DIPLOMARBEIT

Simulation von städtischem Verkehr

Alternativer Titel: A Multi-Agent Micro Simulation of Multiphase Urban Traffic (MAMSMUT)

Ausgeführt am Institut für

Computergraphik und Algorithmen,

Abteilung Computergraphik

der Technischen Universität Wien

unter der Leitung von

Univ.Prof. DI Dr.techn. **Werner Purgathofer**

Univ.Ass. DI DI Dr.techn. **Michael Wimmer**

durch

Gregor Lehninger

Vinzenz-Muchitsch-Straße 34a/60

8020 Graz

_____ _____

Datum Unterschrift

Kurzfassung

Die vorliegende Diplomarbeit thematisiert Simulation von städtischem Verkehr.

Im Rahmen dieser Diplomarbeit wurde die Bibliothek TrafLib (*Tra*ffic *Lib*rary) entwickelt. TrafLib ist eine multi-agent Mikro-Simulation für städtischen Verkehr. Die Bibliothek kann sehr flexibel in Visualisierungssoftware eingesetzt werden. Z.B. Städtevisualisierungs-software, Fahrsimulatoren, Computerspiele, Virtual Reality Software usw. können ansprechender und lebendiger gestaltet werden.

TrafLib benötigt ein Straßennetzwerk in Form einer Straßennetz-Beschreibungsdatei, wobei wahlweise das TrafLib-Format oder das Format von VRMG, einem Programm zur Erstellung der Geometrie großer Straßennetzwerke, verwendet werden kann. Eine Straßennetzwerk-Beschreibungsdatei besteht aus Straßen und Kreuzungen und optional Ampeln und Verkehrszeichen.

Das Problem der Verkehrsberechnung wird durch die Berücksichtigung von Verkehrsregeln laut Straßenverkehrsordnung und durch das Hintereinanderfahren von Fahrzeugen („car–following“) gelöst. Die Fahrzeuge sind als autonome agents modelliert. Das individuelle, menschliche Fahrverhalten wurde durch die Implementierung von 3 verschiedenen Charaktereigenschaften (vorsichtig, normal, aggressiv) nachgebildet.

Die Bewegung der Fahrzeuge findet auf eindimensionalen Fahrspuren statt, wodurch die Flexibilität schwach, aber der Rechenaufwand stark reduziert wird.

TrafLib ist fähig, etwa 5000 Fahrzeuge auf einem 3x3 km großen städtischen Straßennetzwerk auf einem herkömmlichen PC in Echtzeit zu simulieren.

Abstract

The task of this thesis is the simulation of urban traffic.

For this purpose the traffic library TrafLib was developed. TrafLib is a multi-agent, micro simulation for urban traffic. It is suitable for visualisation, e.g. urban visualisation software, driving simulators, computer games, virtual reality software, and so on. It enhances them and makes them more attractive.

TrafLib requires a road-network in the form of an input file. The input file must contain the description of roads and crossings, and optionally traffic lights and traffic signs. Additionally to the own file description format, the format of VRMG is also supported. VRMG is a program for creating geometry of large road-networks.

The problem of traffic calculation is solved by handling traffic rules according to the Highway Code and the “Car-Following-Theory”. The vehicles are modelled as autonomous agents. Human characteristics are imitated by three different types of driving-behaviour (cautious, normal, aggressive).

The movement of the vehicles takes place on 1-dimensional lanes. This reduces the flexibility a little, but simplifies the calculation a lot.

TrafLib is able to simulate more than 5000 vehicles on a road network of the size of 3 times 3 km in real-time on a standard-PC.

Inhaltsverzeichnis

1 Einleitung

Grund für die Entwicklung von Programmen zur Unterstützung von Verkehrsplanern ist das immer höher werdende Verkehrsaufkommen, das man seit dem Ende des 2. Weltkrieges beobachten kann [21]. Das Basiswerk „The Theory of Road Traffic Flow“ [21] wurde bereits 1966 veröffentlicht. Dem höheren Verkehrsaufkommen kann man durch 2 Strategien gerecht werden, nämlich durch neue Straßen und die Regelung des Verkehrsflusses. Beides ist natürlich nur begrenzt möglich.

Ein theoretischer Ansatz zur Ermittlung des Verkehrsflusses ist die „Follow-the-Leader-Theory“, oder kürzer auch „Car-Following“ genannt [21]. Diese ersten Ansätze betrachteten nur jeweils eine Fahrspur mit dichtem Verkehr ohne Überholmanöver. Die Dynamik die durch Starten und Stoppen z.B. bei einer Ampel entsteht, ist nicht einfach zu berechnen. Vor einer Roten Ampel stehen Fahrzeuge mit einem sehr geringen Abstand zueinander. Wenn die Ampel auf grün umschaltet, fährt zunächst das erste Fahrzeug los, dann das zweite, usw. bis zur nächsten Roten Ampel, wo wieder das erste zuerst stehen bleiben muss. Durch dieses „stop&go“ entsteht eine wellenartige Bewegung. Solche Kolonnen wurden später als „platoons“ bezeichnet [5, 11, 12, 33, 34].

Thema dieser Diplomarbeit ist die Simulation von städtischem Verkehr für Echtzeit-Visualisierungssoftware.

Der Hauptteil der gestellten Aufgabe umfasst die Entwicklung der Bibliothek „TrafLib“ (*Tra*ffic *Lib*rary). Die Funktionsweise von TrafLib besteht im Einlesen eines Straßennetzwerkes (d.h. Straßen, Kreuzungen, Verkehrstafeln und Ampeln), dem Platzieren von Fahrzeugen sowie dem Simulieren des Verkehrs unter Einhaltung von Regeln der Straßenverkehrsordnung.

Das Einlesen des Straßennetzes und die Berechnung der Fahrzeugbewegungen sind als Bibliothek ohne Grafik implementiert, sodass sie in Visualisierungssysteme eingebunden werden können. TrafLib ist insbesondere ausgelegt auf Zusammenarbeit mit URBANVIZ (Urban Visualization), einem Projekt für „Real-Time Rendering of Urban Environments“ von Univ. Ass. DI DI Dr. techn. Michael Wimmer und Dr. Peter Wonka, entwickelt am Institut für Computergraphik an der Technischen Universität Wien unter der Leitung von Univ. Prof. DI Dr. techn. Werner Purgathofer.

Des Weiteren gehören zu dieser Diplomarbeit die Entwicklung von Fahrzeug-Modellen und ein einfach gehaltenes Programm zur Demonstration von TrafLib.

Ein Großteil der Verkehrs-Simulationsprogramme wurde zur Unterstützung von Verkehrsplanung entwickelt. Der Anwendungsbereich von TrafLib liegt in der Einbindung in Visualisierungssoftware. Dadurch ergeben sich folgende Unterschiede:

	SW für Verkehrsplanung	**SW für Visualisierung (TrafLib)**
Simulationsmodell	makroskopisch, mikroskopisch, Mischform	mikroskopisch
Realitätsbezug	Stark	gering
Größe des simulierten Ausschnittes	Klein, meist nur eine Kreuzung	Stadtausschnitte mit 3x3 km und größer

Beobachtungsziel	Speziell (zum Beispiel eine Kreuzung, ein Autobahnabschnitt, 2 verbundene Ampelanlagen)	gesamtes Straßennetzwerk

Tabelle 1: Simulation für Verkehrsplanung bzw. Visualisierung

Man unterscheidet grundsätzlich zwei Sichtweisen von Verkehrssimulationsmodellen: die mikroskopische und die makroskopische [8]. Bei einer mikroskopischen Simulation wird das Verhalten jedes einzelnen Fahrzeuges simuliert; der Verkehr ergibt sich somit aus dem kollektiven Verhalten aller Fahrzeuge. Bei makroskopischen Modellen sind Aspekte wie die Durchschnittsgeschwindigkeit aller Fahrzeuge auf einer Straße, die Verkehrsdichte und der Verkehrsfluss von Belang.

Verkehrssimulationen, die der Verkehrsplanung dienen, verwenden oft eine Mischform aus Mikro- und Makrosimulation [11, 5, 34]. Das bedeutet, dass zum Beispiel nur eine Kreuzung simuliert wird und die Anzahl der Autos die auf den Straßen Richtung Kreuzung fahren, wird durch makroskopische Formeln bestimmt. Die Bewegung der Fahrzeuge über die Kreuzung wird als mikroskopische Simulation ausgeführt. Durch Veränderung verschiedener Parameter kann man dadurch zum Beispiel den optimalen Phasen-Zyklus für eine Ampelschaltung herausfinden.

Im Gegensatz dazu hat TrafLib die Aufgabe, Fahrzeuge für Visualisierungssoftware auf einem großen Straßennetz mit vielen Kreuzungen gleichzeitig zu bewegen.

Ein wesentlicher Aspekt für Verkehrssimulationen in der Verkehrsplanung ist, dass sie möglichst nahe an die Realität herankommen. TrafLib bildet jedoch die Realität nur so weit nach, dass ein „optisch guter“ Eindruck entsteht.

Einsatz findet TrafLib bereits in der Software Todo welche von todo an der Technischen Universtität todo für IBM entwickelt wurde. Diese Software wurde mit einer Physik-Engine ausgestattet, um Fahrzeug-Kollisionen und andere physikalische Effekte zu simulieren. Zu diesem Zweck wurden etliche Funktionen für den Eingriff in die Bewegung der Fahrzeuge dem Interface von TrafLib hinzugefügt.

Ein weiteres Tool mit dem TrafLib zusammen arbeitet ist VRMG (Virtual Road todo Generator). Diese Software wurde entwickelt, um Straßengeometrie für große Städte oder Stadtteile zu generieren, welche in Stadtvisualisierungsprogrammen verwendet werden. Ein 3x3 km großer Ausschnitt der Stadt Wien, welcher mit VRMG erzeugt wurde, wird zum Beispiel im bereits erwähnten URBANVIZ eingesetzt. TrafLib beinhaltet einen Konverter, um von VRMG erzeugte Straßengeometrie-Beschreibungsdateien in das TrafLib Format zur Beschreibung von Straßennetzen umzuwandeln.

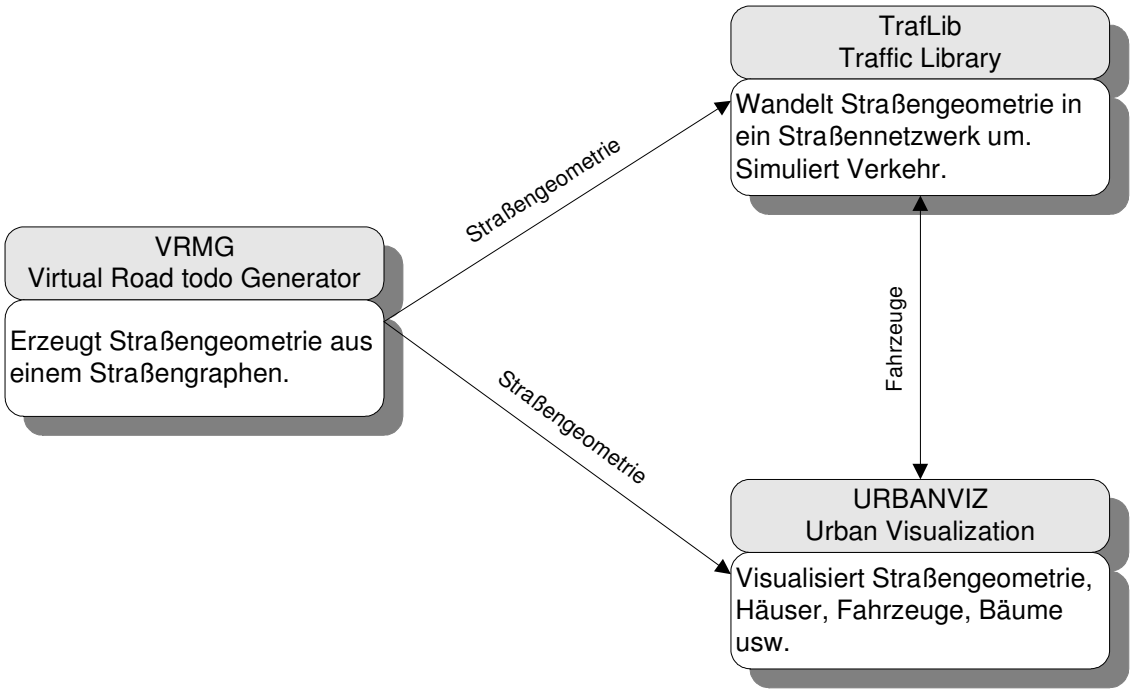


Abb. 1: Zusammenhang TrafLib-VRMG-URBANVIZ

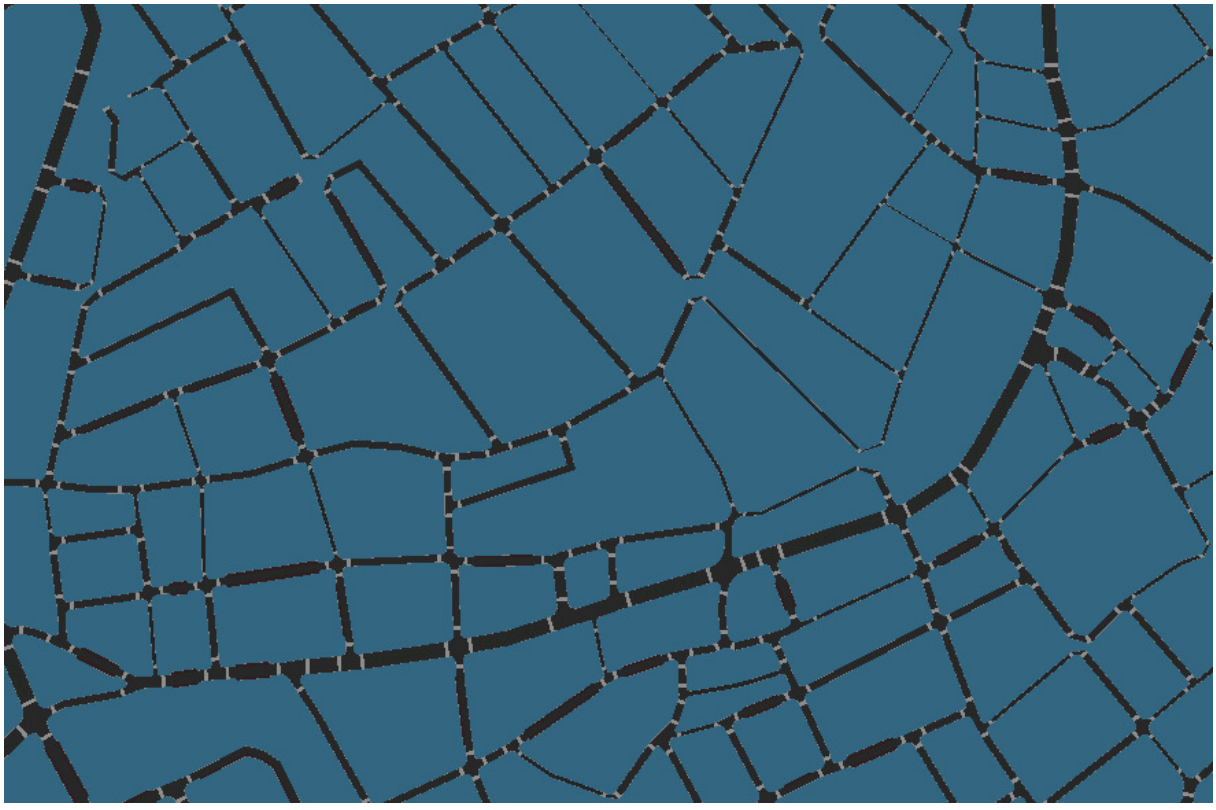


Abb. 2: Straßennetzwerk, Breite ~3 km (ohne Verkehrszeichen)

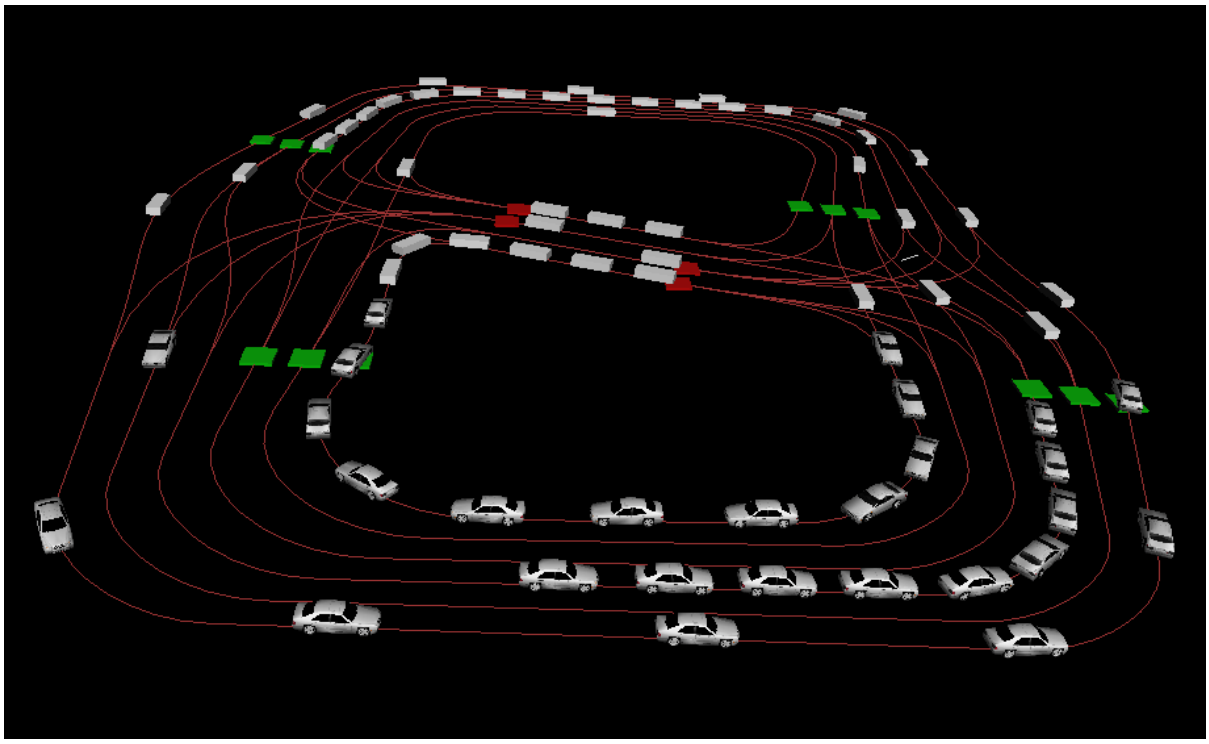


Abb. 3: Beispiel eines Straßennetzes; Fahrzeuge mit LOD Stufen

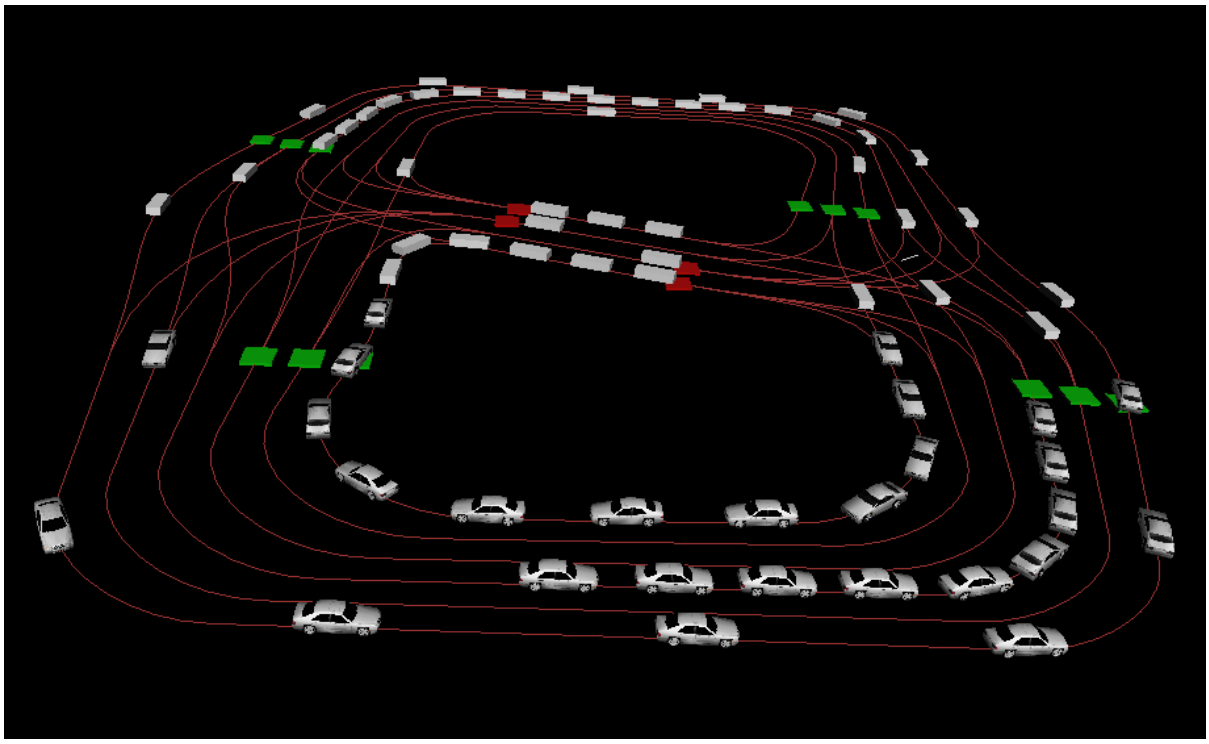


Abb. 4: Kreuzungsbereich aus der Sicht eines Fahrzeuges

Kapiteln kurz beschreiben.

1.1 Anforderungen

Ziel dieser Diplomarbeit ist die Entwicklung einer Bibliothek (TrafLib) für die Simulation von Verkehr. Diese Bibliothek soll sich möglichst einfach in beliebige Echtzeit-Visualisierungssoftware einbinden lassen.

Erforderliche Funktionalität von TrafLib:

- Einfache Einbindbarkeit in Visualisierungssoftware
- Einlesen von Straßennetzwerken aus Dateien, inklusive Ampeln und Verkehrstafeln (optional)
- Umwandlung von Straßengeometrie aus VRMG ins TrafLib-Format
- Generierung von Ampeln und Verkehrstafeln (zumindest Stopp- und Vorrang-Geben Tafeln) sofern sie nicht in der Straßenbeschreibungsdatei definiert sind
- Einhaltung von Verkehrsregeln gemäß Straßenverkehrsordnung

Eine weitere Anforderung ist die Bereitstellung von einigen „geeigneten“ Fahrzeugmodellen. Die Modelle müssen in Formaten vorliegen, die mit den Graphik-Tools Open Inventor [36] und Ogre [38] kompatibel sind. Eine weitere Bedingung ist eine niedrige Anzahl von Dreiecksflächen.

Die Verkehrssimulation ist nicht für die Verkehrsplanung gedacht, weshalb keine detaillierten Vergleiche mit der Realität gezogen werden müssen.

1.2 Anwendungsbereiche

Prinzipiell können Simulationen zum Erforschen, Analysieren und zur Präsentation von Daten verwendet werden. TrafLib beschränkt sich auf die Präsentation, da das Verkehrsmodell nicht exakt auf die Wirklichkeit abgestimmt wurde, wie es zum Beispiel in der Verkehrsplanung erforderlich ist.

Mittels TrafLib lassen sich viele Visualisierungsprogramme durch Hinzufügen von Fahrzeugen lebendiger und ansprechender gestalten:

- Architektur / Städtevisualisierung: Dies umfasst die detailgetreue Nachbildung von Bauwerken, wie z.B. in historischen Städten oder in virtuellen Ansichten zukünftiger Bauten.
- Städteplanung: Mit diesem Begriff ist prinzipiell dasselbe wie mit Städtevisualisierung gemeint, wobei sich allerdings virtuelle Objekte hinzufügen, löschen und manipulieren lassen.
- Computerspiele
- Filmindustrie, Werbeindustrie
- Fahrsimulatoren: Fahrsimulatoren finden Anwendung in der Fahrzeugindustrie, Verkehrsforschung und der Ausbildung und Weiterschulung von Fahrzeuglenkern. In diesem Anwendungsbereich dienen autonom gesteuerte Fahrzeuge der Erhöhung der Realitätstreue von Fahrsimulatoren, wodurch ein realistischeres Verhalten der Benutzer erreicht wird.

- Virtual Reality: Zu Virtual Reality gehört die Verwendung von Head-Mounted-Displays und Datenhandschuhen. Auch hier kann das Realitätsempfinden durch eine Verkehrssimulation erhöht werden.

1.3 Vereinfachungen gegenüber der Realität

Da derzeit die exakte Nachbildung der Realität mittels Simulatoren für Echtzeitanwendungen noch zu komplex ist, werden Vereinfachungen vorgenommen. In erster Linie werden Elemente, welche wenig zum realistischen Eindruck einer Simulation beitragen, ganz oder teilweise weggelassen.

In TrafLib gibt es folgende Vereinfachungen gegenüber der Realität:

- Gleichbleibende Sicht: Es gibt keine Wetterphänomene welche die Sicht einschränken (Nebel, starker Regen) und keine Dämmerung bzw. Dunkelheit. Des Weiteren gibt es keine Sichtbeschränkungen durch Gebäude!
- Gleichbleibende Fahrbahnverhältnisse: Die Formeln zur Berechnung von Geschwindigkeit, Beschleunigung und Bremsung setzen eine trockene Fahrbahn voraus. Es werden also kein Aquaplaning, Schnee, Glatteis, Seitenwind und so weiter simuliert.
- Steigung/Gefälle der Straßen: Eine veränderliche Höhe der Straßen ist in TrafLib möglich, eine Beeinflussung des Fahrverhaltens findet aber nicht statt.
- Unfälle: In TrafLib ist zwar das Fahrverhalten von vorsichtigen, normalen und aggressiven Fahrern nachgebildet, aber selbst das aggressive Fahrverhalten ist so ausgelegt, dass kein Unfall passiert.
- Unfallerkennung: Sollte es in TrafLib wider Erwarten zu einem Unfall kommen, so wird dieser nicht erkannt. Im Gegensatz zur Realität können in TrafLib zwei Körper den selben Platz einnehmen.
- Regelkonformität: Alle Fahrer halten sich strikt an die Verkehrsregeln. Das bedeutet einerseits, dass es keine „Verkehrssünder“ gibt, und andererseits dass es keine „freundlichen“ Fahrer gibt, die bewusst auf ihren Vorrang verzichten, um die Verkehrssituation zu verbessern.
- Einschätzung der Verkehrssituation: Alle Fahrer schätzen gleichzeitig und genau jede ½ Sekunde die Verkehrssituation ein, was sich in einer Änderung der Beschleunigung ausdrückt. In Wirklichkeit sind Menschen aber unterschiedlich aufmerksam beim Beobachten des Verkehrsgeschehens, außerdem hängt die Aufmerksamkeit auch von der jeweiligen (Gefahren-) Situation ab.
- Fußgänger: TrafLib simuliert keine Fußgänger.
- Einschätzung von Geschwindigkeit und Beschleunigung: In TrafLib „wissen“ die Fahrer die Geschwindigkeit und Beschleunigung der anderen Fahrzeuge exakt. In der Realität werden diese Werte nur geschätzt.
- Verkehrsaufteilung bei Kreuzungen: Wenn die Route der Fahrzeuge nicht fix geplant ist sondern zufällig gewählt wird, dann erfolgt in TrafLib die Aufteilung des Verkehrsstromes nach folgenden Wahrscheinlichkeiten: 10% Links-Abbieger, 80% Geradeaus-Fahrer, 10% Rechts-Abbieger [16].
- Geschwindigkeitsbegrenzung: TrafLib ist zur Simulation von städtischem Verkehr entwickelt worden und lässt keine Geschwindigkeiten über 20 m/s (72 km/h) zu. Die

Vorgabegeschwindigkeit liegt bei 50 km/h, nur durch Geschwindigkeits-begrenzungstafeln lässt sich diese auf 72 km/h erhöhen.

- Fahrzeuglänge/-breite: Kein Fahrzeug darf eine Länge von 4,5 m und eine Breite von 2,0 m überschreiten. Der Grund dafür liegt darin, dass TrafLib die Eigenheiten von überdimensionierten Fahrzeugen, wie z.B. das Ausscheren in einer Kurve, nicht berücksichtigt.
- Rückstau in einen Kreuzungsbereich: Sobald ein Fahrzeug den Kreuzungsbereich verlassen hat, nehmen andere Fahrzeuge im Kreuzungsbereich keine Rücksicht mehr auf dieses, außer natürlich beim Hintereinanderfahren. Das wirkt sich zum Beispiel in der Form aus, dass Fahrzeuge, die in die Kreuzung eingefahren sind und während der Grünphase den Kreuzungsbereich nicht mehr verlassen können, den Querverkehr behindern. Dieser Fall tritt jedoch nur bei Stau-Situationen auf.
- Deadlock: In TrafLib kann es wie in der Realität zu Deadlock-Situationen im Kreuzungsbereich kommen. Wenn zum Beispiel in einer ungeregelten, 3-armigen Kreuzung auf jeder Straße gleichzeitig ein Fahrzeug in die Kreuzung einfährt, dann muss aufgrund der Rechtsregel jeder stehen bleiben, weil jeder einen Rechtskommenden hat. In der Realität wird dieses Problem gelöst, in dem einer der Fahrer auf seinen Vorrang verzichtet. In TrafLib werden Fahrzeuge, die sich länger als 2 Minuten bewegungslos in einem Kreuzungsbereich befinden, entfernt.

2 Stand der Technik

Mit dem explosionsartig steigenden Verkehrsaufkommen ab den 60er Jahren wurden Verkehrssimulationen für Verkehrsplaner unumgänglich. Damals war die Leistung von Computern noch zu gering, um einzelne Fahrzeuge simulieren zu können. Deshalb wurden zu dieser Zeit Methoden bzw. Formeln entwickelt, um das Verhalten einer Gruppe von Fahrzeugen zu berechen [21]. Seither wird dies als „makroskopische“ Simulation bezeichnet.

Die heutigen Verkehrssimulatoren können in mikroskopische, makroskopische Simulatoren und Mischformen aus diesen beiden unterteilt werden [8].

Makrosimulatoren modellieren die allgemeinen Aspekte von Verkehrssystemen, wie die durchschnittliche Geschwindigkeit aller Fahrzeuge auf einer Straße, die Verkehrsdichte (= Anzahl der Fahrzeuge pro Längeneinheit) und den Verkehrsfluss (= Anzahl der Fahrzeuge pro Zeiteinheit). Makroskopische Modelle basieren auf der „Flow Theory“ [34]. Diese Sichtweise berücksichtigt jedoch feinere Aspekte wie zum Beispiel das individuelle Verhalten von Fahrern aufgrund ihrer Charaktereigenschaften nicht. Dadurch ist der Rechenaufwand geringer und es lassen sich größere Straßennetze simulieren.

Eine mikroskopische Simulation modelliert jedes einzelne Fahrzeug das in den Verkehr involviert ist. Die Bewegung der Fahrzeuge wird durch Formeln modelliert, die direkt oder indirekt von physikalischen Gesetzen abgeleitet sind. Die Basis für mikroskopische Berechnungsmodelle ist die „Car Following Theory“ [34]. Diese besagt, dass die Änderung von Beschleunigung und Geschwindigkeit eine Funktion ist, die abhängt von Beschleunigung und Geschwindigkeit des vorderen Fahrzeuges und dem Abstand zu ihm. Dabei wird ausgenutzt, dass auf dichtbefahrenen Fahrspuren die Fahrzeuge „Kolonnen“ (platoons) [34] bilden und sich dadurch die Bewegung der gesamten Schlange einfacher berechnen lässt. Die Fahrzeuge bzw. Fahrer können verschiedene Charakteristika haben, wie zum Beispiel die Fahrzeuglänge, die maximale Geschwindigkeit, das maximale Bremsvermögen, das Fahrverhalten und so weiter. Verkehr kann als das kollektive Verhalten von individuellen Fahrzeugen angesehen werden. Vor allem die mikroskopische Simulation profitiert von der immer schneller werdenden Hardware.

Die aktuellen Verkehrssimulatoren für Verkehrsplanung sind bereits so weit fortgeschritten, dass ihre Abweichung von der Realität nur noch gering ist. Zum Beispiel beträgt die Abweichung bei der kommerziellen Software CTSP nur mehr etwa 10% [13].

Wie bereits in der Einleitung erwähnt, benutzen manche Verkehrssimulatoren Mischformen von mikroskopischem und makroskopischem Verhalten [5, 11, 34]. Meist sieht das Modell dann so aus, dass man den Verkehr über Kreuzungen mikroskopisch simuliert, aber bei den Kreuzungszufahrten werden sogenannte Queues verwendet, aus denen eine gewisse Anzahl von Fahrzeugen pro Zeiteinheit in die Kreuzung entlassen werden. Bei einem solchen Modell könnten zum Beispiel die Parameter von Ampeln variiert werden, so lange bis man die beste Ampelschaltung für das entsprechende Verkehrsaufkommen gefunden hat.

Eine spezielle Form der mikroskopischen Simulation sind zelluläre Automaten, wobei das Modell von Nagel/Schreckenberg der bekannteste Vertreter ist [6]. Dort ist das dynamische Verhalten von Fahrzeugen diskret in Zeit und Raum. Dieses Modell ist sehr einfach gehalten und kommt daher mit einem sehr geringen Rechenaufwand aus. Trotzdem weist es realistisches Verhalten auf. Das Original-Modell von Nagel und Schreckenberg wurde für einspurigen Verkehr entwickelt. Später wurde es erweitert, um auch mehrere Fahrspuren [31] und städtischen Verkehr [30] simulieren zu können.

2.1 Kommerzielle Software

Liste der Software aus den Papers rauslesen und googlen.
todo

3 Theoretische Grundlagen

Die im Folgenden angeführten Grundlagen sind erforderlich um TrafLib im Detail zu verstehen. Es werden spezielle Bereiche aus Verkehrswegebau, Verkehrsregelung, Fahrverhalten und Mathematik behandelt.

3.1 Verkehrswegebau

Die Grundlagen des Verkehrswegebaues sind für diejenigen interessant, die für TrafLib ein Straßennetzwerk entwerfen möchten.

In der Realität beginnt die Planung einer Straße mit der Absteckung eines Polygonzuges, welcher der Mittelachse entspricht. Danach führt man die sogenannte Trassierung durch, welche aus dem Polygonzug eine Raumkurve (Krümmung in der Ebene und Senkrechten) erzeugt. Dies geschieht mit Hilfe von den Trassierungselementen Gerade, Übergangsbogen, Kreisbogen. Für nähere Ausführungen siehe VRMG [1].

Für Straßennetzwerke in TrafLib genügt üblicherweise der erste Schritt der Verkehrswegeplanung, die Angabe der Polygonzüge. Die Ausrundung der Ecken in den Polygonzügen erfolgt in TrafLib automatisch mittels Bezier-Kurven, welche für den städtischen Bereich sehr gute Ergebnisse liefern, da die Kurvenradien nur geringe Varianz aufweisen.

An jedem Eckpunkt des Polygonzuges muss für TrafLib der Straßenquerschnitt mit Informationen über die Fahrstreifen angegeben werden.

Die Kreuzungen ergeben sich als Bereiche zwischen den Straßenenden.

3.2 Verkehrsregeln

Zur einfacheren Darstellung wird hier wird Verkehrsregelung in Regelung im Kreuzungsbereich und Regelung außerhalb des Kreuzungsbereiches unterteilt. Im Kreuzungsbereich kommen ausschließlich Ampeln, Stopp-Tafeln, Vorrang-Geben-Tafeln und die Rechts-Regel zum Tragen. Außerhalb des Kreuzungsbereiches befinden sich Tafeln zur Geschwindigkeitsbegrenzung, zur Markierung von Vorrangstraßen usw.

3.2.1 Verkehrsregelung im Kreuzungsbereich

Die Verkehrsregeln bestimmen, ob ein Fahrzeug einen bestimmten Punkt passieren darf oder davor stehen bleiben muss.

Der Verkehr wird im Kreuzungsbereich durch folgende Faktoren geregelt:

1. Rechts-Regel: Die Rechts-Regel besagt, dass der Rechts-Kommende Vorrang hat und ist immer dann gültig, wenn sie nicht durch eine Verkehrstafel oder Ampel aufgehoben wird.
2. Verkehrs-Tafeln: Im Kreuzungsbereich sind letztlich nur Stopp-Tafeln und Vorrang-Geben-Tafeln relevant.
3. Ampeln: Im Normalfall durchläuft eine Ampel folgende Ampelphasen: Grün, Grün blinkend, Orange, Rot. Im Sonderfall kann sie auch Orange blinken oder ausgeschaltet sein. Weitere Sonderfälle sind z.B. eigene Linksabbiege-Ampeln.

Prioritäten für Vorrangregeln:

1. Rote Ampel – höchste Priorität – Ein Fahrzeug muss bei einer Roten Ampel stehen bleiben.
2. Stopp-Tafel – Ein Fahrzeug, das sich einer Stopptafel nähert, hat Nachrang gegenüber allen anderen Fahrzeugen, außer denen, die aufgrund einer Roten Ampel halten müssen. (Dieser Fall tritt selten auf, zum Beispiel im Zusammenhang mit Fußgängerampeln.)
3. Vorrang-Geben-Tafel – Ein Fahrzeug, das vor einer Vorrang-Geben-Tafel steht, hat Vorrang gegenüber anderen, die vor einer Stopp-Tafel oder Roten Ampel stehen.
4. Rechts-Regel – Gibt es weder eine Ampel noch ein anderes Verkehrszeichen, so hat ein Fahrzeug gegenüber Fahrzeugen die unter die Punkte 1-3 fallen Vorrang. Haben mehrere Fahrzeuge keine Ampeln oder Verkehrszeichen, so hat der Rechtskommende Vorrang.

3.2.2 Verkehrsregelung abseits von Kreuzungen

Da die Verkehrsregelung außerhalb von Kreuzungen von geringerer Bedeutung ist, werden von TrafLib nur Geschwindigkeitsbegrenzungs-Tafeln unterstützt.

3.3 Fahrverhalten

Das Fahrverhalten wird einerseits bestimmt durch die Charakteristika von Fahrzeugen, und andererseits durch das individuelle Verhalten der Fahrer.

3.3.1 Technisches – Fahrzeug

Eigenschaften von Fahrzeugen die für eine Verkehrssimulation von Bedeutung sind:

- Höchstgeschwindigkeit: Diese im städtischen Bereich selten maßgebend, da die Höchstgeschwindigkeit der üblichen Pkws höher ist als die Geschwindigkeitsbegrenzungen im Stadtgebiet.
- Maximale Beschleunigung: Ein durchschnittlicher Pkw heutiger Bauart erreicht eine maximale Beschleunigung von 2 m/s^2. In TrafLib wird dieser Wert als Voreinstellung für Fahrzeuge verwendet, wobei die tatsächlich gewählte Beschleunigung von individuellen Parametern der Fahrer abhängt.
  Bei einer Beschleunigung von 2 m/s^2 benötigt ein Fahrzeug etwa 14 Sekunden bzw. 200 Meter um von 0 auf 100 km/h zu kommen.
  In der Realität ist die Beschleunigung eine Funktion von Geschwindigkeit und Getriebeübersetzung, in der Simulation wird sie konstant angenommen.
- Maximale Bremsverzögerung: Ein typischer Pkw erreicht unter günstigsten Straßenverhältnissen etwa eine Bremsverzögerung von 9 m/s^2.
  Bei einer Bremsverzögerung von 9 m/s^2 benötigt ein Fahrzeug etwa 3 Sekunden bzw. 43 Meter, um von 100 km/h zum Stillstand zu kommen.

3.3.2 Individuelles - Mensch

Prinzipiell muss jeder Fahrzeuglenker auf die Gegebenheiten seines Umfeldes reagieren. Er muss Verkehrsregeln und andere Verkehrsteilnehmer beachten um Unfälle zu vermeiden. Die Reaktion auf das Umfeld drückt sich in einer positiven oder negativen Beschleunigung aus [12].

Die technisch mögliche Beschleunigung und Bremsverzögerung wird nur im Notfall verwendet. In der Realität sowie in TrafLib ist die gewählte Beschleunigung bzw. Bremsverzögerung der Fahrzeuge meist deutlich geringer.

Aufgrund der Psychologie des Menschen ist das Fahrverhalten sehr unterschiedlich. Dies drückt sich aus in der gewählten Beschleunigung, Bremsverzögerung und Geschwindigkeit [8], dem Abstand zum Vorderfahrzeug, der Aufmerksamkeit und dem Reaktionsvermögen, dem Hang zu Überholmanövern bis hin zu einer eventuellen Missachtung der Verkehrsregeln.

3.4 Mathematische Grundlagen

Die mathematischen Grundlagen umfassen Bezierkurven, Schnitte von Polygonzügen und Formeln über den Zusammenhang zwischen Geschwindigkeit, Beschleunigung, Weg und Zeit.

3.4.1 Bezierkurven

In TrafLib werden Bezierkurven zur Konstruktion der Kurven im Straßenverlauf verwendet.

In der Realität besteht ein Straßenverlauf aus Geraden, Übergangsbögen (meistens Klothoiden [1, 2, 3]) und Kreisbögen. Klothoiden sind Kurven mit veränderlichem Radius. Zwischen Geraden und Kreisbögen müssen immer Klothoiden (s. Abb. 5) eingeschoben werden um einen stetigen Verlauf der Krümmung zu erreichen.

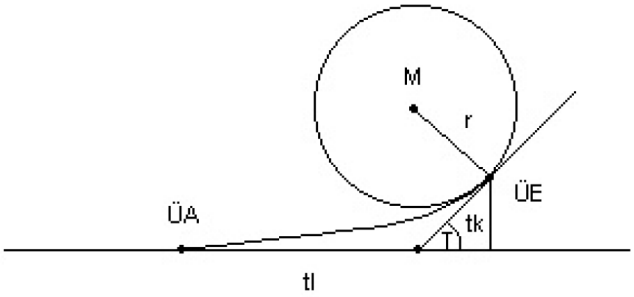


Abb. 5: Klothoide [1]

Eine Klothoide ist durch folgende Formel definiert: $A^2 = r \cdot l$.

A ist der Klothoidenparameter und bestimmt die Stärke der Krümmung. Die Parameter r und l entsprechen dem Radius und der Bogenlänge, wobei mit zunehmender Bogenlänge der Radius kleiner wird.

Klothoiden sind aber relativ kompliziert zu berechnen und werden aufgrund ihrer Ähnlichkeit zu kubischen Splines für Modellierungszwecke durch diese ersetzt [1].

In TrafLib wurde die Erzeugung von Kurven noch weiter vereinfacht und zwar durch die Verwendung von Bezier-Kurven anstelle von kubischem Spline – Kreisbogen – kubischem Spline. Somit verwendet TrafLib ausschließlich Geraden und Bezier-Kurven. Bei einem Vergleich zwischen einer von VRMG [1] erzeugten Straßengeometrie mit den von TrafLib erzeugten Fahrlinien eines 3x3 km großen Stadtausschnittes hat sich gezeigt, dass die Übereinstimmung ziemlich hoch ist.

Formel einer Bezier-Kurve: $B_t = \sum_{i=0}^{n} P_i * C_n^i * t^i * (1-t)^{n-i}$

P_i ... Stützpunkte für die Bezierkurve

t .... Parameter im Bereich [0..1]

n ... Anzahl der Stützpunkte

C_n^i ... Kombination über n und i

3.4.2 Schnitte und Schnittbereiche zwischen Polygonzügen

Der Schnittpunkt zweier Fahrstreifen muss von Fahrzeugen zeitlich hintereinander passiert werden um keinen Unfall zu verursachen. Da Schnitte aber auch schleifend sein können, ist die Berechnung eines Schnittbereiches notwendig.

Es gibt 3 Fälle von Schnitten:

1. Genau ein Schnitt zwischen zwei Polygonzügen. Dies ist der häufigste Fall. Ist der Schnitt schleifend, wird ein Schnittbereich berechnet. Zu diesem Punkt gehört auch der „Schnitt“ am Anfang oder Ende eines Polygonzuges wie er bei Spurvereinigung und Spuraufteilung auftritt. In diesem Fall benötigt man fast immer einen Schnittbereich. Siehe Abb. 6 und Abb. 7.
2. Zwei Schnitte zwischen zwei Polygonzügen. Dieser Fall tritt sehr selten auf. Es werden zwei Schnittpunkte (= Konfliktpunkte) berechnet. Auf einen Schnittbereich wird verzichtet, da sich solche Polygonzüge nach den Schnitten sehr schnell von einander entfernen. Siehe Abb. 8.
3. Kein Schnitt, aber zwei Polygonzüge kommen einander so nahe, dass zwei Autos nicht nebeneinander Platz haben. In diesem Fall wird ein imaginärer Schnittpunkt und Schnittbereich für den zu schmalen Fahrbereich berechnet. Siehe Abb. 9.

Die Berechnung der Schnittbereiche erfolgt folgendermaßen:

Vom Schnittpunkt ausgehend wird in alle 4 Richtungen die Entfernung berechnet, ab der Fahrzeuge wieder nebeneinander Platz haben. Durch die 4 Richtungen ergeben sich 8 Punkte; es werden aber nur die 4 am weitesten entfernten Punkte verwendet. Siehe Abbildung Abb. 6.

In der Bibliothek werden diese Schnittbereiche derart eingesetzt, dass Fahrzeuge, die auf den selben Schnittbereich zusteuern, diesen nur zeitlich hintereinander passieren dürfen.

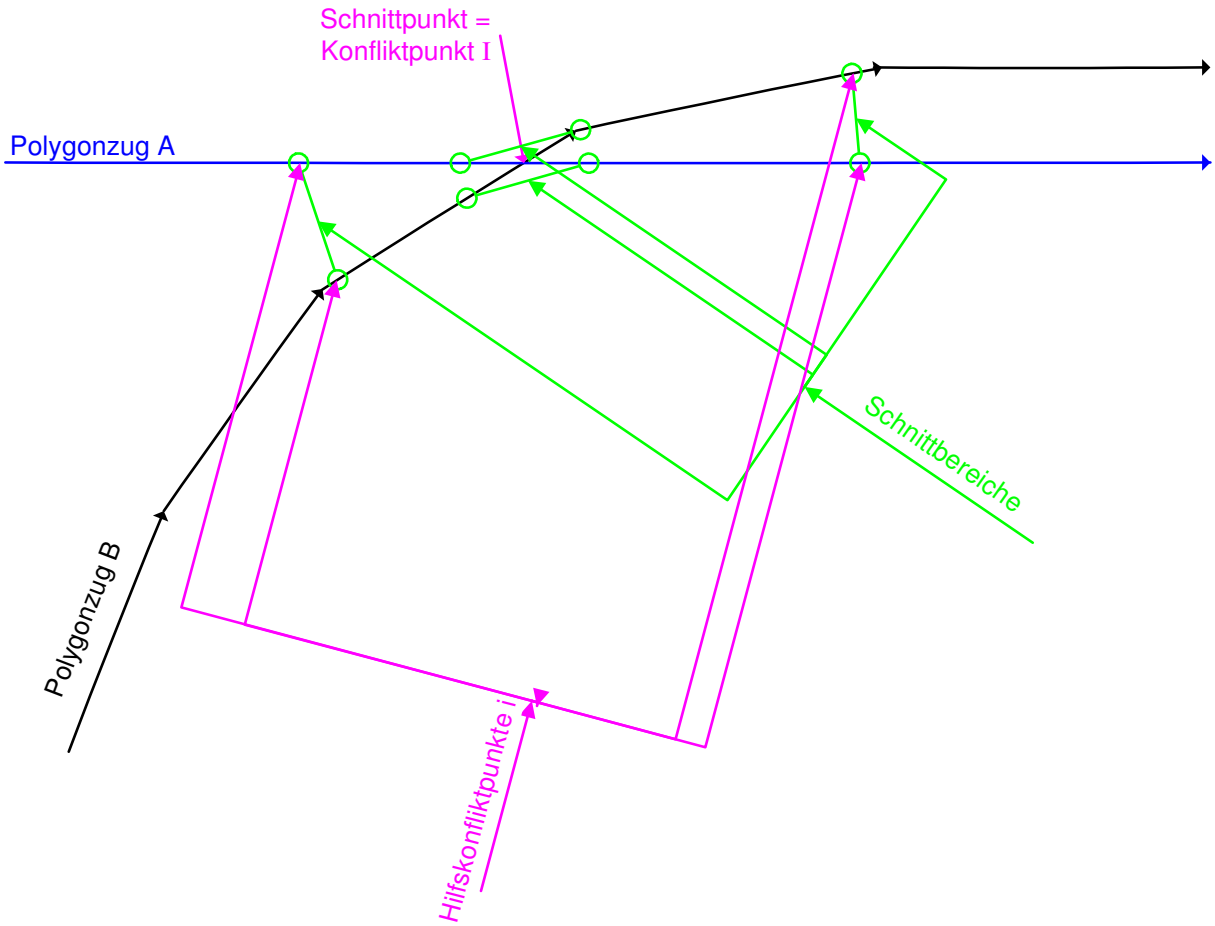


Abb. 6: Schnitte von Polygonzügen Fall 1

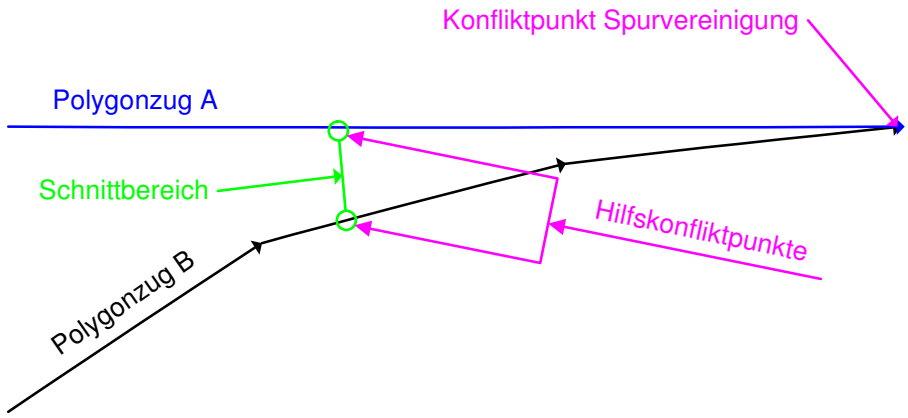


Abb. 7: Schnitte von Polygonzügen Fall 1, Sonderfall

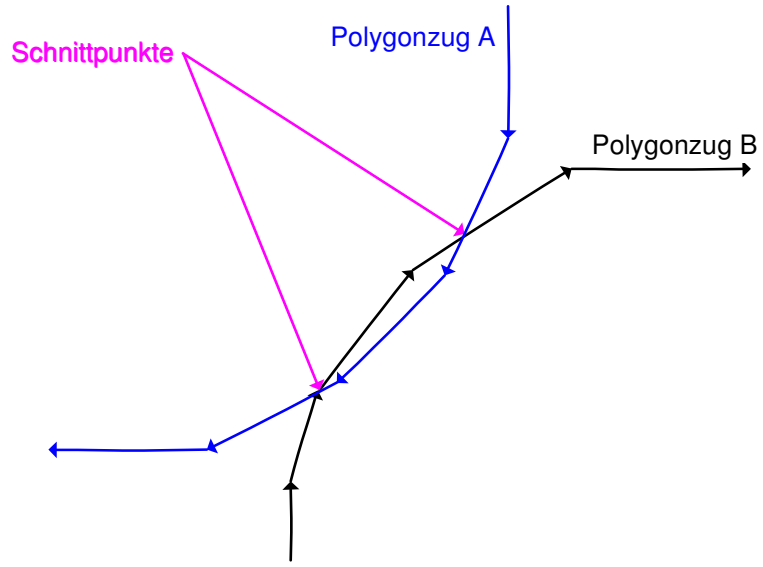


Abb. 8: Schnitte von Polygonzügen Fall 2

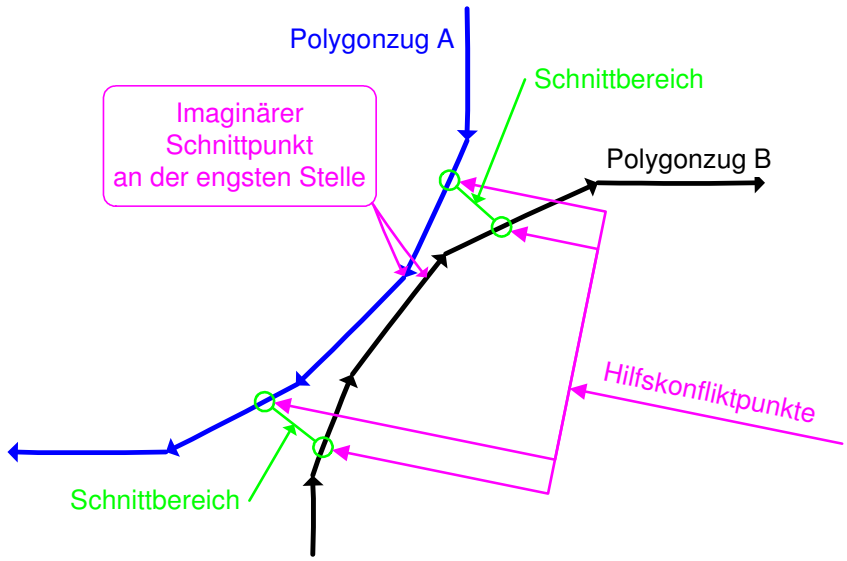


Abb. 9: Schnitte von Polygonzügen Fall 3

3.4.3 Formeln für die Fahrzeugbewegung

Dieses Kapitel behandelt die für TrafLib erforderlichen von der Physik abgeleiteten Formeln, zur Berechnung von Geschwindigkeitsveränderung, Bremsweg, Reaktionszeit usw.

u	...	Initial-Geschwindigkeit
v	...	End-Geschwindigkeit, nach einer Zeit t
a	...	positive oder negative Beschleunigung
a_B	...	maximale Bremsstärke eines Fahrzeuges
t	...	verstrichene Zeit
t_R	...	Reaktionszeit

s ... zurückgelegter Weg

s_R ... Reaktionsweg

v_{VF} ... Geschwindigkeit des Vorderfahrzeuges

a_{VF} ... Beschleunigung des Vorderfahrzeuges

- $v = u + a \cdot t$ ... Berechnet die Veränderung der Geschwindigkeit aufgrund der aktuellen Beschleunigung. Echtzeitanwendungen werden in Schritten von $t = 1/60$ berechnet.
- $s = u \cdot t + \frac{a \cdot t^2}{2}$ ... Zurückgelegter Weg in Abhängigkeit der aktuellen Geschwindigkeit, Beschleunigung und der verstrichenen Zeit.
- $s_R = u \cdot t_R + \frac{a \cdot t_R^2}{2}$ ... Reaktionsweg
- $s = \frac{u^2}{2 \cdot a_B}$ ... Bremsweg
- $a = -\frac{u^2}{2 \cdot s}$ ... Bremsverzögerung um in der Entfernung s zum Stillstand zu kommen. Bei dieser Formel kommt es zu folgendem Problem: Wenn $u = 0$, dann ist auch $a = 0$ und das Fahrzeug legt den Weg s nicht zurück.
- $a = \frac{(v+u) \cdot (v-u)}{2 \cdot s}$ ... Beschleunigung um die Zielgeschwindigkeit nach dem Zurücklegen eines Weges s zu erreichen.

4 TrafLib - Bibliothek

TrafLib ist eine graphiklose Bibliothek zur Simulation von Verkehr.

Jedes Visualisierungsprogramm, das unter anderem Straßen visualisiert, kann mit Hilfe dieser Bibliothek aufgewertet werden, indem es Fahrzeuge entlang der Straßen entsprechend den Verkehrsregeln bewegt. Einziges Erfordernis ist die Bereitstellung des Straßennetzes in Form einer zu TrafLib kompatiblen Datei.

4.1 Einleitung

Bei TrafLib handelt es sich um einen mikroskopischen Verkehrssimulator. Das Straßennetzwerk bzw. die Umwelt werden einmal initialisiert und während der Simulation werden dynamisch Fahrzeuge platziert und entfernt.

Die strukturelle Aufteilung von TrafLib ist ähnlich wie in [14], TrafLib setzt sich aus 4 Komponenten zusammen:

1. Straßennetzwerk: Dieses wird aus einer Datei eingelesen. TrafLib unterstützt auch das Straßengeometrie-Format von VRMG [1], einer Software zur Erzeugung von großen städtischen Straßennetzen. Zum Straßennetzwerk gehören auch die verkehrsregelnden Elemente wie Verkehrstafeln und Ampeln.
2. Fahrzeuge: In TrafLib sind Fahrzeuge imaginäre Punkte auf Linien. Sie haben verschiedene Eigenschaften, welche sich aus technischen Fahrzeug-Charakteristika und individuellen Fahrer-Verhaltensweisen zusammensetzen. Es ist die Aufgabe der Visualisierungssoftware, Fahrzeuge auf diese Punkte zu setzen.
3. Verkehrsberechnung: Die Berechnung der Fahrzeugbewegung findet aufgrund der Fahrzeugcharakteristika unter Einhaltung von Verkehrsregeln statt.
4. Statistik: Die statistische Auswertung ermittelt Werte, welche aus makroskopischen Simulationen bekannt sind, wie Verkehrsfluss, Verkehrsdichte und mittlere Geschwindigkeit pro Straße. Diese Werte werden unter anderem bei der Routenberechnung von Fahrzeugen verwendet, um die zeitlich kürzeste Route zu berechnen.

4.1.1 Fähigkeiten

TrafLib kann ...

- ein Straßennetz aus einer Datei einlesen, optional mit Ampeln, Verkehrstafeln und Korridoren.
- ein Straßennetz, welches der VRMG Spezifikation [1] entspricht, ins TrafLib Format konvertieren. Der Vorteil in der Verwendung von VRMG liegt in der automatisierten Erzeugung von Geometrie für große Straßennetze.
- Ampeln und Verkehrstafeln nach verschiedenen Algorithmen generieren, falls diese nicht in der Straßennetz-Beschreibungsdatei angegeben sind.
- Korridore generieren, falls diese nicht in der Straßennetz-Beschreibungsdatei angegeben sind.
- Fahrzeuge auf zufällige oder bestimmte Positionen setzen. Optional lassen sich fahrzeugspezifische Parameter wie die maximale Geschwindigkeit, Beschleunigung

und Bremsverzögerung angeben. Des Weiteren kann auch ein Zielpunkt vorgegeben werden.

- Fahrzeuge entfernen, zum Beispiel jene die außerhalb der Sichtweite einer Kamera liegen.
- Routen unter Berücksichtigung des aktuellen Verkehrs berechnen.
- Verkehr simulieren, das heißt alle Fahrzeuge bewegen sich entsprechend physikalischer und individueller Parameter und halten sich an alle Verkehrsregeln.
- für Echtzeitanwendungen verwendet werden; je nach Leistungsfähigkeit des PCs können mehrere tausend Fahrzeuge simultan gesteuert werden.
- mit einer flexiblen zeitlichen Steuerung umgehen: In Echtzeitanwendungen wird jede ½ Sekunde die „Fahrzeugsituation“ neu beurteilt und jede 1/60 Sekunde werden die Fahrzeuge bewegt. Da es für das Visualisierungsprogramm möglich ist, die verstrichene Zeit für TrafLib anzugeben, lassen sich Zeitlupe und Zeitraffer einfach modellieren oder die Simulation stoppen. Dadurch können zum Beispiel beliebig viele Fahrzeuge gleichzeitig simuliert werden, mehr als in Echtzeit möglich wäre, und die Simulation kann auf Video aufgenommen und danach schneller abgespielt werden.
- die Anzahl der Fahrzeuge reduzieren, wenn die Anwendung zum Beispiel nicht mehr in Echtzeit ausgeführt werden könnte.

4.1.2 Grenzen

Wie jede Software hat auch TrafLib Einschränkungen:

- Fahrzeuglänge: Für Fahrzeuge die von TrafLib simuliert werden ist eine Länge von 4,5 m vorgesehen. Da sich die Fahrzeuge auf vorgegebenen Rails bewegen, würden sich für überlange Fahrzeuge 3 Probleme ergeben:
  - Die Bewegung um eine Kurve würde unrealistisch ausschauen, da das hintere Ende ausschert.
  - Wenn ein überlanges Fahrzeug um eine enge Kurve fahren würde, würde ein benachbarter Fahrstreifen blockiert werden. Dies wird aber bei TrafLib nicht berücksichtigt.
  - Für alle Abstandsberechnungen, z.B. dem Abstand eines stillstehenden Fahrzeuges zu einer Ampel, dem Abstand zweier Fahrzeuge zueinander, etc., wird die typische Fahrzeuglänge von 4,5 m verwendet.
- Beschränkung auf städtischen Verkehr:
  - Fahrzeuggeschwindigkeiten über 20 m/s bzw. 72 km/h wurden nicht getestet!
- Kreisverkehr: TrafLib unterstützt keinen Kreisverkehr. Eine mögliche Lösung wäre, einen Kreisverkehr durch eine Reihe von T-Kreuzungen mit Vorrang-Geben-Tafeln für die einmündenden Straßen zu modellieren.

4.1.3 Besonderheit

Die Verkehrsregelung und Unfallvermeidung erfolgt in TrafLib wie bereits erwähnt durch Konfliktpunkte. Diese Methode ist meines Wissens noch in keinem Paper veröffentlicht worden.

Am ehesten ist diese Methode mit [9] vergleichbar. Dort wird der Verkehrsfluss mit Hilfe von „control points“ und „fixed points“ geregelt. Dabei werden den „control points“ Geschwindigkeitslimits zugewiesen, ein Limit von 0 bedeutet zum Beispiel, dass ein Fahrzeug bei diesem „control point“ stoppen muss.

4.2 Grundlagen

Wie bereits erwähnt, wird in TrafLib die Fahrzeugbewegung durch Konfliktpunkte und Hintereinanderfahren gesteuert. Die Konfliktpunkte werden unterteilt in verkehrsregelnde und unfallvermeidende.

Die für die Verkehrsregelung verantwortlichen Konfliktpunkte bestimmen immer, ob ein Fahrzeug diesen Punkt passieren darf oder nicht. Dies betrifft zum Beispiel eine Ampel. Ist die Ampel auf Rot, so muss das Fahrzeug vor dem Konfliktpunkt stehen bleiben, ist sie auf Grün, gibt es keine Beeinflussung für das Fahrzeug. Ein weiteres Beispiel ist die Rechtsregel bei einem Schnitt von Fahrspuren – in diesem Fall muss das Fahrzeug so weit vor dem Schnittpunkt stehen bleiben, dass es bevorrangte Fahrzeuge nicht behindert.

Bei den Konfliktpunkten zur Unfallvermeidung wird immer das Reißverschlussprinzip angewendet. Dies tritt zum Beispiel bei Fahrspurreduktionen außerhalb einer Kreuzung auf.

Meist wird das Fahrverhalten eines Fahrzeuges im städtischen Bereich durch das Hintereinanderfahren beeinflusst.

In [18] wird der Verkehrsfluss durch 2 Schichten dargestellt:

- „mental layer“: Simuliert die Verwendung von Strategien, um ein Ziel zu erreichen. Dies beinhaltet zum Beispiel Routenplanung, Wahl des Transportmittels (Pkw, Bus, Fahrrad, ...), tägliche Aktivitätsplanung usw. In TrafLib erfolgt auf dieser Ebene nur die Routenplanung.
- „physical layer“: Simuliert die physikalische Welt in der sich „agents“ bewegen, einander ausweichen, um Hindernisse herum fahren, Staus verursachen usw. Dies entspricht in TrafLib dem Straßennetzwerk, den Fahrzeugen und der Verkehrsberechnung.

4.2.1 Rails, Korridore und Konfliktpunkte

Im Folgenden werden die Begriffe „Rail“, „Korridor“ und „Konfliktpunkt“ definiert und in Abb. 10 dargestellt.

- Rail: Eine „Rail“ entspricht in TrafLib der Mittellinie einer Fahrspur. Fahrzeuge werden in Form von Punkten auf Rails, ähnlich wie eine Eisenbahn auf Schienen, bewegt. Dadurch fällt für die Fahrzeuge ein Freiheitsgrad, nämlich die seitliche Bewegung weg, und die Fahrzeugbewegung vereinfacht sich stark. Somit können mehr Fahrzeuge in Echtzeit simuliert werden. Dies ist dennoch keine grobe Einschränkung, da sich Fahrzeuge in der Realität ohnehin meist innerhalb ihrer Fahrspur bewegen. Ein Spurwechsel ist in TrafLib trotzdem möglich.
- Korridor: Ein Korridor ist eine Rail, die sich im Kreuzungsbereich befindet. Fahrzeuge können sich ausschließlich auf Rails über Kreuzungsbereiche bewegen, bzw. muss für jede Fahrspur über einen Kreuzungsbereich eine Rail existieren.
- Konfliktpunkt: Konfliktpunkte werden in TrafLib auf Stellen platziert, an denen eine Verkehrsregel gültig ist, oder ein Unfall vermieden werden muss.

- Verkehrsregelnde Konfliktpunkte: Diese treten überall dort auf, wo eine Verkehrsregel zum Tragen kommt (Ampel, Verkehrstafel). Es gibt auf jeder Rail zumindest bei jeder Kreuzungseinfahrt einen verkehrsregelnden Konfliktpunkt.
- Unfallvermeidende Konfliktpunkte: Diese treten bei jeder Spuraufteilung, -vereinigung und jedem Spurschnitt auf. Bei Spurschnitten und Spurvereinigungen im Kreuzungsbereich gilt implizit die Rechtsregel, dies bedeutet dass ein Fahrzeug diese Konfliktpunkte nicht passieren darf, wenn es benachrangt ist. Bei Spuraufteilungen und Spurvereinigungen außerhalb eines Kreuzungsbereiches werden Unfälle durch das Reißverschlussprinzip verhindert.

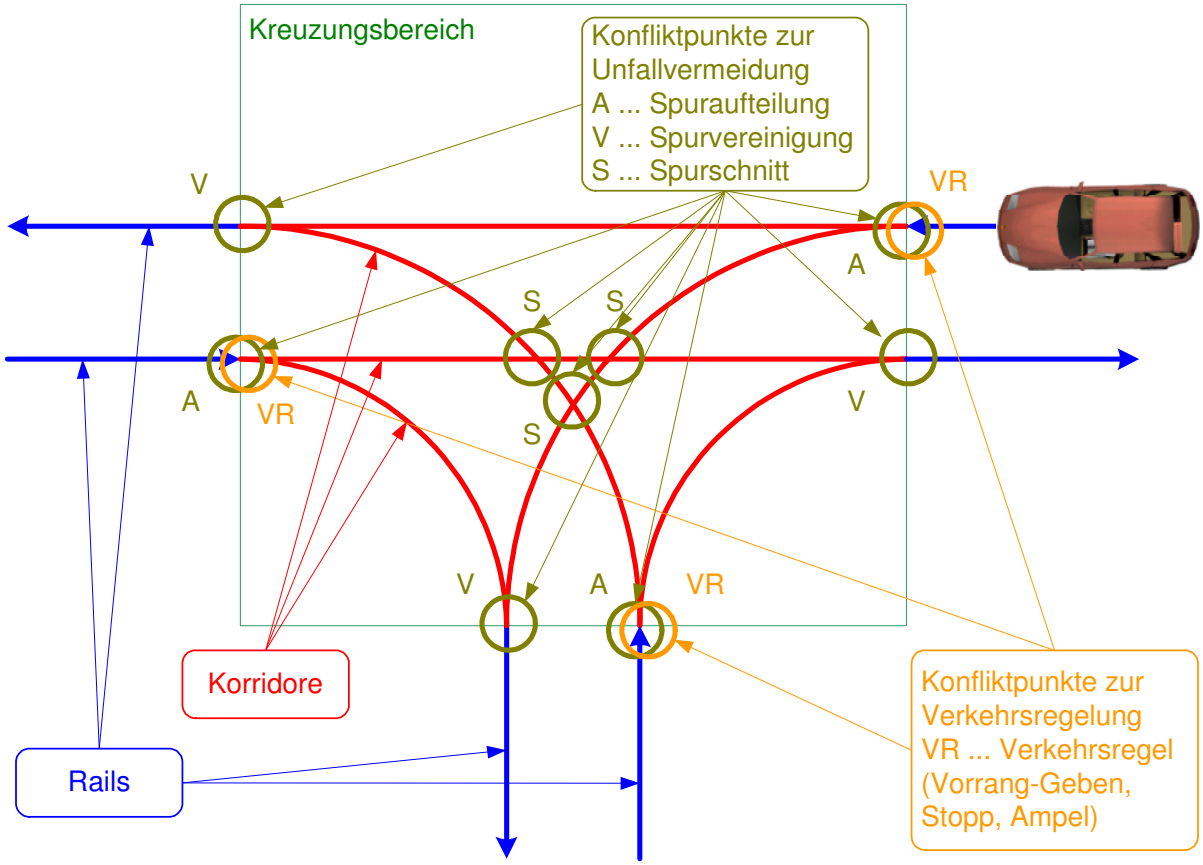


Abb. 10: T-Kreuzung mit Rails, Korridoren und Konfliktpunkten

4.2.2 Fahrzeuge und Einflussbereiche

Jedes Fahrzeug schaut eine bestimmte Entfernung (= Einflussbereich) voraus, und muss auf beeinflussende Fahrzeuge und Verkehrsregeln in diesem Bereich reagieren. Da sich die Fahrzeuge ausschließlich auf „Rails“ bewegen, muss nur der Bereich vor einem Fahrzeug auf der *selben* Rail observiert werden. Befindet sich auf dieser Rail ein Konfliktpunkt, welcher sich auch in dem Einflussbereich eines anderen Fahrzeuges auf einer anderen Rail befindet, so müssen diese Fahrzeuge koordiniert werden, siehe Abb. 11. In diesem Fall kommen Vorrang-Regeln oder das Reißverschlussprinzip zum Tragen. Durch diese Methode reduziert sich der Rechenaufwand zur Findung von Konflikten beträchtlich.

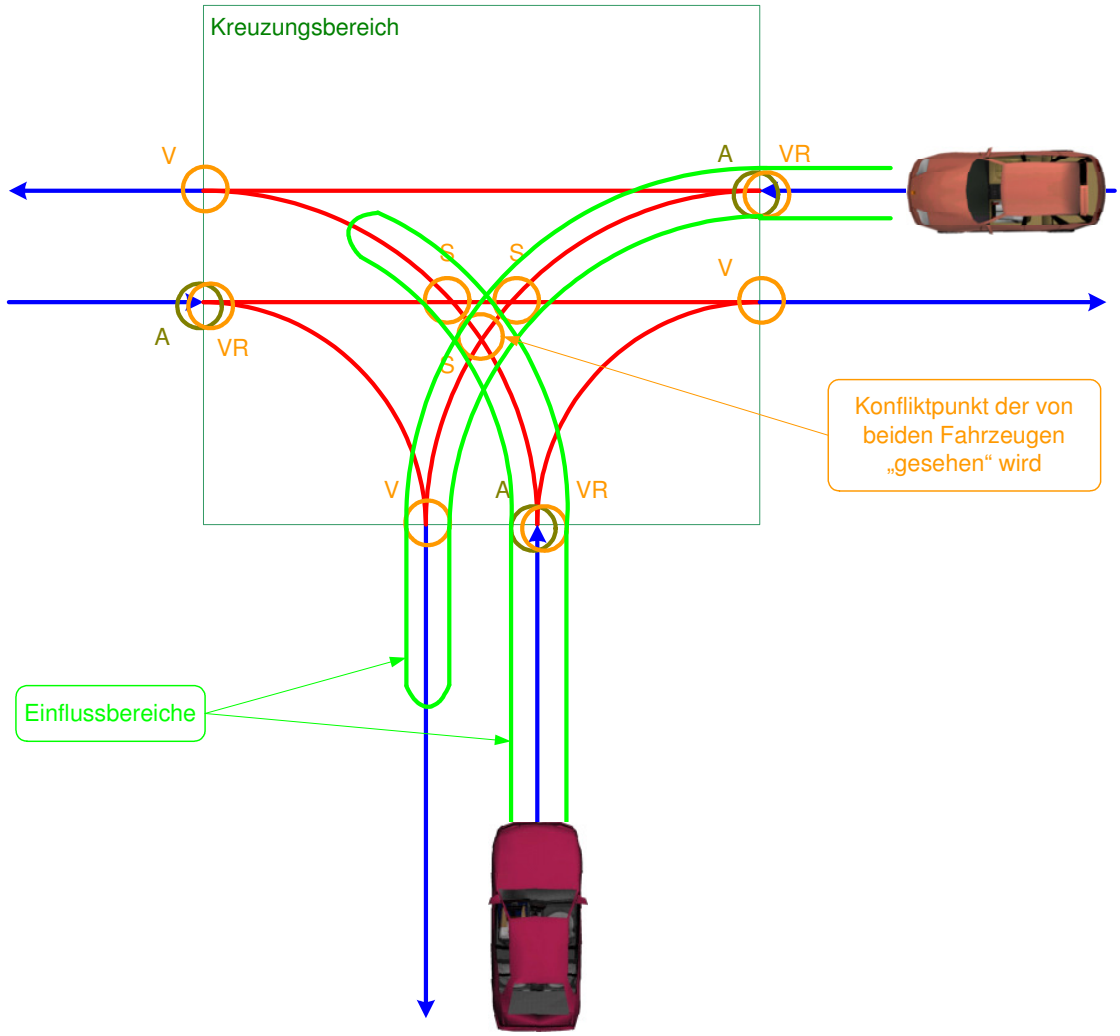


Abb. 11: Einflussbereich

Der Einflussbereich wird einerseits so groß gewählt, dass Fahrzeuge auf Konfliktpunkte rechtzeitig reagieren können, aber andererseits so klein als möglich, um den Rechenaufwand niedrig zu halten.

Berechnet wird er durch die Summe von

- Reaktionsweg = $1 \cdot v$ = Reaktionszeit * Geschwindigkeit
- Bremsweg = $\frac{v^2}{2a}$ = der Weg bis zum Stillstand eines Fahrzeuges. Dieser hängt von der individuellen Bremsstärke des Fahrers ab.
- Sicherheitsfaktor = 5 = allgemeiner Sicherheitswert und jene Entfernung, die ein Fahrzeug bei Stillstand vorausschaut.

$$EB = 5 + 1 \cdot v + \frac{v^2}{2a}$$

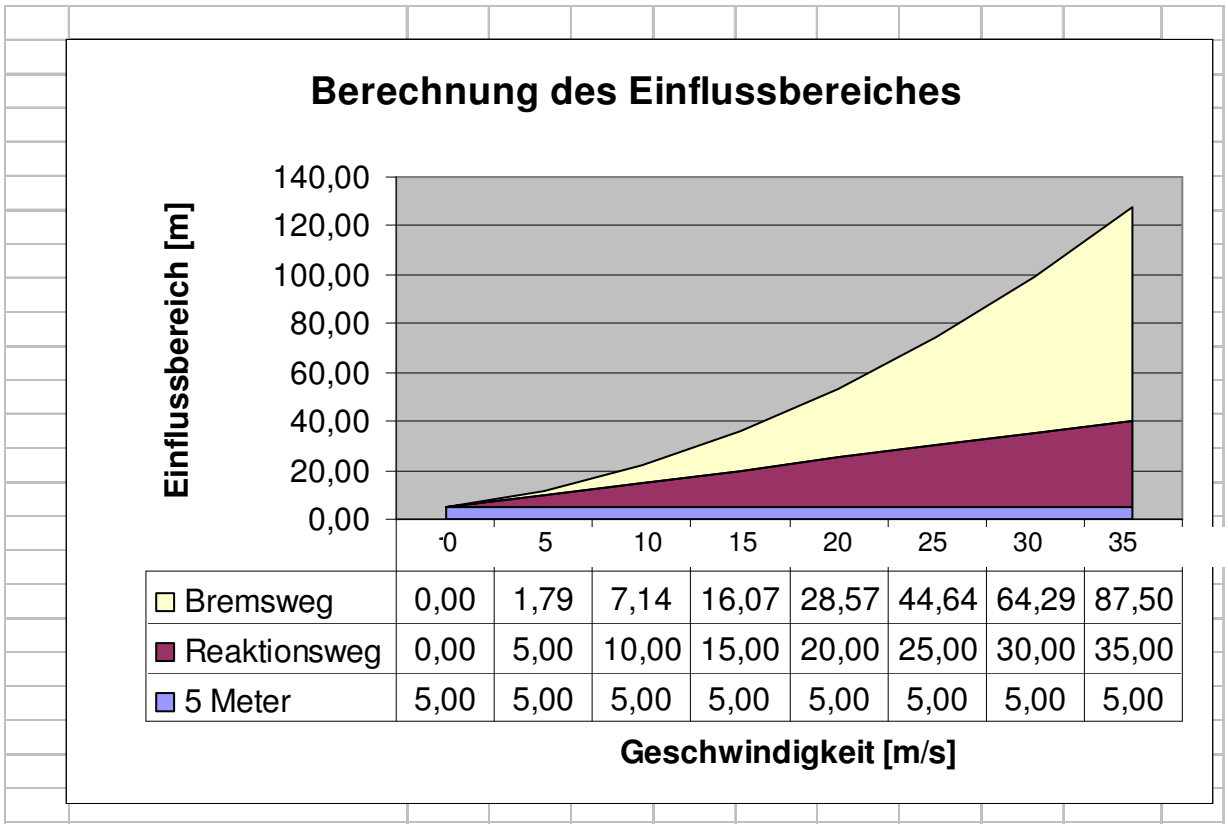


Abb. 12: Einflussbereich bei einer Bremsverzögerung von 7 m/s^2

Eine andere Methode zum Erleichtern des Auffindens von in Konflikt stehenden Fahrzeugen wird in [8] beschrieben. Dort werden Straßen in kleinere Sektoren zerlegt.

4.2.3 Verkehrsregelung und Unfallvermeidung

Die Verkehrsregelung und Unfallvermeidung behandelt das Zusammenspiel von Fahrzeugen und Konfliktpunkten. Man kann diesen Vorgang auch in „vehicle-to-network“ und „vehicle-to-vehicle“ Steuerung unterteilen [9].

Alle Fahrzeuge schauen den für sie berechneten Einflussbereich voraus. Befinden sich in diesem Bereich weitere Fahrzeuge und/oder Konfliktpunkte, so beeinflussen diese das Fahrverhalten:

- Fahrzeug voraus: Es wird versucht, dem Fahrzeug zu folgen, d.h. einen Ideal-Abstand zum Vorderfahrzeug einzuhalten. Bei dichterem Verkehr ergeben sich dadurch „Platoons“ (= Kolonnen) [11]. Z.B. wenn ein Fahrzeug aufgrund einer Roten Ampel seine Geschwindigkeit verringern muss und zum Stillstand kommt, müssen sich die Nachfolgenden adäquat verhalten. Schaltet die Ampel auf Grün um, so beginnt die gesamte Kolonne sich in Bewegung zu setzen. Das Fahrverhalten aller Fahrzeuge, mit Ausnahme des ersten, wird durch ihr Vorderfahrzeug bestimmt.
- Konfliktpunkte zur Verkehrsregelung: Zu diesen Konfliktpunkten zählen Ampeln, Verkehrstafeln und die Rechtsregel. Muss ein Fahrzeug aufgrund einer Roten Ampel oder eines Stoppschildes oder aufgrund von Nachrang gegenüber dem Rechtskommenden anhalten, so muss dies vor diesem Konfliktpunkt geschehen. Bei einer Ampel oder bei Verkehrstafeln liegt dieser Punkt außerhalb des

Kreuzungsbereiches. Bei Anwendung der Rechtsregel darf das Fahrzeug nur so weit in die Kreuzung einfahren, dass es den bevorrangten Verkehr nicht behindert.

- Konfliktpunkte zur Unfallvermeidung: Zu diesen Konfliktpunkten zählen Fahrstreifenvereinigungen und Fahrstreifenaufspaltungen. Hat ein Fahrzeug einen solchen Konfliktpunkt im Einflussbereich, so muss es sich mit allen anderen Fahrzeugen die den gleichen Konfliktpunkt im Einflussbereich haben arrangieren. In TrafLib wird dies durch das Reißverschlussprinzip gelöst, das heißt alle Fahrzeuge die einen solchen Konfliktpunkt sehen fahren abwechselnd hintereinander über diesen Punkt.

Der folgende Entscheidungsbaum stellt dar, wie ein Fahrzeug auf sein Vorderfahrzeug reagieren muss, und welche Konfliktpunkte in seinem Einflussbereich beachtet werden müssen. Der Entscheidungsbaum kann die Anzahl der Konfliktpunkte, die für das Fahrverhalten maßgebend sind, entscheidend reduzieren.

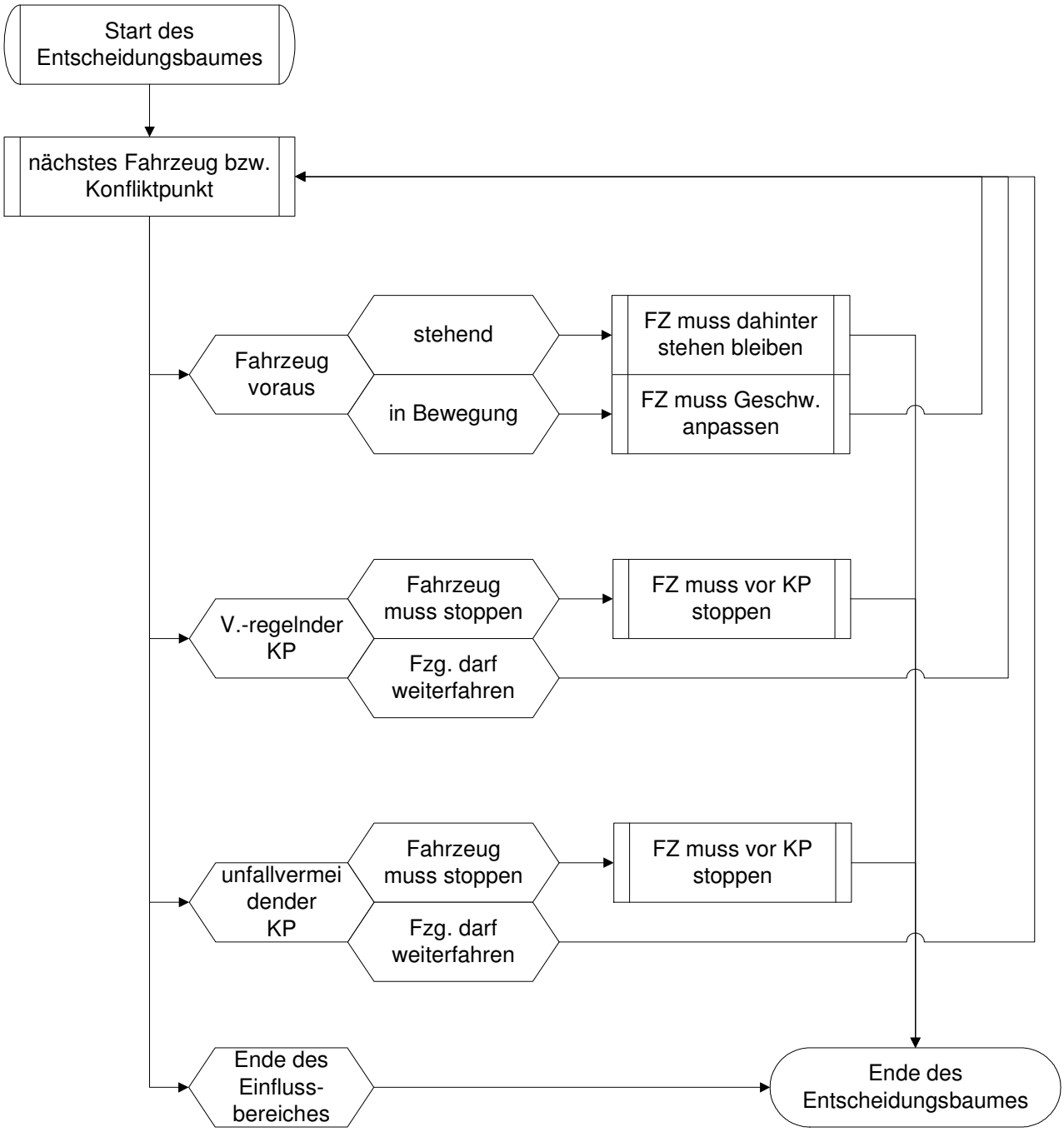


Abb. 13: Einflussbereich-Entscheidungsbaum

4.3 Architektur

TrafLib ist dem Standard gemäß objekt-orientiert struktiert. Eine einfachere Fallstudie dafür wird in [35] beschrieben.

Im Folgenden werden alle Klassen mit ihren Hauptfunktionen angeführt und in Beziehung gesetzt. Dadurch erhält man auch einen guten Überblick über den Ablauf der Verkehrssimulation.

1. cRoadGeometry (optional)

   - liest Straßengeometrie im VRMG-Format [1] ein
   - konvertiert Straßengeometrie in ein Straßennetz
   - speichert ein Straßennetz im TrafLib-Format

2. cRoadNetwork
   - liest ein Straßennetz im TrafLib-Format ein
   - ergänzt Korridore
   - ergänzt Ampeln
   - ergänzt Verkehrstafeln

   } falls nicht in der Datei angegeben

   - speichert das Straßennetz in der internen Datenstruktur cEnvironmentData
3. cEnvironmentAdapter: vervollständigt die interne Repräsentation des Straßennetzes
   - rundet Rails mit Hilfe von Bezier-Kurven aus
   - erzeugt Konfliktpunkte (unfallvermeidende und verkehrsregelnde)
4. cVehicles
   - fügt Fahrzeuge hinzu (optional kann man die fahrzeugspezifischen Parameter angeben, sowie Start- und Zielpunkt bzw. Start- und Zielregion)
   - entfernt Fahrzeuge bei Bedarf (zur Auflösung von Deadlocks und zur Reduzierung der Gesamtanzahl von Fahrzeugen aus Performancegründen)
   - plant Routen (wahlweise mit oder ohne Berücksichtigung des aktuellen Verkehrs)
   - berechnet die Geschwindigkeit und Bewegung der Fahrzeuge entsprechend ihrer aktuellen Beschleunigung
5. cTrafficCalculation: Simuliert sozusagen die Reaktion eines Fahrers nach einem Blick auf das Verkehrsgeschehen
   - berechnet den Einflussbereich aller Fahrzeuge
   - ermittelt die beeinflussenden Konfliktpunkte
   - berechnet die Geschwindigkeitsbegrenzungen hinsichtlich aller Konflikte und dem Hintereinanderfahren
   - berechnet die Beschleunigung für den nächsten Simulationsschritt
6. Schritt 4 und 5 werden kontinuierlich bis zum Ende der Simulation aufgerufen.

Die oben erwähnten Klassen werden im folgenden Diagramm (Abb. 14) übersichtlich dargestellt. Die vier Datenstrukturen cRGData, cRNData, cEnvironmentData und cVehicleData sind im Kapitel 5.3 abgebildet und erklärt.

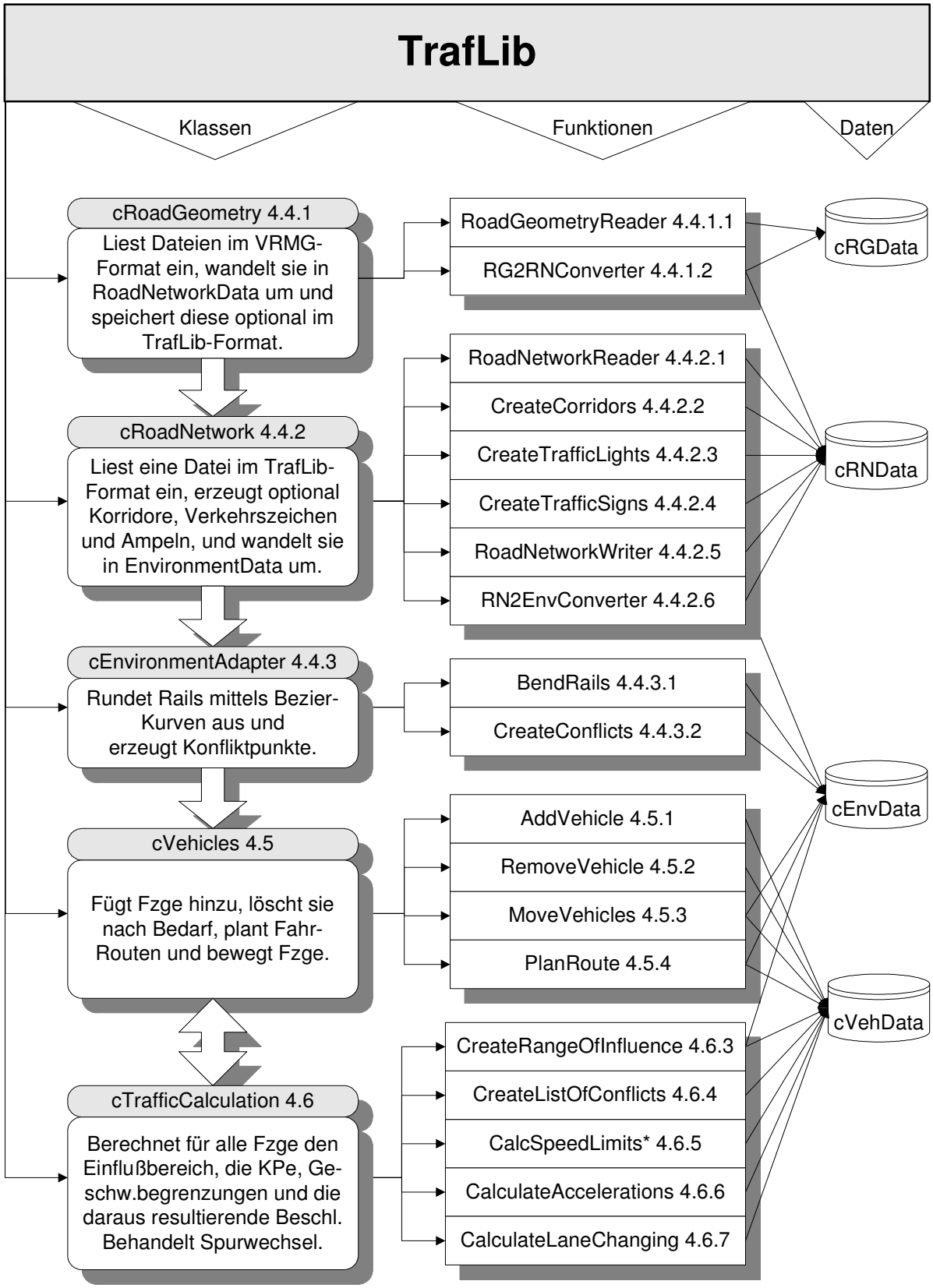


Abb. 14: Objektübersicht von TrafLib

4.4 Straßennetzwerk

Ein Straßennetzwerk im Sinne von TrafLib umfasst Straßen, Kreuzungen, Verkehrszeichen und Ampeln.

TrafLib wurde im Hinblick auf die Zusammenarbeit mit VRMG [1] entwickelt. Um die Straßennetzwerke von VRMG und TrafLib zu unterscheiden, werden im Folgenden drei verschieden Begriffe verwendet, die hier definiert und in Abb. 15 dargestellt werden.

- Straßengeometrie: Bezeichnet Straßennetzwerke im VRMG-Format. Diese beinhalten geometrische Informationen zum Zeichnen von Straßen und Kreuzungen, Fahrbahn-Mittellinien, Fahrbahnteilern, Zebra-Streifen, Gehsteigen usw.
- Straßennetzwerk: Bezeichnet Straßennetzwerke im TrafLib-Format. Diese beinhalten nur jene Informationen, die zum Bewegen von Fahrzeugen notwendig sind, wie Fahrstreifen und verkehrsregelnde Elemente (Verkehrstafeln, Ampeln).
- Rails: Bezeichnet die Mittellinien von Fahrstreifen im TrafLib-internen Speicherformat.

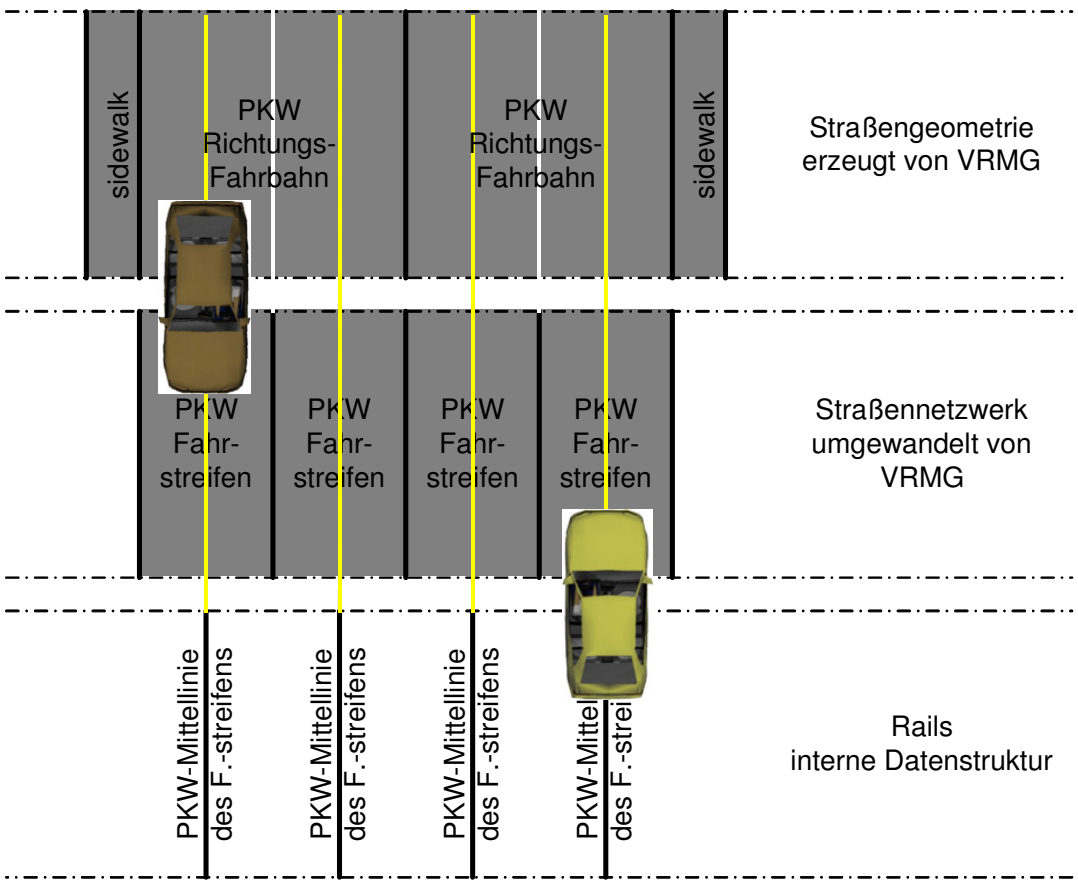


Abb. 15: Vergleich Straßengeometrie-Straßennetzwerk-Rails

Die folgenden 2 Abbildungen zeigen Rails (rote Linien) welche aus einer VRMG-Straßenbeschreibungsdatei (graue Flächen) erzeugt wurden und vergleichen diese.

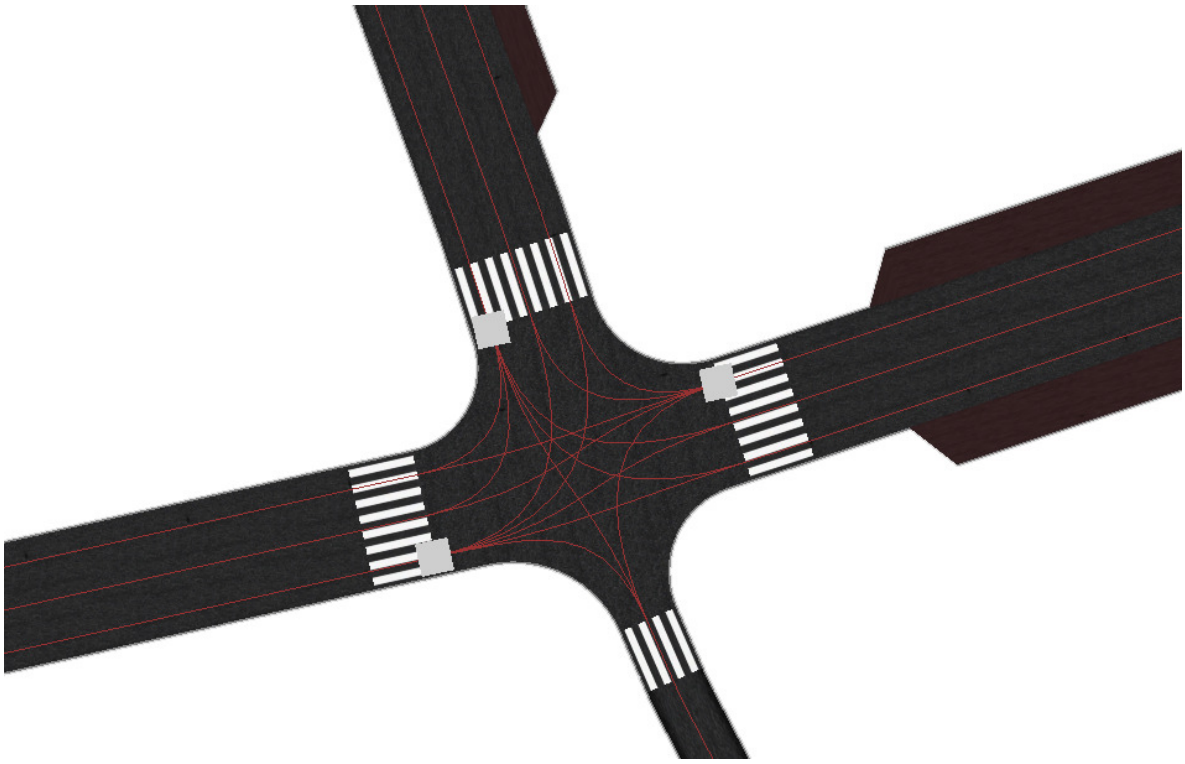


Abb. 16: Vergleich VRMG-TrafLib - Kreuzungsbereich

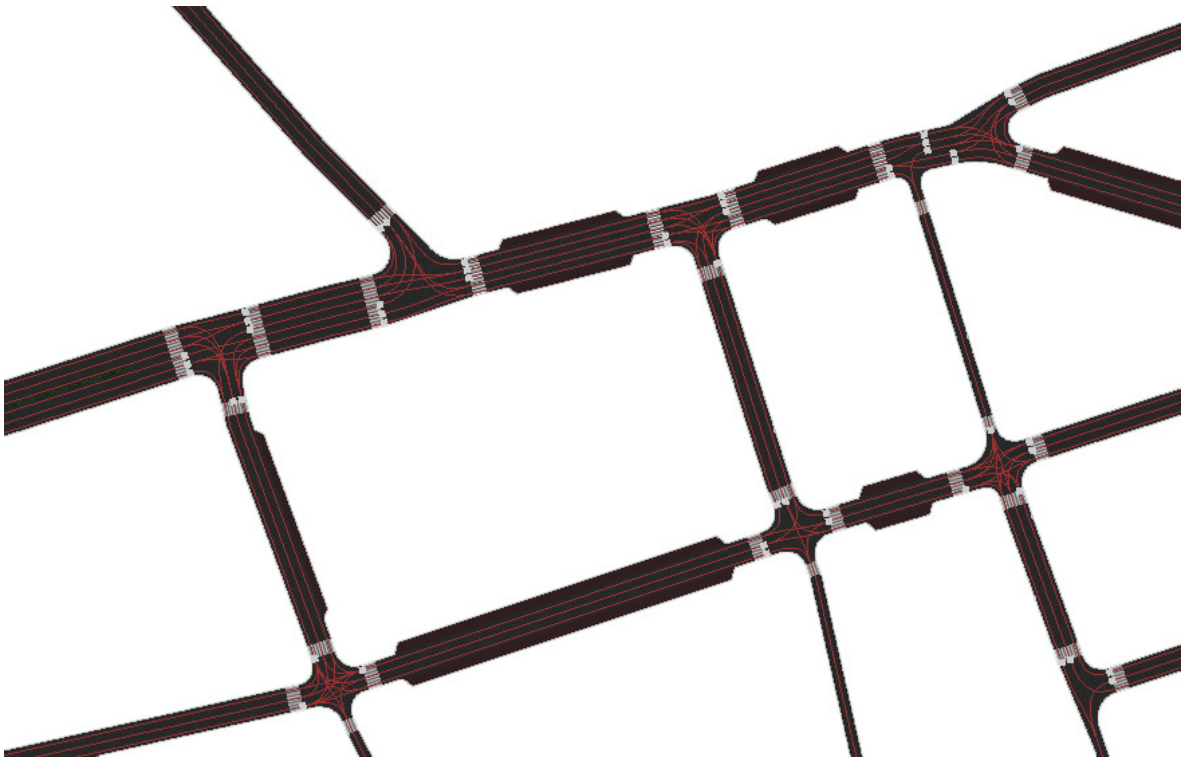


Abb. 17: Vergleich VRMG-TrafLib - Übersicht

4.4.1 Straßen-Geometrie (cRoadGeometry)

In diesem Schritt wird eine Datei mit Straßengeometrie eingelesen, in ein Straßennetzwerk umgewandelt und gespeichert. VRMG steht für „Virtual Road todo Generator“ und wurde

von Gerald Hummel im Rahmen einer Diplomarbeit beim Institut für Computergraphik an der TU Wien entwickelt [1]. Aufgabe des VRMG ist die Erstellung von Straßengeometrie für große Städte oder Stadtteile zur Verwendung in Stadtvisualisierungsprogrammen. Derzeit wird es im bereits erwähnten URBANVIZ eingesetzt. Dadurch kann TrafLib große, von VRMG erzeugte Straßennetze, verwenden.

Das Umwandeln von Geometrie-Dateien muss nur einmal durchgeführt werden. Bei jedem Programmstart muss nur das bereits umgewandelte Straßennetzfile eingelesen werden.

Dieser Schritt ist optional, da natürlich Straßennetz-Dateien auch händisch erzeugt werden können.

Zuständig für die Durchführung dieser Schritte ist die Klasse cRoadGeometry, mit ihren Funktionen „RoadGeometryReader“ und „RG2RNConverter“, die in den nächsten Unterkapiteln erläutert werden.

4.4.1.1 Straßengeometrie einlesen (RoadGeometryReader)

Es wird eine Straßengeometrie-Beschreibungsdatei eingelesen und in der Daten-Klasse RGData (siehe Abb. 23) gespeichert. Das Format muss der Definition von VRMG entsprechen, welches bei [1] in der „Erweiterten Backus-Naur Form“ (EBNF) so definiert ist:

```
ROADNETWORK = { ROAD | CROSSING }
ROAD = road { road_id UINT { ROADPARTS }+ }
ROADPARTS = CT
{     start_midpoints POINT
      {     [ ROADLANES ]
      }
}
ROADLANES = LT
{
      lane_id UINT
      mode MOD
      width DOUBLE
      [ texture   STRING ]
      [ size      DOUBLE ]
      [ align     AT ]
      [ height    DOUBLE ]
      [ edge      ET [ intervals UINT ] [ tension DOUBLE ] [ PL ] ]
      [ dimensions      D ]
}
CROSSING = crossing
{
      crossing_id UINT
      connections
      {
            [ CONNECTIONS ]
      }
      [ texture   STRING ]
      [ size      DOUBLE ]
      { island    POLYGON }
}
CONNECTION = TT
{
      from  UINT CON_STATUS
      to    UINT CON_STATUS
      [ radius    DOUBLE ]
      [ texture   STRING ]
      [ size      DOUBLE ]
      { polyline  UINT [ PL ] [ dimensions D ] }
```

```
}
POLYGON =
{
      [ texture         STRING ]
      [ curbtexture     STRING ]
      [ height          DOUBLE ]
      [ dimensions      D ]
      points      PL
}
UINT = unsigned integer
DOUBLE = double
CON_STATUS = start | end
TT = normal | tcrossing | corner
POINT = DOUBLE DOUBLE [DOUBLE]
D = 2 | 3
CT = straight | circle
MOD = c | w | s
LT = pkw | bike | parking | island | tram | sidewalk
ET = parallel | spline | polyline
AT = not | left | right
PL = [ POINT { , POINT } ]
```

Bedeutung der Symbole und Parameter:

```
[ ]:     optionales Auftreten des geklammerten Arguments
{}:      beliebig häufiges Auftreten des geklammerten Arguments
{}+:     mindestens ein Auftreten des geklammerten Arguments
|:       Auswahloperator (enweder-oder)
unsigned int:    Ganzzahl größer oder gleich Null
double:  Realzahl mit doppelter Genauigkeit
id:      eine einzigartige Identifikationsnummer vom Typ UINT
size:    Größe einer quadratischen(!) Texture in (Maß-)Einheiten
         (meter), (default ist 1)
point:   x,y- oder x,y,z-Werte eines Punktes(2D oder 3D, nach Kontext)
D:       Dimension einer Polyline, default ist 2D. Der z-Wert wird nur
         verwendet, wenn "dimension 3" angegeben wird
width:   Distanz von der des Fahrstreifenrandes zur Mittelline
         („Breitenposition“)
height:  relative Höhendifferenz zwischen aufeinanderfolgenden
         Fahrstreifen oder für Verkehrsinseln als relative
         Höhendifferenz zur Fahrbahn
radius:  Für Abrundungsbögen als Radius, ein Wert von 0 (default), führt
         zu einer programminternen Berechnung mit Verwendung des
         Maximalradius
CT:      Segmentverlauf zum nächsten Straßenmittelpunkt.
texture: Diese muss zumindest für das erste Segment am Straßenanfang in
         Form einer Bilddatei definiert werden, für undefinerte Texturen
         wird die Textur des Vorgängersegments verwendet
LT:      Verwendungszweck des Fahrstreifens z.B. „pkw“ für eine Fahrbahn
MOD:     wird zur Kennzeichnung der Gültigkeit von
         Fahrstreifenpositionen verwendet:
         c: „continuing“ - für das vorige und nächste Segment gültig
         w: „widening“ - nur für nächstes Segment gültig
         s: „shrinking“ - nur für voriges Segment gültig
ET:      Verlauf der Kante eines Fahrstreifens
parallel:  wie Mittellinie(default)
spline:  programminterne generierung einer Spline-Übergangskurve
polyline:  Definition einer beliebigen Verlaufsform durch einen
         Polygonzug
AT:      Ausrichtung der Textur auf einem Fahrstreifen
```

```
left:    an linker Kante(Wiederholung oder Schnitt rechts)
right:   an rechter Kante(Wiederholung oder Schnitt links)
not:     die Textur wird gestreckt oder gezerrt(default)
PL:      Spezifikation einer Kante durch einzelne (geordnete)
         Punktmenge(Polygonzug)
         Achtung: Bei Kreuzungen folgt auf „polyline“ ein UINT-Wert, der
         der Fahrstreifennummer des Abrundungsteils entspricht: dadurch
         wird die Übergangskante „adressiert“: 0 - innerste
         Übergangskante am Fahrbahnteil, numeriert nach außen
con_status:Straßenanschluß an die Kreuzung: start - erster
         Straßenquerschnitt, end - letzter Straßenquerschnitt
TT:      Form des Übergangsteils der Abrundung von Kreuzungen:
         normal: Der Abrundungsteil wird aus dem Randstein und einer
         Fläche zwischen Randstein und Außenrand des Abrundungsteils
         gebildet.
         t-crossing: Vorzugsweise für die durchgehende Straße bei einer
         Einmündung, dabei werden (sofern möglich) alle gemeinsamen
         Streifen der beiden beteiligten Straßenäste fortgeführt.
         corner: Wie Typ -normal-, allerdings wird ein
         Eckpunkt(=Schnittpunkt der Außenkanten der beiden beteiligten
         Straßenäste) verwendet.
Spline-Parameter:
         intervals: Verfeinerungswert für Spline-Kurve, dabei wird das
         Segment in n-Teile gleichmäßig unterteilt und für jede
         Intervallgrenze ein Stützpunkt berechnet(default ist 10).
         tension: Wert kleiner als 1 ergeben bikubischen Spline, große
         Werte approximieren eine Polyline(default ist 1).
```

Beispiel einer Straßengeometrie-Beschreibungsdatei die der EBNF-Syntax von VRMG entspricht:

```
road {
  road_id 2
  straight {
    start_midpoints 0.885831 3.92606 {
      sidewalk {
        lane_id 0
        mode c
        width 8
        texture "../textures/Road/pave.jpg"
      }
      pkw {
        lane_id 1
        mode c
        width 7.8
        texture "../textures/Road/as_stripe2_neutr2.jpg"
        size 10
        height -0.2
        }
        pkw {
          lane_id 5
          mode c
          width 0
          texture "../textures/Road/as_stripe2_neutr2.jpg"
          size 10
          height 0
        }
      sidewalk {
        lane_id 2
        mode c
```

```
        width -7.8
        texture "../textures/Road/pave.jpg"
        height 0.2
      }
      sidewalk {
        lane_id 3
        mode c
        width -8
        texture "../textures/Road/pave.jpg"
      }
    }
  }
  straight {
    …snip…
  }
  straight {
    …snip…
  }
}
…snip…

crossing {
  crossing_id 12
  connections {
    normal {
      from 18 end
      to   19 end
    }
  }
  texture "../textures/Road/as_blank_neutr.jpg"
  size 10
}
…snip…
```

4.4.1.2 Straßengeometrie → Straßennetzwerk (RG2RNConverter)

Die zuvor eingelesene Straßengeometrie (cRGData) muss nun in ein zu TrafLib kompatibles Straßennetzwerk (cRNData) umgewandelt werden.

Folgendes findet bei der Umwandlung statt:

- Umwandlung der Straßen- und Kreuzungsinformationen.
- Entfernung von Texturinformationen, da sie für die Bewegung von Fahrzeugen irrelevant sind.
- Entfernung der Parameter „mode“ und „size“.
- Entfernung aller Straßenflächen die nicht für Pkws vorgesehen sind (Gehsteige, Fahrradwege, ...).
- Aufteilung der Richtungsfahrbahnen in Fahrstreifen anhand der Mindestbreite von 2,5 m. Wenn z.B. im Geometrie-File eine Breite von 8 m für eine Pkw-Fahrbahn angegeben wird, so wird diese in 3 * 2,67 m breite Fahrstreifen unterteilt.

4.4.1.3 Straßennetzwerk speichern (RoadNetworkWriter)

Das im vorigen Kapitel erzeugte Straßennetzwerk wird nun in einer Datei gespeichert. Die verwendete Funktion dafür ist ein Teil der Klasse cRoadNetwork, welche im Kapitel 4.4.2.5 erklärt wird.

4.4.2 Straßennetzwerk (cRoadNetwork)

In diesem Schritt wird eine Straßennetzwerk-Beschreibungsdatei eingelesen, falls notwendig um Korridore, Ampeln und Verkehrszeichen erweitert und wieder gespeichert. Danach wird das Straßennetz in eine geeignete Datenstruktur (cEnvironmentData) für die Bewegung von Fahrzeugen umgewandelt. Die zuständige Klasse ist cRoadNetwork (s. Abb. 14).

Ähnlich wie bei VRMG [1] ist auch in TrafLib das Format der Straßennetzwerks-Beschreibungsdatei durch eine EBNF definiert:

Die EBNF Syntax für ein Straßennetz-Beschreibungsfile sieht folgendermaßen aus:

```
#File format definition for the road network model of TrafLib

ROADNETWORK = { ROAD | CROSSING }+  // list of roads and crossings

ROAD = "road      {"                // one road, from crossing to crossing
      "road_id" ID                  // ID of this road
      "from_id" ID                  // starts in crossing ID
      "to_id" ID                    // ends in crossing ID
      "number_segments" UINT        // number of segments in this road
      "midpoints" { POINT3D }+      // (nr_segments + 1) midpoints
      "road_segments"   "{"
            { ROADSEGMENTS }+       // segments of road
      "}"
"}"

ROADSEGMENTS = CT "{"               // shape of road
      "segm_id" ID                  // ID of segment
      "traffic signs" "{"
      {TRAFFIC_SIGNS}*              // signs along the road; optional
      "}"   "traffic_lights" "{"
            {TRAFFIC_LIGHTS}*       // traffic lights; optional
      "}"
      "number_lanes" UINT           // number of lanes at this profile
      "widths_begin" "{"            // postion of lanes at segment-start
            { FLOAT }+              // -> must equal (number_lanes + 1)
      "}"
      "widths_end" "{"              // position of lanes at segment-end
            { FLOAT }+              // -> must equal (number_lanes + 1)
      "}"
      "roadlanes" "{"   [ROADLANES] "}"   // description of lanes
"}"

ROADLANES = LT "{"                  // usage of lane
      "lane_id" ID                  // id, also used in CORRIDOR
      "mode" MOD                    // mode of lane
      "next_lane_id" "{" ID "}"     // connection to lane of next road segm.
      "prev_lane_id" "{" ID "}"     // connection to lane of prev. road segm
      "height" FLOAT                // height of lane
      "direction" 0/+1/-1           // direction of flow
"}"

TRAFFIC_SIGNS = TST "{"             // type of traffic sign
      "sign_id" ID                  // id of sign
      "position" POINT2D            // 2 distances to position the sign
      "attribute" UINT              // i.e. speed of speed limit sign
"}"

TRAFFIC_LIGHTS = TLT "{"            // type of traffic light
      "traffic_light_id" ID         // id of traffic light
      "position" POINT2D            // like position of SIGNS
```

```
      "affected_lanes" { ID }+      // list of affected lane_ids
      "switching_times" { UINT }7   // 7 numbers are necessary*
"}"

CROSSING = "crossing"   "{"         // definition of one crossing
"crossing_id" ID                    // id of the crossing
      "corridors" "{"               // corridors across a crossing; optional
            { CORRIDOR }+           // list of corridors
      "}"
 "}"

CORRIDOR = „corridor {"             // type of corridor
      "corridor_id" ID              // id of corridor
      "from_road" ID                // road id of start road
      "from_lane" ID                // lane id of ------"---
      "to_road" ID                  // road id of end road
      "to_lane" ID                  // lane id of ------"-----
"}"

UINT =      unsigned integer
ID =        UINT
DOUBLE =    double
POINT2D =   DOUBLE DOUBLE
POINT3D =   DOUBLE DOUBLE DOUBLE
CT =        "straight" | "circle"
LT =        "pkw" | "sidewalk"
MOD =       "c" | "w" | "s"
TST =       "stop" | "giveway" | "speedlimit"
TLT =       "3lights" | "leftarrow" | "rightarrow"

*) traffic light - switching times: 1=time-offsetwhen the traffic light
starts it's period with green. 2=how long the traffic light stays green.
3=how long green flashing. 4=orange. 5=orange flashing. 6=red. 7=off.
Always in seconds
```

Bedeutung der Symbole und Parameter, teilweise von [1] übernommen:

[ ]: optionales Auftreten des geklammerten Arguments

{}: beliebig häufiges Auftreten des geklammerten Arguments

{}+: mindestens ein Auftreten des geklammerten Arguments

|: Auswahloperator (entweder-oder)

CT: „curve type", gibt die Form eines Segments an

LT: „lane type", gibt den Verwendungszweck eines Fahrstreifens an

MOD: „modus", wird zur Kennzeichnung der Gültigkeit von Fahrstreifen verwendet:
- c: „continuing": für das vorige und nächste Segment gültig
- w: „widening": nur für das nächste Segment gültig
- s: „shrinking": nur für voriges Segment gültig

TST: „traffic sign type", gibt den Typ des Verkehrszeichens an

TLT: „traffic light type", gibt den Typ der Verkehrs-Ampel an

Beispiel einer Straßennetzwerk-Beschreibungsdatei die der EBNF-Syntax von TrafLib entspricht:

```
road {
```

```
  road_id 0
  from_id 0
  to_id 1
  number_segments 3
  midpoints {
    -20.0, 0.0, 0.5
    -40.0, 5.0, 0.5
    -40.0, 50.0, 0.5
    -20.0, 50.0, 0.5
  }
  road_segments {
    straight {
      segm_id 0
      number_lanes 6
      widths_begin {
        9 6 3 0 -3 -6 -9
      }
      widths_end {
        9 6 3 0 -3 -6 -9
      }
      road_lanes {
        pkw {
          lane_id 0
          next_lane_id 0
          prev_lane_id 0
          height -0.5
          direction 0
        }
        pkw {
          lane_id 1
          mode c
          next_lane_id 1
          prev_lane_id 1
          height -0.5
          direction 0
        }
        …snip…
      }
    }
    straight {
      segm_id 1
        … snip …
      }
    }
  }
}
crossing {
  crossing_id 0
}
```

4.4.2.1 Straßennetzwerk einlesen (RoadNetworkReader)

Liest ein Straßennetzwerk im TrafLib-Format aus einer Datei ein. Wie bereits erwähnt, ist die Angabe von Korridoren, Verkehrszeichen und Ampeln optional.

4.4.2.2 Korridore erzeugen (CreateCorridors)

Erstellt eine Liste aller Rails von allen Straßen, die in eine Kreuzung münden bzw. aus ihr herausführen. Weiters wird ermittelt, welche Rails miteinander verbunden werden. Die verbundenen Rails ergeben die möglichen Routen über eine Kreuzung und werden in TrafLib

als Korridore bezeichnet. Folgende Formel ermittelt die ausgehenden Rails für die Rails einer einmündenden Straße, siehe auch Abb. 18:

$$in = round\left(\frac{\sum in - 1}{\sum out - 1} \cdot out + C\right)$$

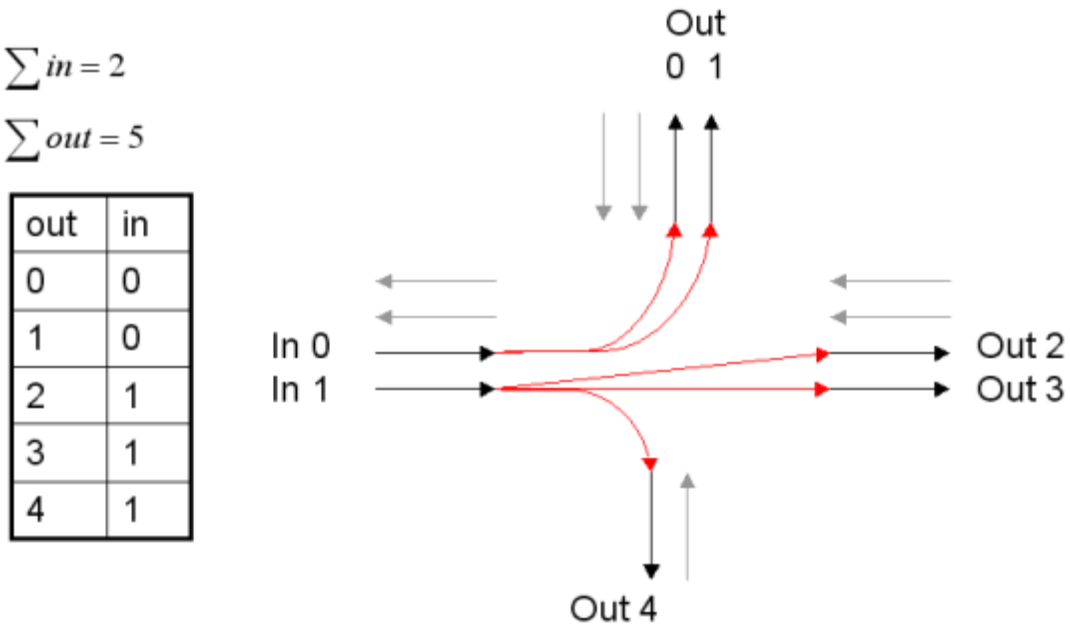
in ... Index einer eingehenden Rail

out ... Index einer ausgehenden Rail

$\sum in$ ... Summe der eingehenden Rails

$\sum out$ ... Summe der ausgehenden Rails

C ... Konstante = 0.2



out	in
0	0
1	0
2	1
3	1
4	1

Abb. 18: Generierung von Korridoren

Wird die Konstante $C = 0.2$ aus obiger Formel weggelassen, werden die eingehenden Fahrstreifen auf alle ausgehenden Fahrstreifen gleichmäßig verteilt. In der Realität gibt es aber eine Bevorzugung der Rechtsabbiegespuren, da aufgrund der Rechts-Regel das Rechtsabbiegen weniger verkehrsbehindernd wirkt als das Linksabbiegen.

Einfluss der Konstanten C:

- ± 0.00 → gleichmäßige Aufteilung
- - 0.49 → Dadurch würden mehr ausgehende Fahrstreifen mit der oder den linken eingehenden Fahrstreifen verbunden werden. Dies hätte den Nachteil, dass Linksabbiegespuren öfter mit geradeaus führenden Spuren zusammengelegt werden würden und der Verkehrsfluss leichter ins Stocken käme.
- + 0.49 → Dadurch würden Rechtsabbiegespuren übermäßig oft mit geradeaus führenden Fahrstreifen zusammengelegt werden.

- + 0.30 → Bei $C = 0.3$ ist der Effekt, der bei $C = 0.49$ entsteht, immer noch zu hoch. Z.B. bei 2 eingehenden und 5 ausgehenden Fahrstreifen würden 4 der 5 Fahrstreifen mit dem rechten der beiden einmündenden Fahrstreifen verbunden werden.
- + 0.2 → Scheint die Realität am ehesten abzubilden. Dazu ist aber zu bemerken, dass in der Realität die Verbindung zwischen eingehenden und ausgehenden Fahrstreifen vom Verkehrsaufkommen abhängt. Zum Beispiel gibt es oft Linksabbiege-Verbote, da Linksabbieger entweder auf der eigenen Fahrspur Staus verursachen können, oder bei Linksabbiege-Ampeln die Wartezeit des Gegenverkehrs erhöhen. TrafLib hat über das Verkehrsaufkommen keine Information.

In seltenen Fällen kommt es vor, dass es mehr eingehende als ausgehende Fahrstreifen gibt. In diesem Fall muss die obige Formel umgedreht werden:

$$out = round\left(\frac{\sum out - 1}{\sum in - 1} \cdot in + C\right)$$

4.4.2.3 Ampeln erzeugen (CreateTrafficLights)

Die erste Ampel wurde 1926 in Großbritannien installiert, 1966 gab es bereits 4000 Stück davon. Manuell bediente „Signalmasten“ gab es ab 1868. [21].

In der Realität macht die steigende Anzahl von städtischem Straßenverkehr eine höhere Anzahl von Verkehrsampeln erforderlich. Ampeln verursachen aber Verzögerungen und Stopps. Diese müssen für einen optimalen Verkehrsfluss die Verzögerungen und Stopps minimiert werden, denn sie sind für die Fahrer ärgerlich, teuer und bedeuten einen Zeitverlust. Außerdem tragen sie auch in hohem Maße zur Luftverschmutzung bei. Straßenbau-Ingenieure versuchen die Ampelsteuerung zusammenhängender Kreuzungen zu koordinieren (z.B.: Grüne Welle), um eine Gruppe von Fahrzeugen über mehrere Kreuzungen möglichst ohne Stopps und einem Minimum an Verzögerungen zu bewegen [26].

Man unterscheidet grundsätzlich zwei Typen von Ampeln: solche mit fixem Ampelphasen-Zyklus, und solche mit einem dem Verkehrsaufkommen angepassten Zyklus. Im zweiten Fall kann der durchschnittliche Verkehrsfluss gesteigert werden. Eine sogenannte Grüne Welle reduziert die Anzahl der Anhalten-Anfahren-Zyklen weiter und ist mit beiden Ampel-Systemen möglich [21].

In TrafLib ist nur die Methode mit fixen Zyklen implementiert. Zusätzlich werden mit Hilfe eines Algorithmus bei der automatischen Generation von Ampeln eine oder mehrere „Hauptstraßen“ gesucht, und für diese die Ampelschaltungen so angepasst, dass Grüne Wellen stadtauswärts entstehen. Dies wird auch in der Realität angewendet um den Verkehr schneller aus einer Stadt hinauszuleiten.

Nachdem TrafLib keine Information über das Verkehrsaufkommen hat, werden Verkehrsampeln aufgrund der Fahrstreifen-Anzahl der Straßen generiert. Betrachtet man das Straßennetz einer Stadt aus der Vogelperspektive, erkennt man, dass es immer wieder größere „Durchzugsstraßen“ gibt, und kleinere Straßen, die den Bereich dazwischen „ausfüllen“, siehe z.B. Abb. 2. Daher wird in TrafLib folgende Methode zur Platzierung von Ampeln angewendet:

1. Suche einen Straßenzug mit möglichst vielen Fahrstreifen, der den gesamten Straßennetz-Ausschnitt durchläuft. Wird ein solcher Straßenzug gefunden, werden Ampeln von der Mitte dieses Straßenzuges stadtauswärts so gesetzt, dass eine grüne Welle in diese Richtung entsteht.

2. Suche einen Straßenzug mit möglichst vielen Fahrstreifen, der am Rande des Straßennetz-Ausschnittes endet (Ausfallstraße). Danach wird eine Grüne Welle Richtung Ausschnitt-Rand erzeugt.
3. Erzeuge bei allen weiteren Kreuzungen nach einem Zufallsgenerator in Abhängigkeit der Anzahl der Fahrspuren Ampeln: $p = (Anz_{Fahrstreifen} - 4) * 3\%$. Dies bedeutet: Ist die Summe der Fahrstreifen, die in eine Kreuzung hinein- bzw. hinausführt unter 5, so wird keine Ampel erzeugt. Für jeden zusätzlichen Fahrstreifen steigt die Wahrscheinlichkeit für das Generieren einer Ampel um 3 Prozent.

Bei allen Ampeln, die in diesen 3 Schritten erzeugt werden, werden eigene Linksabbiege-Ampeln generiert, falls der Linksabbieger mehr als 1 entgegenkommende Fahrspur kreuzen muss. In diesem Fall bekommt auch jeweils der Gegenverkehr eine eigene Linksabbiege-Ampel.

In den seltenen Fällen von 5- oder mehrarmigen Kreuzungen werden in TrafLib keine Ampeln gesetzt.

In TrafLib werden drei Ampeltypen unterschieden, die sich aber nur auf die Visualisierung auswirken. Siehe auch den „traffic light type“ TLT aus der Straßennetzwerk-Beschreibungsdatei in Kapitel 4.4.2:

- „3lights“: Ampel mit 3 Lichtern (Grün-Orange-Rot)
- „leftarrow“: Ampel mit grünem Licht und Pfeil nach links
- „rightarrow“: Ampel mit grünem Licht und Pfeil nach rechts

4.4.2.3.1 Ampelphasen

Längere Grün-Phasen erhöhen zwar den Durchsatz an Fahrzeugen über eine Kreuzung, sie dürfen aber nicht zu lange dauern wegen des Querverkehrs [26].

Da in TrafLib, wie bereits erwähnt, zum Zeitpunkt der Ampel-Generierung nichts über das Verkehrsaufkommen bekannt ist, werden für die Ampelphasen Durchschnittszeiten verwendet. Der Zyklus einer Ampelphase sollte nicht < 25 Sekunden sein [21].

Ampeln durchlaufen folgende Phasen in dieser Reihenfolge:

Grün – Grün blinkend – Orange – Orange blinkend – Rot – Ausgeschaltet

Jede Ampel muss eine Startzeit und für jede Ampelphase eine Zeitdauer gespeichert haben. Kommt eine Ampelphase nicht vor, so ist ihre Zeitdauer 0.

Beispiel: Intervall 30-5-5-0-20-0 bedeutet:

30 Sek. Grün, 5 Sek. Grün blinkend, 5 Sek. Orange, kein Orange blinkend, 20 Sek. Rot und danach wieder vom Anfang an, 30 Sek. Grün...

4.4.2.4 Verkehrstafeln erzeugen (CreateTrafficSigns)

Das Erzeugen von Verkehrstafeln ist genauso wie das Erzeugen von Ampeln und Korridoren optional, da sie bereits im Straßenbeschreibungsfile definiert sein können.

Da in TrafLib nichts über das Verkehrsaufkommen bekannt ist und da die „Hauptstraßen“ im vorhergehenden Kapitel bereits mit Ampeln versehen wurden, werden auf einem bestimmten Prozentsatz der verbliebenen Kreuzungen zufällig Verkehrstafeln verteilt.

Die Höhe dieses Prozentsatzes kann der Benutzer festlegen. Gibt er nichts an, so verwendet TrafLib 90%.

4.4.2.5 Speichern (RoadNetworkWriter)

TrafLib kann eine Straßenbeschreibungsdatei im TrafLib-Format inklusive Korridoren, Ampeln und Verkehrstafeln speichern.

Dies lässt zum Beispiel folgenden Ablauf zu:

1. großes Straßengeometrie-Netzwerk mit VRMG erzeugen
2. Straßengeometrie-Netzwerk mit TrafLib (cRoadGeometry) in Straßennetzwerk umwandeln
3. Straßennetzwerk einlesen, Korridore, Ampeln und Verkehrstafeln erzeugen und speichern
4. Korridore, Ampeln und Verkehrstafeln händisch nachbessern
5. bearbeitetes Straßennetz zur Simulation verwenden

4.4.2.6 Umwandlung (RN2EnvConverter)

Umwandeln der Datenstruktur von cRoadNetworkData zu cEnvironmentData. cRoadNetworkData repräsentiert das Straßennetzwerk in einer sehr ähnlichen Struktur wie die Straßennetzwerk-Beschreibungsdatei. cEnvironmentData ist die interne Datenstruktur für Straßen, welche sich besser für die Verkehrssimulation eignet.

4.4.3 Straßennetz-Aufbereitung (cEnvironmentAdapter)

Um das vorher eingelesene und umgewandelte Straßennetzwerk zur Simulation verwenden zu können, sind noch 2 Aufbereitungsschritte notwendig: die Ausrundung der Rails und das Erzeugen der Konfliktpunkte.

4.4.3.1 Rails ausrunden (BendRails)

Nach der Umwandlung des Straßennetzwerkes in die interne Repräsentation (s. Kapitel 4.4.2.6) fehlt den Rails an den Segmentstößen bzw. den Korridoren im Kreuzungsbereich die Ausrundung.

Die Ausrundung wird in TrafLib auf eine andere Art als in VRMG berechnet. Wie bereits in Kapitel 3.1 erwähnt, wird eine Kurve in VRMG durch eine Abfolge von kubischer Spline – Kreisbogen – kubischer Spline erzeugt. In TrafLib hingegen wurde ein einfacherer Weg gewählt, nämlich die Ausrundung mittels der Bezier-Kurven-Formel (s. Kapitel 3.4.1). Diese beiden Methoden liefern ein ausreichend übereinstimmendes Ergebnis wie in Abb. 17 aufgezeigt ist.

Zur Ausrundung wurde hinsichtlich der Zusammenarbeit mit VRMG ein Kurvenradius von 10 m gewählt, siehe Abb. 19. Dadurch ergeben sich die Punkte A und D auf der Rail. Die Strecken $\overline{AS}$ und $\overline{DS}$ werden halbiert und ergeben somit die Punkte B und C. Die Punkte A, B, C, D dienen als Stützpunkte für die Bezier-Kurve. Auf dieser Bezier-Kurve wird eine gewisse Anzahl von Punkten gewählt und in den Polygonzug (= Rail) eingefügt.

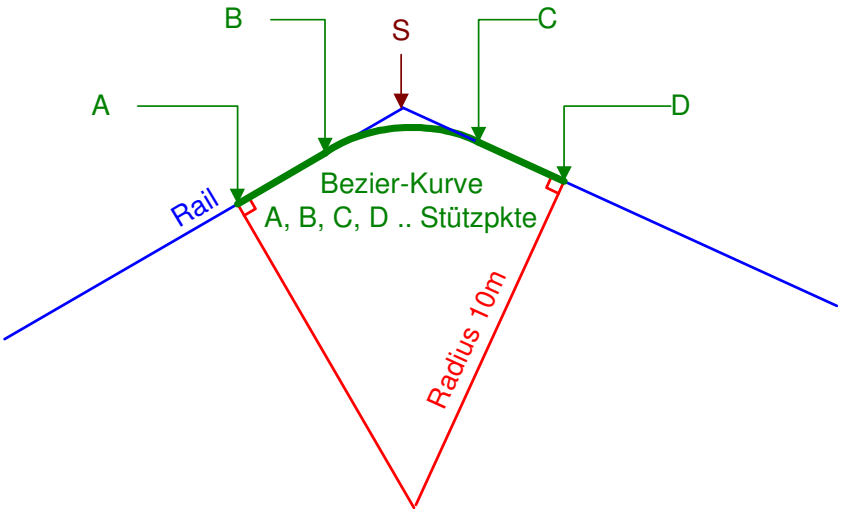


Abb. 19: Ausrundung mittels Bezier-Kurve

4.4.3.2 Konfliktpunkte erzeugen (CreateConflicts)

Dieses Kapitel behandelt die Platzierung von Konfliktpunkten.

Der weitere Vorgang, also wann Fahrzeuge einen Konfliktpunkt „sehen“ und wie sie darauf reagieren, ist im Kapitel 4.6 erklärt.

Bei Fahrspurvereinigungen, -aufteilungen und –schnitten sind Konfliktbereiche (siehe Zeichnungen im Kapitel 3.4.2) erforderlich, da Fahrzeuge innerhalb einer bestimmten Entfernung des Konfliktpunktes bereits nicht mehr nebeneinander fahren dürfen (Gefahr eines seitlichen Zusammenstoßes). Zu diesem Zweck werden Hilfskonfliktpunkte gesetzt.

Für alle drei oben genannten Fälle werden virtuelle Rails erzeugt. Diese werden nicht visualisiert und dienen der Unfallvermeidung. Liegt ein Konfliktpunkt im Einflussbereich mehrerer Fahrzeuge, so werden diese auf eine virtuelle Rail gesetzt und müssen virtuell hintereinander fahren. Näheres dazu im Kapitel 4.6.

In der folgenden Abbildung sind Konfliktpunkte und Hilfskonfliktpunkte dargestellt.

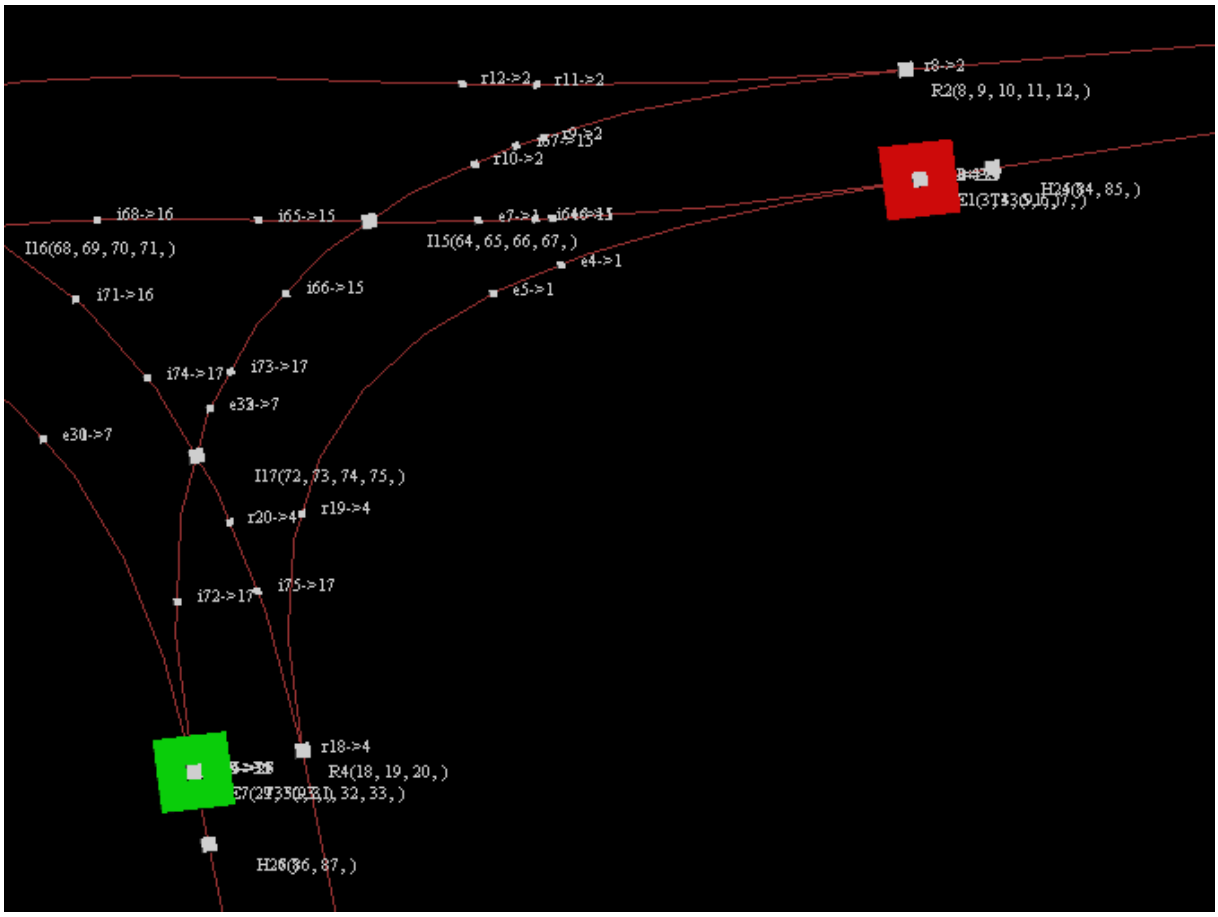


Abb. 20: Illustration von Konfliktpunkten

4.4.3.2.1 Ampeln, Stopp- und Vorrang-Geben-Tafeln

Fahrzeuge halten direkt vor Konfliktpunkten. Aus diesem Grund müssen die Konfliktpunkte ausreichend weit außerhalb des Kreuzungsbereiches platziert werden.

4.4.3.2.2 Fahrspuraufteilung, -vereinigung und -schnitt

Der Konfliktpunkt für eine Fahrspuraufteilung befindet sich direkt bei der Aufteilung, s. Abb. 10 und Abb. 20.

Da sich bei Fahrspuraufteilungen und –vereinigungen schleifende Schnitte ergibt, müssen immer Konfliktbereiche erzeugt werden, s. Abb. 7, im Gegensatz zu Fahrspurschnitten, bei denen nicht immer ein Konfliktbereich erforderlich ist, s. Abb. 8.

Bei jedem dieser Konfliktpunkte gilt die Rechts-Regel. Das bedeutet, dass ein Fahrzeug vor einem Hilfskonfliktpunkt stehen bleiben muss, wenn es Nachrang hat. Sobald es aber den Hilfskonfliktpunkt überschritten hat, wird es auf die virtuelle Rail gesetzt.

4.4.3.2.3 Weitere Verkehrstafeln, enge Kurven, etc.

Eine weitere Verkehrstafel ist zum Beispiel die Geschwindigkeits-Begrenzungstafel. Hier wird der Konfliktpunkt an die Stelle gesetzt, ab der die Verkehrstafel gelten soll.

Bei engen Kurven, die ein Geschwindigkeitslimit erfordern, wird ebenfalls ein Konfliktpunkt gesetzt. Der Kurvenradius wird berechnet, indem durch jeweils drei aufeinanderfolgende

Punkte eines Polygonzuges eine Kurve gelegt wird. Dabei ist der Kleinste aller Radien der Maßgebende.

4.5 Fahrzeuge (cVehicles)

Dieses Kapitel befasst sich mit den Eigenschaften von Fahrzeugen, dem Hinzufügen, Entfernen, Bewegen und Routenplanen. Die Bewegung umfasst nur die Berechnung der Geschwindigkeitsänderung und des zurückgelegten Weges aufgrund der Beschleunigung, welche bei der Verkehrsberechnung ermittelt wird, siehe Kapitel 4.6.

In Paper [9] werden folgende individuelle Eigenschaften erwähnt, die auch in TrafLib Verwendung finden:

- Fahrzeug-Länge – in TrafLib ist diese fix für alle Fahrzeuge
- Höchstgeschwindigkeit
- Maximale Beschleunigung
- Maximale Bremsverzögerung
- Following time – in TrafLib ist die Following Time eine Eigenschaft des Fahrers und wird in Kapitel 4.6 näher erläutert.

Nachdem die Anzahl der Fahrzeuge, die in Echtzeit simuliert werden können begrenzt ist, muss man vor allem bei großen Straßennetzwerken Strategien entwickeln, um ausreichend viele Fahrzeuge in den Sichtbereich der Kamera zu bekommen. Ein Ansatz wird in [17] vorgestellt, wo während der gesamten Simulation an bestimmten Stellen (source points) neue Fahrzeuge erzeugt und an anderen Stellen (drain points) wieder entfernt werden. Diese Generierungs- und Entfernungspunkte können sich mit der Position der Kamera bzw. des Fahrers dynamisch verändern.

Weitere denkbare Strategien:

- Fahrzeuge, die nicht im Sichtbereich der Kamera sind, so dirigieren dass sie wieder in den Sichtbereich kommen.
- Fahrzeuge, die sich weit außerhalb des Sichtbereiches befinden, entfernen und wieder in die Nähe des Sichtbereiches platzieren.
- Fahrzeugdichte im Sichtbereich der Kamera erhöhen.

In TrafLib wird folgende Lösung verwendet:

Die Visualisierungssoftware muss eine Position und einen Kreisdurchmesser angeben. TrafLib plant für alle Fahrzeuge, die sich außerhalb dieses Kreises befinden, eine Route in diesen zurück.

4.5.1 Fahrzeuge hinzufügen (AddVehicle)

Beim Hinzufügen von Fahrzeugen sind verschiedene Parameter möglich:

Parameter	**Auswirkungen in TrafLib auf**	
	Platzierung	**Route**

keine	Zufällig	zufällig
Startposition	Fix	zufällig
Start- und Zielposition	Fix	fix
Startregion	zufällig innerhalb einer Region	zufällig
Start- und Zielregion	zufällig innerhalb einer Region	fix

Tabelle 2: Fahrzeug hinzufügen - Parameter

Des weiteren kann man die Höchstgeschwindigkeit, maximale Bremsverzögerung und die maximale Beschleunigung per Parameter angeben. Dies ermöglicht der Visualisierungssoftware die spezifische Definition von Fahrzeugeigenschaften für ihre Fahrzeugmodelle.

4.5.2 Fahrzeuge entfernen (RemoveVehicle)

Die Visualisierungssoftware kann natürlich über das TrafLib-Interface Fahrzeuge entfernen. Eine automatische Entfernung erfolgt in TrafLib im Falle eines Deadlocks:

Befindet sich ein Fahrzeug im Kreuzungsbereich und bewegt sich länger als zwei Minuten nicht, so wird von einer Deadlock-Situation ausgegangen und dieses Fahrzeug entfernt.

Eine andere Lösung wird bei [8] erwähnt, wo im Falle eines traffic jams (traffic deadlock) ein central controller eingeschaltet wird um den Konflikt zu lösen - ähnlich der Regelung durch einen Polizisten.

4.5.3 Fahrzeugbewegung (MoveVehicle)

Im Kapitel 4.6 wird erläutert, wie aufgrund der Verkehrssituation eine günstige Beschleunigung bzw. Bremsverzögerung für alle Fahrzeuge berechnet wird.

Danach wird

- die Geschwindigkeit aller Fahrzeuge um die Beschleunigung erhöht, bzw. um die Bremsverzögerung verringert und
- der zurückgelegte Weg aller Fahrzeuge berechnet.

Dieser Vorgang muss von der Visualisierungssoftware für jedes gezeichnete Bild initiiert werden, bei Echtzeitanwendungen also 60 mal pro Sekunde. Wird die Funktion „MoveVehicle“ seltener aufgerufen als Bilder gezeichnet werden, so wird ein Ruckeln in der Bewegung der Fahrzeuge sichtbar.

4.5.4 Routenplanung (PlanRoute)

Das makroskopische Ziel eines Fahrzeuges ist die Route zu seinem Zielort [8]. Das mikroskopische Ziel ist die Entscheidungsfindung, die in TrafLib jede ½ Sekunde stattfindet, um das makroskopische Ziel zu erreichen.

Routenfindung wird in der Netzwerk Theorie (NT) als das „Shortest Path Problem“ bezeichnet [7]. Die Route, die durch einen solchen Algorithmus gefunden wird, ist die kürzeste, aber nicht notwendigerweise die beste. Bei [7] wird unter anderem der wissensbasierte Ansatz erklärt, um für einen menschlichen Fahrer „sinnvolle“ Straßen (z.B.

nicht zu viele kleine Gassen) für eine Route auszuwählen. In TrafLib ist über die Straßentypen usw. nichts bekannt. Was aber als zusätzliche Information verwendet werden kann, ist die durchschnittliche Geschwindigkeit auf den Fahrbahnen. Dieses Wissen wird benutzt, um die schnellste Route, welche im Falle von TrafLib die beste ist, zu berechnen.

Zur Berechnung von Routen muss das Straßennetz in einen Graphen abgebildet werden. Dabei entspricht jede Kreuzung einem Knoten und jede Straße einer Kante. Die Gewichtungen an den Kanten werden für verschiedene Algorithmen unterschiedlich berechnet. Für eine verkehrssimulationsbezogene Einführung in die Graphentheorie siehe [23].

Das Straßennetzwerk in TrafLib kann als endlicher, gerichteter, nicht-planarer, schwach-zusammenhängender Graph gesehen werden [24]:

- endlich: Alle Straßen enden in dem Straßennetzwerk.
- gerichtet: Die Fahrbahnen dürfen von den Fahrzeugen nur in eine Richtung benutzt werden.
- planar: Bedeutet, dass sich der Graph aufzeichnen lässt, ohne dass sich Kanten überschneiden. Da es im Straßennetzwerk Über- und Unterführungen geben darf, muss der Graph nicht planar sein.
- schwach zusammenhängend: Zwischen je zwei nicht geordneten Paaren von Knoten muss eine Bahn existieren. (Stark zusammenhängend heißt, dass zwischen jedem geordneten Paar von Knoten eine Bahn existieren muss. Das ist aber im Straßengraph zum Beispiel dann nicht der Fall, wenn man eine Einbahn zu einem Endknoten hat.)

In [29] sind mehrere Algorithmen zur Routenplanung beschrieben. TrafLib beherrscht die folgenden 3 Methoden.

4.5.4.1 zufällige Route

Diese Methode kommt immer dann zum Einsatz, wenn keine bestimmte Route gewählt werden muss. Sie hat gegenüber den anderen den Vorteil, dass sie fast keine Rechenzeit benötigt.

Dabei werden für ein Fahrzeug die nächsten 10 Rails und Korridore zufällig ausgewählt. Hat das Fahrzeug einen Teil seiner Strecke zurückgelegt, so wird die Route wieder zufällig verlängert.

4.5.4.2 A* ohne Verkehrsauslastung

Der A* Algorithmus ist die grundlegende Methode, um in einem Graphen einen Weg von A nach B mit minimaler Summe der Gewichtungen zu ermitteln.

Bei der Verwendung des A* Algorithmus ohne Wissen über die Verkehrsauslastung wird als Kanten-Gewichtung die Länge der Richtungsfahrbahnen verwendet. Dadurch wird die streckenmäßig kürzeste Route gefunden. Mehrere Fahrstreifen einer Richtungsfahrbahn werden nur als eine Kante zwischen zwei Knoten abgebildet.

Dieser Algorithmus entspricht in etwa der Routenplanung eines herkömmlichen Navigationssystems, das über die Verkehrsauslastung keine Information hat.

TrafLib verwendet eine Verbesserung des A* Algorithmus in Anlehnung an Dijkstra [29]: Kanten mit geringen Gewichtungen werden als erstes verfolgt.

4.5.4.3 A* mit Verkehrsauslastung

Dieses Kapitel behandelt die Routenplanung unter Berücksichtigung der Verkehrsauslastung. In [29] wird ein „Trafficmaster“ vorgestellt, welcher Informationen über die Geschwindigkeit von Fahrzeugen auf Straßen sammelt. Es gibt dafür 2 Methoden:

- Auf Hauptstraßen mit mehreren Fahrspuren werden Infrarot Sensoren auf Brücken montiert, welche die Geschwindigkeit der einzelnen Fahrzeuge messen.
- Auf Nebenstraßen werden Kameras an den Straßenrändern positioniert, welche die mittleren 4 Stellen des Nummernschildes erkennen. Werden die Nummerntafeln noch einmal erkannt, lässt sich die Geschwindigkeit des Fahrzeuges ermitteln.

Diese gesammelten Informationen können von herkömmlichen Routenplanern verwendet werden, um im Gegensatz zur entfernungsmäßig kürzesten Strecke die zeitlich kürzeste zu finden. Dies erfolgt über die Durchschnittsgeschwindigkeit der Fahrzeuge. Es lassen sich dadurch realistischere Fahrzeiten ermitteln und Staus umgehen.

Ähnlich wird das Problem in TrafLib angegangen. Es werden statistische Informationen wie zum Beispiel die durchschnittliche Geschwindigkeit der Fahrzeuge auf den Straßen ermittelt. Als Gewichtung im Straßengraphen wird der Quotient aus Fahrstreifenlänge und durchschnittlicher Geschwindigkeit verwendet. Befindet sich kein Fahrzeuge für statistische Berechnungen auf einer Straße, so wird die erlaubte Geschwindigkeit für diese Straße verwendet.

Die durchschnittliche Geschwindigkeit muss über einen längeren Zeitraum beobachtet werden, um den Einfluss der verschiedenen Ampelphasen einzubeziehen.

Im Gegensatz zum A* Algorithmus, der die Verkehrsauslastung nicht berücksichtigt, muss hier jeder Fahrstreifen durch eine Kante dargestellt werden, da jeder Fahrstreifen unterschiedlich frequentiert sein kann. Daraus folgt, dass dieser Algorithmus mehr Rechenzeit benötigt als der vorige.

4.6 Verkehrsberechnung (cTrafficCalculation)

Vekehr ist die Summe der Bewegung aller Fahrzeuge.

Die Fahrzeuge sind Teil der Umwelt [8] und können diese wahrnehmen. Aufgrund ihres Zieles und dem wahrgenommenen Ausschnitt der Umwelt bewegen sie sich entsprechend den Verkehrsregeln auf ihren Zielpunkt zu. Somit sind die Fahrzeuge agents und können individuelle Verhaltensweisen („free will speed“, „free will breaking power“, „free will acceleration“ [8]) besitzen. Bei einer strikt agent-basierten Simulation hat jeder agent volle Kontrolle über sich selbst.

Dies trifft auch auf TrafLib zu. Zwar wird die Berechnung zur Unfallvermeidung in die Konfliktpunkte verlagert, aber dies ist nur programmiertechnisch bedingt und dient der Verkürzung der Berechnungszeit. Die Fahrzeuge verhalten sich trotzdem selbstständig, die Definition von agents wird also nicht verletzt.

In der Realität sieht ein Fahrer ein anderes Fahrzeug, sofern Sichtkontakt möglich ist, schätzt dessen Geschwindigkeit und trifft eine Entscheidung [8]. Dabei werden auch Fahrzeuge wahrgenommen, die die eigene Fahrt nicht beeinflussen. In TrafLib „sieht“ ein Fahrzeug nur jene anderen Fahrzeuge, die sich auf Kollisionskurs befinden, d.h. den gleichen Konfliktpunkt im Einflussbereich haben, auch wenn kein Sichtkontakt besteht. Die Geschwindigkeit der anderen Fahrzeuge wird in der Simulation exakt übernommen.

Die Steuerung der Fahrzeuge basiert auf dem Straßennetz-Layout (vehicle-to-network) und dem Zustand (Geschwindigkeit + relative Position) der anderen Fahrzeuge (vehicle-to-vehicle) [9].

Viele Verkehrssimulations-Softwarepakete basieren nur auf einem vehicle-to-network system, wobei Fahrzeuge fixe Punkte für sich reservieren bzw. wieder freigeben müssen. Fahrzeuge werden nicht individuell gesteuert, können also weder beschleunigen, bremsen, noch frei fahren. Des weiteren ist es sehr schwierig Fahrzeuge unterschiedlicher Länge zu modellieren [9].

TrafLib verwendet sowohl vehicle-to-network als auch vehicle-to-vehicle Steuerung. Dadurch ist die Simulation flexibler und das Verhalten der Fahrzeuge wirkt realistischer. Die „vehicle-to-network“ Steuerung entspricht in TrafLib der Berechnung des Fahrverhaltens aufgrund der verkehrsregelnden Konfliktpunkte. Die „vehicle-to-vehicle“ Steuerung hingegen entspricht den unfallvermeidenden Konfliktpunkten und dem Hintereinanderfahren.

In TrafLib läuft die Verkehrsberechnung folgendermaßen ab:

1. Berechnung des Einflussbereiches aller Fahrzeuge.
2. Ermittlung der Konfliktpunkte im Einflussbereich.
3. Ermittlung des Vorderfahrzeuges.
4. Berechnung der Beschleunigungen aufgrund des Vorderfahrzeuges und aller Konfliktpunkte (Die Anzahl der beeinflussenden Konfliktpunkte lässt sich nachträglich reduzieren, siehe Abb. 13).
5. Die maßgebende Beschleunigung ist die geringste der im Punkt 4. berechneten. Aufgrund dieser Beschleunigung wird das Fahrzeug bis zur nächsten Verkehrsberechnung bewegt.

4.6.1 Micro goal ⇔ macro goal

Der Unterschied zwischen dem mikroskopischen und dem makroskopischen Ziel wurde in 4.5.4 bereits erwähnt.

Das beste micro goal muss nicht die beste Lösung sein um ein macro goal zu erreichen [8]: Wenn zum Beispiel 2 Fahrzeuge auf die selbe Kreuzung gleichzeitig zu fahren, so hat der benachrangte Fahrer 2 Möglichkeiten:

- Er kann seine Geschwindigkeit beibehalten und erst kurz vor der Kreuzung abbremsen, möglicherweise auf eine Geschwindigkeit von 0. Wenn der Bevorrangte die Kreuzung verlassen hat, kann er wieder losfahren bzw. beschleunigen.
- Er kann seine Geschwindigkeit frühzeitig drosseln, sodass er die Kreuzung nach dem Bevorrangten mit einer moderaten Geschwindigkeit passiert, ohne dass er stark abbremsen oder stehen bleiben muss.

Der 1. Fall entspricht eher einem aggressiven Fahrstil, wo die Geschwindigkeit möglichst lange beibehalten und erst recht spät stark gebremst wird.

Der 2. Fall entspricht rücksichtsvollem und vorausschauendem Fahrverhalten.

TrafLib simuliert drei verschiedene Fahrstile, nämlich einen aggressiven, einen normalen und einen vorsichtigen. Der aggressive Fahrstil ist mit Fall 1 zu vergleichen, der vorsichtige kommt in die Nähe von Fall 2 verhält sich aber nicht ganz so „vorausschauend“.

4.6.2 Individuelles Fahrverhalten

Es gibt verschiedene Modelle um das Fahrverhalten von Menschen zu modellieren [37]. Vor 1960 herrschte zur Modellierung des Fahrverhaltens das „skill-based driving model“ vor, danach das „motivational driving model“. Von Michon [19] wurde die „hierarchical control structure“ eingeführt, welche in TrafLib verwendet wird. Dort werden 3 Ebenen der Fahrzeug-Steuerung verwendet:

- strategische Ebene (strategic level): Diese Ebene betrifft i. a. die Routenplanung.
- Manövrierungsebene (maneuvering level): Hier erfolgt die Bestimmung des Fahrverhaltens aufgrund von Verkehrsregeln, Hintereinanderfahren und individuellen Eigenschaften des Fahrers. In TrafLib entspricht dies der Verkehrsberechnung, welche jede ½ Sekunde ausgeführt wird.
- Ausführungsebene (operational level): Diese wird auch als „low-level control“ bezeichnet und behandelt die Anpassung der Geschwindigkeit aufgrund der Beschleunigung und des zurückgelegtes Weg. Dies entspricht in TrafLib der Methode „MoveVehicles“ welche für Echtzeitvisualisierung 60 mal pro Sekunde aufgerufen werden muss.

Auf Ebene der Manövrierung werden üblicherweise ebenfalls 3 Methoden unterschieden [20]:

- rule-based models: Diese basieren auf Wissensdatenbanken, zusammengesetzt aus Regeln, um Entscheidungen fürs Fahren zu treffen. Jede Regel bzw. ein Bündel von Regeln indiziert eine bestimmte Handlung unter bestimmten Voraussetzungen.
- state machine models: State machines repräsentieren eine Anzahl von kleinen Unteraufgaben. In der höheren Hierarchie werden für eine bestimmte Aufgabe, zum Beispiel Links-Abbiegen, mehrere Unteraufgaben in einer bestimmten Reihenfolge ausgeführt.
- probabilistic models: Probabilistische Modelle legen dem Fahrverhalten empirisch ermittelte Werte von echtem Fahrverhalten zugrunde.

Die Verkehrsberechnung von TrafLib entspricht einem rule-based Modell.

In [8] werden Fahrer in vorsichtige, normale und aggressive Fahrer unterteilt. Das wird durch Beschleunigungs- und Bremsstärke, Abstand zum Vordermann und Geschwindigkeit realisiert. Durch die Regelung des Verkehrs mittels Ampeln und Verkehrstafeln fallen die individuellen Eigenschaften der Fahrer weniger ins Gewicht als bei unorganisiertem Verkehr, wie zum Beispiel in Indien [8]. Dort werden die Geschwindigkeit, Beschleunigung und Bremsverzögerung parametriert, um verschiedene Fahrertypen zu modellieren:

type	trait		
	aggressive	normal	cautious
Free will speed (kmph)	≥ 75	≥ 60 & ≤ 75	≤ 60
Free will breaking power (m/s^2)	≥ 8	≥ 6 & ≤ 8	≤ 6
Free will accelaration (m/s^2)	≥ 1.5	≥ 1 & ≤ 1.5	≤ 1

Tabelle 3: Simple classification of psychological traits [8]

TrafLib lehnt sich an dieses Modell an und macht zusätzlich den Abstand beim Hintereinanderfahren vom psychologischen Profil abhängig:

Beispiel		individuell gewählte Werte bei verschiedenen Fahrstilen		
		aggressiv	normal	vorsichtig
erlaubte Geschwindigkeit	50 km/h	+ 10% = 55 km/h	± 0% = 50 km/h	- 10% = 45 km/h
max. Bremsstärke eines Fahrzeuges	9 m/s^2	80% = 7,2 m/s^2	65% = 5,9 m/s^2	50% = 4,5 m/s^2
max. Beschl. eines Fahrzeuges	2 m/s^2	80% = 1,6 m/s^2	65% = 1,3 m/s^2	50% = 1,0 m/s^2
Sicherheitsabstand zum Vordermann	1,5 s	- 33% = 1 s	± 0% = 1,5 s	+ 33% = 2,0 s

Tabelle 4: Parameter für individuelles Fahrverhalten

4.6.3 Einflussbereich berechnen (CreateRangeOfInfluence)

Mit diesem Kapitel beginnt die Beschreibung des Programmablaufs für die Verkehrsberechnung. Es muss zuerst der Einflussbereich für alle Fahrzeuge berechnet werden. Die Formel dazu ist in Kapitel 4.2.2 erklärt.

4.6.4 Konfliktpunkte im Einflussbereich (CreateListOfConflicts)

Bevor im nächsten Kapitel die „Speed Limits“ berechnet werden können, muss für jedes Fahrzeug eine Liste jener Konfliktpunkte, die im Einflussbereich der Fahrzeuge liegen, erstellt werden. Wie die Ermittlung erfolgt, wurde in Kapitel 4.2.2 beschrieben.

4.6.5 Speed Limits (CalculateSpeedLimits)

Jedes Fahrzeug versucht immer auf unendliche Geschwindigkeit zu beschleunigen. Durch verschiedene Speed Limits wird diese aber begrenzt:

- Bauartgeschwindigkeit: Ein Fahrzeug kann natürlich nicht schneller fahren als es physikalisch für dieses Fahrzeugmodell möglich ist.
- zulässige Höchstgeschwindigkeit eines Straßenabschnittes
- zulässige Höchstgeschwindigkeit einer Kurve: In Abhängigkeit des Kurvenradius wird eine Geschwindigkeit berechnet die Fahrzeuge in TrafLib einhalten.
- Vorderfahrzeug
- alle Konfliktpunkte

In TrafLib erhält jedes Fahrzeug eine Liste von Speed-Limits. Diese enthält die jeweiligen Geschwindigkeitsbegrenzungen und die Entfernungen in der diese zu erreichen sind. Daraus wird jeweils eine Beschleunigung berechnet. Die geringste aller Beschleunigungen ist für das

Fahrzeug maßgebend. Liegt die Beschleunigung außerhalb der für ein Fahrzeug zulässigen Grenzen, wird die maximale Bremsverzögerung oder die maximale Beschleunigung verwendet.

4.6.5.1 Bauartgeschwindigkeit

Die Bauartgeschwindigkeit ist die Höchstgeschwindigkeit eines Fahrzeugtyps. Der Standardwert in TrafLib entspricht einem Mittelklasseauto. Die Bauartgeschwindigkeit ist aber letztlich nicht von Belang, da sie viel höher ist , als im Stadtgebiet gefahren werden darf. Dem Benutzer von TrafLib steht aber frei, ein Fahrzeug mit niedrigerer Höchstgeschwindigkeit hinzuzufügen.

4.6.5.2 Geschwindigkeitsbegrenzung

TrafLib ist auf die Simulation von städtischem Verkehr ausgelegt. Daher wurde eine generelle Höchstgeschwindigkeit von 50 km/h (~14 m/s) festgelegt, welche allerdings durch Geschwindigkeitsbegrenzungstafeln geändert werden kann.

4.6.5.3 Geschwindigkeitsbegrenzungen in Kurven

Diese werden ebenfalls durch Konfliktpunkte realisiert. Bei der Initialisierung von TrafLib werden alle Kurvenradien berechnet und ein Geschwindigkeitslimit daraus ermittelt. Falls dieses Limit niedriger als 72 km/h ist, wird ein entsprechender Konfliktpunkt an den Beginn der Kurve gesetzt.

4.6.5.4 Hintereinanderfahren (CalculateFollowingSpeed)

Betrachtet man Stadtverkehr so stellt man fest, dass das Fahrverhalten eines Fahrzeuges viel öfter durch ein Vorderfahrzeug beeinflusst wird als durch Verkehrszeichen. Daher ist das Hintereinanderfahren ein zentraler Punkt in einer Verkehrssimulation und es wurden bereits viele verschiedene Formeln dafür entwickelt [12, 33, 34].

Die Reaktion eines Fahrers beim Hintereinanderfahren drückt sich durch eine Änderung der Beschleunigung aus [5] und ist abhängig von

- dem Abstand zum Vordermann
- dem Geschwindigkeitsunterschied
- den Geschwindigkeiten
- den Fahrzeugeigenschaften
- den Fahrereigenschaften

Für das Hintereinanderfahren von Fahrzeugen wurde für TrafLib folgende Formel entwickelt:

$$v = v_{VF} + F \cdot (d - s_R)$$

$$s_R = \max(0, t_g \cdot v + \frac{t_g^2 \cdot a}{2})$$

v ... angestrebte Geschwindigkeit eines nachfolgenden Fahrzeuges

v_{VF} ... Geschwindigkeit des Vorderfahrzeuges

s_R ... zurückgelegter Weg nach t_g Sekunden

d ... Distanz zwischen zwei Fahrzeugen

t_g ... „time gap“, vom Fahrer gewünschter zeitl. Abstand zum Vorderfzg.

F ... Faktor mit dem sich die „Stärke“ der Geschwindigkeitsanpassung einstellen lässt. In TrafLib gilt $F = 1$.

Die berechnete Geschwindigkeit v geht von der Geschwindigkeit des Vorderfahrzeuges aus v_{VF}. Dazu kommt ein positiver oder negativer Wert, der vom Abstand der beiden Fahrzeuge zueinander abhängt. Dieser Wert setzt sich aus dem Faktor F und dem Ausdruck $d - s_R$ zusammen. Dabei ergibt sich letzterer Ausdruck aus der Differenz des Fahrzeugabstandes und dem Weg, den das Fahrzeug in seinem gewünschten zeitlichen Abstand zurücklegt.

4.6.5.5 Verkehrsregelnde Konfliktpunkte

Wie bereits erwähnt umfassen die verkehrsregelnden Konfliktpunkte im Kreuzungsbereich Ampeln, Stopp-Tafeln und Vorrang-Geben-Tafeln, und außerhalb des Kreuzungsbereiches Geschwindigkeitsbegrenzungs-Tafeln.

Liegt zum Beispiel der Konfliktpunkt einer Ampel im Einflussbereich eines Fahrzeuges, dann wird die momentane Ampelphase bestimmt und falls diese rot ist, wird zum Speed Limit des Fahrzeuges die Geschwindigkeit 0 und die Entfernung zur Ampel hinzugefügt. Stopp-Tafeln und Vorrang-Geben-Tafeln werden analog dazu behandelt.

Bei einer Geschwindigkeitsbegrenzungs-Tafel werden Geschwindigkeitsbegrenzung und Entfernung dieser Tafel zur Speed Limit Liste hinzugefügt.

4.6.5.6 Unfallvermeidende Konfliktpunkte

Zu diesen Konfliktpunkten zählen die Fahrspurvereinigung, -aufteilung und der Fahrspurschnitt.

Eine Fahrspuraufteilung wird so behandelt, dass alle Fahrzeuge, welche sich im Schnittbereich (siehe Abb. 7) befinden, auf einer virtuellen Rail (siehe Kapitel 4.4.3.2) hintereinander fahren müssen.

Bei Fahrspurvereinigungen und Fahrspurschnitten wird implizit die Rechtsregel angewandt. Dies bedeutet, dass ein benachrangtes Fahrzeug vor einem Hilfskonfliktpunkt stoppen muss. Dadurch wird das bevorrangte Fahrzeug nicht behindert. Sobald der Hilfskonfliktpunkt überschritten ist, gilt wieder das Hintereinanderfahren auf einer virtuellen Rail. Dieses Hintereinanderfahren wirkt sich bei einer Spurvereinigung wie das „Reißverschlusssystem“ in der Realität aus. Beim Hintereinanderfahren aufgrund eines Fahrspurschnittes wird der Abstand zwischen den Fahrzeugen erhöht.

In der folgenden Abbildung sind zwei Beispiele zur Unfallvermeidung mittels Konfliktpunkten anhand einer ungeregelten Kreuzung dargestellt.

Das erste Beispiel betrifft die Fahrzeuge A und B, sowie den Spurschnitt S. B hat in diesem Fall Vorrang, weshalb A vor dem Hilfskonfliktpunkt von S anhalten muss, um B nicht zu behindern.

Beispiel zwei betrifft die Fahrzeuge C und D, sowie die Spurvereinigung V. D ist zwar im Vorrang, aber C hat bereits den Hilfskonfliktpunkt überschritten. Deshalb werden beide Fahrzeuge auf die virtuelle Straße zum Konfliktpunkt V gesetzt und die Formeln für das Hintereinanderfahren angewandt.

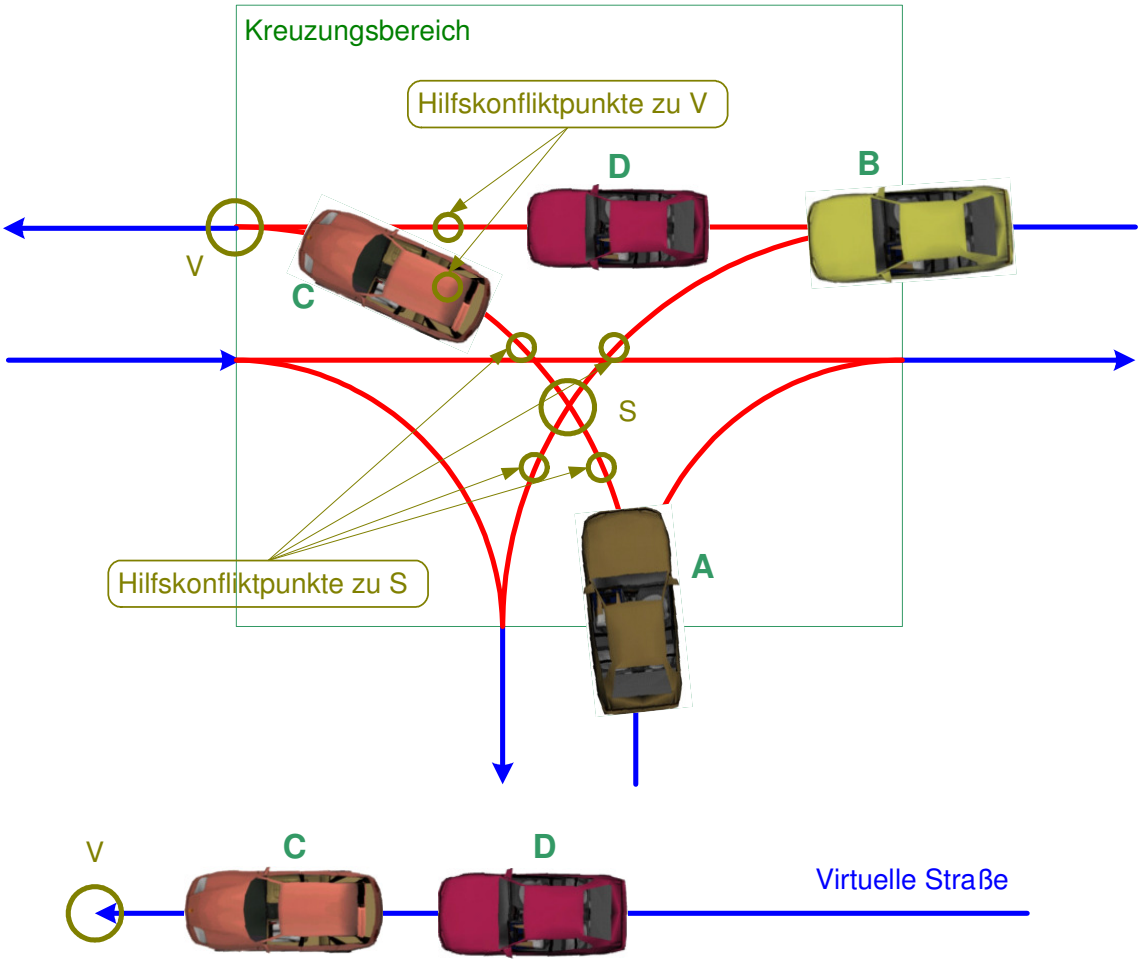


Abb. 21: Einfluss von Konfliktpunkten – Beispiel

4.6.6 Berechnung der Beschleunigungen (CalculateAccelerations)

Im vorigen Kapitel wurde für jedes Fahrzeug eine Liste der Speed Limits erstellt. Nun werden aufgrund der Geschwindigkeiten und Entfernungen, die zuvor berechnet wurden, Beschleunigungen berechnet. Die kleinste der Beschleunigungen ist jeweils für ein Fahrzeug ausschlaggebend.

4.6.7 Spurwechsel (CalculateLaneChanging)

Spurwechsel ist ein sehr häufiger Vorgang im städtischen Verkehr. In TrafLib wurde der Spurwechsel auch aufgrund der Routenplanung implementiert, denn wenn zum Beispiel ein Fahrzeug links abbiegen will, muss es auf eine eventuell vorhandene Linksabbiege-Spur wechseln können.

Die Implementierung von „intelligentem Vorausschauen“ ist sehr schwierig, deshalb wurde in TrafLib ein „erzwingender Spurwechsel“ programmiert, was bedeutet, dass einem Fahrzeug, das die Spur wechseln möchte, Platz gemacht wird.

Folgender Ablauf findet statt:

1. Fahrzeug A möchte die Fahrspur wechseln

2. Das Fahrzeug A wird virtuell auf die zu wechselnde Fahrspur gesetzt.
    a. Bei der Behandlung des Hintereinanderfahrens muss auch das virtuelle Fahrzeug beachtet werden (Fahrzeug C in Abb. 22). Dadurch passt Fahrzeug A seine Geschwindigkeit an seine Vorderfahrzeuge (B und D) auf beiden Fahrspuren an. Weiters schafft jenes Fahrzeug (C), das sich auf der Ziel-Fahrspur hinter dem virtuellen Fahrzeug A befindet, die nötige Lücke für A.
    b. Fahrzeug A muss die Konfliktpunkte auf beiden Fahrspuren beachten.
3. Fahrzeug A kann nun mit angepasster Geschwindigkeit die Spur wechseln.
4. Ist der Spurwechsel abgeschlossen, kann Fahrzeug A von der ursprünglichen Fahrspur entfernt und auf die neue platziert werden.

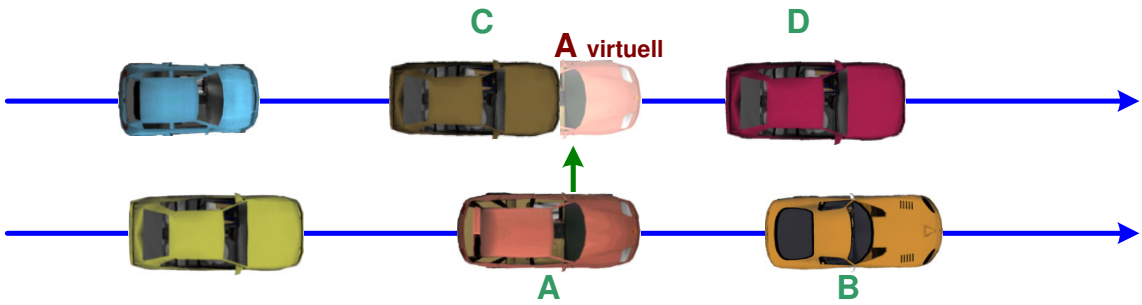


Abb. 22: Spurwechsel

4.6.8 Überholen

Da der Überholvorgang relativ komplex ist und im städtischen Gebiet aufgrund der niedrigen erlaubten Geschwindigkeiten sehr selten vorkommt, wurde in TrafLib auf die Implementation verzichtet. Überholmanöver in der Realität finden zwar statt, aber meist unter Verletzung der Geschwindigkeitsbegrenzungen. Die Durchführung von Überholmanövern ist u.a. in [8] behandelt.

4.7 Statistik

Es bietet sich an, für eine mikroskopische Verkehrssimulation statistische Werte zu berechnen. Diese können mit makroskopischen Parametern verglichen werden. Für Verkehrssimulatoren zur Verkehrsplanung werden diese makroskopischen Parameter [12] entweder in der Realität beobachtetet, oder von geeigneten mikroskopischen Verkehrssimulatoren übernommen.

Zum Beispiel werden in CTSP (City Traffic Simulation Package) [13], folgende Werte als Output berechnet:

- Average waiting times of vehicles
- Average Trip Times
- Average Number of Stoppings
- Average Speed
- Maximum and Average Queue Lengths

- Number of Vehicles entering and leaving each roads

Für TrafLib ist praktisch nur die durchschnittliche Geschwindigkeit der Fahrzeuge auf den Straße interessant, da sie zur Routenberechnung verwendet wird.

Falls erwünscht, kann ein Visualisierungsprogramm folgende zusätzlich von TrafLib berechneten Werte abfragen und ausgeben:

- Anzahl der Autos pro Richtungsfahrbahn
- Anzahl der Autos pro Fahrstreifen
- Durchschnittsgeschwindigkeit aller Fahrzeuge einer Richtungsfahrbahn
- Durchschnittsgeschwindigkeit aller Fahrzeuge eines Fahrstreifens
- Verkehrsdichte (= Anzahl der Fahrzeuge pro Straße bzw. Fahrstreifen dividiert durch die Länge der Straße bzw. des Fahrstreifens)
- Verkehrsfluss (= Verkehrsdichte * Durchschnittsgeschwindigkeit, oder Summe aller Geschwindigkeiten dividiert durch die Streckenlänge)

Der Zusammenhang zwischen Verkehrsdichte und –fluss wurde aus [21] übernommen:

Verkehrsdichte $\rho = \frac{Anz_{Fahrzeuge}}{Streckenlänge}, [1/m]$

Verkehrsfluss $q = \frac{\sum v_{Fahrzeuge}}{Streckenlänge}, [1/s]$

Durschnittsgeschw. $u = \frac{q}{\rho}, [m/s]$

Für die Gewichtungen in der Routenplanung wird jeweils der Quotient aus *Streckenlänge* eines Fahrstreifens und Durchschnittsgeschwindigkeit u seiner Fahrzeuge errechnet. Dieser Quotient gibt an, wie viele Sekunden ein Fahrzeug durchschnittlich benötigt, um eine Fahrstreifenlänge zurückzulegen.

5 Implementierung von TrafLib

In den folgenden Unterkapiteln werden die Entwicklung, Benutzung und die Details der Implementierung beschrieben.

5.1 Entwicklungsplattform und -werkzeuge

Die Bibliothek TrafLib (Kapitel 4) wurde auf Standardhardware unter Verwendung von Microsoft Windows XP und Microsoft Visual Studio 2005 in der Programmiersprache c++ entwickelt. Es wurden keine Microsoft-spezifischen Funktionen verwendet, sodass TrafLib mittels nur geringem Arbeitsaufwand für andere c-Compiler adaptiert werden kann.

Für den Fahrzeugmodell-Konverter (Kapitel 6) wurde zusätzlich Open Inventor 2.6 und Ogre 1.2.0 verwendet, um Fahrzeugmodelle im .iv und .mesh Format umzuwandeln.

Das Demonstrationsprogramm (Kapitel 7) wurde ebenfalls mit Hilfe von Ogre 1.2.0 entwickelt.

5.2 Verwendung von TrafLib

TrafLib ist eine Bibliothek, und kann daher von jeder Visualisierungssoftware über den Source code, oder mittels Header files und vorkompilierter Bibliotheks-Datei (.lib) eingebunden werden.

Erforderlich für die Verkehrssimulation ist ein Straßennetzwerk. TrafLib liest dieses über eine Datei ein, wobei zwei verschiedene Formate unterstützt werden:

- TrafLib-eigenes Format, welches in Kapitel 4.4.2 beschrieben ist
- Outputformat von VRMG [1], welches in Kapitel 4.4.1 beschrieben wurde und sich sehr gut für die Erstellung großer Straßennetzwerke eignet.

Danach müssen über die Schnittstelle von TrafLib Fahrzeuge platziert und der Simulationsablauf über die Funktionen „MoveVehicles“ und „CalculateAccelerations“ gesteuert werden.

TrafLib berechnet verschiedene statistische Werte, welche von der Visualisierungssoftware ausgegeben werden können.

Eine gute Illustration zur Verwendung von TrafLib bietet das ebenfalls im Rahmen dieser Diplomarbeit entwickelte Demonstrationsprogramm (siehe Kapitel 7.

5.3 Klassenbeschreibung und Datenstrukturen

Die für TrafLib erstellten Klassen wurden bereits kurz im Kapitel über die Architektur (4.3) beschrieben und ihr Zusammenhang in Abb. 14 dargestellt.

Die folgenden vier Diagramme stellen die in TrafLib verwendeten Daten-Objekte dar, auf die die Objekte verweisen.

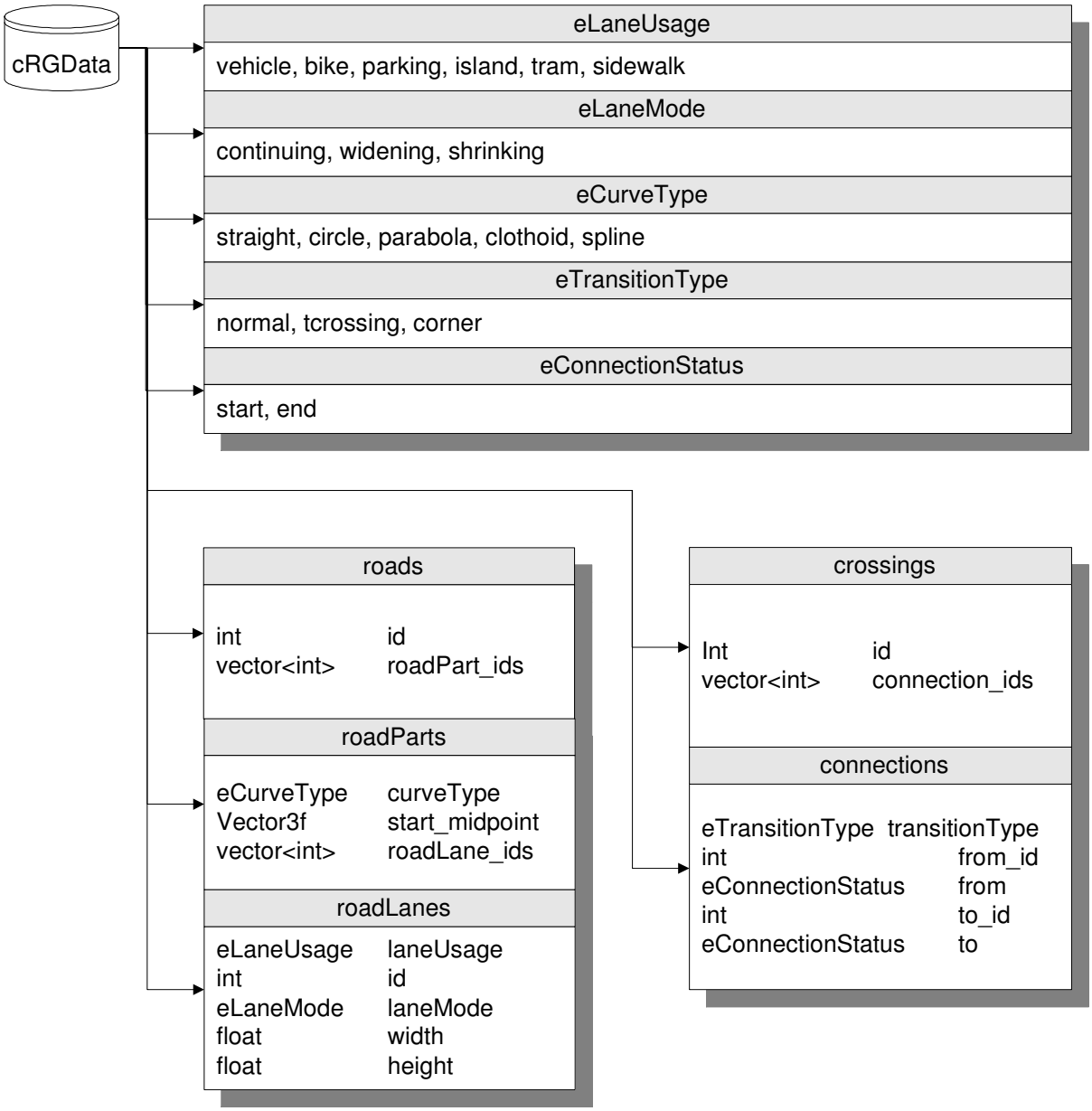


Abb. 23: Daten-Objekt RoadGeometryData

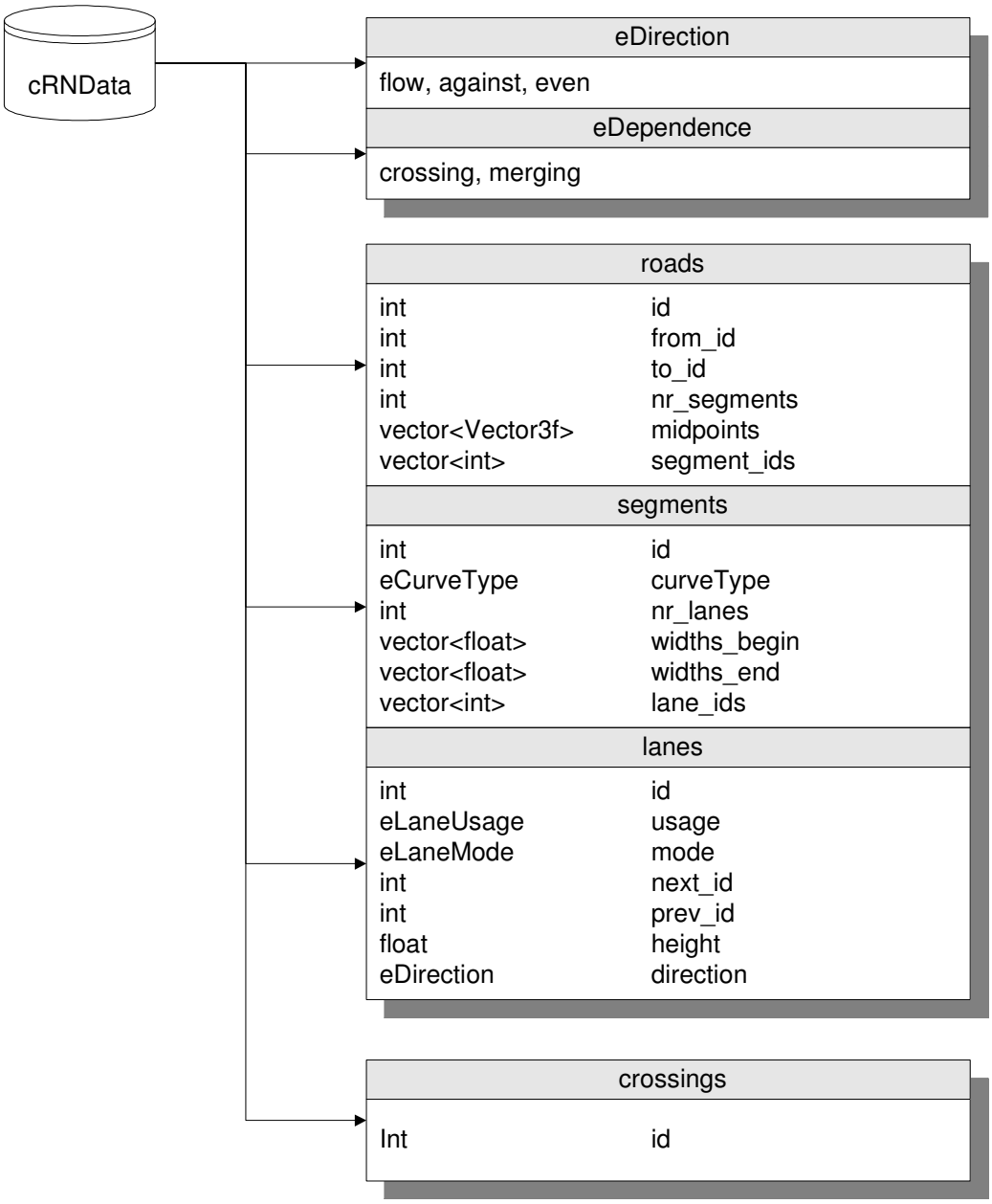


Abb. 24: Daten-Objekt RoadNetworkData

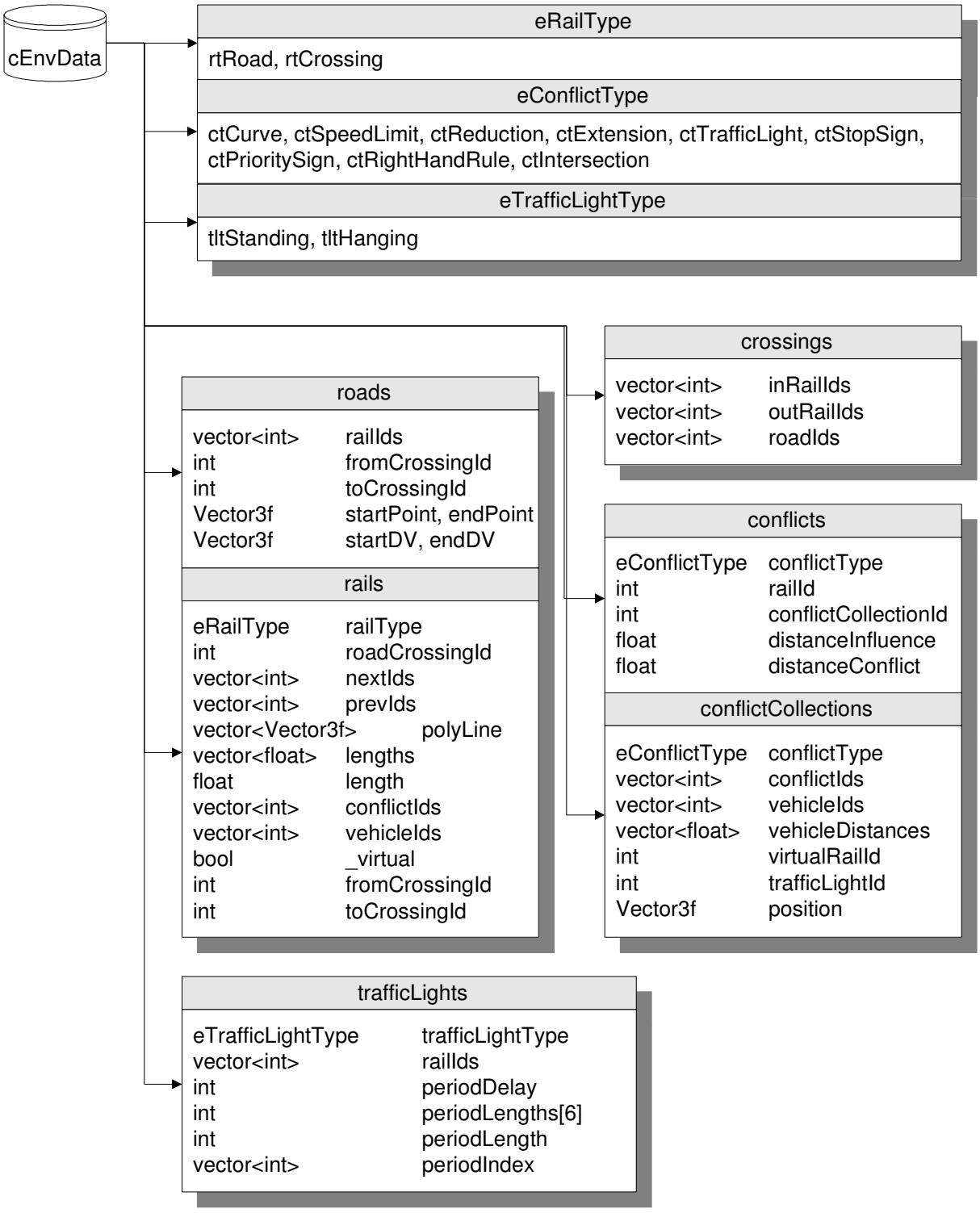


Abb. 25: Daten-Objekt EnvironmentData

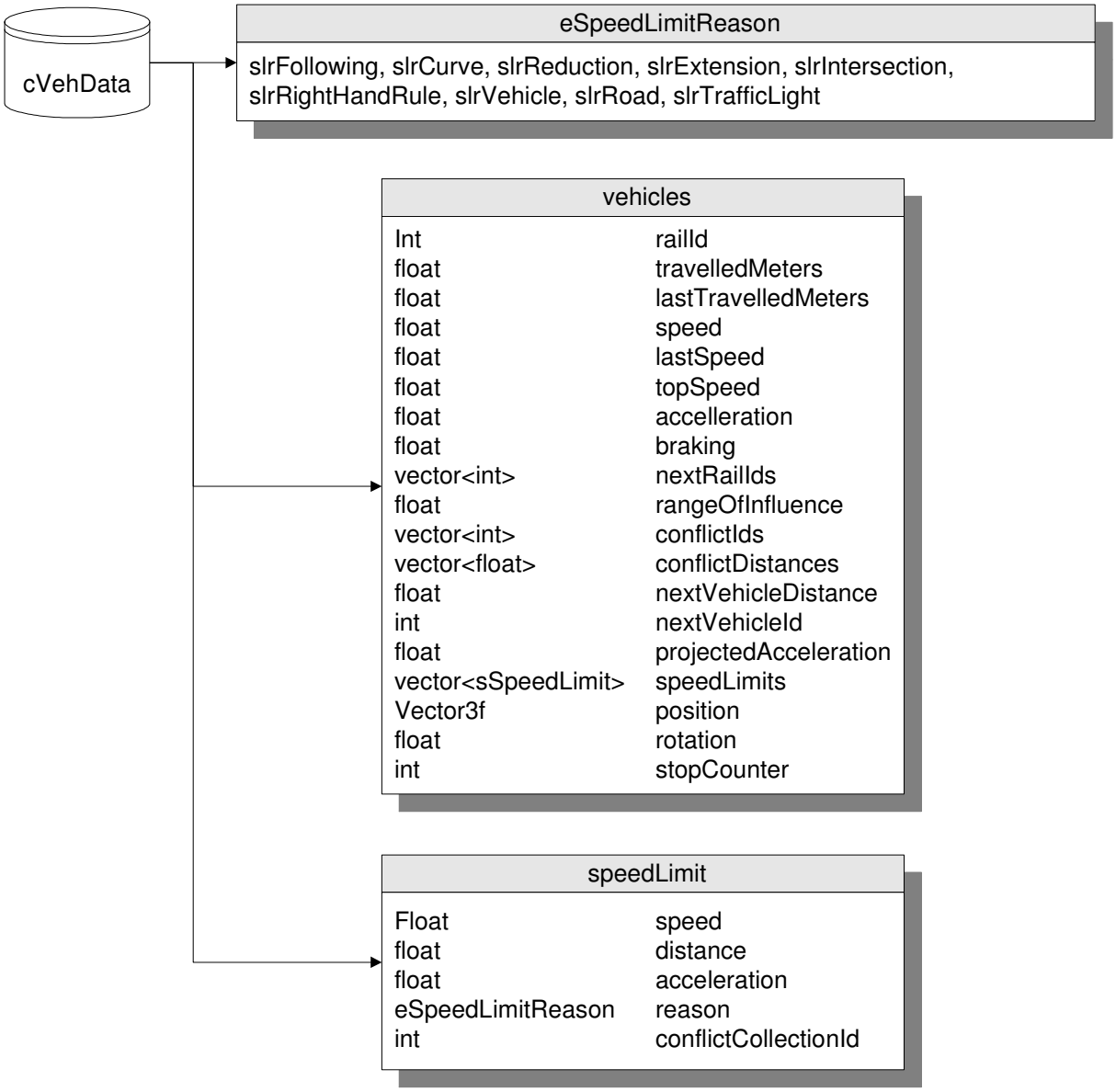


Abb. 26: Daten-Objekt VehicleData

5.4 Schnittstelle

In diesem Kapitel werden die Schnittstellen-Funktionen von TrafLib aufgelistet und kurz erklärt.

Todo – Schnittstelle wird noch erheblich erweitert!

- ***TrafLib()***

Konstruktor

- ***~TrafLib()***

Destruktor

- ***bool ConvertRoadGeometry(char *roadGeometryFileName, char *roadNetworkFileName)***

Liest eine Straßengeometrie-Beschreibungsdatei im VRMG[1] Format ein und speichert sie als Straßennetzwerk-Beschreibungsdatei im TrafLib-Format.

- ***bool CreateRoadNetworkFromFile(char *roadNetworkFileName)***

Liest ein Straßennetzwerk im TrafLib-Format für die Simulation aus einer Datei ein. Zusätzlich werden hier bei Bedarf Korridore, Ampeln und Verkehrszeichen generiert.

- ***void AddVehicle()***

Fügt ein Fahrzeug zur Simulation hinzu. Es wird eine zufällige Position auf einem zufälligen Fahrstreifen (nicht Kreuzungsbereich) gewählt, die nicht zu knapp bei einem anderen Fahrzeug liegt. Wird kein Platz gefunden, so wird kein Fahrzeug hinzugefügt.

- ***void AddVehicle(int maxSpeed, int maxAcceleration, int maxBreakingAction)***

Fügt wie die vorige Funktion ein neues Fahrzeug zur Simulation hinzu. Als Parameter werden die Werte für die maximale Geschwindigkeit, maximale Beschleunigung und maximale Bremsverzögerung übergeben.

- ***void SimulationStep()***

Führt einen Simulationsschritt aus. Diese Funktion sollte bei Echtzeitanwendungen mindestens alle 1/60 Sekunden aufgerufen werden. Todo: Das Hauptprogramm muss die verstrichene Zeit als Parameter übergeben.

Jede 1/2 Sekunde erfolgt die Verkehrsberechnung, durch die das Verhalten der Fahrzeuge angepasst wird.

Jede 1/60 Sekunde werden die Fahrzeuge weiterbewegt. D.h. entsprechend ihres vorher berechneten Verhaltens wird die Geschwindigkeit reduziert oder gesteigert und das Fahrzeug weiterbewegt.

- ***void SetVehicleTopSpeed(int vNr, float speed)***

Setzt die zulässige Höchstgeschwindigkeit für ein bestimmtes Fahrzeug.

- ***void SetVehicleSpeed(int vNr, float speed)***

Überschreibt die von der Bibliothek berechnete Geschwindigkeit für ein bestimmtes Fahrzeug. Dadurch kann Einfluss auf das Fahrverhalten genommen werden.

- ***float GetVehicleSpeed(int vNr)***

Retourniert die derzeitige Geschwindigkeit eines Fahrzeuges. Dies ist zum Beispiel sinnvoll, um Reifendrehung zu simulieren.

- ***int GetVehicleCount()***

Retourniert die Anzahl der derzeit aktiven Fahrzeuge.

- ***Vector3f GetVehicleCoord(int vNr)***

Retourniert die Koordinaten eines bestimmten Fahrzeuges.

- ***float GetVehicleRotation(int vNr)***

Retourniert die Fahrtrichtung eines Fahrzeuges.

6 Fahrzeug-Modell-Konverter (NFSCar2IV)

Eine weitere Aufgabe dieser Diplomarbeit war die Entwicklung von 10 geeigneten Fahrzeugmodellen. Gelöst wurde diese Aufgabe durch einen Konverter, der Fahrzeuge für das Spiel „Need 4 Speed IV“ umwandeln kann. Der Grund für diese Wahl war, dass es für dieses Spiel weit über 1000 frei zugängliche Fahrzeugmodelle gibt. Der Konverter kann diese Modelle in Formate für Open Inventor [36] und Ogre umwandeln.

6.1 Implementation

Verwendet wurden:

Hardware: Standardhardware

Entwicklungsplattform: Microsoft Windows 2000

Entwicklungswerkzeuge: Delphi 5.0, Open Inventor 2.6, Ogre 1.2.0

Der Fahrzeugmodell-Konverter kann Modelle für „Need 4 Speed IV“ im FCE Format einlesen und in die Formate .IV für Open Inventor sowie .MESH für Ogre umwandeln. Verwendet werden alle Modellteile außer den Lichtern und Spezialeffekten wie „Rauchgeneratoren“. Falls im Originalmodell LOD-Stufen vorhanden sind, werden diese ebenfalls konvertiert.

Da die meisten Fahrzeugmodelle von Laien entwickelt wurden, waren über 90% der Modelle nicht geeignet. Die Auswahl der Modelle erfolgte nach folgenden Gesichtspunkten:

- Ästhetik
- Anzahl der Dreiecksflächen (Je weniger desto besser)
- Backface-culling: Nur backface-culling-taugliche Modelle wurden verwendet.
- Modellfehler

6.2 Programmablauf

In Abb. 27 ist das Interface mit allen möglichen Optionen die der Konverter akzeptiert, abgebildet.

1. Programm NFSCar2IV.exe starten
2. Optionen auswählen
   i. Format: IV für Open Inventor oder MESH für Ogre
   ii. Erzeugung von LOD-Stufen an-/abwählen.
   iii. Die weiteren Optionen müssen im Normalfall nicht verändert werden.
3. Button „Convert File“ klicken
4. NFS-Fahrzeugmodell suchen. Unterstützt werden die Komprimierungsformate ZIP und VIV.
5. Der Konverter entpackt nun die VIV Datei und wandelt das Modell und die Texturgrafik in das gewählte Format um.

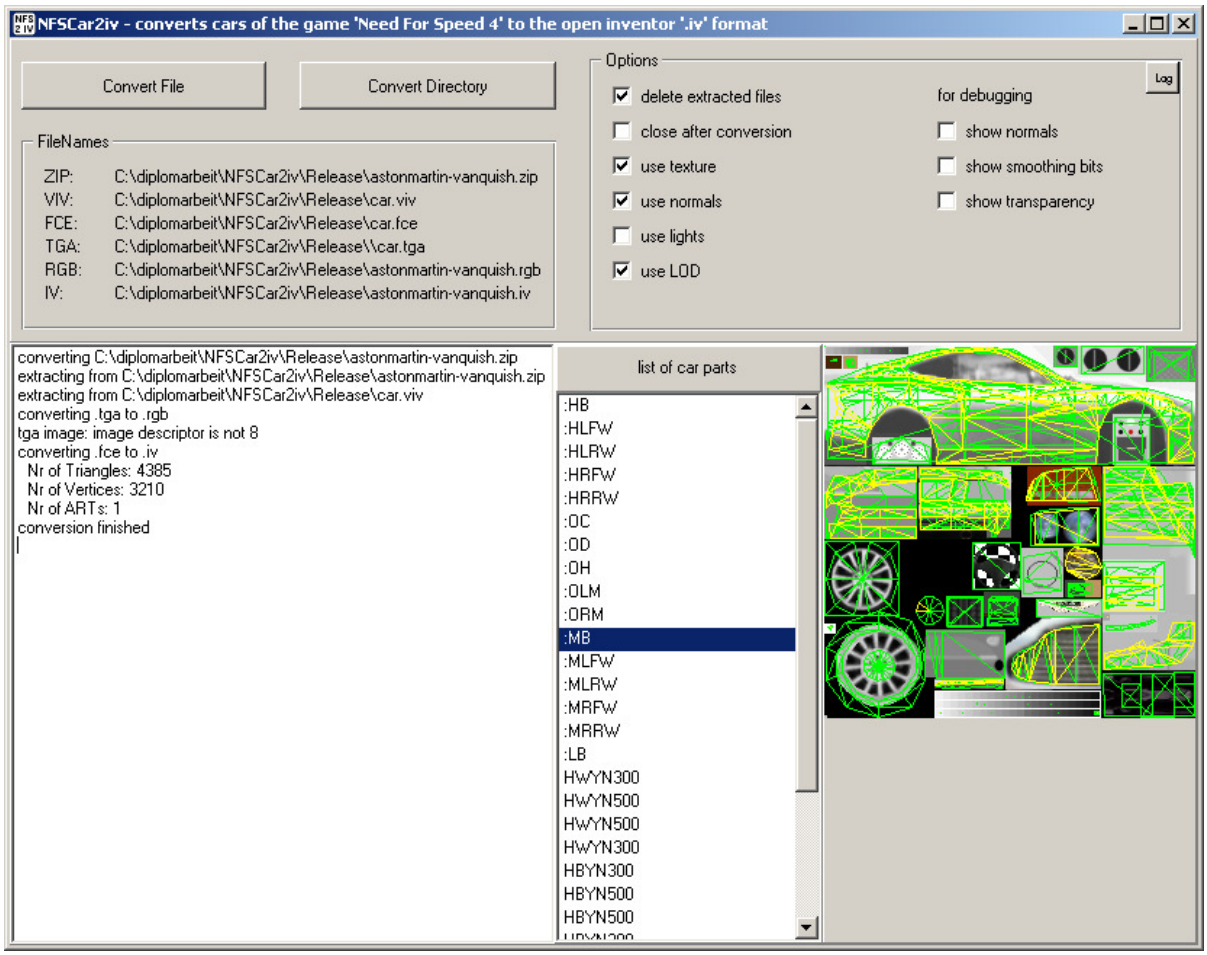


Abb. 27: Interface des Fahrzeug-Konverters

Bei der Konvertierung bekommen alle Modellteile einen Namen. Es ist dadurch möglich, zum Beispiel die Reifen ausfindig zu machen und während der Darstellung zu drehen.

Alle Glasflächen werden semi-transparent modelliert.

Abb. 28: Pkw im Open Inventor Format

Auflistung aller Modell-Teile in einem NFS4 Fahrzeug:

Abk.	**Bezeichnung**	**Beschreibung**
HB	High Body	Karosserie und Reifen in der höchsten Detailstufe
HLFW	High Left Front Wheel	
HRFW	High Right Front Wheel	
HLMW	High Left Middle Wheel	
HRMW	High Right Middle Wheel	
HLRW	High Left Rear Wheel	
HRRW	High Right Rear Wheel	
MB	Medium Body	Karosserie und Reifen in der 2. Detailstufe
MLFW	Medium Left Front Wheel	
MRFW	Medium Right Front Wheel	
MLMW	Medium Left Middle Wheel	
MRMW	Medium Right Middle Wheel	
MLRW	Medium Left Rear Wheel	
MRRW	Medium Right Rear Wheel	
LB	Low Body	Karosserie der LOD Stufe 3
TB	Tiny Body	Karosserie der LOD Stufe 4
OC	Interior	Innenausstattung
OND	Driver's chair and steering wheel	Fahrer + Fahrersitz + Lenkrad

OD	Driver holding steering wheel	
OH	Driver head	
ODL	Dash when lit	beleuchtetes Amaturenbrett
OLM	Left Mirror	Seitenspiegel
ORM	Right Mirror	
OLB	Left Front Brake	Bremsen
ORB	Right Front Brake	
OL	Popup lights	ausklappbare Scheinwerfer
OT	Top of convertibles	Dach eines Cabrios
OS	Optional spoiler	Spoiler

Tabelle 5: Modell-Teile einer FCE-Datei

7 TrafLib Demonstrationsprogramm

Zur Entwicklung von TrafLib war ein Testprogramm zur Entwicklungshilfe notwendig. Dieses Programm kann auch als Vorlage zur Einbindung von TrafLib in andere Software verwendet werden.

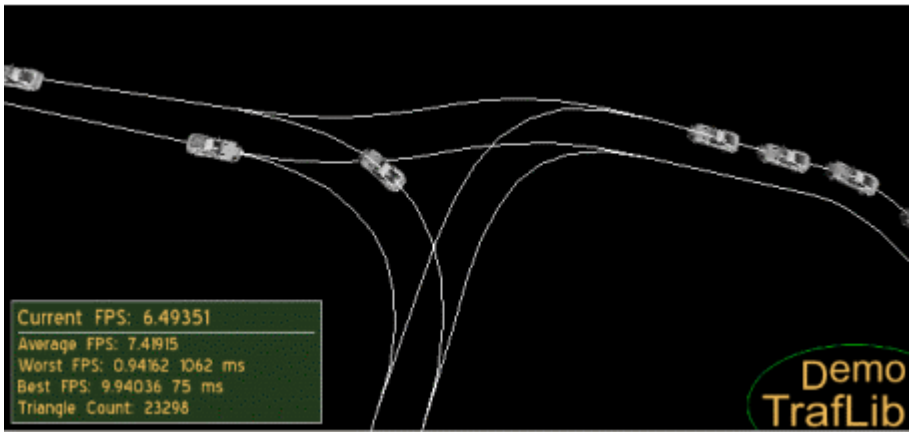


Abb. 29: TrafLib-Demo-Programm

7.1 Implementation

Verwendet wurden:

Hardware: Standardhardware

Entwicklungsplattform: Microsoft Window XP

Entwicklungswerkzeuge: MS Visual Studio 2005, Ogre 1.2.0

Programmiersprache: c++

Nachdem dieses Demoprogramm zum Testen entwickelt wurde, wurde kein Augenmerk auf Performance oder Ästhetik gelegt. Dafür lassen sich aber über Tastenkürzel verschiedene Informationen zum Debugging ein- und ausblenden.

8 Resultate + Bilder

Todo: gibt’s erst wenn TrafLib fertig und in URBANVIZ eingebaut

9 Mögliche Erweiterungen zu TrafLib

Folgende Erweiterungen von TrafLib wären denkbar:

- Unterstützung langer Fahrzeuge: Ausholen bei Kurven, Schleppkurve = kleinerer Radius der Hinterachse
- Fußgänger: Dazu müsste der Parser geändert werden, um Gehsteige (sidewalk) von VRMG [1] zu übernehmen.
- Einsatzfahrzeuge: Denkbar wären z.B. bevorrangte Fahrzeuge, wie Polizei, Rettung, ...
- Physik engine: Dadurch könnten z.B. Kollisionen behandelt werden, unterschiedliche Beschleunigungen bei verschiedenen Drehmomenten oder die Geschwindigkeitsänderung bei Steigungen berücksichtigt werden. Todo - Prag
- Fahrsimulation: Durch die Eingriffsmöglichkeiten über das Interface von TrafLib könnte ein Benutzer die Steuerung über ein Fahrzeug übernehmen.
- Kreisverkehr: Kreuzungen könnten durch Kreisverkehrsanlagen ersetzt werden. In [27] ist eine Möglichkeit zur Unfallvermeidung in Kreisverkehren beschrieben.
- Verkehrsplanung: Um TrafLib für die Verkehrsplanung brauchbar zu machen, müsste das Fahrverhalten mit der Realität verglichen und angepasst werden.

10 Zusammenfassung und Schlussfolgerung

In dieser Diplomarbeit wurden zuerst verschiedene Modelle und der state-of-the-art in der Verkehrssimulation für Verkehrsplanung präsentiert. Auf Grundlage dieser wurde eine spezielle Methode für die Bewegung von Fahrzeugen und die Regelung von Verkehr entwickelt und in den Verkehrssimulator TrafLib implementiert. Es wurde beschrieben, wie mit Hilfe von Konfliktpunkten die „Problemstellen“ aufgelöst wurden um einen unfallfreien Verkehr zu gewährleisten. Die entwickelte Bibliothek ist fähig, über 5000 Fahrzeuge auf einem 3x3 km großen städtischen Straßennetzwerk mit über 150 Kreuzungen in Echtzeit zu simulieren und einen flüssigen Ablauf der Fahrzeugbewegungen zu gewährleisten.

Abschließend kann festgestellt werden, dass die Verkehrssimulation mittels TrafLib einen realistischen Eindruck vermittelt. Die Reduktion der Fahrzeugbewegung auf eine eindimensionale Bewegung von Punkten auf Polygonzügen mindert die Optik in keiner Weise, reduziert den Rechenaufwand aber beträchtlich. Die Fähigkeit von TrafLib, Straßennetzwerke die von VRMG generiert wurden, zu benutzen, ist ein weiterer Vorteil, da mit VRMG große Straßennetze automatisiert erzeugt werden können.

Da TrafLib als Bibliothek implementiert ist, sind Erweiterungen durch andere Bibliotheken vorstellbar. Beispielsweise könnte eine Visualisierungssoftware, die TrafLib verwendet, mit einer weiteren Bibliothek zur Simulation von Fußgängern ausgebaut werden.

11 Literaturverzeichnis

[1] G. Hummel. Modellierung von Straßen für Echtzeitvisualisierung. *Diplomarbeit am Institut für Computergraphik und Algorithmen der Technischen Universität Wien, unter Anleitung von Prof. Dipl.-Ing. Dr. techn. Werner Purgathofer und Univ.-Ass. Dipl.-Ing. Dr .techn. Dieter Schmalstieg, 2001.*

[2] Wolfgang Pietzsch und Günter Wolf. Straßenplanung. 6.Aufl., Werner Ingenieur Texte 37, Copyright Werner Verlag Gmbh Düsseldorf, 2000.

[3] V.S.Rao Sasipalli, G.S.Sasipalli and Koichi Harada. Single Spirals in Highway Design And Bounds for their scaling. *IEICE Transactions on Information and System, Vol. E80-D, No. II, 1997.*

[4] Larry E. Owen, Yunlong Zhang, Lei Rao, Gene McHale. Traffic Flow Simulation using CORSIM. *Proceedings of the 2000 Winter Simulation Conference. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.*

[5] Donald T. Gantz, James R. Mekemson. Flow Profile Comparison of a Microscopic Car-Following Model and a Macroscopic Platoon Dispersion Model for Traffic Simulation. *Proceedings of the 1990 Winter Simulation Conference. Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)*

[6] Tobias Kiesling, Johannes Lüthi. Towards Time-Parallel Road Traffic Simulation. *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*

[7] Muhammad Abaidullah Anwar, Takaichi Yoshida. Integrating OO Road Network Database Cases and Knowledge for Route Finding. *Proceedings of the 2001 ACM symposium on Applied computing. Las Vegas, Nevada, United States. Pages: 215 – 219*

[8] Praveen Paruchuri, Alok Reddy Pullalarevu, Kamalakar Karlapalem. Multi Agent Simulation of Unorganized Traffic. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1 table of contents. Bologna, Italy.*

[9] Joseph C. Brill, Dudley E. Whitney. Development and Application of an Intermodal Mass Transit Simulation with Detailed Traffic Modeling. *Proceedings of the 1997 Winter Simulation Conference, ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson*

[10] Straßenplanung – Trassierung, bearbeitet von der Forschungsanstalt für das Straßenwesen, Arbeitsgruppe „Planung und Verkehr". Ausgabe Mai 1981.

[11] Arie Kaufmann. A Micro-Macro Simulation Model for a Signalized Network. *Annual Simulation Symposium, Florida International University Miami, Florida 33199*

[12] T. Schulze and T. Fliess. Urban Traffic Simulation with psycho-physical vehicle-following models. In *Proceedings of the 1997 Winter Simulation Conference, ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, pages 1222-1229.*

[13] Attila Elci, Ali Zambakoglu. City Traffic Simulation Package and its Utilization. *Proceedings ICD'82 lASTED Conference on Informatics and Control for Development, Tunis, Tunisia, Sept. I-3, 1982.*

[14] Uno Larsson. An Urban Traffic Simulation Model. *Todo*

[15] Shirish S. Joshi, Jeffrey D. Tew. An Improved Response Surface Methodology Algorithm with an Application to Traffic Signal Optimization for Urban Networks. *Proceedings of the 1995 Winter Simulation Conference ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman.*

[16] J. Wesley Barnes, Robert M. Crisp Jr. Simulation of the Four-Way and Two-Way Stop Sign Controlled Intersections. *Digital Government Conference'05, May 15-18, 2005, Atlanta, GA, USA.*

[17] Salvador Bayarri, Marcos Fernandez, and Mariano Perez. Virtual Reality for Driving Simulation. *Todo*

[18] Michael Balmer, Nurhan Cetin, Kai Nagel, Bryan Raney. Towards truly Agent-based Traffic and Mobility Simulations. *Proceedings of the Third International Joint Conference on AAMAS'04, July 19-23, 2004, New York, New York, USA.*

[19] J. A. Michon. A Critical View of Driver Behavior Models: What do we know, what should we do? Todo: *In: L. Evans and R. Schwing, eds. Human Behavior and Traffic Safety, New York, Plenum Press, pp. 485-520.*

[20] Talal Al-Shihabi, Ronald R. Mourant. A Framework for Modeling Human-like Driving Behaviours for Autonomous Vehicles in Driving Simulators. *Proceedings of the fifth international conference on Autonomous agents table of contents. Montreal, Quebec, Canada. Pages: 286 - 291*

[21] Winifred d. Ashton. The Theory of Road Traffic Flow. *London: Methuen & Co Ltd. New York: John Wiley & Sons Inc. 1966.*

[22] Muhammad Abaidullah Anwar, Takaichi Yoshida. Integrating OO Road Network Database, Cases and Knowledge for Route Finding. *Todo*

[23] Thomas M. Liebling. Graphentheorie in Planungs- und Tourenproblemen. *Springer Verlag Berlin – Heidelberg – New York 1970.*

[24] Michael A. Florian. Traffic Equlilibrium Methods. *Proceedings of the International Symposium. Held at the Université de Montréal, November 21-23, 1974.*

[25] Ulrich Klein. Traffic Simulation Based on the High Level Architecture. *Proceedings of the 1998 Winter Simulation Conference, D. J. Medeiros, E. F. Watson, J. S. Carson and M. S. Manivannan, eds.*

[26] Kiyoshi Yamada and Tenny N. Lam. Simulation Analysis of Two Adjecent Traffic Signals. *Proceedings of the 1985 Winter Simulation Conference. D. Gantz, G. Blais., S. Solomon (eds.)*

[27] Willi Bernhard, Peter Portmann. Traffic Simulation of Roundabouts in Switzerland. *Proceedings of the 2000 Winter Simulation Conference. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.*

[28] T. Ishikawa. Development of a road traffic simulator. *Vehicular Technology, IEEE Transactions onVolume 47, Issue 3, Aug. 1998 Page(s):1066 - 1071*

[29] J. Fawcett, P. Robinson. Adaptive routing for road traffic. *Computer Graphics and Applications, IEEEVolume 20, Issue 3, May-June 2000 Page(s):46 - 53*

[30] J. Esser, M. Schreckenberg. Microscopic simulation of urban traffic based on cellular automata. *International Journal of Modern Physics C, 8(5):1025–1036, 1997.*

[31] K. Nagel, D. E. Wolf, P. Wagner, P. Simon. Two-lane traffic rules for cellular automata: A systematic approach. *Physical Review E, 58(2):215–237, 1998.*

[32] J.Barceló, J.L.Ferrer, D. García and R. Grau. Microscopic Traffic Simulation for ATT Systems Analysis a Parallel Computing Version. *Universitat Politècnica de Catalunya, Barcelona, 1998.*

[33] S. O. Simonsson. A Road Traffic Simulation Model For Estimation Of Environmental Effects. *Vehicle Navigation and Information Systems, 1992. VNIS., The 3rd International Conference onSeptember 2-4, 1992 Page(s):483 - 489*

[34] S. O. Simonsson. Car-following as a tool in road traffic simulation. *Vehicle Navigation and Information Systems Conference, 1993., Proceedings of the IEEE-IEE12-15 Oct. 1993 Page(s):150 - 153*

[35] Viera K. Proulx. Traffic Simulation: A Case Study for Teaching Object Oriented Design. *SIGSCE 98 AtlantaGA USA 1998.*

[36] The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2, 1998.

[37] T. A. Ranney. Models of Driving Behavior: A Review of their Evolution. Accident Analysis and Prevention. *Todo: Vol. 26(6), 1994, pp. 733-750.*

[38] Ogre3D: http://www.ogre3d.org, 19.2.1007

12 Abbildungsverzeichnis

13 Tabellenverzeichnis