden ebenfalls ausgewertet, bis schließlich bei einer definierbaren maximalen Rekursionstiefe abgebrochen wird. Diesen Ansatz nennt man Path Tracing. Neben der Stichprobengröße $N_{MC}$ ist die Rekursionstiefe ein wesentlicher Faktor für die Performance und Qualität des Renderings. Die Beiträge zu den Pixelfarben ergeben sich nach diesem Verfahren erst dann, wenn innerhalb der Rekursion ein Ray mit einer Lichtquelle kollidiert. Dazu müssen Lichtquellen entgegen ihrer bisherigen Auffassung im klassischen Rendering volumenbehaftet sein.

Um zu verdeutlichen, wie die Auswertungen an den verschiedenen Hits in unterschiedlichen Rekursionstiefen zur finalen Pixelfarbe beitragen, soll der folgende Pseudocode dienen. Initial wird der Algorithmus 1 aufgerufen. Er bestimmt basierend auf dem Szenegraphen, der Kameraposition und der Bildebene die Farben aller Pixel.

Der Parameter *countPrimRays* definiert die Anzahl der Primärstrahlen, die pro Pixel generiert werden sollen. Diese entspringen am Kameraursprung und durchstoßen das jeweilige Pixel $p$ der Bildebene an einem zufällig ausgewählten Punkt $q$. Sie werden durch den Algorithmus 2 verfolgt, welcher die Farbe, die sich bei Verfolgung des betrachteten Primärstrahls ergibt, zurückliefert. Die ermittelten Farben aller Primärstrahlen eines Pixels werden in der Variablen *col* summiert. Abschließend wird die Farbe des Pixel durch die Normierung der Summe auf die Anzahl der Primärstrahlen gesetzt.

Der wesentliche Path Tracing Gedanke verbirgt sich im Algorithmus 2. Dieser erwartet den Szenegraphen sowie den Ursprung und die Richtung des Rays als Parameter. Wird *traceRay* durch den Algorithmus 1 aufgerufen, so handelt es sich bei den übergebenen

**Algorithm 1** Render Image

```
1: require: scene as BVH-Scenegraph,  image containing pixels,  camOrigin ∈ ℝ³,
   countPrimRays ∈ ℕ^>0
2: function RENDERIMAGE(scene, image, cameraOrigin, countPrimRays)
3:     local variables: q, d ∈ ℝ³,  col ∈ [0, 1]³ RGB-Color
4:     for all Pixel p in image do
5:         col ← 0
6:         for i = 1 to countPrimRays do
7:             q ← randomPointInPixel(p)
8:             d ← norm(q − camOrigin)
9:             col ← col + traceRay(scene, camOrigin, d, 1)
10:        end for
11:        p.color ← col/countPrimRays
12:    end for
13: end function
```

Rayparametern um einen Primärstrahl. Da sich *traceRay* rekursiv erneut aufruft, kann es sich hierbei an späterer Stelle auch um einen Sekundärstrahl handeln. Innerhalb der Funktion wird zunächst geprüft, ob die maximale Rekursionstiefe bereits erreicht ist. In diesem Fall liefert *traceRay* die Farbe schwarz zurück. Das gilt auch, wenn der übergebene Ray nichts trifft. Hierzu wird die $firstHit$-Funktion eingesetzt, die den als BVH organisierten Szenegraphen traversiert. Sie liefert denjenigen Hit zurück, der die geringste Entfernung zum Rayursprung aufweist. Sollte kein Objekt getroffen werden, weist das Attribut $object$ des Hits den Wert *null* auf. Die Ausführung des Algorithmus endet in diesem Fall ebenfalls mit der Rückgabe der Farbe *schwarz*. Schreitet die Programmausführung über die Fallunterscheidung hinweg, liegt ein Hit vor. Durch die Funktion *randomDirection* wird eine zufällige Richtung *dir* für den darauf folgenden Sekundärstrahl bestimmt. Hierbei kann das Importance Sampling Anwendung finden, sodass nicht gleichverteilt über der oberen Hemisphäre des Hits, sondern insbesondere in den wesentlichen Einflussbereichen gesampelt wird. Der neue Ray wird ausgehend vom aktuellen Hit mittels eines rekursiven Aufrufs von *traceRay* in die zufällig bestimmte Richtung *dir* abgesendet. Der Parameter *recDepth* wird inkrementiert, um die Zunahme der Rekursionstiefe zu dokumentieren. Die Rückgabe von *traceRay* wird in der Variablen *indirectCol* zwischengespeichert. Sie repräsentiert das Licht, welches aus Richtung des Sekundärstrahls auf den aktuellen Hit trifft. Durch Anwendung eines BRDF kann ermittelt werden, welcher Anteil dieses Lichts in Richtung des vorhergehenden Strahls (bei $recDepth = 1$ ist es der Primärstrahl) fällt. Dieser Anteil entspricht dem Integranden der Rendergleichung (2-40) und wird in der Variablen $reflectCol$ gespeichert. Der Winkel $\theta_E$ liegt analog zu Gleichung (2-40) zwischen *dir* und der Flächennormalen am Hit. Gemäß der Monte-Carlo-Integration (2-41) muss der Integrand mit der Wahrscheinlichkeit *prob* des Samples (zweite Rückgabe von *randomDirection*) normiert werden. Die Division durch die Stichprobengröße entfällt, da im klassischen Path Tracing nur ein Sekundärstrahl generiert wird. Dementsprechend sind für eine hinreichend dichte Abtastung der Szene viele Primärstrahlen notwendig. Für den Fall, dass sich der Hit auf einer Lichtquelle befindet, wird auf die Rückgabe deren Leuchtdichte addiert, welche im Pseudocode durch das Attribut *emittance* des Hits abgefragt werden kann. Bei nicht leuchtenden Objekten gilt $emittance = 0$.

**Algorithm 2** Trace Ray

```
1: require: scene as BVH-Scenegraph, rayOrigin ∈ ℝ³, rayDirection ∈ ℝ³, recDepth ∈ ℕ^>0
2: function traceRay(scene, rayOrigin, rayDirection, recDepth)
3:     local variables: dir ∈ ℝ³, prob, reflectCoeff ∈ [0, 1], indirectCol, reflectCol ∈ [0, 1]³
4:     if recDepth = maxRecDepth then
5:         return black
6:     end if
7:     hit ← firstHit(scene, rayOrigin, rayDirection)
8:     if hit.object = null then
9:         return black
10:    end if
11:    (dir, prob) ← randomDirection(hit, rayDirection)
12:    indirectCol ← traceRay(scene, hit.position, dir, recDepth + 1)
13:    reflectCoeff ← BRDF(hit, rayDir, dir)
14:    reflectCol ← reflectCoeff · indirectCol · cos θ_E
15:    return reflectCol/prob + hit.emittance          ▷ light source ≡ emittance > 0
16: end function
```

Oft ist in der Praxis die notwendige Menge an Samples und die Tiefe der Rekursion zur Erreichung einer akzeptablen Qualität zu hoch, da die Chance, dass der verfolgte Pfad in einer Lichtquelle endet, abhängig von der Szenengeometrie sehr gering sein kann. Auf diese Weise liefern viele Rekursionszweige die Farbe schwarz zurück. Deshalb weicht man in der Praxis häufig von dieser stringenten Vorgehensweise ab. Stattdessen berechnet man die direkte Beleuchtung jedes Hits durch sogenannte Schattenstrahlen (Shadow Ray). Diese werden ausgehend vom Hit zu jeder Lichtquelle in der Umgebung gebildet. Falls sich der erste Hit eines Schattenstrahls auf der jeweiligen Lichtquelle befindet, ist sicher, dass der betrachtete Szenenpunkt nicht durch andere Objekte verschattet wird und somit das Licht dieser Lichtquelle direkt auf ihn Einfluss nimmt. Der Szenenpunkt wird dann wie im klassischen Rendering anhand des Beleuchtungsmodells ausgewertet. Das Ergebnis bestimmt zusammen mit den Rückgaben der Sekundärstrahlen die finale Erscheinung. Bild 2-46 stellt das Prinzip grafisch dar.

Zur Bestimmung der Farbe eines Pixels werden zufällig verteilte Primärstrahlen (z.B. $r$ in Bild 2-46) vom Augpunkt durch diesen Pixel abgesendet. Dessen erster Hit $h$ befindet sich auf der Kugel. Ausgehend von diesem Hit wird der Sekundärstrahl $r'$ in eine zufällige Richtung ausgesendet. Wahlweise kann hier Importance Sampling eingesetzt werden. Genauso wird ein Schattenstrahl $s$ vom Hit in Richtung Lichtquelle ausgesandt. Im Beispiel befindet sich der erste Hit des Schattenstrahls auf der Lichtquelle, sodass eine Verschattung des Hits $h$ ausgeschlossen werden kann. Das Beleuchtungsmodell wird somit für $h$ ausgewertet und zusammen mit den Ergebnissen des Sekundärstrahls akkumuliert. Der Sekundärstrahl $r'$ trifft zuerst den dargestellten Zylinder (Hit $h'$). Auch an dieser Position wird wieder ein Schattenstrahl $s'$ abgesendet, welcher die Lichtquelle erreicht. Die Lichtquelle nimmt auf die Farbe des Hits $h'$ und somit mittelbar auf die Farbe des Hits $h$ Einfluss, welcher letztlich zur Pixelfarbe beiträgt. Außerdem wird erneut ein Strahl $r''$ von $h'$ in eine zufällige Richtung abgesandt. Der Ray $r''$ wird zurück auf die Kugel reflektiert. Die Schattenprüfung

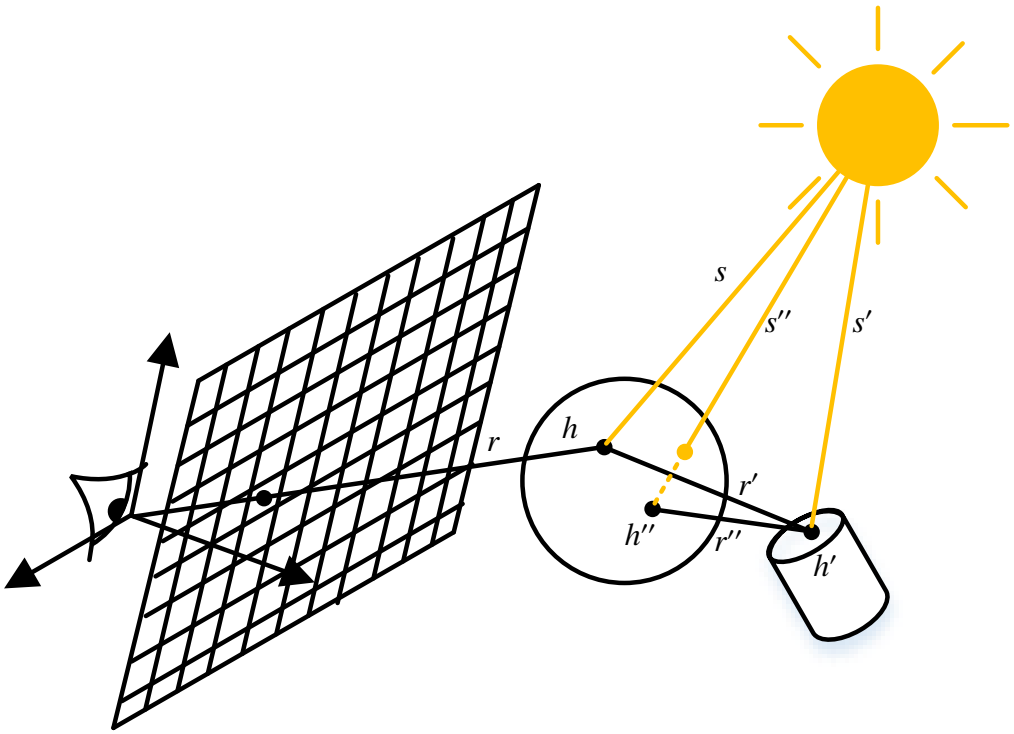


*Bild 2-46*: *Prinzip des Path Tracing.*

zeigt, dass ausgehend von Hit $h''$ kein direkter Sichtkontakt zur Lichtquelle gegeben ist. Sekundärstrahlen werden an $h''$ nicht generiert, da die maximale Rekursionstiefe erreicht ist.

Die Varianten der Raytracing Verfahren sind vielfältig. Alle bewegen sich im Spannungsfeld von Recheneffizienz und der korrekten Wiedergabe verschiedener optischer Phänomene, wie weichen Schatten, Reflexion, Brechung und Streuung von Licht oder der korrekten Berücksichtigung indirekter Beleuchtung. Einige bekannte Verfahren sind rekursives Raytracing, diffuses Raytracing, Light Raytracing oder bidirektionales Path Tracing (Kombination aus Path Tracing und Light Raytracing) [Gla07]. Die Tabelle 2-5 vergleicht die genannten Verfahren hinsichtlich dieser Kriterien und schließt die Übersicht der Raytracing Methoden zur globalen Beleuchtung ab.

### High Dynamic Range

Über lange Zeit wurden zur Speicherung der Farbinformation 8 Bit pro Farbkanal verwendet, wodurch 256 Helligkeitsabstufungen möglich waren. Auch wenn die meisten Ausgabegeräte bis heute über die gleiche Auflösung pro Farbkanal verfügen, stellt der geringe Dynamikumfang von 256 Abstufungen ein Hindernis in der Beschreibung von Szenen dar, welche gleichzeitig dunkle und helle Bereiche beinhalten. Schon während

*Tabelle 2-5*: *Vergleich verschiedener Raytracing Ansätze zur Beleuchtungsberechnung*

| Verfahren | Schatten | Reflexion/Streuung | Beleuchtung |
|---|---|---|---|
| Nur Primärstrahl und ein Schattenstrahl | Nur harte | Keine | Direkt |
| Rekursives Raytracing | Nur harte | Keine | Direkt und Spiegelung |
| Diffuses Raytracing | Vollständig | Spiegelnde Flächen | Direkt und Spiegelung |
| Path Tracing | Vollständig | Vollständig | Vollständig (globale Beleuchtung) |

der Pipeline-Verarbeitung sind helle Bereiche an der oberen Wertegrenzen abgeschnitten oder dunkle Bereiche durch Rundungsfehler verfälscht worden. Man hat deshalb das High Dynamic Range (HDR) Rendering (im Ggs. zu Low Dynamic Range Rendering mit 8 Bit) eingeführt, welches intern mit höherer Präzision rechnet. Mit DirectX 9 [Mic18] wurde die Unterstützung von mindestens 24 Bit im Fragment-Shader vorausgesetzt. Außerdem wurde die Kompatibilität zu Texturen mit 32 Bit pro Farbkanal erforderlich. Dabei wurden die Werte intern nicht mehr durch Fest- sondern Fließkommazahlen codiert, wodurch sich der darstellbare Dynamikumfang weiter erhöht.

Im HDR-Rendering wird die Kompatibilität zum Ausgabegerät, welches bis heute meist noch 8 Bit pro Farbkanal vorsieht (HDR-Monitore stellen hier eine Ausnahme dar), durch Tone Mapping sicher gestellt. Bei Tone Mapping handelt es sich um ein Dynamikkompressionsverfahren, welches den hohen Kontrast eines HDR-Bildes auf den Kontrast der Ausgabe reduziert. Die Farbwerte der Ausgabe werden meist auf das Intervall $[0, 1]$ normiert, sodass eine Mapping-Funktion $f_{TM}$ als Abbildung von HDR-Werten im Intervall $[0, \infty)$ auf das LDR-Intervall $[0, 1]$ verstanden werden kann. Eine besonders einfache Variante eines Tone Mappings entspricht

$$f_{TM} : \mathbb{R}^{\geq 0} \to [0, 1]; \quad c_{LDR}(c_{HDR}) = \frac{c_{HDR}}{c_{HDR} + 1}. \tag{2-42}$$

Nachteilig bei dieser Wahl ist der extreme Kontrastverlust für hohe Werte $c_{HDR}$ des HDR-Bilds, da diese nach dem Mapping sehr nah bei 1 und somit auch beieinander liegen. Für $c_{HDR} \to 0$ werden die Werte hingegen sehr gut differenziert. Diese Diskrepanz wird im grünen Verlauf des Bildes 2-47 wiedergegeben. Unvermeidbar ist, dass der Kontrast des Bildes durch das Tone Mapping verschlechtert wird. Ein häufig verwendetes und schnelles Verfahren ist die $\gamma$-Kompression, welche sich gemäß

$$f_{TM} : [0, A_{TM}^{-\frac{1}{\gamma_{TM}}}] \to [0, 1]; \quad c_{LDR}(c_{HDR}) = A_{TM} \cdot c_{HDR}^{\gamma_{TM}}.$$

beschreiben lässt. Die Parameter $A_{TM} > 0, A_{TM} \in \mathbb{R}$ und $0 < \gamma_{TM} < 1$ sind wählbar und müssen auf die vorliegenden Lichtverhältnisse abgestimmt werden. Sie definieren zum einen, bis zu welchem Maximalwert von $c_{HDR,max} = A_{TM}^{-\frac{1}{\gamma_{TM}}}$ noch Unterschiede im LDR-Bild sichtbar sein werden. Zum anderen beschreiben die Parameter, wie fein zwischen Werten im dunkleren Bereich gegenüber Werten im helleren Bereich differenziert wird. Der Plot 2-47 zeigt drei denkbare Parametersets der $\gamma$-Kompression für ein HDR-Bild, dessen maximale Helligkeit im Bereich von $c_{HDR,max} = 100$ liegt. Zudem stellt der mit „simpel“ bezeichnete Verlauf das Vorgehen nach Gleichung (2-42) dar.

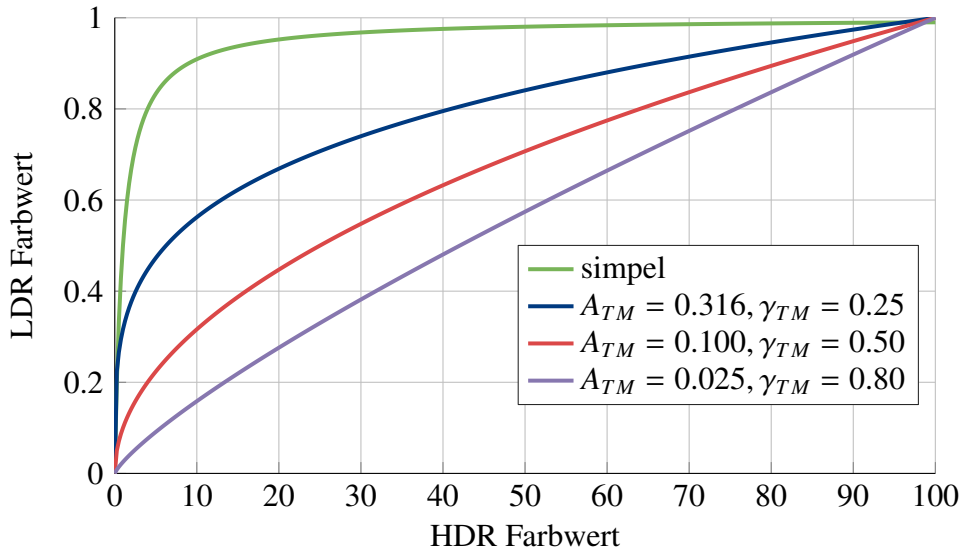


*Bild 2-47*: *Verschiedene Tone Mapping Funktionen zur Konvertierung von Farbwerten aus dem HDR in den LDR Bereich.*

Man stellt fest, dass für kleine $\gamma_{TM}$-Werte die Differenzierung der dunkleren Bereiche zunimmt, während die $\gamma$-Kompression in eine proportionale Beziehung zwischen HDR- und LDR-Wert übergeht, wenn $\gamma_{TM} \to 1$ gilt.

Die $\gamma$-Kompression ist nicht mit der $\gamma$-Korrektur zu verwechseln. Letztere wird unabhängig von HDR auch im normalen Rendering benötigt und berücksichtigt die logarithmische Wahrnehmung des Menschen. Nach der Stevensschen Potenzfunktion nimmt das menschliche Auge dunklere Bereiche im Vergleich zu hellen Bereichen differenzierter wahr [Ste57]. Der Zusammenhang zwischen wahrgenommener ($c_{sens}$) und physikalischer ($c_{phys}$) Helligkeit kann dabei ebenfalls durch eine $\gamma$-Kurve gemäß

$$c_{sens}(c_{phys}) = c_{phys}^{\gamma_e} \tag{2-43}$$

beschrieben werden, wobei dem Auge ein $\gamma$-Wert im Bereich $\gamma_e \in [0.35; 0.5]$ zugeordnet wird. Damit die Helligkeit eines linear arbeitenden Ausgabegeräts linear wahrgenommen wird, wenden Monitore intern eine $\gamma$-Korrektur an, um den Effekt nach Gleichung (2-43) auszugleichen. Der Standardwert für den sRGB-Farbraum wird mit $\gamma_{TM} = 2.2$ angegeben, sodass die Konkattenation der Wahrnehmung nach Gleichung (2-43) und der Ausgabegerät-Korrektur zu einem ungefähr linearen Zusammenhang zwischen dem tatsächlichen Helligkeitswert und der wahrgenommenen Helligkeit führt [WS82].

Die bisher erwähnten Tone Mapping Varianten sind sehr einfach und insbesondere global, d.h. an jeder Stelle im Bild wird die gleiche Funktion $f_{TM}$ angewendet. Neben diesen globalen Ansätzen existieren auch lokale Tone Mapper, welche zu noch besseren Ergebnissen führen können [EUM16].

### 2.3.5 General Purpose Computation on GPU

Mit den weitgehend frei programmierbaren Shadern innerhalb der Programmable Rendering Pipeline und der Unterstützung von Fließkommaoperationen seit 2001 gewann die

GPU auch abseits grafischer Anwendungen an Bedeutung. Mit der Flexibilität heutiger Shader können Berechnungen aus verschiedensten Bereichen von Wissenschaft und Technik erheblich beschleunigt werden. Voraussetzung für einen Performancegewinn gegenüber der klassischen Ausführung auf der CPU ist die hohe Parallelisierbarkeit der Algorithmen. Die Aufgabe muss sich in viele kleine Ausführungsstränge zerlegen lassen, welche untereinander keine oder zumindest wenige Abhängigkeiten haben. Unter dieser Voraussetzung lassen sich viele der Ausführungsstränge auf den Kernen der GPU gleichzeitig ausführen. Die Zweckentfremdung der Grafikkarte in Form solcher Anwendungen nennt man General Purpose Computation on GPU (GPGPU).

Mit dem Einsatz des Grafikchips in GPGPU-Anwendungen wurde auch die softwareseitige Unterstützung grafikferner Berechnungen weiter vorangetrieben. Mit Direct3D 11 und OpenGL Version 4.3 werden seit 2009 bzw. 2012 sogenannte Compute Shader unterstützt. Diese werden auf der Grafikkarte ähnlich den bereits vorgestellten Shader Typen abgearbeitet, sind dabei jedoch völlig losgelößt von der Grafikpipeline. Da Compute-Shader in OpenGL und Direct3D eingebettet sind, lassen sie sich einfach in Grafikanwendungen integrieren und eignen sich beispielsweise zur Vorberechnung von Informationen, welche in einem nachgelagerten Schritt bei der Visualisierung Berücksichtigung finden.

Mit der Einführung von CUDA und OpenCL in 2007 und 2008 wurden Schnittstellen geschaffen, mit welchen die Grafikkarte vollends abstrahiert als hochgradig parallele Recheneinheit gleichwertig zur CPU aufgefasst werden kann. Während CUDA eine Eigenentwicklung von NVIDIA darstellt und nur auf NVIDIA Hardware eingesetzt werden kann, stellt OpenCL einen allgemeinen Standard dar, der keinen Einschränkungen unterliegt. Mittels dieser Schnittstellen können einzelne Ausführungsstränge der Anwendung, beschrieben durch einen Kernel, dynamisch auf die zur Verfügung stehenden Rechenkapazitäten (Devices) aufgeteilt werden. Bild 2-48 zeigt die Architektur aus Sicht des OpenCL-Standards in stark vereinfachter Form. Der Bezug zur Grafikpipeline oder die manuelle Erstellung von Shadern in den entsprechenden Shader-Sprachen entfällt. Stattdessen abstrahieren OpenCL und CUDA die verfügbaren heterogenen Recheneinheiten, wie CPU und GPU, als Compute Devices mit verschieden vielen Compute Units. Die Implementierung kann deshalb vollständig in einer Hochsprache erfolgen. Aufteilbare Programmabschnitte müssen in sogenannten Kernels formuliert werden, welche strukturell Ähnlichkeit zu den zuvor beschriebenen Compute-Shadern aufweisen. Der zur Verfügung stehende Befehlssatz ist in den Kerneln jedoch deutlich umfangreicher. Die Verwendung verschiedener Recheneinheiten bringt den Nachteil mit sich, dass das Speichermanagement nicht mehr ohne Eingriff des Programmierers im Hintergrund erfolgt. Da die Host-Anwendung (Übergreifende Anwendung) und das Device (die zu verwendende Recheneinheit, meist GPU) verschiedene Laufzeitspeicher verwenden, muss der Programmierer die Kopie von Daten, die als Berechnungsgrundlage dienen, vom Host-Speicher (meist Arbeitsspeicher (RAM)) zum Device-Speicher (meist Arbeitsspeicher der GPU (VRAM)) explizit veranlassen. Nach der Berechnung müssen die Ergebnisse wieder zurück zum Host geleitet werden. Die Allokierung und Freigabe von Speicher obliegt dabei ebenfalls dem Entwickler. Diese Speichertransfers können bei großen Datenmengen einen wesentlichen Teil der gesamten Berechnungsdauer ausfüllen und stellen deshalb eine Einschränkung für den Einsatz der GPU in allgemeinen Berechnungen dar.

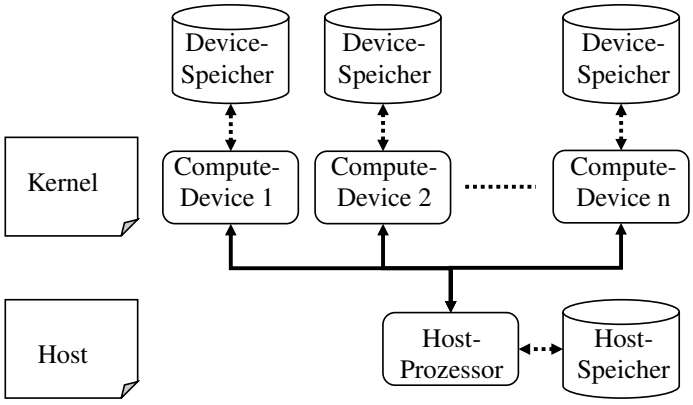


*Bild 2-48*: *Stark vereinfachte Sicht von Recheneinheiten und dedizierten Speicherbereichen in OpenCL.*

# 3 Stand der Technik

Nachdem im Kapitel 2 physikalische, physiologische und technischen Grundlagen, die zur Realisierung einer virtuellen Nachtfahrtsimulation von zentraler Bedeutung sind, ausführlich dargestellt werden, soll sich der Fokus nun auf den heutigen Stand der Technik richten. Dabei stehen moderne Pixel-Scheinwerfersysteme und die simulative Entwicklung dieser im Vordergrund.

## 3.1 Scheinwerfertechnik

Die Hauptscheinwerfer eines Kraftfahrzeugs erfüllen heutzutage eine Vielzahl von Aufgaben. Während zu Beginn des elektrischen Fahrzeugscheinwerfers die Ausleuchtung der vor ihm liegenden Straße bei Dunkelheit im Fokus stand, umfasst das Spektrum der sogenannten Lichtfunktionen inzwischen ein weitaus breiteres Feld. Angefangen mit den Abblend- und Fernlichtfunktionen über Tagfahrlicht, Kurvenlicht und der dynamischen Leuchtweitenregulierung bis hin zu modernen Lichtfunktionen, wie dem blendfreien Fernlicht, der Symbolprojektion, der optischen Spurführung oder des Markierungslichts, verfügen heutige Scheinwerfer über eine immense Leistungsfähigkeit.

### 3.1.1 Lichtverteilungen

Umgesetzt werden die verschiedenen Lichtfunktionen durch die geeignete Gestaltung der Lichtverteilung des jeweiligen Hauptscheinwerfers. Das Bild 3-1 zeigt das Messprinzip, mit dem die Lichtverteilung eines Scheinwerfers vermessen werden kann.

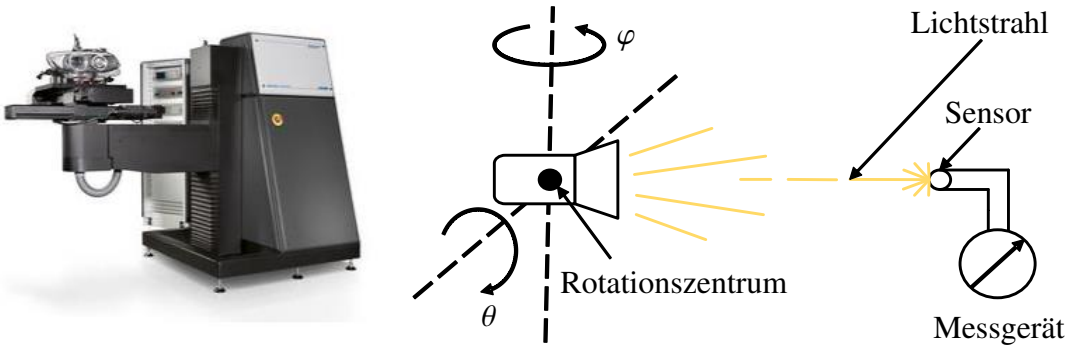


*Bild 3-1*: *Goniophotometer zur Aufzeichung von Lichtstärkeverteilungen von Hauptscheinwerfern (links: Verschwenkeinheit zur Orientierung des Scheinwerfers [Quelle: Instrument Systems Optische Messtechnik GmbH], rechts: Prinzipskizze eines Goniophotometers).*

Als zugrunde liegendes Koordinatensystem zur Beschreibung einer Lichtverteilung hat sich ein Kugelkoordinatensystem etabliert, dessen Ursprung in einem Referenzpunkt innerhalb des Scheinwerfers liegt. Die Abstrahlrichtung des Scheinwerfers kann über den Polarwinkel $\theta$ und den Azimutwinkel $\varphi$ beschrieben werden. Ein Winkelpaar $(\varphi, \theta)$ definiert somit eine Abstrahlrichtung, wobei die Richtung $(0°, 0°)$ per Konvention der Fahrtrichtung bzw. Vorwärtsrichtung entspricht und die Polachse parallel zur Gierachse bzw. Hochachse liegt.

Durch die Rotation des Scheinwerfers um seinen inneren Referenzpunkt bezüglich Polar- und Azimutwinkel, kann durch einen feststehenden Sensor das abgestrahlte Licht in allen Richtungen vermessen werden. Typischerweise wird die Lichtverteilung in Form einer Lichtstärkeverteilung erfasst, sodass die Lichtstärke $I_v$ (siehe Abschnitt 2.1.4) als Messgröße fungiert. Sind neben der Helligkeit auch spektrale Effekte von Bedeutung, so beschreibt man die Lichtverteilung des Scheinwerfers spektral und verwendet anstelle der Lichtstärke die X-, Y- und Z-Koordinaten nach den Vorgaben der CIE (2.2.6). Dabei ist die Lichtstärke in Form der Y-Koordinate als Teilinformation enthalten. Zur Visualisierung der abhängig vom Polar- und Azimutwinkel variierenden Lichtstärke wird häufig eine Falschfarbendarstellung eingesetzt, wie sie in Bild 3-2 visualisiert wird.

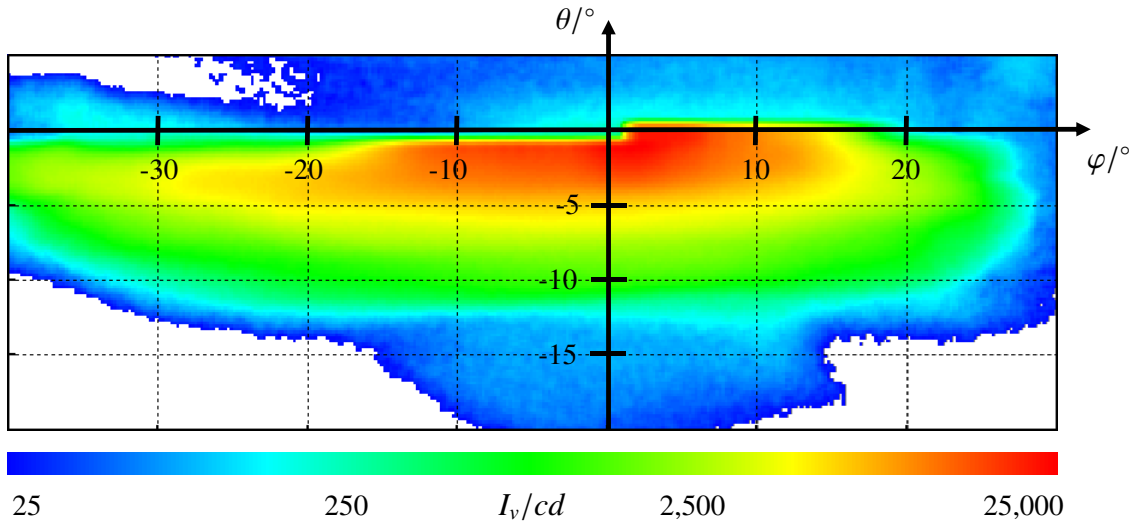


*Bild 3-2: Lichtstärkeverteilung eines HD84-Scheinwerfers mit aktiver Abblendlichtfunktion.*

Beispielhaft wird die Lichtverteilung der Abblendlichtfunktion eines linken HD84-Matrix-Scheinwerfers der HELLA dargestellt. Charakteristisch für jede Abblendlichtverteilung ist die nach rechts ansteigende Hell-Dunkel-Grenze (HDG). Die tiefere vertikale Lage der HDG im linken Bereich der Lichtverteilung stellt sicher, dass der Gegenverkehr auf der linken Fahrspur nicht geblendet wird, während die rechte Fahrspur weiter ausgeleuchtet werden kann, ohne andere Verkehrsteilnehmer zu behindern. Die Fernlichtfunktion ist hingegen symmetrisch zur $\theta$-Achse und hat eine noch höher gelegene HDG.

### 3.1.2 Aufbau und Prinzip

Die Realisierung der Lichtverteilungen für die verschiedenen Lichtfunktionen hat sich in der Vergangenheit maßgeblich verändert. Dabei gab es nicht nur technische Verbesserungen, sondern auch einen fundamentalen Wechsel der eingesetzten lichttechnischen Konzepte.

Die ersten elektrischen Komplettsysteme, bestehend aus Scheinwerfer, Lichtmaschine und Lichtmaschinen-Regler wurden von der Firma Bosch in 1913 mit dem „Bosch-Licht“ auf den Markt gebracht. Mit den ersten elektrischen Scheinwerfern koexistierten weiterhin petroleum- und acetylenbetriebene Scheinwerfer. Erst in den Zwanzigerjahren verdrängte die Elektrik die veralteten Systeme.

Im Jahre 1924 präsentierte die Firma Osram die Zweifaden-Glühlampe (auch Bilux-Lampe). Diese ermöglichte die Erzeugung zweier unterschiedlicher Strahlenbündel und erlaubte so erstmalig die Unterbringung der Abblend- und Fernlichtfunktion im selben Reflektor. Bis 1956 unterschieden sich Abblend- und Fernlicht nur in der Höhe der Hell-Dunkel-Grenze. Später wurde das Abblendlicht als asymmetrische Lichtverteilung mit nach rechts ansteigender Hell-Dunkel-Grenze erzeugt.

In den folgenden Jahrzehnten löste die Halogentechnik die bisherige Glühwendel-Lichtquelle ab. Erstmals kamen Halogen-Glühlampen unter der Bezeichnung H1 in Zusatzscheinwerfern von HELLA zum Einsatz. Ab 1965 wurden die H1-Lampen im Hauptscheinwerfer durch Zweikammersysteme für Abblend- und Fernlicht eingesetzt. Mit der Zweifaden-Halogen-Glühlampe H4, welche die Abblend- und Fernlichtfunktion im selben Reflektor ermöglicht, wurde die Bilux-Lampe 1971 abgelöst.

Das optische Konzept der bisher beschriebenen Scheinwerfer ist die Reflexionstechnik. Das zunächst undefinierte Licht, welches die Lichtquelle aussendet, wird durch den Reflektorschirm zur gewünschten Lichtverteilung geformt. Beispielhaft zeigt Bild 3-3 einen Scheinwerfer mit H4-Lampe, welcher über Abblend- und Fernlichtfunktion verfügt.

Durch die zwei zur Verfügung stehenden Glühfäden können bei H4-Lampen Abblend- und Fernlichtfunktion im selben Reflektor realisiert werden. Dazu verfügt der Glühfaden, welcher für das Abblendlicht genutzt wird (rote Ellipse in Bild 3-3), über eine Abdeckpfanne, welche die untere Reflektorhälfte abschirmt und nur Licht auf die obere Reflektorhälfte fallen lässt. An der oberen Reflektorhälfte wird das Licht vergleichsweise steil auf den Boden reflektiert, wodurch eine blendungskritische Ausleuchtung vermieden wird.

Im Fernlichtfall wird der Fernlicht-Glühfaden (graue Ellipse in Bild 3-3) bestromt. Dieser verfügt nicht über eine Abdeckpfanne und bestrahlt somit den gesamten Reflektorschirm, wie die schwarz gestrichelten Linien in Bild 3-3 visualisieren. Während der obere Teil des Reflektors weiterhin für die fahrzeugnahe Ausleuchtung von Bedeutung ist, dient das Licht, das an der unteren Reflektorhälfte reflektiert wird, zur Ausleuchtung des Fernbereichs.

Die tatsächliche Gestaltung der Lichtverteilung ist insbesondere im Abblendlichtfall komplizierter, als das Bild 3-3 suggeriert. Beispielsweise ist bisher noch nicht beschrieben worden, wie die Asymmetrie der HDG eines Abblendlichts technisch umgesetzt werden kann. Die älteste technische Umsetzung ist der Paraboloid-Scheinwerfer. Dessen Reflektor hat die Geometrie eines Paraboloiden und ist demnach verhältnismäßig einfach gestaltet. Die Lichtverteilung wird erst nach der Reflexion durch Brechung an der Abschlussscheibe

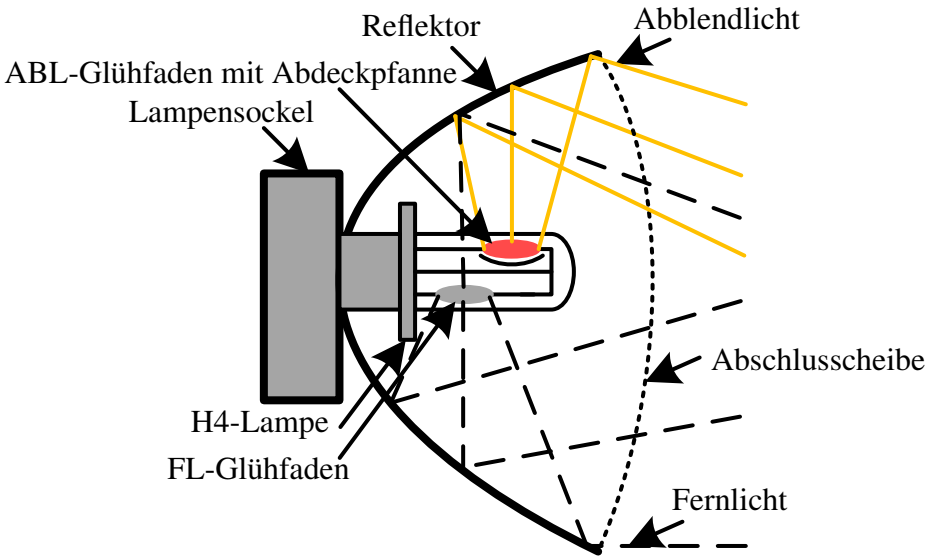


*Bild 3-3*: *Prinzip eines Hauptscheinwerfers mit H4-Lampe und Reflexionstechnik.*

gestaltet. Dazu weist die Abschlussscheibe eine entsprechende Profilierung auf, die das Licht horizontal (senkrechte zylindrische Profilierung) und vertikal (prismatische Profilierung) ablenkt. In Bild 3-4 ist ein solcher Paraboloid-Scheinwerfer eines Audi 100 dargestellt. Die Profilierung der Abschlussscheibe, die aufgrund ihrer optischen Funktion auch als Streuscheibe bezeichnet wird, ist gut erkennbar.

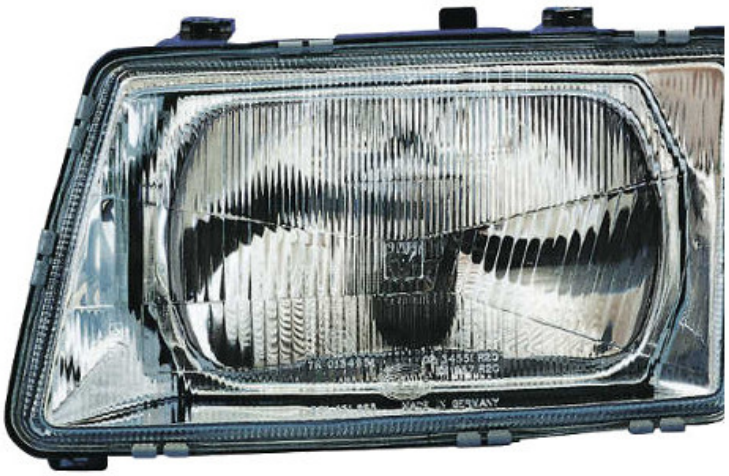

*Bild 3-4*: *Paraboloid-Scheinwerfer eines Audi 100 mit Einkammersystem (H4) [Quelle: HELLA].*

Abgelöst wurden die Paraboloid-Scheinwerfer durch Freiflächen-Scheinwerfer. Die Abschlusscheibe dieser Scheinwerfer weist keine optische Funktion auf. Sie dient lediglich dem Schutz vor Verschmutzung und Witterungseinflüssen. Die Gestaltung der Lichtverteilung wird ausschließlich durch den Reflektor vorgenommen. Die Form des Reflektors

wird computerunterstützt generiert. In Zweikammersystemen kann durch geeignete Form die gesamte Reflektorfläche für das Abblendlicht genutzt werden. Diese Eigenschaft sorgt für eine Verbesserung des Wirkungsgrads gegenüber Paraboloid-Scheinwerfern. Das Bild 3-5 zeigt einen Freiflächen-Scheinwerfer mit Einkammersystem eines VW Bora. Man erkennt die klare Abschlussscheibe und die verschieden orientierten Flächensegmente im Reflektorschirm.

*Bild 3-5*: *Freiflächen-Scheinwerfer eines VW Bora mit Einkammersystem (H4) [Quelle: HELLA].*

Mit der darauffolgenden Generation von Scheinwerfern kam es zu einem Wandel des lichttechnischen Konzepts. Anstelle der bisher angewandten Reflexionstechnik werden von nun an Projektionssysteme genutzt. Der Aufbau eines sogenannten dreiachsigen Ellipsoid(DE)-Scheinwerfers, in welchem die Projektionstechnik genutzt wird, ist in Bild 3-6 dargestellt.

Der Reflektor eines DE-Scheinwerfers wird nicht zur Gestaltung der Lichtverteilung eingesetzt. Seine Aufgabe ist es, das gesamte Licht der Lichtquelle in einem zweiten Brennpunkt zu bündeln. Kurz nach dem zweiten Brennpunkt ist eine Linse montiert. Diese verteilt das gebündelte Licht auf der Straße. Die Abblendlichtverteilung kann mit einem DE-Scheinwerfer realisiert werden, indem eine Blende kurz vor dem zweiten Brennpunkt vorgesehen wird. Diese Blende erzeugt durch ihre Geometrie die Hell-Dunkel-Grenze. Dabei ist zu beachten, dass die Lichtverteilung an der Blende horizontal und vertikal gespiegelt vorliegt. Konzeptbedingt verfügen Projektionsscheinwerfer über eine erheblich kleinere Lichtaustrittsfläche, als sie bei Reflexionsscheinwerfern benötigt wird. Deshalb werden sie auch unter Designaspekten bevorzugt verbaut. Eine Weiterentwicklung des DE-Scheinwerfers ist der Super-DE-Scheinwerfer, welcher auf die Blende verzichten kann, indem er die Lichtverteilung durch einen Freiflächen-Reflektor erzeugt. In diesem Fall kann nicht mehr von einem zweiten Brennpunkt, sondern nur noch von einem ausgedehnten Brennraum vor der Linse gesprochen werden. Aufgrund der fehlenden Blende, welche im DE-Scheinwerfer einen Teil des Lichts absorbiert, weist ein Super-DE-Scheinwerfer einen höheren Wirkungsgrad auf. Im Bild 3-7 ist der Scheinwerfer eines Skoda Superb zu sehen. Er verfügt über ein Freiflächen-Abblendlicht mit Reflexionstechnik und ein Super-DE-Projektionsystem für die Fernlichtfunktion. Der deutlich größere Flächenanteil des Reflexionssystems im linken Bereich ist deutlich erkennbar.

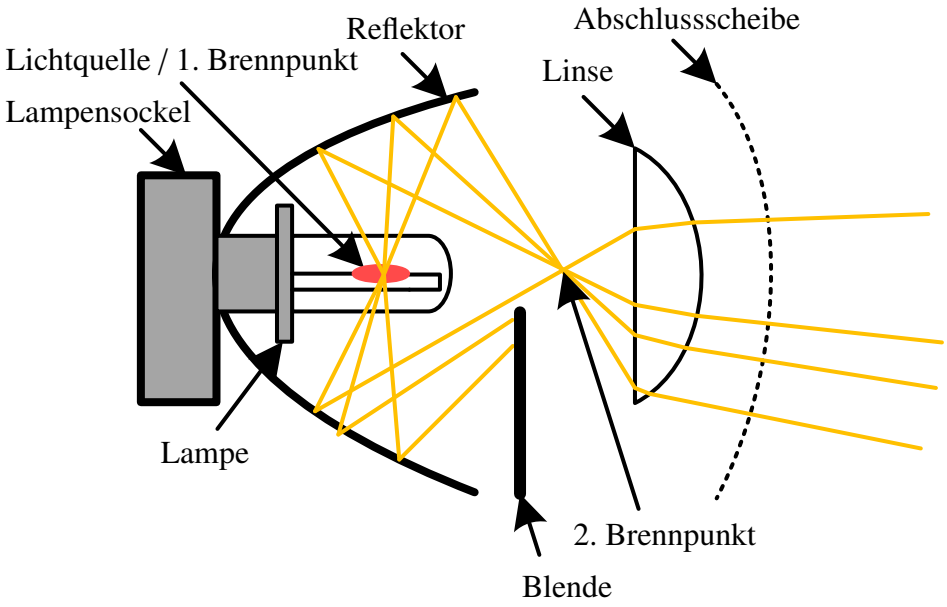


*Bild 3-6*: *Prinzip eines DE-Hauptscheinwerfers mit Projektionstechnik.*

### 3.1.3 Wandel durch Pixellicht

Die bisher vorgestellten Reflexions- und Projektionsscheinwerfer haben alle gemeinsam, dass die umgesetzten Lichtverteilungen statisch sind. Sie sind abhängig vom eingesetzten System durch die Streuscheibe, die Reflektorgeometrie oder die Blendengeometrie festgelegt. Trotzdem verfügen diese Systeme zum Teil über dynamische Lichtfunktionen, wie das Kurvenlicht. Die Dynamik wird dann allerdings durch ein mechanisches Verschwenken des Lichtsystems realisiert, wodurch nicht die Gestalt, sondern nur die Ausrichtung der Lichtverteilung angepasst werden kann. Auch blendfreies Fernlicht kann bereits durch die klassischen Systeme realisiert werden. Hierzu werden ebenfalls elektromechanische Aktuatoren (z.B. verfahrbare Blenden) im Scheinwerfer eingesetzt.

Den Durchbruch in der Gestaltungsfreiheit von Lichtverteilungen und insbesondere ihrer dynamischen Anpassung auf die aktuelle Fahrsituation wurde durch die Idee des Pixellichts geleistet. Der Gedanke einer Lichtverteilung, die durch die Zusammensetzung vieler, voneinander separierter Lichtbausteine, den Pixeln, gestaltet werden kann, wurde von Enders auf der PAL-Konferenz (heute ISAL) bereits 2001 vorgestellt [End01]. Der Hauptscheinwerfer kann dabei als Schwarz-Weiß-Beamer verstanden werden. Umgesetzt wurde dieser Gedanke erst wesentlich später und in kleinen Schritten.

Um den Jahreswechsel 2013/2014 sind mit dem Matrix-LED Scheinwerfer im Audi A8 und dem MULTIBEAM LED Scheinwerfer von Mercedes-Benz die ersten Systeme auf den Markt gekommen, die zumindest die Fernlichtfunktion durch mehrere LED-Lichtquellen abbilden und darauf aufbauend das blendfreie Fernlicht vollelektrisch realisieren [MMH15]. Aufgrund der geringen Auflösung (20 bis 30 einzeln ansteuerbare LEDs pro Scheinwerfer) werden andere Lichtfunktionen, wie das Abblend- oder Abbiegelicht durch

*Bild 3-7*: *Freiflächen-Reflexionssystem für die Abblendlichtfunktion und Super-DE-Projektionssystem für die Fernlichtfunktion im Hauptscheinwerfer eines Skoda Superb [Quelle: HELLA].*

Zusatzlichtquellen geleistet, sodass die Lichtverteilung im relevanten Winkelbereich des blendfreien Fernlichts hinreichend flexibel angepasst werden kann.

Der nächste Meilenstein zur Realisierung der Vision von Pixellicht wurde 2014 auf der VISION in Paris angekündigt [Kle14]. Er ging 2016 mit dem HD84-Matrix-LED Scheinwerfer der HELLA, nachfolgend als HD84-Scheinwerfer bezeichnet, mit der E-Klasse von Mercedes Benz in Serie [SP18]. Das HD84-System verfügt über 84 LED pro Scheinwerfer, welche in drei Zeilen angeordnet sind. Dabei verfügt die untere Zeile (Nahbereich) über 30 LED, die mittlere Zeile über 28 LED und die obere Zeile über 26 LED (Fernbereich). Anders als die bisher vorgestellten Systeme erfüllt die LED-Matrix im HD84-Scheinwerfer nicht nur die Fernlicht-, sondern auch die Abblendlichtfunktion. Letztere wird im Fahrzeugvorfeld durch vier statische Basislichter unterstützt, welche in Bild 3-8 gekennzeichnet sind. Das Kurvenlicht wird ebenfalls ohne mechanische Aktuatoren umgesetzt. Außerdem verfügt der Scheinwerfer über zusätzliche, einzeln ansteuerbare LED an der seitlichen Innenwand, mit welchen eine Abbiegelichtfunktion umgesetzt wird.

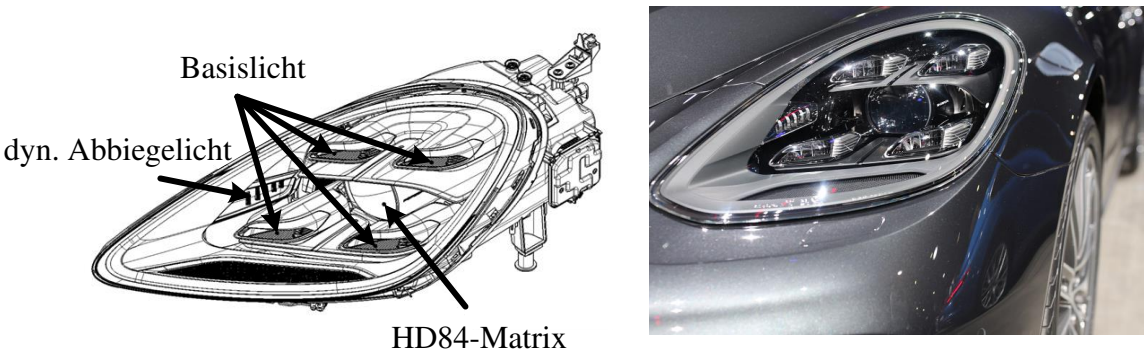


*Bild 3-8*: *HD84-Matrix-LED Scheinwerfer eines Porsche Panamera (links: Skizze, rechts: Foto) [Quelle: HELLA].*

Der durch die Matrix ausgeleuchtete Winkelbereich (auch: Field of View) erstreckt sich über 40° in der Horizontalen und 10° in der Vertikalen. So ergibt sich eine gemittelte Auflö-

sung von etwa $1.3° \times 3.3°$, wobei die horizontale Auflösung des Systems im Zentralbereich höher ist und in den Randbereichen abnimmt. Im Bild 3-9 sind die einzelnen Komponenten des HD84-Moduls dargestellt.

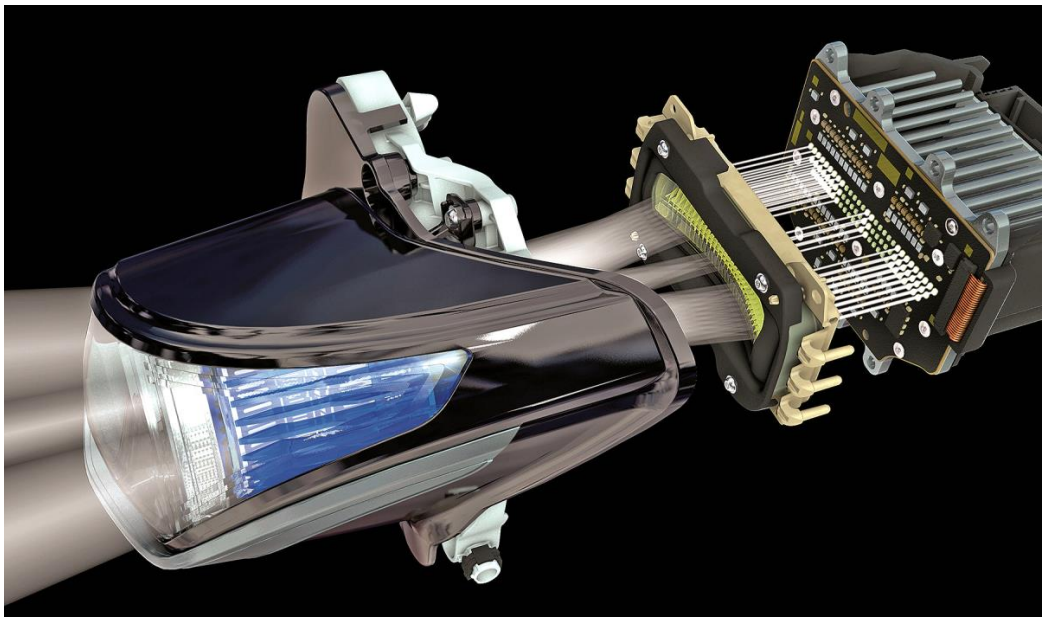

*Bild 3-9*: *Komponenten des HD84-Matrix-LED Moduls [Quelle: HELLA].*

Ganz rechts im Bild 3-9 befindet sich die Lichtquelle. Die 84 LED sind auf einer Platine mit weiterer Ansteuerungselektronik integriert. Rückseitig befindet sich ein Kühlkörper zur Ableitung der thermischen Energie. Das ausgesandte Licht der einzelnen LED wird durch eine Primäroptik aus Silikon voneinander separiert fokussiert und auf eine Linse (Sekundäroptik) geleitet, welche das Lichtbündel auffächert und in die Umgebung aussendet. Die 84 LED können nahezu kontinuierlich zwischen 0% und 100% ihres maximal zulässigen Stroms betrieben werden. Die einzustellenden Dimmwerte werden durch das Scheinwerfersteuergerät mit einer Taktung von 50 Hz vorgegeben.

Inzwischen zeichnet sich ab, dass auch der HD84-Scheinwerfer nur einen Zwischenschritt zum digitalen Licht darstellt. HELLA kündigte Ende 2019 an, die Lichttechnologie SSL|HD (Solid state lighting | high definition) für die Großserienproduktion vorbereitet zu haben [KPW19]. Auf dieser technischen Grundlage werden zehntausende von Lichtquellen pro Scheinwerfer realisiert. Erste Fahrzeuge, die mit diesem Lichtsystem ausgestattet sind, sind in den darauffolgenden drei Jahren zu erwarten. Derartig hoch aufgelöste Scheinwerfer eröffnen erneut die Möglichkeit neuartiger, frei programmierbarer Lichtfunktionen. Beispiele hierfür sind die optische Fahrspurmarkierung, Symbolprojektionen, weitere Individualisierungsmöglichkeiten wie Begrüßungs- und Verabschiedungsanimationen sowie optische Kommunikation mit anderen Fahrzeugen oder Fußgängern. Technisch basieren die SSL|HD-Scheinwerfer auf den Ergebnissen des geförderten Forschungsprojekts „$\mu$-AFS“, welches durch ein Konsortium aus Daimler, HELLA, OSRAM Specialty Lighting and Opto Semiconductors, dem Fraunhofer IZM und IAF sowie Infineon durchgeführt wurde [GPL+15], [MMFG16]. Anstelle der Montage einzelner LED-Lichtquellen auf

einer Platine (Vgl. HD84-Modul) wird dort ein adressierbares LED-Array eingesetzt. Ein einziger fingernagelgroßer $\mu$-AFS Chip verfügt über 1024 Pixel, wobei jedes einzelne unabhängig angesteuert werden kann. Im SSL|HD-Scheinwerfer sind mehrere LED-Arrays verbaut, deren Licht durch nachgeschaltete Optiken auf die Straße projiziert wird. Parallel arbeitet der französische Scheinwerferhersteller VALEO an dieser Technik und stellte in 2019 einen 4.000-Pixel-Prototypen vor [CCRP19]. Im Beitrag wird auch der funktionale Vorteil gegenüber den Matrix-Systemen herausgestellt.

Neben dem SSL|HD-System, welches als konsequente Weiterentwicklung und Miniaturisierung des Matrix-LED-Systems betrachtet werden kann, existieren konzeptionell neue Ansätze. Einer dieser Ansätze wurde im BMBF-geförderten Forschungsprojekt „VoLiFa2020 “erarbeitet [Hes15]. Das Resultat ist ein Scheinwerfer mit Liquid Crystal Display (LCD) Technik. Im Anschluss an das Projekt entwickelte HELLA den entstandenen Prototypen, dargestellt in Bild 3-10, zur Serienreife aus [DF17].

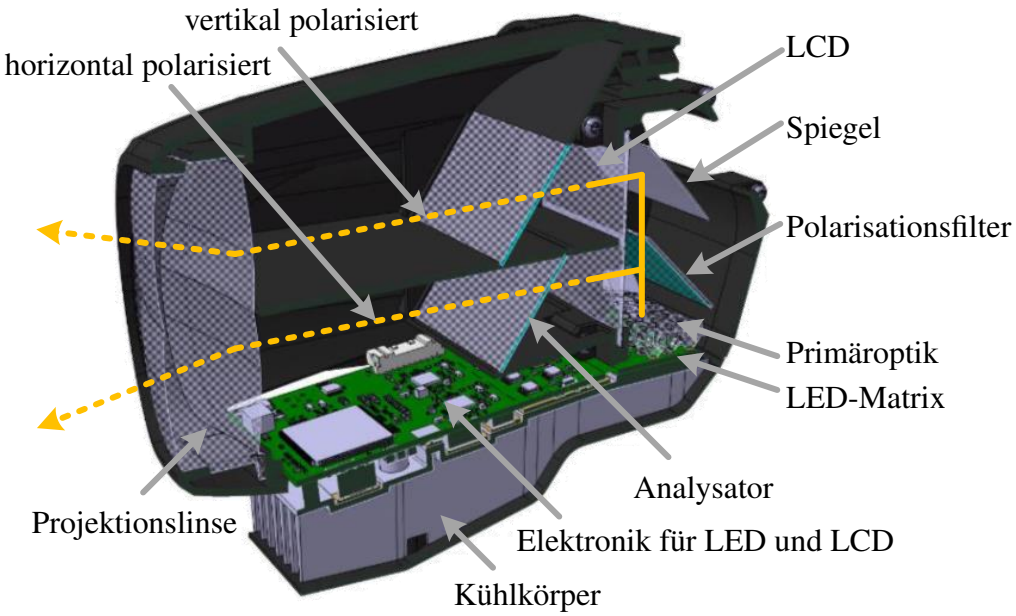


*Bild 3-10*: *Komponenten des LCD-HD-Moduls [Quelle: HELLA].*

Als Lichtquelle des LCD-Moduls werden 25 Hochleistungs-LED eingesetzt, welche sich unten rechts im Bild 3-10 befinden. Diese sind einzeln ansteuerbar, wodurch eine grobe Vorgestaltung der gewünschten Lichtverteilung erfolgt. Ihr Licht wird mittels einer Primäroptik auf einen im 45°-Winkel zur Lichteinfallsrichtung orientierten Polarisationsfilter fokussiert. Dieser spiegelt das horizontal polarisierte Licht und kann von dem vertikal polarisierten Licht durchdrungen werden. Letzteres trifft auf einen ebenfalls im 45°-Winkel positionierten Spiegel und wird dort reflektiert. Auf diese Weise trifft das Licht separiert nach seiner Polarisation auf LC-Displays. Abhängig von ihrer momentanen elektrischen Beschaltung drehen die LC-Displays die Polarisationsrichtung des Lichts, welches anschließend einen Analysator durchqueren muss, der das Licht nur in einer Polarisationsrichtung passieren lässt. Auf diese Weise kann die Lichtverteilung dynamisch angepasst werden. Der Prototyp des VoLiFA2020-Projekts verfügt über eine Auflösung von 0.1° bei einem ausleuchtbaren

Winkelbereich von 30° in der Horizontalen und 10° in der Vertikalen. Neben LCD findet man im gleichen Kontext auch häufiger die Bezeichung Liquid Crystal on Silicon (LCoS), wobei es sich im Wesentlichen um das gleiche Funktionsprinzip handelt. Während das Licht einen LCD durchdringt, wird es am LCoS reflektiert.

Hinsichtlich Effizienz ist das LCD-Modul dem SSL|HD-System unterlegen. Letzteres gestaltet die Gesamtlichtverteilung durch das variable Zuschalten einzelner Lichtquellen. Man spricht auch von einer additiven Komposition der Lichtverteilung. Das LCD-Modul hingegen erzeugt zunächst eine undefinierte Lichtverteilung, welche in einem nachgelagerten Schritt durch Absorption am LC-Display zugeschnitten wird. Das absorbierte Licht wird in Wärme umgesetzt und geht dem System verloren. In diesem Sinne gestaltet ein LCD-Modul die Lichtverteilung subtraktiv.

Die Auflösung des Systems bei gleichzeitig großem Winkelbereich ist für dieses frühe Entwicklungsstadium beachtlich, wie das im Lichtkanal der HELLA aufgenommene Bild 3-11 eindrucksvoll unter Beweis stellt.

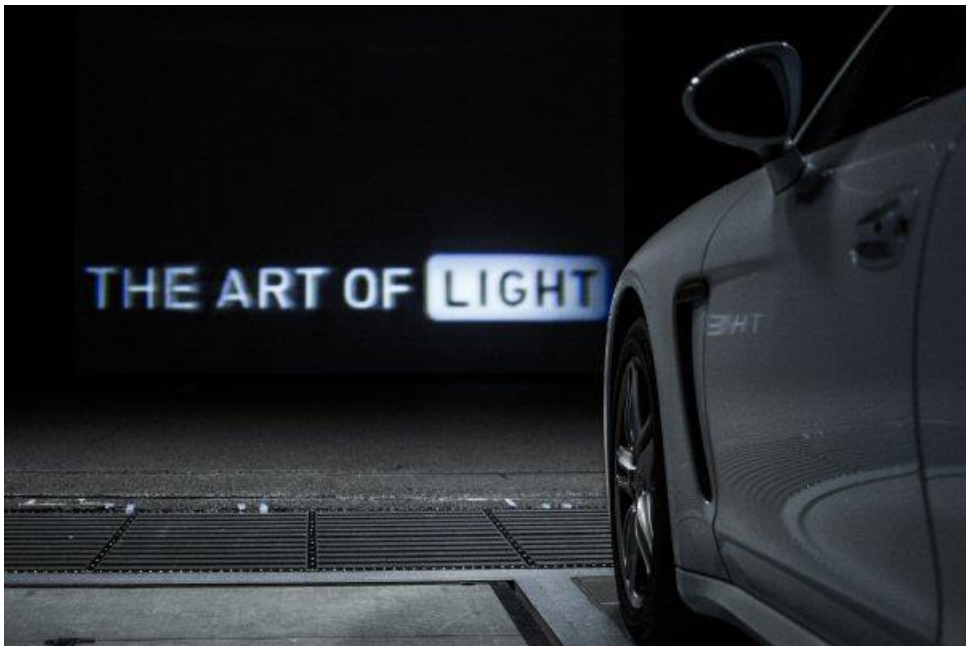


*Bild 3-11*: *Demonstration der Möglichkeiten eines LCD-Scheinwerfers [Quelle: HELLA].*

Ein weiterer Ansatz, der in Verbindung mit Fahrzeugscheinwerfern erstmalig in 2015 durch Texas Instruments prototypisch realisiert wurde, beruht auf der Digital Microscopic Mirror Device (DMD)-Technik [BB15]. Der serienreife Chip wurde in 2018 auf der Vision in Paris vorgestellt [FB18]. Dieses ursprünglich vom selben Unternehmen für Digital Light Processing (DLP)-Projektoren eingeführte Prinzip beruht auf einem Array von mikroskopischen Spiegeln, die elektromechanisch angesteuert und verkippt werden können. Das Bild 3-12 zeigt die Komponenten eines DMD-Moduls in stark vereinfachter Form.

Ausgehend von der Lichtquelle, die meist durch LED realisiert wird, wird das Licht durch eine Primäroptik auf das DMD fokussiert. Die einzelnen Mikrospiegel auf dem DMD

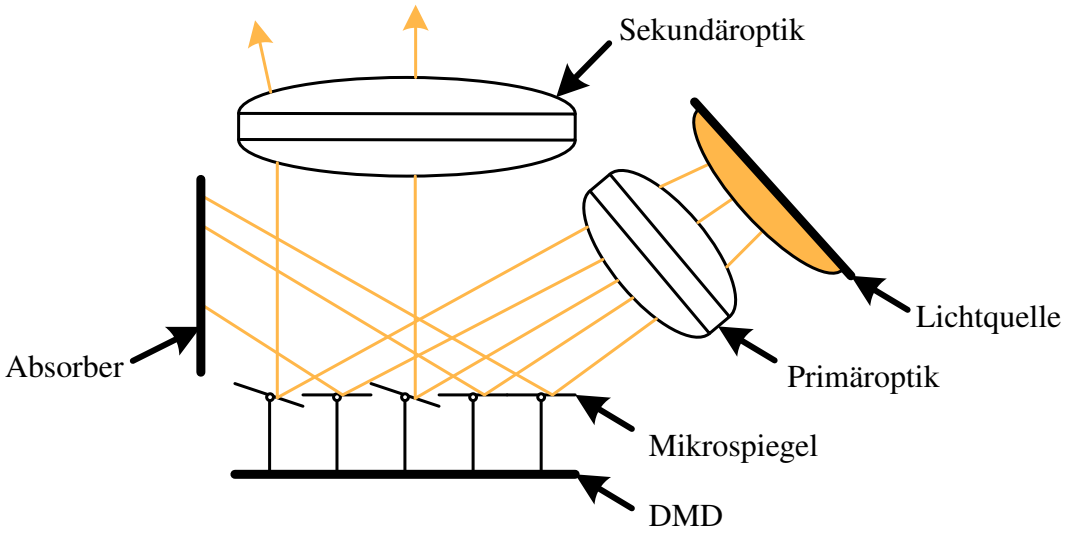


*Bild 3-12*: *Komponenten eines DMD-Scheinwerfer-Moduls.*

bilden die Pixel dieses Lichtsystems. Durch elektrische Beschaltung können die Spiegel verkippt werden. Die Mikrospiegel können dabei nur diskrete Kippwinkel erreichen. Ein kontinuierliches Umlenken von Licht ist also nicht möglich. Stattdessen muss entschieden werden, ob der Anteil des Lichtbündels, der auf den jeweiligen Mikrospiegel fällt, durch die Sekundäroptik auf die Straße projiziert werden soll oder stattdessen durch eine Absorberfläche innerhalb des Scheinwerfergehäuses aus der Lichtverteilung entfernt werden soll. Die DMD-Technik zählt somit zu den subtraktiven Verfahren. Hinzu kommt, dass insbesondere die Sekundäroptik recht komplex ist und eine Gesamtzahl von fünf bis sechs Linsen innerhalb eines DMD-Scheinwerfers erfordert. Ein weiterer Nachteil von DMD-Systemen ist das geringe Field of View, welches aufgrund der geringen optischen Effizienz des Systems vorausgesetzt werden muss, um die erforderliche Beleuchtungsstärke im spezifizierten Winkelbereich erbringen zu können. Hinzu kommt, dass die anfangs eingesetzten Hochleistungs-LED gleichmäßige Lichtverteilungen emittieren, welche von der gewünschten Gestalt von Scheinwerferlichtverteilungen abweichen. Abhilfe kann hier durch die Verwendung von Laserdioden (LD) geschaffen werden, wie Texas Instruments in 2018 zeigt [FB18].

Trotz der vielen Nachteile spricht die extrem hohe Auflösung in der Größenordnung von $10^6$ Pixeln für die Verwendung eines solchen Systems in einem kleinen Winkelbereich mit hohen Anforderungen an die Anpassung der Lichtverteilung, während die umliegenden Bereiche durch andere Systeme beleuchtet werden müssen. Im Bild 3-13 zeigt Mercedes, wie die gewonnene Größenordnung der unterstützten Auflösung Wege für neuartige Lichtfunktionen ebnet. Die erste Kleinserie von Fahrzeugen mit DMD-Technik im Scheinwerfer hat Mercedes-Maybach in 2018 auf den Markt gebracht [Ros18]. Unterstützt wird das in einem kleinen Winkelbereich agierende DMD-Modul durch ein HD84-Modul und drei Basislichter.

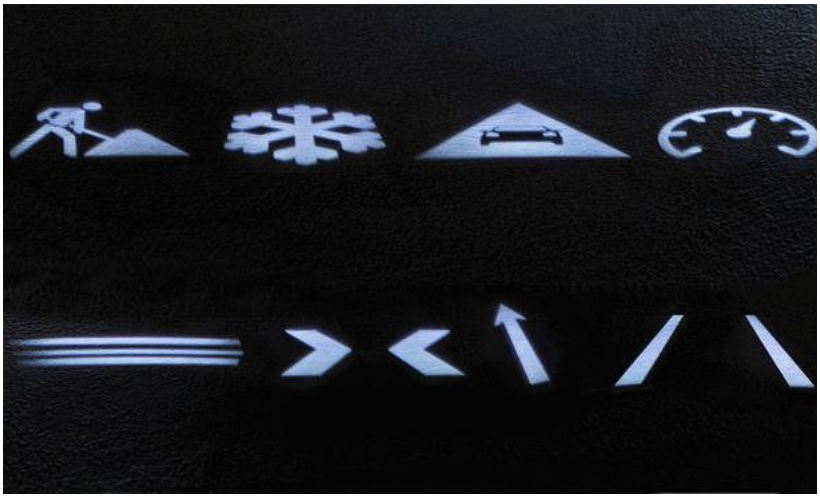

*Bild 3-13*: *Demonstration der Möglichkeiten eines DMD-Scheinwerfers [Quelle: Mercedes].*

Das neueste Konzept zur Realisierung hochauflösender Scheinwerfer stellen Laser-Scanner dar. Der Grundstein für diese Technologie wurde durch das BMBF-geförderte Projekt „Intelligentes Laserlicht für kompakte und hochauflösende adaptive Scheinwerfer“ (iLaS) im Zeitraum von 2015 bis 2017 gelegt [PHG+15], [AHW+16], [HSB+17]. Das Projektkonsortium umfasste die Firmen Audi, Bosch und Osram, sowie das Karlsruher Institut für Technologie (KIT) und die österreichische Firma ZKW. Im Rahmen des Projekts sind erste Demonstratoren mit verbleibenden technischen Schwierigkeiten entstanden. Von 2016 bis 2019 entwickelten der Scheinwerferhersteller HELLA und das Fraunhofer-Anwendungszentrum für Anorganische Leuchtstoffe in Soest im Rahmen des Förderprojekts "Hochinnovative pixelierte Leuchtstoffe für laserbasierte Emissionen im Scheinwerfer"(HipE) des Europäischen Fonds für regionale Entwicklung (EFRE) einen Prototypen, wobei sie schwerpunktmäßig den Phosphor-Konverter optimierten [SHSN18]. Dieser wandelt das insbesondere bei Nacht kaum sichtbare, blaue Laser-Licht in weißes Licht um und hat hohen Einfluss auf den Kontrast, welcher bei Laser-Scanner-Systemen stets als kritische Zielgröße einzustufen ist. Wie auch in den zuvor beschriebenen HD-Scheinwerfersystemen sind die detaillierten physikalischen Aspekte komplex und können auch im Fall des Laser-Scanner-Verfahrens anhand der Skizze 3-14 nur oberflächlich vermittelt werden. Die Laserdiode erzeugt zunächst ein schmalbandiges, hochenergetisches ($\lambda \approx 450\,nm$) Licht, welches ähnlich zur DMD-Technologie auf einen Mikrospiegel (MEMS) trifft, der durch elektrische Signale mechanisch um zwei Rotationsachsen verkippt werden kann. Anders als beim DMD-System genügt der Laser-Scanning-Technologie jedoch ein einziger Spiegel zur Gestaltung der gesamten Lichtverteilungen. Dazu muss der Mikrospiegel ein kontinuierliches Spektrum von Kippwinkel anfahren können. Alternativ lassen sich zwei sequentiell gekoppelte Spiegel verwenden, die jeweils um eine Achse drehbar sind. Das vom Spiegel reflektierte Licht trifft auf einen Leuchtstoff und regt darin befindliche Atome zu höheren Energieniveaus an, welche nach der Anregung Photonen emittieren um in ihren Ruhezustand zurückzugelangen. Der Leuchtstoff ist für die Anwendung so gewählt, dass die vom Leuchtstoff emittierten Photonen in Summe ein weißes Licht erzeugen. Das Laserlicht selbst darf den Scheinwerfer aus Sicherheitsgründen nicht verlassen und wäre

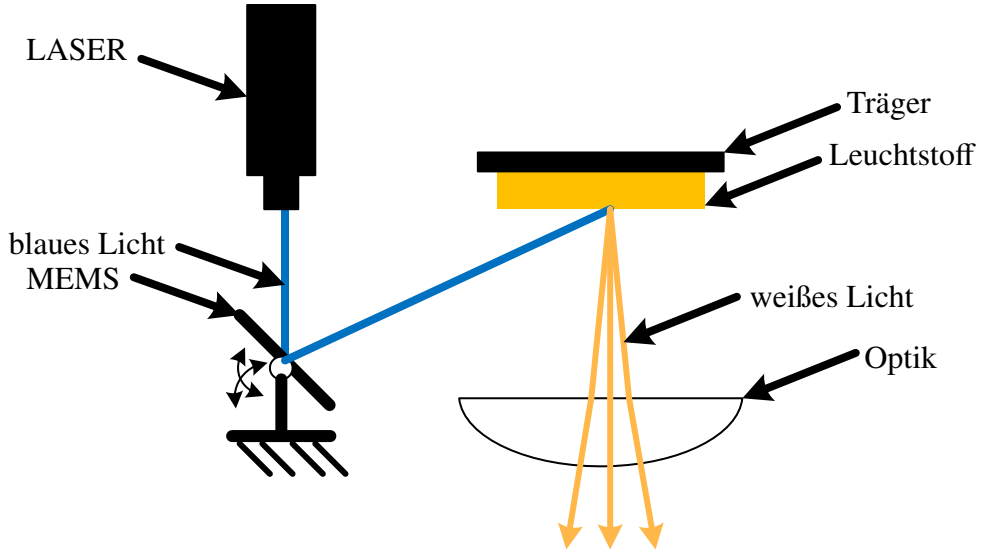


*Bild 3-14*: *Komponenten eines Laser-Scheinwerfer-Moduls.*

aufgrund seiner Wellenlänge für die Ausleuchtung der Straße nicht gewinnbringend (Vgl. $V(\lambda')$-Kurve in Bild 2-9) [LLT+19]. Die vom Leuchtstoff emittierten Photonen werden schließlich über eine Optik auf die Straße projiziert. Durch sehr schnelle Bewegungen des MEMS lässt sich die gesamte zweidimensionale Lichtverteilung beispielsweise zeilenweise scannen. Die Lichtstärke in die jeweilige Raumrichtung kann dabei durch die Belichtungszeit und die Leistung der Laserdiode frei gewählt werden. Konzeptionell handelt es sich also um eine Pulsweitenmodulation in jeder Raumrichtung. Bei hinreichend schnellem Scan nimmt das Auge lediglich den photometrischen Mittelwert wahr.

Ein zentraler Forschungsschwerpunkt ist die Verbesserung der Kontrastschärfe von Laser-Scannern, da das vom Leuchtstoff emittierte Licht nicht so definiert und punktförmig austritt, wie der zuvor aufgetroffene Laserstrahl in den Leuchtstoff eingedrungen ist. Im Projekt „HipE“ konnte der Kontrast bereits deutlich verbessert werden, indem man den Leuchtstoff durch eine Gitterstruktur in kleine Flächenelemente diskretisiert hat.

Ein Alleinstellungsmerkmal der Laser-Scanner-Technologie gegenüber allen anderen beschriebenen Technologien ist die erzielbare Leuchtdichte. Sowohl bei additiven als auch subtraktiven Verfahren muss der insgesamt zur Verfügung stehende Lichtstrom der Lichtquelle bereits beim Design des Scheinwerfers auf den auszuleuchtenden Raumwinkelbereich verteilt werden. Dynamisch kann nur der momentan genutzte Anteil der realisierbaren Gesamtlichtverteilung angepasst werden. Beim Laser-Scanner hingegen kann der maximale Lichtstrom, welcher durch die Leistung der Laserdiode vorgegeben ist, zu jedem Zeitpunkt neu auf den relevanten Winkelbereich verteilt werden. Aus diesem Grund erreichen Laser-Scanner höhere Leuchtdichten als die anderen vorgestellten Verfahren. Diese Eigenschaft macht sie insbesondere für Fernlichtfunktionen besonders interessant, da sie die Reichweiten konkurrierender Systeme deutlich übertreffen. Das Bild 3-15 zeigt den Nutzen dieses Alleinstellungsmerkmal am BMW i8. Es sei darauf

hingewiesen, dass dieses Modell nur über ein statisches Laser-Licht verfügt. Adaptive Laser-Scanning-Systeme sind bislang nicht in Serie.

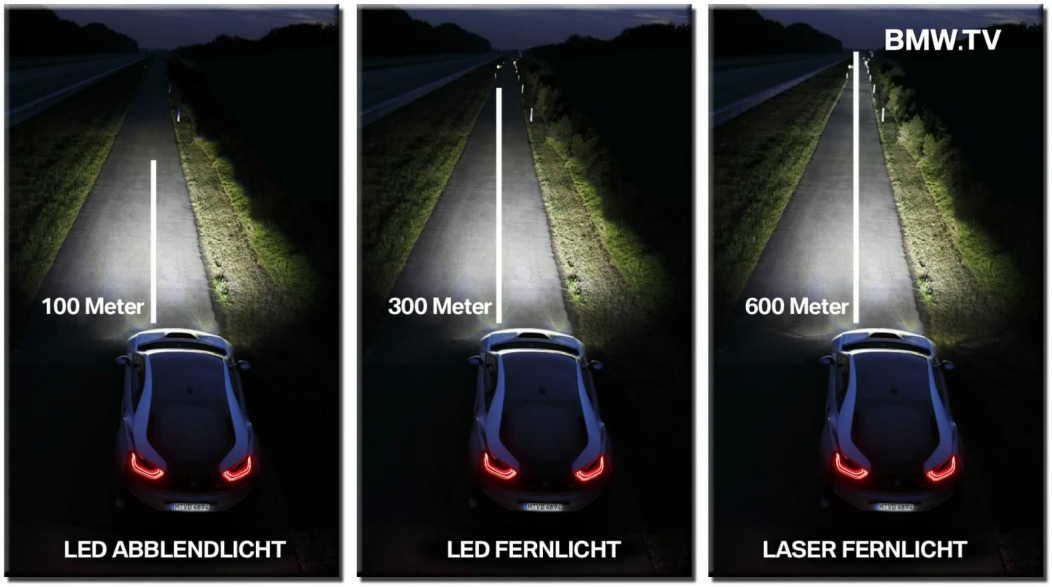


*Bild 3-15*: *Vergleich des Fernlichts eines LED- und eines LD-Scheinwerfers [Quelle: BMW].*

Nachdem die vier momentan diskutierten Technologien zur Realisierung von hochauflösenden Scheinwerfer-Systemen vorgestellt sind, fasst Tabelle 3-1 die wesentlichen Eigenschaften der verschiedenen Ansätze zusammen und vergleicht sie in übersichtlicher Form. Hierbei handelt es sich um eine vereinfachte Zusammenfassung einer umfangreichen Studie von Knöchelmann et al. [KHKL19].

*Tabelle 3-1*: *Wesentliche Eigenschaften verschiedener lichttechnischer Konzepte zur Realisierung hochauflösender Scheinwerfer (Zusammenfassung aus [KHKL19]).*

| Technik | SSL\|HD | LCD | DMD | Scanner |
|---|---|---|---|---|
| Auflösung | $10^4$ | $10^4$ | $10^6$ | anders definiert (hoch) |
| Winkelbereich | groß | mittel | klein | beliebig (meist klein) |
| Komposition | additiv | subtraktiv | subtraktiv | additiv |
| Lichtquelle | LED | LED/LD | LED/LD (weiß) | LD (blau) |
| Opt. Effizienz | hoch | niedrig | niedrig | hoch |
| Komplex. des Linsensystems | mittel | gering | hoch | gering |

## 3.2 Lichtfunktionen von HD-Scheinwerfern

Die im vorhergehenden Abschnitt vorgestellten optischen Systeme stellen die technische Grundvoraussetzung für die nachfolgend beschriebenen Lichtfunktionen dar. Doch erst durch raffinierte und hochgradig adaptive Lichtfunktionen schaffen die Potentiale, die durch HD-Systeme erschlossen werden, einen Gewinn für die Fahrzeuginsassen und andere Verkehrsteilnehmer. Zunächst werden Lichtfunktionen beschrieben, die in aktuellen Fahrzeugen bereits Anwendung finden könnten. Die größte Hürde stellt hierbei nur noch die rechtliche Zulassung dieser neuen Technologien dar. Abschließend wird ein Ausblick auf zukünftige Lichtfunktionen gegeben, die auf wissenschaftlichen Konferenzen bisher nur konzeptionell vorgestellt wurden. Beispiele hierfür sind Lichtfunktionen im Kontext des autonomen Fahrens und der Einsatz von maschinellen Lernmethoden.

Durch HD-Systeme können sowohl bestehende Lichtfunktionen, wie Abblend- und Fernlicht, signifikant optimiert, als auch neuartige fortgeschrittene Lichtfunktionen, wie Straßenprojektionen oder der optische Fahrspurassistent, etabliert werden. Beide Aspekte führen zu einem Sicherheits- und Komfortgewinn. Das System umfasst dabei je nach Lichtfunktion neben dem Scheinwerferpaar verschiedene Sensoren und komplexe Software, die auf dem Steuergerät des Scheinwerfersystems implementiert ist. Erst im Zusammenspiel aller Komponenten gemäß Bild 3-16 kann eine optimale Situationsadaptivität realisiert werden. In diesem Abschnitt werden ausschließlich Forschungsergebnisse mit der Lichtfunktion im Fokus vorgestellt. Aus Gründen der Übersichtlichkeit wird auf die Diskussion der Sensorik und der Perzeption auf Basis der Sensorrohdaten verzichtet. In 2019 präsentie-

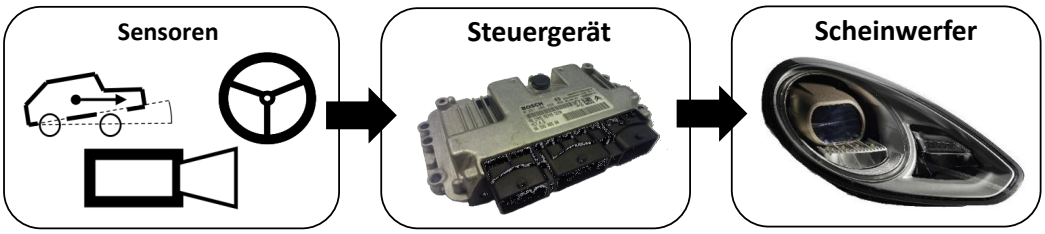


*Bild 3-16*: *Komponenten eines adaptiven Scheinwerfersystems.*

ren Roth et al. die Möglichkeiten eines Micro-LED-Systems anhand eines Demonstrators [RTH+19]. Sie zeigen auf, dass die quadratischen Seitenverhältnisse aktueller LED-Arrays nicht die Anforderungen einer geeigneten Straßenausleuchtung erfüllen. Durch die horizontale Anordnung mehrerer LED-Arrays erzielen sie mit einem horizontalen Field of View (HFOV) von 30° und einem vertikalen Field of View (VFOV) von 10° ein für die Anwendung optimiertes Seitenverhältnis von 3:1. Durch die asymetrische Überlagerung beider Scheinwerfer können sie den HFOV des Gesamtsystems auf 40° ausdehnen, womit der erforderliche Winkelbereich für die meisten HD-Anwendungen bis zu einem Kurvenradius von mindestens 100m gegeben ist [GCN16], [KKBK17]. Gleichzeitig erreichen sie eine Auflösung von 0.3° in der Horizontalen und 0.15° in der Vertikalen. Auf Grundlage dieses und ähnlicher Scheinwerfersysteme leiten Roth et al. mögliche Lichtfunktionen auf Basis von Unfallstatistiken ab. Bei 85% aller nächtlichen Unfälle handelt es sich um Kollisionen zwischen Fahrzeugen an Kreuzungen bzw. bei Wendemanövern oder mit Fußgängern, Auffahrunfälle oder Unfälle aufgrund mangelnder Spurhaltung [GID18]. Zur Prävention dieser Unfalltypen leiten Roth et al. die in Bild 3-17 dargestellten Lichtfunktionen ab.


*Bild 3-17*: *HD-Lichtfunktionen zur Vermeidung der häufigsten nächtlichen Unfalltypen nach Roth et al. [RTH+19].*

Die in den Bildteilen 3-17 a und b visualisierten Lichtfunktionen unterstützen bei der Spurführung. In Bildteil a handelt es sich um ein Baustellenlicht. Diese Lichtfunktion erhält zur Zeit in mehreren Forschungsarbeiten besondere Aufmerksamkeit. Durch die hellen Streifenprojektionen kann der Fahrer die Breite des Fahrzeugs besser einschätzen und somit besser durch schmale Fahrspuren, wie sie häufig auf Autobahnbaustellen vorkommen, navigieren. Im Bildteil b wird der Fahrer auf das Verlassen der Fahrspur hingewiesen. Der nach rechts deutende Pfeil visualisiert die Notwendigkeit einer Lenkbewegung.

Im Bildteil c wird eine typische Kreuzungssituation dargestellt. Das rote Fahrzeug möchte in die Fahrspur des Egofahrzeugs (eigenes Fahrzeug in Abgrenzung zu anderen Verkehrsteilnehmern) eintreten. Durch einen hellen Streifen an der Einmündung wird einerseits der Fahrer auf dieses Fahrzeug aufmerksam gemacht und andererseits dem anderen Fahrzeug signalisiert, dass die Vorfahrt des Egofahrzeugs zu achten ist. Schließlich stellt Bildteil d die Markierung eines Fußgängers dar.

Die vorgestellten Lichtfunktionen sind so gewählt, dass ein Mikro-LED-System, wie es durch Roth et al. beschrieben wird, über eine hinreichende Auflösung zur Realisierung der notwendigen Projektionen und Maskierungen verfügt.

Eine weitere Studie zu den Möglichkeiten von Mikro-LED-Systemen wird durch Lee vorgestellt [Lee19]. Dieser nennt zunächst die in den letzten Jahren entwickelten Lichtfunktionen und kategorisiert sie anhand ihres erzielten Mehrwerts, der zur Implementierung notwendigen Auflösung und des Winkelbereichs, in welchem sie aktiv sind. Die Tabelle 3-2 stellt einen Ausschnitt der von Lee erstellten Zuordnung dar.

Anhand der Winkelbereiche der verschiedenen Lichtfunktionen und der für die jeweilige Funktion erforderlichen Auflösung segmentiert Lee die Gesamtausleuchtung in Felder mit unterschiedlichen Auflösungsanforderungen. Das Bild 3-18 ist an diese Strukturierung angelehnt.

Aus seiner Anforderungsanalyse folgert Lee, dass ein Mikro-LED-System insbesondere im Zentralbereich erhebliche Vorteile gegenüber den bestehenden Matrix-Systemen aufweist.

*Tabelle 3-2*: *Kategorisierung verschiedener Lichtfunktionen nach erzieltem Mehrwert und benötigter Auflösung (Ausschnitt aus [Lee19]).*

| | Nutzen | | | | Auflösung | | |
|---|---|---|---|---|---|---|---|
| | Sicherheit | Komfort | Kommun. | Unterhaltung | > 2° | 0.5° | < 0.1° |
| GFHB | x | x | | | x | x | x |
| Schildent-blendung | x | x | | | | x | x |
| DBL | x | x | | | | x | x |
| Links-/Rechts-verkehr | x | x | | | | x | x |
| Navigations-symbole | | x | x | | | | x |
| Animationen | | | | x | | x | x |

Durch die dort erforderlichen Mindestauflösungen, die er im Bereich von 0.5° bis 0.7° beziffert, genügen die Auflösungen von Matrixsystemen nicht.

Die bereits erwähnten Beiträge zeigen, dass durch HD-Systeme Potentiale für neuartige Lichtfunktionen erschlossen werden können. Die Freiheiten in der Gestaltung sind dabei so weitreichend wie die Unsicherheiten in der Akzeptanz dieser neuen Funktionen. Krieft et al. untersuchten, welche Lichtfunktionen von Probanden als sinnvoll eingeschätzt werden und welche Kriterien erfüllt sein müssen, damit eine neu eingeführte Lichtfunktion auf Akzeptanz stößt [KTW+19]. Im Rahmen des Beitrags wurde ein breites Spektrum von Lichtfunktionen in verschiedenen Probandenstudien untersucht.

In der ersten vorgestellten Studie wurde ein optischer Spurhalte- und Geschwindigkeitsassistent durch zwölf Probanden auf insgesamt ca. 750 gefahrenen Kilometern evaluiert. Beide Systeme wurden akzeptiert (4-5 von 6 Punkten), wobei der Spurassistent besser abgeschnitten hat. Während sich die erste Studie mit der Fahrerassistenz beschäftigte, wurden in der zweiten Studie Lichtfunktionen für das autonome Fahren untersucht. Diese Funktionen haben die Aufgabe, den anderen Verkehrsteilnehmern (Radfahrer, Fußgänger, ...) Hinweise zu geben, weshalb diese anstelle des Fahrers die Bewertung vornehmen. Es zeigt sich, dass die zur Kommunikation verwendeten Lichtzeichen in vielen Fällen unverständlich sind und fehlinterpretiert werden.

In einer dritten Studie wurden Informationsprojektionen zur Kommunikation mit dem Fahrer analysiert. Beispiele hierfür sind Projektionen von Navigationspfeilen, Geschwindigkeitsbegrenzungen oder Willkommens- und Abschiedsanimationen. Die Akzeptanz dieser Projektionen durch die 21 Probanden ist weit gestreut. Sie variiert von starker Akzeptanz für Wildwechsel- und Navigationshinweise bis hin zu fast einstimmiger Ablehnung für eine Animation beim Einschalten des Fernlichts.

Als Fazit folgern Krieft et al., dass Lichtprojektionen durchaus nutzenbringend sein können. Voraussetzung hierfür sind jedoch die Verständlichkeit des Symbols, dessen Positionierung und Form sowie das richtige Timing.

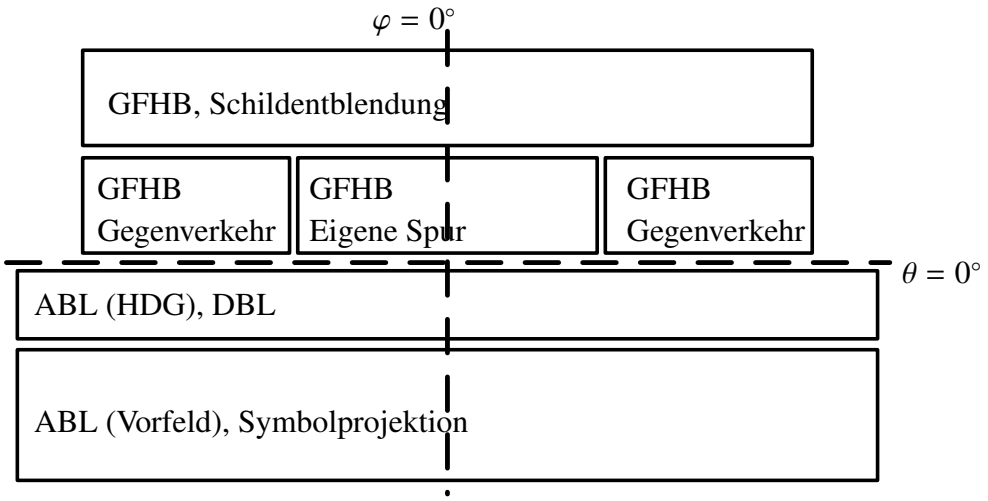


*Bild 3-18*: *Segmentierung des Gesamtausleuchtungsbereichs in die Wirkbereiche verschiedener Lichtfunktionen (orientiert an [Lee19]).*

Ist die Interpretierbarkeit von Symbolprojektionen sichergestellt, gilt es nachfolgend zu prüfen, ob durch die optische Unterstützung des Fahrers Verbesserungen erzielt werden können. Diese Fragestellung haben Budanow und Neumann untersucht [BN19]. Grundlage der Untersuchung waren ein Spurwechsel- und ein Bremsmanöver aufgrund eines Hindernisses auf der Fahrspur (siehe Bild 3-19). Dabei wurde verglichen, wie die Reaktionsfähigkeit des Fahrers ohne und mit der Unterstützung von Lichtfunktionen variiert. In beiden Manövern konnten Verbesserungen von etwa 30% bezüglich der Entfernung zwischen dem Hindernis und dem Beginn der Reaktion verzeichnet werden. Auch der subjektive Eindruck der 82 Testpersonen war mit großer Mehrheit positiv. Sie stellen außerdem heraus, dass die verwendeten Projektionslinien besser interpretierbar sind, als die in anderen Studien (z.B. [RTH+19]) eingesetzten Dreiecke.

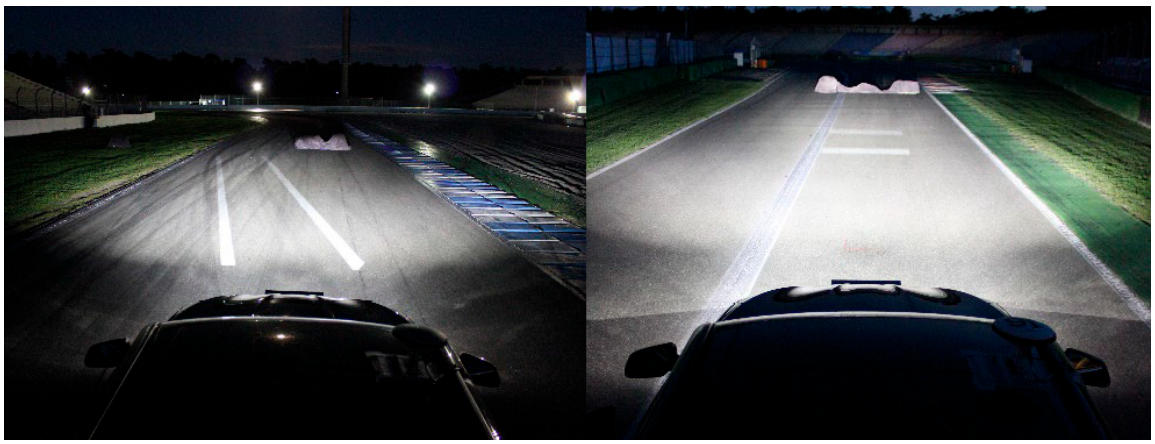

*Bild 3-19*: *Untersuchung des Einflusses optischer Signale auf die Reaktion des Fahrers anhand verschiedener Fahrmanöver aus [BN19] (links: Spurwechsel, rechts: Bremsung).*

Das Baustellenlicht (Construction Zone Light, CZL) wird verstärkt als HD-Lichtfunktion mit herausragendem Mehrwert diskutiert. Hamm hat dazu in 2019 eine umfangreiche Studie vorgestellt, die neben der Bewertung der Vorteile des CZL für den Fahrer auch zulassungsrelevante Aspekte, wie die Auswirkungen auf die Blendung anderer Verkehrsteilnehmer, beleuchtet [Ham19]. Dazu haben 21 Testpersonen ein Überholmanöver innerhalb einer Autobahnbaustelle in verschiedenen Szenarien durchfahren. Verglichen wurden die Lenkeingaben und Gaspedalstellungen mit und ohne CZL. Hamm zeigt, dass die Anzahl der Lenkkorrekturen bei aktivem CZL deutlich reduziert werden können. Außerdem wurde die Gaspedalstellung während des Überholmanövers beobachtet. Es zeigt sich, dass Fahrer ohne CZL häufiger die Gaspedalstellung und damit ihre Geschwindigkeit während des Überholvorgangs ändern. Insgesamt schlussfolgert Hamm, dass CZL zu einem ruhigeren und kontrollierteren Verhalten des Fahrers führt und somit zu einem Sicherheitsgewinn.

Im zweiten Teil der Studie untersucht Hamm mögliche blendungskritische Auswirkungen von CZL. Hierbei zieht er sowohl objektive als auch subjektive Bewertungen heran. Eine Vermessung der Beleuchtungsstärke im Bereich der links vom Egofahrzeug befindlichen Spur für verschiedene Distanzen zeigt, dass ein aktives CZL von einem einfachen Abblendlicht messtechnisch kaum unterscheidbar ist. Diese Beobachtungen werden für trockene und nasse Fahrbahnen bestätigt. Die subjektiven Beurteilungen erfolgten durch 44 Testpersonen, die in einem entgegenkommenden Fahrzeug positioniert wurden. Sie spiegeln die messtechnischen Resultate wider.

*Bild 3-20*: *Construction Zone Light (CZL) zur Unterstützung des Fahrers in engen Fahrspuren von Autobahnbaustellen aus [RL19].*

Rosenhahn und Link bestätigen in einer ähnlichen Untersuchung die positiven Auswirkungen des CZL auf das Fahrverhalten [RL19]. Sie zeigen, dass Fahrer, welche sich ohne optische Unterstützung in 20%-40% der Zeit außerhalb des Sicherheitsbereichs (+/- 25 cm der Idealspur) aufhalten, in ihrer Spurführung stark verbessert werden können. Durch CZL kann der zeitliche Anteil außerhalb des Sicherheitsbereichs auf 10% reduziert werden.

Mit dem Schlechtwetterlicht zeigen Thoma und Vollrath eine weitere HD-Lichtfunktion, die in Teilen bereits durch Matrix-Systeme implementiert werden kann [TV19]. Sie

vergleichen drei verschiedene Ansätze zur Anpassung der Lichtverteilung bei Nebel von Rosenhahn und Schmidt [Ros00], [Sch17]. Dabei stellen die Autoren insbesondere heraus, wie aufwendig und teilweise unmöglich verlässliche Tests durch Realfahrten bei der Erprobung von Lichtfunktionen sein können. Sie greifen deshalb schließlich auf eine künstliche Szenerie in einer Halle mit Nebelmaschinen zur Durchführung der Probandenstudie zurück. Das Bild 3-21 zeigt das Testsetup.

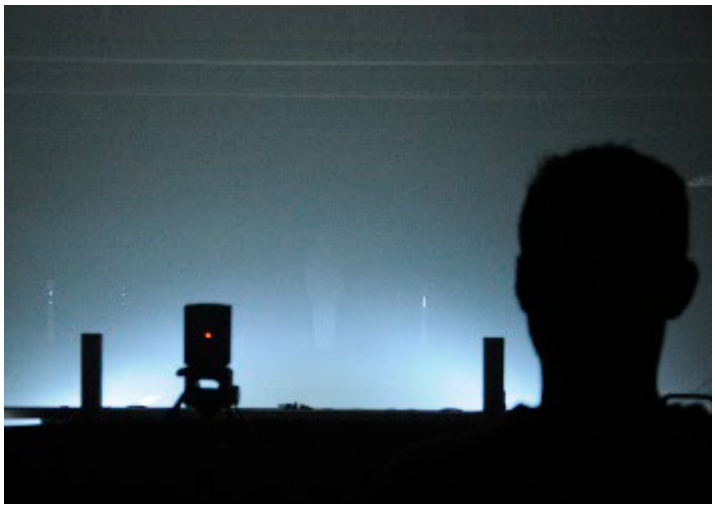

*Bild 3-21*: *Probandenstudie zur Bewertung verschiedener Schlechtwetterlichtfunktionen mit künstlichem Nebel in einer statischen Szenerie aus [TV19].*

Die 34 Teilnehmer bewerten die verschiedenen Lichtfunktionen hinsichtlich der Sichtbarkeit von Objekten, die sich vor dem Fahrzeug befinden, und der Eigenblendung, wie sie durch die Reflexion des Scheinwerferlichts an Nebelpartikeln hervorgerufen wird. Außerdem wird durch die Sichtbarkeitsreichweite ein objektives Kriterium herangezogen. Es zeigt sich, dass eine niedrigere HDG verwendet werden sollte, da eine Fernlichtverteilung eine zu starke Eigenblendung hervorruft. Außerdem zeichnet sich ab, dass eine Dimmung der Vorfeldausleuchtung zu einer besseren Erkennbarkeit von Objekten führt. Da in der Studie das methodische Vorgehen bei der Bewertung von Schlechtwetterlicht im Fokus steht, müssen nachfolgende Arbeiten die konkreten Eigenschaften einer geeigneten Schlechtwetterlichtverteilung präzisieren.

Zum Ende dieses Abschnitts sollen einige HD-Lichtfunktionen vorgestellt werden, die im Kontext des autonomen Fahrens an Bedeutung gewinnen werden. Auch wenn der Scheinwerfer bei autonomen Fahrzeugen des SAE-Levels 5 nicht mehr zur direkten Unterstützung des menschlichen Sehens notwendig ist, kommen ihm in diesem Kontext neuartige funktionale Aufgaben zu.

In autonomen Fahrzeugen stellt der Kamerasensor eine Schlüsselkomponente zur Wahrnehmung der Umgebungssituation dar. Tesla wagt sogar die These, dass ein Sensorsetup, welches ausschließlich aus Kameras besteht, ausreicht, um autonome Fahrfunktionen umzusetzen [Tem19]. Ein Vorteil gegenüber LiDAR- und RADAR-Sensoren ist die höhere örtliche Auflösung [Fec18] und die farbbasierte Objekterkennung. Im Gegensatz zu anderen Sensoren im Kontext des autonomen Fahrens erfordert die Kamera allerdings eine hinreichende Ausleuchtung der zu beobachtenden Umgebung. Bei Nacht muss diese

Ausleuchtung durch das Scheinwerfersystem sichergestellt werden, womit der Fortbestand von Scheinwerfern in autonomen Fahrzeugen gesichert ist.

Gut et al. schlussfolgern vier Funktionen von Scheinwerfern in autonomen Fahrzeugen [GXB19]. Diese sind

- Signal an die Umgebung: Sicherung der Erkennbarkeit durch andere Verkehrsteilnehmer bei Nacht,
- Kommunikation mit der Umgebung: Symbolprojektionen von HD-Scheinwerfern stellen einen Kommunikationskanal zu anderen Verkehrsteilnehmern dar,
- Trajektorienbeleuchtung: Die Fahrzeuginsassen können die zukünftige Bewegung des autonomen Fahrzeugs überwachen und gewinnen ein Sicherheitsgefühl
- Sensorunterstützung: Ausleuchtung der Umgebung zur Kameraerkennung von Objekten, Farben und Wetterverhältnissen bei Nacht.

Die Autoren fokussieren in ihrem Beitrag den letzten Stichpunkt. Sie verdeutlichen, dass die Unterstützung der Kamerasensorik durch spezielle Lichtfunktionen und das damit einhergehende SAE-Level 5 zu einem signifikanten Rückgang der Verkehrstoten führen könnte. Gründe hierfür sind das Entfallen von müdigkeitsbedingten Fehlern des menschlichen Fahrers und die gezielte Beleuchtung von Stellen, die durch LiDAR- und RADAR-Sensoren nicht interpretiert werden können. Als weiteren Punkt nennen die Autoren die Erkennung von Wetterverhältnissen, welche tagsüber bereits durch die Kamera geleistet werden kann. Bei geeigneter Ausleuchtung ist somit zu erwarten, dass diese Erkennung auch auf die Nachtfahrt überführt werden kann. Die vorliegenden Wetterverhältnisse lassen wiederum Rückschlüsse auf die Verlässlichkeit verschiedener Sensoren und die geeignete Reisegeschwindigkeit zu. Zur Implementierung geeigneter Lichtfunktionen leiten Gut et al. die in Tabelle 3-3 aufgeführten Anforderungen an das Scheinwerfersystem ab.

*Tabelle 3-3*: *Anfordernungen an ein Scheinwerfersystem in autonomen Fahrzeugen nach [GXB19].*

| Typ | Anwendung | Anforderungen |
|---|---|---|
| Text | Erkennung von Verkehrsschildern und Straßenmarkierungen | Präzise Auflösung und anpassbare Lichtstärke |
| Farbe | Erkennung von Farben von Verkehrsschildern und von Straßenmarkierungen | Hoher Farbwiedergabeindex (CRI) (möglichst kontinuierliches Spektrum der Lichtquelle) |
| Kontrast | Klassifizierung nicht beleuchteter Objekte und Straßenränder | Hohe Lichtstärke im Bereich der Fahrzeugtrajektorie, hohe Adaptivität der Lichtverteilung |

Auch Böhm stellt eine ähnliche These zur Rolle von Licht in zukünftigen autonomen Fahrzeugen auf [Böh19]. Er spricht vom ersten „Closed-Loop“ Ansatz von Aktoren und Sensoren, der über die klassische Sensorfusion hinaus geht. Einerseits verbessert das Licht die Wahrnehmung der Umgebung durch die Kamera und andererseits kann durch die gewonnenen Umfeldinformationen eine bessere Gestaltung der Lichtverteilung

vorgenommen werden. Während das Scheinwerferlicht heutzutage den Fahrer unterstützt, wird es zukünftig den Sensoren das nächtliche Sehen ermöglichen.

Die getätigten Aussagen unterstützt Böhm durch verschiedene Studien. In einem Versuch vergleicht er die Erkennung eines Fußgängers an einer Landstraße ohne zusätzliche Lichtquellen bei verschiedenen Geschwindigkeiten. In den insgesamt 16 Tests wird der Fußgänger bei Abblendlicht nur zweimal durch den Algorithmus erkannt. Bei Verwendung des Fernlichts konnte der Fußgänger 14 mal detektiert werden. Diese Studie zeigt, dass die Informationsqualität des Kamerasensors maßgeblich durch die Eignung der Lichtverteilungen beider Scheinwerfer bestimmt wird. In einer weiteren Studie zeigt Böhm, dass die Konfidenzintervalle der Objekterkennung bei Nacht von 50% auf über 90% angehoben werden können, wenn anstelle des Abblendlichts eine gezielte Beleuchtung relevanter Bereiche erfolgt. Damit bewegen sich die Konfidenzintervalle im Wertebereich einer Tagfahrt.

Mit der wachsenden Zahl von Umfeldsensoren, wie sie mit der Einführung autonomer Fahrzeuge zu erwarten ist, wird neben der logischen auch die physikalische Integration von Sensoren und Scheinwerfern Gegenstand der Forschung. Mathes und Reiss zeigen auf, dass die Integration der Sensoren in das Scheinwerfergehäuse geometrische, elektronische und physikalische Vorteile liefert [MR18]. Argumente hierfür sind vorhandene Reinigungsanlagen der Scheinwerfersysteme, die günstige Anbaulage an der Fahrzeugfront, welche beispielsweise den „Blick“ um eine Gebäudeecke früh erlaubt, existierende Kühlsysteme der Scheinwerfer oder die Verwendung eines zentralen Steuergeräts für Licht, RADAR und LiDAR. Darüber hinaus beschreiben die Autoren ein Konzept, in welchem dieselben Optiken gemeinsam durch die HD-Lichtquelle des Scheinwerfers und die Kamera genutzt werden.

## 3.3 Simulation in der Scheinwerfertechnik

Die Vielfältigkeit der technischen Lösungen und ihre jeweiligen Vor- und Nachteile erfordern schon in frühen Phasen der Entwicklung eines Scheinwerfersystems die Berücksichtigung verschiedenster Aspekte im Hinblick auf die zu realisierenden Lichtfunktionen. Darüber hinaus zeigt der vorhergehende Abschnitt, dass die Funktionen und Dynamiken des Scheinwerferlichts durch die HD-Technik stark zunehmen. In dem Zuge gewinnt auch der Entwicklungsprozess der Lichtfunktionen an Komplexität. Außerdem ist der Testaufwand im Scheinwerferbereich aufgrund der Anforderungen an Tageszeit und Witterung enorm zeitintensiv und kostspielig. Aufgrund dieser Gegebenheiten und zusätzlichen Sicherheitsaspekten ist die simulationsbasierte Entwicklung moderner Scheinwerfersysteme eine Methode, welche die Realisierung von HD-Lichtfunktionen nicht nur erleichtert, sondern überhaupt erst ermöglicht. Dieser Abschnitt thematisiert, welche simulationsbasierten Methoden in der Entwicklung von HD-Scheinwerfern Anwendung finden und fokussiert dabei virtuelle Nachtfahrtsimulationen, welche das Kernthema dieser Arbeit darstellen.

Bei der Recherche zu virtuellen Nachtfahrtsimulationen stechen verschiedene kommerzielle Lösungen heraus. Diese Produkte werden nachfolgend diskutiert und hinsichtlich ihrer Funktionalität im Kontext von HD-Scheinwerfern verglichen.

Synopsys bietet die Toollandschaft „LucidShape“ an [Syn19]. Hierbei handelt es sich um eine Produktfamilie von miteinander kompatiblen Tools, welche die Lichtentwicklung im automobilen Umfeld in vielen Bereichen unterstüzten. Neben Tools zur Simulation der

Lichtausbreitung im Scheinwerfer, welche zur Auslegung des Scheinwefers und zur simulativen Erzeugung von Lichtstärkeverteilungen dienen, können mit dem Tool „LucidDrive“ virtuelle Nachtfahrten durchgeführt werden [Syn20].

*Bild 3-22*: *Screenshot aus der Nachtfahrtsimulation „LucidDrive“ der Firma Synopsys [Syn20].*

LucidDrive wird fortlaufend weiterentwickelt und verfügt inzwischen über weitreichende Funktionen. Es können monochrome und spektrale Lichtverteilungen dargestellt werden. Letztere haben gegenüber Lichtstärkeverteilungen den Vorteil, dass beispielsweise Farbsäume im Randbereich optischer Linsen sichtbar werden. Diese Phänomene kommen durch die wellenlängenabhängige Brechung des Lichts zustande. Der zugrunde liegende Effekt wird als Dispersion bezeichnet [PPBS05].

Seit September 2019 unterstützt LucidDrive die Simulation hochauflösender Scheinwerfer mit dem „AFS Masking PixelLight“-Feature in idealisierter Form [Syn19]. Das Bild 3-22 zeigt eine Momentaufnahme während der Verwendung dieses Features. Dazu wird initial die Lichtverteilung der HD-Lichtquelle unter maximaler Bestromung aller Pixellichter generiert. Während der Simulation können dann definierbare Winkelbereiche aus dieser Lichtverteilung vollständig oder teilweise abgedunkelt werden. Hierbei handelt es sich um eine stark idealisierte Nachbildung der HD-Lichtquelle, da die physikalische Realisierbarkeit der Maskierung nicht überprüft wird. Aus diesem Grund können Streulichtanteile und Überlappungen der einzelnen Pixellichtquellen mit dem AFSMaskingLight-Feature nicht korrekt wiedergegeben werden. Vorteilhaft bei diesem Ansatz sind die geringeren Anforderungen an die Rechenperformance und die einfache Manipulation der Lichtverteilung.

Seit 2020 unterstützt LucidDrive auch eine physikalisch korrekte Simulation von HD-Scheinwerfern, welche für jedes Pixel der Lichtquelle eine Lichtverteilung vorhält. Hier gibt der Hersteller eine Anzahl von mindestens 12.800 Pixeln an, die in Echtzeit simuliert werden können. Damit beinhaltet LucidDrive die mit Abstand performanteste Pixellichtsimulation am Markt. Jedoch weist die Implementierung auch einen erheblichen Nachteil auf:

spektrale Lichtverteilungen werden im Kontext von HD-Scheinwerfern nicht unterstützt. Somit relativiert sich der sehr hohe Maximalwert gleichzeitig simulierbarer Pixel.

Im Kontext von Fahrsimulatoren kommen häufig mehr Anzeigegeräte zum Einsatz, als ein einziger Rechner versorgen kann. Der Renderaufwand skaliert linear mit der Anzahl von Ausgabegeräten, weshalb zumindest die grafisch relevanten Komponenten auf mehrere Rechner, die in einem gemeinsamen Netzwerk kommunizieren, verteilt werden müssen. LucidDrive unterstützt vornehmlich Büroarbeitsplätze und kleine Simulatoren mit bis zu drei Ausgabegeräten. Eine Unterstützung von Großsimulatoren steht aktuell nicht im Fokus.

Das Fahrzeugmodell ist sehr einfach gehalten und bildet die Dynamik realer Fahrzeuge nicht ab. Witterungseinflüsse, wie Regen, Nebel oder Schnee, können in LucidDrive nicht nachgebildet werden. Zur Analyse von Lichtfunktionen kann die momentane Lichtverteilung durch Falschfarben- und Isolinien-Darstellungen visualisiert werden. Eine aktive Unterstützung bei der Gestaltung von Lichtfunktionen existiert nicht.

Bekannt für umfangreiche Kompetenzen bei der Echtzeitsimulation und insbesondere HiL-Simulation im automobilen Umfeld ist die Firma dSPACE. Der Hersteller bietet sowohl Hardware- als auch Software-Lösungen in diesem Anwendungsfeld an. Mit der Software „MotionDesk“ bietet dSPACE eine Visualisierung der in Echtzeit berechneten Fahrsimulation und unterstüzt seit 2017 und ab Version 4.1 virtuelle Nachtfahrten [dSP17]. Bild 3-23 zeigt beispielhaft einen Screenshot aus MotionDesk.

*Bild 3-23*: *Screenshot einer virtuellen Nachtfahrt im Tool „MotionDesk“ der Firma dSPACE [Quelle: dSPACE].*

Der Fokus von dSPACE liegt jedoch derzeit mehr im Bereich der Simulation von Umfeldsensorik, wie LiDAR-, RADAR- oder Kamera-Sensoren. Im Rahmen der Scheinwerfersimulation werden nur statische Lichtverteilungen unterstützt. Eine Simulation von hochauflösenden Scheinwerfern exisitert bisher nicht. Die Farbe des Scheinwerferlichts kann in MotionDesk variiert werden, wobei keine lokalen Farbunterschiede in der Lichtverteilung möglich sind. Spektrale Messdaten werden somit nicht unterstützt. Analyse- und Designtools zur Beurteilung der Lichtverteilung sind nicht implementiert. Obwohl dSPACE

eine Vielzahl von HiL-Simulationen verschiedenster Steuergeräte des Fahrzeugs unterstützt, können die Reaktionen des Scheinwerfersteuergeräts in MotionDesk mangels HD-Unterstützung nicht visualisiert werden. Die grundsätzliche Anbindung von Steuergeräten ist unter Verwendung der hauseigenen Echtzeithardware sehr flexibel umsetzbar.

Die Anforderungen an die Skalierbarkeit deckt MotionDesk weitestgehend ab. Die Simulation ist sowohl auf einem normalen Desktop-PC, als auch auf der Echtzeithardware ausführbar. Außerdem ist die Verteilung der Visualisierung auf mehrere Rechner möglich, sodass auch Großsimulatoren geeignet betrieben werden können. Als Schwachpunkt ist die Notwendigkeit einer Echtzeithardware zur HiL-Anbindung des Steuergeräts zu nennen. Alternativ bietet dSPACE die Plattform VEOS an, welche neben SiL- und MiL-Tests mit vollständiger CAN Simulation auch reale Steuergeräte mit einem CAN-USB Adapter ankoppeln kann.

MotionDesk unterstützt die Visualisierung von Regen, Schnee und Nebel. Dabei muss jedoch angemerkt werden, dass keine Wechselwirkung des Scheinwerferlichts mit Partikeln in der Luft simuliert wird. Daran wird jedoch aktuell gearbeitet. Ebensowenig ändert sich das Reflektionsverhalten der Umgebungsobjekte bei entsprechender Witterung. Insofern kann nur von einer rudimentären Witterungssimulation gesprochen werden.

Eine besondere Stärke von dSPACE ist das komplexe Fahrzeugmodell der „Automotive Simulation Models“ (ASM) Tool Suite [dSP20]. Hierbei handelt es sich um ein Fahrzeugmodell mit insgesamt 26 Freiheitsgraden, die sich auf den Fahrzeugaufbau, den Antriebsstrang, das Lenksystem und die Räder verteilen. Das Fahrzeugmodell kann über das Tool „ModelDesk“ komfortabel parametriert werden [dSP20]. Ein Ausschnitt aus ModelDesk wird in Bild 3-24 dargestellt.

Als weiterer Vorteil ist die weitgehende Quelloffenheit des Fahrzeugmodells zu nennen. Die Implementierung wird von dSPACE in MATLAB/Simulink vorgenommen und als einsehbares Modell zur Verfügung gestellt. Durch die quelloffene Verfügbarkeit kann der Anwender selbstständig tiefergehende Änderungen an der Modellierung vornehmen, zusätzliche Komponenten ergänzen oder Schnittstellen zu anderen Simulationskomponenten erzeugen. Das Fahrzeugmodell kann für die dSPACE Echtzeithardware kompiliert und mit einer Schrittweite von einer Millisekunde simuliert werden.

Der Scheinwerferhersteller HELLA pflegt eigene Softwaretools zur Unterstützung der Entwicklungsarbeit. Die Toolkette wird unter dem Namen „Helios“ geführt und bietet Berechnungen und Simulationen von lichttechnischen Geräten. Obwohl Automobilität im Fokus steht, werden darüber hinaus auch andere Sektoren, wie z.B. der Flugzeugbau, bedient. Innerhalb der Helios Toolsuite stellt der LightDriver Professional, im Folgenden als LightDriver bezeichnet, die virtuelle Nachtfahrtsimulation dar [Pla12], [WPKB02], [WP01].

Seit 2019 untersützt der LightDriver die physikalische Simulation von Pixel-Scheinwerfern mit etwa 100 Lichtquellen. Die Einzellichtverteilungen können hierbei auch spektral vorliegen. Um höher aufgelöste Systeme zu unterstützen, wird derzeit an der HD-Simulation durch Maskierung gearbeitet, welche zukünftig in den LightDriver integriert werden soll.

Das Fahrzeugmodell des LightDriver ist einfach gehalten und bildet die Fahrdynamik realer Fahrzeuge nicht ab. Eine Witterungssimulation ist ebenfalls nur in stark vereinfachter Form möglich. Eine physikalische Nachbildung von Witterungsphänomenen wird im LightDriver nicht unterstützt.

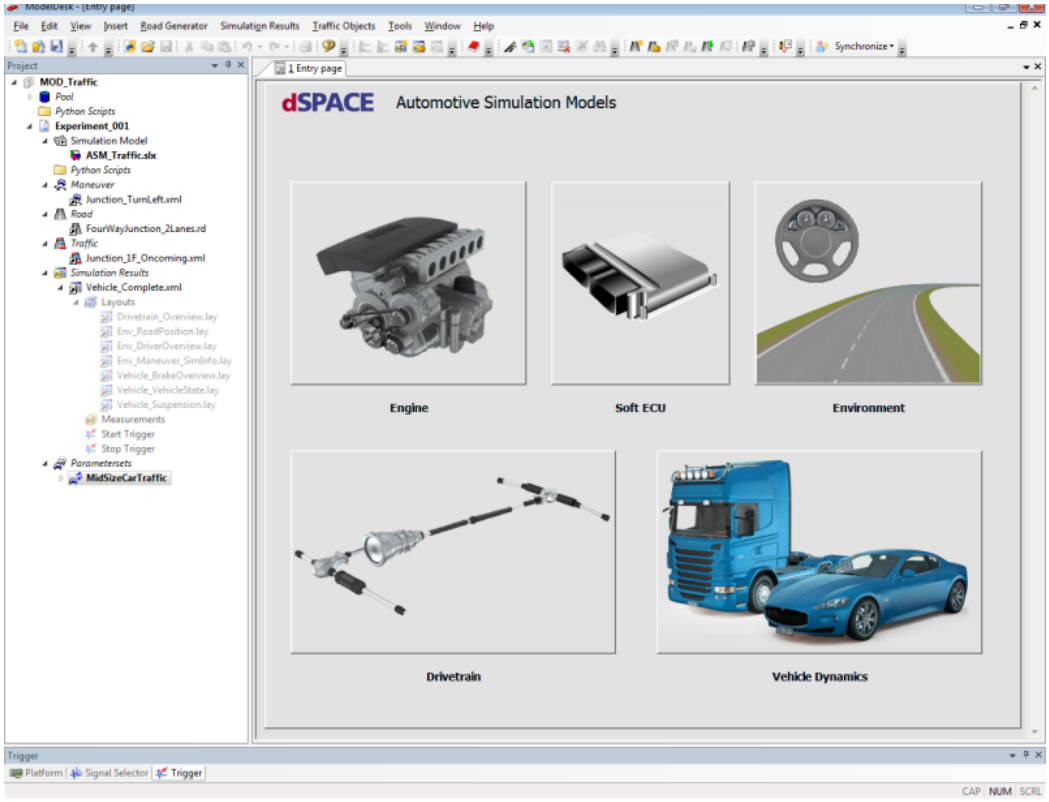


*Bild 3-24*: *Screenshot des Tools „ModelDesk“ zur Parametrierung des Fahrzeugmodells der ASM Toolsuite [Quelle: dSPACE].*

Hinsichtlich der Analysetools verfügt der LightDriver neben dem üblichen Portfolio, wie Isolinien- und Falschfarbendarstellungen von photometrischen Größen, über eine Auswertung der Lichtverteilungen bezüglich gesetzlicher und kundenspezifischer Vorgaben. Eine aktive Unterstützung bei der Lichtfunktionsgestaltung sieht aber auch der LightDriver nicht vor.

Die HiL-Simulation des Scheinwerfersteuergeräts wurde in der Vergangenheit für herkömmliche Scheinwerfersysteme angewendet. Eine Unterstützung von HD-Steuergeräten befindet sich derzeit in Entwicklung.

Der LightDriver kann zum Betrieb von Großsimulatoren eingesetzt werden. Dazu kann er in einer verteilten Client-Server-Architektur betrieben werden. Durch die variable Anzahl an Clients können hinreichend viele Ausgabegeräte versorgt werden. Die Steuerung erfolgt in diesem Betriebsmodus zentral am Server, sodass dieser neben der Koordination des Netzwerkbetriebs die Rolle einer Remote-Applikation übernimmt.

Die Vires Simulationstechnologie GmbH als Teil der Hexagon-Gruppe bietet die Software „Virtual Test Drive“ an. Hierbei handelt es sich um eine Software zur Generierung und Animation von virtuellen Testumgebungen im Bereich der Fahrsimulation. Dabei werden auch Umfeldsensorsimulationen unterstützt. Im Bild 3-26 ist eine Momentaufnahme der Simulation zu sehen. Die vielfältigen Möglichkeiten der Verkehrs- und Umfeldsimulationen werden besonders deutlich.

Im Kontext der physikalisch basierten Umfeldsimulation stellt die korrekte Wiedergabe des Scheinwerferlichts einen zentralen Bestandteil des Portfolios dar. Deshalb arbeitet


*Bild 3-25*: *Screenshot der Nachtfahrtsimulation „LightDriver“ aus der Toolsuite „Helios“ der HELLA [Quelle: HELLA].*

Vires auch an der Simulation von Pixellichtsystemen. Die Einbindung phyiskalischer Lichtquellen im Sinne einzelner Lichtverteilungen ist jedoch auf 100 Elemente beschränkt, wobei auf Rückfrage bei Vires eine Anzahl von 5 bis 20 einzelnen Lichtquellen aufgrund des hohen Rechenaufwands empfohlen wird. Demzufolge ist diese Variante zwar für die Simulation konventioneller Scheinwerfer mit einigen Zusatzlichtquellen geeignet. Sie kann jedoch nicht für die Simulation selbst niedrig aufgelöster Pixelsysteme eingesetzt werden. Um dennoch moderne Systeme mit hoher Auflösung simulieren zu können, bietet Vires das sogenannte Beamerelement an, welches einmalig pro Scheinwerfer eingebunden werden kann. Dieses verfügt über eine Auflösung von maximal $1.000 \times 1.000$ Pixel. Das Beamerelement ist jedoch nicht als eine physikalisch motivierte HD-Simulation, sondern als Maskierung einzuordnen. Eine HiL-Einbindung des Scheinwerfersteuergeräts ist ebenfalls möglich. VTD implementiert die üblichen Analysetools zur Beurteilung von Lichtverteilungen. Designtools für Lichtfunktionen werden nicht bereitgestellt.

Eine Besonderheit von VTD ist die Voraussetzung an das Betriebssystem. Die Software läuft ausschließlich auf Linux-Systemen. Durch die zugrunde liegende Server-Client-Architektur ist eine hohe Skalierbarkeit der Simulation gegeben. Sie kann auf nur einem oder mehreren Rechnern betrieben werden. Entsprechend ist die Anzahl der Ausgabemedien variabel, wodurch die Anwendbarkeit vom Desktop-PC bis hin zum Großsimulator gegeben ist.

Hinsichtlich der Fahrzeugmodellierung unterstützt VTD ein breites Spektrum. Vom Einspurmodell bis hin zum ADAMS-Fahrzeugmodell kann der Anwender entsprechend seiner Anforderungen wählen. Das ADAMS-Fahrzeugmodell wird von dem Unternehmen MSC Software, welches ebenfalls Teil der Hexagon-Gruppe ist, entwickelt und weist eine hohe Komplexität auf. Weitere Informationen zu diesem Modell finden sich in [MSC20].

*Bild 3-26*: *Screenshot des Tools „Virtual Test Drive“ der Vires Simulationstechnologie GmbH [Quelle: MSC Software].*

Eine Witterungssimulation wird in VTD unterstützt. Diese bewegt sich im Kontext der Lichtsimulation allerdings auf einer eher oberflächlichen Ebene und lässt keine Interaktionen zwischen dem Scheinwerferlicht und den Partikeln in der Luft zu.

Umfassende Simulationsanwendungen bietet neben Hexagon auch die in den USA ansässige Firma Ansys an. Die von Ansys geführte Produktfamilie Optis umfasst dabei Simulationstools im Bereich des automotiven Lichts. Innerhalb von Optis bildet die Software VRXPERIENCE (VRX) virtuelle Nachtfahrtsimulationen ab [Ans20]. Die Szenengenerierung wird hierbei durch die Simulation SCANeR des Anbieters AVSimulation realisiert [AVS20]. Eine Momentaufnahme mit aktiver Falschfarbenvisualisierung photometrischer Größen zeigt Bild 3-27.

Ansys unterstützt die Simulation von HD-Scheinwerfern. Die Lichtverteilungen der einzelnen Pixel können photometrisch oder spektral geladen werden. Die maximale Anzahl der Pixel pro Scheinwerfer werden auf 500 begrenzt. Auf Nachfrage teilte das Unternehmen jedoch mit, dass sich die Simulation unter der Annahme einer Gaming-Grafikkarte, wie der GeForce GTX 1080 Ti, eher im Bereich von maximal 120 Pixeln sinnvoll betreiben lässt.

Die HiL-Einbindung des Scheinwerfersteuergeräts ist in VRX möglich. Darüber hinaus kann auch das Steuergerät der Umfeldkamera im In-the-Loop-Betrieb eingebunden werden. Die Skalierbarkeit des Systems ist ebenfalls gegeben. So lassen sich abhängig von der Lizenz bis zu 16 Ausgabegeräte an die Simulation koppeln, womit auch Großsimulatoren hinreichend bedient werden können. Die Single Server Architektur erfordert allerdings die Kopplung aller Ausgabegeräte an einen Rechner, weshalb entsprechende Hardware vorzuhalten ist.

Wie in den meisten bereits diskutierten Anwendungen unterstützt VRX die gängigen Analysetools zur Beurteilung der Lichtverteilungen und -funktionen, bietet aber keine aktive


*Bild 3-27*: *Screenshot des Tools „VRXPERIENCE“ von Ansys [Quelle: Ansys].*

Unterstützung bei der Modifikation bestehender oder der Auslegung neuer Lichtfunktionen. Durch Cosimulation mit dem Tool VRX DS können visuelle Witterungsverhältnisse simuliert werden. Die physikalisch motivierte Simulation von Witterungsbedingungen steht in VRX bisher nicht zur Verfügung. Diese Funktionalität steht aber für 2021 auf der Agenda.

Hinsichtlich der verwendbaren Fahrzeugmodelle ist VRX sehr flexibel. Es erlaubt die Einbindung eigener Modell durch verschiedene Schnittstellen und bietet alternativ mit der hauseigenen Software Twin Builder die Möglichkeit, digitale Zwillinge realer Fahrzeuge zu generieren.

Zum Abschluss der Diskussion der am Markt verfügbaren Nachtfahrtsimulationen, werden sie in Tabelle 3-4 hinsichtlich der relevanten Eigenschaften in übersichtlicher Form verglichen. Die Tabelle wird im nachfolgenden Kapitel die Grundlage zur Ableitung des Handlungsbedarfs bilden.

*Tabelle 3-4*: *Vergleich der Funktionsumfänge verschiedener Nachtfahrtsimulationen.*

| Kriterium | MotionDesk | LightDriver | VTD | LucidDrive | VRX |
|---|---|---|---|---|---|
| Hersteller | dSPACE | HELLA | Vires | Synopsys | Ansys |
| Softwarepaket | ASM | Helios | - | LucidShape | Optis |
| Lichtverteilung | ja | ja | ja | ja | ja |
| Maskierung | nein | nein | ja | ja | ja |
| HD-Simulation | nein | ja | nein | ja | ja |
| HD-Sim. seit | - | 2019 | - | 2019 | 2017 |
| max. Pixelzahl | - | ≈100 | - | ≈12.800 | 120/500 |
| Spektrale HD-Sim. | nein | ja | - | nein | ja |
| Steuergeräte-HiL | (nein) | nein | ja | ja | ja |
| Skalierbarkeit | (ja) | ja | ja | (nein) | ja |
| Lichtfkt. Analyse | nein | ja | ja | ja | ja |
| Lichtfkt. Design | nein | nein | nein | nein | nein |
| Fahrzeugmodell | komplex | einfach | komplex | einfach | komplex |
| Witterung | einfach | einfach | einfach | nein | einfach |

# 4 Anforderungen und Handlungsbedarf

Das vorhergehende Kapitel zeigt, dass derzeit keine umfassende Lösung zur simulativen Erprobung hochauflösender Scheinwerfer und insbesondere zur Auslegung ihrer Lichtfunktionen verfügbar ist. Hieraus ergibt sich die Forschungsfrage: Welche Funktionen und Eigenschaften muss eine softwaretechnische Lösung zur bestmöglichen Unterstützung eines Ingenieurs bei der Auslegung von Lichtfunktionen hochauflösender Scheinwerfer aufweisen und wie können diese implementiert werden? Dieses Kapitel kann als Lastenheft der zu realisierenden Software verstanden werden.

## 4.1 Definition der Anforderungen

In den vorliegenden Unterabschnitten werden die wesentlichen Anforderungen an eine Nachtfahrtsimulation aufgelistet. Konkret unterteilen sich die Anforderungen an die softwaretechnische Lösung in die Kriterien

- visuelle Qualität,
- Technologie-Kompatibilität und Pixel-Skalierbarkeit,
- Echtzeitfähigkeit und X-in-the-Loop Testing,
- Latenz,
- Lichtanalyse und -entwurf,
- Witterung,
- Fahrdynamik,
- Konfigurierbarkeit,
- Parametrierbarkeit
- sowie Remote-Fähigkeit.

Für jedes dieser Kriterien werden nachfolgend im jeweiligen Unterabschnitt konkrete Leistungsmerkmale für eine Nachtfahrtsimulation, wie sie im Kontext von HD-Scheinwerfern zu erbringen sind, abgeleitet.

### 4.1.1 Visuelle Qualität

Die Bewertung von Scheinwerferlicht und Lichtfunktionen erfolgt neben den geltenden Gesetzen und Normen bis heute primär subjektiv. Der visuelle Eindruck des Lichtingenieurs ist ausschlaggebend im Entwicklungsprozess. Um Teile der Entwicklung von der realen in die virtuelle Umgebung verlagern zu können, ist eine realistische Nachbildung des Scheinwerferlichts in der virtuellen Szene grundlegend. Die hinreichende Übereinstimmung zwischen realer und virtueller Lichtverteilung sollte durch die virtuelle Nachbildung einer realen Szene erfolgen. Die gerenderte Szene muss im direkten Vergleich eine gute

Übereinstimmung mit der realen Umgebung aufweisen. Diese Übereinstimmung muss für verschieden geartete Lichtverteilungen nachgewiesen sein.

Der visuelle Gesamteindruck der Lichtverteilung ergibt sich aus verschiedenen Komponenten. Hierbei sind vor allem die geometrische Verteilung der Beleuchtungsstärke, die Wechselwirkungen mit verschiedenen Materialien und der Farbverlauf zu nennen. Zur Abbildung der Beleuchtungsstärke in einer virtuellen Szene sind die **korrekte Nachbildung der Lichtstärkeverteilungen**, die **distanzabhängige Abschwächung der Lichtintensität** und **vorliegende Lichteinfallswinkel** zu berücksichtigen. Um die Wechselwirkung zwischen Licht und verschiedenen Oberflächenmaterialien nachzubilden, müssen die **bidirektionalen Reflektanzverteilungsfunktionen der Materialien** in geeigneter Form approximiert werden. Eine korrekte Nachbildung des Farbverlaufs erfordert die **Unterstützung spektraler Lichtverteilungen**. Zudem muss die **Konvertierung des Quellfarbraums (meist CIE XYZ) in den Zielfarbraum (meist RGB)** bei geeigneter Kalibrierung erfolgen.

### 4.1.2 Technologie-Kompatibilität und Pixel-Skalierbarkeit

Wie Abschnitt 3.1.3 zeigt, existieren derzeit verschiedene technische Konzepte zur Umsetzung hochauflösender Scheinwerfer. Unter diesen Umständen ist eine möglichst große Kompatibilität wünschenswert. Zum einen bedeutet die Verwendung unterschiedlicher Nachtfahrtsimulationen für die verschiedenen technischen Realisierungen einen erheblichen Mehraufwand und zum anderen ist die Vergleichbarkeit derartiger Systeme über verschiedene Simulationen nur eingeschränkt möglich. Als weitere Anforderung lässt sich deshalb ableiten, dass die Implementierung der Lichtsimulation **unabhängig von der konkreten Scheinwerfertechnologie** vorgenommen werden soll.

Verwandt, aber nicht gleichbedeutend mit der Technologie-Kompatibilität ist die Skalierbarkeit bezüglich der Pixelzahl. Jede technische Realisierung von Pixellicht hat individuelle Stärken und Schwächen. Dabei unterscheiden sich verschiedene Technologien bezüglich der Anzahl ihrer Pixel über mehrere Größenordnungen. Bild 4-1 ordnet die Technologien entlang dieses Kriteriums. In den kommenden Jahren werden häufig Kombinationen der verschiedenen Realisierungsmöglichkeiten im Scheinwerfer anzutreffen sein. Gleichzeitig werden veraltete Systeme bei der Marktdurchdringung von den Sport-/Luxusklasse-Fahrzeugen (S/F Segment) bis hin zu Kleinstwagen (A Segment) über viele Jahre am Markt bestehen bleiben. Innerhalb dieses Zeitraums ist davon auszugehen, dass Hardware-Anpassungen und weitere Lichtfunktionen für die bestehenden Systeme zu entwickeln sind. Es leitet sich deshalb die **Skalierbarkeit der HD-Lichtsimulation von wenigen Matrix-Pixeln auf die hochaufgelösten Systeme** als weitere Anforderung ab. Dabei gilt es, auch bei einer großen Anzahl von Pixellichtquellen die Echtzeitfähigkeit zu erhalten.

### 4.1.3 Echtzeitfähigkeit und X-in-the-Loop Testing

Die Komplexität der Scheinwerfersteuergeräte hat sich durch die HD-Technologie wesentlich erhöht. Die Vielzahl der in Abschnitt 3.2 diskutierten Lichtfunktionen müssen auf

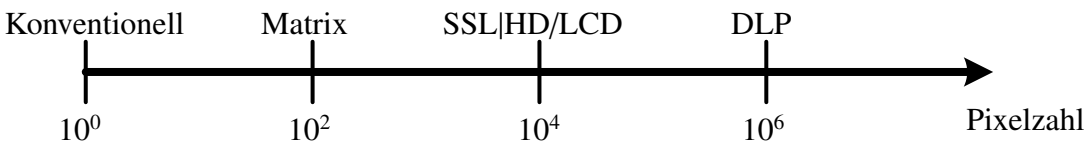


*Bild 4-1*: *Verschiedene HD-Technologien und die jeweiligen Anzahlen von Pixellichtquellen.*

ihnen logisch abgebildet werden. Das Steuergerät und die Scheinwerfer beanspruchen zur Kommunikation so hohe Datenraten, dass über Alternativen zum bisher im Automobil etablierten CAN Bus Protokoll nachgedacht wird [LKVZ11]. In der Konsequenz ist das Steuergerät und die auf ihm ausgeführte Software eine kritische Komponenten im Gesamtsystem und bedarf umfangreicher Erprobung. Diese Erprobung findet nach heutigem automobilen Standard durch In-the-Loop Tests in mehreren Stufen statt. Das Bild 4-2 ordnet die verschiedenen In-the-Loop Techniken in den Entwicklungsprozess ein.

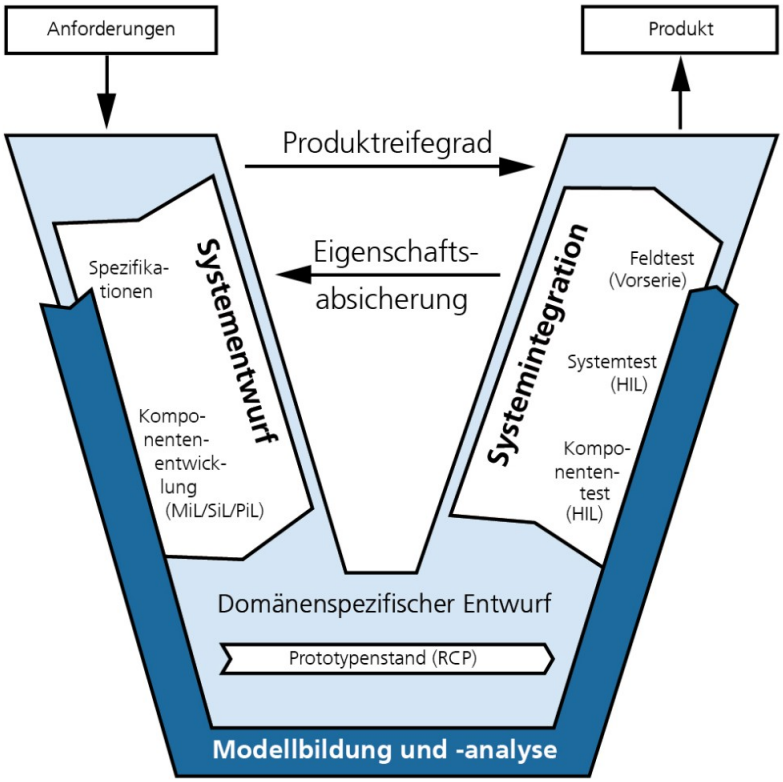


*Bild 4-2*: *Einsatz von In-the-Loop Techniken entlang des Entwicklungsprozesses [Quelle: Fraunhofer IEM].*

In der ersten Stufe werden Model-in-the-Loop (MiL) Tests durchgeführt. Dabei wird zunächst nur die logische Funktion des Steuergeräts in der Simulationssoftware abgebildet. Details der späteren Zielhardware und dessen Schnittstellen werden in diesem ersten Validierungsschritt ausgeblendet. Entspricht das abgebildete Modellverhalten den Erwartungen, so wird das Modell in Steuergerätcode überführt. Hierbei werden technische

Voraussetzungen der Zielhardware berücksichtigt. Der entstehende Code wäre bereits zur Implementierung auf dem Steuergerät verwendbar. Stattdessen wird die Steuergerät-hardware jedoch in der Simulationssoftware emuliert, sodass es sich weiterhin um ein vollständig virtuelles Testsystem handelt. Dieses Vorgehen wird als Software-in-the-Loop (SiL) Testing bezeichnet. Erst bei den Hardware-in-the-Loop (HiL) Tests wird das reale Scheinwerfersteuergerät getestet. Dazu wird die in den SiL-Tests als valide befundene Software auf das Steuergerät übertragen. Zur vollständigen Funktionsabsicherung muss das Steuergerät so umfassend an die Simulation gekoppelt sein, dass dessen wahrgenommene Umgebung der Einbausituation in einem realen Fahrzeug entspricht. Man spricht in diesem Zusammenhang auch von einer Restbussimulation.

Bedingt durch die bereits angesprochene Komplexität von Steuergeräten für HD-Scheinwerfersysteme stellt die **Unterstützung von MiL-, SiL- und HiL-Tests** eine weitere Anforderung an die Simulation dar. Hieraus leitet sich insbesondere ab, dass die Nachtfahrtsimulation eine durch den Steuergerättakt vorgegebene Echtzeitfähigkeit aufweisen muss. Am Beispiel des HD84-Scheinwerfersystems wird das Steuergerät mit einer Taktung von 50 Hz betrieben. Es verarbeitet somit in 20 ms neue Sensorwerte und überführt sie gemäß den Regeln der implementierten Lichtfunktionen in Dimmwerte für alle Pixellichtquellen des Scheinwerferpaars. Demzufolge muss die Simulation die Sensorwerte, welche in einer X-in-the-Loop Konfiguration durch virtuelle Sensoren erzeugt werden, in diesem Zeitrahmen bereitstellen und die vom Steuergerät empfangenen Dimmwerte in korrespondierende Lichtverteilungen überführen.

Neben der korrekten Interaktion mit dem Steuergerät stellt auch die menschliche Wahrnehmung eine Echtzeitanforderung dar. Digitale Ausgabegeräte können nur statische Bilder (Frames) anzeigen. Der Eindruck von Bewegung entsteht durch die hochfrequente Aktualisierung des dargestellten Frames. Ein flüssiger Eindruck entsteht ab etwa 30 Hz. Allerdings steigt die Immersion bei interaktiven Anwendungen mit noch höheren Bildwiederholfrequenzen. Die meisten Ausgabegeräte unterstützen eine Bildwiederholfrequenz von 60 Hz, welche im Idealfall durch die Nachtfahrtsimulation erreicht werden sollte.

Zusammengefasst definiert das **Maximum der idealen Bildwiederholfrequenz von 60 Hz und des Steuergerättakts die zu erfüllende Echtzeitschranke**. Es ist jedoch davon auszugehen, dass sich die Taktungen von Scheinwerfersteuergeräten auch zukünftig in diesem Frequenzbereich aufhalten werden. Höhere Taktfrequenzen würden zu visuell kaum wahrnehmbaren Verbesserungen führen.

### 4.1.4 Latenz

Bei einer HiL-Nachtfahrtsimulation interagieren eine Vielzahl technischer Geräte, wie ein oder mehrere Simulationsrechner, das Scheinwerfersteuergerät, Ein- und Ausgabegeräte sowie der menschliche Fahrer. Jede technische Einheit benötigt einen gewissen Zeitraum für die Datenverarbeitung sowie für das Senden und Empfangen der Daten. In einer interaktiven Fahrsimulation beginnt diese Übertragungskette mit den Eingaben des menschlichen Fahrers, die z.B. durch Lenkrad- und Pedalpositionen erfolgen, und endet mit der visuellen Anzeige auf einem oder mehreren Ausgabegeräten. Die verstrichene Zeitspanne, welche von den Eingaben bis zur Reaktion der Ausgabe verstreicht, wird als Latenz oder Totzeit bezeichnet. Eine zu große Latenz kann zu der sogenannten Simulatorkrankheit führen.

Ursächlich für das Unwohlsein des Fahrers sind die von seinen Erwartungen abweichenden Bewegungen und Bilder. Die Einhaltung einer **möglichst geringen Latenz** stellt deshalb eine weitere Anforderung an die Nachtfahrtsimulation dar.

### 4.1.5 Lichtanalyse und -entwurf

Die realitätsnahe Darstellung der Szene steht im Rahmen der Fahrsimulation im Vordergrund. Abseits von Probandenstudien kann es im Rahmen der Entwicklung sinnvoll sein, von dieser Darstellung abzuweichen und stattdessen künstliche Einblendungen vorzunehmen, die bei der Wahrnehmung sonst schwer erkennbarer Lichteigenschaften unterstützen. Hierzu haben sich verschiedene visuelle Analysewerkzeuge etabliert. Vorrangig ist hier die Hervorhebung verschiedener photometrischer Größen durch Isolinien- oder Falschfarbendarstellungen zu nennen. Konkret eignen sich primär die Lichtstärke, die Beleuchtungsstärke und die Leuchtdichte zur Visualisierung. Im Kontext von HD-Scheinwerfern sind weitere Analysen sinnvoll. So wären beispielsweise die echtzeitfähige Einblendung der Lichtverteilungen beider Scheinwerfer oder der Dimmwerte denkbare Möglichkeiten. Die **Unterstützung von Analysewerkzeugen** wird deshalb in den Anforderungskatalog mit aufgenommen.

Ein ganz neuer Aspekt im Kontext von HD-Systemen sind die hinzu gewonnenen Entwurfsmöglichkeiten. Bisher endete der Entwurfsprozess der Lichtverteilung mit der Konstruktion des Scheinwerfers. Das Zusammenspiel aus Lichtquellen und Reflektorgeometrien bestimmt die Gestalt der Lichtverteilung. Eventuelle Verschwenkaktoren erlauben nachträgliche Anpassungen in geringem Maße. Mit dem Aufkommen der HD-Technologie wandelt sich der Entwurfsprozess maßgeblich. Die Konstruktion des Scheinwerfers stellt nur einen Teil des Entwurfs der Gesamtlichtverteilung dar. Ein zweiter, noch einflussreicherer Teil ergibt sich durch den Entwurfs der Lichtsteueralgorithmen, die auf die Gestalt der Gesamtlichtverteilung und insbesondere auf ihr dynamisches Verhalten im höchsten Maße Einfluss nehmen [KW18]. Vor diesem Hintergrund sollte eine Nachtfahrtsimulation bei der **Auslegung der Lichtsteueralgorithmen aktiv unterstützen** und durch geeignete **Entwurfsfunktionen** eine logische Fortsetzung der Analysefunktionen zur Verfügung stellen.

### 4.1.6 Witterung

Für Lichtfunktionen, wie das Schlechtwetterlicht, ist die reale Erprobung eine besondere Herausforderung. Neben den Schwierigkeiten, die für alle nächtlichen Erprobungsfahrten gelten, kommt hinzu, dass die notwendigen Testbedingungen besonders schwer zu finden sind. Die Wetterverhältnisse sind nicht beeinflussbar. Etwaige Phänomene, wie dichter Nebel, sind darüber hinaus nicht zuverlässig vorhersagbar. Geeignete Orte und Zeitpunkte für die Erprobung in einer Nebelfahrt entstehen daher spontan. In der Konsequenz ist die Vorbereitung eines Erprobungsszenarios kaum möglich. Deshalb werden derartige Lichtfunktionen meist in abgeschlossenen Räumen mit künstlichem Nebel oder Regen erprobt. Hierbei ist aufgrund des begrenzten Platzangebots und der fehlenden Straßenumgebung nur eine statische Validierung möglich.

Diesen Entwicklungsschwierigkeiten entgegenwirkend leitet sich die **Nachbildung verschiedener Witterungsbedingungen** als zusätzliche Anforderung an die Nachtfahrtsimulation ab. Die unterschiedlichen Witterungseinflüsse in der Realität sind vielfältig und komplex. Sie können im Rahmen dieser Arbeit nicht vollständig und detailliert abgebildet werden. Es soll jedoch anhand eines konkreten Beispiels deutlich gemacht werden, dass solche Phänomene zumindest vereinfacht rekonstruierbar sind und bei der simulativen Erprobung der Lichtfunktion berücksichtigt werden können.

### 4.1.7 Fahrdynamik

Oftmals kommt der realistischen Abbildung der Fahrdynamik im Kontext der Scheinwerfersimulation ein untergeordneter Stellenwert zu. Ein Großteil der Lichtfunktionen kann auch bei stark vereinfachter Fahrdynamik hinreichend beurteilt werden. Dennoch gibt es Situationen, in denen das dynamische Verhalten des Fahrzeugaufbaus einen ausschlaggebenden Einfluss auf das Scheinwerferlicht nimmt. Von vorrangiger Bedeutung sind dabei Nickbewegungen. Ein starkes Nicken, welches statisch durch eine asymmetrische Lastverteilung im Fahrzeug oder dynamisch durch starkes Bremsen hervorgerufen werden kann, äußert sich in einer vertikalen Verschiebung der Gesamtlichtverteilung. Bei aktivem Abblendlicht geht damit eine sicherheitskritische Verlagerung der Hell-Dunkel-Grenze einher. Lichtfunktionen zur dynamischen HDG-Einstellung sollen diesen Effekten entgegenwirken. Bei der simulativen Auslegung derartiger Lichtfunktionen kommt der genauen Abbildung der realen Fahrdynamik eine hohe Bedeutung zu. Das Fahrzeugmodell muss dazu hinreichend detailliert und über weite Bereiche parametrierbar sein. Nur so kann das Fahrverhalten des realen Fahrzeugs, welches zukünftig mit dem zu entwickelnden Scheinwerfersystem ausgestattet werden soll, in der Simulation nachgebildet werden. Das ASM Fahrzeugmodell der Firma dSPACE, visualisiert in Bild 4-3, stellt ein Beispiel für ein komplexes Fahrzeugmodell dar.


*Bild 4-3*: *Visualisierung des komplexen Fahrzeugmodells der ASM Toolsuite von dSPACE [Quelle: dSPACE].*

Um eine möglichst umfangreiche Testabdeckung der existierenden und denkbaren Lichtfunktionen zu gewährleisten, stellt die **Option zur Verwendung eines komplexen Fahrzeugmodells** eine weitere Anforderung an die Nachtfahrtsimulation dar.

### 4.1.8 Konfigurierbarkeit

Im Kontext der Nachtfahrtsimulation erweisen sich sehr viele Konfigurationen der zusammenwirkenden Komponenten als sinnvoll. So ist der Einsatz einer Nachtfahrtsimulation am Schreibtisch eines Lichtingenieurs ebenso plausibel, wie der Betrieb eines Großsimulators mit Fahrzeug-Mockup und Mehrkanalausgabe. Im Kontext der Echtzeitfähigkeit wurden bereits XiL-Tests diskutiert. Auch diese Funktionalität stellt hohe Ansprüche an eine flexible Konfigurierbarkeit der Nachtfahrtsimulation. Idealerweise variieren die Möglichkeiten der Lichtsteuerung von statischen Lichtverteilungen über MiL-Steueralgorithmen bis hin zur HiL-Anbindung des realen Steuergeräts via CAN Bus. Als weitere Konfigurationskomponente ist der Fahrmodus zu nennen. Hier stellt die eigene Steuerung des Fahrzeugs die Alternative mit der höchsten Interaktivität dar. Gleichzeitig kann es aber aus Gründen der Ablenkung und der Reproduzierbarkeit sinnvoll sein, den Streckenverlauf durch einen Autopiloten zu befahren. Zur bestmöglichen Vergleichbarkeit mit der realen Erprobungsfahrt ist es darüber hinaus denkbar, reale Fahrzeugtrajektorien aufzuzeichnen und auf der nachgebildeten virtuellen Strecke nachzufahren.

Um einen breiten Einsatz der Nachtfahrtsimulation in verschiedenen Use Cases zu ermöglichen, muss eine **umfassende Konfigurierbarkeit des hard- und softwareseitigen Setups** gegeben sein. Die Konfiguration umfasst dabei unter anderem den Fahrmodus, die Lichtsteuerung sowie die Ein- und Ausgabegeräte.

### 4.1.9 Parametrierbarkeit

Die große Stärke der Simulation ist die flexible Erzeugung beliebiger Testszenarien mit minimalem Aufwand sowie die vollständige Reproduzierbarkeit dieser Szenarien. Auf diese Weise schafft die simulationsbasierte Entwicklung einen erheblichen Zeit- und Kostenvorteil gegenüber der Erprobung in realen Nachtfahrten. Zudem entfällt das hohe Unfallrisiko bei der Fahrt mit nicht ausgereiften Systemen, welche häufig an Racks vor dem Fahrzeug montiert sind und somit keinen Fußgängerschutz gewährleisten.

Damit dieser Vorteil gewinnbringend genutzt werden kann, muss die Nachtfahrtsimulation eine umfassende und reproduzierbare Parametrierung des Testszenarios erlauben. Die Parameter gliedern sich dabei mindestens in die Themenfelder

- Streckenverlauf,
- Streckenumgebung,
- Verkehrsteilnehmer und ihre Bewegungstrajektorien,
- Tageszeit und Witterung,
- Egofahrzeug
- und Ego-Scheinwerfersystem

auf. Um ein konkretes Szenario fehlerfrei reproduzieren zu können, sollten sich die Werte aller Parameter sichern und zu einem späteren Zeitpunkt erneut laden lassen.

Insbesondere kann es bei einzelnen Parametern sinnvoll sein, zur Laufzeit Manipulationen vorzunehmen. So wird beispielsweise der direkte Vergleich zweier Scheinwerfersysteme bzw. Lichtverteilungen unter sonst gleichen Bedingungen möglich. Es empfiehlt sich deshalb eine weitere Untergliederung der Parameter in Offline-Parameter, welche vor Beginn der Simulation festgelegt werden und zur Laufzeit konstant sind, und Online-Parameter, die während der Simulation angepasst werden können.

Zusammengefasst ergibt sich die **umfassende, sicher reproduzierbare und zur Laufzeit manipulierbare Parametrierung** der Nachtfahrtsimulation als weitere Anforderung.

### 4.1.10 Remote-Fähigkeit

Innerhalb der Anforderung „Konfigurierbarkeit“ wurde bereits der Betrieb von Großsimulatoren angesprochen. Bei der Verwendung derartiger Großsimulatoren verfolgt man das Ziel, eine möglichst hohe Immersion des Fahrers zu erzeugen. Das gelingt, wenn alle im Kontext des Fahrens relevanten Sinne durch realistische und zueinander konsistente Reize angesprochen werden. Es ist selbstverständlich, dass der Fahrer dabei keine Aufgaben übernehmen sollte, die nicht der eigentlichen Fahraufgabe entsprechen. Administrative Aufgaben, wie die Initialisierung, das Starten oder Stoppen und die Anpassung von Parametern innerhalb der Simulation, sollten von einer weiteren Person in der Rolle eines Versuchsleiters übernommen werden. Hierzu muss die Nachtfahrtsimulation ferngesteuert von einem Leitstand aus betrieben werden können. Als letzte Anforderung ergibt sich die Existenz einer **Remote-Bedienung, welche mit der eigentliche Nachtfahrtsimulation über eine Netzwerkverbindung in Kontakt steht** und über alle notwendigen Funktionen zur Fernbedienung der Simulation verfügt.

## 4.2 Zusammenfassung der Anforderungen

Zur besseren Übersichtlichkeit und der späteren Referenzierbarkeit der definierten Anforderungen erfolgt nachfolgend eine indexierte Auflistung:

- **A1** Korrekte Abbildung photometrischer Zusammenhänge
- **A2** Unterstützung spektrale Lichtverteilungen
- **A3** Unabhängigkeit von der Scheinwerfer-Technologie
- **A4** Skalierbarkeit bezüglich der Pixelanzahl
- **A5** Unterstützung von XiL-Techniken
- **A6** Echtzeitfähigkeit und Latenz
- **A7** Lichtanalyse
- **A8** Lichtentwurf
- **A9** Unterstützung verschiedener Witterungsbedingungen

- **A10** Verwendbarkeit eines komplexen Fahrzeugmodells
- **A11** Konfigurierbarkeit des Hard- und Software-Setups
- **A12** Umfangreiche, reproduzierbare Parametrierung mit Laufzeitanpassung
- **A13** Remote-Bedienbarkeit

Nachfolgend wird auf die aufgelisteten Anforderungen nur noch in dieser Kurzform verwiesen, worunter die im Abschnitt 4.1 ausführlich diskutierten Bedeutungen zu verstehen sind.

## 4.3 Analyse bestehender Simulationen

Nachdem die Anforderungen an eine Nachtfahrtsimulation festgelegt sind, gilt es, die in Abschnitt 3.3 vorgestellten Nachtfahrtsimulationen hinsichtlich dieser Anforderungen zu prüfen. Hierbei soll herausgearbeitet werden, welche Elemente bereits in guter Qualität gelöst sind und an welchen Stellen prinzipielle Defizite vorliegen, deren weitere Betrachtung sich im Rahmen dieser Arbeit als gewinnbringend erweisen könnte.

Da alle genannten Nachtfahrtsimulationen im produktiven Umfeld seit mehreren Jahren erfolgreich eingesetzt werden, ist davon auszugehen, dass sie die Anforderungen **A1** und **A6** hinreichend erfüllen. Die Importfunktionen von Lichtverteilungen in allen Simulationen versichern die prinzipielle Gestaltbarkeit der Lichtverteilung. Abgesehen von MotionDesk unterstützen alle Simulation spektrale Lichtverteilungen zumindest bei der Simulation konventioneller Scheinwerfer. Insofern ist anzunehmen, dass auch farbliche Aspekte der Lichtverteilung korrekt abgebildet werden können.

Alle am Markt verfügbaren Nachtfahrtsimulationen weisen Defizite bezüglich der Unterstützung von HD-Scheinwerfersystemen auf. Zu Beginn dieser Forschungsarbeit konnte keine der genannten Simulationen eine Funktion vorweisen, welche die Simulation von HD-Systemen ermöglicht. Doch auch bis heute ist die Unterstützung lückenhaft realisiert. Hinsichtlich der Pixelzahl bewegen sich alle Nachtfahrtsimulationen mit Ausnahme von LucidDrive deutlich unter 1.000 Einzellichtquellen. Diese Begrenzung erlaubt zwar die Simulation von Matrix-Systemen, sie stößt jedoch bei HD|SSL- und LCD-Systemen an ihre Grenzen. Ebenso können DMD- oder Scanner-Systeme nicht simulativ abgebildet werden. In dieser Hinsicht erfüllen sie die Anforderungen **A3** und **A4** nicht. LucidDrive hingegen erlaubt mit mindestens 12.800 Pixellichtquellen pro Scheinwerfer eine wesentlich höher aufgelöste Darstellung. Auf dieser Basis ist zumindest die Simulation von $\mu$AFS- und LCD-Systemen in vielen Fällen möglich. Nachteilig hierbei ist jedoch, dass spektrale Lichtverteilungen im Rahmen der HD-Simulation nicht unterstützt werden. Somit erfüllt LucidDrive die Anforderungen **A3** und **A4** deutlich besser, kann jedoch der Anforderung **A2** nicht standhalten.

Drei der fünf vorgestellten Nachtfahrtsimulationen unterstützen die HiL-Einbindung des Scheinwerfersteuergeräts (**A5**) und ermöglichen somit eine hohe Testabdeckung. Gleiches gilt für die Skalierbarkeit. Leider lässt sich LucidDrive als beste Simulation bezüglich der Anforderung **A4** nicht vollständig skalieren. Die maximale Anzahl von Ausgabegeräten ist auf drei begrenzt. Ein Betrieb von Großsimulatoren ist deshalb im Regelfall nicht möglich. Beispielsweise verfügt der ATMOS Fahrsimulator des Heinz Nixdorf Instituts

über elf Ausgabemedien. LucidDrive erfüllt die Anforderung **A11** somit nicht. Diejenigen Simulationen, die für den Großsimulatorbetrieb geeignet sind, bieten auch eine Remote-Bedienbarkeit gemäß der Anforderung **A13**. Für die übrigen Simulationen besteht keine Notwendigkeit einer solchen Remote-Bedienbarkeit.

Mit Ausnahme von MotionDesk verfügen alle Nachtfahrtsimulationen über die üblichen Analysetools zur Visualisierung von photometrischen Größen. Das sind vorrangig Visualisierungen von Isolinien- oder Falschfarbendarstellungen der photometrischen Größen. Die Anforderung **A7** kann für die übrigen Simulationen als erfüllt betrachtet werden. Eine andere Situation ergibt sich für die aktive Unterstützung beim Entwurf von Lichtfunktionen. Die Anforderung **A8** muss als gänzlich ungelöst betrachtet werden. Keine der Nachtfahrtsimulationen stellt Funktionen zur Auslegung der Lichtsteueralgorithmen bereit.

Bei Überprüfung der Anforderung **A9** ist ein vergleichbarer Missstand auffindbar. Auch wenn der Großteil der Nachtfahrtsimulationen verschiedene Witterungsbedingungen darstellen kann, sind die zugrunde liegenden Ansätze derart vereinfacht, als dass die Witterungseinflüsse auf keinem physikalischen Modell basieren und insofern nicht mit dem Scheinwerferlicht interagieren. Diese Interaktion ist jedoch essentiell, um jene reale Phänomene abzubilden, denen mit fortschrittlichen Lichtsteueralgorithmen zu begegnen ist. Die Anforderung **A9** muss insofern ebenfalls als nicht erfüllt betrachtet werden.

Auch bezüglich der Tiefe der Fahrzeugmodellierung zeigen sich die vorgestellten Nachtfahrtsimulationen sehr divers. Sie reichen von sehr einfachen, kinematischen Modellen in den Simulationsumgebungen LightDriver und LucidDrive bis hin zu hochkomplexen, detaillierten Fahrzeugmodellen, wie ASM oder ADAMS. Die Anforderung **A10** wird demnach von drei der fünf Nachtfahrtsimulationen vollständig erfüllt.

Da alle vorgestellten Simulationen seit langem im industriellen Umfeld eingesetzt werden, stellen sie durch hochwertige grafische Benutzeroberflächen eine zugängliche Bedienung sicher. Dazu gehören auch die Möglichkeit einer umfangreichen, reproduzierbaren Parametrierung sowie die Anpassung von Parametern während der Simulation. Die Anforderung **A12** kann somit als unkritisch betrachtet werden.

## 4.4 Ableitung des Handlungsbedarfs

Die Gegenüberstellung verfügbarer Nachtfahrtsimulationen an den in Abschnitt 4.2 zusammengefassten Anforderungen hat gezeigt, dass einige Punkte bereits in hoher Qualität geleistet werden, während andere noch lückenhaft oder gar nicht gelöst worden sind. Aus dieser Motivation heraus soll die hier vorgestellte Arbeit eine Nachtfahrtsimulation hervorbringen, die insbesondere in den Punkten überzeugt, die aktuell als ungelöst betrachtet werden müssen.

Eine bisher nur lückenhaft gelöstes Problem stellt die echtzeitfähige Simulation von HD-Systemen dar, die weit über die Auflösung von Matrix-Systemen mit wenigen hundert Pixeln hinaus gehen. Als einziger Anbieter liefert Synopsys mit LucidDrive eine Nachtfahrtsimulation, welche Simulationen von Systemen mit über 10.000 Pixeln erlaubt. LucidDrive unterstützt jedoch keine spektralen Simulationen, obwohl wellenlängenabhängige Gestaltunterschiede in der Lichtverteilung insbesondere im Kontext von LED-basierten Systemen

zu berücksichtigen sind. Als zweiter wesentlicher Nachteil von LucidDrive ist die geringe Konfigurierbarkeit zu nennen, welche den Betrieb in Großsimulatoren mit mehr als drei Ausgabegeräten verhindert.

Die Untersuchung der bestehenden Simulationen hat außerdem gezeigt, dass der durch die HD-Technologie neu gewonnene Entwurfsfreiheitsgrad in Form der Lichtsteueralgorithmen bisher in keiner Software Berücksichtigung findet. Die hier entstehende Nachtfahrtsimulation soll diesen Aspekt aufgreifen und den Entwickler aktiv bei der Auslegung von Lichtsteueralgorithmen unterstützen.

Ein drittes großes Defizit ist die zu stark vereinfachte Modellierung von Witterungseinflüssen. Keine der am Markt verfügbaren Simulationen modelliert Witterungseinflüsse auf einer physikalischen Basis. In der Konsequenz wird die Wechselwirkung zwischen dem Scheinwerferlicht und den Witterungseinflüssen nicht berücksichtigt, weshalb die Witterungssimulation für die Auslegung darauf abgestimmter Lichtfunktionen ungeeignet ist. Ziel muss es sein, Witterungsverhältnisse so zu modellieren, dass es zu einer visuell wahrnehmbaren und physikalisch plausiblen Wechselwirkung mit dem Scheinwerferlicht kommt. Die Ausprägungen der Witterungseinflüsse sind in der Realität vielfältig. Nebel, Regen und Schnee können in verschiedenen Ausprägungen und ausgelöst durch vollkommen unterschiedliche Phänomene Einfluss auf die Sicht des Fahrers nehmen. Innerhalb der Arbeit wird deshalb mit dem Nebel beispielhaft ein Witterungseinfluss herausgegriffen, um die Machbarkeit einer physikalisch motivierten Modellierung zu untersuchen.

Zusammengefasst stellen die echtzeitfähige spektrale HD-Simulation, die aktive Unterstützung beim Entwurf von Lichtsteueralgorithmen und die physikalisch basierte Modellierung von Witterungseinflüssen die bisher noch ungelösten Problemstellungen dar, aus welchen sich der wissenschaftliche Handlungsbedarf dieser Arbeit ableitet. Im Sinne einer umfassenden und nutzbaren Lösung, müssen außerdem die übrigen in Abschnitt 4.2 aufgeführten Anforderungen durch die im Rahmen der Arbeit entwickelten Nachtfahrtsimulation erfüllt werden. Da diese nicht den wissenschaftlichen Kern der vorliegenden Arbeit darstellen, werden sie im Rahmen des nachfolgenden Kapitels 5 zur Architektur der Simulation oberflächlich beschrieben, während die wissenschaftlichen Kernthemen in den Kapiteln 6 und 7 im Detail diskutiert werden.

# 5 Architektur der Simulation

Gegenstand dieses Kapitels ist die Vorstellung der unter den Vorgaben aus Kapitel 4 entwickelten Nachtfahrtsimulation, welche fortführend auch als „Hyperion“ bezeichnet wird. Das Architekturbild 5-1 ist möglichst allgemeingültig gehalten, um einen Großteil der möglichen Konfigurationen von Hyperion zu erfassen. Dennoch können nicht alle Use Cases durch Bild 5-1 erfasst werden. Im Abschnitt 5.8 werden spezifischere Architekturen zu konkreten Konfigurationen dargestellt.

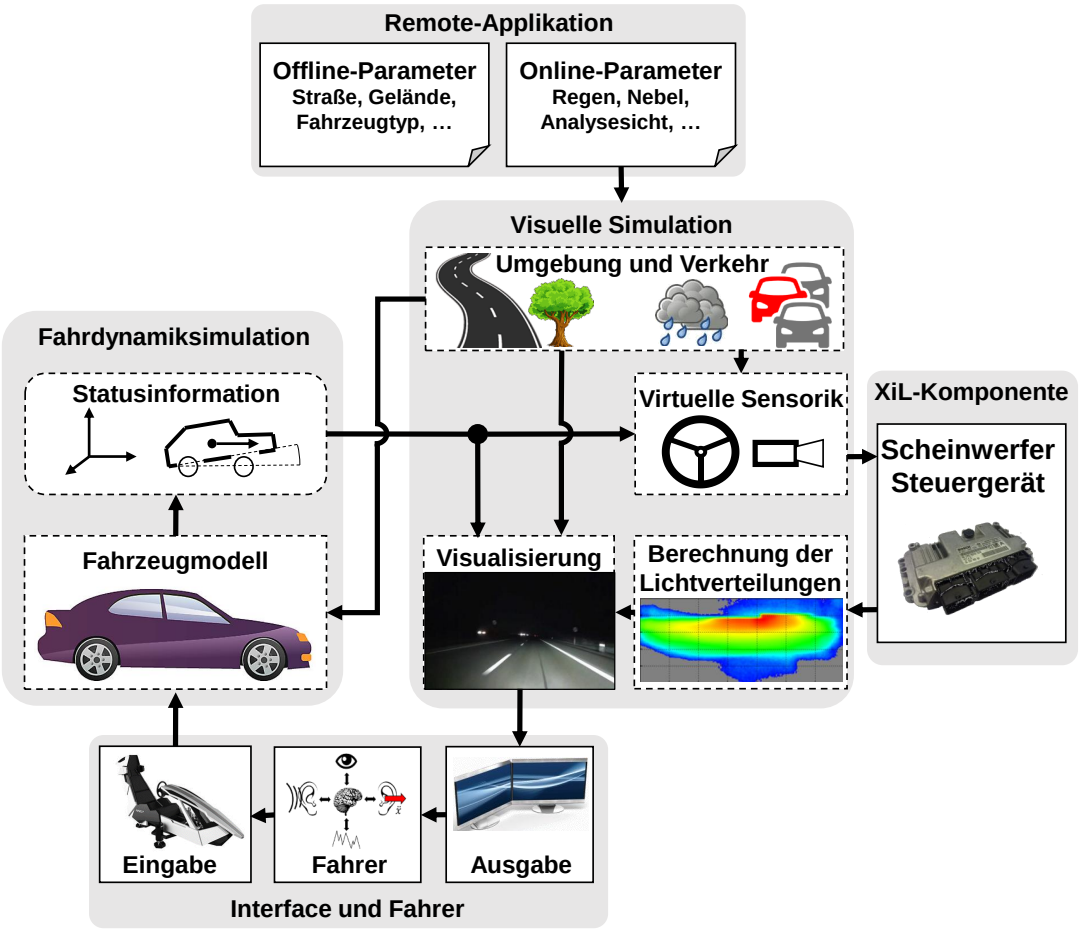


*Bild 5-1*: *Abstraktes Architekturdiagramm der Nachtfahrtsimulation „Hyperion“.*

Die grau hinterlegten Bereiche fassen logische Einheiten der Simulationsarchitektur zusammen. Abhängig von der gewählten Konfiguration kann jede logische Einheit auf einem separaten technischen System ablaufen oder zusammen mit anderen logischen Einheiten auf demselben System ausgeführt werden. Die logischen Einheiten sind im Detail die visuelle Simulation, die Fahrdynamiksimulation, das Steuergerät als XiL-Komponente, das Ein- und Ausgabeinterface für den Fahrer bzw. Entwickler und die optionale Remote-Applikation zur Fernsteuerung von Hyperion. Zwischen den logischen Einheiten existie-

ren verschiedene Wechselwirkungen und Abhängigkeiten. Insbesondere findet sich im Architekturbild ein Regelkreis wieder, welcher durch die logischen Einheiten „Interface und Fahrer“, „Fahrdynamiksimulation“, „Visuelle Simulation“ und „XiL-Komponente“ geschlossen wird. Der Fahrer nimmt dabei die Rolle des Reglers ein, wobei seine visuelle Wahrnehmung die Regelgröße und die Fahreingaben die Stellgröße darstellen. Es wird in diesem Kontext auch von Human-in-the-Loop-Anwendungen gesprochen.

Der folgende Abschnitt erklärt die ausgewählte Entwicklungsumgebung und die Hintergründe für diese Entscheidung. Im Anschluss werden die Komponenten der in Bild 5-1 dargestellten Architektur beschrieben.

## 5.1 Entwicklung in Unity3D

Im Rahmen dieses Abschnitts kann nur ein kleiner Ausschnitt der Entwicklungsumgebung und der Konzepte, die der Unity-Engine zugrunde liegen, beschrieben werden. Für eine tiefergehende Auseinandersetzung mit Unity3D stellt Unity Technologies eine umfangreiche Plattform bereit, die über eine Vielzahl von Kursen, Tutorien und Beispielprojekten verfügt [Uni20].

In Abschnitt 2.3.1 wurde bereits diskutiert, dass bei der Implementierung einer umfangreichen 3D-Anwendung auf eine Engine zurückgegriffen werden sollte, welche bereits viele vorgefertigte Inhalte und Funktionen bereitstellt. Im Rahmen der Nachtfahrtsimulation ist insbesondere die Wahl einer Spiele-Engine sinnvoll, da diese zusätzlich bei der Modellierung physikalischer Sachverhalte und weiteren wiederkehrenden Erfordernissen, wie beispielsweise bei dem Management des Szenengraphen, unterstützt. Derzeit wird der Markt durch verschiedene Spiele-Engines angeführt, von denen Unity3D und Unreal ohne Lizenzgebühren eingesetzt werden dürfen [Uni20], [Epi20]. Beide Produkte sind grundsätzlich geeignet, um eine Nachtfahrtsimulation zu implementieren. Die Funktionsumfänge von Unity3D und Unreal befinden sich auf Augenhöhe, wobei sie sich in einigen Details unterscheiden [Vis20]. Am Heinz Nixdorf Institut bestand zu Beginn der Arbeiten durch das zuvor durchgeführte Traffis-Projekt eine größere Wissensbasis in Unity3D [Gau15]. Deshalb fiel die Entscheidung auch im Rahmen der vorliegenden Arbeit auf diese Engine, welche darüber hinaus als leichter zu erlernen gilt.

Da nachfolgend mehrfach Bezug auf Unity3D genommen wird, soll die Spiele-Engine und ihre zugehörige Entwicklungsumgebung nun vorgestellt werden. Der hier vorgestellte Stand von Hyperion wird mit der Version 2019.3.12f1 von Unity3D betrieben. Das Bild 5-2 zeigt die Benutzeroberfläche der Entwicklungsumgebung.

Die Oberfläche setzt sich aus verschiedenen Elementen zusammen. Im linken Bereich (s. Bild 5-2a) findet man den **Szenegraphen** (auch: Hierarchie). Hierbei handelt es sich um eine hierarchische Anordnung von Szenenelementen. Das können neben physischen Objekten oder Lichtquellen der Szene auch nicht sichtbare, logische Elemente sein. Ein Element des Szenengraphs wird in Unity3D als **Gameobject** bezeichnet. Jedes Gameobject kann **Children** haben. Hierbei handelt es sich um weitere Gameobjects, die einem anderen Gameobject (**Parent**) in der hierarchischen Struktur untergeordnet sind. Diese Unterordnung äußert sich darin, dass bestimmte Veränderungen eines Parent auch auf dessen Children angewandt werden. So werden sie beispielsweise bei jeder Veränderung der Position, Rotation oder Skalierung ihres Parent ebenfalls verschoben, rotiert und skaliert. Jedes

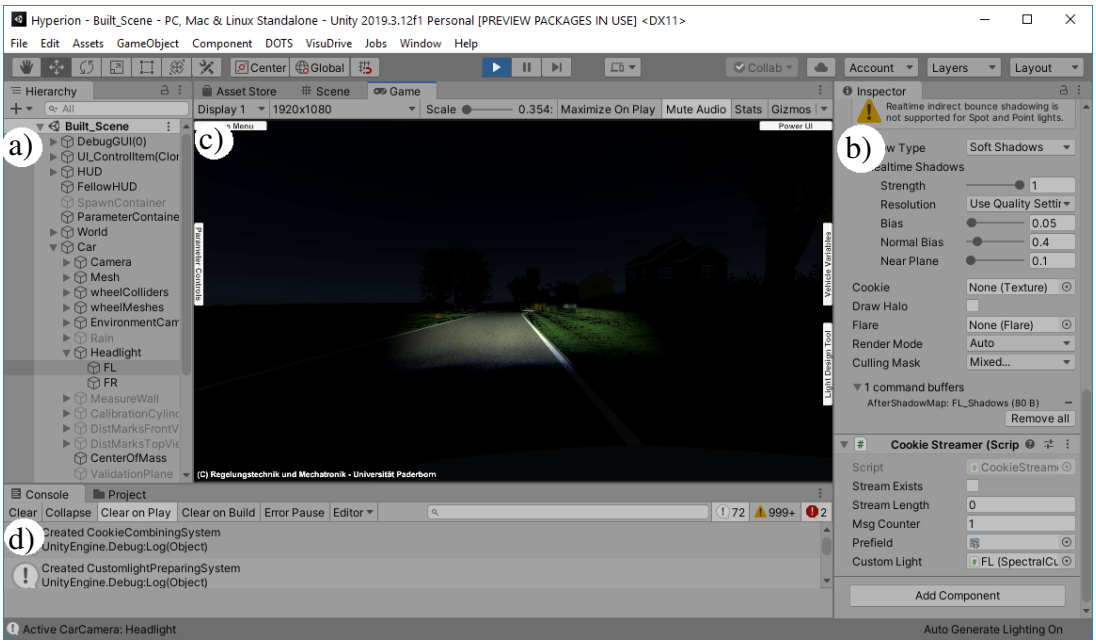


*Bild 5-2*: *Benutzeroberfläche der Entwicklungsumgebung und Spiele-Engine Unity3D in der Version 2019.3.12f1.*

Gameobject kann ein oder mehrere **Components** beinhalten. Erst durch die Components bekommt das Gameobject eine Funktion. Man kann ein Gameobject auch als Container für Components begreifen. Durch die Selektion eines Gameobjects im Szenegraphen werden dessen Components im **Inspector** angezeigt (s. Bild 5-2b). Das einzige unbedingt notwendige Component ist das **Transform**-Component. Es enthält alle Informationen zur Formulierung einer Transformationsmatrix, welche die Lage, Orientierung und Skalierung des Gameobjects innerhalb der Szene bzw. bezüglich des World-Space festlegen. In Bild 5-3 ist die Darstellung des Inspectors bei Selektion eines Gameobjects zu sehen, welches lediglich über ein Transform-Component verfügt. Die Checkbox neben dem Bezeichner „Leeres GameObject“ kennzeichnet die Aktivität des Gameobject. Durch das Deaktivieren verhält sich die grafische Ausgabe, als wäre das entsprechende Gameobject nicht im Szenegraphen enthalten. Darunter findet sich eine Auflistung aller Components, über die das selektierte Gameobject verfügt. Typischerweise besitzen Components Attribute, die komfortabel im Inspector angepasst werden können. Im Fall des Transform-Components sind das die Position, die Rotation und die Skalierung bezüglich der Raumachsen. Durch den „Add Component“-Button (s. Bild 5-3 unten) können weitere Components an ein Gameobject geknüpft werden. Dabei lassen sich zusätzlich zu den Standard-Components der Unity-Engine eigene Components durch sogenannte **Skripte** programmieren. Diese Skripte können in der Programmiersprache C# formuliert werden und müssen von der Basisklasse „MonoBehaviour“ aus der Unity-Engine abgeleitet sein. Damit wird sichergestellt, dass die Anbindung an ein Gameobject möglich ist und zentrale Methoden, wie das Updateverhalten pro Frame oder die Initialisierung beim Programmstart durch den Entwickler spezifiziert werden können.

Der zentrale und größte Bereich in Bild 5-2c wird durch den **Scene- und Gameview** eingenommen, zwischen welchen durch die darüber befindlichen Reiter gewechselt werden kann. Der Sceneview ist immer verfügbar. Dieser wird für eine beispielhafte Szene aus

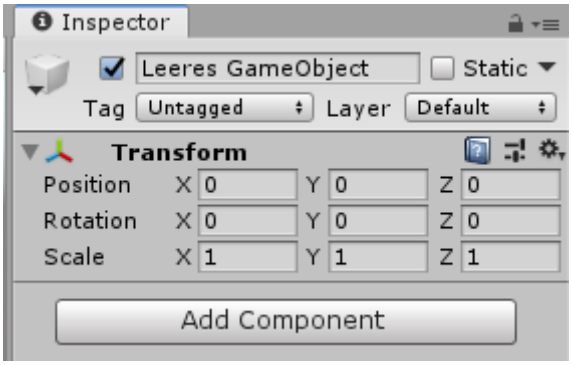


*Bild 5-3*: *Ein leeres Gameobject im Inspector der Entwicklungsumgebung Unity3D.*

Hyperion im oberen Teil von Bild 5-4 dargestellt. Er kann als anschaulichere Alternative zum Szenegraphen verstanden werden und gibt einen Eindruck, wie die Szene in der späteren Anwendung aussehen wird. Im Sceneview werden die Gameobjects der Szene bereits in der richtigen Lage und Orientierung dargestellt. Im Unterschied zum Gameview werden jedoch keine aufwendigen Rendering-Operationen ausgeführt. Außerdem zeigt der Sceneview weitere Informationen (**Gizmos**), die im Gameview zwar verborgen bleiben sollen, jedoch hilfreich bei der Implementierung sind. Beispiele hierfür sind die Kamerapositionen (siehe weiße Kamerasymbole im Bereich des Fahrzeugs in Bild 5-4 oben) oder die Ursprünge und Ausrichtungen der jeweiligen Objektkoordinatensysteme (siehe roten, grünen und blauen Achsenpfeil in Bild 5-4 oben). Welche Anteile dargestellt werden sollen, kann über die Aus- und Abwahl der Gizmos weitgehend frei parametriert werden. Der Sceneview soll den Entwickler in der Gestaltung des Szenegraphen unterstützen. Während der Gameview die Szene aus Sicht einer definierten Kamera darstellt, kann sich der Entwickler im Sceneview frei durch die Szene bewegen. Die direkte Manipulation des Szenegraphen ist im Sceneview ebenfalls möglich. Objekte können mit direktem visuellen Feedback verschoben, rotiert und skaliert werden. Genauso lassen sich Gameobjects kopieren oder löschen.

Der Gameview, dargestellt im unteren Teil von Bild 5-4, zeigt den aktuellen Stand der Anwendung, wie sie ein potentieller Benutzer wahrnehmen würde. Insofern dient der Gameview als Kontrollinstrument für den Entwickler. Die Szene wird aus Sicht der Kamera und der an ihr gebundenen Parameter und Components dargestellt. Die freie Bewegung in der Szene und die Manipulation des Szenegraphen sind nicht möglich. Es sind nur durch den Entwickler vorgesehene Aktionen verfügbar. So sind beispielsweise Schaltflächen (siehe weiße Felder am Bildrand in Bild 5-4 unten), die zur Interaktion mit dem Benutzer dienen, vorhanden. Alle Gameobjects und ihre Components verhalten sich wie in der laufenden Anwendung. Benutzereingaben werden verarbeitet und durch entsprechende Reaktionen ausgeführt. So kann der Entwickler das Verhalten der Anwendung schon vor dem Kompilieren umfänglich testen.

In Bild 5-2d finden sich die **Konsole** und der **Projektexplorer**. Letzterer dient ähnlich einem Dateiexplorer eines Betriebssystems zur Navigation in der Ordnerstruktur eines Unity-Projekts. Der Projektexplorer ermöglicht die komfortable Verknüpfung des Szenegraphen mit den Projektdateien. Ein Beispiel hierfür ist die Zuweisung einer als Bilddatei vorliegenden Textur auf die Oberfläche eines geometrisch ausgedehnten Gameobjects.


*Bild 5-4*: *Gegenüberstellung von Sceneview (oben) und Gameview (unten) in der Entwicklungsumgebung Unity3D.*

Die Konsole dient dem Debugging während der Anwendungsentwicklung. Sie zeigt textuelle Ausgaben der Unity-Engine, Fehlermeldungen, Warnungen und weitere durch den Entwickler ausgelöste Meldungen an. Da derartige Meldungen nicht in der grafischen Ausgabe für den Benutzer sichtbar sein sollten, bietet die Konsole eine komfortable Möglichkeit des Monitorings des Programmverhaltens ohne in die grafische Oberfläche der Anwendung eingreifen zu müssen.

## 5.2 Fahrdynamiksimulation

Aufgrund der überschaubaren Abhängigkeiten von anderen Logikeinheiten wird die Fahrdynamiksimulation als erste Architektur-Komponente diskutiert. Ihre Aufgabe ist die Modellierung und fortlaufende Berechnung des Fahrzeugzustands in Abhängigkeit von

den Fahrereingaben und den Wechselwirkungen mit der Fahrbahn und Umgebung. Von besonderem Interesse sind hierbei die Lage, Orientierung und weitere lichtrelevante Größen des Egofahrzeugs, da diese von anderen logischen Einheiten der Nachtfahrtsimulation benötigt werden.

Im Rahmen der Simulation wird die Fahrdynamiksimulation in zwei Varianten mit unterschiedlichen Komplexitätsgraden unterstützt. Zum einen existiert ein internes, vereinfachtes Fahrzeugmodell. Hierbei wird über den Simulationsrechner hinaus keine zusätzliche Hardware benötigt. Die logischen Einheiten „Fahrdynamik“ und „Visuelle Simulation“ werden auf einer gemeinsamen Recheneinheit betrieben und bilden eine Applikation. Dieses Setup empfiehlt sich beispielsweise zur Verwendung an einem normalen Büroarbeitsplatz, an dem typischerweise keine Echtzeithardware zur Verfügung steht. Bei dieser Variante werden viele Vereinfachungen getroffen, die eine detaillierte Nachbildung eines realen Fahrzeugs nicht ermöglichen. Da viele Lichtfunktionen weitestgehend unbeeinflusst von der Fahrdynamik agieren, genügt diese Variante dennoch, um eine Vielzahl von Lichtfunktionen simulativ zu erproben. Ein besonderer Vorteil des internen Fahrzeugmodells ist die Kompaktheit der Gesamtkonfiguration und die Unabhängigkeit von Drittanbieter-Software. Es wird in Unterabschnitt 5.2.1 detaillierter vorgestellt.

Um der Anforderung **A10** (komplexes Fahrzeugmodell) gerecht zu werden, kann alternativ zum internen Fahrzeugmodell die Anbindung an ein dSPACE-Echtzeitsystem mit ASM Fahrzeugmodell erfolgen. Das ASM Fahrzeugmodell wurde in Abschnitt 3.3 bereits kurz vorgestellt. Es handelt sich um ein komplexes Modell mit vielen Freiheitsgraden und Parametern. Durch die Berechnung des Modells auf einem Echtzeitsystem ist die echtzeitfähige Simulation mit hoher Taktrate sichergestellt. Der Unterabschnitt 5.2.2 stellt das Modell detaillierter vor und beschreibt die notwendigen Erweiterungen und Schnittstellen zur Integration in Hyperion.

Den Abschluss dieses Abschnitts bilden die fremdgesteuerten Fahrmodi. Diese Steuerungsvarianten benötigen kein Fahrzeugmodell, da die Trajektorie des Fahrzeugs entweder exakt der Sollspur bei vorgegebenen Wunschgeschwindigkeiten folgt, oder die Trajektorie und alle weiteren lichtrelevanten Fahrzeuggrößen im Vorhinein aufgezeichnet wurden. Vorteile der fremdgesteuerten Fahrmodi sind zum einen das Entfallen der Fahraufgabe, welche eine stärkere Konzentration auf das Licht zulässt, und zum anderen die vollständige Reproduzierbarkeit der gefahrenen Trajektorie.

### 5.2.1 Internes Fahrzeugmodell

Das interne Fahrzeugmodell wird vollständig in der Unity3D-Entwicklungsumgebung modelliert. Hierzu kann auf einige Components der Unity-Engine zurückgegriffen werden, die verschiedene Aspekte des Gesamtfahrzeugmodells abbilden. Bei dem internen Fahrzeugmodell handelt es sich um ein Mehrkörpersystem bestehend aus dem Fahrzeugaufbau $m_A$ und den vier Rädern der Masse $m_R$. Im Szenegraphen ist jedes Fahrzeug ein Gameobject, welches zumindest die folgende Hierarchie aufweist:

- **Car** mit Components: *Transform, Rigidbody, Car Model, Car Input*
  - **COG** mit Components: *Transform*
  - **Mesh** mit Components: *Transform, Mesh Filter, Mesh Renderer, Mesh Collider*

- **Wheel Colliders** mit Components: *Transform*
  - **ColliderFL** mit Components: *Transform, Wheel Collider*
  - **ColliderFR** mit Components: *Transform, Wheel Collider*
  - **ColliderRL** mit Components: *Transform, Wheel Collider*
  - **ColliderRR** mit Components: *Transform, Wheel Collider*
- **Wheel Meshes** mit Components: *Transform*
  - **MeshFL** mit Components: *Transform, Mesh Filter, Mesh Renderer*
  - **MeshFR** mit Components: *Transform, Mesh Filter, Mesh Renderer*
  - **MeshRL** mit Components: *Transform, Mesh Filter, Mesh Renderer*
  - **MeshRR** mit Components: *Transform, Mesh Filter, Mesh Renderer*

In dieser Auflistung sind Gameobjects fett und ihre zugehörigen Components kursiv geschrieben. Das Gameobject COG spezifiziert in seinem Transform-Component die Lage des Fahrzeugschwerpunkts. Mesh Filter und Mesh Renderer dienen nur der grafischen Darstellung des jeweiligen Gameobject und haben keine physikalische Relevanz. Der Mesh Filter verknüpft sein Gameobject mit dem jeweiligen Polygonnetz (auch: Mesh), welches dessen Geometrie modelliert. Dazu muss das Mesh im Projektverzeichnis liegen. Der Mesh Renderer definiert die Visualisierung des Meshes. Er verknüpft es mit einem Material, sodass unter anderem eventuelle Texturen, Shader und Beleuchtungsgrößen für das Rendering festgelegt sind. Durch die Manipulation des Transform-Components kann das Mesh zum Koordinatenursprung des Gameobject ausgerichtet werden. Das Mesh des Fahrzeugaufbaus wird durch den Mesh Filter des Car-Gameobject referenziert. Die Visualisierungen der vier Räder sind im Wheel Meshes-Gameobject organisiert. Jedes Rad wird durch ein eigenes untergeordnetes Gameobject abgebildet. Die physikalische Berücksichtigung der Räder ist aus praktischen Gründen davon unabhängig und findet sich innerhalb des Wheel Collider-Gameobjects.

### Fahrzeugaufbau

Zur physikalischen Modellierung des Fahrzeugaufbaus eignet sich ein Rigidbody-Component. Dieses Standard-Component von Unity dient zur generellen Modellierung von Festkörpern, welche von Kräften beeinflusst werden und sich unter deren Einfluss bewegen. Das Bild 5-5 zeigt die zur Verfügung stehende Parametrierung des Rigidbody-Components.

Der Parameter *Mass* definiert die Masse des Festkörpers in Kilogramm. Zur Abbildung des Luftwiderstands können die Parameter *Drag* und *Angular Drag* verwendet werden. Dabei bezieht sich *Drag* auf den Widerstand bei translatorischen und *Angular Drag* auf den Widerstand bei rotatorischen Bewegungen. Die interne Berücksichtigung dieser Werte ist jedoch nicht physikalisch korrekt. In Hyperion werden deshalb beide Parameter auf null gesetzt. Stattdessen wird der Luftwiderstand über ein eigenes Skript, bezeichnet mit „Air Resistance“, berücksichtigt. Als einzige Parameter können der $c_w$-Wert und die Stirnfläche $A_S$ des Fahrzeugs in diesem Script-Component definiert werden. Der

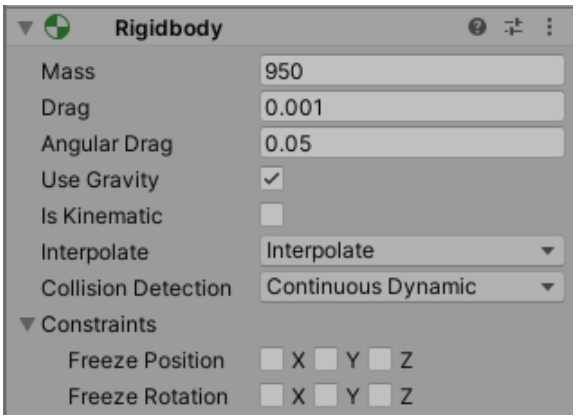


*Bild 5-5*: *Attribute des Rigidbody-Components der Unity-Engine.*

rotatorische Luftwiderstand wird vernachlässigt. Innerhalb des Skripts werden Betrag und Richtung der Luftwiderstandskraft $F_L$ gemäß

$$F_L = -\frac{1}{2} \cdot \rho_L \cdot c_w \cdot A_S \cdot |v_{Fzg}| \cdot v_{Fzg}$$

berechnet. Dabei bezeichnet $\rho_L$ die Dichte der Luft und $v_{Fzg} \in \mathbb{R}^3$ den Vektor der Fahrzeuggeschwindigkeit. Der Kraftvektor $F_L$ wirkt am Schwerpunkt des Fahrzeugaufbaus.

Mit der Checkbox „Use Gravity“ in Bild 5-5 kann bestimmt werden, ob der Festkörper dem Einfluss der Schwerkraft unterliegt. Für das Fahrzeug ist diese Checkbox aktiv. In Unity wirkt die Schwerkraft, sofern nicht anders definiert, in Richtung der negativen $y$-Achse des Weltkoordinatensystems. Ihr Angriffspunkt ist am Schwerpunkt des Rigidbody. Unity berechnet den Schwerpunkt aus allen Kollisionsgeometrien (auch Collider), die an das Rigidbody gebunden sind. Alternativ kann der Schwerpunkt über das Attribut *centerOfMass* des Rigidbody manuell definiert werden. Diese Variante wird hier genutzt. Zur intuitiven Festlegung des Schwerpunkts wird dieser als Child-Gameobject „COG“ des Fahrzeugs mit eigenem Transform abgebildet. So kann die momentane Lage des Schwerpunkts im Sceneview visualisiert und manipuliert werden. Die nachfolgende „Is Kinematic“-Checkbox muss deaktiviert werden, damit das Rigidbody durch Kräfte beeinflusst werden kann. Ist diese Checkbox aktiv, so kann das Fahrzeug nur durch die direkte Manipulation des Transform durch Skripte oder vorgefertigte Animationen bewegt werden.

Das Dropdown-Menü „Interpolate“ legt die Interpolation der Bewegung von Frame zu Frame fest. Da die Physik-Engine von Unity mit einer festen Framerate operiert, während das Rendering abhängig von der Rechenlast und der zur Verfügung stehenden Leistung eine variable Framerate aufweist, sind die momentane Fahrzeugposition und die grafische Ausgabe nicht exakt synchronisiert. Zur Auswahl stehen die Modi *None*, *Interpolate* und *Extrapolate*. Um störende visuelle Artefakte zu vermeiden, die insbesondere dann auftreten, wenn die Kamera das betreffende Rigidbody verfolgt, sollte für das Egofahrzeug eine Interpolation durchgeführt werden.

Soll das derzeitige Rigidbody in die Kollisionsprüfung einbezogen werden, so kann durch das zweite Dropdown-Menü aus Bild 5-5 die Art und Weise der Berücksichtigung spezifiziert werden. Für sich schnell bewegende Objekte wird die Einstellung *Continous Dynamic* empfohlen. Voraussetzung für die Durchführbarkeit der Kollisionsprüfung ist

das Vorhandensein eines Colliders. Diese stehen in Unity in verschieden komplexen Formen zur Verfügung. Besonders flexibel ist ein Mesh Collider, wie er in dieser Arbeit für den Fahrzeugaufbau verwendet wird. Mit wachsender Polygonzahl wird die Kollisionsprüfung jedoch immer rechenintensiver. Deshalb approximiert man das grafische Mesh meist grob mit einem zweiten Mesh, das ausschließlich von dem Collider verwendet wird. Dieser besitzt also ein eigenes Mesh Filter für dieses Kollisions-Mesh. In Bild 5-6 wird das zur Kollisionsprüfung eingesetzte Mesh des Fahrzeugs dargestellt, indem dessen Polygonkanten durch grüne Linien hervorgehoben werden (auch Wireframe).

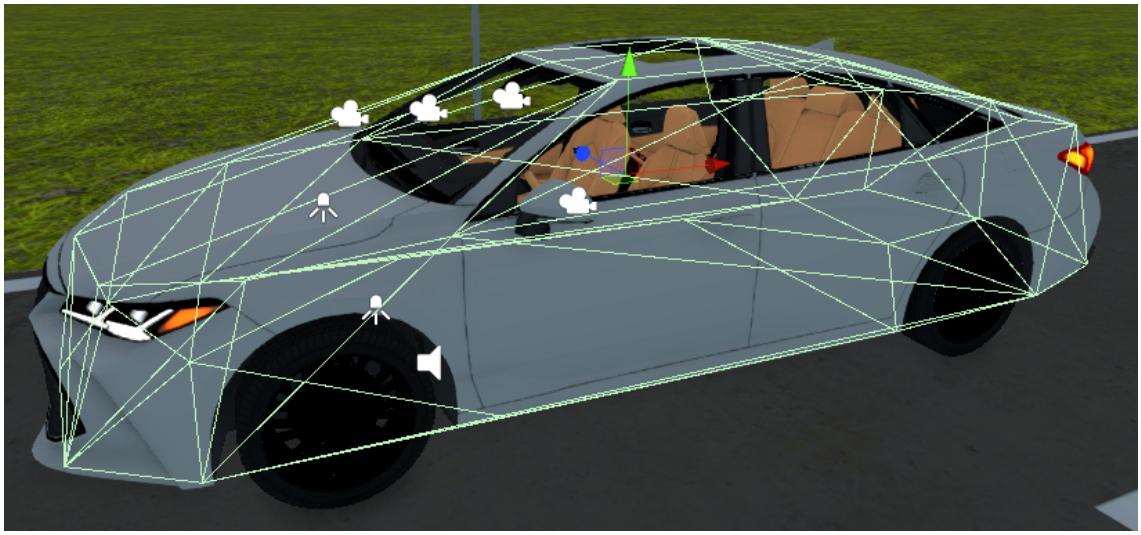

*Bild 5-6*: *Mesh zur Kollisionsprüfung des Fahrzeugs visualisiert durch grünen Wireframe.*

### Rad und Reifen

Neben dem Fahrzeugaufbau stellen die vier Räder die weiteren Massen des Fahrzeugmodells dar. Zur Modellierung von Rädern stellt Unity spezielle Wheel Collider Components zur Verfügung, welche die Radaufhängung am Fahrzeugaufbau und den Rad-Straße-Kontakt abbilden. In Bild 5-7 werden die im Inspector dargestellten Parametriermöglichkeiten des Wheel Colliders gezeigt.

Der erste Parameter „Mass“ bezeichnet die Radmasse in Kilogramm. Es schließt sich der dynamische Radradius in Metern an. Die Wheel Damping Rate nimmt Einfluss auf die rotatorische Bewegung des Rads. Sie hat keinen Einfluss auf die Vertikaldynamik. Da die genaue Einflussnahme dieses Werts nicht dokumentiert ist, wird er für das hier verwendete Fahrzeugmodell auf den Minimalwert von $10^{-4}$ gesetzt.

Durch die Zuweisung des Wheel Collider besitzt das Rad nur einen translatorischen Freiheitsgrad relativ zum Fahrzeugaufbau entlang der lokalen $y$-Achse bzw. Hochachse. Der zulässige Federweg entlang dieses Freiheitsgrads kann durch den Parameter „Suspension Distance“ vorgegeben werden. Weitere Konfigurationen bezüglich des Feder-Dämpfer-Systems können in der Gruppe „Suspension Spring“ vorgenommen werden. Einstellbar sind die Feder- und Dämpferkonstante in den Einheiten $\frac{N}{m}$ bzw. $\frac{Ns}{m}$ und die Ausfederung des unbelasteten Rads relativ zum Gesamtfederweg. Ein Wert von null für „Target Position“ entspricht der vollständigen Ausfederung, während ein Wert von eins einer Ruhelage bei

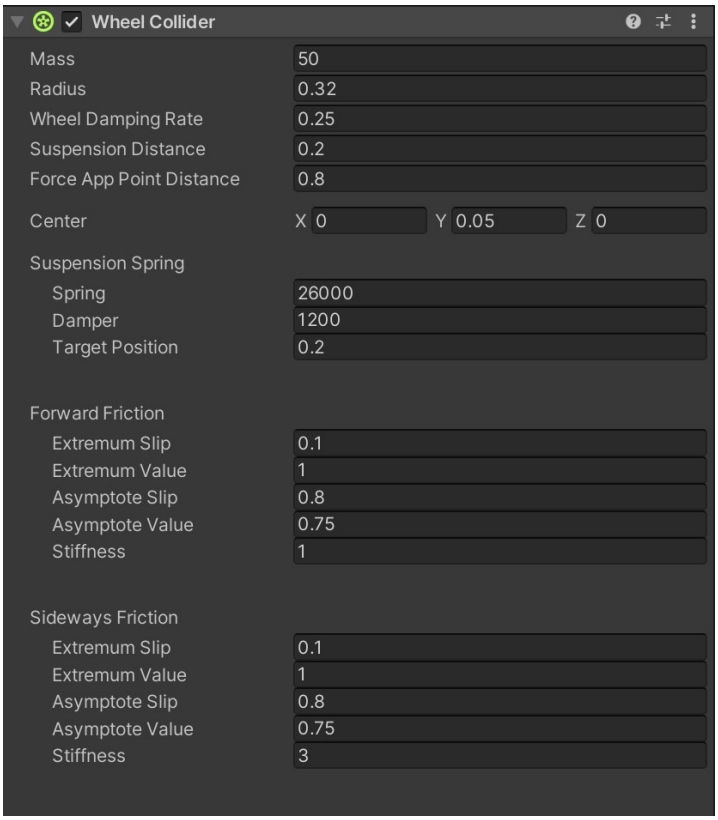


*Bild 5-7*: *Attribute des Wheel Collider-Components der Unity-Engine.*

vollständiger Einfederung entspricht. Die Federkraft ergibt sich durch die Abweichung von dieser Ruhelage.

Der Parameter „Force App Point Distance“ spezifiziert den Krafteinleitungspunkt der Kräfte, die durch das jeweilige Rad hervorgerufen auf den Fahrzeugaufbau wirken. Neben den Feder- und Dämpferkräften gehören hierzu auch die Längs- und Seitenführungskräfte des Rad-Straße-Kontakts. Der Krafteinleitungspunkt liegt stets auf einer Parallelen zur $y$-Achse bzw. Hochachse des Rigidbody, welche die Ruheposition des Rads beinhaltet. Durch den genannten Parameter wird die genaue Lage des Krafteinleitungspunkts auf dieser Geraden bestimmt, indem er den Abstand dieses Punkts zur Ruhelage des Rads in Metern definiert.

Zur Ausrichtung der Collider und der grafischen Meshes der Räder zueinander kann der Parameter „Center“ eingesetzt werden. Da die Meshes und Collider der Räder des hier vorgestellten Fahrzeugmodells in separaten Gameobjects mit eigenen Transforms verwaltet werden, werden $x$-, $y$- und $z$-Koordinate des Radzentrums auf null gesetzt und die Ausrichtung über das Transform vorgenommen.

Es schließen sich zwei weitere Parametergruppen an, welche den Längs- und Querschlupf spezifizieren. Die Prinzipien sind für beide Gruppen dieselben. Es liegt eine nichtlineare Reifenkraftkennlinie zugrunde. Der Extrempunkt und das asymptotische Verhalten dieser Kennlinie kann parametriert werden. Das Bild 5-8 gibt Aufschluss über die genaue Bedeutung der einzelnen Parameter.

Der genaue Verlauf der Kurve ergibt sich durch einen Spline durch die Punkte $(0, 0)$, $(F_{max}, s_{max})$ sowie $(F_{\infty}, s_{\infty})$ und der Forderung, dass die Steigung der Kurve für $s = s_{max}$

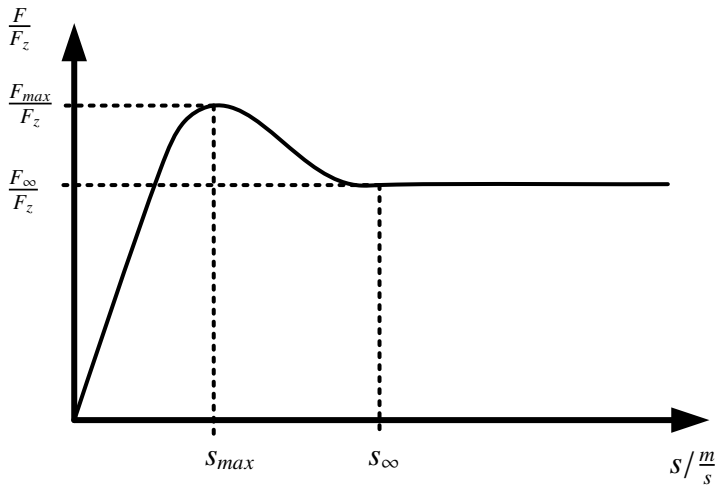


*Bild 5-8*: *Schlupfkurve des Wheel Collider der Unity-Engine.*

und $s > s_\infty$ null ist. Auch wenn hier nicht die klassische Formulierung der Kennlinie über Arcustangens-Funktionen zugrunde liegt, ist der im Wheel Collider verwendete Reifenkraftverlauf dennoch eng an der Magic Formula nach Pacejka angelehnt [PB12]. Die Zuordnung der Parameter des Wheel Collider auf die charakteristischen Größen der Kennlinie aus Bild 5-8 ist wie folgt:

- Extremum Slip: $s_{max}$
- Extremum Value: $\frac{F_{max}}{F_z}$
- Asymptote Slip: $s_\infty$
- Asymptote Value: $\frac{F_\infty}{F_z}$

Die Kraftwerte werden mithilfe der Radaufstandskraft $F_z$ normiert, sodass die Reifenkraftkennlinie unabhängig von dieser beschrieben werden kann. Der Schlupf wird als Relativgeschwindigkeit zwischen Radumfangsgeschwindigkeiten und Radnabengeschwindigkeit in der Einheit $\frac{m}{s}$ angegeben. Die aus dem Schlupf resultierende Kraft wird in der Einheit $N$ angegeben. Nachteilig ist, dass eine direkte Reibwertvorgabe im Modell nicht vorgesehen ist. Allerdings können die Werte $F_{max}$ und $F_\infty$ durch den Parameter „Stiffness“ skaliert werden. Da dieser Parameter zur Laufzeit angepasst werden kann, ist eine Berücksichtigung eines veränderlichen Reibwerts möglich. Im Rahmen dieser Arbeit wird allerdings angenommen, dass der Reibwert konstant ist.

### Antrieb und Bremse

Zur Vervollständigung des Fahrzeugmodells werden weitere Komponenten benötigt, die nicht durch vorgefertigte Components der Unity-Engine abgedeckt werden. Zur Modellierung dieser weiteren Aspekte wird ein eigenes Script-Component „Car Model“ eingesetzt. Da das interne Fahrzeugmodell nicht das Ziel hat, das reale Fahrzeugverhalten detailgetreu abzubilden, werden die weiteren Komponenten stark vereinfacht modelliert.

Eine wesentliche Komponente ist der Antriebsstrang, welcher zur Momentenübertragung vom Motor, über Kupplung, Getriebe und Differential, bis hin zum Rad verantwortlich ist. Die Kupplung und das Getriebe werden vernachlässigt. Der Motor wird über zwei

Parameter beschrieben. Zum einen ist das maximale Motormoment festzulegen und zum anderen ist der Momentenaufbau mit einer PT1-Dynamik versehen, deren Zeitkonstante den zweiten Parameter darstellt. Stellgröße ist die normierte Gaspedalstellung.

Unter der Annahme eines Achsdifferentials wird das Motormoment zu gleichen Teilen auf die Räder der antreibende Achse verteilt. Die Wahl der Antriebsachse ist dem Anwender überlassen. Zudem ist die Simulation eines Allradfahrzeugs möglich. Dann ist die Momentenaufteilung zwischen Vorder- und Hinterachse jedoch statisch vorzugeben. Die sich ergebenden Antriebsmomente für die einzelnen Räder können dem Attribut „motorTorque“ (Einheit *Nm*) des jeweiligen Wheel Collider in jedem Zeitschritt zugewiesen werden.

Auch das Bremsmodell ist im internen Fahrzeugmodell sehr einfach gehalten und weist eine hohe Ähnlichkeit zum Antriebsmodell auf. Die Bremskraft wird zwischen dem linken und rechten Rad gleich verteilt. Die Aufteilung zwischen Vorder- und Hinterachse erfolgt wieder durch eine statische Vorgabe. Genau wie der Motor wird auch das Bremsmoment mit einer Aufbaudynamik versehen. Diese Aufbaudynamik dient zur numerischen Stabilisierung und ist physikalisch begründbar, da sich die Normalkraft zwischen Bremsbelag und Scheibe bei der Betätigung des Bremspedals ebenfalls stetig erhöht. Die Zeitkonstante dieser Aufbaudynamik sollte jedoch deutlich niedriger als die Zeitkonstante der Motordynamik gewählt werden. Außerdem wird das betragsmäßige Bremsmoment nach oben begrenzt. Die Drehrichtung ist der Raddrehung entgegengesetzt. Die Bremsmomente der einzelnen Räder können dem Attribut „brakeTorque“ (Einheit *Nm*) des jeweiligen Wheel Collider in jedem Zeitschritt zugewiesen werden.

### Lenkung

Zur Modellierung der Lenkung sieht der Wheel Collider der Unity-Engine bereits das Attribut „steerAngle“ vor. Dieses Attribut bezeichnet den Verdrehwinkel des Rads um die lokale *y*-Achse bzw. Hochachse in Grad. Im Rahmen der Arbeit wird von einer Vorderachslenkung ausgegangen, wie sie in den meisten PKW üblich ist. Genau wie die Antriebs- und Bremsmomente wird auch der Lenkwinkel mit einer Aufbaudynamik versehen. So werden sprunghafte Änderungen des Lenkwinkels vermieden, welche zu Stabilitätsproblemen führen können. Gleichzeitig verbessert sich die Bedienbarkeit durch Eingabegeräte, die keine kontinuierliche Stellgrößenvorgabe erlauben, wie es beispielsweise für die Tastatur der Fall ist.

Die Lenkwinkel an den Spuren eines zweispurigen Fahrzeugs sind bedingt durch die Achskinematik unterschiedlich. Da das kurveninnere Rad stärker eingeschlagen werden muss, um der Bahnkurve folgen zu können, wird der Querschlupf in Kurvenfahrten durch diese Maßnahme minimiert. Auch in diesem stark vereinfachten internen Modell sollte dieser Effekt berücksichtigt werden, um hohe Schlupf- und damit Kraftwerte an den Rädern zu vermeiden. Im Rahmen des internen Modells wird ausschließlich der Ackermann-Fall betrachtet, welcher nur bei sehr langsamer Fahrt ohne Schlupf exakt gilt. Nach Ackermann werden die Vorderräder gemäß Bild 5-9 so eingeschlagen, dass sich alle Räder um einen gemeinsamen Momentanpol *M* drehen. Es ergeben sich die Lenkwinkel $\delta_o$ für das kurvenäußere und $\delta_i$ für das kurveninnere Rad. Abhängig von der Spurbreite $b_{Fzg}$ und dem Radstand $l_{Fzg}$ stehen diese Lenkwinkel nach [PH11] in Beziehung:

$$\cot \delta_o = \cot \delta_i + \frac{b_{Fzg}}{l_{Fzg}}. \tag{5-1}$$

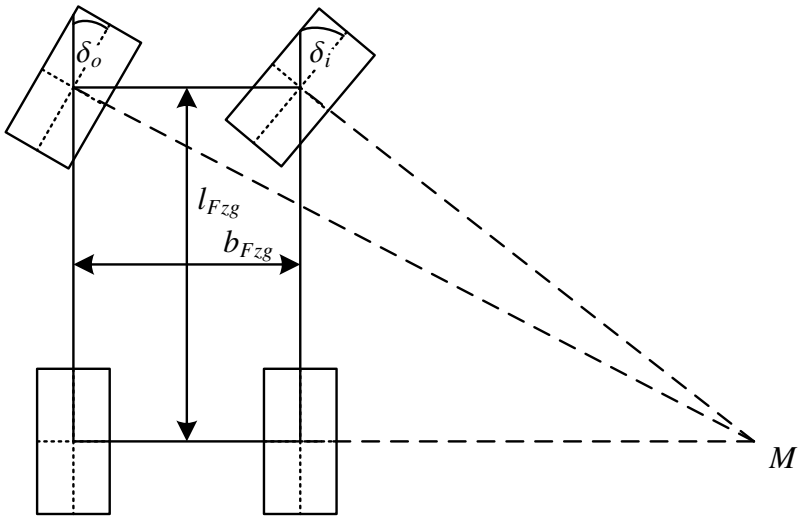


*Bild 5-9*: *Unterschiedliche Lenkwinkel des kurveninneren und kurvenäußeren Rads nach Ackermann.*

Im internen Fahrzeugmodell wird $\delta_i$ durch die Fahrereingabe vorgegeben, während $\delta_o$ gemäß Gleichung (5-1) berechnet wird.

### Fahrzeugparametrierung

Zur vollständigen Definition eines Fahrzeugs bedarf es selbst für das vereinfachte interne Modell einer Vielzahl von Parametern. Um eine komfortable, übersichtliche und reproduzierbare Parametrierung verschiedener Fahrzeuge zu ermöglichen, werden alle fahrzeugrelevanten Parameter in einer zentralen JSON-Datei zusammengefasst. Diese Datei dient zur zentralen Manipulation der Fahrzeugparameter, zur Speicherung und zum Laden der Fahrzeugdaten. Ein Beispiel für eine derartige Parametrierung findet sich im Anhang A1.1.

### Fahrzeugsteuerung

Noch ungeklärt ist, wie die normierten Fahreingaben für Motor, Bremse und Lenkung zustande kommen. Die Berechnung dieser Stellgrößen ist abhängig vom gewählten Eingabemedium. Zur Zeit stehen die Tastatur und eine Lenkrad-Pedal-Kombination als Eingabegeräte zur Auswahl.

Bei Verwendung der Lenkrad-Pedal-Kombination gestaltet sich die Ermittlung besonders einfach, da dieses Eingabegerät bereits kontinuierliche Größen liefert. Diese müssen lediglich durch geeignete Proportionalitätsfaktoren in den Wertebereich $[0, 1]$ (Motor und Bremse) bzw. $[-1, 1]$ (Lenkung) überführt werden.

Eine Tastatur eignet sich wesentlich schlechter als Eingabemedium. Allerdings ist sie an jedem Büroarbeitsplatz verfügbar. Ihre Unterstützung trägt somit maßgeblich zur Erfüllung

der Anforderung **A11** (Konfigurierbarkeit) bei. Nachteil der Tastatur sind die nicht kontinuierlichen bzw. binären Informationen der Tasten. Es können nur die Zustände „Taste gedrückt“ und „Taste nicht gedrückt“ unterschieden werden. Die Zustandsvariable der entsprechenden Taste sei mit $k_{in} \in \{0, 1\}$ bezeichnet, wobei der Wert „1“ den Zustand „Taste gedrückt“ repräsentiert.

Zur Überführung dieses binären Zustands in eine kontinuierlichen Fahreingaben $u_{in}$ werden die Tastenzustände gemäß

$$u_{in} = \max(0, \min(u_{in,alt} + c_{in} \cdot \Delta T \cdot k_{in}, 1)) \tag{5-2}$$

zeitlich diskret integriert. Dabei bezeichnet $u_{in,alt}$ die entsprechende Fahreingabe zum vorhergehenden Zeitpunkt, $k_{in}$ den aktuellen Zustand der Taste, $\Delta T$ die Schrittweite der zeitlichen Diskretisierung und $c_{in}$ die Empfindlichkeit der Fahreingabe auf den Tastendruck. Je höher der Wert von $c_{in} > 0$, desto schneller ändert sich die Fahreingabe $u_{in}$. Die Minimum- und Maximum-Funktionen stellen die Einhaltung des gültigen Wertebereichs für die normierten Fahreingaben sicher. Die Gleichung (5-2) eignet sich unmittelbar für die Motor- und Bremsmomentvorgaben. Für die Lenkvorgaben sind zwei Tasten und eine angepasste Verarbeitung notwendig.

### 5.2.2 ASM Fahrzeugmodell

Im Abschnitt 5.2.1 sind eine Reihe von zum Teil schwerwiegenden Vereinfachungen getroffen worden. Es ist deshalb nicht möglich, die Fahrdynamik realer Fahrzeuge mit dem internen Fahrzeugmodell detailliert nachzubilden. Sollen mit der Simulation Aspekte einer Lichtfunktion untersucht werden, die kritisch durch das Fahrverhalten beeinflusst werden, wird die Verwendung des wesentlich komplexeren ASM Fahrzeugmodells empfohlen. Im Rahmen der Arbeit ist eine vollständige Beschreibung dieses Fahrzeugmodells nicht möglich. Detaillierte Informationen finden sich in [dSP20]. In den nachfolgenden Unterabschnitten soll lediglich eine Übersicht gegeben werden.

#### ASM Toolsuite

Das ASM Fahrzeugmodell ist in eine umfangreiche Toolsuite eingebettet, die zur Simulation von verschiedensten Fahrzeugkomponenten bis hin zur Fahrzeugumgebung eingesetzt werden kann. Die Toolsuite weist detaillierte und variantenreiche Modelle in den Bereichen Antriebsstrang, Fahrdynamik, elektrische Komponenten und Fahrzeugumgebung auf, welche zueinander kompatibel sind und so die zugeschnittene Konfiguration des Gesamtfahrzeugs erlauben.

dSPACE implementiert die Modelle in MATLAB/Simulink, sodass diese in der Developer-Version weitgehend quelloffen sind und eine Manipulation durch den Anwender möglich ist. Diese Quelloffenheit garantiert eine hohe Flexibilität in anwenderspezifischen Vorhaben. Einzelne Modellkomponenten können beispielsweise angepasst oder ausgetauscht werden. Außerdem kann das ASM Modell um geeignete Schnittstellen erweitert werden, die eine Kopplung zusätzlicher anwenderspezifischer Modelle ermöglichen.

Neben diesen tiefen Eingriffen in die Modellarchitektur wird durch das Tool „ModelDesk“ eine intuitive Benutzeroberfläche zur vollständigen Offline-Parametrierung von Fahrzeug und Umgebung bereitgestellt (s. Bild 3-24). Die Vielzahl der einstellbaren Parameter werden in Kategorien übersichtlich organisiert und können durch geeignete Eingabefelder manipuliert werden. Stellenweise werden die Bedeutungen der Parameter durch grafische Schaubilder visualisiert. Die Parametrierung in ModelDesk steht im direkten Zusammenhang mit dem zugrunde liegenden Simulink-Modell. Durch die Übernahme einer Parametrierung werden entsprechende Anpassungen im Modell automatisiert vorgenommen. Hierbei kann es sich um Wertänderungen einzelner Kenngrößen oder sogar um den Wechsel von einem Verbrennungs- auf einen Elektromotor handeln.

Die Verwendung von ModelDesk findet vor der eigentlichen Simulation statt und dient der Offline-Parametrierung. Während der Simulation wird das Controlling durch das Tool „ControlDesk“ ermöglicht. Dessen Benutzeroberfläche wird in Bild 5-10 dargestellt.


*Bild 5-10*: *ControlDesk-Benutzeroberfläche der ASM Toolsuite.*

ControlDesk ermöglicht die Echtzeitvisualisierung des Modellzustands. Es können anwenderspezifische Layouts konfiguriert werden. Hierzu stehen verschiedene Elemente, wie numerische Ausgaben, Ladebalken, Zeitplots und Ähnliches zur Verfügung. Das Tachoinstrument und die darunter befindlichen Plots in Bild 5-10 sind ein Beispiel für ein derartiges Layout. Zur Verknüpfung der Layout-Elemente mit den Modellgrößen kann durch die gesamte Modellhierarchie navigiert werden. Einzelne Parameter und Signale des Simulink-Modells können den Anzeigeelementen zugewiesen werden. Neben der Anzeige aktueller Modellgrößen wird in ControlDesk die Interaktion mit dem Modell ermöglicht. Die Modellparameter können zur Laufzeit durch Eingabefelder im Layout manipuliert werden. Neben der Visualisierung des Modellzustands und dem Eingriff in das Modell stellt die Aufzeichnung von Signalverläufen eine dritte wichtige Funktionalität von ControlDesk dar. Während der Simulation können Messungen aufgenommen werden, in denen beliebige Signale zeitgleich und synchronisiert über einen gewählten Erfassungszeitraum aufgezeichnet werden. Die entstehenden Aufnahmen sind kompatibel zu MATLAB/Simulink

und können dort auf verschiedene Weisen visualisiert und weiterverarbeitet werden. Auf diese Weise lassen sich die Wirkzusammenhänge einzelner Größen im Modell detailliert analysieren.

Die Visualisierung der Fahrsimulation erfolgt innerhalb der ASM Toolsuite durch MotionDesk. Da MotionDesk den hier gestellten Anforderungen einer Nachtfahrtsimulation nicht genügt (s. Tabelle 3-4), wird auf dieses Tool im Rahmen der vorliegenden Arbeit nicht zurückgegriffen.

### Schnittstelle zu Hyperion

Wie erwähnt, handelt es sich bei dem ASM Fahrzeugmodell um ein quelloffenes Modell, dessen Implementierung in MATLAB/Simulink vorgenommen wird. Durch diese Eigenschaft ist die Herstellung einer Schnittstelle möglich, welche die Einbindung des ASM Fahrzeugmodells in Hyperion ermöglicht. Die meisten Signale des ASM Modells sind in zentralen Bussen organisiert. Aus diesen Signalbussen müssen die außerhalb der Fahrdynamiksimulation relevanten Informationen extrahiert und weitergegeben werden. Das ASM Modell auf Wurzelebene ist grau umstrichelt in Bild 5-11 dargestellt. Es wird in die Subsysteme „ASM Fahrzeugmodell“ und „Benutzerschnittstelle des ASM Modells“ untergliedert. Letzteres ist für die Ankopplung anwendungsspezifischer Komponenten besonders geeignet, da es die wesentlichen Signale und Parameter des Fahrzeugmodells enthält und diese in zugänglicher Form kategorisiert.

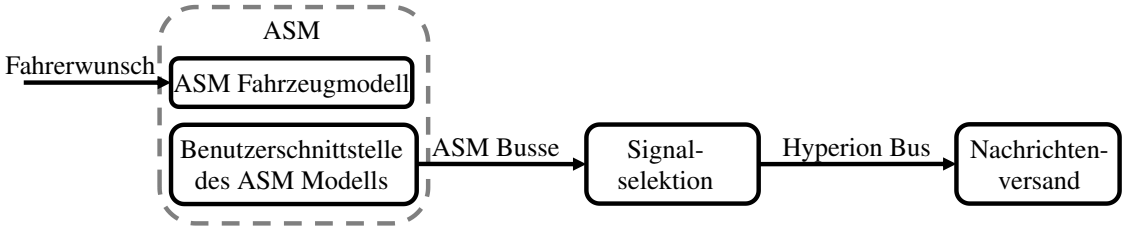


*Bild 5-11*: *Modifiziertes ASM Modell mit Schnittstelle zu Hyperion auf der Wurzelebene.*

Zur Herstellung einer Schnittstelle zu den Umgebungskomponenten der Gesamtarchitektur (s. Bild 5-1) werden die benötigten ASM Busse aus der ASM Benutzerschnittstelle ausgeleitet und dem Subsystem „Signalselektion“ zugeführt. Dieses wählt die relevanten Signale aus den verschiedenen ASM Bussen aus und führt sie zu einem Hyperion Bus zusammen. Die Signale im Hyperion Bus werden schließlich in einem definierten Takt an die visuelle Simulation versendet.

Das Bild 5-12 zeigt die Signalausleitung aus dem ASM Modell detaillierter. Zu sehen ist eine abstrahierte Darstellung der Inhalte des Subsystems „Benutzerschnittstelle des ASM Modells“.

In diesem Subsystem werden die Signale entsprechend der Fahrzeugkomponenten „Steuergerät“, „Motor“, „Antriebsstrang“, „Fahrdynamik“ sowie der „Fahrzeugumgebung“ untergliedert. Jede Komponente verwaltet einen zentralen Bus, welcher alle Signale im Kontext dieser Komponente enthält. Alle für die Gesamtsimulation relevanten Signale können aus

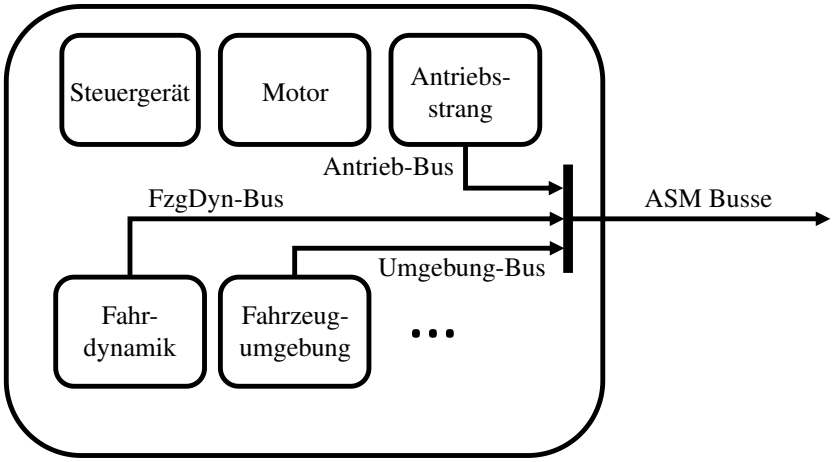


*Bild 5-12*: *Ausleitung der in Hyperion relevanten Signalbusse aus der Benutzerschnittstelle des ASM Modells.*

den Bussen des Antriebsstrangs „Antrieb-Bus“, der Fahrdynamik „FzgDyn-Bus“ und der Fahrzeugumgebung „Umgebung-Bus“ extrahiert werden. Deshalb werden die entsprechenden Busse aus den Fahrzeug- und Umgebungskomponenten ausgeleitet und in einem Sammelbus, welcher in Bild 5-12 mit „ASM Busse“ bezeichnet wird, zusammengeführt.

Die gesammelten ASM Busse werden der Signalselektion zugeführt. Deren Struktur wird in Bild 5-13 gezeigt. Der Sammelbus wird zunächst in die Komponentenbusse aufgeteilt. Aus diesen werden dann die einzelnen Signale ausgewählt und in neue Busse mit den Bezeichnungen „Ego Info“ und „Fellow Info“ überführt. Zusätzlich wird das Signal „Manöverzeit“ durchgereicht. Die neu erzeugten Busse und die Manöverzeit bilden den Hyperion Bus, welcher alle für die Restsimulation relevanten Signale in einem Bus zusammenfasst.

Der neu erzeugte Bus „Ego Info“ enthält alle Informationen, die im Kontext des Egofahrzeugs von Bedeutung sind. Zur Positionierung in der virtuellen Umgebung werden die Position und Rotation des Egofahrzeugs bezüglich des Inertialkoordinatensystems benötigt. Bei diesen Signalen handelt es sich jeweils um dreidimensionale Vektoren, wobei jedes Element vom Datentyp „double“ ist und somit durch acht Byte codiert wird.

Eine weitere logische Gruppe bilden die lichtrelevanten Parameter. Diese Gruppe setzt sich am Beispiel des HD84-Systems aus der Gierrate des Fahrzeugaufbaus, den Radumfangsgeschwindigkeiten des linken und rechten Vorderrads und dem mittleren Radeinschlagwinkel beider Vorderräder zusammen. Mit diesen und weiteren Größen, die in der visuellen Simulation erzeugt werden, ermittelt das Steuergerät eine geeignete Lichtverteilung.

Darüber hinaus werden Parameter übermittelt, die ausschließlich Einfluss auf die Tonwiedergabe nehmen. Diese Parameter sind die betragsmäßige Geschwindigkeit des Fahrzeugaufbaus, die Motordrehzahl und die Gaspedalstellung. Auch wenn die akustische Simulation im Rahmen dieser Arbeit nicht beschrieben wird, seien diese Größen zur Vollständigkeit erwähnt.

Die drei logischen Gruppen der Fahrzeuglage, der lichtrelevanten Daten und der akustisch relevanten Daten bilden den Ego Info Bus. Darunter wird der Fellow Info Bus gebildet.

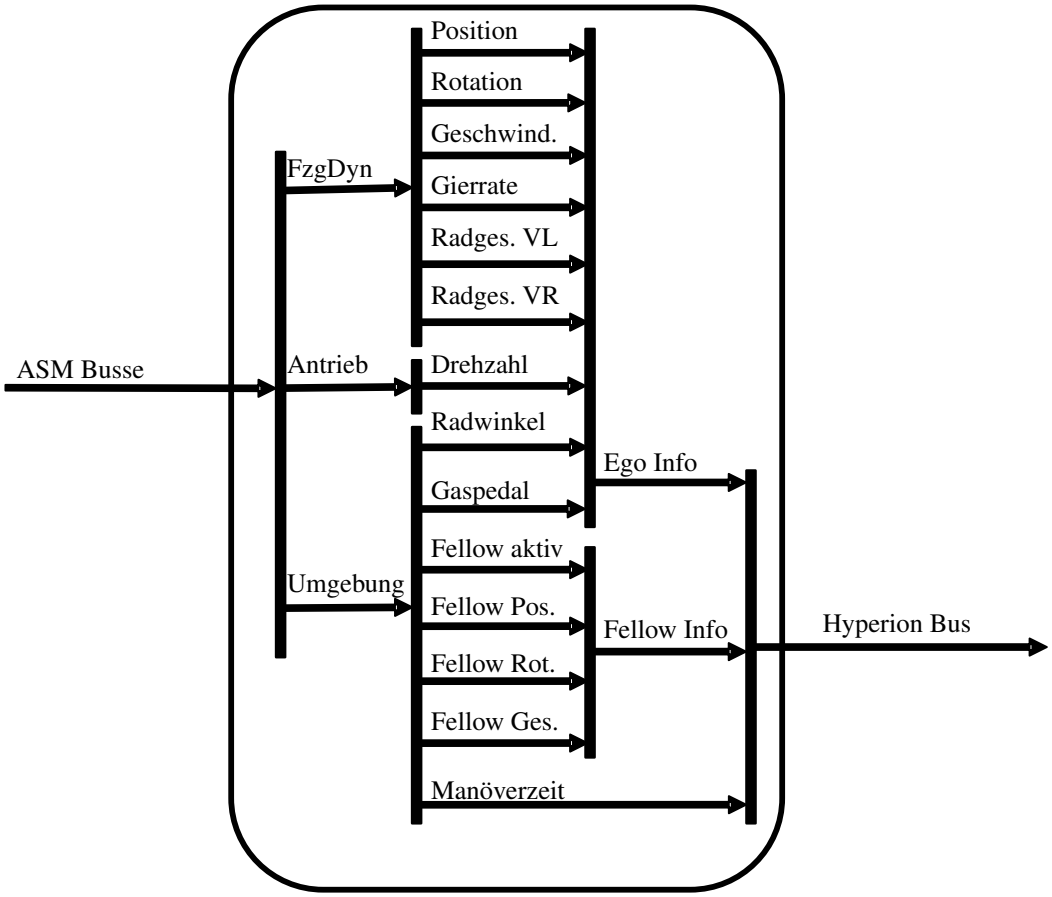


*Bild 5-13*: *Erzeugung des Hyperion-Signalbusses in der Signalselektion.*

Dieser kapselt alle Informationen, die im Kontext weiterer Verkehrsteilnehmer (auch „Fellow“) relevant sind.

Das ASM Fahrzeugmodell ist in der Lage bis zu 15 Fellows zu simulieren. Deshalb wird durch das Signal „Fellow aktiv“ codiert, welche der maximal 15 Verkehrsteilnehmer in der aktuellen Simulation aktiv sind. Das Signal besteht aus einer Folge von 15 Bytes. Das für den jeweiligen Fellow repräsentative Byte trägt den Wert 1, wenn dieser aktiv ist, und den Wert 0 bei Inaktivität. Zur Darstellung der Fellows in der virtuellen Umgebung werden ihre jeweiligen Positionen und Rotationen benötigt. Die Signale „Fellow Pos.“ und „Fellow Rot.“ beinhalten diese Daten. Strukturell handelt es bei diesen Signalen um je 15 dreidimensionale Vektoren. Zur akustischen Simulation werden durch das Signal „Fellow Ges.“ zusätzlich die betragsmäßigen Geschwindigkeiten aller Fellows übermittelt.

Neben den Ego und Fellow Info Bussen wird zudem die Manöverzeit durchgereicht. Dieser Wert beinhaltet die Dauer seit Simulationsstart und kann zur Auslösung zeitgesteuerter Events verwendet werden. Insbesondere bei fremdgesteuerten Fahrmodi, welche im nachfolgenden Abschnitt thematisiert werden, kann auf diese Weise sichergestellt werden, dass sich Online-Parameter stets in der gleichen Fahrsituation ändern. Somit trägt die Übermittlung der Manöverzeit zur Erfüllung der Anforderung **A12** (Reproduzierbarkeit) bei.

Die Ego und Fellow Info Busse sowie die Manöverzeit bilden gemeinsam den Hyperion Bus, dessen Signale zur weiteren Verarbeitung in der visuellen Simulation benötigt werden. Zur Weitergabe wird der Hyperion Bus deshalb dem Nachrichtenversand zugeführt. Dieser Block wird in Bild 5-14 dargestellt.

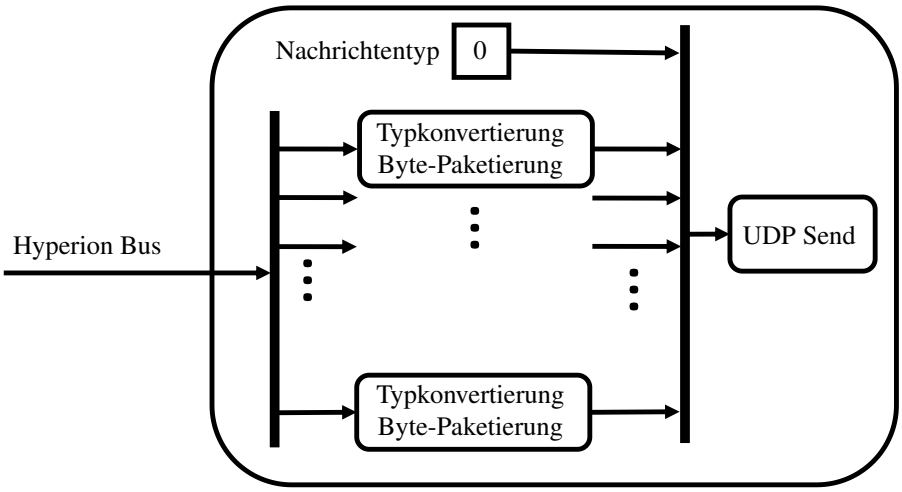


*Bild 5-14*: *Versand des Hyperion-Signalbusses via UDP.*

Da der Hyperion-Server verschiedene Arten von UDP (User Datagram Protocol) Nachrichten entgegen nehmen kann, enthält jede UDP Nachricht im ersten Byte der Nutzdaten einen Typflag. Dieser Typflag codiert eine Nummer, welche die Anzahl und Interpretation der nachfolgenden Bytes festlegt. Für eine ASM Zustandsnachricht, wie sie hier dargestellt wird, trägt der Typflag den Wert 0. Nachfolgend werden sämtliche Signale des Hyperion Busses aneinandergereiht. Hierzu kann der Simulink-Standardblock „Byte Pack“ verwendet werden. Da dieser jedoch nicht mit allen Datentypen umgehen kann, müssen double-Werte zunächst in single-Werte konvertiert werden. Außerdem müssen Ganzzahltypen durch Konvertierung als solche deklariert werden, wenn sie zuvor als Fließkommawerte behandelt wurden.

Die aneinandergereihten Bytefolgen des Nachrichtentyps und aller Signale werden als Nutzdaten in eine UDP Nachricht überführt und an den Hyperion-Server übermittelt, welcher die visuelle Simulation ausführt.

Die Frequenz der Nachrichten kann in Simulink durch einen sogenannten „Rate Transition“ Block bestimmt werden. Dieser muss vor dem „UDP Send“ angeordnet sein und ermöglicht das Überspringen mehrere Zeitschritte des vorgelagerten Modells. Da das ASM Fahrzeugmodell mit einer Schrittweite von 1 $ms$ gerechnet wird, während für die visuelle Simulation gemäß Anforderung **A6** Frequenzen im Bereich von 60 $Hz$ ausreichen, dient der „Rate Transition“ Block zur Vermeidung einer unnötig hohen Netzwerkauslastung.

### 5.2.3 Fremdgesteuerte Fahrmodi

Sowohl beim einfachen internen Fahrzeugmodell, als auch beim komplexen ASM Fahrzeugmodell wurde bislang davon ausgegangen, dass der Anwender bzw. der Proband das

Fahrgeschehen durch seine Eingaben interaktiv beeinflusst. Auch wenn auf diese Weise die maximale Immersion des Fahrers sichergestellt wird, ist seine Konzentration zu einem wesentlichen Teil auf die Fahraufgabe gerichtet. Hinzu kommt, dass die gefahrenen Trajektorien in den verschiedenen Simulationsdurchläufen Unterschiede aufweisen, die letztlich die Vergleichbarkeit der Durchläufe mindern. In der Konsequenz ist es sinnvoll, alternativ zu den selbstgesteuerten Fahrmodi aus den Abschnitten 5.2.1 und 5.2.2 auch fremdgesteuerte Fahrmodi anzubieten, die dem Ingenieur die Lichtbeurteilung in ungestörter Atmosphäre erlauben.

### Interner Autopilot

Die erste und einfachste Variante des fremdgesteuerten Fahrens bildet der interne Autopilot. Er berechnet die Trajektorie des Fahrzeugs ausschließlich anhand des Streckenverlaufs. Dabei berücksichtigt er Sollgeschwindigkeiten und Verzögerungen in Kurven, die als weitere Parameter vorgegeben werden können. Neben der optionalen Steuerung des Egofahrzeugs übernimmt der interne Autopilot auch die Fremdverkehrssimulation, wenn diese nicht durch ASM vorgenommen wird.

Die Verkehrsführung ist nicht eindeutig. Auf mehrspurigen Straßen und Kreuzungen stehen verschiedene Optionen zur Weiterfahrt zur Verfügung. Zur Definition der Solltrajektorie wird deshalb ein XML-basiertes Format eingeführt, dass die Fahrzeugtrajektorie mit dem zugrunde liegenden Straßennetz verknüpft. Im Anhang A1.2 findet sich die ausführliche Dokumentation und eine Beispieldatei gemäß dieses Formats. Das Bild 5-15 zeigt die Struktur eines gültigen XML-Baums zur Manöverdefinition.

Die Trajektorien aller Fahrzeuge für eine gegebene Streckendefinition werden unter dem Begriff „Manöver“ zusammen gefasst. Zur Streckendefinition werden im nachfolgenden Abschnitt detaillierte Informationen folgen. Das Manöver untergliedert sich logisch in Fahrzeuge und Routen. Um die Verkehrsteuerung des ASM Modells und des internen Autopiloten in anderen Simulationskomponenten auf gleiche Weise behandeln zu können, wird das Limit von 16 Fahrzeugen (Ego und max. 15 Fellows) aus ASM auch für den internen Autopiloten übernommen. Die Fahrzeuge haben zwei Attribute. Zum einen tragen sie eine ID. Die ID -1 kennzeichnet das Egofahrzeug, während die Werte 0 bis 14 Fellow-Fahrzeuge repräsentieren. Für jedes Fahrzeug kann außerdem ein Offset definiert werden, an welchem das Fahrzeug relativ zum Nullpunkt der ihm zugewiesenen Route startet. So kann vermieden werden, dass Fahrzeuge mit einer gemeinsamen Route permanent die gleichen Positionen aufweisen.

Die andere Komponente eines Manövers sind Routen, von denen beliebig viele, aber mindestens eine zu definieren sind. Jede Route trägt eine Liste von Fahrzeug-IDs als Attribut. So können die 16 Fahrzeuge den verschiedenen Routen zugewiesen werden. Zudem ist festzulegen, welche Straßen Bestandteil der Route sind. Jede Straße hat vier Attribute. Die ID der Straße muss ihrer ID in der Streckendefinition entsprechen. Weiterhin wird spezifiziert, in welche Richtung und an welcher Bogenlänge beginnend die Straße zu befahren ist.

Optional kann eine Straße mehrspurig sein. In diesem Fall kann durch die Spuren definiert werden, auf welcher Spur die Fahrzeuge, welcher der jeweiligen Route zugewiesen sind, fahren sollen. Dazu hat jede Spur eine ID, die ihrer ID in der Streckendefinition entsprechen

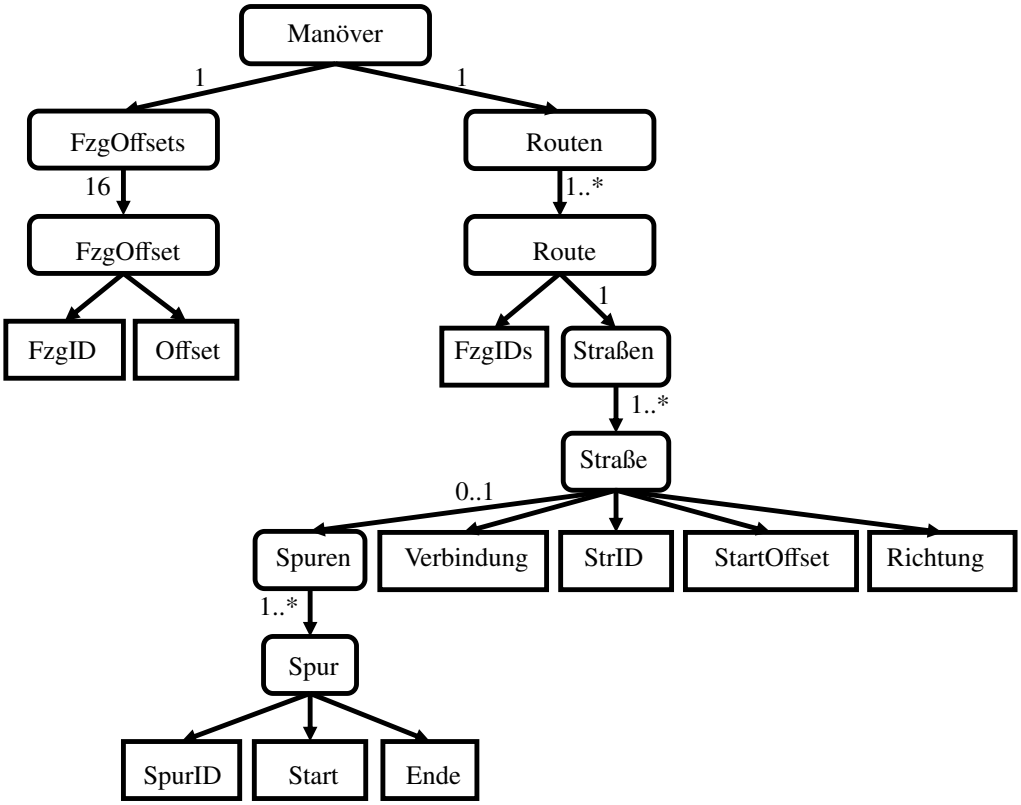


*Bild 5-15*: *Struktur der Manöverdefinition des internen Autopiloten.*

muss. Außerdem ist festzulegen, in welchem Bogenlängenbereich die Spur zu befahren ist. Wird das Attribut „Ende“ der Spur nicht gesetzt, bleibt die Spur bis zum Ende der Straße aktiv.

In einem besonderen Fall können Straßen definiert werden, welche nicht in der Streckendefinition vorhanden sind. Dieser Sonderfall zeichnet sich durch das Setzten des Attributs „Verbindung“ auf den Boolschen Wert *true* aus. In diesem Fall wird eine Verbindungslinie zwischen der vorhergehenden und nachfolgenden Straße interpoliert. Dementsprechend müssen die Attribute „StrID“, „StartOffset“ und „Richtung“ nicht gesetzt werden. Stattdessen ist durch die Attribute „Start“ und „Ende“ zu spezifizieren, an welcher Stelle die vorhergehende Straße verlassen werden soll und an welcher Stelle die Verbindungsstraße in die nachfolgende Straße münden soll. Die Attribute beziehen sich auf die Bogenlängenparameter der jeweiligen Straße. Ein Anwendungsbeispiel für diesen Sonderfall stellen Kreuzungen dar. Das Fahrzeug folgt beim Überfahren der Kreuzung einer Verbindungsstraße, um sich von der einmündenden auf die ausgehende Straße zu bewegen.

Es treten Situationen auf, in denen die zu fahrende Bahnkurve nicht eindeutig aus der Streckendefinition abgeleitet werden kann. Neben Kreuzungen stellen Spurwechsel ein weiteres Beispiel dar. Im Rahmen dieser Arbeit werden solche Manöversegmente als dynamisch bezeichnet. Um geeignete Trajektorien für dynamische Abschnitte zu generieren, werden im Rahmen von Hyperion sogenannte NURBS-Kurven (nicht-uniforme rationale B-Splines) eingesetzt [Ger11]. NURBS sind eine Verallgemeinerung nicht-rationaler B-Splines und Bézier-Kurven. Sie sind in der Lage einen beliebigen stetigen Linienzug

abzubilden. Anwendung finden sie vor allem im Bereich der computergenerierten Bildgebung (CGI) und der computergestützten Konstruktion (CAD). Ihre Stützpunkte treffen NURBS im Allgemeinen nicht exakt, sondern nähern sie nur an. Diese Eigenschaft kann zu Problemen in der Stetigkeit bzw. der stetigen Differenzierbarkeit an den Übergängen zwischen statischen und dynamischen Manöversegmenten führen. Um diese Probleme zu umgehen, werden mehrere Stützpunkte in den Übergangsbereichen für zwei aufeinander folgende Segmente gleich gewählt. Die notwendige Anzahl der identischen Stützpunkte korreliert mit der Ordnung der NURBS. Man kann zeigen, dass abhängig von der Ordnung stets nur eine Teilmenge der Stützpunkte lokal Einfluss nimmt [Sel07]. Diese Eigenschaft der NURBS erlaubt die Sicherstellung der stetig differenzierbaren Umschaltung zwischen den Segmenten.

Die Streckendefinition legt gemeinsam mit der Manöverspezifikation und dem beschriebenen Verhalten in dynamischen Manöversegmenten fest, entlang welcher Bahnkurve sich ein Fahrzeug fortbewegt. Noch nicht definiert ist die zeitliche Komponente und somit die Geschwindigkeit, mit der das Fahrzeug die Bahnkurve abfährt. Die Geschwindigkeitsvorgabe wird im internen Autopiloten sehr einfach realisiert und greift nicht auf das interne Fahrzeugmodell zurück.

Zunächst kann der Anwender eine Sollgeschwindigkeit $v_{soll}$ vorgeben, mit der das Fahrzeug fährt, wenn die Gegebenheiten es erlauben. Gründe für eine Reduzierung der Geschwindigkeit stellen enge Kurven, Abbiegevorgänge oder langsamer vorausfahrende Fahrzeuge dar.

Die beiden erstgenannte Fälle werden berücksichtigt, indem der Autopilot ähnlich einem menschlichen Fahrer einen Blick auf den vor ihm liegenden Streckenverlauf wirft. Dazu wird zunächst ein Bogenlängenparameter $s_f$ berechnet, der auf der bezüglich $s$ parametrisierten Kurve $c_{Tr}(s)$ vor dem Fahrzeug liegt, welches sich derzeit auf der Position $c_{Tr}(s_p)$ befindet. Das Bild 5-16 veranschaulicht diesen Zusammenhang.

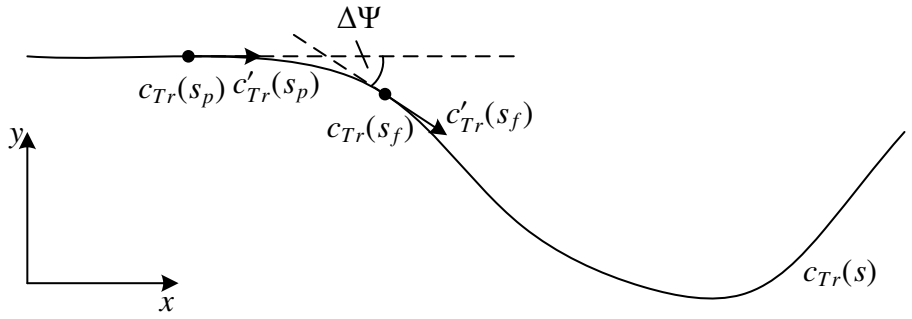


*Bild 5-16*: *Geschwindigkeitsanpassung des internen Autopiloten in Kurven oder vor Abbiegevorgängen.*

Die Lage des Vorausschaupunkts $c_{Tr}(s_p)$ kann über die Zeitlücke $\Delta T_{gap}$ zur momentanen ($t_p$) Position parametriert werden. Unter der Annahme einer konstanten Fahrzeuggeschwindigkeit im Zeitintervall $[t_p, t_p + \Delta T_{gap}]$ ergibt sich der Bogenlängenparameter $s_f = s_p + v_{ist} \cdot \Delta T_{gap}$ des Vorausschaupunkts. Nun werden die Tangentenvektoren der Bahnkurve $c_{Tr}(s)$ für die Parameter $s_p$ und $s_f$ hinsichtlich ihrer Richtung verglichen. Konkret wird die Winkeldifferenz $\Delta\Psi$ beider Vektoren bezüglich der Hochachse ermittelt.

Sie entspricht gleichzeitig der Drehung des Fahrzeugs im Zeitintervall $\Delta T_{gap}$ und stellt auf diese Weise ein Maß für die Schärfe einer eventuell vorausliegenden Kurve dar. Auf einer Geraden ist die Winkeldifferenz $\Delta\Psi = 0°$. Im Fall einer unmittelbar bevorstehenden 90°-Kurve kann $\Delta\Psi$ abhängig vom Parameter $\Delta T_{gap}$ und der zum Zeitpunkt $t_p$ vorliegenden Fahrzeuggeschwindigkeit $v_{ist}$ zwischen 0° und 90° betragen. Die Wahl von $\Delta T_{gap}$ sollte also einerseits so groß sein, dass voraus liegende Kurven berücksichtigt werden können. Andererseits darf $\Delta T_{gap}$ nicht so groß sein, dass eine bevorstehende S-Kurve vollständig übergangen wird und so fälschlicherweise keine Geschwindigkeitsanpassung erfolgt. Versuche haben gezeigt, dass sich ein Wert von 3 $s$ für den Großteil aller Streckenverläufe gut eignet.

Der Zusammenhang zwischen der Winkeldifferenz $\Delta\Psi$ und der angepassten Sollgeschwindigkeit $v_{soll,dyn}$ des Fahrzeugs lautet

$$v_{soll,dyn} = \max(v_{soll} - \frac{dv}{d\Psi} \cdot \Delta\Psi, v_{min}),$$

wobei das Bremsverhalten $\frac{dv}{d\Psi}$ (Standard-Wert: $0,2\,\frac{m/s}{°}$) des Fahrers beim Auftritt einer Kurve und die Minimalgeschwindigkeit $v_{min}$ (Standard-Wert: $15\,\frac{km}{h}$) weitere Auslegungsparameter darstellen. Die tatsächliche Geschwindigkeit des Fahrzeugs folgt der Sollgeschwindigkeit mit einer PT1-Dynamik entsprechend

$$v_{ist} = T^\star \cdot (v_{soll,dyn} - v_{ist,alt}) + v_{ist,alt}$$
$$\text{wobei } T^\star = \frac{1}{\frac{T_v}{\delta T} + 1}.$$

Die Zeitkonstante $T_v$ dieser PT1-Dynamik stellt den letzten Auslegungsparameter des Autopiloten dar. Ihr Standardwert in Hyperion liegt bei 3 $s$. Auch wenn das Geschwindigkeitsverhalten des Autopiloten einheitlich dargestellt wird, können die genannten Parameter fahrzeugspezifisch vorgegeben werden.

### ASM Autopilot

Im Kontext der Fahrzeugmodelle wurde bereits die Einbindung des ASM Fahrzeugmodells in Hyperion diskutiert. Die ASM Toolsuite bietet zudem die Möglichkeit der automatischen Steuerung des Ego- und der Fremdfahrzeuge. Hierzu stellt das Tool ModelDesk eine benutzerfreundliche Oberfläche zur Festlegung der Fahrzeugtrajektorien bereit, von welcher in Bild 5-17 ein Screenshot zu sehen ist.

Das Vorgehen bei der Manöverdefinition in ModelDesk ist in [dSP20] genau beschrieben und soll hier nicht detaillierter ausgeführt werden. Prinzipiell gleicht die zugrunde liegende Logik dem im vorhergehenden Abschnitt vorgestellten Autopiloten.

### GPS Autopilot

Die hinreichende Übereinstimmung der virtuellen Umgebung und der realen Erlebnisse ist die zentrale Anforderung jeder Fahrsimulation. Nur wenn diese Übereinstimmung gegeben

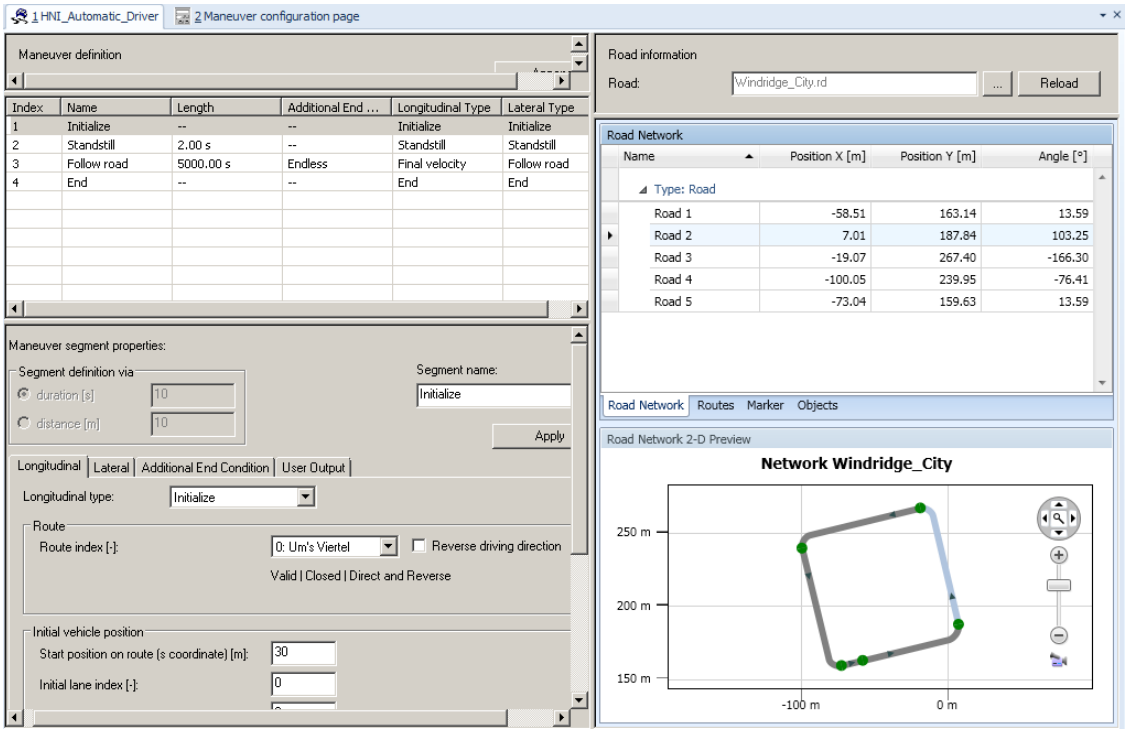


*Bild 5-17*: *Manöverdefinition in ModelDesk aus der ASM Toolsuite.*

ist, lassen sich zeitintensive, kostspielige und zum Teil gefährliche reale Tests im Entwicklungsprozess durch simulative Erprobungen ersetzen. Zur Validierung einer Simulation muss die hinreichende Übereinstimmung mit der Realität nachgewiesen werden. Diese Vergleichbarkeit ist nur dann gegeben, wenn virtuelle Abbildungen realer Umgebungen in der Simulation zur Verfügung stehen. Vollständig wird die Vergleichbarkeit dann, wenn die Fahrzeugbewegungen der Realität durch die Simulation wiedergegeben werden.

Hyperion ermöglicht diese Gegenüberstellung von Realität und Simulation, indem reale Fahrzeugtrajektorien, welche beispielsweise durch ein Messfahrzeug bezüglich GPS-Koordinaten und weiteren Fahrzeugdaten aufgezeichnet wurden, in der Simulation geladen werden können. Steht für die befahrene Strecke ein virtuelles Abbild zur Verfügung (s. Abschnitt 5.3), kann die geladene Trajektorie nachgefahren werden.

Ausgangspunkt für den Import der realen Fahrzeugtrajektorie sind die Messdaten. Diese müssen zur Festlegung der Fahrzeuglage den Längen- und Breitengrad umfassen. Da eine Höhenmessung in der Regel nicht mit hinreichender Genauigkeit möglich ist, wird diese abhängig von der horizontalen Lage durch das Streckenprofil ermittelt. Ist die korrekte Abbildung der Hubbewegung des Fahrzeugaufbaus erwünscht, muss in diesem Fahrmodus deshalb eine dedizierte Messung dieser Größe erfolgen. Zur Festlegung der Fahrzeugorientierung sind außerdem Roll-, Nick- und Gierwinkel aufzuzeichnen. Bei Verwendung des GPS Autopiloten ist kein Fahrzeugmodell aktiv. Dementsprechend müssen auch Fahrzeuggrößen, welche relevant für das Scheinwerfersteuergerät sind, aufgezeichnet werden. Alle genannten Daten werden über die Fahrtdauer mit einer vorgegebenen Frequenz abgetastet und liegen als Zeitreihen vor. Schließlich muss die Aufzeichnung über einen Verweis auf die korrespondierende virtuelle Strecke verfügen.

Im nächsten Schritt müssen die Längen- und Breitengrade in die kartesischen Koordinaten des Weltkoordinatensystems der Unity-Szene überführt werden. Dazu werden die Werte zuerst mittels Mercator-Projektion aus dem kugelförmigen WGS84-System in ein planares System überführt. Zur Eliminierung des Messrauschens werden die so gewonnenen X- und Y-Koordinaten und alle weiteren Messdaten durch eine gleitende Mittelwertbildung gefiltert. Die Phasenfreiheit der Filterung wird dabei sichergestellt, indem ein gegebenes Element mit gleich vielen zeitlichen Vorgängern und Nachfolgern ermittelt wird. Die Fensterbreite des Filters kann für jeden Messwerttyp parametriert werden. Bei einer Messfrequenz von 50 Hz haben sich für die planaren Koordinaten eine Fensterbreite von etwa 25 Zeitschritten und für die Orientierungen eine Fensterbreite von etwa 10 Zeitschritten als geeignet herausgestellt.

Im Rahmen dieser Arbeit wurde der GPS Autopilot anhand einer Messfahrt in der Nähe von Paderborn erprobt. Der Rundkurs durch Borchen und Etteln mit innerörtlichen Abschnitten, Landstraßen und einem Autobahnabschnitt eignet sich aufgrund seiner Diversität gut für den Eignungsnachweis. Die aufgezeichnete Fahrzeugtrajektorie wird auf der Kartenansicht in Bild 5-18 als blaue Linie überlagert.

Die auf diesem Weg generierte Fahrzeugtrajektorie wird als Zeitreihe im Verzeichnis der zugehörigen virtuellen Strecke abgelegt und kann im Rahmen einer Simulation durch das Egofahrzeug vollständig reproduzierbar nachgefahren werden.

## 5.3 Streckengenerierung

Innerhalb von Hyperion können virtuelle Strecken auf zwei unterschiedliche Arten erzeugt werden. Jede Ausprägung verfügt über individuelle Vor- und Nachteile. Die beiden Möglichkeiten werden in den nachfolgenden Unterabschnitten näher beleuchtet.

### 5.3.1 Import von OpenDRIVE-Strecken

Die schnellste und einfachste Möglichkeit zur Generierung virtueller Strecken stellt der Import von Streckendefinitionen nach dem OpenDRIVE-Standard dar. OpenDRIVE ist ein offenes XML-basiertes Format zur logischen Beschreibung von Straßennetzwerken. Das Format zielt auf den Einsatz in der Fahrsimulation ab und erlaubt daher eine präzise geometrische Beschreibung des Streckenverlaufs durch stückweise definierte mathematische Funktionen. Es unterstützt Höhen- und Querneigungsprofile für Straßen und erlaubt vielfältige Spurkonstellationen. Entwickelt wird das offene Format durch ein Expertengremium aus dem Bereich der Fahrsimulation, welches sich sowohl aus Personen öffentlicher Einrichtungen als auch der Industrie zusammensetzt. Weiterführende Informationen können der Spezifikation und der Website des Standards entnommen werden [DHB19],[VIR20].

Hyperion stellt eine Funktionalität bereit, welche die textuelle Streckendefinition gemäß des Standards ohne manuelles Einwirken in eine virtuelle Szene überführt. Die im Standard definierte Straße wird dabei geometrisch sehr genau wiedergegeben. Ebenso werden die Spuren auf den Teilabschnitten korrekt dargestellt. Übergänge zwischen verschiedenen Teilabschnitten, wie Kreuzungen, Wechsel der Spurzahlen und ähnliches sind im Rahmen der automatischen Generierung aufgrund der Vielzahl ihrer möglichen Ausprägung nur

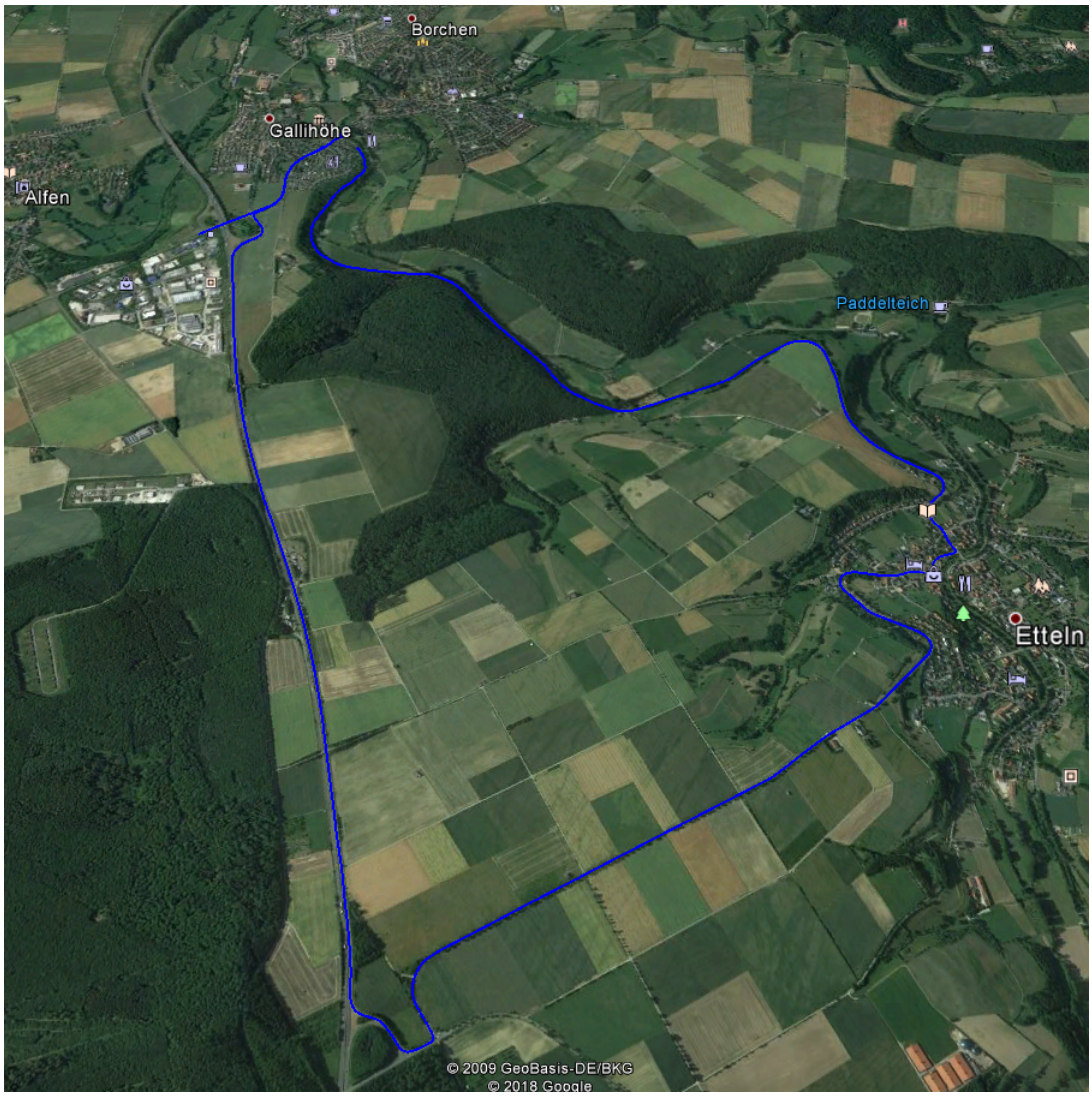


*Bild 5-18*: *Aufzeichnung einer Realfahrt bei Paderborn zur späteren Nachverfolgung der Fahrzeugtrajektorie innerhalb der Simulation.*

unter sehr hohem Aufwand umsetzbar. Da die Streckengenerierung nicht den Forschungsfokus dieser Arbeit adressiert, werden derartige Elemente durch Polygone mit Asphalt-Textur, jedoch ohne Straßenmarkierungen oder ähnliches, dargestellt.

Zur Generierung der virtuellen Strecke werden auf Basis der geometrischen Primitive des Standards (Linien, Kreisbögen, Klothoide und Polygone) Stützstellen entlang des gesamten Streckenverlaufs berechnet. Diese verfügen über eine räumliche Position und Orientierung. Auf diese Weise kann die Lage, Richtung, Steigung und Querneigung der Straße an jeder Stützstelle exakt wiedergegeben werden. Die Abstände der Stützstellen sind parametrierbar. Mit dem Paket „SuperSplines“ aus dem Unity Asset Store können Listen von Stützstellen durch Splines approximiert werden [Evo13]. Hierbei können verschiedene Splinetypen verwendet werden. In Hyperion fällt die Wahl auf B-Spline-Kurven [Boo01]. Hierbei handelt es sich um stückweise definierte Polynomfunktionen unter Verwendung bestimmter Basisfunktionen (Basis-Spline). Durch das Aneinanderlegen von Mesh-Primitiven entlang des Splines erhält die Straße ihre dreidimensionale Ausdehnung. Auf Geraden sind die

Primitive flache Quader, die in Kurven und bei einem Wechsel der Spurzahlen entsprechend verzerrt werden. Die Anpassung der Breite innerhalb eines Splines, wie er bei dem Wechsel der Spuranzahl notwendig wird, ist nicht Bestandteil von SuperSplines, weshalb dieses Paket um weitere Funktionalitäten ergänzt wurde. Genauso ist in SuperSplines die Verwendung der gleichen Textur für alle Primitive vorgesehen. Die unterschiedlichen Straßenmarkierungen, die unter anderem von der Anzahl der Fahrspuren und möglichen Fahrrichtungen abhängen, haben ebenfalls eine Erweiterung des Pakets erfordert. Bild 5-19 zeigt das Ergebnis der automatischen Straßengenerierung auf Basis der einfachen OpenDRIVE-Definition aus Anhang A1.3.

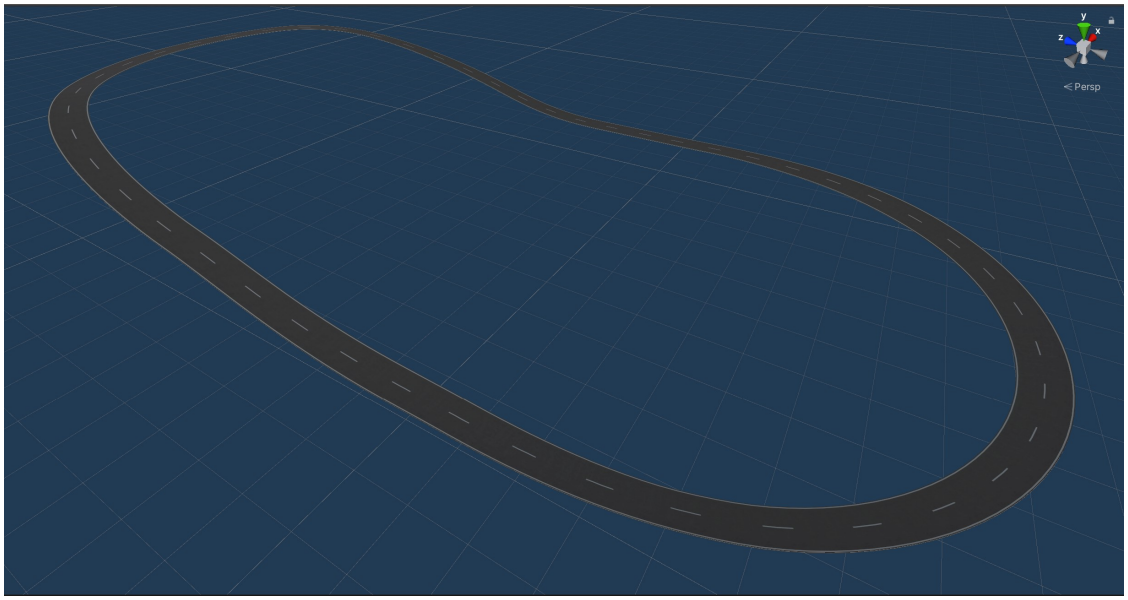


*Bild 5-19*: *Automatisch generierter Straßenverlauf durch Import einer OpenDRIVE-Definition.*

Während der OpenDRIVE-Standard den Straßenverlauf bezüglich aller Freiheitsgrade exakt spezifiziert, enthält er keine Informationen über das Höhenprofil des umliegenden Geländes. Die Existenz eines Untergrunds im Bereich der Straße ist für die realitätsnahe Abbildung des Scheinwerferlichts jedoch unabdingbar. Um diese Informationslücke zu schließen, wird das Höhenprofil im Bereich der Strecke durch Interpolation generiert. Die äquidistanten Stützpunkte der Straße dienen auch hierfür als Grundlage. In Bild 5-20 ist ein beispielhafter Stützpunkt $s$ mit seiner Position $x_s, y_s$ und $z_s$ und Orientierung, welche sich durch die Winkel $\alpha_s, \beta_s$ und $\gamma_s$ der Straße an diesem Punkt ergibt, dargestellt. Seine Orientierung kann gemäß

$$\frac{dy_s}{dx_s} = \cos(-\beta_s) \cdot \tan\gamma_s + \sin(-\beta_s) \cdot \tan\alpha_s \quad \text{und}$$
$$\frac{dy_s}{dz_s} = \sin(-\beta_s) \cdot \tan\gamma_s - \cos(-\beta_s) \cdot \tan\alpha_s$$

in die Gradienten $\frac{dy_s}{dx_s}$ und $\frac{dy_s}{dz_s}$ bezüglich der horizontalen Achsen überführt werden. Mit diesen Informationen kann eine Ebene definiert werden, welche das ideale Höhenprofil des Untergrunds im lokalen Bereich des jeweiligen Stützpunkts darstellt. Sie wird in Bild 5-20 für den Stützpunkt $s$ als Straßenstück visualisiert.

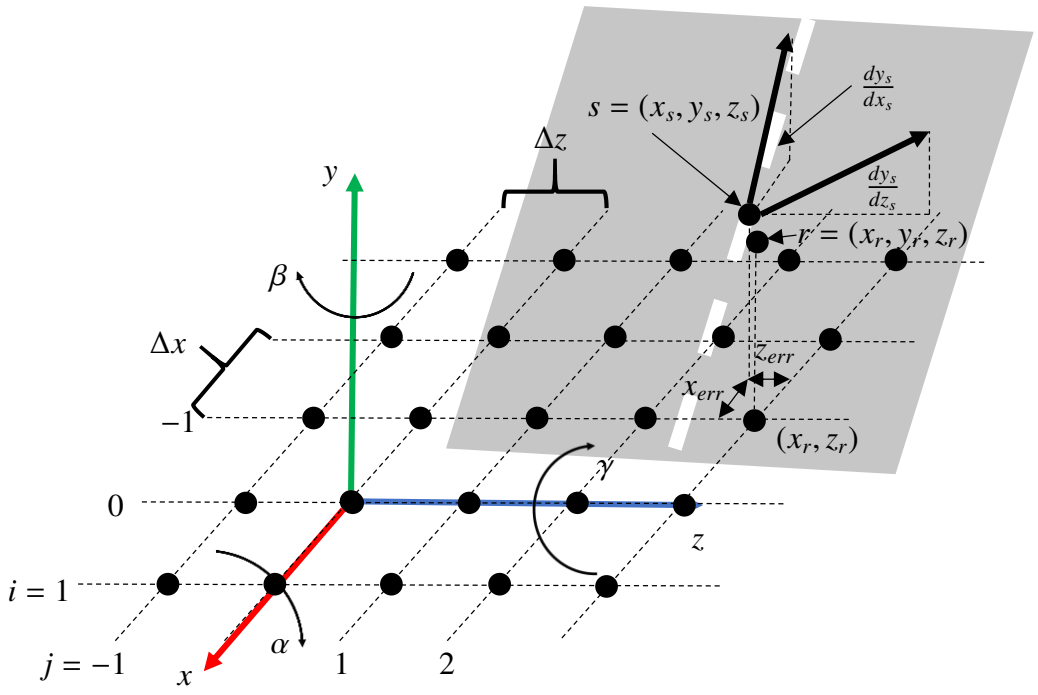


*Bild 5-20*: *Skizze zur Interpolation des Geländehöhenprofils auf der Basis einzelner Stützpunkte mit Lage- und Orientierungsinformationen.*

Das Höhenprofil des Geländes wird durch eine diskretisierte Fläche abgebildet. Dazu werden entlang der horizontalen Raumrichtungen in einer Schrittweite von $\Delta x$ bzw. $\Delta z$ Höhenwerte vorgegeben. Setzt man den Nullpunkt für beide horizontalen Richtungen an eine beliebige Stelle, so kann jeder Punkt des Rasters in der horizontalen Ebene alternativ zu den kontinuierlichen Koordinaten $x$ und $z$ mit den Indizes $i$ und $j$ ausgedrückt werden, indem sein Versatz zum Nullpunkt in $x$- bzw. $z$-Richtung durch die jeweilige Schrittweite $\Delta x$ bzw. $\Delta z$ geteilt wird. Nachfolgend sei der Nullpunkt so gesetzt, dass er auf einer Ecke des Geländes liegt und nur in positive $x$- und $z$-Richtung weitere Stützpunkte existieren. Die Fläche wird diskretisiert, indem ihre Höhe nur für ganzzahlige Paarungen von $i$ und $j$ festgehalten wird, während Zwischenräume durch die vier umliegenden Rasterpunkte interpoliert werden. Insgesamt kann das Höhenprofil der diskretisierten Fläche somit als ein zweidimensionales Array $H(i, j)$ mit $i \in 0, i_{max}$ und $j \in 0, j_{max}$ aufgefasst werden, wobei der Eintrag $h_{ij} \in \mathbb{R}$ die Höhe des Geländes an den diskreten Koordinaten $i$ und $j$ bzw. den kontinuierlichen Koordinaten $x_i = \Delta x \cdot i$ und $z_j = \Delta z \cdot j$ beschreibt.

Im nächsten Schritt werden die Stützpunkte des Streckenverlaufs auf die nächstgelegenen Rasterpunkte des Höhenprofils übertragen. Der Stützpunkt $s$ weist zum nächstgelegenen Rasterpunkt $r$ einen Offset von $x_{err}$ in $x$-Richtung und $z_{err}$ in $z$-Richtung auf. Um die Höhe von $s$ unter Berücksichtigung dieses Offsets korrekt auf $r$ zu übertragen, muss sie entsprechend der Gradienten korrigiert werden. Es ergibt sich die folgende Höhe für den Rasterpunkt $r$:

$$y_r = y_s + x_{err} \cdot \Delta x \cdot \frac{dy_s}{dx_s} + z_{err} \cdot \Delta z \cdot \frac{dy_s}{dz_s}.$$

Dabei dienen die Multiplikationen mit $\Delta x$ bzw. $\Delta z$ zur Überführung der Gradienten, welche die Steigung pro Meter wiedergeben, in das Einheitensystem des Rasters. Da der Rasterpunkt seine Daten direkt von einer Stützstelle der Straße erhält, wird dieser Rasterpunkt als Stützpunkt gekennzeichnet. Er erhält über den Zeitraum der Generierung des Höhenprofils zusätzlich die Gradienteninformationen von $s$ normiert auf das Einheitensystem des Rasters:

$$\frac{dy_r}{dx_r} = \Delta x \cdot \frac{dy_s}{dx_s} \text{ und } \frac{dy_r}{dz_r} = \Delta z \cdot \frac{dy_s}{dz_s}.$$

Nachdem dieser Vorgang für alle Stützstellen der Strecke durchgeführt wurde, existiert eine Teilmenge $R_{init}$ aus der Menge aller Rasterpunkte $R$, welche bereits Höheninformationen durch die unmittelbaren Streckenstützstellen erhalten haben. Jeder Punkt $r$ aus $R_{init}$ schlägt nun einen Höhenwert für jeden Rasterpunkt $t$ aus der Differenzmenge $R \setminus R_{init}$ gemäß

$$y_{t,r} = y_r + \frac{dy_r}{dx_r} \cdot (x_t - x_r) + \frac{dy_r}{dz_r} \cdot (z_t - z_r)$$

vor. Für den Rasterpunkt $t$ werden auf diese Weise alle Vorschläge der Punkte aus $R_{init}$ gesammelt. Die finale Höhe $y_t$ wird schließlich aus allen Vorschlägen $y_{t,r}$ mit $r \in R_{init}$ aggregiert, wobei eine abstandsabhängige Gewichtung erfolgt. Konkret ergibt sich der Wert $y_t$ gemäß

$$\begin{aligned} y_t = & \frac{1}{\sum_{r \in R_{init}} w(t,r)} \cdot \sum_{r \in R_{init}} w(t,r) \cdot y_{t,r} \\ & \text{mit } w(t,r) = \frac{1}{|r-t|^3} \\ & \text{und } |r-t| = \sqrt{(x_r - x_t)^2 + (z_r - z_t)^2}, \end{aligned} \tag{5-3}$$

wobei die Gewichtungsfunktion $w$ anpassbar ist. Für die meisten Streckenprofile hat sich die hier gewählte kubische Bewertung des Abstands als geeignet erwiesen. In jedem Fall sollte die Gewichtung mit zunehmendem Abstand von $t$ und $r$ monoton fallen.

Nachdem die Höhe aller Punkte $t \in R \setminus R_{init}$ nach Gleichung (5-3) festgelegt wurde, verfügt das gesamte Höhenraster $R$ über gültige Höhenwerte. Die Geometrie des Geländes ist somit vollständig beschrieben und kann als Polygonnetz mit geeigneten Texturen visualisiert werden. Bild 5-21 zeigt ein Beispiel.

Neben der Straße und dem Geländeprofil stellt die Randbebauung einen dritten wichtigen Aspekt bei der Bewertung des Scheinwerferlichts dar. Durch umliegende Objekte werden vertikale Projektionen der Lichtverteilung sichtbar, wodurch die Hell-Dunkel-Grenze besonders scharf wahrgenommen wird. Aus diesem Grund unterstützt Hyperion das Einfügen von Objekten im Randbereich der Straße. Um den Anwender hierbei bestmöglich zu unterstützen, ist auch dieser Schritt über weite Bereich automatisiert.

Der OpenDRIVE-Standard erlaubt die Definition sogenannter Custom User Tags, welche die Kennzeichnung von Streckenabschnitten mit eigenen Schlüsselwörtern erlauben. In Hyperion wird diese Möglichkeit genutzt, um die Streckenabschnitte mit Meta-Informationen zu ihrer Randbebauung zu versehen. So kann für einen Abschnitt beispielsweise definiert werden, ob und an welcher Seite Bäume, Leitpfosten oder ähnliches existieren. Die Tabelle 5-1 zeigt die Gesamtheit der hierfür spezifizierten Tags.

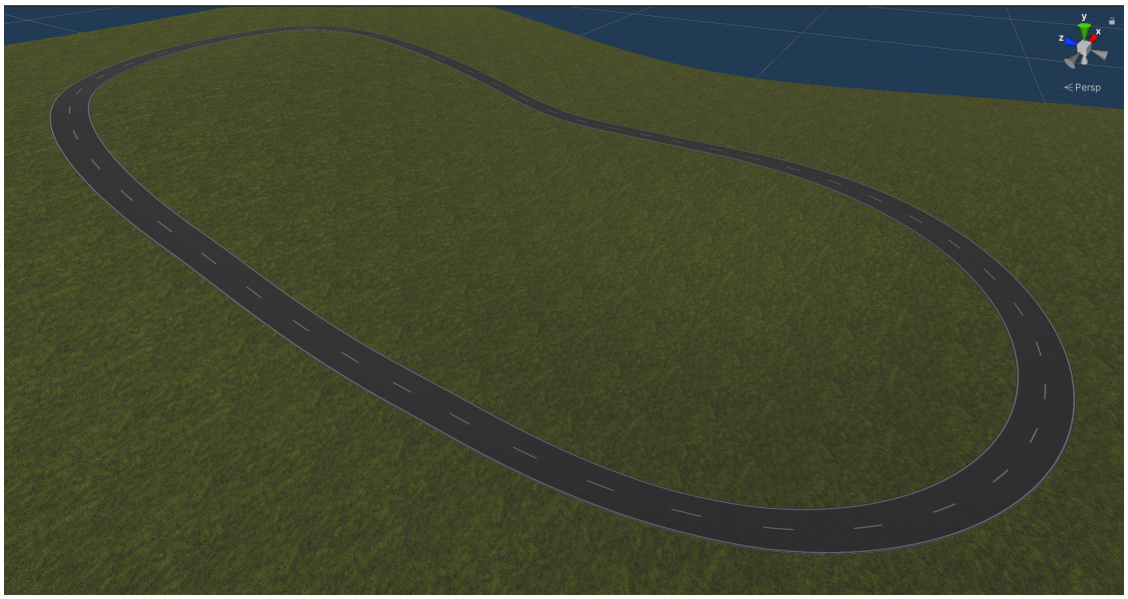


*Bild 5-21*: *Automatisch generierter Straßenverlauf mit interpoliertem Gelände durch Import einer OpenDRIVE-Definition.*

Bei der Generierung werden die vorgesehenen Objekte automatisiert positioniert. Die Software berücksichtigt dabei den geeigneten Abstand zur Straße und bringt, sofern es sinnvoll ist (Bsp. Bäume), zufällige Positionsabweichungen mit ein. In Bild 5-22 sind die Bürgersteige, Straßenlaternen und Leitplanken automatisiert erzeugt worden.

Schließlich muss mit Objekten umgegangen werden, die sich für die automatische Generierung nicht eignen oder nur mit immensem Aufwand in diese integriert werden könnten. So sind Gebäude beispielsweise schwer zu handhaben, da sie durch ihre große Grundfläche eine Einbettung in unebenes Gelände erschweren. Außerdem gibt es anwenderspezifische Wünsche, die nicht umfassend durch die Objektbibliothek von Hyperion bereitgestellt werden können. Abhilfe schafft hier, dass die automatische Szenegenerierung einen Szenegraphen liefert, der nachträglich im Unity-Editor beliebig angepasst werden kann. Das nachträgliche, manuelle Einfügen weiterer Objekte kann also jederzeit erfolgen.

### 5.3.2 Import von OpenStreetMap-Strecken

Die im Abschnitt 5.3.1 beschriebene Variante des Streckenimports stellt den schnellsten und einfachsten Weg zur Generierung frei definierbarer Strecken dar. Dennoch kann es im Rahmen des Virtual Prototyping sinnvoll sein, auf die Freiheiten der Simulation zu verzichten und die Gegebenheiten realer Teststrecken möglichst getreu nachzubilden. Die Nachbildung von Realstrecken ist insbesondere zur Validierung der Simulation von zentraler Bedeutung. Des Weiteren erfordert der in Abschnitt 5.2.3 vorgestellte GPS Autopilot ein virtuelles Abbild der real abgefahrenen Strecke. Motiviert durch diese Überlegungen erlaubt Hyperion den Import von realen Strecken. Aufgrund der Komplexität dieses Vorgangs sind jedoch deutlich mehr manuelle Eingriffe notwendig, als es die OpenDRIVE-Strecken erfordern.

*Tabelle 5-1*: *User Data Tags zur Erweiterung des Informationsgehalts einer OpenDRIVE-Defintion.*

| Tag | Datentyp | Kommentar |
|---|---|---|
| ReflectorPosts | Boolean | Leitpfosten an den Seiten |
| ReflectorPostDistance | Float | Abstand zwischen Leitpfosten |
| Guardrails | Boolean | Leitplanken an den Straßenrändern |
| GuardrailPostDistance | Float | Abstand der Pfosten der Leitplanken |
| StreetLamps | Boolean | Straßenlaternen an den Seiten |
| StreetLampDistance | Float | Abstand der Straßenlaternen |
| SideTrees | {None,Left,Right,Both} | Bäume an den Seiten |
| TreeRoadDistance | Float | Abstand vom Straßenrand zu Bäumen |
| TreeBaseDistance | Float | Mittl. Abstand zwischen Bäumen |
| TreeDistanceVariant | Float | Max. Variation des Abtands $TreeDistanceVariant \leq \frac{TreeBaseDistance}{2}$ |
| Sidewalk | {None,Left,Right,Both} | Bürgersteige an den Seiten |
| SidewalkWidth | Float | Breite des Bürgersteigs |

Als Eingangsdaten für den Import realer Strecken wird der relevante Bereich im OpenStreetMap (OSM) Format benötigt [OSM20a]. Bei OpenStreetMap handelt es sich um ein in 2004 gegründetes internationales Projekt und um einen freien Standard, welcher vornehmlich zur Navigation entwickelt wurde. Der Streckenverlauf wird lediglich durch grob gewählte Stützpunkte und geraden Verbindungslinien zwischen diesen modelliert. Zur Fahrdynamiksimulation ist der OSM Standard zu ungenau und erfordert eine nachgelagerte Aufbereitung der Daten. In Hyperion wird dennoch auf diese Datenquelle zurückgegriffen, da OSM Daten flächendeckend und kostenfrei verfügbar sind. Der relevante Kartenbereich kann mit einem Editor selektiert und extrahiert werden. Es existieren verschiedene Editoren, die den OSM Standard unterstützen. Im Rahmen dieser Arbeit wurde mit dem Java OpenStreetMap Editor (JOSM) gearbeitet [OSM20b]. Ein Screenshot des Editors wird in Bild 5-23 gezeigt. Anschließend sollten nicht relevante Straßen im Selektionsbereich entfernt werden. So kann die zu verarbeitende Datenmenge zu diesem frühen Zeitpunkt bereits auf das Notwendigste beschränkt werden.

Das verbleibende Straßennetz kann mit dem Kommandozeilentool „netconvert“ aus der Verkehrssimulation „Simulation of Urban Mobility“ (SUMO) von OSM in OpenDRIVE konvertiert werden. Im Anschluss kann der Workflow des Imports von OpenDRIVE-Strecken gemäß Abschnitt 5.3.1 durchlaufen werden. Nach der Generierung der virtuellen Strecke gemäß dieses Verfahrens ist das Ergebnis hinsichtlich zweierlei Kriterien zu prüfen. Einerseits muss die Anzahl der Fahrspuren auf den relevanten Straßenbereich mit den den real vorhandenen verglichen werden. Zumindest zum aktuellen Zeitpunkt befindet sich netconvert noch in der Entwicklung. Außerdem sind die zugrunde liegenden OSM Spezifikation nicht immer vollständig und korrekt. Sollten hier Unterschiede zwischen der realen Umgebung und der virtuellen Szene vorhanden sein, müssen die entsprechenden Tags im OpenDRIVE-Dokument nachträglich angepasst bzw. ergänzt werden. Soll der GPS Autopilot verwendet werden, muss andererseits überprüft werden, ob dessen Trajektorie mit dem generierten Straßenverlauf übereinstimmt. Ist das nicht der Fall, kann die Trajektorie im OSM Editor eingeblendet werden. Der Editor erlaubt die Verschiebung der vorhandenen

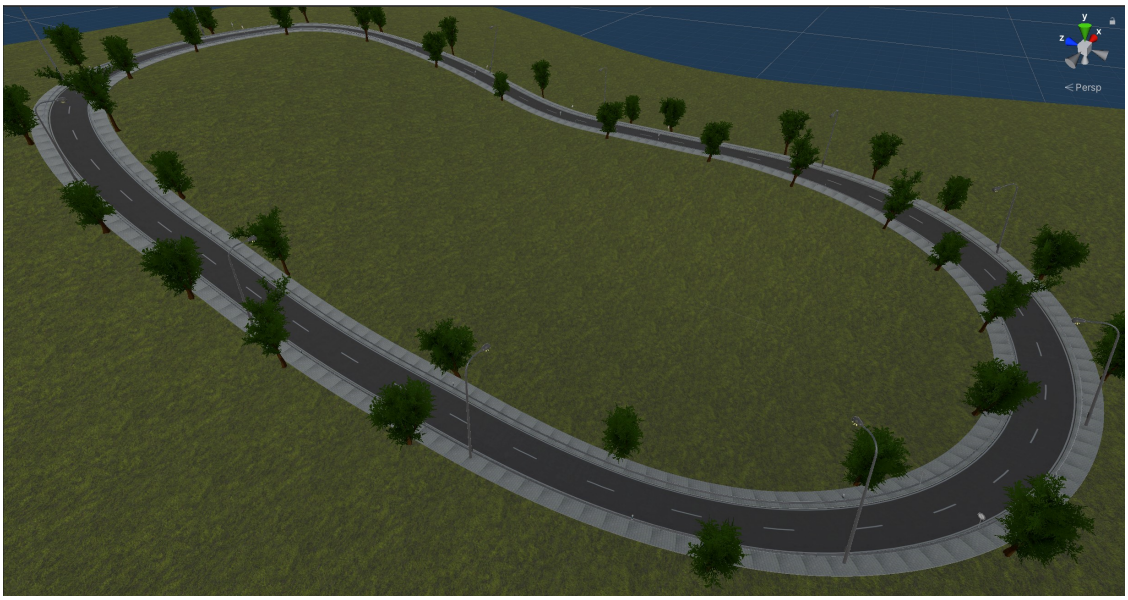


*Bild 5-22*: *Automatisch generierter Straßenverlauf mit interpoliertem Gelände und Randobjekten durch Import einer OpenDRIVE-Definition.*

und die Ergänzung neuer Stützpunkte zur Annäherung an den realen Streckenverlauf. Nach der Korrektur ist der bereits beschriebene Workflow zu wiederholen.

Auch wenn der OSM Standard Höheninformationen grundsätzlich unterstützt, sind sie in den meisten Fällen nicht verfügbar. Diese sind für die Nachbildung von Realstrecken allerdings unverzichtbar. Steilhänge am Straßenrand, Gräben und der generelle An- oder Abstieg, sowie die Querneigung der Straße werden vom Fahrer deutlich wahrgenommen. Nur wenn diese Eindrücke in die virtuelle Szene übertragbar sind, wird ein realistischer Eindruck der Fahrstrecke erzielt. Um das bis hierhin planare Straßennetz um die Höheninformationen anzureichern, erlaubt Hyperion die Integration der Daten aus Höhendatenbanken. Im hier betrachteten Anwendungsbeispiel des Rundkurses Borchen-Etteln (siehe Bild 5-18) wird der Prozess unter Rückgriff auf die Höhendatenbank NRW beschrieben [Inf20].

Die Höhendatenbank enthält Höhendaten von NRW mit einer Genauigkeit von 1 m × 1 m in Universal Transverse Mercator (UTM) Koordinaten. UTM Koordinaten sind bis auf einen konstanten Offset kompatibel zu den Positionsdaten des generierten Streckenverlaufs, da die Mercator-Projektion bereits angewandt wurde. Aufgrund der Fülle von Daten ist das Höhenprofil von NRW in Dateien mit einer Ausdehnung von 2 km × 2 km unterteilt. Abhängig von der befahrenen Strecke müssen die benötigten Dateien aus der Höhendatenbank geladen werden. Für den Rundkurs Borchen-Etteln sind acht dieser Dateien erforderlich. Nach diesen Vorarbeiten kann Hyperion die Höhendaten automatisiert in ein virtuelles Gelände überführen. Aufgrund der hohen Auflösung werden auch Straßengräben abgebildet. Das Bild 5-24 zeigt das Ergebnis für den Rundkurs Borchen-Etteln.

Das erzeugte Gelände wird darüber hinaus genutzt, um die aus der OSM-Definition generierte Straße mit Höhendaten anzureichern. Dazu wird über alle Stützpunkte aller Straßen iteriert. An jedem Stützpunkt werden mehrere Abfragen der Geländehöhe zwischen dem linken und rechten Fahrbahnrand vorgenommen. Der sich ergebende Mittelwert bildet die Höhe der Straßenmitte. Darüber hinaus wird über die Höhenunterschiede zwischen den

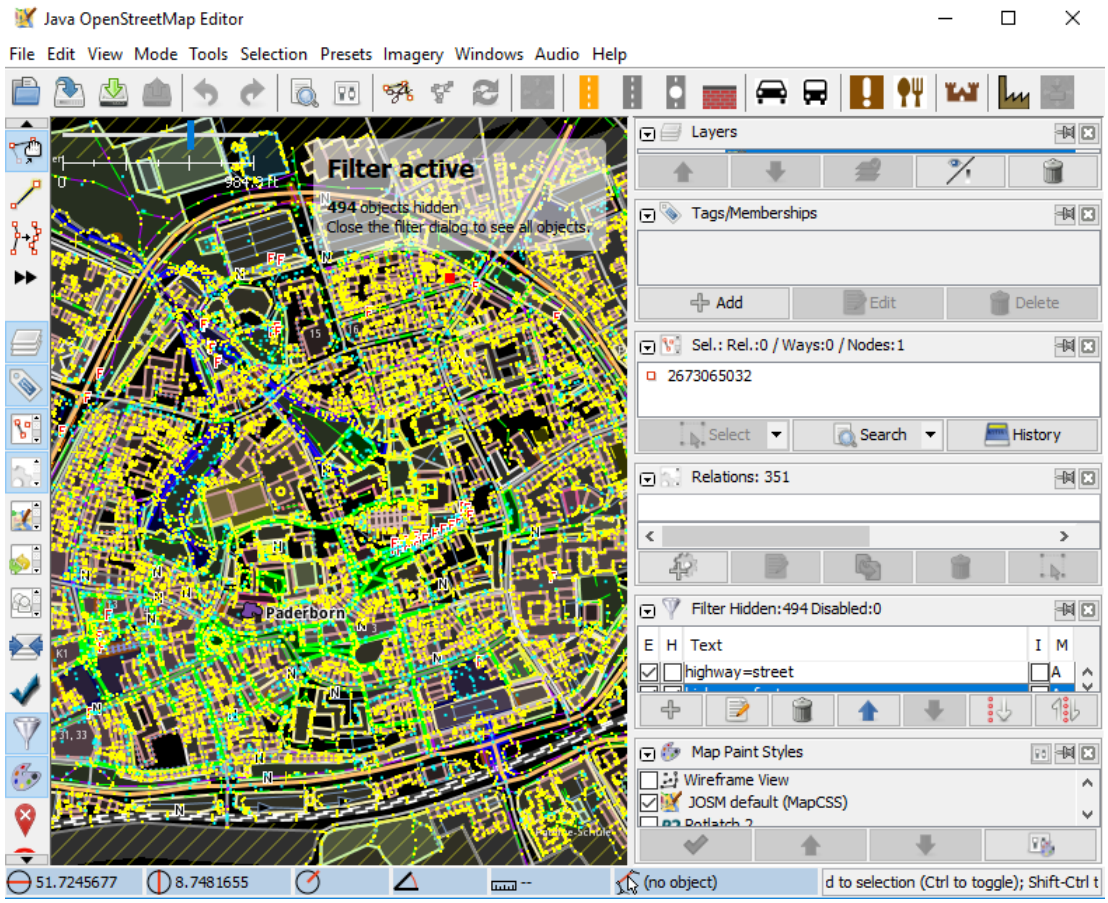


*Bild 5-23*: *Screenshot des Java OpenStreetMap Editors (JOSM) mit geladenem Kartenmaterial der Paderborner Innenstadt.*

Abfragen im linken und rechten Bereich der Straße eine geeignete Querneigung der Straße ermittelt. Schließlich wird über die Höhenunterschiede von Stützpunkt zu Stützpunkt die Steigung der Straße generiert. Hierbei werden unrealistisch hohe Änderungen, die beispielsweise durch Brücken über der Fahrbahn zustande kommen können, übersprungen. Hyperion bietet einen Export des mit diesen Informationen angereicherten Straßennetzes im OpenDRIVE-Format. Da das Format Höhenprofile und Querneigungsprofile der Straße als stückweise definierte kubische Polynome beschreibt, müssen die Daten an den diskreten Stützstellen durch Curve Fitting approximiert werden.

Von hier an liegen das Geländeprofil und die Strecke im OpenDRIVE-Format vollständig vor. Letztere kann mit dem in Abschnitt 5.3.1 beschriebenen Verfahren visualisiert werden. Zur besseren Wiedererkennbarkeit der realen Fahrstrecke bietet sich die manuelle Nachbearbeitung der generierten Szene an. So können markante Elemente entlang der Strecke und Gebäude in innerörtlichen Bereichen ergänzt werden. Die in Tabelle 5-1 gelisteten User Tags können weiterhin zur automatisierten Positionierung von Straßenrandobjekte eingesetzt werden. Das Bild 5-25 zeigt das Ergebnis für einen Ausschnitt des Rundkurses Borchen-Etteln.

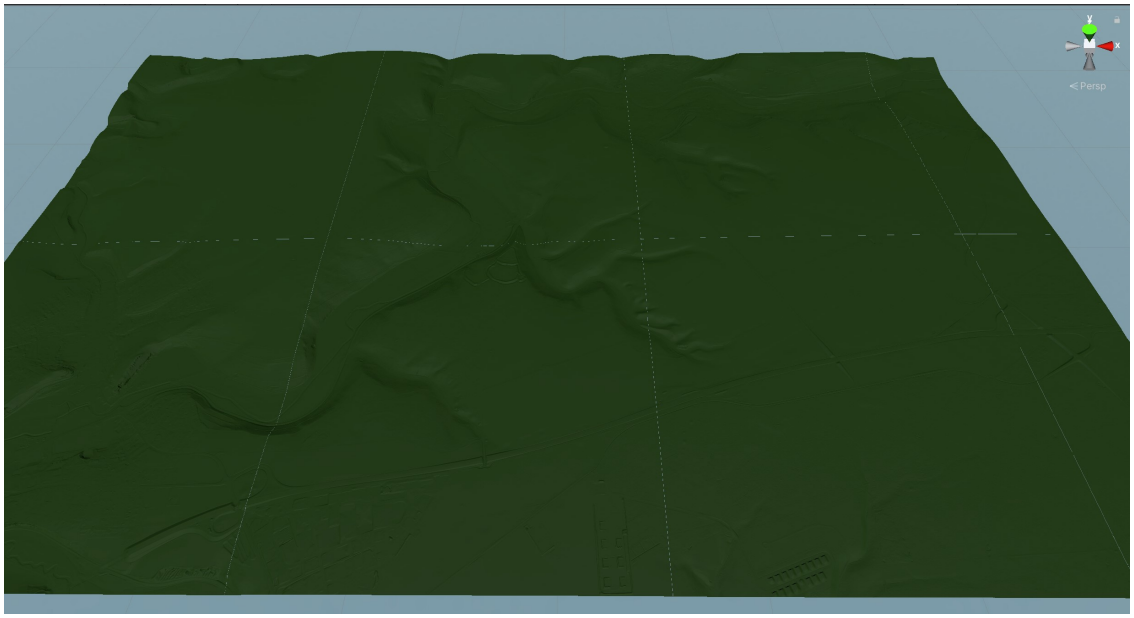

*Bild 5-24*: *Automatisch generiertes Gelände mit Höhenprofil für den Rundkurs Borchen-Etteln.*

## 5.4 Visuelle Simulation

In den nachfolgenden Unterabschnitten werden die Komponenten der visuellen Simulation, dargestellt im mittleren Bereich des Architekturbilds 5-1, beschrieben. Die visuelle Simulation ist der zentrale Baustein zur Beantwortung der in Abschnitt 4.4 formulierten Forschungsfrage. Nachdem im Unterabschnitt 5.4.1 kurz auf einige allgemeine Aspekte der Visualisierung eingegangen wird, schließt sich ein Abschnitt zur virtuellen Sensorik an. Diese generiert die vom Scheinwerfersteuergerät benötigten Eingangsgrößen. Die Rückgaben des Steuergeräts, welche sich in Dimmwerten der Pixellichter äußern, werden in der visuellen Simulation visualisiert. Der Unterabschnitt 5.4.3 beschreibt das Lichtquellenmodell, das zur Virtualisierung von Pixel-Scheinwerfersystemen entwickelt wurde, hinsichtlich seiner Integration in die Gesamtarchitektur. Eine ausführliche Diskussion der Lichtsimulation folgt in Kapitel 6. Die Komponente „Umgebung und Verkehr“ wird nicht in einem gesonderten Abschnitt dargestellt, da Streckengenerierung und die Steuerung der anderen Verkehrsteilnehmer bereits in den Abschnitten 5.3 und 5.2.3 beschrieben wurden. Noch nicht diskutiert wurde die Simulation von Witterungseinflüssen, auf welche aufgrund ihrer engen Verzahnung mit der Lichtsimulation in Kapitel 6 näher eingegangen wird.

### 5.4.1 Visualisierung

Einleitend sollen in diesem Abschnitt allgemeine Informationen zum Aufbau und zur grundlegenden Funktion der Visualisierung vorgestellt werden. Spezifiziert durch die Offline-Parametrierung, welche in Abschnitt 5.7 Erwähnung findet, wird zum Simulationsstart die Szenerie geladen. Unmittelbar danach startet die Simulation. Das Bild 5-26 zeigt eine Momentaufnahme der laufenden Simulation.

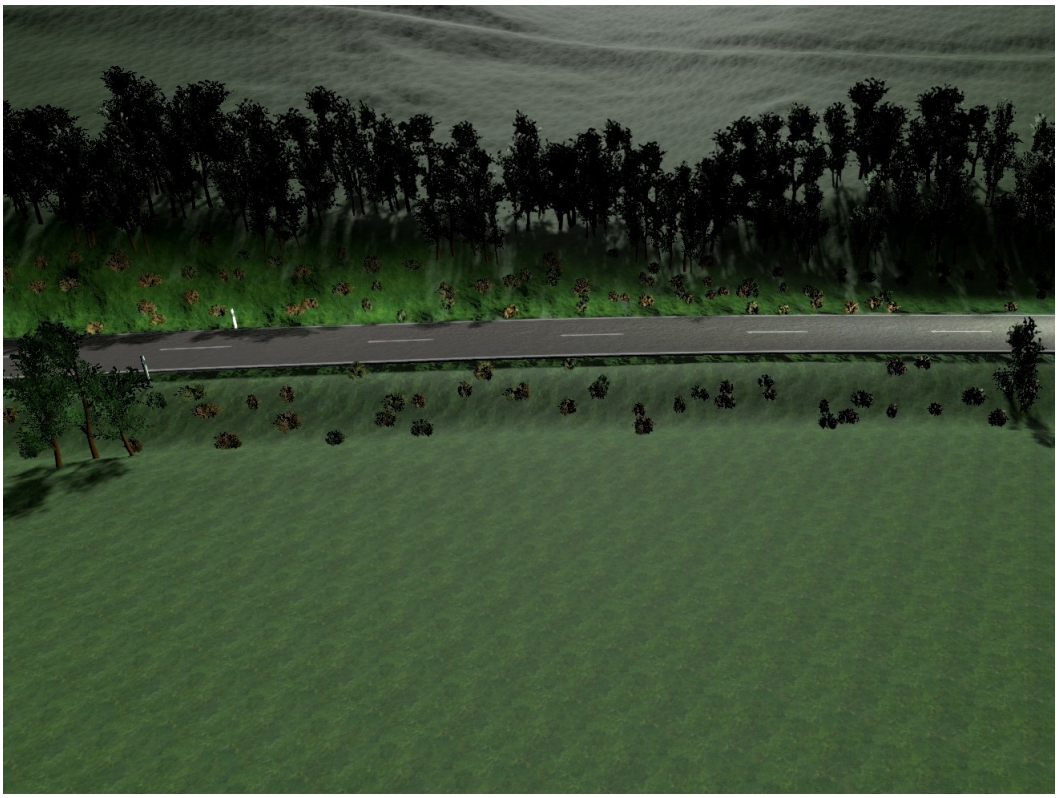

*Bild 5-25*: *Realstrecke mit eingebettetem OSM-Straßennetz und generierter Randbebauung in Hyperion.*

Um das Ausgabegerät bestmöglich zu nutzen, wird das ganze Bild durch die Simulationsausgabe ausgefüllt. Zur Interaktion mit der Anwendung befinden sich an den seitlichen und oberen Rändern Schaltflächen, durch welche verschiedene Menüs geöffnet werden können. Zum Beispiel werden in Bild 5-26 Fahrzeugzustände und Fahreingaben dargestellt, indem zuvor die Schaltfläche „Vehicle Variables“ am rechten Rand der Anzeige betätigt wurde. Weitere Schaltflächen dienen zur Manipulation der Online-Parameter, zur Analyse und zum Design der Lichtfunktion oder zum Setzen von Tastaturkürzeln. Nachfolgend werden die einzelnen Funktionalitäten detaillierter beschrieben.

### 5.4.2 Virtuelle Sensorik

Zur Bestimmung einer Lichtverteilung, die für die vorliegende Verkehrssituation adäquat ist, benötigt das Steuergerät sowohl Informationen über das eigene Fahrzeug als auch über das Fahrzeugumfeld. Die Sensoren stellen dabei die Sinnesorgane des Steuergeräts zur Wahrnehmung des Umfelds dar. In der Simulation müssen die realen Sensoren durch virtuelle ersetzt werden, um für das im HiL-Betrieb eingebundene Steuergerät gleiche Umgebungsbedingungen sicherzustellen. Dieser Abschnitt beschreibt die Umsetzung der virtuellen Sensoren.

Das von HELLA bereitgestellte Steuergerät eines HD84-Pixelsystems dient zur Erprobung der Implementierung. Dieses Steuergerät bestimmt die Dimmwerte der Pixel beider Schein-


*Bild 5-26*: *Stadtfahrt mit Abblendlicht in Hyperion (rechts: Fahrzeuggrößen).*

werfer unter Berücksichtigung der in Tabelle 5-2 zusammengefassten Fahrzeugsensorwerte.

*Tabelle 5-2*: *Im Scheinwerfersteuergerät berücksichtigte Fahrzeugsensorwerte.*

| Wert | Datentyp | Kommentar |
|---|---|---|
| $v_{FL}$ | Float | Radgeschw. vorne links in km/h |
| $v_{FR}$ | Float | Radgeschw. vorne rechts in km/h |
| $\phi_{Steer}$ | Float | Mittlerer Radeinschlag in ° |
| $\dot{\Phi}$ | Float | Gierrate in $°/s$ |

Neben den Informationen über das eigene Fahrzeug muss das Umfeld sensiert werden. Dazu verwendet das Steuergerät des HD84-Systems eine Umfeldkamera, welche über ein zwischengeschaltetes Kamerasteuergerät angekoppelt ist. Das Kamerasteuergerät extrahiert aus den Bildsequenzen des Kamerasensors Objektlisten, welche es an das Lichtsteuergerät weiterleitet. Im betrachteten Fall können maximal acht Objekte gleichzeitig erkannt werden, wobei für jedes Objekt die in Tabelle 5-3 aufgelisteten Informationen bereitgestellt werden. In diesem Zusammenhang schlüsselt Tabelle 5-4 die Typziffern auf. Die Winkelgrößen beziehen sich auf den Ursprung des Kamerakoordinatensystems. Zum besseren Verständnis werden sie in Bild 5-27 visualisiert.

Da die Umfeldkamera und ihr Steuergerät nicht Teil des zu erprobenden Systems sind, kann ihre Virtualisierung einfach umgesetzt werden [Goh18]. Die Positionen und Orientierungen aller Fahrzeuge sind in der Simulation zu jedem Zeitpunkt vollständig bekannt. Es genügt, die Positionen der Scheinwerfer und Rücklichter der anderen Verkehrsteilnehmer in das Kamerakoordinatensystem des Egofahrzeugs zu überführen und dann die in Tabelle 5-3 aufgelisteten Größen zu ermitteln.

*Tabelle 5-3*: *Objektdaten eines Objekts aus der Objektliste der Umfeldkamera.*

| Wert | Datentyp | Kommentar |
|---|---|---|
| ID | 0, 127 | Über die Sichtbarkeitsdauer eindeutige ID |
| Typ | 0, 7 | Typisierung des Objekts |
| HorPos | $[-25.5°; 25.5°]$ | Horizontaler Winkel zur Objektmitte |
| HorDim | $[0°; 10°]$ | Horizontale Ausdehnung des Objekts |
| VerPos | $[-12.75°; 12.75°]$ | Vertikaler Winkel zur Objektuntergrenze |

*Tabelle 5-4*: *Aufschlüsselung der Typziffern von detektierten Objekten.*

| Typziffer | Bedeutung |
|---|---|
| 0 | kein Objekt erkannt |
| 1 | einzelner Scheinwerfer |
| 2 | einzelnes Rücklicht |
| 3 | Scheinwerferpaar |
| 4 | Rücklichtpaar |
| 5 | Objektcluster |
| 6/7 | bisher nicht definiert |

Ein Verkehrsteilnehmer wird jedoch nur dann in die Objektliste aufgenommen, wenn dieser im Sichtbarkeitsbereich der Kamera liegt. Hierzu werden verschiedene Überprüfungen durchgeführt. Zunächst wird gemäß den Ungleichungen (5-4) geprüft, ob das erkannte Objekt im View Frustum der Umfeldkamera liegt.

$$\begin{aligned} minHorPos \leq\ & HorPos \leq maxHorPos \\ minVerPos \leq\ & VerPos \leq maxHorPos \\ & Dist \leq maxDist \end{aligned} \tag{5-4}$$

Die Grenzen für die winkelbezogenen Größen können Tabelle 5-3 entnommen werden. Als maximale Distanz *maxDist* wird initial ein Wert von 120m verwendet. Alle genannten Größen sind anpassbar. Außerdem muss sichergestellt werden, dass das Objekt nicht durch andere Szenenelemente verdeckt wird. Dazu werden Raycasts auf das Objekt angewendet. Hierbei handelt es sich um Strahlen, die von der Umfeldkamera in Richtung des Objekts ausgesendet werden. Betrachtet wird die Kollision dieser Strahlen mit Objekten innerhalb der virtuellen Szene. Nur, wenn diese nicht im Vorhinein mit anderen Objekten der Szene kollidieren, wird das betrachtete Objekt in der Objektliste berücksichtigt.

Die bisher beschriebenen Tests werden für jeden Scheinwerfer und jedes Rücklicht aller anderen Verkehrsteilnehmer durchgeführt. Sollte nur ein getestetes Element eines Verkehrsteilnehmers als sichtbar eingestuft werden, so wird dieses durch die Wahl der entsprechenden Typziffer gemäß Tabelle 5-4 berücksichtigt.

Neben der Objektliste liefert die Kamera noch eine Information darüber, ob sich das Egofahrzeug zurzeit inner- oder außerorts befindet. Diese Funktion wird durch die Erkennung der Straßenbeleuchtung realisiert und durch den Wert „CityDetection“ an das Lichtsteuergerät weitergegeben. Die zulässigen Werte sind die Ziffern 0 bis 4, welche die Bedeutungen „nicht aktiv“, „außerorts“, „innerorts“ und „nicht definiert“ tragen.

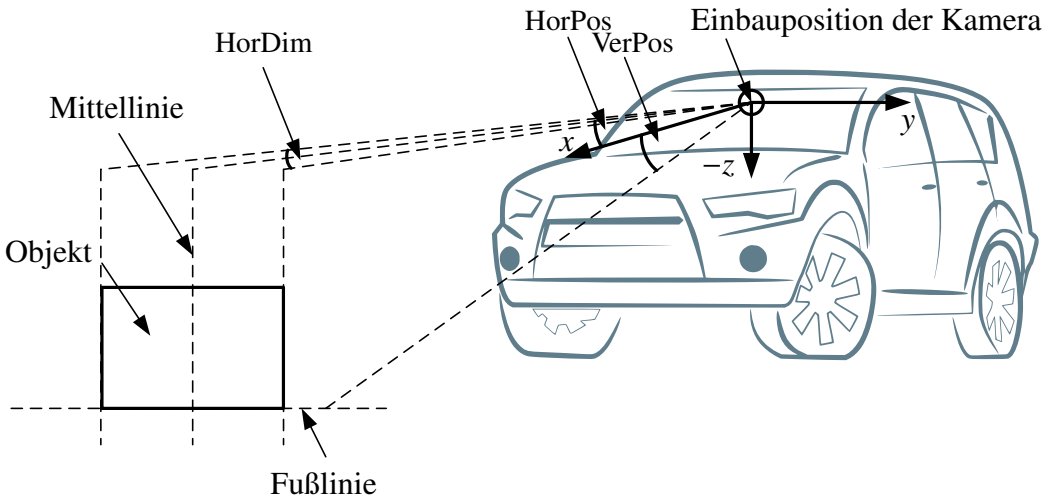


*Bild 5-27*: *Koordinatensystem der Umfeldkamera und Zuordnung geometrischer Größen der Objektliste.*

Die in diesem Abschnitt genannten Signale, bilden die Eingänge der verschiedenen Lichtfunktionen, die auf dem Steuergerät implementiert sind. Insofern nimmt die Qualität der Sensorik einen erheblichen Einfluss auf die Leistungsfähigkeit und die Funktionalität des Scheinwerfersystems. Beispielsweise wäre es sinnlos, hoch aufgelöste Lichtmodule für ein blendfreies Fernlicht zu verbauen, wenn die Umfeldkamera die Lage anderer Verkehrsteilnehmer nur mit viel gröberer Auflösung beschreiben kann. Die Zuführung der Sensorsignale an das Steuergerät wird im Abschnitt 5.5 beschrieben.

### 5.4.3 Virtuelle Lichtquelle

Eine der wesentlichen Herausforderungen bei der Gestaltung einer Nachtfahrtsimulation ist die adäquate Virtualisierung der Fahrzeugscheinwerfer. Tatsächlich eignet sich keine der Lichtquellentypen aus der Unity-Engine für diese Anwendung. Am ehesten infrage käme das Spotlicht, dessen grundsätzliche Funktion bereits in Abschnitt 2.3.4 erläutert wurde. Problematisch hierbei ist jedoch die Berücksichtigung der dynamischen Lichtverteilung des Scheinwerfers. In Unity lassen sich lokale Anpassungen der Lichtintensität durch Cookies realisieren, welche als Transparente mit lokal variierender Durchlässigkeit über die Lichtquelle gelegt werden. Genaueres hierzu wurde ebenfalls in Abschnitt 2.3.4 diskutiert. Die geometrische Handhabung eines Cookies ist im Fall des Spotlichts insofern problematisch, als dass es die Grundfläche einer Pyramide darstellt, in deren Spitze die Lichtquelle sitzt. Lichtverteilungen von KFZ-Scheinwerfern, wie sie in Abschnitt 3.1.1 beschrieben wurden, werden bezüglich Kugelkoordinaten mit der Lichtquelle als Drehzentrum aufgezeichnet.

Für einen gegebenen Spitzenwinkel der Pyramide und einer gegebenen Rastergröße des Cookies könnte die Lichtintensität für jedes Rasterelement des Cookies ermittelt werden. Allerdings ergibt sich hierbei ein Auflösungsproblem. Während die bezüglich Polar- und Azimutwinkel dargestellte Lichtverteilung in allen Raumrichtungen mit gleichbleibender Rastergröße auflöst, wird das Cookie des Unity Spotlichts mit zunehmender Nähe zur Lichtmittelachse ungenauer bezüglich der Raumwinkel. Die Skizze 5-28 visualisiert diesen

Effekt, wobei die Darstellung aus Gründen der Übersichtlichkeit auf zwei Dimensionen reduziert ist. Links in Bild 5-28 wird die Interpretation der Lichtverteilung bzw. des Cookies

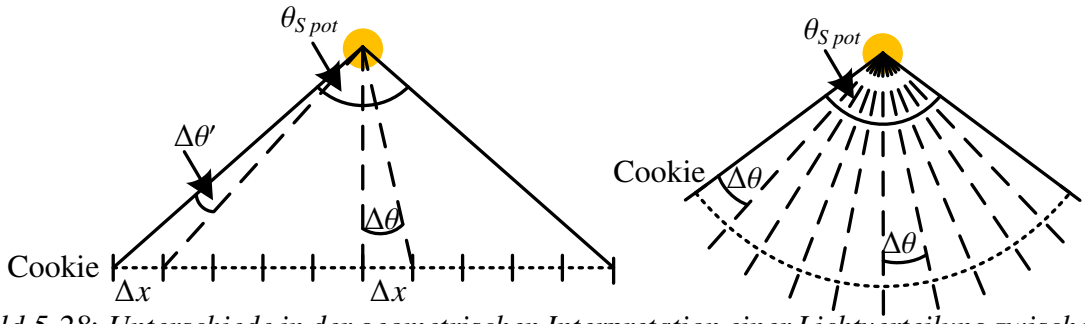


*Bild 5-28*: *Unterschiede in der geometrischen Interpretation einer Lichtverteilung zwischen Unity's Spotlicht und einem KFZ-Scheinwerfer.*

durch ein Spotlicht der Unity-Engine visualisiert. Die Lichtverteilung wird geometrisch als Pyramidengrundfläche verstanden. Die Auflösung innerhalb der Pyramidengrundfläche ist konstant. Für die dargestellte Dimension beträgt der Abstand zweier Messpunkte $\Delta x$. Geometrisch bedingt werden die Raumwinkel, für welche die Messpunkte gültig sind, mit zunehmendem Abstand zur Lichtmittelachse kleiner. In der Skizze findet sich dieser Zusammenhang durch die Beziehung $\Delta\theta' < \Delta\theta$ wieder.

Außerdem ist problematisch, dass einige Lichtverteilungen über einen Azimutintervall von $-90°$ bis $+90°$ vermessen werden. Um diese Lichtverteilungen vollständig abbilden zu können, müsste die Pyramide des Spotlichts einen nicht realisierbaren Spitzenwinkel von $180°$ haben. Derartige Lichtverteilungen ließen sich auf ein Spotlicht nur durch vorheriges Beschneiden der Lichtverteilung und unter starken Auflösungseinbußen im Zentralbereich anwenden. Dieses Vorgehen ist mit der Anforderung **A1** (Korrekte Abbildung photometrischer Zusammenhänge) jedoch nicht in Einklang zu bringen.

Rechts in Bild 5-28 ist hingegen eine fiktive Lichtquelle dargestellt, welche die Lichtverteilung mit einem geeigneten geometrischen Ansatz interpretiert. Dazu wird die Lichtverteilung bzw. das Cookie nicht als planare Fläche, sondern als Kugelflächenstück aufgefasst. Der Winkel $\Delta\theta$ zwischen benachbarten Messpunkten bleibt in jeder Raumrichtung unverändert.

Zusammengefasst findet sich keine geeignete Lösung zur physikalisch korrekten Nachbildung von KFZ-Scheinwerfern innerhalb der Unity-Engine. Aus diesem Grund wird eine eigene Lichtquelle implementiert, welche auf die Anforderungen im vorliegenden Kontext zugeschnitten ist.

Ein weiterer Vorteil einer Eigenentwicklung ist die freie Manipulierbarkeit aller zugrunde liegenden Funktionen. Diese Möglichkeit wird im Rahmen von Hyperion beispielsweise zur Unterstützung von Retroreflexion eingesetzt. Hierbei handelt es sich um die Wechselwirkung von Licht mit Reflektoren, wie sie als Folie auf Straßenschildern oder an Leitpfosten zu finden sind. Außerdem werden die in Kapitel 7.1 vorgestellten Analysesichten in der Hyperion-Lichtquelle integriert.

Bei der Implementierung dieser nachfolgend als Basislichtquelle bezeichneten Komponente wird auf Ray Tracing Verfahren verzichtet und stattdessen klassische Rastergrafik eingesetzt. Der primäre Grund für diese Entscheidung ist die Anforderung **A6** (Echtzeitfähigkeit). Zu Beginn der Arbeiten in 2017 existierten noch keine Grafikkarten am Markt, welche Ray Tracing durch Hardware-Unterstützung beschleunigten. Ray Tracing basierte Verfahren sind insbesondere aufgrund der Traversierung des Szenegraphen bei der Kollisionsprüfung sehr rechenintensiv. Genaueres hierzu kann in Abschnitt 2.3.4 nachgelesen werden. Heute werden Ray Tracing Verfahren für einzelne Effekte in Echtzeitanwendungen wie Computerspielen eingesetzt. In den kommenden Jahren ist mit einer vollständigen Durchdringung zu rechnen. Wesentliche Treiber sind hierbei die Ende 2018 neu aufgekommene Hardware-Unterstützung durch die Grafikkarte (z.B. GeForce RTX-Serie) und intensive Forschungsarbeiten im Bereich der zugrunde liegenden Algorithmik. Unter den genannten Gegebenheiten wurde die Architektur der Lichtsimulation so gewählt, dass eine nachträgliche Überführung der Basislichtquelle auf Ray Tracing Verfahren ohne fundamentale Anpassungen möglich ist. Dies wird vor allem durch die Entkopplung der Gesamtlichtverteilungsermittlung von der Lichtquelle sichergestellt.

Nachdem Ray Tracing Verfahren ausgeschlossen wurden, verbleibt die Entscheidung zwischen Forward und Deferred Rendering. Die Gegenüberstellung dieser Verfahren wurde bereits in Abschnitt 2.3.3 vorgenommen. In der vorliegenden Anwendung ist mit einem hohen Aufkommen dynamischer Lichter in der Szene zu rechnen. Zunächst verfügt jedes Auto über vier dynamische Lichtquellen, da die Scheinwerfer jedes Fahrzeugs ihre Lage in der Szene zusammen mit dem Fahrzeugaufbau verändern. Darüber hinaus sind die Scheinwerfer in doppelter Hinsicht dynamisch, da neben ihrer Position auch ihre Lichtverteilung zeitlichen Änderungen unterliegt. Es lassen sich deshalb keine Preprocessing-Verfahren zur Aufwandsreduzierung anwenden. Aufgrund der hohen Anzahl dynamischer Lichtquellen fällt die Entscheidung auf die Deferred Rendering Pipeline, welche das Beleuchtungsmodell nur auf die tatsächlich ausgegebenen Pixel anwendet und somit bei dieser Vielzahl von Lichtquellen zur Einhaltung der Anforderung **A6** (Echtzeitfähigkeit) beiträgt.

Technisch angebunden wird die virtuelle Lichtquelle durch eine Skript-Komponente mit der Bezeichnung „Headlight-Controller“, welches unmittelbar an das Gameobject des Fahrzeugs angebunden ist. Der Headlight-Controller stellt die einzige Komponente dar, die der Anwender in den Szenegraphen einbinden muss, um hochauflösende Scheinwerfer an einem virtuellen Fahrzeug simulieren zu können. Beim Simulationsstart vervollständigt der Headlight-Controller die Architektur der Lichtquellen im Szenegraphen selbstständig. Nachfolgend ist die auf diese Weise generierte Struktur innerhalb des Szenegraphen dargestellt. Das Wurzelobjekt „Car“ ist mit dem im Abschnitt 5.2.1 gleich benannten Gameobject gleichzusetzen. Die dort aufgeführten Child-Elemente und Components von Car werden hier der Übersichtlichkeit halber durch Auslassungspunkte ersetzt.

- **Car** mit Components: *Headlight-Controller, ...*
  - **Headlight** mit Components: -
    - **FL** mit Components: *Headlamp, CustomLight, (Combiner)*
    - **FR** mit Components: *Headlamp, CustomLight, (Combiner)*
  - **...**

Zuerst erzeugt der Headlight-Controller ein Child-Gameobject des Fahrzeugs und bezeichnet dieses mit dem Namen „Headlight“. Dieses dient ausschließlich der Übersichtlichkeit,

indem es die weiteren notwendigen Elemente kapselt. Darin werden weitere Child-Objekte mit den Bezeichnungen „FL“ und „FR“ erzeugt. Ihre Positionen und Orientierungen relativ zum lokalen Koordinatensystem des Fahrzeugs können durch Parameter vorgegeben werden. Auf diese Weise lassen sich beliebige Einbaulagen abbilden. FL und FR erhalten außerdem jeweils ein Script-Component mit der Bezeichnung „Headlamp“. Dieses Skript verwaltet einen einzelnen Scheinwerfer. Es erzeugt am selben Gameobject ein weiteres Script-Component mit der Bezeichnung „CustomLight“, welches die Lichtquelle zur Virtualisierung von HD-Scheinwerfern repräsentiert. Außerdem erzeugt das Headlamp-Script eine Instanz der Klasse „Combiner“. Der Combiner taucht nicht im Szenegraphen auf, da er die Klasse „MonoBehaviour“ nicht implementiert. Seine Aufgabe ist die Ermittlung der Gesamtlichtverteilung des Scheinwerfers aus den momentanen Dimmwerten seiner Lichtquellen. Diese Gesamtlichtverteilung kann im Anschluss als Cookie des zugehörigen CustomLight verwendet werden. Er ist somit hierarchisch auf der gleichen Stufe und wird eingeklammert in obiger Struktur aufgeführt.

Der Headlight-Controller übernimmt neben der Initialisierung des Scheinwerfersystems auch das Management zur Laufzeit. So werden zum Beispiel über ihn Dimmwerte an die Scheinwerfer weitergegeben oder Ausfälle einzelner LEDs simuliert. Konkret gibt er die Anweisungen an das jeweilige Headlamp-Script weiter, welches wiederum entsprechenden Manipulationen des CustomLight oder des Combiner vornimmt. Eine detailliertere Darstellung der Scheinwerfersimulation findet sich in Kapitel 6.

## 5.5 XiL-Betrieb des Steuergeräts

Primäres Ziel bei der simulativen Erprobung ist die optimale Auslegung von Lichtfunktionen. Technisch werden Lichtfunktionen durch Algorithmen abgebildet, welche auf dem Lichtsteuergerät implementiert sind. Zum Test des Steuergeräts stehen dem Entwickler verschiedene Einbindungsvarianten zur Verfügung. Hyperion deckt mit der Möglichkeit von Model(MiL)-, Software(SiL)- und Hardware(HiL)-in-the-Loop-Tests alle Stufen ab und erfüllt somit die Anforderung **A5** (XiL-Techniken).

Die MiL-Einbindung des Steuergeräts erfordert weder reale Fahrzeugkomponenten, noch eine Umgebung, die das Steuergeräteumfeld detailgetreu simuliert. Hyperion sieht die MiL-Einbindung durch die Implementierung der Steuergerätelogik in einer C# -Klasse vor, welche unmittelbar in die Klassenhierarchie der Gesamtsimulation eingebunden werden kann. Die Kompatibilität zwischen der Steuergeräte-Klasse und Hyperion wird über die Vorgabe sogenannter Interfaces sichergestellt, welche durch die Steuergeräte-Klasse implementiert werden müssen. Somit ist sichergestellt, dass zur Funktionalität unverzichtbare Schnittstellen zur Verfügung stehen, ohne deren innere Logik vorzugeben. Die folgende Aufzählung listet die wichtigsten Schnittstellenmethoden auf:

- **IHeadlightECU** ECU-Interface
  - *InitSensorSetup (SensorList)* Übergabe der Sensorreferenzen
  - *Calculate ()* Berechnung der Dimmwerte
  - *IsReady () : Boolean* Berechnung abgeschlossen?
  - *CurrentValues () : ValueList* Ausgabe der aktuellen Dimmwerte

  - *LastValues () : ValueList* Ausgabe der Dimmwerte des vorigen Takts
  - *IsDirty () : Boolean* Dimmwerte nach letztem Takt geändert?
  - *ActivateFunction (LightFunction)* Lichtfunktion aktivieren
  - *DeactivateFunction (LightFunction)* Lichtfunktion deaktivieren
  - *GetActiveFunctions () : LightFunctionList* Ausgabe aktiver Lichtfunktionen
- **IHeadlightECUInject** Interface zur externen Manipulation
  - *SetCurrentValues (ValueList)* Dimmwerte vorgeben
  - *DimPixel (PixelID, DimValue)* Dimme Lichtquelle mit ID *PixelID*
  - *DegradePixels (DegrationList)* Degradierung der Lichtquellen vorgeben
  - *DegradePixel (PixelID, Degration)* Degradiere Lichtquelle mit ID *PixelID*

Wie aus der Auflistung hervorgeht, sind zwei Interfaces für die MiL-Einbindung der Steuergerätelogik vorgesehen. Dabei ist die Implementierung des Interfaces **IHeadlightECU** zwingend erforderlich, während das Interface **IHeadlightECUInject** nicht notwendigerweise implementiert werden muss. Letzteres ist vorgesehen, um über die Vorgaben des Steuergeräts hinaus externe Eingriffe vornehmen zu können. Diese Schnittstelle kann beispielsweise genutzt werden, um den Ausfall oder die Degradierung verschiedener Lichtquellen zu simulieren.

Die eigentlichen Lichtfunktionen bindet der Entwickler in der *Calculate*-Methode ein. Die im Abschnitt 5.4.2 beschriebenen Sensorwerte stehen ihm dabei als Eingangswerte zur Verfügung. Abhängig davon, welche Sensoren durch das Steuergerät tatsächlich genutzt werden, können diese durch die Methode *InitSensorSetup* an die Steuergeräte-Instanz übergeben werden. Ausgabe der Algorithmen sind die Dimmwerte aller Pixel des linken und rechten Scheinwerfers, welche durch die Methode *CurrentValues* abgefragt werden können. Diese werden schließlich über den Headlight-Controller an die virtuellen Scheinwerfer weitergegeben, sodass die Auswirkungen der implementierten Lichtsteuerung in der virtuellen Szene beobachtbar werden.

MiL-Tests stellen die agilste Variante der XiL-Einbindungen dar. Durch die einfache Einbindung können innerhalb kürzester Zeit Anpassungen und Parameterstudien durchgeführt werden. Auf diese Weise kann der Entwickler zielgerichtet ein Modell der Lichtsteuerung entwerfen, dass den Anforderungen gerecht wird.

Im nächsten Schritt wird dieses Modell in die Steuergerätesoftware übersetzt und für die Ausführung auf einer Echtzeithardware kompiliert. Darüber hinaus müssen die Umgebungsbedingungen des realen Steuergeräts nachgebildet werden. Vorrangig handelt es sich hierbei um die CAN Kommunikation mit anderen Fahrzeugkomponenten. Hyperion weist hierzu eine CAN Schnittstelle auf und emuliert das Restfahrzeug. Als Hardware wird ein USB-CAN Adapter der PEAK-System Technik GmbH verwendet, weshalb der einfache Betrieb an einem normalen Desktop-PC ohne weitere Hardwareanforderungen möglich ist [PEA20]. Beispielsweise werden die Werte der virtuellen Sensoren mit dem vorgegebenen Takt an das Lichtsteuergerät gesendet. Umgekehrt empfängt Hyperion die Nachrichten des Steuergeräts, welche die Dimmwerte aller Pixel beider Scheinwerfer enthalten. Diese Konfiguration stellt einen SiL-Test dar.

In der finalen Teststufe wird der HiL-Test des Steuergeräts vollzogen. Dazu wird dieses prototypisch realisiert und anstelle der Echtzeithardware an die Simulation gekoppelt. Hinsichtlich der Einbindung seitens Hyperion besteht kein Unterschied zwischen SiL- und HiL-Tests. Zusammenfassend stellt das Bild 5-29 die drei möglichen XiL-Einbindungen skizzenhaft dar.

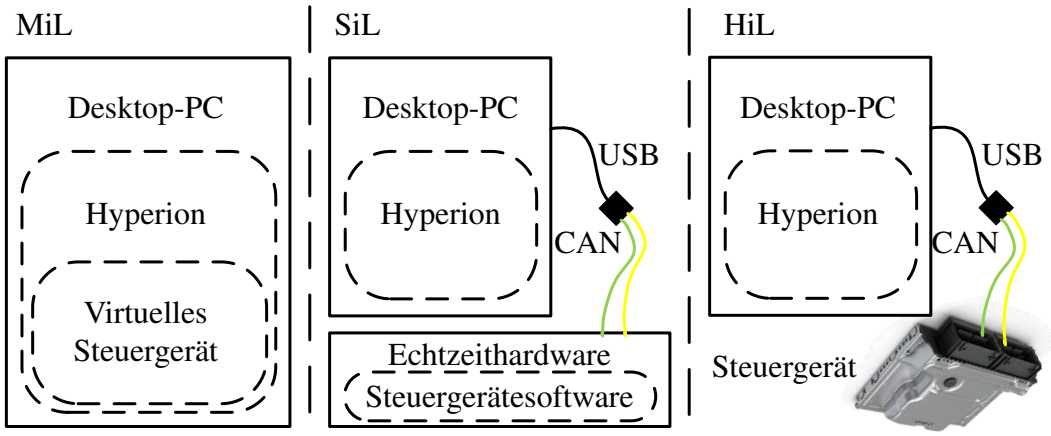


*Bild 5-29*: *XiL-Einbindungsmöglichkeiten der Lichtsteuerung in Hyperion.*

## 5.6 Simulator-Interface

Die Interaktion des Anwenders mit der Fahrsimulation geschieht über das Simulator-Interface. Dieses ist bidirektional und gliedert sich deshalb in zwei Aspekte. Zum einen fließen durch die visuelle Ausgabe Informationen von der Simulation zum Anwender. Andererseits kann der Anwender durch Bedienelemente Einfluss auf die zukünftige Entwicklung nehmen. Beide Aspekte werden in den nachfolgenden Unterabschnitten beschrieben.

### 5.6.1 Ausgabe

Als Anwender der Nachtfahrtsimulation muss der Entwickler oder Testfahrer bestmöglich über das aktuelle Verhalten der Lichtsteuerung und dessen Wechselwirkung mit der Umwelt informiert sein. Diese Kopplung zwischen der Nachtfahrtsimulation und dem Menschen geschieht durch die visuelle Ausgabe. Um der Anforderung **A11** (Konfigurierbarkeit) gerecht zu werden, kann die visuelle Ausgabe durch Hyperion auf verschiedene Art und Weise erfolgen. Grundsätzlich kann zwischen Desktop- und Simulator-Varianten unterschieden werden.

Die Desktop-Varianten sind auf den Betrieb der Nachtfahrtsimulation am Büroarbeitsplatz zugeschnitten. Konkret kann der Anwender zwischen 1-, 2- und 3-Monitor-Betrieb wählen. In Bild 5-30 werden diese Modi visualisiert.

Der 1-Monitor-Betrieb (oben links in Bild 5-30) stellt die geringsten Anforderungen an Rechenleistung und Hardware-Verfügbarkeit. Er lässt sich deshalb an jedem Desktop-PC

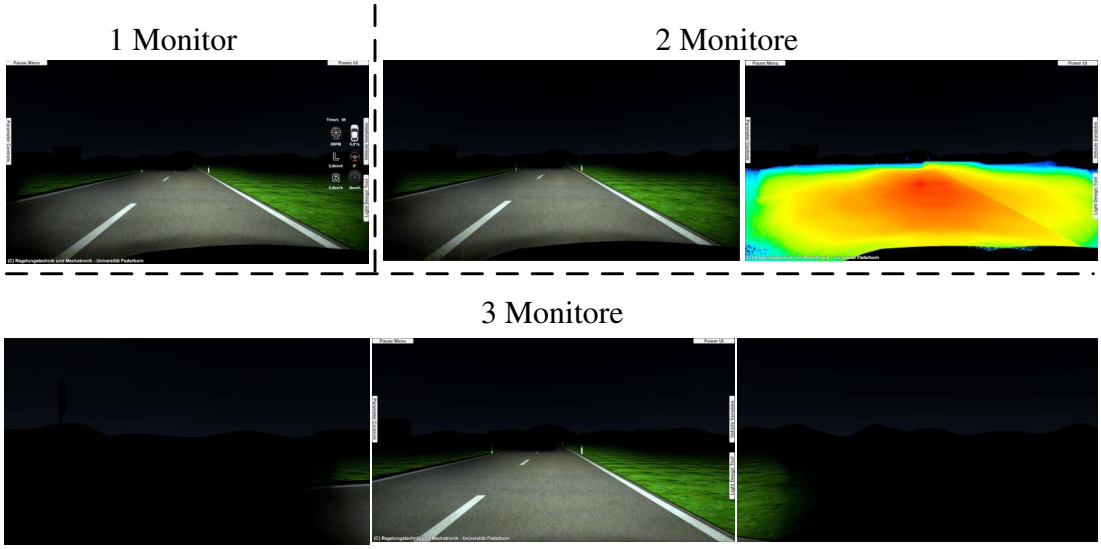


*Bild 5-30*: *Verschiedene Ausgabemodi von Hyperion im Desktop-Betrieb.*

mit hinreichender Grafikleistung betreiben. Der Monitor zeigt die Szene aus Sicht einer virtuellen Kamera. Deren Perspektive kann zur Laufzeit variiert werden. Im Randbereich der Anzeige können auf Wunsch verschiedene Menüs eingeblendet werden, die sämtliche verfügbare Interaktionen mit dem Benutzer ermöglichen, welche über das normale Fahren hinausgehen. Insbesondere kann anstelle des normalen Renderings eine Isolinien- oder Falschfarbendarstellung der Szene erfolgen, durch welche verschiedene photometrische Größen visualisiert werden können.

Häufig sind Arbeitsplätze mit zwei Monitoren ausgestattet. In diesem Fall erlaubt Hyperion die Wahl eines anderen Ausgabemodus. Dieser kann den Zustand der vorliegenden Verkehrssituation auf beiden Ausgaben vollkommen unabhängig voneinander visualisieren. Beispielsweise lassen sich verschiedene Kameraperspektiven einstellen. Noch interessanter ist allerdings die normale Darstellung auf einem Ausgabegerät, während die zweite Ausgabe die Szene aus der gleichen Perspektive, jedoch in einer Analyseansicht zeigt. Eine derartige Konfiguration wird im oberen rechten Bereich des Bilds 5-30 visualisiert. Details zu möglichen Analysefunktionen folgen in Kapitel 7. Außerdem können auf der zweiten Ausgabe die verschiedenen Menüs bedient werden, ohne die Darstellung der virtuellen Szene auf dem ersten Ausgabegerät zu beschneiden.

Schließlich verfügt Hyperion über eine 3-Monitor-Ausgabe, deren Verwendung sich für kompakte Fahrsimulatoren eignet. Dieser Modus ist im unteren Bereich des Bilds 5-30 dargestellt. In dieser Variante werden die virtuellen Kameras, welche zur Bildgenerierung für die verschiedenen Ausgabegeräte dienen, so parametriert, dass sich die Einzelbilder zu einem Gesamtbild der Szene zusammenfügen. Die Kameraeinstellungen sind in diesem Fall von den Beschaffenheiten und den Ausrichtungen der einzelnen Monitore abhängig und können in Hyperion parametriert werden. Durch die Verteilung der Ausgabe auf mehrere Geräte erhält der Anwender einen besseren Überblick und erfährt die Fahrsimulation immersiver.

Um auch Großsimulatoren bedienen zu können, verfügt Hyperion über einen weiteren hochgradig konfigurierbaren Ausgabemodus. Der besondere Anspruch in diesem Fall ist

die Vielzahl der Ausgabegeräte, die in einem Großsimulator typischerweise zum Einsatz kommen. Der Atlas Motion System (ATMOS) Fahrsimulator des Heinz Nixdorf Instituts diente zur Erprobung von Hyperion im Großsimulator-Betrieb. Das Bild 5-31 zeigt ein Foto des Simulators. Er verfügt über elf Ausgabegeräte.


*Bild 5-31*: *Foto des ATMOS Fahrsimulators in der Laborhalle des Heinz Nixdorf Instituts der Universität Paderborn.*

Im Detail versorgen acht Beamer eine 240°-Rundprojektion. Drei weitere Displays ersetzen die Außen- und den Innenspiegel des Fahrzeugmockups. Zur Versorgung dieser Vielzahl von Ausgabegeräten genügt ein einzelner Desktop-PC nicht. Um auf anwendungsspezifische und kostenintensive Hardware verzichten zu können, muss das Rendering der vielen virtuellen Kameras auf mehrere Rechner verteilt werden. Hierzu verfügt Hyperion über eine Netzwerkfähigkeit, welche durch die in Bild 5-32 dargestellte Master-Slave-Architektur umgesetzt wird.

Die mit „Master“ oder „Slave“ gekennzeichneten Einheiten können durch normale Desktop-PC mit hinreichender Grafikleistung dargestellt werden. Auf jedem dieser Rechner läuft eine Instanz von Hyperion, wobei sich die Initialisierungen der Instanzen rechnerspezifisch unterscheiden. Der Master ist für das gesamte Management der Simulation zuständig. Nur er bindet, sofern vom Anwender gewünscht, das Scheinwerfersteuergerät im HiL-Betrieb ein oder koppelt sich via Ethernet mit dem ASM Modell auf einer externen Echtzeithardware. Die Zustandsermittlung der Simulation zur Laufzeit erfolgt ebenfalls zentral am Master. So werden zum Beispiel die Lagen des Egofahrzeugs und der weiteren Verkehrsteilnehmer ermittelt. Auch die Interaktion des Versuchsleiters mit der Software erfolgt am Master. Hierzu verfügt der Master über eigene Ein- und Ausgabegeräte, durch welche die virtuelle Szene visualisiert und Eingaben entgegen genommen werden können.

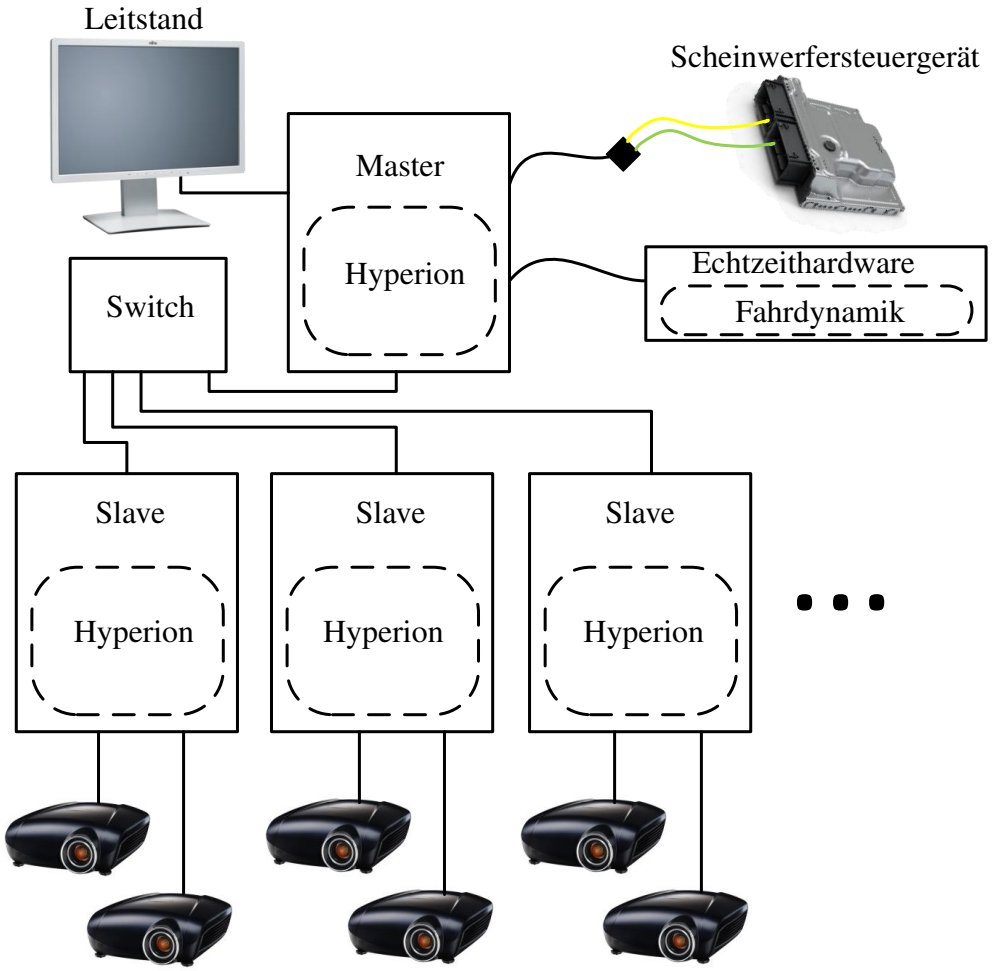


*Bild 5-32*: *Netzwerkfähigkeit durch Master-Slave-Architektur beim Einsatz von Hyperion an Großsimulatoren.*

In regelmäßigen Zeitabständen, die idealerweise der Bildwiederholrate entsprechen, synchronisiert der Master den momentanen Simulationszustand mit den Slaves durch eine UDP-Broadcast-Nachricht. Da diese Zustandsnachricht nicht spezifisch für einen einzelnen Slave gilt, wird die Performance durch eine hohe Anzahl von Slave-Rechnern nicht beeinträchtigt. Neben den Zustandsnachrichten gibt der Master auch vom Versuchsleiter vorgenommene Manipulationen an Online-Parametern durch eine UDP-Broadcast-Nachricht weiter. Im Gegensatz zur Zustandsnachricht wird die Parameternachricht allerdings nicht zeitgesteuert, sondern einmal pro Manipulation versendet.

Für jeden Slave können individuell virtuelle Kameras konfiguriert werden, welche die Szene aus einer bestimmten Perspektive rendern. Hierbei können bis zu drei Kameras pro Slave eingesetzt werden. Auf diese Weise kann die hohe Belastung des Renderings für viele Ausgabegeräte, wie sie in Großsimulatoren stets gegenwärtig ist, durch eine Skalierung der Hardware-Architektur angemessen aufgefangen werden. Im Beispiel des ATMOS Fahrsimulators werden fünf Slave-Rechner eingesetzt. Die Slaves 1 bis 4 rendern die Ausgaben von jeweils zwei Beamern der Rundprojektion, während Slave 5 den Innen- und die Außenspiegel mit Bildmaterial versorgt.

### 5.6.2 Eingabe

Neben den verschiedenen Ausgabemodi muss Hyperion zur Erfüllung der Anforderung **A11** (Konfigurierbarkeit) auch im Bereich der Eingabe entsprechend anpassbar gestaltet sein. Das Spektrum vom einfachen Arbeitsplatz-PC ohne besondere Hardwareausstattung bis hin zum Großsimulator müssen abgedeckt werden.

Zur grundlegenden Ausstattung eines Desktop-PC gehören Tastatur und Maus. Diese Eingabegeräte erlauben bereits die vollständige Bedienung der Nachtfahrtsimulation. Während die Maus vor allem zur Navigation und Auswahl in den verschiedenen Menüs eingesetzt wird, erfolgt die Fahrzeugsteuerung über die Tastatur. Zusätzlich können Shortcuts definiert werden, welche häufig verwendete Aktionen mit Tastenkombinationen verknüpfen.

Auch wenn diese Konfiguration der Eingabegeräte eine Minimalanforderung darstellt, die eine Verwendung der Software unter nahezu allen Bedingungen ermöglicht, stellt sie nicht den Idealfall dar. Ein großer Nachteil liegt in der binären Charakteristik der Tasteneingaben. Diese können nur die Zustände „Taste gedrückt“ und „Taste nicht gedrückt“ annehmen. Für kontinuierliche Stellgrößen, wie die Gaspedalstellung oder den Lenkwinkel, ist diese Art der Eingabe denkbar ungeeignet und führt zu einer schlechten Steuerbarkeit des Fahrzeugs. Damit begründet sich die Unterstützung einer Lenkrad-Pedal-Kombination als weiteres mögliches Eingabegerät in Hyperion. Näheres zur Verarbeitung der Fahreingaben wurde bereits in Abschnitt 5.2.1 thematisiert.

Die Lenkrad-Pedal-Kombination kann sowohl für die verschiedenen Desktop-Varianten als auch in Großsimulatoren verwendet werden. Bei letzteren wird jedoch meist ein komplexes Fahrzeugmodell eingesetzt, um die Sollvorgaben für das Bewegungssystem möglichst realitätsnah generieren zu können und die Immersion zu maximieren. In diesem Fall sind die Eingabegeräte gar nicht an Hyperion angebunden. Stattdessen wirken die Eingaben direkt auf die extern angebundene Echtzeithardware, auf welcher das Fahrzeugmodell simuliert wird. Hyperion wird durch den Fahrzeugzustand nur indirekt durch die Fahreingaben beeinflusst.

## 5.7 Remote-Applikation

Das Bild 5-32 skizziert die Architektur von Hyperion im Großsimulator-Betrieb. In diesem Betriebsmodus interagieren oft mehrere Personen in verschiedenen Rollen mit der Anwendung. Einerseits erfüllt ein Proband die eigentliche Fahraufgabe. Er nimmt die visuelle Ausgabe wahr und agiert durch Eingabeinstrumente, wie die Pedale oder das Lenkrad. Das Durchführen weiterer Aufgaben würde den Immersionsgrad für den Probanden reduzieren. Deshalb interagiert in einer zweiten Rolle der Versuchsleiter mit der Simulation. Dieser übernimmt die übergeordnete Steuerung der Simulation. Um den Rechnerverbund aus Bild 5-32 ortsungebunden und zentral steuern zu können, verfügt Hyperion über eine Remote-Applikation und erfüllt somit die Anforderung **A13** (Remote-Bedienbarkeit).

Der Master- und sämtliche Slave-Rechner stellen Netzwerkschnittstellen bereit, über welche die Remote-Applikation Anweisungen verteilen kann. Gleichzeitig teilen alle Rechner der Remote-Applikation ihre Zustände mit. Der Versuchsleiter erhält somit eine zentrale Kontrollmöglichkeit über die verteilte Simulation.

Der Funktionsumfang der Remote-Applikation kann in drei Gruppen untergliedert werden. Für jede Gruppe steht innerhalb der Applikation eine eigene Ansicht zur Verfügung. Die allgemeine Rechner-Steuerung bildet die erste Gruppe. Ihre Ansicht in der Remote-Applikation wird in Bild 5-33 dargestellt.

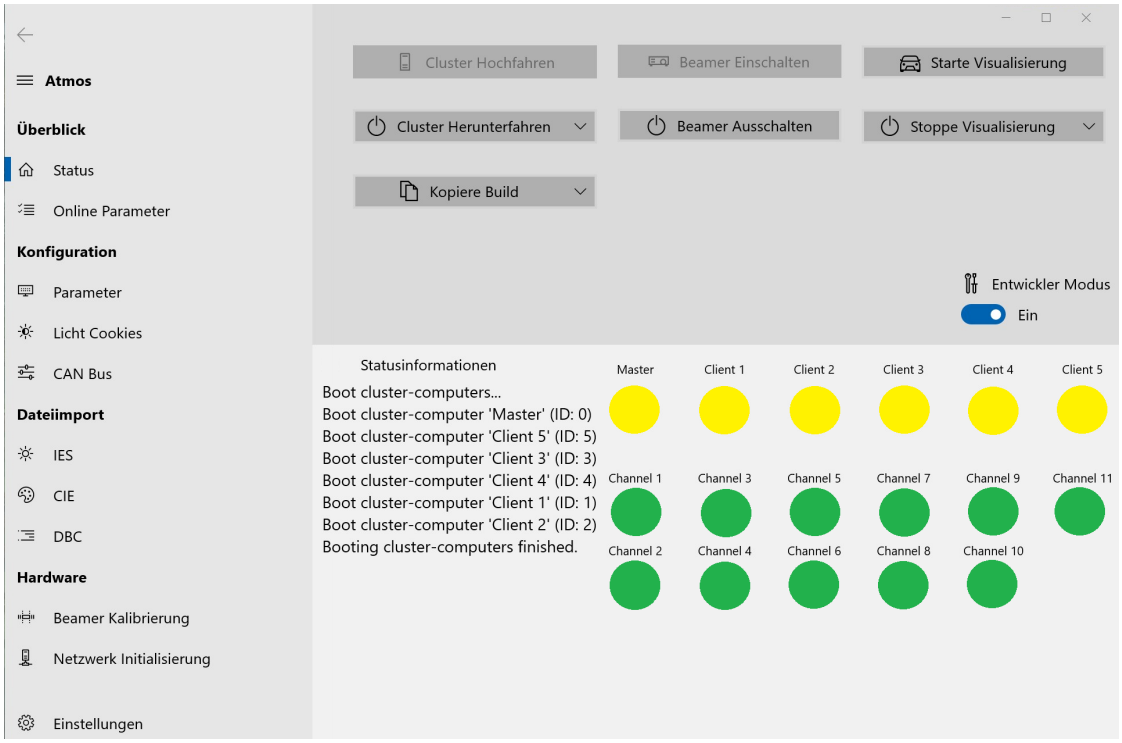


*Bild 5-33*: *Allgemeine Statusanzeige und Steuerung von Hyperion in der Remote-Applikation.*

Im oberen Bereich befinden sich die Aktionsmöglichkeiten, während der untere Bereich Informationen der einzelnen Rechner des Clusters zurückliefert. Innerhalb der Rechnersteuerung können alle Rechner hoch- und heruntergefahren werden. Genauso lassen sich die netzwerkfähigen Ausgabegeräte ein- und ausschalten. Nach dem Starten der Rechner besteht schließlich die Möglichkeit Hyperion zu laden. Als weitere Funktion können neue Versionen von Hyperion automatisiert auf allen Rechner des Clusters installiert werden.

Der untere Bereich aus Bild 5-33 umfasst die Rückmeldung an den Versuchsleiter. Rechts werden die Zustände aller Rechner und Ausgabegeräte durch Statusampeln gekennzeichnet. Er kann so auf einen Blick den Zustand des Gesamtsystems erfassen. Zusätzlich findet sich links eine chronologische Auflistung aller Statusmeldungen, wodurch alle vom Versuchsleiter eingeleiteten Aktionen durch ein Feedback jedes einzelnen Cluster-Rechners quittiert werden können.

In der zweiten Ansicht kann die Offline-Parametrierung vorgenommen werden. Dort stehen dem Versuchsleiter alle Parameter, die Hyperion zur Manipulation freigibt, zur Verfügung. Im Kontext der Offline-Parametrierung sind aber vor allem diejenigen Parameter von Interesse, die zur Laufzeit nicht mehr angepasst werden können. Dazu gehören beispielsweise die zu befahrende Strecke oder das verwendete Scheinwerfersystem. Der Versuchsleiter kann die Parametrierung in dieser Ansicht komfortabel vornehmen. Die Parameter

sind entsprechend ihrer Semantik gruppiert. Außerdem stellen entsprechende Eingabemasken sicher, dass die eingegebenen Werte im Gültigkeitsbereich des entsprechenden Parameters liegen. Nach Fertigstellung der Offline-Parametrierung können die Parameter durch einen Mausklick an alle Rechner übertragen werden. Außerdem ist es möglich, die Wertekonfiguration zu speichern und zu einem späteren Zeitpunkt erneut zu laden. Mit den genannten Funktionen wird ein wichtiger Beitrag zur Erfüllung der Anforderung **A12** (Reproduzierbarkeit und Laufzeitanpassung) geleistet. Eine Momentaufnahme der Initialisierungsansicht wird in Bild 5-34 dargestellt.

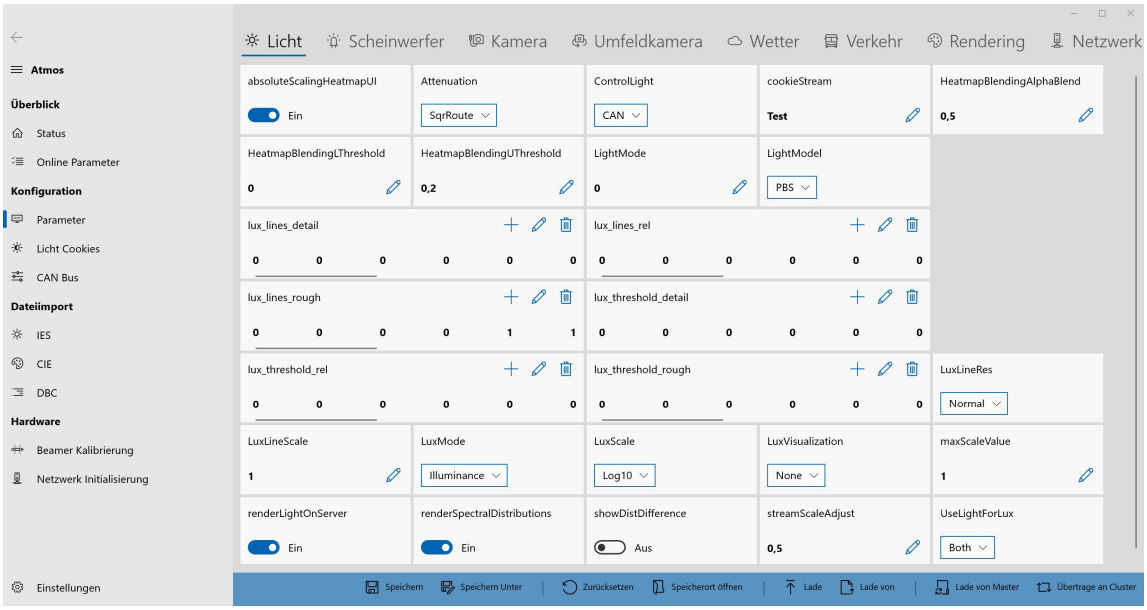


*Bild 5-34*: *Fernparametrierung von Hyperion über die Remote-Applikation.*

Während sich die zweite Ansicht zur Initialisierung der Simulation eignet und somit vor dem Start von Hyperion herangezogen wird, eignen sich die in der dritten Ansicht aufgeführten Online-Parameter zur Laufzeitmanipulation. Dementsprechend zeigt diese Ansicht auch nur eine Teilmenge der insgesamt zur Verfügung stehenden Parameter. Strukturell entspricht die dritte Ansicht jedoch weiterhin der in Bild 5-34 dargestellten. Manipulationen einzelner Online-Parameter werden von der Remote-Applikation ausschließlich an den Master geleitet. Dieser reicht die Anpassung an alle Slaves weiter. Die Unterstützung der Online-Parameter stellt ebenfalls einen Beitrag zur Erfüllung der Anforderung **A12** (Reproduzierbarkeit und Laufzeitanpassung) dar.

## 5.8 Konfigurationen

Wie aus den zurückliegenden Abschnitten hervorgeht, verfügt Hyperion über eine Skalierbarkeit, die Einsätze vom Arbeitsplatz-Rechner bis hin zum Großsimulator ermöglicht. Diese Skalierbarkeit gelingt, indem die abstrakten Funktionsbausteine der Gesamtsimulation, wie sie in Bild 5-1 zu sehen sind, in unterschiedlichen technischen Abstufungen realisiert werden können. Um der Anforderung **A11** (Konfigurierbarkeit) gerecht zu werden, werden in diesem letzten Abschnitt des Kapitels beispielhaft drei Konfigurationen vorgestellt, die sich im Rahmen der Entwicklung als besonders geeignet erweisen. Sie

werden nachfolgend als „Desktop-“, „Mini-Simulator-“ und „Großsimulator“-Varianten bezeichnet.

Die Desktop-Variante stellt die Minimalkonfiguration dar. Sie kann an jedem Arbeitsplatz zur Anwendung kommen, sofern die verfügbare Grafikleistung ausreicht. Wie aus Bild 5-35 hervorgeht, genügt eine normale Arbeitsplatzausstattung als Hardware-Setup. Diese Eigenschaft erlaubt einen breiten Einsatz in der Entwicklung. Aufgrund des Verzichts auf geeignete Eingabeinstrumente empfiehlt es sich, fremdgesteuerte Fahrmodi zu verwenden. Der Anwender kann sich somit ausschließlich auf das Scheinwerferlicht konzentrieren. Die MiL-Einbindung des Steuergeräts erlaubt schnelle Anpassungen, welche aufgrund der Fremdsteuerung unter idealen Bedingungen verglichen werden können. Insbesondere in frühen Phasen der Entwicklung, in denen Machbarkeitsstudien und grobe Vorauslegungen stattfinden, stellt diese Realisierung von Hyperion ein adäquates, kostengünstiges und zeiteffizientes Mittel dar.

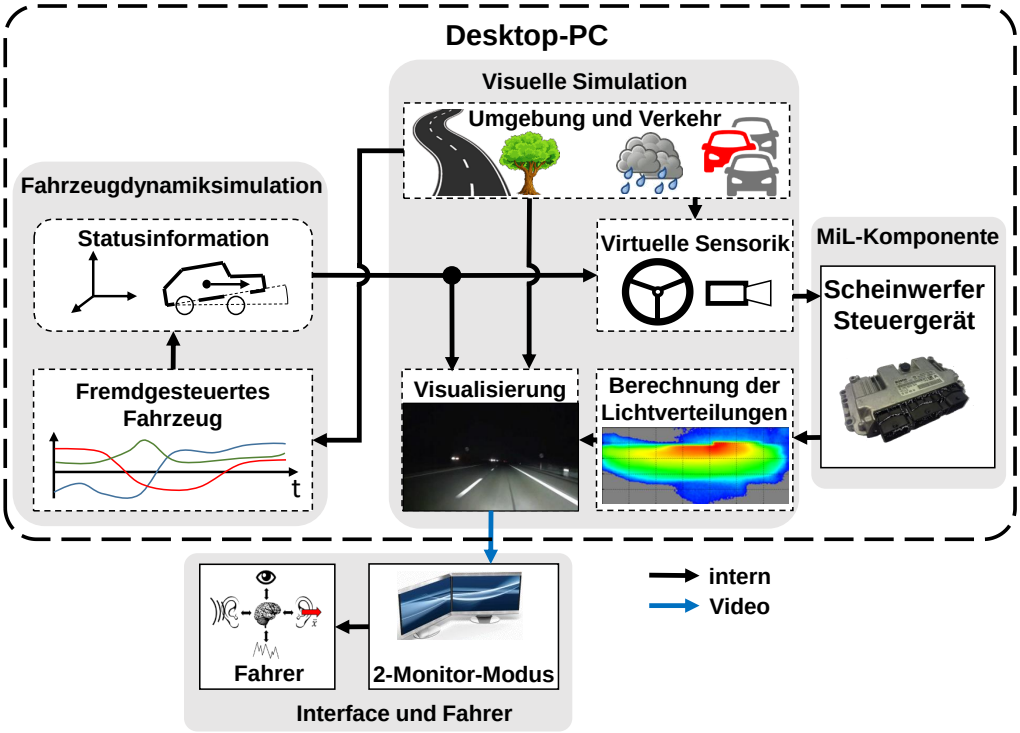


*Bild 5-35*: *Beispielkonfiguration „Desktop“ zum Betrieb von Hyperion an normalen Arbeitsplätzen.*

Auch wenn die Desktop-Variante aufgrund ihrer niedrigen Anforderungen sehr breit eingesetzt werden kann, erlaubt sie aufgrund der fehlenden Interaktion mit einem menschlichen Fahrer keine vollständige Testabdeckung. Die direkte Reaktion des Scheinwerfersystems auf Fahreingaben und die subjektive Wahrnehmung eines Lichtexperten stellen wesentliche Kriterien bei der Bewertung eines Scheinwerfersystems dar. Derartige Erprobungen sind nur in einem Simulator-Umfeld möglich.

Da Großsimulatoren erhebliche Kosten verursachen und ihre Verfügbarkeit begrenzt ist, erlaubt Hyperion mit der Mini-Simulator-Konfiguration den Betrieb einer kostengünstigen Alternative. Anders als in Großsimulatoren wird hierbei auf ein Bewegungssystem ver-

zichtet. Als Eingabeinstrumente dienen Lenkrad-Pedal-Kombinationen aus dem Gaming-Bereich. Zusätzlich sollte der Mini-Simulator über drei Monitore verfügen, um die visuelle Immersion hinreichend sicherzustellen. Da eine Echtzeithardware ebenfalls hohe Kosten verursacht und die Anbindung der Eingabeinstrumente an diese zusätzlichen Aufwand darstellt, schlägt die in Bild 5-36 vorgestellte Konfiguration die Verwendung des internen Fahrzeugmodells vor. Der Betrieb des ASM Modells auf einer Echtzeithardware ist alternativ möglich. In der visualisierten Konfiguration ist das Steuergerät als HiL-Komponente eingebunden. Sollte zum Zeitpunkt der Erprobung noch kein Prototyp vorhanden sein, kann dieses analog zu Bild 5-35 im MiL-Betrieb eingebunden werden.

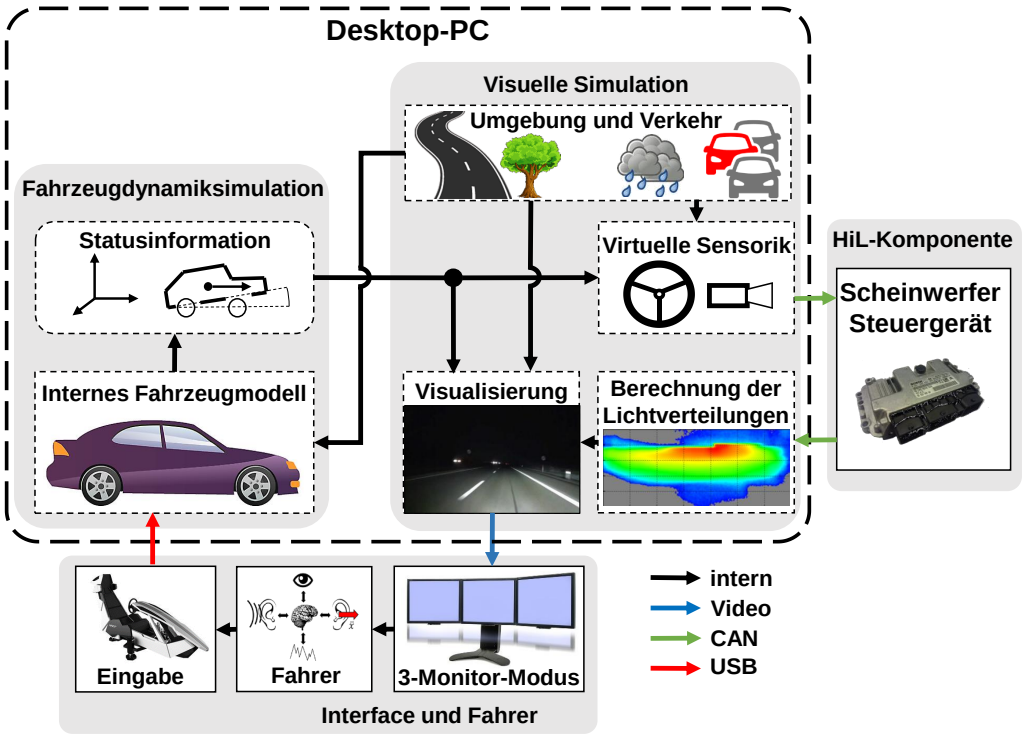


*Bild 5-36*: *Beispielkonfiguration „Mini-Simulator“ zum Betrieb von Hyperion an kleinen Fahrsimulatoren.*

Die in Bild 5-37 dargestellte Konfiguration entspricht der im ATMOS Fahrsimulator verwendeten. Sie ist aber in gleicher Weise auf andere Großsimulatoren übertragbar. Der Hardware-Aufwand ist erheblich größer, als es für den Mini-Simulator der Fall ist. Gleichzeitig ist jedoch auch die Immersion wesentlich höher.

In der Großsimulator-Konfiguration sind ein Rechencluster, die Echtzeithardware, das reale Steuergerät, ein in der Regel aus mehreren Einheiten bestehendes Ausgabesystem und ein typischerweise als Fahrzeug-Mockup realisiertes Eingabesystem die wesentlichen Hardware-Komponenten. Die Anzahl der Rechner im Cluster orientiert sich an der Anzahl der Ausgabegeräte und an ihrer Auflösung. Im ATMOS Fahrsimulator werden der Master- und fünf Slave-Rechner betrieben.

Neben dem Hardware-Aufwand entsteht auch ein personeller Aufwand, da mindestens eine weitere Person in der Rolle des Versuchsleiters involviert ist. Der Versuchsleiter

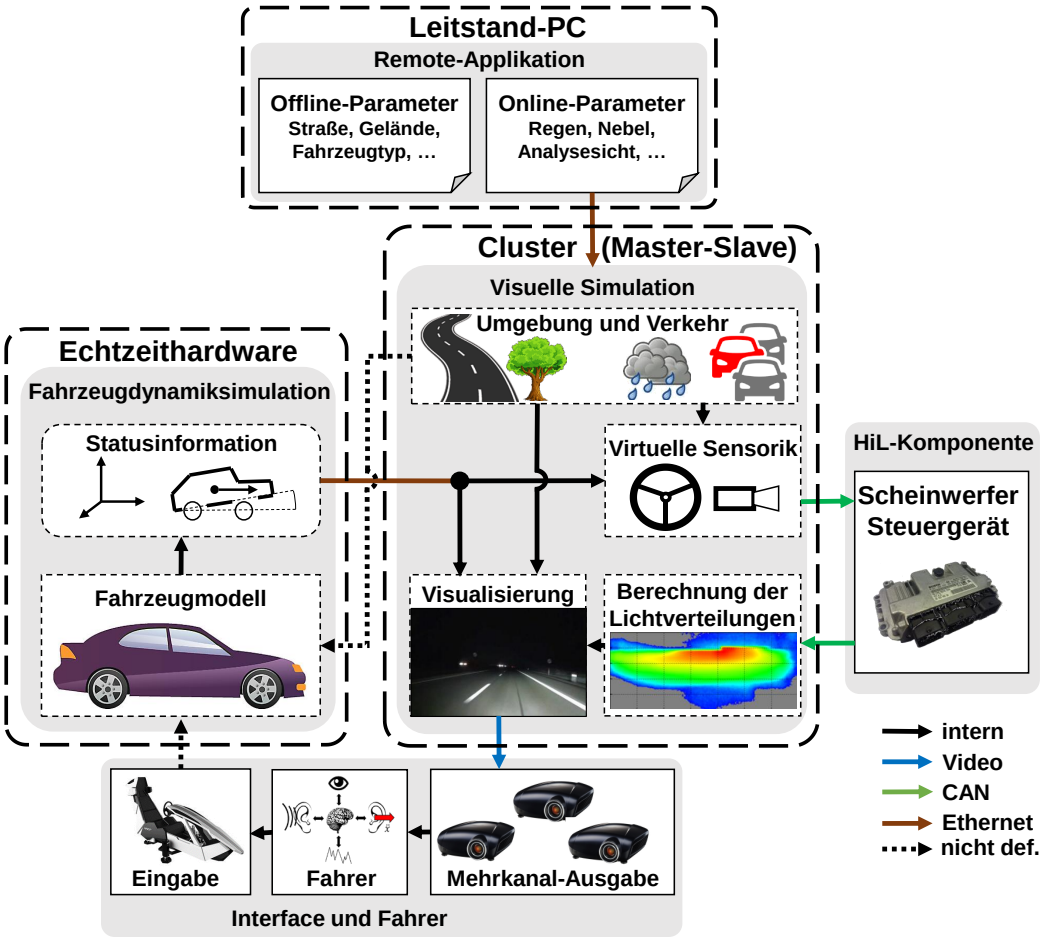


*Bild 5-37*: *Beispielkonfiguration „Großsimulator“ zum Betrieb von Hyperion an Großsimulatoren.*

übernimmt die administrative Steuerung der Simulation durch die Remote-Applikation um den Probanden zu entlasten und dessen Immersion zu erhöhen.

# 6 Rendering von HD-Scheinwerferlicht

Das nun folgende Kapitel stellt gemeinsam mit Kapitel 7 den Forschungskern der vorliegenden Arbeit dar. Es beinhaltet die Vorstellung des methodischen Vorgehens und der technischen Implementierung zur Simulation des Lichts von Pixel-Scheinwerfersystemen. Nach der Vorwegnahme einiger Designentscheidungen durch Abschnitt 5.4.3, wird in Abschnitt 6.1 zunächst die Beschaffenheit eines Datensatzes zur Spezifikation eines HD-Scheinwerfersystems vorgestellt.

Die Simulation des Lichts erfolgt in zwei Stufen. Zuerst wird die von den momentan vorliegenden Dimmwerten abhängige Gesamtlichtverteilung des Scheinwerfers ermittelt. Das hierzu notwendige Vorgehen wird in Abschnitt 6.2 eingeführt. Mit Abschnitt 6.3 schließt sich die Implementierung der Lichtquelle an, mit welcher die ermittelte Gesamtlichtverteilung in die Szene projiziert wird. Das erarbeitete Verfahren wird abschließend validiert (Abschnitt 6.4) und hinsichtlich seiner Laufzeit diskutiert (Abschnitt 6.5). In Abschnitt 6.6 wird die Eignung der Lösung zur physikalisch motivierten Nachbildung von Witterungsbedingungen am Beispiel von Nebel nachgewiesen.

## 6.1 Datensatz eines HD-Scheinwerfersystems

Die dynamische Lichtverteilung eines HD-Scheinwerfers kann durch die Gesamtheit der Lichtverteilungen all seiner Lichtquellen und die zum aktuellen Zeitpunkt vorliegenden Dimmwerte aller Lichtquellen vollständig beschrieben werden. Diese Aussage gilt zunächst nur für additive Systeme mit einzelnen Lichtquellen, wie Matrix- oder SSL|HD-Systeme (siehe Tabelle 3-1). Sie kann auf Scanner-Systeme erweitert werden, wenn man jede diskrete Position des MEMS als einzelne Lichtquelle interpretiert (siehe Abschnitt 3.1.3). Auf ähnliche Weise lassen sich subtraktive Systeme in gleicher Weise datentechnisch darstellen, indem man jedes diskrete Ausblendungselement als eigene Lichtquelle auffasst. Am Beispiel eines LCD-Scheinwerfers entspricht die Anzahl der fiktiven Lichtquellen somit der Pixelzahl des LCD-Moduls. Zusammenfassend zeigt sich, dass die datentechnische Beschreibung eines HD-Scheinwerfers durch die Gesamtheit seiner Einzellichtverteilungen als universell angesehen werden kann. Hyperion verwendet diese Repräsentation des Scheinwerfers zur bestmöglichen Erfüllung der Anforderung **A3** (Technologie-Unabhängigkeit).

Die grundsätzliche Gestalt von Lichtverteilungen wurde bereits in Abschnitt 3.1.1 diskutiert. Dort wurden sowohl monochrome Lichtstärkeverteilungen als auch spektrale Lichtverteilungen beschrieben. Der wesentliche Unterschied ist, dass im monochromen Fall die Lichtstärke als eindimensionaler Wert und im spektralen Fall der dreidimensionale Farbvektor für jedes Winkelpaar $(\theta, \varphi)$ aufgezeichnet wird. Dabei wird der Farbvektor typischerweise durch die X-, Y- und Z-Koordinaten gemäß der CIE beschrieben (siehe Abschnitt 2.2.6). Nach Anforderung **A2** (Spektrale Lichtverteilungen) soll Hyperion beide Varianten unterstützen. Zur Sicherstellung eines einheitlichen Workflows werden Lichtstärkeverteilungen in spektrale Lichtverteilungen überführt. Dazu wird der Lichtstärkewert als Y-Koordinate des spektralen Pendants übernommen, während die Farbart der gesamten

Lichtverteilung, spezifiziert durch die X- und Z-Koordinaten, durch den Anwender gewählt werden kann. Lokale Farbunterschiede finden sich in der so erzeugten Lichtverteilung erwartungsgemäß nicht.

Technisch liegen die Lichtverteilungen im monochromen Fall als Dateien im IES-Format (Illuminating Engineering Society) vor [ANS01]. Neben den eigentlichen Lichtstärkewerten und den Winkelpaaren, auf die sie sich beziehen, enthält eine IES-Datei weitere Header-Informationen. Hierzu gehören zum Beispiel die Anzahl der Messschritte in horizontaler und vertikaler Richtung oder der Bezeichner dieser Messung. Im Anhang A2.1 findet sich ein Ausschnitt einer Beispieldatei im IES-Format. Spektrale Lichtverteilungen liegen stattdessen als Textdateien vor. Sie weisen jedoch ein ähnliches Schema auf. Zuerst wird im Header spezifiziert, wie die Polachse liegt, um die Bedeutung von Polar- und Azimutwinkel zu definieren. Weiterhin wird spezifiziert, dass die Farben nach dem Standard der CIE im XYZ-Farbraum (s. Abschnitt 2.2.6) vorliegen. Abschließend erfolgt die Benennung des vermessenen Winkelbereichs und der Auflösung. Nach dem Header folgen die eigentlichen Daten, wobei die Farbkoordinaten X, Y und Z über alle Messpunkte hinweg aneinander gereiht sind. Die Zählfolge der Messpunkte ist parametrierbar. Wichtig ist, dass sie beim Speichern und Laden der Daten in gleicher Weise eingehalten wird. Der Anfangsbereich einer CIE-Datei kann in Anhang A2.2 eingesehen werden.

Da ein HD-Scheinwerfer über eine Vielzahl von Lichtquellen verfügt, ist ein vollständiger Satz der Lichtverteilungen aller Einzellichtquellen notwendig, um einen derartigen Scheinwerfer zu beschreiben. Der Datensatz eines HD-Scheinwerfers verfügt somit über eine Vielzahl von IES- bzw. CIE-Dateien. Die Zuordnung der Lichtverteilungen zu den Lichtquellen im Scheinwerfer und damit zu den Dimmwerten, die vom Steuergerät vorgegeben werden, erfolgt in den zur Verfügung stehenden Daten über die Dateinamen. Eine Hinterlegung im Dateiheader wäre ebenfalls möglich und erscheint robuster. Darüber hinaus werden die Einzellichtquellen nicht zwangsläufig über den gleichen Winkelbereich vermessen. Im Fall des zur Erprobung verwendeten HD84-Systems liegen beispielsweise die Messdaten der Matrix im Winkelintervall $[-6°, 6°] \times [-25°, 25°]$ vor, während die Vorfeldleuchten im Bereich $[-30°, 15°] \times [-90°, 90°]$ vermessen wurden. Die Messbereiche orientieren sich an den unterschiedlichen Bereichen der Einflussnahme der verschiedenen Lichtquellen. Um maximale Kompatibilität und die Erfüllung der Anforderung **A3** sicherzustellen, erlaubt Hyperion neben variierenden Vermessungsbereichen auch unterschiedliche Auflösungen der Messungen.

## 6.2 Bestimmung der Gesamtlichtverteilung

In Abschnitt 5.4.3 wurde die Hyperion-Lichtquelle innerhalb der Gesamtarchitektur eingeordnet. Nun erfolgt die Beschreibung der zugrunde liegenden Methodik und der technischen Implementierung. Dieser Abschnitt bezieht sich auf die Bestimmung der Gesamtlichtverteilung, während Abschnitt 6.3 die zweite Stufe des Licht-Renderings erläutert.

Die in Abschnitt 6.1 beschriebenen Lichtverteilungen stellen die zeitinvariante Komponente der datentechnischen Repräsentation eines HD-Scheinwerfers dar. Dynamik erhält die Gesamtlichtverteilung eines solchen Scheinwerfers, indem die Einzellichtverteilungen auf oder abgedimmt werden. Auf diese Weise fügen sich die Einzellichtverteilungen in unterschiedlicher Weise zur Gesamtlichtverteilung zusammen und erlauben eine bausteinartige

Gestaltung des Lichts. Technisch wird die Dimmstufe durch das Scheinwerfersteuergerät vorgegeben. Der nachfolgende Unterabschnitt beschreibt genauer, wie die Gesamtlichtverteilung eines Scheinwerfers basierend auf den zeitinvarianten Einzellichtverteilungen und den dynamischen Dimmwerten ermittelt werden kann. Dabei wird zunächst die intuitive Vorgehensweise der Ermittlung vorgestellt. Anschließend wird in Unterabschnitt 6.2.3 die tatsächliche Implementierung des Verfahrens in Hyperion beschrieben. Hierbei wurde zugunsten der Anforderungen **A4** (Skalierbarkeit der Pixelanzahl) und **A6** (Echtzeitfähigkeit) erheblich von der intuitiven Vorgehensweise abgewichen.

### 6.2.1 Funktionsprinzip

Die von einzelnen Pixeln eines HD-Scheinwerfers ausgeleuchteten Winkelbereiche sind so gestaltet, dass sie den insgesamt ausgeleuchteten Bereich durch eine Rasterstruktur ausfüllen. Bei der Auslegung der Lichtverteilung einzelner Pixel ist ein Kompromiss zwischen der scharfen Abgrenzung zu den benachbarten Pixeln und sanften Ausläufen im Randbereich zu treffen. Während die erstgenannte Eigenschaft zusammen mit der Auflösung des Systems die Genauigkeit bestimmt, stellen sanfte Ausläufe eine homogene Lichtverteilung ohne störende Artefakte im Sichtfeld sicher. Das Bild 6-1 visualisiert den Ausleuchtungsbereich einzelner Pixel der Matrix und deren Komposition zu verschiedenen Gesamtlichtverteilungen am Beispiel des HD84-Systems. Beim HD84-System handelt es sich im Vergleich zu den modernsten Technologien um ein niedrig aufgelöstes Scheinwerfersystem. Zur Anschauung ist es jedoch gut geeignet und dient deshalb als Anwendungsbeispiel. Das nachfolgend beschriebene Verfahren gilt in gleicher Weise für hoch aufgelöste Scheinwerfersysteme.

Im oberen Bereich des Bildes 6-1 werden die LED der HD84-Matrix entsprechend ihrer Einbauanordnung im Scheinwerfer visualisiert. Die darin befindliche Nummerierung entspricht der Indexierung der 84 Pixellichter. Die verschiedenen Zeilen verfügen über unterschiedlich große Reichweiten. Die untere Zeile (1) dient ausschließlich der Ausleuchtung des Nahbereichs vor dem Fahrzeug, während die mittlere Zeile (2) für die Reichweite des Abblendlichts ausreicht. Die obere Zeile (3) wird hingegen nur bei aktivem Fernlicht eingesetzt. Dementsprechend liegt der Schwerpunkt der Lichtstärkeverteilung im positiven Bereich des Polarwinkels $\theta$. Genau wie die vertikale Anordnung steht auch die horizontale Anordnung der Lichtquellen im direkten Bezug zum ausgeleuchteten Winkelbereich. Auf der linken Seite angeordnete LED strahlen zum Beispiel primär auf die linke Fahrbahnseite ($\varphi < 0$). Zur besseren Nachvollziehbarkeit sind die Lichtstärkeverteilungen der LED 1, 45 und 84 im mittleren Bereich des Bildes 6-1 dargestellt. Der Zusammenhang zwischen Einbaulage und Ausleuchtungsbereich ist klar erkennbar. Außerdem wird deutlich, dass eine einzelne LED exklusiv ein kleines Winkelsegment der Gesamtlichtverteilung bestrahlt. Diese Eigenschaft eines HD-Systems bildet die Grundlage zur freien Gestaltung der Gesamtlichtverteilung. Am Beispiel des HD84-Systems stellt man fest, dass LED 45 einen wesentlich kleineren Bereich ausleuchtet, als es für die in den Randzonen angeordneten LED 1 und 84 der Fall ist. Dieses Design ist den richtungsabhängig variierenden Anforderungen an die Auflösung geschuldet. Beispielsweise muss das Scheinwerfersystem zur Ausblendung von Gegenverkehr im Zentralbereich schärfer als für ein Kurvenlicht im Seitenbereich auflösen.

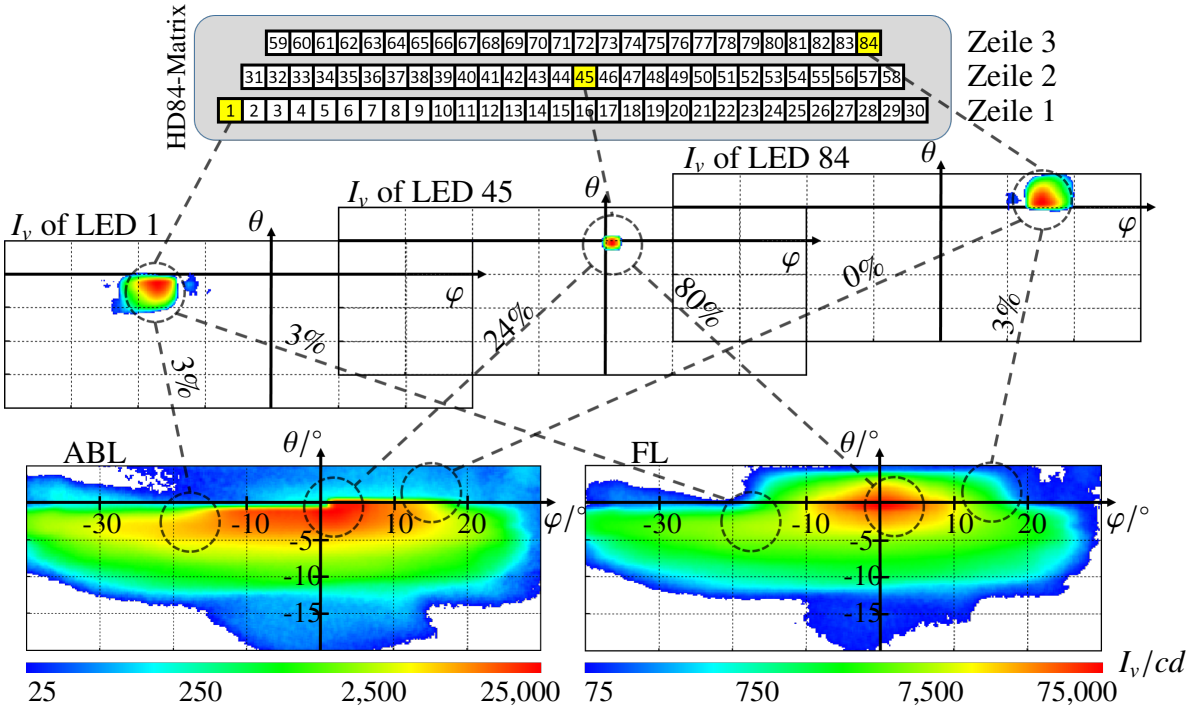


*Bild 6-1*: *Lichtstärkeverteilungen einzelner Pixel und ihre mit Dimmwerten gewichtete Komposition zur Gesamtlichtstärkeverteilung (links: Abblendlicht, rechts: Fernlicht).*

Die Lichtstärkeverteilungen der einzelnen Pixel werden bei maximaler Bestromung aufgezeichnet. Während des Betriebs kann jede Pixellichtquelle quasikontinuierlich im Bereich von 0-100% gedimmt werden, indem die zugeführte Leistung durch Pulsweitenmodulation (PWM) angepasst wird. Im unteren, linken Bereich des Bildes 6-1 ist die Abblendlichtverteilung des HD84-Systems dargestellt. Sie entsteht durch die geeignete Bestromung der Pixellichter. Beispielhaft sind die Beiträge der Pixel 1, 45 und 84 zur Gesamtlichtverteilung visualisiert. Während LED 1 (3%) und 45 (24%) eingeschaltet sind, trägt LED 84 nicht zum Abblendlicht bei. Im Vergleich dazu findet sich auf der rechten Seite die Fernlichtverteilung des HD84-Systems. In diesem Fall ist auch LED 84 (3%) aktiv. Ebenso wird der Strom der LED 45 im Vergleich zum Abblendlicht angehoben um eine höhere Lichtstärke zu erzielen, da die Fernlichtverteilung im Schwerpunkt etwa dreimal stärker als das Abblendlicht strahlt. Generell fällt auf, dass die LED 1 und 84 mit nur 3% sehr schwach betrieben werden. Diese Leistungsreserven sind im Randbereich nötig, da wesentlich höhere Werte erzielt werden müssen, wenn beispielsweise die Kurvenlichtfunktion den Lichtschwerpunkt horizontal verschiebt.

### 6.2.2 Formalismus

Aus den vorangegangenen Erläuterungen geht das Funktionsprinzip hervor. Zur genauen Berechnung der Gesamtlichtverteilung eines Scheinwerfers gilt es, die angesprochenen Merkmale eines HD-Scheinwerfers geeignet zu formalisieren. Darauf aufbauend kann die Gesamtlichtverteilung als Rechenvorschrift formuliert werden.

Dazu sei die Anzahl der Pixellichtquellen nachfolgend mit $K$ bezeichnet. Zusatzlichtquellen, wie Vorfeld- und Kurvenleuchten oder Zusatzfernlichter, können im beschriebenen Formalismus ebenfalls als Pixellichtquellen behandelt werden, obwohl sie nicht Teil des HD-Moduls sind. Für jede Lichtquelle $k \in \{1, ..., K\}$ kann die Lichtverteilung in diskretisierter Form als zweidimensionales Array $L_k$ abgebildet werden. Die Diskretisierung ist kein eigener Zwischenschritt im gesamten Verfahren. Sie ergibt sich durch die Messung oder modellbasierte Berechnung inhärent. Darüber hinaus ist die Information über den Vermessungsbereich der Lichtquelle $k$ unverzichtbar. Er kann durch die unteren und oberen Grenzen der Polar- und Azimutwinkel $\underline{\theta}_k, \bar{\theta}_k, \underline{\varphi}_k, \bar{\varphi}_k$ beschrieben werden, welche in Bild 6-2 visualisiert sind. Aus der Zeilenzahl $M_k$, der Spaltenzahl $N_k$ und den Winkelgrenzen ergibt sich die Auflösung der diskretisierten Lichtverteilung $L_k$. Die horizontale Schrittweite beträgt $\Delta\varphi = \frac{\bar{\varphi}_k - \underline{\varphi}_k}{N_k - 1}$ und die vertikale Schrittweite beträgt $\Delta\theta = \frac{\bar{\theta}_k - \underline{\theta}_k}{M_k - 1}$. Diese Schrittweiten werden für alle Lichtverteilungen als einheitlich angenommen. Sind sie es nicht, wird die Einheitlichkeit durch eine vorgelagerte bilineare Interpolation herbeigeführt. Der Eintrag $l_k(m, n)$ aus $L_k$, welcher sich in Zeile $m$ und Spalte $n$ befindet, entspricht dem Wert der Lichtverteilung der Lichtquelle $k$ für das Winkelpaar $(\theta, \varphi) = (\underline{\theta}_k + m \cdot \Delta\theta, \underline{\varphi}_k + n \cdot \Delta\varphi)$. Auch die Bedeutung dieses Bezeichners wird in 6-2 veranschaulicht. Um spektrale Lichtverteilungen vollständig abbilden zu können, repräsentieren die Einträge $l_k \in \mathbb{R}^3_{\geq 0}$ Farbvalenzen. Werden monochrome Lichtstärkeverteilungen verarbeitet, wird nur ein Eintrag des Farbvalenzvektors benötigt. Zum Erhalt eines einheitlichen Verfahrens wird die vektorielle Struktur der Einträge aus $L_k$ dennoch beibehalten.

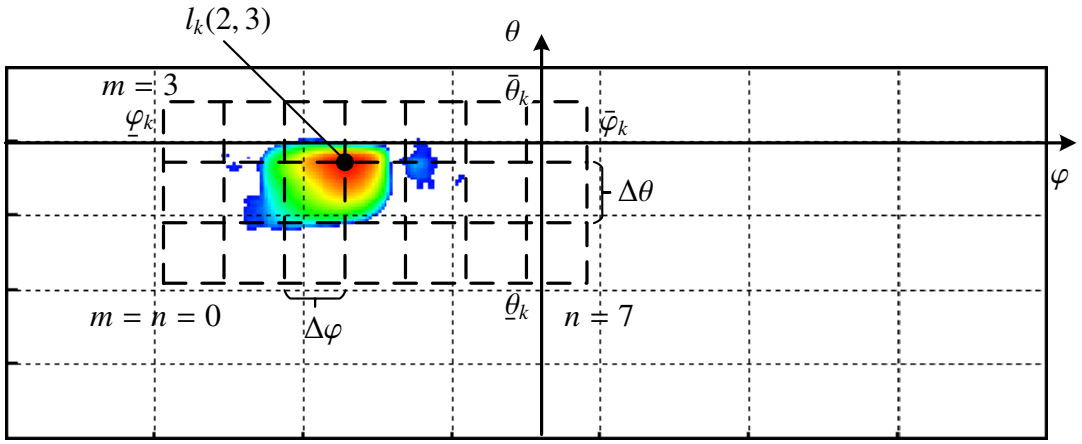


*Bild 6-2*: *Diskretisierung einer kontinuierlichen Lichtverteilung durch ein zweidimensionales Array.*

Mit den beschriebenen Größen kann jede Lichtquelle des Scheinwerfers hinsichtlich ihrer statischen Eigenschaften spezifiziert werden. Die Menge der Daten skaliert mit der Anzahl der Lichtquellen, den Größen der Vermessungsbereiche und den Diskretisierungsschrittweiten innerhalb der Lichtverteilung. Nachfolgend wird angenommen, dass das Diskretisierungsraster bezüglich der Polar- und Azimutwinkel über alle Lichtverteilungen des Scheinwerfers hinweg einheitlich ist. Formal wird dadurch vorausgesetzt, dass die horizontalen und vertikalen Winkeldifferenzen zweier Messpunkte $(\theta_1, \varphi_1)$ und $(\theta_2, \varphi_2)$ stets ganzzahlig durch die Diskretisierungsschrittweiten $\Delta\theta$ bzw. $\Delta\varphi$ teilbar sind. Das gilt

insbesondere dann, wenn diese Messpunkte aus verschiedenen Lichtverteilungen stammen. Wird diese Forderung durch den zur Verfügung stehenden Datensatz nicht erfüllt, können die Daten durch bilineare Interpolation in das gewünschte Diskretisierungsraster überführt werden.

Alle bisher eingeführten Größen sind bezogen auf das betrachtete Scheinwerfersystem unveränderlich. Die dynamische Komponente des Scheinweferlichts wird durch die zeitveränderlichen Dimmwerte geleistet. Der Dimmwert der Lichtquelle $k$ sei durch $d_k(t) \in [0, 1]$ bezeichnet. Dieser stellt einen globalen Skalierungsfaktor für alle Einträge des Arrays $L_k$ dar. Die momentan vorliegende, diskretisierte Lichtverteilung der Lichtquelle $k$ ergibt sich somit zu

$$L_k(t) = d_k(t) \cdot L_k. \tag{6-1}$$

In Gleichung (6-1) kann $L_k$ als Matrix aufgefasst werden. Die Multiplikation mit dem skalaren Dimmwert $d_k(t)$ ist dann entsprechend der üblichen Rechenregeln komponentenweise durchzuführen.

Basierend auf den genannten Definitionen und Annahmen kann die Gestalt der Gesamtlichtverteilung $L_\sum$ des Scheinwerfers beschrieben werden. Der Messbereich der Gesamtlichtverteilung soll der kleinst mögliche sein, der alle Einzellichtverteilungen vollständig enthält. Die Winkelgrenzen ergeben sich deshalb gemäß

$$\begin{aligned} \underline{\theta} &= \min_{k \in 1,...,K} \underline{\theta}_k, \quad \bar{\theta} = \max_{k \in 1,...,K} \bar{\theta}_k \text{ und} \\ \underline{\varphi} &= \min_{k \in 1,...,K} \underline{\varphi}_k, \quad \bar{\varphi} = \max_{k \in 1,...,K} \bar{\varphi}_k. \end{aligned} \tag{6-2}$$

Die Auflösung innerhalb dieses Bereichs entspricht der Auflösung aller Einzellichtverteilungen, sodass sich die Zeilen- und Spaltenzahlen von $L_\sum$ zu

$$M_\sum = \frac{\bar{\theta} - \underline{\theta}}{\Delta\theta} + 1 \text{ und } N_\sum = \frac{\bar{\varphi} - \underline{\varphi}}{\Delta\varphi} + 1$$

ergeben. Die Einträge $l_\sum(m, n)$ mit $m \in \{1, ..., M_\sum\}, n \in \{1, ..., N_\sum\}$ der Gesamtlichtverteilung ergeben sich durch die Komposition der momentanen Einzellichtverteilungen $L_k(t)$ mit $k \in 1, ..., K$ und sind somit ebenfalls zeitveränderlich. Zur Herleitung von $L_\sum$ wird zunächst die bereichsweise Matrizenaddition eingeführt. Sie wird fortlaufend mit dem Operator $+_\square$ bezeichnet. Zur Definition der bereichsweisen Addition müssen im Vorhinein die Zeilen- und Spaltengültigkeitsbereiche einer Matrix eingeführt werden. Eine Matrix $A$, welche über die Zeilenindizes $\underline{m}_A$ bis $\bar{m}_A$ und die Spaltenindizes $\underline{n}_A$ bis $\bar{n}_A$ gültig ist, wird im Folgenden als

$$A[\underline{m}_A, \bar{m}_A, \underline{n}_A, \bar{n}_A]$$

notiert. $A$ muss in diesem Beispiel $\bar{m}_A - \underline{m}_A + 1$-viele Zeilen und $\bar{n}_A - \underline{n}_A + 1$-viele Spalten aufweisen. Die bereichsweise Addition zweier Matrizen $A$ und $B$ kann nun wie folgt notiert werden:

$$C[\underline{m}_C, \bar{m}_C, \underline{n}_C, \bar{n}_C] = A[\underline{m}_A, \bar{m}_A, \underline{n}_A, \bar{n}_A] +_\square B[\underline{m}_B, \bar{m}_B, \underline{n}_B, \bar{n}_B]. \tag{6-3}$$

Für den Gültigkeitsbereich der Matrix $C$ gilt

$$\begin{aligned} \underline{m}_C &= \min\{\underline{m}_A, \underline{m}_B\}, \\ \bar{m}_C &= \max\{\bar{m}_A, \bar{m}_B\}, \\ \underline{n}_C &= \min\{\underline{n}_A, \underline{n}_B\} \text{ und} \\ \bar{n}_C &= \max\{\bar{n}_A, \bar{n}_B\}. \end{aligned} \tag{6-4}$$

Die Konstruktion von $C$ erfolgt durch die vorhergehende Anpassung der Summanden $A$ und $B$. Diese Anpassung basiert auf den Resultaten der Minimums- und Maximumsfunktionen nach Gleichung (6-4). Für den Fall $\underline{m}_A < \underline{m}_B$ wird $B$ um $\underline{m}_B - \underline{m}_A$-viele Nullzeilen nach unten erweitert. Gilt hingegen $\underline{m}_A > \underline{m}_B$ wird $A$ um $\underline{m}_A - \underline{m}_B$-viele Nullzeilen nach unten erweitert. Im Fall $\underline{m}_A = \underline{m}_B$ müssen weder $A$, noch $B$ manipuliert werden. Neben den unteren Grenzen der Zeilengültigkeitsbereiche werden auch die oberen Grenzen verglichen. Gilt $\bar{m}_A < \bar{m}_B$ wird $A$ um $\bar{m}_B - \bar{m}_A$-viele Nullzeilen nach oben erweitert. Das Schema entspricht dem Vorgehen für die unteren Grenzen. Zusätzlich werden die Spaltengültigkeitsbereiche verglichen. Im Fall $\underline{n}_A < \underline{n}_B$ wird $B$ um $\underline{n}_B - \underline{n}_A$-viele Nullspalten nach links ergänzt. Die weiteren Vergleiche erfolgen analog zu dem Vorgehen für die Zeilengültigkeitsbereiche.

Die beschriebene Anpassung der Summanden $A$ und $B$ führt dazu, dass die Dimensionen beider Matrizen identisch sind. Sie können anschließend anhand der üblichen Rechenregeln zur Addition zweier Matrizen addiert werden. Außerdem gilt, dass die bereichsweise Matrizenaddition kommutativ und assoziativ ist.

Die Gleichung (6-5) veranschaulicht die bereichsweise Matrizenaddition an einem einfachen Beispiel.

$$\begin{aligned} A[0,1,0,1] &= \begin{pmatrix} 3 & 4 \\ 1 & 2 \end{pmatrix} \\ B[1,2,1,2] &= \begin{pmatrix} 7 & 8 \\ 5 & 6 \end{pmatrix} \\ C[0,2,0,2] = A[0,1,0,1] +_{\square} B[1,2,1,2] &= \begin{pmatrix} 0 & 7 & 8 \\ 3 & 9 & 6 \\ 1 & 2 & 0 \end{pmatrix} \end{aligned} \tag{6-5}$$

Basierend auf der eingeführten Rechenoperation kann die Gesamtlichtverteilung durch die Einzellichtverteilungen und den vorliegenden Dimmwerten ausgedrückt werden. Dazu werden die bisher als Arrays beschriebenen Einzellichtverteilungen gemäß der Beziehung (6-6) in bereichsweise definierte Matrizen überführt.

$$\begin{aligned} L_k \rightarrow & L_k[\underline{m}_k, \bar{m}_k, \underline{n}_k, \bar{n}_k] \text{ mit} \\ & \underline{m}_k = \frac{\underline{\theta}_k - \underline{\theta}}{\Delta\theta}, \bar{m}_k = \frac{\bar{\theta}_k - \underline{\theta}}{\Delta\theta}, \underline{n}_k = \frac{\underline{\varphi}_k - \underline{\varphi}}{\Delta\varphi} \text{ und } \bar{n}_k = \frac{\bar{\varphi}_k - \underline{\varphi}}{\Delta\varphi} \end{aligned} \tag{6-6}$$

$L_k[\underline{m}_k, \bar{m}_k, \underline{n}_k, \bar{n}_k]$ hat per Definition die gleiche Dimension wie das Array $L_k$ und übernimmt die Einträge aus $L_k$ unverändert. Schließlich ergibt sich die Gesamtlichtverteilung $L_\sum$ zu

$$L_\sum[\underline{m}_\sum, \bar{m}_\sum, \underline{n}_\sum, \bar{n}_\sum](t) = \sum_{k=1}^{K}{}_{\square}\, d_k(t) \cdot L_k[\underline{m}_k, \bar{m}_k, \underline{n}_k, \bar{n}_k], \tag{6-7}$$

wobei das Summenzeichen mit nachgestelltem $\square$ die bereichsweise Matrizenaddition nach Gleichung (6-3) meint. Konstruktionsbedingt gilt für die Gültigkeitsbereiche von $L_\Sigma$

$$L_\Sigma[\underline{m}_\Sigma, \bar{m}_\Sigma, \underline{n}_\Sigma, \bar{n}_\Sigma] = L[0, \frac{\bar{\theta} - \underline{\theta}}{\Delta\theta}, 0, \frac{\bar{\varphi} - \underline{\varphi}}{\Delta\varphi}].$$

Die Rückführung der Bereichsmatrix $L[\underline{m}_\Sigma, \bar{m}_\Sigma, \underline{n}_\Sigma, \bar{n}_\Sigma]$ nach Gleichung (6-7) in das diskretisierte Lichtverteilungsarray $L_\Sigma$ geschieht durch die bloße Übernahme der Einträge. Die Winkelgrenzen von $L_\Sigma$ entsprechen den globalen Minima und Maxima der Polar- und Azimutwinkel nach Gleichung (6-2).

### 6.2.3 Implementierung

Im Abschnitt 6.2.2 wird das formale Vorgehen zur Berechnung der zeitveränderlichen Gesamtlichtverteilung aus den Einzellichtverteilungen eines Scheinwerfers beschrieben. Dieser Formalismus diente in einer ersten Implementierung als direkter Leitfaden. Dabei wurden die zeitaufwendigen und hochgradig parallelisierbaren Matrixadditionen auf dem Grafikprozessor ausgeführt. Das hat neben der kürzeren Berechnungszeit den Vorteil, dass die Gesamtlichtverteilung als Ergebnis des Verfahrens für das weitere Rendering bereits im Speicher der Grafikkarte zur Verfügung steht. Trotz der vorgenommenen Laufzeitoptimierung stellte sich heraus, dass die Implementierung des Verfahrens orientiert am Formalismus nach Abschnitt 6.2.2 auf Desktop-Rechnern, die für heutige Verhältnisse über eine sehr gute Grafikleistung verfügen, nur bis auf wenige hundert Lichtquellen pro Scheinwerfer skaliert. Die Implementierung eignet sich insofern ausschließlich für niedrig aufgelöste Matrix-Systeme, womit die Anforderungen **A3** (Technologie-Unabhängigkeit) und **A4** (Pixel-Skalierbarkeit) durch diese erste Realisierungsvariante nur unzureichend erfüllt werden können.

Auf der Suche nach Lösungen, die den Anforderungen **A3** und **A4** gerecht werden, entstand schließlich eine Implementierung, die in wesentlichen Schritten von dem Formalismus aus Abschnitt 6.2.2 abweicht und weniger intuitiv erscheint. Die neu gestaltete Implementierung sieht jedoch an keiner Stelle Vereinfachungen vor und führt zur gleichen Gesamtlichtverteilung $L_\Sigma$. In den folgenden Unterabschnitten wird diese laufzeit- und speicheroptimierte Vorgehensweise vorgestellt. Das Verfahren befindet sich derzeit unter dem Aktenzeichen DE 10 2020 110 860.5 im Anmeldeverfahren beim Deutschen Patent- und Markenamt (DPMA).

#### Import

Die Datenvorbereitung wird eingeleitet, indem der Anwender den Datensatz eines Scheinwerfersystems importiert. Hierzu steht ihm der in Bild 6-3 gezeigte Import-Dialog zur Verfügung. Es wird nachfolgend von einer spektralen Lichtverteilung ausgegangen. Eine Lichtstärkeverteilung kann als Spezialfall einer spektralen Lichtverteilung angesehen werden. Wie diese in eine spektrale Verteilung umgeformt werden kann, wurde bereits in Abschnitt 6.1 diskutiert.

Im Import-Dialog können verschiedene Einstellungen getroffen werden, von denen die wichtigsten nachfolgend vorgestellt werden. Durch die Auswahl des Dateityps (CIE/IES)

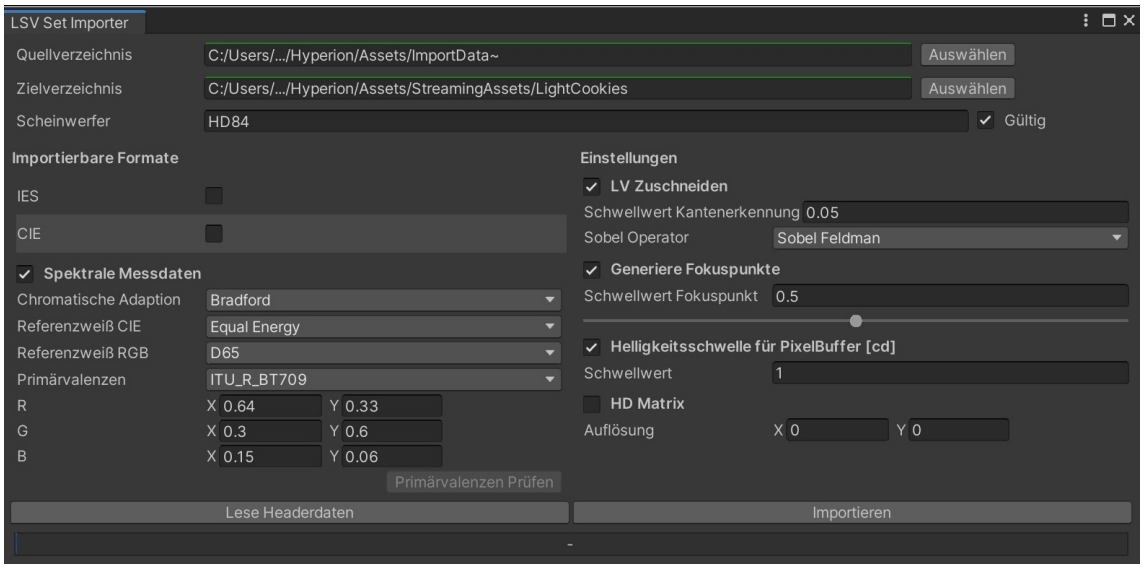


*Bild 6-3*: *Dialog zum Importieren neuer Scheinwerferdatensätze in Hyperion.*

wird spezifiziert, ob es sich um monochrome oder spektrale Lichtdatensätze handelt. Weiterhin muss angegeben werden, wo der Weißpunkt der Quelldaten bezüglich der Primärvalenzen des CIE-Farbraums liegt. Durch die zusätzliche Angabe des Weißpunkts des Zielfarbraums kann auf Wunsch ein Weißpunktwechsel vollzogen werden (s. Abschnitt 2.2.6). Der Zielfarbraum selbst ist ebenfalls parametrierbar, wobei der Standard-RGB-Raum die übliche Wahl darstellt (s. Abschnitt 2.2.7). Bei Verwendung moderner Monitore kann durch die Wahl eines Farbraums mit größerem Gamut-Dreieck ein besseres Ergebnis erzielt werden. Codiert werden die einzelnen Farbkanäle durch Fließkommawerte (Float, 32 Bit). Hiermit wird die darstellbare Bittiefe heutiger Ausgabegeräte weit überschritten. Die hohe Genauigkeit ist dennoch aufgrund von Rundungsfehlern und Tone Mapping von Vorteil (s. Abschnitt 2.3.4). Zur Laufzeitoptimierung kann ein Schwellwert definiert werden. Messpunkte, die eine Lichtstärke unterhalb dieses Schwellwerts aufweisen, werden nicht berücksichtigt. Auf diese Weise kann das Rendering beschleunigt werden. Bei einem zu großen Schwellwert besteht jedoch die Gefahr, dass Streulicht in der Simulation nicht korrekt wiedergegeben wird. Soll auf diese Vereinfachung verzichtet werden, kann der Anwender den Wert des Schwellwerts bei 0 belassen. Schließlich kann ein Bezeichner für den importierten Datensatz vorgesehen werden. Dieser dient bei zukünftigen Simulationen zur Identifikation des Scheinwerfersystems.

Im Anschluss kann der Import-Vorgang gestartet werden. Der Vorgang wird in Bild 6-4 durch ein Ablaufdiagramm veranschaulicht. Daneben ist die während des Imports entstehende Datenstruktur visualisiert. Zu Beginn wird die Wurzel der aufzubauenden Datenstruktur angelegt, welche als logisches Abbild des gesamten Scheinwerfersystems verstanden werden kann. Durch einen zuvor gewählten Bezeichner kann dieses System in späteren Simulation ausgewählt werden. Darüber hinaus enthält das System weitere Meta-Informationen. Beispiele hierfür sind die Information, ob die Lichtverteilungen der einzelnen Lichtquellen monochrom oder spektral sind, und die Art und Weise der Dimmwertvorgabe. Während die Dimmwerte eines Matrix-Systems typischerweise durch die einzelnen Indizes der Lichtquellen vorgegeben werden, ist dieses Vorgehen bei hoch aufgelösten Systemen nicht handhabbar. Alternativ können die Dimmwerte hoch aufgelöster

Systeme durch Texturen mit nur einem Farbkanal beschrieben werden. Die Farbwerte der einzelnen Pixel entsprechen in diesem Fall den Dimmwerten der Lichtquellen. Es ergeben sich zusammengefasst index- und texturbasierte Methoden zur Vorgabe der Dimmwerte.

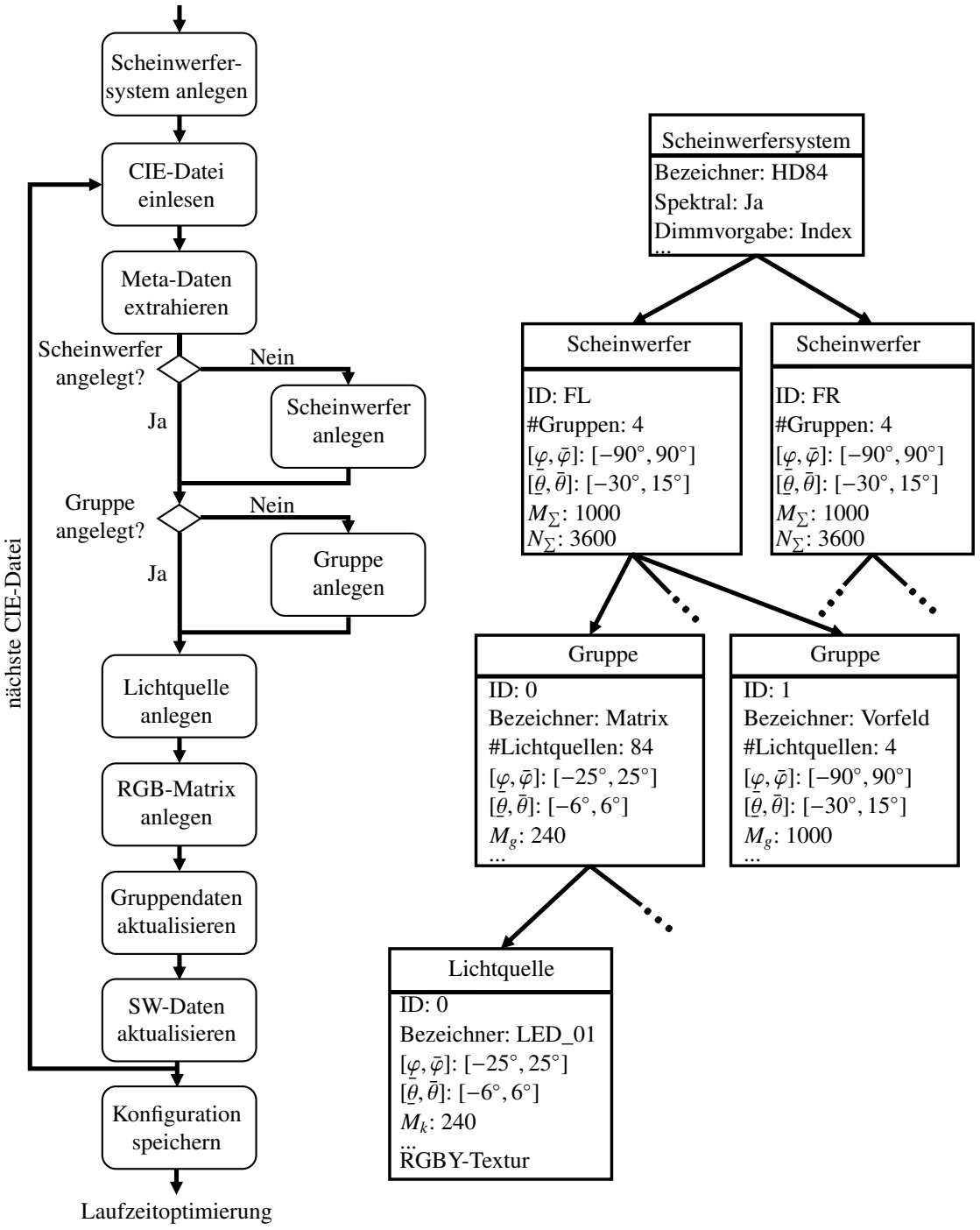


*Bild 6-4*: *Import des Datensatzes eines HD-Scheinwerfersystems.*

Nachdem das Scheinwerfersystem als logische Kapsel für die nachfolgenden Daten vorhanden ist, wird die erste CIE-Datei im ausgewählten Ordner eingelesen. Für das End-

ergebnis ist es unerheblich, in welcher Reihenfolge die CIE-Dateien eingelesen werden. Aus der Datei werden zunächst die Meta-Daten extrahiert. Aus dem Dateinamen geht die Scheinwerferzugehörigkeit und der Index, welcher den Bezug zum Dimmwert in späteren Simulationen herstellt, hervor. Je nach Datensatz kann die Gruppenzugehörigkeit ebenfalls aus dem Dateinamen oder dem vermessenen Winkelbereich abgeleitet werden. Die Einteilung der Pixellichtquellen in Gruppen ist nicht zwingend erforderlich. Abhängig vom Scheinwerfersystem kann sie die Geschwindigkeit des Renderings jedoch erheblich verbessern. Bei der Gruppierung wird ausgenutzt, dass sämtliche Lichtquellen des HD-Moduls in der Regel innerhalb eines Winkelbereichs wirken, der wesentlich kleiner als der gesamte Wirkbereich des Scheinwerfers ist. Es führt deshalb zu einer erhöhten Effizienz, erst in einem reduzierten Winkelbereich die Gesamtlichtverteilung des HD-Moduls zu berechnen, bevor diese mit den übrigen Lichtquellen zur Gesamtlichtverteilung des Scheinwerfers zusammengeführt wird.

Anschließend wird geprüft, ob der Scheinwerfer oder die Gruppe, die der gerade verarbeiteten Lichtquelle zuzuordnen sind, bereits angelegt wurden. Das ist der Fall, wenn zuvor verarbeitete Lichtquellen zum gleichen Scheinwerfer oder zur gleichen Gruppe gehörten. Sind diese Datenstrukturen bisher nicht vorhanden, werden sie neu angelegt. Im anderen Fall wird die Lichtquelle in der in Bild 6-4 als Baum organisierten Datenstruktur an der richtigen Stelle eingeordnet.

Das logische Abbild der Lichtquelle enthält neben der eigentlichen Lichtverteilung weitere Daten. Zur Identifikation wird jede LED durch einen Bezeichner gekennzeichnet, welcher aus dem Dateinamen abgeleitet wird. Parallel dazu trägt die Lichtquelle zur technischen Identifizierung eine ID. Diese ID ist maßgeblich, um die zeitveränderlichen Dimmwerte korrekt auf die Lichtquellen zuzuweisen. Sie ist nicht nur innerhalb der Gruppe, sondern für das gesamte Scheinwerfersystem eindeutig. Weitere wesentliche Informationen sind die horizontalen und vertikalen Winkelbereiche $[\underline{\varphi}, \bar{\varphi}]$ und $[\underline{\theta}, \bar{\theta}]$, in denen die Lichtquelle vermessen wurde. Gemeinsam mit der Zeilen- und Spaltenzahl (Vgl. $M_k$ und $N_k$ in Abschnitt 6.2.2) ergibt sich die Auflösung der Lichtverteilung.

Die Einzellichtverteilung wird formal bereits in Abschnitt 6.2.1 als bereichsweise definierte Matrix $L_k$ eingeführt. Der Index $k$ korrespondiert mit der bereits angesprochenen ID. Datentechnisch wird die Lichtverteilung als Textur mit vier Farbkanälen pro Texel abgebildet. Hierbei enthalten die ersten drei Kanäle die Farbe im spezifizierten Zielfarbraum (meist sRGB). Der vierte Kanal beinhaltet die Lichtstärke (Y-Koordinate des Quellfarbraums CIE-XYZ). Diese Information, welche an späterer Stelle benötigt wird, kann auch aus dem RGB-Wert ermittelt werden. Dennoch wird sie zur Laufzeitoptimierung redundant in der Textur vorgehalten. Fortlaufend werden die so strukturierten Farbkanäle durch das Kürzel „RGBY“ referenziert. Die RGBY-Textur enthält $M_k$-viele Zeilen und $N_k$-viele Spalten. So können die Daten der Lichtverteilung verlustfrei abgebildet werden.

Das Hinzufügen einer Lichtquelle erfordert die Aktualisierung der übergeordneten Gruppe und des Scheinwerfers. Innerhalb der Gruppe wird die Anzahl der enthaltenen Lichtquellen inkrementiert. Sofern erforderlich werden der Winkelbereich sowie die Dimensionen $M_g$ und $N_g$ der Gruppe vergrößert. Gleiches gilt für den Scheinwerfer.

Im Anschluss wird das beschriebene Vorgehen für die nächste CIE-Datei wiederholt. Diese Iteration terminiert, nachdem alle CIE-Dateien des Scheinwerfersystems durchlaufen wurden. Ist das der Fall, gibt die baumartige Datenstruktur aus Bild 6-4 den Datensatz des

Scheinwerfersystems vollständig wieder. Im finalen Schritt der Datenvorbereitung wird der gewonnene Datensatz gesichert. Hierbei kann zwischen der logischen Strukturierung samt der Metadaten und den Lichtverteilungsdaten in Form der RGBY-Texturen unterschieden werden.

Die erstgenannten Informationen werden mittels JavaScript Object Notation (JSON) durch eine textbasierte Datei gesichert. Bei JSON handelt es sich um eine gut lesbare und komfortable Variante zur Serialisierung und Deserialisierung strukturierter Daten. JSON ist in den Standards [ISO01] und [ECMA01] dokumentiert. Im Vergleich zu XML (Extensible Marup Language)-basierten Dateien überzeugt JSON durch die besonders kompakte Darstellung und die bessere Unterstützung komplexerer Datentypen. Die RGBY-Texturen werden hingegen binär codiert gespeichert. Auf diese Weise sind sie kompakt und können schneller gespeichert und geladen werden. Die textbasierte Einsehbarkeit bringt aufgrund der hohen Datenmenge ohnehin keinen Vorteil.

Hiermit ist der Import-Vorgang eines Scheinwerferdatensatzes vollständig abgeschlossen. Die gewonnenen Daten sind gesichert. Auf sie kann jederzeit zurückgegriffen werden. An die Datenvorbereitung schließt sich die Datenoptimierung an.

### Datenoptimierung

Die Echtzeitanforderung **A6** ist kritisch für die Qualität der Nachtfahrtsimulation. Sie sollte unter allen Umständen eingehalten werden. Um Echtzeitfähigkeit gewährleisten zu können, wird die im Abschnitt 6.2.3 vorgestellte Datenbasis im Rahmen eines Preprocessings so aufbereitet, dass die Laufzeitberechnungen mit minimalem Aufwand durchgeführt werden können. Die Datenoptimierung muss für jedes zu simulierende Scheinwerfersystem nur einmalig durchgeführt werden. Die aufbereiteten Daten stehen anschließend für die Simulation zur Verfügung.

Anhand von Pseudocode soll das Vorgehen beim Preprocessing erläutert werden. Der Algorithmus 3 organisiert die Datenoptimierung auf oberster Ebene. Eingang des Algorithmus stellen die importierten Daten dar, welche im Pseudocode durch den Parameter *headlampSystem* bezeichnet sind. Diese umfassen sowohl die Datenstruktur entsprechend Bild 6-4 als auch die RGBY-Matrizen als datentechnische Abbildungen der einzelnen Lichtverteilungen. Wie aus den geschachtelten Iterationen in Algorithmus 3 hervorgeht, werden die Daten des Scheinwerfersystems gruppenweise verarbeitet. Algorithmisch spielt es keine Rolle, ob die derzeit bearbeitete Gruppe zum linken oder rechten Scheinwerfer gehört. Gleichwohl ist diese Information bei den späteren Laufzeitberechnung relevant und kann aus den Metadaten extrahiert werden.

Für jede Gruppe von Lichtquellen wird in Zeile 6 der Algorithmus 4 aufgerufen. Seine Aufgabe ist es, die Daten ausgabeorientiert umzustrukturieren. Hiermit ist gemeint, dass die Daten in eine Struktur gebracht werden, welche ideal zur Ermittlung der Gesamtlichtverteilung gestaltet ist. Bisher sind alle Einzellichtverteilungen separiert voneinander dargestellt. Stattdessen soll nun die Komposition einzelner Einträge aus verschiedenen Einzellichtverteilungen zum jeweiligen Pixel der Gesamtlichtverteilung maßgeblich für die Strukturierung sein. Im Bild 6-5 wird beispielhaft dargestellt, wie ein Pixel der Gruppenlichtverteilung (weißes Kästchen im oberen Teil von Bild 6-5) durch drei verschiedene

**Algorithm 3** Preprocess Headlampsystem

```
1:  require: headlampSystem as Datastructure like Fig. 6-4
2:  function PREPROCESSHEADLAMPSYSTEM(headlampSystem)
3:      local variables: reorganizedData as Set of Datastructures, bufferdData as Set
    of Buffers
4:      for all Headlamp lamp in headlampSystem do
5:          for all Group group in lamp do
6:              reorganizedData ← reorganizeGroupdata(group)
7:              bufferedData ← createBuffers(reorganizedData)
8:              saveToDisk(bufferedData)
9:          end for
10:     end for
11: end function
```

Einzellichtverteilungen (LED X, Y und Z) entsteht. Die relevanten Pixel in den Einzellichtverteilungen (weiße Kästchen im unteren Teil von Bild 6-5) liegen abhängig vom vermessenen Winkelbereich an verschiedenen Positionen.

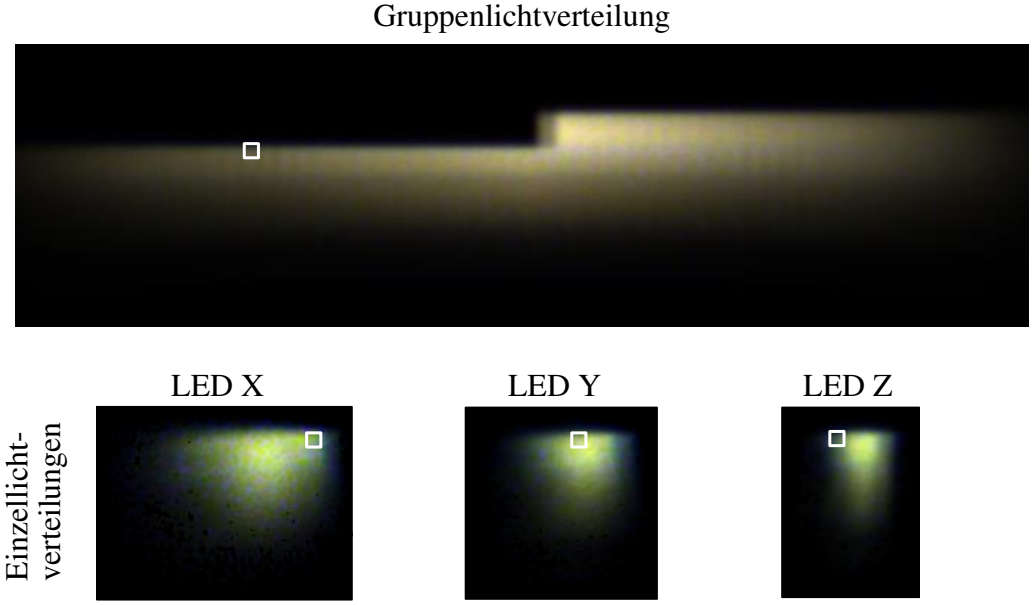


*Bild 6-5*: *Komposition der Einzellichtverteilungen zur Gruppenlichtverteilung.*

In Form des Parameters *group* erhält der Algorithmus 4 alle Daten, welche die aktuelle Gruppe und die in ihr enthaltenen Lichtquellen betreffen. Der Algorithmus iteriert über alle Einzellichtquellen innerhalb der Gruppe (Zeile 6). Die Zeilen- und Spaltenzahlen aus der Datenstruktur nach Bild 6-4 bestimmen die Größe der Gesamtlichtverteilung dieser Gruppe. Abhängig von den Winkelbereichen der einzelnen Lichtquellen sind diese gegebenenfalls nur in einem Teilbereich der Gruppenlichtverteilung vermessen. Deshalb wird durch die Funktion *calcOffset* der Versatz der Einzellichtverteilung zur Gruppenlichtverteilung entsprechend der bereichsweisen Matrizenaddition ermittelt. Unter der getroffenen Annahme eines einheitlichen Diskretisierungsrasters und der bekannten Winkelbereiche der Einzellichtverteilung und der Gruppenlichtverteilung ist diese Berechnung einfach möglich. *calcOffset* gibt dementsprechend einen 2-elementigen Vektor mit positiven und ganzzah-

**Algorithm 4** Reorganize Groupdata

```
1: require: group as Datastructure like Fig. 6-4, minLumIntens ∈ ℝ lowest considered
   Luminous Intensity
2: function REORGANIZEGROUPDATA(group, minLumIntens)
3:     local variables: totalSrcPixel ∈ ℕ, trgLocList as List of ℕ², pixelOffset ∈
   ℕ², trgLoc ∈ ℕ², srcRGBYAtTrg as Dictionary from ℕ² to List of ℝ⁴,
   srcIDAtTrg as Dictionary from ℕ² to List of ℕ
4:     totalSrcPixel ← 0
5:     trgLocList ← ∅
6:     for all Light light in group do
7:         pixelOffset ← calcOffset(light.angleBounds, group.angleBounds)
8:         for m = 1 to light.M do
9:             for n = 1 to light.N do
10:                trgLoc.m ← pixelOffset.m + m
11:                trgLoc.n ← pixelOffset.n + n
12:                if light.RGBYMat(m,n).Y > minLumIntens then
13:                    totalSrcPixel + +
14:                    if trgLoc ∉ trgLocList then
15:                        trgLocList.Add(trgLoc)
16:                        srcRGBYAtTrg(trgLoc) ← new empty List
17:                        srcIDAtTrg(trgLoc) ← new empty List
18:                    end if
19:                    srcRGBYAtTrg(trgLoc).Add(light.RGBYMat(m, n))
20:                    srcIDAtTrg(trgLoc).Add(light.ID)
21:                end if
22:            end for
23:        end for
24:    end for
25:    result ← totalSrcPixel, trgLocList, srcRGBYAtTrg, srcIDAtTrg
26:    return result
27: end function
```

ligen Einträgen zurück, welcher durch die Variable *pixelOffset* für die Berechnungszeit der aktuellen Lichtquelle *light* gesichert wird.

Anschließend wird durch verschachtelte Iteration durch alle Pixel der aktuellen Einzellichtverteilung *light* iteriert. Für jedes Pixel wird der Schleifenrumpf von Zeile 10 bis 25 durchlaufen. Im ersten Schritt wird die Entsprechung des aktuellen Pixels der Einzellichtverteilung innerhalb der Gruppenlichtverteilung ermittelt. Die korrekte Zuordnung ist dann gegeben, wenn beide Pixel dem gleichen Winkelpaar $(\theta, \varphi)$ entsprechen. Diese Koordinatentransformation von der Einzellichtverteilung in die Gruppenlichtverteilung kann durch die Addition des zuvor berechneten *pixelOffset* auf die Koordinaten $m$ und $n$ des Pixels geleistet werden. Die Lage des Pixels in der Gruppenlichtverteilung wird durch die Variable *trgLoc* (Zielposition) gesichert. Der Lichteintrag des Pixels wird aus der RGBY-Matrix an den Koordinaten $(m, n)$ ausgelesen und stellt einen Vektor mit vier Einträgen dar.

Im nächsten Schritt wird überprüft, ob die Lichtstärke $light.RGBYMat(m, n).Y$ der Lichtquelle *light* in der gerade betrachteten Richtung $(m, n)$ einen konfigurierbaren Schwellwert *minLumIntens* überschreitet (s. Bild 6-3). Durch das Setzen dieses Schwellwerts können Effekte durch Messrauschen und die Datenmenge reduziert werden. Ist diese Reduktion nicht gewünscht, wählt man $minLumIntens = 0$. Lichtstärkewerte, welche die in Zeile 13 definierte Bedingung nicht erfüllen, führen zum direkten Übergang zum nächsten Pixel der Einzellichtverteilung und werden nicht weiter berücksichtigt.

Erfüllt $light.RGBYMat(m, n).Y$ hingegen die Bedingung, ist der Beitrag relevant für die Gruppenlichtverteilung und muss datentechnisch abgebildet werden. In diesem Fall wird zunächst der Zähler *totalSrcPixel* inkrementiert, welcher die Anzahl aller Pixel aus Einzellichtverteilungen zählt, die einen Beitrag zur Gruppenlichtverteilung leisten. Der weiterführende Ablauf hängt davon ab, ob ein zuvor verarbeitetes Pixel einer anderen Einzellichtverteilung bereits einen Beitrag zum Pixel an der Position *trgLoc* in der Gruppenlichtverteilung geleistet hat.

Ist das nicht der Fall, so wird das Pixel der Gruppenlichtverteilung mit seiner Position *trgLoc* in die Liste *trgLocList* aufgenommen. Zusätzlich werden für das Pixel der Gruppenlichtverteilung die Listen *srcRGBYAtTrg(trgLoc)* und *srcIDAtTrg(trgLoc)* angelegt. *srcRGBYAtTrg(trgLoc)* erhält mit *light.RGBYMat(m,n)* den ersten Eintrag. Genauso erhält die Liste *srcIDAtTrg(trgLoc)* mit der ID der aktuell betrachteten Lichtquelle ihren ersten Eintrag. Die Reihenfolgen innerhalb der Listen müssen erhalten bleiben, um eine spätere Zuordnung der IDs zu den Lichteinträgen gewährleisten zu können.

Im anderen Fall wurde das Pixel *trgLoc* der Gruppenlichtverteilung schon vorher durch einen Pixel einer Einzellichtverteilung getroffen. Insofern existiert der Eintrag *trgLoc* in *trgLocList* und die Listen *srcRGBYAtTrg(trgLoc)* und *srcIDAtTrg(trgLoc)* wurden bereits angelegt. Es genügt dann, den Lichteintrag des gerade betrachteten Pixels der Einzellichtverteilung und deren ID in die entsprechenden Listen aufzunehmen.

Wurde das beschriebene Verfahren durchlaufen, enthalten der Zähler *totalSrcPixel*, die Liste *trgLocList* sowie die Listensammlungen *srcRGBYAtTrg* und *srcIDAtTrg* alle Informationen, die für den weiteren Ablauf benötigt werden. Es sei angemerkt, dass *trgLocList* nicht zwangsläufig alle Pixel der Gruppenlichtverteilung beinhalten muss. Wird ein Pixel der Gruppenlichtverteilung nicht durch mindestens ein Pixel einer Einzellichtverteilung, dessen Lichteintrag den Schwellwert *minLumIntens* überschreitet, erfasst, findet dieses Pixel im

weiteren Verlauf keine Berücksichtigung mehr. Das ist wünschenswert, da es keinen Beitrag zur Gruppenlichtverteilung leistet.

Im Anschluss an die Reorganisation der Daten ruft der übergeordnete Algorithmus 3 den Algorithmus 5 auf. Dessen Aufgabe ist es, die ausgabeorientierten Daten in eine Struktur zu überführen, durch welche die Gruppenlichtverteilung zur Laufzeit mit bestmöglicher Effizienz berechnet werden kann. Neben der Pseudocode-Darstellung wird das zugrunde liegende Prinzip des Algorithmus 5 in Bild 6-6 grafisch visualisiert.

**Algorithm 5** Create Buffers

1: **require:** $totalSrcPixel \in \mathbb{N}$, $trgLocList$ as List of $\mathbb{N}^2$, $srcRGBYAtTrg$ as Dictionary from $\mathbb{N}^2$ to List of $\mathbb{R}^4$, $srcIDAtTrg$ as Dictionary from $\mathbb{N}^2$ to List of $\mathbb{N}$
2: **function** createBuffers($totalSrcPixel, trgLocList, srcRGBYAtTrg, srcIDAtTrg$)
3: **local variables:** $totalTrgPixel \in \mathbb{N}$, $trgBuffer$ as Buffer of $\mathbb{N}^4$, $srcRGBYBuffer$ as Buffer of $\mathbb{R}^4$, $srcIDBuffer$ as Buffer of $\mathbb{N}$, $srcBufferOffset \in \mathbb{N}$, $trgLoc \in \mathbb{N}^2$, $srcPixelAtTrg \in \mathbb{N}$
4: $totalTrgPixel \leftarrow count(trgLocList)$
5: $trgBuffer \leftarrow newBuffer(Type : \mathbb{N}^4, Size : totalTrgPixel)$
6: $srcRGBYBuffer \leftarrow newBuffer(Type : \mathbb{R}^4, Size : totalSrcPixel)$
7: $srcIDBuffer \leftarrow newBuffer(Type : \mathbb{N}, Size : totalSrcPixel)$
8: $srcBufferOffset \leftarrow 1$
9: **for** $i = 1$ to $totalTrgPixel$ **do**
10: $trgLoc \leftarrow trgLocList[i]$
11: $srcPixelAtTrg \leftarrow count(srcRGBYAtTrg(trgLoc))$
12: $trgBuffer[i] \leftarrow (trgLoc, srcBufferOffset, srcPixelAtTrg)$
13: **for** $j = 1$ to $srcPixelAtTrg$ **do**
14: $srcRGBYBuffer[srcBufferOffset] \leftarrow srcRGBYAtTrg(trgLoc)[j]$
15: $srcIDBuffer[srcBufferOffset] \leftarrow srcIDAtTrg(trgLoc)[j]$
16: $srcBufferOffset++$
17: **end for**
18: **end for**
19: **return** $trgBuffer, srcRGBYBuffer, srcIDBuffer$
20: **end function**

Als Eingangsparameter erwartet Algorithmus 5 die Rückgabe des zuvor ausgeführten Algorithmus 4. Die diskretisierte Gruppenlichtverteilung wird in Bild 6-6 oben links dargestellt. Jeder Kasten repräsentiert einen Pixel. Die $x$- und $y$-Koordinaten dieser Pixel sind in der Liste *trgLocList* enthalten, sofern mindestens eine Einzellichtverteilung einen Lichteintrag an diesem Pixel der Gruppenlichtverteilung aufweist. Rechts im Bild 6-6 sind beispielhaft die Listen der Lichteinträge (*srcRGBYAtTrg*) und der Lichtquellen-IDs (*srcIDAtTrg*) für drei Pixel der Gruppenlichtverteilung dargestellt. Die Gruppenlichtverteilung am Koordinatenpaar $(1, 1)$ wird durch die Einzellichtquellen mit den IDs 1, 2 und 4 bestimmt. Dabei hat die Lichtquelle mit ID 1 bei voller Bestromung einen Eintrag von $val_1$, die Lichtquelle mit ID 2 einen Eintrag von $val_2$ und die Lichtquelle mit ID 4 einen Eintrag von $val_3$. Die übrigen Lichtquellen nehmen auf die Koordinaten $(1, 1)$ keinen Einfluss. Das Koordinatenpaar $(1, 2)$ wird hingegen durch keine Lichtquelle beeinflusst. Dem entsprechend sind die

Listen *srcRGBYAtTrg(1,2)* und *srcIDAtTrg(1,2)* leer. Als abschließendes Beispiel wird das Koordinatenpaar $(1, 3)$ der Gruppenlichtverteilung durch fünf Einzellichtquellen mit den IDs 2, 3, 4, 6 und 7 beeinflusst.

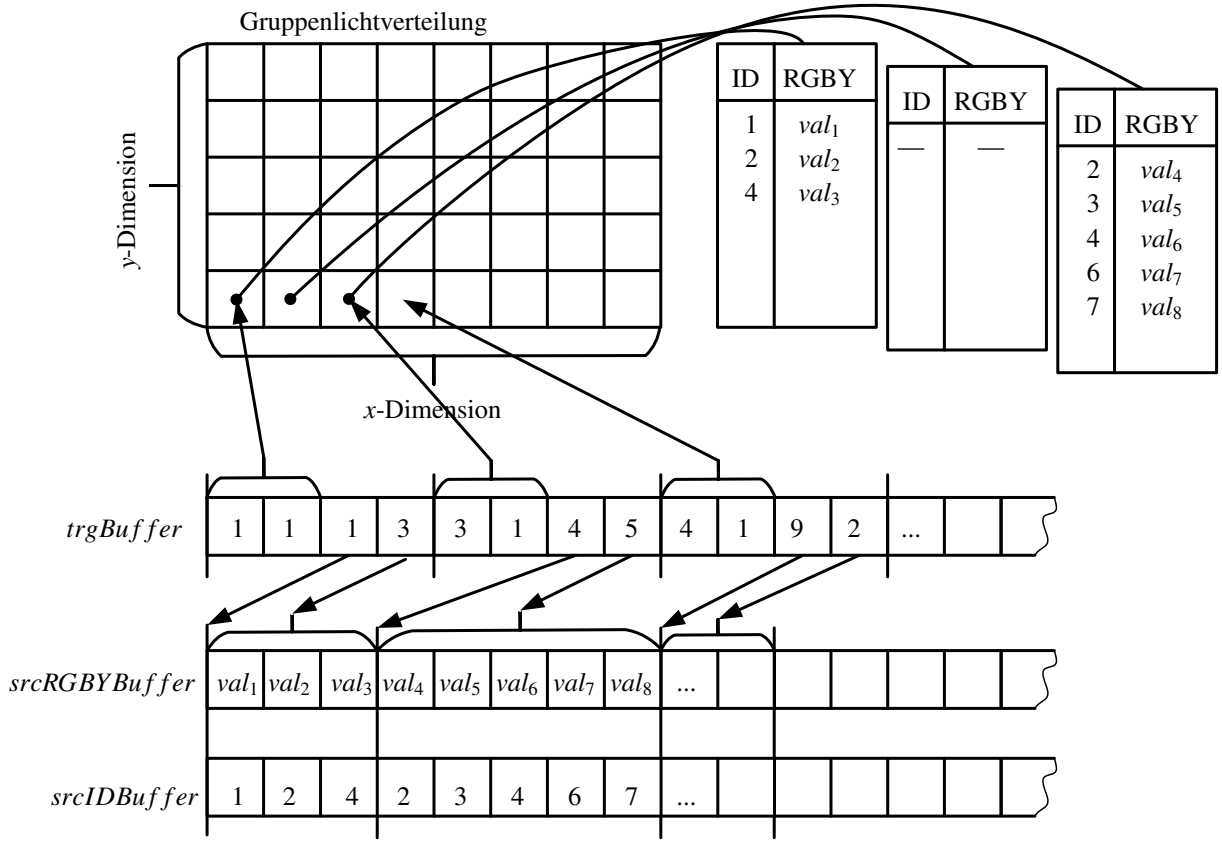


*Bild 6-6*: *Komposition von Einzellichtverteilungen zur Gesamtlichtverteilung durch lineare Buffer.*

Algorithmus 5 iteriert nun über alle Koordinaten der Liste *trgLocList* und somit über alle Pixel der Gruppenlichtverteilung, die durch mindestens eine Lichtquelle ausgeleuchtet werden. Bei der Iteration befüllt der Algorithmus die Buffer *trgBuffer*, *srcRGBYBuffer* und *srcIDBuffer*, welche initial leer sind. Sie werden im unteren Bereich von Bild 6-6 dargestellt. In der äußeren Schleife wird der Buffer *trgBuffer* beschrieben. Seine Länge korreliert mit der Anzahl der Einträge in *trgLocList*. Für jedes Koordinatenpaar *trgLoc* in *trgLocList* werden in *trgBuffer* vier Werte gespeichert. Die ersten beiden Werte stellen die Koordinaten *trgLoc* selbst dar. Der dritte Wert *srcBufferOffset* gibt die Adresse an, an der die übrigen Buffer ausgelesen werden müssen, um die für das aktuell betrachtete Pixel relevanten Werte zu erhalten. Der vierte Wert *srcPixelAtTrg* gibt an, wie viele Werte angefangen von *srcBufferOffset* sich auf das aktuelle Pixel beziehen. *srcPixelAtTrg* entspricht der Anzahl von Lichtquellen, die an der Koordinate *trgLoc* auf die Gruppenlichtverteilung Einfluss nehmen. In der inneren Schleife werden die Buffer *srcRGBYBuffer* und *srcIDBuffer* beschrieben. *srcRGBYBuffer* und *srcIDBuffer* weisen konstruktionsbedingt stets die gleichen Längen auf. Nach jedem neuen Eintrag in diese Buffer wird die Zählvariable *srcBufferOffset* inkrementiert. So ist der Buffer-Offset für *srcIDBuffer* und *srcRGBYBuffer*

bei der Verarbeitung des nächsten Koordinatenpaars in *trgLocList* bekannt. In Bild 6-6 sind die semantisch zusammenhängenden Buffereinträge durch verlängerte vertikale Linien gruppiert. Während *trgBuffer* stets Gruppen aus vier Elementen beinhaltet, variieren die Gruppengrößen der anderen Buffer mit dem Koordinatenpaar. Im Beispiel aus Bild 6-6 führt das Pixel mit den Koordinaten (1, 1) innerhalb der Gruppenlichtverteilung zu jeweils 3 Einträgen in *srcRGBYBuffe*r und *srcIDBuffer*. Da es das erste verarbeitete Koordinatenpaar darstellt, wird in *trgBuffer* der Buffer-Offset 1 eingetragen. Der nachfolgende Wert 3 gibt an, dass beginnend bei Index 1 ingesamt 3 Werte der Buffer *srcRGBYBuffer* und *srcIDBuffer* auf das Pixel mit den Koordinaten (1, 1) Bezug nehmen. Die RGBY-Werte und die IDs der Einzellichtquellen finden sich entsprechend der visualisierten Tabelle im oberen rechten Bereich des Bilds 6-6 in den Buffern *srcRGBYBuffer* und *srcIDBuffer*. Im Beispiel wird das Pixel mit den Koordinaten (2, 1) durch keine Einzellichtquelle beeinflusst. Es findet sich deshalb in der Buffer-Struktur nicht wieder. Stattdessen schließen sich die Daten zu dem Pixel mit den Koordinaten (3, 1) an.

Der Algorithmus terminiert, nachdem alle Pixel der Gruppenlichtverteilung (*trgLocList*), die den Schwellwert der Lichtstärke überschreiten, durchlaufen worden sind. Anschließend werden die Inhalte der vorher komplexen Datenstruktur durch drei eindimensionale Buffer wiedergegeben. Wie sich nachfolgend zeigen wird, ist diese ausgabeorientierte Umstrukturierung mit einem erheblichen Performancegewinn der kritischen Laufzeitberechnungen verbunden.

### Initialisierung

In den vorhergehenden Unterabschnitten werden die notwendigen Schritte zur Vorbereitung eines Scheinwerfersystems für die Nachtfahrtsimulation beschrieben. Diese Arbeitsschritte erfolgen einmalig für das jeweilige System, sodass die resultierenden Daten anschließend für die Simulation zur Verfügung stehen. In diesem Unterabschnitt werden die Abläufe thematisiert, welche zu Beginn jeder Simulation erforderlich sind. Durch sie wird der erforderlich Kontext für die Laufzeitberechnungen erzeugt.

Im Rahmen der Offline-Parametrierung wählt der Anwender das zu simulierende Scheinwerfersystem aus. Die Identifikation des Systems erfolgt durch die Bezeichnung, welche beim Import des Scheinwerferdatensatzes vergeben wurde. Zu Beginn der Simulation wird der Szenengraph erzeugt, welcher das Egofahrzeug und die hierarchisch darunter angeordnete, in Abschnitt 5.4.3 diskutierte, Struktur für das Scheinwerfersystem beinhaltet. Jeder Scheinwerfer innerhalb der Szene enthält ein Headlamp-Script. Das Headlamp-Script beinhaltet wiederum eine Instanz der Klasse „Combiner“. Die Bestimmung der Gesamtlichtverteilung geschieht exklusiv in dieser Klasse. Mit dem Simulationsbeginn durchläuft der Combiner den in Bild 6-7 als Flussdiagramm visualisierten Vorgang, welcher den datentechnischen Kontext für die Laufzeitberechnungen generiert. Diese Schritte werden für jeden Scheinwerfer durchlaufen.

Zunächst werden die Daten der ersten Gruppe des jeweiligen Scheinwerfers geladen. Hierzu gehören zum einen die optimierten Buffer als Ergebnis des Algorithmus 5 und zum anderen die Metadaten der Gruppen, wie sie in der Baumstruktur des Bilds 6-4 visualisiert werden. Da die Bestimmung der Gesamtlichtverteilung durch einen hochgradig parallelen Ansatz auf dem Grafikprozessor erfolgen soll, werden nun die Buffer in Form sogenannter

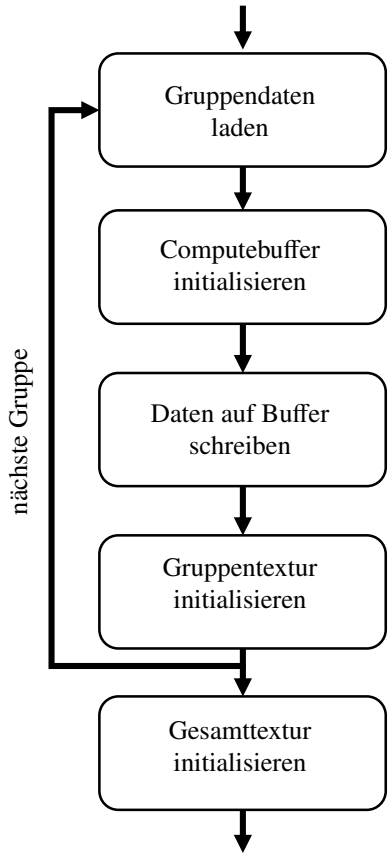


*Bild 6-7*: *Initialisierung des Combiners.*

Computebuffer auf den Grafikspeicher transferiert. Vorbereitend werden dafür entsprechende Computebuffer mit der richtigen Länge und den erforderlichen Datentypen erzeugt. Anschließend werden die Daten der Buffer *trgBuffer*, *srcRGBYBuffer* und *srcIDBuffer* in die Computebuffer des Grafikspeichers kopiert. Zusätzlich wird ein vierter Buffer *dimValBuffer* erzeugt. Dessen Länge entspricht der Anzahl von Lichtquellen innerhalb der aktuellen Gruppe. Die Daten dieses Buffers sind Fließkommawerte, welche im Intervall $[0, 1]$ liegen. Dieser Buffer dient zur Aufnahme der momentanen Dimmwerte. Er kann zum jetzigen Zeitpunkt nicht mit Daten befüllt werden, da die Dimmwerte zeitlichen Änderungen unterliegen und im Verlauf der Simulation permanent angepasst werden müssen. Schließlich wird die Zieltextur der aktuellen Gruppe initialisiert. Ihre Größe ist entsprechend der Zeilen- und Spaltenzahl der Gruppe zu wählen (s. Bild 6-4).

Dieses Verfahren wird für alle Gruppen des Scheinwerfers wiederholt. Die Zieltexturen der Gruppen werden in einer Liste mit der Bezeichnung *groupTextures* gesammelt. Nachdem alle Gruppen durchlaufen wurden, wird abschließend die Zieltextur für die Gesamtlichtverteilung erzeugt. Ihre Größe entspricht der $x$- und $y$-Dimension des Scheinwerfers. Sie trägt den Namen $L_\Sigma$. Sowohl die Gruppen- als auch die Gesamtzieltexturen beinhalten Elemente aus der Menge $\mathbb{R}^4$, da es sich um Summen von RGBY-Vektoren handelt.

### Laufzeitberechnungen

Nach der vollständigen Initialisierung der Simulation beginnt ein sich stetig wiederholender Berechnungszyklus. In kurzen Zeitschritten wird der Zustand des eigenen Fahrzeugs und der übrigen Umgebung angepasst. Die sich verändernden Simulationszustände und Fahreingaben führen auch zu neuen Dimmwertvorgaben des Scheinwerfersteuergeräts. Dementsprechend unterliegen sie permanenten zeitlichen Änderungen und können erst zur Laufzeit einfließen. In diesem Abschnitt wird beschrieben, wie die Dimmwerte und die bereits beschriebenen, statischen Buffer zur Gesamtlichtverteilung kombiniert werden. Formal wird die Gleichung (6-7) gelöst. Zur möglichst zeiteffizienten Lösung wird dazu ein hochgradig paralleler Ansatz gewählt, welcher ausgeführt auf den vielen Rechenkernen des Grafikprozessors innerhalb kürzester Zeit zur Gesamtlichtverteilung führt.

Die Berechnung der Gesamtlichtverteilung erfolgt zweistufig. Im ersten Schritt werden die Gruppenlichtverteilungen für alle Gruppen des Scheinwerfersystems ermittelt. Nachfolgend werden die Gruppenlichtverteilungen zur Gesamtlichtverteilung aggregiert. Auch wenn die wesentlichen Berechnungen auf der GPU durchgeführt werden, dirigiert die CPU sämtliche Abläufe. Der entsprechende Code zur Koordination der Berechnung ist als Methode „combine“ in der Klasse „Combiner“ verankert. Der Pseudocode 6 gibt die wesentlichen Inhalte wieder.

**Algorithm 6** Combine - Script-Component „Combiner“

```
1:  require: headlampSystem as Datastructure like Fig. 6-4, dimValues ∈ [0,1]^K,
    groupLightDistList as List of (ℝ^4)^(M_g × N_g)
2:  function COMBINE(dimValues)
3:      local variables: L_∑ ∈ (ℝ^4)^(M_∑ × N_∑)
4:      dimValBuffer ← asComputeBuffer(dimValues)
5:      for all Group grp ∈ headlampSystem.groups do
6:          trgBuffer ← grp.trgBuffer
7:          srcRGBYBuffer ← grp.srcRGBYBuffer
8:          srcIDBuffer ← grp.srcIDBuffer
9:          groupLightDistList[grp.ID] ← dispatch(combineGroup, sizeOf(trgBuffer) / threadGroupSize)
10:         L_∑ ← Blit(groupLightDistList[grp.ID], L_∑, combineOverall)
11:     end for
12: end function
```

Ein einleitendes Zurücksetzen der Gesamtlichtverteilung $L_\Sigma$ ist nicht erforderlich, da alle sich dynamisch ändernden Einträge in jeder Iteration überschrieben werden. Zurücksetzen bedeutet in diesem Zusammenhang, dass sämtliche RGBY-Einträge der Zieltextur durch den Nullvektor ersetzt werden. Zuerst werden die Dimmwerte in den *dimValBuffer* geschrieben, sodass sie dem Grafikprozessor zur Verfügung stehen. Anschließend wird die Gesamtlichtverteilung gruppenweise ermittelt. Dazu wird über alle Lichtquellen-Gruppen des Scheinwerfers iteriert. Im Schleifenrumpf ist das initiale Zurücksetzen der Gruppenlichtverteilung mit dem bereits erwähnten Argument im Kontext der Gesamtlichtverteilung nicht erforderlich. Es stehen nun alle Daten auf dem Grafikspeicher bereit, um die Gruppenlichtverteilung zu ermitteln. In Zeile 9 wird der Computeshader durch einen *dispatch*-Befehl gestartet. Neben der Methode *combineGroup* des Shaders wird als

zweiter Parameter die Anzahl der Threadgruppen erwartet. Durch diesen wird indirekt die Größe der Threadgruppen definiert, welche für den Datenaustausch zwischen den Threads relevant ist. Der hier formulierte Algorithmus wurde bewusst so formuliert, dass ein Datenaustausch unter den Threads nicht erforderlich ist, da derartige Abhängigkeiten die Performance erheblich beeinträchtigen können. Vor diesem Hintergrund ist hier die kleinste empfohlenen Gruppengröße zu wählen. Diese ist von der Grafikhardware abhängig und liegt häufig bei 32 oder 64 Threads pro Gruppe. Da es eine hardwarespezifische obere Schranke für die Anzahl der Threadgruppen gibt, muss die Gruppengröße bei sehr hohen Dimensionen der Gruppenlichtverteilung weiter erhöht werden.

Der Computeshader, dessen Pseudocode im Algorithmus 7 dargestellt ist, wird auf der GPU ausgeführt. Entsprechend den Parametern des *dispatch*-Befehls werden genau so viele nebenläufige Threads gestartet, wie Einträge im *trgBuffer* vorhanden sind. Jeder dieser Threads durchläuft den in Algorithmus 7 gezeigten Pseudocode.

**Algorithm 7** Group Combiner Shader (Compute)

1: **require:** $instID \in \mathbb{N}$, $dimValBuffer \in [0,1]^K$, $trgBuffer$ as Buffer of $\mathbb{N}^4$, $srcRGBYBuffer$ as Buffer of $\mathbb{R}^4$, $srcIDBuffer$ as Buffer of $\mathbb{N}$, $groupLightDist \in (\mathbb{R}^4)^{M_g \times N_g}$
2: **function** COMBINEGROUP($instID, dimValBuffer$)
3: **local variables:** $trgLoc \in \mathbb{N}^2$, $srcBufferOffset, srcBufferCount \in \mathbb{N}$, $srcVal, tmpVal \in \mathbb{R}^4$, $srcID \in \mathbb{N}$
4: $trgLoc.m \leftarrow trgBuffer[instID][1]$
5: $trgLoc.n \leftarrow trgBuffer[instID][2]$
6: $srcBufferOffset \leftarrow trgBuffer[instID][3]$
7: $srcBufferCount \leftarrow trgBuffer[instID][4]$
8: $tmpVal \leftarrow (0,0,0,0)$
9: **for** $i = srcBufferOffset$ to $srcBufferOffset + srcBufferCount - 1$ **do**
10: $srcVal \leftarrow srcRGBYBuffer[i]$
11: $srcID \leftarrow srcIDBuffer[i]$
12: $tmpVal \leftarrow tmpVal + dimValBuffer[srcID] \cdot srcVal$
13: **end for**
14: $groupLightDist[trgLoc] \leftarrow tmpVal$
15: **end function**

Die einzige threadspezifische Komponente ist der Parameter *instID*. Er enthält eine ID, die eindeutig dem Thread zugewiesen werden kann. Die IDs der Threads werden von 0 an fortlaufend durchnummeriert. Durch diese Information kann sichergestellt werden, dass die Berechnungen einerseits vollständig und andererseits nicht doppelt erfolgen. Konkret wird die *instID* in den Zeilen 4 bis 7 als Index zum Auslesen des *trgBuffer* eingesetzt. Jeder Thread berechnet also den RGBY-Vektor für ein Koordinatenpaar *trgLoc* der Gruppenlichtverteilung. Die Schreibweise *trgBuffer[x][y]* meint die *y*-te Komponente des *x*-ten Eintrags aus *trgBuffer*. Auf die Bedeutung der Komponenten der Vektoren aus $\mathbb{N}^4$ wird bereits in der Beschreibung von Algorithmus 5 ausführlich eingegangen. Durch die For-Schleife in den Zeilen 9 bis 13 wird über alle Einzellichtquellen iteriert, die am Koordinatenpaar *trgLoc* einen Beitrag zur Gruppenlichtverteilung leisten. Im Detail wird der RGBY-Vektor der Einzellichtverteilung ausgelesen, der für die betrachteten

Zielkoordinaten relevant ist. Er wird in *srcVal* gesichert und muss anschließend mit dem Dimmwert skaliert werden (Zeile 12). Dazu wird die ID *srcID* der Einzellichtquelle aus dem *srcIDBuffer* ausgelesen, welche in Zeile 12 wiederum als Index dient, um den zur Lichtquelle gehörigen Dimmwert aus *dimValBuffer* zu erhalten. Das Produkt aus diesem Dimmwert und dem RGBY-Vektor liefert den momentanen Beitrag der Einzellichtquelle zum betrachteten Pixel der Gruppenlichtverteilung. Die Beiträge aller Einzellichtquellen werden in der lokalen Varibale *tmpVal* summiert. Abschließend wird der aus der Gesamtheit aller Einzellichtquellen resultierende RGBY-Vektor an das Koordinatenpaar *trgLoc* der Gruppenlichtverteilung *groupLightDist* geschrieben.

Da jeder Thread des Computeshaders den RGBY-Vektor eines Koordinatenpaars *trgLoc* der Gruppenlichtverteilung exklusiv ermittelt, ist die Gruppenlichtverteilung nach dem vollständigen Durchlauf aller Threads vollständig bestimmt.

Die ermittelte Gruppenlichtverteilung wird schließlich durch Zeile 10 des Algorithmus 6 in die Gesamtlichtverteilung des Scheinwerfers integriert. Diese Integration erfolgt durch einen *Blit*-Befehl, welcher es ermöglicht, Texturen ohne Bezug zu Geometrien mit der Grafikpipeline zu rendern. Dazu wird ein geometrisches Primitiv erzeugt, welches das Renderziel vollständig ausfüllt.

In der hier betrachteten Anwendung ist die Gruppenlichtverteilung die Textur, welche in die Gesamtlichtverteilung als Renderziel geschrieben wird. Die geeignete Gestaltung der Vertex und Fragment Shader erlauben die korrekte Integration der Lichtverteilungen. Der Vertex-Shader wird als Pseudocode in Algorithmus 8 wiedergegeben. Gleichzeitig veranschaulicht Bild 6-8 die implementierten Vorgänge.

Da das Primitiv durch seine Rechteck-Geometrie aus nur vier Vertices besteht, werden lediglich vier Threads des im Algorithmus 8 formulierten Vertex-Shaders aufgerufen. Die vier Vertices sind in Bild 6-8 mit $v_1, v_2, v_3$ und $v_4$ bezeichnet. Nach der Transformation in den Clipspace (Zeile 8) haben sie die $x$- und $y$-Koordinaten $v_1 = (-1, -1), v_2 = (1, -1), v_3 = (1, 1)$ und $v_4 = (-1, 1)$. Ziel ist es, die Vertices so zu verschieben, dass das von ihnen beschriebene Rechteck nur den Bereich bedeckt, den die Gruppenlichtverteilung in der Gesamtlichtverteilung beeinflusst. Zuerst wird in Zeile 10 die in Bild 6-8a dargestellte Koordinatentransformation vorgenommen. Die Gruppenlichtverteilung, welche sich zuvor über den Bereich $[-1, 1] \times [-1, 1]$ erstreckte, entspricht nun dem Bereich $[0, 1] \times [0, 1]$. Entsprechend werden aus den Vertices $v_1, v_2, v_3$ und $v_4$ die transformierten Vertices $v'_1 = (0, 0), v'_2 = (1, 0), v'_3 = (1, 1)$ und $v'_4 = (0, 1)$. Zur korrekten Beschreibung des Einflussbereichs der Gruppenlichtverteilung erfolgt in den Zeilen 11 und 12 eine weitere Verschiebung der Vertices. Diese Verschiebung basiert auf den Winkelgrenzen der Gruppen- und Gesamtlichtverteilungen in horizontaler und vertikaler Richtung. Auf Basis dieser Werte lässt sich der Einflussbereich der Gruppenlichtverteilung normiert auf den Winkelbereich der Gesamtlichtverteilung formulieren. Die Vertices $v''_1, v''_2, v''_3$ und $v''_4$ in Bild 6-8b geben die Lage der Gruppenlichtverteilung gemäß der Verschiebungsvorschrift wieder. Mit Zeile 13 wird die Koordinatentransformation aus Bild 6-8a rückgängig gemacht (s. Bild 6-8c). Die Lage und Skalierung der Gruppenlichtverteilung relativ zur Gesamtlichtverteilung ist damit sichergestellt. Die *uv*-Koordinaten, die zur Adressierung der Textur dienen, können übernommen werden, da sie normiert vorliegen. Sie haben die Werte $(0, 0), (1, 0), (1, 1)$ und $(0, 1)$ und bilden somit die Gruppenlichtverteilung vollständig auf das transformierte Rechteck ab.

**Algorithm 8** Overall Combiner Shader (Vertex Stage)

**require:** $globalBounds \in \mathbb{R}^4$, $groupBounds \in \mathbb{R}^4$
**function** COMBINEOVERALLVERT($v_{in} \in \mathbb{R}^4, {}_T u, {}_T v \in \mathbb{R}$)
  **local variables:** $v_c, v_{out} \in \mathbb{R}^4$, $v, v', v'', v''' \in \mathbb{R}^2$
  $globalWidth \leftarrow globalBounds.right - globalBounds.left$
  $globalHeight \leftarrow globalBounds.top - globalBounds.down$
  $groupWidth \leftarrow groupBounds.right - groupBounds.left$
  $groupHeight \leftarrow groupBounds.top - groupBounds.down$
  $v_c \leftarrow P \cdot V \cdot M \cdot v_{in}$
  $v \leftarrow \begin{pmatrix} v_c.x \\ v_c.y \end{pmatrix}$
  $v' \leftarrow 0.5 \cdot v + \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$
  $v''.x \leftarrow \frac{groupBounds.left + groupWidth \cdot v'.x - globalBounds.left}{globalWidth}$
  $v''.y \leftarrow \frac{groupBounds.down + groupHeight \cdot v'.y - globalBounds.down}{globalHeight}$
  $v''' \leftarrow 2 \cdot \left( v'' - \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} \right)$
  $v_{out} \leftarrow \begin{pmatrix} v'''.x \\ v'''.y \\ v_c.z \\ v_c.w \end{pmatrix}$
  **return** $v_{out}, {}_T u, {}_T v$
**end function**

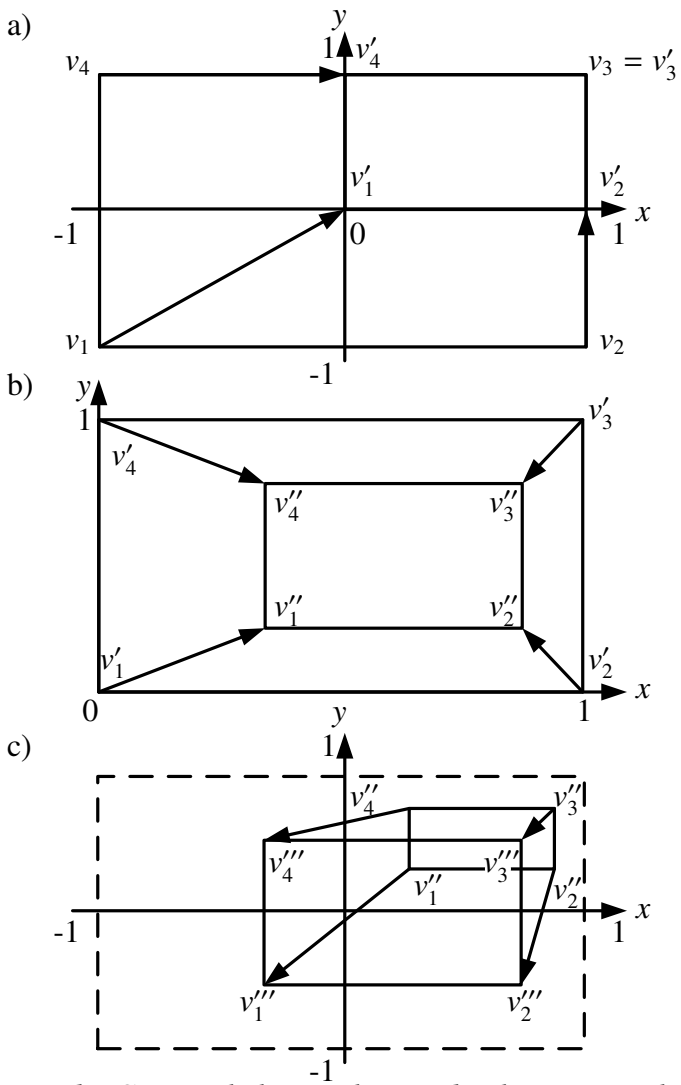


*Bild 6-8*: *Komposition der Gruppenlichtverteilungen durch geometrische Transformationen in der Vertex-Stufe des Combiner-Shaders.*

Schließlich werden der transformierte Positionsvektor $v_{out}$ und die unangetasteten *uv*-Koordinaten zurückgegeben. Im Rahmen der Rasterung wird das Rechteck in viele Fragmente diskretisiert. Die *uv*-Koordinaten werden basierend auf den Werten in den Vertices über alle Fragmente interpoliert. Es schließt sich der durch den Algorithmus 9 formulierte Fragment-Shader an. Für jedes Fragment des Rechtecks wird ein Thread des Shaders initiiert.

Der Fragment-Shader liest lediglich für das jeweilige Fragment die Textur *groupLightDist*, welche die Gruppenlichtverteilung repräsentiert, an den entsprechenden *uv*-Koordinaten aus. Da der Texturzugriff über *uv*-Koordinaten erfolgt, sind die Werte ${}_Tu$ und ${}_Tv$ nicht als Zeilen- und Spaltenindizes, sondern als kontinuierliche Koordinaten zu verstehen. Der in der Variabeln *val* gespeicherte RGBY-Vektor ergibt sich also durch die bilineare Interpolation der vier umgebenden Einträge in *groupLightDist*. Der Fragment Shader gibt *val* bei aktivem additiven Blending zurück. So wird der bisherige Wert in der Gesamtlichtverteilung nicht überschrieben, sondern auf den Beitrag der Gruppenlichtverteilung addiert. Ein

**Algorithm 9** Overall Combiner Shader (Fragment Stage)

1: **require:** Blend Mode: Additive Blending, $groupLightDist \in (\mathbb{R}^4)^{M_g \times N_g}$
2: **function** COMBINEOVERALLFRAG($_T u$, $_T v$)
3: **local variables:** $val \in \mathbb{R}^4$
4: $val \leftarrow groupLightDist[_T u, _T v]$
5: **return** val
6: **end function**

Rücksetzen der Gesamtlichtverteilung darf nur zu Beginn der Lichtberechnung erfolgen und wird entsprechend durch Algorithmus 6 vorgenommen.

Die vorgestellten Compute-, Vertex- und Fragment-Shader operieren auf allen Gruppenlichtverteilungen. Mit dem Terminieren der Schleife in Algorithmus 6 sind die Gruppenlichtverteilungen vollständig berücksichtigt worden. Als finales Resultat findet sich die gesuchte Gesamtlichtverteilung in der Textur $L_\Sigma$ des Algorithmus 6. Sie ist die Basis für das Rendering der Scheinwerfereinflüsse innerhalb der Szene.

### 6.2.4 Vorteile der Implementierung

Die vorgestellte Implementierung weicht vom intuitiven Vorgehen ab. Es soll abschließend aufgezeigt werden, warum dieses Vorgehen sowohl hinsichtlich der Laufzeit als auch des Speicherbedarfs von Vorteil ist.

Um diesen Vergleich anstellen zu können, muss zunächst definiert werden, wie eine intuitive Implementierung gestaltet sein könnte. Es liegt nahe, sich am mathematischen Formalismus nach Abschnitt 6.2.2 zu orientieren. Entsprechend Gleichung (6-7) werden die Lichtverteilungen aller Einzellichtquellen des Scheinwerfers sukzessive addiert und führen auf diese Weise zur Gesamtlichtverteilung. Auch im Rahmen dieser Arbeit folgte eine erste Umsetzung nach diesem Prinzip. Dazu können prinzipiell die Vertex- und Fragment-Shader 8 und 9 eingesetzt werden. Eine CPU-seitige Organisation des Ablaufs folgt dann der in Algorithmus 10 formulierten Logik. Die Einzellichtquellen des Scheinwerfers

**Algorithm 10** Intuitive Combine

1: **require:** $headlampSystem$, $lightDistList$ as List of $(\mathbb{R}^4)^{M_k \times N_k}$, $dimValues \in [0,1]^K$, $L_\Sigma \in (\mathbb{R}^4)^{M_\Sigma \times N_\Sigma}$
2: **function** INTUITIVECOMBINE($dimValues$)
3: $clear \quad L_\Sigma$
4: **for all** $Light\ light \in headlampSystem.lights$ **do**
5: $IntuitiveCombinerShader.dimValue \leftarrow dimValues[light.ID]$
6: $L_\Sigma \leftarrow Blit(lightDistList[light.ID], L_\Sigma, IntuitiveCombine)$
7: **end for**
8: **end function**

werden durch den *Intuitive Combiner Shader* iterativ zur Gesamtlichtverteilung addiert. Eine Gruppierung der Lichtquellen findet nicht statt. Ebenso entfällt die in Abschnitt 6.2.3 beschriebene Neustrukturierung der importierten Daten. Der *Intuitive Combiner*

*Shader* weist eine hohe Ähnlichkeit zum *Overall Combiner Shader* auf. Es handelt sich ebenfalls um einen Vertex- und einen Fragment-Shader im Kontext der Rendering Pipeline. Der Vertex-Shader ist nahezu identisch zu Algorithmus 8 und wird deshalb nicht durch Pseudocode formuliert. Als einziger Unterschied ist zu erwähnen, dass die Variable *groupBounds* anstelle der Winkelgrenzen der Gruppenlichtverteilung die Winkelgrenzen der Einzellichtverteilung der gerade betrachteten Lichtquelle *light* beinhalten muss.

Innerhalb des Fragment Shaders sind hingegen Änderungen erforderlich, um eine Berücksichtigung des momentanen Dimmwerts zu ermöglichen. In den Gruppenlichtverteilungen des zuvor präsentierten Verfahrens sind diese bereits inkludiert, da der vorgeschaltete Compute-Shader sie berücksichtigt. Um die Dimmwerte in der intuitiven Implementierungsvariante einzubeziehen, liest der in Algorithmus 11 formulierte Fragment-Shader den RGBY-Vektor aus der Lichtverteilung aus und skaliert ihn mit dem momentanen Dimmwert *dimValue* der Lichtquelle. Die CPU setzt diesen Dimmwert durch Zeile 5 im Algorithmus 10, bevor sie den Shader aufruft. Im Übrigen bleibt die Funktionsweise des Fragment-Shaders ebenfalls nah an Algorithmus 9. Insbesondere muss additives Blending eingesetzt werden, damit jede neu verarbeitete Lichtquelle die bisherigen Werte innerhalb der Gesamtlichtverteilung nicht überschreibt, sondern addiert.

**Algorithm 11** Intuitive Combiner Shader (Fragment Stage)

1: **require:** Blend Mode: Additive Blending, $lightDist \in (\mathbb{R}^4)^{M_k \times N_k}$, $dimValue \in [0, 1]$
2: **function** INTUITIVECOMBINEFRAG($_T u$, $_T v$)
3: **local variables:** $val \in \mathbb{R}^4$
4: $val \leftarrow dimValue \cdot lightDist[_T u, _T v]$
5: **return** val
6: **end function**

Stellt man den intuitiven Ansatz und die Implementierung nach Abschnitt 6.2.3 gegenüber, ergeben sich prinzipielle Unterschiede, aus welchen die Überlegenheit der hier verwendeten Lösung resultiert. Sie äußert sich sowohl in einer reduzierten Berechnungsdauer zur Laufzeit, als auch in einem geringeren Speicherbedarf. Diese Vorteile werden in den folgenden Unterabschnitten herausgearbeitet.

### Komplexität

Zur Erfüllung der Anforderungen **A4** (Skalierbarkeit bezüglich der Pixelanzahl) und **A6** (Echtzeitfähigkeit und Latenz) ist eine kurze Berechnungsdauer zur Laufzeit von höchster Priorität. Tests des intuitiven Ansatzes zeigen, dass die Anzahl der Lichtquellen auf wenige hundert pro Scheinwerfer beschränkt sein muss, um den Anforderungen gerecht zu werden. Mit Blick auf die modernen Scheinwerfertechnologien (Vgl. Bild 4-1) eignet sich der intuitive Ansatz lediglich für die Simulation niedrig aufgelöster Matrixsysteme.

Das prinzipbedingte Problem des intuitiven Ansatzes ist seine Eingangssensitivität. Seine Berechnungsdauer steht in einem proportionalen Verhältnis zu der Anzahl von Lichtquellen $K$. Orientiert an Algorithmus 10 ist für jede Lichtquelle ein erneuter Aufruf des Vertex- und Fragment-Shaders notwendig. Der Aufwand für den Vertex-Shader ist vernachlässigbar, da dieser auf nur vier Vertices pro Einzellichtverteilung operiert. Die Threads des

Fragment-Shaders richten sich hingegen nach der Auflösung der Gesamtlichtverteilung $M_\sum \times N_\sum$ und dem Einflussbereich der gerade betrachteten Einzellichtverteilung innerhalb der Gesamtlichtverteilung. Deshalb entsteht für die Bestimmung der Gesamtlichtverteilung insgesamt ein Berechnungsaufwand von $O(K \cdot M_\sum \cdot N_\sum)$.

Der in Abschnitt 6.2.3 vorgestellte Ansatz umgeht den linearen Zusammenhang zwischen der Lichtquellenanzahl $K$ und dem Berechnungsaufwand durch die Umstrukturierung der Daten gemäß Abschnitt 6.2.3. Da der Import und das Preprocessing des Scheinwerfersystems vor der Simulation durchgeführt werden können und nicht für jede Simulation wiederholt werden müssen, wird die Erfüllung der Anforderungen **A4** und **A6** nicht gefährdet. Die zusätzliche Verarbeitungszeit des Preprocessings stellt somit keinen nennenswerten Nachteil dar. Vernachlässigt man die Gruppierung der Lichtquellen, ist zur Berechnung der Gesamtlichtverteilung nur ein Aufruf des Compute-Shaders notwendig. Er ist somit nicht eingangssensitiv. Es gilt jedoch zu beachten, dass innerhalb des Compute-Shaders eine For-Schleife implementiert ist. Demzufolge ist die Ausführungsdauer eines einzelnen Threads nicht konstant. Sie unterliegt der Anzahl von Iterationen der For-Schleife. Diese Anzahl sei mit $K_o$ bezeichnet und entspricht der maximalen Anzahl von Lichtquellen, die einen Lichtbeitrag zu dem betrachteten Pixel der Gruppen- bzw. Gesamtlichtverteilung liefern. Der Berechnungsaufwand des Algorithmus beträgt damit $O(K_o \cdot M_\sum \cdot N_\sum)$. Am Beispiel des HD84-Systems beträgt $K_o$ im Mittel 2,8. Generell gilt $K_o << K$. Insbesondere existiert keine Proportionalität zwischen der Lichtquellenzahl $K$ und dem Wert $K_o$. Mit wachsendem $K$ rücken zwar die Ausleuchtungsbereiche benachbarter Lichtquellen enger zusammen. Gleichzeitig werden sie jedoch kleiner und mit ihnen auch die Überlappungsbereiche. Es ist somit davon auszugehen, dass der Wert $K_o$ unabhängig von der Lichtquellenzahl durch eine verhältnismäßig kleine Konstante nach oben abgeschätzt werden kann, womit der Berechnungsaufwand des Verfahrens auf $O(M_\sum \cdot N_\sum)$ reduziert wird. Der Algorithmus ist somit ausschließlich ausgabesensitiv. Auf diese Weise ist es dem intuitiven Vorgehen mit einem Berechnungsaufwand von $O(K \cdot M_\sum \cdot N_\sum)$ insbesondere hinsichtlich der Anforderungen **A4** und **A6** prinzipbedingt überlegen.

Bisher wurde das zweistufige Kombinationsverfahren, das im ersten Schritt Gruppenlichtverteilungen und nachgelagert die Gesamtlichtverteilung ermittelt, ausgeblendet. Stattdessen wurde angenommen, dass das Scheinwerfersystem aus nur einer Gruppe besteht. Diese Annahme kann implementiert werden. Die vorherige Gruppierung reduziert den Rechenaufwand jedoch weiter, indem sie Lichtquellen mit lokaler Zusammengehörigkeit vorgelagert kombiniert und so die Auflösung $M_\sum \cdot N_\sum$ der Gesamtlichtverteilung auf den lokalen Bereich $M_g \cdot N_g$ der Gruppe $g$ verkleinert. Gleichzeitig entsteht neuer Aufwand durch die anschließende Fusion der Gruppen zur Gesamtlichtverteilung. Zusammengefasst ergibt sich ein Berechnungsaufwand von $O(\sum_{g=1}^{G} M_g \cdot N_g + G \cdot M_\sum \cdot N_\sum)$, wobei $G$ die Anzahl der Gruppen darstellt. Mit $M_g < M_\sum$ und $N_g < N_\sum \forall g \in 1, ..., G$ kann der Aufwand durch $O(G \cdot M_\sum \cdot N_\sum)$ abgeschätzt werden. Nimmt man weiterhin an, dass die Gruppenzahl $G$ durch eine kleine Konstante beschränkt ist, ergibt sich $O(M_\sum \cdot N_\sum)$. Die Gruppierung der Lichtquellen stellt also keinen prinzipiellen Vorteil dar. Durch die Gestaltung der Lichtmodule im Scheinwerfer führt sie in der Praxis dennoch zu deutlichen Laufzeiteinsparungen.

### Speicherbedarf

Auch hinsichtlich des Speicherbedarfs kann das hier vorgestellte Verfahren deutliche Vorteile gegenüber dem intuitiven Ansatz vorweisen. Bei der Abschätzung des erforderlichen Speichers genügt es, die Lichtverteilungen zu betrachten. Metadaten sind im Vergleich dazu vernachlässigbar. Der intuitive Ansatz hält sämtliche Lichtverteilungen des Scheinwerfers voneinander separiert im Speicher vor. Der minimale Speicherbedarf einer Lichtverteilung $L_k$ ergibt sich aus dessen Auflösung $M_k \times N_k$ und der Codierung der Datenpunkte. Jeder Datenpunkt stellt einen RGBY-Vektor dar, wobei die Einträge als Fließkommazahlen mit je 4 Byte codiert werden. Es ergibt sich somit eine Speicherbedarf von $16 \cdot M_k \cdot N_k$ Byte pro Lichtverteilung. Nimmt man vereinfachend an, dass alle Lichtverteilungen die gleiche Auflösung und den gleichen Vermessungsbereich vorweisen, ergibt sich mit der Lichtquellenzahl $K$ ein Speicherbedarf von $16 \cdot M_\sum \cdot N_\sum \cdot K$ Byte für einen Scheinwerfer. Am Beispiel des HD84-Systems mit einer Auflösung von $1000 \times 240$ und $K = 84$ werden für das HD-Modul etwa 308 $MiB$ (1 $MiB$ (Mebibyte) entspricht $2^{20}$ Byte) benötigt.

Der hier präsentierte Ansatz hält die Lichtverteilungen nicht in ihrer ursprünglichen Form im Speicher vor. Durch das in Abschnitt 6.2.3 vorgestellte Preprocessing werden die Informationen aller Lichtverteilungen in wenigen großen Buffern geschickt zusammengeführt. Insbesondere werden Pixel der Lichtverteilungen, die keinen oder einen vernachlässigbaren Lichtbeitrag liefern, nicht durch die neue Struktur abgebildet. Maßgeblich für den Speicherbedarf ist die Anzahl der Pixel aller Einzellichtquellen, die einen Lichtbeitrag liefern. Um einen fairen Vergleich mit dem intuitiven Ansatz vornehmen zu können, werden nur diejenigen Pixel verworfen, die einen Beitrag von exakt 0 liefern. In anderen Worten wird der Parameter „minLuminousIntensity“ in Algorithmus 4 auf 0 gesetzt. Am Beispiel des HD84-Systems existieren über alle Einzellichtquellen hinweg 543.519 Pixel, die einen Beitrag zur Lichtverteilung leisten. Hieraus können unmittelbar die Größen der Buffer *srcRGBYBuffer* und *srcIDBuffer* ermittelt werden, da sie dementsprechend 543.519 Elemente haben. Elemente des *srcRGBYBuffer* sind RGBY-Vektoren, wobei jeder Vektoreintrag durch 4 Byte (Float) codiert wird. Im *srcIDBuffer* sind Ganzzahlwerte enthalten, die ebenfalls durch jeweils 4 Byte (Integer) codiert werden. Es ergibt sich für beide Buffer somit ein Speicherbedarf von $543.519 \cdot 20\,Byte \approx 10\,MiB$. Davon entfallen ca. *2 MiB* auf den *srcIDBuffer* und ca. *8 MiB* auf den *srcRGBYBuffer*.

Die Größe des *trgBuffer* hängt hingegen von der Anzahl der Pixel in der Gruppen- bzw. Gesamtlichtverteilung ab, die von mindestens einer Einzellichtquelle einen Lichtbeitrag erhalten. Am Beispiel des HD84-Systems ist das für 175.575 Pixel der ingesamt $1000 \times 240$-Pixel großen Lichtverteilung der Fall. Das entspricht 73,15%. *trgBuffer* enthält für jeden der 175.575 Pixel vier Ganzzahlwerte. Es ergibt sich somit ein Speicherbedarf von $175.575 \cdot 16\,Byte \approx 3\,MiB$ für *trgBuffer*.

In Summe werden für alle Buffer insgesamt etwa *13 MiB* benötigt. Das entspricht nur 4,2% des Speicherbedarfs des intuitiven Ansatzes. Die Einsparungen durch das Preprocessing sind somit erheblich. Ursächlich hierfür ist vor allem das frühzeitige Verwerfen nicht relevanter Messpunkte in den Einzel- und Gruppen- bzw. Gesamtlichtverteilungen. Diese Messpunkte sind zwangsläufig in den Eingangsdaten enthalten, da die Konturen der Lichtverteilung nicht die Messbereichsgrenzen widerspiegeln. So haben Lichtverteilungen in der Regel runde oder ovale Geometrien, während die Messbereiche typischerweise eine rechteckige Gestalt bezüglich der Kugelkoordinaten aufweisen. Hinzu kommt, dass der

Messbereich bewusst deutlich größer gewählt wird, als es der Ausleuchtungsbereich der Lichtquelle erfordern würde, um unbeabsichtigtes Streulicht der Lichtquelle, welches zum Teil in stark abweichende Richtungen abgegeben wird, einzufangen.

## 6.3 Beleuchtung der Szene

Mit der Gesamtlichtverteilung ist die Charakteristik des Scheinwerferlichts vollständig beschrieben. Nun muss die Interaktion des Lichts mit der Umgebung modelliert werden. Aus den in Abschnitt 5.4.3 genannten Gründen kann dazu nicht auf eine existierende Lichtquelle der Unity-Engine zurückgegriffen werden. Stattdessen wird eine eigene Lichtquelle in die Engine integriert, deren grundsätzliche Struktur am Spotlicht der Unity-Engine orientiert ist.

Im nachfolgenden Abschnitt 6.3.1 wird dargestellt, wie die Integration einer selbst implementierten Lichtquelle innerhalb der Unity-Engine und insbesondere der Deferred Rendering Pipeline gelingt. Daran angeschlossen wird in Abschnitt 6.3.2 das Vorgehen der Beleuchtungsberechnung innerhalb der Szene thematisiert. Hierbei handelt es sich zum Großteil um etablierte Konzepte des Deferred Lighting. Da an diversen Stellen Modifikationen vorgenommen wurden, wird der grundsätzliche Ablauf dennoch vollständig dargestellt.

### 6.3.1 Integration der Lichtquelle

Im Rahmen des Architekturkapitels 5 wird in Abschnitt 5.4.3 gefolgert, dass im Rahmen von Hyperion eine neue Lichtquelle zu entwickeln ist, um die Anforderungen bei der virtuellen Abbildung eines HD-Scheinwerferlichts zu erfüllen. Weiterhin wird argumentiert, dass die Verwendung der Deferred Rendering Pipeline in Verbindung mit der hohen Zahl dynamischer Lichtquellen unvermeidbar ist. In der Konsequenz gilt es, ein eigenes Lichtquellenmodell in die Deferred Rendering Pipeline der Unity-Engine zu integrieren.

Zur Integration eigener Befehlssequenzen innerhalb der Rendering Pipeline sieht Unity sogenannte Commandbuffer vor [Uni20]. Diese erlauben an verschiedenen Stellen die Injektion eigener Befehle innerhalb der normalen Befehlsfolge des Renderings. In diesem Fall ist der Injektionsort „After Lighting“ von besonderem Interesse. In Bild 2-36 wurde bereits der grobe Ablauf des Deferred Renderings dargestellt. Die Scheinwerferlichtquellen werden im Anschluss an den Lighting Pass integriert und ebenfalls durch einen Shader mit Vertex- und Fragment-Stufe implementiert. Dieser Shader wird nachfolgend als Headlamp-Shader bezeichnet. Das Bild 6-9 stellt die modifizierte Deferred Rendering Pipeline dar. Über den Ort der Injektion „After Lighting“ wird sicher gestellt, dass der Headlamp-Shader den gleichen Kontext wie die Buit-In-Shader für die Lichtquellen der Unity-Engine vorfindet. Auf der einen Seite stehen ihm die Daten aus dem G-Buffer zur Verfügung und auf der anderen Seite ist sein Rendertarget der Frame-Buffer und somit die finale Bildschirmausgabe. Der Headlamp-Shader kann somit orientiert an den Built-In-Shadern implementiert werden. Eine weiterer Vorteil der Integration durch Commandbuffer ist der Erhalt der Kompatibilität zu den Standard-Lichtquellen der Unity-Engine, welche vor Beginn des Headlamp-Shaders durchlaufen werden. Der Headlamp-Shader wird einmal pro

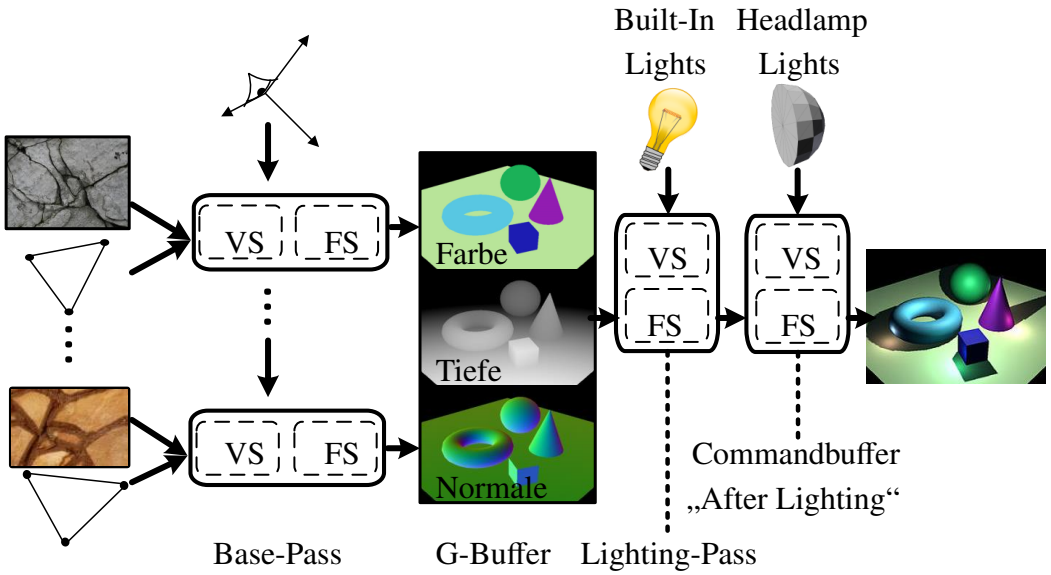


*Bild 6-9*: *Durch Commandbuffer-Injektion modifizierte Deferred Pipeline.*

Scheinwerfer ausgeführt. Insofern stellt der zu injizierende Commandbuffer eine Iteration von Shader-Aufrufen dar. Die Erstellung des Commandbuffers geschieht CPU-seitig und erfolgt im CustomLight-Skript, welches eine Komponente des jeweiligen Scheinwerfers innerhalb des Szenengraphen ist (s. Abschnitt 5.4.3). Das Flussdiagramm 6-10 veranschaulicht die sich zur Laufzeit stetig wiederholende Erzeugung des Commandbuffers.

Einmalig zu Beginn der Simulation wird das Material für das Rendering des Scheinwerferlichts erzeugt. Dieses beinhaltet die Elemente, welche die Art und Weise des Renderings einer Geometrie festlegen. Hierzu gehören insbesondere der Headlamp-Shader und die zuvor bestimmte Gesamtlichtverteilung des Scheinwerfers, aber auch Parameter, wie der horizontale und vertikale Winkelbereich der Ausleuchtung und die maximale Reichweite des Scheinwerferlichts.

Nach der Initialisierung beginnt ein sich in jedem Frame wiederholender Ablauf. Wie in Abschnitt 2.3.3 beschrieben, wird zur Selektion der vom Licht beeinflussten Fragmente des G-Buffers ein fiktives Lichtvolumen gerendert. Da die optimale Form dieses Lichtvolumens vom Vermessungsbereich des jeweiligen Scheinwerfersystems abhängt, wird dieses Lichtvolumen in Hyperion prozedural erstellt. Im Bild 6-11 werden die Lichtvolumen für drei unterschiedliche Winkelbereiche dargestellt. Dabei sind die Ausrichtungen der Koordinatensysteme, in deren Ursprung sich die Lichtquelle befindet, spaltenweise konstant. Weiterhin entspricht die $z$-Achse der Lichtmittelachse ($\theta = \varphi = 0°$). Polachse der Kugelkoordinaten für die Lichtverteilungen ist die $y$-Achse. Die maximal möglichen Ausleuchtungswinkelintervalle sind horizontal und vertikal $[-90°, 90°]$. Sie führen zu einer Halbkugel, wie sie in Bild 6-11a dargestellt ist. In der Variante c wird der asymmetrische Ausleuchtungsbereich der HD84-Matrix zugrunde gelegt, welcher horizontal den vollen Messbereich ausschöpft, während sich der vertikale Messbereich über nur 45° erstreckt. Neben den Winkelbereichen kann die Tesselierung vorgegeben werden, welche die Anzahl der Vertices bestimmt, die zur Approximation der Form verwendet werden. Für den beschriebenen Zweck ist eine eher grobe Tesselierung ausreichend.

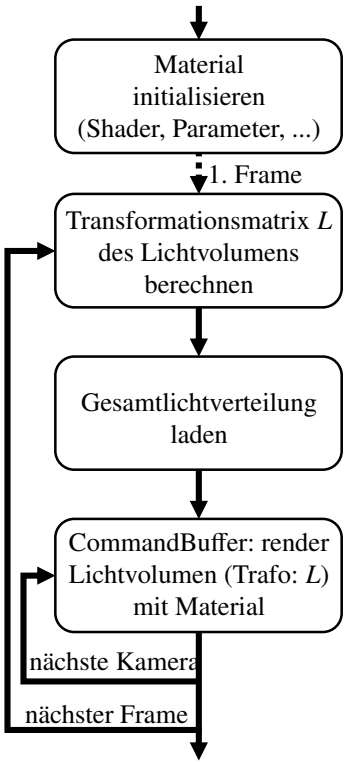


*Bild 6-10*: *Iterative Erzeugung des Commandbuffers zur Laufzeit.*

Um das so gestaltete Lichtvolumen gemäß der beschriebenen Lage zu orientieren, wird die Transformationsmatrix $L$ vom Licht- in das Weltkoordinatensystem gebildet. Neben der Translation und Rotation wird die maximale Reichweite der Lichtquelle durch die gleichmäßige Skalierung des Lichtvolumens entlang aller Achsen bezüglich des Zentrums $x = y = z = 0$ in $L$ berücksichtigt. Dieses Vorgehen entspricht einer Anpassung des Kugelradius. $L$ unterliegt aufgrund der Fahrzeugbewegung dynamischen Änderungen. Genauso kann sich die Gesamtlichtverteilung permanent ändern, weshalb sie ebenfalls in jedem Frame aktualisiert werden muss. Schließlich wird das Rendering des Lichtvolumens durch den Headlamp-Shader und den im Material spezifizierten Parametern als Anweisung für die GPU in den „After Lighting“-Commandbuffer geschrieben. Sollten mehrere Kameras aktiv sein, wird das Rendering für jede Kamera separiert vorgenommen. In diesem Fall muss der Render-Befehl für jede Kamera injiziert werden. Ausgeführt werden die im Commandbuffer aufgelisteten Befehle nach dem eigentlichen Lighting-Pass der Unity-Engine (s. Bild 6-9). Der beschriebene Ablauf wiederholt sich Frame für Frame.

### 6.3.2 Implementierung

In Abschnitt 6.3.1 wird die Integration der eigenen Lichtquelle zum Rendering von Scheinwerferlicht innerhalb der Deferred Pipeline beschrieben. Dieser Abschnitt thematisiert nun die Vorgänge innerhalb des Headlamp-Shaders, welcher die Beleuchtung der virtuellen Szene durch das Scheinwerferlicht implementiert. Entsprechend dem üblichen Vorgehen im Lighting-Pass der Deferred Pipeline setzt sich der Headlamp-Shader aus Vertex- und Fragment-Stufe zusammen. Diese werden nachfolgend detailliert erläutert. In Bild 6-12

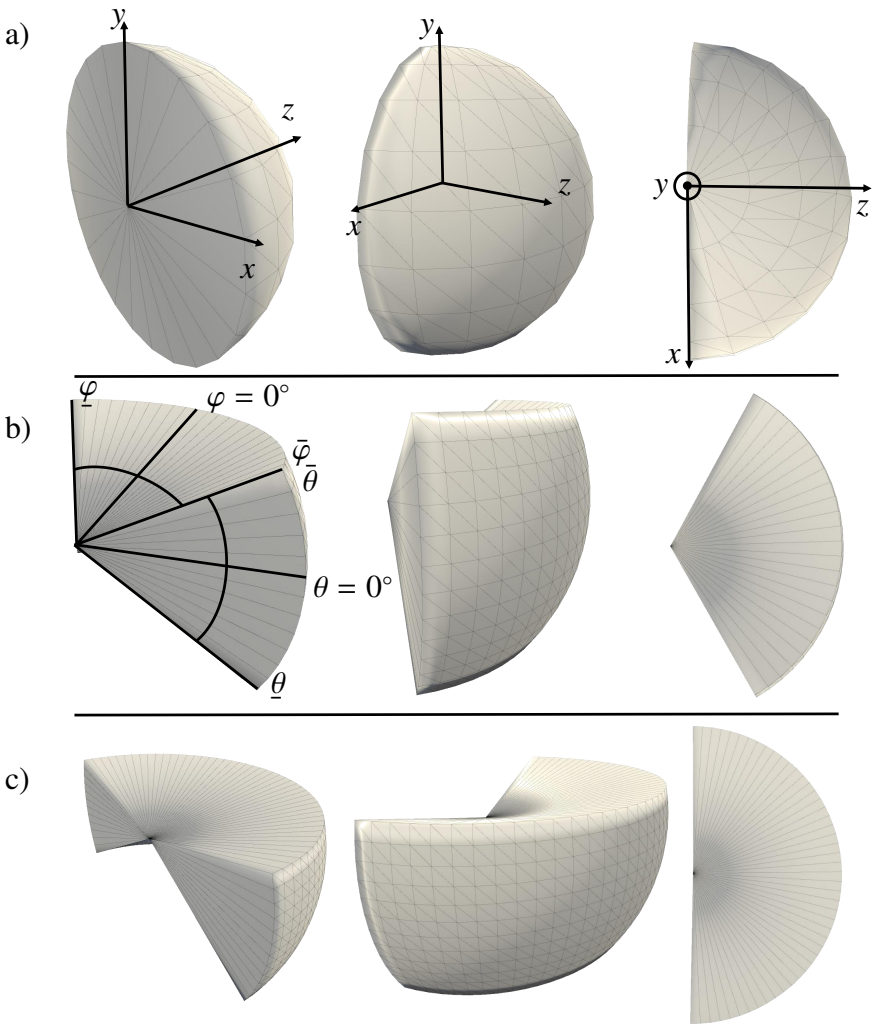


*Bild 6-11*: *Prozedural generiertes Lichtvolumen zur optimalen Anpassung an die horizontalen und vertikalen Winkelbereiche* $[\varphi, \bar{\varphi}] \times [\underline{\theta}, \bar{\theta}]$ *der Lichtverteilung ((a)* $[-90°, 90°] \times [-90°, 90°]$*; b)* $[-60°, 60°] \times [-30°, 30°]$*; c)* $[-90°, 90°] \times [-30°, 15°]$*).*

ist eine stark reduzierte Szene skizziert, welche die nachfolgend verwendeten Größen in einen bildlichen Zusammenhang stellt und beim Verständnis unterstützen soll.

Die relativen Lagen aller Szenenelemente werden im Weltkoordinatensystem *w* beschrieben. Das Kamerakoordinatensytem *e* beschreibt die Elemente der Szene aus Sicht der Kamera. Das View Frustum, welches das Volumen der sichtbaren Punkte begrenzt, ist in Bild 6-12 in blau dargestellt (s. Abschnitt 2.3.1). Dabei hat die Near Clipping Plane den Abstand *n* und die Far Clipping Plane den Abstand *f* zur Kamera. Um darüber hinaus die relevanten geometrischen Größen bei der Anwendung des Beleuchtungsmodells an einem Objekt der Szene visualisieren zu können, wird in Bild 6-12 beispielhaft ein Zylinder betrachtet. Er verfügt über das lokale Objektkoordinatensystem *m*. Im Zentrum des in gelb skizzierten Lichtvolumens befindet sich die Lichtquelle. Im Lighting-Pass ist dieses Lichtvolumen mit seinem lokalen Lichtkoordinatensystem *l* das zu rendernde Objekt. Die übrige Szene wurde bereits im Base-Pass in den G-Buffer gerendert.

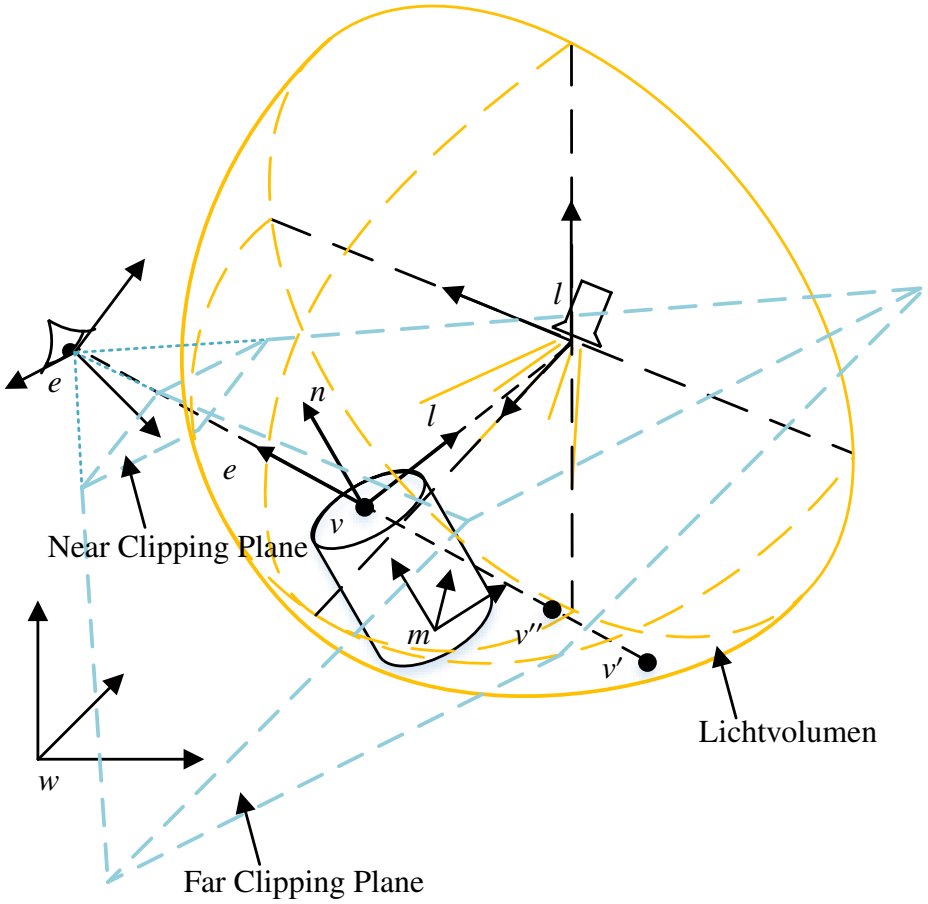


*Bild 6-12*: *Einfache Szene mit Weltkoordinatensystem w, Kamerakoordinatensystem e, Lichtkoordinatensystem l und einem beispielhaften Objekt mit dem Objektkoordinatensystem m.*

### Vertex-Shader

Der Vertex-Shader operiert auf allen Vertices des in Bild 6-11 dargestellten Lichtvolumens, dessen genaue Gestalt zuvor prozedural generiert wurde. Das Lichtvolumen schlüpft in die Rolle eines gewöhnlichen Szenenobjekts im Base-Pass. In Bild 6-12 ist das Lichtvolumen als ideale Halbkugel (Vgl. 6-11a) in gelb dargestellt. Auf diesem wird beispielhaft der Vertex $v'$ betrachtet. Gemäß den Abschnitten 2.3.1 und 2.3.2 ist die primäre Aufgabe des Vertex-Shaders die Transformation der Vertices vom lokalen Koordinatensystem über Welt- und Kamerakoordinatensystem bis in den Clipspace. Zur Abgrenzung von gewöhnlichen Szenenobjekten wird die Transformationsmatrix von lokalen ins Weltkoordinatensytem im Fall des Lichtvolumens mit $L$ (anstelle von $M$) bezeichnet. Die Gestalt von $L$ wurde in Abschnitt 6.3.1 bereits diskutiert. Auch wenn das Lichtvolumen im gerenderten Frame nicht sichtbar sein wird, ist dessen Transformation in den Clipspace notwendig, um das Volumen zu umreißen, welches potentiell durch die Lichtquelle beeinflusst werden kann. Durch die nachgelagerte Rasterung wird aus diesem Volumen ein Bereich von Fragmenten, für welche die Beleuchtungsberechnungen zu vollziehen sind.

Die genauen Berechnungsschritte des Vertex-Shaders werden in Algorithmus 12 durch Pseudocode beschrieben. Da im Deferred Rendering zwangsläufig Per-Fragment-Lighting

eingesetzt wird, gestalten sich die Berechnungen in der Vertex-Stufe verhältnismäßig übersichtlich. Die eigentliche Anwendung des Beleuchtungsmodells erfolgt im nachgelagerten Fragment-Shader.

An jeden Thread des Vertex-Shaders wird ein Vertex ${}_l v'$ des Lichtvolumens aus Bild 6-11 übergeben. Dieses liegt initial im Lichtkoordinatensystem $l$ vor, da dieses dem Objektkoordinatensystem des Lichtvolumens entspricht. Durch Multiplikation mit den verschiedenen Transformationsmatrizen kann ${}_l v'$ zu ${}_e v'$ überführt werden und liegt somit im Clipspace vor. Im Detail erfolgt die Transformation vom Licht- in das Weltkoordinatensystem durch die Matrix $L$, vom Welt- in das Kamerakoordinatensystem durch die Matrix $V$ und vom Kamerakoordinatensystem in den Clipsspace durch die Matrix $P$ (s. Abschnitt 2.3.1). Der Clipspace $c$ wird in Bild 6-12 nicht aufgeführt. Entsprechend den Ausführungen in Abschnitt 2.3.1 hat er den gleichen Ursprung und die gleiche Achsenausrichtung wie $e$. Die Koordinaten werden jedoch perspektivisch verzerrt, um das nachfolgende Clipping und die Überführung in Bildschirmkoordinaten zu vereinfachen. Die Vertexkoordinaten ${}_c v'$ bilden die Grundlage für die Rasterung, die zwischen Vertex- und Fragment-Stufe stattfindet. ${}_c v'$ zählt deshalb auch zu den Rückgabewerten der Vertex-Stufe. Darüber hinaus wird die normierte Frameposition ${}_c v'_{uv}$ des Vertex $v'$ ermittelt. Wenn ${}_c v'$ im View Frustum enthalten ist, so liegen seine $x$- und $y$-Koorinaten vor der perspektivischen Division im Intervall $[-{}_c w, {}_c w]$. Durch die Zeilen 5 und 6 und der späteren perspektivischen Division im Fragment-Shader bewegen sich die $x$- und $y$-Koordinaten von ${}_c v'_{uv}$ im Intervall $[0, 1]$, wenn $v'$ sich innerhalb des View Frustum befindet. Somit können die Koordinaten von ${}_c v'_{uv}$ als Texturkoordinaten zur Addressierung des G-Buffers genutzt werden. Aus diesem Grund stellen sie die zweite Rückgabe der Vertex-Stufe dar. Letztlich wird der Vektor ${}_e v'$ von der Kamera zum Vertex $v'$ durch Multiplikation von ${}_l v'$ mit $L$ und $V$ bestimmt. Dieser Wert wird neben ${}_c v'_{uv}$ zur Rekonstruktion der räumlichen Position der Fragmente innerhalb der Fragment-Stufe benötigt und bildet deshalb die dritte Ausgabe der Vertex-Stufe. Auf die beschriebene Weise werden alle Vertices des Lichtvolumens parallel verarbeitet.

**Algorithm 12** Headlamp Shader (Vertex Stage)

1: **require:** ${}_l v' \in \mathbb{R}^4$ vertex of light volume as homogeneus coordinates in $l$
2: **function** HEADLAMPVERT(${}_l v' \in \mathbb{R}^4$)
3: **local variables:** ${}_c v', {}_c v'_{uv}, {}_e v' \in \mathbb{R}^4$
4: ${}_c v' \leftarrow P \cdot V \cdot L \cdot {}_l v'$ ▷ vertex in $c$
5: ${}_c v'_{uv}.x \leftarrow \frac{1}{2} \cdot {}_c v'.x + \frac{1}{2} \cdot {}_c v'.w$ ▷ transform to screen space
6: ${}_c v'_{uv}.y \leftarrow \frac{1}{2} \cdot {}_c v'.y + \frac{1}{2} \cdot {}_c v'.w$ ▷ transform to screen space
7: ${}_c v'_{uv}.z \leftarrow {}_c v'.z$
8: ${}_c v'_{uv}.w \leftarrow {}_c v'.w$
9: ${}_e v' \leftarrow V \cdot L \cdot {}_l v'$ ▷ vertex in $e$
10: **return** ${}_c v', {}_c v'_{uv}, {}_e v'$
11: **end function**

Nach der Vertex-Stufe wird zunächst das Clipping vollzogen. Es verbleiben die Vertices, welche sich innerhalb des View Frustums befinden. Auf diese wird die perspektivische Division angewendet, sodass ${}_c v'$ zu ${}_n v'$ im NDC-System überführt wird (s. Abschnitt 2.3.1). Anhand der Koordinaten von ${}_n v'$ erfolgt die Rasterung. Die übrigen Rückgaben der Vertex-Stufe werden bilinear auf die Fragmente interpoliert.

### Fragment-Shader

Im Anschluss an die vorangegangenen Schritte wird die in Algorithmus 13 als Pseudocode notierte Fragment-Stufe ausgeführt. Die Fragment-Stufe prozessiert alle Fragmente, die durch das gerasterte Lightvolumen getroffen werden. Als Parameter erhält jeder Thread des Fragment-Shaders die Interpolationsergebnisse der Rückgaben ${}_{c}v'_{uv}$ und ${}_{e}v'$ der Vertex-Stufe, für welche weiterhin die gleichen Bezeichner verwendet werden.

Der Pseudocode kann in fünf logische Blöcke unterteilt werden. Im ersten Block (Zeile 4 bis 10) wird die räumliche Position des im aktuellen Fragment sichtbaren Oberflächenelements der Szene rekonstruiert. Bezogen auf das Minimalbeispiel 6-12 handelt es sich um den Punkt $v$ auf dem zylinderförmigen Szenenobjekt. Der G-Buffer allein würde diese Rekonstruktion nicht ermöglichen. Durch die zusätzlichen Ausgaben der Vertex-Stufe können die fehlenden Informationen jedoch ergänzt werden. Ausgelöst wurde der betrachtete Thread des Fragment-Shaders durch den Punkt $v'$, welcher zur Oberfläche des Lichtvolumens gehört. Da das Lichtvolumen nicht dargestellt werden soll, ist die genaue Lage von $v'$ nicht von Bedeutung. Lediglich die Blickrichtung auf $v'$, welche durch die schwarz gestrichelte Linie in Bild 6-12 visualisiert wird, ist maßgeblich. Sie kennzeichnet alle Punkte im Raum, die der gleichen Bildschirmposition entsprechen. Dort, wo die Linie ausgehend von der Kamera erstmalig auf ein Szenenobjekt trifft, befindet sich der anstelle von $v'$ zu rendernde Punkt $v$. Die Blickrichtung der Kamera auf die Punkte $v$, $v'$ und $v''$ ist durch den Vektor ${}_{e}v'$ bekannt. Zudem enthält der G-Buffer die auf die Far Clipping Plane normierten Tiefenwerte aller Fragmente. Nach der perspektivischen Division in Zeile 5 können die Einträge aus ${}_{c}v'_{uv}$ genutzt werden, um den G-Buffer an der korrekten Stelle auszulesen (Zeile 6). Da die Tiefe des Punkts $v$ bisher nur in normierter Form bekannt ist, wird in einem Zwischenschritt der Punkt ${}_{e}v''$ ermittelt, welcher sich in der gleichen Blickrichtung befindet, aber exakt auf der Far Clipping Plane liegt. Diesen erhält man durch die Skalierung des Punkts ${}_{e}v'$ mit dem Quotienten aus der Tiefe $f$ der Far Clipping Plane und der $z$-Koordinate des Punkts ${}_{e}v'$ (Zeile 4). Die anschließende Skalierung des Punkt ${}_{e}v''$ mit dem normierten Tiefenwert $z_{uv}$ des G-Buffers führt schließlich auf die Position von $v$ im Kamerakoordinatensystem $e$. Diese kann durch Multiplikation mit der Inversen von $V$ in das Weltkoordinatensystem überführt werden (Zeile 8). Neben der Position von $v$ ist auch die Blickrichtung auf $v$ für das Beleuchtungsmodell relevant (s. Vektor $e$ in Bild 2-41). In den Zeilen 9 und 10 wird der normierte Blickvektor ${}_{w}e$ bezogen auf das Weltkoordinatensystem berechnet.

Neben der Blickrichtung spielt auch die Lichteinfallsrichtung eine zentrale Rolle bei der Auswertung des Bleuchtungsmodells. Der Lichtvektor wird in den Zeilen 12 bis 14 bestimmt. Zuerst wird die Position ${}_{w}l$ der Lichtquelle im Weltkoordinatensystem aus der Matrix $L$ extrahiert (Zeile 23). Hierbei wird ausgenutzt, dass der Headlamp-Shader innerhalb der Deferred Pipeline ausschließlich das Lichtvolumen rendert, sodass die Transformationsmatrix $L$ vom Licht- zum Weltkoordinatensystem $w$ über alle Aufrufe des Fragment-Shaders konstant ist. Die Einträge von $L$ ergeben sich aus den Translationen, Rotationen und Skalierungen, die zur Positionierung des Lichtvolumens im Weltkoordinatensystem erforderlich sind. Allgemein kann ein Punkt ${}_{w}v$ in Weltkoordinaten durch die

**Algorithm 13** Headlamp Shader (Fragment Stage)

**require:** ${}_c v'_{uv}, {}_e v' \in \mathbb{R}^4$ position of $v'$ of light volume in $c$ and $e$, $f \in \mathbb{R}$ distance camera $\leftrightarrow$ far clipping plane, $G_{depth}, G_{normal}, G_{material}$ G-Buffer information, ${}_w v_{cam} \in \mathbb{R}^4$ camera position in $w$, $L_\Sigma \in (\mathbb{R}^4)^{M_\Sigma \times N_\Sigma}$ overall light distribution of current headlamp, $globalBounds \in \mathbb{R}^4$ angle bounds of $L_\Sigma$, $globalWidth, globalHeight \in \mathbb{R}$ angle ranges of $L_\Sigma$, $intens \in \mathbb{R}$ scale factor for brightness of headlight

**function** HEADLAMPFRAG(${}_c v'_{uv}, {}_e v'$)

**local variables:** ${}_e v'' \in \mathbb{R}^4$ position of $v''$ in $e$, ${}_e v, {}_w v \in \mathbb{R}^4$ position of $v$ in $e$ and $w$, ${}_T u_B, {}_T v_B \in \mathbb{R}$ G-Buffer coords of current fragment, $z_{uv} \in \mathbb{R}$ depth of current fragment, ${}_w l, {}_l l \in \mathbb{R}^4$ position of headlamp in $w$ and $l$, ${}_w v_{c,m} \in \mathbb{R}^4$ vector $v \rightarrow {}_w v_{cam}$, ${}_w v_{l,m} \in \mathbb{R}^4$ vector $v \rightarrow {}_w l$, $\theta, \varphi \in \mathbb{R}$ polar and azimuth angle of light ray, ${}_T u_L, {}_T v_L \in \mathbb{R}$ texture coords of light ray in $L_\Sigma$, ${}_w e \in \mathbb{R}^4$ eye vector at $v$ for light model, ${}_w n \in \mathbb{R}^4$ nprmal at $v$ for light model, $C_l, c_{l,att} \in \mathbb{R}^4$ (attenuated) light color as RGB-vector, $C_m \in \mathbb{R}^4$ reflection of material

${}_e v'' \leftarrow \frac{f}{{}_e v'.z} \cdot {}_e v'$ ▷ scale to far clipping distance (f)

$({}_T u_B, {}_T v_B) \leftarrow \frac{1}{{}_c v'_{uv}.w} \cdot {}_c v'_{uv}.xy$ ▷ buffer coords of $v'$ (same for $v$)

$z_{uv} \leftarrow G_{depth}({}_T u_B, {}_T v_B)$ ▷ norm. depth at screen position $({}_T u_B, {}_T v_B)$

${}_e v \leftarrow z_{uv} \cdot {}_e v''$ ▷ position of $v$ in $e$

${}_w v \leftarrow V^{-1} \cdot {}_e v$ ▷ position of $v$ in $w$

${}_w v_{c,m} \leftarrow {}_w c - {}_w v$ ▷ vector v→camera in $w$

${}_w e \leftarrow \frac{{}_w v_{c,m}}{|{}_w v_{c,m}|}$ ▷ direction v→camera in $w$

${}_w l \leftarrow L[1:4, 4]$ ▷ position of light in $w$

${}_w v_{l,m} \leftarrow {}_w l - {}_w v$ ▷ vector v→light in $w$

${}_w l \leftarrow \frac{{}_w v_{l,m}}{|{}_w v_{l,m}|}$ ▷ direction v→light in $w$

${}_l l \leftarrow -L^{-1} \cdot {}_w l$ ▷ direction light→v in $l$

$\theta \leftarrow \frac{\pi}{2} - \arccos {}_l l.y$ ▷ polar and azimuth angle

$\varphi \leftarrow \text{atan2}({}_l l.x, {}_l l.z)$ ▷ between light axis and ${}_l l$

${}_T u_L \leftarrow \frac{\varphi - globalBounds.left}{globalWidth}$ ▷ Light-Cookie $u$-coordinate

${}_T v_L \leftarrow \frac{\theta - globalBounds.down}{globalHeight}$ ▷ Light-Cookie $v$-coordinate

$C_l \leftarrow L_\Sigma({}_T u_L, {}_T v_L)$ ▷ light power in specific direction

$att \leftarrow \frac{1}{intens^2} \cdot {}_w v_{l,m} \cdot {}_w v_{l,m}$ ▷ light attenuation

$C_{l,att} \leftarrow att \cdot C_l$ ▷ attenuated light color

${}_w n \leftarrow G_{normal}({}_T u_B, {}_T v_B)$ ▷ normal at screen position $({}_T u_B, {}_T v_B)$

$C_m \leftarrow G_{material}({}_T u_B, {}_T v_B)$ ▷ material at screen position $({}_T u_B, {}_T v_B)$

**return** lightingModel($C_m, {}_w n, {}_w e, {}_w l, C_{l,att}$)

**end function**

Multiplikation seiner Lichtkoordinaten ${}_{l}v$ mit der homogenen Transformationsmatrix $L$ von $l$ nach $w$ transformiert werden:

$$ {}_{w}v = L \cdot {}_{l}v \text{ with } L = \begin{bmatrix} l_{11} & \dots & l_{14} \\ \vdots & & \vdots \\ l_{41} & \dots & l_{44} \end{bmatrix}. $$

Nach Gleichung (2-24) entsprechen die Einträge $l_{14}$, $l_{24}$ und $l_{34}$ der Translation von $l$ nach $w$ entlang der $x$-, $y$- und $z$-Richtung. Da sich die Lichtquelle im Ursprung des Lichtkoordinatensystems $l$ befindet, entspricht die Translation in $L$ gerade der Lichtposition ${}_{w}l$ im Weltkoordinatensystem. ${}_{w}l$ kann somit aus der vierten Spalte von $L$ ausgelesen werden. Auf dieser Basis kann der normierte Lichtvektor ${}_{w}l$ vom Objekt zur Lichtquelle im Weltkoordinatensystem bestimmt werden (Zeilen 13 und 14).

In den Zeilen 16 bis 21 findet die durch den Combiner-Shader bestimmte Gesamtlichtverteilung $L_{\sum}$ des Scheinwerfers Berücksichtigung. Zur Bestimmung der spektralen Lichtabgabe des Scheinwerfers auf das betrachtete Fragment sind zunächst der Polar- und Azimutwinkel des Lichtstrahls aus Sicht der Lichtquelle zu ermitteln (s. Abschnitt 3.1.1). Zu diesem Zweck wird die Lichteinfallsrichtung ${}_{w}l$ durch Multiplikation mit $L^{-1}$ in das Lichtkoordinatensystem $l$ übertragen und durch Negation in die Lichtausbreitungsrichtung gedreht. Der Polar- und Azimutwinkel können nun durch Trigonometrie aus ${}_{l}l$ ermittelt werden (Zeilen 17 und 18). Der Zugriff auf die Lichtverteilung $L_{\sum}$ muss durch normierte Texturkoordinaten erfolgen. Deshalb wird das Winkelpaar $(\theta, \varphi)$ in den Zeilen 19 und 20 bezüglich des Vermessungsbereichs der Gesamtlichverteilung normiert. Über die normierten Koordinaten $({}_{T}u_L, {}_{T}v_L)$ kann schließlich der Texturzugriff auf die Gesamtlichtverteilung $L_{\sum}$ erfolgen. Der spektrale Lichtbeitrag in die betrachtete Richtung wird in $C_l$ (RGBY-Vektor) zwischengespeichert und bildet eine weitere wichtige Größe bei der Auswertung des Beleuchtungsmodells.

Bevor das Beleuchtungsmodell ausgewertet werden kann, muss letztlich noch die distanzabhängige Abschwächung des Lichts Berücksichtigung finden. Nach Gleichung (2-7) nimmt die Beleuchtungsstärke auf einem Objekt mit zunehmender Entfernung zur Lichtquelle quadratisch ab. Die quadratische Entfernung kann durch das Skalarprodukt des Vektors ${}_{w}v_{l,m}$ mit sich selbst auf effiziente Weise berechnet werden. Darüber hinaus wird dieser Wert durch das Quadrat der durch den Anwender einstellbaren Lichtquellenintensität *intens* geteilt. Diese dient zur globalen Skalierbarkeit des Scheinwerferlichts und kann zur Harmonisierung innerhalb der Szene verwendet werden. Um die Simulationsergebnisse nicht zu verzerren, sollte *intens* für alle Scheinwerfer gleich gewählt werden. Durch Multiplikation der Gesamtabschwächung *att* mit dem RGBY-Vektor $C_l$ aus der Lichtverteilung $L_{\sum}$ resultiert schließlich der am Objekt wirkende Lichtbeitrag $C_{l,att}$.

Abschließend kann das Beleuchtungsmodell basierend auf den zuvor bestimmten Größen ausgewertet werden. Hierzu wurde kein eigenes Beleuchtungsmodell entworfen, sondern auf ein physikalisch basiertes Beleuchtungsmodell der Unity-Engine zurückgegriffen [Rus20]. Dieses benötigt die normierten Richtungsvektoren ${}_{w}e$ und ${}_{w}l$, die Objektnormale ${}_{w}n$ und Materialeigenschaften $C_m$, welche an den Koordinaten $({}_{T}u_B, {}_{T}v_B)$ aus dem G-Buffer ausgelesen werden können (Zeilen 26 und 27) sowie den distanzgeschwächten Lichteintrag $C_{l,att}$. Basierend auf diesen Daten bestimmt das Beleuchtungsmodell die resultierende Farbe des betrachteten Oberflächenpunkts bzw. Fragments. Zuvor durch andere Lichtquellen

bestimmte Farben werden additiv berücksichtigt. Mit Abschluss des Lighting-Pass sind alle Lichtquellen durchlaufen. Jedes Fragment wird durch die Summe aller einwirkenden Lichteinflüsse gefärbt. Der finale Frame kann schließlich auf dem Ausgabegerät dargestellt werden.

## 6.4 Validierung

Im Anschluss an die detaillierte Beschreibung der Implementierung gilt es nachzuweisen, dass die entwickelte Lösung einerseits die Gesamtlichtverteilung korrekt bestimmt und andererseits einen Lichteindruck innerhalb der Szene realisiert, welcher die Realität hinreichend gut abbildet. Die Validierung erfolgt in zwei Schritten.

In Unterabschnitt 6.4.1 werden die Resultate der Algorithmik mit einer gemessenen Gesamtlichtverteilung des HD84-Systems verglichen. Das gelingt, indem zuvor eine künstliche Szene konstruiert wird, welche den Großteil der Fremdeinflüsse auf die Lichtverteilung eliminiert und somit eine gute Vergleichbarkeit zu den Messdaten erlaubt.

Um über diesen grundlegenden Nachweis hinaus zu überprüfen, ob der Lichteindruck in einer realistischen Szene den Erwartungen entspricht, wird die vorgestellte Implementierung in Unterabschnitt 6.4.2 mit der Nachtfahrtsimulation „LightDriver“ verglichen (s. Abschnitt 3.3). Da dieses Tool seit Jahren erfolgreich in der Lichtentwicklung der HELLA im Einsatz ist, kann es als Referenz für eine Bewertung von Hyperion herangezogen werden. Zum Zeitpunkt der Validierung konnte die Vielzahl der Lichtquellen eines HD84-Systems nicht im LightDriver simuliert werden. Aus diesem Grund wurden die zu vergleichenden Dimmwerteinstellungen für den LightDriver im Vorhinein in Gesamtlichtverteilungen überführt und dann durch eine virtuelle Lichtquelle simuliert.

### 6.4.1 Bestimmung der Gesamtlichtverteilung

Wie in Abschnitt 3.1.1 beschrieben, kann die photometrische Charakteristik eines Scheinwerfers durch das Auftragen der Lichtstärke über dem Polarwinkel $\theta$ und dem Azimutwinkel $\varphi$ vollständig erfasst werden. Dieselbe Beschreibungsform könnte für die spektrale Erfassung des Lichts herangezogen werden, wenn man anstelle der Lichtstärke die Farbwerte der Primärvalenzen aufträgt. In der beschriebenen Implementierung werden die spektralen Informationen zusammen mit der Lichtstärke in einem RGBY-Vektor zusammengefasst. Insofern werden auf beide Beschreibungsformen die gleichen Operationen angewendet. Zur besseren Darstellbarkeit wird deshalb auf die Visualisierung aller Größen des Vektors verzichtet und stattdessen lediglich die Lichtstärke (Y-Komponente) zur Validierung verwendet.

Zur Validierung der Lichtverteilung wird die in Bild 6-13 dargestellte konstruierte Szene verwendet. Sie enthält nur drei Elemente – eine einfarbige graue Hohlkugel, einen linken Scheinwerfer mit HD84-Modul im Mittelpunkt der Hohlkugel und eine Kamera, welche die Perspektive für das Rendering definiert. Die Kamera wird dabei ebenfalls am Mittelpunkt der Kugel positioniert. Zudem entspricht ihre Blickrichtung der Lichtmittelachse des Scheinwerfers. Die Lage von Kamera und Scheinwerfer sind in Bild 6-13 durch das Koordinatensystem mit den blauen, roten und grünen Pfeilen beschrieben, wobei der blaue

Pfeil entlang der Lichtmittelachse bzw. der Blickrichtung zeigt. Das horizontale Field of View der Kamera wird zu 120° gewählt. In Form des gelben Netzes wird das prozedural generierte Lichtvolumen des Scheinwerfers visualisiert. Die Hohlkugel hat einen Radius von 10m. Dementsprechend sollte der Radius des Lichtvolumens darüber liegen. Der exakte Werte hat im Übrigen keinen Einfluss auf das Ergebnis.

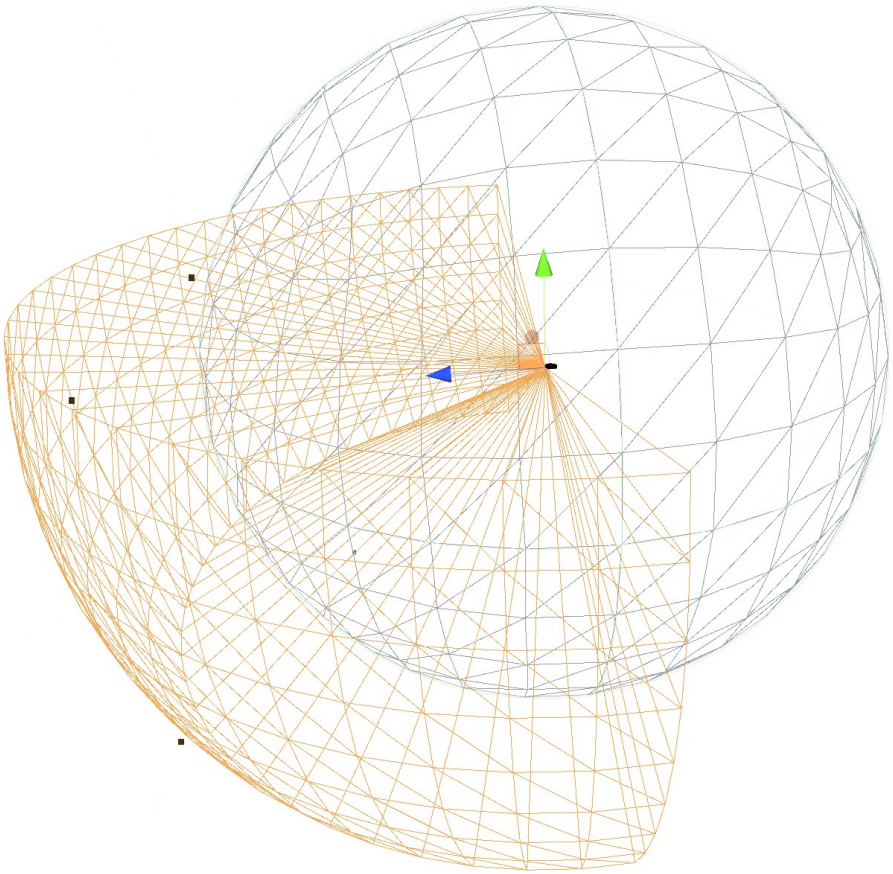

*Bild 6-13*: *Wireframe-Darstellung der Szene zur Validierung der Gesamtlichtverteilung.*

Entsprechend der Szenendefinition bestrahlt der Scheinwerfer die Innenwand der Hohlkugel. Gibt man die Dimmwerte zur Ausgabe einer Abblendlichtverteilung vor (Vgl. Variable *dimValues* in Algorithmus 6), so erhält man das im Bild 6-14a dargestellte Rendering-Ergebnis. Zur genaueren Beurteilung zeigt Bild 6-14b die in logarithmisch skalierten Falschfarben visualisierte Beleuchtungsstärke auf der Kugelinnenwand. In Bild 6-14c wird die gemessene Referenzlichtverteilung, die es simulativ nachzubilden gilt, in logarithmischer Falschfarbenskalierung dargestellt.

Das Rendering-Ergebnis weist die klassischen Eigenschaften einer Abblendlichtverteilung auf. Hierbei ist vor allem die für ein HD84-System typische Stufe in der Hell-Dunkel-Grenze hervorzuheben, welche sicherstellt, dass einerseits eine gute Ausleuchtung der eigenen Fahrbahn und andererseits keine Blendung des Gegenverkehrs gegeben ist. Weiterhin kann beobachtet werden, dass Licht, welches erst in höherer Entfernung auf die Straße fallen würde (große Werte für $\varphi$), auch eine höhere Beleuchtungsstärke auf der Kugelfläche erzielt.

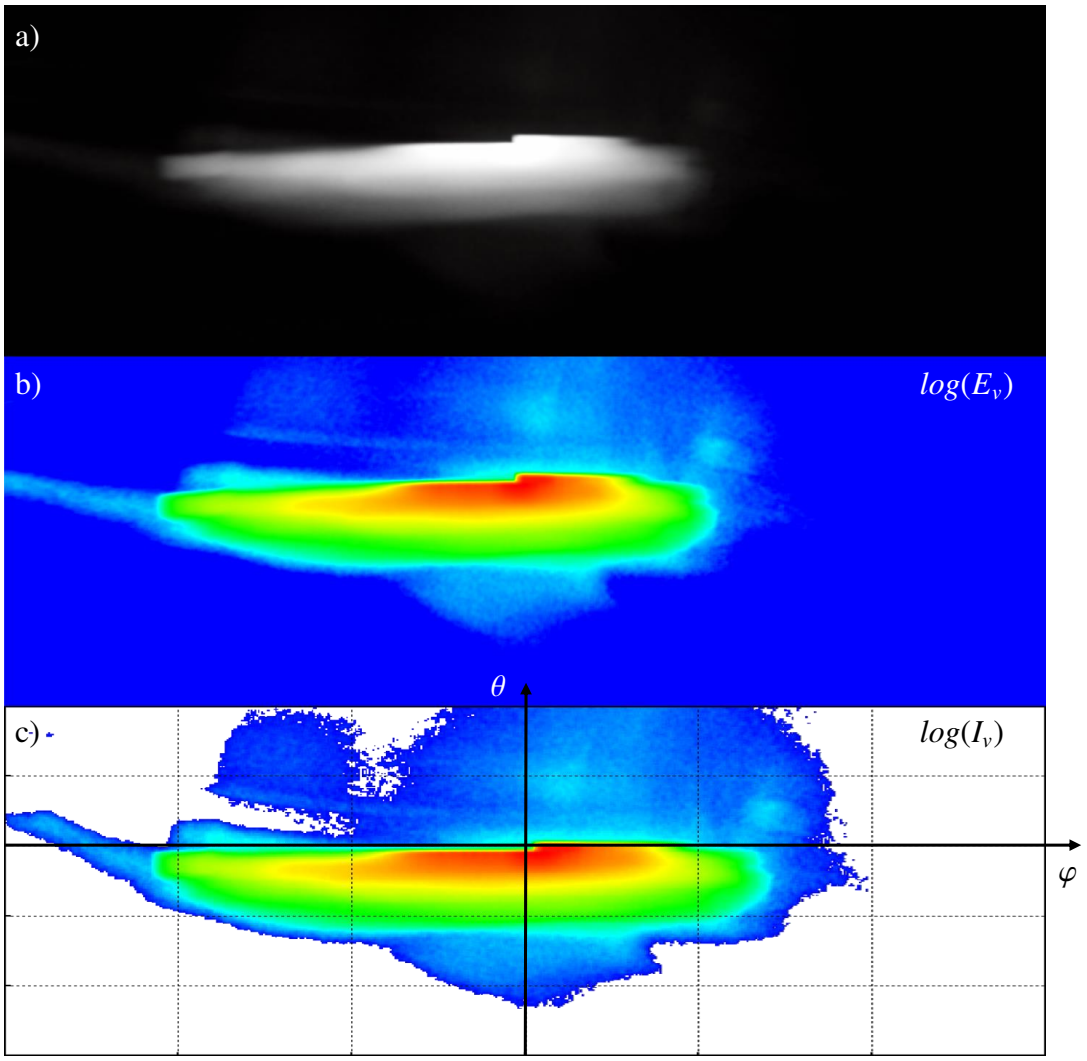


*Bild 6-14*: *Vergleich des gerenderten Lichts a) eines HD84-Moduls im Abblendlicht-Modus (Falschfarben-Darstellung der Beleuchtungsstärke in Bildteil b)) mit den Messdaten c) bei gleicher Konfiguration.*

Der direkte Vergleich des Simulationsergebnisses aus Bild 6-14a mit der Referenz aus Bild 6-14c verdeutlicht, dass die Übereinstimmung von Simulation und Realität auch im Detail gewährleistet ist. Gleichzeitig muss festgestellt werden, dass ein Rendering in der normalen Ansicht, wie es in Bild 6-14a gezeigt wird, nicht ausreicht, um Helligkeitsabstufungen innerhalb der Lichtverteilung und Streulichtanteile detailliert zu analysieren. Diese kommen in der Falschfarbenvisualisierung deutlich genauer zum Vorschein.

Um diesem Problem zu begegnen, wird die logarithmisch skalierte Falschfarbendarstellung der Beleuchtungsstärke mit einem alternativen Shader bei sonst gleicher Konfiguration gerendert. Die Beleuchtungsstärke $E_v$ und die Lichtstärke $I_v$ stehen nach Gleichung (2-13) im Zusammenhang. Durch die spezifische Definition der Szene ist sowohl der Abstand zwischen Lichtquelle und bestrahlter Fläche als auch der Lichteinfallswinkel an jedem Punkt konstant. Es ergibt sich eine direkte Proportionalität zwischen der Lichtstärke $I_v$ und der Beleuchtungsstärke $E_v$. Somit ist eine direkte Vergleichbarkeit zwischen der Analyse-

sicht des Renderings (Bild 6-14b), welche die Beleuchtungsstärke $E_v$ visualisiert, und der gemessenen Referenzlichtverteilung, welche bezüglich der Lichtstärke $I_v$ aufgetragen ist, gegeben.

Wie sich herausstellt, ist die Korrespondenz zwischen der simulierten Lichtverteilung und der Referenz in der Falschfarbenrepräsentation noch überzeugender. Sowohl die Abstufungen innerhalb der zentralen Lichtkeule als auch die Ausprägungen des Streulichts im Randbereich sind in beiden Darstellungen nahezu identisch. Es sei darauf hingewiesen, dass die Achsenskalierungen der Polar- und Azimutwinkel $\theta$ und $\varphi$ von Bild 6-14c nicht auf die Bilder 6-14a und b übertragen werden können, da diese die Szene aus der perspektivischen Sicht der Kamera zeigen. Die Position und Ausdehnung der Lichtverteilung wurde deshalb in einem separaten Versuch überprüft und bestätigt. Zusammengefasst kann die Lichtverteilung als valide eingestuft werden.

### 6.4.2 Beleuchtung der Szene

Wenngleich die Korrektheit der Implementierung im Sinne der mathematisch korrekten Wiedergabe der Lichtverteilung durch den Abschnitt 6.4.1 nachgewiesen wird, muss auch die optische Erscheinung des Lichts in realitätsnahen Szenen beurteilt werden. Schließlich nimmt die subjektive Bewertung des Lichts speziell in der Scheinwerferentwicklung einen hohen Anteil der Gesamtevaluierung ein. Bei einer derartigen Betrachtung nehmen deutlich mehr Faktoren Einfluss. Die Einschränkungen des Ausgabegeräts hinsichtlich Leuchtdichte und Kontrast, die Modellierungstiefe der Szene hinsichtlich Geometrien, Diskretisierungen, Texturen, Normal Maps, Reflexionseigenschaften und die Komplexität des verwendeten Beleuchtungsmodells sind nur einige Beispiele.

Da die Erstellung realer Versuche gepaart mit der hoch genauen Nachbildung der realen Umgebungen einen nicht darstellbaren Aufwand bedeuten würde, wird der HELLA LightDriver (s. Abschnitt 3.3) als Referenz verwendet. Die Eigenentwicklung der Hella wird seit einigen Jahren erfolgreich im Entwicklungsprozess neuer Scheinwerfer und Lichtfunktionen eingesetzt. Im Gegensatz zur hier vorgestellten Implementierung, erlaubt der LightDriver (Stand: Juli 2017) jedoch nicht die dynamische Anpassung der Lichtverteilung basierend auf den Dimmwerten der Pixellichtquellen, wie es für ein HD-System erforderlich ist. Dennoch schmälert diese Eigenschaft nicht die Eignung des LightDriver als Referenz zur Validierung. Die gewünschte Gesamtlichtverteilung des HD-Scheinwerfers wird im Vorhinein berechnet und als statische Lichtverteilung im LightDriver geladen.

Bild 6-15 vergleicht die Abblendlichtverteilungen des HD84-Systems (linker und rechter Scheinwerfer) der Hyperion-Implementierung (unten) mit der Abblendlichtverteilung des LightDriver (oben). Die zur Validierung herangezogenen Bilder können in Anhang A4 vergrößert und um weitere Informationen angereichert in Augenschein genommen werden. Um die Reproduktion der Szene des LightDriver in Hyperion mit vertretbarem Aufwand zu ermöglichen, wurde eine einfache zweispurige Straße gewählt. Rechts im Bild 6-15 wird die Szene um eine Messwand mit roten Kontrolllinien ergänzt. Diese befindet sich in Fahrtrichtung 10m vom Scheinwerfersystem entfernt. Eine solche Messwand ist ein typisches Analysewerkzeug bei der Bewertung von Lichtverteilungen, da die vertikale Projektion des Lichts eine wesentlich bessere Erkennbarkeit der Konturen ermöglicht. Die vertikalen Kontrolllinien fluchten mit den Einbaupositionen der Scheinwerfer be-

züglich der Fahrzeugquerachse. Die drei horizontalen Kontrolllinien befinden sich aus Sicht der Kugelkoordinatensysteme der Scheinwerfer bei den Polarwinkeln $0°$, $-0.57°$ und $-1°$. Die genaue Szenendefinition konnte nicht direkt aus dem LightDriver extrahiert werden. Sie musste deshalb manuell nachgebildet werden. Aus diesem Grund können die Szenen (Texturen, Farben, ...) im Detail voneinander abweichen. Weiterhin liegt dem LightDriver ein anderes Beleuchtungsmodell zu Grunde. Für die Validierung der Hyperion-Implementierung sollte die Übereinstimmung jedoch genügen.

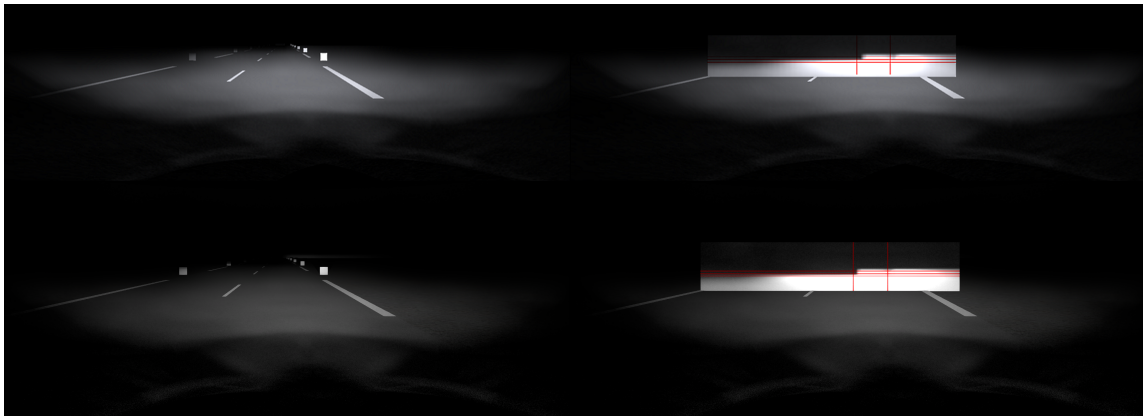

*Bild 6-15*: *Vergleich des LightDriver (oben) mit Hyperion (unten) bei der Simulation einer Abblendlichtverteilung in einer einfachen Straßenszene (links) und mit einer 10m entfernten Messwand (rechts).*

Vergleicht man die Abblendlichtverteilungen links im Bild 6-15, ist insgesamt eine gute Übereinstimmung festzustellen. Bei genauerem Hinsehen sind jedoch leichte Unterschiede erkennbar. Die Farbe bzw. die Helligkeit auf der Straße erscheint in den beiden Darstellung leicht unterschiedlich. Ursächlich für die erwähnten Unterschiede können verschiedenste Einflussfaktoren sein, wie sie bereits eingangs erwähnt werden.

Abgesehen von den leichten Unterschieden stimmen beide Simulationen gut überein, so dass die in Hyperion eingesetzte Rendering-Methode auch in realistischen Szenen validiert werden kann. Mit Hilfe der Fahrbahnmarkierungen wird die qualitative Formgleichheit der beleuchteten Straßenbereiche beider Implementierungen deutlich. Darüber hinaus sind in beiden Darstellungen die unterschiedlichen Lichtreichweiten auf der linken und rechten Fahrbahn zu erkennen. Die weißen Kontrollfelder an den Fahrbahnrändern zeigen diese für Abblendlicht typische Asymmetrie besonders deutlich und für beide Implementierung auf gleiche Weise.

Auch wenn das Licht auf der Straße das zentrale Bewertungskriterium aus Sicht des Fahrers ist, kann die Analyse von Lichtverteilungen an einer vertikalen Messwand vor allem zu Vergleichszwecken hilfreich sein. Die Hell-Dunkel-Grenze wird durch die vertikale Projektion scharf dargestellt, wodurch die unterschiedlichen Ausleuchtungen der Fahrspuren als Stufen in der Lichtverteilung sichtbar werden. Unter Zuhilfenahme der roten Kontrolllinien im rechten Teil des Bilds 6-15 ist die Ausprägung der HDG in beiden Simulationen identisch.

Nachdem die Abblendlichtverteilungen in beiden Simulationen verglichen wurden, zeigt Bild 6-16 einen ähnlichen Vergleich für das Fernlicht. Für den LightDriver wurde die

Lichtstärkeverteilung erneut vorberechnet und als statische Gesamtlichtverteilung geladen. Wie beim Abblendlicht lässt sich neben Detailunterschieden zwischen den Bildern eine gute Gesamtübereinstimmung feststellen. Wie zu erwarten war, hat die Ausleuchtung der entfernten Bereiche im Vergleich zum Abblendlicht in Bild 6-15 zugenommen. Besonders stark kann dieser Unterschied auf der Gegenfahrbahn und den daran entlang positionierten Kontrollfeldern wahrgenommen werden. Dieser Effekt wird durch eine grundlegende Änderung der Lichtverteilung erreicht, wie an den Messwänden auf der rechten Seite von Bild 6-16 zu erkennen ist. Die Stufe der HDG des Abblendlichts verschwindet bei Verwendung des Fernlichts. Stattdessen wird eine symmetrische Lichtverteilung mit einem so genannten Fernlichtkegel erzeugt, der auch weit entfernte Bereiche vor dem Fahrzeug ausleuchtet.

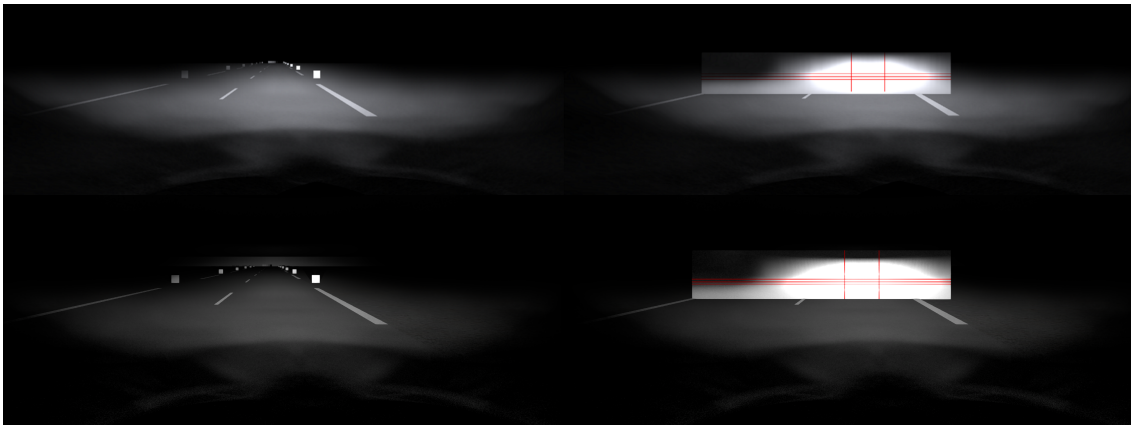

*Bild 6-16*: *Vergleich des LightDriver (oben) mit Hyperion (unten) bei der Simulation einer Fernlichtverteilung in einer einfachen Straßenszene (links) und mit einer 10m entfernten Messwand (rechts).*

## 6.5 Laufzeit

Im Hinblick auf die Anwendung des Verfahrens innerhalb einer Nachtfahrtsimulation ist **A6** (Echtzeitfähigkeit) eine kritische Anforderung. Um nachzuweisen, dass Hyperion diese Anforderung erfüllt, wird die Laufzeit des Renderings analysiert. Dabei können die Stufen der zweistufigen Lichtsimulation isoliert voneinander betrachtet werden. Im nachfolgenden Abschnitt wird die Laufzeit zur Bestimmung der Gesamtlichtverteilung diskutiert, während Abschnitt 6.5.2 die Laufzeit des Shadings innerhalb der Szene betrachtet.

### 6.5.1 Bestimmung der Gesamtlichtverteilung

In Abschnitt 6.2.4 wurden die Komplexität und der Speicherbedarfs des hier eingesetzten Verfahrens bereits auf theoretischer Ebene diskutiert. Nachfolgend sollen die theoretischen Überlegungen durch reale Laufzeitmessungen untermauert werden.

Ausschließlich Vorgänge, die während der Simulation wiederholt ausgeführt werden müssen, stellen kritische Elemente bei der Laufzeitbetrachtung dar. Eine Analyse des Imports

und der Datenoptimierung von Lichtverteilungen sowie der Initialisierung ist deshalb nicht erforderlich. Lediglich die in Abschnitt 6.2.3 vorgestellten Laufzeitoperationen sind für die nachfolgende Betrachtung relevant.

Untersucht man den auf der CPU ausgeführten Algorithmus 6, so beschränkt sich die Aufgabe der CPU auf die Übergabe der momentan vorliegenden Dimmwerte an die GPU und dem Aufruf des GPU-seitig ausgeführten GroupCombiner-Shaders für jede Gruppe. Die durch die CPU beanspruchte Berechnungszeit kann deshalb als vernachlässigbar angesehen werden.

Deutlich kritischer ist hingegen der GPU-seitig ausgeführte Algorithmus 7. Betrachtet man den Pseudocode, ergeben sich im Kontext der Anwendung vier relevante Parameter, welche potentiell Einfluss auf die Laufzeit nehmen könnten. Diese sind

- die Anzahl der Lichtquellen innerhalb eines Scheinwerfers,
- die Größe des auszuleuchtenden Winkelbereichs,
- die Auflösung innerhalb dieses Winkelbereichs
- und die Überlappung der Lichtverteilungen einzelner Lichtquellen.

Um die Einflüsse der genannten Parameter auf die Laufzeit quantifizieren zu können, müssen zunächst Maße für diese Größen definiert werden. Für den ersten Parameter ist die Festlegung eines Maßes trivial, da die Zahl der Lichtquellen unmittelbar verwendet werden kann. Die Größe des auszuleuchtenden Winkelbereichs muss hingegen in einen eindimensionalen Wert überführt werden, der die Ausdehung bezüglich des Polar- und Azimutwinkels $[\underline{\theta}, \bar{\theta}] \times [\underline{\varphi}, \bar{\varphi}]$ quantifiziert. Letztlich wirkt sich der Winkelbereich in der Anzahl der Datenpunkte innerhalb der Gesamtlichtverteilung aus. Er wird deshalb durch das nachfolgend als „Winkelfläche“ bezeichnete Produkt $(\bar{\theta} - \underline{\theta}) \cdot (\bar{\varphi} - \underline{\varphi})$ beschrieben. Die Auflösung der Lichtverteilung wird in der Einheit „Winkelgrad pro Pixel“ $[^\circ/px]$ gemessen. Da die Veränderung der Auflösung technisch den gleichen Effekt wie die Anpassung des Winkelbereichs hat, wird diese nicht in einer gesonderten Parameterstudie betrachtet. Die Erkenntnisse über den Zusammenhang zwischen Laufzeit und Winkelbereich können direkt auf den Einfluss der Auflösung überführt werden. Als Maß für die Überlappung wird die Ausdehnung der Lichtverteilung einer Einzellichtquelle bezogen auf den exklusiv durch sie zu bestrahlenden Bereich angegeben. Der exklusive Einflussbereich ergibt sich dabei durch das Rastern des Gesamtwinkelbereichs entsprechend der insgesamt vorliegenden Lichtquellenzahl (z.B. 20 Zeilen und 50 Spalten für ein 1.000 Pixel-System). Beispielhaft wird er durch das durchgezogene Rechteck in Bild 6-17 visualisiert. Die konkrete Bedeutung der relativen Überlappung $o_{rel}$ wird im Bild ebenfalls visualisiert. Sie bewirkt, dass die Lichtquelle über ihren exklusiven Bereich (durchgezogenes Rechteck in Bild 6-17) hinaus strahlt und so insgesamt das gestrichelt dargestellte Rechteck bezüglich der Raumwinkeldomäne ausleuchtet. Die bestrahlte Winkelfläche $(\bar{\theta}_k - \underline{\theta}_k) \cdot (\bar{\varphi}_k - \underline{\varphi}_k)$ der Lichtquelle $k$ vergrößert sich auf diese Weise um den Faktor $(2o_{rel} + 1)^2$. Im Randbereich der Gesamtlichtverteilung ist die Ausdehnung der Lichtquelle nicht zu allen Seiten möglich. In diesem Fall wird sichergestellt, dass die gesamte Flächenzunahme unverändert bleibt. Der relative Wert $o_{rel}$ der Überlappung kann alternativ auf die mittlere Anzahl von Lichtquellen, die auf den gleichen Pixel der Gesamtlichtverteilung strahlen, überführt werden. Nachfolgend werden stets beide Kenngrößen genannt.

Da mit dem HD84-System im Rahmen dieser Arbeit nur ein realer Datensatz zur Verfügung steht, ist die Generierung weiterer Scheinwerfer-Datensätze erforderlich, um die

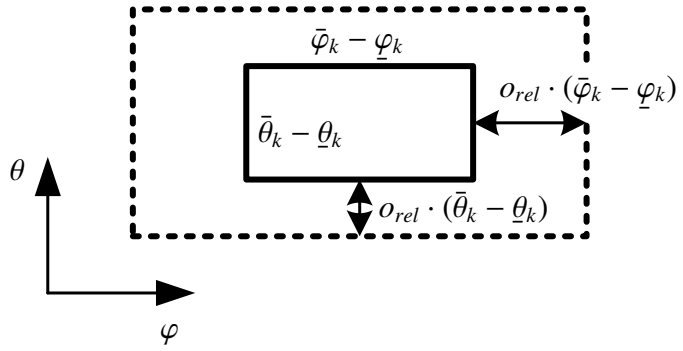


*Bild 6-17*: *Maß für die relative Überlappung $o_{rel}$ benachbarter Einzellichtquellen.*

Einflüsse der zuvor genannten Größen auf die Laufzeit untersuchen zu können. Hinzu kommt, dass die isolierte Betrachtung von Einflüssen nur möglich ist, wenn ein Parameter variiert wird und alle anderen konstant gehalten werden. Weitere reale Datensätze wären somit nur bedingt hilfreich. Aus diesem Grund werden für jede Parameterstudie fiktive Scheinwerferdaten generiert, welche die laufzeitrelevanten Eigenschaften realer Lichtverteilungen nachbilden. Der Benutzerdialog für diese Generierung, die wählbaren Parameter und ihre Bedeutung für das resultierende Scheinwerfersystem können in Anhang A3.1 eingesehen werden. Die fiktiven Datensätze sind stets Abwandlungen der in Tabelle 6-1 zusammengefassten Standardkonfiguration.

*Tabelle 6-1*: *Standardkonfiguration eines Scheinwerfersystems zur Parameterstudie im Rahmen der Laufzeitanalyse der Lichtverteilungsberechnung.*

| Eigenschaft | Standardwert | Einheit |
|---|---|---|
| Lichtquellenzahl | 1.000 | - |
| Winkelbereich | $[-60, 60] \times [-30, 30]$ | $^\circ$ |
| Winkelfläche | 7.200 | $^\circ\circ$ |
| Auflösung | 0.05 | $\frac{^\circ}{px}$ |
| Datenpunkte | 2.880.000 | - |
| Überlappung | 67 | % |
| Lichtquellen/Pixel | $\approx 2.8$ | $\frac{1}{px}$ |
| Spektral | Ja | - |
| Bit/Farbkanal | 32 | Bit |
| Dimmwertbuffer | 8 | kB |
| Threads/Gruppe | 64 | - |
| Threadgruppen | 45.000 | - |

Sämtliche Messungen werden auf einem Notebook durchgeführt, dessen Hard- und Softwarespezifikation in Tabelle 6-2 angegeben ist. Zur Messung wird der Profiler der Unity-Engine eingesetzt, welcher in der angegebenen Version als eingeständiger Prozess aufgerufen werden kann und das Laufzeitverhalten der eigentlichen Anwendung nicht beeinflusst.

*Tabelle 6-2: Hard- und Softwarespezifikation im Rahmen der Laufzeitanalyse.*

| | |
|---|---|
| Betriebssystem | Windows 10 Pro 64-bit (10.0, Build 16299) |
| Grafikengine | Unity3D, Version 2020.1.6f1 |
| Modell | Dell Precision 7710 |
| Prozessor | Intel(R) Core(TM) i7-6820HQ CPU @2.7 GHz |
| Arbeitsspeicher | 16384 MB RAM |
| DirectX Version | DirectX 11 |
| Grafikchip | NVIDIA Quadro M3000M |
| Videospeicher | 4062 MB VRAM (+ 8133 MB Shared) |

Zuerst wird der Einfluss der Lichtquellenzahl auf die Laufzeit untersucht. Dafür werden sechs HD-Scheinwerfer generiert, die über 10, 100, 1.000, 10.000, 100.000 und 1.000.000 Lichtquellen verfügen. Alle weiteren Eigenschaften entsprechen der in Tabelle 6-1 gezeigten Standardkonfiguration. Die sich ergebenden Buffergrößen und weitere Informationen zu den einzelnen Systemen fasst die Tabelle A3-1 im Anhang A3.2 zusammen.

Jedes Scheinwerfersystem wird simuliert. Dabei werden die Dimmwerte sämtlicher Lichtquellen der Scheinwerfer in jedem Frame zufällig gewählt. Somit ist eine ständige Neuberechnung der Gesamtlichtverteilung erforderlich. Der in blau dargestellte Plot in Bild 6-18 zeigt die Berechnungszeiten $t_{comb}$ der Lichtverteilungen pro Scheinwerfer auf der GPU.

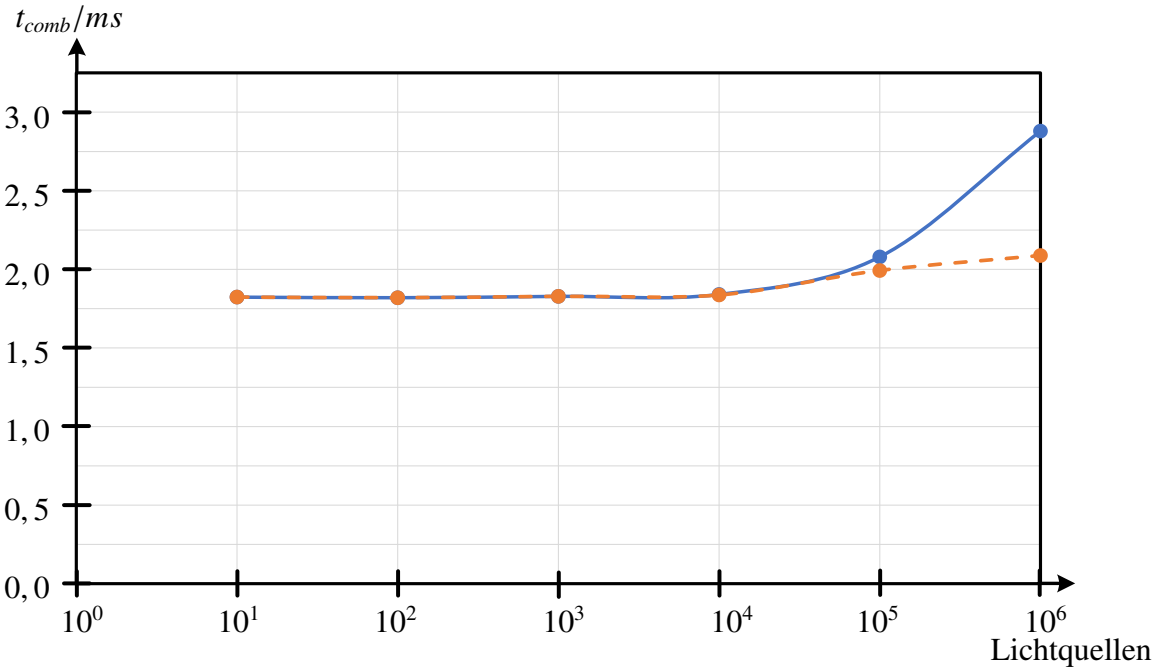


*Bild 6-18: Laufzeit der Lichtverteilungsberechnung in Abhängigkeit von der Lichtquellenzahl mit (blau) und ohne (orange) stetiger Dimmwertaktualisierung.*

Wie sich zeigt, nimmt die Anzahl der Lichtquellen bis zu einer Größenordnung von $10^4$ keinerlei Einfluss auf das Laufzeitverhalten. Das ist nach Abschnitt 6.2.4 auch zu erwarten, da die Lichtquellenzahl den ausgabesensitiven Algorithmus in seiner Komplexität nicht

beeinflusst. Tatsächlich kann man jedoch ab einer Größenordnung von $10^5$ Lichtquellen einen rapiden Anstieg beobachten. Trotz dieser Zunahme der Berechnungszeit ist selbst das 1.000.000 Lichtquellensystem mit unter 3 ms echtzeitfähig, wodurch die Eignung des Verfahrens für sämtliche praxisrelevanten Scheinwerfer-Technologien nachgewiesen ist.

Weitere Untersuchungen haben gezeigt, dass die primäre Ursache für den Laufzeitanstieg ab $10^5$ Lichtquellen die Datenübertragung zwischen CPU und GPU ist. Die Größe des Dimmwert-Buffers ist direkt proportional mit der Anzahl von Lichtquellen und muss in jedem Frame von der CPU an die GPU übergeben werden. Das in Tabelle 6-2 spezifizierte Testsystem weist hierfür ab etwa $10^5$ Lichtquellen eine zu geringe Datenübertragung auf. Ein Nachweis dieser Ursache gelingt durch die Neuberechnung der Lichtverteilung bei unverändertem Dimmwert-Buffer. Die sich ergebenden Laufzeiten zeigt der orange dargestellte Plot in Bild 6-18. Wie sich zeigt, ist dieser deutlich flacher. Dennoch verbleibt ein geringer, linearer Anstieg, welcher durch die zunehmende Häufigkeit von Cache-Misses innerhalb der *srcRGBY*- und *srcID*-Buffer erklärt werden kann.

Als Nächstes soll der Einfluss des Messbereichs und der Auflösung der Gesamtlichtverteilung untersucht werden. Beide Größen bestimmen letztlich die Anzahl der Datenpunkte der Gesamtlichtverteilung, welche sich aus dem Quotienten der Winkelfläche und der quadratischen Auflösung ergibt. Es genügt somit, den Einfluss der Datenpunktzahl auf die Laufzeit zu untersuchen. Auch hierfür wurden verschiedene Systeme generiert, die weitestgehend der Standardkonfiguration aus Tabelle 6-2 entsprechen. Nur die Anzahl der Datenpunkte wurde durch die Variation des Messbereichs von 160.000 bis hin zu 12.960.000 vergrößert. Weiterführende Informationen zu den resultierenden Buffergrößen finden sich in Tabelle A3-2 im Anhang A3.2. Der Einfluss auf die Laufzeit wird durch den Plot 6-19 visualisiert. Auch wenn die Auflösung der Lichtverteilung nicht im direkten Zusammenhang mit der Ausgabeauflösung steht, werden im Plot zusätzlich die Anzahlen der Pixel verschiedener standardisierter Ausgabeauflösungen markiert.

Der Zusammenhang zwischen der Anzahl von Datenpunkten und der Berechnungsdauer ist erwartungsgemäß linear. Jeder Datenpunkt wird durch einen einzelnen Thread des in Abschnitt 6.2.3 vorgestellten Compute-Shaders exklusiv bearbeitet. Da die GPU bei weitem nicht genug Ausführungseinheiten vorweist, um alle Datenpunkte parallel zu verarbeiten, kommt es zu einer sequentiellen Verarbeitung von Datenpunktgruppen, wobei die Datenpunkte innerhalb einer Gruppe parallel berechnet werden. Die Messung bestätigt auch die Komplexitätsbetrachtung in Abschnitt 6.2.4. Auch für sehr hohe Datenpunktzahlen im Bereich von Ultra HD ($3.840 \times 2.160$ Datenpunkte) terminiert der vorgestellte Algorithmus in unter $10\,ms$. In Anbetracht der verwendeten Rechnerspezifikation nach Tabelle 6-2 ist die Echtzeitfähigkeit auch für hochaufgelöste Systeme sichergestellt.

Zuletzt ist der Einfluss der Überlappung benachbarter Lichtquellen auf die Laufzeit zu untersuchen. Auch hierfür wurden verschiedene Systeme generiert, welche mit Ausnahme der Überlappung die in Tabelle 6-1 aufgeführte Spezifikation vorweisen. Die Überlappungen der einzelnen Systeme und die damit einhergehenden mittleren Anzahlen von Lichquellen pro Datenpunkt werden in Tabelle 6-3 gegenüber gestellt. Die Buffergrößen der einzelnen Systeme und weiterführende Informationen können in Tabelle A3-3 im Anhang A3.2 eingesehen werden. Der blaue Plot in Bild 6-20 zeigt die Berechnungsdauern in Abhängigkeit von der Überlappung benachbarter Lichtquellen, während der orange Plot selbige in Abhängigkeit von der mittleren Anzahl an Lichquellen pro Pixel darstellt. Wenngleich die relative Überlappung benachbarter Pixel hinsichtlich der Bedeutung greifbarer

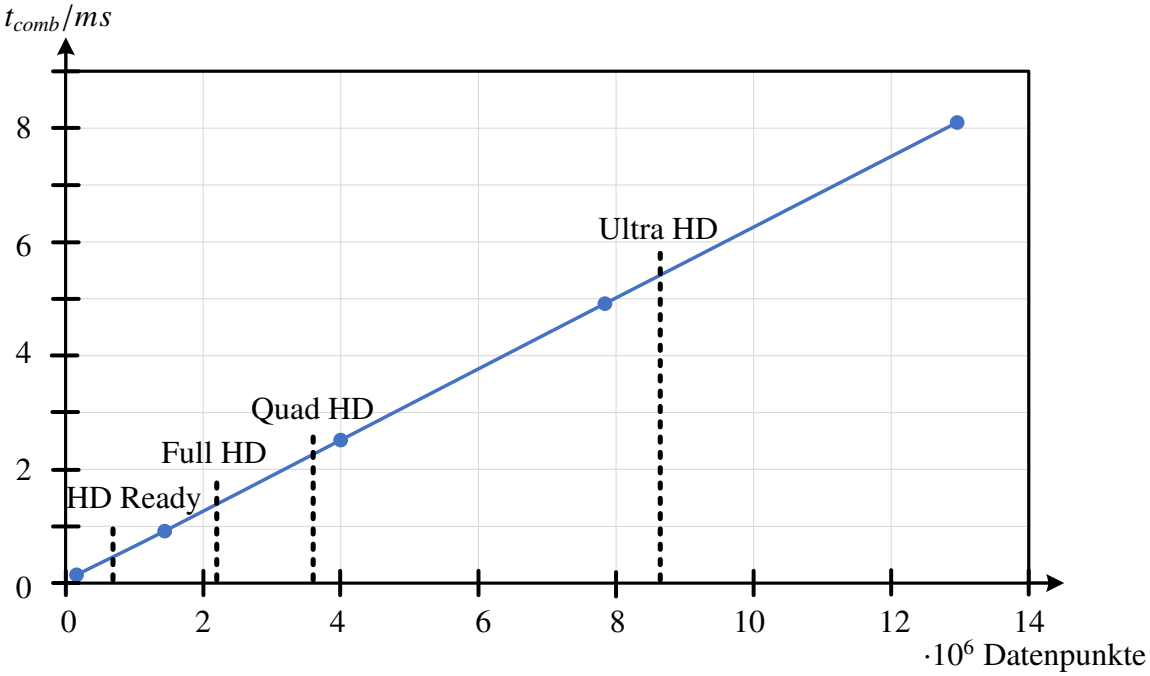


*Bild 6-19*: *Laufzeit der Lichtverteilungsberechnung in Abhängigkeit von der Anzahl der Datenpunkte in der Gesamtlichtverteilung.*

*Tabelle 6-3*: *Relative Überlappungen und die mittlere Anzahl von Lichtquellen pro Datenpunkt der zur Laufzeitanalyse generierten Scheinwerfersysteme.*

| Überlappung(%) | 0 | 40 | 80 | 120 | 160 | 200 |
|---|---|---|---|---|---|---|
| Lichtquellen/Pixel | 1,0 | 2,0 | 3,2 | 4,9 | 6,8 | 9,0 |

ist, stellt die mittlere Anzahl von Lichtquellen, die auf einen Datenpunkt der Gesamtlichtverteilung strahlen, den kritischen Wert bezüglich der Laufzeit dar. Die Diskussion der Laufzeitmessung soll deshalb anhand des orangen Plots in Bild 6-20 erfolgen.

Erwartungsgemäß steigt die Berechnungsdauer mit der Anzahl von Lichtquellen, die auf einen Datenpunkt Einfluss nehmen. Schließlich stellt diese Kenngröße die mittlere Anzahl der Iterationen dar, welche innerhalb des Compute-Shaders (Algorithmus 7) für jeden Datenpunkt durchlaufen werden müssen. Mit dieser Begründung wäre jedoch ein linearer Zusammenhang zu erwarten. Der Plot zeigt hingegen einen leicht progressiven Anstieg. Diese Progression ist auf die zunehmende Größe der *srcRGBY*- und *srcID*-Buffer zurückzuführen, welche linear mit der mittleren Lichtquellenzahl pro Pixel wachsen. Auch wenn die Buffer zeitlich unveränderlich sind und zur Laufzeit nicht aktualisiert werden müssen, führt ihre zunehmende Größe zu häufigeren Cache-Misses, welche ein Nachladen von Daten erfordern und somit zeitliche Verzögerungen hervorrufen.

Zusammenfassend kann festgehalten werden, dass die Berechnung der Gesamtlichtverteilung auf dem in Tabelle 6-2 spezifizierten Testsystem, welches gemessen an heutigen Maßstäben nur eine mittelmäßige Performance aufweist, für praxisrelevante Konfigurationen echtzeitfähig ist. Somit wird die Anforderung **A6** (Echtzeitfähigkeit) erfüllt. Die in Abschnitt 6.2.4 bereits diskutierte Ausgabesensitivität der Implementierung konnte durch

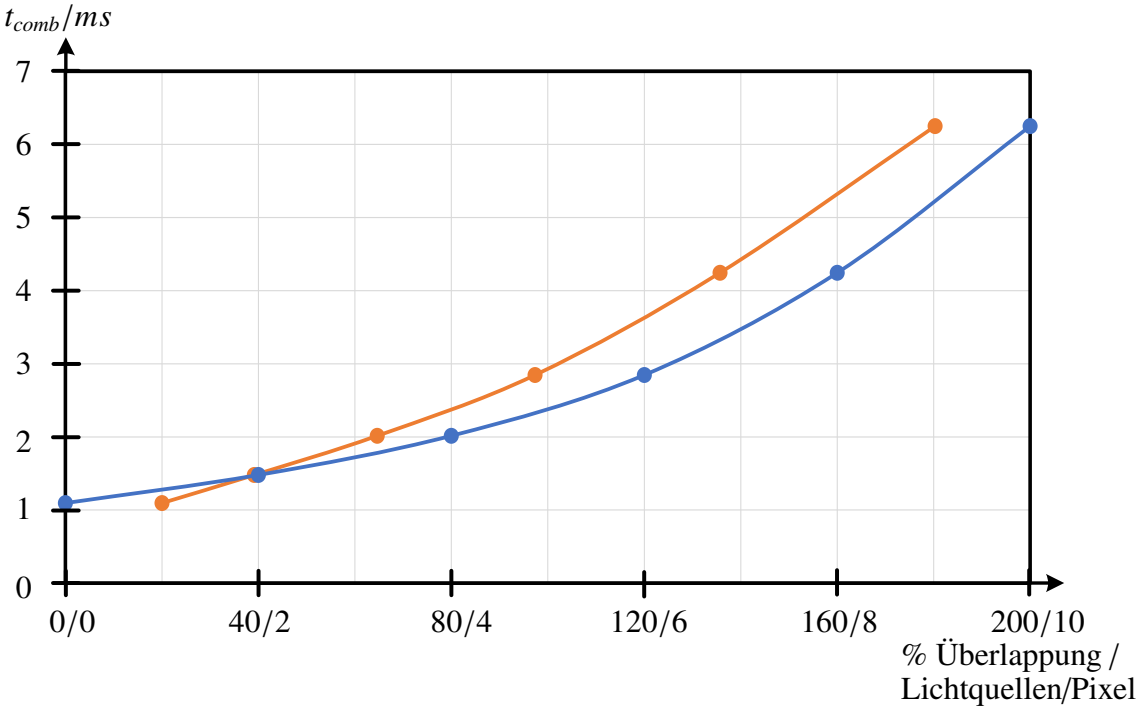


*Bild 6-20*: *Laufzeit der Lichtverteilungsberechnung in Abhängigkeit von der Überlappung der Einzellichtverteilungen (blau: Laufzeit über relativer Überlappung, orange: Laufzeit über mittlerer Anzahl von Lichtquellen pro Datenpunkt).*

die Laufzeitmessungen unter Vernachlässigung der Speicherverwaltung nachgewiesen werden. Die Laufzeit bleibt somit durch die Lichtquellenzahl weitgehend unbeeinflusst, womit die Erfüllung der Anforderung **A4** (Pixel-Skalierbarkeit) sichergestellt ist.

### 6.5.2 Beleuchtung der Szene

Basierend auf der ermittelten Gesamtlichtverteilung eines Scheinwerfers schließt sich die Beleuchtung der virtuellen Szene an, welche durch die in Abschnitt 6.3 vorgestellte Lichtquelle erfolgt. Da es sich bei dieser um eine Modifikation des Spotlichts der Unity-Engine handelt, gelten die gleichen Zusammenhänge, wie für die Built-In-Lights der Unity-Engine innerhalb der Deferred Pipeline. Nennenswerte Unterschiede zum Spotlicht stellen das in Bild 6-11 dargestellte Lichtvolumen und die Interpretation der Lichtverteilung in Kugelkoordinaten dar. Diese sollten jedoch keine erheblichen Auswirkungen auf die Berechnungsdauer haben.

Betrachtet man die Implementierung der Lichtquelle und deren Einbindung in die Deferred Pipeline, so können vier maßgebliche Einflussfaktoren auf die Berechnungsdauer identifiziert werden. Diese sind

- die Auflösung der Ausgabe,
- die Kameraperspektive,
- der Messbereich des Scheinwerfers und
- die Messauflösung des Scheinwerfers.

Den offensichtlichsten Einfluss hat der zuerst genannte Punkt. Die Ausgabeauflösung bestimmt die Anzahl der Fragmente, auf die der in Algorithmus 13 beschriebene Shader maximal angewendet werden muss. Abhängig davon, ob das Lichtvolumen der Lichtquelle die Ausgabe nur anteilig überdeckt, muss der Shader auch nur auf einen Anteil der insgesamt im G-Buffer enthaltenen Fragmente angewendet werden. Da diese Überdeckung stark von der Kameraperspektive und dem Messbereich abhängt, werden Lage und Orientierung der Kamera für die Laufzeitmessung identisch zum simulierten Scheinwerfer gewählt. So genügt die Variation des Messbereichs, welcher die Ausdehnung des Lichtvolumens und damit dessen Überdeckung mit der Ausgabe spezifiziert. Der Messbereich wird deshalb als zweiter Parameter bezüglich seiner Einflussnahme auf die Laufzeit untersucht. In Kombination mit dem Messbereich legt die Messauflösung des Scheinwerfersystems die Anzahl der vermessenen Datenpunkte der Gesamtlichtverteilung fest und bestimmt damit die Größe der Gesamtlichtverteilung. Bedingt durch das Laden der Daten vor der Anwendung des Headlamp Shaders und den mit wachsender Größe zunehmenden Cache-Misses hat auch die Messauflösung Einfluss auf die Laufzeit. Sie stellt den dritten Parameter dar, den es zu untersuchen gilt. Insbesondere sei nochmal darauf hingewiesen, dass die Komplexität der Szene den Berechnungsaufwand des Shaders aufgrund seiner Einbindung innerhalb der Deferred Pipeline nicht beeinflusst. Die nachfolgend vorgestellten Laufzeitmessungen haben somit eine generelle Aussagekraft.

Die Tabelle 6-4 fasst die in den nachfolgenden Messungen verwendete Konfiguration zusammen. Mit Ausnahme der zu variierenden Parameter gelten die nachfolgend aufgeführten Eigenschaften für Kamera und Scheinwerfersystem. Innerhalb der Standardkonfiguration überdeckt das Scheinwerferlicht 91% des durch die Kamera einsehbaren Bereichs. Diese Überdeckung ergibt sich aus der Wahl des FOV der Kamera und des Winkelbereichs des Scheinwerfers. Es sei jedoch darauf hingewiesen, dass die Berechnung der Überdeckung für eine gegebene Winkelkonfiguration nicht trivial ist und unter Berücksichtigung der verschiedenen Geometrien von View Frustum und Lichtvolumen erfolgen muss. Für die verwendete Hard- und Software gilt die in Tabelle 6-2 definierte Spezifikation weiterhin.

Bei der Laufzeitanalyse kann die Berechnungsdauer auf der CPU, welche lediglich die Transformationsmatrix des Lichtvolumens und einige Parameter vorgibt, außer Acht gelassen werden. Sie ist im Vergleich zum Berechnungsaufwand der GPU vernachlässigbar.

Zuerst wird der Einfluss des Messbereichs auf die Laufzeit des Shadings diskutiert. Bei konstanter Messauflösung führt ein größer werdender Messbereich einerseits zu einer höher werdenden Anzahl von Datenpunkten innerhalb der Gesamtlichtverteilung und andererseits zu einer zunehmenden Überdeckung des einsehbaren Bereichs. Insofern ist in jedem Fall mit einer Laufzeitzunahme zu rechnen. Der blaue Plot in Bild 6-21 visualisiert die gemessenen Berechnungszeiten in Abhängigkeit der Anzahl von Datenpunkten. Diese ergibt sich durch Division der Winkelfläche durch die quadratische Messauflösung. Im Anhang A3.3 finden sich weitere Details zu dieser und den nachfolgenden Messreihen.

Bis zu einer Zahl von etwa zwei Millionen Datenpunkten weist die Berechnungsdauer $t_{light}$ einen proportionalen Zusammenhang zu den Datenpunkten auf. Hiermit war zu rechnen, da die Datenpunkte die Größe des Ausleuchtungsbereichs innerhalb des einsehbaren Bereichs widerspiegeln. Die Anzahl der Datenpunkte stellt deshalb zeitgleich die Anzahl der Aufrufe des Fragment Shaders 13 dar. Für noch höhere Datenpunktzahlen ändert sich der Verlauf schlagartig. Die Laufzeit bleibt ab etwa zwei Millionen Datenpunkten mit ca.

*Tabelle 6-4*: *Standardkonfiguration eines Scheinwerfersystems zur Parameterstudie im Rahmen der Laufzeitanalyse der Lichtverteilungsberechnung.*

| Eigenschaft | Standardwert | Einheit |
|---|---|---|
| **Kamera** | | |
| Seitenverhältnis | 16:9 | - |
| FOV Horizontal | 91,5 | $^\circ$ |
| FOV Vertical | 60 | $^\circ$ |
| Pixel Horizontal | 1.920 | - |
| Pixel Vertikal | 1.080 | - |
| Pixelzahl | 2.073.600 | - |
| **Scheinwerfer** | | |
| Winkelbereich | $[-45, 45] \times [-25, 25]$ | $^\circ$ |
| Winkelfläche | 4.500 | $^{\circ\circ}$ |
| Messauflösung | 0.05 | $\frac{^\circ}{px}$ |
| Datenpunkte | 1.800.000 | - |
| Überdeckung | 91 | % |
| Spektral | Ja | - |
| Bit/Farbkanal | 32 | Bit |

$0,5\,ms$ konstant. Dieses Phänomen kann erklärt werden, indem man die Überdeckung der Kamerasicht durch den Einflussbereichs des Lichts betrachtet. Diese wird durch den orangen Plot in Bild 6-21 visualisiert. Wie aus seinem Verlauf hervorgeht, beginnt das Scheinwerferlicht ab einem Winkelbereich von ungefähr $[-45°, 45°] \times [-30°, 30°]$ den einsehbaren Bereich der Kamera auszufüllen. Eine weitere Vergrößerung des Winkelbereichs erhöht die Anzahl der zu berechnenden Fragmente nicht und wirkt sich deshalb nicht auf die Laufzeit aus. Außerdem zeigt sich, dass nicht benötigte Bereiche innerhalb der Gesamtlichtverteilung keinen negativen Einfluss auf die Laufzeit nehmen. Sie werden nicht in den Cache geladen. Insgesamt kann man festhalten, dass die Performance des Shadings auch für große Winkelbereiche mit ca. $0,5\,ms$ sehr gut ist.

Anstelle des Winkelbereichs wird nun die Messauflösung variiert. Diese führt ebenfalls zu einer Veränderung der Datenpunktzahl. Im Unterschied zur Anpassung des Winkelbereichs liegt dieses Mal eine konstante Überdeckung (91%) der Kamerasicht vor. Der Plot in Bild 6-22 zeigt die Laufzeit des Shadings für Messauflösungen von $0,02°$ bis $0,10°$ pro Datenpunkt.

Da die Verfeinerung der Messauflösung quadratischen Einfluss auf die Anzahl der Datenpunkte nimmt, ist ein umgekehrt quadratischer Zusammenhang mit der Berechnungszeit zu erwarten. Diese Erwartung wird durch den Verlauf des Plots in Bild 6-22 widergespiegelt. Wie sich zeigt, reagiert die Laufzeit sehr empfindlich auf feine Messauflösungen. Gleichzeitig kann die maximal gemessene Laufzeit von $1,2\,ms$ im praxisrelevanten Bereich als vollkommen akzeptabel angesehen werden.

Zuletzt wird der Einfluss der Ausgabeauflösung untersucht. Diese Größe ist im Kontext der Deferred Pipeline besonders ausschlaggebend, da sie die Fragmentzahl des Bilds und damit auch die insgesamt zu beleuchtenden Fragmente vorgibt. Insgesamt wurden mit HD

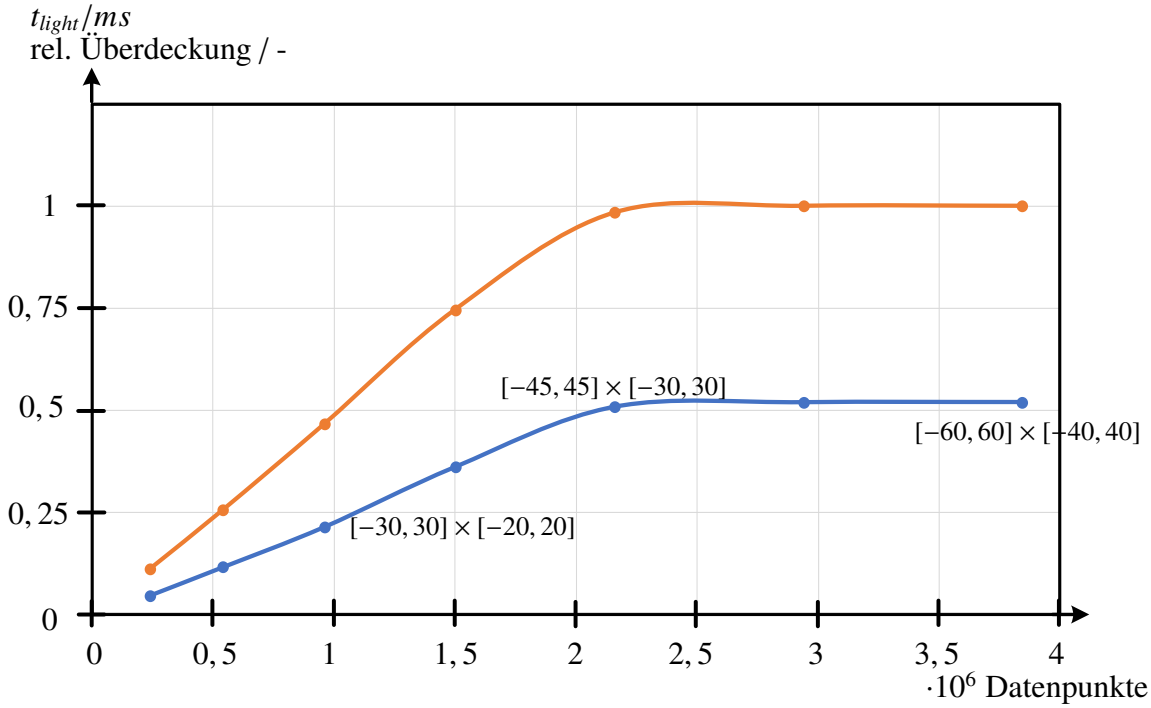


*Bild 6-21*: *Laufzeit der Szenenbelichtung in Abhängigkeit von den Datenpunkten der Gesamtlichtverteilung (blau) und die normierte Überdeckung zwischen Lichtkeule und View Frustum der Kamera (orange).*

Ready, Full HD, Quad HD (QHD) und Ultra HD (4K) vier standardisierte Auflösungen gemessen. Das Bild 6-23 zeigt die Ergebnisse.

Die Laufzeit wird bezüglich der Pixelzahl aufgetragen. Diese ergibt sich durch die Multiplikation von Zeilen- und Spaltenzahl des Bildrasters. Bei konstanter Überdeckung besteht zwischen der Pixelzahl und der Anzahl von Fragment-Shader-Aufrufen ein linearer Zusammenhang. Der Plot in Bild 6-23 entspricht diesen Überlegungen uneingeschränkt. Selbst bei UHD-Auflösung mit über acht Millionen Pixeln kann das Shading in einem Zeitfenster von $1,6\,ms$ abgewickelt werden.

Zusammenfassend kann das Shading durch die in Abschnitt 6.3 vorgestellte Lichtquelle in Bezug auf die Anforderung **A6** (Echtzeitfähigkeit) als unkritisch betrachtet werden. Für typische Konfigurationen liegt der Berechnungsaufwand pro Scheinwerfer bei etwa $1\,ms$. Darüber hinaus werden moderne Hardwarekonfigurationen die Berechnungsdauer noch weiter reduzieren.

## 6.6 Witterung am Beispiel von Nebel

Bei der Analyse bestehender Nachtfahrtsimulationen in Abschnitt 3.3 stellt sich heraus, dass keine der derzeit verfügbaren Lösungen die physikalisch motivierte Simulation von Witterungsbedingungen unterstützt. In Abschnitt 4.1.6 wird jedoch herausgearbeitet, dass die simulative Erprobung insbesondere solcher Fahrsituationen einen hohen Mehrwert in der Entwicklung liefert. Hyperion soll diese Lücke schließen, weshalb die Simulation von Witterungseinflüssen als Anforderung **A9** fixiert wurde. Die Ausprägung verschiedener

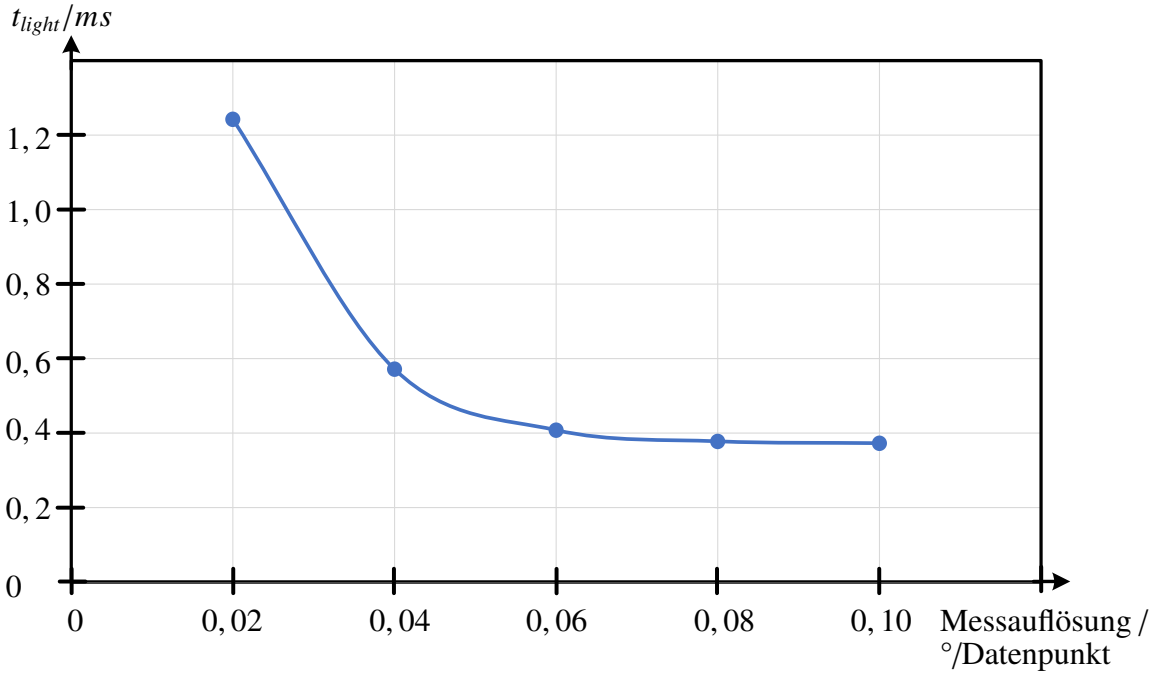


*Bild 6-22: Laufzeit der Szenenbelichtung in Abhängigkeit von der Messauflösung bei konstantem Messbereich bzw. konstanter Überdeckung.*

Wetterphänomene und ihre Wechselwirkungen mit dem Scheinwerferlicht sind sehr komplex. Es ist deshalb nicht möglich, alle Witterungseinflüsse im Rahmen dieser Arbeit zu modellieren. Um die grundsätzliche Eignung der Lichtimplementierung zur Nachbildung von witterungsbedingten Phänomenen nachzuweisen, wird im Folgenden gezeigt, wie eine nächtliche Nebelfahrt implementiert werden kann.

### 6.6.1 Wechselwirkung mit Licht

Jedem Autofahrer ist die starke Sehbeeinträchtigung bei einer Nebelfahrt bekannt. Besonders eingeschränkt ist die Sicht, wenn der Nebel bei Nacht auftritt. In diesem Abschnitt sollen die Ursachen für das erschwerte Sehen bei einer nächtlichen Nebelfahrt angesprochen werden, bevor darauf aufbauend im nachfolgenden Abschnitt 6.6.2 ein geeignetes Nebelmodell abgeleitet wird.

Bei einer klaren Nacht durchdringt das Licht beginnend am jeweiligen Scheinwerfer den Raum bis zum Auftreffen auf ein Umgebungsobjekt. Dort wird es reflektiert, wobei Anteile des reflektierten Lichts auf die Augen des Fahrers treffen und so zur Sichtbarkeit des Objekts führen. Unter Nebelbedingungen gilt diese Aussage nicht. Als Nebel werden viele kleine Wassertropfen, die als Aerosole in der Luft schweben, bezeichnet. Anteile des von den Scheinwerfern ausgesandten Lichts treffen auf ihrem Weg in unterschiedlichen Abständen auf diese Wassertropfen. Dort werden sie absorbiert oder reflektiert. Letzteres führt zu neuen Lichtstrahlen, die sich in zufällige Richtungen ausbreiten. Einige von ihnen bewegen sich entgegengesetzt zum ursprünglichen Lichtstrahl und führen somit zu einer Blendung des Fahrers. Man spricht in dem Fall auch von Eigenblendung, da der Fahrer

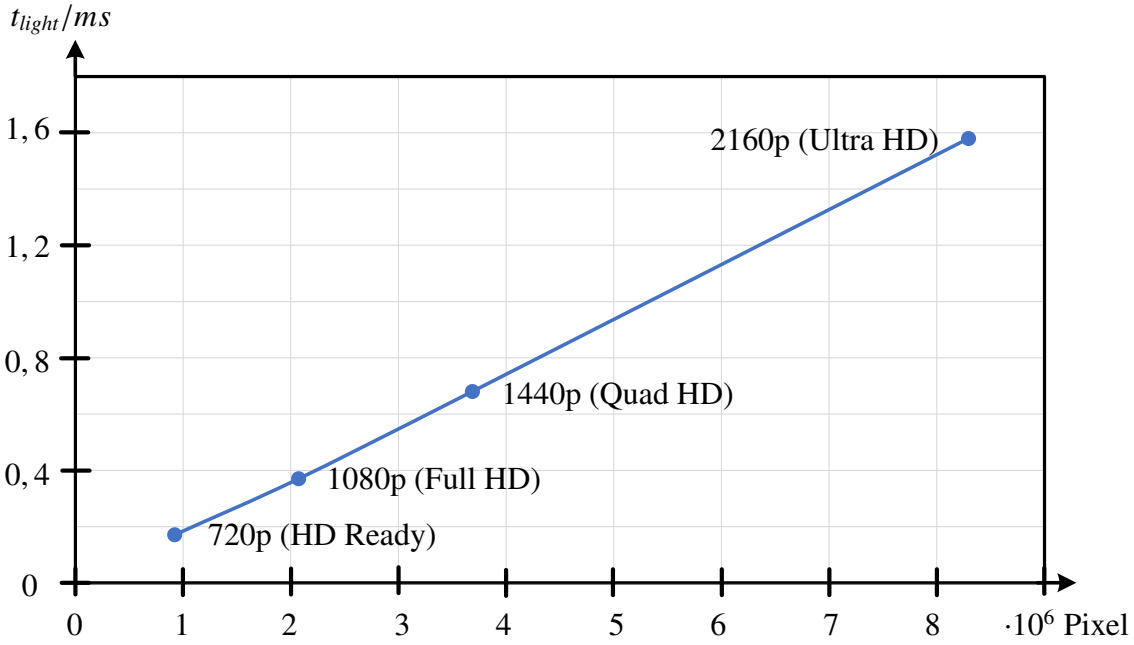


*Bild 6-23*: *Laufzeit der Szenenbelichtung in Abhängigkeit von der Ausgabeauflösung.*

letztlich vom Licht seiner eigenen Scheinwerfer geblendet wird. Die Skizze 6-24 soll diese prinzipiellen Überlegungen veranschaulichen.

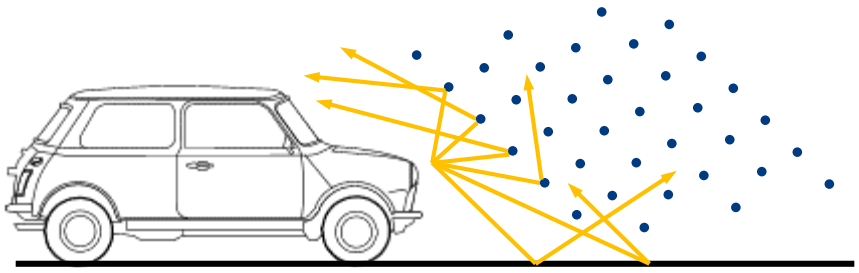

*Bild 6-24*: *Wechselwirkung des Scheinwerferlichts mit Nebel.*

Neben der Eigenblendung ist die erhöhte Abschwächung des Lichts mit zunehmendem Ausbreitungsweg der zweite Effekt, der zu einer eingeschränkten Sicht führt. Licht, das vor dem Auftreffen auf einem Szenenobjekt absorbiert oder in eine andere Richtung reflektiert wurde, reduziert die Beleuchtungsstärke gegenüber dem Erwartungswert unter klaren Sichtverhältnissen. Außerdem wird das vom Objekt reflektierte Licht auf dem Weg zum Fahrer erneut anteilig absorbiert oder abgelenkt, wodurch die Erkennbarkeit des Objekts weiter verschlechtert wird.

Die Möglichkeiten zur Anpassung der Lichtverteilung auf die Wetterbedingungen sind noch Gegenstand der Forschung. Insbesondere die Einführung hochauflösender Scheinwerfersysteme und die damit gewonnene Flexibilität hat der Entwicklung des Schlechtwetterlichts neuen Auftrieb verliehen. Die bereits in Kapitel 3 diskutierten Arbeiten beweisen

die Brisanz dieses Themas. Umso wichtiger ist es, dass theoretische Überlegungen zur wetterabhängigen Anpassung der Lichtverteilung durch Simulation schnell, kostengünstig und gefahrlos evaluiert werden können.

### 6.6.2 Modellierung

Beim bisher vorgestellten Rendering des Licht wird angenommen, dass die Leuchtdichte entlang eines Lichtstrahls konstant ist und Wechselwirkungen zwischen Licht und Szene nur an den Objektoberflächen auftreten. Für mit Licht wechselwirkenden Medien, wie Nebel, gilt diese Annahme nicht. Stattdessen kann das Licht überall im Raum in Interaktion mit diesem Medium treten. Da die Medien Transparenz aufweisen, kann die Frage, ob und wo ein Lichtstrahl mit dem Medium interagiert, nicht sicher vorhergesagt werden. Stattdessen lässt sich eine Kollisionswahrscheinlichkeit $\tau$ angeben. $\tau$ bezeichnet die Wahrscheinlichkeit der Kollision eines Lichtstrahls mit dem Medium während seiner Ausbreitung entlang einer Längeneinheit. Im betrachteten Anwendungsfall korrespondiert $\tau$ mit der Nebeldichte. Im weiteren Verlauf wird eine konstante Nebeldichte angenommen. Grundsätzlich ist das vorgestellte Konzept jedoch auch auf lokal variierende Nebeldichten, wie z.B. Nebelschwaden oder inhomogenen Nebel, erweiterbar, indem $\tau$ ortsabhängig definiert wird. Zur Abbildung der Zusammenhänge wird die in Abschnitt 2.3.4 vorgestellte Rendergleichung zur volumetrischen Rendergleichung oder Strahlungstransportgleichung [SSS08] erweitert:

$$\frac{dL_v(v(s),\Omega)}{ds} = -\tau L_v(v(s),\Omega) + \tau a_{ref} \int_{\Omega' \in \Omega^+} L_v(v(s),\Omega') p_{Ph}(\Omega',\Omega) d\Omega'. \tag{6-8}$$

Das Bild 6-25 unterstützt beim Verständnis der Größen. In Gleichung (6-8) bezeichnet $L_v(v(s),\Omega)$ die Leuchtdichte am Ort $v(s)$ in Richtung $\Omega$. Betrachtet wird ein Lichtstrahl der am Ort $v(0)$ beginnt und sich in Richtung $\Omega$ ausbreitet. Initial weist dieser Strahl die Leuchtdichte $L_v(v(0),\Omega)$ auf. Bei der Ausbreitung durch das wechselwirkende Medium verändert sich die Leuchtdichte des Strahls. Betrachtet man ein differentielles Streckenelement $ds$, so kann die Veränderung des Lichtstrahls entlang dieses Streckenelements durch die rechte Seite der Gleichung (6-8) beschrieben werden. Der rot dargestellte Term $\tau L_v(v(s),\Omega)$ ist der Verlustterm. Er umfasst den Anteil, welcher mit dem Medium kollidiert und von diesem entweder absorbiert oder in eine andere Richtung reflektiert wird. In beiden Fällen ist die Leuchtdichte aus Sicht des betrachteten Lichtstrahls entlang $v(s)$ verloren. Das in grün visualisierte Integral umfasst die Zunahme der Leuchtdichte. Sie kommt zustande, indem Strahlen, die sich entlang anderer Pfade ausbreiten den Pfad $v(s)$ kreuzen und Anteile dieser Strahlen am Kreuzungspunkt kollidieren. Im Zuge der Kollision leistet jedoch nur der Anteil einen Beitrag zum betrachteten Strahl $v(s)$, der in Richtung $\Omega$ reflektiert wird. Die Wahrscheinlichkeit, dass die Kollision zur Reflexion führt, wird durch den Parameter $a_{ref}$ beschrieben. Weiterhin beschreibt die Phasenfunktion $p_{Ph}(\Omega',\Omega)$ die Wahrscheinlichkeit, dass ein aus Richtung $\Omega'$ eintreffender Strahl in Richtung $\Omega$ reflektiert wird. Um eintreffende Strahlen aus allen Richtungen $\Omega'$ zu berücksichtigen, wird über den vollen Raumwinkel $\Omega^+$ integriert.

Mit Gleichung (6-8) kann die Wechselwirkung des Lichts mit den Nebelpartikeln modelliert werden. Problematisch ist jedoch, dass es sich um eine Integro-Differential-Gleichung

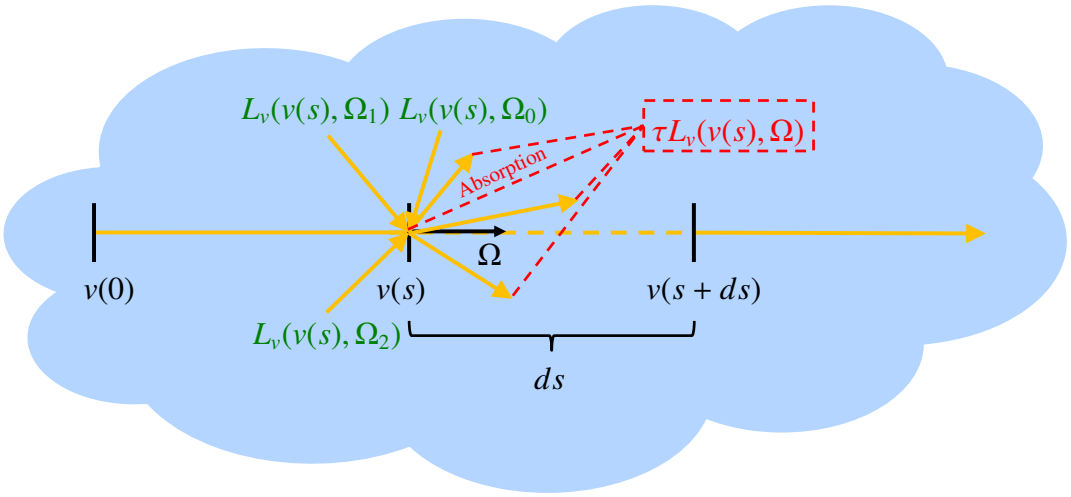


*Bild 6-25*: *Visualisierung der Größen in der Strahlungstransportgleichung.*

handelt, welche nicht analytisch gelöst werden kann. Im Gebiet des fotorealistischen Renderings wird die Lösung der Gleichung (6-8) durch Monte-Carlo-Methoden approximiert [SSS08]. Als Anwendung im Echtzeit-Rendering ist diese Vorgehensweise jedoch ungeeignet. Um die Wechselwirkung des Lichts in Echtzeit zu simulieren, ist es notwendig, einige Vereinfachungen zu treffen [TU09].

**Vereinfachung 1:** Nachfolgend wird angenommen, dass jeder Lichtstrahl auf dem Weg von der Lichtquelle bis zum Auftreffen auf einem Objekt der Szene maximal einmal reflektiert wird. Mit dieser Annahme wird die Realität offensichtlich verfälscht. Da jedoch eine Vielzahl von Lichtstrahlen betrachtet wird und reale Lichtstrahlen mit zunehmender Anzahl von Reflexionen an Leuchtdichte verlieren, kann diese Vereinfachung getroffen werden, ohne sich gänzlich von den realen Gegebenheiten zu entfernen. Sie hat rechentechnisch positive Auswirkungen auf den grün dargestellten Term der Gleichung (6-8), da nun davon ausgegangen werden kann, dass die darin aufgeführten Leuchtdichten $L_v(v(s),\Omega')$ stets unmittelbar von einer Lichtquelle der Szene stammen. Der grün dargestellte Term kann somit gemäß

$$\tau a_{ref} \int_{\Omega' \in \Omega^+} L_v(v(s),\Omega') p_{Ph}(\Omega',\Omega) d\Omega' \rightarrow \sum_{i \in I} \tau a_{ref} \frac{I_{v,i}(\Omega_i)}{r_i^2} e^{-\tau r_i} p_{Ph}(\Omega_i,\Omega) =: L_{v,i}(v(s),\Omega) \tag{6-9}$$

neu formuliert werden. Anstelle des Integrals über alle Raumrichtungen genügt nun die Summe über alle Lichtquellen der Szene, welche in der Indexmenge $I$ zusammengefasst werden. Im Bild 6-26 werden die Größen der Gleichung (6-9) für eine Beispiellichtquelle visualisiert. Die von einer Lichtquelle $i$ am Ort $v(s)$ hervorgerufene Leuchtdichte kann durch deren Lichtverteilung $I_{v,i}$ und den Abstand $r_i$ von der Lichtquellenposition $v_i$ zum Punkt $v(s)$ beschrieben werden. Dazu muss die Lichtstärke von der Lichtquelle in Richtung $\Omega_i$ durch das Quadrat des Abstands zwischen $v_i$ und $v(s)$ geteilt werden. Hinzu kommt, dass Anteile des Lichts auf dem Weg von der Lichtquelle bis hin zu $v(s)$ bereits absorbiert oder in andere Richtungen reflektiert wurden. Zur Berücksichtigung dieser Verluste muss die Leuchtdichte mit $e^{-\tau r_i}$ multipliziert werden.

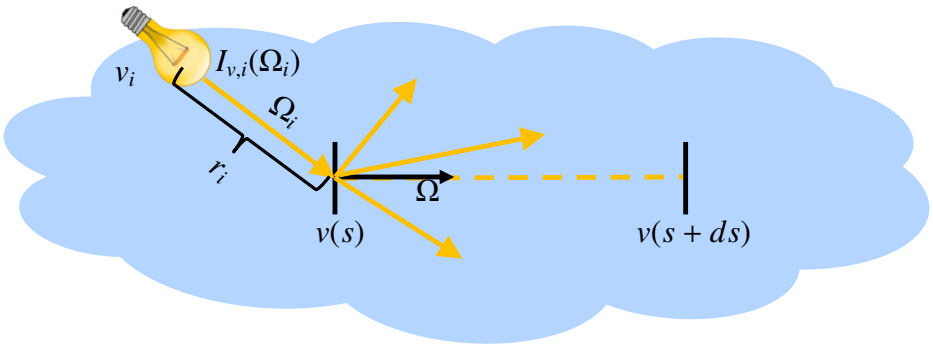


*Bild 6-26*: *Vereinfachung 1 zur Erlangung einer echtzeitfähigen Lösung der volumetrischen Rendergleichung.*

Die volumetrische Rendergleichung vereinfacht sich unter der getroffenen Annahme zu

$$\frac{dL_v(v(s),\Omega)}{ds} = -\tau L_v(v(s),\Omega) + L_{v,i}(v(s),\Omega)$$

und nimmt die Gestalt einer gewöhnlichen Differentialgleichung an. Ihre analytische Lösung kann nun formuliert werden:

$$L_v(v(s),\Omega) = e^{-\tau s} L_v(v(0),\Omega) + \int_{l=0}^{s} L_{v,i}(v(l),\Omega) e^{-\tau(s-l)} dl. \tag{6-10}$$

**Vereinfachung 2:** Verglichen mit der ursprünglichen volumetrischen Rendergleichung (6-8) stellt die Lösung (6-10) einen deutlich effizienteren Ausdruck dar. Das Integral der Gleichung (6-10) ist jedoch ebenfalls nicht analytisch lösbar. Es wird gemäß

$$\int_{l=0}^{s} L_{v,i}(v(l),\Omega) e^{-\tau(s-l)} dl \approx \sum_{n=0}^{N} L_{v,i}(v(l_n),\Omega) e^{-\tau(s-l_n)} \Delta l \text{ mit } l_n = n\Delta l, N = \lfloor \frac{s}{l} \rceil$$

durch eine Riemann-Summe approximiert. Die Schrittweite $\Delta l$ ist parametrierbar und beeinflusst die Genauigkeit der Approximation.

**Vereinfachung/Annahme 3:** Die Phasenfunktion $p_{Ph}(\Omega_l, \Omega)$ wurde bislang nicht näher spezifiziert. Sie beschreibt die Wahrscheinlichkeit, dass ein unter dem Raumwinkel $\Omega_l$ einfallender Lichtstrahl an einem Nebelpartikel in Richtung des Raumwinkels $\Omega$ reflektiert wird. Vereinfachend wird angenommen, dass $\Omega_l$ und $\Omega$ nicht korrelieren. Die Wahrscheinlichkeit ist unabhängig von der Einfallsrichtung für alle Reflexionsrichtungen gleich. Da die Phasenfunktion eine Wahrscheinlichkeitsdichtefunktion ist, muss sie die Voraussetzungen für eine solche erfüllen. Insbesondere muss das Integral der Funktion über der Grundmenge 1 bzw. 100% sein. Die Grundmenge stellt hier den vollen Raumwinkel von $4\pi$ dar. Um die zuvor genannten Annahmen mathematisch abzubilden, muss die Phasenfunktion als Konstante formuliert werden:

$$p_{Ph}(\Omega_l, \Omega) := \frac{1}{4\pi}.$$

Diese Definition hat zur Folge, dass die $L_{v,i}$-Terme ihre Abhängigkeit von $\Omega$ verlieren.

Unter der Berücksichtigung aller drei Vereinfachungen ergibt sich abschließend die vereinfachte volumetrische Rendergleichung

$$L_v(v(s), \Omega) = e^{-\tau s} L_v(v(0), \Omega) + \sum_{n=0}^{N} L_{v,i}(v(l_n)) e^{-\tau(s-l_n)} \Delta l$$
$$\text{mit } L_{v,i}(v) = \tau a_{ref} \sum_{i \in I} \frac{I_v(\Omega_i)}{4\pi r_i^2} e^{-\tau r_i}. \tag{6-11}$$

### 6.6.3 Implementierung

Die vereinfachte volumetrische Rendergleichung (6-11) kann nun durch einen Ray Marching Ansatz implementiert werden [KH84]. Das Bild 6-27 visualisiert das damit verbundene Vorgehen. Ausgehend von der Kameraposition wird durch jedes Pixel des Frame Buffers ein Ray verfolgt. Dieser Ray wird mit einer gleich bleibenden Schrittweite gerastert und an den diskreten Rasterstellen (s. blaue Punkte in Bild 6-27) ausgewertet. Beginnend am getroffenen Szenenobjekt werden die Lichtbeiträge der Rasterstellen entsprechend den einzelnen Summanden in der Summenformel aus Gleichung (6-11) summiert.

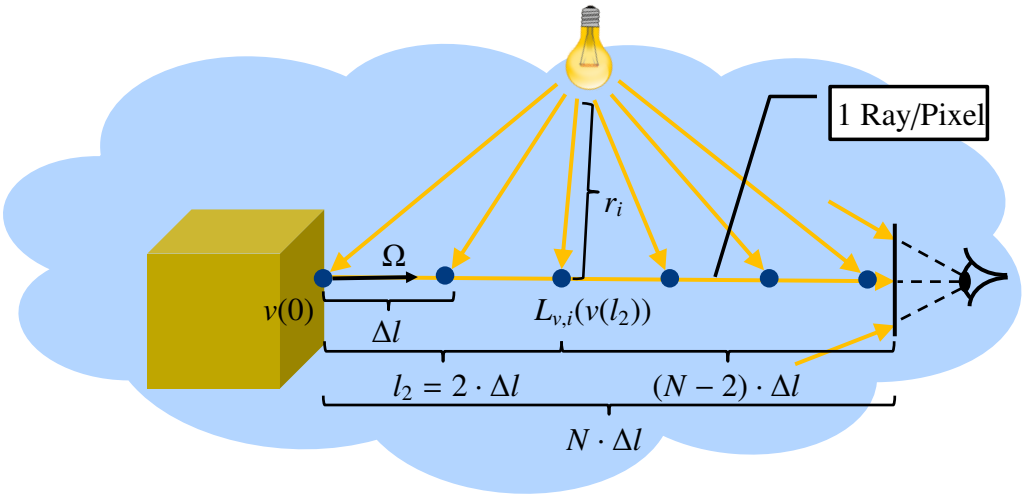


*Bild 6-27*: *Implementierung der vereinfachten volumetrischen Rendergleichung mittels Ray Marching.*

Das genaue Vorgehen wird im Algorithmus 14 formuliert. Begonnen wird am Ende $v(0)$ des Rays und somit am getroffenen Szenenobjekt. Eine umgekehrte Vorgehensweise beginnend an der Kamera wäre ebenfalls möglich. Der Leuchtdichtebeitrag $L_v(v(0), \Omega)$ am Punkt $v(0)$ entspricht der Beleuchtungssituation ohne Nebel. Insofern kann das Beleuchtungsmodell bei $v(0)$ auf herkömmliche Weise ausgewertet werden, wobei die Abschwächung des Lichts von der Quelle zum Punkt $v(0)$ durch den Faktor $e^{-\tau r_i}$ zu berücksichtigen ist (Zeile 4). Die Beiträge aller Rasterpunkte werden in der Variablen $L_{sum}$ summiert. Sie wird in Zeile 5 mit dem Lichtbeitrag bei $v(0)$ initialisiert. Zusätzlich wird durch den Faktor $e^{-\tau s}$ berücksichtigt, dass das am Szenenobjekt reflektierte Licht auf dem Weg bis zur Kamera

durch den Nebel geschwächt wird. Bis hierhin beinhaltet $L_{sum}$ den linken Summanden der Gleichung (6-11).

Durch die nachfolgende For-Schleife wird die Summenformel bestimmt. Dazu wird der Ray beginnend beim nächsten Rasterpunkt $n = 1$ bis zum letzten Rasterpunkt vor der Near Clipping Plane abgeschritten und das Argument der Summenformel aus Gleichung (6-11) ausgewertet (Zeile 7). Die Werte werden auf den bestehenden Wert von $L_{sum}$ addiert. Nach der Terminierung dieser Iteration beinhaltet $L_{sum}$ die Lichtbeiträge aller Rasterpunkte und somit den Lichtbeitrag, der am betrachteten Pixel durch die Kamera wahrgenommen wird. $L_{sum}$ wird als Resultat zurückgegeben.

**Algorithm 14** Simplified Volumetric Rendering

1: **require:** $v_{cam}$ position of camera, $v_i$ position of each light $i \in I$, $I_{v,i}$ luminous intensity distribution of each light $i \in I$, $\tau \in \mathbb{R}$ fog density, $a_{ref} \in \mathbb{R}$ reflection probability, $\Delta l \in \mathbb{R}$ step size
2: **function** RENDERVOLUMETRIC($v(0) \in \mathbb{R}^3$ position of ray start)
3: **local variables:** $L_v(v(0), \Omega) \in \mathbb{R}^4$ light at object surface, $L_{sum} \in \mathbb{R}^4$
4: $L_v(v(0), \Omega) \leftarrow L_{\cancel{fog}}(v(0), \Omega)$ with $I_{v,i}(\Omega_i)e^{-\tau r_i}$
5: $L_{sum} \leftarrow e^{-\tau N \Delta l} L_v(v(0), \Omega)$
6: **for** $n = 1$ to $N$ **do**
7: $L_{sum} \leftarrow L_{sum} + L_{v,i}(v(n \cdot \Delta l))e^{-\tau(N-n)\Delta l}\Delta l$
8: **end for**
9: **return** $L_{sum}$
10: **end function**

Der Algorithmus 14 stellt das Vorgehen des volumetrischen Renderings von Nebel im Kern dar. Bei der praktischen Umsetzung müssen einige weitere Details beachtet werden, die hier nur kurz angeführt werden sollen.

**Rekonstruktion der räumlichen Lagebeziehungen:** Die Umsetzung des Algorithmus 14 erfolgt in Hyperion durch einen Image Effect Shader. Derartige Shader kommen erst nach dem vollständigen Rendering der Szene zum Einsatz. Bei Verwendung der Deferred Rendering Pipeline wird der Nebeleinfluss somit erst nach Abschluss des Lighting Pass berücksichtigt. Zur Rekonstruktion der Lagebeziehungen, welche zur Implementierung des Algorithmus 14 bekannt sein müssen, empfiehlt sich deshalb das für den Headlamp Shader (Algorithmus 12) diskutierte Vorgehen. Die Vertices des Lichtvolumens stellen den Eingang für die Vertex Stufe des Nebelrenderings dar. Innerhalb der Vertex Stufe wird ein Ray-Vektor generiert, welcher von der Kamera auf den gerade verarbeiteten Vertex des Lichtvolumens zeigt. In der späteren Fragment Stufe kann die Interpolation dieses Ray-Vektors gemeinsam mit den Texturkoordinaten des betrachteten Fragments und dem z-Buffer genutzt werden, um die räumliche Lage zu rekonstruieren.

**Start und Ende der Rays:** Der in Bild 6-27 beispielhaft skizzierte Ray beginnt an der Near Clipping Plane und endet am Szenenobjekt. Diese Situation trifft nicht immer zu. So kann es beispielsweise sein, dass ein Ray kein Objekt der Szenen trifft und deshalb spätestens an der Far Clipping Plane beendet werden muss. Außerdem führen sehr lange Rays zu sehr vielen Rasterpunkten, wodurch die Anzahl der Schleifeniterationen in Algorithmus 14 sehr hoch werden kann. Es sollte deshalb eine geeignete Obergrenze vorgesehen werden, welche die Ausführung in Echtzeit zulässt.

**Zufälliger Startoffset der Rays:** Das deterministische Vorgehen der Rasterung aller Rays, wie es in Algorithmus 14 beschrieben wird, führt zu einer Auswertung des Nebeleinflusses in dünnen Scheiben mit dazwischen befindlichen Leerräume der Dicke $\Delta l$. Zur besseren Verteilung der Rasterpunkte im auszuwertenden Volumen empfiehlt es sich, jeden Ray mit einem im Intervall $[0, \Delta l]$ zufällig gewählten Startoffset zu versehen. In der hier gewählten Implementierung wird hierzu eine Blue-Noise-Textur verwendet, welche abhängig vom gerade betrachteten Pixel an unterschiedlichen Stellen ausgelesen wird [Uli88].

**Globaler Nebel:** Das hier beschriebene Vorgehen beschreibt eine Methode zur Darstellung der Wechselwirkung zwischen dem Scheinwerferlicht und den Nebelpartikeln. Prinzipiell könnte diese Methode auf alle lokalen Lichtquellen angewendet werden. Zur Sicherung der Echtzeitfähigkeit sollte hiervon jedoch abgesehen werden. Hinzu kommt, dass globale Lichtquellen, wie gerichtetes oder ambientes Licht, mit der beschriebenen Methode nicht abgebildet werden können. Es ist deshalb notwendig neben der beschriebenen Implementierung einen globalen Nebel vorzusehen. In Hyperion wird hierzu eine erweiterte Variante der Implementierung aus der Unity-Engine eingesetzt. Die Abstimmung des globalen und volumetrischen Nebels zueinander wird durch geeignete Parameter, wie z.B. der Nebeldichte, ermöglicht.

**Weichzeichnung:** In der beschriebenen Implementierung wird ein Ray pro Pixel der Ausgabe erzeugt. Hierbei werden die Bildinformationen im Vergleich zu den realen Gegebenheiten stark reduziert. Mit steigender Auflösung oder der Interpolation zwischen mehreren Rays innerhalb eines Pixels wird das Rendering-Ergebnis genauer. Gleichzeitig haben derartige Maßnahmen einen negativen Einfluss auf die Berechnungszeit und gefährden somit die Anforderung **A6** (Echtzeitfähigkeit). Stattdessen kann durch das nachträgliche Weichzeichnen ein vergleichbares Resultat ohne nennenswerte Laufzeiteinbußen erzielt werden. In der hier vorgestellten Lösung wird dazu ein Gauß-Filter eingesetzt [Jäh05].

### 6.6.4 Renderergebnis

Als Abschluss des Abschnitts 6.6 wird das Ergebnis der Nebelimplementierung vorgestellt. Hierzu sei zunächst auf Bild 6-28 verwiesen, welches einen globalen Nebel zeigt, der durch Unity-Bordmittel ohne weiteren Implementierungsaufwand erzeugt werden kann. Wie sich zeigt, ist dieser globale Nebel für die Entwicklung von Scheinwerfersystemen absolut ungeeignet, da er keine Wechselwirkung mit Licht erlaubt. Der in Unterabschnitt 6.6.1 diskutierte Effekt der weißen Wand bleibt gänzlich aus. Auch wenn globaler Nebel sehr effizient berechnet werden kann, erfüllt er nicht die qualitativen Voraussetzungen der Anforderung **A9** (Simulation von Witterungseinflüssen).

Im Vergleich dazu zeigen die Bilder 6-29c und 6-30 die Resultate der hier vorgestellten volumetrischen Implementierung für eine Abblendlicht- und eine Fernlichtverteilung. Der Unterschied zum globalen Nebel der Unity-Engine ist deutlich erkennbar. Vor den Frontscheinwerfern bilden sich Lichtkeulen aus. Besonders erwähnenswert ist, dass diese Lichtkeulen ihre Gestalt der momentanen Ansteuerung des Scheinwerfers anpassen. So kann die Stufe der Hell-Dunkel-Grenze in der Abblendlichtverteilung als weiche Kante in den Lichtkeulen der Scheinwerfer wahrgenommen werden. Genauso ist klar erkennbar, dass der wesentlich höhere Lichtstrom einer Fernlichtverteilung zu helleren Lichtkeulen vor


*Bild 6-28*: *Globaler Nebel der Unity-Engine.*

dem Fahrzeug führt. Der Effekt einer weißen Wand kann demnach durch die vorgestellte Implementierung reproduziert werden.

In den Bildern 6-29a und 6-29b sind Zwischenergebnisse des Renderings visualisiert, wie sie auf dem Weg zum finalen Szenenbild 6-29c entstehen. Konkret zeigt Bild 6-29a die zunächst isoliert gerenderten Lichtbeiträge der angestrahlten Nebelpartikel. Hierbei werden bereits die vorliegende Perspektive und eventuellen Verdeckungen berücksichtigt, um das spätere Blending der Nebeltextur mit der Szenentextur zu ermöglichen. In Bild 6-29a werden bereits die zum Ende von Abschnitt 6.6.3 kurz erwähnten zufälligen Startoffsets der einzelnen Rays verwendet. Hierdurch kann die Scheibenbildung vermieden werden. Die strahlenden Nebelpartikel wirken gleichmäßig verteilt. Dennoch führt die diskrete Abtastung zu erkennbaren Rauscheffekten. Diese werden in einem nachgelagerten Schritt durch die ebenfalls am Ende des Abschnitt 6.6.3 erwähnte Weichzeichnung eliminiert. Das Bild 6-29b zeigt die Weichzeichnung des Bilds 6-29a durch dreimaliges Anwenden des örtlichen Tiefpassfilters. Das Filterergebnis kann letztlich durch additives Blending in die Szene integriert werden.

Die vorgestellte Umsetzung eines volumetrischen Nebel zeigt die grundsätzliche Machbarkeit der Echtzeit-Simulation von Witterungsbedingungen im Rahmen von Hyperion. Die genaue Validierung der simulierten Nebeleffekte anhand realer Referenzen wurde im Rahmen der Arbeit nicht betrachtet. Es sei jedoch erwähnt, dass die hier vorgestellte Implementierung bereits in anderen Untersuchungen herangezogen wurde [Tho21].

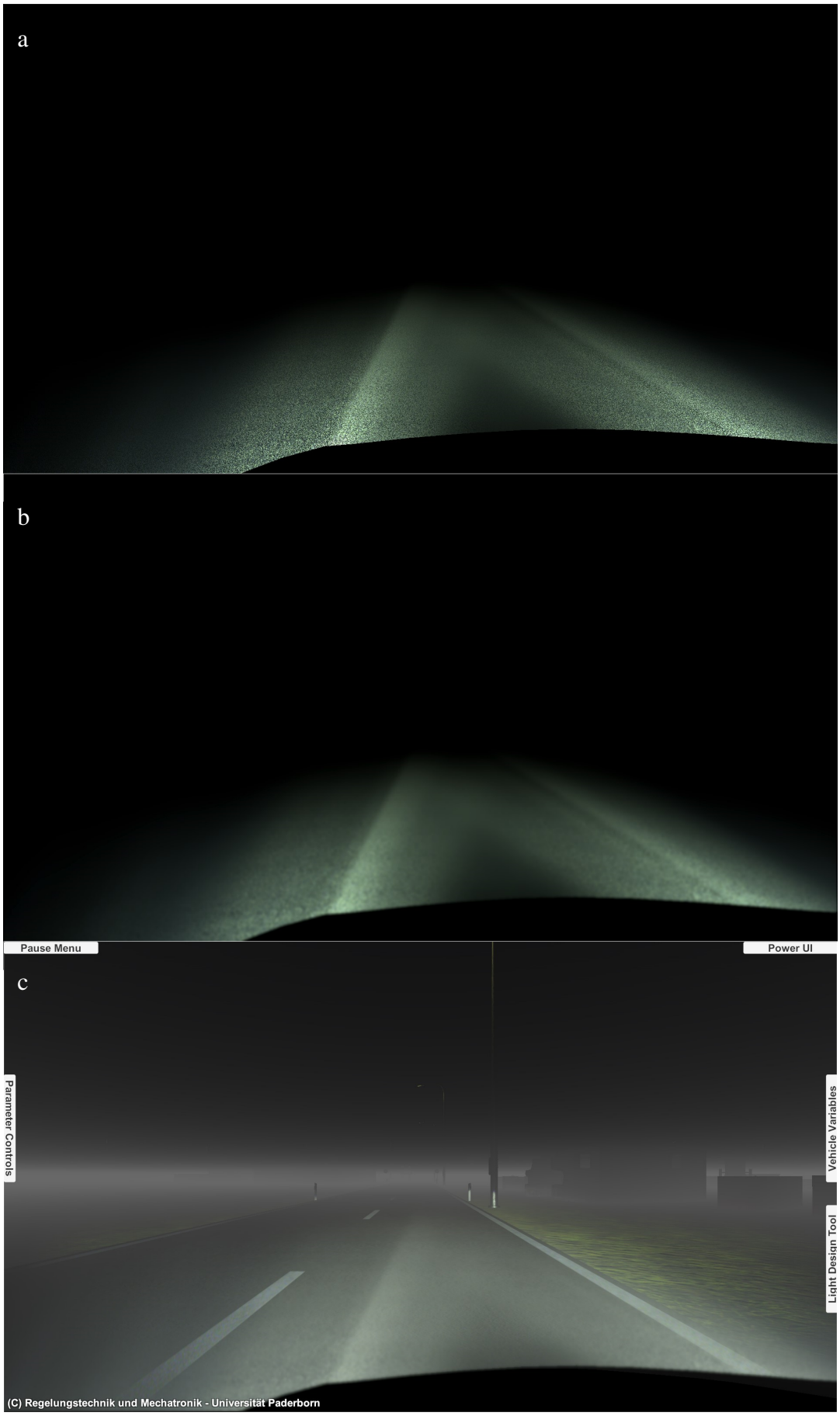


*Bild 6-29*: *Renderergebnisse des volumetrischen Nebels bei Abblendlicht.*


*Bild 6-30*: *Renderergebnisse des volumetrischen Nebels bei Fernlicht.*

# 7 Analyse und Entwurf von Lichtfunktionen

Die echtzeitfähige Implementierung des Lichts von HD-Scheinwerfern innerhalb der Nachtfahrtsimulation „Hyperion“ wurde im vorangegangenen Kapitel detailliert vorgestellt. Sie ermöglicht eine Erprobung virtueller Prototypen bereits vor der physischen Realisierung. HD-Systeme sind hochflexibel und können dynamisch zwischen verschiedensten Lichtverteilungen wechseln. Dementsprechend kommt den Lichtfunktionen, welche die momentane Gestalt der Lichtverteilung bestimmen, bei der Erprobung ein besonderer Stellenwert zu. Fallen im Rahmen dieser Erprobung Fehler oder Schwachstellen auf, gilt es diese zu korrigieren. Dieses Kapitel umfasst Tools und Ansätze, die den Entwickler sowohl bei der Erkennung als auch bei der Beseitigung derartiger Mängel unterstützen.

## 7.1 Analyse

Der erste Teil des Kapitels befasst sich mit der Analyse eines Scheinwerfersystems. Die detaillierte Erfassung des Ist-Zustands steht im Vordergrund. Dazu benennt Abschnitt 7.1.1 die photometrischen Größen, die zur Analyse des Scheinwerferlichts zur Verfügung stehen und erläutert ihre Aussagekraft. Der nachfolgende Abschnitt 7.1.2 stellt die in Hyperion zur Verfügung stehenden Visualisierungen dieser Größen vor. Da die Analyse des Lichts nur den Ausgang des Scheinwerfersystems erfasst, umfasst Abschnitt 7.1.3 Visualisierungen der Systemeingänge, welche im betrachteten Kontext durch Sensorinformationen gebildet werden.

### 7.1.1 Photometrische Größen

Bevor die verschiedenen Visualisierungen im Abschnitt 7.1.2 thematisiert werden, beschreibt dieser Abschnitt die Größen, deren Visualisierungen bei der Analyse eines Scheinwerfersystems hilfreich sein kann. In der Photometrie sind hier die Lichtstärke, die Beleuchtungsstärke und die Leuchtdichte örtlich aufgelöst in der Szene zu nennen. Jede dieser Größen hat Vorzüge und Nachteile bei der Analyse, da verschieden viele Einflussgrößen Berücksichtigung finden. Diese sind in Tabelle 7-1 aufgeführt. Die Wahrnehmung

*Tabelle 7-1*: *Photometrische Größen und ihre Abhängigkeiten von der Umgebung.*

| Größe | Einflüsse |
|---|---|
| Lichtstärke $I_v$ | Polarwinkel, Azimutwinkel |
| Beleuchtungsstärke $E_v$ | + Distanz, Einfallswinkel |
| Leuchtdichte $L_v$ | + Materialeigenschaften (BRDF) |

des menschlichen Auges ist im photometrischen Sinne an die Leuchtdichte gebunden. Diese ist jedoch bedingt durch die vielen Einflussgrößen sehr komplex. Neben geometrischen Faktoren, welche auch die Lichtstärke und Beleuchtungsstärke bestimmen, fließen

Materialeigenschaften ein, welche das Reflexionsverhalten der Oberfläche beeinflussen. Beispiele hierfür sind die Rauigkeit der Oberfläche, die spektralen Eigenschaften und die Dichte des Materials, die Anisotropie und weitere Einflussgrößen [Geb03]. Gerade diese Materialeigenschaften sind sehr komplex und können im Rahmen dieser Arbeit nur grob approximiert Berücksichtigung finden. In der Konsequenz kann der simulativ ermittelte Leuchtdichtewert eines Szenenpunkts stark vom realen Wert abweichen. Auf eine Visualisierung der Leuchtdichte wird deshalb verzichtet.

Hinzu kommt, dass der Lichtingenieur oftmals weniger an der Leuchtdichte als an der Beleuchtungsstärke interessiert ist. Die Beleuchtungsstärke gibt die örtlich aufgelösten Lichtbeiträge des Scheinwerfers basierend auf der Lichtstärkeverteilung und der Geometrie der Szene (Distanz und Einfallswinkel) wieder. Auf diese Weise wird der Einfluss des Scheinwerferlichts nicht durch unterschiedlich stark reflektierende Objekte der Szene verfälscht. Auf das Reflexionsverhalten kann im realen Umfeld ohnehin keinen Einfluss genommen werden. Neben der Leuchtdichte ist die Beleuchtungsstärke somit eine wichtige Analysegröße für die Erprobung eines Scheinwerfersystems.

Doch auch die Beleuchtungsstärke verzerrt die Lichtstärkeverteilung bedingt durch die Variation der Distanzen und der Einfallswinkel innerhalb der Szene erheblich. Möchte der Lichtingenieur grundlegend prüfen, ob die Konturen der Lichtstärkeverteilung für eine gegebene Situation korrekt gestaltet sind, kann es sinnvoll sein, auch die Einflüsse auf die Beleuchtungsstärke außer Acht zu lassen und gezielt die Lichtstärke zu betrachten. In diese fließt ausschließlich die Richtung ein, die vom jeweiligen Scheinwerfer auf das betrachtete Szenenelement weist. Diese Richtung wird formal durch den Polar- und Azimutwinkel im lokalen Scheinwerferkoordinatensystem beschrieben.

Das Bild 7-1 fasst die angestellten Überlegungen zusammen und visualisiert die Diskrepanz zwischen der menschlichen Wahrnehmung und der technischen Beeinflussbarkeit, der Situationsunabhängigkeit sowie der Präzision der simulativen Berechnungen. Aufgrund dieses Kompromisses hat jede der betrachteten photometrischen Größen eigene Vorzüge und Nachteile bei der Scheinwerferanalyse.

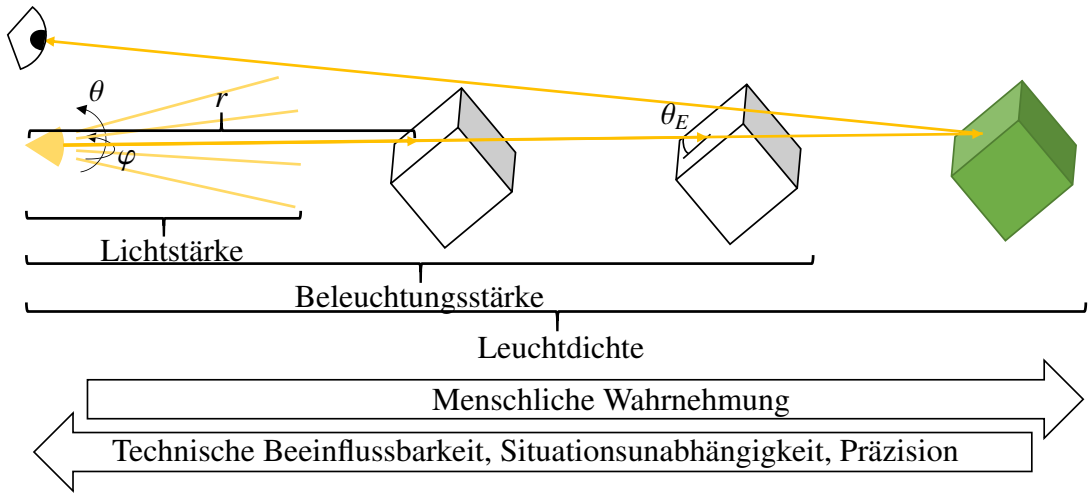


*Bild 7-1*: *Photometrische Größen zur Analyse von Scheinwerfersystemen.*

### 7.1.2 Analysesichten

Zur Erfüllung der Anforderung **A7** können die im Abschnitt 7.1.1 beschriebenen Analysegrößen in Hyperion auf unterschiedliche Weise visualisiert werden. Derartige Analysesichten etablierten sich in der Scheinwerferentwicklung auch schon vor dem Aufkommen der HD-Technologie. Typischerweise werden die photometrischen Größen durch Isolinien- oder Falschfarbendarstellungen visualisiert. Im Rahmen dieser Arbeit wurden die bestehenden Konzepte lediglich in die HD-Domäne überführt. Die Bilder im Anhang A5.1 zeigen einige Impressionen. Da die Lichtstärkeverteilungen der Scheinwerfer nach der Bestimmung der Gesamtlichtverteilung gemäß Abschnitt 6.2.3 vorliegen, unterscheidet sich die Implementierung der Analysesichten prinzipiell nicht von klassischen Scheinwerfersystemen. Dementsprechend werden die etablierten Analysesichten nicht ausführlicher betrachtet. Neben den Visualisierungen der photometrischen Größen stellt Hyperion weitere etablierte Analysetools, wie beispielsweise die Einblendung von Distanzmarken oder einer Messwand, zur Verfügung. Sie können im Anhang A5.2 eingesehen werden.

Mit dem Wandel der Scheinwerfertechnik im Zuge der HD-Technologie entstehen darüber hinaus neue Möglichkeiten zur Analyse. Im Folgenden soll eine Analysesicht vorgestellt werden, die insbesondere bei der Beurteilung der Lichtsteueralgorithmen hilft, indem sie die Dimmwerte der Pixellichtquellen im Scheinwerfer visualisiert. Sie wird im weiteren Verlauf als Dimmwertanalyse bezeichnet. Konkret werden zwei Varianten der Dimmwertanalyse vorgestellt, die abhängig von der Auflösung des Scheinwerfersystems ausgewählt werden sollten.

Das Bild 7-2 zeigt die Dimmwertanalyse für das HD84-System. Aufgrund der verhältnismäßig geringen Anzahl von Pixellichtquellen dieses Scheinwerfersystems wird die Dimmwertanalyse in seiner ersten Variante gezeigt. Diese zeichnet sich dadurch aus, dass die Dimmwerte aller Pixellichter in voneinander separierten Kästchen visualisiert werden (s. Bild 7-2 unten). Die Anordnung der Kästchen entspricht den Einbaupositionen der Pixellichter im Matrix-Modul. Die Position eines Pixellichts gibt gleichzeitig Aufschluss über den Raumwinkelbereich, in dem es strahlt. Außerdem tragen die Kästchen Indizes und erlauben darüber eine direkte Zuordnung zu den Lichtquellen der Scheinwerfer. Erwartungsgemäß werden die Dimmwerte des linken Scheinwerfers in der unteren linken Ecke der Anzeige dargestellt. Entsprechend verhält es sich für den rechten Scheinwerfer. Der Dimmwert der jeweiligen Pixellichtquelle wird durch Falschfarben visualisiert. Ist die Lichtquelle ausgeschaltet, so ist das entsprechende Kästchen tiefblau gefärbt. Bei maximaler Bestromung ist das Kästchen tiefrot. Alle Abstufungen ordnen sich im Farbverlauf von blau über grün, gelb und orange bis rot ein. Parallel zu den Dimmwerten werden im oberen Bereich die sich ergebenden Lichtstärkeverteilungen der Scheinwerfer bezüglich des Polar- und Azimutwinkels durch Falschfarbendarstellungen visualisiert.

Der Lichtingenieur erhält auf diese Weise ein direktes Feedback über die Ausgabe der Lichtsteueralgorithmen und kann hieraus gezielt Anpassungen ableiten. Des weiteren stellt die Dimmwertanalyse Möglichkeiten zur Fehlerinjektion bereit. Durch Rechtsklick auf ein Kästchen, kann die korrespondierende Pixellichtquelle von der Funktionstüchtigkeit bis zum Totalausfall stufenweise degradiert werden. Die Konsequenzen für die Lichtverteilungen und die Ausleuchtung der Szene werden unmittelbar sichtbar. Darüber hinaus kann die Dimmwertanalyse parallel zu den bereits genannten, klassischen Analysesichten eingesetzt werden.

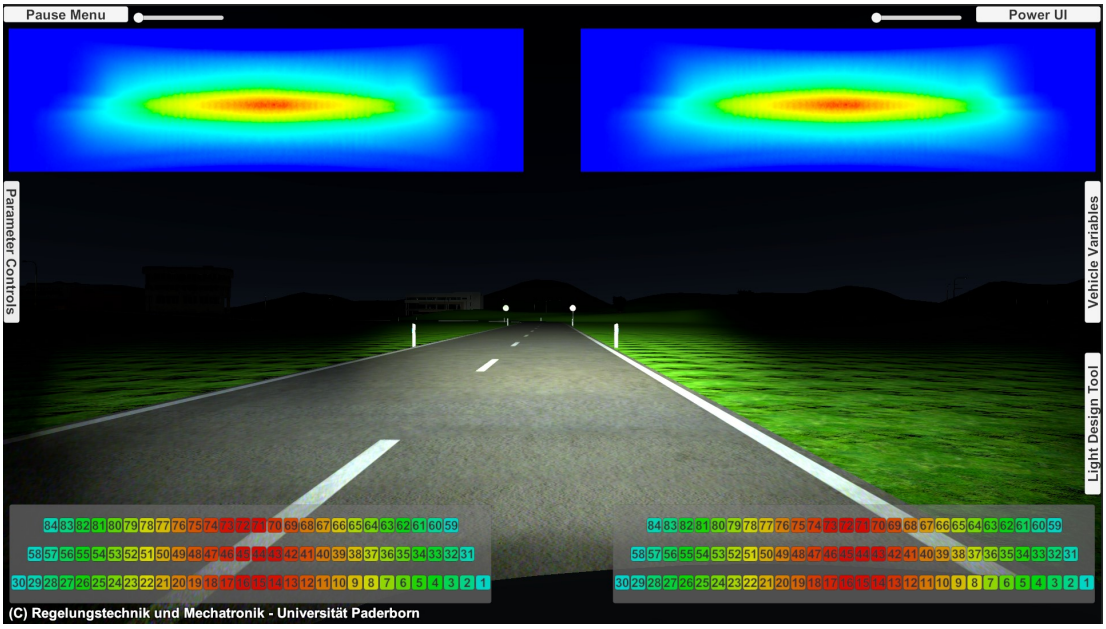


*Bild 7-2*: *Dimmwertanalyse eines niedrig aufgelösten Scheinwerfersystems (Bsp.: HD84).*

Für höher aufgelöste Systeme ist die Repräsentation der Pixellichter durch einzelne Kästchen nicht übersichtlich darstellbar. Aus diesem Grund stellt die Dimmwertanalyse die Dimmwertbelegung für höher aufgelöste Systeme durch eine Textur dar. Die Pixellichtquellen entsprechen dabei einzelnen Texeln dieser Textur, welchen abhängig vom Dimmwert eine entsprechende Farbe analog zur ersten Variante zugewiesen wird. Die Lichtstärkeverteilungen im oberen Bereich behalten unverändert ihre Bedeutung. Diese zweite Variante der Dimmwertanalyse wird in Bild 7-3 am Beispiel eines fiktiven Systems mit 1200 Lichtquellen pro Scheinwerfer gezeigt.

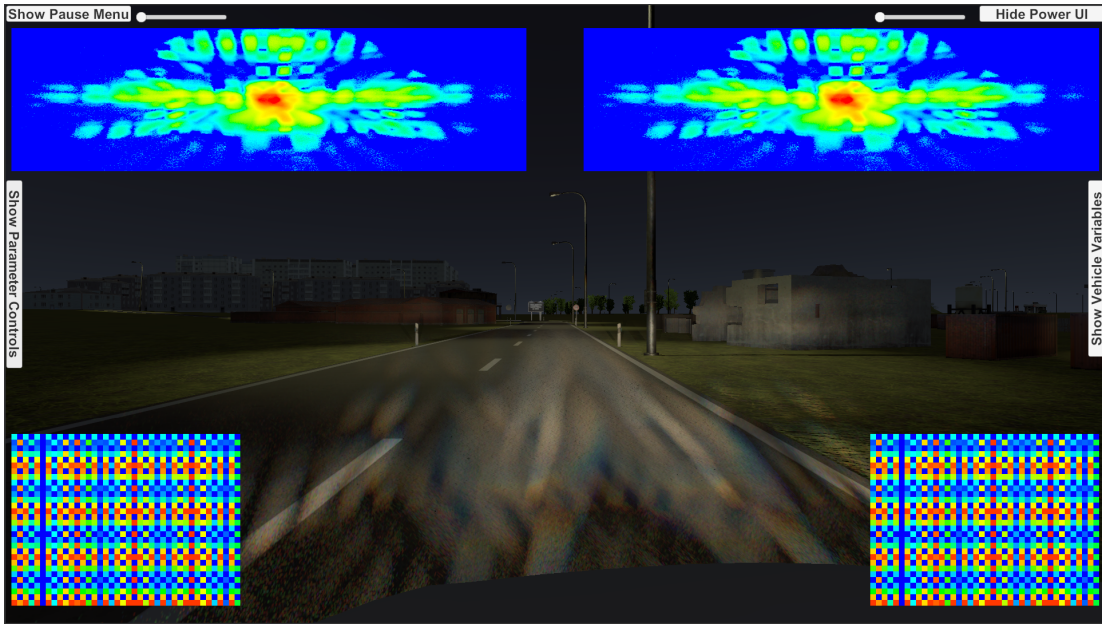


*Bild 7-3*: *Dimmwertanalyse eines fiktiven hoch aufgelösten Scheinwerfersystems.*

Sollten die Anzahlen der Pixel innerhalb einer Zeile variieren, so werden die Dimensionen der Textur als das Maximum der vorkommenden Zeilen- und Spaltenzahlen gewählt. Diejenigen Texel, die keiner Lichtquelle entsprechen, werden transparent dargestellt.

### 7.1.3 Sensorgrößen

Die Visualisierungen der photometrischen Größen und der Dimmwerte genügen nicht, um das Systemverhalten vollständig zu analysieren. Sie stellen nur die Ausgänge des Systems dar. Ebenso zentral ist die Analyse der Systemeingänge, welche letztlich zu den Ausgangsgrößen führen. Im Kontext eines Scheinwerfersystems setzen sich die Eingänge aus sensorisch erfassten Informationen über den Zustand des Fahrzeugs und dessen Umgebung zusammen (s. Abschnitt 5.4.2). Auch diese müssen auf intuitive Weise visualisiert werden.

Hyperion bietet zur Darstellung des Fahrzeugzustands ein Dashboard im rechten Bereich der Anzeige. Dieses Dashboard ist in Bild 7-4 zu sehen und kann auf Wunsch ein- oder ausgeblendet werden.


*Bild 7-4*: *Dashboard des Fahrzeugzustands und Visualisierung der Umfelderkennung zur Analyse lichtrelevanter Eingangsgrößen.*

Visualisiert werden die lichtrelevanten Größen. Das sind die Größen, die durch das Lichtsteuergerät bei der Dimmwertvorgabe berücksichtigt werden. Im dargestellten Beispiel werden die Umfangsgeschwindigkeiten der Vorderräder, die Fahrzeuggeschwindigkeit und -gierrate sowie der Lenkwinkel visualisiert. Abhängig davon, welcher Fahrmodus aktiv ist, werden diese Größen intern berechnet, über die entsprechende Schnittstelle vom ASM Fahrzeugmodell übertragen oder lediglich als aufgezeichnete Größen angezeigt.

Insbesondere für moderne Lichtfunktionen sind neben den Fahrzeuggrößen Umfeldinformationen ausschlaggebend. Wie in Abschnitt 5.4.2 am Beispiel der Umfeldkamera

beschrieben, müssen die benötigten Umfeldsensoren als virtuelle Sensoren in die Simulation integriert werden. Um den Einfluss der Umfeldinformationen auf die Lichtfunktionen bestmöglich überblicken zu können, müssen auch die wahrgenommenen Informationen der Umfeldsensoren transparent einsehbar sein. Vor diesem Hintergrund zeigt Bild 7-4 zusätzlich die Visualisierung der durch die Umfeldkamera erkannten Fahrzeuge. Sie wird über das durch die linke Schaltfläche erreichbare Parametermenü aktiviert. Die in Abschnitt 5.4.2 genannten Objektinformationen werden unmittelbar neben jedem erkannten Objekt eingeblendet.

## 7.2 Entwurf

Die in den Abschnitten 7.1.2 und 7.1.3 beschriebenen Visualisierungen unterstützen den Entwickler bei der Fehlererkennung. In klassischen Systemen genügt die Anwendung dieser Analysesichten in einigen ausgewählten Situationen, um die wenigen, darstellbaren Lichtverteilungen zu optimieren. Durch die Flexibilität von HD-Systemen ist die Menge der darstellbaren Lichtverteilungen jedoch so stark gewachsen, dass konventionelle Entwicklungsmethoden die Potentiale moderner Systeme nicht voll ausschöpfen können. Stattdessen muss der Lichtingenieur durch softwaretechnische Automatismen aktiv beim Entwurf von Lichtverteilungen unterstützt werden. Dieser Abschnitt umfasst die Methoden, die zur Unterstützung und Automatisierung des Auslegungsprozesses von Lichtfunktionen entwickelt wurden und adressiert damit die Anforderung **A8**. Das konkrete Scheinwerfersystem wird dabei abstrahiert, sodass die Auslegung der Lichtfunktion universell vorgenommen werden kann. Die Anpassung der generierten Lichtfunktionen auf verschiedene Systeme ist anschließend automatisiert möglich.

### 7.2.1 Methodik

Die nachfolgend vorgestellte Methodik dient nicht nur der Handhabbarkeit der vielen Freiheitsgrade beim Entwurf von Lichtfunktionen für HD-Systeme, sondern automatisiert den Entwurfsprozess über weite Bereiche. Außerdem erlaubt sie die direkte Vorgabe photometrischer Sollwerte und führt auf diese Weise schneller zur gesuchten Lichtfunktion, als es in iterativen Annäherungsverfahren möglich ist. Die grundlegende Wirkkette ist in Bild 7-5 dargestellt.

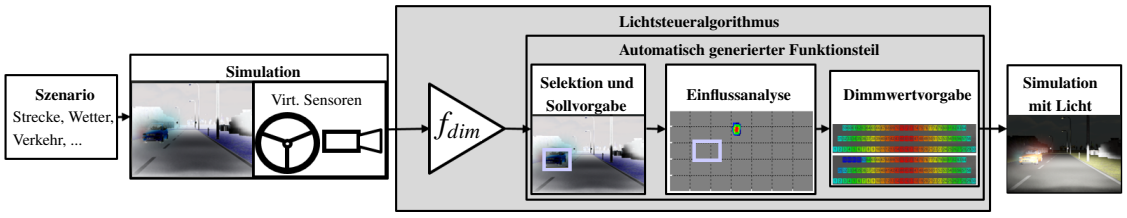


*Bild 7-5*: *Methodik für den Entwurf von Lichtfunktionen für HD-Scheinwerfersysteme.*

Der Entwurfsprozess beginnt mit der Definition des Testszenarios und der anschließenden Simulation dessen in Hyperion. Innerhalb der Nachtfahrtsimulation werden alle lichtrelevanten Fahrzeugzustände (z.B. Lenkwinkel, Gierrate, Geschwindigkeit, ...) und Umfeld-

sensoren (z.B. Umfeldkamera, ...) simuliert (s. Abschnitt 5.4.2). Die Informationen der Fahrzeugsensorik bestimmen das Verhalten der verschiedenen Lichtfunktionen. Betrachtet man beispielsweise das blendfreie Fernlicht, so werden die Dimmwerte der Pixellichter primär durch die Umfeldkamera bestimmt. $f_{dim}$ gibt für die einzelnen Lichtquellen im Regelfall die Dimmwerte für volles Fernlicht vor. Sobald jedoch durch die Umfeldsensorik ein entgegenkommendes Fahrzeug erkannt wird, müssen die Dimmwerte der auf das Fahrzeug Einfluss nehmenden Lichtquellen reduziert werden. Eine Lichtfunktion kann insofern als Mapping-Funktion $f_{dim}$ verstanden werden (s. Bild 7-5). Sie verweist im Allgemeinen von der Gesamtheit der momentan vorliegenden Sensordaten auf die Dimmwerte des HD-Scheinwerfers. Aufgrund der hochdimensionalen Ausgangsgröße von $f_{dim}$, welche abhängig vom Scheinwerfersystem zwischen einigen hundert bis mehreren zehntausend Dimmwerten variieren kann, kann diese Mapping-Funktion auf direktem Wege nicht adäquat formuliert werden. Die in Bild 7-5 veranschaulichte Methodik reduziert die Dimension der Mapping-Funktion $f_{dim}$ durch das Einfügen einer Abstraktionsschicht vor dem Scheinwerfersystem erheblich. Diese Abstraktionsschicht, bestehend aus Einflussanalyse und Dimmwertvorgabe, ist in der Lage räumliche Selektionen im Fahrzeugumfeld und darin einzuhaltende photometrische Sollwerte in Dimmwerte der Pixellichter zu transformieren. Die Implementierung dieser Abstraktionsschicht, deren essentielle Komponente die Einflussanalyse einzelner Lichtquellen auf den Selektionsbereich darstellt, wird im nachfolgenden Abschnitt beschrieben.

Den methodischen Kern stellt die drastische Verringerung der Dimension der Mapping-Funktion $f_{dim}$ dar. Aufgrund der Abstraktionsschicht beschränkt sich die Aufgabe des Lichtingenieurs auf die Definition einer Zuordnung von Sensorinformationen auf Selektionsbereiche und photometrische Sollwerte. In Bild 7-5 wird diese Zuordnung durch den mit $f_{dim}$ beschrifteten Block abgebildet. Abhängig von der zu entwerfenden Lichtfunktion kann $f_{dim}$ ganz unterschiedlicher Gestalt sein und wird in dieser prototypischen Umsetzung nicht näher diskutiert. Ist $f_{dim}$ gegeben, erfolgen alle nachfolgenden Schritte, welche nötig sind, um die Dimmwerte der Pixellichter zu ermitteln, vollautomatisiert in der Abstraktionsschicht.

### 7.2.2 Implementierung

Nach der Darstellung methodischer Grundlagen erfolgt nun die Diskussion der wesentlichen Aspekte der Implementierung. Zuerst wird das Preprocessing des Scheinwerfersystems beschrieben. Im Rahmen dieses Vorgangs werden die essentiellen Eigenschaften aus den Lichtverteilungen der einzelnen Pixellichter extrahiert. Insbesondere wird dabei die konkrete Scheinwerferrealisierung abstrahiert, sodass die nachfolgenden Schritte davon unabhängig implementiert werden können. Im Anschluss wird die Einflussanalyse der einzelnen Pixellichter auf den räumlichen Selektionsbereich vorgestellt. Die hierzu verwendete Implementierung ist echtzeitfähig und eignet sich auch zur Integration auf einem Steuergerät. Abschließend werden Designparameter aufgeführt, welche vom Lichtingenieur spezifiziert werden können und sowohl einen Kompromiss zwischen Robustheit und Präzision ermöglichen als auch zur Individualisierung der Lichtfunktion beitragen.

### Preprocessing

Ziel des Preprocessings ist ähnlich wie in Kapitel 6 die Extraktion der wesentlichen Informationen, um so zur Laufzeit schnell agieren zu können. Zu Beginn liegen die Lichtverteilungen aller Pixellichter vor. Im monochromen Fall handelt es sich hierbei um Lichtstärkewerte und im spektralen Fall um CIE XYZ Koordinaten, welche die Charakteristik des Pixellichts abhängig von Polar- und Azimutwinkel wiedergeben. Für die nachfolgenden Ausführungen wird ausschließlich die Lichtstärke verwendet, welche im spektralen Fall im vierten Farbkanal der RGBY-Textur enthalten ist. Beispielhaft zeigt das Bild 7-6 die Lichtstärkeverteilung der LED 1 (untere Zeile, links) eines HD84-Scheinwerfers. In Bild 7-6 sind die typischen Merkmale eines Pixellichts erkennbar. Die betrachtete Lichtquelle bestrahlt nur einen kleinen Ausschnitt des insgesamt auszuleuchtenden Raumwinkels.

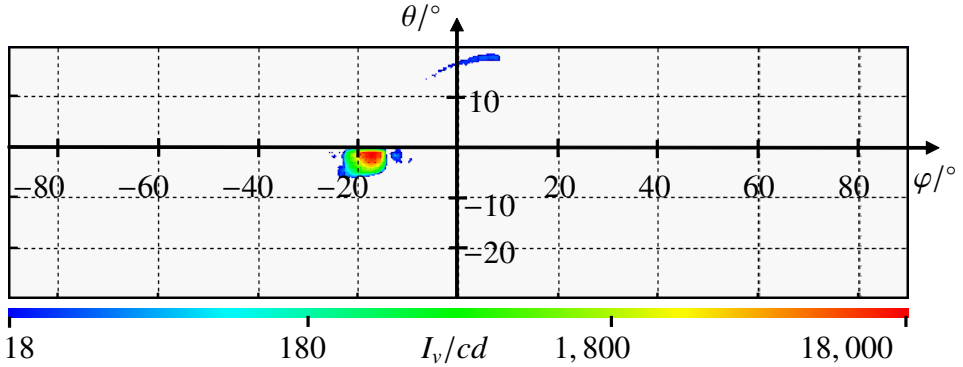


*Bild 7-6*: *Lichtstärkeverteilung der LED 1 der HD84-Matrix.*

Um die nachfolgend benötigten Merkmale eines Pixellichts zu extrahieren, wird im ersten Schritt zu jedem Pixel (Index $k$) der bestrahlte Raumwinkelbereich $R_{l,k} \subset [-\frac{\pi}{2}, \frac{\pi}{2}]^2$ bestimmt. Ziel ist die Identifikation der Konturen der Lichtkeule (s. Bereich bei ca. $\theta = -2°, \varphi = -20°$ in Bild 7-6). Einerseits sollte die Lichtkeule in $R_{l,k}$ vollständig enthalten sein, andererseits sollten das umgebende Streulicht und Abbildungsfehler nicht zu einem übergroßen Raumwinkelbereich führen. Die Tiefpass-Filterung der ursprünglichen Lichtverteilung schafft hier Abhilfe, da Streulichtanteile aufgrund ihrer hohen örtlichen Variation stark abgeschwächt werden. So kann beispielsweise der bei $\theta \approx 17°$ und $\varphi \approx 0°$ befindliche Streulichtschweif (s. Bild 7-6) durch die Tiefpassfilterung erheblich abgeschwächt werden, da er nur sehr konzentriert auftritt und seine direkte Umgebung kein Licht empfängt. In Hyperion wird dazu auf jeden $7 \times 7$-Block benachbarter Texel der Lichtverteilung ein Gauß-Kernel der Größe $7 \times 7$ mit einer Standardabweichung von $\sigma = 3$ angewendet [Jäh05]. Die hier getroffene Wahl wurde auf die vorliegenden Messdaten optimiert und hängt von der Auflösung der Lichtverteilung und den optischen Eigenschaften des Scheinwerfersystems, insbesondere der Randschärfe der Ausleuchtungsbereiche und der Charakteristik des Streulichts (, ab. In Bild 7-7a wird die Lichtverteilung der LED 1 gezeigt. Auch wenn die dargestellte Lichtverteilung spektral ist, wird das Preprocessing ausschließlich auf Basis der Lichtstärke vollzogen, welche im nicht visualisierten $\alpha$-Kanal neben den drei Farbkanälen (RGB) ausgelesen werden kann. Das Bild 7-7b zeigt die Gradientendarstellung des Bilds 7-7a. Diese kann durch die Anwendung des Sobel-Operators erzeugt werden, welcher typischerweise zur Kantenerkennung eingesetzt wird [Jäh05]. In der Gradientendarstellung wird das Streulicht deutlicher erkennbar (s. linker Randbereich in Bild 7-7b). Das Ergebnis

der Tiefpassfilterung wird in Bild 7-7c ebenfalls in der Gradientendarstellung gezeigt. Der Vergleich der Bilder 7-7b und 7-7c zeigt, dass das Streulicht durch die Tiefpass-Filterung deutlich reduziert werden kann ohne die Konturen der Lichtkeule signifikant zu beeinflussen.

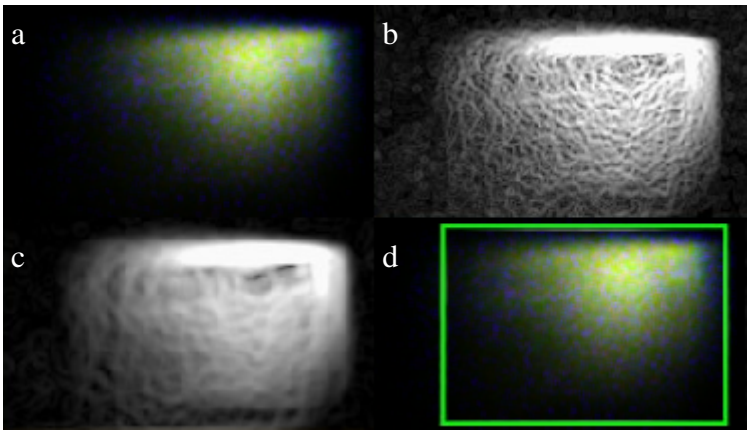


*Bild 7-7*: *Spektrale Lichtverteilung der LED 1 des HD84-Systems (a), deren Gradientendarstellung (b),deren Gradientendarstellung nach der Tiefpass-Filterung (c) und ihr umhüllendes Rechteck in sphärischen Koordinaten (d).*

Im nächsten Schritt wird der maximal auftretende Lichtstärkewert $\bar{I}_{v,k}$ innerhalb der betrachteten Lichtverteilung der LED $k$ bestimmt. Bezogen auf $\bar{I}_{v,k}$ wird eine relative Schranke $b_{cut} \in [0,1]$ definiert. Lichtstärkewerte, welche unterhalb $b_{cut} \cdot \bar{I}_{v,k}$ liegen, werden als irrelevant betrachtet. Die vorgelagerte Tiefpass-Filterung stellt sicher, dass auch für kleine Werte von $b_{cut}$ kein Streulicht als relevant eingestuft wird. Die Wahl der Schranke definiert vorrangig, wie scharf die Lichtkeule ausgeschnitten wird. Sie muss spezifisch für das jeweilige Scheinwerfersystem gewählt werden. Der Abschnitt 7.2.3 wird zeigen, dass die Wahl $b_{cut} = 50\%$ im Falle des HD84-Systems zu einer guten Segmentierung des gesamten Ausleuchtungsbereichs führt (s. Bild 7-10).

Um in den nachfolgenden Schritten effizient mit den identifizierten Winkelbereichen $R_{l,k}$ umgehen zu können, werden sie durch Rechteckgeometrien bezüglich der sphärischen Koordinaten beschrieben. Somit lassen sie sich sehr kompakt durch ihren unteren linken Eckpunkt $(\underline{\theta}_{l,k}, \underline{\varphi}_{l,k})$ und ihren oberen rechten Eckpunkt $(\bar{\theta}_{l,k}, \bar{\varphi}_{l,k})$ beschreiben. Zur Konstruktion des einhüllenden Rechtecks $R_{l,k}$ für eine gegebene Lichtverteilung, wird dieses mit einer Größe von einem Texel an dem zuvor ermittelten Maximalwert $\bar{I}_{v,k}$ initialisiert. Dementsprechend gilt initial

$$(\underline{\theta}_{l,k}, \underline{\varphi}_{l,k}) = (\bar{\theta}_{l,k}, \bar{\varphi}_{l,k}) = (\theta_{max}, \varphi_{max}),$$

wobei $(\theta_{max}, \varphi_{max})$ die Texelkoordinaten von $\bar{I}_{v,k}$ sind. Nun wird über alle Texel der Lichtverteilung iteriert. Für jedes Texel, dessen Lichtstärkewert $I_{v,\theta^\star,\varphi^\star}$ oberhalb der Schranke $(I_{v,\varphi^\star,\theta^\star} > b_{cut} \cdot \bar{I}_{v,k})$ liegt, wird geprüft, ob es im bisherigen Rechteck $R_{l,k}$ enthalten ist. Sollte das Texel außerhalb von $R_{l,k}$ liegen, werden die Eckpunkte von $R_{l,k}$ so angepasst, dass das neue Rechteck $R^\star_{l,k}$ der kleinsten Rechteckgeometrie entspricht, für die gilt

$$R_{l,k} \subset R^\star_{l,k} \wedge (\varphi^\star, \theta^\star) \in R^\star_{l,k}.$$

Das Bild 7-7d zeigt das grün dargestellte Rechteck $R_{l,1}$, welches nach Iteration über alle Texel der Lichtverteilung der LED 1 entsteht. Wie aus dem Bild hervorgeht, kann der vom Pixellicht bestrahlte Raumwinkel durch dieses Verfahren extrahiert werden.

Neben den bestrahlten Winkelbereichen stellt der Lichtstrom $\Phi_{v,k}$, welcher vom Pixellicht $k$ bei voller Bestromung innerhalb des Bereichs $R_{l,k}$ abgegeben wird, ein weiteres wichtiges Merkmal der Lichtverteilung dar. Durch Integration der Lichtstärke über dem Raumwinkelbereich $R_{l,k}$ kann $\Phi_{v,k}$ bestimmt werden:

$$\Phi_{v,k} = \int_{\varphi=\underline{\varphi}}^{\bar{\varphi}} \int_{\theta=\underline{\theta}}^{\bar{\theta}} I_{v,k}(\theta,\varphi) \cos\theta d\theta d\varphi. \tag{7-1}$$

Da keine kontinuierliche Form der Lichtstärkeverteilung $I_{v,k}(\theta,\varphi)$ besteht, muss das Integral in Gleichung (7-1) durch die Riemann-Summe der diskreten Texel approximiert werden:

$$\begin{aligned} \Phi_{v,k} &= \sum_{m=1}^{M_k} \sum_{n=1}^{N_k} I_{v,k}(\theta_m,\varphi_n) \cos\theta_n \Delta\varphi\Delta\theta \\ \theta_m &= \underline{\theta} + m \cdot \Delta\theta, \varphi_n = \underline{\varphi} + n \cdot \Delta\varphi. \end{aligned} \tag{7-2}$$

Wenn $\Phi_k$ über einen großen Winkelbereich $R_{l,k}$ verteilt ist, so ergibt sich eine niedrige mittlere Lichtstärke $I_{v,k,\varnothing}$. In der Konsequenz ist auch die am bestrahlten Objekt vorliegende Beleuchtungsstärke niedrig. Zur Berücksichtigung dieses Effekts wird die Größe des bestrahlten Raumwinkelbereichs ebenfalls bestimmt:

$$\begin{aligned} \Omega_k &= \int_{\theta=\underline{\theta}}^{\bar{\theta}} \int_{\varphi=\underline{\varphi}}^{\bar{\varphi}} \cos\theta d\varphi d\theta \\ &= (\bar{\varphi} - \underline{\varphi}) \cdot (\sin\bar{\theta} - \sin\underline{\theta}). \end{aligned} \tag{7-3}$$

Aufbauend auf den Gleichungen (7-2) und (7-3) kann die mittlere Lichtstärke im identifizierten Bestrahlungsbereich $R_{l,k}$ zu

$$I_{v,k,\varnothing} = \frac{\Phi_{v,k}}{\Omega_k} \tag{7-4}$$

bestimmt werden. Das Preprocessing ist damit abgeschlossen.

### Einflussanalyse

Die im Preprocessing extrahierten Merkmale $R_{l,k}$, $\Phi_{v,k}$, $\Omega_k$ und $I_{v,k,\varnothing}$ jeder Lichtquelle werden für die Transformation der räumlichen Selektion und der Sollwertvorgabe in Dimmwerte der einzelnen Pixellichter benötigt. Zuvor muss die räumliche Selektion jedoch in die sphärischen Koordinaten der Scheinwerfer überführt werden. Details zu dieser Transformation können [RBM+19a] entnommen werden. Es wird im weiteren Verlauf angenommen, dass der Selektionsbereich analog zu den Lichtkeulen der Pixellichter als Rechteck $R_{sel}$ bezüglich der sphärischen Koordinaten des jeweiligen Scheinwerfers interpretiert werden kann. Neben der Selektion muss der Sollwert $E_{v,ref}$ der Beleuchtungsstärke spezifiziert werden. Die Wahl dieses Sollwerts ist abhängig von der betrachteten Lichtfunktion und den vorliegenden Sensorinformationen. Im Falle des blendfreien Fernlichts ist zum Beispiel

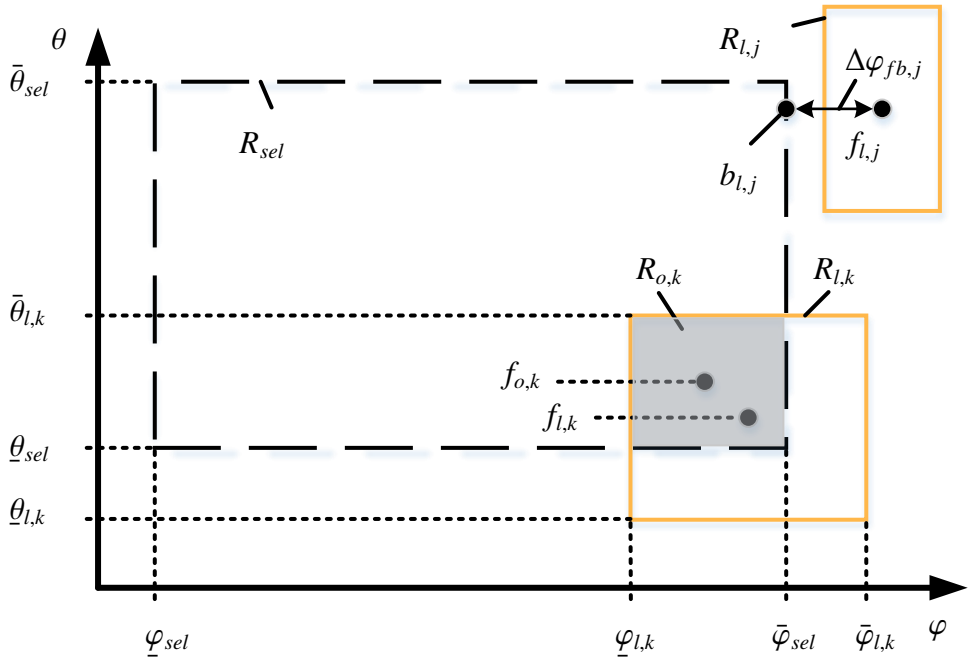


*Bild 7-8: Überlappungsbereich zwischen der Selektion $R_{sel}$ und der Lichtkeule $R_{l,k}$.*

die Vorgabe $E_{v,ref} = 0\,lx$ im Bereich der Windschutzscheibe des entgegenkommenden Fahrzeugs sinnvoll.

Im nächsten Schritt muss die Einflussnahme jedes Pixellichts auf $R_{sel}$ analysiert werden. Betrachtet man die Lichtquelle $k$, wird zuerst die Überlappung $R_{o,k}$ zwischen dem Ausleuchtungsbereich $R_{l,k}$ der Lichtquelle und der Selektion $R_{sel}$ bestimmt. Es werden zwei Typen von Überlappungssituationen unterschieden.

Im ersten Fall nimmt der Überlappungsbereich $R_{o,k}$ mindestens $o_{cut} = 50\%$ des Ausleuchtungsbereichs $R_{l,k}$ ein. Dieser Fall trifft auf die Lichtquelle $k$ in Bild 7-8 zu. Zur Erreichung der Sollwertvorgabe $E_{v,ref}$ im Bereich $R_{sel}$ muss für die Lichtquelle $k$ ein geeigneter Dimmwert $d_k \in [0, 1]$ gefunden werden. Da die Beleuchtungsstärke nicht direkt beeinflusst werden kann, muss die Sollvorgabe zunächst in eine Referenzlichtstärke $I_{v,ref}$ überführt werden. Gemäß Gleichung (2-13) stehen diese Größen über die Distanz $r$ und den Lichteinfallswinkel $\theta_E$ in Zusammenhang. Während die Distanz zu Umgebungsobjekten durch moderne Fahrzeugsensorik erfasst werden kann, kann die genaue Kenntnis von $\theta_E$ nicht zwangsläufig vorausgesetzt werden. Ist $\theta_E$ unbekannt, empfiehlt es sich, den Wert von $\theta_E$ abhängig von der Lichtfunktion festzulegen (z.B. blendfreies Fernlicht $\theta_E = 0°$). Die Lichtverteilung eines einzelnen Pixellichts kann durch den Dimmwert nur global skaliert werden. Deshalb wird die Distanz $r_{o,k}$ zwischen der Lichtquelle $k$ und der beleuchteten Fläche in Richtung des Fokuspunkts $f_{o,k}$ der Überlappung als Approximation der mittleren Entfernung verwendet. Da der außerhalb $R_{sel}$ liegende Anteil der Lichtkeule $R_{l,k}$ keinen Beitrag zur Beleuchtungsstärke im relevanten Bereich liefert (s. Bild 7-8), wird die Richtung $f_{o,k}$ anstelle des Fokuspunkts $f_{l,k}$ der Lichtquelle präferiert. Falls $R_{sel}$ die Lichtkeule

$R_{l,k}$ vollständig beinhaltet, sind $f_{o,k}$ und $f_{l,k}$ identisch. Basierend auf diesen Überlegungen ergibt sich die Referenzlichtstärke zu

$$I_{v,k,ref} = \frac{E_{v,ref} \cdot r_{o,k}^2}{\cos \theta_E} \tag{7-5}$$

und der gesuchte Dimmwert $d_k$ der Lichtquelle $k$ entsprechend zu

$$d_k = \frac{I_{v,k,ref}}{I_{v,k,\varnothing}}. \tag{7-6}$$

Im anderen Fall hat der Überlappungsbereich $R_{o,j}$ maximal einen Anteil von $o_{cut}$ ($< 50\%$) am Ausleuchtungsbereich $R_{l,j}$, wie es Bild 7-8 am Beispiel der Lichtquelle $j$ verdeutlicht wird. Würden Lichtquellen auf die bereits beschriebene Weise behandelt werden, so würden die unerwünschten Veränderungen außerhalb der Selektion den gewünschten Veränderungen im Selektionsbereich überwiegen. Dennoch sollten Lichtquellen im Randbereich der Selektion berücksichtigt werden. Einerseits kann das Dimmen im Randbereich Sensorungenauigkeiten kompensieren. Andererseits ist die Schärfe der Hell-Dunkel-Grenze, welche durch das Dimmverhalten gerade dieser Lichtquellen parametriert werden kann, eine Individualisierungsmöglichkeit für die verschiedenen Fahrzeughersteller.

Als Maß der Entfernung zwischen einer Lichtkeule $R_{l,j}$ und der Selektion $R_{sel}$ wird die horizontale ($\Delta\varphi_{fb,j}$) und vertikale ($\Delta\theta_{fb,j} = 0°$ in Bild 7-8) Winkeldifferenz zwischen dem Fokuspunkt $f_{l,j}$ und dem nächstgelegenen Grenzpunkt $b_{l,j}$ der Selektion verwendet. Aus den Differenzen $\Delta\varphi_{fb,j}$ und $\Delta\theta_{fb,j}$ wird ein Interpolationsfaktor $c_j \in [0, 1]$ berechnet. Der einzustellende Dimmwert $d_j$ ergibt sich schließlich aus der Interpolation des Dimmwerts $d_j^\star$ nach Gleichung (7-6) und einem Standard-Dimmwert $d_{std,j}$ (z.B. Fernlicht ohne Ausblendungen):

$$d_j = (1 - c_j) \cdot d_{std,j} + c_j \cdot d_j^\star. \tag{7-7}$$

Zur Berechnung von $d_j^\star$ wird die Entfernungsmessung zur bestrahlten Fläche in Richtung des Randpunkts $b_{l,j}$ vorgenommen, da $b_{l,j}$ dem Fokuspunkt $f_{l,j}$ am nächsten gelegen ist. Der Interpolationsfaktor $c_j$ in Gleichung (7-7) resultiert aus $\Delta\varphi_{fb,j}$ und $\Delta\theta_{fb,j}$ sowie den frei wählbaren Maximalabständen $\Delta\bar{\varphi}_{fb}$ und $\Delta\bar{\theta}_{fb}$, unter denen noch eine Beeinflussung der Lichtquellen durch die Selektion gewünscht wird:

$$c_j = \max(1 - \frac{\Delta\varphi_{fb,j}}{\Delta\bar{\varphi}_{fb}}, 1 - \frac{\Delta\theta_{fb,j}}{\Delta\bar{\theta}_{fb}}, 0). \tag{7-8}$$

### Designparameter

In den vorhergehenden Abschnitten wird die Dimmwertermittlung als striktes Verfahren beschrieben. Zur Sicherstellung der Robustheit gegenüber Sensorfehlern und zur Eröffnung von Individualisierungsmöglichkeiten sollte dem Lichtingenieur an geeigneten Stellen Spielraum eingeräumt werden. Diese Freiheitsgrade werden in der vorgestellten Implementierung durch wählbare Designparameter bereitgestellt.

Die Designparameter $\Delta\bar{\varphi}$ und $\Delta\bar{\theta}$, welche die Kantenschärfe der Dimmbereiche spezifizieren, werden bereits in Gleichung (7-8) angeführt. Große Werte für $\Delta\bar{\varphi}$ und $\Delta\bar{\theta}$ führen

zu weichen Übergängen vom Dimmbereich in die übrige Lichtverteilung. Wählt man $\Delta\bar{\varphi} = \Delta\bar{\theta} = 0°$, können Situationen auftreten, in welchen ein Pixellicht vollständig ausgeschaltet ist, während ein benachbartes mit voller Bestromung versorgt wird. Hieraus resultieren scharfe Übergänge zwischen Dimmbereich und restlicher Lichtverteilung. Insbesondere ist es sinnvoll, die horizontale und vertikale Kantenschärfe unabhängig voneinander parametrieren zu können, da die Rasterung der Gesamtlichtverteilung entlang der horizontalen Winkel häufig feiner ist. Im Kontext von Sensorungenauigkeiten kann durch die Parameter $\Delta\bar{\varphi}_{fb}$ und $\Delta\bar{\theta}_{fb}$ ein Sicherheitsbereich beschrieben werden, welcher die eigentliche Selektion umrahmt.

Neben örtlicher Tiefpass-Filterung, erlaubt die hier vorgestellte Lösung auch zeitliche Tiefpass-Filterung. Hierzu werden PT1-Filter eingesetzt. Da die Dimmwerte der Pixellichter zu diskreten Zeitpunkten bestimmt werden, werden die Dimmfilter in ihrer diskreten Form

$$d_{k+1} = T^\star \cdot (d_{ref,k} - d_k) + d_k \text{ mit } T^\star = \frac{1}{\frac{T_{dim}}{\Delta t} + 1} \tag{7-9}$$

verwendet. In Gleichung (7-9) ist $d_{ref,k}$ der Referenzdimmwert nach Gleichung (7-6) bzw. (7-7), $d_k$ der Dimmwert des vorhergehenden Zeitschritts und $\Delta t$ die zeitliche Diskretisierungsschrittweite. Der Lichtingenieur kann die Dimmzeitkonstanten $T_{up}$ (falls $d_k < d_{ref,k}$) und $T_{down}$ (falls $d_k > d_{ref,k}$), welche die Zeitkonstante $T_{dim}$ in Gleichung (7-9) ersetzen, unabhängig voneinander wählen.

Auch die zeitliche Tiefpass-Filterung schafft Raum für Individualisierung. Beispielsweise könnte ein OEM die Aufmerksamkeit des Kunden durch die technischen Möglichkeiten des Scheinwerfersystems erregen wollen und deshalb schnelle und scharfe Veränderungen in der Lichtverteilung bevorzugen. Ein anderer OEM verfolgt hingegen ein möglichst unauffälliges Scheinwerfersystem und möchte ein hektisches Gefühl bei der Nachtfahrt in jedem Fall vermeiden. In diesem Fall empfiehlt es sich, größere Zeitkonstanten zu wählen. Zwischen der Ab- und Aufdimmzeitkonstanten ($T_{down}$ und $T_{up}$) wird neben der Entwurfsfreiheit auch aus Sicherheitsgründen unterschieden. Die Abdimmgeschwindigkeit muss hoch genug sein, um z.B. entgegenkommenden Verkehr nicht aufgrund einer zu hohen Latenz zu blenden. Sie besitzt somit eine höhere Sicherheitsrelevanz als die Aufdimmgeschwindigkeit.

### 7.2.3 Validierung

Zur Validierung des beschriebenen Entwurfsmethodik wird nun überprüft, in welcher Qualität die Sollvorgabe umgesetzt werden kann. Neben der Korrektheit der Implementierung wird dabei auch der Einfluss des Scheinwerfersystems mit seinen Möglichkeiten und Grenzen diskutiert.

#### Szenario

In der hier vorgestellten Lösung gibt der Lichtingenieur den Sollzustand in Form der Beleuchtungsstärke $E_{v,ref}$ in einem Selektionsbereich $R_{sel}$ vor. Dementsprechend sollte

die Beleuchtungsstärke im Rahmen der Validierung als Gütemaß fungieren. Zu diesem Zweck wird die durch das Scheinwerfersystem realisierte Beleuchtungsstärke $E_{v,\varnothing}$ im Selektionsbereich gemittelt und mit der Sollvorgabe $E_{v,ref}$ verglichen. Außerdem wird eine Falschfarbendarstellung über Polar- und Azimutwinkel herangezogen, um lokale Über- und Unterschreitungen visualisieren zu können.

Um die Ergebnisse der Entwurfsmethodik möglichst isoliert von anderen Störeinflüssen, wie Orientierungs- und Entfernungsunterschiede zu den bestrahlten Flächen, bewerten zu können, wird eine stark reduzierte Szene zur Validierung erstellt. Die in Bild 7-9 gezeigte Szene enthält einen linken HD84-Scheinwerfer als einzige Lichtquelle. Dieser Scheinwerfer befindet sich im Zentrum einer Hohlkugel mit einem Radius von $r_s = 10\,m$ und bestrahlt deren Innenwand. Demzufolge haben alle bestrahlten Flächenelemente die gleiche Distanz und die gleiche Orientierung zur Lichtquelle. Somit kann die Sollvorgabe $E_{v,ref}$ unabhängig vom betrachteten Szenenpunkt in die Referenzlichtstärke $I_{v,k,ref}$ aus Gleichung (7-5) überführt werden.

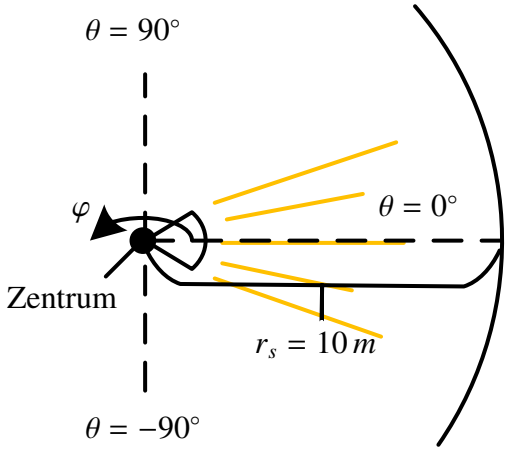


*Bild 7-9*: *Szene zur Validierung der implementierten Entwurfsmethode.*

Der Selektionsbereich sollte einerseits im Arbeitsbereich des Scheinwerfersystems enthalten sein und andererseits die Analyse von störenden Effekten im Grenzbereich der Ausleuchtung erlauben. Er wird deshalb zu $R_{sel}$ mit $\varphi \in [-10.5°, -3.0°]$ und $\theta \in [-4.1°, +3.6°]$ gewählt. Innerhalb von $R_{sel}$ soll eine Beleuchtungsstärke $E_{v,ref}$ von $100\,lx$ eingestellt werden. Um die Resultate der in Abschnitt 7.2.2 beschriebenen Implementierung möglichst transparent zu zeigen, wird keine lokale Filterung angewendet ($\Delta\bar{\varphi} = \Delta\bar{\theta} = 0°$). Aus demselben Grund werden die Standarddimmwerte $d_{std,j}$ nach Gleichung (7-7) zu 0% gewählt. Die Einstellungen der zeitlichen Filter sind aufgrund der statischen Szene im Rahmen der Validierung nicht relevant. Zur Erprobung der Implementierung wird der Datensatz des HD84-Systems genutzt. Aufgrund der überschaubaren Anzahl von Lichtquellen lassen sich die Berechnungsergebnisse für einzelne Pixellichter anschaulich visualisieren.

### Ergebnisse

Auf das HD84-System wird zunächst das in Abschnitt 7.2.2 beschriebene Preprocessing angewendet. Im ersten Schritt werden dabei die Einflussbereiche $R_{l,k}$ sämtlicher Pixellichter

ermittelt. Diese Bereiche weisen bezüglich des Polar- und Azimutwinkels eine rechteckige Gestalt auf. Der Relativwert $b_{cut}$, welcher die Trennung zwischen Nutz- und Streulicht maßgeblich bestimmt, ist dabei so zu wählen, dass weder Lücken noch Überlappungen zwischen den einzelnen Einflussbereichen dominieren. Für das betrachtete HD84-System führt eine Wahl von $b_{cut} = 50\%$ auf die in Bild 7-10 visualisierte Segmentierung.

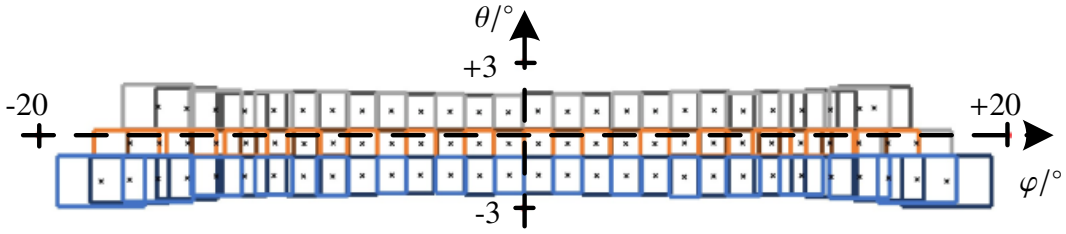


*Bild 7-10*: *Segmentierung des Ausleuchtungsbereichs eines HD84-Systems in die Einflussbereiche einzelner Pixellichtquellen.*

Wie sich zeigt, ergibt sich im Zentralbereich $-10° < \varphi < +10°$ eine sehr gute Segmentierung, während in den Randbereichen größere Überlappungen zu verzeichnen sind. Diese sind jedoch der optischen Auslegung des Systems geschuldet. Eine Minimierung der Überlappungen durch eine andere Wahl von $b_{cut}$ würde mit einer Verschlechterung im Zentralbereich einhergehen. Die Gestaltung des Systems in drei Zeilen kann in der Segmentierung ebenfalls sehr gut wiedererkannt werden. Zur besseren Erkennbarkeit werden die obere Zeile in grau, die mittlere Zeile in orange und die untere Zeile in blau dargestellt. Innerhalb der Zeilen werden zur Unterscheidung der einzelnen Lichtquellen gerade Indizes in einem dunkleren und ungerade Indizes in einem helleren Farbton visualisiert.

Nach Gleichung (7-3) können die Raumwinkel $\Omega_k$ der zuvor identifizierten Einflussbereiche ermittelt werden. Das Bild 7-11 zeigt sämtliche Werte der drei Zeilen $i = 1 \ldots 30$ (unten), $i = 31 \ldots 58$ Mitte) und $i = 59 \ldots 84$ (oben) des HD84-Moduls. Die Ergebnisse entsprechen den Erwartungen. Da $\theta$ aus Gleichung (7-3) über die Zeilen hinweg nur geringen Veränderungen unterliegt, sind die Raumwinkel $\Omega_k$ nahezu proportional zu den Flächeninhalten der Rechtecke aus Bild 7-10. Darüber hinaus zeigt Bild 7-10, dass die Lichtquellen der mittleren Zeile kleinere Einflussbereiche als die äußeren Zeilen vorweisen. Diese Eigenschaft wird bei der optischen Auslegung beabsichtigt, da im Zentralbereich die höchsten Flexibilitätsanforderungen gelten.

Neben den Raumwinkeln wird im Preprocessing der Lichtstrom $\Phi_{v,k}$ jeder Lichtquelle gemäß Gleichung (7-2) ermittelt. Die Ergebnisse dieser Berechnungen werden in Bild 7-12 zusammengefasst. Da es sich bei allen Lichtquellen um LED des gleichen Typs handelt, wird über alle Pixellichter hinweg ein gleichbleibender Lichtstrom mit zufälligen leichten Schwankungen erwartet. Tatsächlich stellt man jedoch fest, dass die mittlere Zeile im Zentralbereich systematisch höhere Werte erreicht. Erklärt werden kann diese Anomalie durch die Fokussierung des Lichts auf einen kleineren Raumwinkel, welche zu weniger abgeschnittenem Streulicht führt. Die Anstiege des Lichtstroms im Randbereich der unteren und oberen Zeilen resultieren aus den vergrößerten Integrationsbereichen, wie

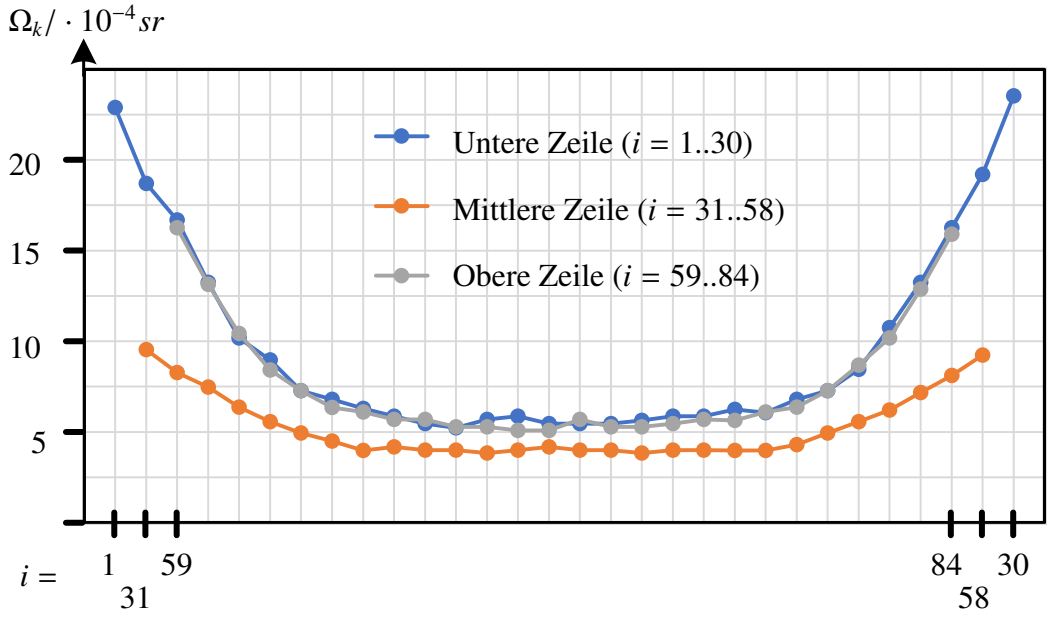


*Bild 7-11*: *Raumwinkel der Pixellichter eines HD84-Systems.*

sie bereits in Bild 7-11 beobachtet werden konnten. Der signifikant höhere Lichtstrom der LED $i = 84$ kann nicht erklärt werden. Er liegt bereits in den Vermessungsdaten vor.

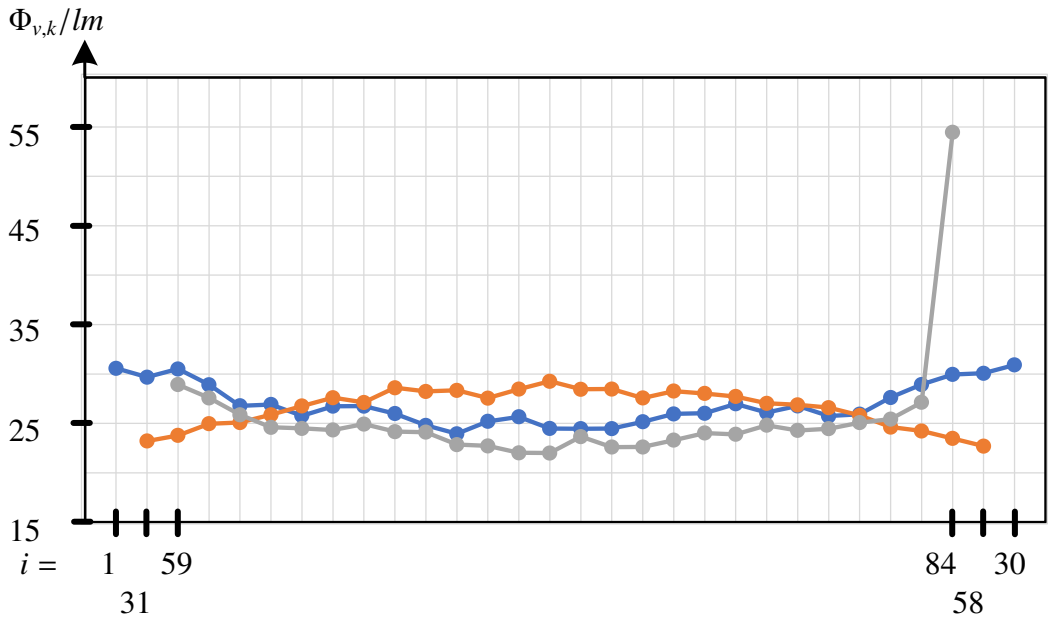


*Bild 7-12*: *Lichtstrom der Pixellichter eines HD84-Systems.*

Abgeschlossen wird das Preprocessing durch die Bestimmung der mittleren Lichtstärke $I_{v,k,\varnothing}$, welche sich nach Gleichung (7-4) aus dem Verhältnis des Lichtstroms und des Raumwinkels der jeweiligen Lichtquelle ergibt. In Bild 7-13 werden die Quotienten über die drei Zeilen des Moduls visualisiert. Es zeigt sich, dass die kleineren Raumwinkel und der erhöhte Lichtstrom der mittleren Zeile in Kombination zu einer besonders hohen mittleren Lichtstärke in ihrem Einflussbereich führen. Die Anomalie der LED 84 ist auch in der mittleren Lichstärke deutlich erkennbar. Allerdings erscheint sie durch den großen bestrahlten Raumwinkel schwächer als in Bild 7-12.

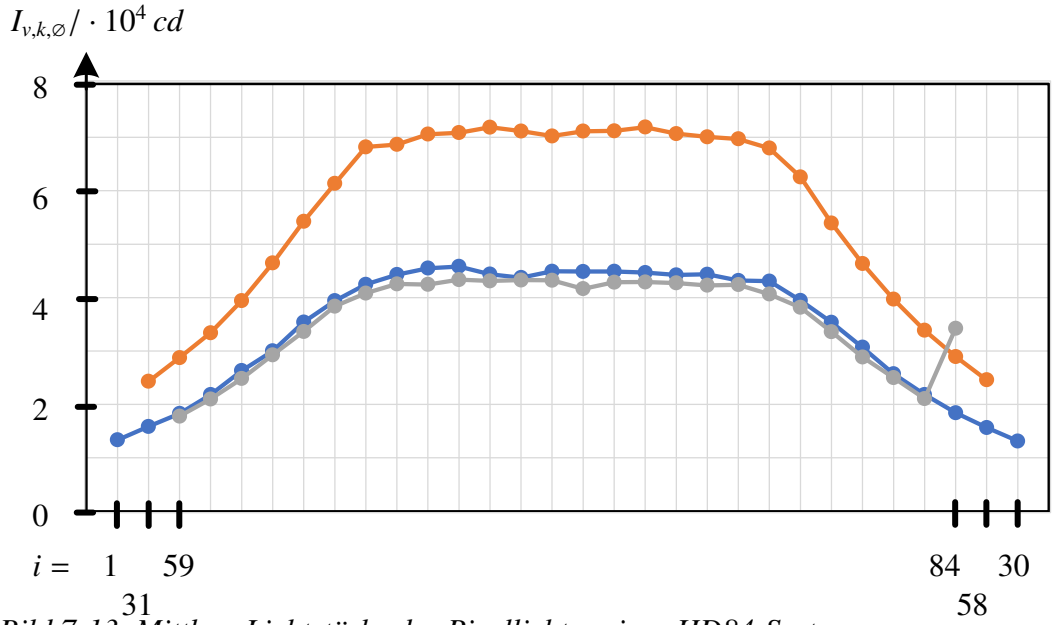


*Bild 7-13: Mittlere Lichtstärke der Pixellichter eines HD84-Systems.*

Basierend auf den extrahierten Merkmalen bestimmt das Verfahren die Dimmwerte, welche für die Umsetzung der Referenzbeleuchtungsstärke $E_{v,ref}$ im Selektionsbereich $R_{sel}$ erforderlich sind. In der Validierungsszene kann der Kugelradius $r_s = 10\,m$ zur Konvertierung von $E_{v,ref}$ auf die Referenzlichtstärke $I_{v,ref} = 10.000\,cd$ verwendet werden. Um $I_{v,ref}$ einzustellen, werden die in Bild 7-14 gezeigten Dimmwerte gemäß Gleichung (7-6) bestimmt. Es stellt sich heraus, dass die Lichtquellen $i = 7 \ldots 13$ der unteren Zeile, $i = 36 \ldots 42$ der mittleren Zeile und $i = 63 \ldots 69$ der oberen Zeile als relevant für den Selektionsbereich $R_{sel}$ eingestuft werden. Da die lokale Filterung inaktiv ist, behalten alle übrigen Lichtquellen ihren Standarddimmwert $d_{std,i} = 0\%$. Der Algorithmus kompensiert den Abfall der mittleren Lichtstärke im Randbereich aller Zeilen durch die Erhöhung des Dimmwerts mit zunehmender Randnähe. Zusätzlich wird die mittlere Zeile mit weniger Leistung versorgt um die Lichtstärke der unteren und oberen Zeile anzugleichen.

Die berechneten Dimmwerte ergeben letztlich die Gesamtlichtverteilung des Scheinwerfers, welche unter Berücksichtigung der Szenengeometrie auf die Beleuchtungsstärke führt. Zur Bewertung der eingestellten Lichtverteilung werden zwei Maße herangezogen. Zuerst wird die mittlere Beleuchtungsstärke $E_{v,\varnothing}$ innerhalb des Selektionsbereichs $R_{sel}$ mit der Referenzbeleuchtungsstärke $E_{v,ref}$ verglichen. Analog zur mittleren Lichtstärke (s. Gleichung (7-4)) kann $E_{v,\varnothing}$ durch die diskrete Integration der lokalen Beleuchtungsstärkewerte im Selektionsbereich gemäß

$$E_{v,\varnothing} = \frac{1}{\Omega_{sel}} \sum_{k=1}^{84} \int_{\varphi=\underline{\varphi}}^{\bar{\varphi}} \int_{\theta=\underline{\theta}}^{\bar{\theta}} \frac{I_{v,k}(\theta,\varphi)}{r(\theta,\varphi)^2} \cos\alpha \cos\theta d\theta d\varphi$$

bestimmt werden, wobei $\Omega_{sel}$ dem Raumwinkel der Selektion $R_{sel}$ entspricht. Die Distanz der Lichtquellen zu den beleuchteten Flächen wird für jede diskrete Richtungskomponente $(\theta, \varphi)$ ermittelt. Im beschriebenen Validierungsszenario wird eine mittlere Lichtstärke von $E_{v,\varnothing} = 96.2\,lx$ erreicht, sodass eine relative Abweichung von unter 4% erreicht wird. Es sei jedoch darauf hingewiesen, dass die Abweichungen signifikant ansteigen, sobald die Grenzen des Gesamtausleuchtungsbereichs erreicht werden oder der Selektionsbereich schlecht durch die Segmentierung des Systems nachgebildet werden kann.

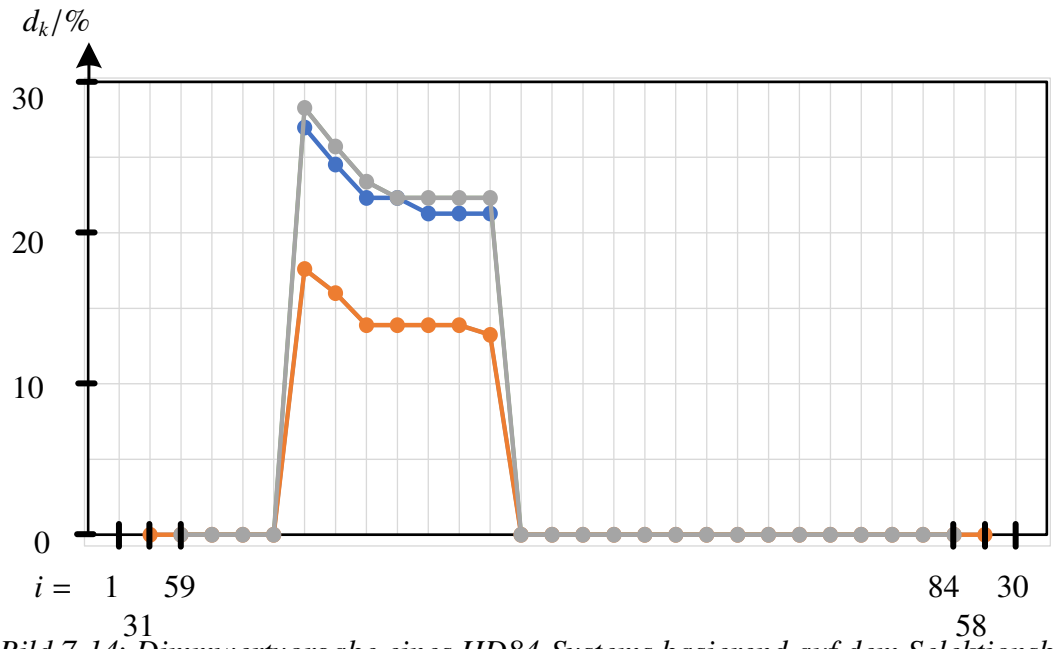


*Bild 7-14: Dimmwertvorgabe eines HD84-Systems basierend auf dem Selektionsbereich $R_{sel}$ und der Sollbeleuchtungsstärke $E_{ref}$.*

Neben der mittleren Abweichung wird der Soll-Ist-Wertabgleich auch lokal aufgelöst durchgeführt. In Bild 7-15 wird die relative Abweichung des Istwerts von der Referenz $E_{v,ref}$ bezüglich des Polar- und Azimutwinkels aus Sicht des Scheinwerfers in Falschfarben visualisiert. Zunächst lässt sich feststellen, dass eine weitgehende Abdeckung des selektierten Ausleuchtungsbereichs $R_{sel}$ gegeben ist. Obwohl $E_{v,ref}$ im Mittel sehr gut reproduziert wird, treten in lokal begrenzten Bereichen signifikante Abweichungen vom Sollwert auf. Diese Abweichungen lassen sich primär auf Hardware-Limitierungen zurückführen. Konkret sind hierbei die begrenzte Auflösung des HD84-Systems, die Inhomogenitäten der Lichtverteilungen einzelner Pixellichter und die Überlappung der einzelnen Ausleuchtungsbereiche im Randbereich der Matrix zu nennen. Letzteres äußert sich in der erheblichen Überschreitung des Sollwerts im Bereich $\varphi = -9°$. Die Inhomogenitäten spiegeln sich in den horizontalen roten Linien der einzelnen Zeilen wieder, wobei der Sollwert im Bereich der mittleren Zeile besser getroffen wird.

### 7.2.4 Globale Optimierung

Der bisher vorgestellte Ansatz ermittelt den Dimmwert jeder Lichtquelle unabhängig von den Dimmwerten aller anderen Lichtquellen beider Scheinwerfer. Diese Lokalität und die Eigenschaft, dass aufwendige Berechnungen zu einem Großteil in das Preprocessing ausgelagert werden können, erlauben eine hochperformante Implementierung der bisher vorgestellten Lösung, die sich in dieser Form problemlos für die Integration in einem Scheinwerfersteuergerät eignet. Ein Nachteil des lokalen Ansatzes ist jedoch, dass die Wechselwirkungen zwischen den einzelnen Lichtquellen außer Acht gelassen werden. Licht, das eine Lichtquelle $k$ außerhalb des zugewiesenen Einflussbereichs $R_{l,k}$ strahlt, wird von dem lokalen Verfahren vollkommen außer Acht gelassen, weshalb die Ergebnisse nicht optimal sind.

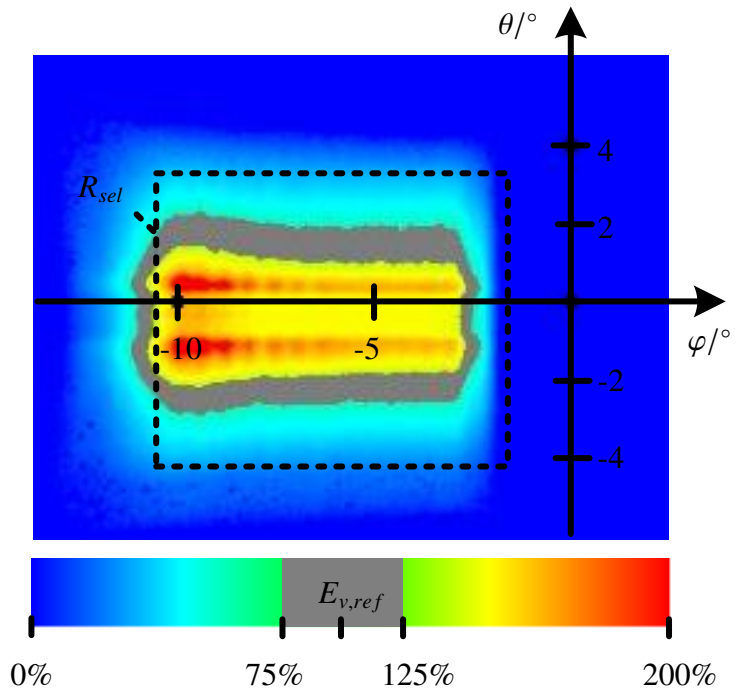


*Bild 7-15*: *Falschfarbendarstellung der Abweichung von der Referenzbeleuchtungsstärke* $E_{v,ref}$.

Nun soll ein globaler Ansatz vorgestellt werden, der die gleiche Problemstellung auf eine andere Weise löst. Den Ausgangspunkt bildet der relevante Ausleuchtungsbereich des Scheinwerfersystems, welcher ohne Beschränkung der Allgemeinheit vereinfachend als rechteckig bezüglich der Raumwinkelkoordinaten angenommen wird. Er wird durch das umschließende Rechteck in Bild 7-16 dargestellt.

Dieser Ausleuchtungsbereich wird nun durch kleinere Rechtecke segmentiert. Im Beispiel aus Bild 7-16 wird der Gesamtbereich in $N_s$ äquidistante Einheiten entlang der Horizontalen und $M_s$ äquidistante Einheiten entlang der Vertikalen unterteilt. Der nachfolgend vorgestellte Algorithmus kann jedoch auf gleiche Weise eingesetzt werden, wenn die Segmentierung inhomogen ist. Für jedes Segment $(m, n)$ wird nun die Einflussnahme aller Lichtquellen bei maximaler Bestromung in Form des einwirkenden Lichtstroms bestimmt. Der Lichtstrom $\Phi_{mn,k}$, welcher von der Lichtquelle $k$ bei maximaler Bestromung in das Segment $(m, n)$ abgegeben wird, kann analog zur Gleichung (7-2) gemäß

$$\Phi_{mn,k} = \int_{\varphi=\underline{\varphi}_{mn}}^{\bar{\varphi}_{mn}} \int_{\theta=\underline{\theta}_{mn}}^{\bar{\theta}_{mn}} I_{v,k}(\theta, \varphi) \cos\theta d\theta d\varphi$$

berechnet werden. Dabei bezeichnen $\underline{\varphi}_{mn}$, $\bar{\varphi}_{mn}$, $\underline{\theta}_{mn}$ und $\bar{\theta}_{mn}$ die Begrenzungen des Segments $(m, n)$. Für einen gegebenen Dimmwert $d_k$ und einer im Segment $(m, n)$ als ungefähr konstant angenommenen Entfernung $r_{mn}$ zwischen Scheinwerfer und bestrahlter Fläche kann basierend auf $\Phi_{mn,k}$ die realisierte Beleuchtungsstärke

$$E_{v,mn,k} = h_{mn,k} \cdot d_k \text{ mit } h_{mn,k} = \frac{1}{r_{mn}^2} \cdot \frac{\Phi_{mn,k}}{\Omega_{mn}} \tag{7-10}$$

bestimmt werden. Dabei ist $\Omega_{mn}$ der durch das Segment $(m, n)$ überdeckte Raumwinkelbereich, welcher analog zur Gleichung (7-3) auf Basis der Winkelgrenzen berechnet werden

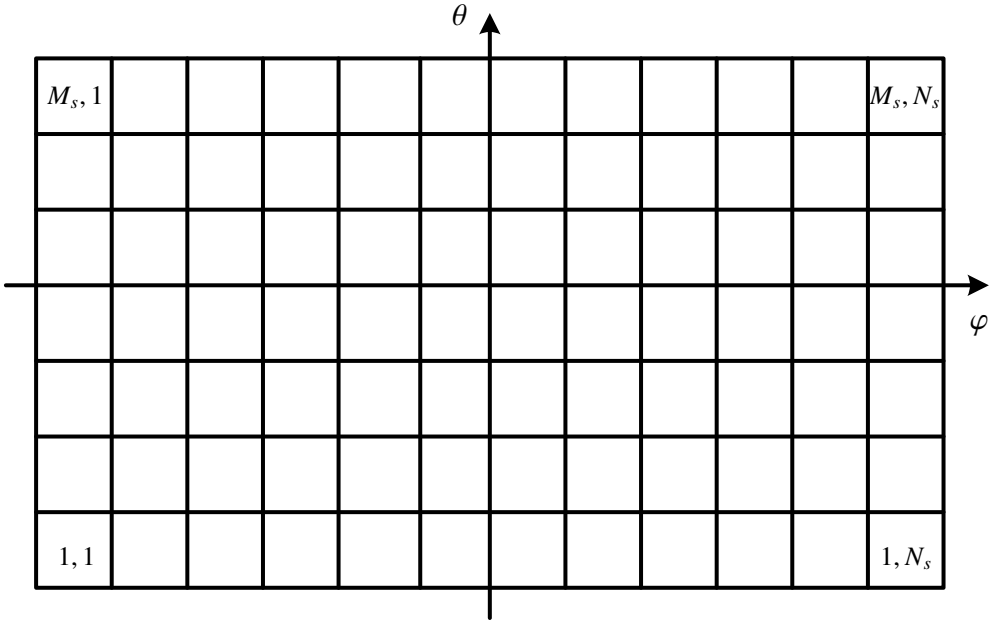


*Bild 7-16*: *Segmentierung des Ausleuchtungsbereichs eines HD-Scheinwerfers zur Formulierung der Dimmwertvorgabe als globales Optimierungsproblem.*

kann. Die einzustellende Lichtverteilung kann nun durch die Vorgabe eines Referenzwerts $E_{v,mn,ref}$ für alle Segmente $(m, n)$ des Ausleuchtungsbereichs beschrieben werden. Der Istwert eines Segments $(m, n)$ ergibt sich durch die Summation der Beiträge aller $K$ Lichtquellen aus Gleichung (7-10) zu

$$E_{v,mn} = \sum_{k=1}^{K} h_{mn,k} \cdot d_k.$$

Um nun die global optimale Dimmwertauswahl zu finden, kann mithilfe der angestellten Vorüberlegungen ein Optimierungsproblem formuliert werden, welches die Abweichungen von den Referenzwerten über den gesamten Ausleuchtungsbereich hinweg minimiert. Eine intuitive Variante dieser Formulierung wird durch das Optimierungsproblem (7-11) dargestellt.

$$\begin{aligned} \min_{d_1,\ldots,d_K \in \mathbb{R}} \; & w_{11}(E_{v,11,ref} - E_{v,11})^2 + \cdots + w_{M_sN_s}(E_{v,M_sN_s,ref} - E_{v,M_sN_s})^2 \\ \text{mit } E_{v,11} &= h_{11,1} \cdot d_1 + \cdots + h_{11,K} \cdot d_K \\ \vdots &= \vdots \\ E_{v,M_sN_s} &= h_{M_sN_s,1} \cdot d_1 + \cdots + h_{M_sN_s,K} \cdot d_K \\ \text{u.d.N. } 1 &\geq d_1, \ldots, d_K \\ 0 &\leq d_1, \ldots, d_K \end{aligned} \tag{7-11}$$

Die Optimierungsvariablen sind die normierten Dimmwerte $d_1$ bis $d_K$. Alle übrigen Größen mit Ausnahme der Entfernungsgrößen $r_{mn}$, welche durch die Umfeldsensorik für jede Richtung $(m, n)$ in der jeweiligen Fahrsituation ausgewertet werden müssen, sind bekannt. Durch die Gewichtungsfaktoren $w_{mn}$ können die einzelnen Segmente verschieden stark berücksichtigt werden. Hierbei empfiehlt es sich, diejenigen Segmente, welche durch die

vorhandenen Lichtquellen aufgrund zu hoher Entfernung oder eines zu geringen Lichtstroms kaum beeinflusst werden können, schwächer zu bewerten. Um den tatsächlichen Einflussbereich des Scheinwerfers in den Fokus der Optimierung zu rücken, wird deshalb die Wahl

$$w_{mn} = \frac{\sum_{k=1}^{K} h_{mn,k}}{h_{max}} \text{ mit } h_{max} = \max_{m\in\{1,\ldots,M_s\},n\in\{1,\ldots,N_s\}} \sum_{k=1}^{K} h_{mn,k}$$

für die Gewichtungsfaktoren vorgeschlagen. Bei diesem Vorgehen erhält das am besten beeinflussbare Segment $(m_{max}, n_{max})$ die Gewichtung $w_{m_{max},n_{max}} = 1$, während die übrigen Segmente Gewichtungsfaktoren aus dem Intervall $[0, 1]$ erhalten. Die Normierung der Gewichtungsfaktoren auf $h_{max}$ ist für das Optimierungsergebnis unerheblich. Sie wird nur zur leichteren Interpretation der Werte vorgenommen.

Weitere Modifikationen des Optimierungsproblems nach Gleichung (7-11) sind denkbar. Beispielsweise könnte die Segmentierung optimiert werden, indem Winkelbereiche mit hohen Dynamik- oder Präzisionsanforderungen feiner segmentiert werden, als es für weniger relevante Bereiche der Fall ist. Damit wird gleichzeitig die Flexibilität bei der Sollvorgabe beeinflusst, welche in der gleichen Segmentierung vorgenommen wird.

Die hier vorgestellte Lösung soll das zugrunde liegende Prinzip verdeutlichen und verzichtet auf die genannten Optimierungsmöglichkeiten. Der grundsätzliche Vorteil gegenüber dem lokalen Verfahren ist die gleichzeitige Berücksichtigung aller Lichtquellen. Hierdurch werden die Einflüsse der Lichtquellen nicht nur in ihren primären Ausleuchtungsbereichen, sondern über den gesamten relevanten Bereich hinweg berücksichtigt und auf optimale Weise kombiniert.

Weiterhin muss nachgewiesen werden, dass das Optimierungsproblem konvex ist. Nur dann ist sicher gestellt, dass die gefundene Lösung dem globalen Optimum entspricht. Im Falle eines quadratischen Optimierungsproblems, wie es in (7-11) vorliegt, genügt es nachzuweisen, dass die quadratische Matrix $Q_{opt}$ der in (7-12) notierten Standardform eines quadratischen Optimierungsproblems positiv semidefinit ist.

$$\begin{aligned}
&\min_{d_1,\ldots,d_K\in\mathbb{R}} x_{opt}^T \cdot Q_{opt} \cdot x_{opt} + c_{opt}^T \cdot x_{opt} + f_{opt} \\
&\text{u.d.N. } 1 \geq d_1, \ldots, d_K \\
&\qquad 0 \leq d_1, \ldots, d_K \\
&\quad \text{mit } x_{opt}^T = \begin{pmatrix} d_1 & \ldots & d_K \end{pmatrix} \\
&\qquad Q_{opt} = \begin{pmatrix} q_{11} & \ldots & q_{1K} \\ \vdots & & \vdots \\ q_{K1} & \ldots & q_{KK} \end{pmatrix}, q_{ij} \in \mathbb{R} \\
&\qquad c_{opt}^T = \begin{pmatrix} c_1 & \ldots & c_K \end{pmatrix}^T, c_k \in \mathbb{R} \\
&\qquad f_{opt} \in \mathbb{R}
\end{aligned} \tag{7-12}$$

Um die Gestalt von $Q_{opt}$ genauer beschreiben zu können, wird das ursprünglich formulierte Optimierungsproblem (7-11) in die Standardform überführt. Durch Auflösen der quadratischen Terme in der Zielfunktion des Optimierungsproblems (7-11), Einsetzen

der Nebenbedingungen und Umsortierung findet man, dass $Q_{opt}$, $c_{opt}$ und $f_{opt}$ für das betrachtete Problem die Gestalt

$$Q_{opt} = (q_{kk'})_{k,k'=1,\ldots,K} \text{ mit } q_{kk'} = \sum_{m=1}^{M_s}\sum_{n=1}^{N_s} w_{mn} h_{mn,k} h_{mn,k'}$$

$$c_{opt}^T = (c_k)_{k=1,\ldots,K} \text{ mit } c_k = -2 \cdot \sum_{m=1}^{M_s}\sum_{n=1}^{N_s} w_{mn} E_{v,mn,ref} h_{mn,k}$$

$$f_{opt} = \sum_{m=1}^{M_s}\sum_{n=1}^{N_s} w_{mn} E_{v,mn,ref}^2$$

$$\text{mit } h_{mn,k} = \frac{1}{r_{mn}^2} \cdot \frac{\Phi_{mn,k}}{\Omega_{mn}} > 0$$

annehmen. Um die positive Semidefinitheit von $Q_{opt}$ nachzuweisen, wird $Q_{opt}$ zunächst in $M_s \cdot N_s$-viele Summanden zerlegt:

$$Q_{opt} = \sum_{m=1}^{M}\sum_{n=1}^{N_s} Q_{mn} \text{ mit } Q_{mn} = (q_{mn,kk'})_{k,k'=1,\ldots,K} = w_{mn} h_{mn,k} h_{mn,k'}.$$

Sind alle Summanden $Q_{mn}$ positiv semidefinit, so ist auch $Q_{opt}$ positiv semidefinit. Es genügt somit, einen allgemeingültigen Summanden $Q_{mn}$ zu betrachten. Die Matrix $Q_{mn}$ kann durch das Vektorprodukt

$$Q_{mn} = w_{mn} h_{mn} \cdot h_{mn}^T \text{ mit } h_{mn} = \begin{pmatrix} h_{mn,1} & \ldots & h_{mn,K} \end{pmatrix}^T$$

formuliert werden. Basierend auf den Vorüberlegungen, kann die positive Semidefinitheit von $Q_{mn}$ leicht nachgewiesen werden. Mit

$$x^T \cdot Q_{mn} \cdot x = w_{mn} x^T h_{mn} h_{mn}^T x = w_{mn} \sum_{k=1}^{K} x_k h_{mn,k} \cdot \sum_{k=1}^{K} x_k h_{mn,k} = w_{mn} (\sum_{k=1}^{K} x_k h_{mn,k})^2 \geq 0$$

und der Eigenschaft $w_{mn} > 0 \forall m \in \{1, \ldots, M_s\}, n \in \{1, \ldots, N_s\}$ ist die positive Semidefinitheit eines beliebigen Summanden $Q_{mn}$ und damit auch die positive Semidefinitheit der Matrix $Q_{opt}$ selbst gezeigt. Schlussfolgernd ist das Optimierungsproblem (7-11) konvex, womit die Konvergenz des Optimierers im globalen Optimum sichergestellt ist.

# 8 Evaluierung

Im Kapitel 4 wurden Anforderungen definiert, welche durch eine Nachtfahrtsimulation erfüllt werden müssen, um den Ingenieur bei der simulationsbasierten Entwicklung von HD-Scheinwerfern bestmöglich zu unterstützen. Die Analyse hat gezeigt, dass derzeit keine Lösung existiert, die das gesamte Spektrum der Anforderungen adäquat abdeckt. Somit wurde letztlich ein Handlungsbedarf abgeleitet, den es im Rahmen dieser Arbeit zu erfüllen galt. Daraus entstanden ist die Nachtfahrtsimulationsumgebung „Hyperion“. Innerhalb dieses Kapitel wird bewertet, inwieweit die im Abschnitt 4.2 zusammengefassten Anforderungen durch Hyperion erfüllt werden.

## 8.1 Überprüfung der Anforderungen

In den folgenden Unterabschnitten wird Hyperion entlang der Kriterien der Anforderungsdefinition analog zu Abschnitt 4.1 evaluiert. Im jeweiligen Unterabschnitt wird auf die korrespondierenden Detailanforderungen **A1** bis **A13** direkt Bezug genommen.

### 8.1.1 Visuelle Qualität

In Abschnitt 4.1.1 ist die hohe Relevanz der subjektiven Bewertung des Lichtingenieurs bereits hervorgehoben worden. Hieraus wurde abgeleitet, dass die visuelle Qualität der Simulation durch eine hohe Übereinstimmung mit den realen Lichtverhältnissen überzeugen muss. Die Validierung des vorgestellten Rendering-Verfahrens in Abschnitt 6.4 hat gezeigt, dass Lichtverteilungen in Hyperion mit guter Übereinstimmung wiedergegeben werden können. Als Referenz wurde hierbei anstelle realer Aufnahmen die Nachtfahrtsimulation „LightDriver“ verwendet. Diese Wahl der Referenz ist zulässig, da die Software „LightDriver“ seit vielen Jahren erfolgreich im produktiven Umfeld genutzt wird. Sie bringt den Vorteil, dass die Fahrzeugumgebung detailgetreu nachgebildet werden kann, womit die Grundlage für einen aussagekräftigen Vergleich gegeben ist. Die Nachbildung realer Umgebungen wäre aufgrund der komplexen Oberflächen und Geometrien entweder nicht hinreichend genau oder mit enormem messtechnischen Aufwand verbunden gewesen.

Dieser Punkt führt auch zu einer Schwachstelle von Hyperion, die im Rahmen der Evaluierung nicht unerwähnt bleiben soll. Während das Rendering des HD-Scheinwerferlichts, welches den Forschungskern dieser Arbeit bildet, detailliert entwickelt und beschrieben wird, kam der Modellierung von virtuellen Umgebungen eine untergeordnete Rolle zu. Hyperion weist in der bisherigen Entwicklungsstufe wesentlich einfachere Umgebungsmodelle als andere in Abschnitt 3.3 vorgestellte Simulationen auf. Dennoch ist das hier präsentierte Rendering-Verfahren des Lichts in gleicher Weise auch auf komplexe Szenen anwendbar, weshalb der wissenschaftliche Wert der implementierten Lösung durch diesen Aspekt nicht beeinträchtigt wird.

Ein weiterer wesentlicher Aspekt ist die Simulation spektraler Lichtverteilungen. Hyperion stellt diese Funktionalität vollständig zur Verfügung. Darüberhinaus unterstützt ein

Assistent beim Import spektraler Datensätze und stellt die korrekte Konvertierung vom Quell- in den Zielfarbraum sicher.

Zusammengefasst kann die Anforderung **A1** (Korrekte Abbildung photometrischer Zusammenhänge) als weitestgehend erfüllt betrachtet werden. Eine Einschränkung stellen die rudimentären Szenenmodelle dar, welche im wissenschaftlichen Sinne als nebensächlich eingestuft werden können. Die Anforderung **A2** (Unterstützung spektraler Lichtverteilungen) wird ohne Einschränkungen erfüllt.

### 8.1.2 Technologie-Kompatibilität und Pixel-Skalierbarkeit

Im Rahmen der Anforderungsdefinition wurde herausgestellt, dass am Markt unterschiedlichste Scheinwerfer-Technologien zum Einsatz kommen werden. Als weitere Anforderung wurde deshalb die Kompatibilität der Nachtfahrtsimulation zu den verschiedenen Technologien in Form der Anforderung **A3** festgehalten. Das in Hyperion eingesetzte Verfahren zur Bestimmung der Gesamtlichtverteilung des Scheinwerfers nach Abschnitt 6.2.2 kann für jede HD-Technologie eingesetzt werden, deren einzelne Lichtquellen durch statische Lichtverteilungen beschrieben werden können. Diese Form der Beschreibung eignet sich sowohl für physikalische Lichtquellen, wie sie bei LED-Matrix-Systemen existieren, als auch für logische Pixellichter am Beispiel von LCD-Systemen. Insofern ist die Anforderung **A3** (Technologie-Kompatibilität) in Hyperion weitestgehend erfüllt. Es sei jedoch darauf hingewiesen, dass die vorgestellte Implementierung nur eingeschränkt verwendbar ist, wenn Scanner-Verfahren zum Einsatz kommen. Bei diesen Systemen wird der gesamte Ausleuchtungsbereich hochfrequent durch einen konzentrierten, meist quasikontinuierlichen Lichtspot abgescannt. Die Vorgänge laufen dabei so schnell ab, dass der Mensch diese nicht sieht und nur die Gesamtlichtverteilung wahrnimmt, deren Gestalt aus den Belichtungszeiten der verschiedenen Winkelbereiche resultiert. Prinzipbedingt kann die Gesamtlichtverteilung nicht aus der Summe fixer Einzellichtverteilungen beschrieben werden, wodurch sich die eingeschränkte Eignung erklärt.

Einher mit den verschiedenen Scheinwerfer-Technologien gehen die unterschiedlichen Größenordnungen der individuell ansteuerbaren Pixellichter. Insofern stellt die Skalierbarkeit der Nachtfahrtsimulation hinsichtlich der Anzahl von Pixellichtquellen unter Erhalt der Echtzeitfähigkeit die zentrale Anforderung **A4** dar. Durch eine völlige Neustrukturierung der Daten, die eine hochgradig parallele Bestimmung der Gesamtlichtverteilung durch die GPU erlaubt, gelingt in Hyperion eine Echtzeit-Simulation in der Größenordnung von einer Millionen Lichtquellen. Direkten Einfluss auf die Laufzeit nehmen die Rechenleistung der Hardware, die Gestalt der Einzellichtverteilungen, der Raumwinkel der Gesamtlichtverteilung und die Winkelauflösung. Details dazu wurden in Abschnitt 6.5 ausführlich untersucht. Verglichen mit den am Markt existierenden Nachtfahrtsimulationen ordnet sich Hyperion damit an der Spitze ein. Alle praxisrelevanten Systeme können in Echtzeit simuliert werden, ohne Einbußen bezüglich der Auflösung oder die Vernachlässigung von Streulicht der einzelnen Pixellichter in Kauf nehmen zu müssen. Bild 8-1 greift das Bild 4-1 aus der Anforderungsdefinition auf und stellt die Kompatibilität von Hyperion mit den verschiedenen Technologien bzw. Größenordnungen grafisch dar.

Wie sich zeigt, können grundsätzlich alle Systeme durch Hyperion simuliert werden. Ursächlich hierfür ist die Formulierung eines ausgabesensitiven Algorithmus, welcher

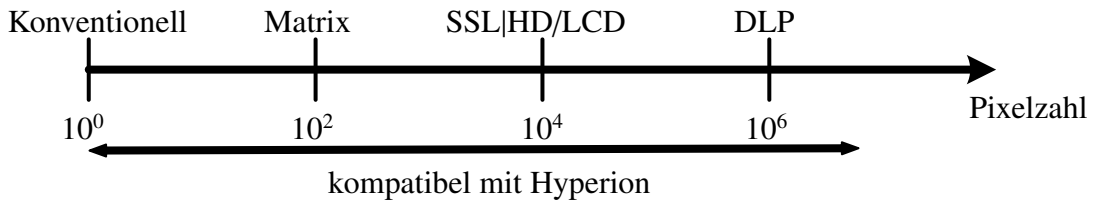


*Bild 8-1*: *Kompatibilität von Hyperion zu verschiedenen HD-Technologien und den jeweiligen Anzahlen von Pixellichtquellen.*

algorithmisch nicht von der Lichtquellenzahl abhängt. Sowohl die theoretische Komplexitätsbetrachtung in Abschnitt 6.2.3 als auch die Laufzeitmessungen in Abschnitt 6.5.1 haben gezeigt, dass der Algorithmus bezüglich der Lichtquellenzahl nicht sensitiv ist. Die Anforderung **A4** (Skalierbarkeit bezüglich der Pixelanzahl) kann somit als vollständig erfüllt betrachtet werden.

### 8.1.3 Echtzeitfähigkeit und X-in-the-Loop Testing

Mit dem Einzug der HD-Technologie im automobilen Licht ist die Komplexität des Scheinwerfersteuergeräts erheblich gewachsen. Aus diesem Grund wurde in Abschnitt 4.1.3 die Anforderung **A5** formuliert, welche die Unterstützung von In-the-Loop-Tests fordert. Wie in Abschnitt 5.5 ausgeführt wurde, unterstützt Hyperion das gesamte Spektrum von MiL-, SiL- und HiL-Tests. Die Anforderung **A5** kann insofern als vollständig erfüllt betrachtet werden.

Eine Grundvoraussetzung für SiL- und HiL-Tests stellt die Echtzeitfähigkeit der gesamten Simulation dar. Diese beinhaltet den Kreislauf aus der Simulation von Fahrzeug und Umgebung, der Berechnung virtueller Sensorsignale, der Übermittlung dieser Sensorwerte an das Steuergerät sowie der Darstellung der Steuergeräteausgaben in Form des Scheinwerferlichts in der virtuellen Umgebung. Die Echtzeitanforderung bezüglich des Steuergerätetakts wird somit durch **A5** impliziert. Wie jedoch in Abschnitt 4.1.3 ausgeführt wird, muss im Rahmen der Nachtfahrtsimulation noch eine weitere Echtzeitschranke Beachtung finden. Diese begründet sich in der Immersion des menschlichen Fahrers, dessen visuelle Wahrnehmung die Simulatorausgabe erst bei hinreichend vielen Frames pro Sekunde als flüssiges Video wahrnimmt. Zur Berücksichtigung beider Echtzeitschranken wurde deshalb zusätzlich die Anforderung **A6** definiert, welche eine Bildwiederholfrequenz von etwa 60 Hz fordert.

Wie aus der Laufzeitbetrachtung in Abschnitt 6.5 hervorgeht, gelingt das Licht-Rendering, welches die mit Abstand rechenintensivste Komponente der Gesamtsimulation darstellt, mit deutlichen Reserven zur Echtzeitschranke. Die übrigen Komponenten der Simulation können zum Teil parallel erfolgen oder sind bezüglich ihres Laufzeitbedarfs vernachlässigbar. Der Echtzeit-Aspekt der Anforderung **A6** kann demzufolge als erfüllt betrachtet werden.

### 8.1.4 Latenz

Gemäß Abschnitt 4.1.4 muss neben der Dauer einer Simulationsiteration, welche maßgeblich für die Echtzeitfähigkeit der Anwendung ist, auch die Latenz beachtet werden. Die Zeitspanne von den Eingaben des Fahrers bis zur Reaktion durch die visuelle Ausgabe ergibt sich durch die Interaktion einer Vielzahl von technischen Komponenten, von denen jede einen gewissen Zeitraum für das Empfangen, Verarbeiten und Senden von Daten benötigt. Im Laufe der Implementierung von Hyperion wurden die iterativ erforderlichen Operationen durch den flächendeckenden Einsatz von Preprocessing minimiert. Außerdem wurde die Kommunikation zwischen den verschiedenen technischen Komponenten der Architektur optimiert. Die verbleibende Latenz ist so gering, dass sie nicht wahrgenommen werden kann. Somit kann auch der Latenz-Aspekt der Anforderung **A6** als erfüllt betrachtet werden.

### 8.1.5 Lichtanalyse und -design

Während die Nachtfahrtsimulation im Regelfall das Ziel verfolgt, die realen Licht- und Sichtverhältnisse möglichst exakt wiederzugeben, kann es abseits von Probandenstudien auch sinnvoll sein, die Wahrnehmung schwer erkennbarer Lichteigenschaften durch künstliche Einblendungen zu unterstützen. Die Anforderung **A7** umfasst derartige Analysetools und fordert zudem den Ausbau der etablierten Analysen um die neuen Aspekte im Zuge der HD-Technologie.

In Hyperion werden alle etablierten Analysen angeboten. Zudem werden HD-spezifische Analysesichten implementiert, welche unter dem Begriff „Dimmwertanalyse“ zusammengefasst werden. Dazu gehören die Darstellung der Dimmwerte aller Pixellichter und der daraus resultierenden Gesamtlichtverteilungen der Scheinwerfer in Falschfarben. Die analysierbaren Größen und deren Visualisierungsmöglichkeiten werden in den Abschnitten 7.1.1 und 7.1.2 ausführlich dargestellt. Mit der HD-Technologie haben die Berücksichtigung von Sensorgrößen und insbesondere der Umfelderkennung im Steuergerät an Bedeutung gewonnen. Die Visualisierung derartiger Informationen wird deshalb ebenfalls in Hyperion unterstützt und wird in Abschnitt 7.1.3 ausgeführt. Zusammengefasst kann die Anforderung **A7** als erfüllt betrachtet werden.

Als logische Folge der Analysewerkzeuge werden in Abschnitt 4.1.5 die Designwerkzeuge genannt, welche den Ingenieur beim Entwurf neuer Lichtfunktionen aktiv unterstützen sollen. Ursächlich hierfür ist die Zweiteilung des Entwurfsprozesses, die mit dem Einzug der HD-Technologie einher geht. Gegenüber der physikalischen Auslegung des Scheinwerfers gewinnen die Lichtsteueralgorithmen an Bedeutung. Gleichzeitig ist die direkte Formulierung dieser Algorithmen ohne unterlagerte Abstraktionsschichten zur logischen Reduzierung der Lichtquellenmenge des Scheinwerfers schlicht nicht praktikabel.

In Abschnitt 7.2 wird eine neuartige Methodik vorgestellt, mit welcher die Vielzahl der Lichtquellen eines HD-Scheinwerfers beherrschbar wird. Weiterhin erlaubt die dort angewendete Abstraktion des realen Scheinwerfers eine weitgehend automatisierte Überführung von Lichtfunktionen auf andere HD-Systeme. Ausbaufähig ist die Unterstützung bei der Gestaltung der Mapping-Funktion von Sensorausgaben auf Selektionsbereiche. Hier sind weitere Automatismen denkbar, wobei diese dann spezifisch für die Art der Lichtfunk-

tion gestaltet werden müssten. Die Anforderung **A8** wird deshalb als weitgehend erfüllt betrachtet.

### 8.1.6 Witterung

Wie in Abschnitt 4.1.6 dargestellt wurde, gestaltet sich die Erprobung von Szenarien mit spezifischen Witterungsbedingungen, wie Regen, Nebel oder Schnee, in einer realen Umgebung sehr schwierig. Derartige Wetterverhältnisse, darunter ganz besonders Nebel, sind schwer vorhersagbar, wodurch anstelle von nächtlichen Erprobungsfahrten vermehrt auf statische Tests in geschlossenen Räumen mit künstlich erzeugten Witterungsbedingungen zurückgegriffen werden muss. Motiviert durch dieses Erschwernis in der Entwicklung wurde die Anforderung **A9** (Unterstützung verschiedener Witterungsbedingungen) definiert.

Zur Nachbildung einzelner Witterungsbedingungen innerhalb der Simulation sind ganz unterschiedliche Vorgehen erforderlich, wobei die Einflüsse jeder Wetterlage auf die Licht- und Sichtverhältnisse sehr komplex sind. Im Rahmen von Hyperion wird in Form des Nebels deshalb nur eine Witterungssituation nachgebildet, welche die grundsätzliche Eignung des Licht-Renderings zur Darstellung derartiger Phänomene nachweisen soll. Die konkrete Implementierung des Nebels in Hyperion wird im Abschnitt 6.6 ausführlich beschrieben. Wesentliche Effekte, wie die Eigenblendung, die Lichtkeulen der Scheinwerfer und die eingeschränkte Sicht, können reproduziert werden. Zukünftig wäre die Nachbildung weiterer Witterungen, wie Regen oder Schnee, interessant. Die Anforderung **A9** wird deshalb als teilweise gelöst eingestuft.

### 8.1.7 Fahrdynamik

Auch wenn die exakte Nachbildung der realen Fahrdynamik in der Nachtfahrtsimulation eine untergeordnete Rolle spielt, existieren Lichtfunktionen, wie die dynamische Leuchtweitenregelung, die durch das dynamische Verhalten des Fahrzeugs maßgeblich beeinflusst werden. Aus diesem Grund wurde im Abschnitt 4.1.7 die Anforderung **A10** (Verwendbarkeit eines komplexen Fahrzeugmodells) eingeführt.

Hyperion erfüllt die Anforderung **A10**, indem es die Kopplung des ASM Fahrzeugmodells erlaubt. Wie in Abschnitt 5.2.2 beschrieben, kann der Simulationsrechner via Ethernet an ein Echtzeitsystem gekoppelt werden, welches das ASM-Fahrzeugmodell berechnet. Die Lage des Egofahrzeugs wird mit Hyperion synchronisiert und entsprechend visualisiert. Die Anforderung **A10** kann somit als erfüllt betrachtet werden. Da ein so aufwendiges Fahrzeugmodell für viele Anwendungen nicht erforderlich ist, bietet Hyperion zusätzlich die in Abschnitt 5.2 beschriebenen Alternativen an.

### 8.1.8 Konfigurierbarkeit

In Abschnitt 4.1.8 wurde mit der Anforderung **A11** die Konfigurierbarkeit des Hard- und Software-Setups eingefordert. Begründet wird diese Anforderung durch verschiedene Aspekte. Ein wesentlicher Kern ist die Einsatzmöglichkeit von Hyperion auf dem Desktop-

PC eines Ingenieurs bis hin zum Großsimulator mit Bewegungssystem, Fahrzeugmock-up und mehrkanaliger Ausgabe. Ein weiterer Grund sind die bereits angeführten MiL-, SiL- und HiL-Testmöglichkeiten, die Hyperion mitbringt. Die genannten und weitere Einsatzzwecke erfordern ganz unterschiedliche Hard- und Softwarekonfigurationen. Hyperion stellt diese Flexibilität durch eine modulare Architektur sicher. Wie Abschnitt 5.8 nachweist, lässt sich Hyperion in einer Vielzahl von Konfigurationen betreiben. Die Anforderung **A11** kann somit als erfüllt betrachtet werden.

### 8.1.9 Parametrierbarkeit

Als große Stärke der Simulation wird in Abschnitt 4.1.9 der Anforderungsdefinition die Erzeugung beliebiger Testszenarien mit geringem Aufwand und deren vollständige Reproduzierbarkeit hervorgehoben. Ein Szenario setzt sich dabei aus verschiedenen Parametern, wie dem Streckenverlauf, der Streckenumgebung, den Verkehrsteilnehmern und ihren Trajektorien, der Tageszeit und Witterung sowie dem Scheinwerfersystem zusammen. Hinzu kommt, dass die Manipulation einzelner Parameter zur Laufzeit beispielsweise für einen direkten Vergleich zweier Szenarien sinnvoll sein kann. Unter der Anforderung **A12** (Umfangreiche, reproduzierbare Parametrierung mit Laufzeitanpassung) werden all diese Merkmale zusammengefasst.

Zur Erfüllung der Anforderung **A12** existieren in Hyperion verschiedene Komponenten. Einerseits verfügt die Simulation über eine umfangreiche Streckengenerierung, die in Abschnitt 5.3 detailliert dargestellt wurde. Diese Generierung erlaubt die automatische Erzeugung des dreidimensionalen Straßenverlaufs, des Geländes und der Randbebauung. Zur Streckendefinition wird das standardisierte Format „OpenDRIVE“ unterstützt. Außerdem können Ausschnitte aus der OpenStreetMap-Karte importiert werden. Hier müssen im Regelfall jedoch manuelle Korrekturen am Kartenmaterial vorgenommen werden. Insgesamt bietet Hyperion also die Möglichkeit einer schnellen und komfortablen Szenengenerierung. Dem Ingenieur ist es somit einfach möglich, das Scheinwerfersystem auf verschiedenen Strecken zu erproben.

Darüber hinaus verfügt Hyperion über eine umfangreiche Parameterverwaltung, mit welcher auf alle Bereiche der Simulation Einfluss genommen werden kann. Einmal definierte Parametersätze lassen sich jederzeit speichern und laden. Zudem wird zwischen Offline- und Online-Parametern unterschieden, wobei letztere zur Laufzeit manipuliert werden können. Zur Parametrierung des Szenarios steht zum einen in der Simulation selbst eine Parameterliste zur Verfügung, die vom linken Bildrand aus erreicht werden kann. Zum anderen kann Hyperion über eine Netzwerk-Schnittstelle durch einen entfernten Remote-Client parametriert werden.

Durch die Kombination der Streckengenerierung und des umfangreichen Parameter-Interfaces mit Lade- und Sicherungsfunktionen von vordefinierten Parametersätzen erfüllt Hyperion die Anforderung **A12** vollständig.

### 8.1.10 Remote-Fähigkeit

Ein Aspekt der Anforderung **A11** (Konfigurierbarkeit des Hard- und Softwaresetups) sind die verschiedenen Einsatzbereiche von Hyperion, die von der Nutzung auf einem Desktop-PC am Arbeitsplatz des Ingenieurs bis zum Betrieb eines Großsimulators reichen. Letzterer erfordert in der Regel mindestens zwei Personen, wobei eine Person die Rolle des Testfahrers einnimmt, während eine zweite Person den Simulator betreibt und als Versuchsleiter agiert. In der Konsequenz wurde in Abschnitt 4.1.10 die Anforderung **A13** (Remote-Bedienbarkeit) definiert. Sie fordert eine Netzwerkschnittstelle, durch welche alle im Großsimulator-Betrieb erforderlichen Eingriffe aus der Ferne getätigt werden können.

Um diese Funktionalität zu erbringen, wurde eine Remote-Applikation entworfen, die sich mit Hyperion permanent synchronisiert. In Abschnitt 5.7 werden die Funktionen dieser Applikation umfänglich beschrieben. Sie erlaubt das Hoch- und Herunterfahren der Hardware-Komponenten, das Starten und Stoppen der Simulation, die Manipulation der Offline- und Online-Parameter sowie ihre Übergabe an die verteilten Recheneinheiten. Zudem zeigt sie den aktuellen Status der im Verbund beteiligten Hard- und Softwarekomponenten an. Zusammengefasst erfüllt Hyperion die Anforderung **A13** vollständig.

## 8.2 Zusammenfassung der Evaluierung

Abschließend sollen die Evaluierungsergebnisse in übersichtlicher Form zusammengefasst werden. Tabelle 8-1 stellt dazu das Evaluierungsergebnis bezüglich der einzelnen Anforderungen aus Abschnitt 4.2 dar.

*Tabelle 8-1*: *Evaluierungsergebnisse der Nachtfahrtsimulation „Hyperion“ bezüglich der gestellten Anforderungen.*

| Erfüllungsgrad | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **A7** | **A8** | **A9** | **A10** | **A11** | **A12** | **A13** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| gar nicht | | | | | | | | | | | | | |
| teilweise | | | | | | | | | x | | | | |
| weitestgehend | x | | x | | | | | x | | | | | |
| vollständig | | x | | x | x | x | x | | | x | x | x | x |

Wie die Tabelle zeigt, wird der Großteil aller Anforderungen vollständig erfüllt. Nur Anforderung **A9** (Unterstützung verschiedener Witterungsbedingungen) wird als „teilweise erfüllt“ eingestuft. Hintergrund dieser Einstufung ist, dass mit dem Nebel nur eine der relevanten Witterungstypen in hinreichender Tiefe modelliert wurde. Aufgrund der Komplexität der verschiedenen Wetterphänomene und ihrer Wechselwirkungen mit Licht musste der Aufwand im Rahmen dieser Arbeit auf ein Beispiel beschränkt werden. Die grundsätzliche Eignung der Lichtmodellierung zur Integration dieser Phänomene wurde dennoch nachgewiesen.

# 9 Zusammenfassung und Ausblick

Im letzten Kapitel dieser Arbeit werden die zuvor dargestellten Inhalte unter Nennung der wesentlichen Teilergebnisse zusammengefasst. Außerdem wird ein Fazit aus den Ergebnissen gezogen. Daran angeschlossen gibt Abschnitt 9.2 einen Ausblick auf weitere Ausbaumöglichkeiten der hier vorgestellten Nachtfahrtsimulation „Hyperion“.

## 9.1 Zusammenfassung

Mit dem Einzug der HD-Technologie ist die Komplexität von KFZ-Scheinwerfern erheblich angestiegen. Die gewonnenen Potentiale der neuen Technologie können nur dann ausgeschöpft werden, wenn die ohnehin erschwerten Bedingungen bei der Scheinwerferentwicklung durch geeignete Werkzeuge verbessert werden. In diesem Kontext ist die simulative Erprobung eine Schlüsselkomponente. Aufgrund der Zeit- und Kosteneffizienz, der Reproduzierbarkeit sowie der Sicherheit werden KFZ-Scheinwerfer seit geraumer Zeit simulationsgestützt entwickelt. Den wesentlichen Teil der Entwicklungsarbeit stellte vor der HD-Technologie die Auslegung des physischen Scheinwerfers dar. Die Gestaltung der Lichtquellen und Optiken waren die klassischen Kernaufgaben innerhalb der Auslegung eines Scheinwerfersystems. Mit der dynamischen Anpassung der Lichtverteilung eröffnen HD-Scheinwerfer ein völlig neues Feld und verschieben damit den Schwerpunkt der Entwicklungstätigkeit. Lichtsteueralgorithmen und die durch sie umgesetzten Lichtfunktionen (s. Abschnitt 3.2) rücken durch die gewonnene Flexibilität in den Fokus. Die Abdeckung dieser neuen Technologie in der simulativen Entwicklung von Scheinwerfern ist somit zwingend erforderlich.

Gemäß der Darstellungen in Abschnitt 3.3 arbeiten die Hersteller der etablierten Nachtfahrtsimulationen an der Unterstützung von HD-Systemen. Um die am Markt verfügbaren Lösungen bewerten zu können, wurden in Kapitel 4 Anforderungen definiert, die eine Nachtfahrtsimulation für HD-Scheinwerfersysteme erfüllen sollte. Die anschließende Bewertung und Gegenüberstellung der Simulationen hat gezeigt, dass keine der verfügbaren Lösungen die verschiedenen Kriterien in hinreichender Weise erfüllt. Hieraus wurde der Handlungsbedarf abgeleitet, welcher zu der vorliegenden Arbeit und der darin vorgestellten Nachtfahrtsimulation „Hyperion“ führte.

In Kapitel 5 wird einleitend die Hard- und Softwarearchitektur von Hyperion vorgestellt. Das Bild 5-1 visualisiert die wesentlichen Komponenten der Architektur und ihre Wechselwirkungen untereinander in übersichtlicher Form. Auf oberster Ebene kann zwischen der Fahrdynamiksimulation, der visuellen Simulation, dem Steuergerät als XiL-Testkomponente und dem Simulator-Interface unterschieden werden. Die verschiedenen Elemente werden anschließend genauer diskutiert. Ein besonderes Augenmerk kommt der Streckengenerierung zu, welche die virtuellen Szenen zur simulativen Erprobung weitestgehend automatisiert erzeugt. Am Ende des Kapitels werden verschiedene Konfigurationen vorgestellt, in denen Hyperion betrieben werden kann. Die Modularität der Architektur erlaubt Einsätze vom Arbeitsplatzrechner eines Lichtingenieurs bis hin zum Großsimulatorbetrieb.

Das Rendering des Lichts hochauflösender Scheinwerfer wird in Kapitel 6 als wissenschaftlicher Kern der Arbeit detailliert beschrieben. Einleitend wird die typische datentechnische Abbildung von HD-Scheinwerfern vorgestellt, welche durch die Codierung der Einzellichtverteilungen aller Lichtquellen eines Scheinwerfers realisiert wird. Die Lichtverteilungen können abhängig davon, ob sie nur photometrische oder auch spektrale Informationen beinhalten, in verschiedenen Formaten vorliegen. Nachfolgend wird das formale Vorgehen zur Bestimmung der Gesamtlichtverteilung dargelegt. Die einzelnen Lichtverteilungen werden als Matrizen formalisiert und linear kombiniert, wobei die Koeffizienten der Linearkombination durch die normierten Dimmwerte der Lichtquellen vorgegeben sind.

Betrachtet man das intuitive Vorgehen zur Bestimmung der Gesamtlichtverteilung wird deutlich, dass eine echtzeitfähige Implementierung für hohe Lichtquellenzahlen nicht gewährleistet werden kann. Aus diesem Grund unterscheidet sich das implementierte Vorgehen prinzipiell von der intuitiven Vorgehensweise. Dennoch führen beide Varianten zur gleichen Gesamtlichtverteilung. Kernelement der Implementierung ist die völlige Neustrukturierung der Scheinwerferdaten. Im Rahmen eines Preprocessings werden die vielen Einzellichtverteilungen auf geschickte Weise durch drei Datenstrukturen (Compute Buffer) abgebildet. Diese Reorganisation führt zu einer erheblichen Speicherreduktion, wie es am Beispiel des HD84-Systems (95,8% Speicherreduktion) gezeigt werden konnte. Gleichzeitig erlaubt sie die Formulierung eines effizienten Algorithmus zur Durchführung der Berechnungen zur Laufzeit. In Abschnitt 6.2.4 wurde nachgewiesen, dass der Berechnungsaufwand zur Laufzeit durch das vorgelagerte Preprocessing unabhängig von der Lichtquellenzahl wird. Dieser theoretische Komplexitätsnachweis des ausschließlich ausgabesensitiven Algorithmus konnte durch Laufzeitmessungen untermauert werden. Allerdings zeigt sich in der Praxis ein Einfluss der Lichtquellenzahl ab ca. 10.000 Lichtquellen, welcher auf das Speichermanagement der Grafikkarte zurückgeführt werden kann (s. Abschnitt 6.5.1). Dieser ist im Bereich praxisrelevanter Lichtquellenzahlen jedoch so gering, dass er die Echtzeitfähigkeit der Implementierung nicht gefährdet.

Nachdem die Gesamtlichtverteilung bekannt ist, müssen die darauf basierenden Einflüsse des Lichts in der virtuellen Szene dargestellt werden. Hierzu wurde ein eigenes Lichtquellenmodell mittels Command Buffer in die Deferred Pipeline injiziert. Auf diese Weise können Anpassungen am Lichtquellenmodell vorgenommen werden, ohne die Kompatibilität zu bestehenden Lichtquellen der Unity-Engine zu gefährden. Die implementierte Lichtquelle ist eng am Spotlicht der Unity-Engine orientiert, sodass die Laufzeiteigenschaften und die Korrektheit des implementierten Lichtquellenmodells weitgehend abgesichert sind. Dennoch werden die Ergebnisse durch einen Vergleich mit dem LightDriver validiert (s. Abschnitt 6.4.2). Neben der qualitativen Validierung wird in Abschnitt 6.5.2 das Laufzeitverhalten durch Parameterstudien untersucht. Zusammengefasst ist die Kombination der Gesamtlichtverteilungsberechnung und des Lichtquellenmodells über alle praxisrelevanten Lichtquellenzahlen hinweg echtzeitfähig, ohne dabei Vereinfachungen zu treffen oder verlustbehaftete Komprimierungen vorzunehmen.

Den Abschluss von Kapitel 6 bildet die Simulation von Witterungsbedingungen. Am Beispiel des Nebels wird gezeigt, dass die physikalisch motivierte Modellierung und Implementierung einer Witterungsbedingung in Hyperion vorgenommen werden kann. Wie sich zeigt, kann die Nebelsimulation die real auftretenden Effekte der Eigenblendung und Sichtbeeinträchtigung nachbilden. Im direkten Vergleich mit der Nebelvisualisierung der Unity-Engine sind die Vorteile der hier vorgestellten Variante deutlich erkennbar. Aufgrund

der Vielschichtigkeit realer Witterungsphänomene wird im Rahmen der vorliegenden Arbeit auf die Integration weiterer Witterungsmodellierungen verzichtet.

Aufbauend auf der implementierten Lichtsimulation schließen sich in Kapitel 7 Analyse- und Entwurfswerkzeuge für HD-Scheinwerfer an. Zunächst werden die relevanten photometrischen Größen sowie ihre Vor- und Nachteile bei der Bewertung eines Scheinwerfersystems diskutiert. Die etablierten Visualisierungen dieser Größen durch Isolinien oder Falschfarben werden in die HD-Domäne überführt. Außerdem wird eine neue Analysesicht eingeführt, welche die momentanen Dimmwerte der Scheinwerfer und die daraus resultierenden Lichtverteilungen in Echtzeit visualisiert. Abhängig von der Auflösung des Systems existieren zwei Varianten dieser Dimmwertanalyse, welche in den Bildern 7-2 und 7-3 abgebildet sind. Neben der Analyse des Scheinwerferlichts können lichtrelevante Werte der Fahrzeug- und Umfeldsensorik ebenfalls visualisiert werden.

Während die Analysen bei der Erfassung des Ist-Zustands unterstützen, folgen in Abschnitt 7.2 Entwurfswerkzeuge. Zur Realisierung dieser Werkzeuge wurde in Abschnitt 7.2.1 eine Methodik entwickelt, die wesentliche Teile der Lichtsteueralgorithmen automatisiert generiert, indem eine Abstraktionsschicht zwischen der Vielzahl von Lichtquellen des Scheinwerfers und den Vorgaben des Lichtingenieurs geschaffen wird. Diese Abstraktionsschicht reduziert einerseits die Dimension der Stellgrößen erheblich und führt auf diese Weise zur Beherrschbarkeit der hohen Lichtquellenzahl. Andererseits abstrahiert er das konkrete HD-System und erlaubt damit die Überführung bereits definierter Lichtfunktionen auf andere Scheinwerfersysteme. Es werden zwei Implementierungen der Methodik vorgestellt. Der erste, lokale Ansatz ist echtzeitfähig und eignet sich in der vorgestellten Form zur Implementierung auf einem Steuergerät. Im Gegensatz zum zweiten Ansatz, welcher auf einer globalen Optimierung beruht, führt er im Allgemeinen jedoch zu suboptimalen Ergebnissen. Für den globalen Ansatz wird das quadratische Optimierungsproblem in Gleichung (7-11) formuliert und dessen Konvexität nachgewiesen. Diese stellt sicher, dass der Optimierer im globalen Optimum konvergiert. Zur Implementierung auf einem Steuergerät eignet sich das globale Verfahren ohne vorherige Anpassungen und Vereinfachungen jedoch nicht.

Schließlich wird Hyperion entlang der zuvor aufgestellten Anforderungen **A1** bis **A13** evaluiert. Die Evaluierungsergebnisse werden in Tabelle 8-1 zusammengefasst. Als einzige Anforderung muss **A9**, die Witterungssimulation, als „teilweise erfüllt“ eingestuft werden, da bisher nur Nebel modelliert wurde. Die Modellierung sämtlicher Witterungsphänomene hätten den wissenschaftlichen Kern der vorliegenden Arbeit verfehlt. Im Übrigen zeigt sich, dass die restlichen Anforderungen als weitestgehend oder vollständig erfüllt betrachtet werden können. Das gilt insbesondere für die Anforderungen **A2** bis **A8**, welche den wissenschaftlichen Kern der Arbeit umreißen.

## 9.2 Ausblick

Im Verlauf der Bearbeitung der wissenschaftlichen Fragestellung haben sich verschiedene Stoßrichtungen ergeben, deren weitere Verfolgung zu gewinnbringenden Ergebnissen führen könnte. Zum Abschluss der Arbeit soll dieser Ausblick eine Motivation zur Weiterentwicklung der Nachtfahrtsimulation „Hyperion“ liefern.

Bisher wird zur Bestimmung der Farbwerte aller Pixel auf ein physikalisch basiertes Beleuchtungsmodell der Unity-Engine zurückgegriffen. Der breite Einsatz in VR-Anwendungen beweist die hohe Qualität dieses Beleuchtungsmodells. Es eignet sich für verschiedenste Beleuchtungskonstellationen und ist somit universell einsetzbar. In der vorliegenden Arbeit sind im Vergleich dazu nur wenige Beleuchtungskonstellationen von besonderem Interesse. Zum einen trifft das Scheinwerferlicht meist unter einem sehr flachen Winkel auf die beleuchteten Flächen. Außerdem sind die Beschaffenheiten dieser Flächen oft detailliert beschreibbar. So könnte die Integration dedizierter Beleuchtungsmodelle für einzelne häufig auftretenden Materialien zu einer weiteren Qualitätssteigerung der visuellen Ausgabe führen. Ein Beleuchtungsmodell für Asphalt wäre ein konkretes Beispiel. Diese dedizierten Beleuchtungsmodelle könnten dann mit größerer Komplexität modelliert werden, als es für ein universelles Beleuchtungsmodell der Unity-Engine unter Echtzeitanforderungen zulässig wäre.

Die vorliegende Arbeit beschreibt die Berechnung der Lichteinflüsse durch die klassischen Methoden der Rastergrafik (Per-Fragment-Lighting). Gleichzeitig wird herausgestellt, dass die vorgestellte Implementierung zur Gesamtlichtverteilungsberechnung, welche den wissenschaftlichen Kern der Arbeit bildet, davon entkoppelt ist und sich in gleicher Weise für die Anwendung eines Ray Tracers eignet. Die Ersetzung der klassischen Vertex und Fragment Shader des Lichtquellenmodells durch einen Ray Tracing Ansatz erscheint insofern vielversprechend. Die nun am Markt erhältliche Hardware ließe eine echtzeitfähige Implementierung eines Ray Tracers für die Scheinwerfer zu. Durch die damit mögliche Berücksichtigung von Mehrfachreflexion ist mit einer Verbesserung der grundsätzlichen visuellen Ausgabe und insbesondere der Witterungssimulation zu rechnen.

Eine dritte Erweiterungsmöglichkeit der Nachtfahrtsimulation „Hyperion“ ergibt sich durch die zunehmende Berücksichtigung der Umfeldsituation in neuartigen Lichtfunktionen. Im Abschnitt 3.2 wurden im Kontext fortgeschrittener Fahrerassistenzsysteme und des autonomen Fahrens Lichtfunktionen vorgestellt, deren Verhalten maßgeblich durch die Fahrzeugumgebung bestimmt wird. Im Rahmen der vorliegenden Arbeit stand die Umfeldsensorik nicht im Fokus, weshalb bislang nur eine rudimentäre Implementierung einer Umfeldkamera existiert. Die detaillierte Modellierung des Kamerasensors und die Ergänzung von Sensorsimulationen für LiDAR- und RADAR-Systeme würde die ganzheitliche Erprobung des Scheinwerfersystems von der Sensorwahrnehmung über die Datenverarbeitung bis hin zur Lichtdarstellung erlauben. Ein derart umfassendes Werkzeug stellt einen hohen Mehrwert für die Entwicklung zukünftiger Sensor- und Scheinwerfersysteme dar. Es soll im BMWi-geförderten Verbundprojekt „Robuste Sensorik für hochautomatisiertes Fahren“ (RoSSHAF) durch die Kooperation aus dem Heinz Nixdorf Institut, dem Fraunhofer IEM, der HELLA GmbH & Co. KGaA, der dSPACE GmbH, der Smart Mechatronics GmbH und der RTB GmbH & Co. KG erarbeitet werden.

# 10 Literaturverzeichnis


[Adv20] Advanced Micro Devices: *AMD Ryzen Threadripper*. 2020. https://www.amd.com/de/products/ryzen-threadripper (besucht am 08. 12. 2020)

[AFKN95] Apteker, R. T.; Fisher, J. A.; Kisimov, V. S.; Neishlos, H.: Video Acceptability and Frame Rate. *IEEE MultiMedia* 2 (1995), Nr. 3, S. 32–40

[AFS+13] Akeley, K.; Foley, J.; Sklar, D.; McGuire, M.; Hughes, J.; van Dam, A.; Feiner, S.: *Computer Graphics*. 3rd edition. Addison-Wesley Professional, 2013

[AHW+16] Ansorg, P.; Hager, J.; Woisetschläger, O.; Seitz, M.; Stötzler, A.; Jahn, P.; Mitterlehner, T.; Reisinger, B.; Bemmer, C.: Advantages of laser scanners for automotive headlamps. *SIA VISION* (2016)

[ANS01] ANSI/IES LM-63-19:2019. Aufl., *Approved Method: IES Standard File Format for the Electronic Transfer of Photometric Data and Related Information*

[Ans20] Ansys: *VRXPERIENCE*. 2020. https://www.ansys.com/de-de/products/systems/ansys-vrxperience (besucht am 29. 09. 2020)

[ASU20] ASUSTek Computer: *Rog Swift 360 Hz PG259QN*. 2020. https://rog.asus.com/de/monitors/23-to-24-5-inches/rog-swift-360hz-pg259qn-model/spec/ (besucht am 08. 12. 2020)

[AVS20] AVSimulation: *SCANeR Studio*. 2020. https://www.avsimulation.com/scanerstudio/ (besucht am 29. 09. 2020)

[BB06] Bender, M.; Brill, M.: *Computergrafik: Ein anwendungsorientiertes Lehrbuch*. 2., überarb. Aufl. München: Hanser, 2006

[BB15] Bhakta, V. R.; Ballard, B.: High resolution adaptive headlight using Texas Instruments DLP technology. *ISAL* (2015)

[Bli77] Blinn, J. F.: Models of light reflection for computer synthesized pictures. *SIGGRAPH* (1977), S. 192–198

[BN19] Budanow, M.; Neumann, C.: Success of Driver Assistance through Light Projections on the Road. *ISAL* (2019)

[Böh19] Böhm, G.: Future of Automotive Headlamps - Light for Sensors. *ISAL* (2019)

[Boo01] Boor, C. de: *A practical guide to splines*. Rev. ed. Applied mathematical sciences. New York: Springer-Verlag, 2001

[Bou70] Bouknight, J.: A procedure for generation of three-dimensional half-toned computer graphics presentations. *Communications of the ACM* 13 (1970), S. 527–536

[Bre65] Bresenham, J. E.: Algorithm for computer control of a digital plotter. *IBM Systems Journal* 4 (1965), Nr. 1, S. 25–30

[BS75] Blevin, W. R.; Steiner, B.: Redefinition of the Candela and the Lumen. *IOP Publishing* 11 (1975), S. 97–104

[CB78] Cyrus, M.; Beck, J.: Generalized two- and three-dimensional clipping. *Computers & Graphics* (1978), S. 23–28

[CCRP19] Cladé, S.; Courcier, M.; Roels, S.; Pellarin, M.: 4K Pixel Solid State Glare Free High Beam. *ISAL* (2019)

[CIE19] CIE: *The basis of physical photometry*. 3rd edition. Bd. Technical report. Technical report / CIE. Vienna: CIE Central Bureau, 2019

[DF17] Duhme, D.; Fischer, B.: Next Generation LCD Module. *ISAL* (2017)

[DHB19] Dupius, M.; Hekele, E.; Biehn, A.: *OpenDRIVE Format Specification, Rev. 1.5*. 2019

[DIN01] DIN 5031:1982-03, *Strahlungsphysik im optischen Bereich und Lichttechnik*

[DIN02] DIN 13032-1:2012-06, *Licht und Beleuchtung - Messung und Darstellung photometrischer Daten von Lampen und Leuchten*

[DIN03] DIN EN ISO 80000:2013. Aufl., *Größen und Einheiten*

[DIN04] DIN 5033-1:2017-10, *Farbmessung*

[DIN05] DIN EN ISO 11664-1:2011-07, *CIE farbmetrische Normalbeobachter*

[dSP17] dSPACE: *MotionDesk Scene Animation Version 4.1*. 2017

[dSP20] dSPACE: *User Documentation Portal*. 2020. https://www.dspace.com/de/gmb/home/support/documentation.cfm

[ECMA01] ECMA 404:2017. Aufl., *The JSON Data Interchange Syntax*

[End01] Enders, M.: Pixellight. *PAL Conference* (2001)

[Epi20] Epic Games, I.: *Unreal Engine Website*. 2020. https://www.unrealengine.com (besucht am 06. 04. 2020)

[EUM16] Eilertsen, G.; Unger, J.; Mantiuk, R. K.: Evaluation of Tone Mapping Operators for HDR Video. *High Dynamic Range Video*. Elsevier Science and Technology Books, Inc, 2016, S. 185–207

[Evo13] Evolue Technologies: *SuperSplines: Unity3D Asset*. 2013. https://assetstore.unity.com/packages/tools/level-design/superspline s-2020 (besucht am 22. 06. 2020)

[FB18] Farris, J.; Ballard, B.: Trends in High Resolution Headlamps. *SIA VISION* (2018)

[Fec18] Fechner, T.: Camera Based Lost Cargo Detection for Automated Driving. *VISION* (2018)

[FK03] Fernando, R.; Kilgard, M. J.: *The Cg tutorial*. Boston, Mass.: Addison-Wesley, 2003

[Gau15] Gausemeier, S.: *Test- und Trainingsumgebung für fortgeschrittene Fahrerassistenzsysteme*. 2015. https://www.hni.uni-paderborn.de/rtm/forschung/fahrerassistenzsysteme/traffis/ (besucht am 01. 10. 2020)

[GCN16] Gut, C.; Cristea, J.; Neumann, C.: High-resolution headlamp. *Advanced Optical Technologies* 5 (2016), Nr. 2, S. 109–116

[Geb03] Gebhardt, N.: *Einige BRDF Modelle*. 2003. http://www.irrlicht3d.org/papers/BrdfModelle.pdf (besucht am 16. 11. 2020)

[Ger11] Gershenfeld, N.: *The nature of mathematical modeling*. Cambridge: Cambridge Univ. Press, 2011

[GID18] GIDAS: *German in-depth accident study: data from 2000-2018, Interner Bereich*. 2018. http://www.gidas.org (besucht am 09. 09. 2020)

[Gla07] Glassner, A.: *An introduction to ray tracing*. Transferred to digital print. San Francisco, Calif.: Morgan Kaufmann, 2007

[GPL+15] Grötsch, S.; Pfeuffer, A.; Liebetrau, T.; Oppermann, H.; Brink, M.; Fiederling, R.; Möllers, I.; Moisel, J.: Integrated High Resolution LED Light Sources in an AFS/ADB Headlamp. *ISAL* (2015)

[GR17] Gausemeier, S.; Rüddenklau, N.: *Smart Headlamp Technology*. 2017. https://www.hni.uni-paderborn.de/rtm/forschung/fahrerassistenzsysteme/smart-headlamp-technology/ (besucht am 01. 12. 2020)

[GXB19] Gut, C.; Xilu, Z.; Boeke, B.: DIGITAL LIGHT – The Future Light Distribution for Automated Vehicles. *ISAL* (2019)

[Ham19] Hamm, M.: Real Driving Benefits and Research Findings with Digital Light Functions. *ISAL* (2019)

[Hen82] Hentschel, H.-J.: *Licht und Beleuchtung: Theorie und Praxis der Lichttechnik*. 2., vollkommen überarb. Aufl. Heidelberg: Hüthig, 1982

[Hes15] Hesse, H.: BMBF-Project VOLIFA 2020 - High resolution light distribution by using a LCD. *ISAL* (2015)

[HH75] Hammersley, J. M.; Handscomb, D. C.: *Monte Carlo methods*. London: Methuen, 1975

[HSB+17] Hager, J.; Seitz, M.; Bemmer, C.; Jahn, P.; Ansorg, P.; Woisetschläger, O.; Buchmann, F.; Sprenger, D.; Vogl, M.; Hering, O.; Artzner, J.; Glück, M.; Reisinger, B.; Mitterlehner, T.; Schantl, P.; Weber, L.; Berlitz, S.; Neumann, C.: Handling 17W of scanning laser power – three years of exploration in the iLaS project. *ISAL* (2017)

[Hun98] Hunt, R. W. G.: *Measuring colour*. 3rd ed. Kingston-upon-Thames: Fountain, 1998

[ICG86] Immel, D. S.; Cohen, M. F.; Greenberg, D. P.: A radiosity method for non-diffuse environments. *SIGGRAPH* (1986)

[Inf20] Information und Technik Nordrhein-Westfalen: *Geobasisdaten*. 2020. https://www.opengeodata.nrw.de/produkte/geobasis/ (besucht am 18. 06. 2020)

[ISO01] ISO/IEC 21778:2017. Aufl., *Information technology — The JSON data interchange syntax*

[Jäh05] Jähne, B.: *Digital Image Processing*. 6th revised and extended ed. Berlin und Heidelberg: Springer, 2005

[Kaj86] Kajiya, J. T.: The rendering equation. *SIGGRAPH* (1986), S. 143–150

[KBR17] Kessenich, J.; Baldwin, D.; Rost, R.: *The OpenGL Shading Language: Language Version: 4.50*. 2017. https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.50.pdf (besucht am 07. 04. 2020)

[KH84] Kajiya, J. T.; Herzen, B. P. V.: Ray tracing volume densities. *SIGGRAPH* (1984)

[KHKL19] Knöchelmann, M.; Held, M. P.; Kloppenburg, G.; Lachmayer, R.: High-resolution headlamps - technology analysis and system design. *Advanced Optical Technologies* 8 (2019), Nr. 1, S. 33–46

[Khr19] Khronos OpenCL Working Group: *The OpenCL Specification*. 2019. https://www.khronos.org/registry/OpenCL/specs/2.2/pdf/OpenCL_API.pdf (besucht am 06. 04. 2020)

[KKBK17] Kobbert, J.; Kosmas, K.; Bursasiu, D.; Khanh, T. Q.: High Beam Optimization for Low Resolution Glare Free High Beam. *ISAL* (2017)

[Kle14] Kleinkes, M.: LED-Matrix-Systems - Revolution and Evolution. *VISION* (2014)

[KPW19] Kleinkes, M.; Pohlmann, W.; Wilks, C.: Boost Safety & Styling - New HD-LED Systems for front and rear. *ISAL* (2019)

[KTW+19] Krieft, F.; Thoma, A.; Willeke, B.; Kubitza, B.; Kaup, M.: Symbol Projection: Gain or Gadget? *ISAL* (2019)

[KW18] Kubitza, B.; Wilks, C.: Digital Light as Support for the Driver. *ATZ* 120 (2018), S. 54–57

[Lee19] Lee, H.: The Study of Functionality for Now and Future High Definition Lighting. *ISAL* (2019)

[LKVZ11] Lim, H.-T.; Krebs, B.; Volker, L.; Zahrer, P.: Performance evaluation of the inter-domain communication in a switched Ethernet based in-car network. *IEEE 36th Conference on Local Computer Networks, Bonn* (2011), S. 101–108

[LLT+19] Li, R.; Liu, Q.; Tang, Z.; Feng, J.; Deng, L.; Fan, J.; Zheng, Z.; Chen, W.; Cao, D.; Zhiming, Y.: The photo-biological safety study of phosphor converted white laser diode applied in automotive lighting. *ISAL* (2019)

[Mic18] Microsoft: *DirectX graphics and gaming: Part of Microsoft Docs*. 2018. https://docs.microsoft.com/en-us/windows/win32/directx (besucht am 12. 04. 2020)

[Mic20] Microsoft: *Xbox Series X Spezifikation*. 2020. https://www.xbox.com/de-DE/consoles/xbox-series-x#specs (besucht am 08. 12. 2020)

[MMFG16] Möllers, I.; Moisel, J.; Fiederling, R.; Grötsch, S.: An efficient and high-resolution ADB headlamp based on micro-integrated LED arrays. *VDI - Optische Technologien in der Fahrzeugtechnik* (2016)

[MMH15] Maier, M.; Moisel, J.; Herold, F.: Multibeam-Scheinwerfer in der Mercedes Benz CLS-Klasse. *ATZ* 2 (2015), S. 17–20

[MR18] Mathes, J.; Reiss, B.: Merging Lighting and Sensing: a new path toward more automated vehicles. *SIA VISION* (2018)

[MSC20] MSC Software: *ADAMS*. 2020. https://www.mscsoftware.com/product/adams (besucht am 23. 06. 2020)

[NRH+77] Nicodemus, F. E.; Richmond, J. C.; Hsia, J. J.; Ginsberg, I. W.; Limperis, T.: *Geometrical Considerations and Nomenclature for Reflectance*. U.S. Department of Commerce - National Bureau of Standards, 1977

[Nvi07] Nvidia Corporation: *CUDA C++ Programming Guide*. 2007. https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (besucht am 06. 04. 2020)

[Nvi20] Nvidia Corporation: *GEFORCE RTX 3090*. 2020. https://www.nvidia.com/de-de/geforce/graphics-cards/30-series/rtx-3090/ (besucht am 08. 12. 2020)

[OSM20a] OSM Foundation: *Java OpenStreetMap Editor*. 2020. https://josm.openstreetmap.de/ (besucht am 22. 06. 2020)

[OSM20b] OSM Foundation: *OpenStreetMap*. 2020. https://www.openstreetmap.org (besucht am 22. 06. 2020)

[PB12] Pacejka, H. B.; Besselink, I.: *Tire and vehicle dynamics*. 3rd ed. Amsterdam und Boston: Elsevier/BH, 2012

[PEA20] PEAK-System Technik GmbH: *CAN-Interface für USB*. 2020. https://www.peak-system.com/PCAN-USB.199.0.html (besucht am 03. 07. 2020)

[PH11] Pfeffer, P.; Harrer, M.: *Lenkungshandbuch: Lenksysteme, Lenkgefühl, Fahrdynamik von Kraftfahrzeugen*. Praxis ATZ/MTZ-Fachbuch. Wiesbaden: Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden GmbH, Wiesbaden, 2011

[PHG+15] Petersen, A.; Hager, J.; Gut, C.; Jahn, P.; Seitz, M.; Schwaiger, S.; Schlöder, U.; Helmer, M.; Berlitz, S.; Neumann, C.; Hering, O.: Challenges for MEMS based Scanning Laser System. *ISAL* (2015)

[Pho75] Phong, B. T.: Illumination for computer generated pictures. *Communications of the ACM* 18 (1975), Nr. 6, S. 311–317

[Pit67] Pitteway, M. L. V.: Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotter. *The Computer Journal* 10 (1967), Nr. 3, S. 282–289

[Pla12] Plattfaut, C.: Using a night time driving simulation for the development of camera controlled light algorithms. *Ilisisis Symposium* (2012)

[PPBS05] Pedrotti, F.; Pedrotti, L.; Bausch, W.; Schmidt, H.: *Optik für Ingenieure: Grundlagen ; mit 28 Tabellen*. 3., bearb. und aktualisierte Aufl. Berlin, Heidelberg und New York: Springer, 2005

[Ree62] Reeb, O.: *Grundlagen der Photometrie*. Karlsruhe: Braun, 1962

[Ric81] Richter, M.: *Einführung in die Farbmetrik*. De Gruyter, 1981

[RL19] Rosenhahn, E.-O.; Link, F.: Traffic Safety Benefits provided by High Resolution Headlamp Systems. *ISAL* (2019)

[Ros00] Rosenhahn, E.-O.: *Dissertation: Entwicklung von lichttechnischen Anforderungen an Kraftfahrzeugscheinwerfer für Schlechtwetterbedingungen*. München: Herbert Utz Verlag, 2000

[Ros18] Rosenhahn, E.-O.: New Systems for Safety and Comfort Improvement by High Resolution Flexibility. *SIA VISION* (2018)

[RTH+19] Roth, J.; Thamm, M.; Held, M. P.; Lachmayer, R.; Kleinert, B.: Micro-Pixel-LED-Headlights. *ISAL* (2019)

[Rus20] Russel, J.: *Basic Theory of Physically-Based Rendering*. 2020. https://marmoset.co/posts/basic-theory-of-physically-based-rendering/ (besucht am 28. 09. 2020)

[SA06] Segal, M.; Akeley, K.: *The OpenGL Graphics System: A Specification: Version 4.5 (Core Profile)*. Hrsg. von Frazier, C.; Leech, J.; Brown, P. 2006. https://www.khronos.org/registry/OpenGL/specs/gl/glspec45.core.pdf (besucht am 06. 04. 2020)

[Sch07] Schanda, J.: *Colorimetry: Understanding the CIE system / edited by János Schanda*. Hoboken, N.J.: Wiley und Chichester, 2007

[Sch17] Schmidt, C.: Adverse Weather Light: New Chances for a technically unsolved problem. *ISAL* (2017)

[Sel07] Selimovic, I.: Beiträge zu globalen Fragen in der NURBS-Technik. *Dissertation, Fachbereich Mathematik, Universität Dortmund* (2007), S. 7–8

[SHSN18] Stroop, P.; Hilling, B.; Steudel, F.; Nolte, P. W.: Laser scanning lighting systems - optimization of contrast and further challenges. *SIA VISION* (2018)

[Son20] Sony Interactive Entertainment: *PlayStation 5*. 2020. https://www.playstation.com/de-de/ps5/ (besucht am 08. 12. 2020)

[SP18] Scheffels, G.; Prawitz, S.: *Von der Glühlampe zur Matrix-LED*. 2018. https://www.automobil-industrie.vogel.de/von-der-gluehlampe-zur-matrix-led-a-712366/ (besucht am 21. 04. 2020)

[SR17] Schneider, J.; Rautek, P.: A Versatile and Efficient GPU Data Structure for Spatial Indexing. *IEEE Transactions on Visualization and Computer Graphics* (2017)

[SSS08] Szirmay-Kalos, L.; Szécsi, L.; Sbert, M.: *GPU-based techniques for global illumination effects*. Synthesis lectures in computer graphics and animation. San Rafael, Calif.: Morgan & Claypool, 2008

[Ste57] Stevens, S. S.: On the psychophysical law. *Psychological review* 64 (1957), Nr. 3, S. 153–181

[Syn19] Synopsys: *Press Release: Synopsys Announces New Release of LucidShape Software for Automotive Lighting Design and Analysis*. 2019. https://news.synopsys.com/2019-09-17-Synopsys-Announces-New-Release-of-LucidShape-Software-for-Automotive-Lighting-Design-and-Analysis (besucht am 08. 05. 2020)

[Syn20] Synopsys: *Information zum Tool LucidDrive aus der Produktfamilie LucidShape der Firma Synopsys*. 2020. https://www.synopsys.com/optical-solutions/lucidshape/luciddrive.html (besucht am 08. 05. 2020)

[Tem19] Templeton, B.: *Elon Musk's War On LIDAR: Who Is Right And Why Do They Think That?* 2019. https://www.forbes.com/sites/bradtempleton/2019/05/06/elon-musks-war-on-lidar-who-is-right-and-why-do-they-think-that/#1e84a8162a3b (besucht am 29. 05. 2019)

[The14] The Khronos Vulkan Working Group: *Vulkan 1.0.136 - A Specification*. 2014. https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html (besucht am 06. 04. 2020)

[Tho21] Thoma, A.: *Dissertation: Optimierung der Scheinwerferlichtverteilung zur Verbesserung der Sichtverhältnisse in nächtlichen Schlechtwettersituationen*. Braunschweig: https://doi.org/10.24355/dbbs.084-202106041025-0, 2021

[TU09] Tóth, B.; Umenhoffer, T.: Real-time Volumetric Lighting in Participating Media. *EUROGRAPHICS* (2009)

[TV19] Thoma, A.; Vollrath, M.: Adverse Weather Light – New Approaches to Evaluate Adaptive Light Functions. *ISAL* (2019)

[Uli88] Ulichney, R. A.: Dithering with blue noise. *Proceedings of the IEEE* 76 (1988), Nr. 1

[Uni20] Unity Technologies: *Unity Learn*. 2020. https://learn.unity.com/ (besucht am 20. 05. 2020)

[VIR20] VIRES: *OpenDRIVE Website*. 2020. http://www.opendrive.org/index.html (besucht am 22. 06. 2020)

[Vis20] VisCircle GmbH: *Unreal Engine vs. Unity: welche Software sollten Gamedeveloper wählen?* 2020. https://viscircle.de/unreal-engine-vs-unity-welche-software-sollten-gamedeveloper-waehlen/ (besucht am 18. 05. 2020)

[Wik20] Wikipedia: *Leuchtdichte*. 2020. https://de.wikipedia.org/wiki/Leuchtdichte (besucht am 20. 02. 2020)

[WP01] Weber, T.; Plattfaut, C.: Interaktiver Simulator für Nachtfahrten am PC. *ATZ* 103 (2001), S. 1058–1061

[WPKB02] Weber, T.; Plattfaut, C.; Kleinkes, M.; Berssenbrügge, J.: Virtual Nightdrive. *Driving Simulation Conference* (2002)

[WS82] Wyszecki, G.; Stiles, W. S.: *Colour science: Concepts and methods, quantitative data and formulae / Günter Wyszecki, W.S. Stiles*. 2nd ed. Wiley series in pure and applied optics. New York und Chichester: Wiley, 1982

[YL14] Yin, M.; Li, S.: Fast BVH construction and refit for ray tracing of dynamic scenes. *Multimedia Tools and Applications* 72 (2014)

## Literaturverzeichnis der studentischen Arbeiten

Diese Dissertation wurde von verschiedenen studentischen Arbeiten begleitet, von denen einige im Rahmen der Dissertationsschrift keinen Beitrag geliefert haben. Zwei dieser studentischen Arbeiten haben die Simulationsumgebung Hyperion um weitere hier nicht thematisierte Komponenten erweitert [Sim19],[Sol21]. In anderen Arbeiten wurde Hyperion zur simulationsgestützten Untersuchung anderer wissenschaftlicher Fragestellungen verwendet [Pet21],[Hin22].

Die Definition der Zielsetzung, die Erarbeitung von Lösungsansätzen sowie die Auswertung, Interpretation und Visualisierung der in den Arbeiten hervorgebrachten Ergebnisse erfolgte unter wissenschaftlicher Anleitung des Autors dieser Dissertationsschrift.


[Goh18] Gohar, A.; Rüddenklau, N. (BetreuerIn): *Modellierung einer virtuellen Sensorik zur HiL-Simulation eines Scheinwerfer-Steuergerätes*. Unveröffentlichte Masterarbeit. Fakultät für Maschinenbau, Universität Paderborn, 2018

[Hin22] Hinzmann, S.; Thoma, A. (BetreuerIn); Rüddenklau, N. (BetreuerIn): *Untersuchung des Einflusses individueller Merkmale auf die Geschwindigkeitswahrnehmung beim Führen eines Kraftfahrzeuges im Nebel*. Unveröffentlichte Masterarbeit. Fakultät für Sozial- und Verhaltenswissenschaften, Friedrich-Schiller-Universität Jena, 2022

[Pet21] Peters, D.; Rüddenklau, N. (BetreuerIn): *Entwicklung adaptiver Scheinwerferlichtverteilungen für unterschiedliche Straßenbeleuchtungssituationen*. Unveröffentlichte Masterarbeit. Fakultät für Maschinenbau, Universität Paderborn, 2021

[Sim19] Simon, C.; Biemelt, P. (BetreuerIn); Rüddenklau, N. (BetreuerIn): *Integration akustischer Bewegungshinweise zur Immersionssteigerung in der interaktiven Fahrsimulation*. Unveröffentlichte Projektarbeit. Fakultät für Maschinenbau, Universität Paderborn, 2019

[Sol21] Soltani, O.; Brodehl, C. (BetreuerIn); Rüddenklau, N. (BetreuerIn): *Prozessierung und Auswertung von Messdaten eines Radars mit synthetischer Apertur*. Unveröffentlichte Bachelorarbeit. Fakultät für Maschinenbau, Universität Paderborn, 2021

# Anhang

## Inhaltsverzeichnis

# A1 Definitionsdateien

## A1.1 Parameterdatei für das interne Fahrzeugmodell

```
{
  "CreatedOn": "Montag, 30. September 2019",
  "Name": "ExampleCar",
  "Version": "1.0",
  "Mass": 950.0,
  "CenterOfMass": [0.0, 0.35, 0.2],
  "LeftHeadlight": [-0.52, 0.78, 1.08],
  "RightHeadlight": [0.52, 0.78, 1.08],
  "HeadPosition": [-0.3941, 1.1448, 0.04],
  "EnvironmentCameraPosition": [0.0, 1.2916, 0.556],
  "FrontSpringConstant": 26000.0,
  "RearSpringConstant": 18000.0,
  "FrontTargetPosition": 0.4,
  "FrontDamperConstant": 1200.0,
  "RearDamperConstant": 950.0,
  "RearTargetPosition": 0.4,
  "FrontSuspensionMass": 40.0,
  "RearSuspensionMass": 40.0,
  "WheelRadius": 0.32,
  "SuspensionDistance": 0.2,
  "ForceAppDistance": 0.3,
  "ModelHasSteeringWheel": true,
  "ModelHasKmhNeedle": true,
  "ModelHasRpmNeedle": true,
  "IsFrontWheelDrive": true,
  "IsRearWheelDrive": false,
  "IsFrontSteering": true,
  "IsRearSteering": false,
  "HasEnvironmentCamera": true,
  "WheelBase": 1.7,
  "OverrideEnvCameraPos": false,
  "OverrideHeadlightPos": false,
  "InertiaRadius": {
    "x": 0.64,
    "y": 1.13,
    "z": 1.15
  },
  "FrontSurfaceArea": 2.0,
  "Cw": 0.3,
  "TrackWidth": 1.65
}
```

Listing A1.1: *JSON-Definition des internen Fahrzeugmodells*

## A1.2 Definition der Fahrzeugtrajektorien des internen Autopiloten

```
<?xml version="1.0" encoding="utf-8"?>
<maneuver xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="maneuvers.xsd">
   <routes>
      <route ids="-1,0,4,1,3,12" minDistance="30.0">
         <roads>
            <road id="01" direction="forward" >
               <laneOffsets>
                  <laneOffset s="0" a="0" b="0.5" c="0" d="0"/>
                  <laneOffset s="8" a="4" b="0" c="0" d="0"/>
                  <laneOffset s="50" a="4" b="-0.5" c="0" d="0"/>
                  <laneOffset s="58" a="0" b="0" c="0" d="0"/>
               </laneOffsets>
            </road>
            <road id="03" direction="backward" />
            <road id="04" direction="forward" />
            <road id="05" direction="forward" />
         </roads>
      </route>
      <route ids="5,10,11,7,2,6,8,9,13,14" minDistance="60">
         <roads>
            <road id="02" direction="forward" />
            <road id="10" direction="forward" minDistance="30.0" />
         </roads>
      </route>
   </routes>
</maneuver>
```

Listing A1.2: *XML-Definiton der Fahrzeugmanöver für eine gegebene Strecke*

- Ein Manöver (*maneuver*) kann mehrere Routen enthalten.
- Für jede Route (*route*) werden im Feld *ids* die IDs der Fahrzeuge aufgelistet, die auf dieser Route fahren. Fremdfahrzeuge werden durch IDs zwischen 0 und 14 identifiziert. Das Egofahrzeug ist ID -1.
- Jede Route beinhaltet das Feld *minDistance*, welches den Mindestabstand aufeinander folgender Fahrzeuge in Metern vorgibt.
- Eine Route kann sich über mehrere Straßen erstrecken.
- Jede Straße (*road*) ist über das Feld *id* mit der Streckendefinition verknüpft. Die IDs der Manöver- und der Streckendefinition müssen übereinstimmen.
- Innerhalb der Straße kann der Wert des Felds *minDistance*, der durch die Route vorgegeben wurde, optional überschrieben werden.
- Für jede Straße muss definiert werden, ob sie vorwärts oder rückwärts (*direction=„forward/backward“*) befahren werden soll, wobei „vorwärts“ die Fahrt entlang aufsteigender Bogenlänge meint.
- Jede Straße kann keinen, einen oder mehrere Spurversätze (*laneOffset*) enthalten. Dies ist der laterale Abstand, den das Fahrzeug von der Referenzlinie hat. Auf diese Weise können Fahrzeuge auf die Spuren einer Straße verteilt werden.

- Jeder Spurversatz wird als eine Polynomfunktion 3. Grades mit den Koeffizienten $a$ bis $d$ angegeben: $a + b \cdot x + c \cdot x^2 + d \cdot x^3$. Sein Gültigkeitsbereich beginnt bei der Bogenlänge $s$ und endet am Beginn des nächsten Spurversatzes oder dem Ende der Straße. Die Spurversätze müssen entlang des Felds $s$ geordnet sein.

## A1.3 Definition des Streckenverlaufs nach dem OpenDRIVE-Standard

```
<?xml version="1.0" encoding="utf-8"?>
<OpenDRIVE xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance">
  <header revMajor="1" revMinor="4" name="Rundkurs_ModelDesk" date=
      "2018-02-20A10:08:12" vendor="dSPACE GmbH">
    <geoReference><![CDATA[]]></geoReference>
  </header>
  <road name="Road 1" length="713.685" id="183349851" junction="-1"
      >
    <link>
      <predecessor elementType="road" elementId="183349851" />
      <successor elementType="road" elementId="183349851" />
    </link>
    <type s="0" type="unknown">
      <speed max="200" unit="km/h" />
    </type>
    <type s="300" type="unknown">
      <speed max="200" unit="km/h" />
    </type>
    <type s="800" type="unknown">
      <speed max="200" unit="km/h" />
    </type>
    <planView>
      <geometry s="0" x="0" y="0" hdg="0" length="100">
        <line />
      </geometry>
      <geometry s="100" x="100" y="0" hdg="0" length="100">
        <spiral curvStart="0" curvEnd="0.02" />
      </geometry>
      <geometry s="200" x="190.452" y="31.026" hdg="0.999" length="
          57.1">
        <arc curvature="0.02" />
      </geometry>
      <geometry s="257.1" x="190.452" y="85.074" hdg="2.1418"
          length="100">
        <spiral curvStart="0.02" curvEnd="0" />
      </geometry>
      <geometry s="357.1" x="99.993" y="116.082" hdg="-3.141"
          length="100">
        <line />
      </geometry>
      <geometry s="457.1" x="-0.006" y="116.061" hdg="-3.141"
          length="100">
        <spiral curvStart="0" curvEnd="0.02" />
      </geometry>
      <geometry s="557.1" x="-90.452" y="85.015" hdg="-2.141"
          length="57.1">
```

```
40          <arc curvature="0.02" />
41        </geometry>
42        <geometry s="614.2" x="-90.441" y="30.968" hdg="-0.999"
              length="99.485">
43          <paramPoly3 aU="0" bU="95.596" cU="-18.028" dU="-2.618" aV=
                "0" bV="0" cV="97.600" dV="-38.260" pRange="normalized"
                />
44        </geometry>
45      </planView>
46      <elevationProfile>
47        <elevation s="0" a="0" b="0.009" c="0" d="0" />
48        <elevation s="104" a="1" b="0.010" c="0" d="0" />
49        <elevation s="201" a="2" b="0.017" c="0" d="0" />
50        <elevation s="257" a="3" b="-0.009" c="0" d="0" />
51        <elevation s="358" a="2" b="-0.020" c="0" d="0" />
52        <elevation s="457.1" a="0" b="0" c="0" d="0" />
53        <elevation s="563" a="0" b="0" c="0" d="0" />
54        <elevation s="615" a="0" b="0" c="0" d="0" />
55      </elevationProfile>
56      <lateralProfile>
57        <superelevation s="0" a="0" b="0" c="0" d="0" />
58      </lateralProfile>
59      <lanes>
60        <laneOffset s="0" a="-1.75" />
61        <laneSection s="0">
62          <left>
63            <lane id="1" type="driving">
64              <width sOffset="0" a="3.5" b="0" c="0" d="0" />
65              <roadMark type="solid" width="0.25" />
66            </lane>
67          </left>
68          <center>
69            <lane id="0" type="driving">
70              <roadMark type="solid" width="0.25" />
71            </lane>
72          </center>
73        </laneSection>
74      </lanes>
75    </road>
76  </OpenDRIVE>
```

Listing A1.3: *XML-Definition eines Streckenverlaufs nach dem OpenDRIVE-Standard*

# A2 Lichtverteilungen

## A2.1 Messdaten einer Lichtverteilung im IES-Format

```
IESNA:LM-63-1995
[TEST]
[MANUFAC]

[LUMCAT]
[LUMINAIRE]
[LAMPCAT]
[LAMP]

[OTHER] created: Wed Nov 29 09:48:07 2017
[DATE]

TILT=NONE
1 -1 1
8 8 3 2
0 0 0
1 1 0
-14.0 -10.0 -6.0 -2.0
2.0 6.0 10.0 14.0
-28.0 -20.0 -12.0 -4.0
4.0 12.0 20.0 28.0
0 0 0 0 0 0 0 0
0 0 0.1 0.1 0.1 0.1 0 0
0.1 0.2 0.3 0.4 0.4 0.3 0.2 0.1
0.1 0.2 0.3 0.4 0.4 0.3 0.2 0.1
0.1 0.2 0.3 0.4 0.4 0.3 0.2 0.1
0.1 0.2 0.3 0.4 0.4 0.3 0.2 0.1
0 0.1 0.1 0.2 0.1 0.1 0.2 0
0 0 0 0 0 0 0 0
```

Listing A2.1: *Reduzierte Beispieldatei einer Lichtstärkeverteilung im IES-Format*

## A2.2 Messdaten einer Lichtverteilung im CIE-Format

```
1  Pole axis,Y
2  Pole angle,vertical
3  Units,CIE X Y Z,
4  Rows,4,
5  Columns,4,
6  Vertical min,-5
7  Vertical max,5
8  Horizontal min,-10
9  Horizontal max,10
10 1.0,2.0,1.0,1.5,3.0,0.8,1.0,2.0,1.0,1.5,3.0,0.8,
11 1.1,3.0,1.1,1.5,3.2,0.6,2.0,5.0,1.0,1.5,3.0,0.6,
12 1.0,2.0,0.8,1.5,7.1,0.8,1.0,2.0,1.9,1.5,3.0,0.8,
13 1.5,2.0,1.0,1.5,3.0,0.9,4.0,2.0,1.4,1.5,7.0,1.8
```

Listing A2.2: *Reduzierte Beispieldatei einer spektralen Lichtverteilung im CIE-Format*

# A3 Laufzeitanalyse und Speicherbedarf

## A3.1 Generierung von Lichtverteilungsdatensätzen

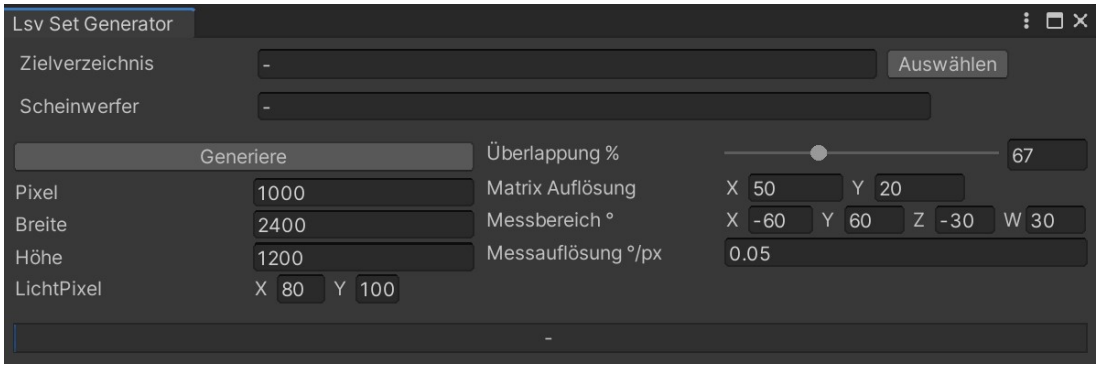


*Bild A3-1*: *Grafische Benutzeroberfläche zur Generierung von Lichtverteilungsdatensätzen*

Bild A3-1 zeigt die Benutzeroberfläche zur Generierung von Lichtverteilungsdatensätzen. Diese Generierung wurde zur Erzeugung der fiktiven Scheinwerfersysteme eingesetzt, welche nachfolgend aufgeführt werden. Zuerst wird das Zielverzeichnis ausgewählt, in welchem das generierte System gespeichert werden soll. Der zweite Eintrag dient zur Benennung des Scheinwerfersystems und ermöglicht eine Identifizierung und Referenzierung in anschließenden Simulationen.

Der rechts darunter befindliche Block erlaubt die technische Beschreibung des Scheinwerfersystems. Das in Bild A3-1 gezeigte Beispiel führt zu einem HD-System mit 1.000 Lichtquellen pro Scheinwerfer. Die Gesamtlichtverteilung, welche sich über einen Winkelbereich von $-60°$ bis $+60°$ in der Horizontalen und $-30°$ bis $+30°$ in der Vertikalen erstreckt, wird durch die Lichtquellen in 50 Spalten und 20 Zeilen segmentiert. Mit einer Überlappung von 67% zwischen benachbarten Lichtquellen, ist diese Segmentierung nicht scharf (s. Abschnitt 6.5.1). Die Winkelbereich der Gesamtlichtverteilung wird in $0,05°$-Schritten diskretisiert.

Im linken unteren Block werden einige resultierende Systemeigenschaften angzeigt. Hierzu gehören die Anzahl der Lichtquellen (1.000), die Anzahl der diskreten Datenpunkte entlang der Horizontalen (2.400) und Vertikalen (1.200), sowie die durch eine Lichtquelle beeinflussten Datenpunkte entlang der Horizontalen (80) und Vertikalen (100).

Ein Klick auf den Button „Generiere“ erzeugt das spezifizierte System. Der Fortschritt wird durch den Ladebalken veranschaulicht, welcher sich am unteren Rand des Bilds A3-1 befindet.

## A3.2 Laufzeitanalyse der Gesamtlichtverteilungsberechnung

*Tabelle A3-1*: *Parameter, Berechnungsdauer und Speicherbedarf der Lichtdatensätze mit variierender Lichtquellenzahl.*

| Lichtquellen | 10 | $10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|---|---|
| Laufzeit (ms) | 1,823 | 1,820 | 1,828 | 1,841 | 2,079 | 2,880 |
| Lichter/Pixel | 2,80 | 2,79 | 2,79 | 2,79 | 2,79 | 2,79 |
| $srcValBuffer(kB)$ | 128.480 | 128.223 | 127.903 | 127.904 | 127.904 | 127.872 |
| $srcIDBuffer(kB)$ | 32.120 | 32.055 | 31.975 | 31.976 | 31.976 | 31.968 |
| $trgBuffer(kB)$ | 45.964 | 45.945 | 45.907 | 45.907 | 45.907 | 45.907 |
| $dimmValBuffer(kB)$ | 0,08 | 0,8 | 8 | 80 | 800 | 8.000 |
| Abdeckung (%) | 100,00 | 99,96 | 99,88 | 99,87 | 99,87 | 99,87 |
| Laufzeit (konst. Dimmwerte) (ms) | 1,823 | 1,820 | 1,828 | 1,834 | 1,986 | 2,076 |

*Tabelle A3-2*: *Parameter, Berechnungsdauer und Speicherbedarf der Lichtdatensätze mit variierender Datenpunktzahl.*

| Datenpunkte | 160 | 1.440 | 4.000 | 7.840 | 12.960 |
|---|---|---|---|---|---|
| $[\underline{\theta}, \bar{\theta}]$ | $[-10°, 10°]$ | $[-30°, 30°]$ | $[-50°, 50°]$ | $[-70°, 70°]$ | $[-90°, 90°]$ |
| $[\underline{\varphi}, \bar{\varphi}]$ | $[-10°, 10°]$ | $[-30°, 30°]$ | $[-50°, 50°]$ | $[-70°, 70°]$ | $[-90°, 90°]$ |
| Threads/Gruppe | 64 | 64 | 64 | 128 | 256 |
| Gruppen | 2.500 | 22.500 | 62.500 | 61.250 | 50.625 |
| Laufzeit (ms) | 0,148 | 0,917 | 2,517 | 4,916 | 8,100 |
| Lichter/Pixel | 2,714 | 2,789 | 2,801 | 2,808 | 2,790 |
| $srcValBuffer(kB)$ | 6.843 | 63.936 | 178.916 | 351.785 | 577.727 |
| $srcIDBuffer(kB)$ | 1.710 | 15.984 | 44.729 | 87.946 | 144.431 |
| $trgBuffer(kB)$ | 2.521 | 22.924 | 63.872 | 125.260 | 207.072 |
| Abdeckung (%) | 99,50 | 99,83 | 100,00 | 100,00 | 99,97 |

*Tabelle A3-3*: *Parameter, Berechnungsdauer und Speicherbedarf der Lichtdatensätze mit variierender Überlappung.*

| Überlappung (%) | 0 | 40 | 80 | 120 | 160 | 200 |
|---|---|---|---|---|---|---|
| Lichter/Pixel | 1,00 | 1,96 | 3,23 | 4,87 | 6,78 | 9,02 |
| Laufzeit (ms) | 1,095 | 1,480 | 2,034 | 2,851 | 4,248 | 6,276 |
| $srcValBuffer(kB)$ | 46.022 | 89.967 | 148.504 | 223.744 | 311.850 | 414.374 |
| $srcIDBuffer(kB)$ | 11.505 | 22.491 | 37.126 | 55.936 | 77.962 | 103.593 |
| $trgBuffer(kB)$ | 45.964 | 45.945 | 45.945 | 45.964 | 45.964 | 45.964 |
| Abdeckung (%) | 100,00 | 99,96 | 99,96 | 100,00 | 100,00 | 100,00 |

## A3.3 Laufzeitanalyse der Szenenbeleuchtung

*Tabelle A3-4*: *Parameter und Berechnungsdauer der Messreihe mit variierendem Messbereich.*

| **Messbereich** | | | | | | | |
|---|---|---|---|---|---|---|---|
| Horizontal(°) | ±15 | ±22,5 | ±30 | ±37,5 | ±45 | ±52,5 | ±60 |
| Vertikal(°) | ±10 | ±15 | ±20 | ±25 | ±30 | ±35 | ±40 |
| Winkelfläche (°°) | 600 | 1.350 | 2.400 | 3.750 | 5.400 | 7.350 | 9.600 |
| Datenpunkte($\cdot 10^6$) | 0,24 | 0,54 | 0,96 | 1,50 | 2,16 | 2,94 | 3,84 |
| Abdeckung (%) | 11,1% | 25,5% | 46,6% | 74,6% | 98,4% | 100,0% | 100,0% |
| Laufzeit (ms) | 0,047 | 0,116 | 0,214 | 0,361 | 0,508 | 0,519 | 0,519 |

*Tabelle A3-5*: *Parameter und Berechnungsdauer der Messreihe mit variierender Messauflösung.*

| Messauflösung (°) | 0,02 | 0,04 | 0,06 | 0,08 | 0,10 |
|---|---|---|---|---|---|
| Datenpunkte ($\cdot 10^6$) | 11,25 | 2,81 | 1,25 | 0,703 | 0,450 |
| Laufzeit (ms) | 1,242 | 0,572 | 0,408 | 0,378 | 0,373 |

*Tabelle A3-6*: *Parameter und Berechnungsdauer der Messreihe mit variierender Ausgabeaufloesung.*

| Bezeichnung | HD Ready | Full HD | Quad HD | Ultra HD |
|---|---|---|---|---|
| Pixel Horiz. | 1.280 | 1.920 | 2.560 | 3.840 |
| Pixel Vert. | 720 | 1.080 | 1.440 | 2.160 |
| Pixelzahl ($\cdot 10^6$) | 0,921 | 2,074 | 3,686 | 8,294 |
| Laufzeit (ms) | 0,171 | 0,370 | 0,680 | 1,580 |

## A4 Validierung

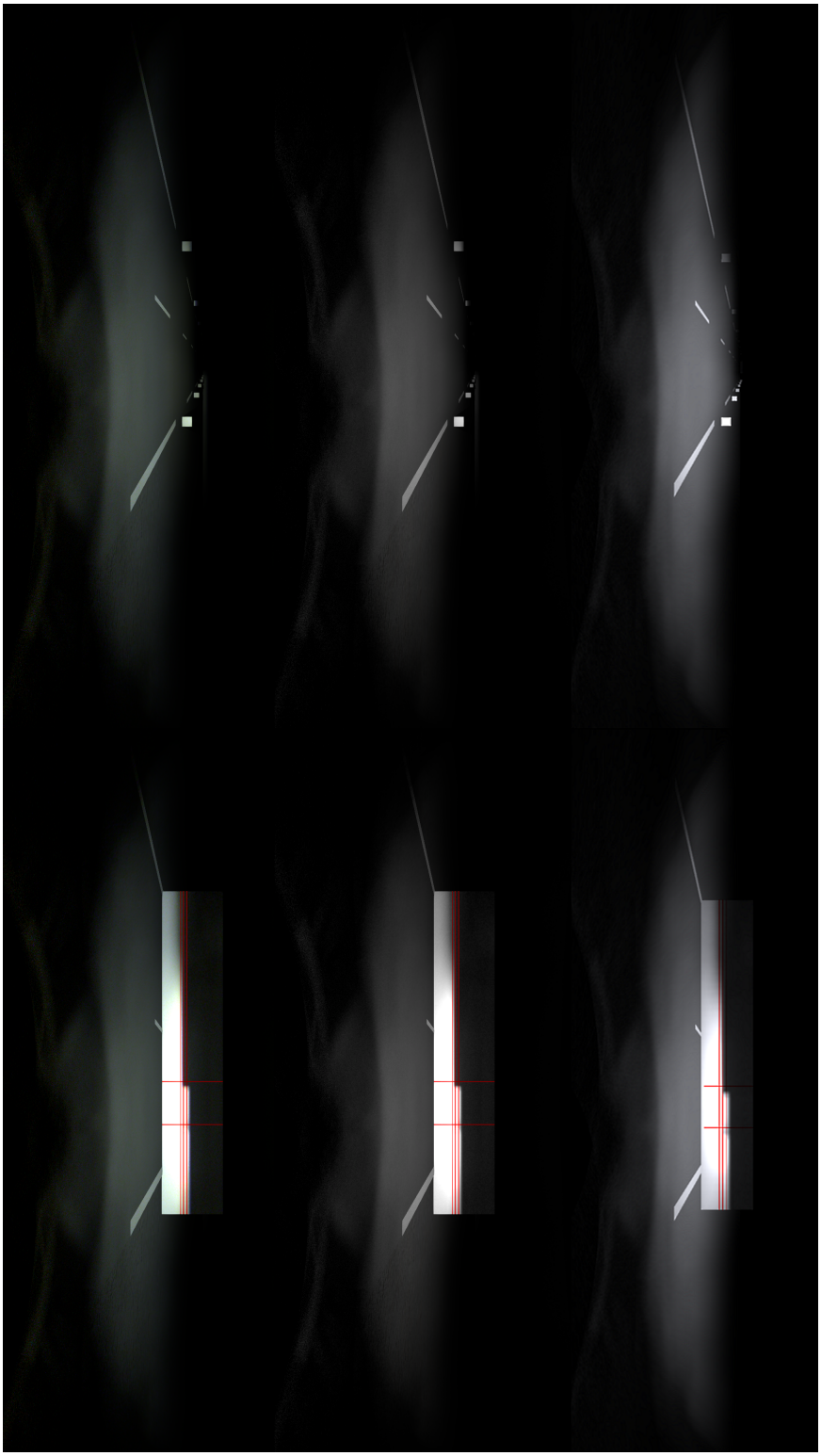

*Bild A4-1*: *Vergleich des LightDriver (oben) mit Hyperion (mittig mit monochromer und unten mit spektraler Lichtverteilung) bei der Simulation einer Abblendlichtverteilung in einer einfachen Straßenszene (links) und mit einer 10 m entfernten Messwand (rechts).*

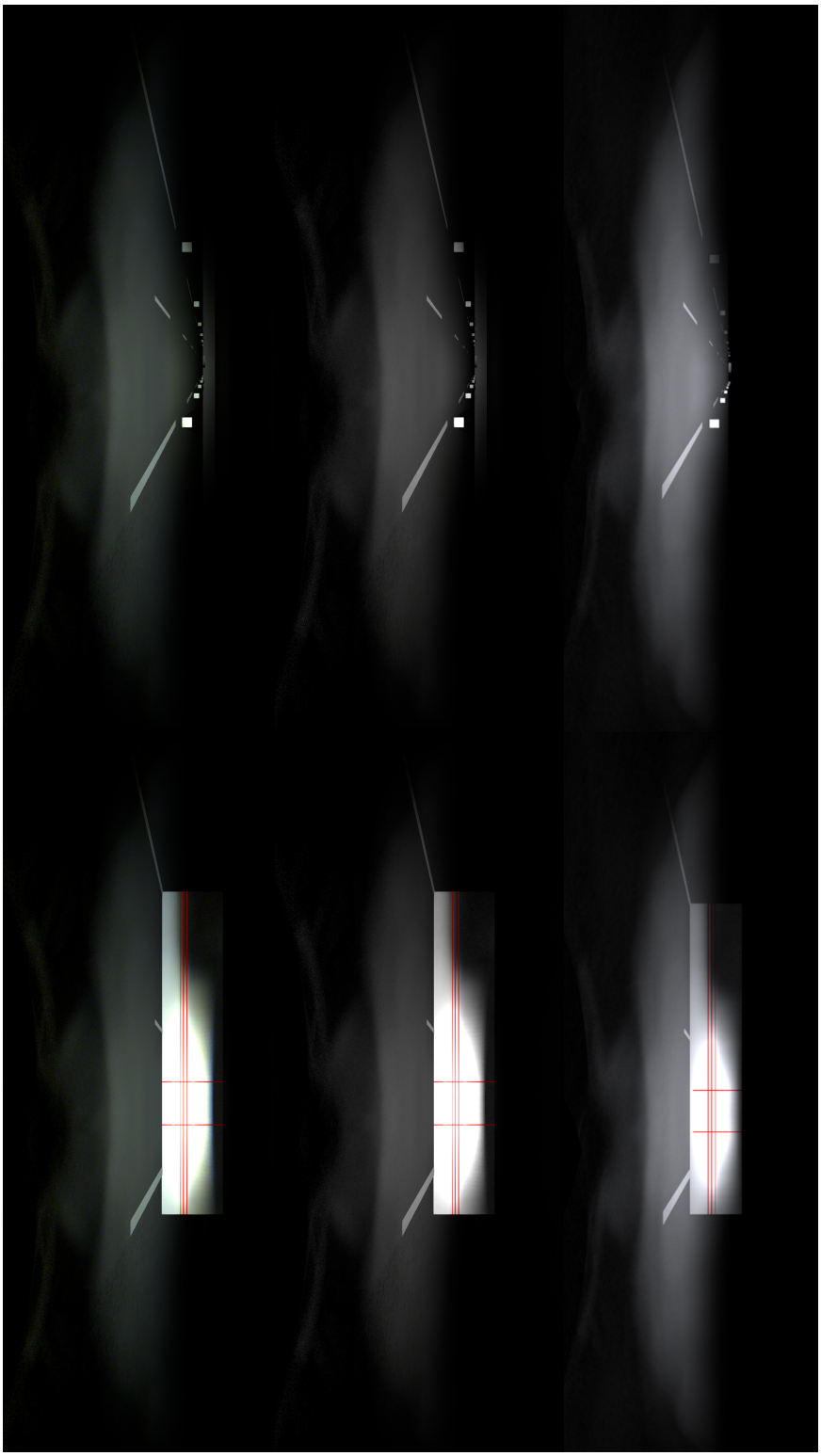

*Bild A4-2*: *Vergleich des LightDriver (oben) mit Hyperion (mittig mit monochromer und unten mit spektraler Lichtverteilung) bei der Simulation einer Fernlichtverteilung in einer einfachen Straßenszene (links) und mit einer 10 m entfernten Messwand (rechts).*

# A5 Analyse und Entwurf

## A5.1 Isolinien- und Falschfarbendarstellungen

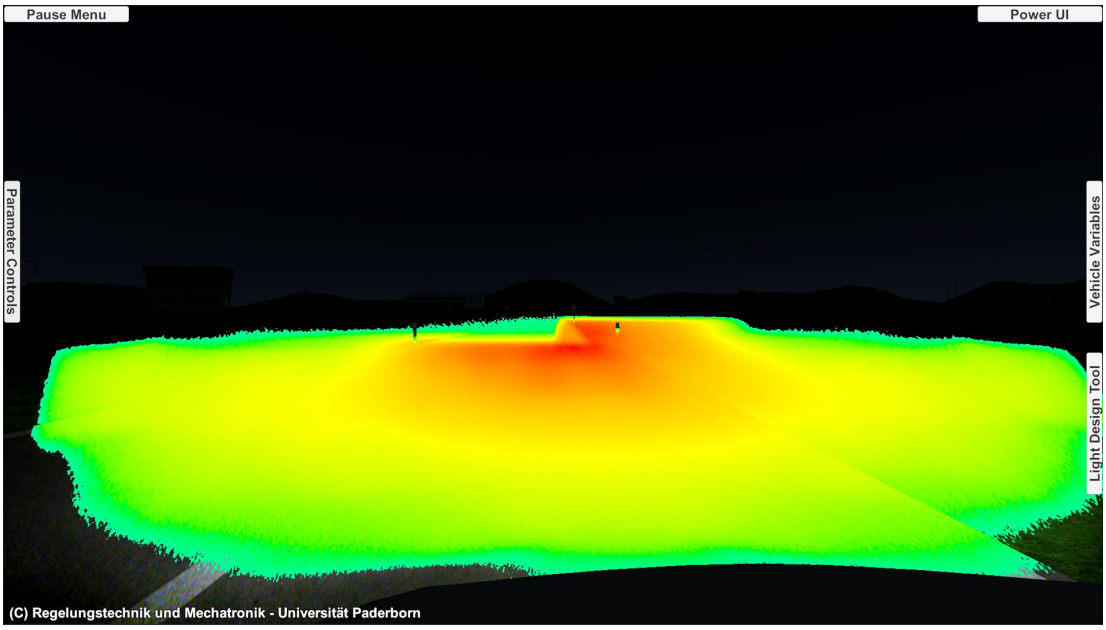


*Bild A5-1*: *Falschfarbendarstellung der Lichtstärke eines HD84-Systems.*

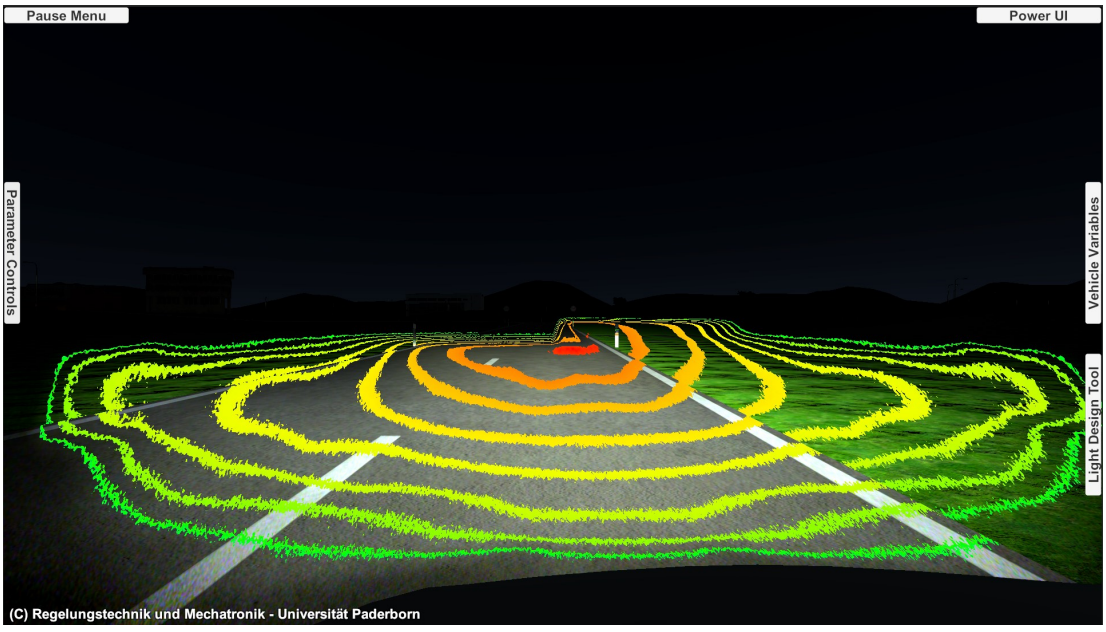


*Bild A5-2*: *Isoliniendarstellung der Lichtstärke eines HD84-Systems.*

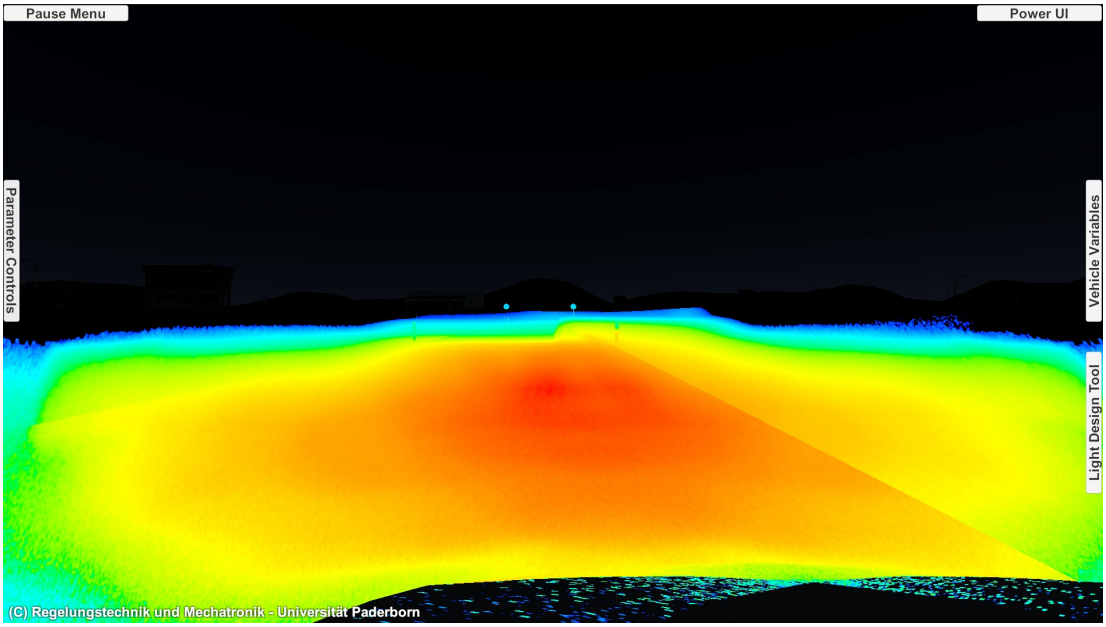


*Bild A5-3*: *Falschfarbendarstellung der Beleuchtungsstärke eines HD84-Systems.*

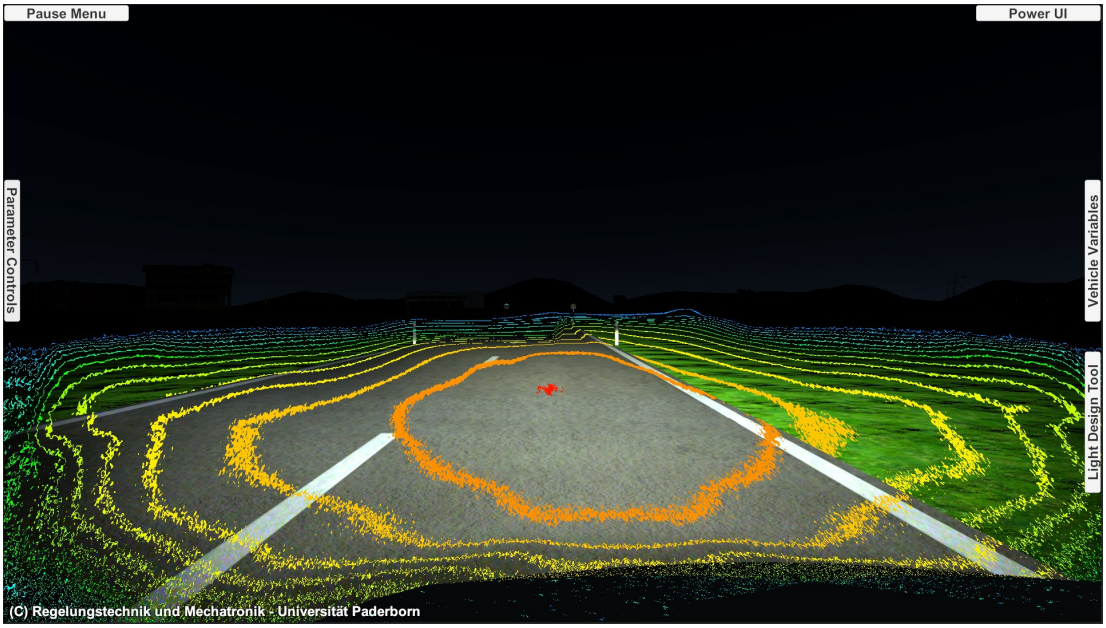


*Bild A5-4*: *Isoliniendarstellung der Beleuchtungsstärke eines HD84-Systems.*

## A5.2 Sonstige Analysesichten

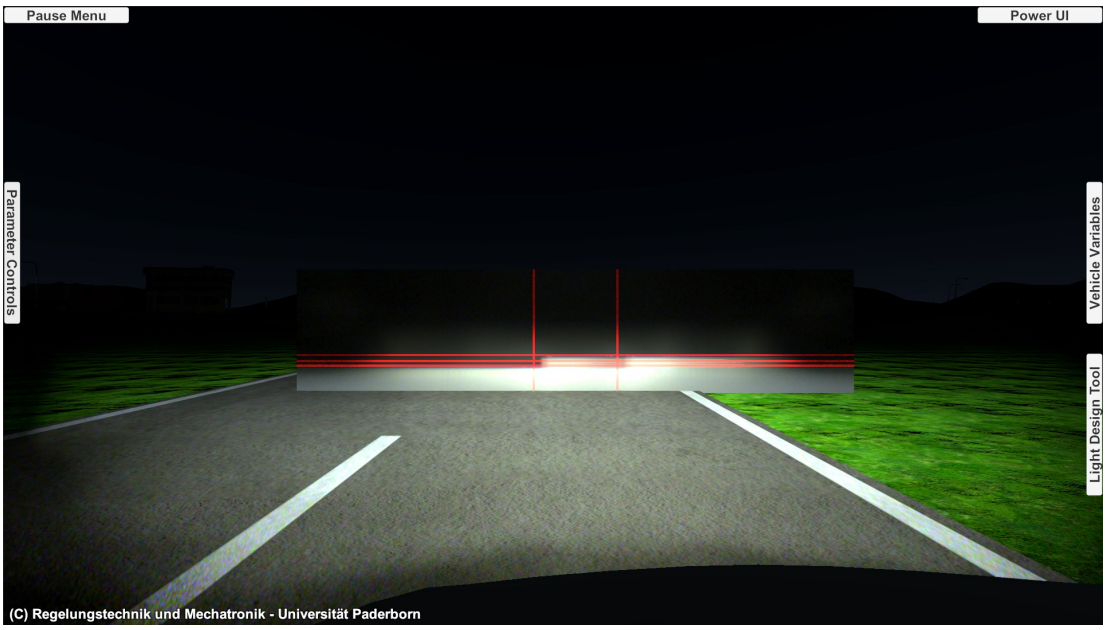


*Bild A5-5*: *Einblendung einer Messwand in 10 m Entfernung zur Kontrolle der Hell-Dunkel-Grenze des Scheinwerfersystems (rote Kontrolllinien an spezifischen Positionen dienen der Orientierung).*

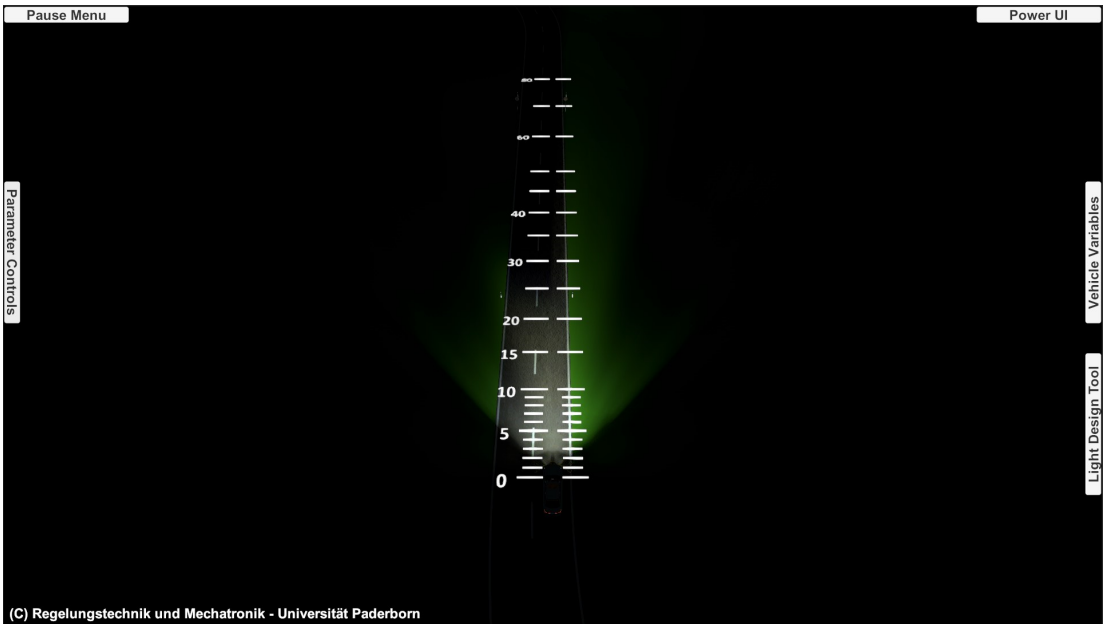


*Bild A5-6*: *Einblendung von Distanzmarken zur Beurteilung der Reichweite des Scheinwerfersystems.*

## Das Heinz Nixdorf Institut – Interdisziplinäres Forschungszentrum für Informatik und Technik

Das Heinz Nixdorf Institut ist ein Forschungszentrum der Universität Paderborn. Es entstand 1987 aus der Initiative und mit Förderung von Heinz Nixdorf. Damit wollte er Ingenieurwissenschaften und Informatik zusammenführen, um wesentliche Impulse für neue Produkte und Dienstleistungen zu erzeugen. Dies schließt auch die Wechselwirkungen mit dem gesellschaftlichen Umfeld ein.

Die Forschungsarbeit orientiert sich an dem Programm „Dynamik, Mobilität, Vernetzung: Eine neue Schule des Entwurfs der technischen Systeme von morgen“. In der Lehre engagiert sich das Heinz Nixdorf Institut in Studiengängen der Informatik, der Ingenieurwissenschaften und der Wirtschaftswissenschaften.

Heute wirken am Heinz Nixdorf Institut acht Professoren mit insgesamt 130 Mitarbeiterinnen und Mitarbeitern. Pro Jahr promovieren hier etwa 15 Nachwuchswissenschaftlerinnen und Nachwuchswissenschaftler.

## Heinz Nixdorf Institute – Interdisciplinary Research Centre for Computer Science and Technology

The Heinz Nixdorf Institute is a research centre within the Paderborn University. It was founded in 1987 initiated and supported by Heinz Nixdorf. By doing so he wanted to create a symbiosis of computer science and engineering in order to provide critical impetus for new products and services. This includes interactions with the social environment.

Our research is aligned with the program “Dynamics, Mobility, Integration: Enroute to the technical systems of tomorrow.” In training and education the Heinz Nixdorf Institute is in-volved in many programs of study at the Paderborn University. The superior goal in education and training is to communicate competencies that are critical in tomorrows economy.

Today eight Professors and 130 researchers work at the Heinz Nixdorf Institute. Per year approximately 15 young researchers receive a doctorate.

## Zuletzt erschienene Bände der Verlagsschriftenreihe des Heinz Nixdorf Instituts

Bd. 384 DÜLME, C.: Systematik zur zukunftsorientierten Konsolidierung variantenreicher Produktprogramme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 384, Paderborn, 2018 – ISBN 978-3-947647-03-3

Bd. 385 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 14. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 8. und 9. November 2018, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 385, Paderborn, 2018 – ISBN 978-3-947647-04-0

Bd. 386 SCHNEIDER, M.: Spezifikationstechnik zur Beschreibung und Analyse von Wertschöpfungssystemen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 386, Paderborn, 2018 – ISBN 978-3-947647-05-7

Bd. 387 ECHTERHOFF, B.: Methodik zur Einführung innovativer Geschäftsmodelle in etablierten Unternehmen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 387, Paderborn, 2018 – ISBN 978-3-947647-06-4

Bd. 388 KRUSE, D.: Teilautomatisierte Parameteridentifikation für die Validierung von Dynamikmodellen im modellbasierten Entwurf mechatronischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 388, Paderborn, 2019 – ISBN 978-3-947647-07-1

Bd. 389 MITTAG, T.: Systematik zur Gestaltung der Wertschöpfung für digitalisierte hybride Marktleistungen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 389, Paderborn, 2019 – ISBN 978-3-947647-08-8

Bd. 390 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 15. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 21. und 22. November 2019, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 390, Paderborn, 2019 – ISBN 978-3-947647-09-5

Bd. 391 SCHIERBAUM, A.: Systematik zur Ableitung bedarfsgerechter Systems Engineering Leitfäden im Maschinenbau. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 391, Paderborn, 2019 – ISBN 978-3-947647-10-1

Bd. 392 PAI, A.: Computationally Efficient Modelling and Precision Position and Force Control of SMA Actuators. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 392, Paderborn, 2019 – ISBN 978-3-947647-11-8

Bd. 393 ECHTERFELD, J.: Systematik zur Digitalisierung von Produktprogrammen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 393, Paderborn, 2020 – ISBN 978-3-947647-12-5

Bd. 394 LOCHBICHLER, M.: Systematische Wahl einer Modellierungstiefe im Entwurfsprozess mechatronischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 394, Paderborn, 2020 – ISBN 978-3-947647-13-2

Bd. 395 LUKEI, M.: Systematik zur integrativen Entwicklung von mechatronischen Produkten und deren Prüfmittel. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 395, Paderborn, 2020 – ISBN 978-3-947647-14-9

Bd. 396 KOHLSTEDT, A.: Modellbasierte Synthese einer hybriden Kraft-/Positionsregelung für einen Fahrzeugachsprüfstand mit hydraulischem Hexapod. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 396, Paderborn, 2021 – ISBN 978-3-947647-15-6

Bd. 397 DREWEL, M.: Systematik zum Einstieg in die Plattformökonomie. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 397, Paderborn, 2021 – ISBN 978-3-947647-16-3

---

Bezugsadresse:
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn

## Zuletzt erschienene Bände der Verlagsschriftenreihe des Heinz Nixdorf Instituts

Bd. 398 FRANK, M.: Systematik zur Planung des organisationalen Wandels zum Smart Service-Anbieter. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 398, Paderborn, 2021 – ISBN 978-3-947647-17-0

Bd. 399 KOLDEWEY, C.: Systematik zur Entwicklung von Smart Service-Strategien im produzierenden Gewerbe. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 399, Paderborn, 2021 – ISBN 978-3-947647-18-7

Bd. 400 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 16. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 2. und 3. Dezember 2021, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 400, Paderborn, 2021 – ISBN 978-3-947647-19-4

Bd. 401 BRETZ, L.: Rahmenwerk zur Planung und Einführung von Systems Engineering und Model-Based Systems Engineering. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 401, Paderborn, 2021 – ISBN 978-3-947647-20-0

Bd. 402 WU, L.: Ultrabreitbandige Sampler in SiGe-BiCMOS-Technologie für Analog-Digital-Wandler mit zeitversetzter Abtastung. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 402, Paderborn, 2021 – ISBN 978-3-947647-21-7

Bd. 403 HILLEBRAND, M.: Entwicklungssystematik zur Integration von Eigenschaften der Selbstheilung in Intelligente Technische Systeme. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 403, Paderborn, 2021 – ISBN 978-3-947647-22-4

Bd. 404 OLMA, S.: Systemtheorie von Hardware-in-the-Loop-Simulationen mit Anwendung auf einem Fahrzeugachsprüfstand mit parallelkinematischem Lastsimulator. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 404, Paderborn, 2022 – ISBN 978-3-947647-23-1

Bd. 405 FECHTELPETER, C.: Rahmenwerk zur Gestaltung des Technologietransfers in mittelständisch geprägten Innovationsclustern. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 405, Paderborn, 2022 – ISBN 978-3-947647-24-8

Bd. 406 OLEFF, C.: Proaktives Management von Anforderungsänderungen in der Entwicklung komplexer technischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 406, Paderborn, 2022 – ISBN 978-3-947647-25-5

Bd. 407 JAVED, A. R.: Mixed-Signal Baseband Circuit Design for High Data Rate Wireless Communication in Bulk CMOS and SiGe BiCMOS Technologies. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 407, Paderborn, 2022 – ISBN 978-3-947647-26-2

Bd. 408 DUMITRESCU, R, KOLDEWEY, C.: Datengestützte Projektplanung. Fachbuch. Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 408, Paderborn, 2022 – ISBN 978-3-947647-27-9

Bd. 409 PÖHLER, A.: Automatisierte dezentrale Produktionssteuerung für cyber-physische Produktionssysteme mit digitaler Repräsentation der Beschäftigten. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 409, Paderborn, 2022 – ISBN 978-3-947647-28-6

---

Bezugsadresse:
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn

Die vorliegende Arbeit befasst sich mit dem simulationsbasierten Entwurf hochauflösender Pixel-Scheinwerfersysteme durch virtuellen Nachtfahrten. Nach einer Darstellung der notwendigen theoretischen Grundlagen wird zunächst der derzeitige Stand der Technik beschrieben. Besonderes Augenmerk erhält die simulationsgestützte Entwicklung von Pixel-Scheinwerfersystemen. Die existierenden Lösungen werden vorgestellt und bewertet. Abgeleitet aus den vorhandenen Schwächen derzeitiger Nachtfahrtsimulationen wird ein Anforderungskatalog erarbeitet, den die hier vorgestellte Lösung bestmöglich erfüllen soll. Es folgt die Beschreibung der zu diesem Zweck entwickelten Nachtfahrtsimulation „Hyperion“. Nach einer Darstellung der Gesamtarchitektur, der daz gehörigen Komponenten und ihrer Wechselwirkungen, wird der Forschungskern der vorliegenden Arbeit detailliert betrachtet. Hierzu gehört im ersten Schritt die echtzeitfähige und qualitativ hochwertige Nachbildung des Lichts von Pixel-Scheinwerfern in einer virtuellen Umgebung. Bei der Virtualisierung von Pixel-Systemen werden neben dem ausgesandten Licht beider Scheinwerfer auch das Steuergerät und damit verbundene Sensoren betrachtet. Im zweiten Schritt werden darauf aufbauend Analyse- und Entwurfsverfahren für diese Systeme methodisch eingeführt, prototypisch implementiert und validiert. Hierbei liegt der Fokus nicht auf der optischen Auslegung des Scheinwerfers, sondern auf dem En wurf von Lichtfunktionen zur situationsadaptiven Steuerung der zahlreichen Lichtquellen eines Pixel-Scheinwerfers. Schließlich wird die entwickelte Lösung anhand des zuvor angefertigten Anforderungskatalogs bewertet. Zum Abschluss werden die zentralen Ergebnisse der Arbeit zusammengefasst. In einem Ausblick werden weitere Potentiale und Ausbaumöglichkeiten der Nachtfahrtsimulation diskutiert.