Carsten Schmidt

Hardware-in-the-Loop-gestützte Entwicklungsplattform für Fahrerassistenzsysteme –

Modellierung und Visualisierung des Fahrzeugumfeldes

Cuvillier Verlag Göttingen
Internationaler wissenschaftlicher Fachverlag

Hardware-in-the-Loop-gestützte Entwicklungsplattform für Fahrerassistenzsysteme – Modellierung und Visualisierung des Fahrzeugumfeldes

Dissertation zur

Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

im Fachbereich Elektrotechnik / Informatik (FB16)

der Universität Kassel

vorgelegt von Carsten Schmidt (geb. am 16.05.1979 in Wolfenbüttel)

Eingereicht am: 27. Juli 2010

Disputation am: 25. Februar 2011

1. Gutachter: Prof. Dr. rer. nat. Ludwig Brabetz

2. Gutachter: Prof. Dr. rer. nat. Albert Zündorf

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet über http://dnb.d-nb.de abrufbar.
1. Aufl. - Göttingen: Cuvillier, 2011
Zugl.: Kassel, Univ., Diss., 2011

978-3-86955-727-4

Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

1. Auflage 2011
Gedruckt auf säurefreiem Papier.

978-3-86955-727-4

Vorwort

Die vorliegende Arbeit ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter des Fachgebietes *Fahrzeugsysteme und Grundlagen der Elektrotechnik* an der Universität Kassel innerhalb des EU Forschungsprojektes *DECOS* entstanden.

Zu Beginn meiner Tätigkeit stand das vakante Fachgebiet unter der kommissarischen Leitung von Prof. Dr.-Ing. Heinz Theuerkauf, der auch die laufenden Forschungsprojekte betreute. Ich möchte Prof. Theuerkauf für seine Anregungen und Unterstützung meiner Arbeit danken.

Herrn Dr.-Ing. Mhemed Ayeb, dem Leiter des an der Universität Kassel angesiedelten Arbeitspaketes im Projekt *DECOS*, möchte ich für die vielen hilfreichen Diskussionen zu Fragen der Echtzeitsimulation und Fahrzeugelektronik und für die angenehme Projektleitung danken.

Mit der Berufung von Prof. Dr.rer.nat. Ludwig Brabetz zum Leiter des Fachgebietes fand ein Wechsel in der Betreuung der wissenschaftlichen Arbeiten statt. Für das entgegen gebrachte Interesse, insbesondere der laufenden Arbeiten, und die Übernahme des Vorsitz der Prüfungskommission möchte ich Prof. Brabetz danken.

Weiterhin gilt mein Dank Prof. Dr.rer.nat. Albert Zündorf vom Fachgebiet *Software Engineering* für die Übernahme des Zweitgutachters.

Für das angenehme Arbeitsklima möchte ich vor allem meinen Projektkollegen Christian O. Schmidt und Dirk Tellmann danken, die durch ihre fachliche und menschliche Kompetenz nicht nur zum Gelingen des Projektes beigetragen haben, sondern darüber hinaus dafür gesorgt haben, daß mir die Zeit in Kassel immer in guter Erinnerung bleiben wird.

In diesem Sinne gilt mein Dank auch Patrick Gräbel, Dominik Holler, Michael Meyer und Klaus Simon, die durch ihre studentischen Arbeiten zum Gelingen dieser Arbeit beigetragen haben.

Den damaligen Mitarbeitern des Fachgebietes, Alexandra Burger, Fränk Diegmüller, Stefan Fröhlich, Ralf Gemmerich, Oliver Haas, Judith Keuch, Dirk Schneider und Thomas Waldmann, sei für das stets angenehme Arbeitsklima gedankt.

Ein besonderer Dank gilt meinen Eltern, Karl und Marianne Schmidt, und meinen Schwestern, Carla und Cornelia Schmidt, die mir das Studium ermöglicht haben und mir stets mit Rat und Tat bei den alltäglichen Dingen zur Seite standen.

Achim, im April 2011 Carsten Schmidt

Erklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation selbständig und ohne unerlaubte Hilfe angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Kein Teil dieser Arbeit ist in einem anderen Promotions oder Habilitationsverfahren verwendet worden.

Inhaltsverzeichnis

1. Einleitung

Moderne Kraftfahrzeuge werden zunehmend mit Fahrerassistenzsystemen ausgestattet, die nicht nur den Bewegungszustand des Eigenfahrzeuges zur Vermeidung oder Minderung von Unfällen heranziehen, sondern ebenfalls das Fahrzeugumfeld in eine Regelstrategie mit einbeziehen.

So vermeidet z.B. eine adaptive Geschwindigkeitsregelung durch rechtzeitige Verzögerung des Eigenfahrzeuges einen Auffahrunfall. Das Fahrzeugumfeld wird dazu mit einem Radar- oder Lidar-Sensor erfasst und das vorausfahrende Fahrzeug zur Berechnung des Bremseingriffs beobachtet. Ermöglicht wird dieser Bremseingriff durch die vorhandene Steuerelektronik automatischer Blockierverhinderer oder Stabilitätsregelsysteme, die durch die Vernetzung der elektronischen Fahrzeugkomponenten Sollwerte für die Verzögerung des Fahrzeuges entgegen nehmen können.

Am Beispiel der adaptiven Geschwindigkeitsregelung wird deutlich, daß hohe Anforderungen bezüglich der Verkehrs- und Betriebssicherheit an solche Steuergeräte gestellt werden müssen, da diese den Fahrer bei der Fahrzeugführung auch in sicherheitsrelevanten Situationen entlasten sollen.

Die Herstellung eines Fahrerassistenzsystems, das das Eigenfahrzeug als Bestandteil bzw. als Komponente in einem Verkehrsfluß betrachtet, stellt einen Zulieferer bereits bei der Entwicklung vor neue Herausforderungen. Konnte für Assistenzsysteme zur Traktionskontrolle die Funktionalität durch ausreichend genaue Fahrdynamiksimulationen validiert werden, erfordern Assistenzsysteme, die die Fahrt des Eigenfahrzeuges im Verkehrsfluß regeln, bereits eine ungleich aufwendigere Systemumgebung.

Die Systemumgebung eines solchen Assistenzsystem besteht i.d.R. aus den drei Komponenten Verkehrsumfeld, Sensorik und Eigenfahrzeug.

1. Das Verkehrsumfeld des Eigenfahrzeuges bedingt sowohl einen Ruhe- als auch einen Reglerbetrieb des Assistenzsystems.

   Im Ruhebetrieb beobachtet das Assistenzsystem den Verkehrsfluß und überlässt dem Fahrer die alleinige Führung des Fahrzeuges, da keine Bedrohung für einen Verkehrsteilnehmer besteht.

Wird durch fortlaufende Beobachtung eine Gefahrensituation erkannt, greift das Assistenzsystem durch einen Regeleingriff, z.B. eine Verzögerung in die Fahrzeugführung ein.

2. Durch eine geeignete Sensorik erfasst das Assistenzsystem das unmittelbare Verkehrsumfeld des Eigenfahrzeuges. Die aufbereiteten Meßdaten werden der Reglerfunktion zur Verfügung gestellt, um die momentane Verkehrssituation hinsichtlich eines Regeleingriffs zu bewerten.

3. Die Ausgabe der Stellgröße, z.B. eines Lenk- oder Bremseingriffs, wirkt sich auf die Dynamik des Eigenfahrzeuges aus und verändert somit die Position des Eigenfahrzeuges relativ zum Fremdverkehr. Es entsteht u.U. eine Rückkopplung mit dem Verkehrsfluß.

Während der Entwicklung solcher Assistenzsysteme ist es für einen Zulieferer wünschenswert bereits die Integration in die oben beschriebene Systemumgebung zu berücksichtigen, um durch gezielte Testfälle die Dynamik des Assistenzsystems im Zusammenspiel mit dem Eigenfahrzeug auch in Grenzsituationen sicherzustellen. Dies ist insbesondere durch die bereits erwähnte Sicherheitsrelevanz solcher Systeme erforderlich.

Ein Automobilhersteller, für den dieses Assistenzsystem nur eine Teilkomponente seines Produktes darstellt, beschäftigt sich eher mit der Integration dieser Komponente in seine Fahrzeugflotte. Dazu zählt die Systemapplikation, d.h. die Anpassung der Komponente an die verschiedenen Fahrzeugkonzepte, aber auch die Bewältigung der Komponentenvielfalt unterschiedlicher Funktionen, z.B. einem Bremsensteuergerät verschiedener Hersteller. Bei einem Automobilhersteller wird ein Test der beschriebenen Assistenzsysteme in der Prozeßverfolgung als Funktions- und Integrationstest umgesetzt.

An der Universität Kassel wurde eine Entwicklungsplattform geschaffen, die eine Funktionsentwicklung und einen Funktionstest von Fahrerassistenzsystemen, die auf obige Systemumgebung angewiesen sind, unterstützen soll.

Basis der Entwicklungsplattform ist ein *HiL*-Simulator, an den das Assistenzsystem als Echtteil angeschlossen wird, s. Abbildung 1.1. Verschiedene Software-Module des Simulators bilden die Systemumgebung des Assistenzsystems nach.

Assistenzsysteme erwarten für einen funktionsgerechten Betrieb Informationen über das Fahrzeugumfeld. Diese Informationen bestehen i.d.R. aus Objektlisten, die durch Verarbeitung und Fusionierung der Meßergebnisse verschiedener Umfeldsensoren generiert werden. Jedem Objekt, das einem

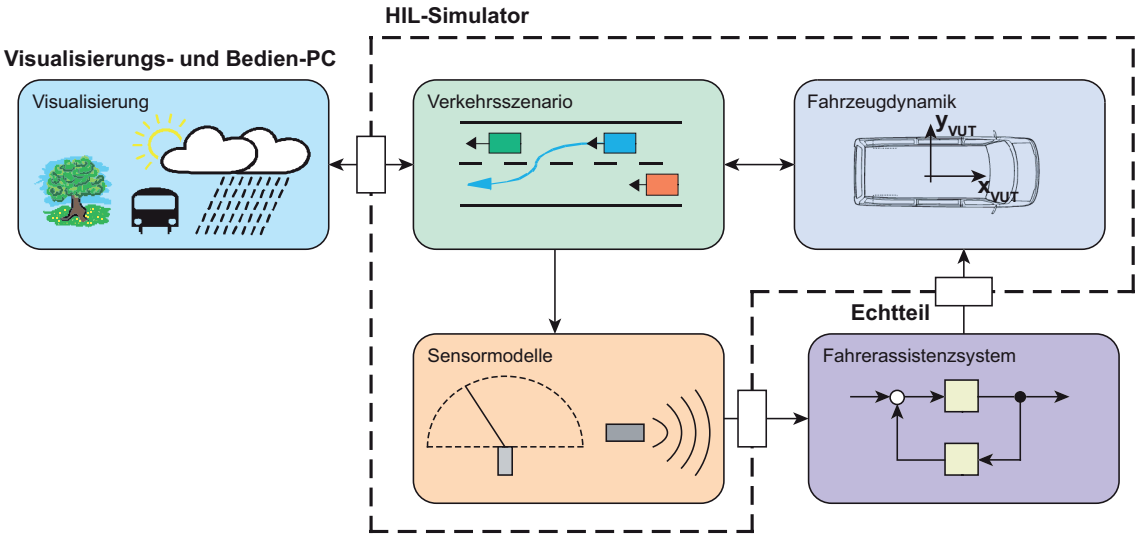

Abbildung 1.1.: Struktur der Entwicklungsplattform.

Fahrzeug des Verkehrsumfeldes entspricht, wird dabei der geschätzte Bewegungszustand relativ zum Eigenfahrzeug zugeordnet, so daß ein Assistenzsystem die aktuelle Verkehrssituation bzgl. ihres Gefahrenpotentials bewerten und ggf. einen Regeleingriff vornehmen kann.

Die Entwicklungsplattform bildet die Umfeldsensorik durch sog. Sensormodelle nach, die aus dem simulierten Verkehrsumfeld anhand des Bewegungszustandes und der Geometrie eines Verkehrsteilnehmers die erforderlichen Informationen für das Assistenzsystem zur Verfügung stellen. Benötigt ein Assistenzsystem Informationen mehrerer Sensoren werden deren Modelle parallel auf dem Echtzeitsimulator berechnet. Die Informationen werden dann einzeln oder durch eine vorausgehende Datenfusion an das Assistenzsystem übertragen.

Um die Systemumgebung des Assistenzsystems realitätsnah nachzubilden, simuliert ein weiteres Software-Modul den Fremdverkehr, in dem sich auch das Eigenfahrzeug des Assistenzsystems gewegt. Der Verkehrsfluß ist i.d.R. für die Verkehrsteilnehmer unkritisch, so daß kein Regeleingriff erforderlich ist. Um jedoch eine Reaktion des Assistenzsystems zu provozieren, wird beliebigen Verkehrsteilnehmern eine Bewegungstrajektorie aufgeprägt, die zu einer für das Eigenfahrzeug kritischen Verkehrssituation führt.

Das aktuelle Verkehrsszenario als Resultat der Verkehrssimulation und der Erzeugung kritischer Verkehrssituationen wird von den Sensormodellen zur Stimulation des Assistenzsystems verarbeitet.

Die Reaktion des Assistenzsystems wird von einer Fahrdynamiksimulation verarbeitet, deren Ergebnisse wiederum an die Verkehrsflußsimulation weitergegeben werden. Von den Sensormodellen wird der aktualisierte Bewegungszustand benötigt, um die Relativbewegung des Fremdverkehrs zum Eigenfahrzeug zu bestimmen.

Eine Anpassung der Entwicklungsplattform an weitere Assistenzsysteme, Sensoren oder Versuchsfahrzeuge ist durch Anpassung der entsprechenden Software-Module möglich. Zur Adaption des Echtzeitsimulators an ein anderes Versuchsfahrzeug ist nur die Positionierung der Sensoren in den Sensormodellen und die Parametrierung des Fahrzeuges in der Fahrdynamiksimulation anzupassen.

Der modulare Aufbau der Entwicklungsplattform gestattet darüber hinaus die Integration bereits existierender Lösungen zum Betrieb weiterer Steuergeräte am Echtzeitsimulator. Eine Erweiterung des Simulators bis hin zu einer Gesamtfahrzeugsimulation ist somit realisierbar.

1.1. Stand der Technik

Der naheliegendste Ansatz zum Funktionstest eines Fahrerassistenzsystems besteht in der Durchführung von Fahrversuchen, die durch Einsatz der realen Systemumgebung das Zusammenspiel zwischen der Umfeldsensorik, dem Assistenzsystem und dem Eigenfahrzeug exakt nachbilden. Der Verkehrsfluß muß dabei durch mehrere Fremdfahrzeuge nachgebildet werden.

Eine Reaktion des Assistenzsystems wird vor allem durch Verkehrssituationen mit hohem Gefahrenpotential provoziert, wie sie bei Kolonnenfahrten oder Spurwechseln entstehen können. Solche Situationen müssen vom Testpersonal durch das Zusammenspiel zwischen Eigenfahrzeug und Fremdfahrzeugen erzeugt werden. Dabei werden Personal und Versuchsfahrzeuge einem hohen Risiko ausgesetzt. Darüber hinaus können solche Fahrmanöver nicht exakt durch menschlichen Einsatz reproduziert werden, was für eine Validierung der Funktionalität des Assistenzsystems erforderlich ist.

Um bei einem Funktionstest eines Fahrerassistenzsystems das Risiko für das Testpersonal und das Versuchsfahrzeug zu minimieren, kann das Testfahrzeug in eine *HiL*-ähnliche Umgebung integriert werden, so daß auf Testfahrten im Straßenverkehr und eine reale Erzeugung kritischer Verkehrssituationen verzichtet werden kann. Dieser Ansatz wird dann als *Vehicle-in-the-Loop* bezeichnet.

[Verburg u. a. 2002] betreiben dazu das Versuchsfahrzeug auf einem Rollenprüfstand. Der Fremdverkehr wird durch Trägerplattformen nachgebildet, die die Karosserie eines realen Fahrzeuges in longitudinaler und lateraler Richtung zum Rollenprüfstand transportieren können.

Der Betrieb des Testfahrzeuges auf einem Rollenprüfstand erlaubt es die Fremdfahrzeuge nunmehr mit der gewünschten Relativgeschwindigkeit zum Testfahrzeug zu bewegen. Verkehrsszenarien können durch Messung der Geschwindigkeit des Versuchsfahrzeuges und Einprägung von Trajektorien auf die Trägerplattformen erzeugt werden. Da reale Karosserieteile durch diese Trägerplattformen bewegt werden, erzeugen die Umfeldsensoren des Assistenzsystems reale Meßwerte. Die Rückkopplung des Gesamtsystems wird nach Abbildung 1.1 zwischen dem Eigenfahrzeug und seiner Bewegung im Verkehrsfluß aufgetrennt.

Einen ähnlichen Ansatz verfolgen [Bock u. a. 2005]. Hier wird jedoch das Gesamtsystem zwischen der Umfeldsensorik und dem Assistenzsystem aufgetrennt, s. Abbildung 1.1.

Das Versuchsfahrzeug mit einem Assistenzsystem bewegt sich dabei auf einer Freifläche. Der Bewegungszustand wird durch geeignete Meßverfahren aufgenommen und an eine Verkehrssimulation übergeben. Diese Verkehrssimulation berechnet durch ein Sensormodell die vom Assistenzsystem benötigten Eingangswerte, die über die Schnittstelle des Assistenzsystems zum Antriebsstrang des Versuchsfahrzeuges entgegen genommen werden.

Beide vorgestellten Ansätze reduzieren das Risiko für das Personal und das Versuchsfahrzeug auf ein Minimum, benötigen zur Realisierung jedoch ein Versuchsfahrzeug mit integriertem Assistenzsystem. Der erste Ansatz benötigt zudem noch einen Rollenprüfstand und bewegliche Trägerplattformen zur Nachbildung des Fremdverkehrs. Die Reproduzierbarkeit der Testszenarien zur Stimulation eines Assistenzsystems ist bei beiden Ansätzen gegeben.

Die in dieser Arbeit vorgestellte Entwicklungsplattform ist hingegen auf einem *HiL*-Simulator verfügbar, der ohne weitere Fahrzeugkomponenten auskommt. Kapitel 5 zeigt weiterhin, daß die Entwicklungsplattform ebenfalls als *SiL*-Umgebung eingesetzt werden kann. Dadurch ist ein Funktionstest nicht erst durch einen funktionalen Prototypen möglich, sondern kann bereits während der Modellbildung in systemrelevanter Umgebung erfolgen.

1.2. Ziele der Arbeit

Die Entwicklungsplattform ist durch die Zusammenarbeit mehrerer Mitarbeiter der Universität Kassel entstanden.

[Tellmann 2011] beschreibt die Entwicklung parametrierbarer Sensormodelle zur Nachbildung der Umfeldsensorik eines Assistenzsystems sowie die Entwicklung einer angepassten Verkehrssimulation, die zur Stimulation des Assistenzsystems durch die Sensormodelle verarbeitet wird.

Die Erzeugung reproduzierbarer für das Eigenfahrzeug kritischer Testszenarien beschreibt [Schmidt 2010]. Dabei wird insbesondere die *Kritikalität* zur Bemessung des Gefahrenpotentials der Bewegung mehrerer Verkehrsteilnehmer zueinander definiert. Die *Kritikalität* ist dabei abhängig von der Funktion des Assistenzsystems, der Umfeldsensorik und den möglich Stellgrößen des Regeleingriffs, z.B. der maximal möglichen Verzögerung des Eigenfahrzeuges.

Ziel dieser Arbeit ist die Entwicklung einer dreidimensionalen Visualisierung des Fahrzeugumfeldes und einer Simulationsumgebung zur Konfiguration der verschiedenen Software-Module der Entwicklungsplattform.

Kapitel 2 erläutert das verwendete Straßenmodell, das die Basis zur Positionierung der Verkehrsteilnehmer durch die verschiedenen Software-Module bildet. Die in diesem Kapitel vorgestellten Koordinatensysteme werden von allen Software-Modulen zur Interaktion mit den Verkehrsteilnehmern verwendet.

Die vorhandene Verkehrssimulation macht eine Visualisierung des Verkehrsszenarios erforderlich, da der Bewegungszustand vieler Verkehrsteilnehmer nur schwer anhand ihrer Zustandstrajektorien beurteilt werden kann. Die Visualisierung ist dazu mit der Bedienung des *HiL*-Simulators zu koppeln, um die Simulationsergebnisse möglichst in Echtzeit verarbeiten zu können. In Kapitel 3 werden die eingesetzten Algorithmen zur Visualisierung des Fahrzeugumfeldes erläutert und die Verarbeitung der Geometriedaten durch das eingesetzte Graphiksystem beschrieben.

Um einem Entwicklungsingenieur eine konsistente Versuchsbeschreibung zu ermöglichen, lädt jedes Software-Modul eine Simulationsumgebung, die den momentan durchgeführten Versuch des zu untersuchenden Assistenzsystems konfiguriert. Die Komponenten dieser Simulationsumgebung werden in Kapitel 4 beschrieben. Die Simulationsumgebung abstrahiert bewußt von den in Kapitel 2 und Kapitel 3 gelegten Grundlagen, um einem Anwender der Entwicklungsplattform einen einfachen Zugang zur Konfiguration des Versuchsaufbaus zu ermöglichen.

Erste Erfahrungen über die Anwendung der Visualisierung und der Simulationsumgebung zum Test des Prototypen eines Assistenzsystems werden in Kapitel 5 geschildert. Dieser Prototyp wurde innerhalb eines Forschungsprojektes in Zusammenarbeit mit mehreren Automobilzulieferern entwickelt und konnte durch offengelegte Schnittstellen als Testsystem verwendet werden.

2. Straßenmodell

Dieses Kapitel erläutert die mathematische Beschreibung des Rundkurses der Simulationsumgebung. Zum Verständnis einiger Abschnitte wird die Kenntnis von Anhang A vorausgesetzt.

2.1. Einleitung

Kraftfahrzeuge werden durch ihre Fahrer überwiegend im öffentlichen Straßennetz bewegt. Dieses Netz ist an die benötigten Transportkapazitäten eines Landes bzw. einer Region angepasst.

Z.B. werden Ballungszentren der Bundesrepublik Deutschland i.d.R. mit Autobahnen verbunden, die den täglich anstehenden Güter- und Personenverkehr auf mehreren Fahrspuren pro Fahrtrichtung abwickeln können. Dünn besiedelte Regionen hingegen werden aus Kostengründen nur durch Bundes- oder Landstraßen mit einer Fahrspur pro Richtung verbunden.

Diese Aufteilung führt zur Bildung verschiedener Straßenklassen, zwischen denen der Verkehr durch geeignete Verbindungselemente vermittelt und durch die sog. *Verkehrsregeln* reglementiert wird. Der Straßenverlauf, die sog. *Trassierung*, beschreibt dabei die geometrischen Eigenschaften des Straßennetzes, die Verkehrsregeln die sich selbst von den Verkehrsteilnehmern auferlegten Konventionen zur gefahrlosen Nutzung dieses Straßennetzes.

In diesem Abschnitt wird eine geometrische Beschreibung der Trassierung vorgestellt, wie sie z.B. zur Entwicklung oder zum funktionalen Test von Fahrerassistenzsystemen verwendet werden kann. Ziel dieser Beschreibung ist dabei nicht eine exakte Nachbildung des Straßennetzes, dessen Eigenschaften in der Bundesrepublik durch die *Richtlinie für die Anlage von Straßen*[1] festgelegt sind, sondern vielmehr die Schaffung eines Entwicklungswerkzeuges, das die bereits erwähnten Aufgaben unterstützen kann.

Für einen funktionalen Test von Fahrerassistenzsystemen zur Überwachung des Fahrzeugumfeldes ist eine Nachbildung der verschiedenen Stra-

[1] Abk.: RAS

ßenklassen und insbesondere deren Verbindung durch Knotenpunkte, z.B. von Autobahnkreuzen, nicht erforderlich.

Einscherende oder überholende Fahrzeuge können bereits auf einem Kurs mit mehreren Fahrspuren nachgebildet werden. Die Anzahl der erzeugten Fahrspuren entspricht dann evtl. nicht der durch die RAS festgelegten Konvention, was für eine Bewertung der Dynamik von Fahrerassistenzsystemen jedoch keine Rolle spielt.

Um den Einfluß der Fahrbahngeometrie auf einige Assistenzsysteme, z.B. einer Geschwindigkeitsregelung untersuchen zu können, muß ein Straßenverlauf mit Bereichen unterschiedlicher Krümmung, Steigung und Querneigung erzeugt werden. Dabei ist ein kontinuierlicher Übergang zwischen Bereichen unterschiedlicher Geometrie zu ermöglichen, um Unstetigkeiten in der Simulation zu vermeiden.

Der Straßenverlauf soll ebenfalls einfach an die Bedürfnisse eines speziellen Tests angepasst werden können. Daher wird einem Anwender die Möglichkeit zur synthetischen Erzeugung einer Trassierung durch Platzierung mehrerer Stützstellen, die den Straßenverlauf interpolieren, gegeben. Darüber hinaus können diese Stützstellen aus digitalisiertem Kartenmaterial oder Messungen bei Testfahrten gewonnen werden.

Unter diesen Voraussetzungen soll sich bei dem Straßenverlauf auf einen geschlossenen Rundkurs beschränkt werden, dessen Stützstellen gerade oder gekrümmte Abschnitte interpolieren. Für jede Fahrtrichtung kann dabei die Anzahl der möglichen Fahrspuren sowie deren Breite vorgegeben werden.

2.2. Struktur des Rundkurses

Ein Rundkurs besteht dabei aus N Stützstellen, die durch Geraden oder Kurven miteinander verbunden sind. Jeder Stützstelle $\vec{r}_i$, mit $i = 1, \ldots, N$, wird ein Parameterwert t_i zugeordnet, der einen Punkt auf der Mittellinie des Rundkurses interpoliert. Für die Parameter muß dabei $t_i < t_{i+1}$ gelten.

Abbildung 2.1 zeigt den Abschnitt eines beliebigen Rundkurses.

Die Abschnitte des Rundkurses müssen den Verlauf der Fahrbahnmittellinie approximieren können. Für gerade Abschnitte können die Stützstellen einfach durch Geraden verbunden werden, s. $\vec{r}_i$ und $\vec{r}_{i+1}$ in Abbildung 2.1.

Zur Approximation gekrümmter Abschnitte, s. $\vec{r}_{i+1}$ bis $\vec{r}_{i+4}$ in Abbildung 2.1, existiert eine Vielzahl mathematischer Beschreibungen, s. [Holler 2005]. Zur Approximation der Krümmung des Rundkurses wurde das Verfahren von [Wang 2005] verwendet.

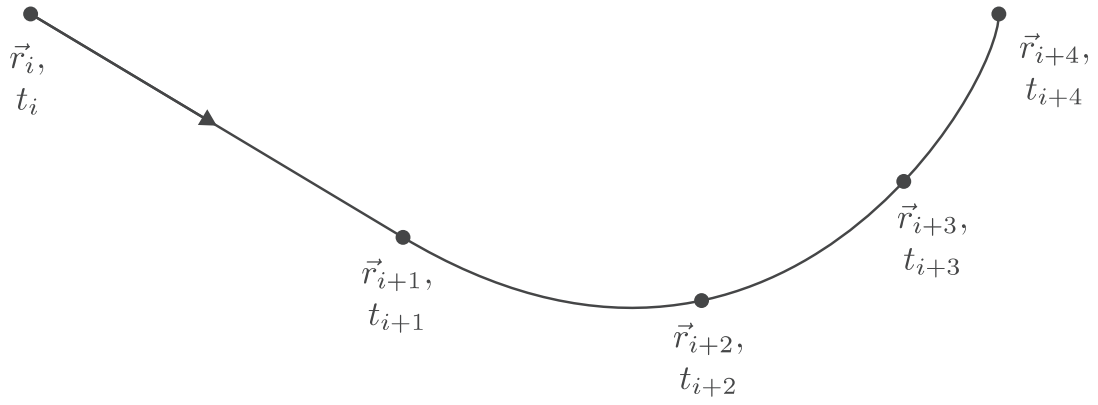

Abbildung 2.1.: Abschnitt eines beliebigen Rundkurses.

Um einen stetigen Übergang zwischen geraden und gekrümmten Abschnitten zu gewährleisten, ist über die gemeinsame Stützstelle hinaus die Stetigkeit der 1. und möglicherweise der 2. Ableitung zu fordern, damit beim Durchfahren einer Stoßstelle keine Unstetigkeit in der Simulation auftritt. In diesem Fall spricht man von einer $\mathcal{C}_1$- bzw. $\mathcal{C}_2$-Stetigkeit.

Damit Fahrzeuge durch eine Fahrdynamiksimulation oder einer Verkehrssimulation auf dem Rundkurs bewegt werden können, ist es notwendig, die Fahrtrichtung zu definieren. Es bietet sich an die Fahrtrichtung mit aufsteigenden Parameterwerten zu definieren. Der Tangentenvektor an die Fahrbahnmittellinie zeigt dann von $\vec{r}_i$ nach $\vec{r}_{i+1}$, s. Abbildung 2.1.

Es ist ausreichend die N Parameterwerte zu $t_1 = 0, \ldots, t_N = N - 1$ zu setzen. Dadurch können auf der Fahrbahnmittellinie Ortsvektoren $\vec{r}$ interpoliert werden. Für eine physikalisch plausible Anwendung, z.B. einer Verkehrssimulation, ist eine Parametrierung nach Bogenlänge wünschenswert, da der zurückgelegte Weg auf der Mittellinie durch die Verwendung gekrümmter Abschnitte i.d.R. nichtlinear vom Parameterwert t abhängig ist. Wird eine dreidimensionale Kurve durch die Funktion $\vec{S}(t)$ beschrieben, errechnet sich die Bogenlänge $L_{0,1}$ im Intervall $[t_0, t_1]$ zu

$$L_{0,1} = \int_{t_0}^{t_1} \left| \vec{S}\,'(t) \right| \, dt = \int_{t_0}^{t_1} \sqrt{S'^2_x(t) + S'^2_y(t) + S'^2_z(t)} \, dt. \tag{2.1}$$

Eine Parametrierung des Rundkurses nach zurückgelegter Bogenlänge ℓ auf der Mittellinie ist somit wünschenswert. Die Parameterwerte t_i sind folglich auf äquivalente Bogenlängen ℓ_i abzubilden.

Die einzelnen Abschnitte des Rundkurses können dabei sequentiell nach Bogenlänge parametriert werden. Die zum Startpunkt des Rundkurses gehörige Bogenlänge ist von einem Anwender vorzugeben oder wird zu 0 gesetzt.

2.3. Approximation gerader Abschnitte

Die Approximation gerader Abschnitte ist trivial. Soll der Abschnitt zwischen den Stützstellen i und $i+1$ approximiert werden, so kann der Richtungsvektor der Geraden zu

$$\vec{g} = \frac{\vec{r}_{i+1} - \vec{r}_i}{|\,\vec{r}_{i+1} - \vec{r}_i\,|}$$

bestimmt werden. Die Bogenlänge eines geraden Abschnitts ist durch

$$L_{i,i+1} = \ell_{i+1} - \ell_i = |\,\vec{r}_{i+1} - \vec{r}_i\,|$$

festgelegt. Die Funktion des Abschnitts lautet damit

$$\vec{S}(\ell) = \vec{r}_i + (\ell - \ell_i) \cdot \vec{g} \qquad \forall \quad \ell \in [\ell_i, \ell_{i+1}] \tag{2.2}$$

und ist in Bogenlänge parametriert.

2.4. Approximation gekrümmter Abschnitte

Gekrümmte Abschnitte werden nach dem Verfahren von [Wang 2005] durch sog. kubische Splines approximiert. Es werden dabei N_S aufeinanderfolgende Stützstellen des Rundkurses zu einem Spline zusammengefasst. Zwei benachbarte Stützstellen werden durch ein kubisches Polynom, einer sog. Spline-Funktion, interpoliert.

Die Indizierung der Laufvariablen i wird in diesem Abschnitt zu $i = 1, \ldots, N_S$ gewählt, um eine handliche Definition der Spline-Funktionen zu ermöglichen. Die N_S Stützstellen werden nach der Berechnung der Spline-Koeffizienten fortlaufend in den Rundkurs integriert.

2.4.1. Interpolation der Stützstellen

Durch N_S Stützstellen werden $N_S - 1$ Parameterintervalle definiert, die die Spline-Funktionen definieren. Die Breite eines Intervalls i ist durch $h_i = t_{i+1} - t_i$ festgelegt. Für den eindimensionalen Fall ist die Spline-Funktion, deren 1. und 2. Ableitung wie folgt gegeben, s. [Bronstein u. a. 2005].

$$\begin{aligned} S_i(t) &= (t-t_i)^3 d_i + (t-t_i)^2 c_i + (t-t_i) b_i + a_i, \qquad i = 1, \ldots, N_S - 1 \\ S_i'(t) &= 3(t-t_i)^2 d_i + 2(t-t_i) c_i + b_i \\ S_i''(t) &= 6(t-t_i) d_i + 2c_i \end{aligned} \tag{2.3}$$

Die Bestimmung der Spline-Koeffizienten a_i, b_i, c_i und d_i erfolgt nach [Bronstein u. a. 2005] durch Aufstellen von vier Bedingungen.

1. Die kubischen Polynome verlaufen durch die Stützstellen r_i.

$$S_i(t_i) = a_i = r_i\,, \qquad i = 1, \ldots, N_S - 1$$

2. Die 2. Ableitung ist an den Stoßstellen stetig.

$$\begin{aligned} S_i''(t_{i+1}) &= S_{i+1}''(t_{i+1})\,, \qquad i = 1, \ldots, N_S - 1 \\ d_i &= \frac{c_{i+1} - c_i}{3h_i} \end{aligned}$$

3. Zwei Spline-Funktionen stoßen an einer Stützstelle zusammen.

$$\begin{aligned} S_i(t_{i+1}) &= S_{i+1}(t_{i+1})\,, \qquad i = 1, \ldots, N_S - 1 \\ b_i &= \frac{a_{i+1} - a_i}{h_i} - h_i \frac{c_{i+1} + 2c_i}{3} \end{aligned}$$

4. Die 1. Ableitung ist an den Stoßstellen stetig.

$$\begin{aligned} &S_i'(t_{i+1}) = S_{i+1}'(t_{i+1})\,, \qquad i = 1, \ldots, N_S - 2 \\ &h_{i+1} c_{i+2} + 2(h_{i+1} + h_i) c_{i+1} + h_i c_i = \\ &\qquad\qquad 3\left(\frac{a_{i+2} - a_{i+1}}{h_{i+1}} - \frac{a_{i+1} - a_i}{h_i}\right) \end{aligned}$$

Diese Bedingung führt auf ein tridiagonales Gleichungssystem, das z.B. mit dem LU-Algorithmus aus [Press u. a. 2007] gelöst werden kann.

$$\begin{bmatrix} 2(h_2+h_1) & h_2 & 0 & \dots & 0 \\ h_2 & 2(h_3+h_2) & h_3 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & h_{N_S-3} & 2(h_{N_S-2}+h_{N_S-3}) & h_{N_S-2} \\ 0 & \dots & 0 & h_{N_S-2} & 2(h_{N_S-1}+h_{N_S-2}) \end{bmatrix}$$

$$\cdot \begin{pmatrix} c_2 \\ c_3 \\ \vdots \\ c_{N_S-2} \\ c_{N_S-1} \end{pmatrix} = \begin{pmatrix} 3\left(\frac{a_3-a_2}{h_2} - \frac{a_2-a_1}{h_1}\right) - h_1 c_1 \\ 3\left(\frac{a_4-a_3}{h_3} - \frac{a_3-a_2}{h_2}\right) \\ \vdots \\ 3\left(\frac{a_{N_S-1}-a_{N_S-2}}{h_{N_S-2}} - \frac{a_{N_S-2}-a_{N_S-3}}{h_{N_S-3}}\right) \\ 3\left(\frac{a_{N_S}-a_{N_S-1}}{h_{N_S-1}} - \frac{a_{N_S-1}-a_{N_S-2}}{h_{N_S-2}}\right) - h_{N_S-1} c_{N_S} \end{pmatrix}$$

Die Koeffizienten a_i können direkt aus der ersten Bedingung bestimmt werden. Die Koeffizienten b_i und d_i werden hingegen als Funktionen der Koeffizienten a_i und c_i formuliert, wobei das obige, lineare Gleichungssystem zur Bestimmung der c_i unterbestimmt ist.

Durch Setzen von $c_1 = c_{N_S} = 0$ erhält man einen sog. *natural Spline*[2], dessen Krümmung durch den Verlauf der interpolierten Punkte r_i vorgegeben ist.

Eine weitere Möglichkeit ist die Vorgabe der Steigung des Splines an den Randpunkten. Man erhält den sog. *clamped Spline*[3] und gibt dann die Koeffizienten

[2] dt.: natürlicher Spline
[3] dt.: eingespannter Spline

$$
\begin{aligned}
c_1 &= \frac{1}{2}\left[\frac{3}{h_1}\left(\frac{a_2-a_1}{h_1}-b_1\right)-c_2\right] \qquad \text{und} \\
c_{N_S} &= \frac{1}{2}\left[\frac{3}{h_{N_S-1}}\left(b_{N_S}-\frac{a_{N_S}-a_{N_S-1}}{h_{N_S-1}}\right)-c_{N_S-1}\right]
\end{aligned}
\tag{2.4}
$$

vor. b_1 und b_{N_S} sind die Steigungen an den Randpunkten.

Um dreidimensionale Kurven zu interpolieren, werden drei Spline-Funktionen vektoriell überlagert.

$$
\vec{S}_i(t) = (t-t_i)^3 \cdot \vec{d}_i + (t-t_i)^2 \cdot \vec{c}_i + (t-t_i) \cdot \vec{b}_i + \vec{a}_i
$$

Die Koeffizienten können dann separat für jede Dimension bestimmt werden. Die Steigungen b_1 und b_{N_S} des Splines an seinen Enden sind dann vektoriell als Tangentenvektoren $\vec{b}_1$ und $\vec{b}_{N_S}$ zu interpretieren. Diese Tangentenvektoren können dazu verwendet werden den Übergang an den Stoßstellen zwischen einem geraden und gekrümmten Abschnitt $\mathcal{C}_1$-stetig vorzugeben.

2.4.2. Parametrierung nach Bogenlänge

Bisher wurden die Spline Funktionen allgemein durch den Parameter t parametriert, für den monoton steigende Werte vorgegeben werden können. [Wang 2005] stellt einen Algorithmus zur Parametrierung nach Bogenlänge vor. Dabei werden die Koeffizienten der Spline-Funktionen

$$
\vec{S}_i(\ell) = (\ell-\ell_i)^3 \cdot \vec{d}_i + (\ell-\ell_i)^2 \cdot \vec{c}_i + (\ell-\ell_i) \cdot \vec{b}_i + \vec{a}_i \tag{2.5}
$$

bestimmt. Der Parameter ℓ entspricht dann der auf der Fahrbahnmittellinie zurückgelegten Bogenlänge.

[Wang 2005] geht dabei in drei Schritten vor.

1. Zuerst wird ein sog. Ausgangs-Spline mit den Spline-Funktionen

$$
\tilde{\vec{S}}_j(t) = (t-\tilde{t}_j)^3 \cdot \tilde{\vec{d}}_j + (t-\tilde{t}_j)^2 \cdot \tilde{\vec{c}}_j + (t-\tilde{t}_j) \cdot \tilde{\vec{b}}_j + \tilde{\vec{a}}_j, \quad j = 1, \ldots, M_S - 1\,, \tag{2.6}
$$

den zu interpolierenden Stützstellen $\tilde{\vec{r}}_j$ und aufsteigender Parametrierung $\tilde{t}_j = j$ bestimmt.

Jeder Stützstelle kann die Bogenlänge

$$\tilde{\ell}_j \quad = \quad \int_{\tilde{t}_1}^{\tilde{t}_j} \left| \tilde{\vec{S}}'(t) \right| \, dt, \qquad j = 1, \ldots, M_S \tag{2.7}$$

zugeordnet werden.

Die Gesamtbogenlänge $\tilde{L}$ des Ausgangs-Splines entspricht dann $\tilde{\ell}_{M_S}$.

2. Jetzt können N_S äquidistant entfernte Stützstellen $\vec{r}_i$ auf dem Ausgangs-Spline bestimmt werden, die dann die Stützstellen des Rundkurses bilden. Dazu ist die Bestimmung zugehöriger Parameterwerte t_i auf dem Ausgangs-Spline notwendig.

   Jeder Stützstelle i werden die Bogenlängen

$$\ell_i = (i-1)\frac{\tilde{L}}{N_S - 1}, \qquad i = 1, \ldots, N_S$$

   zugewiesen.

   Zu jeder Bogenlänge ℓ_i kann mit der Bedingung

$$\tilde{\ell}_j \leq \quad \ell_i \quad < \tilde{\ell}_{j+1}$$

   das Intervall j des Parameterwertes t_i innerhalb des Ausgangs-Splines bestimmt werden, s. Gleichungen (2.6) und (2.7).

   Die Bestimmung der Bogenlänge ℓ_i lässt sich damit umformen.

$$\ell_i = \int_{\tilde{t}_1}^{t_i} \left| \tilde{\vec{S}}'(t) \right| \, dt = \int_{\tilde{t}_j}^{t_i} \left| \tilde{\vec{S}}'(t) \right| \, dt + \tilde{\ell}_j = \Delta_i(t_i) + \tilde{\ell}_j$$

   Durch Anwendung des Bisektionsverfahrens kann der Parameterwert t_i bestimmt werden.

Dazu wird das Intervall j halbiert. Die Integration wird zunächst im Bereich $t_i \in [\tilde{t}_j, \frac{\tilde{t}_j+\tilde{t}_{j+1}}{2}]$ durchgeführt. Ist

$$\Delta_i(t_i) < \ell_i - \tilde{\ell}_j,$$

befindet sich t_i in der oberen Hälfte. Andernfalls in der unteren Hälfte. Die Bisektion des Suchintervalls ist durchzuführen, bis eine gewünschte Toleranz der berechneten Bogenlänge $\Delta_i(t_i)$ zur vorgegebenen Bogenlänge $\ell_i - \tilde{\ell}_j$ eingehalten wird.

Die Parameterwerte t_i definieren jetzt äquidistante Stützstellen $\vec{r}_i$ mit den zugehörigen Bogenlängen ℓ_i.

3. Mit den N_S Stützstellen $\vec{r}_i$ und den zugehörigen Bogenlängen ℓ_i sind die Spline-Funktionen des Ziel-Splines in der Form von Gleichung (2.5) bestimmt.

Bei der Parametrierung nach Bogenlänge wurde bisher die Bedeutung der Tangentenvektoren $\vec{b}_1$ und $\vec{b}_{N_S}$ vernachlässigt. Da ein Spline nicht nur Stützstellen, sondern auch die 1. Ableitung einer Kurve interpoliert, müssen die Tangentenvektoren aus Gleichung (2.4) entsprechend angepasst werden.

Für eine Kurve $\vec{S}(\ell)$, die nach Bogenlänge parametriert ist, gilt

$$\left| \vec{S}\,'(\ell) \right| = 1\,. \tag{2.8}$$

Die Tangentenvektoren des Ziel-Splines müssen folglich die Bedingung

$$\left| \vec{b}_1 \right| = \left| \vec{b}_{N_S} \right| = 1$$

erfüllen.

Während der Erstellung des Ausgangs-Splines liegt jedoch keine nach Bogenlänge parametrierte Kurve vor. Die Tangentenvektoren $\vec{b}_1$ und $\vec{b}_{N_S}$, die vom Anwender vorgegeben werden, müssen daher gewichtet werden, um einen fehlerhaften Einfluß des Betrags der Tangentenvektoren auf die Krümmung der Kurve zu vermeiden. Die Tangentenvektoren $\tilde{\vec{b}}_1$ und $\tilde{\vec{b}}_{M_S}$ des Ausgangs-Splines werden daher zu

$$\begin{aligned} \tilde{\vec{b}}_1 &= \left| \tilde{\vec{r}}_2 - \tilde{\vec{r}}_1 \right| \cdot \vec{b}_1 \qquad \text{und} \\ \tilde{\vec{b}}_{M_S} &= \left| \tilde{\vec{r}}_{M_S} - \tilde{\vec{r}}_{M_S-1} \right| \cdot \vec{b}_{N_S} \end{aligned}$$

gesetzt.

2.4.3. Beispiel

Abbildung 2.2 zeigt die Parametrierung nach Bogenlänge am Beispiel eines analytischen Kurvenverlaufes.

Als Beispiel eines analytischen Kurvenverlaufes wird die Archimedische Spirale verwendet, deren Radius $a \cdot \varphi$ mit steigendem Parameterwert φ wächst. In diesem Beispiel wird die Konstante a zu $a = 10m$ gesetzt und der Parameter φ im Intervall $\varphi \in [0, 2\pi]$ variiert.

Der Verlauf der analytischen Kurve lautet dann

$$\vec{S}(\varphi) = \begin{pmatrix} a \cdot \varphi \cdot \cos\varphi \\ a \cdot \varphi \cdot \sin\varphi \\ 0 \end{pmatrix}$$

und ist in Abbildung 2.2 schwarz dargestellt.

Auf dieser Kurve werden die M_S Stützstellen des Ausgangs-Splines in äquidistanten Parameterintervallen $\Delta\varphi = \frac{2\pi}{M_S - 1}$ bestimmt, so daß für jede Stützstelle

$$\tilde{\vec{r}}_j = \vec{S}\left((j-1) \cdot \Delta\varphi\right), \quad j = 1, \ldots, M_S$$

gilt. Die zugehörigen Parameterwerte werden zu einer monoton steigenden Zahlenfolge gewählt, d.h. $\tilde{t}_j = j$.

Mit den Stützstellen $\tilde{\vec{r}}_j$, die in Abbildung 2.2 blau dargestellt sind, und den zugehörigen Parametern $\tilde{t}_j$, sind die Spline-Funktionen nach Gleichung (2.6) bestimmt.

Jetzt werden durch den Algorithmus aus Abschnitt 2.4.2 die N_S Stützstellen $\vec{r}_i$ des Ziel-Splines mit den zugehörigen Parameterwerten ℓ_i berechnet. Diese Stützstellen sind mit äquidistanter Bogenlänge zueinander auf dem Spline angeordnet und werden in Abbildung 2.2 rot dargestellt.

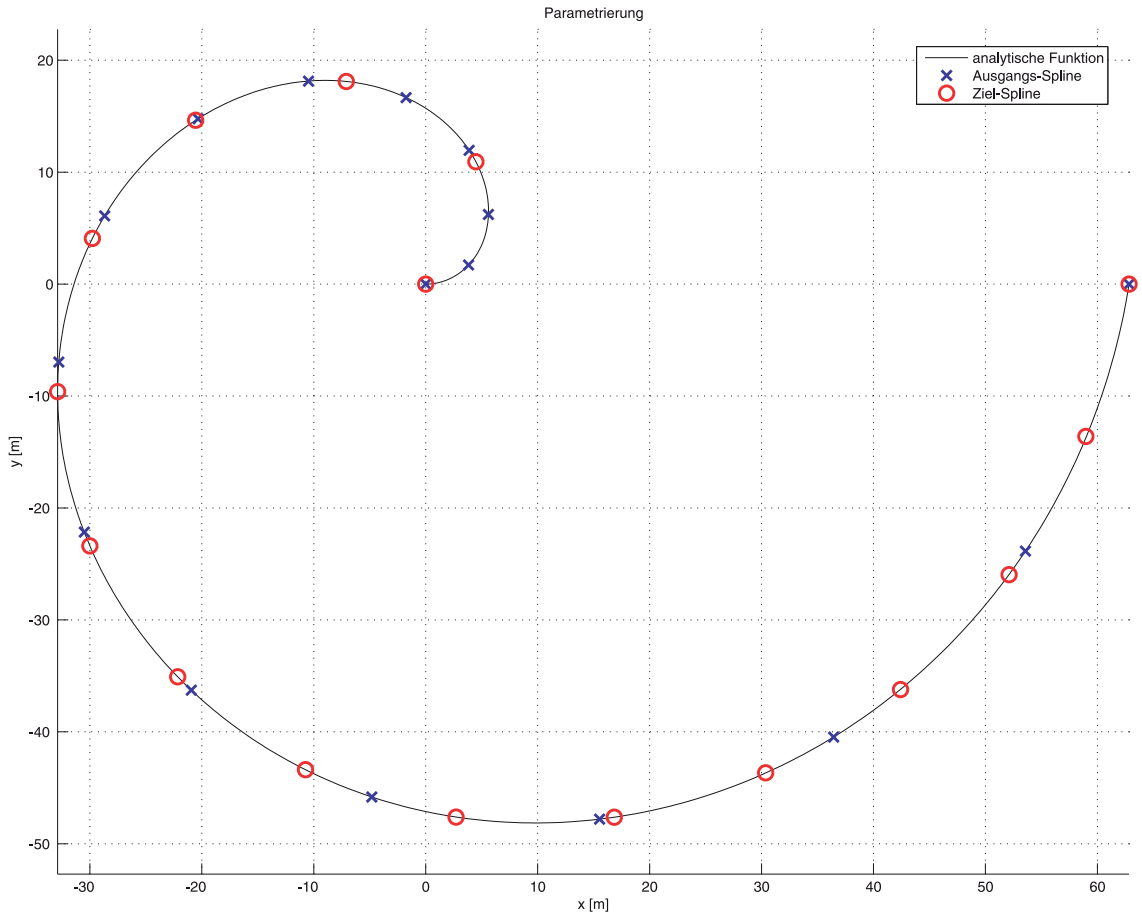

Abbildung 2.2.: Parametrierung nach Bogenlänge am Beispiel eines analytischen Kurvenverlaufes.

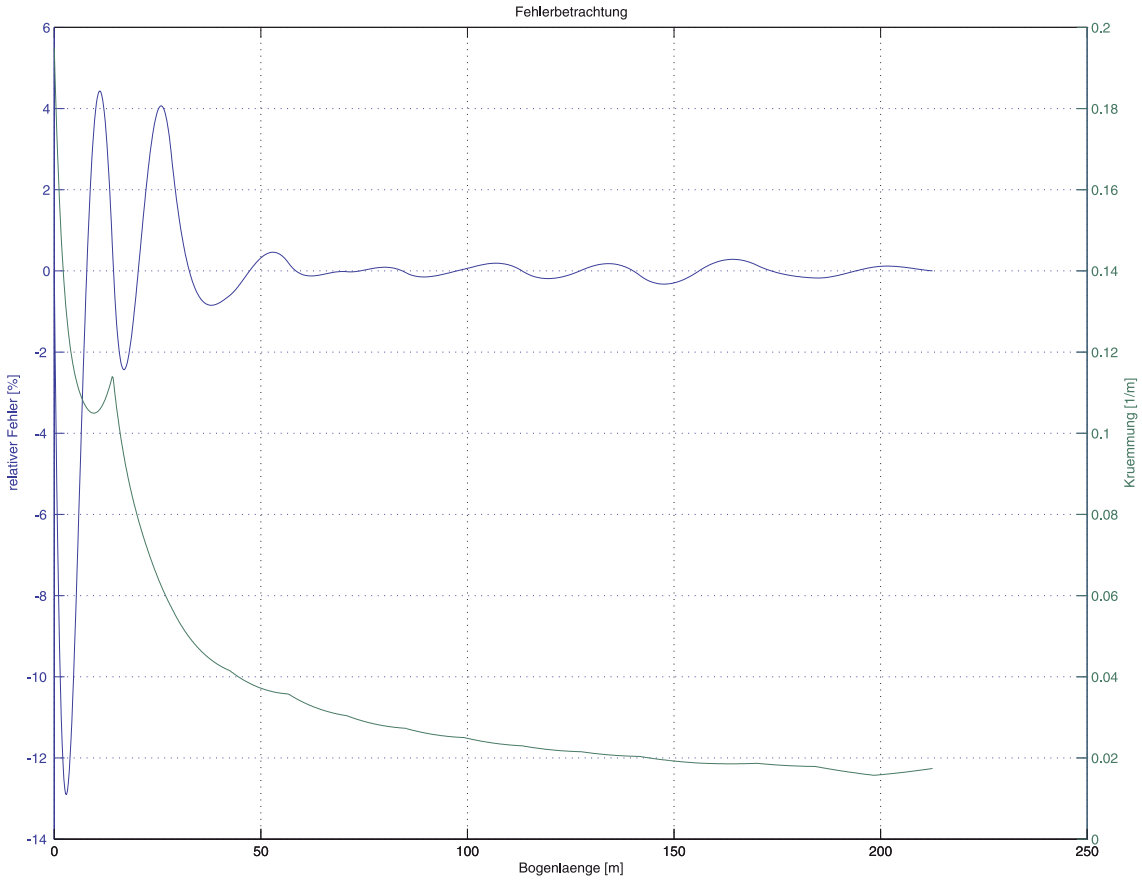

Abbildung 2.3.: Fehlerbetrachtung der Parametrierung ($M_S = 16$, $N_S = 16$).

Für die Parametrierung des Ziel-Splines soll nun eine Fehlerbetrachtung durchgeführt werden. Aus Gleichung (2.8) kann die Fehlerfunktion

$$e(\ell) \quad = \quad \left|\vec{S}\,'(\ell)\right| - 1$$

hergeleitet werden, die die Abweichung der Parametrierung von ihrem Sollwert beschreibt. Diese Funktion ist in Abbildung 2.3 blau dargestellt.

Parallel zum relativen Fehler der Parametrierung ist in Abbildung 2.3 die berechnete Krümmung

$$\kappa(\ell) \quad = \quad \frac{\left|\vec{S}\,'(\ell) \times \vec{S}\,''(\ell)\right|}{\left|\vec{S}\,'(\ell)\right|^3}$$

des Ziel-Splines in grün aufgetragen.

In diesem Beispiel kann für den analytischen Kurvenverlauf die Krümmung κ zu

$$\kappa(\varphi) \quad = \quad \frac{1}{a \cdot \varphi}$$

angegeben werden. Eine Grenzwertbetrachtung liefert

$$\begin{aligned} \kappa(\varphi) \quad &\xrightarrow[\varphi \to 0]{} \quad \infty \qquad \text{und} \\ \kappa(\varphi) \quad &\xrightarrow[\varphi \to \infty]{} \quad 0\,. \end{aligned}$$

Wird der Verlauf des relativen Fehlers in Abbildung 2.3 mit dem der berechneten Krümmung verglichen, liegt der größte Fehler der Parametrierung im Bereich der stärksten Krümmung der zu interpolierenden Kurve. Die hier gewählten Anzahl an Stützstellen des Ziel-Splines ist somit nicht ausreichend, um dem Verlauf der Archimedischen Spirale exakt zu folgen. Für eine hinreichend große Anzahl an Stützstellen kann der Fehler jedoch weiter gesenkt werden.

Die Unstetigkeitsstelle im Verlauf der Krümmung ist durch das numerisch instabile Verhalten der Krümmung des analytischen Kurvenverlaufes begründet, daß nicht durch eine numerische Interpolation nachgebildet werden kann.

Eine ausführliche Fehlerbetrachtung des Algorithmus aus Abschnitt 2.4.2 gibt [Wang 2005].

2.5. Erstellung des Rundkurses

[Holler 2005] stellt einen graphischen Editor vor, mit dem der Verlauf eines Rundkurses durch Platzieren von Stützstellen vorgegeben werden kann, s. Abbildung 2.4.

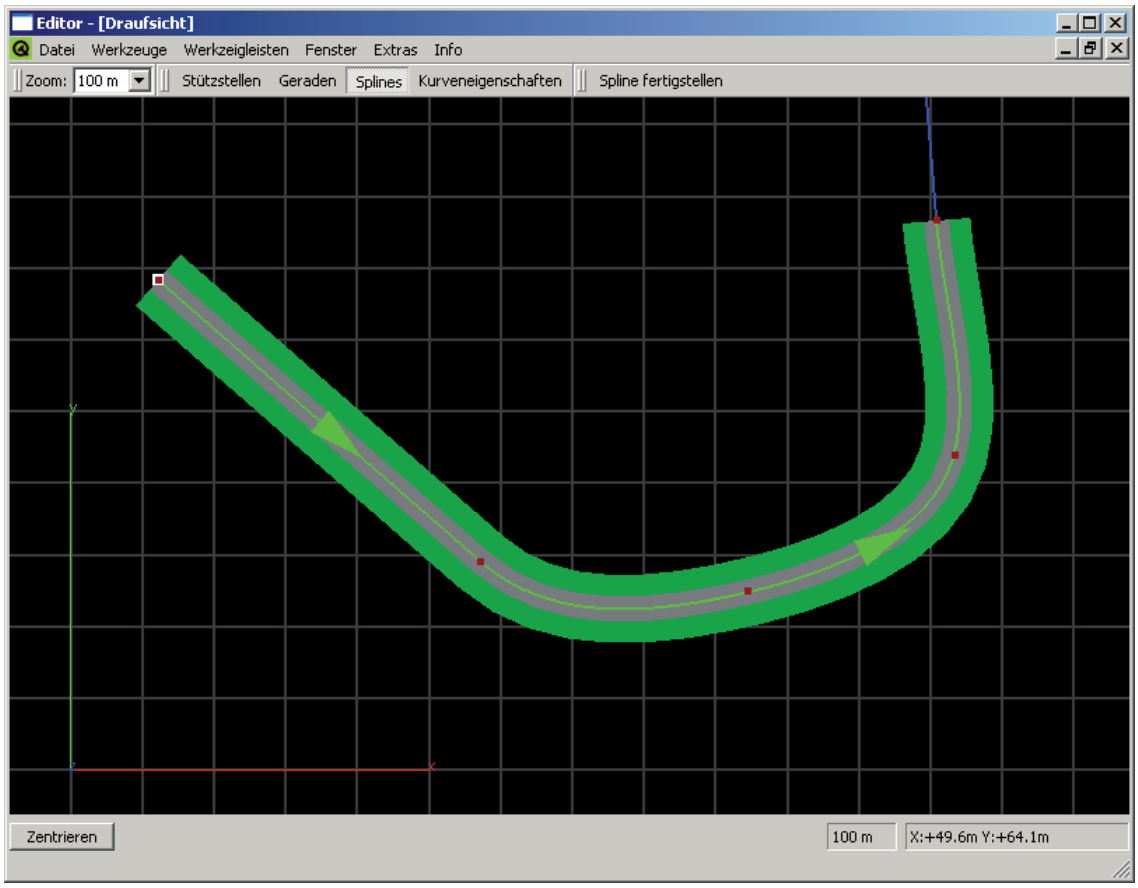

Abbildung 2.4.: Unvollständiger Rundkurs bei Bearbeitung im graphischen Editor.

Zwei Stützstellen können dabei zu einem geraden Abschnitt des Rundkurses verbunden werden. Mehrere Stützstellen, zumindest vier, können zu einem gekrümmten Abschnitt verbunden werden.

Eine ausgewählte Stützstelle muß als Startpunkt des Rundkurses definiert werden. An ihr fällt die Stützstelle mit der kleinsten, der sog. Startbogenlänge und der größten Bogenlänge zusammen. Diese Startbogenlänge muß dabei vom Benutzer vorgegeben werden.

Die Geradendefinition nach Gleichung (2.2) interpoliert Stützstellen relativ zur Anfangsbogenlänge ℓ_i.

Bei den gekrümmten Abschnitten muß zu jeder der berechneten Bogenlängen ℓ_i die Anfangsbogenlänge des Abschnitts hinzuaddiert werden, um

die Integration in den Rundkurs abzuschließen. Die zur Bogenlänge ℓ zugehörige Spline-Funktion i lässt sich dann durch eine einfache Rechnung bestimmen.

$$i(\ell) = \min\left(\left\lfloor \frac{\ell - \ell_1}{\ell_{N_S} - \ell_1} \cdot (N_S - 1) + 1 \right\rfloor, N_s - 1\right)$$

Dabei wurde die äquidistante Entfernung der Stützstellen zueinander ausgenutzt. Da zur Interpolation der letzten Stützstelle bei $i = N_S$ keine Spline-Funktion definiert werden kann, wird die letzte Stützstelle aus der vorherigen Funktion $i = N_S - 1$ bestimmt.

Der Übergang zwischen einem geraden und einem gekrümmten Abschnitt kann nur $\mathcal{C}_1$-stetig gestaltet werden. Der Richtungsvektor des geraden Abschnitts bestimmt dann den Tangentenvektor des Splines an der Stoßstelle. Ein stetiger Übergang zwischen zwei Geraden ist nur durch dieselbe Richtung realisierbar. Bei Splines ist die Wahl des Anfangs- und Endtangentenvektors beliebig, durch den graphischen Editor kann eine Anpassung an die Richtung bzw. Tangente des vorherigen Abschnitts vorgenommen werden.

Zu jeder Stützstelle ist zusätzlich die Querneigung der Fahrbahnoberfläche gegenüber der x-y-Ebene anzugeben. Dieser Winkel wird zur Erzeugung der Fahrbahnkoordinaten und der Berechnung eines Drahtgittermodells des Rundkurses verwendet.

2.6. Erzeugung des Fahrbahnkoordinatensystems

Bisher beschränkte sich die Beschreibung des Rundkurses auf eine mathematische Modellierung der Fahrbahnmittellinie. Zur Navigation auf einer Fahrbahn ist jedoch eine Beschreibung der Fahrbahnoberfläche erforderlich. Dazu wird aus der analytischen Formulierung der Mittellinie an jeder Stelle ℓ des Rundkurses ein Koordinatensystem erzeugt, das die lokale Änderung der Fahrbahnoberfläche gegenüber dem Inertialsystem beschreibt, s. Abbildung 2.5.

Die Längsachse wird durch den normierten Tangentenvektor an die Kurve $\vec{S}(\ell)$ des Rundkurses gebildet. Für gerade Abschnitte ist dies der Vektor $\vec{g}$ aus Gleichung (2.2). Bei gekrümmten Abschnitten entspricht die Längsachse der 1. Ableitung $\vec{S}'_i(\ell)$ der Spline-Funktion i, s. Gleichungen (2.3) und (2.5).

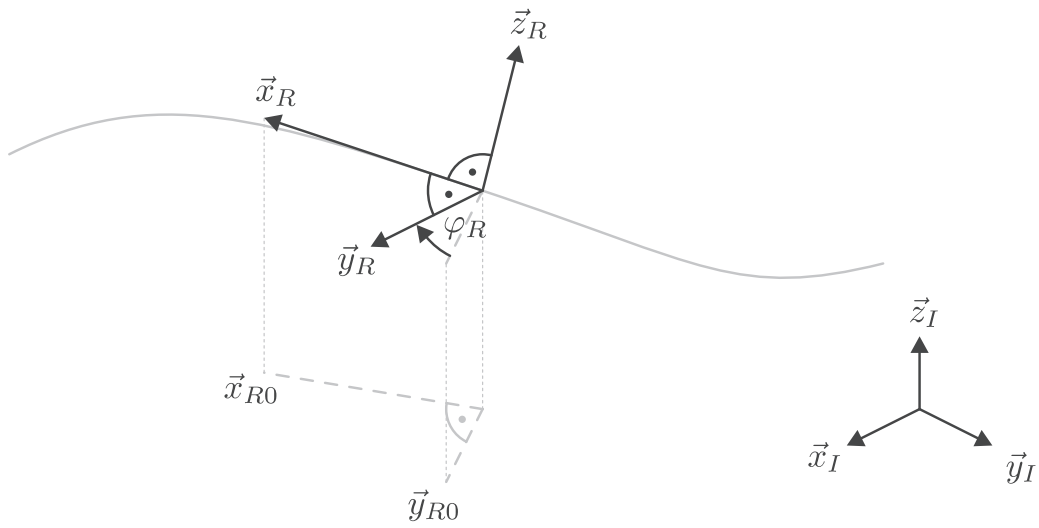

Abbildung 2.5.: Erzeugung des Fahrbahnkoordinatensystems.

Damit folgt

$$\vec{x}_R(\ell) = \frac{\vec{S}\,'(\ell)}{\left|\,\vec{S}\,'(\ell)\,\right|}\,.$$

Durch Projektion des Vektors $\vec{x}_R$ in den Vektor $\vec{x}_{R0}$ der x-y-Ebene des Inertialsystems und anschließende Rotation mit dem Winkel $\frac{\pi}{2}$ um die Hochachse des Inertialsystems wird der temporäre Vektor $\vec{y}_{R0}$ erzeugt.

$$\vec{x}_{R0} = \begin{pmatrix} x_{R,x} \\ x_{R,y} \\ 0 \end{pmatrix} \qquad \Rightarrow \qquad \vec{y}_{R0} = \begin{pmatrix} -x_{R,y} \\ x_{R,x} \\ 0 \end{pmatrix} \tag{2.9}$$

Die Rotation des Vektors $\vec{y}_{R0}$ mit dem Querneigungswinkel $\varphi_R(\ell)$ um die Längsachse $\vec{x}_R$ der Fahrbahn erzeugt die Querachse des Fahrbahnkoordinatensystems. Eine geeignete Rotationsmatrix ist durch Gleichung (A.12) in Abschnitt A.6 gegeben.

$$\vec{y}_R(\ell) = \frac{\mathbf{R}(\vec{x}_R, \varphi_R) \cdot \vec{y}_{R0}}{|\,\mathbf{R}(\vec{x}_R, \varphi_R) \cdot \vec{y}_{R0}\,|}$$

Die Hochachse des Fahrbahnkoordinatensystems kann einfach durch das Kreuzprodukt bestimmt werden.

$$\vec{z}_R(\ell) = \frac{\vec{x}_R \times \vec{y}_R}{|\,\vec{x}_R \times \vec{y}_R\,|}$$

Es bietet sich an die Basisvektoren des Fahrbahnkoordinatensystems nach dem Schema aus Gleichung (A.15) in einer Rotationsmatrix $\mathbf{R}_{IR}(\ell)$ zu formulieren, wobei zu beachten ist, daß das Fahrbahnkoordinatensystem in Abhängigkeit der Bogenlänge ℓ berechnet wird.

$$\mathbf{R}_{IR}(\ell) \quad = \quad \begin{bmatrix} x_{R,x} & y_{R,x} & z_{R,x} \\ x_{R,y} & y_{R,y} & z_{R,y} \\ x_{R,z} & y_{R,z} & z_{R,z} \end{bmatrix} \tag{2.10}$$

2.7. Berechnung der Fahrbahnkoordinaten

In diesem Abschnitt werden sog. Fahrbahn- oder auch Bandkoordinaten eingeführt. Mit diesen Koordinaten lässt sich die Position eines kartesischen Punktes anschaulich in Relation zur Fahrbahnoberfläche beschreiben. Einige Aufgabenstellungen der Gesamtsimulation, z.B. die Verkehrssimulation, lassen sich in Fahrbahnkoordinaten einfacher formulieren als in kartesischen Koordinaten.

2.7.1. Definition

Jedem dreidimensionalen Punkt $\vec{p}$ können die Fahrbahnkoordinaten D, O und L zugeordnet werden, s. Abbildung 2.6. Diese Bezeichnung ist identisch mit der in [Wang 2005].

Die auf der Fahrbahnmittellinie zurückgelegte Bogenlänge wird durch die D[4]-Koordinate beschrieben. Die Querablage zur Mittellinie wird dann mit O[5], die Höhe über der Fahrbahnoberfläche mit L[6] bezeichnet.

Damit ist der Koordinatenvektor des Punktes $\vec{p}$ in Fahrbahnkoordinaten vollständig definiert.

[4] engl.: *distance*
[5] engl.: *offset*
[6] engl.: *loft*

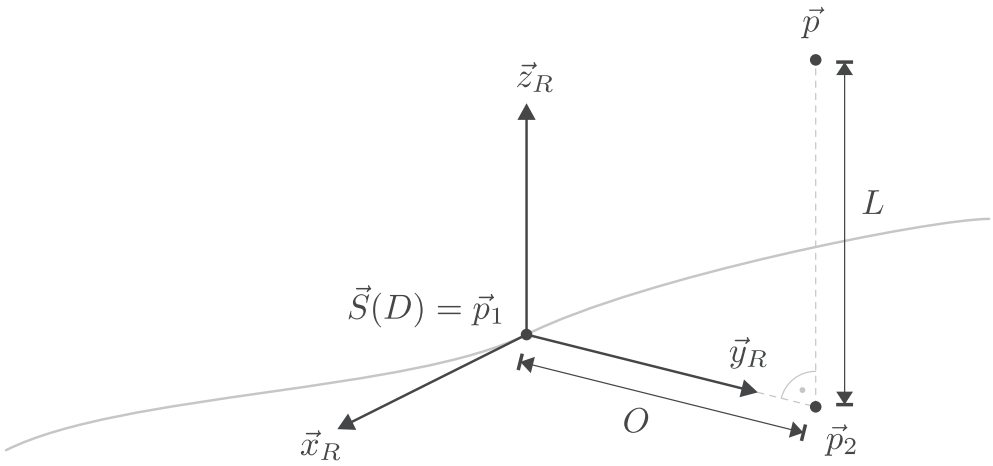

Abbildung 2.6.: Geometrische Bedeutung der Fahrbahnkoordinaten D, O und L.

$$_R\vec{p} = \begin{pmatrix} D \\ O \\ L \end{pmatrix}$$

2.7.2. Umrechnung in kartesische Koordinaten

Sind die Fahrbahnkoordinaten bekannt, kann durch den Zusammenhang

$$\vec{p} = \underbrace{\vec{S}(D)}_{\vec{p}_1} + O \cdot \vec{y}_R(D) + L \cdot \vec{z}_R(D) \tag{2.11}$$

der kartesische Punkt $\vec{p}$ bestimmt werden.

Dazu wird an der Stelle D des Rundkurses der Punkt $\vec{p}_1$ interpoliert und das lokale Fahrbahnkoordinatensystem $\mathbf{R}_{IR}(D)$ bestimmt. Der Punkt $\vec{p}_1$ liegt bereits im Inertialsystem vor, da die Stützstellen des Rundkurses in diesem angegeben wurden.

In diesem Punkt können die Querablage O und die Höhe L der Fahrbahnkoordinaten entlang der Quer- und der Hochachse des Fahrbahnkoordinatensystems $\mathbf{R}_{IR}$ durch Vektoraddition überlagert werden. Der resultierende Vektor $O \cdot \vec{y}_R + L \cdot \vec{z}_R$ bestimmt mit dem Punkt $\vec{p}_1$ den kartesischen Punkt $\vec{p}$.

2.7.3. Umrechnung von kartesischen Koordinaten

Die Bestimmung der Fahrbahnkoordinaten von kartesischen Koordinaten erfolgt nach [Wang 2005] in drei Schritten. Dabei wird auf dem Rundkurs der Punkt $\vec{p}_1$ gesucht, der einem beliebigen Punkt $\vec{p}$ am nächsten liegt. Zur Bestimmung dieses Punktes $\vec{p}_1$ ist die Kenntnis der zu $\vec{p}_1$ gehörigen Bogenlänge ℓ_1 erforderlich, da

$$\vec{p}_1 = \vec{S}(\ell_1)$$

gilt. Ist $\vec{p}_1$ bekannt, werden die fehlenden Fahrbahnkoordinaten O und L analytisch bestimmt. Mit $D = \ell_1$ gilt die Korrespondenz

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad \Leftrightarrow \qquad {}_R\vec{p} = \begin{pmatrix} D \\ O \\ L \end{pmatrix} .$$

Bestimmung des zugehörigen Abschnittes

Zur Bestimmung des Punktes $\vec{p}_1$ werden nur die Abschnitte des Rundkurses berücksichtigt, deren achsenparallele Begrenzungsvolumina den Punkt $\vec{p}$ enthalten. Das Begrenzungsvolumen kann durch einen Vektor $\vec{\varrho}_{min}$ der minimalen und einen weiteren Vektor $\vec{\varrho}_{max}$ der maximalen Koordinaten beschrieben werden. Der Punkt $\vec{p}$ muß dann für jede Koordinate die Bedingung

$$\vec{\varrho}_{min} \leq \vec{p} \leq \vec{\varrho}_{max}$$

erfüllen.

Für gerade Abschnitte können die Abmaße des Volumens durch Vergleich der Koordinaten von Anfangs- und Endpunkt bestimmt werden.

Jeder Spline-Funktion gekrümmter Abschnitte, s. Gleichung (2.5), kann ein Begrenzungsvolumen zugeordnet werden, in dem der Punkt $\vec{p}$ zu suchen ist. Bestimmt werden die Koordinaten eines Begrenzungsvolumens durch analytische Berechnung der Maxima und Minima jeder Spline-Funktion aller Dimensionen, s. Gleichung (2.3). Die Abmaße aller Begrenzungsvolumina

eines Splines können zu einem Begrenzungsvolumen zusammengefasst werden, um die Anzahl der Vergleichsoperationen zwischen den Koordinaten des Punktes $\vec{p}$ und der Begrenzungsvolumina durch einen initialen Test mit diesem Volumen zu minimieren.

In Abhängigkeit der definierten Fahrspuren pro Fahrtrichtung N_L und der Fahrspurbreite w_L werden alle Begrenzungsvolumina vergrößert. Der minimale Kurvenradius muß somit $N_L \cdot w_L$ betragen, um eine eindeutige Zuordnung des Punktes $\vec{p}$ zu den Begrenzungsvolumina gekrümmter Abschnitte zu ermöglichen. Der Punkt $\vec{p}$ wird durch den Test

$$\varrho_{min,x} - N_L \cdot w_L \leq p_x \leq \varrho_{max,x} + N_L \cdot w_L \tag{2.12}$$

einem Abschnitt des Rundkurses zugeordnet, wobei Gleichung (2.12) analog für die y- und die z-Koordinate gilt. Für gekrümmte Abschnitte muß ggf. geprüft werden, welcher Spline-Funktion $\vec{p}$ zugeordnet werden kann.

Die beiden nachfolgenden Schritte werden nur durchgeführt, wenn der Punkt $\vec{p}$ einem Abschnitt zugeordnet werden kann. Evtl. sind die nachfolgenden Schritte für mehrere Abschnitte durchzuführen, falls $\vec{p}$ nicht eindeutig einem Abschnitt zugeordnet werden kann. Es ist der Punkt $\vec{p}_1$ des Rundkurses zu verwenden, der den geringsten Abstand zum Punkt $\vec{p}$ aufweist.

Minimierung der Abstandsfunktion

Die Entfernung des Punktes $\vec{p}$ zum Rundkurs kann durch die quadratische Abstandsfunktion $\Delta(\ell) = \left| \vec{S}(\ell) - \vec{p} \right|^2$ beschrieben werden. Der gesuchte Punkt $\vec{p}_1$ auf dem Rundkurs ist durch den minimalen Wert dieser Funktion bestimmt.

Für gerade Straßenabschnitte ist der minimale Wert durch den Lotfußpunkt des Punktes $\vec{p}$ auf die Gerade bestimmt. Der Parameter ℓ_{min} des Lotfußpunktes errechnet sich mit

$$\ell_{min} = \ell_i + \frac{(\vec{p} - \vec{r}_i) \bullet \vec{g}}{|\,\vec{g}\,|^2}\,,$$

s. Gleichung (2.2). Die zugehörige quadratische Abstandsfunktion $\Delta(\ell_{min})$ lautet

$$\Delta(\ell_{min}) = |\,\vec{r}_i + (\ell_{min} - \ell_i) \cdot \vec{g} - \vec{p}\,|^2 \ .$$

Die Minimierung der Abstandsfunktion lässt sich für gekrümmte Abschnitte nicht analytisch lösen, da

$$\Delta(\ell) = (S_x(\ell) - p_x)^2 + (S_y(\ell) - p_y)^2 + (S_z(\ell) - p_z)^2 \tag{2.13}$$

gilt. Da es sich bei den Spline-Funktionen $S_i(\ell)$ um kubische Polynome handelt, ist $\Delta(\ell)$ eine Funktion vom Grad 6. Der Verlauf der Abstandsfunktion $\Delta(\ell)$ ähnelt jedoch einer quadratischen Funktion, s. Abbildung 2.7.

Die Minimierung der Abstandsfunktion erfolgt nach [Wang 2005] in zwei Schritten. Zunächst wird eine quadratische Funktion minimiert, die den Startwert für eine nachfolgende Minimierung mit dem Newton-Verfahren liefert. Dabei wird ausgenutzt, daß das Newton-Verfahren bei Vorgabe hinreichend genauer Startwerte schnell gegen das Minimum konvergiert, s. [Wang 2005].

Eine quadratische Funktion $y = f(x)$ wird allgemein durch

$$y \quad = \quad a \cdot x^2 + b \cdot x + c$$

beschrieben. Für drei Wertepaare (x_c, y_c) lässt sich das lineare Gleichungssystem

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \quad = \quad \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

aufstellen. Damit können die Koeffizienten $a = f(x_i, y_i)$, $b = f(x_i, y_i)$ und $c = f(x_i, y_i)$ bestimmt werden. Die Funktion $f(x)$ nimmt dann die Form

$$y \quad = \quad \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3$$

an. Das Minimum $\frac{d\,f(x)}{dx} = 0$ liegt bei

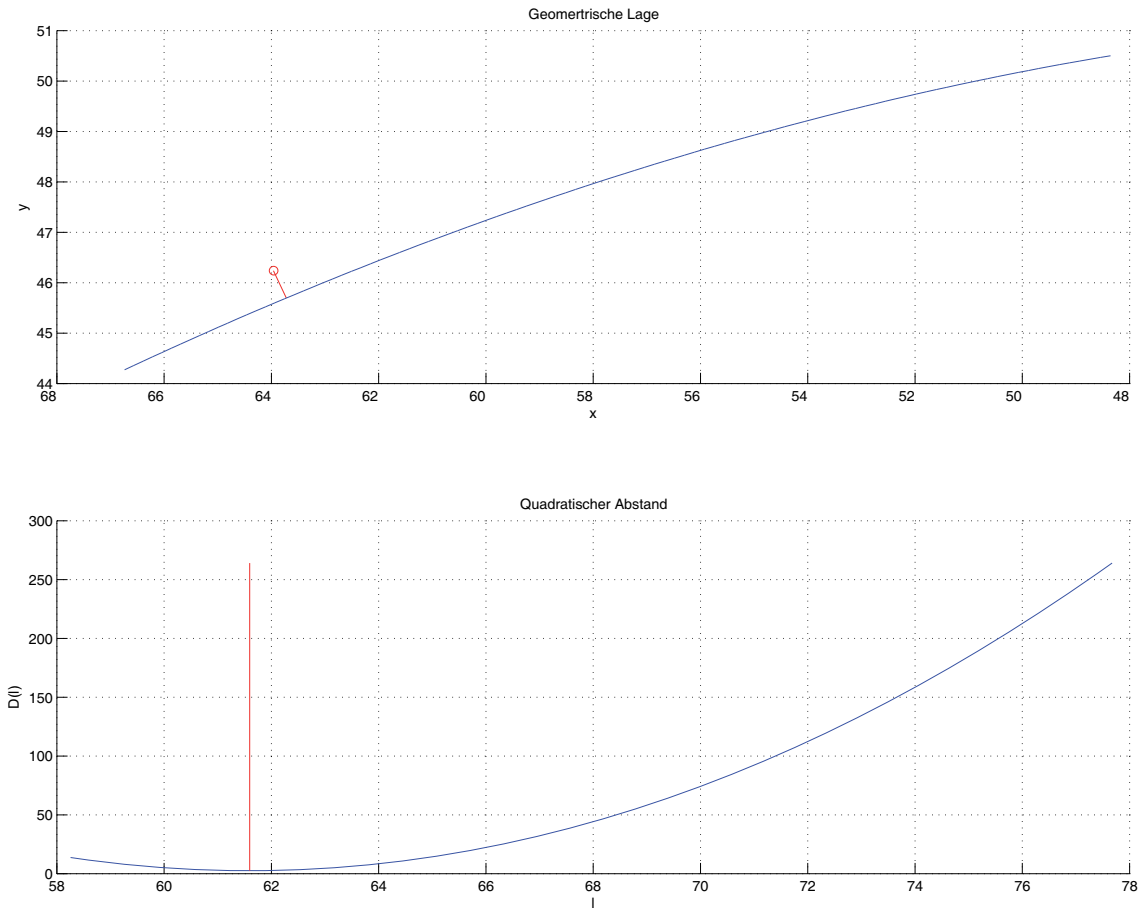

Abbildung 2.7.: Lage eines Punktes $\vec{p}$ zu einer Spline-Funktion (oben). Zugehörige Abstandsfunktion (unten).

$$x_{min} = \frac{1}{2} \cdot \frac{(x_2^2 - x_3^2)y_1 + (x_3^2 - x_1^2)y_2 + (x_1^2 - x_2^2)y_3}{(x_2 - x_3)y_1 + (x_3 - x_1)y_2 + (x_1 - x_2)y_3} \,. \tag{2.14}$$

Zur Minimierung der Abstandsfunktion werden Wertepaare mit $x_c = \ell_c$ und $y_c = \Delta(\ell_c)$ verwendet. Als Startwerte der quadratischen Minimierung werden die Parameterwerte ℓ_i, $\frac{\ell_i + \ell_{i+1}}{2}$ und ℓ_{i+1} der Spline-Funktion verwendet, in deren Begrenzungsvolumen sich der Punkt $\vec{p}$ befindet. Es werden drei Iterationen $c = 4, 5, 6$ mit Gleichung (2.14) durchgeführt, wobei nach jeder Iteration das Wertepaar mit der größten Abstandsfunktion verworfen wird. Der Parameterwert ℓ_c mit der geringsten Abstandsfunktion bildet den Startwert für das Newton-Verfahren.

Das Newton-Verfahren zur Bestimmung von Nullstellen der Funktion $f(x)$ lautet

$$x^{(c+1)} = x^{(c)} - \frac{f(x^{(c)})}{f'(x^{(c)})} \,.$$

Für die Minimierung der Abstandsfunktion nach Gleichung (2.13) gilt

$$\Delta'(\ell_{min}) = 0 \,.$$

Zur iterativen Berechnung mit dem Newton-Verfahren wird die 1. und die 2. Ableitung der Abstandsfunktion benötigt. Mit Gleichung (2.3) folgt

$$\begin{aligned}
\Delta(\ell) &= \left[\vec{S}(\ell) - \vec{p}\right] \bullet \left[\vec{S}(\ell) - \vec{p}\right] \\
\Delta'(\ell) &= 2 \cdot \vec{S}\,'(\ell) \bullet \left[\vec{S}(\ell) - \vec{p}\right] \\
\Delta''(\ell) &= 2 \cdot \left\{\vec{S}\,''(\ell) \bullet \left[\vec{S}(\ell) - \vec{p}\right] + \vec{S}\,'(\ell) \bullet \vec{S}\,'(\ell)\right\} \,.
\end{aligned}$$

Für die Iterationen c des Newton-Verfahrens folgt damit

$$\ell_{min}^{(c+1)} = \ell_{min}^{(c)} - \frac{\Delta'(\ell_{min}^{(c)})}{\Delta''(\ell_{min}^{(c)})} \,.$$

Durch die Verwendung des Startwerts aus der quadratischen Minimierung kann eine geringe Anzahl an Iterationen $c_{max} = 4$ gewählt werden, s. [Wang 2005].

Berechnung der Querablage und der Höhe

Durch die Anwendung von quadratischer Minimierung und Newton-Verfahren ist der zum Punkt $\vec{p}$ gehörende Parameterwert ℓ_1 bestimmt worden. Dieser Wert entspricht der auf der Mittellinie des Rundkurses zurückgelegten Entfernung D. Die fehlenden Koordinaten O und L werden nach Abbildung 2.6 analytisch bestimmt.

Der Punkt $\vec{p_1}$ wird durch den Rundkurs interpoliert.

$$\vec{p_1} = \vec{S}(\ell_1)$$

Mit dem Richtungsvektor $\vec{d}_{p_1 p} = \vec{p} - \vec{p_1}$ ist die Querablage zur Mittellinie bestimmt.

$$O = \vec{d}_{p_1 p} \bullet \vec{y}_R(\ell_1)$$

Die Basisvektoren des Fahrbahnkoordinatensystems $\mathbf{R}_{IR}(\ell_1)$ sind durch Gleichung (2.10) festgelegt.

Als Hilfsvariable wird der Punkt $\vec{p_2}$ eingeführt.

$$\vec{p_2} = \vec{p_1} + O \cdot \vec{y}_R(\ell_1)$$

Der Richtungsvektor $\vec{d}_{p_2 p} = \vec{p} - \vec{p_2}$ dient zur Bestimmung der Höhe L.

$$L = \vec{d}_{p_2 p} \bullet \vec{z}_R(\ell_1)$$

Die Fahrbahnkoordinaten ${}_R\vec{p}$ des kartesischen Punktes $\vec{p}$ sind damit vollständig bestimmt.

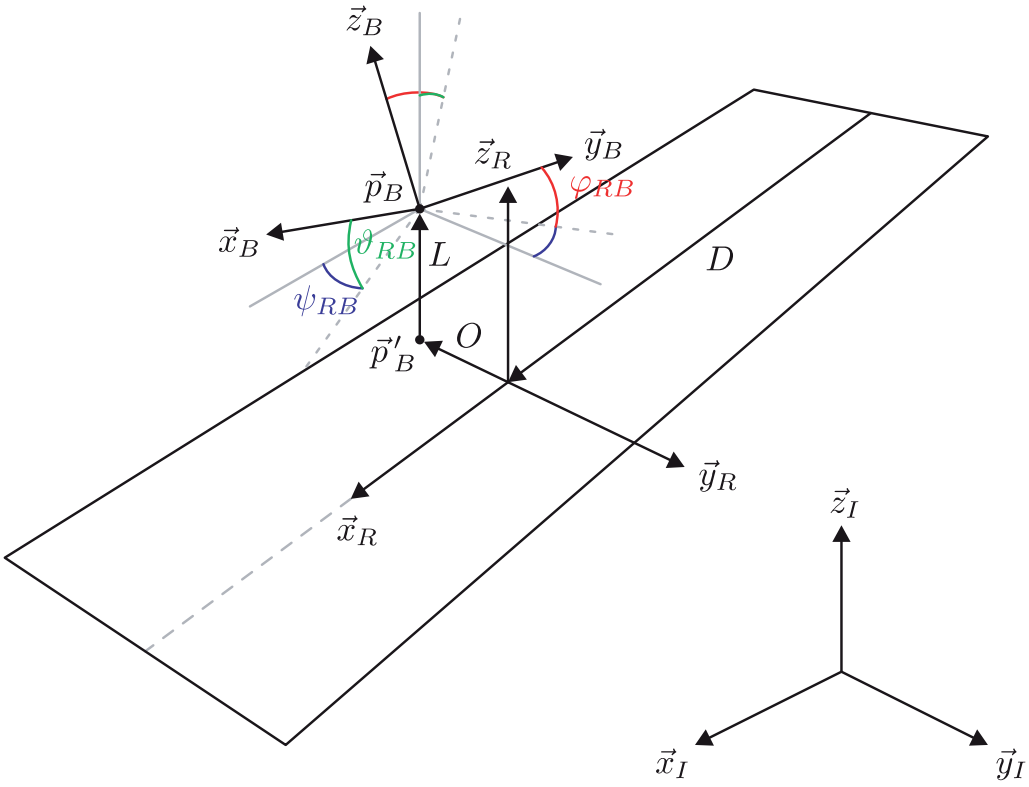

Abbildung 2.8.: Zur Berechnung der Rollbewegung eines Fahrzeuges gegenüber der Fahrbahn.

2.8. Berechnung der Fahrzeugkoordinaten

Zur Visualisierung ist evtl. die Kenntnis der Relativbewegung eines Fahrzeuges zur Fahrbahnoberfläche erforderlich, um die Rollbewegung des Fahrzeuges plausibel darstellen zu können.

Eine Fahrdynamiksimulation berechnet i.d.R. den Ortsvektor $\vec{p}_B$ und die Rotation $\mathbf{R}_{IB}$ des aufbaufesten Koordinatensystems B gegenüber dem Inertialsystem I. Das Fahrzeug bewegt sich dabei nicht zwingend entlang des Fahrbahnkoordinatensystems R, s. Abbildung 2.8.

Durch das Verfahren aus Abschnitt 2.7 kann der Ortsvektor $\vec{p}_B$ des Aufbaukoordinatensystems B der auf der Fahrbahn zurückgelegten Bogenlänge D zugeordnet werden. Die Projektion $\vec{p}'_B$ des Ortsvektors $\vec{p}_B$ des Aufbaus auf die Fahrbahnoberfläche ist damit festgelegt.

Zur Bestimmung der Rollbewegung des Aufbaus relativ zur Fahrbahnoberfläche ist die Matrix $\mathbf{R}_{RB}$ zu bestimmen, die den Kurswinkel ψ_{RB} den

Nickwinkel ϑ_{RB} und den Wankwinkel φ_{RB} nach [FAKRA 1994] festlegt.

Dazu wird die Rotationsmatrix $\mathbf{R}_{IB}$ durch die Rotation $\mathbf{R}_{RB}$ gefolgt von der Rotation $\mathbf{R}_{IR}$ ausgedrückt.

$$\mathbf{R}_{IB} = \mathbf{R}_{IR} \cdot \mathbf{R}_{RB} \tag{2.15}$$

Die Matrix $\mathbf{R}_{IR}$ beschreibt die Rotation der Fahrbahnkoordinaten relativ zum Inertialsystem, s. Gleichung (2.10).

Gleichung (2.15) kann nach $\mathbf{R}_{RB}$ aufgelöst werden. Die Winkel der relativen Rollbewegung von Fahrzeug zur Fahrbahn ergeben sich dann aus Gleichung (A.11) in Abschnitt A.5.2.

$$\begin{aligned}
\mathbf{R}_{RB} &= \mathbf{R}_{IR}^T \cdot \mathbf{R}_{IB} = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} \\
\varphi_{RB} &= \arctan\left(\frac{r_{21}}{r_{22}}\right) \\
\vartheta_{RB} &= -\arcsin(r_{20}) \\
\psi_{RB} &= \arctan\left(\frac{r_{10}}{r_{00}}\right)
\end{aligned} \tag{2.16}$$

3. Visualisierung

In diesem Kapitel wird die Visualisierung des dreidimensionalen Fahrzeugumfeldes beschrieben. Die Darstellung des Rundkurses erfordert die Kenntnis des in Abschnitt 2.6 eingeführten Fahrbahnkoordinatensystems. Abschnitt A.9 erläutert die zur Beschreibung der geometrischen Transformationen benötigte Starrkörpertransformation.

Die in diesem Kapitel verwendete Terminologie orientiert sich an den im Bereich Computer Graphik bzw. OpenGL üblichen Begriffen. Einige dieser Begriffe, die in den Natur- bzw. Ingenieurwissenschaften fest mit einem physikalischen Zusammenhang verbunden sind, werden hier in einem anderen Kontext, unabhängig von jeder physikalischen Bedeutung, gebraucht.

3.1. Einleitung

Die Darstellung dreidimensionaler Szenarien erfolgt in der Computergraphik durch sog. Rasteralgorithmen. Dabei werden die Oberflächen von Objekten einer virtuellen Welt durch eine beliebige Anzahl von Dreiecken approximiert. Zur Darstellung auf einem diskreten Anzeigegerät, z.B. einem Flüssigkristallbildschirm, werden diese Dreiecke zunächst auf eine zweidimensionale Fläche projiziert, um danach durch die sog. Rasterung in einzelne Bildelemente[1] zerlegt zu werden, s. [Foley u. a. 1992].

Üblicherweise wurde zur Bearbeitung verschiedener Forschungs- und Industrieprojekte jeweils eine proprietäre Visualisierungssoftware entwickelt, die die jeweilige Problemstellung lösen sollte. Die Unterstützung einer einheitlichen Programmierschnittstelle durch die Graphikkartenhersteller war somit nicht gegeben. Mit der Einführung OpenGLs[2] 1992 durch SGI steht eine offene Programmierschnittstelle zur Verfügung, die sowohl zur Darstellung zwei- als auch dreidimensionaler Graphiken eingesetzt werden kann. Viele Graphikkartenhersteller unterstützen durch geeignete Treiber eine direkte Verarbeitung der OpenGL-Befehle.

[1] engl.: *pixel*
[2] engl.: *Open Graphics Language*

Als Alternative zu OpenGL wurde von Microsoft die betriebssystemspezifische Programmierschnittstelle Direct3D entwickelt, die nur auf den firmeneigenen Windows-Betriebssystemen verfügbar ist. Im Gegensatz zu OpenGL sind neuere Versionen von Direct3D nicht abwärtskompatibel zu älteren Versionen. Dadurch sind ältere Programme u.U. nicht mehr auf neueren Windows-Betriebssystemen lauffähig.

OpenGL wird durch seinen offenen Standard häufig in Forschungsprojekten eingesetzt und ist Bestandteil vieler Grundlagenvorlesungen zum Thema Computergraphik. Im Gegensatz zu Direct3D ist die Programmierschnittstelle nicht an ein bestimmtes Betriebssystem gebunden. Bei einem Wechsel des Betriebssystems müssen somit nur die Programmteile neu implementiert werden, die das Anzeigegerät des jeweiligen Betriebssystems zur Verwendung durch OpenGL initialisieren.

Zur Realisierung der Visualisierung wurde OpenGL als Programmierschnittstelle gewählt, da diese bereits bekannt war und ein Wechsel des zugrunde liegenden Betriebssystems zu einem abschätzbaren Arbeitsaufwand führt.

3.2. OpenGL-Grundlagen

In diesem Abschnitt sollen die wichtigsten Funktionen OpenGLs erläutert werden. Einen vollständigen Überblick über die Funktionalität geben [Shreiner u. a. 2006].

OpenGL ist als Zustandsautomat realisiert. So kann Geometrie, die zur Darstellung dieselben Parameter benötigt, mit einem Minimum an Befehlen dargestellt werden. I.d.R. ist dann nur der Transport der Geometrie zur Graphikkarte erforderlich. Die Darstellung von Objekten mit unterschiedlichen Parametern führt jedoch meist zu komplexen Zustandswechseln, die ebenfalls in der Graphikkarte vorgenommen werden müssen und so die Datenverarbeitung verzögern. Ein Entwurfsziel bei der Entwicklung mit OpenGL ist daher die Minimierung der Zustandswechsel.

Listing 3.1 zeigt die OpenGL-Befehle zur Darstellung eines Dreiecks.

Listing 3.1: Programm zur Darstellung eines Dreiecks mit OpenGL.

```
1 glBegin(GL_TRIANGLES);
  glColor3f(0, 0, 1);
  glNormal3f(0, 0, 1);
  glVertex3f( 1, 0, 0);
```

```
5  glVertex3f( 0, 1, 0);
   glVertex3f(-1, 0, 0);
   glEnd();
```

Die Definition einer graphischen Grundform[3] muß dabei in einem speziellen Definitionsmodus erfolgen, s. Zeilen 1 und 7. In Listing 3.1 wird die Graphikkarte aufgefordert, Dreiecke zu zeichnen. In diesem Modus werden jeweils drei aufeinanderfolgende Punkte, die durch den Befehl `glVertex()` definiert werden, zu einem Dreieck zusammengefasst. Jeder Aufruf dieses Befehls sendet den entsprechenden Datenvektor an die Graphikkarte und startet die Verarbeitung des entsprechenden Punktes durch OpenGL. In Listing 3.1 wurde der Zustand zur Verarbeitung der Punkte durch Setzen eines Normalenvektors und eines Farbwertes, s. `glNormal()` und `glColor()`, beeinflusst. Soll jeder Punkt eines Dreiecks mit unterschiedlichen Normalen- bzw. Farbwerten ausgestattet werden, so sind die entsprechenden Datenvektoren vor jeder Definition eines Punktes erneut zu setzen. Die an die Graphikkarte zu sendende Datenmenge erhöht sich dadurch beträchtlich.

Das Senden eines Punktes an die Graphikkarte startet die Datenverarbeitung durch die Graphikpipeline OpenGLs in Abhängigkeit des momentanen Zustandes. Abbildung 3.1 zeigt die wichtigsten Stationen dieser Graphikpipeline, wie sie zum Verständnis der hier beschriebenen Visualisierung erforderlich sind. Moderne Graphikkarten realisieren eine solche Pipeline bereits durch ihre Architektur. Dadurch können viele Graphikoperationen unabhängig vom Prozessor des Visualisierungs-PCs verarbeitet werden.

Um das Laden von Geometrieinformationen in die Graphikkarte zu beschleunigen, unterstützt OpenGL eine direkte Ablage dieser Daten im Speicher der Graphikkarte. Zum Zeichnen eines Objektes sind dann nicht mehr die vollständigen Datensätze aus Farbwert, Normalenvektor und Koordinaten an die Graphikkarte zu senden, sondern nur noch Indizes, die die Datenfelder im Graphikspeicher referenzieren.

Es folgt eine Beschreibung der Funktionalität der wichtigsten Pipelinestufen, wie sie für die Visualisierung verwendet werden. Es wird dabei nicht auf mögliche Erweiterungen durch neuere OpenGL-Versionen eingegangen.

Vektoren Jeder Orts- und Richtungsvektor $\vec{v}_i$, der an die Graphikkarte gesendet wurde erfährt in dieser Pipelinestufe die durch OpenGL konfigurierte geometrische Transformation.

[3] engl.: *primitive*; OpenGL unterstützt Punkte, Linien, Drei- und Vierecke

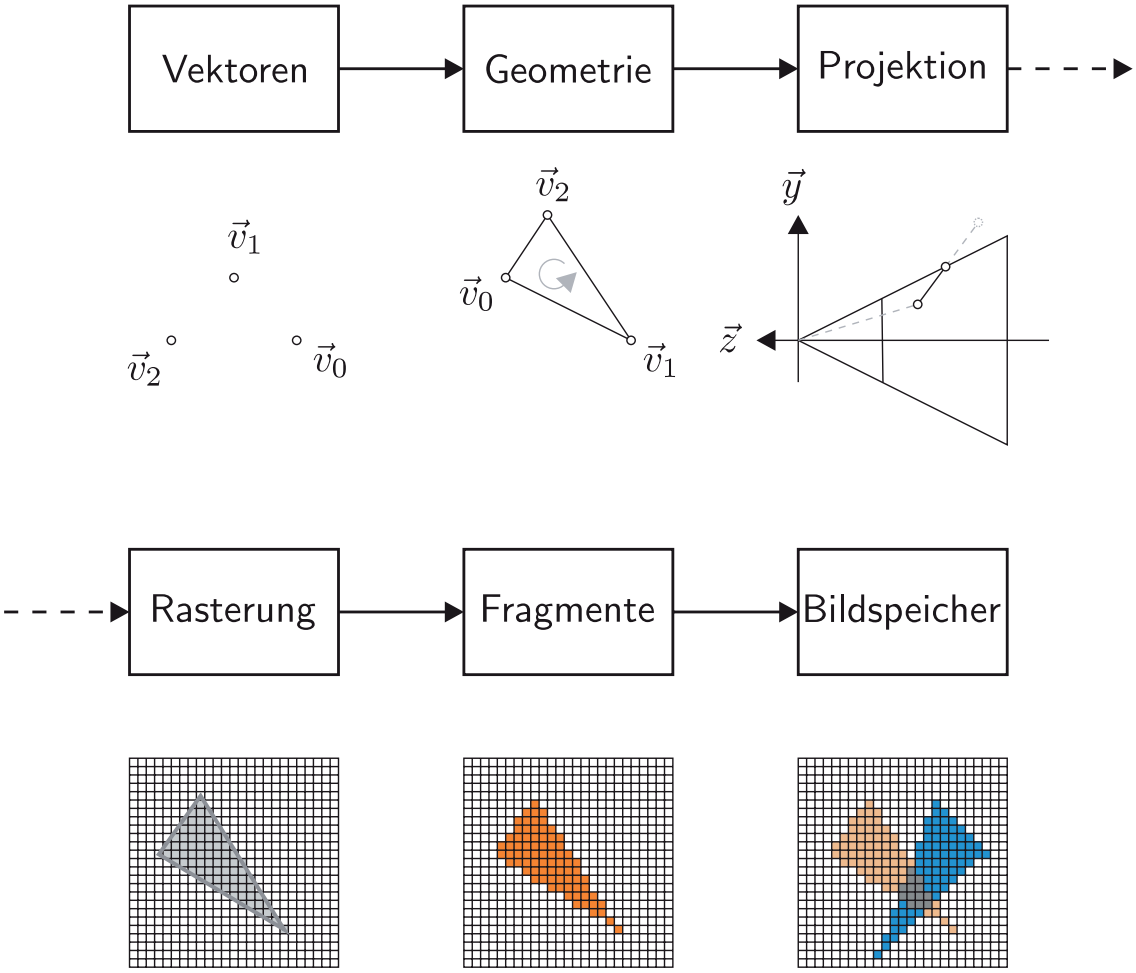

Abbildung 3.1.: Datenverarbeitung durch die Graphikpipeline OpenGLs.

$$\vec{v}'_i = \xi_{MVP} \cdot \vec{v}_i \tag{3.1}$$

Die Betrachtermatrix[4] ξ_{MVP} ist dabei in Abhängigkeit der gewünschten Position, des gewünschten Koordinatensystems und der Größe des Anzeigegerätes zu konfigurieren. Diese Konfiguration ist abhängig vom Verwendungszweck der zu entwickelnden Graphiksoftware. Abschnitt 3.3 erläutert die zur Visualisierung gewählte Konfiguration der Betrachtermatrix.

Als Datenquelle stehen dieser Pipeline-Stufe die durch die Befehle in Listing 3.1 angegebenen Datenvektoren zur Verfügung. OpenGL interpretiert dabei alle Datenvektoren als homogene Vektoren der Dimension 4×1. Ortsvektoren, die die Punkte[5] der graphischen Grundformen festlegen, werden mit dem Befehl `glVertex()` erzeugt, wobei eine nicht explizit angegebene homogene Komponente zu $w = 1$ gesetzt wird. Normalenvektoren hingegen repräsentieren immer Richtungsvektoren mit der homogenen Komponente $w = 0$.

Durch Transformation mit der Betrachtermatrix ξ_{MVP} werden die Vektoren $\vec{v}_i$ auf die Vektoren $\vec{v}'_i$ des Einheitsvolumens $(\pm 1, \pm 1, \pm 1)^T$ abgebildet. Die homogene Komponente, s. Abschnitt A.9, der Vektoren $\vec{v}'_i$ enthält dann den Korrekturfaktor zur perspektivischen Projektion.

Weiterhin wird für jeden Punkt der Farbwert und der Normalenvektor zur Berechnung des Lichtmodells initialisiert, s. Abschnitt 3.4.

Der OpenGL-Standard gestattet es durch wenige Befehle die Matrix ξ_{MVP} an die jeweilige Problemstellung anzupassen. Moderne Graphikkarten gestatten darüber hinaus die Programmierung dieser Pipelinestufe durch eine Hochsprache. Die Graphikkarte ist dadurch in der Lage z.B. die Wellenbewegung einer Wasseroberfläche durch Modulation der Geometriedaten mit einer Sinusfunktion zu berechnen. Durch massive Parallelverarbeitung des Graphikprozessors werden durch ein solches Vertex-Programm mehrere Vertices gleichzeitig verarbeitet.

[4] engl.: *model-view-projection matrix*
[5] engl.: *vertex*, pl. *vertices*

Geometrie Diese Pipelinestufe verbindet mehrere Punkte zu der durch den Programmierer festgelegten Form. Dabei ist für die darauffolgende Pipelinestufe die Reihenfolge der Eckpunkte von elementarer Bedeutung.

Es ist geplant in zukünftigen OpenGL-Versionen diese Pipelinestufe durch ein sog. Geometrie-Programm in ihrer Verwendung flexibler zu gestalten. Dadurch ist es z.B. möglich eine Form in eine andere umzuwandeln oder eine Form durch Entfernen aus der Pipeline zu ignorieren.

Projektion Durch die bekannte Reihenfolge der Eckpunkte einer Form können vom Betrachter abgewandte Flächen aus der Pipeline entfernt werden.

Formen, die im Blickfeld nur teilweise sichtbar sind, müssen auf das Volumen des Blickfeldes begrenzt werden. Dazu sind evtl. Eckpunkte neu zu berechnen oder zu entfernen. Dieses Vorgehen wird durch die Normierung der Vektoren auf das Einheitsvolumen $(\pm 1, \pm 1, \pm 1)^T$ erleichtert.

Die Eckpunkte werden danach durch die sog. perspektivische Division mit der w-Koordinate auf die zweidimensionale Fläche des Anzeigegerätes projiziert.

Letztendlich erfolgt die Umrechnung in sog. Gerätekoordinaten, die den Koordinaten der zweidimensionalen Projektion auf dem Anzeigegerät entsprechen.

Rasterung Durch die bekannten Gerätekoordinaten kann eine Form nun zur Darstellung auf dem Anzeigegerät ausgetastet werden. Dazu werden die Schnittpunkte zweier Kanten der Form mit einer diskreten Zeile des Anzeigegerätes berechnet. Es dürfen im folgenden nur die Bildelemente des Anzeigegerätes bearbeitet werden, die sich innerhalb dieser Schnittpunkte befinden. Algorithmen zur Austastung verschiedener Formen werden von [Foley u. a. 1992] beschrieben.

Parallel zur Austastung findet für jedes Bildelement die Interpolation der Parameter des Lichtmodells und der Entfernung zum Betrachter, die auch als Tiefe bezeichnet wird, statt. Ein Lichtmodell benötigt zur Schattierung gekrümmter Flächen einen Farbwert und einen Normalenvektor, um den resultierenden Farbwert eines Bildelementes plausibel berechnen zu können, s. Abschnitt 3.4. Die Tiefe wird

zur Bestimmung der Sichtbarkeit eines ausgetasteten Bildelementes benötigt.

Das durch die Austastung erzeugte Bildelement wird auch als Fragment bezeichnet. Es besteht u.a. aus den Gerätekoordinaten, der Tiefe, einem Farbwert und einem Normalenvektor.

Moderne Graphikkarten erzeugen mehrere Fragmente gleichzeitig. Die nachfolgenden Pipelinestufen sind dann mehrfach vorhanden und werden parallel verwendet.

Fragmente Für jedes Fragment kann jetzt der resultierende Farbton im Bildspeicher des Anzeigegerätes berechnet werden. Dies geschieht unter Berücksichtigung des verwendeten Lichtmodells, s. Abschnitt 3.4, und eines Texturelementes. Durch die bekannte Tiefe kann ebenfalls ein Nebeleffekt erzeugt werden, indem weiter vom Betrachter entfernte Bildpunkte stärker mit einem Grauton vermischt werden.

Die Berechnung des resultierenden Farbwertes lässt sich auf modernen Graphikkarten durch ein Hochsprachenprogramm beliebig anpassen. So ist es z.B. möglich die Helligkeit einer Oberfläche mit einer zeitlichen Sinusfunktion zu modulieren, um eine defekte Lichtquelle zu simulieren.

Bildspeicher Der Bildspeicher besteht unter OpenGL aus mehreren Feldern, deren Dimensionen der der Anzeigefläche entsprechen. Diese Felder speichern u.a. die Farbwerte, die Tiefe und eine Maske, die verwendet werden kann, um Bereiche des Anzeigegerätes für eine Aktualisierung des Bildspeichers zu sperren.

Ob ein Fragment tatsächlich in den Bildspeicher geschrieben wird ist abhängig von seiner Tiefe. I.d.R. werden nur die Fragmente in den Bildspeicher übernommen, deren Tiefe kleiner ist als die des schon im Bildspeicher vorhandenen Fragmentes.

Zusätzlich lassen sich nur in dieser Stufe Transparenzeffekte berechnen, da diese die Kenntnis schon vorhandener Fragmente erfordern.

Die Möglichkeiten zur Visualisierung einer dreidimensionalen Szenerie sind somit an die Funktionalität OpenGLs gekoppelt. Für bestimmte Anwendungsgebiete, z.B. die Erzeugung von Schatten, ist somit eine Kenntnis der Graphikpipeline unabdingbar.

In den nachfolgenden Abschnitten soll auf die Verwendung von OpenGL zur Visualisierung des dreidimensionalen Fahrzeugumfeldes eingegangen werden. Dazu wird zunächst die Konfiguration der Betrachtermatrix und des Lichtmodells beschrieben. Danach werden die Datenstrukturen der dreidimensionalen Objekte der virtuellen Welt erläutert.

3.3. Konfiguration der Betrachtermatrix

Da der Begriff Kamera oder Kamerakoordinatensystem i.d.R. von Computergraphiksystemen verwendet wird, um das Koordinatensystem des Sichtfeldes zu beschrieben, wird hier die Bezeichnung Betrachter eingeführt, um Verwechselungen zu vermeiden.

Durch die Betrachtermatrix ξ_{MVP}, s. Gleichung (3.1), werden Vektoren in Abhängigkeit von Position und Orientierung eines Betrachters der virtuellen Welt auf das Einheitsvolumen $(\pm 1, \pm 1, \pm 1)^T$ abgebildet. Diese Abbildung kann als Verkettung mehrerer Starrkörpertransformationen aufgefasst werden, die die Vektoren durch aufeinanderfolgende Rotationen und Translationen transformieren.

Eine einzelne Starrkörpertransformation ist als 4×4-Matrix homogener Koordinaten aufzufassen, s. Abschnitt A.9, die dabei einen Vektor durch eine Rotation gefolgt von einer Translation transformiert.

Die Betrachtermatrix ξ_{MVP} wird als Verkettung der Matrizen in Gleichung (3.2) beschrieben.

$$\xi_{MVP} = \xi_P \cdot \xi_{CO} \cdot \xi_{OW} \tag{3.2}$$

Vektoren durchlaufen die Transformationen dabei von rechts nach links.

Ziel dieser Struktur der Betrachtermatrix ist die Minimierung von Wechseln der Transformationsmatrix ξ_{MVP}. Die Geometriedaten statischer Objekte, z.B. des Rundkurses, können direkt durch diese Betrachtermatrix transformiert werden, da sie bereits in Weltkoordinaten vorliegen. Für dynamische Objekte, z.B. den Verkehrsteilnehmern, ist diese Transformationsmatrix durch die Starrkörpertransformation vom Objekt- ins Weltkoordinatensystem, wie sie durch eine Simulation berechnet wird, zu ergänzen. Das Inertialsystem I der Simulation ist dabei identisch mit dem Weltkoordinatensystem W.

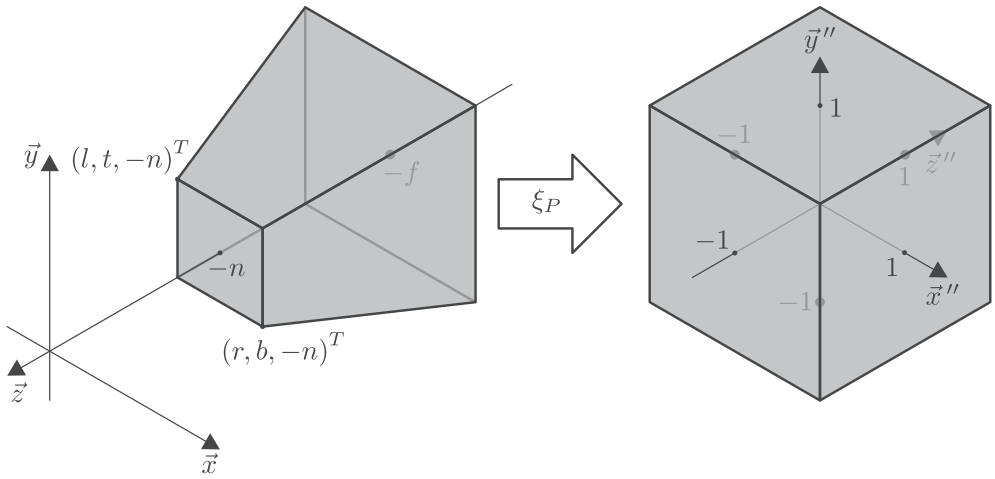

Abbildung 3.2.: Die Projektionsmatrix ξ_P bildet den sichtbaren Bereich OpenGLs, der die Form eines Kegelstumpfes aufweist, auf einen Einheitswürfel ab.

3.3.1. Projektion

Mit der Projektionsmatrix ξ_P, die die letzte Transformation der Betrachtermatrix ξ_{MVP} darstellt, werden in OpenGL die Koordinaten des sichtbaren Bereiches, die sog. Augenkoordinaten[6], auf eine Projektionsfläche bei $z = -n$ projiziert und in normierte Gerätekoordinaten[7] umgerechnet. Der sichtbare Bereich, das statische Kamerasystem C, hat dabei die Form eines Kegelstumpfes, der durch die Transformation ξ_P in einen Einheitswürfel überführt wird, s. Abbildung 3.2.

Das Sichtfeld des Kegelstumpfes ist dabei entlang der negativen z-Achse orientiert. Die Projektionsfläche befindet sich an der Stelle $z = -n$, die Begrenzung des Sichtfeldes an der Stelle $z = -f$, wobei bei der OpenGL-üblichen Angabe der Abmaße auf die Bedingung $0 < n < f$ geachtet werden muß. In horizontaler Richtung wird die Projektionsfläche im Bereich $x = [l, r]$ begrenzt, in vertikaler Richtung im Bereich $y = [b, t]$.

Die Anwendung der Projektionsmatrix ξ_P durch OpenGL erfolgt in zwei Schritten. Zuerst werden die dreidimensionalen Augenkoordinaten $(x, y, z, 1)^T$ in sog. Schnittkoordinaten[8] $(x', y', z', w')^T$ überführt, s. Gleichung (3.3).

[6] engl.: *eye coordinates*
[7] engl.: *normalized device coordinates (NDC)*

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \xi_P \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.3}$$

Durch Anwendung der Projektionsmatrix ξ_P werden dreidimensionale Koordinaten auf homogene Koordinaten abgebildet, d.h. $w' \neq 1$.

Nach dieser Transformation findet die perspektivische Korrektur oder auch perspektivische Division statt. Die Schnittkoordinaten werden dabei durch ihre homogene Komponente w' dividiert, um die normierten Gerätekoordinaten $(x'', y'', z'')^T$ zu erzeugen, s. Gleichung (3.4).

$$\begin{pmatrix} x'' \\ y'' \\ z'' \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} /w' \tag{3.4}$$

Die Herleitung der Projektionsmatrix ξ_P orientiert sich an [Ahn 2008]. Ausgangspunkt der Herleitung bildet die Projektion des Punktes $\vec{p} = (x_p, y_p, z_p, 1)^T$, s. Abbildung 3.3.

Für die projizierte y-Koordinate $\overline{y}_p$ des Punktes $\vec{p}$ folgt aus Abbildung 3.3 unmittelbar

$$\overline{y}_p = n \cdot \frac{y_p}{-z_p} \,,$$

oder allgemein

$$\overline{y} = n \cdot \frac{y}{-z} \,. \tag{3.5}$$

Durch Interpolation an einer Geraden wird der Bereich $[b, t]$ der projizierten Koordinaten $\overline{y}$ auf den Bereich $[-1, 1]$ der normierten Gerätekoordinaten y'' umgerechnet, s. Abbildung 3.3. Mit den Bedingungen

[8] engl.: *clip coordinates*

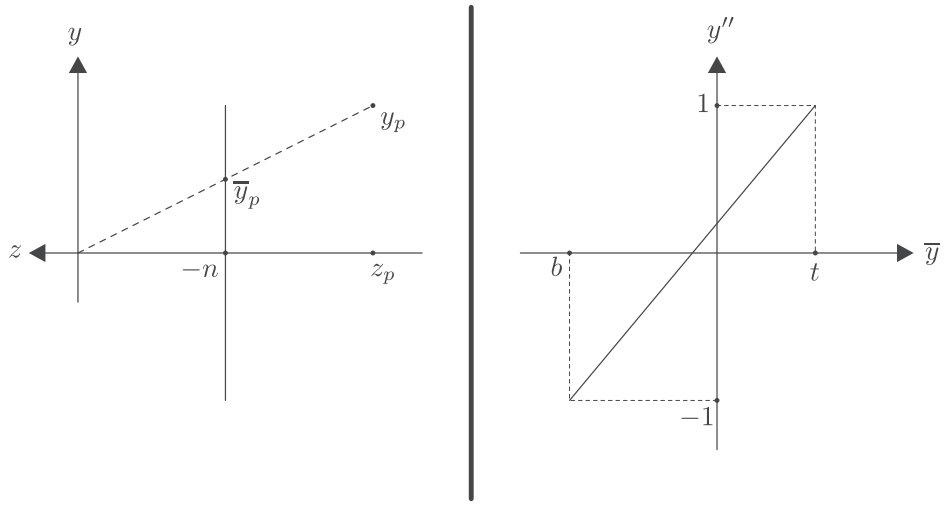

Abbildung 3.3.: Projektion eines Punktes $\vec{p}$ (links) und Normierung der projizierten Koordinaten auf normierte Gerätekoordinaten (rechts).

$$\begin{aligned} y'' &= \frac{2}{t-b} \cdot \overline{y} + \beta \quad \text{und} \\ 1 &= \frac{2}{t-b} \cdot t + \beta \end{aligned}$$

folgt für die normierten Gerätekoordinaten

$$y'' = \frac{2}{t-b} \cdot \overline{y} - \frac{t+b}{t-b} \,.$$

Einsetzen von Gleichung (3.5) liefert den Zusammenhang

$$y'' = \frac{2 \cdot n}{t-b} \cdot \frac{y}{-z} - \frac{t+b}{t-b}$$

zwischen Augen- und normierten Gerätekoordinaten. Durch Umstellen dieser Gleichung wird der Zusammenhang aus Gleichung (3.4) deutlich:

$$y'' = \underbrace{\left(\frac{2 \cdot n}{t-b} \cdot y + \frac{t+b}{t-b} \cdot z\right)}_{=y'} / \underbrace{(-z)}_{=w'}$$

Damit sind die 2. und die 4. Zeile der Matrix ξ_P bestimmt, s. Gleichung (3.3) und Gleichung (3.4). Ein analoges Vorgehen liefert einen ähnlichen Zusammenhang für die 1. Zeile.

Unter der Annahme, daß die z''-Koordinaten nicht von x-y-Koordinaten abhängig sind, kann aus den Gleichungen (3.3) und (3.4) die Bedingung

$$z'' = \frac{z'}{w'} = \frac{A \cdot z + B}{-z}$$

aufgestellt werden. Mit der Festlegung, daß $z = -n$ auf $z'' = -1$ und $z = -f$ auf $z'' = 1$ abgebildet werden, folgen

$$\begin{aligned} A &= -\frac{f+n}{f-n} \quad \text{und} \\ B &= -\frac{2 \cdot f \cdot n}{f-n}\,. \end{aligned}$$

Die Projektionsmatrix ξ_P ist damit vollständig bestimmt, s. [Shreiner u. a. 2006], und lautet somit

$$\xi_P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}. \tag{3.6}$$

Für eine Anwendung müssen die sechs Parameter dieser Matrix problemangepasst bestimmt werden.

Üblicherweise wird das Sichtfeld durch vier Kenngrößen konfiguriert.

1. Dem horizontalen Öffnungswinkel φ_H des Kegelstumpfes.

2. Der Tiefe d des Kegelstumpfes.

3. Dem Seitenverhältnis $a = \frac{b'}{h'}$ zwischen der Breite und der Höhe der Anzeigefläche; gemessen in Bildelementen.

4. Einem Faktor c zur Umrechnung von normierten Koordinaten des Einheitsvolumens auf Längeneinheiten der virtuellen Welt.

Mit diesen Größen lassen sich die Variablen der Matrix ξ_P mit den Hilfsvariablen w und h, die der Breite und Höhe der Anzeigefläche in Längeneinheiten der virtuellen Welt entsprechen, berechnen.

$$
\begin{aligned}
h &= c \\
w &= a \cdot h \\
l &= -\frac{w}{2} \\
r &= \frac{w}{2} \\
b &= -\frac{h}{2} \\
t &= \frac{h}{2} \\
n &= \frac{w}{2 \cdot \tan\left(\frac{\varphi_H}{2}\right)} \\
f &= n + d
\end{aligned}
$$

3.3.2. Wechsel des Koordinatensystems

Im vorherigen Abschnitt wurde das Sichtfeld OpenGLs durch einen Kegelstumpf konfiguriert. Die Position und Orientierung dieses Kegelstumpfes ist jedoch statisch an das Koordinatensystem OpenGLs gebunden. Zur Visualisierung des Fahrzeugumfeldes ist es vorteilhaft die Koordinaten der virtuellen Welt so zu formulieren, daß sie einem Anwender aus der Automobilindustrie vertraut sind, s. Abschnitt A.1.

Die Starrkörpertransformation ξ_{CO} in Gleichung (3.2), die das Betrachtersystem O in das statische Kamerasystem C abbildet, muß dazu entsprechend konfiguriert werden. Ein Betrachter soll sich im Koordinatenursprung befinden und entlang der positiven x-Achse ausgerichtet sein, was eine Anwendung der Koordinaten aus [FAKRA 1994] erleichtert. Die Transformation der Koordinaten nach [FAKRA 1994] in OpenGL-Koordinaten erfolgt dann in drei Schritten, s. Abbildung 3.4.

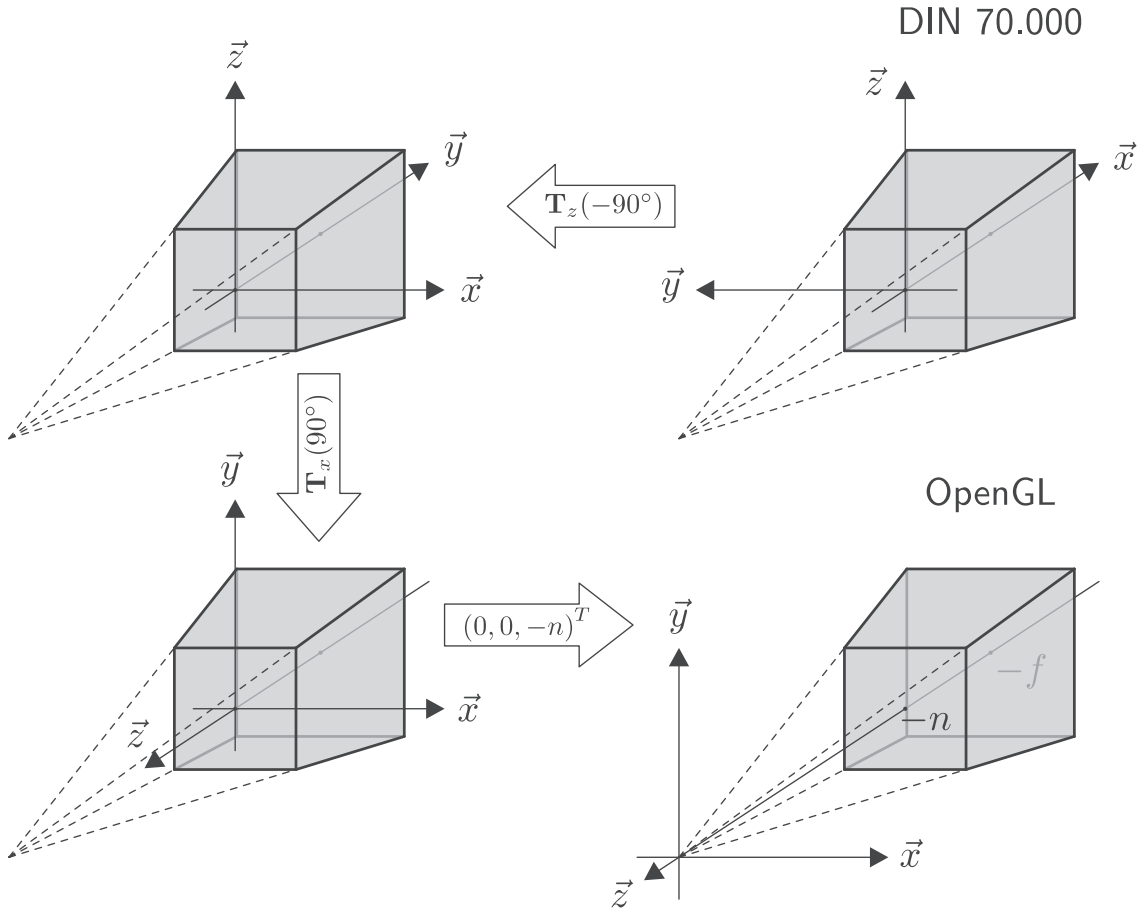

Abbildung 3.4.: Transformation der Koordinaten von DIN 70.000 nach OpenGL.

Zuerst erfolgt eine Basisrotation mit dem Winkel $-90°$ um die z-Achse. Danach wird eine Basisrotation mit dem Winkel $90°$ um die x-Achse des *neuen* Koordinatensystems durchgeführt. Abschließend wird die Projektionsfläche des Kegelstumpfes um $-n$ entlang der z-Achse aus dem Koordinatenursprung verschoben, womit der Kegelstumpf nun die für OpenGL gebräuchliche Lage aufweist. Die resultierende Starrkörpertransformation lautet damit

$$\xi_{CO} = \xi(\,\mathbf{T}_x(90°) \cdot \mathbf{T}_z(-90°), (0, 0, -n)^T\,)\,.$$

3.3.3. Bewegung des Betrachters

Bisher wurde in den beiden vorherigen Abschnitten das Koordinatensystem des Betrachters auf das Kamerakoordinatensystem OpenGLs abgebildet und die Matrix zur Projektion der dreidimensionalen Szenerie auf die Anzeigefläche konfiguriert.

Der Betrachter ist identisch mit einem weiteren Koordinatensystem, daß durch Rotation und Translation in der virtuellen Welt platziert wird. Gesucht wird eine Transformation, die das Weltkoordinatensystem W in das Koordinatensystem O des Betrachters überführt. Einem Anwender ist es dann durch seine Eingabe möglich dieses Koordinatensystem zu bewegen. Durch nachfolgende Anwendung der Transformationsvorschrift scheint sich der Anwender in der virtuellen Welt zu bewegen.

Zur Rotation des Betrachtersystems ist die sog. *Tracking*-Sequenz, s. [Kuipers 2002], geeignet. Sie kombiniert die Rotation mit dem Winkel α um die Hochachse mit einer nachfolgenden Rotation mit dem Winkel β um die Querachse des temporären Systems, s. Abbildung 3.5.

Die resultierende Matrix der Basisrotation lautet

$$\mathbf{T}_{OW} = \mathbf{T}_y(\beta) \cdot \mathbf{T}_z(\alpha)\,. \tag{3.7}$$

Mit dem Zusammenhang aus Gleichung (A.16) können die Basisvektoren des rotierten Koordinatensystems gewonnen werden, die zur Platzierung eines Betrachters verwendet werden können.

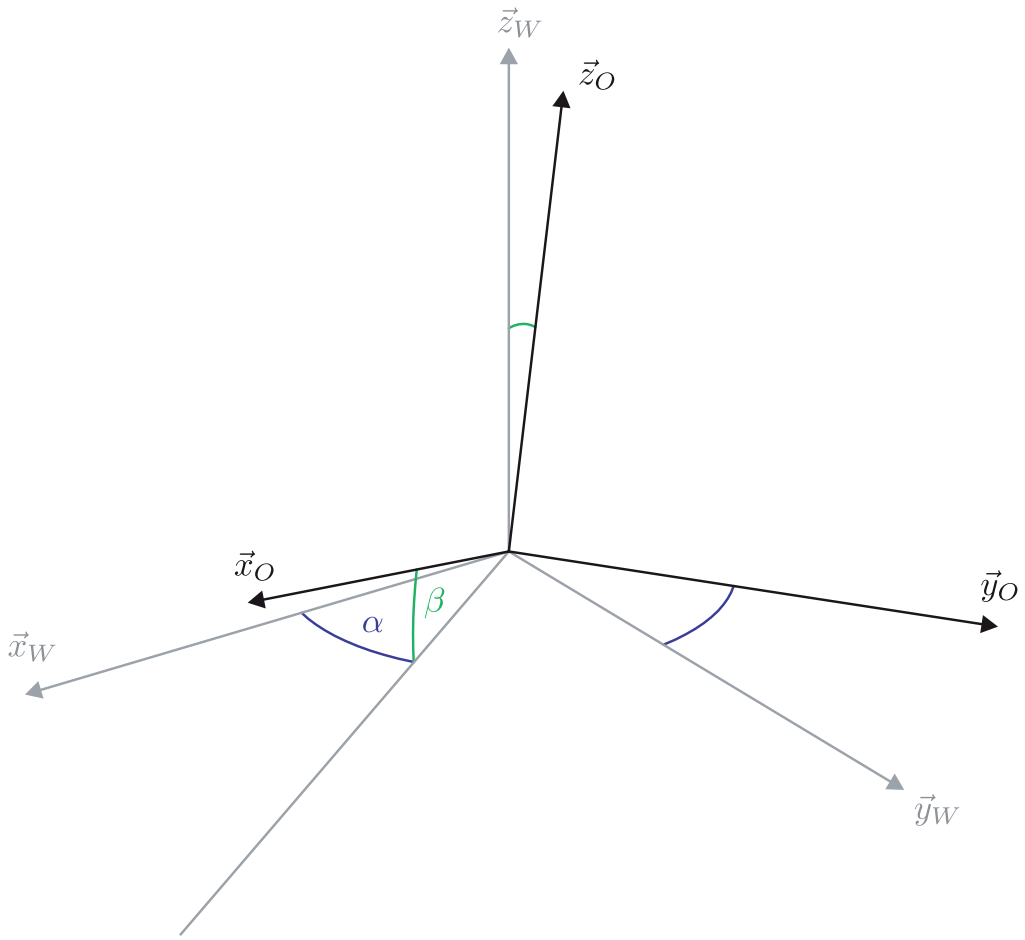

Abbildung 3.5.: Ausrichtung eines Betrachters durch die *Tracking*-Sequenz.

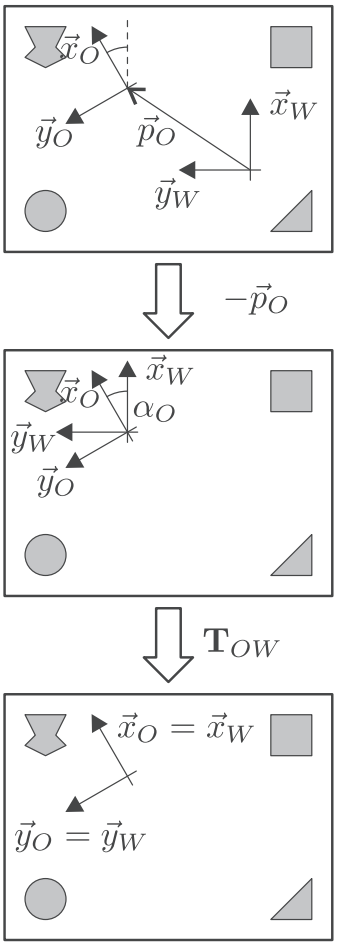

Abbildung 3.6.: Bewegung eines Betrachters durch eine virtuelle Welt.

Bewegung in Weltkoordinaten

Die Bewegung innerhalb einer virtuellen Welt lässt sich durch eine Transformation nach Abbildung 3.6 realisieren. Zuerst werden die Koordinatenursprünge des Betrachter- und des Weltkoordinatensystems durch eine Translation ineinander überführt. Danach wird das Betrachtersystem entlang des Weltkoordinatensystems ausgerichtet.

Die gesuchte Starrkörpertransformation ξ_{OW} lautet damit

$$\xi_{OW} = \xi(\,\mathbf{T}_{OW}, \vec{0}\,)\cdot\xi(\,\mathbf{I}, -\vec{p}_O\,) = \xi^{-1}(\,\mathbf{T}_{OW}^T, \vec{p}_O\,) = \xi(\,\mathbf{T}_{OW}, -\mathbf{T}_{OW}\cdot\vec{p}_O\,)\,.$$

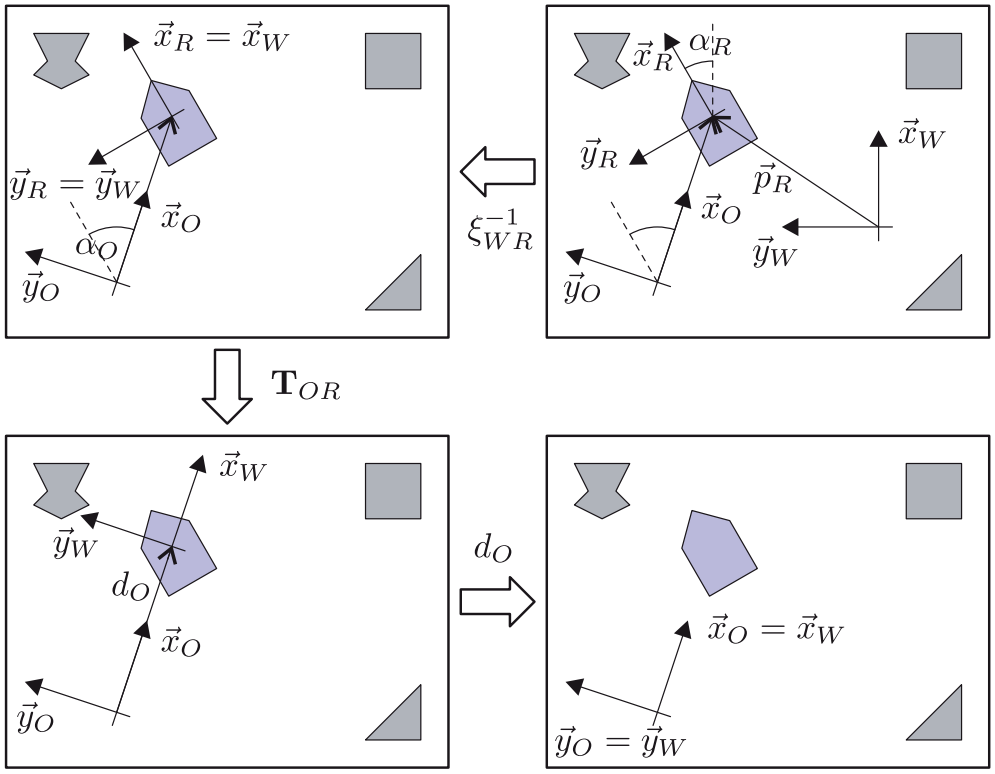

Abbildung 3.7.: Bewegung eines Betrachters um ein Referenzsystem.

Die Matrix $\mathbf{T}_{OW}$ der Basisrotation wird aus der *Tracking*-Sequenz gewonnen, s. Gleichung (3.7), die Position $\vec{p}_O$ des Betrachters ist durch die Translation entlang der Basisvektoren des Betrachtersystems festgelegt.

Bewegung relativ zu einem Referenzsystem

Zur Visualisierung einer Verkehrssimulation ist es wünschenswert einen Betrachter relativ zu einem Verkehrsteilnehmer bewegen zu können, um das Verkehrsgeschehen aus der Sicht eines speziellen Teilnehmers beobachten zu können. Abbildung 3.7 veranschaulicht die Platzierung eines Betrachterkoordinatensystems relativ zu einem Referenzsystem R.

Zuerst werden die Weltkoordinaten durch die Matrix $\xi_{WR}^{-1} = \xi_{RW}$ in das Referenzsystem transformiert. Danach wird das Referenzsystem durch die Basisrotation $\mathbf{T}_{OR}$ entlang des Betrachtersystems ausgerichtet. Letztendlich wird das Betrachterkoordinatensystem noch um die Strecke $(d_O, 0, 0)^T$ entlang der x-Achse verschoben. Man erhält die gesuchte Starrkörpertransformation ξ_{OW}.

$$\xi_{OW} = \xi_{OR} \cdot \xi_{WR}^{-1} = \xi(\,\mathbf{T}_{OR}, (d_O, 0, 0)^T\,) \cdot \xi_{RW}$$

Durch Anwendung der Transformation ξ_{OR} hat ein Betrachter die Möglichkeit das Referenzsystem auf beliebigen Radien d_O einer Kugel zu umkreisen. Die Transformation ξ_{WR} des Referenzsystems wird aus den Koordinaten eines Verkehrsteilnehmers berechnet.

3.4. Ausleuchtung der Szenerie

Die Betrachtermatrix sorgt für ein geometrisch korrektes Verhältnis der virtuellen Objekte zueinander. Eine korrekte optische Wiedergabe der Szene bedingt jedoch die Berechnung eines Farbtones für jedes Bildelement der Anzeigefläche.

Der Mensch nimmt die Krümmung von Oberflächen durch deren Schattierung war. Es ist also durch geeignete Verfahren die Krümmung von Oberflächen zur Schattierung der Szenerie zu simulieren.

Die Position eines Objektes im Verhältnis zu seiner Umgebung wird vom Menschen durch den Objektschatten bestimmt. Eine effiziente Berechnung zur Erzeugung von Schatten ist somit wünschenswert, damit ein Anwender der Visualisierung ein Objekt nicht "schwebend" in der Umwelt wahrnimmt.

In diesem Abschnitt soll das eingesetzte Beleuchtungsmodell und die Erzeugung dreidimensionaler Schatten erläutert werden. Es wird dabei insbesondere auf das Zusammenspiel mit der OpenGL-Pipeline eingegangen.

3.4.1. Beleuchtungsmodell

OpenGL unterstützt ein lokales Beleuchtungsmodell. Lokale Beleuchtungsmodelle schattieren die Oberfläche eines Objektes nur anhand der Eigenschaften der Oberfläche und der einfallenden Lichtquellen. Es findet keine Interaktion mit anderen Oberflächen, z.B. durch Reflexion, statt.

Die durch OpenGL bereitgestellte Lichtgleichung, s. [Shreiner u. a. 2006], unterstützt die Modellierung verschiedener Lichtquellen und Reflexionen. Diese Visualisierung verwendet jedoch nicht alle Möglichkeiten der Lichtgleichung. Die vereinfachte Form der Lichtgleichung lautet

$$c = a_w \cdot a_m + \sum_i \left(a_i \cdot a_m + \max(\vec{L}_i \bullet \vec{n}, 0) \cdot d_i \cdot d_m \right) . \tag{3.8}$$

In der Vektor-Stufe der OpenGL-Pipeline wird diese Gleichung für jeden Eckpunkt einer Form ausgewertet. Das Ergebnis c steht dabei stellvertretend für die Farbkomponenten Rot, Grün und Blau der drei Grundfarben. Für jede Grundfarbe können die Parameter a_x und d_x der Gleichung individuell vorgegeben werden.

Durch Gleichung (3.8) wird sowohl der Einfluß des Hintergrundleuchtens[9] als auch der Einfluß gerichteter Lichtquellen berücksichtigt.

Das Hintergrundleuchten ist ein Resultat der Mehrwegeausbreitung des Lichts. Durch fortwährende Reflexion der Lichtstrahlung an Oberflächen ist die Lichtquelle einer beleuchteten Fläche nicht mehr zu bestimmen. Diese indirekte Beleuchtung wird in Gleichung (3.8) durch den Faktor a_w für das Hintergrundleuchten der virtuellen Welt berücksichtigt. Jede Lichtquelle i trägt durch den Faktor a_i ebenfalls zur Hintergrundbeleuchtung bei. Die Empfindlichkeit der Oberfläche auf das Hintergrundleuchten kommt durch den Faktor a_m[10] zum Ausdruck.

Wird eine diffuse Oberfläche von einer gerichteten Lichtquelle beleuchtet, ist die Intensität des reflektierten Lichtes abhängig vom Einfallswinkel ϑ.

$$I = I_{max} \cdot \cos\vartheta = I_{max} \cdot (\vec{L} \bullet \vec{n})$$

Dieser Zusammenhang wird auch als Lambert'sches Gesetz bezeichnet. Dabei ist $\vec{n}$ der Normalenvektor der Oberfläche und $\vec{L}$ der Richtungsvektor der Lichtquelle.

OpenGL implementiert diesen Zusammenhang durch den Ausdruck

$$\max(\vec{L}_i \bullet \vec{n}, 0) \cdot d_i \cdot d_m$$

der Lichtgleichung. Durch Beschränkung auf positive Einfallswinkel werden von der Lichtquelle i abgewandte Oberflächen nicht durch diffuses Licht beeinflusst. Die Faktoren d_i und d_m geben die Intensität des diffusen Lichtes bzw. die Empfindlichkeit einer Oberfläche für diffuses Licht an. Zu beachten

[9] engl.: *ambient lighting*
[10] engl.: *material*

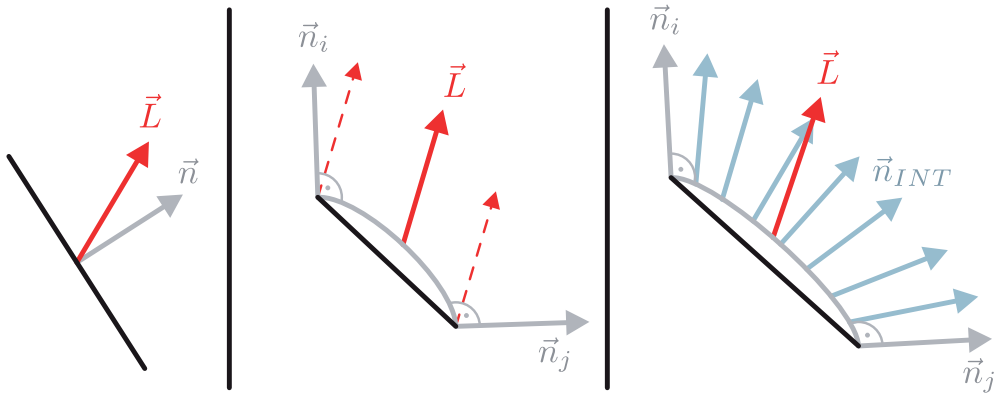

Abbildung 3.8.: Verschiedene Verfahren zur Schattierung einer Fläche. Flat- (links), Gouraud- (mitte) und Phong-Shading (rechts).

bleibt, daß Richtungsvektoren von Lichtquellen bei OpenGL immer auf die Lichtquelle gerichtet sind.

Damit die dargestellten Flächen ein möglichst realitätsnahes Aussehen aufweisen, sind die Parameter der Lichtgleichung in Abhängigkeit des Oberflächenmaterials anzupassen. Eine Anpassung an "reale" Verhältnisse gestaltet sich jedoch oft schwierig, s. z.B. [Rost 2006]. Zur Visualisierung des Fahrzeugumfeldes wurde ein Verhältnis zwischen Hintergrundleuchten und diffusem Licht von 20% zu 80% gewählt. Wird ein dreidimensionales Objekt durch eine entsprechende Anzahl an Dreiecken approximiert, nimmt ein Betrachter keine Unstetigkeiten der Objektoberfläche wahr.

Gleichung (3.8) wird von OpenGL in der Vektor-Stufe für jeden Eckpunkt einer Form ausgewertet. Der resultierende Farbwert wird in der Fragment-Stufe dem ausgetasteten Bildelement der Form zugewiesen. Es existieren verschiedene Verfahren zur Berechnung des resultierenden Farbwertes der Fragment-Stufe, s. Abbildung 3.8.

Beim Flat-Shading wird die Lichtgleichung einmal pro Fläche ausgewertet. Der resultierende Farbwert wird in der Fragment-Stufe dem ausgetasteten Bildelement zugewiesen. Dieses Verfahren benötigt die Angabe eines Normalenvektors pro Fläche und wird direkt durch OpenGL unterstützt.

Das Gouraud-Shading wird ebenfalls direkt von OpenGL unterstützt, benötigt jedoch die Angabe eines Normalenvektors pro Eckpunkt einer Fläche. In der Vektor-Stufe findet dann eine Berechnung der Lichtgleichung pro Eckpunkt statt. Der resultierende Farbwert wird während der Austastung in der Rasterung-Stufe linear über die Oberfläche interpoliert. Das Ergebnis der

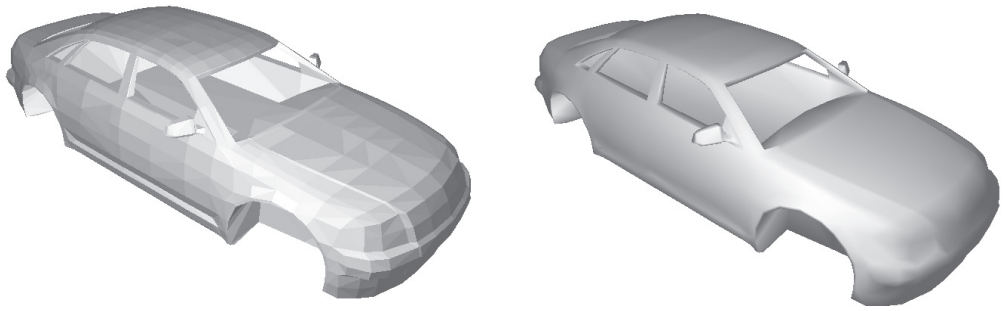

Abbildung 3.9.: Vergleich zwischen Flat- (links) und Gouraud-Shading (rechts).

Interpolation wird in der Fragment-Stufe als Farbwert des Bildelementes verwendet.

Aufwendiger gestaltet sich das Phong-Shading, bei dem zur Austastung in der Rasterung-Stufe die Normalenvektoren der Eckpunkte linear über eine Oberfläche interpoliert werden. Die Lichtgleichung wird dann erst während der Fragment-Stufe ausgewertet. Dadurch können Lichteffekte in der Auflösung des Anzeigegerätes berechnet werden. Dieser Ansatz ist jedoch nur mit der programmierbaren Pipeline OpenGLs möglich, s. [Rost 2006].

Die für das Gouraud- und Phong-Shading benötigten Normalenvektoren sind nicht mit den Flächennormalen der Dreiecke eines darzustellenden Drahtgittermodells identisch. Im Gegensatz zu den Flächennormalen stehen sie Senkrecht auf der durch die resultierende Schattierung zu approximierenden gekrümmten Oberfläche.

Um eine ansprechende Darstellung gekrümmter Flächen mit einer geringen Anzahl an Dreiecken darstellen zu können, ist zumindest der Einsatz des Gouraud-Shading erforderlich. Dazu muß jeder Eckpunkt des Drahtgittermodells über einen Normalenvektor verfügen, der die Krümmung approximiert. Einen Vergleich zwischen Flat- und Gouraud-Shading zeigt Abbildung 3.9.

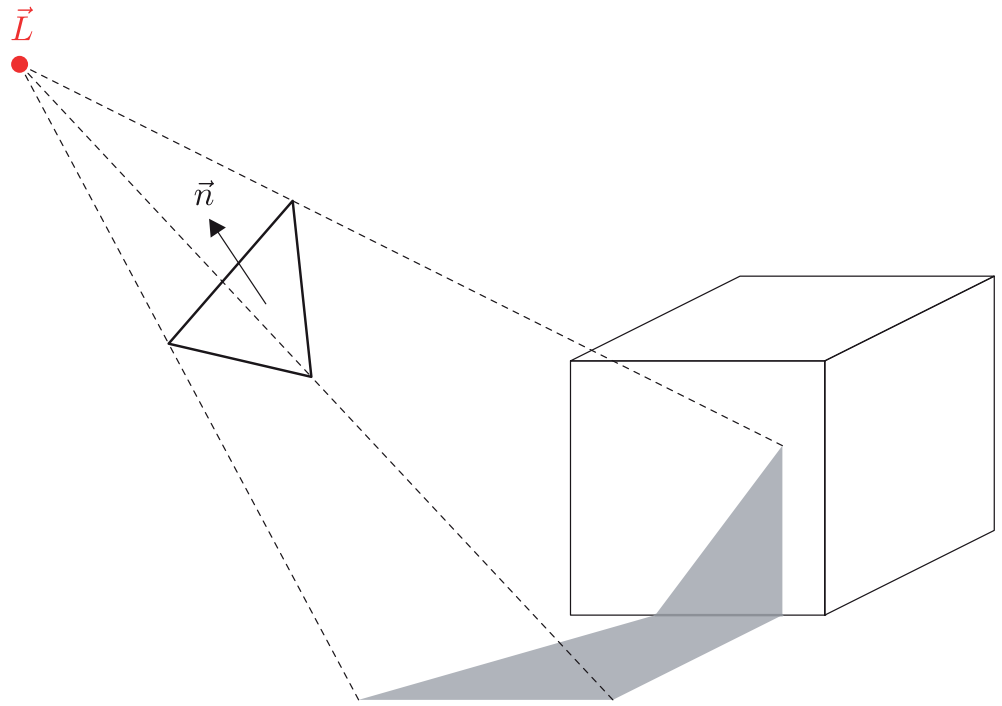

Abbildung 3.10.: Schattenvolumen einer punktförmigen Lichtquelle.

3.4.2. Berechnung dreidimensionaler Schattenvolumina

Die Schattierung von Objektoberflächen wird dazu verwendet, die Krümmung der Objekte für einen Betrachter sichtbar zu machen. Dies ermöglicht es einem Betrachter jedoch nicht ein geometrisches Verhältnis zwischen dem Objekt und der Szenerie herzustellen. Dazu ist die Erzeugung der zugehörigen Objektschatten erforderlich. Der Schatten ermöglicht es einem Betrachter ein Objekt in seiner Umwelt zu "verankern".

In der Computergraphik werden Schatten durch Unterdrückung der Schattierung der im Schattenraum befindlichen Bildelemente erzeugt. Die Kanten einer Oberfläche, die die Lichtausbreitung blockiert, bilden dabei die Seitenflächen eines sog. Schattenvolumens, Abbildung 3.10. Das Schattenvolumen wird dabei in Richtung der Lichtquelle durch die lichtundurchlässige Oberfläche begrenzt. Die lichtabgewandte Begrenzung des Schattenvolumens ist so zu wählen, daß die gesamte Szene durch das Schattenvolumen abgedeckt werden kann.

Der Test, ob sich ein Bildelement in einem Schattenvolumen befindet, kann in der Fragmente-Stufe der OpenGL-Pipeline stattfinden. Dazu wird

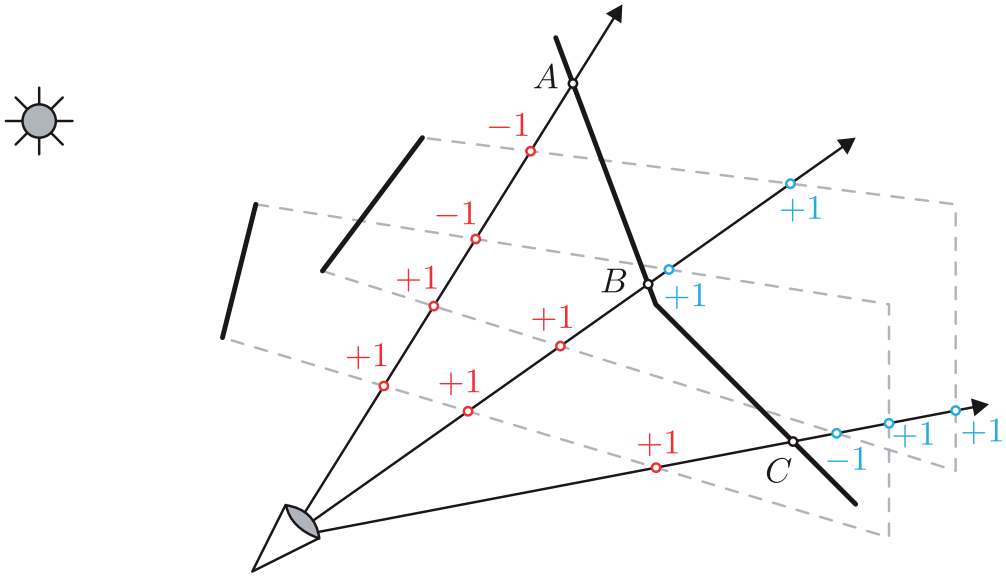

Abbildung 3.11.: Zur Erzeugung von Schattenvolumen durch den Masken- und Tiefenspeicher. Dargestellt sind die Ergebnisse des *zPass*- und des *zFail*-Algorithmus (nach [Akenine-Möller u. Haines 2002]).

die dreidimensionale Koordinate des Bildelementes auf eine Zugehörigkeit zum Schattenvolumen geprüft und entsprechend schattiert.

Komplexe Szenerien verhindern jedoch den Einsatz dieses Vorgehens, da evtl. mehrere Millionen Bildelemente mit mehreren Schattenvolumina geprüft werden müssen. [Everitt u. Kilgard 2002] stellen einen Algorithmus vor, der den Masken- und den Tiefenspeicher der OpenGL-Pipeline verwendet. Operationen mit diesen Speichern werden direkt durch die Graphik-Hardware unterstützt.

Beschreibung des Algorithmus

Abbildung 3.11 dient zur Verdeutlichung des Algorithmus. Es soll geprüft werden, ob sich die Elemente A, B und C innerhalb eines Schattenvolumens befinden.

Befindet sich die Tiefeninformation der Szene im Bildspeicher OpenGLs, kann der Maskenspeicher zum Zählen der Schnitte eines Sehstrahls mit den Begrenzungsflächen eines Schattenvolumens verwendet werden. Initialisiert wird jedes Element des Maskenspeichers mit 0. Während des Zählens werden Schreibzugriffe auf den Tiefenspeicher unterbunden.

Es existieren zwei Varianten dieses Algorithmus, die sich in Aufwand und Robustheit unterscheiden. Beide Algorithmen nutzen die Austastung der Begrenzungsflächen eines Schattenvolumens, um die Zugehörigkeit eines Bildelementes zu einem Schattenvolumen zu bestimmen. Die Austastung muß dabei für den Betrachter zu- und abgewandte Begrenzungsflächen separat durchgeführt werden.

Die erste Möglichkeit besteht darin, den Maskenspeicher zu inkrementieren bzw. zu dekrementieren, falls eine Begrenzungsfläche vor der bereits berechneten Szenerie liegt und somit der Tiefentest jedes erzeugten Fragmentes in der Bildspeicher-Stufe der Pipeline erfolgreich ist. In Anlehnung an dieses Vorgehen wird diese Variante als *zPass*-Algorithmus bezeichnet.

Zuerst wird für die erzeugten Fragmente der dem Betrachter zugewandten Begrenzungsflächen der Maskenspeicher inkrementiert, danach für abgewandte Begrenzungsflächen wieder dekrementiert, falls der Tiefentest erfolgreich ist. Ein Bildelement befindet sich somit nicht im Schatten, wenn der Maskenspeicher einen Wert von 0 aufweist. Ein fiktiver Sehstrahl hat dann ein Schattenvolumen gleich häufig betreten und wieder verlassen, s. Element A in Abbildung 3.11.

Eine andere Variante ist durch das Inkrementieren und Dekrementieren des Maskenspeichers bei fehlgeschlagenem Tiefentest gegeben. Dann befinden sich die Begrenzungsflächen der Schattenvolumina hinter der Szenerie. Diese Variante wird auch als *zFail*-Algorithmus bezeichnet.

Bei fehlgeschlagenem Tiefentest wird der Maskenspeicher zunächst für alle dem Betrachter abgewandte Begrenzungsflächen inkrementiert und danach für dem Betrachter zugewandte Begrenzungsflächen dekrementiert. Auch hier befinden sich Elemente mit einem Maskenspeicherinhalt von 0 nicht im Schatten.

Zu bemerken bleibt, daß die Summe der Ein- bzw. Austritte eines Sehstrahl in die Schattenvolumina durch beide Varianten identisch berechnet werden, s. Element C in Abbildung 3.11.

Die *zPass*-Variante ist einfacher zu implementieren, da die Schattenvolumina durch die Tiefe der Szenerie begrenzt werden und somit keine Ausdehnung der Begrenzungsflächen über den Bereich der Szene zu berechnen ist. Bei der *zFail*-Variante hingegen sind die Schattenvolumina über den Bereich der Szene auszudehnen, da der eigentliche Test von der Verdeckung der Schattenvolumina durch die Szenerie abhängig ist.

Beim *zPass*-Ansatz muß der Maskierungsspeicher in Abhängigkeit der Position des Betrachters mit der Anzahl der Ein- und Austritte der Sehstrahlen

aus den Schattenvolumina initialisiert werden. Da sich ein Betrachter durch eine Kollisionsabfrage i.d.R. nicht hinter der Szenerie bewegen kann, ist der *zFail*-Ansatz unabhängig von der Position des Betrachters.

[Everitt u. Kilgard 2002] schlagen die Verwendung der *zFail*-Variante vor. Um die Schattenvolumina über den Maximalbereich einer Szene auszudehnen, werden die Begrenzungsflächen nach Unendlich projiziert, um die lückenlose Abdeckung einer beliebigen Szene zu gewährleisten.

Anpassung der Projektionsmatrix

OpenGLs Projektionsmatrix ist durch Gleichung (3.6) gegeben. Die Matrix zur Projektion nach Unendlich ist durch

$$\xi_{P,\infty} = \lim_{f\to\infty} \xi_P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.9}$$

gegeben, wobei die rückseitige Fläche des Sichtfeldes durch einen Grenzübergang nach Unendlich verschoben wurde.

Dieser Ansatz ist möglich, da OpenGL Vektoren durch sog. homogene Koordinaten beschriebt, s. Abschnitt A.9. Homogene Vektoren bestehen aus einem dreidimensionalen Vektoranteil und der homogenen Komponente w.

$$\vec{v} \Rightarrow \begin{pmatrix} \vec{v} \\ v_w \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ v_w \end{pmatrix}$$

Ortsvektoren besitzen eine homogene Komponente von $w = 1$, Richtungsvektoren von $w = 0$. Daher ermöglicht Gleichung (3.9) die Projektion eines Vektors nach Unendlich. Die w-Komponente ist für den entsprechenden Vektor auf 0 zu setzen.

Tabelle 3.1 zeigt Beispiele der Projektion durch Anwendung der Matrix $\xi_{P,\infty}$. Die Vektoren mit $z = -n$ und $z = -f$ entsprechen Punkten auf den Deckflächen des Sichtfeldes entlang der negativen z-Achse. Der Bereich des Sichtfeldes wird somit von $[-n, -f]$ auf $[-1, 1 - 2\frac{n}{f}]$ abgebildet. Richtungsvektoren mit $w = 0$, die unendlich weit vom Betrachter entfernt sind, werden auf 1 abgebildet.

Vektor	Projektion durch Matrix $\xi_{P,\infty}$	Perspektivische Division $1/w$
$\begin{pmatrix} -n \\ 1 \end{pmatrix}$	$\begin{pmatrix} n-2n \\ n \end{pmatrix}$	$\begin{pmatrix} -1 \\ 1 \end{pmatrix}$
$\begin{pmatrix} -f \\ 1 \end{pmatrix}$	$\begin{pmatrix} f-2n \\ f \end{pmatrix}$	$\begin{pmatrix} 1-2\frac{n}{f} \\ 1 \end{pmatrix}$
$\begin{pmatrix} -z \\ 1 \end{pmatrix}$	$\begin{pmatrix} z-2n \\ z \end{pmatrix}$	$\begin{pmatrix} 1-2\frac{n}{z} \\ 1 \end{pmatrix}$
$\begin{pmatrix} -z \\ 0 \end{pmatrix}$	$\begin{pmatrix} z \\ z \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$

Tabelle 3.1.: z- und w-Komponenten verschiedener Orts- und Richtungsvektoren bei der Projektion nach Gleichung (3.9). Die z-Komponenten wurden mit $0 > -n > -z > -f$ entlang der negativen z-Achse gewählt.

Erzeugung eines Schattenvolumens

Das Schattenvolumen eines Dreiecks, s. Abbildung 3.10, wird durch das Dreieck selbst, seiner Projektion nach Unendlich und den drei Seitenflächen begrenzt.

OpenGL unterscheidet zwischen punktförmigen und gerichteten Lichtquellen, für deren Position ein homogener Vektor mit $w = 1$ oder $w = 0$ angegeben werden muß. Der Richtungsvektor einer gerichteten Lichtquelle ist dabei immer in Richtung der Quelle orientiert.

Mit diesem Wissen können die Seitenflächen des Schattenvolumens eines Dreiecks berechnet werden, s. Abbildung 3.12.

Die Kante eines Dreiecks sei dabei durch die Eckpunkte A und B festgelegt, die bereits zwei der vier Eckpunkte einer Seitenfläche festlegen. Die Projektion des Punktes A nach Unendlich wird durch Berechnung mit dem homogenen Positionsvektor der Lichtquelle durchgeführt.

$$A_\infty \Rightarrow \begin{pmatrix} L_w \cdot \vec{A} - \vec{L} \\ 0 \end{pmatrix} \tag{3.10}$$

Durch Multiplikation des Eckpunktes A mit der homogenen Komponente L_w der Lichtquelle wird für eine punktförmige Lichtquelle mit $L_W = 1$ der

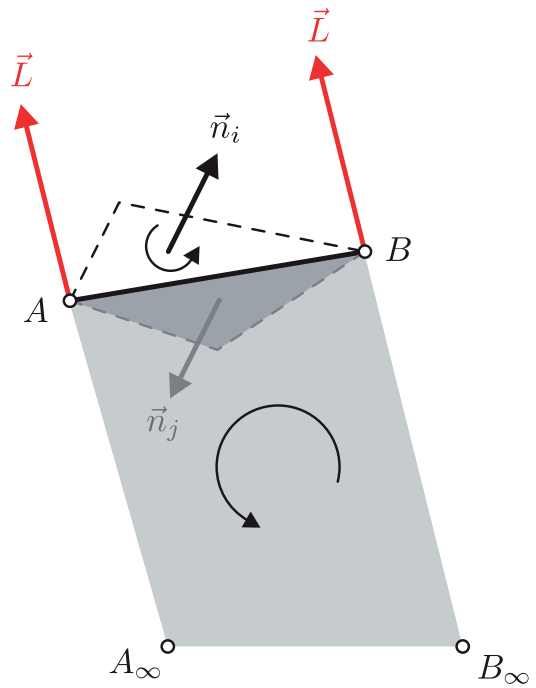

Abbildung 3.12.: Erzeugung der Seitenfläche eines Schattenvolumens.

Richtungsvektor zwischen der Quelle und dem Eckpunkt gebildet. Bei einer gerichteten Lichtquelle mit $L_w = 0$ wird deren Richtungsvektor direkt verwendet. Durch Nullsetzen der homogenen Komponente des resultierenden Vektors erhält man einen Richtungsvektor, der nach Unendlich zeigt.

Die Berechnung des Punktes B_∞ erfolgt analog.

Das resultierende Viereck der Seitenfläche muß in Abhängigkeit des Windungssinns des Dreiecks erzeugt werden, damit dem Betrachter zu- und abgewandte Seitenflächen von der Graphikkarte unterschieden werden können. In Abbildung 3.12 wird der Windungssinn des Dreiecks durch den Normalenvektor $\vec{n}_i$ festgelegt. Die Seitenfläche ist dann in der Reihenfolge $B \rightarrow A \rightarrow A_\infty \rightarrow B_\infty$ zu erzeugen.

Insgesamt müssen drei Vierecke als Seitenflächen des Schattenvolumens eines Dreiecks erzeugt werden. Die beiden Deckflächen werden durch die Eckpunkte A, B und C des Dreiecks und deren Projektion nach Unendlich, A_∞, B_∞ und C_∞, erzeugt.

Für komplexe Szenen ist dieser Ansatz zur Erzeugung der Seitenflächen suboptimal, da durch den Beitrag einer Kante, die in zwei Dreiecken vor-

handen ist, der Maskierungsspeicher einmal inkrementiert und danach dekrementiert wird.

Daher werden nur die Kanten eines Objektes zur Berechnung des Schattenvolumens herangezogen, die die Silhouette eines Objektes bezüglich der Lichtquelle bilden. Solche Kanten weisen benachbarte Dreiecke auf, von denen eines der Lichtquelle zu und das andere der Lichtquelle abgewandt ist. Die Bedingung für Silhouettenkanten lautet somit nach Abbildung 3.12

$$\operatorname{sign}(\vec{L} \bullet \vec{n}_i) \neq \operatorname{sign}(\vec{L} \bullet \vec{n}_j)\,.$$

Integration des Algorithmus

Üblicherweise wird das zweidimensionaler Abbild einer Szene durch den einmaligen Durchlauf eines Algorithmus berechnet. Der *zFail*-Algorithmus muß jedoch u.U. das Schattenvolumen eines Objektes für mehrere Lichtquellen berechnen und zu einem Ergebnis zusammenfassen. Daher ist es erforderlich die Szene mehrfach, unter Einfluß der verschiedenen Lichtquellen, zu berechnen. Algorithmus 1 verdeutlicht die Modifikation der Berechnungsvorschrift einer Szene durch den *zFail*-Algorithmus.

Die Schritte 1 bis 3 sind, bis auf den fehlenden Einfluß verschiedener Lichtquellen, mit einem konventionellen Algorithmus zur Berechnung einer Szene, identisch. Durch die fehlenden Lichtquellen berechnen sich die Elemente des Farbspeichers nur durch den Anteil $a_w \cdot a_m$ aus Gleichung (3.8).

Die Verwendung des *zFail*-Algorithmus setzt einen deaktivierten Farb- und Tiefenspeicher voraus, s. Schritt 4. Der Tiefentest findet jedoch weiterhin für jedes erzeugte Fragment statt.

Jetzt wird in den Schritten 5 bis 13 der Einfluß aller Lichtquellen unter Berücksichtigung der Objektschatten berechnet.

Dazu wird für jede Lichtquelle i der Maskierungsspeicher mit 0 initialisiert und zusammen mit dem Farbspeicher zur Verwendung durch den *zFail*-Algorithmus konfiguriert.

Die Objektschatten können jetzt in den Schritten 8 bis 10 berechnet werden. Nach dieser Berechnung enthält der Maskierungsspeicher Werte ungleich 0, falls sich ein Bildelement innerhalb eines Schattenvolumens befindet.

Der Einfluß der Lichtquelle wird in Schritt 11 zur erneuten Berechnung der Szene aktiviert. Diese Berechnung, s. Schritt 12, addiert zu den bereits im Farbspeicher errechneten Werten den Anteil $a_i \cdot a_m + \max(\vec{L}_i \bullet \vec{n}, 0) \cdot d_i \cdot d_m$

Algorithmus 1 Integration des *zFail*-Algorithmus in die Berechnung der dargestellten Szene.

1: Setze den Tiefen- und den Farbspeicher zurück.
2: Lade die Betrachtermatrix ξ_{MVP}.
3: Berechne die Szene nur unter Einfluß des Hintergrundleuchtens. Aktualisiere dabei den Tiefen- und den Farbspeicher.
4: Unterbinde die Aktualisierung des Tiefenspeichers und deaktiviere das Hintergrundleuchten.
5: **for** Lichtquelle i **do**
6: Unterbinde die Aktualisierung des Farbspeichers und konfiguriere den Maskierungsspeicher zur Verwendung des *zFail*-Algorithmus.
7: Setze den Maskierungsspeicher auf 0.
8: **for** Objekt j **do**
9: Erzeuge das Schattenvolumen j durch Lichtquelle i.
10: **end for**
11: Aktiviere die Lichtquelle i.
12: Berechne die Szene unter Berücksichtigung des Hintergrundleuchtens und des diffusen Anteils der Lichtquelle i. Aktualisiere nur die Elemente des Farbspeichers, die sichtbar sind und sich nicht in einem Schattenvolumen befinden.
13: **end for**

aus Gleichung (3.8), der durch den Einfluß der Lichtquelle i entsteht. Dabei werden nur die Elemente berücksichtigt, die sichtbar sind, d.h. deren Tiefe der bereits im Tiefenspeicher berechneten Tiefe entspricht und die sich nicht in einem Schattenvolumen befinden, d.h. deren Maskierungswert gleich 0 ist.

Die durch den Algorithmus erzeugten Schattenvolumen besitzen scharfe Kanten, d.h. es wird nur der Kernschatten[11] eines Objektes nachgebildet. Die Erzeugung des Halbschattens[12] erfordert Lichtquellen mit endlicher Ausdehnung. OpenGL unterstützt bisher jedoch nur infinitesimal kleine Lichtquellen, so daß die Ausdehnung einer Lichtquelle durch den Anwendungsentwickler in den programmierbaren Pipeline-Stufen nachgebildet werden muß.

3.5. Darstellung des Fahrbahnverlaufes

Das in Kapitel 2 vorgestellte Straßenmodell beschreibt den dreidimensionalen Verlauf einer infinitesimal dünnen Fahrbahnmittellinie. Zur Darstellung auf einem Anzeigegerät ist aus diesem Verlauf ein räumliches Drahtgittermodell zu erzeugen, das durch eine Graphikkarte dargestellt werden kann.

Dazu wird in drei Schritten vorgegangen. Zuerst werden aus der analytischen Beschreibung des Fahrbahnverlaufes Stützpunkte und Normalenvektoren berechnet, die die räumlichen Abmaße der Fahrbahn festlegen. Diese werden im nächsten Schritt in einer Datenstruktur zur Beschreibung der Fahrbahngeometrie zusammengefasst, um diese Informationen effektiv durch OpenGL verarbeiten zu können. Letztlich werden diese Datenstrukturen nochmals sortiert, um schon auf der Anwendungsebene eine Optimierung bei der Auswahl der darzustellenden Bereiche vorzunehmen. Das Resultat zeigt Abbildung 3.13 und wird als Damm bezeichnet.

3.5.1. Erzeugung des Drahtgittermodells

Die Stützpunkte des Drahtgittermodells werden aus dem Verlauf der Fahrbahnmittellinie berechnet, s. Abbildung 3.14.

Der analytische Verlauf der Fahrbahnmittellinie wird zur Erzeugung des Drahtgittermodells in äquidistanten Abständen $\Delta\ell$ ausgewertet. Jede Auswertung definiert einen Schnitt s durch das Drahtgittermodell. Ausgehend von einem Punkt $\vec{S}(\ell)$ auf der Fahrbahnmittellinie, werden die den Schnitt

[11] lat.: *Umbra*
[12] lat.: *Penumbra*

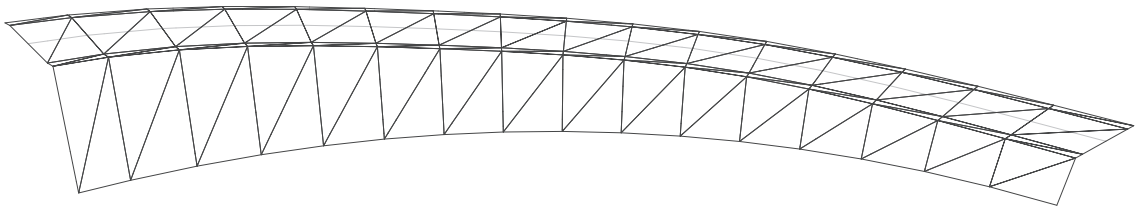

Abbildung 3.13.: Drahtgittermodell des Rundkurses. Der Verlauf der Fahrbahnmittellinie ist grau dargestellt.

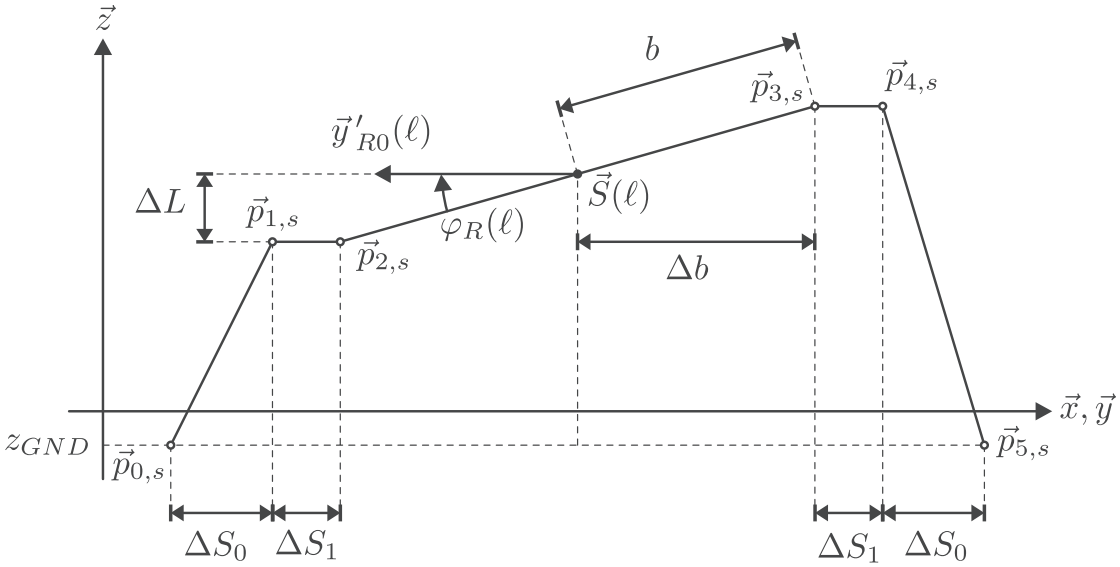

Abbildung 3.14.: Zur Berechnung der Stützpunkte des Drahtgittermodells. Die Querneigung der Fahrbahn ist hier negativ ($\varphi_R(\ell) < 0$).

s definierenden Stützpunkte $\vec{p}_{i,s}$ berechnet. Mit der Querneigung $\varphi_R(\ell)$, der Anzahl der Fahrspuren pro Fahrtrichtung N_L und der Fahrspurbreite w_L kann die Überhöhung ΔL des Fahrbahnrandes bezüglich der Fahrbahnmittellinie und die horizontale Breite Δb der geneigten Fahrbahnoberfläche berechnet werden.

$$\begin{aligned} b &= N_L \cdot w_L \\ \Delta b &= b \cdot \cos \varphi_R(\ell) \\ \Delta L &= b \cdot \sin \varphi_R(\ell) \end{aligned}$$

Mit Gleichung (2.9) wird die y-Achse des Fahrbahnkoordinatensystems in der Horizontalen berechnet.

$$\vec{y}\,'_{R0}(\ell) = \frac{\vec{y}_{R0}(\ell)}{|\,\vec{y}_{R0}(\ell)\,|}$$

Die Randpunkte der Fahrbahn $\vec{p}_{2,s}$ und $\vec{p}_{3,s}$ sind damit festgelegt.

$$\begin{aligned} \vec{p}_{2,s} &= \vec{S}(\ell) + \Delta b \cdot \vec{y}\,'_{R0}(\ell) + \begin{pmatrix} 0 \\ 0 \\ \Delta L \end{pmatrix} \\ \vec{p}_{3,s} &= \vec{S}(\ell) - \Delta b \cdot \vec{y}\,'_{R0}(\ell) - \begin{pmatrix} 0 \\ 0 \\ \Delta L \end{pmatrix} \end{aligned}$$

Mit dem inneren Rand ΔS_1 und dem äußeren Rand ΔS_0 werden die verbleibenden Punkte des Schnitts s berechnet.

$$\begin{aligned} \vec{p}_{1,s} &= \vec{p}_{2,s} + \Delta S_1 \cdot \vec{y}\,'_{R0}(\ell) \\ \vec{p}_{0,s} &= [\vec{p}_{1,s} + \Delta S_0 \cdot \vec{y}\,'_{R0}(\ell)]_{z=z_{GND}} \\ \vec{p}_{4,s} &= \vec{p}_{3,s} - \Delta S_1 \cdot \vec{y}\,'_{R0}(\ell) \\ \vec{p}_{5,s} &= [\vec{p}_{4,s} - \Delta S_0 \cdot \vec{y}\,'_{R0}(\ell)]_{z=z_{GND}} \end{aligned}$$

Für die äußersten Punkte eines Schnitts wird die z-Koordinate zu z_{GND} gesetzt. Dadurch wird dem Drahtgittermodell des Rundkurses, das alle

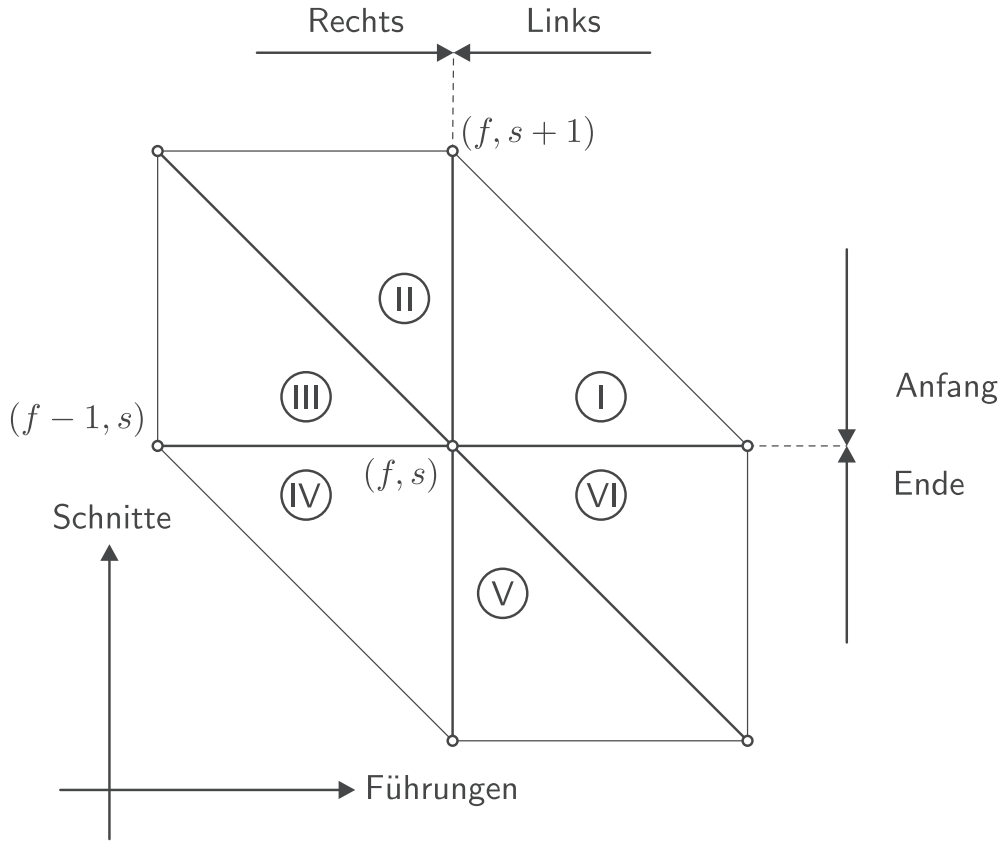

Abbildung 3.15.: Indizierung der Stützpunkte.

Schnitte enthält, das Aussehen eines Damms verliehen. Die Fahrbahnoberfläche erscheint somit für den Betrachter nicht als infinitesimal dünnes, räumliches Band.

Das Drahtgittermodell des Rundkurses kann somit berechnet werden. Um eine ansprechende Darstellung zu erzielen, sind jedoch noch Normalenvektoren für jeden berechneten Stützpunkt zu bestimmen und die Fahrbahnoberfläche mit einer Textur zu belegen.

Um die Normalenvektoren jedes Stützpunktes zu berechnen, ist es notwendig die Nachbarn eines Stützpunktes zu betrachten, s. Abbildung 3.15.

Alle Stützpunkte entlang der y-Achse des Fahrbahnkoordinatensystems werden zu einem Schnitt s zusammengefasst; Stützpunkte gleicher Entfernung zur Mittellinie zu Führungen f. Somit können alle Stützpunkte des Damms durch die Koordinaten (f, s) eindeutig indiziert werden.

Zur Anwendung des Lichtmodells, s. Abschnitt 3.4.1, muß für jeden Stützpunkt ein Normalenvektor vorhanden sein. Für einen Randpunkt des Damms

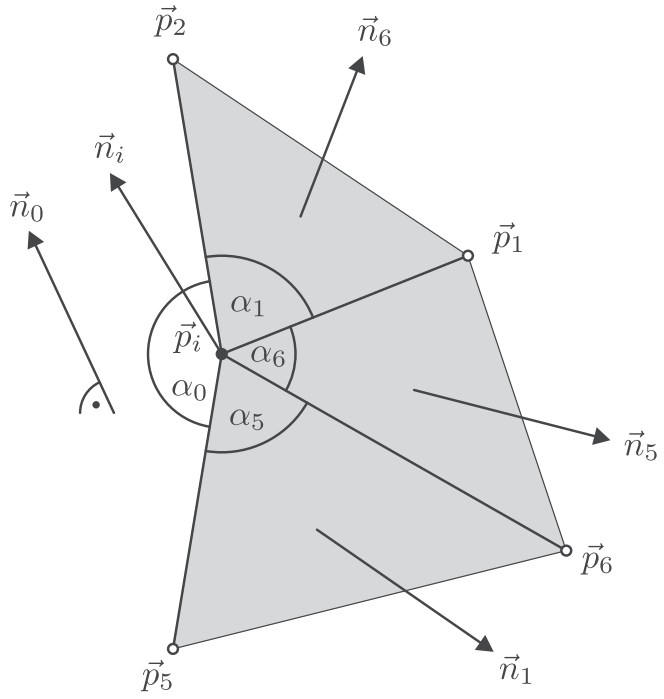

Abbildung 3.16.: Berechnung des Normalenvektors $\vec{n}_i$ eines Randpunktes $\vec{p}_i$. Alle Indizes entsprechen der jeweiligen Sektornummer.

soll exemplarisch die Berechnung des Normalenvektors erläutert werden, s. Abbildung 3.16.

In Anlehnung an [Shankel u. Treglia 2002] wird der Normalenvektor als gewichtete Summe der an den Punkt grenzenden Flächennormalen berechnet. Eine Flächennormale wird aus dem Kreuzprodukt der die Fläche aufspannenden Richtungsvektoren berechnet. Zugunsten der Übersicht wird hier auf eine explizite Kennzeichnung des Schnitts und der Führung verzichtet. Die Stützpunkte und Normalenvektoren werden durch die jeweilige Sektornummer gekennzeichnet.

Die benötigten Richtungsvektoren werden direkt aus den Stützpunkten berechnet, z.B.

$$\vec{d_1} = \vec{p}_1 - \vec{p}_i \,.$$

Mit diesen Richtungsvektoren sind dann die Normalenvektoren und die Öffnungswinkel der Grenzflächen am Stützpunkt $\vec{p}_i$ zu berechnen, z.B.

$$\vec{n}_1 = \frac{\vec{d}_1 \times \vec{d}_2}{\left|\vec{d}_1 \times \vec{d}_2\right|} \quad \text{und}$$

$$\alpha_1 = \arccos\left(\frac{\vec{d}_1 \bullet \vec{d}_2}{\left|\vec{d}_1\right| \cdot \left|\vec{d}_2\right|}\right) .$$

Jetzt kann der Normalenvektor $\vec{n}_i$ durch Wichtung aller Flächennormalen berechnet werden.

$$\vec{n}\,'_i = \alpha_0 \cdot \vec{n}_0 + \alpha_1 \cdot \vec{n}_1 + \alpha_5 \cdot \vec{n}_5 + \alpha_6 \cdot \vec{n}_6$$

$$\vec{n}_i = \frac{\vec{n}\,'_i}{|\,\vec{n}\,'_i\,|}$$

Die abschließende Normierung macht eine Skalierung der Wichtungsfaktoren mit $2 \cdot \pi$ unnötig.

Da es sich bei bei dem Stützpunkt $\vec{p}_i$ um einen Randpunkt handelt, wird eine der Grenzflächen durch die x-y-Ebene gebildet. Der zugehörige Normalenvektor $\vec{n}_0$ ist mit dem Richtungsvektor der z-Achse identisch, der Öffnungswinkel berechnet sich aus

$$\alpha_0 = 2 \cdot \pi - \alpha_1 - \alpha_5 - \alpha_6 .$$

Die Belegung der Fahrbahnoberfläche mit einer Textur erfolgt durch Berechnung geeigneter Textur-Koordinaten und Zuweisung dieser Koordinaten zu den Stützpunkten, s. Abbildung 3.17.

Jedem dreidimensionalen Punkt können in OpenGL die Koordinaten eines zweidimensionalen Bildes zugewiesen werden, das dann die Bewegung des Punktes erfährt. Durch solche Texturen ist es möglich Oberflächen mit einer Vielzahl an Details auszustatten, ohne die Anzahl der dazu benötigten Geometriedaten zu erhöhen. OpenGL verwendet die Koordinaten $(s, t) = [0, 1]$, um auf Elemente einer Textur[13] zugreifen zu können.

Nach Abbildung 3.17 ist die s-Koordinate aller Stützpunkte der Führung 2 auf $s = 0$, die der Führung 3 auf $s = 1$ zu setzen. Für quadratische Tex-

[13] engl.: *texel – texture element*

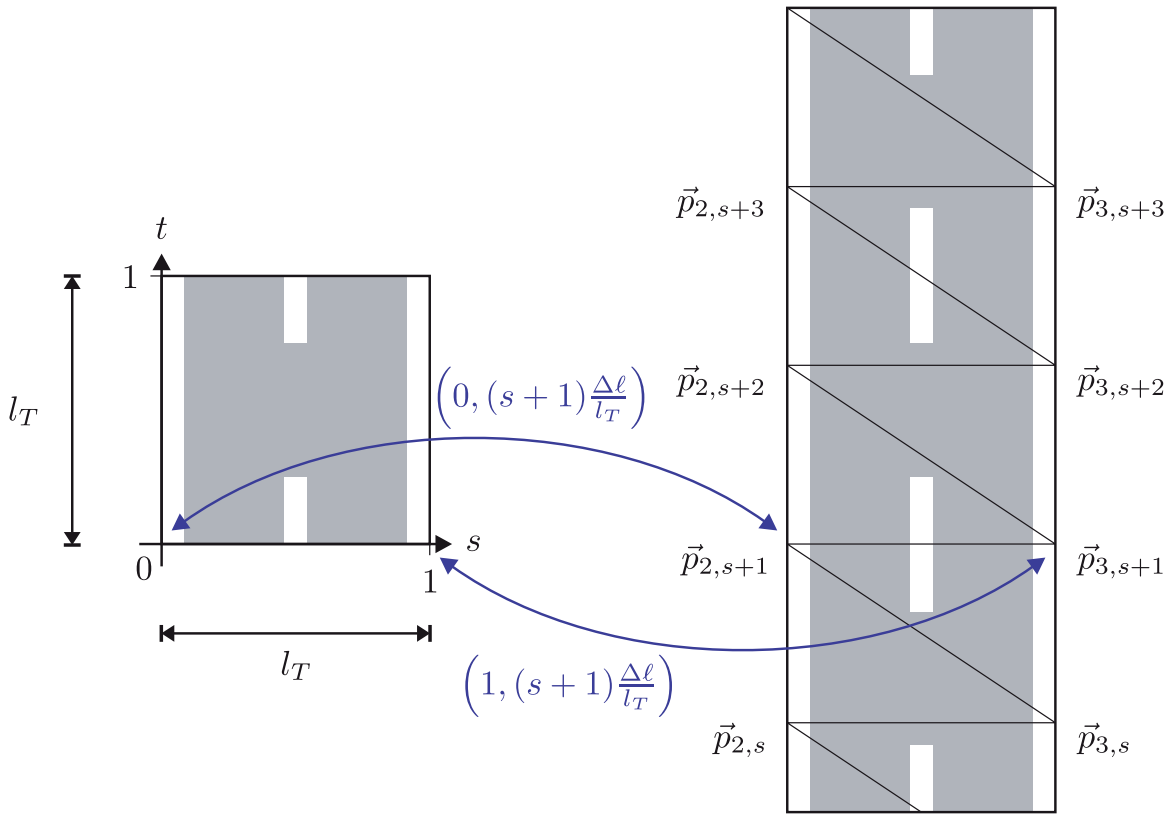

Abbildung 3.17.: Zur Berechnung der Textur-Koordinaten.

turen, deren Länge in Weltkoordinaten zu l_T angenommen wird, lassen sich die t-Koordinaten durch die bekannte Bogenlänge des Schnitts s berechnen. Es gilt

$$t = s\frac{\Delta\ell}{l_T}\,,$$

wobei s die Laufnummer der Schnitte ist und $\Delta\ell$ die Schrittweite zur Erzeugung der Schnitte. OpenGL verwendet nur die Nachkommastellen der Textur-Koordinaten und berechnet einen Überlauf, so daß die Fahrbahnoberfläche fortlaufend mit der Textur belegt ist.

Ein Eckpunkt des Drahtgittermodells wird somit durch seinen Stützpunkt, den zugehörigen Normalenvektor und im Falle der Führungen 2 und 3 durch die Textur-Koordinaten beschrieben. Die Einzelvektoren eines solchen Datensatzes werden auch als Attribut bezeichnet.

Die Schritte zur Darstellung des Fahrbahnverlaufes zeigt Abbildung 3.18.

3.5.2. Optimierung des Datenstromes

Die OpenGL-Pipeline ist zur Darstellung von Dreiecken optimiert, die von der Anwendung an die Graphikkarte gesendet werden müssen. Jeder der drei Eckpunkte ist dazu explizit anzugeben, s. Listing 3.1. Für N Dreiecke sind somit $3 \cdot N$ Punkte an die Graphikkarte zu senden. Diese Anzahl verdoppelt sich, falls ebenfalls die Normalenvektoren zur Berechnung eines Lichtmodells an die Graphikkarte gesendet werden sollen.

Durch die Verwendung sog. Dreiecksbänder[14] kann die zu übertragene Datenmenge minimiert werden, s. Abbildung 3.19.

Dabei wird ein Dreieck nach der Definition zweier Eckpunkte durch jeden weiteren Punkt definiert, wobei die Zählung bei 1 beginnt. Dreiecke mit ungerader Laufnummer werden durch die Punkte n, $n+1$ und $n+2$ definiert, Dreiecke mit gerader Laufnummer durch die Punkte $n+1$, n und $n+2$. Für N Dreiecke sind somit $\frac{N}{3} + 2$ Eckpunkte zu übertragen.

Mit der Konvention aus Abbildung 3.15 kann ein Dreiecksband aus zwei benachbarten Führungen erzeugt werden. Die erforderliche Sequenz der Stützpunkte lautet allgemein

[14] engl.: *triangle strips*

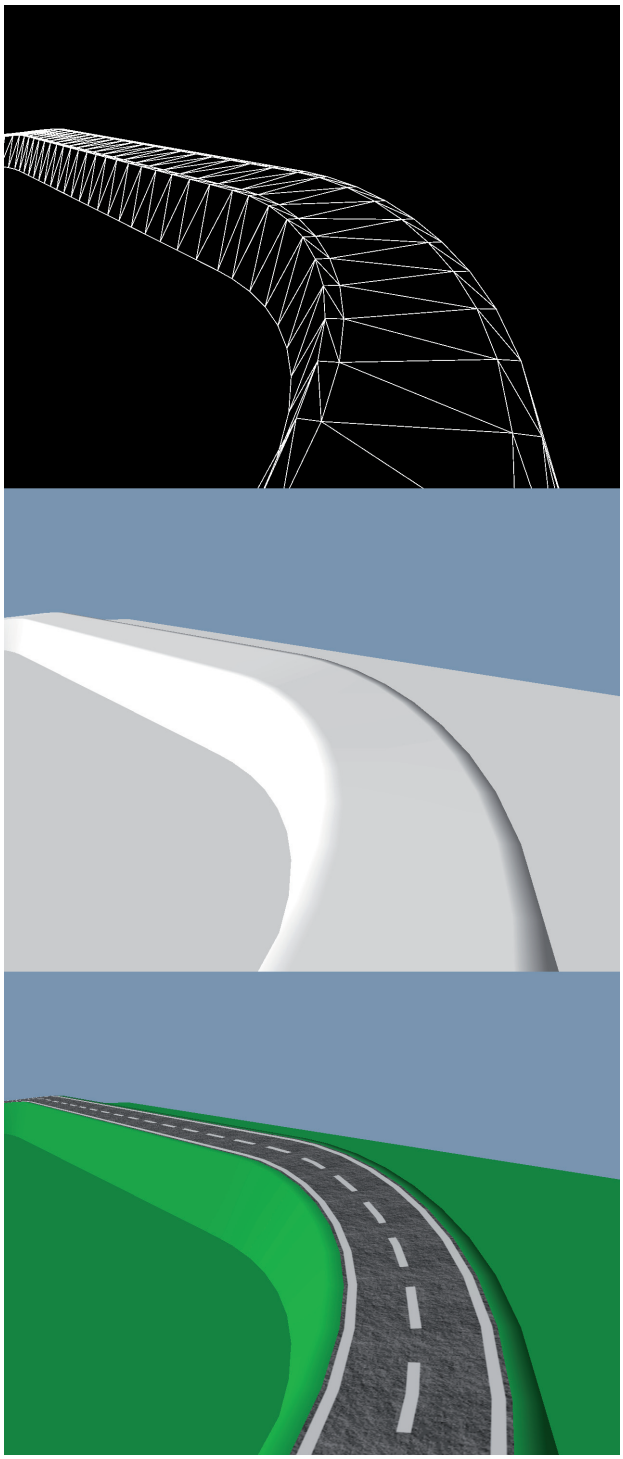

Abbildung 3.18.: Schritte zur Berechnung des Fahrbahnverlaufes. Drahtgittermodell (oben), Beleuchtungsmodell (mitte) und Belegung mit einer Textur (unten).

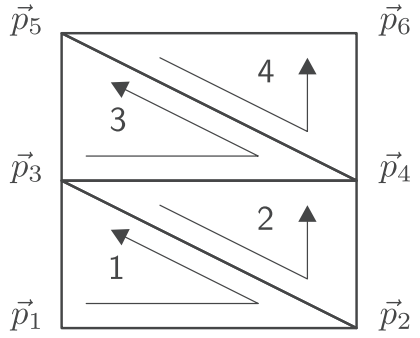

Abbildung 3.19.: Zur Erzeugung eines Dreiecksbandes.

$$\vec{p}_{f,s}, \vec{p}_{f+1,s}, \vec{p}_{f,s+1}, \vec{p}_{f+1,s+1}, \cdots \quad .$$

Um den Graphikrechner weiter zu entlasten, bietet OpenGL die Möglichkeit Datenfelder direkt im Hauptspeicher der Graphikkarte zu verwalten. Zur Berechnung einer Szene werden diese Felder durch ein weiteres Indexfeld, das von der Anwendung an die Graphikkarte gesendet wird, referenziert, um die gewünschte Geometrie zu erzeugen, s. Abbildung 3.20.

Durch die Anwendung werden die Datenfelder beim Laden des Rundkurses mit ihren Elementen initialisiert. Es werden drei Datenfelder für die Stützpunkte, die Normalenvektoren und die Textur-Koordinaten benötigt. Die Dimension jedes Feldes ist dabei $N_F \cdot N_S$, wobei $N_F = 6$ die Anzahl der Führungen festlegt und N_S die Anzahl der Schnitte pro Führung.

Da zwei Führungen zu einem Dreiecksband zusammengefasst werden, müssen durch die Anwendung $N_F - 1 = 5$ Indexfelder erstellt und initialisiert werden. Diese Indexfelder befinden sich jedoch nicht im Speicher der Graphikkarte, sondern werden mit jeder Aktualisierung der Darstellung an diese gesendet. Die Graphikkarte empfängt die Indizes eines Feldes und kann durch simultanes Referenzieren der Daten in allen drei Datenfeldern die gewünschte Geometrie erzeugen. Ein Transport der Geometriedaten findet somit nur auf dem internen Bus der Graphikkarte statt. Vom Graphikrechner zur Graphikkarte werden zur Aktualisierung einer Szene nur die Indizes in die Datenfelder der Geometrie transportiert.

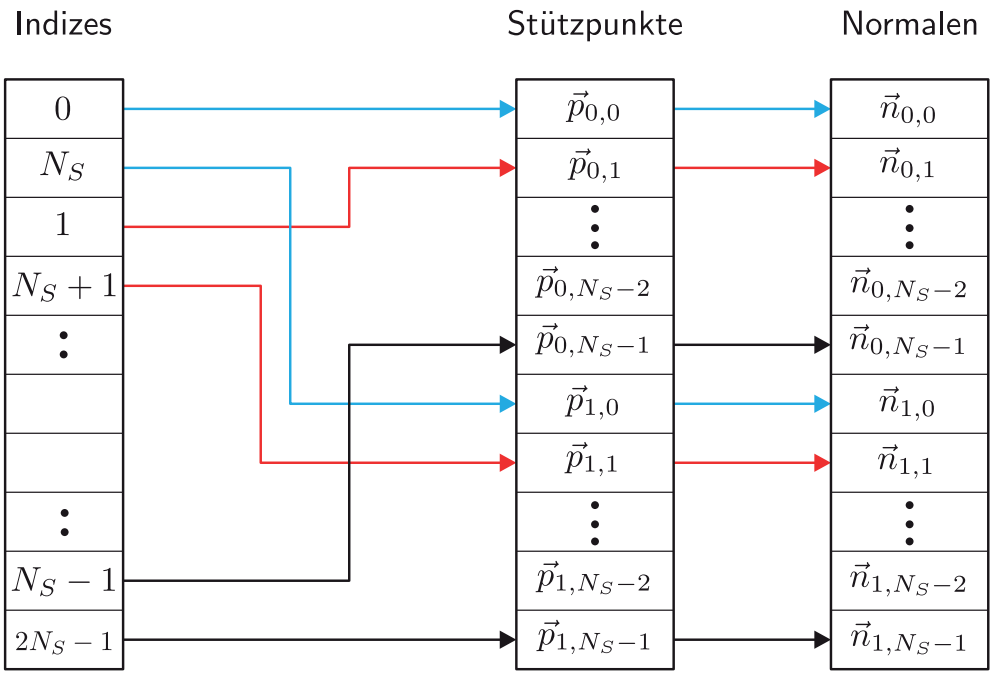

Abbildung 3.20.: Indizierung der Geometriedaten.

3.5.3. Vorausberechnung der sichtbaren Teilvolumina

Zur Erzeugung des Damms müssen pro Schnitt 10 Dreiecke berechnet werden, s. Abbildung 3.14 und Abbildung 3.19. Für einen Rundkurs der Länge L müssen somit

$$10 \cdot \frac{L}{\Delta\ell}$$

Dreiecke verarbeitet werden, wobei $\Delta\ell$ der Abstand zweier Schnitte ist. Es ist daher von Vorteil die Anzahl der durch die Graphikkarte zu verarbeitenden Dreiecke zu begrenzen. Dazu wird der dreidimensionale Raum der Szenerie iterativ unterteilt und die Geometrie in Abhängigkeit ihrer Begrenzungsvolumina den Unterräumen zugeordnet. Besonders effektiv ist dabei eine Unterteilung des Raumes durch sog. Binärbäume[15], s. [Abrash 1997].

Der Damm eines Rundkurses wird durch einen sog. achsenparalleler Binärbaum unterteilt. Da der Rundkurs nur einen zweidimensionalen Verlauf zulassen soll, d.h. es wird auf die Modellierung von Brücken oder Unter-

[15]engl.: *BSP – binary space partitioning*

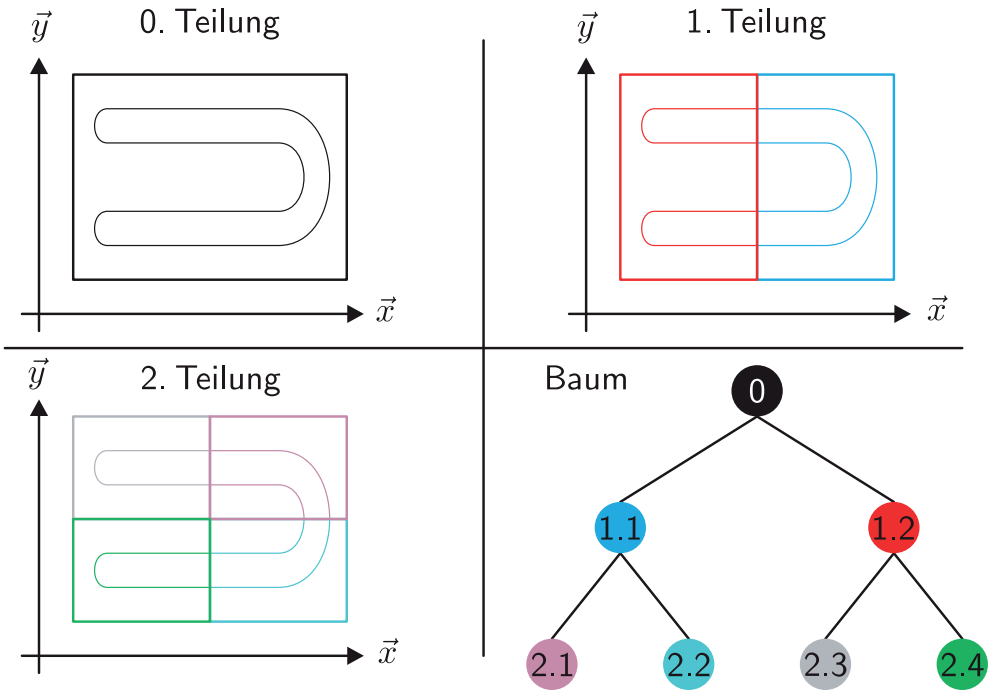

Abbildung 3.21.: Erzeugung des Binärbaumes.

führungen verzichtet, kann die z-Achse ignoriert werden. Die Fläche des Damms wird iterativ entlang der Koordinatenachsen halbiert, wobei die x- und die y-Achse im Wechsel als Bezugsachse der Teilung verwendet werden.

Als Beispiel soll der Verlauf des Rundkurses in Abbildung 3.21 unterteilt werden.

Die erste Teilung halbiert den Damm entlang der x-Achse in die Teilräume 1.1 und 1.2. Die Geometrie der Schnitte, die Stützpunkte, die Normalenvektoren und die Textur-Koordinaten, werden entsprechend in die beiden Teilvolumina sortiert. Der Abschnitt eines Damms innerhalb eines Teilvolumens wird als Teildamm bezeichnet.

Durch die zweite Teilung werden die beiden Teilräume 1.1 und 1.2 entlang der y-Achse wiederum halbiert. Beim erneuten Sortieren der Geometrie ist darauf zu achten, daß evtl. mehrere Teildämme innerhalb eines Unterraumes entstehen können, s. z.B. Teilraum 2.1.

Eine erneute Halbierung der vier Teilräume erfolgt dann wieder entlang der x-Achse. Die Unterteilung ist solange fortzusetzen, bis die maximalen Abmaße eines Teilraumes einen vorgegebenen Wert unterschreiten. Diese kleinsten Teilräume bilden die Blätter des Binärbaumes. Sie enthalten die

Geometrie, die durch Strukturierung nach Abschnitt 3.5.2 effektiv durch die Graphikkarte verarbeitet werden kann.

Zur Darstellung des Damms wird der Baum aus Abbildung 3.21 beginnend mit der Wurzel durchlaufen. Jeder Knoten des Baumes enthält dabei die Abmaße des entsprechenden Teilraumes in Weltkoordinaten. Die Wurzel enthält die Abmaße des gesamten Damms.

Die Abmaße des Sichtfeldes in Weltkoordinaten können mit den Variablen aus Abschnitt 3.3.1 berechnet werden. Mit

$$\begin{aligned} {}_W w &= w \cdot \frac{f}{n} \quad \text{und} \\ {}_W h &= h \cdot \frac{f}{n} \end{aligned}$$

können die Eckpunkte des achsenparallelen Begrenzungsvolumens zu

$$\begin{pmatrix} 0 \\ -\frac{{}_W w}{2} \\ -\frac{{}_W h}{2} \end{pmatrix} \qquad \text{und} \qquad \begin{pmatrix} d \\ \frac{{}_W w}{2} \\ \frac{{}_W h}{2} \end{pmatrix}$$

bestimmt werden. Dieses Begrenzungsvolumen wird durch die Matrix ξ_{OW}^{-1} aus Abschnitt 3.3.3 auf die aktuelle Kameraposition transformiert.

Das Begrenzungsvolumen jedes Knotens wird beim Abstieg des Baumes auf eine Überlappung mit dem transformierten Begrenzungsvolumen des Sichtfeldes überprüft. Existiert keine gemeinsame Schnittmenge zwischen den beiden Volumina, wird die Verarbeitung des Teilbaumes abgebrochen. Im Fall einer Überlappung werden alle Kinder des geprüften Knotens wieder auf eine Überlappung geprüft, bis die Blätter des Baumes erreicht sind. Befindet sich ein Blatt des Baumes innerhalb des Sichtfeldes, ist dessen Geometrie darzustellen.

Der Test aller Teilvolumina auf einen Schnitt mit dem Sichtfeld erfolgt auf dem Graphikrechner und muß daher effizient implementiert sein. Die Verwendung achsenparalleler Teilräume benötigt zum Test zweier Volumina sechs Vergleiche. Ist die Geometrie eines Blattes sichtbar, d.h. es existiert eine Schnittmenge zwischen dem transformierten Sichtfeld und den Abmaßen des Blattes, sind nur die Indizes aller Dreiecksbänder an die Graphikkarte zu senden. Die zu verarbeitende Datenmenge bzgl. Abschnitt 3.5.2 wird so nochmals optimiert, da die Graphikkarte nur den sichtbaren Bereich zu

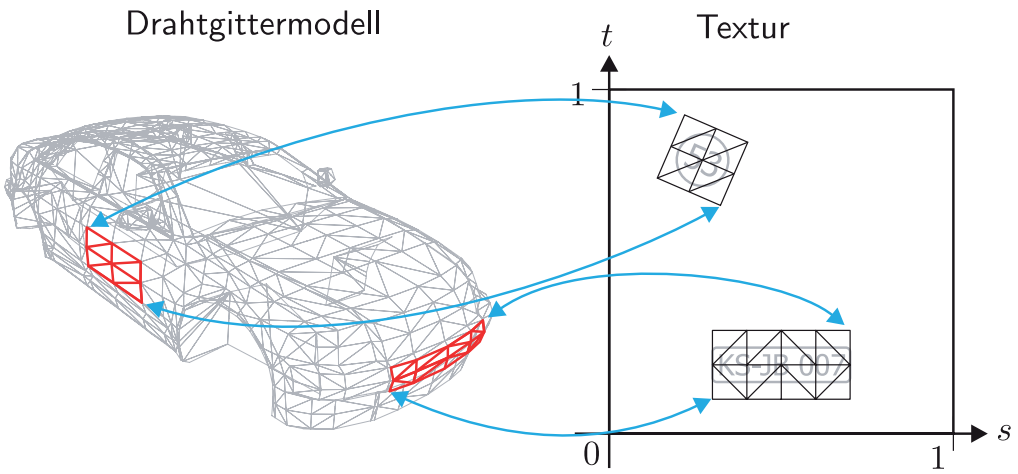

Abbildung 3.22.: Abbildung dreidimensionaler Oberflächen auf eine zweidimensionale Textur.

verarbeiten hat. Unsichtbare Bereiche des Rundkurses werden nicht berücksichtigt.

3.6. Darstellung beliebiger Objekte

Zur Darstellung beliebiger dreidimensionaler Objekte, z.B. der Aufbau oder die Räder eines Fahrzeuges, wird eine Datenstruktur benötigt, die durch OpenGL verarbeitet werden kann.

Als Basis eines Objektes wird wieder ein Drahtgittermodell verwendet, das z.B. mit der Visualisierungs-Software *Maya* der Firma Autodesk erstellt wird. Die zur Darstellung benötigten Informationen, die Indexliste der Dreiecke, die Stützpunkte, die Normalenvektoren und die Textur-Koordinaten, können in ein proprietäres Dateiformat exportiert werden, um durch die Visualisierung verarbeitet zu werden.

Die zweidimensionalen Textur-Koordinaten (s, t) werden dabei jedem Eckpunkt eines Dreiecks individuell zugewiesen, s. Abbildung 3.22.

Dieser Ansatz realisiert eine planare Abbildung der Textur-Fläche auf ein Dreieck und ermöglicht die beliebige Platzierung der Dreiecke des Drahtgittermodells auf der zweidimensionalen Textur.

Durch die Bindung der Textur-Koordinaten an die Eckpunkte der Dreiecke erhöht sich die benötigte Datenmenge von der Anzahl der Stützpunkte N_V auf die dreifache Anzahl der Dreiecke $3 \cdot N_T$. Für das Fahrzeug in Abbildung 3.22 beträgt dieses Verhältnis

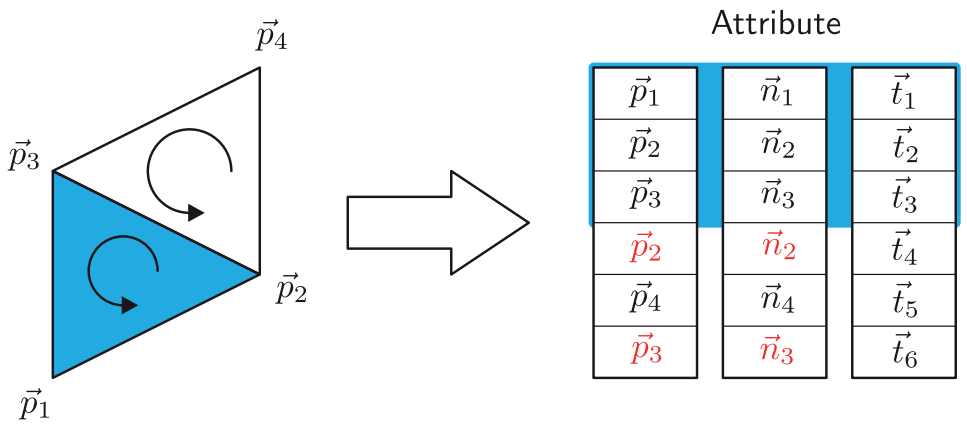

Abbildung 3.23.: Anordnung der Attribute zweier Dreiecke im Graphikspeicher. Redundante Attribute sind rot dargestellt.

$$\frac{3 \cdot N_T}{N_V} = \frac{3 \cdot 2298}{1137} \approx 6,06\,.$$

Da die Graphikkarte jede Komponente eines Vektors als 32 Bit Gleitkommazahl speichert, werden für dieses Fahrzeug

$$3 \cdot 2298 \cdot (3 + 3 + 2) \cdot 4\,\text{Bytes} = 220.608\,\text{Bytes}$$

Hauptspeicher benötigt, wobei die Stützpunkte und Normalenvektoren 3 und die Textur-Koordinaten 2 Vektorkomponenten besitzen.

Der erhöhte Speicherbedarf wird bewusst in Kauf genommen, da ein Objekt durch Laden unterschiedlicher Transformationsmatrizen in beliebigen Positionen dargestellt werden kann und somit nur eine Instanz des Objektes im Graphikspeicher erforderlich ist. Die Anordnung der Attribute im Graphikspeicher erfolgt sortiert nach den Eckpunkten der zu erzeugenden Dreiecke, s. Abbildung 3.23.

Im Gegensatz zu Abschnitt 3.5.2, wo ein Attribut pro Stützpunkt des Drahtgittermodells definiert wurde, werden die Attribute hier für jeden Eckpunkt eines Dreiecks definiert, um obige Zuweisung der Textur-Koordinaten zu ermöglichen. In diesem Fall wird kein Indexfeld benötigt, da die Attribute der Dreiecke bereits sortiert vorliegen. Zur Darstellung der Dreiecke

wird OpenGL die Länge $3 \cdot N_T$ der Attributfelder mitgeteilt. Die Graphikkarte berechnet dann N_T Dreiecke, wobei immer drei aufeinanderfolgende Attribute ein Dreieck definieren.

3.7. Bereitstellung von Objektreferenzen

Der vorherige Abschnitt beschreibt die Definition einer Objektgeometrie, wie sie zur Verarbeitung durch OpenGL erforderlich ist. Zur Erzeugung von Fahrzeugen sind zumindest zwei Teilobjekte zu kombinieren, um den Aufbau und die Räder eines Fahrzeuges darstellen zu können. Die beiden Teilobjekte werden mit Texturen belegt, um Türen oder Felgen darstellen zu können, und müssen entsprechend den Fahrzeugabmaßen relativ zueinander positioniert werden.

3.7.1. Darstellung eines Fahrzeuges

In diesem Abschnitt wird ein Algorithmus vorgestellt, der das Mehrkörpersystem "Fahrzeug", das aus einem starren Aufbau und vier starren Rädern besteht, für die zwei Anwendungsfälle Verkehrssimulation und Fahrdynamiksimulation visualisieren kann. Die beiden Anwendungsfälle unterscheiden sich nur durch zwei Winkel, die zum Ausgleich der Rollbewegung im Fall der Fahrdynamiksimulation benötigt werden. Diese Winkel werden bei der Verkehrssimulation mit 0 belegt, was die Rotationsmatrix der Rollbewegung in eine Einheitsmatrix überführt.

Bei der Darstellung eines Fahrzeuges müssen der Aufbau und die Räder relativ zueinander in Position gebracht werden. Der Ursprung des Bezugssystems V des Fahrzeuges soll sich dabei auf der Fahrbahnoberfläche befinden, was eine Darstellung der Fahrzeugbewegung sowohl mit als auch ohne Rollbewegung ermöglicht.

Der Aufbau B und die Räder T sind in Abbildung 3.24 dargestellt.

Die Menge der Stützpunkte $\vec{\varrho}_B$ des Aufbaus und $\vec{\varrho}_T$ des Rades wird um die Koordinatenursprünge B und T beschrieben, so daß

$$\begin{aligned} \vec{\varrho}_B &\in [\vec{\varrho}_{B,min}, \vec{\varrho}_{B,max}] \qquad \text{und} \\ \vec{\varrho}_T &\in [\vec{\varrho}_{T,min}, \vec{\varrho}_{T,max}] \end{aligned}$$

gilt.

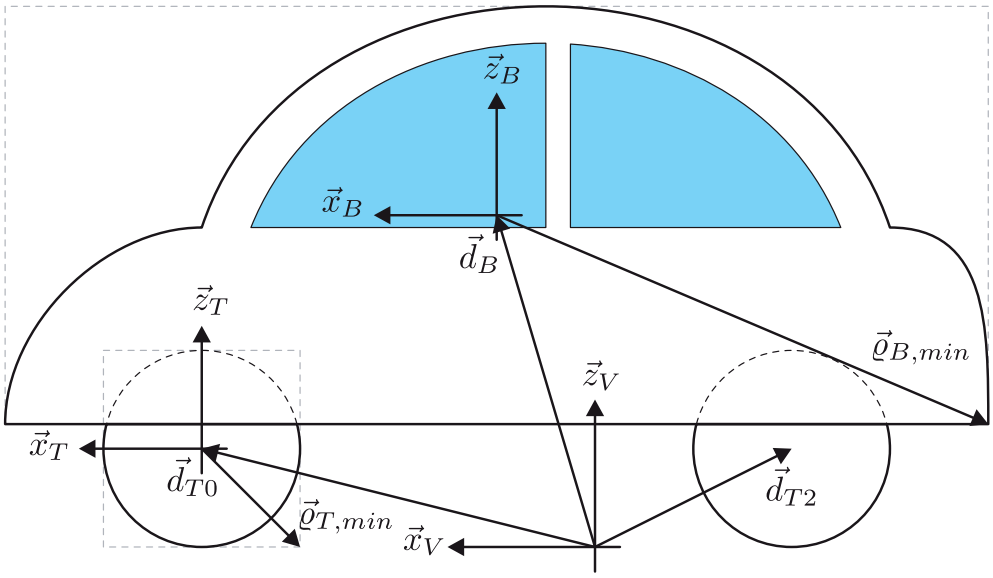

Abbildung 3.24.: Zur Positionierung der Teilobjekte eines Fahrzeuges.

Im Ruhezustand, d.h. ohne Lenk-, Antriebs- und Rolleinfluß wird die Position der Koordinatensysteme B und T gegenüber dem Koordinatensystem V durch die Ortsvektoren $\vec{d}_B$ für den Aufbau und $\vec{d}_{Ti}$, $i \in [0, 3]$, für die Räder definiert. Die Radgeometrie muß dabei, um den Einfluß der Lenk- und Antriebsmomente nachzubilden, für jedes Rad i entsprechend transformiert werden.

Transformationen werden unter OpenGL durch einen Stapel von Matrizen verwaltet, dessen aktuelles Element ξ_i der Transformationsmatrix für Geometriedaten entspricht. Das Hinzufügen einer Matrix ξ zum Stapel entspricht dabei der Matrizenmultiplikation $\xi_i = \xi_{i-1} \cdot \xi$. Die von OpenGL verwendeten 4×4 Matrizen können zum Speichern der Starrkörpertransformation verwendet werden, s. Abschnitt A.9.

Algorithmus 2 zeigt die Transformationen, die von der Aufbaugeometrie B und der Radgeometrie T durchlaufen werden. Der Funktionsparameter E entspricht einer Datenstruktur, die die Aufbau- und Radgeometrie sowie deren statischen Ortsvektoren im Koordinatensystem V enthält. Vor dem Aufruf zum Zeichnen eines Fahrzeuges ist die Betrachtermatrix ξ_{MVP} zu laden, damit die Lage der Kamera berücksichtigt wird.

Beim Zeichnen des Aufbaus und der Räder müssen zwei Fälle unterschieden werden.

Algorithmus 2 Algorithmus zum Zeichnen eines Fahrzeuges.

Require: Die Betrachtermatrix ξ_{MVP} muß geladen sein.

1: **procedure** DrawEntity(E)
2: PushMatrix(ξ_{IV}) ▷ Anfang Fahrzeugsystem

3: PushMatrix($\xi(\mathbf{I}, \vec{p}_B - (0, 0, \varrho_{B,min,z} + \varrho_{T,min,z})^T)$)
4: DrawObject(B) ▷ Aufbau zeichnen
5: PopMatrix( )

6: PushMatrix($\xi(\mathbf{R}_x(-\varphi_{RB}) \cdot \mathbf{R}_y(-\vartheta_{RB}), \vec{0})$) ▷ Anfang Fahrbahnsystem

7: PushMatrix($\xi(\mathbf{R}_z(\delta_{T0}) \cdot \mathbf{R}_y(\varphi_{T0}), \vec{p}_{T0} - (0, 0, \varrho_{T,min,z})^T)$)
8: DrawObject(T) ▷ Rad vorne links zeichnen
9: PopMatrix( )

10: PushMatrix($\xi(\mathbf{R}_z(\pi + \delta_{T1}) \cdot \mathbf{R}_y(-\varphi_{T1}), \vec{p}_{T1} - (0, 0, \varrho_{T,min,z})^T)$)
11: DrawObject(T) ▷ Rad vorne rechts zeichnen
12: PopMatrix( )

13: PushMatrix($\xi(\mathbf{R}_y(\varphi_{T2}), \vec{p}_{T2} - (0, 0, \varrho_{T,min,z})^T)$)
14: DrawObject(T) ▷ Rad hinten links zeichnen
15: PopMatrix( )

16: PushMatrix($\xi(\mathbf{R}_z(\pi) \cdot \mathbf{R}_y(-\varphi_{T3}), \vec{p}_{T3} - (0, 0, \varrho_{T,min,z})^T)$)
17: DrawObject(T) ▷ Rad hinten rechts zeichnen
18: PopMatrix( )

19: PopMatrix( ) ▷ Ende Fahrbahnsystem

20: PopMatrix( ) ▷ Ende Fahrzeugsystem
21: **end procedure**

1. Die Starrkörpertransformation ξ_{IV} des Fahrzeuges wurde aus dem Fahrbahnkoordinatensystem berechnet oder

2. die Starrkörpertransformation ξ_{IV} des Fahrzeuges wurde durch eine Fahrdynamiksimulation berechnet.

Im ersten Fall wird die Transformation ξ_{IV} direkt aus einen Punkt auf der Fahrbahnoberfläche bestimmt. Dieser Ansatz kann z.B. durch eine Verkehrssimulation verwendet werden, deren Hauptaufgabe in der Berechnung von Fahrzeugpositionen relativ zueinander liegt. Der Aufbau und die Räder eines Fahrzeuges können dann im Fahrbahnkoordinatensystem ausgerichtet werden, um die Rollbewegungen der Fahrzeuge zu vernachlässigen.

$$\xi_{IV} \quad = \quad \xi(\,\mathbf{R}_{IR}(D), {}_R\vec{p}_R = (D, O, L = 0)^T\,)$$

Die Position $\vec{p}_R$ des Fahrzeuges auf der Fahrbahn ist dabei durch den zurückgelegten Weg D und die Querablage O festgelegt; die Überhöhung L wurde explizit zu 0 gesetzt. Der Ortsvektor $\vec{p}_R$ entspricht dem Ursprung des Koordinatensystems V.

Durch Gleichung (2.10) ist die Rotationsmatrix $\mathbf{R}_{IR}$ festgelegt. Der Nickwinkel ϑ_{RB} und der Wankwinkel φ_{RB} werden dann zu 0 gesetzt.

Zeile 2 in Algorithmus 2 lädt die Transformation ξ_{IV}. Durch Nullsetzen der Rollbewegung in Zeile 6 folgen Aufbau und Räder dem Fahrbahnkoordinatensystem bzw. der Transformation ξ_{IV}.

Der zweite Fall erfordert die Einrechnung der Rollbewegung des Fahrzeuges gegenüber der Fahrbahn, s. Abschnitt 2.8.

Die durch die Fahrdynamik berechnete Rotation $\mathbf{R}_{IB}$ des Aufbaus gegenüber dem Inertialsystem und dessen Position $\vec{p}_B$ legen die Transformation ξ_{IV} fest.

$$\xi_{IV} \quad = \quad \xi(\,\mathbf{R}_{IB}, \vec{p}\,'_B\,), \qquad \vec{p}\,'_B \equiv {}_R\vec{p}_B|_{L=0}$$

$\vec{p}\,'_B$ entspricht dem auf die Fahrbahnoberfläche projizierten Ortsvektor $\vec{p}_B$ des Aufbaukoordinatensystems B.

Der Ausgleich der Rollbewegung erfolgt durch Berechnung des Nickwinkels ϑ_{RB} und des Wankwinkels φ_{RB} aus der Matrix $\mathbf{R}_{RB}$, s. Abschnitt 2.8, durch Gleichung (2.16).

Zeile 2 des Algorithmus 2 positioniert die Fahrzeuggeometrie durch die Transformation ξ_{IV} absolut im Inertialsystem. In Zeile 6 wird die Rollbewegung ausgeglichen, s. Gleichung (A.10).

In den Zeilen 3 bis 5 wird die Aufbaugeometrie B im ruhenden Koordinatensystem V platziert und gezeichnet, s. Abbildung 3.24. Durch die Organisation der OpenGL-Transformationen in einem Stapel, erfährt die Geometrie B nach der Platzierung im Koordinatensystem V die Transformation ξ_{IV}.

Die Radgeometrie wird für jedes Rad i individuell im Koordinatensystem V platziert und gezeichnet, s. Abbildung 3.24. Dabei ist zu berücksichtigen, daß die Radkappen der rechten Räder nach außen zeigen müssen, was durch eine Rotation mit einem Winkel von 180° um die Hochachse realisiert wird. Die Rotationswinkel φ_T der rechten Räder bedingt durch das Antriebsmoment müssen dann jedoch negativ berücksichtigt werden. Im Gegensatz zu den Hinterrädern können die Vorderräder um den Lenkwinkel δ_T eingeschlagen werden.

Durch die Berücksichtigung der Rollbewegung werden die Räder relativ zum Aufbau entlang der Fahrbahnoberfläche ausgerichtet. Eine Hubbewegung des Fahrzeuges wird durch Algorithmus 2 nicht berücksichtigt.

3.7.2. Benutzerschnittstelle

Damit sich ein Anwender nicht mit den OpenGL-Interna auseinandersetzen muß, wird durch die Visualisierung eine Datenbank bereitgestellt, die eine logische Ansicht auf sog. Visualisierungsobjekte ermöglicht. Einem Anwender wird somit ermöglicht Fahrzeuge und deren Texturen über symbolische Namen zu erzeugen. Weitere Visualisierungsobjekte, z.B. Häuser und Bäume, können ebenfalls über die Visualisierungsdatenbank bereitgestellt werden.

Die Visualisierungsdatenbank ist durch eine XML-Datei realisiert, die eine Liste von Texturen und eine Liste von Fahrzeugen konfiguriert.

Zur Konfiguration der Texturen wird jedem symbolischen Textur-Namen ein vollständiger Dateipfad zugeordnet. Wird eine Textur durch den Anwender verwendet, lädt die Visualisierung die Textur in den Graphikspeicher und liefert dem Anwender eine Referenz zurück. Bei einer erneuten Verwendung wird die Referenz der bereits im Graphikspeicher vorhandenen Textur zurückgeliefert, um die vorhandenen Betriebsmittel optimal auszunutzen.

Fahrzeuge benötigen im Vergleich zu Texturen einen umfangreichen Datensatz zur Konfiguration. Es müssen zumindest die Dateipfade zu den Geo-

metriedaten der Teilobjekte und die Vektoren $\vec{p}_B$ und $\vec{p}_{Ti}$ zu deren Positionierung festgelegt werden. Darüber hinaus können Positionen und Abmaße von Blinkern oder Scheinwerferkegeln definiert werden. Diese Informationen werden in einer Datenstruktur gespeichert, die einem symbolischen Fahrzeugnamen zugeordnet ist, s. Variable E in Algorithmus 2. Ein Anwender lädt zur Visualisierung die Datenstruktur E über ihren symbolischen Namen. Eine erneute Verwendung liefert wieder eine Referenz auf die bereits geladene Struktur zurück. Mit einer Referenz für die Aufbau-Textur und einer weiteren Referenz für die Rad-Textur, ist die Darstellung eines Fahrzeuges vollständig beschrieben.

4. Simulationsumgebung

Nach Abbildung 1.1 bildet die vorgestellte Entwicklungsplattform die Systemumgebung eines Fahrerassistenzsystems zur Fahrzeugumfeldüberwachung durch verschiedene Software-Module eines *HiL*-Simulators nach. Diese Modularität gestattet eine einfache Anpassung an die Anforderungen verschiedenster Assistenzsysteme und ermöglicht darüber hinaus eine Integration in bereits bestehende *HiL*-Konfigurationen zum Funktionstest weiterer Fahrzeugkomponenten.

Die Systemumgebung des Echtteils wird durch das Zusammenspiel dieser Software-Module nachgebildet. Um ein für das Echtteil plausibles Verhalten der Systemumgebung zu gewährleisten, wird eine konsistente Beschreibung des Fahrzeugumfeldes benötigt. Diese Beschreibung wird durch eine Simulationsumgebung realisiert, die in Form einer Datenstruktur verschiedene Parameter der Systemumgebung, z.B. die Anzahl der Verkehrsteilnehmer oder die Positionierung der Sensoren am simulierten Eigenfahrzeug, einem Anwender der Entwicklungsplattform zur Verfügung stellt.

Das Prinzip der Simulationsumgebung wird durch Abbildung 4.1 verdeutlicht.

Auf einem geschlossenen Rundkurs werden die Verkehrsteilnehmer durch die Verkehrs- und die Fahrdynamiksimulation bewegt. In Abhängigkeit des Bewegungszustandes ändern sich die Koordinaten jedes Verkehrsteilnehmers.

Diese Koordinaten werden von allen Software-Modulen benötigt, um ein Fahrzeug absolut zu positionieren. Diese Positionierung wird von der Verkehrssimulation mit den Abmaßen des Fahrzeuges zur Kollisionsabfrage verwendet.

Zur Erfassung der Fremdfahrzeuge durch die nachgebildete Umfeldsensorik des simulierten Eigenfahrzeuges wird ebenfalls die Ausrichtung und die Abmaße der Fremdfahrzeuge benötigt. Weiterhin müssen die Sensormodelle bezüglich der am Eigenfahrzeug nachgebildeten Sensorik parametriert werden.

Letztendlich erfordert eine Visualisierung des simulierten Verkehrsszenarios die Kenntnis der Koordinaten und der Geometrie der Verkehrsteilnehmer sowie der Geometrie des Rundkurses.

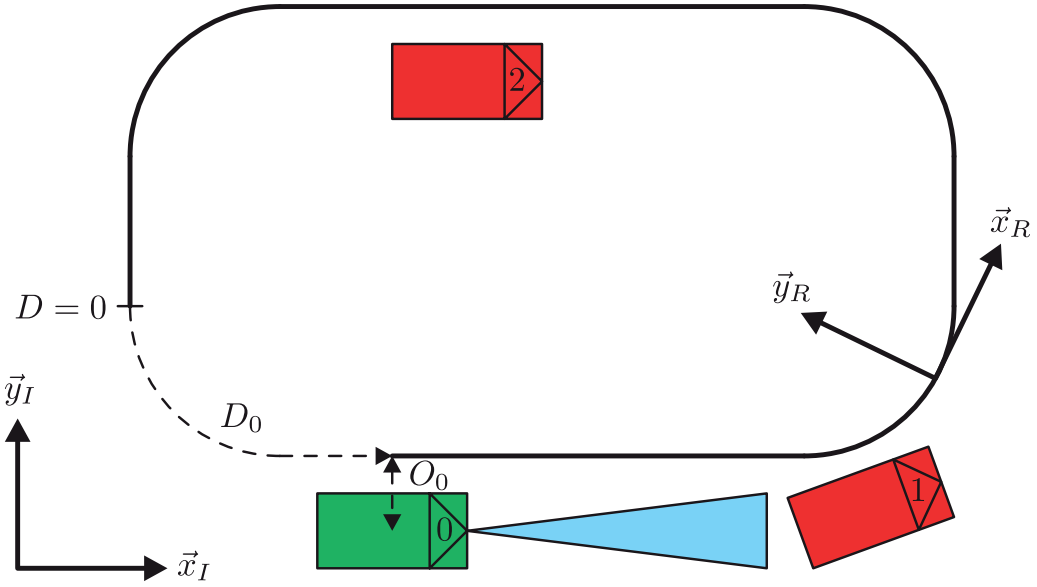

Abbildung 4.1.: Prinzip der Simulationsumgebung.

Die aufgezählten Komponenten werden den Software-Modulen und der Visualisierung durch die Simulationsumgebung in Form einer Konfigurationsdatei zur Verfügung gestellt. Die folgenden Abschnitte beschreiben die Struktur bzw. die Komponenten der Simulationsumgebung, deren Integration in den Echtzeitsimulator sowie die funktionale Kopplung der einzelnen Software-Module durch Vereinbarung eines Koordinatensatzes zur Beschreibung jedes Verkehrsteilnehmers.

[Tellmann 2011] und [Schmidt 2010] verwenden die Simulationsumgebung zur Simulation und Erfassung des Fremdverkehrs durch die Sensormodelle sowie der Erzeugung reproduzierbarer Fahrmanöver definierten Gefahrenpotentials.

4.1. Struktur

Die Simulationsumgebung zur Konfiguration der Entwicklungsplattform wird einem Anwender in Form eines Baumes präsentiert, s. Abbildung 4.2, dessen Blätter und Teilbäume die verschiedenen Komponenten der nachgebildeten Systemumgebung des Echtteils parametrieren.

Die Anordnung der Parameter des Fahrzeugumfeldes in einer Baumstruktur ermöglicht es den Software-Modulen und der Visualisierung während

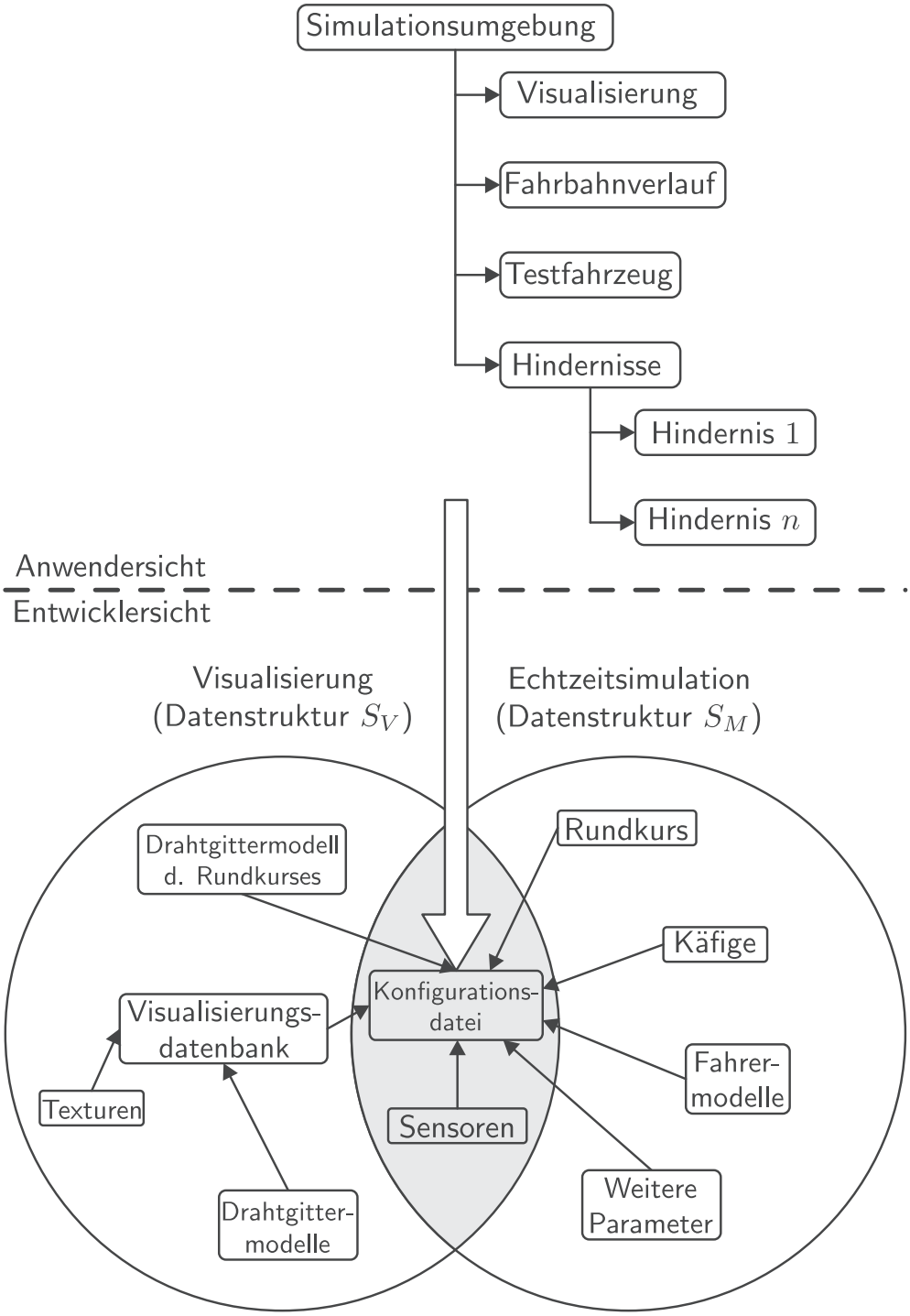

Abbildung 4.2.: Die Komponenten der Simulationsumgebung präsentieren sich einem Anwender in Form eines Baumes (oben). Eine Konfigurationsdatei speichert die Parameter der Simulationsumgebung und referenziert benötigte Dateien (unten).

der Initialisierungsphase nur die Informationen zu verarbeiten, die für einen reibungslosen Betrieb erforderlich sind. Der Baum aus Abbildung 4.2 wird durch Verarbeitung einer *XML*-Datei, der sog. Konfigurationsdatei, auf eine Datenstruktur der entsprechenden Anwendung, der Echtzeitsimulation oder der Visualisierung, abgebildet.

Um eine möglichst große Wiederverwendbarkeit einzelner Punkte eines konfigurierten Testablaufes zu gewährleisten, werden einzelne Komponenten der Simulationsumgebung in separaten Dateien konfiguriert, die dann durch die Konfigurationsdatei referenziert werden, s. Abbildung 4.2. Die Pfeile sind jeweils auf die referenzierende Datei gerichtet. I.d.R. ist das Verhältnis der verschiedenen Dateien zur Konfigurationsdatei nur für einen Testentwickler von Interesse. Eine Modifikation des Testablaufes kann bereits in der Anwendersicht durch einen Prüfingenieur erfolgen.

Zur Echtzeitsimulation des Fahrzeugumfeldes werden keine Informationen über die visuellen Eigenschaften der Fahrzeuge benötigt. Die aus der Konfigurationsdatei erzeugte Datenstruktur S_M, die in den Software-Modulen des Echtzeitsimulators eingesetzt wird, enthält dementsprechend keine hochauflösenden Geometriedaten und Texturen zur Darstellung der Fahrzeuge, sondern ein reduziertes Drahtgittermodell mit einer definierbaren Sende- und Empfangscharakteristik, wie sie zur Verarbeitung durch die Sensormodelle benötigt wird.

Analog dazu erzeugt die Visualisierung aus der Konfigurationsdatei eine Datenstruktur S_V, die nur die zur Darstellung der Fahrzeuge und der Fahrbahn benötigten Informationen enthält.

Abbildung 4.3 illustriert den Unterschied eines Drahtgittermodells, wie es zur Visualisierung eines Fahrzeuges verwendet wird, und eines sog. Käfigs, der die Geometrie eines Fahrzeuges zur Verarbeitung durch die Sensormodelle approximiert.

Die Parameter des Fahrzeugumfeldes werden dabei *direkt* oder *indirekt* in der Konfigurationsdatei angegeben. Eine direkte Angabe eines Parameters erfolgt dabei in der Konfigurationsdatei selbst, z.B. die Anzahl der Fremdfahrzeuge. Indirekte Parameter verweisen auf eine weitere Datei, die zur Erzeugung der Datenstrukturen S_M oder S_V verarbeitet werden muß.

Im folgenden sollen die verschiedenen Komponenten der Simulationsumgebung und ihre Bedeutung für die Entwicklungsplattform erläutert werden.

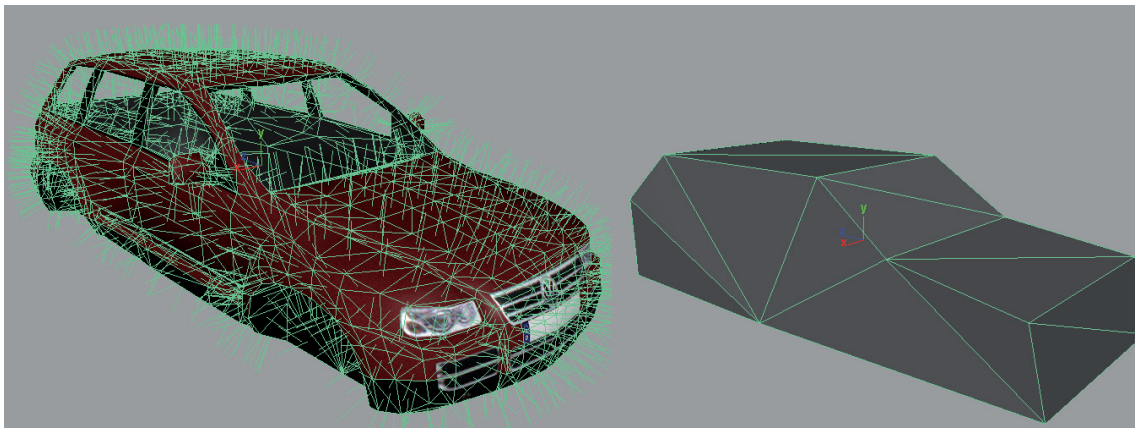

Abbildung 4.3.: Vergleich eines Drahtgittermodells zur Visualisierung (links, ca. 2000 Dreiecke) mit einem Käfig zur Verarbeitung durch die Sensormodelle (rechts, ca. 30 Dreiecke).

4.1.1. Visualisierung

Die Konfiguration der Visualisierung erfolgt nur durch Angabe der Visualisierungsdatenbank, s. Abschnitt 3.7, die wiederum die bereitgestellten Texturen und Fahrzeuge zur Benutzung durch den Anwender bereithält.

Eine Erweiterung der Visualisierungsdatenbank ist i.d.R. nur bei Aufnahme eines neuen Fahrzeuges oder einer neuen Textur, z.B. zur Änderung des Fahrbahnbelages, nötig. Für viele Anwendungen kann eine bereits bestehende Datenbank genutzt werden.

4.1.2. Fahrbahnverlauf

Die Software-Module des Echtzeitrechners benötigen zur Berechnung der Trajektorien der Verkehrsteilnehmer die analytische Fahrbahnbeschreibung aus Kapitel 2. Dieser Rundkurs wird durch Referenzierung einer weiteren Datei in der Konfigurationsdatei eingebunden und den Modulen in der Datenstruktur S_M zur Verfügung gestellt.

Zur Visualisierung des Fahrbahnverlaufes wird hingegen der Damm aus Abschnitt 3.5 benötigt, der vor Benutzung der Simulationsumgebung berechnet werden kann und dann durch eine Dateireferenz der Konfigurationsdatei in die Datenstruktur S_V geladen wird. Die Textur der Fahrbahnoberfläche wird als symbolischer Name der Visualisierungsdatenbank direkt in der Konfigurationsdatei angegeben.

In Abhängigkeit der Anwendung, Visualisierung oder Echtzeitsimulation, werden die entsprechenden Blätter des Teilbaums *Rundkurs*, s. Abbildung 4.2, der Simulationsumgebung verarbeitet bzw. ignoriert.

4.1.3. Testfahrzeug

Die visuellen Eigenschaften des Eigenfahrzeuges, das hier als Testfahrzeug bezeichnet wird, werden in der Konfigurationsdatei direkt durch die symbolischen Namen der Visualisierungsdatenbank angegeben.

Zur Erfassung des simulierten Fremdverkehrs benötigen die Sensormodelle Informationen über die Eigenschaften der nachzubildenden Sensoren. Diese Informationen werden durch eine weitere Datei beschrieben, die durch die Konfigurationsdatei referenziert wird. Die geometrischen Eigenschaften dieser Sensoren werden zudem von der Visualisierung zur Darstellung der Sensorbereiche verwendet.

Die Sensorkonfiguration stellt den einzigen Parameter der Simulationsumgebung dar, der sowohl für die Visualisierung als auch für die Echtzeitsimulation verwendet wird.

Weitere Parameter des Testfahrzeuges, z.B. der Käfig, die Parametrierung eines Fahrermodells oder Eigenschaften des zu testenden Assistenzsystems, werden durch Dateireferenzen in der Konfigurationsdatei eingebunden. Der Käfig des Testfahrzeuges dient dabei zur Kollisionsabfrage durch die Fahrermodelle der Verkehrssimulation.

Die Starteigenschaften des Testfahrzeuges, z.B. die initiale Fahrspur oder Sollgeschwindigkeit, werden direkt in der Konfigurationsdatei der Simulationsumgebung angegeben.

4.1.4. Hindernisse

Zur Konfiguration der Verkehrssimulation dient der Teilbaum *Hindernisse* der Simulationsumgebung. Die einzelnen Verkehrsteilnehmer, die hier als Hindernis bezeichnet werden, bilden wiederum Unterbäume des Teilbaumes *Hindernisse*, s. Abbildung 4.2. Während der Verarbeitung der Konfigurationsdatei wird das Auftreten der Teilbäume *Hindernis* gezählt, um die durch den Anwender konfigurierte Anzahl an Hindernissen zu ermitteln.

Jedes Hindernis verfügt dabei über die folgenden Parameter.

Ähnlich dem Testfahrzeug werden symbolische Namen der Visualisierungsdatenbank benötigt, um die visuellen Eigenschaften eines Hindernisses für die Visualisierung zu definieren.

Die Parametrierung eines Fahrers und der Käfig des Hindernisses werden als Referenzen auf weitere Dateien in der Konfigurationsdatei gespeichert. Der Käfig dient hier sowohl zur Kollisionsabfrage durch die Verkehrssimulation als auch zur Erfassung durch die Sensormodelle.

In der Konfigurationsdatei werden für jedes Hindernis direkt die gewünschten Starteigenschaften, ähnlich des Testfahrzeuges, angegeben.

Darüber hinaus kann für jedes Hindernis eine Liste von Fahrmanövern angegeben werden, die zur Stimulation des zu testenden Assistenzsystems verwendet werden. Eines dieser Fahrmanöver wird dem Hindernis durch den Situationsgenerator der Verkehrssimulation aufgeprägt, um ein gewünschtes Gefahrenpotential für das Testfahrzeug zu erzeugen.

4.2. Implementierung

4.2.1. Beschreibung der Systemarchitektur

Die Simulationsumgebung wurde für einen Echtzeitsimulator der Firma carts realisiert, s. [Woermann 1994]. Abbildung 4.4 zeigt die Struktur der zugrunde liegenden Systemarchitektur.

Ein *VME*[1]-Rechner, der sog. *Echtzeitrechner*, bildet die Basis des Echtzeitsimulators. Ein auf dem Echtzeitrechner ausgeführtes Echtzeitbetriebssystem steuert den Ablauf der Echtzeitsimulation, verwaltet die Simulationsergebnisse und verarbeitet die Ein- und Ausgangssignale der angeschlossenen Echtteile[2].

Zentrale Komponente des Echtzeitrechners ist die *Datenbasis*. Alle Signale des Echtzeitsimulators, die durch physikalische Ein- und Ausgänge sowie der funktionalen Kopplung der *Simulationsprogramme* entstehen, werden in der Datenbasis auf *Parameter* abgebildet. Jeder Parameter entspricht dabei einer eindeutig benannten Variablen, der zur Darstellung ein Datentyp, z.B. Festkomma- oder Gleitkommazahl, zugeordnet wird.

Die Echtzeitsimulation, die die Systemumgebung eines Echtteils nachbildet, besteht aus einem oder mehreren Simulationsprogrammen. Ein Simulationsprogramm entspricht einem Prozess des Echtzeitbetriebssystems und nimmt die Zustände nach Abbildung 4.5 an.

[1] *VERSAmodule Eurocard*
[2] Anm.: Die Anbindung von Echtteilen wird in Abbildung 4.4 nicht dargestellt.

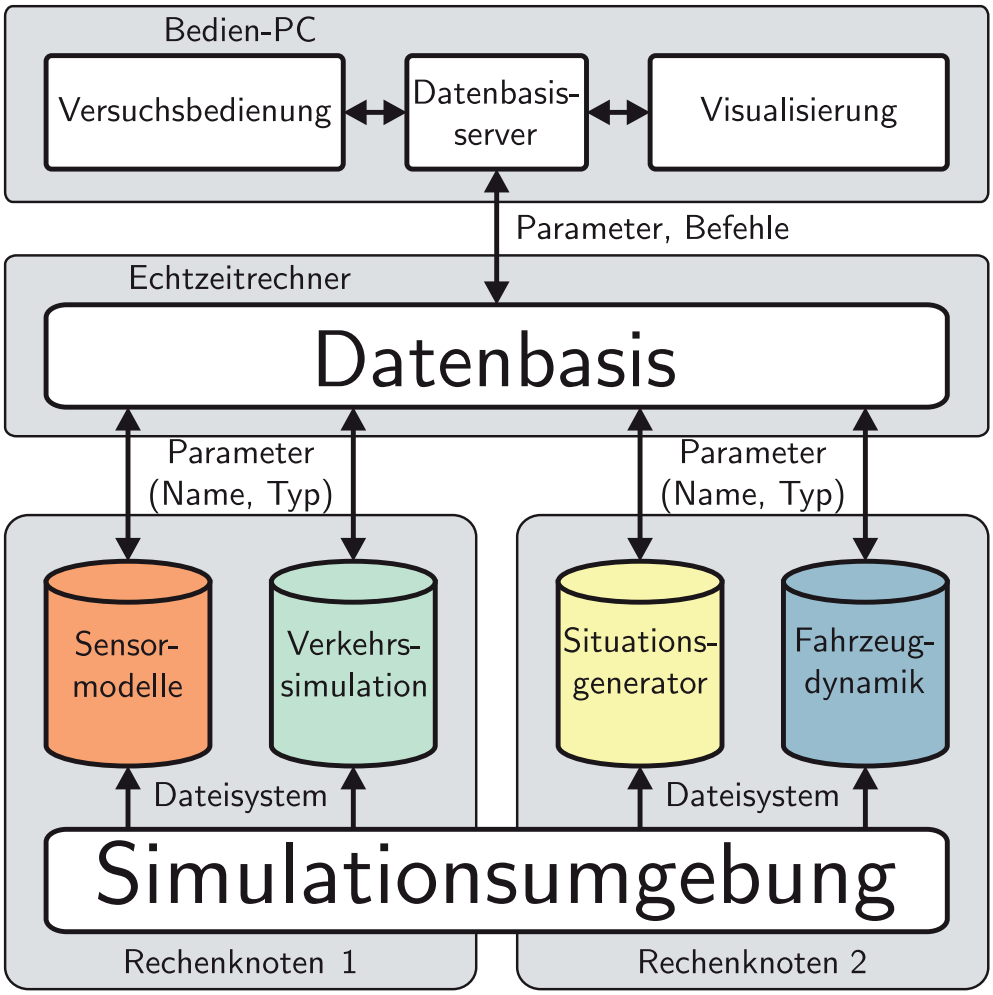

Abbildung 4.4.: Struktur der verteilten Systemarchitektur des Echtzeitsimulators.

Jeder Zustand wird durch Implementierung einer Rückruffunktion[3] der Simulationsprogramme realisiert. Die Zustandswechsel werden bei einem manuellen Versuch durch Aktionen des Benutzers eingeleitet.

Nach dem Laden der Datenbasis, das aus der Initialisierung der Parameter, der Konfiguration der Ein-/Ausgabe-Schnittstellen und der Übertragung der Simulationsprogramme auf den Echtzeitrechner besteht, werden die Simulationsprogramme geladen und der Zustand *Initialisierung* durchlaufen.

[3]engl.: *callback function*

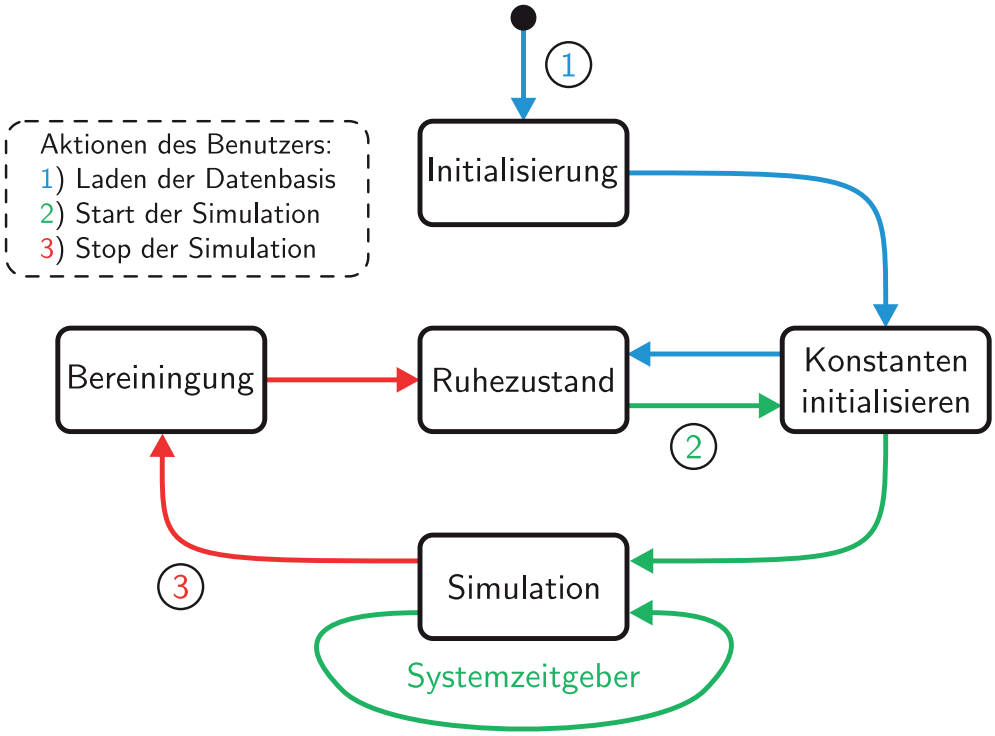

Abbildung 4.5.: Zustände eines Simulationsprogramms (vereinfacht).

Der Initialisierungszustand dient dem Simulationsprogramm dazu Parameter der Datenbasis als Ein- bzw. Ausgänge zu definieren. Mit jedem Eintritt des Simulationsprogramms in einen der weiteren Zustände werden die während der Initialisierung bei der Datenbasis angemeldeten Eingangsparameter auf Variablen des Simulationsprogramms abgebildet. Bei Verlassen eines Zustandes werden die Ausgangsparameter in die Datenbasis übernommen.

Zur Initialisierung von Konstanten, die von Parametern der Datenbasis abhängig sein können, wird der Zustand *Konstanten initialisieren* beim Wechsel vom Initialisierungs- in den Ruhezustand und beim Übergang vom Ruhezustand in die Simulation aufgerufen.

Der *Ruhezustand* wird benötigt, um die Echtteile mit den der Systemumgebung entsprechenden Signalen zu versorgen ohne bestimmte Betriebsszenarien nachzubilden. Bei Ausbleiben konsistenter Eingangssignale wird ein Echtteil u.U. in einen Fehlerzustand wechseln und den weiteren Betrieb verweigern.

Beim Start der Simulation durch einen Anwender wird in den Zustand *Simulation* gewechselt. Der Systemzeitgeber sorgt durch zyklische Aufrufe

der Simulationsprogramme für eine fortlaufende Berechnung der Simulationsergebnisse.

Die Simulation wird durch den Anwender beendet. Vor dem Wechsel in den Ruhezustand wird eine *Bereinigung* durchgeführt, die von einem Simulationsprogramm z.B. zur Freigabe von angefordertem Speicher genutzt werden kann.

Zur Erweiterung der Rechenleistung bietet der Echtzeitsimulator die Möglichkeit *Rechenknoten* durch Vernetzung in den Simulator zu integrieren, s. Abbildung 4.4 und [Meyer 2006]. Diese Rechenknoten führen dabei nur Simulationsprogramme aus. Die Datenbasis des Echtzeitrechners wird dazu auf allen Rechenknoten synchronisiert, um eine globale Verfügbarkeit aller Signale des Echtzeitsimulators zu gewährleisten. Eine Kopplung der Rechenknoten findet dabei direkt über den *VME*-Bus oder *Gigabit Ethernet* statt.

Die Steuerung des Echtzeitsimulators erfolgt durch den *Datenbasisserver*, der auf dem *Bedien*-PC ausgeführt wird. Eine Anbindung des Echtzeitrechners erfolgt dabei über die Netzwerkschnittstelle. Der Datenbasisserver steuert den Echtzeitsimulator und führt eine zyklische Spiegelung der Datenbasis des Echtzeitrechners auf dem Bedien-PC aus. Da der Datenbasisserver über eine Schnittstelle zur Interprozeßkommunikation verfügt, kann der Echtzeitrechner von verschiedenen Programmen gesteuert werden. Ein Benutzer verwendet dazu meist die *Versuchsbedienung* des Echtzeitsimulators, die zum Laden der Datenbasis, zur Vorgabe von Parametern und Auswertung der Simulationsergebnisse verwendet wird.

4.2.2. Schnittstellen der Simulationsprogramme

Eine funktionale Kopplung der Simulationsprogramme kann über die Parameter der Datenbasis erfolgen, die während der Echtzeitsimulation auf allen Rechenknoten des Simulators verfügbar sind. Da sich die geometrischen Parameter der Verkehrsteilnehmer i.d.R. nicht ändern, ist es ausreichend die 6 Koordinaten zur Positionierung eines Starrkörpers, s. Abschnitt A.9, für jeden Verkehrsteilnehmer durch die Datenbasis zur Kopplung der Simulationsprogramme zu verwenden.

Die Starrkörpertransformation ξ_{IV}, die beliebige Vektoren $\vec{\varrho}$ vom Fahrzeugsystem V in das Inertialsystem I transformiert, weist folgende Gestalt auf.

Parametername	Beschreibung	Datentyp, Einheit
`prefix_X`	x-Koordinate	`double`, $[m]$
`prefix_Y`	y-Koordinate	`double`, $[m]$
`prefix_Z`	z-Koordinate	`double`, $[m]$
`prefix_ROTX`	Rotationswinkel φ_{IV} um die x-Achse	`double`, $[rad]$
`prefix_ROTY`	Rotationswinkel ϑ_{IV} um die y-Achse	`double`, $[rad]$
`prefix_ROTZ`	Rotationswinkel ψ_{IV} um die z-Achse	`double`, $[rad]$
`prefix_D`	zurückgelegte Bogenlänge D (optional)	`double`, $[m]$
`prefix_O`	Querablage O (optional)	`double`, $[m]$
`prefix_L`	Höhe L (optional)	`double`, $[m]$
`prefix_ROLLX`	Wankwinkel φ_{RB} relativ zur Fahrbahnoberfläche (optional)	`double`, $[rad]$
`prefix_ROLLY`	Nickwinkel ϑ_{RB} relativ zur Fahrbahnoberfläche (optional)	`double`, $[rad]$

Tabelle 4.1.: Koordinatensatz zur Beschreibung eines Verkehrsteilnehmers.

$$\vec{\varrho} = \xi_{IV}(\mathbf{R}_{IV}, \vec{p}_V) \cdot {}_V\vec{\varrho}$$

Auf die Berechnung der Rotationsmatrix $\mathbf{R}_{IV}$ und des Ortsvektors $\vec{p}_V$ wurde bereits in den Abschnitten 2.8 und 3.7.1 eingegangen. Können die Komponenten x, y und z des Ortsvektors $\vec{p}_V$ direkt als Parameter der Datenbasis verwendet werden, ist die Rotationsmatrix $\mathbf{R}_{IV}$ mit der Konvention

$$\mathbf{R}_{IV} = \mathbf{R}_z(\psi_{IV}) \cdot \mathbf{R}_y(\vartheta_{IV}) \cdot \mathbf{R}_x(\varphi_{IV})$$

durch die Parameter φ_{IV}, ϑ_{IV} und ψ_{IV} definiert, was einer Anwendung von Gleichung (A.10) entspricht.

Tabelle 4.1 zeigt die Namenskonvention zur Darstellung der Parameter in der Datenbasis.

Für jeden Verkehrsteilnehmer sind somit 6 Parameter in der Datenbasis anzulegen, die dessen Koordinaten im Inertialsystem beschreiben. Der

Präfix `prefix` der Variablen in Tabelle 4.1 wird dabei für das Testfahrzeug zu `VUT`[4] und für die Hindernisse zu `OBS_`i[5] gesetzt. Die Laufvariable $i = 1, \ldots, N$ kennzeichnet eindeutig jedes Hindernis.

In der Datenbasis sind somit zur Anwendung der Simulationsumgebung für jeden Verkehrsteilnehmer 6 Parameter anzulegen. Um eine redundante Berechnung verschiedener Koordinatendarstellungen, z.B. der Fahrbahnkoordinaten, s. Abschnitt 2.7.1, zu vermeiden, können für jeden Verkehrsteilnehmer weitere Parameter eingeführt werden, deren Benutzung durch die Simulationsprogramme und die Visualisierung optional ist.

4.2.3. Integration der Simulationsumgebung

Eine Nachbildung des Fahrzeugumfeldes erfolgt durch die Simulationsprogramme *Sensormodelle*, *Verkehrssimulation*, *Situationsgenerator* und *Fahrzeugdynamik*, s. Abbildung 4.4. Der Situationsgenerator dient zur Erzeugung relevanter Verkehrsszenarien zum Test des Assistenzsystems und wurde aus der Verkehrssimulation herausgelöst. Zur optimalen Ausnutzung der verfügbaren Betriebsmittel können die Simulationsprogramme auf die vorhandenen Rechenknoten des Echtzeitsimulators verteilt werden.

Der Einsatz der Simulationsumgebung durch die Simulationsprogramme wird durch eine *C++*-Bibliothek ermöglicht, die die Konfigurationsdatei verarbeitet und die zur Echtzeitsimulation erforderlichen Dateien auf die Datenstruktur S_M abbildet. Diese Bibliothek wird zum Übersetzungszeitpunkt an jedes Simulationsprogramm statisch gebunden und steht somit zur Laufzeit zur Verfügung.

Damit die Simulationsumgebung von den Simulationsprogrammen geladen werden kann, sind die zur Echtzeitsimulation erforderlichen Dateien an die entsprechenden Rechenknoten des Echtzeitsimulators zu übertragen. Die Verarbeitung der Dateien erfolgt durch die Simulationsprogramme unabhängig voneinander auf dem jeweiligen lokalen Rechenknoten.

Mit Eintritt eines Simulationsprogramms in die Initialisierung beginnt die Verarbeitung der Simulationsumgebung. Jedes Simulationsprogramm erzeugt zunächst aus der lokal vorhandenen Konfigurationsdatei eine Instanz der Datenstruktur S_M, die die Struktur aus Abbildung 4.2 aufweist. Mit der jetzt bekannten Anzahl der Verkehrsteilnehmer kann die funktionale

[4] engl.: *vehicle-under-test*
[5] engl.: *obstacle*

Kopplung der Simulationsprogramme untereinander vorgenommen werden. Dazu wird, in Abhängigkeit der Funktion des Simulationsprogramms, der Koordinatensatz aus Abschnitt 4.2.2 für jeden Verkehrsteilnehmer mit der Datenbasis als Ein- bzw. als Ausgang vereinbart.

Da die Kopplung eines Simulationsprogramms mit der Datenbasis nur im Initialisierungszustand des Echtzeitsimulators vorgenommen werden kann und die Anzahl der benötigten Koordinatensätze erst nach Verarbeitung der Simulationsumgebung bekannt ist, muß die Erzeugung der Datenstruktur S_M vor der Kopplung des Simulationsprogramms an die Datenbasis ebenfalls im Initialisierungszustand erfolgen. In der Datenbasis sind im Voraus durch den Anwender die benötigte Anzahl an Parametern zu erzeugen, da sonst keine Kopplung der Simulationsprogramme untereinander möglich ist.

Durch Verteilung der Simulationsumgebung auf alle Rechenknoten des Echtzeitsimulators, die vor einem Laden der Datenbasis des Echtzeitrechners zu erfolgen hat, stehen die statischen Parameter des Fahrzeugumfeldes jedem Simulationsprogramm zur Verfügung. Die Koordinaten der Verkehrsteilnehmer hingegen werden den Simulationsprogrammen als dynamische Parameter durch die Datenbasis zur Verfügung gestellt.

Von den Simulationsprogrammen werden auf den Rechenknoten des Echtzeitsimulators nur die Bestandteile der Simulationsumgebung verarbeitet, die zur Echtzeitsimulation des Fahrzeugumfeldes erforderlich sind, s. Abbildung 4.2. Die *Visualisierung* ist hingegen ein eigenständiges Programm, das auf dem Bedien-PC des Echtzeitsimulators ausgeführt wird, s. Abbildung 4.4, und über den Datenbasisserver an den Echtzeitsimulator gekoppelt ist.

Zur Darstellung des Fahrzeugumfeldes wird von der Visualisierung die Konfigurationsdatei der Simulationsumgebung auf dem Bedien-PC verarbeitet und die Datenstruktur S_V erzeugt. Nachdem die Datenbasis auf den Echtzeitrechner geladen und die Echtzeitsimulation durch den Anwender gestartet wurde, ist die Visualisierung mit der gespiegelten Datenbasis des Datenbasisservers zu verbinden. Die in Abschnitt 4.2.2 vereinbarten Koordinatensätze werden von der Visualisierung zyklisch aus der gespiegelten Datenbasis aktualisiert und direkt zur Positionierung der Verkehrsteilnehmer verwendet.

5. Anwendung

Eine erste Anwendung, deren Ergebnisse in diesem Kapitel präsentiert werden, haben die vorgestellte Visualisierung und Simulationsumgebung im EU-Forschungsprojekt *DECOS* erfahren.

5.1. Zielsetzung des DECOS-Projektes

Die Funktionsvielfalt moderner Verkehrsmittel und industrieller Anlagen wird in zunehmendem Maße durch den Einsatz elektronischer Steuergeräte realisiert, denen eine sog. föderierte Systemarchitektur zugrunde liegt.

Mit einer föderierten Systemarchitektur wird das Gesamtsystem durch mehrere verteilte Anwendungen realisiert. Jede dieser Anwendungen ist auf genau einem Steuergerät implementiert. Eine funktionale Kopplung der einzelnen Anwendungen erfolgt z.B. mit dem *CAN*-Bus. Die föderierte Systemarchitektur bietet u.a. Vorteile bei der Fehlereindämmung und der Flexibilität des Gesamtsystems.

Ein fehlerhaftes Steuergerät betrifft zunächst nur die Funktion der auf ihm ausgeführten Anwendung. Der Fehler kann somit lokal begrenzt werden, da die Schnittstelle zu den anderen Anwendungen bzw. Steuergeräten, also die Kommunikation, i.d.R. durch jede Anwendung separat diagnostiziert wird.

Durch die Zuordnung einer Anwendung pro Steuergerät entstehen enorme Vorteile bei der Konfiguration des Gesamtsystems, dessen Funktionsvielfalt durch die Präsenz der einzelnen Steuergeräte definiert ist. Verschiedene Varianten des Gesamtsystems lassen sich somit durch hinzufügen und entfernen von Steuergeräten bzw. der Anwendungen realisieren. Eine Bindung des Gesamtsystems an den Hersteller eines Steuergerätes einer bestimmten Anwendung ist ebenfalls nicht gegeben, da für eine bestimmte Anwendung nur die Schnittstelle zum Gesamtsystem angepasst werden muß.

Nachteilig wirkt sich bei der föderierten Systemarchitektur der heterogene Entwicklungsprozess der verschiedenen Hersteller von Steuergeräten aus. So wird der Hersteller des Gesamtsystems durch dessen zunehmende Kom-

plexität und Funktionsvielfalt ständig vor neue Herausforderungen bei der Systemintegration und dem Test der einzelnen Steuergeräte gestellt.

Um dem Trend der zunehmenden Komplexität der Entwicklung, Inbetriebnahme und dem Test von vernetzten, eingebetteten Systemen zu begegnen, wurde von der Europäischen Union das *DECOS*[1]-Projekt initiiert, das innerhalb des 6. Rahmenprogramms gefördert wurde, s. [Kopetz u. a. 2004].

Ziel des *DECOS*-Projektes ist es einen ganzheitlichen Ansatz bei der Entwicklung, Implementierung und Verteilung eingebetteter, verteilter und sicherheitsrelevanter Systeme zur Verfügung zu stellen. Dazu wurde die sog. *DECOS*-Technologie entwickelt, die den Anforderungen solcher Systeme gerecht wird. Diese Technologie wird auch als integrative Systemarchitektur bezeichnet, da bereits während der Entwicklung und während des Tests einer Anwendung deren Integration in das Gesamtsystem berücksichtigt wird.

Eine Anwendung, die im *DECOS*-Kontext als Software-Komponente zu verstehen ist, wird nicht durch ein diskretes Steuergerät realisiert, sondern kann auf einem beliebigen Steuergerät in das Gesamtsystem integriert werden. Dadurch wird eine bessere Ausnutzung der Betriebsmittel, z.B. des Prozessors oder des Hauptspeichers eines Steuergerätes, der physikalischen Vernetzung der Steuergeräte untereinander und der Verwendung vorhandener Sensorik bzw. Aktorik angestrebt. Sicherheitsrelevante Anwendungen können durch eine Verteilung auf mehrere Steuergeräte ebenfalls redundant ausgeführt werden.

Am Beispiel der Vernetzung eines modernen Kraftfahrzeuges soll die föderierte der integrativen Systemarchitektur gegenübergestellt werden, s. Abbildung 5.1.

Bei der momentan in Kraftfahrzeugen eingesetzten Systemarchitektur wird jedem Steuergerät genau eine Anwendung zugeordnet. In Abhängigkeit der Anforderungen an die einzelnen Anwendungen wird die Vernetzung in mehrere Teilnetze aufgespalten, um die vorhandenen Betriebsmittel der eingesetzten Bussysteme optimal einsetzen zu können. Ein sog. *Gateway*-Steuergerät dient dann zur Vermittlung zwischen den einzelnen Netzwerken.

Eine integrative Systemarchitektur, die mit der *DECOS*-Technologie realisiert ist, kann die verschiedenen Anwendungen auf beliebige Steuergeräte verteilen. Anwendungen, die auf demselben Steuergerät untergebracht sind, sind durch eine Laufzeitumgebung, dem Betriebssystem, gegeneinander gekapselt. Die Kommunikation erfolgt untereinander z.B. durch geteilte

[1] *Dependable Embedded Components and Systems*

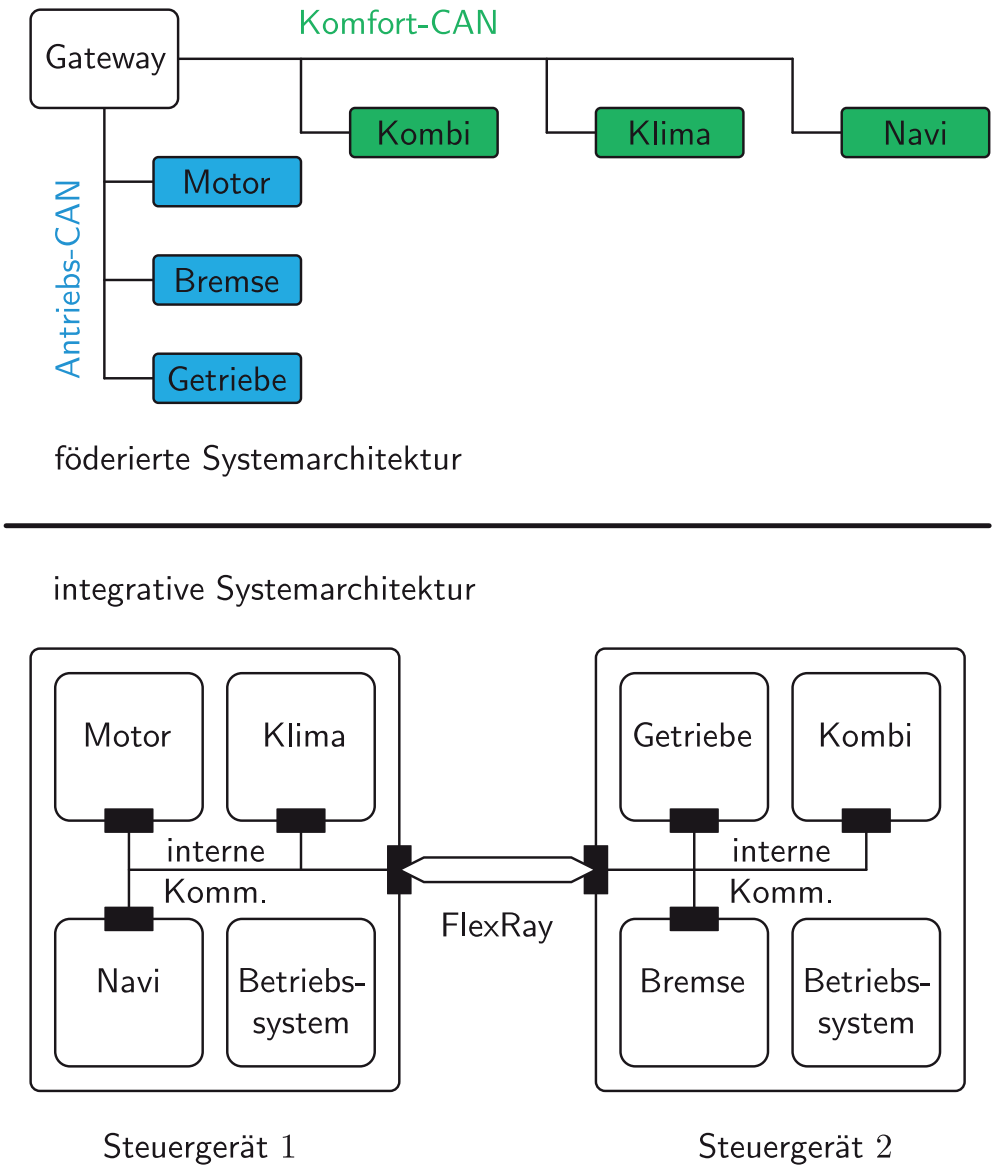

Abbildung 5.1.: Gegenüberstellung der föderierten (oben) und der integrativen (unten) Systemarchitektur am Beispiel der Vernetzung von Anwendungen eines Kraftfahrzeuges. Die Anbindung von Steuergeräten an das Bordnetz und an die Sensorik bzw. Aktorik des Fahrzeuges wird dabei nicht berücksichtigt.

Speicherbereiche[2]. Der Austausch von Informationen über die Grenzen eines Steuergerätes hinaus erfolgt hier durch *FlexRay*, was eine Zeitsteuerung der einzelnen Anwendungen ermöglicht.

Im folgenden sollen die Hauptthemen der *DECOS*-Technologie erläutert werden.

Systemarchitektur Die Grundlage der mit den *DECOS*-Methoden und -Werkzeugen entwickelten Anwendungen bildet eine gekapselte, zeitgesteuerte Systemarchitektur.

Gekapselt bedeutet dabei das die Anwendungen eines Steuergerätes räumlich voneinander getrennt sind und auch im Fehlerfall keine Beeinflussung zweier unabhängiger, lokaler Anwendungen auftreten kann. Eine Möglichkeit dazu bieten z.B. die bekannten Speicherschutzmaßnahmen. Eine Beeinflussung von Anwendungen anderer Steuergeräte, z.B. durch die Kopplung über Kommunikationskanäle, bleibt ungeachtet dieser Maßnahmen möglich.

Unter einer Zeitsteuerung versteht man das deterministische Aufrufen der einzelnen Anwendungen, die auf verschiedene Steuergeräte verteilt sind. Diese Aufrufe geschehen durch einen zeitgesteuerten Bus, der mit einer definierten Zykluszeit die Anwendung aufruft und den benötigten Datentransfer abwickelt.

Innerhalb des *DECOS*-Projektes wird eine Kombination aus dem Infineon *TriCore* Prozessor als System-*CPU* und den zeitgesteuerten Bussen *FlexRay* und *TTP*[3] eingesetzt.

Werkzeuge Um den Entwicklungsprozeß ganzheitlich, d.h. von der Anforderungsaufnahme bis zur Verteilung auf reale Steuergeräte zu unterstützen, wird eine Werkzeugkette entwickelt, die bereits beim Entwurf einer Anwendung deren Integration in die *DECOS*-Technologie gestattet.

Diese Integration beschränkt sich dabei nicht nur auf die Definition von Wortbreiten, Datentypen oder physikalischen Dimensionen der erforderlichen Ein- und Ausgangssignale einer Anwendung, sondern legt bereits bei der Entwicklung Anforderungen an die benötigten Betriebsmittel fest.

[2] engl.: *shared memory*
[3] *Time Triggered Protocol*

Die heute üblicherweise automatische Codeerzeugung von Funktionen für elektronische Steuergeräte kann durch die *DECOS*-Werkzeuge mit den Informationen zur Systemintegration gekoppelt werden und führt so zu einer vereinfachten Verteilung einer Anwendung auf beliebige Steuergeräte.

Methodik Durch eine einheitliche Entwicklungsmethodik wird der gesamte Entwicklungsprozeß gesteuert. Sie beschreibt die Anwendung der Schnittstellen zur *DECOS*-Technologie, Richtlinien zum Test und der Validierung von Anwendungen und spezifiziert die Anbindung von *DECOS*-fremden Technologien, z.B. dem *CAN*-Bus.

Ein üblicher Arbeitsvorgang beginnt mit der Entwicklung einer Anwendung, z.B. in *SimuLink*. Danach folgt der Test und die Validierung mit *SCADE*[4]. Innerhalb von *SCADE* werden auch die Schnittstellen zur Systemintegration definiert, womit im nächsten Schritt die Verteilung auf ein Steuergerät geschehen kann. Letztendlich wird ein Skript zum Übersetzen und Binden des Codes erstellt, der dann auf dem Steuergerät ausgeführt werden kann.

Um die Funktion und die Möglichkeiten der *DECOS*-Technologie aufzuzeigen, wurden drei Anwendungen definiert, die den Einsatz der entwickelten Technologie bei der Funktionsentwicklung und -implementierung, dem Test und der Integration in das Gesamtsystem demonstrieren sollen. Im Einzelnen wurden Anwendungen der Automobilindustrie, der Luft- und Raumfahrttechnik und einer industriellen Regelung realisiert.

Als Anwendung in der Luft- und Raumfahrttechnik und der industriellen Regelung wurde die Regelung einer Landeklappe und die Positionierung eines Silicium-Wafers realisiert. Beide Anwendungen stellen verteilte Regelungen dar, die hohe Anforderungen an die Echtzeitfähigkeit der Systemarchitektur und die Sicherheitsrelevanz für Benutzer und Material stellen.

Das *DECOS*-Projekt wurde in Kooperation mehrerer Hochschulen und Industrieunternehmen bearbeitet. Die Universität Kassel beteiligte sich am Gesamtprojekt mit dem Aufbau eines Echtzeitsimulators zur Nachbildung der Systemumgebung von Fahrerassistenzsystemen zur Fahrzeugumfeldüberwachung. Mithilfe der *DECOS*-Technologie sollen die Funktionen solcher Assistenzsysteme implementiert, validiert und in einem Gesamtsystem zur Demonstration einer Anwendung der Automobilindustrie integriert werden.

[4] *Safety Critical Application Development Environment*

5.2. Struktur des Demonstrators

Zur Demonstration der *DECOS*-Technologie wurden drei Assistenzsystemfunktionen verwendet, die den Fahrer aktiv bei der Fahrzeugführung unterstützen sollen. Dazu wird die Bewegung des Fahrzeuges auf der Fahrbahn und im Verkehrsfluß mit Sensoren beobachtet und ggf. ein Regeleingriff durch das Assistenzsystem vorgenommen.

Als Assistenzsysteme wurden die Funktionen einer Distanzregelung, einer Spurhaltung und eines Kurvenlichts implementiert. Diese Funktionen werden bereits in Serienfahrzeugen implementiert und stellen in ihrem Verbund eine verteilte Regelung der Fahrzeugführung dar. Dabei bilden die Stellgrößen einer Funktion die Eingangsgrößen einer anderen Funktion; z.B. der Lenkwinkel zwischen Spurhaltung und Kurvenlicht.

Im folgenden sollen die Funktionsweise der implementierten Assistenzsystemfunktionen erläutert werden, um die Kopplung mit der Entwicklungsplattform aufzuzeigen.

Distanzregelung Eine Distanzregelung stellt eine Erweiterung der Geschwindigkeitsregelung dar. Die Fahrbahn wird dazu in Fahrtrichtung durch einen Radarsensor beobachtet und der Abstand des vorausfahrenden Verkehrsteilnehmers bestimmt. Unterschreitet dieser Abstand den für die Wunschgeschwindigkeit des Fahrers erforderlichen Sicherheitsabstand, wird das Fahrzeug verzögert, um einen Auffahrunfall zu vermeiden.

Moderne Systeme, wie die im *DECOS*-Projekt eingesetzte Funktion, verzögern das Eigenfahrzeug bis zum Stillstand, um danach selbständig wieder anzufahren. Daher werden solche Funktionen auch als Stauassistent bezeichnet.

Zum Test der Funktion sind Kolonnenfahrten mit variierender Geschwindigkeit und Stausituationen nachzubilden. Die Systemfunktion erwartet die Entfernung und Geschwindigkeit des vorausfahrenden Fahrzeuges, die durch die Entwicklungsplattform auf dem Echtzeitsimulator berechnet werden.

Als Stellgröße wird ein Verzögerungs- bzw. ein Beschleunigungseingriff berechnet, der vom Echtzeitsimulator in einer Fahrdynamiksimulation verarbeitet wird.

Spurhaltung Durch eine Videokamera wird die Krümmung der Fahrbahn bestimmt und durch Überlagerung eines Lenkwinkels das Fahrzeug in seiner Spur gehalten.

Eine Nachbildung der Kamerasensorik wurde innerhalb des *DECOS*-Projektes nicht realisiert. Die für die Funktion erforderliche Fahrbahnkrümmung wurde direkt aus dem analytischen Straßenmodell auf dem Echtzeitrechner bestimmt und der Funktion als Eingangswert übergeben. Der Lenkwinkel als Stellgröße wird wieder von der Fahrdynamiksimulation verarbeitet.

Kurvenlicht Zur optimalen Ausleuchtung der Fahrbahn bei Kurvenfahrt wird der Lenkwinkel des Eigenfahrzeugs beobachtet und das Abblendlicht durch Schwenken der Scheinwerfer um die Hochachse nachgeführt.

Der zur Funktion erforderliche Lenkwinkel wird bereits von der Spurhaltung berechnet und kann direkt vom Kurvenlicht verwendet werden.

Diese Funktion besteht aus zwei Teilen. Der mit der *DECOS*-Technologie realisierte Teil berechnet die Krümmung der Bahnkurve des Eigenfahrzeugs, der durch den zweiten Funktionsteil im Lichtsteuergerät, das als Echtteil eines Scheinwerfers vorhanden war, zur Berechnung der Schwenkwinkel beider Frontscheinwerfer verwendet wird. Diese Schwenkwinkel können dann von der Visualisierung zur Positionierung zweier Scheinwerferkegel verwendet werden.

Alle Assistenzsystemfunktionen wurden mit den *DECOS*-Werkzeugen implementiert und zu einem Gesamtsystem auf einem Rechen-Cluster, der mit dem Kommunikationssystem *FlexRay* und der *DECOS*-Laufzeitumgebung ausgestattet ist, integriert, s. Abbildung 5.2.

Dieser Rechen-Cluster besteht aus mehreren Knoten mit einer *TriCore*-CPU und einer *FlexRay*-Schnittstelle. Die *DECOS*-Laufzeitumgebung wird auf jedem Knoten zur Verfügung gestellt und integriert die Assistenzsystemfunktionen als zeitgesteuerte Anwendungen, deren Kommunikation untereinander durch den globalen Zeitplan des Rechen-Clusters vorgegeben ist.

Die Kommunikation mit dem Echtzeitrechner, die zur Kopplung mit der Entwicklungsplattform erforderlich ist, wird über den *CAN*-Bus realisiert, s. CAN 1 in Abbildung 5.2, an den ebenfalls der Scheinwerfer als Echtteil angeschlossen ist. Die Schwenkwinkel der Frontscheinwerfer werden über eine Diagnoseschnittstelle des Scheinwerfers an den Echtzeitsimulator übertragen, s. CAN 2 in Abbildung 5.2.

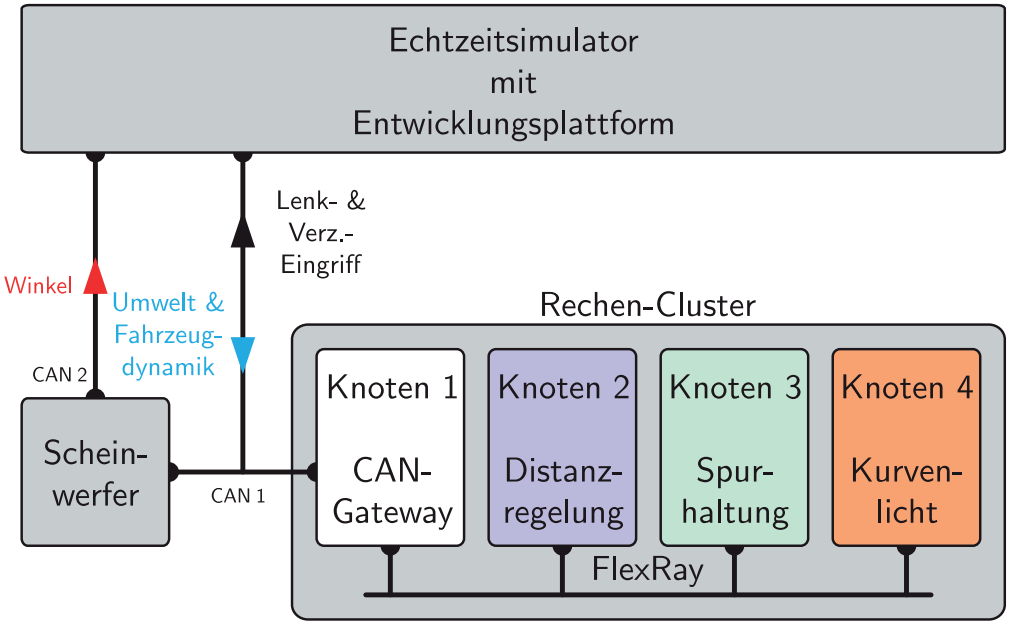

Abbildung 5.2.: Struktur des automobilen Demonstrators.

Da der *CAN*-Bus eine *DECOS*-fremde Technologie darstellt, wird zur Kopplung des Echtzeitsimulators ein sog. *Gateway* definiert, das den Datenverkehr des innerhalb des globalen Zeitplans auf Nachrichten des *CAN*-Bus abbildet.

Durch Anwendung der Entwicklungsplattform wird das Umfeld eines Fahrzeuges durch Einsatz eines Echtzeitsimulators zum Betrieb der Assistenzsystemfunktionen nachgebildet. Ein reales Testfahrzeug ist somit nicht erforderlich.

5.3. Nachbildung des Fahrzeugumfeldes

Durch Bewertung des Fahrzeugumfeldes in Relation zum Eigenfahrzeug können die in Abschnitt 5.2 vorgestellten Assistenzsysteme zur Vermeidung bzw. Verminderung von Unfällen beitragen. Die Funktion jedes Einzelsystems bedingt jedoch seine Stimulation mit Eingangssignalen, wie sie von Sensoren zur Erfassung des Fahrzeugumfeldes generiert werden.

Kapitel 1 beschreibt mehrere Software-Module, die zur Nachbildung des Fahrzeugumfeldes durch einen Echtzeitsimulator verwendet werden können. In diesem Abschnitt soll die Konfiguration dieser Software-Module, wie sie

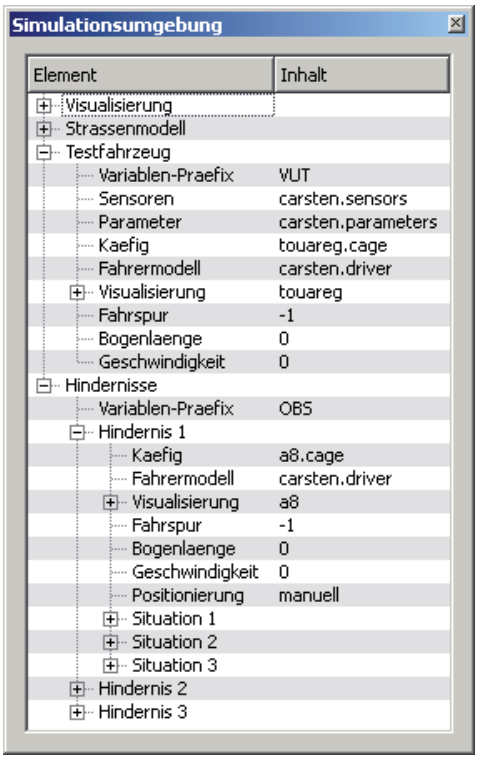

Abbildung 5.3.: Ansicht der Simulationsumgebung in *PRAETORIA*.

zur Realisierung des *DECOS*-Projektes eingesetzt wurden, durch die in Kapitel 4 vorgestellte Simulationsumgebung beschrieben werden.

Zur Konfiguration der Simulationsumgebung wurde das Werkzeug *PRAETORIA* entwickelt. Ähnlich Abbildung 4.2 werden dem Anwender die möglichen Einstellungen der Simulationsumgebung präsentiert, s. Abbildung 5.3.

Die Teilbäume, die die Komponenten der Simulationsumgebung darstellen, können versteckt oder angezeigt werden, um die präsentierten Informationen auf die interessierenden Einstellungen zu reduzieren. Zum Test eines Assistenzsystems ist es i.d.R. nicht erforderlich die Visualisierung oder den Fahrbahnverlauf der Simulationsumgebung für jede Testreihe anzupassen, so daß diese Einstellungen häufig übernommen werden können.

Zur Konfiguration einer Komponente ist ein entsprechender Dialog zu öffnen, der die zugehörigen Einstellungen zur Anpassung durch den Anwender bereithält. Die resultierende Konfiguration wird dann in den Blättern des zugehörigen Teilbaums angezeigt.

Der Test der Distanzregelung des *DECOS*-Projektes erfordert die Erzeugung von Stau- und Kolonnenfahrten durch die Verkehrssimulation. Um

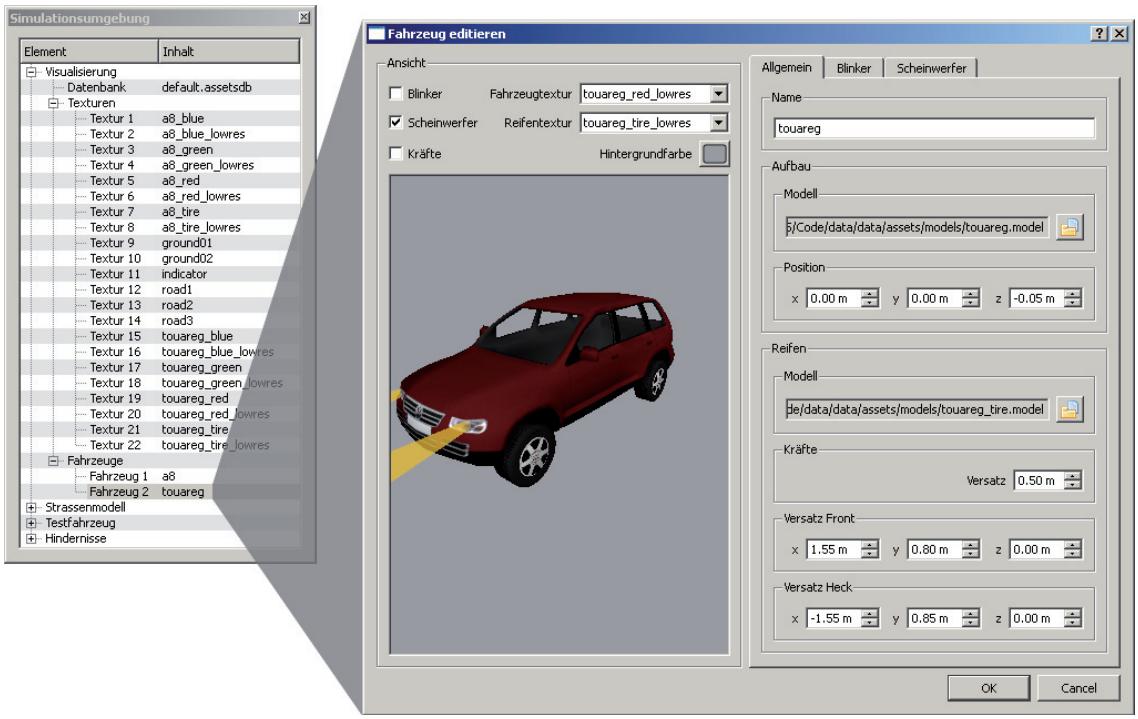

Abbildung 5.4.: Konfiguration eines Fahrzeuges zur Visualisierung.

einem Betrachter während der verschiedenen Versuchsreihen ein möglichst abwechslungsreiches Fahrzeugumfeld zu bieten, wurden in der Visualisierungsdatenbank zwei Fahrzeuge erzeugt, die jeweils mit verschiedenen Texturen belegt werden konnten, um mehrere Verkehrsteilnehmer mit unterschiedlicher Fahrzeuglackierung darstellen zu können, s. Abbildung 5.4.

Die Visualisierungsdatenbank wird als externe Dateireferenz in die Simulationsumgebung geladen. *PRAETORIA* integriert die Verwaltung der Datenbank in den Baum der Simulationsumgebung, so daß die Konfiguration der gesamten Simulationsumgebung an zentraler Stelle vorgenommen werden kann. Die Definition von Texturen erfolgt über die eindeutige Zuordnung eines Textur-Namens zu der gewünschten Bilddatei, die im Dateisystem des Bedien-PCs vorhanden sein muß.

Der Dialog in Abbildung 5.4 dient zur Konfiguration eines Fahrzeuges. Jedem Fahrzeug ist ein eindeutiger Name zugeordnet, der bei der Konfiguration des Testfahrzeuges und der Hindernisse verwendet wird. Nach Abschnitt 3.7.1 werden die Drahtgittermodelle des Aufbaus und der Räder relativ zueinander platziert. Die dreidimensionale Ansicht stellt das Fahr-

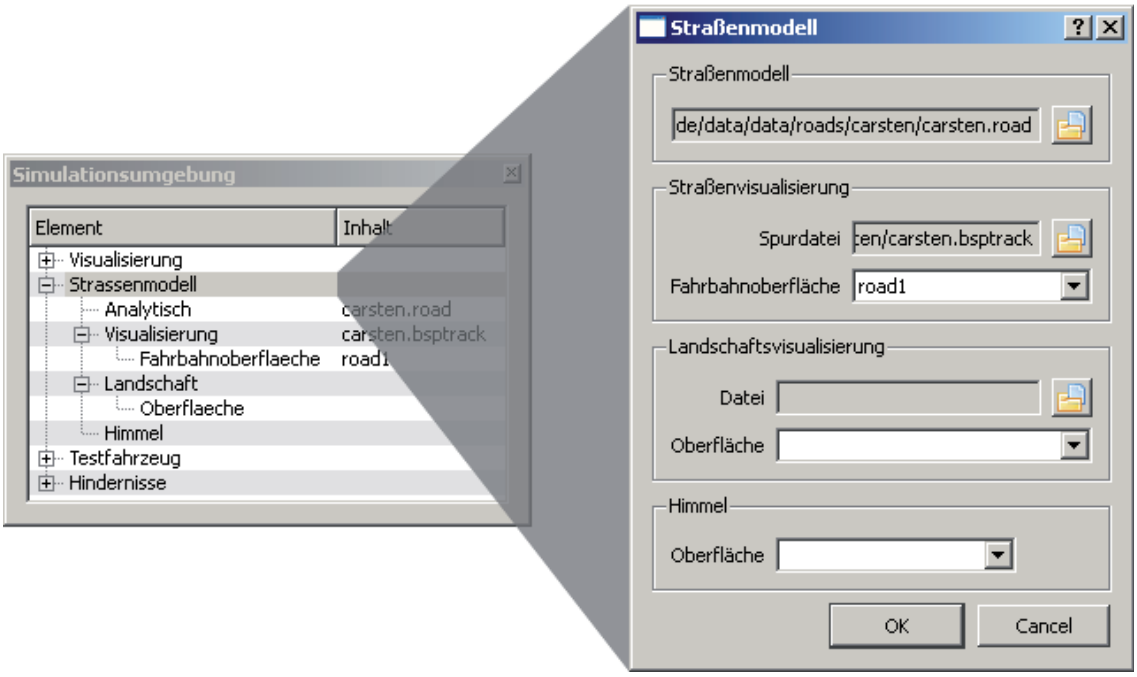

Abbildung 5.5.: Konfiguration des Rundkurses.

zeug entsprechend den vorgenommenen Einstellungen dar, wobei bereits auf Texturen der Datenbank zur Belegung des Aufbaus und der Reifen zurückgegriffen werden kann.

Darüber hinaus kann jedes Fahrzeug mit Scheinwerfern und Blinkern ausgestattet werden. Damit können die Schwenkwinkel des adaptiven Lichtsystems in der Visualisierung dargestellt werden. Der Verkehrssimulation wird durch Setzen der Blinker ermöglicht einen Spurwechsel auch graphisch anzuzeigen.

Die Konfiguration des Fahrbahnverlaufes benötigt lediglich drei Parameter, s. Abbildung 5.5.

Im zugehörigen Dialog wird das analytische Straßenmodell der Echtzeitsimulation, das Drahtgittermodell zur Visualisierung und der Name der Fahrbahntextur aus der Visualisierungsdatenbank angegeben. Der Rundkurs wird mit dem Editor aus Abschnitt 2.5 erstellt. Das zur Darstellung verwendete Drahtgittermodell inklusive dessen Optimierung durch einen Binärbaum wird ebenfalls von diesem Editor erstellt.

Für zukünftige Erweiterungen sind Einstellungen des Reliefs sowie Texturen der Bodenoberfläche und des Horizonts vorgesehen.

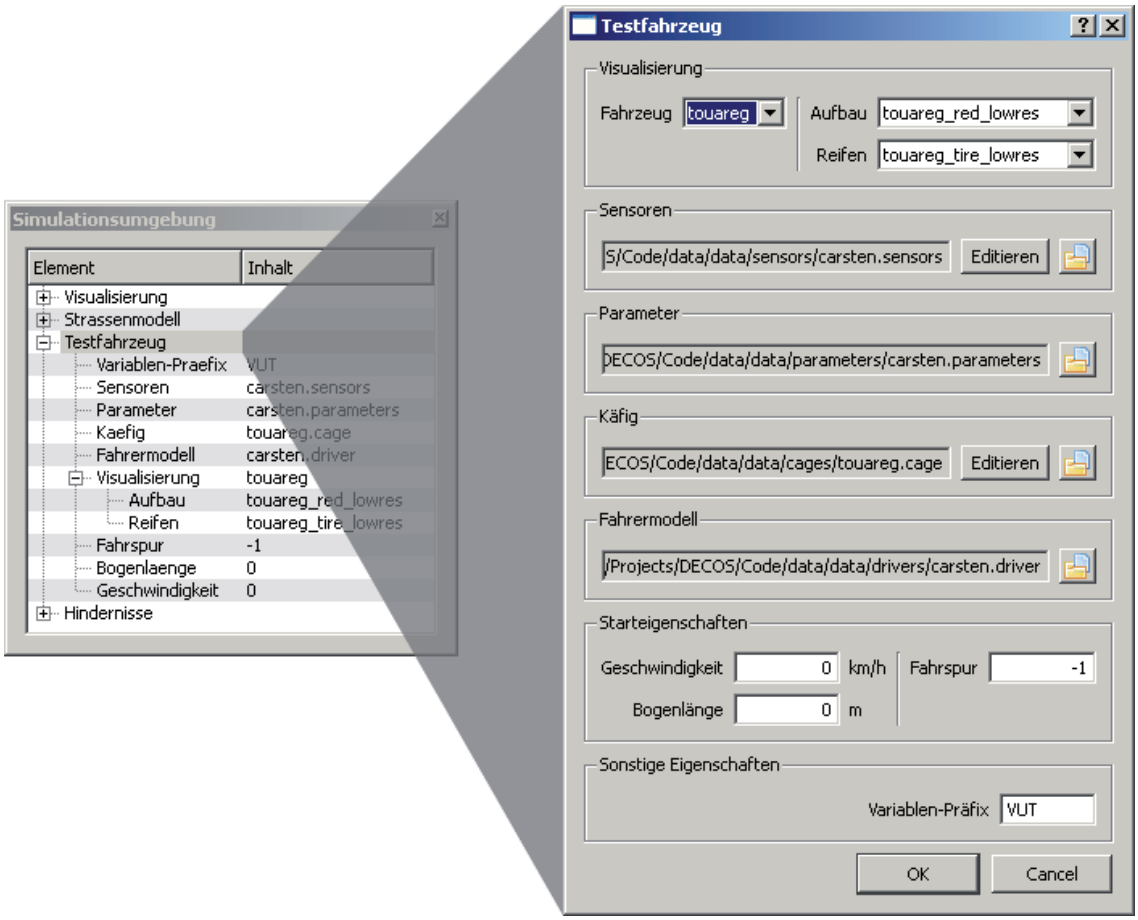

Abbildung 5.6.: Konfiguration des Testfahrzeuges.

Den Dialog zur Konfiguration des Testfahrzeuges zeigt Abbildung 5.6.

Zur Bedeutung der Parameter sei auf Abschnitt 4.1.3 verwiesen. Die Einstellungen der Visualisierung stellen symbolische Namen der Visualisierungsdatenbank dar. Hinzugekommen ist der Variablenpräfix der Parameternamen zur Kopplung der Simulationsmodule des Echtzeitsimulators, s. Abschnitt 4.2.2.

Eine Besonderheit des Testfahrzeuges ist seine Ausstattung mit Sensoren, die das Fahrzeugumfeld erfassen und deren Funktion durch die Sensormodelle auf dem Echtzeitsimulator nachgebildet werden. Zur Konfiguration dieser Sensormodelle dient der Dialog in Abbildung 5.7, der in den Einstellungen des Testfahrzeuges aufgerufen werden kann.

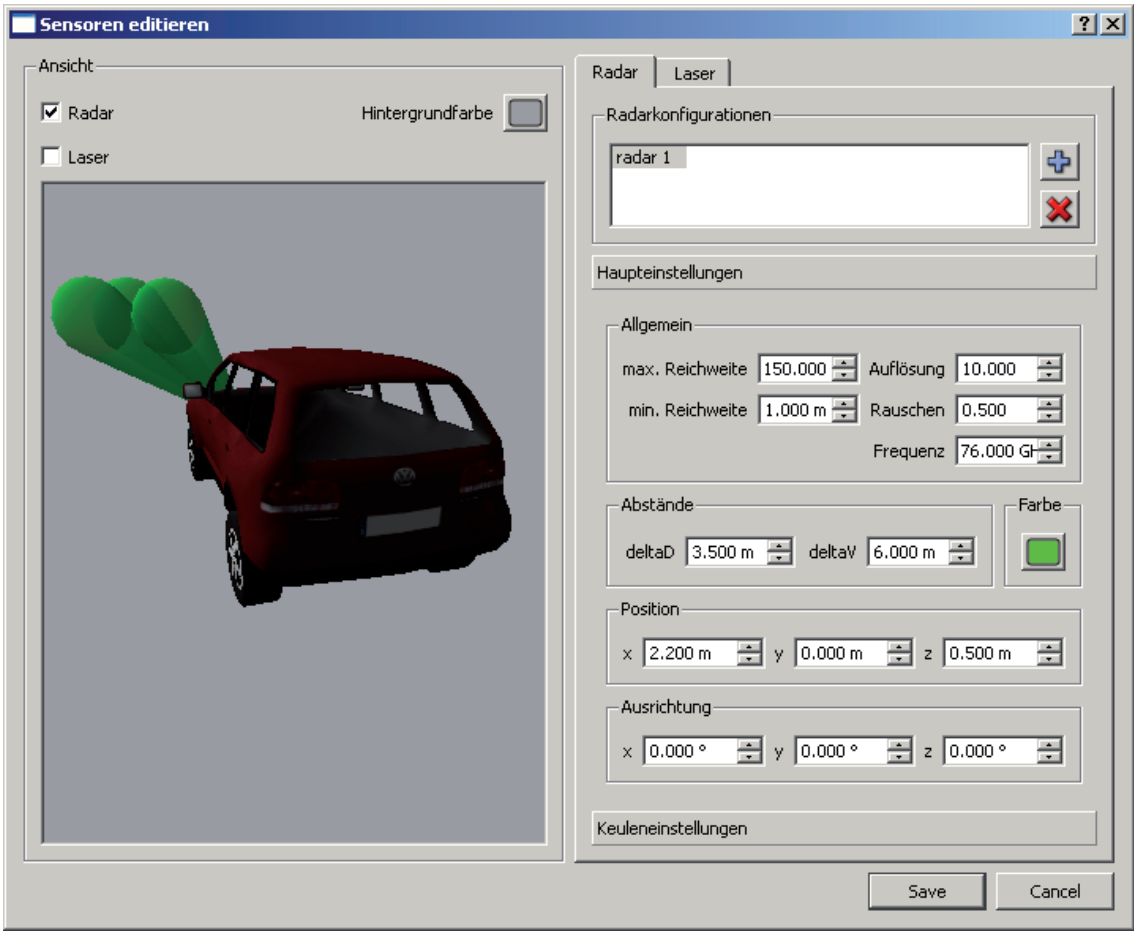

Abbildung 5.7.: Konfiguration der Sensorik des Testfahrzeuges.

Der Dialog gestattet die Konfiguration von Radar- und Lidar-Sensoren. Das Testfahrzeug kann dabei mit mehreren Sensoren beider Sensortypen ausgestattet werden. Jedem Sensor müssen seine Koordinaten relativ zum Fahrzeugkoordinatensystem und die Sende- bzw. Empfangscharakteristika zugeordnet werden.

Radar-Sensoren können über mehrere Keulen verfügen, die relativ zu der Grundausrichtung des Sensors orientiert werden können, s. Abbildung 5.7. Zum Einsatz innerhalb des *DECOS*-Projektes wurde jedoch nur ein Radar-Sensor bestehend aus einer Keule benötigt.

Jedem Lidar-Sensor können mehrere Strahlen zugeordnet werden, die die verschiedenen Ebenen zur Abtastung der Umwelt durch Kreissegmente geometrisch beschreiben. Die einzelnen Ebenen können durch Rotation aus der Grundausrichtung erzeugt werden.

Die Konfiguration des Fremdverkehrs wird durch die Platzierung mehrerer Hindernisse vorgenommen, s. Abbildung 5.8, deren Variablenpräfix in Analogie zum Testfahrzeug ebenfalls vorgegeben werden kann.

In Abhängigkeit der Visualisierungsdatenbank wird das Fahrzeug und die Texturen des Aufbaus und der Reifen definiert. Zur Erläuterung der weiteren Optionen sei auf Abschnitt 4.1.4 verwiesen.

Eine Besonderheit stellt der Käfig der Hindernisse dar, der in der Verkehrssimulation zur Kollisionsabfrage zwischen allen Verkehrsteilnehmern verwendet wird. Daher ist das Testfahrzeug ebenfalls mit einem Käfig ausgestattet.

Die Sensormodelle verwenden den Käfig, der ein reduziertes Drahtgittermodell des Fahrzeuges darstellt, zur Bestimmung der Relativbewegung der Hindernisse zum Testfahrzeug. Dazu kann der Käfig nach Abbildung 5.9 parametriert werden.

Zunächst ist der Käfig relativ zum Fahrzeugkoordinatensystem zu platzieren. Da den Sensormodellen durch Kopplung an die Datenbasis des Echtzeitsimulators die Koordinaten der Hindernisse bekannt sind, können die Dreiecke des Käfigs absolut im Inertialsystem positioniert werden.

In den Sensormodellen werden die Detektionsbereiche der Sensoren in einzelne Strahlen diskretisiert, die einzeln mit den Dreiecken des Käfigs zum Schnitt gebracht werden. Tritt ein solcher Schnitt auf, stehen dem Sensormodell pro Dreieck mehrere Parameter zur Verfügung, die zur Nachbildung des Echos verwendet werden können. Die Parameterwerte können im Dialog zur Konfiguration eines Käfigs für jedes Dreieck unabhängig voneinander vorgegeben werden, s. Abbildung 5.9. Ihre Bedeutung hängt von der Implementierung des jeweiligen Sensormodells ab.

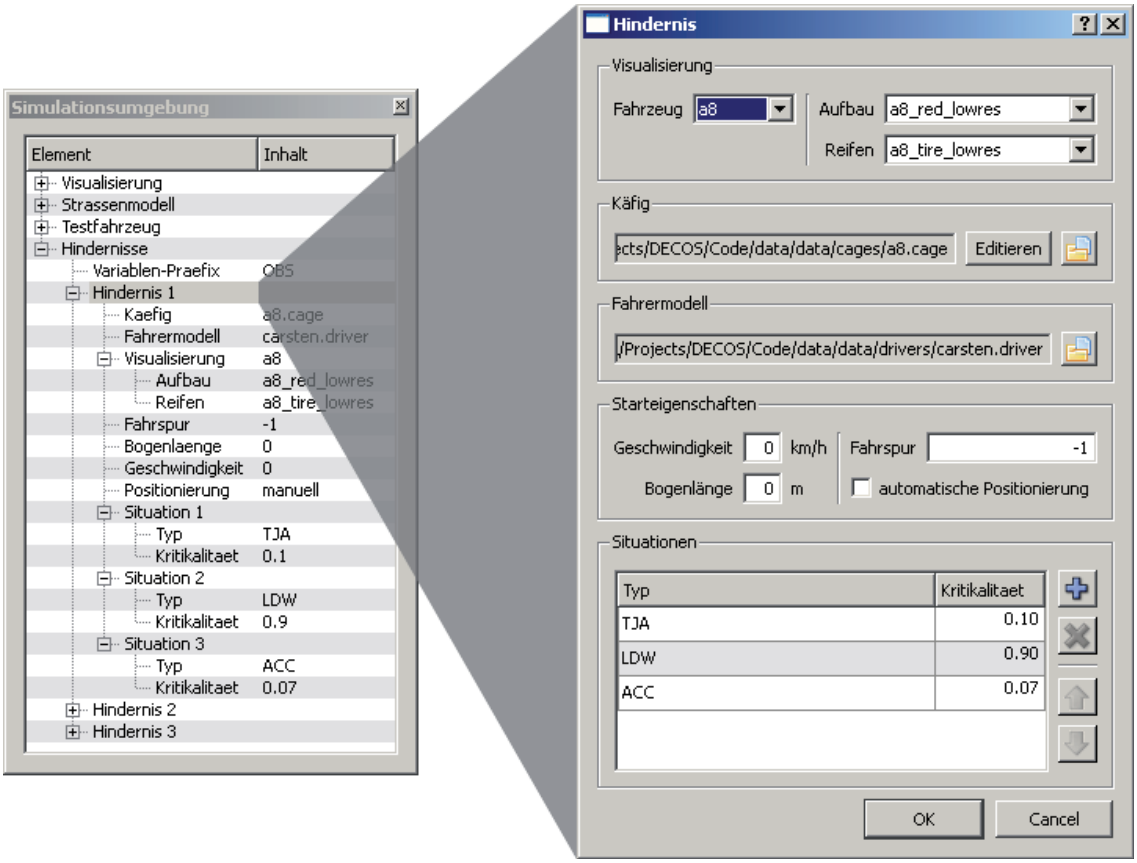

Abbildung 5.8.: Dialog zur Konfiguration eines Hindernisses.

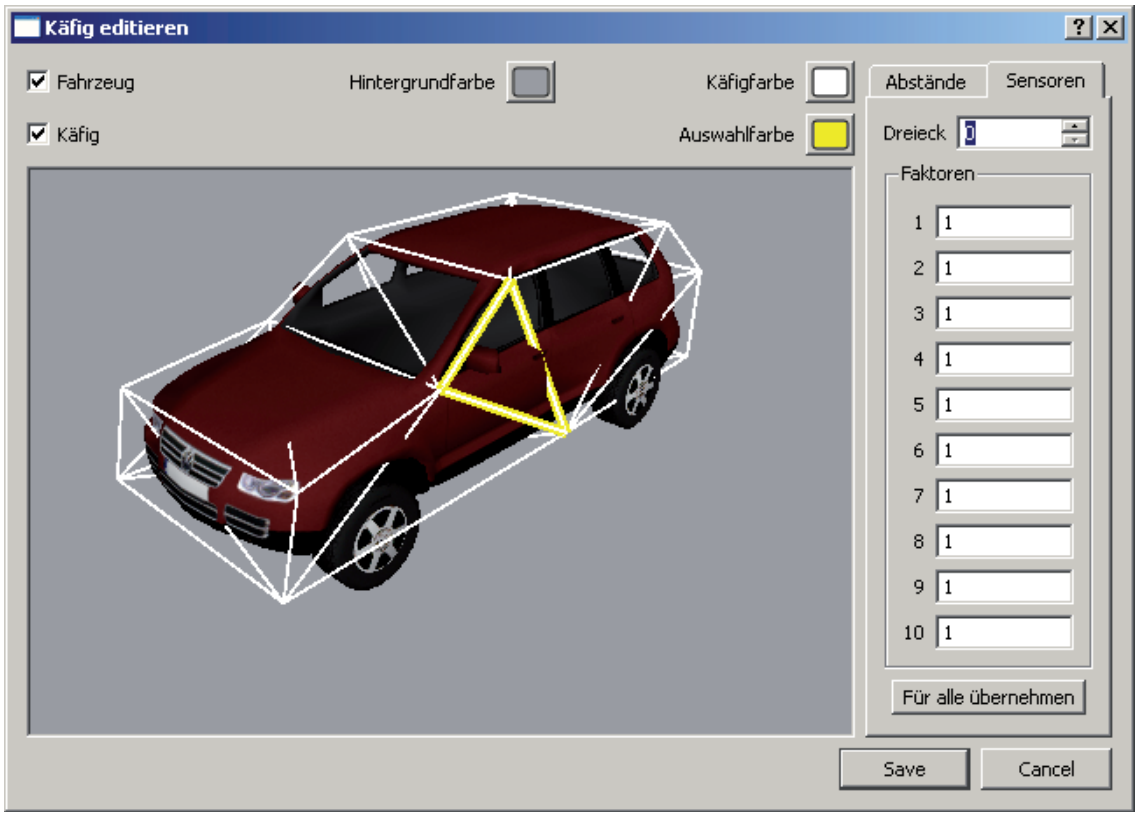

Abbildung 5.9.: Konfiguration des Käfigs eines Fahrzeuges.

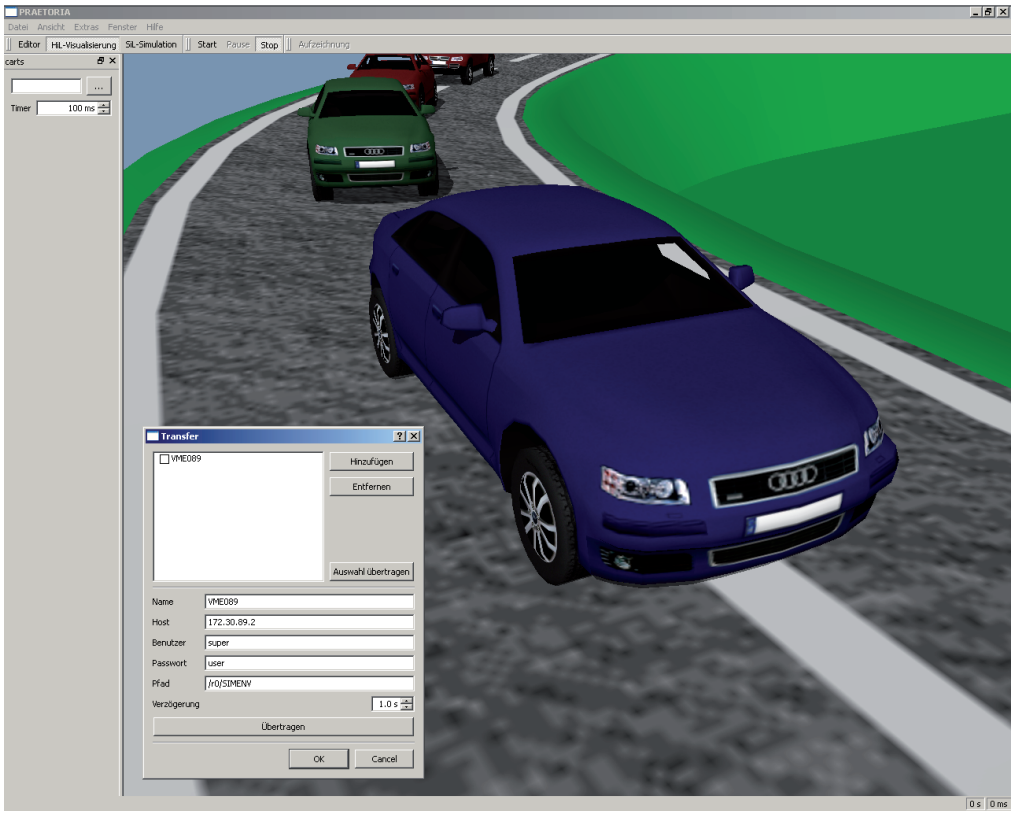

Abbildung 5.10.: Benutzeroberfläche *PRAETORIAs* zur Visualisierung der Echtzeitsimulation.

Das Werkzeug *PRAETORIA* gestattet es einem Anwender, alle Einstellungen der Simulationsumgebung, bis auf die Parameter des Testfahrzeuges und die Konfiguration der Fahrer, über eine graphische Oberfläche zu konfigurieren. Darüber hinaus dient *PRAETORIA* zur Visualisierung der Echtzeitsimulation und Übertragung der Simulationsumgebung an die Rechenknoten des Echtzeitsimulators. Einem Anwender wird somit die Konfiguration und Visualisierung der Entwicklungsplattform durch ein zentrales Werkzeug zur Verfügung gestellt. Abbildung 5.10 zeigt die Benutzeroberfläche zur Visualisierung der Echtzeitsimulation.

5.4. Software-in-the-Loop-Simulation

Zum Test eines Assistenzsystems mit der vorgestellten Entwicklungsplattform wird die auf einem Steuergerät implementierte Systemfunktion und ein Echtzeitsimulator benötigt. Dazu ist zunächst die Funktionalität des Assi-

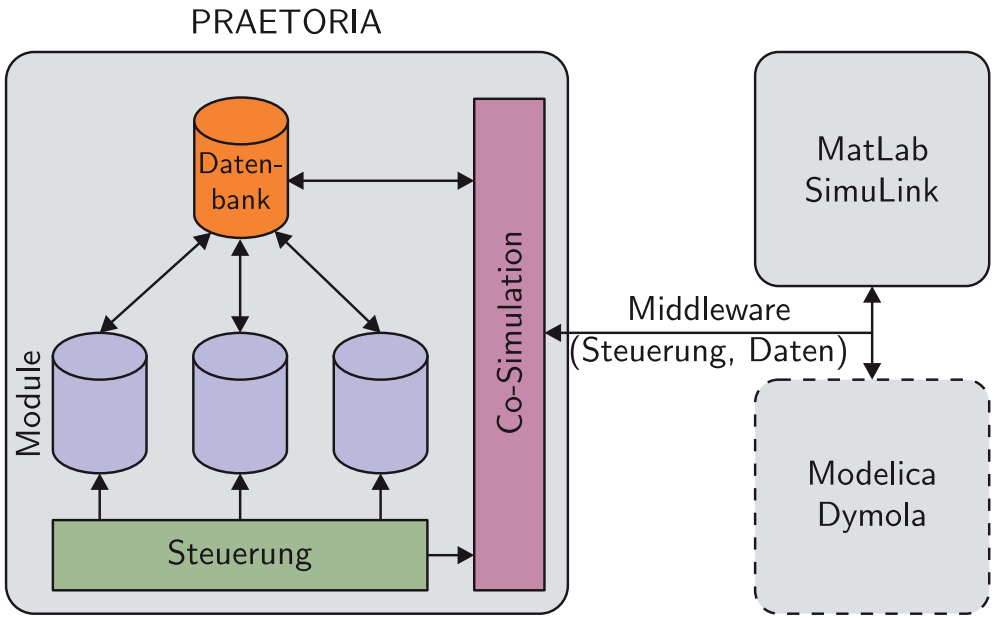

Abbildung 5.11.: Struktur der Software-in-the-Loop Simulation.

stenzsystems zu entwickeln und auf einem Steuergerät zu implementieren. Die Entwicklung von Systemfunktionen findet dabei zunehmend mit sog. *Rapid Prototyping*-Werkzeugen statt.

Die Simulationsumgebung der Entwicklungsplattform wurde als Programmbibliothek mit der Programmiersprache *C++* entwickelt und wird zur Übersetzungszeit an die Simulationsprogramme, die später auf dem Echtzeitsimulator ausgeführt werden, gebunden. Bei der Verarbeitung der Simulationsumgebung kommen nur Funktionen der Standard-Programmbibliotheken zum Einsatz. Die Simulationsprogramme implementieren mathematische Modelle des Fahrzeugumfeldes, die ebenfalls mit Funktionen der Standard-Programmbibliotheken realisiert werden können. Eine Abhängigkeit des resultierenden Simulationsprogramms besteht daher nur zur Laufzeitumgebung des Echtzeitsimulators.

Das Programm *PRAETORIA* wurde daher um eine Möglichkeit zum Ausführen von Simulationsprogrammen auf dem Bedien-PC und eine Co-Simulation erweitert, s. [Gräbel 2007]. Die Struktur dieser Software-in-the-Loop-Umgebung ist in Abbildung 5.11 dargestellt.

Zentrales Element der Software-in-the-Loop-Umgebung ist eine *Datenbank*, in der die Variablen zur Kopplung der *Simulationsmodule* zusammengeführt werden. Simulationsmodule können durch Neu-Übersetzen der Si-

mulationsprogramme des Echtzeitsimulators für die Systemarchitektur des Bedien-PCs erzeugt werden und stellen dynamische Bibliotheken dar.

Die *Steuerung* der Simulation übernimmt ein Unterprogramm, das den Zustandsautomaten nach Abbildung 4.5 implementiert. In jedem Zustand wird die entsprechende Rückruffunktion eines Simulationsmoduls ausgeführt. Von der Steuerung wird ebenfalls die Laufzeitumgebung des Echtzeitsimulators bereitgestellt, die es den Modulen gestattet Verbindungen von Variablen zur Datenbank herzustellen.

Gesteuert wird der Simulationsablauf durch einen Zeitgeber des Bedien-PCs. Mit Beginn jedes Simulationsschrittes werden für alle Simulationsmodule die Eingangsvariablen aus der Datenbank aktualisiert, der Simulationsschritt durchgeführt und die Ergebnisse in die Datenbank zurück geschrieben. In Abhängigkeit der Komplexität der Simulation kann auf dem Bedien-PC bereits Echtzeit erreicht werden. Praktische Versuche haben jedoch gezeigt, daß die Kombination aus Visualisierung, Verkehrssimulation und Sensormodellen nicht vom Bedien-PC in Echtzeit verarbeitet werden kann.

Die Funktion eines Fahrerassistenzsystems kann jetzt als weiteres Simulationsmodul entwickelt werden und an die Entwicklungsplattform, die auf dem Bedien-PC ausgeführt wird, gekoppelt werden. Um jedoch die Entwicklung des Assistenzsystems mit Werkzeugen des *Rapid Prototyping* zu ermöglichen, wurde die Datenbank und die Steuerung um eine Schnittstelle zur *Co-Simulation* erweitert.

Unter einer Co-Simulation versteht man die Kopplung mehrerer Simulationswerkzeuge zur Modellierung eines Gesamtsystems. Jedes Werkzeug kann dabei unter Berücksichtigung seiner Stärken zur Beschreibung bestimmter Problemstellungen eingesetzt werden. Durch eine zentrale Instanz werden die Simulationsschritte aller Teilnehmer zeitlich synchronisiert und die gekoppelten Zustandsgrößen verteilt, um einen konsistenten Simulationszustand zu gewährleisten.

Um verschiedene Simulationswerkzeuge miteinander zu synchronisieren, verwendet die durch *PRAETORIA* bereit gestellte Co-Simulation die standardisierte Programmierschnittstelle *MPI*[5], die zum Nachrichtenaustausch zwischen Anwendungen zur parallelen Datenverarbeitung entwickelt wurde. Momentan wird eine Schnittstelle zum Programmsystem *MatLab/SimuLink* angeboten. Eine Erweiterung zur Kopplung weiterer Simulationswerkzeuge, z.B. *Modelica/Dymola*, ist möglich, s. Abbildung 5.11.

[5]engl.: *Message Passing Interface*

Die Zustände der nachgebildeten Laufzeitumgebung des Echtzeitsimulators, s. Abbildung 4.5, werden von der Steuerung an alle Teilnehmer der Co-Simulation übertragen. Während der Initialisierung werden die benötigten Lese- und Schreibzugriffe der Teilnehmer auf die Datenbank ermittelt und mit der Datenbank *PRAETORIAs* abgeglichen. Jeder Teilnehmer kann so eine Kopie der globalen Datenbank erstellen, die jedoch nur die von ihm benötigten Variablen enthält. Mit Eintritt in jeden weiteren Zustand wird durch *MPI* der Zustandswechsel signalisiert, auf die Bearbeitung des Zustandes durch die externen Teilnehmer gewartet und die Schreibzugriffe aller Teilnehmer auf die Datenbank synchronisiert. Durch globale Bekanntgabe der Ausgangsgrößen aller Teilnehmer, kann der für die Schreibzugriffe erforderliche Datenaustausch mit einer minimalen Datenmenge abgewickelt werden.

Eine Integration *SimuLinks* in die Co-Simulation wird durch einen spezialisierten *SimuLink*-Block realisiert. Zur Konfiguration dieses Blocks ordnet der Anwender den Ein- und Ausgängen Variablen der Datenbank zu. Lesende Zugriffe in die Datenbank werden dabei auf Ausgänge abgebildet, schreibende Zugriffe auf Eingänge. Die Anzahl der verwendeten Ein- bzw. Ausgänge wird automatisch an den Wunsch des Anwenders angepasst.

Für die automobile Anwendung innerhalb des *DECOS*-Projektes sollte die Leistungsfähigkeit des eingesetzten Kommunikationssystems *FlexRay* bzgl. der Zeitsteuerung und Fehlertoleranz untersucht werden. Dazu wurden die Funktionen der Assistenzsysteme als Modelle in *SimuLink* implementiert und, entsprechend ihrer Konfiguration auf dem Rechen-Cluster, durch Nachbildung des globalen Zeitplanes ausgeführt.

Die zur plausiblen Stimulation der Assistenzsysteme benötigten Eingangssignale der Assistenzsysteme wurden durch die auf demselben PC ausgeführte Entwicklungsplattform erzeugt und durch die Co-Simulation *PRAETORIAs* mit den Modellen in *SimuLink* synchronisiert, s. Abbildung 5.12.

5.5. Leistungsfähigkeit der Visualisierung

In diesem Abschnitt soll die Leistungsfähigkeit der Visualisierung diskutiert werden. Dazu wurden mehrere Fahrzeuge mit der Geschwindigkeit v entlang der Mittellinie eines Rundkurses bewegt. Mit der Abtastzeit τ kann die auf der Mittellinie zurückgelegte Bogenlänge $D(t) = D(t-\tau) + v \cdot \tau$ aktualisiert werden. Die Daten des Rundkurses und des verwendeten Bedien-PCs zeigt Tabelle 5.1, die Anordnung der Fahrzeuge ist in Abbildung 5.13 zu sehen.

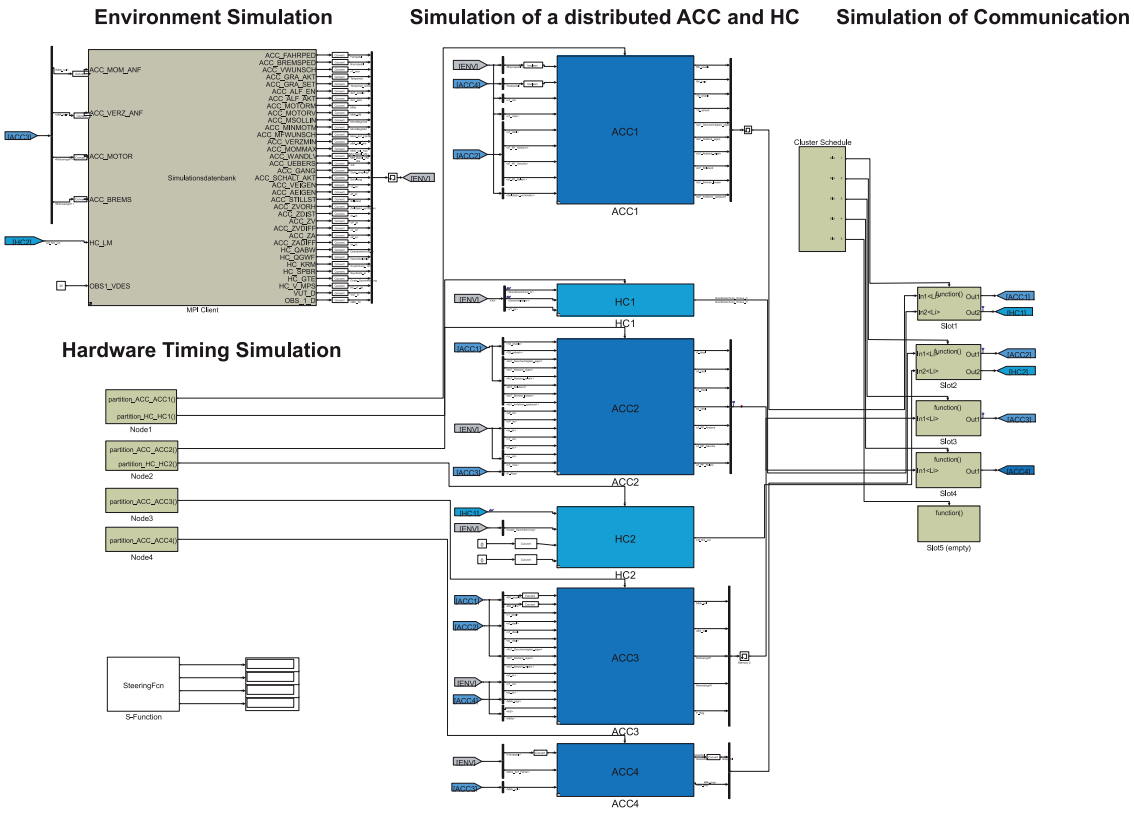

Abbildung 5.12.: Integration von *SimuLink* in die Co-Simulation (Dipl.-Inf. Thorsten Frunzke, Audi Electronics Venture GmbH).

Länge	$13516.5m$
Abmaße	$3111.3m \times 5273.9m$
Teildämme	102
Abmaße eines Teilvolumens	$194.4m \times 164.8m$
Dreiecke	32420
Prozessor	Pentium M 750, $1.86GHz$, $2MB$ L2-Cache
Hauptspeicher	$2GB$, $266MHz$ DDR2
Graphikkarte	GeForce Go 6800, $256MB$ RAM, PCI-Express x16

Tabelle 5.1.: Daten des Rundkurses (oben) und des Bedien-PCs (unten).

Abbildung 5.13.: Anordnung der Fahrzeuge zur Bestimmung der Leistungsfähigkeit der Visualisierung.

Der Verlauf des Rundkurses wurde aus der digitalisierten Karte eines Autobahnrings erzeugt; die minimalen Abmaße eines Teilvolumens wurden auf $100m \times 100m$ begrenzt.

Bei einer Tiefe des Sichtfeldes von $400m$ werden 4 Teilvolumina berechnet, falls der Betrachter entlang einer Koordinatenachse ausgerichtet ist. In der Praxis wurden jedoch im Mittel $6-8$ Teilvolumina berechnet, da das achsenparallele Begrenzungsvolumen, mit dem die Teilvolumina auf Sichtbarkeit getestet werden, aus den maximalen bzw. den minimalen Koordinaten des rotierten Sichtfeldes erzeugt wird.

Wird eine Überlappung des Sichtfeldes mit einem Teilvolumen festgestellt, wird die Verarbeitung aller Dreiecke des Teilvolumens durch die Graphikpipeline eingeleitet. Die Verwendung achsenparalleler Begrenzungsvolumina führt meist zu einer erhöhten Anzahl der als sichtbar erkannten Dreiecke, ist jedoch eine effektive Methode, um die Graphikpipeline vor einem zu großen Datenaufkommen zu bewahren. Dies betrifft insbesondere die vor der Projektions-Stufe befindlichen Pipeline-Stufen, da diese noch keine Kenntnis über die Sichtbarkeit der Geometrie besitzen und somit alle eingehenden Daten verarbeiten müssen.

Als Fahrzeug der Verkehrsteilnehmer kommt ein Drahtgittermodell zum Einsatz, das aus ca. 2300 Dreiecken besteht. Mit den 4 Reifen, die jeweils aus 200 Dreiecken bestehen, sind so pro Verkehrsteilnehmer ca. 3100 Dreiecke zu verarbeiten. Das Begrenzungsvolumen eines Verkehrsteilnehmers wird nach seiner Rotation in ein achsenparalleles Volumen umgewandelt und auf Überlappung mit dem achsenparallelen Sichtfeld geprüft. Die zugehörigen Dreiecke werden ggf. durch die Graphikpipeline verarbeitet. Da die Kommunikation zwischen Echtzeitrechner und Bedien-PC auf eine Übertragung von 255 Parameter beschränkt ist, lassen sich mit einem minimalen Satz von 6 Parametern pro Fahrzeug, s. Abschnitt 4.2.2, 42 Fahrzeuge visualisieren. Da einige Parameter zur Steuerung des Echtzeitsystems benötigt werden, wurde die maximale Anzahl der Fahrzeuge auf 40 beschränkt.

Die Leistungsfähigkeit von Visualisierungssystemen wird i.d.R. in "Bilder pro Sekunde" angegeben. Zur Bestimmung dieses Wertes wurde die Anzahl der Verkehrsteilnehmer auf dem beschriebenen Rundkurs variiert und die Anzahl der Berechneten Bilder pro Sekunde durch das verwendete Graphiksystem ermittelt. Jede Messung in Abhängigkeit der dargestellten Verkehrsteilnehmer wurde mit und ohne Verwendung des Binärbaumes durchgeführt, um den Einfluß des beschriebenen Verfahrens zu verdeutlichen. Die Ergebnisse sind in Tabelle 5.2 und Abbildung 5.14 zusammengefasst.

Fahrzeuge	*BSP* verw.	*BSP* nicht verw.
5	120	74
10	67	49
15	47	38
20	35	30
25	28	24
30	23	20
35	20	18
40	18	16

Tabelle 5.2.: Anzahl der pro Sekunde berechneten Bilder in Abhängigkeit der dargestellten Fahrzeuge (Ergebnisse).

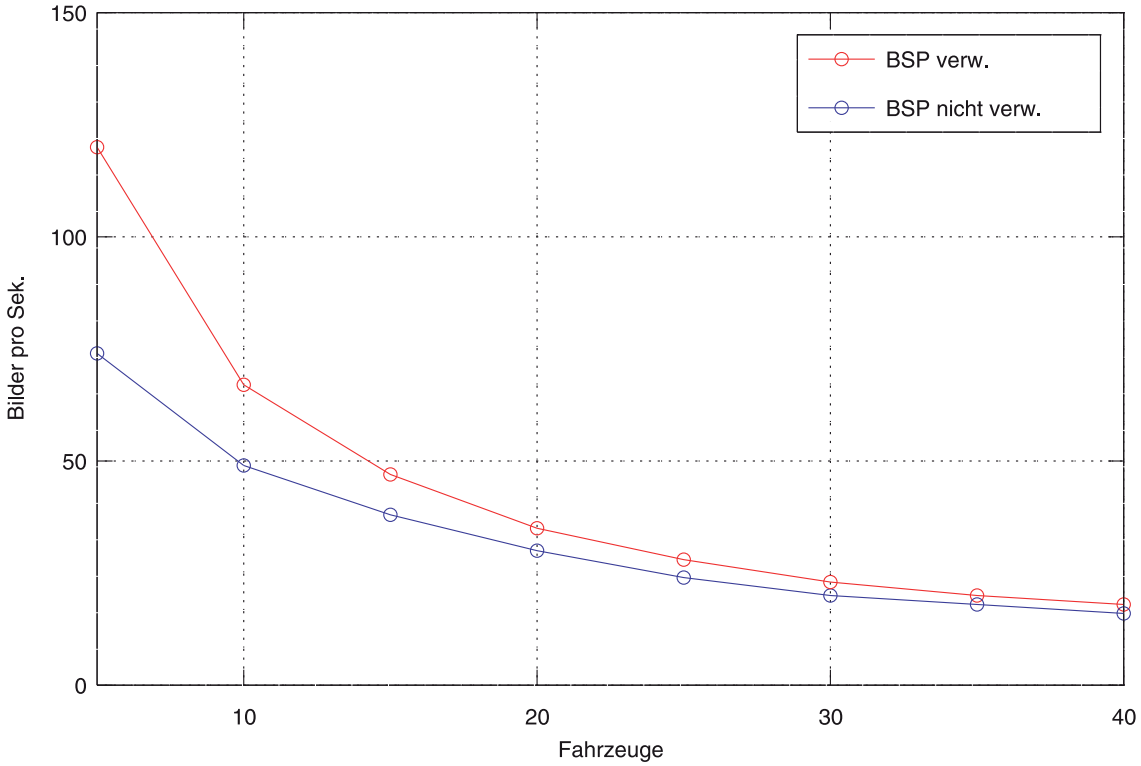

Abbildung 5.14.: Anzahl der pro Sekunde berechneten Bilder in Abhängigkeit der dargestellten Fahrzeuge (Tendenz).

Erwartungsgemäß wird durch Verwendung des Binärbaumes eine höhere Bildrate erzielt. Mit zunehmender Anzahl an Fahrzeugen nähern sich die beiden Bildraten jedoch an. Die Anzahl der Dreiecke der Fahrzeuge, $40 \cdot 3100 = 124000$, beträgt in beiden Fällen ein vielfaches der Dreiecke des Rundkurses.

Ohne Binärbaum werden mehr Dreiecke in die Graphikpipeline eingeleitet, nicht sichtbare Dreiecke jedoch nach der Projektions-Stufe wieder aus ihr entfernt. Da die Dreiecke aller Fahrzeuge sichtbar sind, dominiert ihre Anzahl die Leistungsfähigkeit der Visualisierung.

5.6. Ergebnisse

Das Ergebnis dieser Arbeit ist die Umsetzung der plattformübergreifenden Simulationsumgebung aus Kapitel 4 zur Parametrierung der Simulationsmodule und die Entwicklung des Programmsystems *PRAETORIA* zur Steuerung des Echtzeitsimulators und zur Visualisierung des Verkehrsgeschehens nach Kapitel 3.

Mit diesen Werkzeugen konnten die Verkehrssimulation und die zu deren Erfassung benötigten Sensormodelle sowie der Situationsgenerator zur Erzeugung geeigneter Testszenarien realisiert werden.

In diesem Abschnitt soll die Anwendung der entwickelten Werkzeuge in den oben genannten Gebieten kurz erläutert werden.

5.6.1. Anwendung zur Modellierung der Umfeldsensorik und Entwicklung von Fahrermodellen

Die Parametrierung der Fahrermodelle erfolgt durch die Simulationsumgebung individuell für jedes Hindernis. Dabei sind die Eigenschaften eines Fahrers in einer externen Datei gespeichert, so daß problemlos dieselben Eigenschaften auf mehrere Fahrer übertragen werden können. Darüber hinaus werden in der Simulationsumgebung für jedes Hindernis die Starteigenschaften, z.B. die Fahrspur und Geschwindigkeit, festgelegt.

Für die Sensormodelle ist der Käfig, der jedem Hindernis zugeordnet werden muß, von entscheidender Bedeutung. Mit ihm wird die durch die Sensorik erfasste Geometrie und deren Reflexionseigenschaften bezüglich eines Sensors definiert, s. Abbildung 4.3. Damit diese Käfige von den Sensormodellen erfasst werden können, müssen sie in der virtuellen Umwelt platziert werden. Dazu sind die sechs kartesischen Koordinaten aus Abschnitt 4.2.2

ausreichend, die von der Verkehrssimulation und der Fahrdynamiksimulation des Testfahrzeuges berechnet werden müssen. Der Erfassungsbereich eines Sensores ergibt sich dann aus seiner relativen Ausrichtung zum Testfahrzeug und dessen Orientierung in der virtuellen Umwelt.

Die Konfiguration der eingesetzten Sensoren wird in der Simulationsumgebung am Testfahrzeug vorgenommen, dessen Bestandteil das zu testende Assistenzsystem inklusive seiner Sensoren ist. Konfiguriert werden können die Ausrichtung und die Eigenschaften von Laser- und Radarsensoren, s. Abbildung 5.7.

Eine vollständige Beschreibung der eingesetzten Fahrer- und Sensormodelle gibt [Tellmann 2011].

Abbildung 5.15 zeigt die Ausrichtung dreier Radarsensoren. Damit während des Tests eine Identifikation der Hindernisse stattfinden kann, sind die Erfassungsbereiche der Sensoren transparent visualisiert.

5.6.2. Anwendung zur Erzeugung kritischer Verkehrsszenarien

Zum Test von Fahrerassistenzsystemen werden reproduzierbare Testszenarien benötigt, die einen Eingriff des Assistenzsystems provozieren. Dazu kann den Fahrzeugen des simulierten Fremdverkehrs, den sog. Hindernissen, durch einen Situationsgenerator eine Trajektorie aufgeprägt werden, deren Kurs zu einer Kollision mit dem simulierten Eigenfahrzeug führt. Die Bewertung eines solchen Manövers erfolgt dabei durch die sog. Kritikalität, die eine Aussage darüber trifft, ob die vom Fahrerassistenzsystem zu bewertende Situation mit oder ohne Eingriff in die Fahrzeugführung zu einem Schaden für die involvierten Verkehrsteilnehmer führt.

In der Simulationsumgebung ist für jedes Hindernis eine Liste mit kritischen Verkehrssituationen und deren zugehöriger Kritikalität hinterlegt, die beim Einlesen der Simulationsumgebung dem Situationsgenerator zur Verfügung steht. Dieser kann dann diese Liste sequentiell für jedes Hindernis abarbeiten, um den Testablauf durchzuführen. Dabei wird immer nur ein Hindernis zur Erzeugung einer kritischen Verkehrssituation aktiviert.

Die Steuerung der Hindernisse wird zwischen den Modulen Situationsgenerator und Verkehrssimulation durch die vereinbarte Schnittstelle in Abschnitt 4.2.2 koordiniert, wobei die in diesem Abschnitt als optional gekennzeichneten Fahrbahnkoordinaten dann zwingend von den beiden Simulationsmodulen berechnet und ausgewertet werden müssen.

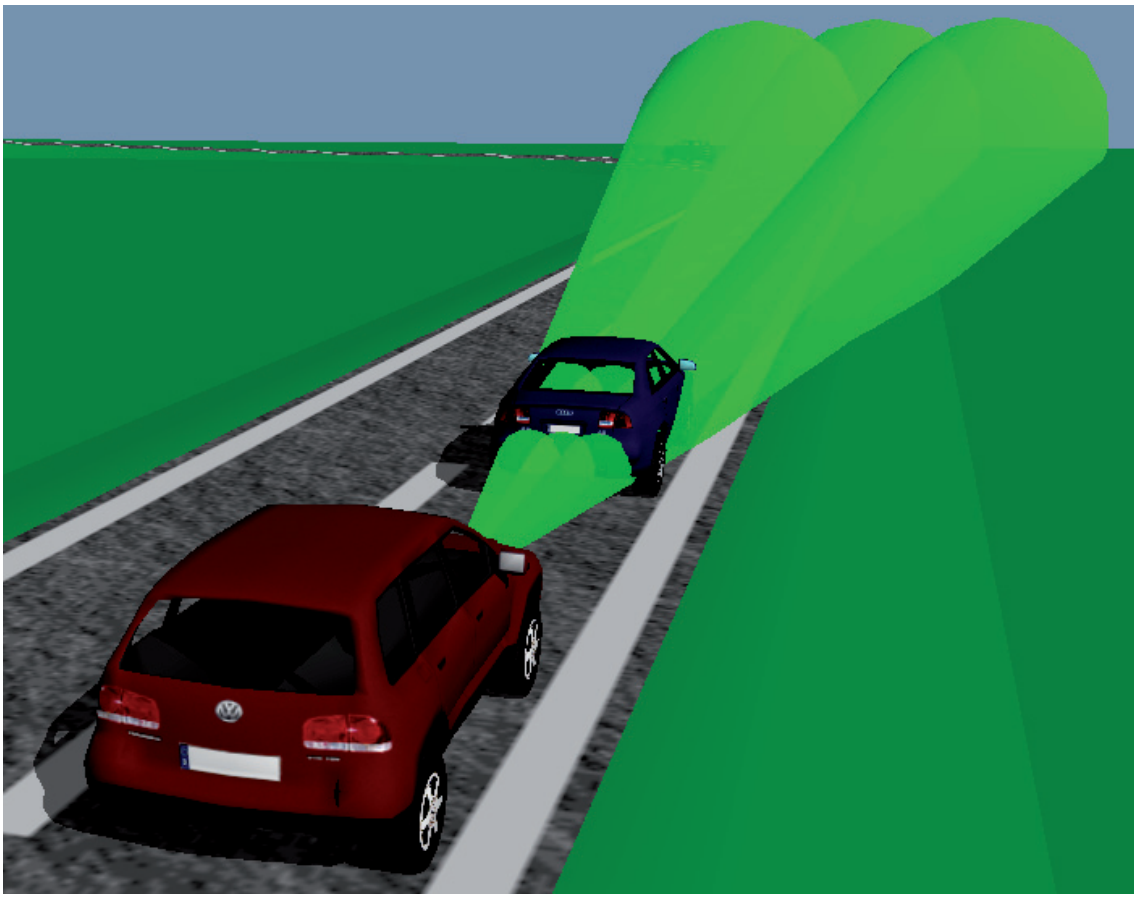

Abbildung 5.15.: Konfiguration der Ausrichtung dreier Radarsensoren.

Eine umfassende Beschreibung des Situationsgenerators und die Definition der Kritikalität ist in [Schmidt 2010] zu finden.

Abbildung 5.16 zeigt die Benutzeroberfläche *PRAETORIAs* im Software-in-the-Loop-Betrieb. Als Simulationsmodule sind ein Fahrerassistenzsystem, die Sensormodelle, der Situationsgenerator und die Verkehrssimulation geladen. Als kritische Situation wird gerade ein Spurwechsel durchgeführt. Damit ein Prüfingenieur eine direkte Rückwirkung zum durchgeführten Fahrmanöver erhält, wird der Spurwechsel zusätzlich durch Setzen der entsprechenden Blinker angezeigt.

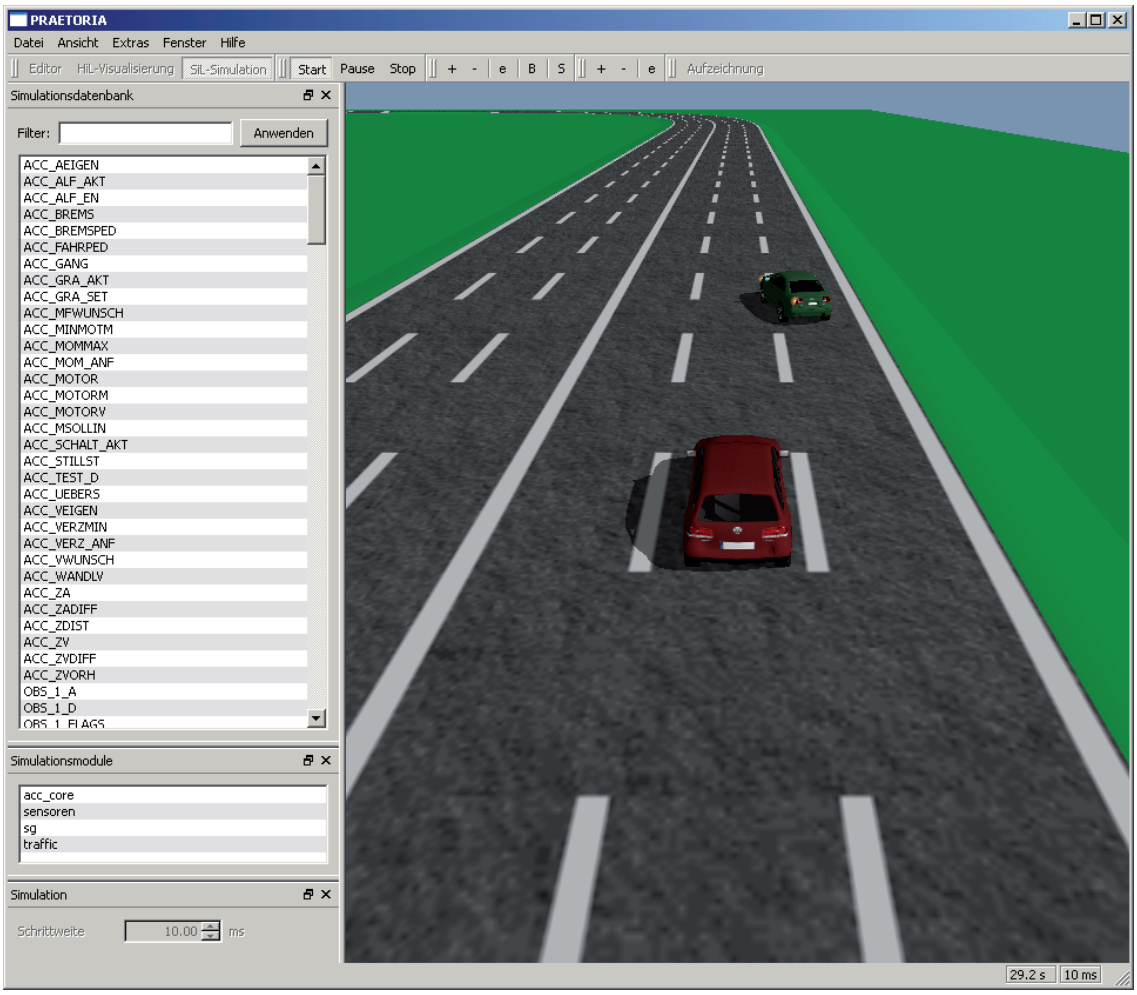

Abbildung 5.16.: Software-in-the-Loop-Simulation des Situationsgenerators, s. [Schmidt 2010].

6. Zusammenfassung

Die durch diese Arbeit entwickelte Simulationsumgebung hat bereits eine vielfältige Anwendung in einem Forschungsprojekt erfahren.

Dabei war vor allem das Werkzeug *PRAETORIA* von zentraler Bedeutung, mit dem bereits während der Entwicklungsphase die Interaktion der einzelnen Simulationsmodule getestet werden konnte. Eine nachfolgende Übertragung der Simulationsmodule auf den Echtzeitsimulator war durch die plattformübergreifende Natur der Simulationsumgebung problemlos möglich.

Zur Steuerung des Echtzeitsimulators und zur Visualisierung konnte ebenfalls auf *PRAETORIA* zurückgegriffen werden. Eine mit diesem Programm parametrierte Simulationsumgebung konnte direkt auf die verschiedenen Rechenknoten des Echtzeitsimulators übertragen werden, so daß alle Simulationsmodule über eine konsistente Konfiguration der Testfälle verfügten. Die Visualisierung eines Testablaufes findet dabei in Echtzeit zeitgleich zur Durchführung des Tests statt.

In den Kapiteln 2 und 3 wurden die Grundlagen der Simulationsumgebung zum echtzeitfähigen Test von Fahrerassistenzsystemen präsentiert.

Das in Kapitel 2 verwendete Straßenmodell von [Wang 2005] wurde dabei um eine kinematische Beschreibung des Fahrbahnkoordinatensystems erweitert und deren Anwendung in der Fahrdynamiksimulation aufgezeigt.

Die in Kapitel 3 vorgestellte Visualisierung der Umwelt bietet dabei das größte Potential zur Erweiterung. Zunächst sollte geprüft werden inwieweit sich die propriäteren Dateiformate der Fahrzeuge und des Rundkurses durch standardisierte Formate, wie sie z.B. in der Unterhaltungsindustrie eingesetzt werden, ersetzen lassen. Das Fahrzeugumfeld sollte um ein Relief und Wettereffekte erweitert werden, um eine ansprechendere Präsentation des Verkehrsgeschehens zu erzeugen.

Als zentraler Ausgangspunkt einer Testdurchführung kann die Simulationsumgebung aus Kapitel 4 betrachtet werden. Mit ihr wird der Testumfang und -ablauf, sowie dessen visuelle Darstellung konfiguriert. Durch die Unterteilung der verwendeten Daten in die Anwendungsbereiche *Visualisierung* und *Simulation* ist eine getrennte Erweiterung beider Teilgebiete möglich.

Eine erste Anwendung des Konzepts *Simulationsumgebung* wird in Kapitel 5 präsentiert. Die zur Durchführung des *DECOS*-Projektes entwickelten Werkzeuge bieten eine stabile Entwicklungsumgebung, die beim Test von Fahrerassistenzsystemen eingesetzt werden kann. Eine Anwendung der Entwicklungsumgebung beschränkt sich dabei durch die vorgestellte Co-Simulation nicht nur auf die Echtzeitsimulation, sondern kann bereits in früheren Entwicklungsstadien eines Assistenzsystems erfolgen.

A. Koordinatensysteme

Dieses Kapitel soll die Grundlagen zur Transformation von Vektoren durch Rotationsmatrizen erläutern und in die verwendete Namenskonvention dieses Textes einführen.

A.1. Verwendetes Koordinatensystem

Es wird das Koordinatensystem aus [FAKRA 1994] verwendet, um die beschriebene Simulationsumgebung für Anwender der Kraftfahrzeugindustrie leichter zugänglich zu machen.

Die x-Achse zeigt in Blickrichtung der Kamera bzw. in Fahrtrichtung eines Fahrzeuges. Die y-Achse ist dann nach links, die z-Achse nach oben gerichtet, Abbildung A.1.

Durch die Verwendung eines rechtshändigen Koordinatensystems werden auch die Vorzeichen der drei Rotationswinkel um die Elementarachsen festgelegt. Eine Rotation um die Hochachse mit dem Winkel ψ wird als Gieren, eine Rotation um die Querachse mit dem Winkel ϑ wird als Nicken und eine Rotation um die Längsachse mit dem Winkel φ wird als Wanken bezeichnet.

A.2. Vektorrotation und Koordinatentransformation

[Kuipers 2002] zeigt, daß eine Rotation durch zwei verschiedene Drehmatrizen dargestellt werden kann. Für eine Drehung um die z-Achse sind folgende Matrizen möglich, wobei ein positiver Drehwinkel im Sinne eines rechtshändigen Koordinatensystems angenommen wird.

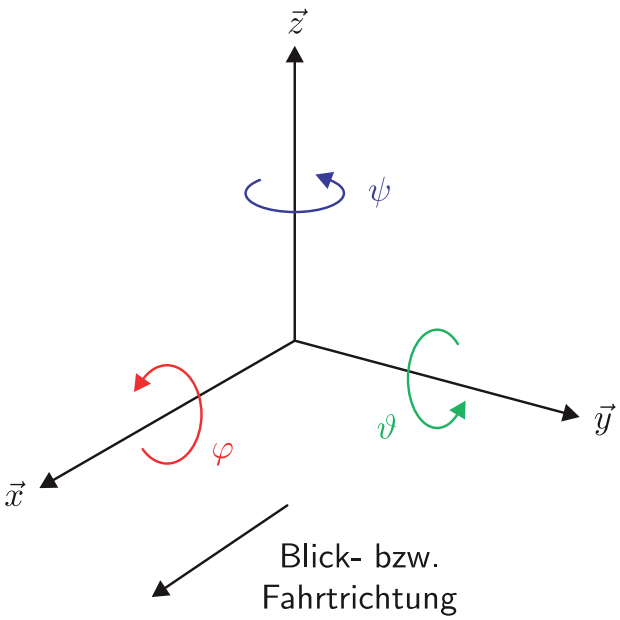

Abbildung A.1.: Verwendetes Koordinatensystem.

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_z(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Die Matrix $\mathbf{R}_z(\psi)$ rotiert dabei einen Vektor $\vec{v}$ um die z-Achse des Inertialsystems in den Vektor $\vec{w}$.

$$\vec{w} = \mathbf{R}_z(\psi) \cdot \vec{v} \tag{A.1}$$

Mit der Matrix $\mathbf{T}_z(\psi)$ hingegen wird eine Rotation der Vektorbasis beschrieben. Ein Vektor wird dann z.B. vom Inertialsystem I in ein System S transformiert. Seine Koordinaten sind dann bezüglich der Basisvektoren des Systems S angegeben.

$$_S\vec{v} = \mathbf{T}_{SI,z}(\psi) \cdot \vec{v} \tag{A.2}$$

Der Index SI der Matrix $\mathbf{T}_{SI,z}$ gibt dabei Ziel und Quelle der Koordinatentransformation an. Das Koordinatensystem, in dem ein Vektor dargestellt ist, wird durch einen vorangestellten Index gekennzeichnet; z.B. $_S\vec{v}$ für den Vektor $\vec{v}$ im Koordinatensystem S. Vektoren, die im Inertialsystem dargestellt sind werden nicht explizit gekennzeichnet; $_I\vec{v} \equiv \vec{v}$.

Werden alle Vektoren, eines gegenüber dem Inertialsystem I rotierten Koordinatensystems S, durch eine Rotation transformiert, wird ebenfalls auf die Schreibweise mit Quell- und Ziel-Indizes zurückgegriffen. Gleichung (A.1) lautet damit

$$\vec{v} = \mathbf{R}_{IS,z}(\psi) \cdot {}_S\vec{v}\,.$$

Um dasselbe Verhältnis zwischen resultierendem Vektor und Koordinatensystem sowohl mit der Matrix $\mathbf{R}_z(\psi)$ als auch mit der Matrix $\mathbf{T}_z(\psi)$ zu erhalten, gilt der Zusammenhang

$$\mathbf{R}_z(\psi) = \mathbf{T}_z(-\psi). \tag{A.3}$$

Die beiden möglichen Operationen werden gemäß ihren Funktionen Vektor- bzw. Basisrotation genannt, s. Abbildung A.2. In der Computergraphik kommt dabei meist die Vektorrotation zum Einsatz, da die zu visualisierende Szenerie nur um ein festes Kamerasystem rotiert werden kann. Basisrotationen kommen dafür überwiegend in der Luftfahrttechnologie zum Einsatz, da sie die Wirkung eines Bezugssystems, z.B. dem Inertialsystem, auf ein beobachtetes System abbilden können, z.B. die Wirkung der Erdbeschleunigung $\vec{g}$ auf ein Flugzeug.

Bietet ein Programmsystem nur eine der beiden Möglichkeiten an, so kann mit Gleichung (A.3) die jeweilige andere Darstellung erzeugt werden. Der gleiche Effekt lässt sich durch Transponieren der Matrix erzielen, s. Gleichung (A.4).

$$\mathbf{R}_z(\psi) = \mathbf{T}_z^T(\psi) \tag{A.4}$$

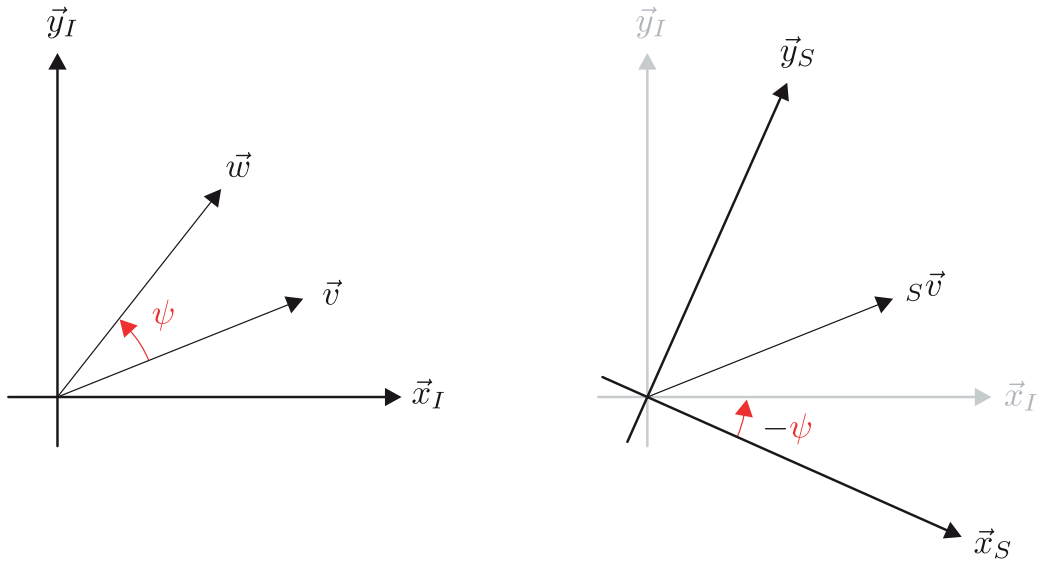

Abbildung A.2.: Das Verhältnis eines rotierten Vektors $\vec{w}$ bzw. ${}_S\vec{v}$ zu seiner Vektorbasis bleibt erhalten (links: Vektorrotation, rechts: Basisrotation).

Die hier vorgestellten Zusammenhänge gelten allgemein für Rotationsmatrizen, s. [Kuipers 2002].

A.3. Eigenschaften von Rotationsmatrizen

Rotationsmatrizen zählen zu den speziellen orthogonalen Matrizen im $\mathbb{R}^3$, der sog. Gruppe $\mathbf{SO}(3)$. Damit weisen sie folgende Eigenschaften auf. Die entsprechenden Beweise sind z.B. [Kuipers 2002] zu entnehmen. Diese Eigenschaften sind allgemein für jede Rotationsmatrix gültig.

Orthogonalität Orthogonale Matrizen sind durch die Eigenschaft

$$\mathbf{R}^{-1} = \mathbf{R}^T$$

definiert. Damit gilt

$$\mathbf{R} \cdot \mathbf{R}^{-1} = \mathbf{R} \cdot \mathbf{R}^T = \mathbf{I}. \tag{A.5}$$

D.h. die Anwendung einer Rotation gefolgt von der inversen Rotation lässt einen Vektor unverändert.

Vektorrotation		Basisrotation	
$\mathbf{R}_x(\varphi) =$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix}$	$\mathbf{T}_x(\varphi) =$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi \\ 0 & -\sin\varphi & \cos\varphi \end{bmatrix}$
$\mathbf{R}_y(\vartheta) =$	$\begin{bmatrix} \cos\vartheta & 0 & \sin\vartheta \\ 0 & 1 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta \end{bmatrix}$	$\mathbf{T}_y(\vartheta) =$	$\begin{bmatrix} \cos\vartheta & 0 & -\sin\vartheta \\ 0 & 1 & 0 \\ \sin\vartheta & 0 & \cos\vartheta \end{bmatrix}$
$\mathbf{R}_z(\psi) =$	$\begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\mathbf{T}_z(\psi) =$	$\begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Tabelle A.1.: Elementardrehungen.

Drehachse Da die Anwendung einer Rotation auf seine Drehachse diesen Vektor unverändert lässt, besitzt jede Rotationsmatrix einen Eigenwert $\lambda_d = +1$, der die Drehachse $\vec{d}$ definiert.

$$(\mathbf{R} - \lambda_d \mathbf{I})\,\vec{d} = \vec{0}$$

Drehwinkel Der Drehwinkel φ einer Rotation um die Drehachse $\vec{d}$ kann mit der Spur der Rotationsmatrix berechnet werden.

$$\operatorname{spur}\mathbf{R} = 1 + 2\cos\varphi$$

A.4. Elementardrehungen

Im Folgenden sollen die Elementardrehungen um die 3 Achsen eines Koordinatensystems sowohl als Vektor- als auch als Basisrotation angegeben werden, s. Tabelle A.1. Der nachfolgende Abschnitt präsentiert einige Anwendungen der Elementardrehungen.

A.5. Rotationssequenzen

Nach dem Euler-Theorem können zwei unabhängige Koordinatensysteme durch maximal drei Rotationen um die Elementarachsen ineinander überführt werden, wobei zwei aufeinanderfolgende Rotationen nicht um dieselbe Achse vorgenommen werden dürfen, s. [Kuipers 2002].

A.5.1. Vektorrotation

Eine in der Computergraphik häufig angewendete Rotationssequenz ist durch Gleichung (A.6) gegeben. Dabei ist die Reihenfolge der Elementardrehungen um ein ruhendes Koordinatensystem von rechts nach links zu lesen, d.h. zuerst wird eine Drehung um die z-Achse durchgeführt.

$$\begin{aligned}
\mathbf{R} &= \mathbf{R}_x(\varphi) \cdot \mathbf{R}_y(\vartheta) \cdot \mathbf{R}_z(\psi) \\
&= \begin{bmatrix} \cos\vartheta\cos\psi & -\cos\vartheta\sin\psi & \sin\vartheta \\ \begin{pmatrix}\cos\varphi\sin\psi \\ +\sin\varphi\sin\vartheta\cos\psi\end{pmatrix} & \begin{pmatrix}\cos\varphi\cos\psi \\ -\sin\varphi\sin\vartheta\sin\psi\end{pmatrix} & -\sin\varphi\cos\vartheta \\ \begin{pmatrix}\sin\varphi\sin\psi \\ -\cos\varphi\sin\vartheta\cos\psi\end{pmatrix} & \begin{pmatrix}\sin\varphi\cos\psi \\ +\cos\varphi\sin\vartheta\sin\psi\end{pmatrix} & \cos\varphi\cos\vartheta \end{bmatrix}
\end{aligned} \tag{A.6}$$

Die Drehwinkel der Elementardrehungen können mit Gleichung (A.7) zurückgewonnen werden. Für einen Drehwinkel von $\vartheta = \pm\frac{\pi}{2}$ werden die Gleichungen singulär.

$$\begin{aligned}
\varphi &= -\arctan\left(\frac{r_{12}}{r_{22}}\right) \\
\vartheta &= \arcsin(r_{02}) \\
\psi &= -\arctan\left(\frac{r_{01}}{r_{00}}\right)
\end{aligned} \tag{A.7}$$

A.5.2. Basisrotation

Die Rotationssequenz der Basisrotationen aus Gleichung (A.8) wird in [FAKRA 1994] zur Rotation des erdfesten Koordinatensystems in das bewegte Aufbaukoordinatensystem verwendet. Eine weitere Anwendung existiert in

der Luftfahrt, wobei zu beachten ist, daß die Orientierung der Koordinatenachsen eine andere als die hier gewählte ist.

$$
\begin{aligned}
\mathbf{T} &= \mathbf{T}_x(\varphi) \cdot \mathbf{T}_y(\vartheta) \cdot \mathbf{T}_z(\psi) \\
&= \begin{bmatrix} \cos\vartheta\cos\psi & \cos\vartheta\sin\psi & -\sin\vartheta \\ \begin{pmatrix}\sin\varphi\sin\vartheta\cos\psi \\ -\cos\varphi\sin\psi\end{pmatrix} & \begin{pmatrix}\cos\varphi\cos\psi \\ +\sin\varphi\sin\vartheta\sin\psi\end{pmatrix} & \sin\varphi\cos\vartheta \\ \begin{pmatrix}\sin\varphi\sin\psi \\ +\cos\varphi\sin\vartheta\cos\psi\end{pmatrix} & \begin{pmatrix}\cos\varphi\sin\vartheta\sin\psi \\ -\sin\varphi\cos\psi\end{pmatrix} & \cos\varphi\cos\vartheta \end{bmatrix}
\end{aligned}
\tag{A.8}
$$

Die Drehwinkel können ebenfalls durch Anwendung von Gleichung (A.9) ermittelt werden. Die Singularität ist dabei auch hier zu beachten.

$$
\begin{aligned}
\varphi &= \arctan\left(\frac{t_{12}}{t_{22}}\right) \\
\vartheta &= -\arcsin(t_{02}) \\
\psi &= \arctan\left(\frac{t_{01}}{t_{00}}\right)
\end{aligned}
\tag{A.9}
$$

Die Wirkung von Gleichung (A.8) zeigt Abbildung A.3.

Die entsprechende Rotationsmatrix erhält man durch Anwendung von Gleichung (A.4).

$$
\begin{aligned}
\mathbf{R} &= \mathbf{T}^T = (\mathbf{T}_x(\varphi) \cdot \mathbf{T}_y(\vartheta) \cdot \mathbf{T}_z(\psi))^T = \mathbf{T}_z^T(\psi) \cdot \mathbf{T}_y^T(\vartheta) \cdot \mathbf{T}_x^T(\varphi) \\
&= \mathbf{R}_z(\psi) \cdot \mathbf{R}_y(\vartheta) \cdot \mathbf{R}_x(\varphi) \\
&= \begin{bmatrix} \cos\vartheta\cos\psi & \begin{pmatrix}\sin\varphi\sin\vartheta\cos\psi \\ -\cos\varphi\sin\psi\end{pmatrix} & \begin{pmatrix}\sin\varphi\sin\psi \\ +\cos\varphi\sin\vartheta\cos\psi\end{pmatrix} \\ \cos\vartheta\sin\psi & \begin{pmatrix}\cos\varphi\cos\psi \\ +\sin\varphi\sin\vartheta\sin\psi\end{pmatrix} & \begin{pmatrix}\cos\varphi\sin\vartheta\sin\psi \\ -\sin\varphi\cos\psi\end{pmatrix} \\ -\sin\vartheta & \sin\varphi\cos\vartheta & \cos\varphi\cos\vartheta \end{bmatrix}
\end{aligned}
\tag{A.10}
$$

Die Bestimmung der Drehwinkel geschieht durch folgenden Zusammenhang.

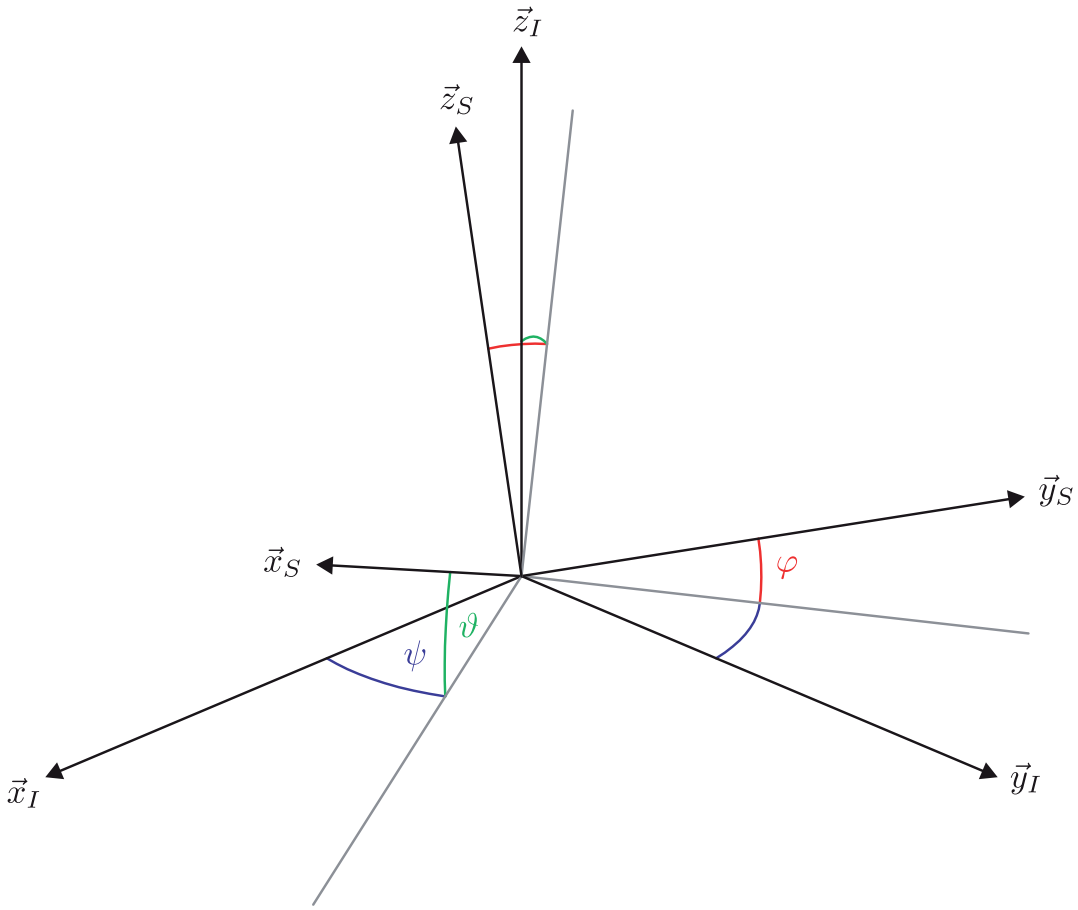

Abbildung A.3.: Basisrotation des Inertialsystems I in das System S.

$$
\begin{aligned}
\varphi &= \arctan\left(\frac{r_{21}}{r_{22}}\right) \\
\vartheta &= -\arcsin(r_{20}) \\
\psi &= \arctan\left(\frac{r_{10}}{r_{00}}\right)
\end{aligned}
\tag{A.11}
$$

A.6. Rotation um eine beliebige Achse

In [Akenine-Möller u. Haines 2002] wird eine Matrix zur Vektorrotation mit dem Winkel φ um die normierte Drehachse $\vec{d}$ angegeben, s. Gleichung (A.12).

$$
\begin{aligned}
c &= \cos\varphi \\
s &= \sin\varphi \\
\mathbf{R}(\vec{d},\varphi) &= \begin{bmatrix} c+(1-c)d_x^2 & (1-c)d_xd_y - sd_z & (1-c)d_xd_z + sd_y \\ (1-c)d_xd_y + sd_z & c+(1-c)d_y^2 & (1-c)d_yd_z - sd_x \\ (1-c)d_xd_z - sd_y & (1-c)d_yd_z + sd_x & c+(1-c)d_z^2 \end{bmatrix}
\end{aligned}
\tag{A.12}
$$

Mit dem Roberson-Operator $\tilde{a}$, der das Kreuzprodukt

$$
\vec{a} \times \vec{b} = \underbrace{\begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}}_{\tilde{a}} \cdot \vec{b} = \tilde{a} \cdot \vec{b}
$$

vermittelt und den Vektor $\vec{a}$ in die schiefsymmetrische Matrix[1] $\tilde{a}$ umwandelt, lässt sich Gleichung (A.12) wie folgt schreiben, s. [Schiehlen u. Eberhard 2004].

$$
\begin{aligned}
\mathbf{R}(\vec{d},\varphi) &= \vec{d}\cdot\vec{d}^T + (\mathbf{I} - \vec{d}\cdot\vec{d}^T)\cdot\cos\varphi + \tilde{d}\cdot\sin\varphi \\
&= \mathbf{I}\cdot\cos\varphi + \vec{d}\cdot\vec{d}^T\cdot(1-\cos\varphi) + \tilde{d}\cdot\sin\varphi
\end{aligned}
$$

[1] Für schiefsymmetrische Matrizen gilt $\mathbf{M} = -\mathbf{M}^T$.

A.7. Beschreibung der Rotation durch den Richtungskosinus

Die Darstellung der sog. Richtungskosinusmatrix orientiert sich an [Durham 2004]. Es soll untersucht werden, wie ein System S durch Rotation in das Inertialsystem I überführt werden kann.

Ein Vektor $\vec{v}$ habe im System S die Komponenten ${}_Sv_x$, ${}_Sv_y$ und ${}_Sv_z$. Durch die Rotation der beiden Koordinatensysteme zueinander tragen alle drei Vektorkomponenten im Systems S durch Überlagerung auf allen drei Achsen des Inertialsystems zu dem resultierenden Vektor $\vec{v}$ im Inertialsystem bei. Die Gewichtung der einzelnen Überlagerung wird durch die 9 Winkel aller 6 Achsen zueinander bestimmt.

$$\begin{aligned}
\vec{v} &= \begin{bmatrix} \cos\varphi_{\vec{x}_I\vec{x}_S} & \cos\varphi_{\vec{x}_I\vec{y}_S} & \cos\varphi_{\vec{x}_I\vec{z}_S} \\ \cos\varphi_{\vec{y}_I\vec{x}_S} & \cos\varphi_{\vec{y}_I\vec{y}_S} & \cos\varphi_{\vec{y}_I\vec{z}_S} \\ \cos\varphi_{\vec{z}_I\vec{x}_S} & \cos\varphi_{\vec{z}_I\vec{y}_S} & \cos\varphi_{\vec{z}_I\vec{z}_S} \end{bmatrix} \cdot \begin{pmatrix} {}_Sv_x \\ {}_Sv_y \\ {}_Sv_z \end{pmatrix} \\
&= \underbrace{\begin{bmatrix} \vec{x}_I \bullet \vec{x}_S & \vec{x}_I \bullet \vec{y}_S & \vec{x}_I \bullet \vec{z}_S \\ \vec{y}_I \bullet \vec{x}_S & \vec{y}_I \bullet \vec{y}_S & \vec{y}_I \bullet \vec{z}_S \\ \vec{z}_I \bullet \vec{x}_S & \vec{z}_I \bullet \vec{y}_S & \vec{z}_I \bullet \vec{z}_S \end{bmatrix}}_{\mathbf{R}_{IS}} \cdot {}_S\vec{v}
\end{aligned}$$

Damit ist die Richtungskosinusmatrix $\mathbf{R}_{IS}$ der Rotation bestimmt.

$$\mathbf{R}_{IS} = \begin{bmatrix} \vec{x}_I \bullet \vec{x}_S & \vec{x}_I \bullet \vec{y}_S & \vec{x}_I \bullet \vec{z}_S \\ \vec{y}_I \bullet \vec{x}_S & \vec{y}_I \bullet \vec{y}_S & \vec{y}_I \bullet \vec{z}_S \\ \vec{z}_I \bullet \vec{x}_S & \vec{z}_I \bullet \vec{y}_S & \vec{z}_I \bullet \vec{z}_S \end{bmatrix} \tag{A.13}$$

Die Richtungskosinusmatrix lässt sich ebenfalls für eine Basisrotation aufstellen. Mit Gleichung (A.4) gilt

$$\mathbf{T}_{SI} = \mathbf{R}_{IS}^T = \begin{bmatrix} \vec{x}_S \bullet \vec{x}_I & \vec{x}_S \bullet \vec{y}_I & \vec{x}_S \bullet \vec{z}_I \\ \vec{y}_S \bullet \vec{x}_I & \vec{y}_S \bullet \vec{y}_I & \vec{y}_S \bullet \vec{z}_I \\ \vec{z}_S \bullet \vec{x}_I & \vec{z}_S \bullet \vec{y}_I & \vec{z}_S \bullet \vec{z}_I \end{bmatrix}. \tag{A.14}$$

Abbildung A.4 veranschaulicht die Wirkung der Matrix $\mathbf{T}_{SI}$.

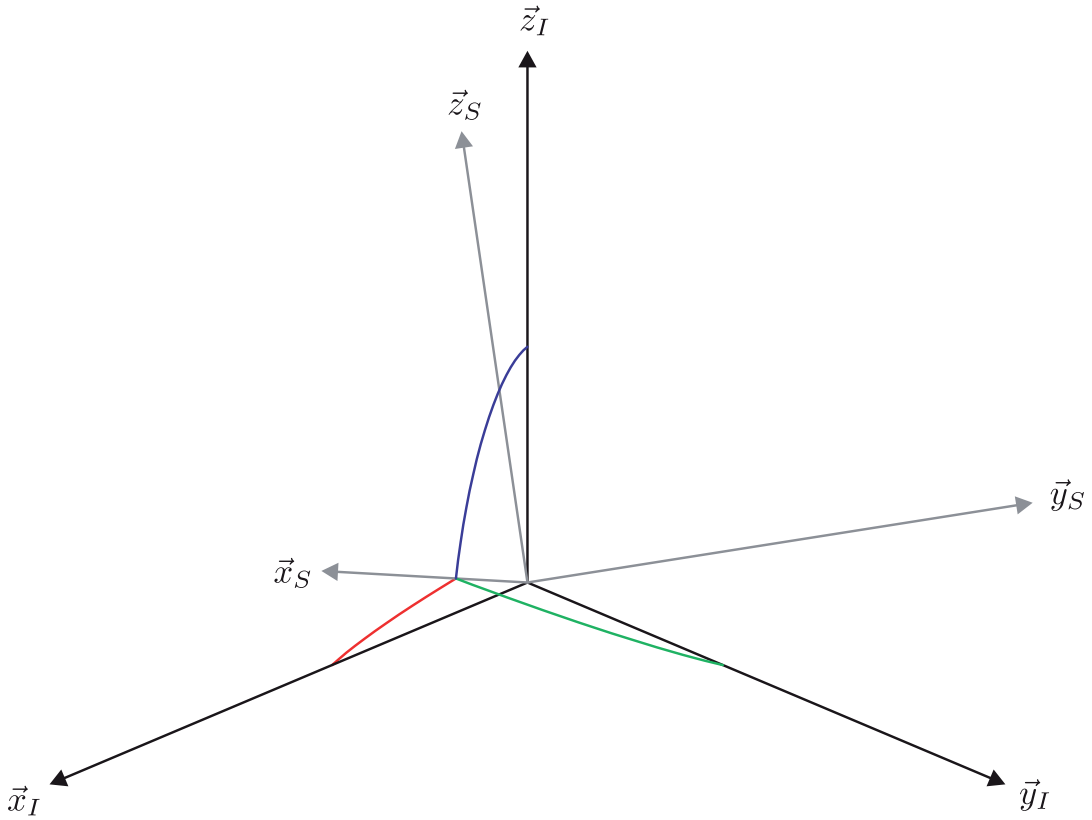

Abbildung A.4.: Richtungskosinus der x-Achse des Systems S zum Inertialsystem I.

Ist das Inertialsystem mit der standard Orthonormalbasis identisch, enthalten die Spalten und Zeilen der Matrizen $\mathbf{R}$ und $\mathbf{T}$ gerade die Komponenten der Basisvektoren des Systems S im Inertialsystem I.

$$\mathbf{R}_{IS} = \begin{bmatrix} x_{S,x} & y_{S,x} & z_{S,x} \\ x_{S,y} & y_{S,y} & z_{S,y} \\ x_{S,z} & y_{S,z} & z_{S,z} \end{bmatrix} \tag{A.15}$$

$$\mathbf{T}_{SI} = \begin{bmatrix} x_{S,x} & x_{S,y} & x_{S,z} \\ y_{S,x} & y_{S,y} & y_{S,z} \\ z_{S,x} & z_{S,y} & z_{S,z} \end{bmatrix} \tag{A.16}$$

Durch die Richtungskosinusmatrix können zwei beliebig zueinander orientierte Koordinatensysteme ineinander überführt werden sofern ihre Basisvektoren in der standard Orthonormalbasis dargestellt sind.

A.8. Beschreibung der Rotation durch Quaternionen

Die Quaternionen wurden von Sir William Rowan Hamilton[2] 1843 als Verallgemeinerung der Rotation in der Ebene mit komplexen Zahlen entdeckt, s. [Hamilton 1844]. Quaternionen sind Hyperkomplexe Zahlen im $\mathbb{R}^4$, die drei Imaginärteile aufweisen und somit Rotationen im $\mathbb{R}^3$ eindeutig beschreiben können.

Eine Einführung in die Anwendung der Quaternionen gibt [Kuipers 2002]. Ausführliche mathematische Beweise der hier aufgeführten Eigenschaften sind in [Dam u. a. 1998] ausgeführt.

A.8.1. Definition

Quaternionen bestehen aus einem Real- und drei Imaginärteilen und werden auch als Hamilton-Zahlen bezeichnet. Das Quaternion $\hat{q} \in \mathbb{H}$ ist wie folgt definiert.

$$\hat{q} \equiv q_w + q_x \cdot \vec{i} + q_y \cdot \vec{j} + q_z \cdot \vec{k} = q_w + \vec{q} = q_w + \begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} = [q_w, \vec{q}]$$

[2] 1805-1865

Die Vektoren $\vec{i}$, $\vec{j}$ und $\vec{k}$ bilden dabei die standard Orthonormalbasis und bilden die Imaginärteile des Quaternions. Sie unterliegen den Hamilton-Regeln.

$$\vec{i}^{\,2} = \vec{j}^{\,2} = \vec{k}^{\,2} = \vec{i} \cdot \vec{j} \cdot \vec{k} = -1$$

Häufig wird auch nur von einem Imaginärteil gesprochen, dem Vektor $\vec{q}$.

A.8.2. Eigenschaften

Im folgenden werden die wichtigsten Eigenschaften von Quaternionen aufgelistet.

Addition Die Addition bzw. Subtraktion erfolgt durch Addition der Einzelkomponenten.

$$\hat{q} = \hat{r} \pm \hat{s} = (r_w \pm s_w) + (\vec{r} \pm \vec{s})$$

Multiplikation Die Multiplikation zweier Quaternionen ist nicht kommutativ.

$$\hat{q} = \hat{r} \cdot \hat{s} = \underbrace{r_w \cdot s_w - \vec{r} \bullet \vec{s}}_{q_w} + \underbrace{r_w \cdot \vec{s} + s_w \cdot \vec{r} + \vec{r} \times \vec{s}}_{\vec{q}} \tag{A.17}$$

Konjugation Die Konjugation erfolgt ähnlich der der komplexen Zahlen.

$$\hat{q}^* = q_w - \vec{q}$$

Norm Die Cayley-Norm ist wie folgt definiert.

$$|| \, \hat{q} \, || = \hat{q} \cdot \hat{q}^* = q_w^2 + q_x^2 + q_y^2 + q_z^2$$

Durch die Norm werden die sog. Einheitsquaternionen definiert.

$$\hat{q} \in \mathbb{H}_1 \qquad \forall \qquad ||\,\hat{q}\,|| = 1 \tag{A.18}$$

Länge Die Länge eines Quaternions im $\mathbb{R}^4$ berechnet sich zu

$$|\,\hat{q}\,| = \sqrt{||\,\hat{q}\,||} = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}.$$

Inversion Die Inversion gestattet es durch ein Quaternion zu dividieren.

$$\hat{q}^{-1} = \frac{\hat{q}^*}{||\,\hat{q}\,||} \tag{A.19}$$

Für ein Einheitsquaternion nimmt das Inverse eine besonders einfache Gestalt an.

$$\hat{q}^{-1} = \hat{q}^*, \qquad \hat{q} \in \mathbb{H}_1$$

A.8.3. Rotation

[Kuipers 2002] leitet die Bedeutung eines Quaternions für die Rotation eines Vektors $\vec{v}$ über die Beziehung

$$\vec{w} = \hat{q} \cdot \vec{v} \cdot \hat{q}^{-1}, \qquad \hat{q} \in \mathbb{H}_1 \tag{A.20}$$

her, wobei die Regeln zur Multiplikation zweier Quaternionen beachtet werden müssen, s. Gleichung (A.17). Der Vektor $\vec{v}$ ist hier als Quaternion mit dem Realteil $v_w = 0$ zu verstehen.

Die Basisrotation eines Vektors $\vec{v}$ in das System S ist dann durch

$${}_S\vec{v} = \hat{q}^{-1} \cdot \vec{v} \cdot \hat{q}, \qquad \hat{q} \in \mathbb{H}_1 \tag{A.21}$$

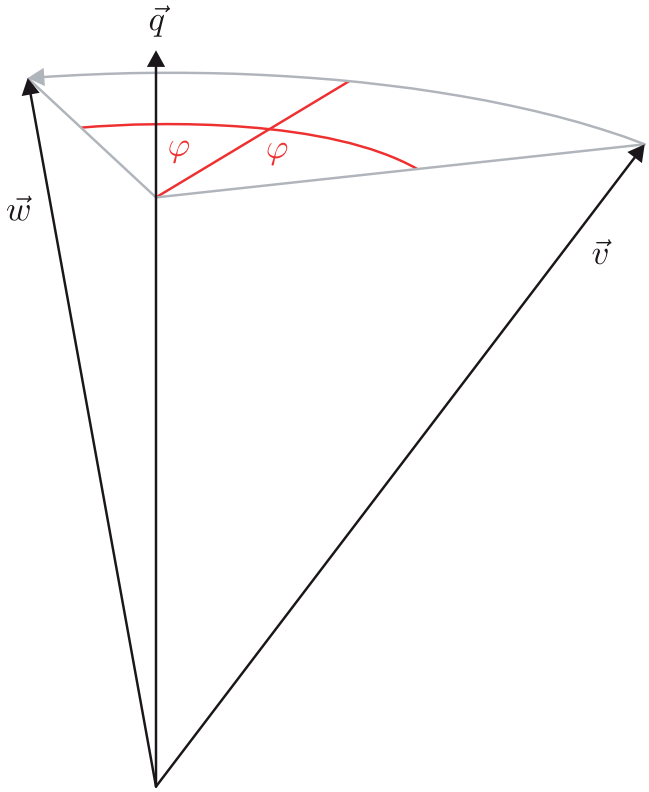

Abbildung A.5.: Rotation eines Vektors $\vec{v}$ durch das Quaternion $\hat{q}$.

gegeben, wobei berücksichtigt werden kann, daß für Einheitsquaternionen $\hat{q}^{-1} = \hat{q}^*$ gilt, s. Gleichung (A.19).

Das Einheitsquaternion muß dabei zu

$$\hat{q} = \cos\varphi + \sin\varphi \cdot \vec{q}, \qquad |\,\vec{q}\,| = 1$$

gewählt werden. Der Einheitsvektor $\vec{q}$ entspricht dann der Drehachse und der Winkel φ dem doppelten Drehwinkel um diese Achse, s. Abbildung A.5.

Die Transformationen aus Gleichung (A.20) und Gleichung (A.21) können ebenfalls als 3×3 Matrizen formuliert werden, s. Gleichung (A.22) und Gleichung (A.23), um die Rotation mit Quaternionen ohne Gleichung (A.17) durchzuführen. Üblicherweise werden Verkettungen von Rotationen mit Quaternionen berechnet, da die Multiplikation aus Gleichung (A.17) effektiver ist als die direkte Multiplikation zweier 3×3 Matrizen. Die Umwandlung in Rotationsmatrizen erfolgt erst unmittelbar vor der Transformation von Vek-

	Vektorrotation	Basisrotation
Transformation	$\vec{w} = \hat{q} \cdot \vec{v} \cdot \hat{q}^*$	$\vec{v}\,' = \hat{q}^* \cdot \vec{v} \cdot \hat{q}$
Verkettung	$\vec{w} = (\hat{q}_2 \cdot \hat{q}_1) \cdot \vec{v} \cdot (\hat{q}_2 \cdot \hat{q}_1)^*$	$\vec{v}\,' = (\hat{q}_1 \cdot \hat{q}_2)^* \cdot \vec{v} \cdot (\hat{q}_1 \cdot \hat{q}_2)$
Matrix	$\mathbf{R}$	$\mathbf{T}$

Tabelle A.2.: Zusammenhänge bei der Rotation mit Quaternionen.

toren. Im Gegensatz zu den Matrizen aus Abschnitt A.5 sind Quaternionen singularitätsfrei, s. [Kuipers 2002].

$$\mathbf{R} = \begin{bmatrix} 2(q_x^2 + q_w^2) - 1 & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 2(q_y^2 + q_w^2) - 1 & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 2(q_z^2 + q_w^2) - 1 \end{bmatrix} \tag{A.22}$$

$$\mathbf{T} = \mathbf{R}^T \tag{A.23}$$

Oftmals ist es wünschenswert die Rotationsbeschreibung eines Quaternions aus einer Matrix, z.B. der Richtungskosinusmatrix, s. Abschnitt A.7, zu gewinnen. [Shoemake u. Eberly 2004] präsentieren dazu einen stabilen Algorithmus, der aus den Einzelelementen einer Rotationsmatrix

$$\mathbf{R} = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}$$

durch Vergleich mit der Matrix aus Gleichung (A.22) das Einheitsquaternion $\hat{q}$ bestimmt. Dazu wird zunächst in der Hauptdiagonalen das größte Element des Quaternions $\hat{q}$ bestimmt. Durch gezielte Addition der Elemente in den Nebendiagonalen und Division durch das aus der Hauptdiagonalen bestimmte Element erhält man dann das vollständige Quaternion $\hat{q}$, s. Algorithmus 3. Die Wahl des Vorzeichens für q_w in Zeile 2 ist beliebig, da die Quaternionen $\hat{q}$ und $-\hat{q}$ dieselbe Rotation repräsentieren.

Tabelle A.2 stellt die beiden Transformationen nochmals gegenüber und zeigt die Regeln zur Verkettung von Rotationen mit Quaternionen. Dabei wird die Rotation $\hat{q}_1$ gefolgt von der Rotation $\hat{q}_2$ durchgeführt.

Algorithmus 3 Berechnung des Quaternions $\hat{q}$ aus der Matrix $\mathbf{R}$.

1: **if** $\operatorname{spur}\mathbf{R} \geq 0$ **then** ▷ $|\, q_w \,| \geq \frac{1}{2}$, d.h. q_w dominiert
2: $\quad q_w := \pm\frac{1}{2}\sqrt{\operatorname{spur}\mathbf{R} + 1}$
3: $\quad q_x := \frac{r_{21}-r_{12}}{4q_w}$
4: $\quad q_y := \frac{r_{02}-r_{20}}{4q_w}$
5: $\quad q_z := \frac{r_{10}-r_{01}}{4q_w}$
6: **else if** $r_{00} \geq r_{11} \wedge r_{00} \geq r_{22}$ **then** ▷ q_x dominiert
7: $\quad t := \sqrt{r_{00} - r_{11} - r_{22} + 1}$
8: $\quad q_x := \frac{1}{2}t$
9: $\quad q_y := \frac{r_{10}+r_{01}}{2t}$
10: $\quad q_z := \frac{r_{02}+r_{20}}{2t}$
11: $\quad q_w := \frac{r_{21}-r_{12}}{2t}$
12: **else if** $r_{11} \geq r_{00} \wedge r_{11} \geq r_{22}$ **then** ▷ q_y dominiert
13: $\quad t := \sqrt{r_{11} - r_{00} - r_{22} + 1}$
14: $\quad q_y := \frac{1}{2}t$
15: $\quad q_x := \frac{r_{10}+r_{01}}{2t}$
16: $\quad q_z := \frac{r_{21}+r_{12}}{2t}$
17: $\quad q_w := \frac{r_{02}-r_{20}}{2t}$
18: **else if** $r_{22} \geq r_{00} \wedge r_{22} \geq r_{11}$ **then** ▷ q_z dominiert
19: $\quad t := \sqrt{r_{22} - r_{00} - r_{11} + 1}$
20: $\quad q_z := \frac{1}{2}t$
21: $\quad q_x := \frac{r_{02}+r_{20}}{2t}$
22: $\quad q_y := \frac{r_{21}+r_{12}}{2t}$
23: $\quad q_w := \frac{r_{10}-r_{01}}{2t}$
24: **end if**

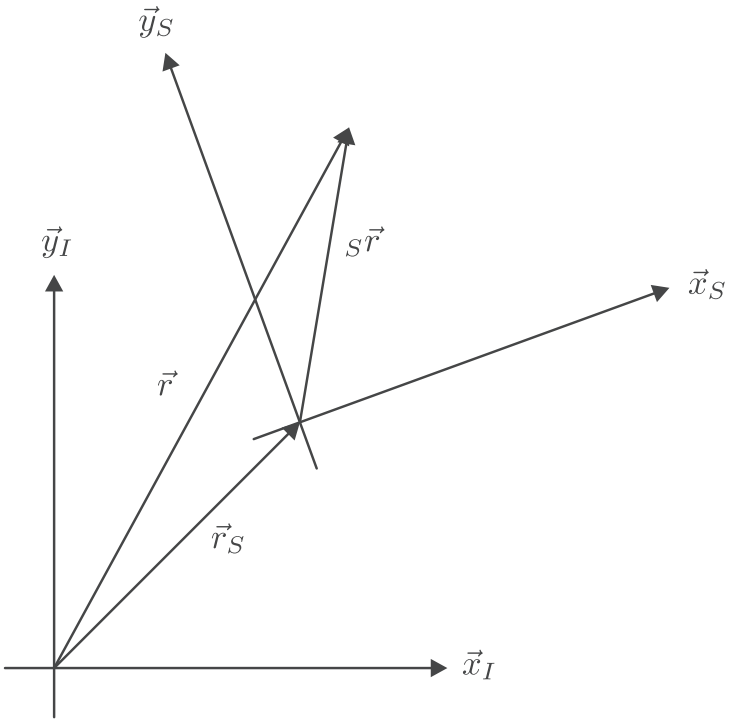

Abbildung A.6.: Zur Starrkörpertransformation.

A.9. Starrkörpertransformation

Betrachtet wird ein Vektor $\vec{r}$, der im Koordinatensystem S dargestellt ist, s. Abbildung A.6.

Im Inertialsystem I kann der Vektor $\vec{r}$ durch

$$\vec{r} = \mathbf{R}_{IS} \cdot {}_S\vec{r} + \vec{r}_S \tag{A.24}$$

beschrieben werden, wobei wieder auf eine spezielle Kennzeichnung der Vektoren im Inertialsystem verzichtet wurde. $\vec{r}_S$ beschreibt die Verschiebung des Koordinatensystems S zum Inertialsystem I und darf nicht mit dem Vektor ${}_S\vec{r}$ verwechselt werden, der den Vektor $\vec{r}$ bezüglich des Koordinatensystems S beschreibt. In der Computergraphik wird diese Transformation auch als Starrkörpertransformation bezeichnet, da sie in der Physik zur Beschreibung der Orientierung eines starren Körpers angewendet wird, s. [Schiehlen u. Eberhard 2004].

Die Starrkörpertransformation wird in der Computergraphik durch Übergang auf homogene Koordinaten beschrieben, s. [Akenine-Möller u. Haines 2002], wobei zuerst eine Rotation $\mathbf{R}$ gefolgt von einer Translation $\vec{t}$ ausgeführt wird. Die dazu benötigte 4×4-Matrix ist der Funktionswert der Funktion $\xi(\mathbf{R}, \vec{t})$. Für beliebige 4×4-Matrizen wird die Variable ξ ohne Funktionsargumente verwendet.

$$\begin{aligned}
\xi(\mathbf{R}, \vec{t}) &= \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{00} & r_{01} & r_{02} & 0 \\ r_{10} & r_{11} & r_{12} & 0 \\ r_{20} & r_{21} & r_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Vektoren müssen dann durch eine homogene Komponente w ergänzt werden, die nicht mit dem Realteil eines Quaternions verwechselt werden darf, s. Abschnitt A.8.1. Mit der homogenen Komponente wird angezeigt, ob ein Vektor einen Ortsvektor ($w = 1$) oder einen Richtungsvektor ($w = 0$) darstellen soll. Gleichung (A.24) kann dann durch

$$\begin{pmatrix} r_x \\ r_y \\ r_z \\ 1 \end{pmatrix} = \xi(\mathbf{R}_{IS}, \vec{r}_S) \cdot \begin{pmatrix} {}_S r_x \\ {}_S r_y \\ {}_S r_z \\ 1 \end{pmatrix}$$

ausgedrückt werden.

Die inverse Starrkörpertransformation ist durch

$$\xi^{-1}(\mathbf{R}, \vec{t}) = \xi(\mathbf{R}^T, -\mathbf{R}^T \cdot \vec{t})$$

gegeben. Für die Verkettung der Transformation ξ_1 gefolgt von der Transformation ξ_2 folgt

$$\xi_2(\mathbf{R}_2, \vec{t}_2) \cdot \xi_1(\mathbf{R}_1, \vec{t}_1) = \xi_{21}(\mathbf{R}_2 \cdot \mathbf{R}_1, \mathbf{R}_2 \cdot \vec{t}_1 + \vec{t}_2)\,.$$

Durch Verkettung von Starrkörpertransformationen können fortlaufende Wechsel des Koordinatensystems effektiv zusammengefasst werden. Zu beachten bleibt, daß alle Matrizen ξ einen homogenen Vektor berechnen, der 4 Komponenten aufweist. Richtungsvektoren werden durch die Translation nicht beeinflusst ($w = 0$). Ortsvektoren müssen eine homogene Komponente $w = 1$ besitzen, um einen Punkt im $\mathbb{R}^3$ zu beschreiben. Dies kann durch Division aller Komponenten durch die w-Komponente erreicht werden.

Die Starrkörpertransformation kann durch drei rotatorische und drei translatorische Koordinaten beschrieben werden, s. [Schiehlen u. Eberhard 2004]. Zur Beschreibung der Rotation können z.B. die Winkel aus Abschnitt A.5 verwendet werden. Mit der Bedingung aus Gleichung (A.18) können ebenfalls drei Komponenten eines Quaternions zur Beschreibung der Rotation verwendet werden. Eine ähnliche Aussage kann mit Gleichung (A.5) für drei Komponenten der Richtungskosinusmatrix getroffen werden.

Literaturverzeichnis

[Abrash 1997] Abrash, Michael: *Michael Abrash's Graphics Programming Black Book Special Edition*. Coriolis Group Books, 1997. – ISBN 1–576–10174–6

[Ahn 2008] Ahn, Song H.: *OpenGL Projection Matrix*. `http://www.songho.ca/opengl/gl_projectionmatrix.html`, 2008

[Akenine-Möller u. Haines 2002] Akenine-Möller, Tomas ; Haines, Eric: *Real-Time Rendering*. 2nd Edition. A K Peters, Ltd., 2002. – ISBN 1–56881–182–9

[Bock u. a. 2005] Bock, T. ; Siedersberger, K.-H. ; Zavrel, M. ; Breu, A. ; Maurer, M.: Simulations- und Testumgebung für Fahrerassistenzsysteme – Vehicle in the Loop (ViL), 2005 (VDI-Berichte 1900)

[Bronstein u. a. 2005] Bronstein, I.N. ; Semendjajew, K.A. ; Musiol, G. ; Mühlig, H.: *Taschenbuch der Mathematik*. 6. Auflage. Verlag Harri Deutsch, 2005. – ISBN 3–8171–2006–0

[Dam u. a. 1998] Dam, Erik B. ; Koch, Martin ; Lillholm, Martin: Quaternions, Interpolation and Animation / University of Copenhagen. 1998. – Forschungsbericht

[Durham 2004] Durham, Wayne: *Aircraft Dynamics & Control*. Aerospace & Ocean Engineering, Virgiana Tech, 215 Randolph Hall, Blacksburg, VA 24061 : `http://www.aoe.vt.edu/~durham/AOE5214/`, August 2004

[Everitt u. Kilgard 2002] Everitt, Cass ; Kilgard, Mark J.: *Practical and Robust Shadow Volumes for Hardware-Accelerated Rendering*. `http://developer.nvidia.com/`, März 2002

[FAKRA 1994] FAKRA, Normenausschuß Kraftfahrzeuge: DIN 70000, Straßenfahrzeuge – Fahrzeugdynamik und Fahrverhalten – Begriffe / Deutsches Institut für Normung e.V. 1994. – dt. Übersetzung der ISO 8855 Ausgabe 1991, modifiziert

[Foley u. a. 1992] Foley, James D. ; Dam, Andires van ; Feiner, Steven K. ; Hughes, John F.: *Computer Graphics – Principles and Practice.* 2nd Edition. Addison-Wesley Publishing Company, 1992. – ISBN 0–201–12110–7

[Gräbel 2007] Gräbel, Patrick: *Entwicklung und Implementierung eines Co-Simulationsansatzes für eine verteilte Software-in-the-Loop Simulationsumgebung*, Universität Kassel, Diplomarbeit, Juli 2007

[Hamilton 1844] Hamilton, William R.: On Quaternions, or on a New System of Imaginaries in Algebra. In: *Philosophical Magazine.* 1844-1850

[Holler 2005] Holler, Dominik: *Modellierung dreidimensionaler Straßenverläufe zum Einsatz in Echtzeitsimulationssystemen*, Universität Kassel, Diplomarbeit, 2005

[Kopetz u. a. 2004] Kopetz, H. ; Obermaisser, R. ; Peti, P. ; Suri, N.: *From a Federated to an Integrated Architecture for Dependable Real-Time Embedded Systems.* `http://www.decos.at/`, August 2004

[Kuipers 2002] Kuipers, Jack B.: *Quaternions and rotation sequences.* Princeton University Press, 2002. – ISBN 0–691–10298–8

[Meyer 2006] Meyer, Michael: *Erstellung eines Knotenrechners für ein Echtzeitnetzwerk zur Umgebungssimulation für Fahrerassistenzsysteme*, Universität Kassel, Diplomarbeit, 2006

[Press u. a. 2007] Press, William H. ; Teukolsky, Saul A. ; Vetterling, William T. ; Flannery, Brian P.: *Numerical Recipes – The Art of Scientific Computing.* 3rd Edition. Cambridge University Press, 2007. – ISBN 0–521–88068–8

[Rost 2006] Rost, Randi J.: *OpenGL Shading Language.* 2nd Edition. Addison-Wesley – Pearson Education, 2006. – ISBN 0–321–33489–2

[Schiehlen u. Eberhard 2004] Schiehlen, Werner ; Eberhard, Peter: *Technische Dynamik.* B. G. Teubner Verlag, 2004. – ISBN 3–519–12365–7

[Schmidt 2010] Schmidt, Christian O.: *Hardware-in-the-Loop-gestützte Entwicklungsplattform für Fahrerassistenzsysteme – Analyse und Generierung kritischer Verkehrsszenarien*, Universität Kassel, Diss., 2010

[Shankel u. Treglia 2002] *Kapitel* 4.2 Fast Heightfield Normal Calculation. In: Shankel, Jason ; Treglia, Dante: *Game Programming Gems 3*. 2002. – ISBN 1–58450–233–9, S. 344–348

[Shoemake u. Eberly 2004] *Kapitel* 10.4 From Rotation Matrices to Quaternions. In: Shoemake, Ken ; Eberly, David H.: *Game Physics*. 2004. – ISBN 1–55860–740–4, S. 522–538

[Shreiner u. a. 2006] Shreiner, Dave ; Woo, Mason ; Neider, Jackie ; Davis, Tom: *OpenGL Programming Guide*. 5th Edition. Addison-Wesley – Pearson Education, 2006. – ISBN 0–321–33573–2

[Tellmann 2011] Tellmann, Dirk: *Hardware-in-the-Loop-gestützte Entwicklungsplattform für Fahrerassistenzsysteme – Modelle der Umfeldsensorik und angepasste Fahrermodelle*, Universität Kassel, Diss., 2011

[Verburg u. a. 2002] Verburg, D.J. ; Knaap, A.C.M. v. d. ; Ploeg, J.: VeHiL – Developing and Testing Intelligent Vehicles, 2002 (Proceedings IEEE Intelligent Vehicle Symposium)

[Wang 2005] Wang, Hongling: *Efficient Roadway Modeling and Behaviour Control for Real-Time Simulation*, University of Iowa, Diss., Mai 2005

[Woermann 1994] Woermann, Rolf: *Ein Beitrag zur Echtzeitsimulation technischer Systeme hoher Dynamik mit diskreten Modellen*, Universität Kassel, Diss., 1994