

# Nutzung von projektiven Texturen auf einer GPU

## zur Distanzmessung für automotive Sensorsimulationen

Christian Berger<sup>1,2</sup>, Bernhard Rumpe<sup>1,2</sup>

<sup>1</sup>Institut für Software Systems Engineering, TU Braunschweig

<sup>2</sup>Lehrstuhl für Software Engineering, RWTH Aachen  
<http://www.se-rwth.de>

### Kurzfassung

Software-gestützte Testumgebungen für Sensorfusionsalgorithmen und intelligente, sensorbasierte Fahrerassistenzsysteme sind bereits frühzeitig in der Entwicklungsphase für Erprobungen und Parametrisierungen geeignet. Allerdings lassen sich unterschiedliche Sensortypen und Verbaupositionen aufgrund durchzuführender Kalibrierungen nur aufwändig testen. In diesem Beitrag wird ein Verfahren vorgestellt, das heutige leistungsstarke, programmierbare Graphikprozessoren zur Generierung von Sensorrohdaten verwendet, um bereits frühzeitig den Softwareentwicklungsprozess zu unterstützen.

### 1. Einleitung

Die Entwicklung komplexer, intelligenter Fahrerassistenzsysteme zur Schaffung neuen Kundenmehrerts in Fahrzeugen basiert stark auf leistungsfähiger Sensorik zur Detektion des Fahrzeugumfeldes. Die von Sensoren produzierten Messwerte werden in nachgelagerten Verarbeitungsstufen durch Sensordatenfusionen zu einer abstrakten Darstellung aggregiert. Diese Darstellung wird von den Algorithmen der Fahrerassistenz ausgewertet und unterstützt den Fahrer indirekt beispielsweise durch akustische Abstandswarnung oder direkt wie etwa beim Einparkassistenten.

Die Softwareentwicklung von Sensorfusionsalgorithmen und weiteren intelligenznahen Algorithmen erfordert die Verarbeitung von Rohdaten der Sensoren. Hierzu werden zunächst Messfahrten geplant und durchgeführt, um die verbauten Sensoren während des Betriebs aufzuzeichnen. Diese Messergebnisse werden im Anschluss verwendet, um die erstellten Algorithmen zu entwickeln, zu erproben und zu optimieren.

Dieses Vorgehen ist relativ einfach und ressourcenschonend durchführbar, wenn angenommen werden kann, dass Sensorik und Verbaupositionen vorab bekannt sind. Es ist allerdings sehr aufwändig, verschiedene Sensortypen und Verbaupositionen zu variieren und ihre Abdeckungsbereiche zu evaluieren, da für aussagekräftige Vergleiche sowohl die Umweltbedingungen wiederholbar sein müssen, als auch für die montierten Sensoren jeweils eine Kalibrierung durchgeführt werden muss.



Eine Möglichkeit ist es, verfügbare Spezifikationen des Sensorrohdatenformats eines Herstellers einschließlich geeigneter Fehler- und Rauschannahmen zu verwenden. Damit können benötigte Eingabedaten synthetisch bereitgestellt und bei gleichzeitiger Erhöhung der Testtiefe zeitaufwändige Messfahrten deutlich reduziert werden.

Zur Synthetisierung der Sensorrohdaten ist der Systemkontext des Fahrzeugs zu modellieren. Diese virtuelle Umwelt wird dann verwendet, um das verwendete Messprinzip eines Sensors nachzubilden.

Zur Generierung eines Sensordatensatzes für Time-of-Flight-Sensoren kann beispielsweise das Prinzip des Raytracings verwendet werden. Da im Vorfeld bereits alle Parameter zur Berechnung bekannt sind, kann jeder emittierte Laserstrahl mit jedem Objekt der Szene geschnitten werden. Für einen Laserstrahl wird für den zu erstellenden Sensordatensatz der Schnittpunkt verwendet, der am nächsten am Sensor liegt.

Dieses Vorgehen ist für einfache Systemkontexte bestehend aus wenigen und einfachen Objekten mit einer kleinen Anzahl an Time-of-Flight-Sensoren ein praktikables Verfahren. Bei ansteigender Komplexität der Szene erweist sich dieses Vorgehen jedoch als nicht performant genug, um den Softwareentwicklungsprozess geeignet zu unterstützen, da die Laufzeit für  $k$  Objekte der Umwelt, die mit  $n$  Sensoren bestrahlt werden, die jeweils maximal  $m$  Lichtstrahlen emittieren, im ungünstigsten Fall in  $O(n*m*k*k)$  liegt.

## 2. GPU-gestütztes Verfahren zur Generierung von Sensorrohdaten

Eine Alternative zum CPU-basierten Verfahren des soeben beschriebenen Raytracings bietet der Einsatz moderner programmierbarer Graphikprozessoren (GPUs). Hierbei wird die rechenintensive Abstandsberechnung von der CPU auf eine GPU verlagert, so dass nur noch die eigentliche Auswertung und Transformation in das Sensorrohdatenformat auf der CPU durchzuführen ist.

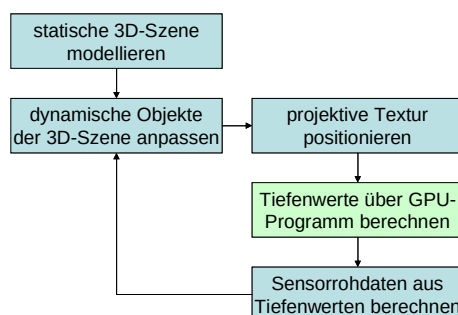


Abbildung 1: Verfahren zur Generierung von Sensorrohdaten für Time-of-Flight-Sensoren.

Moderne Graphikprozessoren sind grundsätzlich auf performante Bildsynthese ausgelegt und erlauben nur indirekt eine Bildanalyse, wie es die Generierung von Sensorrohdaten erfordert. Daher ist die durchzuführende Berechnung als Bildsyntheseproblem zu formulieren, um die Vorzüge einer modernen Graphikkarte zur Generierung von Sensorrohdaten zu nutzen. In Abbildung 1 wird überblicksartig das beschriebene Verfahren dargestellt.

### **Modellierung des Systemkontextes**

Zunächst ist es erforderlich, den gewünschten Systemkontext geeignet zu formulieren. Für GPU-basierte Algorithmen kann beispielsweise OpenGL als Modellierungs- und Berechnungssprache verwendet werden. Hierzu wird entweder durch Zeichnungs-primitive eine Welt modelliert oder vorhandene Modelle von CAD- oder 3D-Zeichnungsprogrammen in die Szene geladen. Abbildung 2 zeigt eine texturlose 3D-Szene der Innenstadt von Braunschweig.



Abbildung 2: Texturlose Darstellung der Innenstadt von Braunschweig.

Diese Szene bildet die Grundlage für die Generierung von Sensorrohdaten. Dabei hängt die Güte der generierten Sensorrohdaten direkt von dem Detaillierungsgrad der 3D-Szene ab. Die in Abbildung 2 dargestellte Szene enthält nur Gebäude. Die Szene kann aber aufgrund ihrer hierarchischen Modellierung leicht um aufwändigere Details wie Bäume, Fahrräder und ähnliches ergänzt werden.

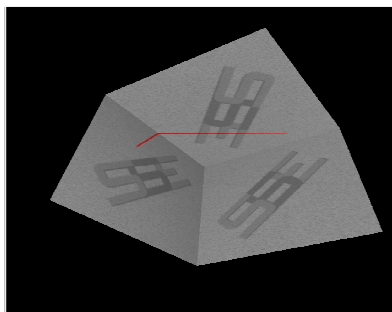
## Modellierung eines Time-of-Flight-Sensors

Ein Time-of-Flight-Sensor, wie beispielsweise ein Laser, emittiert aktiv Licht, um über die Laufzeit des Lichts Informationen über das Umfeld zu erhalten. Dieses Verfahren wird gleichfalls modelliert, um Abstandsdaten aus der Szene zu berechnen. Hierzu wird eine zweidimensionale Textur definiert, die Binärinformation über die Verteilung der vom Sensor emittierten Lichtstrahlen enthält.

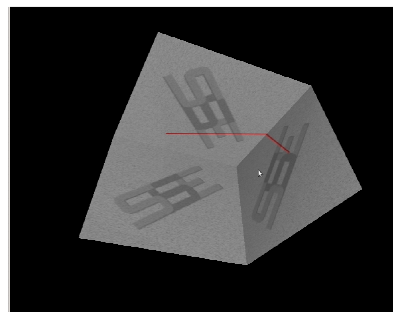
In der Szene wird ein Projektor definiert, dessen Aufgabe darin besteht, die definierte Textur in die 3D-Welt zu projizieren. Hierzu muss zunächst die Textur skaliert und verschoben werden, damit sie sich zentriert vor dem Projektor befindet. Anschließend wird der Projektor in der Szene platziert und von der gewünschten Kameraposition betrachtet. In Formel (1) wird die vollständige Transformation dargestellt.

$$T = \begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1.0 \end{bmatrix} * P_p * V_p * V_e^{-1} \quad (1)$$

T beschreibt die zu berechnenden Zielkoordinaten der Textur, das Produkt aus  $P_p$  und  $V_p$  bezeichnet die Modellprojektion in die Szene aus Projektorsicht. Hierfür können direkt die Daten `GL_PROJECTION_MATRIX` und `GL_MODELVIEW_MATRIX` aus dem OpenGL-Kontext verwendet werden. In  $V_e^{-1}$  ist die Transformation der errechneten Texturkoordinaten in die gewünschten Kamerakoordinaten beschrieben. Mittels projektiver Texturen ist es möglich, mehrere unabhängige Projektoren in einer 3D-Szene zu definieren und zu betrachten.



(a)



(b)

Abbildung 3: Illustration einer projektiven Textur.

In Abbildung 3 ist die Funktionsweise eines Projektors dargestellt. Beide Abbildungen zeigen einen Blick in einen Würfel, dessen Innenseite mit einer grauen Textur versehen ist. Die rote Gerade stellt die Projektion der Textur einer schematischen Scanlinie dar, die in den Würfel hineinprojiziert wird.

Der Würfel wird zwischen der Abbildung 3.a) und Abbildung 3.b) mit der Maus nach links gezogen, ohne dass sich die Position des Projektors verändert. Dadurch verschiebt sich die Figur unter der Scanlinie, die selbst in den rechten Bereich des Würfels strahlt.

Nachdem die 3D-Szene durch einen Projektor angestrahlt wird, kann ein spezielles auf der GPU ausgeführtes Programm verwendet werden, um Abstände für die angestrahnten Bereiche zu bestimmen. Diese Programme, die grundsätzlich für die Berechnung von Farben eingesetzt werden, heißen Shader und gliedern sich in zwei Teile. Der erste Teil führt Berechnungen an den Knoten der Geometrie eines Objekts durch und wird daher als Vertex-Shader bezeichnet. Der zweite Teil interpoliert fehlende Fragmente zwischen den Knoten eines Objekts und wird Pixel- oder Fragment-Shader bezeichnet.

Zur Berechnung der Abstände wird ein Bildsyntheseproblem formuliert. In OpenGL wird die Anordnung und somit die räumliche Tiefe zwischen Objekten über den z-Buffer, der auch als Tiefenwertspeicher bezeichnet wird, berechnet. Zur Bestimmung der Abstände von angestrahnten Objekten werden im Vertex-Shader zunächst die Daten aus dem z-Buffer in Form einer Tiefenbildtextur an den Fragment-Shader übertragen.

Im Fragment-Shader wird die eingangs definierte Binärtextur verwendet, um die angestrahnten Bereiche der Szene zu ermitteln. In diesen Bereichen wird der korrespondierende Wert des z-Buffers aus der Tiefenbildtextur gelesen und als Ergebnis der Bildsynthese in die Ausgabe der Graphikkarte übertragen. Die Bestimmung der korrekten Entfernung ist hierbei unter Berücksichtigung der Nah- und Fernbereiche des sichtbaren Raumes durchzuführen. In Formel (2) wird der Zusammenhang dargestellt, der bei der Berechnung im Fragment-Shader durchzuführen ist.

$$z = (w - near) / (1 - \frac{near}{far}) \quad (2)$$

Das synthetisierte Bild enthält an den angestrahnten Bereichen die durch die GPU bestimmten Tiefenwerte. Diese Werte werden durch eine nachgelagerte Bildanalyse aus der Graphikkarte gelesen und anschließend umgerechnet. Hierbei werden die zweidimensionalen Koordinaten des Bildes unter Verwendung der Tiefeninformationen, die als Grauwerte im Bild vorliegen, in dreidimensionale Koordinaten relativ zum Ursprung der Szene zurückgerechnet. In Formel (3) wird die erforderliche Umrechnung dargestellt.

Das Produkt aus den Matrizen P und M berechnet die Projektion in Weltkoordinaten, das Tripel (x, y, z) bezeichnet die umzurechnenden Koordinaten. Das Quadrupel  $(\hat{x}, \hat{y}, \bar{w}, \bar{h})$  benennt Versatz sowie Breite und Höhe des verwendeten Bildausschnitts.

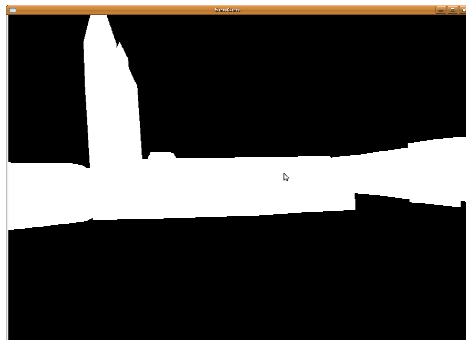
$$C = (P * M)^{-1} * \begin{pmatrix} \frac{2 * (x - \hat{x})}{\bar{w}} - 1 \\ \frac{2 * (y - \hat{y})}{\bar{h}} - 1 \\ 2 * z - 1 \\ 1 \end{pmatrix} \quad (3)$$

Mit dem Ergebnis C in Weltkoordinaten kann unter Berücksichtigung der virtuellen Verbauposition ein Sensorrohdatensatz für einen Time-of-Flight-Sensor berechnet werden. Dieser kann in nachgelagerten Verarbeitungsstufen noch durch künstliche Störgrößen modifiziert werden.

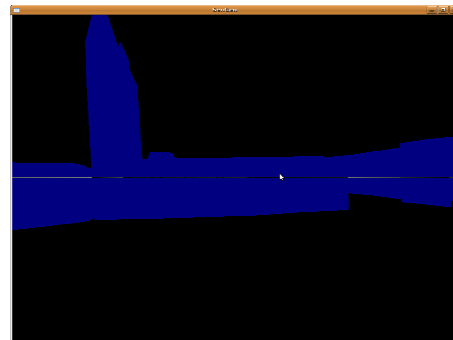
Der gewählte Systemkontext bedingt eine Reihe unausweichlicher Hard- sowie Software-Entscheidungen, die zur Gestaltung einer Gesamtsystemarchitektur führen. Im Detail werden diese in [3] beschrieben, wobei für das weitere Verständnis dieser Arbeit hier nur einzelne Teilaspekte erläutert werden.

### 3. Anwendungsbeispiel

Im Folgenden wird die Funktionsweise des beschriebenen Prinzips dargestellt.



(a)



(b)

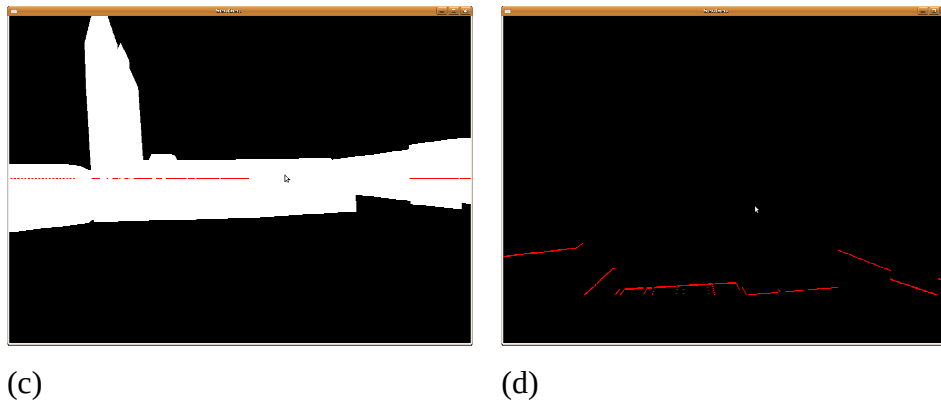


Abbildung 4: Abstandsbestimmung durch Bildsynthese am Beispiel des Braunschweig Doms.

In Abbildung 4.a) wird der Braunschweiger Dom aus der vorherigen Abbildung 1 aus der Nähe dargestellt. Abbildung 4.b) zeigt das aktivierte Shader-Programm auf der GPU. Bereiche, die nicht durch die projizierte Textur beschienen werden, würden normalerweise aus der 3D-Szene entfernt werden. Zum besseren Verständnis wurden diese Bereiche blau eingefärbt. In der Mitte des Bildes verläuft eine Linie, in der bereits die Tiefenwerte als Grauwerte kodiert sind.

In Abbildung 4.c) wurde das Shader-Programm deaktiviert und das Ergebnis der Tiefenwertbestimmung wird gleichfalls zu den regulären Objekten in die 3D-Szene gezeichnet. In Abbildung 4.d) wurden die ursprünglichen Objekte ausgeblendet und die Kameraposition der positiven z-Achse entlang nach oben verlagert, um die generierten Messwerte besser einsehen zu können.

#### 4. Auflösung und Genauigkeit des Verfahrens

Erste Testreihen der Anwendung des Verfahrens erlauben Generierungszyklen von etwa 110 Hz. Im Vergleich zum SICK Laserscanner [1] liegt damit ein Geschwindigkeitsvorteil eines virtuellen Sensors von über 40% vor.

Prinzipiell kann bei virtuellen Sensoren eine beliebig hohe Genauigkeit erzielt werden, jedoch hängt die Genauigkeit eines virtuellen Sensors direkt von der Definition der Nah- und Fernebenen ab. Hierbei ist zu beachten, dass mit zunehmender Distanz des gemessenen Objekts vom Projektor die Genauigkeit sinkt, da der z-Buffer keine lineare Abbildung zwischen Nah- und Fernebene, sondern den perspektivlosen Quotienten  $z/w$  enthält [2].

Die berechneten Abstände zu den Objekten können durch beliebige Filter noch künstlich verrauscht werden. Dadurch lässt sich eine potentielle Messungenauigkeit der Sensorik modellieren, um die Arbeit mit realer Sensorik zu ergänzen.

## **6. Verwandte Arbeiten**

In [3] wird ein Verfahren zur Absicherung intelligenter Algorithmen dargestellt. Dieses erlaubt im Gegensatz zum hier dargestellten Vorgehen nur den Einsatz von Testumgebungen für den Verarbeitungsbereich nach einer Sensordatenaufbereitung. In [4] und [5] werden hierbei Erweiterungen sowie Anwendung des Prinzips in der DARPA Urban Challenge 2007 [6] dargelegt.

In [7] wird das grundsätzliche Verfahren für projektive Texturen dargestellt. Im Fokus steht hierbei die Bildsynthese. Diese Arbeit bildete die Basis für das hier dargestellte Verfahren.

Die Autoren von [8] beschreiben ein verteiltes Robotikframework, das ebenfalls in der Lage ist, Sensordaten zu generieren. Das dort implementierte Verfahren basiert auf dem Prinzip des Raytracings und nutzt nicht die Vorteile eines GPU-basierten Algorithmus.

Neben dem Team CarOLO der Technischen Universität Braunschweig nahm auch ein Team aus Berlin an der DARPA Urban Challenge 2007 teil. In [9] illustrieren die Autoren ein Verfahren zur Sensordatengenerierung, das CPU-gestützt ausschließlich auf dem vollständigen Tiefenwertbild einer 3D-Szene basiert.

Die Firma aus [10] stellt mit CarMaker ein Programm zur Verfügung, das es erlaubt, virtuelle Teststrecken dreidimensional zu befahren. Der Anwendungsschwerpunkt liegt hierbei auf der Fahrdynamiksimulation.

In [11] stellen die Autoren ein verteiltes System zur Visualisierung von Sensordaten vor. Der Fokus liegt hierbei ausschließlich auf der Darstellung und nicht auf der Synthese von Sensorrohdaten.

Die Simulationsumgebung in [12] ermöglicht Analysen über Verkehrsflüsse auf mehrspurigen Richtungsfahrbahnen wie beispielsweise Autobahnen. Hierbei steht die Simulation des Verkehrsflusses und nicht die Simulation einer Umgebung für ein einzelnes Fahrzeug im Vordergrund.

In [13] stellen die Autoren ein Simulations-Framework vor, das wie [12] auch Untersuchungen an Verkehrsströmen zulässt. Hierbei können zwar individuelle Verkehrsteilnehmer im Fokus stehen, jedoch werden Verhaltensmodelle und ihre Auswirkungen im Gesamtsystem untersucht.

Die Arbeit aus [14] stellt einen verwandten Ansatz für Photonic Mixer Devices dar. Im Gegensatz zum hier dargestellten Verfahren werden allerdings keine projektiven Texturen verwendet, wodurch nur ein Photonic Mixer Device zu einem Zeitpunkt simuliert werden kann.

Die Autoren der Arbeit aus [15] beschreiben ein Verfahren zur Erzeugung von Radarziellisten. Hierbei werden Objekte und Fahrzeuge maßgeblich über Reflexionspunkte beschrieben und durch künstliches Rauschen modifiziert. Ziel der Arbeit ist die Unterstützung von Verarbeitungsalgorithmen von Radarobjektlisten.

## 7. Zusammenfassung

Die Entwicklung komplexer, intelligenter Fahrerassistenzsysteme verwendet eine Vielzahl verschiedener Sensoren zur verlässlichen Detektion des Fahrzeugumfeldes. Für die Software-Entwicklung bietet es sich an, nicht nur mit Rohdaten der Sensoren zu arbeiten, sondern diese auch synthetisch bereitzustellen.

In diesem Beitrag wurde ein Verfahren gezeigt, das es erlaubt, Sensorrohdaten von virtuellen Time-of-Flight-Sensoren zu generieren. Damit ist es möglich, unterschiedliche Sensortypen und Verbaupositionen am Versuchsträger zur Ermittlung geeigneter Sensorabdeckungsbereiche zunächst virtuell zu bestimmen, ohne eine aufwändige Montage oder Kalibrierung durchzuführen. Das dargestellte Verfahren lässt sich auf einem modernen Graphikprozessor implementieren, um als Bildsyntheseproblem die CPU zu entlasten.

Das vorgestellte Verfahren ist damit ein weiterer Baustein zur Virtualisierung des Prüfstands für intelligente und damit notwendigerweise stark Software-lastige Fahrerassistenz, die in ihrer Komplexität auf allein traditionelle Weise nicht mehr ausreichend getestet werden kann.

## Literatur

- [1] SICK AG, [www.sick.com](http://www.sick.com), Oktober 2008.
- [2] E. Catmull: *A subdivision algorithm for computer display of curved surfaces*. Dissertation, Universität von Utah, 1974.
- [3] C. Basarke, C. Berger, K. Homeier, B. Rumpe: *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*. 11th Automotive Technology Conference, Virtual Vehicle Creation 2007, Vieweg Technologie Forum, 2007.
- [4] C. Basarke, C. Berger, B. Rumpe: *Software & Systems Engineering Process and Tools for the Development of Autonomous Driving Intelligence*. Journal of Aerospace Computing, Information, and Communication (JACIC) Vol. 4, Nr. 12, S. 1158-1174, Oktober 2007.
- [5] F. Rauskolb, K. Berger, C. Lipksi, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher J-M. Wille, P. Hecker, T. Nothdurft, M. Doering, K. Homeier J. Morgenroth, L. Wolf, C. Basarke, C. Berger, T. Gülke, F. Klose, B. Rumpe: *Caroline: An Autonomously Driving Vehicle for Urban Environments*. Journal of Field Robotics, Wiley Periodicals, Vol. 25, Nr. 9, S. 674-724, September 2008.

- [6] DARPA Urban Challenge, <http://www.darpa.mil/grandchallenge/index.asp>, August 2007.
- [7] C. Everitt: *Projective Texture Mapping*. NVidia Whitepaper, 2001.
- [8] B. Gerkey, R. T. Vaughan, A. Howard: The Player/Stage Project: *Tools for Multi-Robot and Distributed Sensor Systems*. Proceedings of the 11th International Conference on Advanced Robotics, S. 317-323, 2003.
- [9] R. Rojas: *Forschungsaktivitäten beim Urban Challenge Team Berlin*. In: 9. Braunschweiger Symposium für Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel, 2008.
- [10] IPG Automotive GmbH, *CarMaker Vehicle Dynamic Simulation*. <http://www.ipg.de>, November 2008.
- [11] J. Klandt, G. Anders: *3D-Visualisierungswerkzeuge für die Entwicklung und Präsentation von Fahrerassistenzsystemen*. In: 9. Braunschweiger Symposium für Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel, 2008.
- [12] Bundesautobahnsimulator, [www.babsim.de](http://www.babsim.de), Oktober 2008.
- [13] J. J. Olstam, J. Lundgren, M. Adlers, P. Matstoms: *A Framework for Simulation of Surrounding Vehicles in Driving Simulators*. In: ACM Transactions on Modeling and Computer Simulation, Vol. 18, Nr. 3, S. 1-24, Juli 2008.
- [14] M. Keller, J. Orthmann, A. Kolb, V. Peters: *A Simulation Framework for Time-Of-Flight Sensors*. In: Proceedings of the International Symposium on Signals, Circuits and Systems, 2007.
- [15] M. Bühren, B. Yang: *Simulation of automotive radar target lists using a novel approach of object representation*. In: Proceedings of IEEE Intelligent Vehicle Symposium, S. 314-319, 2006.