Konzeption und Entwicklung eines graphisch interaktiven Editors für Straßennetze

Diplomarbeit im Hauptfach
Methoden der Modellierung und Simulation (CSE)

vorgelegt von:
cand. tema. Frank Nägele

Betreuer:
Dipl.-Ing. Ingo Krems
Dipl.-Ing. Uwe Wössner

Stuttgart August 2010

Nr.: HLRS-D-0054

Vorwort

Die vorliegende Diplomarbeit bildet den Abschluss meines Studiums des Technologiemanagements an der Universität Stuttgart. Sie entstand im Rahmen einer Zusammenarbeit zwischen der Dr. Ing. h.c. F. Porsche AG und dem Höchstleistungsrechenzentrum Stuttgart (HLRS), Universität Stuttgart.
Mein Dank geht an Ingo Krems und Rainer Bernhard vom Porsche Entwicklungszentrum Weissach, die mir das Schreiben dieser Diplomarbeit ermöglichten und mich dabei unterstützten, sowie an Dr.-Ing. Uwe Wössner und seine Mitarbeiter von der Abteilung Visualisierung des HLRS, die mir ebenfalls während der Erstellung dieser Arbeit zur Seite standen, und natürlich an Professor Dr.-Ing. Michael Resch, dem Leiter des HLRS.

Frank Nägele, Stuttgart den 23. August 2010

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im modernen Entwicklungsprozess von Fahrzeugen gewinnt die Virtuelle Produktentwicklung aufgrund des hohen Kosten- und Zeitdrucks sowie stetig wachsender Kundenanforderungen zunehmend an Bedeutung. Bereits früh in der Entwicklungsphase lassen sich virtuelle Komponenten- oder Fahrzeugprototypen simulieren, visualisieren und analysieren - lange bevor die ersten realen Prototypen hergestellt werden.

Neben der getrennten Untersuchung von einzelnen Bauteilen, Komponenten und Softwaresystemen ist es sehr wichtig, auch den Menschen im Gesamtsystem Fahrzeug zu betrachten. Hier können virtuelle Fahrsimulatoren eingesetzt werden, beispielsweise zur Beurteilung der Nützlichkeit und Akzeptanz von Fahrerassistenzsystemen oder der Ergonomie von Anzeige- und Bedienelementen.

Insbesondere die Evaluierung von aktiven Sicherheitssystemen zur Unfallvermeidung ist in der Realitat nur mit großem Aufwand und hohen Kosten möglich. Über den Kostenaspekt hinaus finden sich aber auch eine ganze Reihe weiterer Vorteile, die für einen Einsatz von Fahrsimulatoren sprechen. So lassen sich am Simulator Gefahrensituationen erproben, ohne Probanden einem Sicherheitsrisiko aussetzen zu müssen. Die Reproduzierbarkeit der Testszenarien wird ebenso erleichtert wie die Bestimmung vieler für die Auswertung relevanter Kennwerte. Nicht zuletzt lassen sich Varianten verschiedener Komponenten oder Algorithmen am Simulator direkter vergleichen als in der Realität, da sie mit wenigen Handgriffen beziehungsweise Mausklicks auswechselbar sind. Aus einem großen Variantenspektrum kann so eine gute Vorauswahl getroffen werden, die schließlich in den realen Versuchen teilweise nur noch verifiziert werden muss.

Für die Simulation muss eine virtuelle Welt erschaffen werden, um dem Fahrer den Eindruck zu vermitteln, sich in einer echten Fahrsituation zu befinden. Hierzu gehören grafische Elemente wie die Fahrbahn, die unmittelbare Fahrbahnumgebung mit Leitplanken und Verkehrsschildern, aber auch Objekte in der näheren Umgebung wie Gebäude oder Bäume. Wichtig für die Akzeptanz der virtuellen Testszenarien sind außerdem andere Verkehrsteilnehmer, deren Verhalten durch künstliche Intelligenz gesteuert werden muss. Diese Fahrzeuge benötigen Informationen über die Streckenlogik, wie beispielsweise die örtliche Lage der Fahrbahn, die Anzahl der Fahrspuren oder den Gültigkeitsbereich von Geschwindigkeitsbeschränkungen.

Abbildung 1.1: Virtueller Fahrerplatz der Abteilung EID der Porsche AG im Entwicklungszentrum Weissach.

Der Virtuelle Fahrerplatz der Dr. Ing. h.c. F. Porsche AG (Abbildung 1.1) und der Fahrsimulator am Höchstleistungsrechenzentrum Stuttgart (HLRS) verwenden für die Definition der Streckenlogik das von der Daimler AG initiierte OpenDRIVE Format. In einer vorhergehenden Diplomarbeit wurde das Format stellenweise erweitert, um den Einsatz von intelligenten Fremdfahrzeugen zu erleichtern.

Ziel dieser Arbeit war es, ein Programm zu konzipieren und zu erstellen, das es erlaubt, Straßennetze nach dem OpenDRIVE Format inklusive besagten Erweiterungen zu modellieren.

Eine grafische Benutzeroberfläche soll dem Anwender dabei eine einfache und effiziente Bedienung ermöglichen. Das Rückgängigmachen und Wiederherstellen einzelner Arbeitsschritte und das Einblenden von Straßenkarten oder Satellitenbildern sollen den Arbeitsprozess ebenso erleichtern wie die Möglichkeit, Prototypen von komplexen Streckenteilen zu erstellen und somit wiederzuverwenden.

Kapitel 2 gibt einen kurzen Überblick über die grafische Benutzeroberfläche und die Gesamtstruktur des Programms. Darauf werden in Kapitel 3 die einzelnen Komponenten des Programms und ihre Kommunikation untereinander genauer beschrieben. In Kapitel 4 werden einige ausgewählte Problemstellungen der Implementierung und ihre Lösung vorgestellt. Abschließend wird in Kapitel 5 ein Ausblick und in Kapitel 6 eine Zusammenfassung gegeben.

Kapitel 2

Grundlagen

Bevor in den nachfolgenden Kapiteln detailliert auf einzelne Bereiche des im Rahmen dieser Diplomarbeit entstandenen Programms **ODD: OpenDRIVE Designer** eingegangen wird, soll hier die grobe Struktur des Programms vorgestellt werden. Abschnitt 2.1 gibt einen Überblick über die Benutzeroberfläche, die mithilfe der Klassenbibliothek Qt erstellt wurde. Anschließend wird in Abschnitt 2.2 auf das Architekturmuster Model-View-Controller eingegangen und erläutert, wie es als Grundlage für den Aufbau der Programmstruktur verwendet wurde. In Kapitel 2.3 schließlich wird das OpenDRIVE Format vorgestellt, in dem die Modelldaten importiert und exportiert werden.

2.1 Grafische Benutzeroberfläche

Eine grafische Benutzeroberfläche (engl. *graphical user interface*) dient als Schnittstelle zwischen Mensch und Computer. Sie soll vorallem auch unerfahrenen Benutzern die Datenverarbeitung leichter und effizienter gestalten, als dies mit Texteditoren oder rein kommandozeilenbasierten Programmen oft möglich ist.

Für die Programmierung der Benutzeroberfläche wurde die Klassenbibliothek Qt verwendet, die bis 2008 von der norwegischen Softwarefirma Trolltech entwickelt wurde, welche seitdem zum Nokia-Konzern gehört [1]. Mit Qt lässt sich plattformunabhängig programmieren. So werden mit Unix/Linux (mit X11), Mac OS X und Windows die derzeit gängigsten Betriebssysteme unterstützt. Ein duales Lizenzsystem erlaubt die Entwicklung sowohl von kom-

merzieller Software mit einer proprietären Lizenz als auch von freier Software mit der GNU General Public License (GPL). Seit Kurzem ist Qt auch mit der GNU Lesser General Public License (LGPL) erhältlich.

Als Programmiersprache wird für Qt vorrangig C++ verwendet, wobei Nokia mit Qt Jambi auch die Möglichkeit bietet, Java zu verwenden und eine Reihe unabhängiger Projekte Schnittstellen unter Anderem zu Python, Perl oder PHP anbieten. Da einige Konzepte von Qt, beispielsweise der *signals and slots* Mechanismus[1], über den Sprachstandard von C++ hinaus gehen, wird vor der eigentlichen Kompilierung ein Präprozessor, der sogenannte *meta object compiler* ausgeführt, der konformen C++ Quellcode erstellt.

Um den Programmcode sauber, flexibel und nachvollziehbar zu gestalten wurden zahlreiche Richtlinien und Empfehlungen von Meyers [2] sowie Henricson und Nyquist [3] umgesetzt. Die wichtigsten davon sind in Anhang A zusammengefasst und sollen als Programmierrichtlinien für Nachfolgearbeiten dienen.

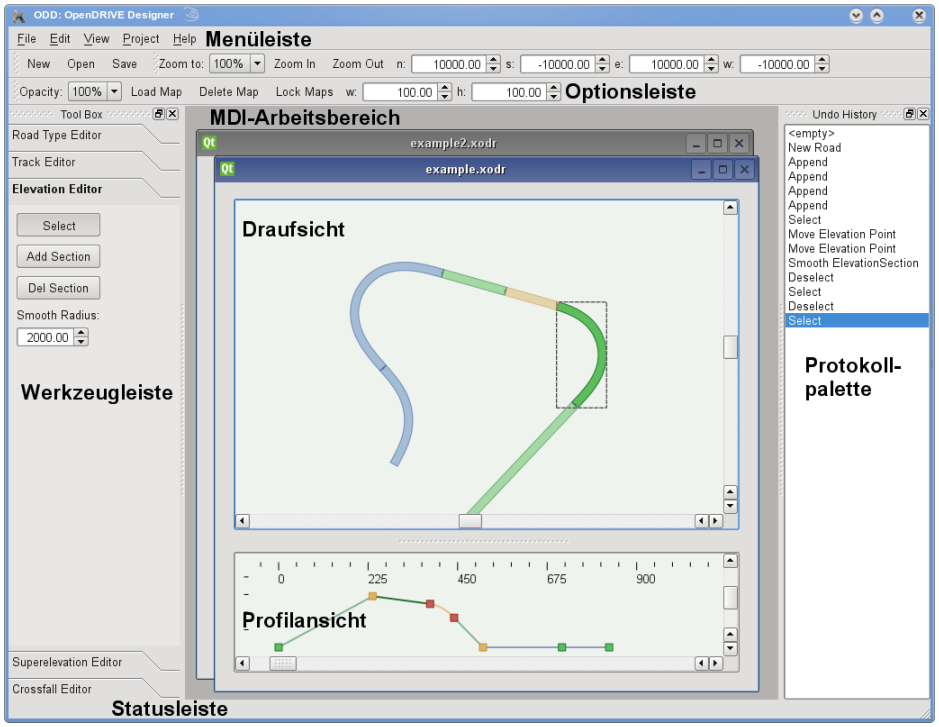


Abbildung 2.1: Grafische Benutzeroberfläche von ODD mit Multiple Document Interface.

[1] http://doc.trolltech.com/signalsandslots.html

Ein Screenshot des Programms ist in Abbildung 2.1 dargestellt. Ein Multiple Document Interface (MDI) ermöglicht es, gleichzeitig an mehreren Projekten zu arbeiten, wobei die Anwendung nur ein einziges Mal gestartet werden muss. Die Dokumente können dabei auf dem MDI-Arbeitsbereich über- oder nebeneinander angeordnet werden. Zwischen den Dokumenten kann mit der Tastenkombination `Strg+Tab` durchgeschaltet werden.

Der Arbeitsbereich eines Dokuments besteht aus einer Draufsicht sowie bei einigen Editoren zusätzlich aus einer Profilansicht, die im unteren Teil des Arbeitsbereichs eingeblendet wird.

Um den MDI-Arbeitsbereich herum sind die einzelnen Bedienelemente angeordnet:

- Die **Menüleiste** (*menu bar*) enthält Schaltflächen für wichtige allgemeine Funktionen, beispielsweise zum Speichern und Laden von Dateien, Rückgängigmachen und Wiederherstellen einzelner Arbeitsschritte sowie zum Zoomen der Ansicht.

- Die **Optionsleiste** (*tool bar*) ermöglicht den schnellen Zugriff auf ausgewählte Funktionen der Menüleiste.

- In der **Statusleiste** (*status bar*) werden aktuelle Informationen angezeigt wie zum Beispiel die Position des Mauszeigers oder Warnmeldungen.

- In der **Werkzeugleiste** (*toolbox*) befinden sich, aufgeteilt auf sechs Editoren, die einzelnen Werkzeuge, die zur Bearbeitung der Straßensysteme benötigt werden.

- Die **Protokollpalette** (*undo history*) zeigt eine Liste der zuletzt durchgeführten Arbeitsschritte.

2.2 Programm-Architektur

Der grundsätzliche Aufbau des Programms basiert auf dem Model-View-Controller Architekturmuster, es bietet sich aber auch eine Sichtweise als Schichtenmodell an. Die beiden Konzepte sollen in den nächsten Abschnitten kurz vorgestellt werden.

2.2.1 Model-View-Controller

Das Model-View-Controller (MVC) Architekturmuster wurde 1978/79 von Trygve Reenskaug am Xerox Palo Alto Research Laboratory (PARC) konzipiert [4]. Ein großer Teil heutiger Softwareprojekte verwendet MVC zur groben Strukturierung, wobei das Konzept jedoch sehr unterschiedlich interpretiert wird.

Zentral ist die Aufteilung des Programms in die drei Teile Modell (engl. *model*), Präsentation (*view*) und Steuerung (*controller*). Das Modell enthält dabei ausschließlich Daten, ist also unabhängig von der Visualisierung der Daten und von der Verarbeitung der Benutzerinteraktionen. Diese Entkopplung führt zu weniger Abhängigkeiten in der Programmstruktur und erleichtert damit Änderungen im Programmcode.

Die Aufgabe der Präsentation ist es, ausgewählte Daten des Modells darzustellen und Benutzerinteraktionen an die Steuerung weiterzugeben, die diese verarbeitet und gegebenenfalls die Daten des Modells manipuliert. Eine Trennung von Präsentation und Steuerung wird jedoch oft nicht strikt umgesetzt. Zudem schreibt das ursprüngliche MVC-Konzept nicht vor, wo die sogenannte Geschäftslogik untergebracht wird, also derjenige Teil, der die Eingaben des Nutzers interpretiert und entscheidet, welche Daten des Modells geändert werden sollen. Oft ist die Geschäftslogik Teil der Steuerung, sie kann aber auch zum Modell gehören oder in einer Schicht dazwischen liegen.

Abbildung 2.2 zeigt ein stark reduziertes Klassendiagramm des Programms und verdeutlicht, wie das MVC-Konzept im Rahmen dieser Arbeit umgesetzt wurde. Beim Programmstart wird ein Objekt der Klasse `MainWindow` erstellt, dem sämtliche grafischen Bedienelemente wie Menüs und Werkzeugleisten zugeordnet werden. Ebenfalls ein Teil des `MainWindow` ist der MDI-Arbeitsbereich vom Typ `QMdiArea`. Für jedes vom Benutzer geöffnete oder neu erstellte Dokument wird ein `ProjectWidget`-Objekt angelegt, das wiederum die Objekte der MVC-Klassen enthält. Zu jedem Projekt gehört demnach ein `ProjectWidget`-Objekt mit einem **eigenen** Satz an MVC-Klassen.

2.2.2 Schichtenmodell

Eine weitere Perspektive soll den Aufbau des Programms und die Unabhängigkeit der Modellklassen verdeutlichen. In Abbildung 2.3 ist ein Schichtenmodell des Programms dargestellt. Die erste Schicht (`ProjectData`) enthält sämtliche Daten des Modells, diese können von allen anderen Schich-

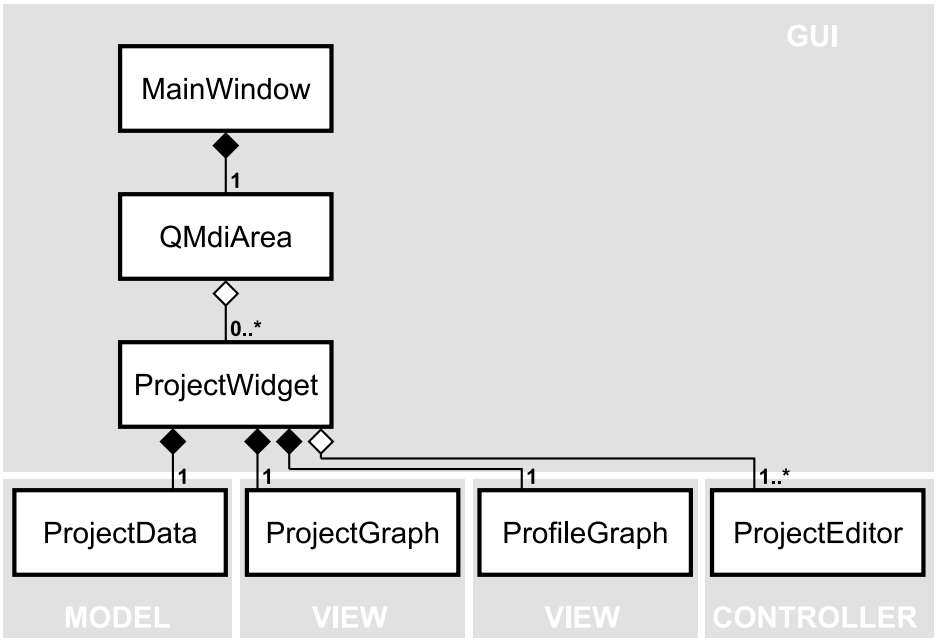


Abbildung 2.2: Klassendiagramm des Programms.

ten gelesen werden (gestrichelte Linien).

Die zweite Schicht wird durch `Command`- und `Visitor`-Unterklassen gebildet. Nur von der `Command`-Basisklasse abgeleitete Unterklassen sind berechtigt, Daten des Modells zu manipulieren. Jede Veränderung der Modelldaten ist somit in einem Objekt gekapselt, wodurch das Rückgängigmachen und Wiederherstellen einzelner Arbeitsschritte ermöglicht wird (siehe Kapitel 3.1.3). Das Visitor-Pattern vereinfacht das Durchschreiten der Objekthierarchie, was sich einige Hilfsfunktionen zu Nutzen machen (siehe Kapitel 3.1.2).

Zur dritten Schicht gehören schließlich die Klassen der Präsentation und der Steuerung. Die Präsentation wird mithilfe des Observer-Pattern (Kapitel 3.1.4) über Änderungen der Modelldaten informiert. Auch in dieser Schicht werden `Visitor`-Unterklassen verwendet.

An der Richtung der eingetragenen Pfeile lässt sich nun deutlich ablesen, dass jede Schicht nur auf Objekte der vorhergehenden Schichten und der eigenen Schicht Zugriff hat (mit Ausnahme der sehr eingeschränkten `Acceptor-Visitor` und `Subject-ChangeManager-Observer` Verbindungen). Dies verdeutlicht, dass eine Änderung in der zweiten und dritten Schicht keinerlei Einfluß auf die `ProjectData`-Schicht haben kann und somit Änderungen im Programmcode wie erwähnt erleichtert werden.

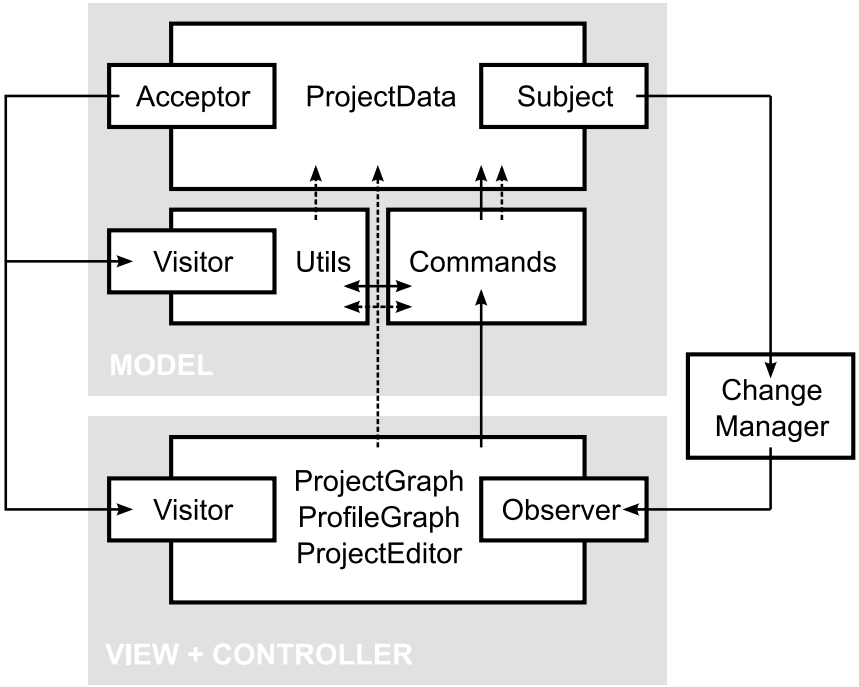


Abbildung 2.3: Schichtenmodell des Programms. Durchgezogene Linien stellen Schreibzugriff dar, gestrichelte Linien Lesezugriff.

2.3 Datenmodell: OpenDRIVE

Das OpenDRIVE Format geht auf eine Initiative der VIRES Simulationstechnologie GmbH und des Daimler Driving Simulator zurück [5, 6]. Ziel war es, einen Standard für die Beschreibung der Logik von Straßennetzen zu schaffen, um den Datenaustausch zwischen den Betreibern von Fahrsimulatoren zu erleichtern. Eine erste Version wurde im Januar 2006 veröffentlicht, die aktuelle Version 1.2 im Januar 2008.

Dieses Kapitel soll die grundlegenden Konzepte des OpenDRIVE Formats erklären. Abschnitt 2.3.1 beschreibt, wie aus einzelnen Straßen und Kreuzungen Straßennetze erstellt werden, Abschnitt 2.3.2 den allgemeinen Aufbau einer Straße. In den Abschnitten 2.3.3 bis 2.3.7 werden schließlich die einzelnen Bestandteile einer Straße genauer betrachtet.

2.3.1 Verknüpfung von Straßen

Eindeutige Verknüpfungen

Besitzt eine Straße maximal einen Vorgänger (engl. *predecessor*) und maximal einen Nachfolger (engl. *successor*), so ist die Verknüpfung eindeutig. Zu beachten ist, dass jede Straße eine Richtung besitzt. Das heißt, jede Straße speichert zusätzlich zu der Information, welche Straße vorangeht beziehungsweise welche folgt, ob sie mit deren Ende oder Anfang verbunden ist. Abbildung 2.4 zeigt ein Beispiel mit drei Straßen. In Tabelle 2.1 sind dazu die Verknüpfungen aufgelistet. Das Ende der Straße R_1 ist beispielsweise mit dem Anfang von R_2 verbunden, das Ende von R_2 mit dem Ende von R_3.

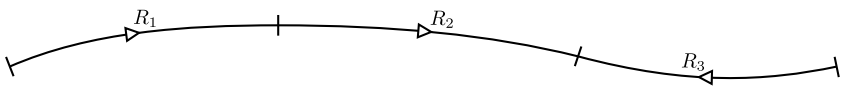


Abbildung 2.4: Eindeutige Verknüpfung von Straßen.

In der vorangegangenen Arbeit von Seybold [7] wurde das OpenDRIVE Format um sogenannte Fiddleyards erweitert. Diese können als Quellen und Senken für Fremdfahrzeuge agieren. Sie werden wie normale Straßen verbaut und verknüpft.

Tabelle 2.1: Beispiel zur eindeutigen Verknüpfung von Straßen: Liste der Vorgänger und Nachfolger.

Predecessor	Road	Successor
-	R_1	R_2 start
R_1 end	R_2	R_3 end
-	R_3	R_2 end

Mehrdeutige Verknüpfungen

Endet eine Straße an einer Kreuzung, so ist der Nachfolger nicht mehr eindeutig. In Abbildung 2.5 ist eine Straße R_1 abgebildet, die sich verzweigt. Um diese Verzweigung zu repräsentieren, wird ein Kreuzungsobjekt J_1 erstellt (engl. *junction*), das bei der Straße als Nachfolger eingetragen wird (siehe Tabelle 2.2). Die Kreuzung selber besteht aus Pfaden, im Beispiel die Pfade P_1, P_2 und P_3, die außer der Kennzeichnung als Pfad wie normale Straßen behandelt werden. Vorgänger und Nachfolger der Pfade sind im Beispiel wieder eindeutig und können dementsprechend normal definiert werden.

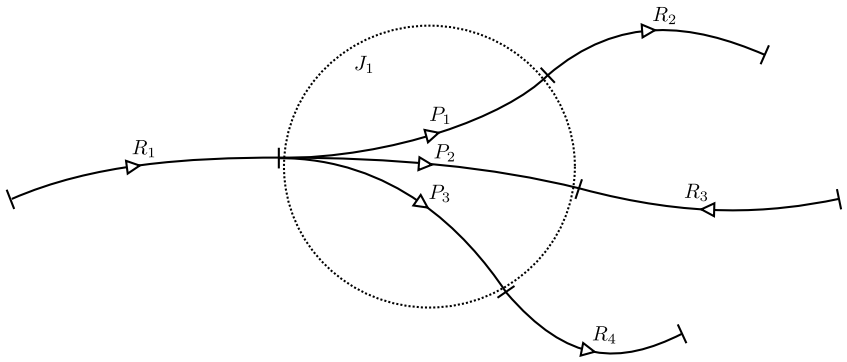


Abbildung 2.5: Mehrdeutige Verknüpfung von Straßen durch Kreuzungen.

2.3.2 Straßensegmente

Die Trasse einer Straße (OpenDRIVE: *centerline*, *chord line*, *track*) wird durch Geometrieelemente in der xy-Ebene definiert, wobei laut Definition x nach Osten, y nach Norden und z nach oben zeigt. Es bietet sich aber

Tabelle 2.2: Beispiel zur mehrdeutigen Verknüpfung von Straßen: Liste der Vorgänger und Nachfolger.

Predecessor	Road	Successor
-	R_1	J_1
R_1 end	P_1	R_2 start
R_1 end	P_2	R_3 end
R_1 end	P_3	R_4 start
P_1 end	R_2	-
-	R_3	P_2 end
P_3 end	R_4	-

Tabelle 2.3: Beispiel zur mehrdeutigen Verknüpfung von Straßen: Liste der Verbindungen der Kreuzung J_1.

Incoming Road	Connecting Path
R_1	P_1 start
R_1	P_2 start
R_1	P_3 start

an, auch ein relatives Straßenkoordinatensystem zu definieren, wobei s in Longitudinalrichtung also entlang der Trasse zeigt und t in Transversalrichtung quer zur Trasse. In Abbildung 2.6 sind zur Veranschaulichung die drei Straßen R_1, R_2 und R_3 jeweils mit ihrem eigenen Straßenkoordinatensystem dargestellt.

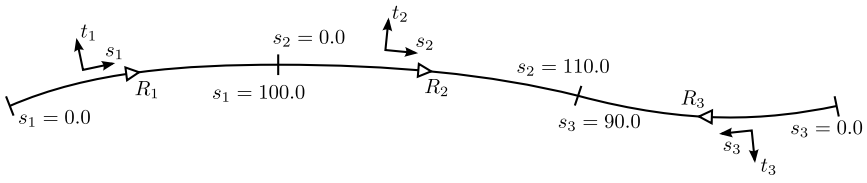


Abbildung 2.6: Drei Straßen jeweils mit eigenem Straßenkoordinatensystem.

Die Beschaffenheit einer Straße wird durch Segmente definiert, wobei jedem Straßensegment (engl. *road section*) S_i eine Startkoordinate s_{S_i} zugeordnet wird, bei der es beginnt. Ein Segment ist von s_{S_i} an solange gültig, bis Seg-

ment S_{i+1} beginnt (oder die Straße endet), also im Interval $[s_{S_i}, s_{S_{i+1}})$. Das erste Segment sollte bei $s_{S_1} = 0.0$ beginnen. In Abbildung 2.7 ist eine Straße R_1 mit drei Segmenten dargestellt.

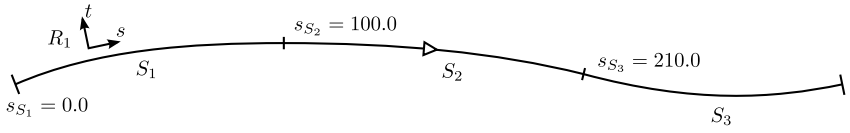


Abbildung 2.7: Eine Straße R_1 mit drei Straßensegmenten S_1, S_2 und S_3.

Es gibt neun Arten von Straßensegmenten, die eine Straße komplett beschreiben, wobei sechs davon in der Arbeit von Seybold [7] umgesetzt wurden. Diese sechs Segmentarten sollen in den folgenden fünf Abschnitten kurz erläutert werden.

2.3.3 Lageplan (planView)

Wie bereits erwähnt wird die Lage in der xy-Ebene durch Geometrieelemente beschrieben. Diese setzen sich aus den Grundelementen Gerade, Kreisbogen und Klothoide zusammen.

Da Geradenstücke die Krümmung

$$\kappa_{line} = 0.0 \tag{2.1}$$

besitzen und Kreisbögen die konstante Krümmung

$$\kappa_{arc} = \frac{1}{r_{arc}}, \tag{2.2}$$

wobei r_{arc} der Kreisradius ist, muss zwischen Geradenstücken und Kreisbögen ein Übergangselement verbaut werden, um eine ruckfreie Fahrt und eine gleichmäßige Lenkradbewegung zu ermöglichen. Hierfür bieten sich Klothoiden an, deren Krümmung

$$\frac{\mathrm{d}\kappa_{spiral}}{\mathrm{d}s} = const \tag{2.3}$$

linear mit dem Weg zunimmt [8].

Abbildung 2.8 zeigt einen solchen Übergangbogen TS_2 zwischen dem Geradenstück TS_1 und dem Kreisbogenstück TS_3. Eine genauere mathematische Betrachtung der Grundelemente findet sich in Kapitel 4.1.

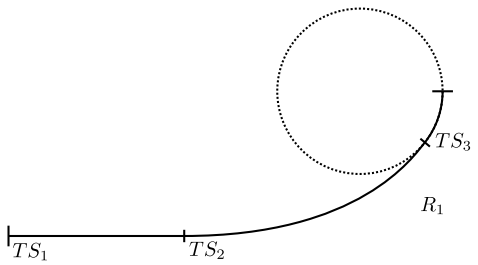


Abbildung 2.8: Eine Straße R_1 mit den drei Geometrieelementen TS_1 (Gerade), TS_2 (Klothoide) und TS_3 (Kreisbogen).

2.3.4 Höhenplan (elevationProfile)

Das Höhenprofil einer Straße wird komplett unabhängig vom Lageplan definiert und besteht aus mehreren Höhensegmenten ES_i. Für jedes Höhensegment der Straße wird ein Polynom dritten Grades angegeben. Die Höhe eines beliebigen Punktes s der Straßenachse berechnet sich nun zu

$$z = a_i + b_i\, ds_i + c_i\, ds_i^2 + d_i\, ds_i^3 \tag{2.4}$$

wobei

$$ds_i = s - s_i \tag{2.5}$$

der Abstand des Punktes s zum Beginn des aktuellen Höhensegmentes ES_i ist.

2.3.5 Querneigung (lateralProfile)

Bei der Berechnung der Höhe von Punkten abseits der Straßenachse muss die Querneigung berücksichtigt werden. Diese soll den Abfluss von Regenwasser begünstigen und während einer Kurvenfahrt die auf die Insassen wirkenden Kräfte reduzieren [9]. Der OpenDRIVE Standard berücksichtigt zwei Arten von Querneigungen, die sich überlagern lassen. Abbildung 2.9 zeigt eine Neigung der kompletten Fahrbahn, wie sie beispielsweise in Kurven verwendet wird. Dabei beschreibt ein positiver Winkel α ein Abkippen nach rechts. Abbildung 2.10 zeigt ein sogenanntes Dachprofil, das sich speziell für Geraden eignet.

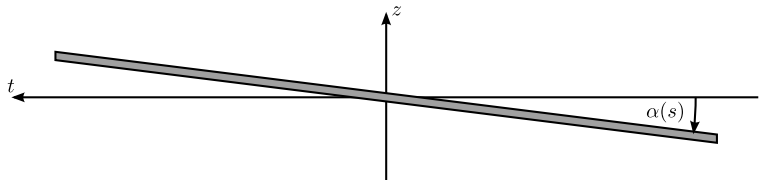


Abbildung 2.9: Querneigung der Fahrbahn.

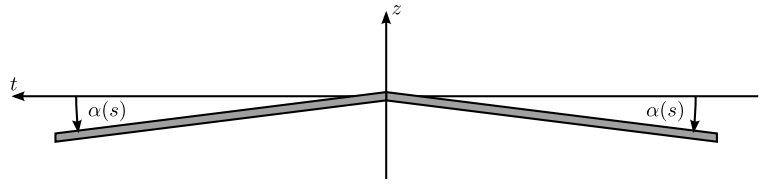


Abbildung 2.10: Querneigung in Form eines Dachprofils.

2.3.6 Querschnitt (lanes)

Die Fahrbahn einer Straße teilt sich auf in Fahr-, Rand-, Trenn- und Seitenstreifen sowie Rad- und Gehwege [10]. In OpenDRIVE werden diese Streifen (engl. *lanes*) von der Mitte aus durchnummeriert, in t-Richtung positiv, in negative t-Richtung dementsprechend negativ. Die Straßenachse hat den Index 0 und per Definition die Breite 0.0. Für alle anderen Streifen lässt sich eine Breite und ein Typ angeben sowie gegebenenfalls die Art der Fahrbahnmarkierungen und die erlaubte Geschwindigkeit. Jedes Segment hat über seine gesamte Länge eine konstante Anzahl an Streifen, die Indices müssen fortlaufend sein und dürfen keine Lücken haben. In Abbildung 2.11 ist beispielhaft eine Straße mit den zwei Segmenten LS_1 und LS_2 dargestellt, wobei Segment LS_1 aus sieben Streifen besteht, Segment LS_2 aus acht.

2.3.7 Straßentyp (type)

Die letzte Segmentart, die hier beschrieben werden soll, gibt den Straßentyp an. Hier stehen fünf Typen zur Auswahl, die in Deutschland Fußgänger-

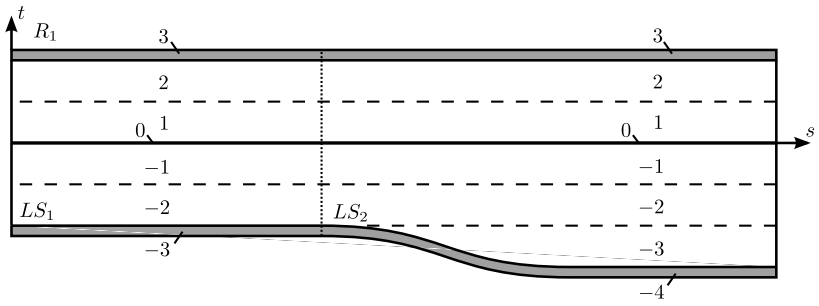
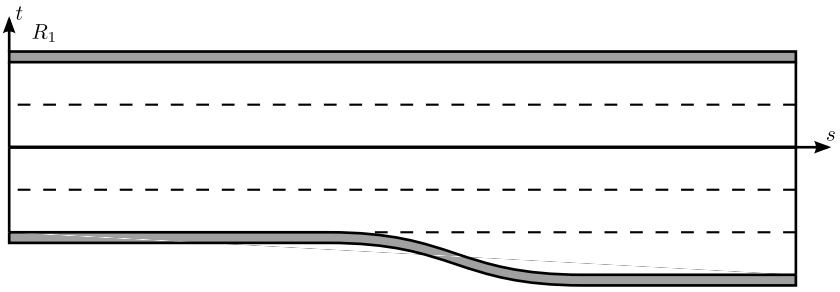


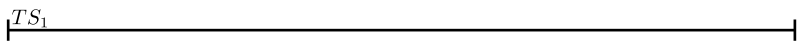
Abbildung 2.11: Straße mit mehreren Fahrstreifen.

zonen (`pedestrian`), Tempo-30-Zonen (`lowSpeed`), Innerorts- (`town`) und Außerorts-Straßen (`rural`) sowie Autobahnen (`motorway`) entsprechen.

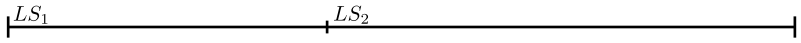
Abschließend soll ein einfaches Beispiel gegeben werden, um zu verdeutlichen, wie sich eine Straße aus den vorgestellten Arten an Segmenten zusammensetzt. Die Straße R_1 ist in Abbildung 2.12a dargestellt und besteht aus einem Geradenstück TS_1 (2.12b). Da die Anzahl der Fahrstreifen einer Straßenseite von zwei auf drei zunimmt, sind zwei Querschnittsegmente LS_1 und LS_2 nötig (2.12c). Die Verbreiterung der Straße soll den Übergang einer Landstraße in eine Autobahn darstellen. Das Straßentypsegment RS_1 ist demnach vom Typ `rural`, Segment RS_2 vom Typ `motorway` (2.12d). Zuletzt ist noch ein Höhenprofil angegeben, das aus vier einzelnen Segmenten besteht (2.12e).



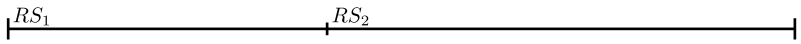
(a) Die Straße R_1.



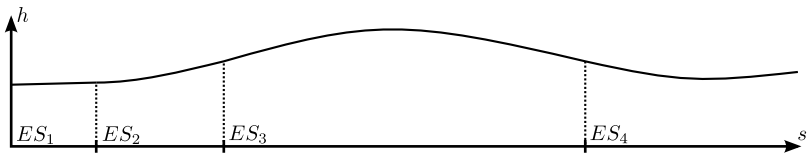
(b) Ein TrackElement TS_1.



(c) Zwei LaneSections LS_1 und LS_2.



(d) Zwei RoadTypeSections RS_1 und RS_2.



(e) Vier ElevationSections ES_1 bis ES_4.

Abbildung 2.12: Beispiel einer Straße mit verschiendenen Segmenten.

Kapitel 3

Programm-Design

In Kapitel 2 wurde bereits die grobe Struktur des Programms vorgestellt. Im Vordergrund stand dabei die Trennung des Programmcodes in Modell sowie Präsentation und Steuerung. In diesem Kapitel soll nun die detaillierte Beschreibung einzelner Komponenten vorgenommen werden.

Es hat sich bei der Entwicklung der Systemkomponenten als sehr hilfreich erwiesen, auf bewährte Entwurfsmuster (engl. *design patterns*) der Softwareentwicklung zurückzugreifen. Die Benennung der Entwurfsmuster orientiert sich dabei an den Namen, wie sie im Standardwerk *Design Patterns - Elements of Reusable Object-Oriented Software* [11] verwendet werden.

Kapitel 3.1 beschreibt die allgemeine Klassenstruktur des Modells und geht dann auf einige zentrale Problemstellungen ein und wie sie mithilfe von Entwurfsmustern gelöst wurden. Analog wird anschließend in Kapitel 3.2 auf die Klassen der Präsentation und der Steuerung eingegangen.

3.1 Modell (model)

Abbildung 3.1 zeigt ein Klassendiagramm der obersten Ebenen des Modells. Die Hierarchie teilt sich unterhalb der zentralen Klasse `ProjectData` in die Bereiche RoadSystem, SceneSystem und VehicleSystem. Während das RoadSystem sich größtenteils an der OpenDRIVE Struktur orientiert (mit Ausnahme der Fiddleyards), sind im Scenery- und VehicleSystem die Ergänzungen von Seybold und die in dieser Arbeit hinzugekommenen SceneryMaps zusammengefasst (siehe Kapitel 4.4 und Anhang B).

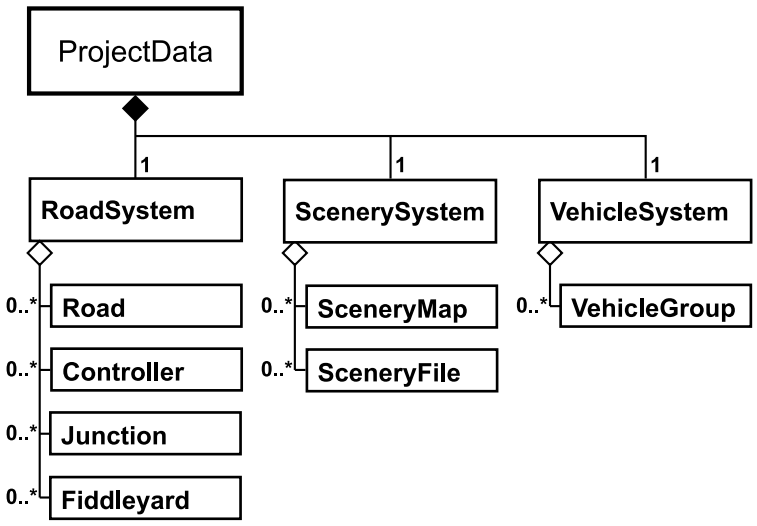


Abbildung 3.1: Klassendiagramm ProjectData.

Das Klassendiagramm in Abbildung 3.2 stellt die Hierarchie der Straßendaten dar. Zu jeder Straße können beliebig viele Straßensegmente der sechs Segmentarten hinzugefügt werden, die in Kapitel 2.3 vorgestellt wurden. Es muss jedoch mindestens je ein Objekt der Typen `TrackComponent` und `LaneSection` vorhanden sein, sodass die Straße eine räumliche Lage und Ausdehnung besitzt.

Die Klasse `DataElement` ist die Basisklasse für sämtliche Klassen des Modells (Abbildung 3.3). Sie erbt wiederum von den Klassen `Acceptor` und `Subject` und stattet damit die Klassen des Modells mit der Funktionalität des Visitor-Pattern (Abschnitt 3.1.2) und des Observer-Pattern (Abschnitt 3.1.4) aus.

3.1.1 Composite-Pattern

Mit dem Composite-Pattern (dt. *Kompositum-Entwurfsmuster*) lassen sich sogenannte Teil-Ganzes-Hierarchien realisieren [11]. Es wird eine Baumstruktur aus Blatt- (engl. *leaf*) und Verbundknoten (engl. *composite*) erstellt, wobei alle Objekte der Baumstruktur gleich behandelt werden können, da sie die gemeinsame Basisklasse `Component` besitzen. Ein Klassendiagramm dazu ist in Abbildung 3.4 dargestellt.

Abbildung 3.5 zeigt, wie das Entwurfsmuster eingesetzt wurde, um eine Hier-

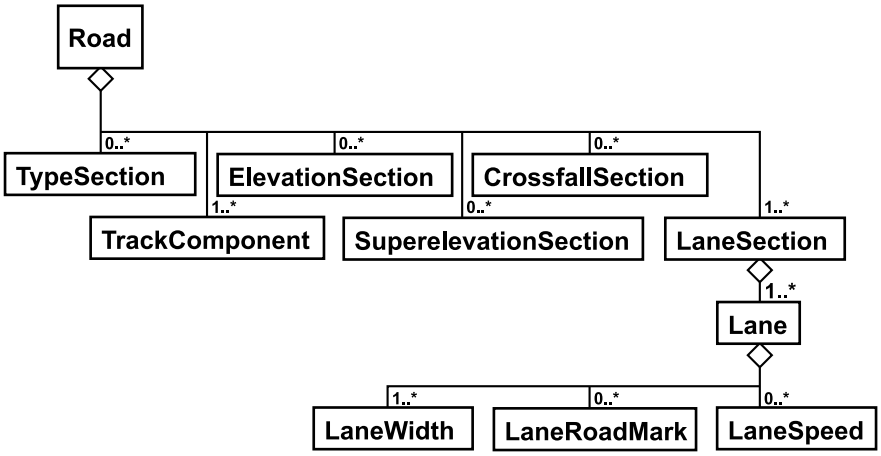


Abbildung 3.2: Klassendiagramm Road.

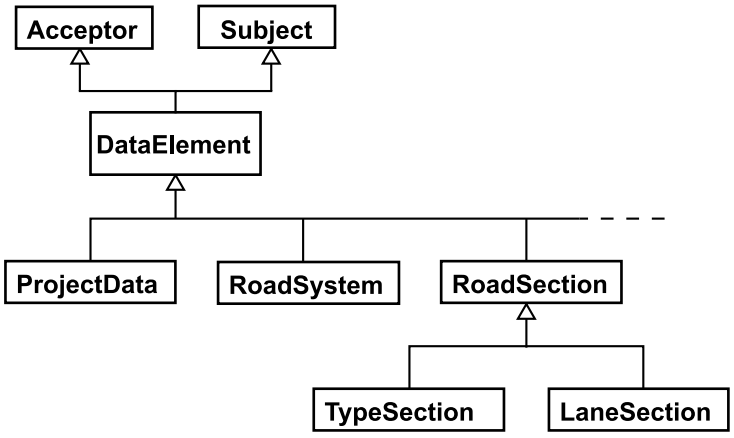


Abbildung 3.3: Vererbungshierarchie DataElement.

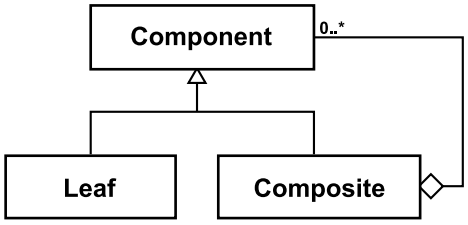


Abbildung 3.4: Klassendiagramm zum Composite-Pattern.

archie aus geometrischen Elementen zu erschaffen. Die Hierarchie besteht aus mehreren Blattknoten, welche die in Kapitel 2.3.3 vorgestellten geometrischen Grundelemente Gerade, Kreisbogen und Klothoide repräsentieren. Diese Grundelemente lassen sich zu Verbunden zusammenschließen. Bisher wurde eine Klothoide-Kreisbogen-Klothoide-Verbundkurve umgesetzt, die das Arbeiten mit Klothoiden erleichtert, wie in Kapitel 4.1 beschrieben wird.

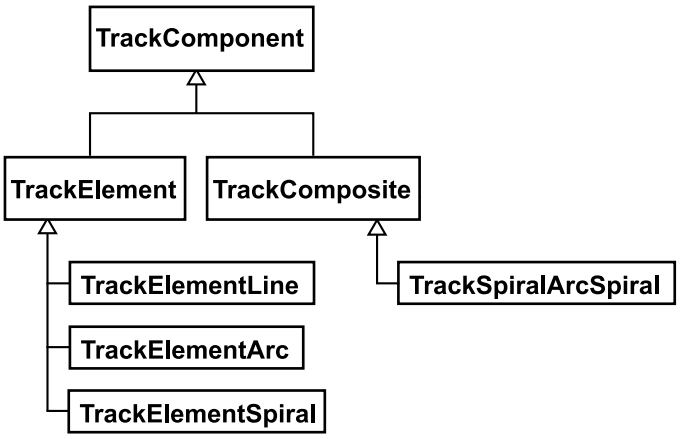


Abbildung 3.5: Vererbungshierarchie TrackComponent.

3.1.2 Visitor-Pattern

Wie in Kapitel 2.2 beschrieben wurde, ist es sinnvoll, das Modell möglichst schlank und unabhängig zu halten, damit das Programm flexibel und leicht erweiterbar bleibt. Deshalb sollten beispielsweise Funktionen zum Export

von Daten nicht in die Klassen des Modells selbst integriert werden, da sich ansonsten mit jedem hinzukommenden Exportformat das Modell weiter aufblähen würde. Änderungen an den Exportfunktionen würden sich schwierig gestalten, da diese über das gesamte Modell verteilt sind. Auch würde das Hinzufügen eines neuen Satzes an Funktionen bedeuten, dass ein großer Teil des Modells neu kompiliert werden müsste.

All diese Probleme lassen sich durch die Verwendung des Visitor-Pattern (dt. *Besucher-Entwurfsmuster*) lösen [11]. Dabei werden ähnliche Funktionen nicht über das Modell verteilt, sondern zentral in einer `Visitor`-Klasse definiert. Die Klassen der Modellhierarchie müssen nur noch jeweils eine Funktion accept(Visitor *) definieren, die einen Visitor annimmt, indem sie die Funktion visit() des Visitors mit sich selbst als Argument aufruft.

Man erhält demnach zwei Klassenhierarchien. Einerseits die Hierarchie der Modellklassen, die von der Basisklasse `Acceptor` abgeleitet wurden, andererseits die Hierarchie der Visitor-Klassen mit der gemeinsamen abstrakten Basisklasse `Visitor`. In den Abbildungen 3.6 und 3.7 sind die zwei Klassenhierarchien des Visitor-Pattern bildlich dargestellt.

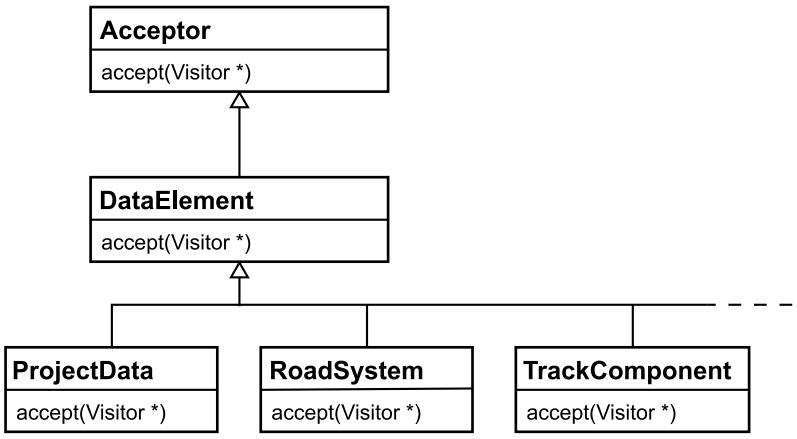


Abbildung 3.6: Klassendiagramm zum Visitor-Pattern: Acceptor-Hierarchie.

Dieses Verfahren wird auch *double dispatch* genannt. Welche Funktion letztendlich aufgerufen wird, ist sowohl vom konkreten Visitor-Objekt als auch vom konkreten Modell-Objekt abhängig und kann erst zur Laufzeit bestimmt werden.

Zusammenfassend bietet das Visitor-Pattern die Möglichkeit, sehr leicht

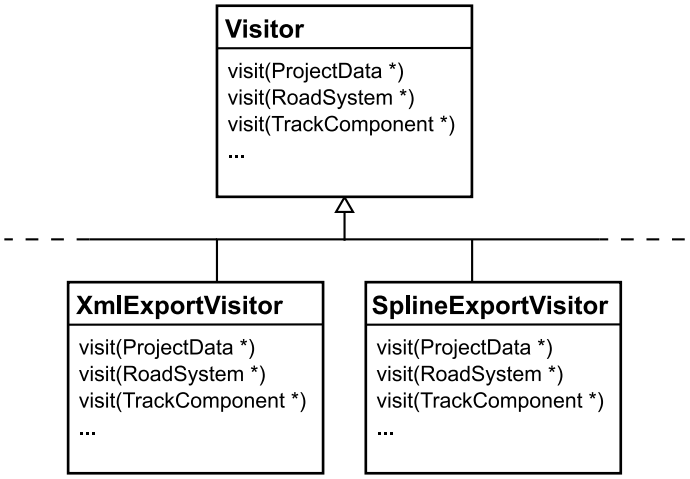


Abbildung 3.7: Klassendiagramm zum Visitor-Pattern: Visitor-Hierarchie.

den Modellklassen neue Funktionalität durch die Erstellung neuer Visitor-Subklassen hinzuzufügen, ohne jedoch die Klassen der Modellhierarchie selbst zu verändern, da der Code im Visitor und nicht in den Modellklassen abgelegt wird. Der Nachteil dabei ist, dass eine Änderung der Modellhierarchie mitunter mit erhöhtem Aufwand verbunden ist, da gegebenenfalls die Visitor-Subklassen angepasst werden müssen. Dies ist allerdings akzeptabel, da sich die Struktur des Modells in den meisten Fällen selten ändert.

Der Einsatz von Visitors ist in allen Schichten der Architektur sinnvoll. Umgesetzt wurden beispielsweise ein Visitor zum Exportieren der Daten in das OpenDRIVE Format und ein Visitor zum Exportieren von Punkten auf der Straßenachse zur Verwendung in einem 3D-Grafikprogramm. Ein weiterer Visitor untersucht eine neu geladene Szenerie auf Klothoide-Kreisbogen-Klothoide-Kombinationen und fasst diese zu `TrackSpiralArcSpiral`-Verbundkurven zusammen, wie sie in Kapitel 3.1.1 beschrieben wurden.

3.1.3 Command-Pattern

Das Command-Pattern (dt. *Kommando-Entwurfsmuster*) beschreibt die Kapselung eines Befehls durch ein Objekt [11]. Dies ermöglicht es, eine Undo/Redo-Funktion zu implementieren, die das Rückgängigmachen und

Wiederherstellen einzelner Arbeitsschritte erlaubt.

In Abbildung 3.8 ist die Basisklasse `QUndoCommand` dargestellt, wie sie von der Qt-Klassenbibliothek zur Verfügung gestellt wird. Jedes abgeleitete Command muss eine Funktion redo() und eine Funktion undo() definieren. Soll ein Command ausgeführt werden, so wird die Funktion redo() ausgeführt. Soll der Befehl rückgängig gemacht werden, wird die Funktion undo() aufgerufen. Das Command-Objekt muss dabei gegebenenfalls zusätzliche Daten speichern, um das Modell auf den vorherigen Zustand zurückbringen zu können. Dabei ist zu beachten, dass es sinnvoller ist, absolute Werte zu speichern, da es bei relativen Werten durch wiederholtes Aufrufen der Funktionen undo() und redo() ansonsten zu einem Abdriften kommen kann, insbesondere wenn dabei viele numerische Berechnungen durchgeführt werden, bei denen Rundungsfehler entstehen.

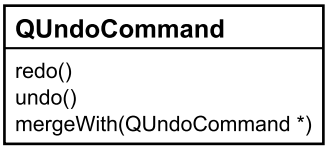


Abbildung 3.8: Die Basisklasse QUndoCommand der Qt-Klassenbibliothek.

Voraussetzung für die stabile Anwendung der Undo/Redo-Funktion ist zudem, dass das Modell ausschließlich über Commands geändert wird. Sollte ein Command beispielsweise einen Zeiger auf ein Objekt speichern, das anschließend manuell gelöscht wird, so kommt es beim (erneuten) Ausführen des Commands zu einem Programmabsturz.

Mehrere Commands lassen sich zu einem Macro-Command zusammenfügen. Wird die Funktion redo() des Macro-Commands aufgerufen, so ruft diese die redo() Funktionen der einzelnen Commands der Reihe nach auf. Beim Aufruf der Funktion undo() geschieht dies ebenso, jedoch in umgekehrter Reihenfolge.

Um mehr als einen Arbeitsschritt rückgängig machen zu können, muss ein Stapelspeicher (engl. *stack*) angelegt werden, wie er in Abbildung 3.9 zu sehen ist. Die Qt-Bibliothek bietet dazu die `QUndoStack`-Klasse an (Abbildung 3.10). Mit den Funktionen undo() und redo() kann man die aktuelle Position im Stapel verändern. Dabei führt die Funktion undo() des Stapels entsprechend die Funktion undo() des Commands der aktuellen Position aus und

verringert den Index um eins. Mit redo() lässt sich der Index um eins erhöhen, dabei wird die Funktion redo() des Commands über der aktuellen Position ausgeführt. Die Funktion setIndex(int i) erlaubt es, auf eine bestimmte Position i zu springen, dazu müssen aber alle Commands zwischen der aktuellen Position und der Zielposition der Reihe nach ausgeführt werden.

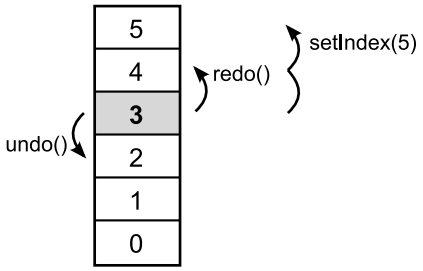


Abbildung 3.9: Ein Stapelspeicher mit sechs Commands.

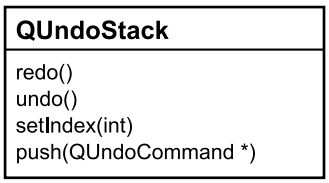


Abbildung 3.10: Die Klasse QUndoStack der Qt-Klassenbibliothek.

Neue Commands werden wie bei Stapelspeichern üblich von oben auf den Stapel gelegt. Dabei ist zu beachten, dass zuerst sämtliche Commands über der aktuellen Position gelöscht werden müssen. In Abbildung 3.11 ist dazu ein Beispiel aufgeführt. Begonnen wird mit einem Stapel aus sechs Commands (3.11a), auf dem ein neues Command abgelegt werden soll. Der aktuelle Index ist $i = 3$. Da sich zwei Commands über der aktuellen Position befinden, müssen diese entfernt werden (3.11b). Anschließend kann das neue Command hinzugefügt und mit redo() ausgeführt werden (3.11c). Der neue Index des Stapels ist nun $i = 4$ (3.11d).

Wird ein Command auf dem Stapel abgelegt, so lässt es sich auch mit dem vorangehenden Command zusammenfassen. Hierfür wird die Funktion mergeWith(QUndoCommand *) des alten Commands mit dem neuen Command als Argument aufgerufen. In dieser Funktion kann nun überprüfen wer-

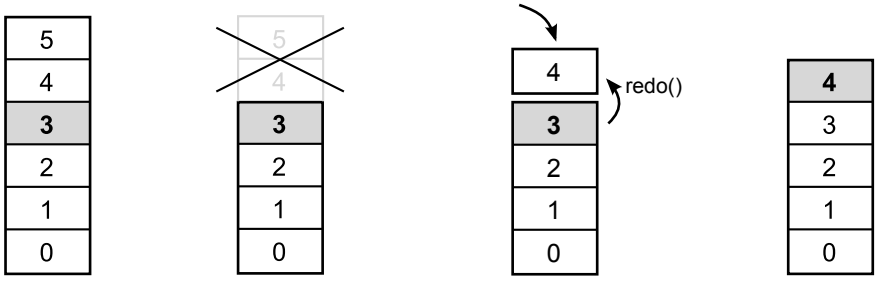


Abbildung 3.11: Hinzufügen eines Commands auf den Stapelspeicher.

den, ob beide Commands vom gleichen Typ sind und wenn dies der Fall ist, ob ein Zusammenfassen der Commands sinnvoll ist. Hat man beispielsweise ein Command, das ein Objekt um zehn Einheiten nach rechts verschiebt und fügt ein neues Command hinzu, welches das selbe Objekt um weitere fünf Einheiten verschiebt, kann man die Commands so zu einem einzelnen Command verschmelzen, welches das Objekt um 15 Einheiten verschiebt.

3.1.4 Observer-Pattern

In Kapitel 2.2 wurde verdeutlicht, dass es sinnvoll ist, die Klassen des Datenmodells von den anderen Klassen des Programms zu isolieren, um es flexibel zu halten. Dennoch muss es für das Modell möglich sein, die Klassen der Präsentation und der Steuerung über eventuelle Änderungen im Modell zu informieren.

Hierfür kann das Observer-Pattern (dt. *Beobachter-Entwurfsmuster*), auch bekannt als Publish-Subscribe-Pattern, verwendet werden [11]. Dazu werden die Klassen des Modells von der Klasse `Subject` abgeleitet, wie in Abbildung 3.12 dargestellt. Das Subject bietet die Funktionen attach(Observer *) und detach(Observer *) an, mit denen sich ein Observer an- beziehungsweise abmelden kann, der an Änderungen des Subjects interessiert ist. Das Subject kennt dabei nur das abstrakte Interface der Basisklasse `Observer`, weiß also nicht, von welchem konkreten Typ der Observer ist. Somit ist es nach wie vor von Änderungen der `Observer`-Subklassen außerhalb des Modells unabhängig.

In der einfachsten Variante des Observer-Pattern informiert ein geändertes

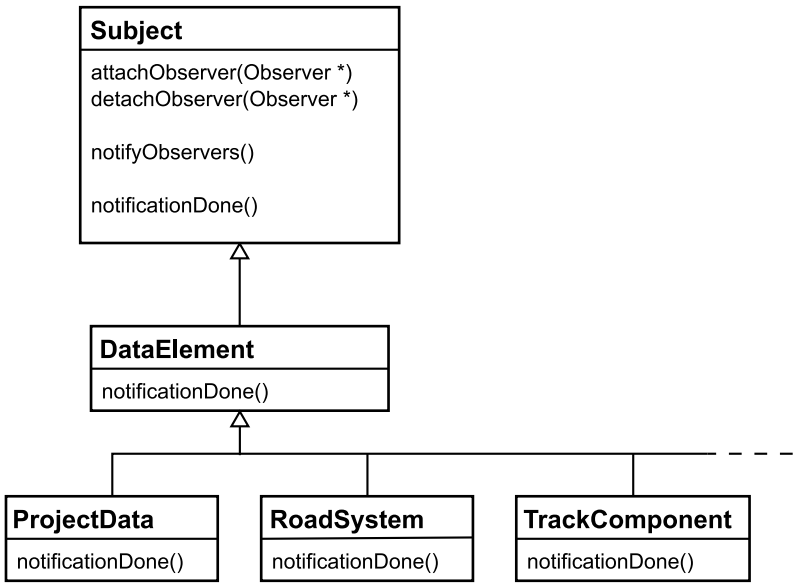


Abbildung 3.12: Klassendiagramm zum Observer-Pattern: Subject-Hierarchie.

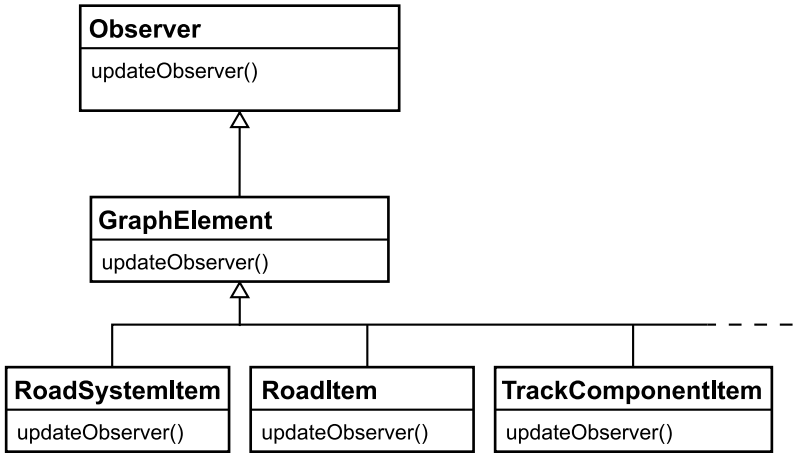


Abbildung 3.13: Klassendiagramm zum Observer-Pattern: Observer-Hierarchie.

Subject direkt die bei ihm registrierten Observer, indem es die virtuelle Funktion updateObserver() der `Observer`-Basisklasse aufruft. Die konkreten Subklassen können dann auf die Änderungen der Modellklassen reagieren. Der direkte Aufruf ist jedoch in vielen Fällen nicht zufriedenstellend, da so Observer mitunter mehrfach hintereinander benachrichtigt werden, wenn sich bei ihrem Subject mehrere Werte ändern, oder der Observer mehrere Subjects beobachtet, die sich ändern.

Sinnvoller ist es in solchen Fällen, zunächst sämtliche Änderungen des Modells vorzunehmen und anschließend all diejenigen Observer einmalig zu benachrichtigen, deren Subjects sich geändert haben. Dazu wurde die in Abbildung 3.14 dargestellte Klasse `ChangeManager` eingeführt, die den gesamten Benachrichtigungsvorgang steuert.

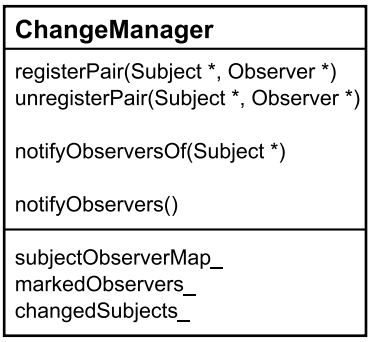


Abbildung 3.14: Die Klasse ChangeManager.

Abbildung 3.15 zeigt in einem Sequenzdiagramm beispielhaft die Interaktionen eines möglichen Ablaufs.

1. Das Objekt observerA meldet sich bei subjectA und subjectB an, worauf die Subjects die Observer-Subject-Verknüpfung dem ChangeManager mitteilen, der diese in der Map[1] `subjectObserverMap_` speichert.

2. Wird nun ein Subject geändert, informiert es den ChangeManager darüber, der darauf mithilfe der `subjectObserverMap_` ermittelt, welche Observer benachrichtigt werden müssen. Diese werden in der Liste `markedObservers_` zwischengespeichert. Außerdem wird das geänderte Subject in der Liste `changedSubjects_` eingetragen.

[1] Eine Map ist ein assoziatives Array, also ein Datenfeld mit Schlüssel-Datenwert Paaren.

3. Wurden alle Änderungen am Modell vorgenommen, wird die Funktion notifyObservers() des ChangeManager aufgerufen. Diese setzt nun die Observer der Liste `markedObservers_` davon in Kenntnis, dass mindestens eines ihrer Subjects geändert wurde. Anschließend wird den Subjects der Liste `changedSubjects_` mitgeteilt, dass der Benachrichtigungsvorgang abgeschlossen wurde. Zuletzt werden schließlich die beiden Listen geleert.

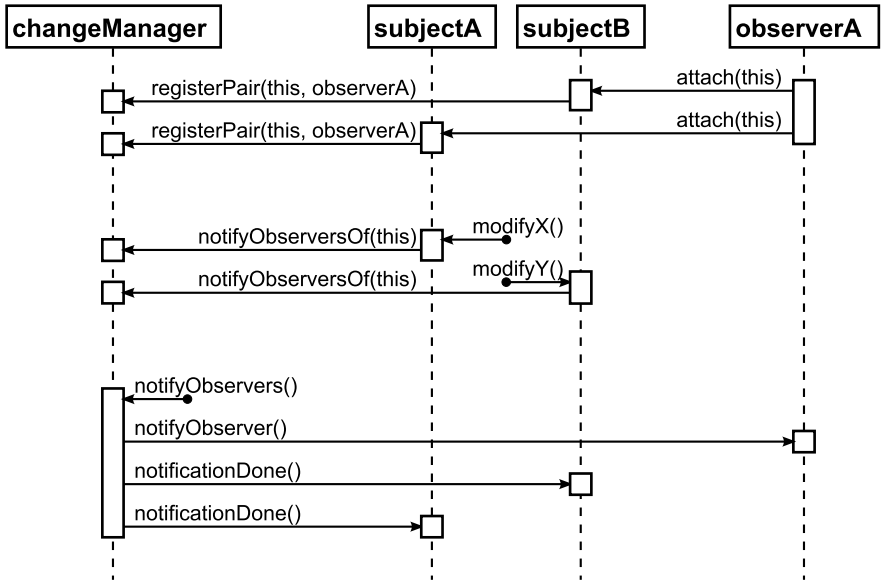


Abbildung 3.15: Sequenzdiagramm zum Observer-Pattern.

Der Aufwand für die Aktualisierung der Observer lässt sich oft deutlich reduzieren, indem den Observern mitgeteilt wird, welcher Aspekt des beobachteten Subjects sich ändert. Soll sich beispielsweise nur die Farbe eines Straßensegments ändern, wäre es überflüssig auch die komplette Geometrie neu zu berechnen. Generell unterscheidet man zwischen einem *push model* und einem *pull model*, wobei es auch Mittelwege gibt. Beim push model werden dem Observer sehr umfangreiche Informationen über die Änderungen geliefert, während beim reinen pull model dem Observer nur mitgeteilt wird, dass sich irgendetwas geändert hat. Das push model ist auf den ersten Blick effizienter, da der Observer nicht erst herausfinden muss, was sich wirklich geändert hat. Allerdings braucht nicht jeder Observer immer alle Informationen. Außerdem ist diese Variante oft unflexibler gegenüber Veränderungen.

Stattdessen wurde ein Kompromiss umgesetzt, bei dem ein Subject je nach Änderung bestimmte Flags setzt, die vom Observer bei Bedarf abgefragt werden können. Die Flags werden bei jeder Änderung vom Subject selbst gesetzt. Sobald der ChangeManager nach Abschluss des Benachrichtigungsvorgangs die notificationDone() Funktionen der geänderten Subjects aufruft, werden die Flags wieder auf Null zurückgesetzt.

Die Klasse `DataElement` hat etwa die Flags `CDE_DataElementCreated` und `CDE_DataElementDeleted`, um zu signalisieren, dass das DataElement-Objekt neu erstellt oder gelöscht wurde. Die Flags `CDE_DataElementAdded` und `CDE_DataElementRemoved` werden gesetzt, wenn das Objekt einem Elternobjekt zugeordnet wurde (beispielsweise ein Straßensegment einer Straße) beziehungsweise die Zuordnung aufgehoben wurde. Das Setzen des fünften Flags `CDE_SelectionChange` bedeutet schließlich, dass das Objekt selektiert oder deselektiert wurde.

Der Aktualisierungsvorgang auf Seiten der Präsentation und der Steuerung wird in Kapitel 3.2.1 betrachtet.

3.2 Präsentation (view) und Steuerung (controller)

Abbildung 3.16 zeigt nocheinmal das Klassendiagramm eines geöffneten Dokuments. Das `ProjectWidget`-Objekt enthält dabei alle zum Dokument gehörenden Komponenten. Zum einen ist dies das Modell (`ProjectData`), das im vorherigen Kapitel genauer betrachtet wurde. Dazu kommen die beiden Präsentationen `ProjectGraph` und `ProfileGraph`, die eine Draufsicht und eine Profilansicht beinhalten sowie die Steuerung in Form von `ProjectEditor`-Subklassen.

Für die Präsentationen wird der Qt Graphics View Framework verwendet. Mit dem Framework lässt sich eine große Anzahl an 2D-Elementen (engl. *items*) visualisieren. Diese Items können geometrische Primitive wie Linien, Rechtecke und Kreise sein, aber auch geladene Bilddateien oder frei definierbare Polygonzüge und Pfade.

Der Framework verwendet dafür eine Model-View Architektur (dt. *Modell-Präsentation*). Die Klasse `QGraphicsScene` dient dabei als Behälter für sämtliche Items des Modells und stellt Funktionen zum Verwalten der Items zur Verfügung. Mithilfe einer `QGraphicsView` können die Items einer Szene

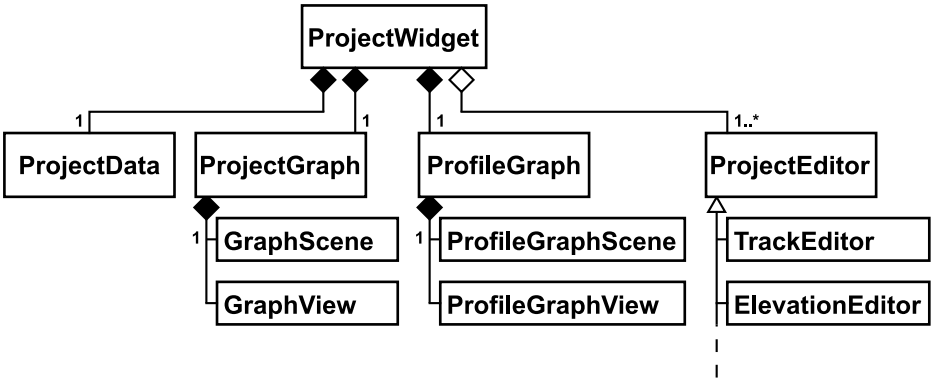


Abbildung 3.16: Klassendiagramm mit den wichtigsten Klassen eines Projektes.

schließlich dargestellt sowie Maus- und Tastatur-Events entgegengenommen und an die Szene und die Items weitergegeben werden.

Erzeugt werden die Items von Subklassen des `ProjectEditor`. Sechs Editoren wurden bisher implementiert, unter Anderem der `TrackEditor`, mit dem die Lage der Straßenachse definiert wird und der `ElevationEditor`, mit dem Höhenprofile erstellt werden.

Die Hierarchie der Items spiegelt die Hierarchie des Modells wider. Analog zu den Klassen `RoadSystem` und `Road` existieren beispielsweise die Items `RoadSystemItem` und `RoadItem`. Diese Items erfüllen Aufgaben, die für alle Editoren benötigt werden. Für einige Editoren wurde jedoch ein eigener Satz an Items erstellt, die von den Standard-Items abgeleitet werden. Dies ermöglicht es, weitere Funktionalitäten wie Kontextmenüs hinzuzufügen, die nur bei manchen Editoren erforderlich sind. Abbildung 3.17 zeigt eine Auswahl an Items, die vom ElevationEditor beziehungsweise vom TrackEditor verwendet werden.

Analog zur Klasse `DataElement` des Modells haben auch die Items der Präsentation eine gemeinsame Basisklasse `GraphElement`, wie in Abbildung 3.18 zu sehen ist. Diese Klasse wiederum besitzt die drei Basisklassen `Observer`, `QGraphicsPathItem` und `QObject`. Einem Objekt vom Typ `QGraphicsPathItem` kann man einen Pfad zuweisen, der den Umriss eines Items definiert. Außerdem kann man die Farbe und ein Füllmuster für das Item bestimmen sowie Funktionen definieren, die auf Maus- und Tastatur-Events reagieren. Die Items lassen sich dann der Szene hinzufügen. Durch die

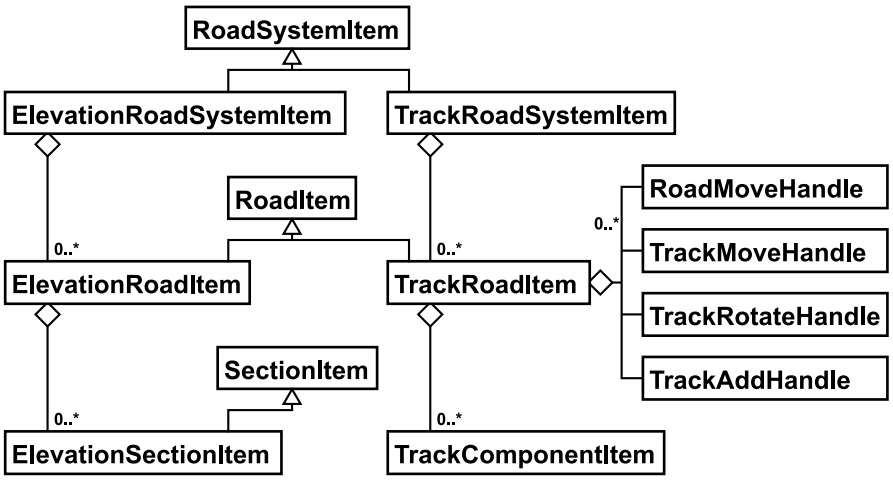


Abbildung 3.17: Klassendiagramm mit Items der Präsentation, die vom ElevationEditor und TrackEditor verwendet werden.

Ableitung von der Klasse `QObject` kann der *signals and slots* Mechanismus von den Items verwendet werden. Zuletzt werden die Items von der Klasse `Observer` abgeleitet, um auf Änderungen des Modells reagieren zu können. In Kapitel 3.1.4 wurde das Observer-Pattern auf Seiten des Modells bereits vorgestellt. Im nächsten Abschnitt wird im Hinblick auf die Präsentation weiter darauf eingegangen.

3.2.1 Chain-of-Responsibility-Pattern

Die Chain-of-Responsibility (dt. *Zuständigkeitskette*) wurde entwickelt, um ein Objekt, das eine Anfrage sendet, von demjenigen Objekt zu entkoppeln, das die Anfrage entgegen nimmt. Dazu wird den empfangenden Objekten die Möglichkeit gegeben, eine Anfrage an ein anderes Objekt weiter zu geben, wenn sie die Anfrage nicht bearbeiten können. Es entsteht eine Kette an Objekten, entlang derer Anfragen weiter gereicht werden. Das sendende Objekt muss also nicht das konkrete Objekt kennen, das die Anfrage bearbeitet, sondern nur eine Basisklasse, welche die Verkettung ermöglicht. Diese Basisklasse muss eine Schnittstelle bieten, mit der eine Anfrage an die Kette gegeben werden kann. Von dort aus wird die Anfrage entlang der Kette weiter gereicht, bis die Kette zu Ende ist oder die Anfrage vollständig bearbeitet wurde.

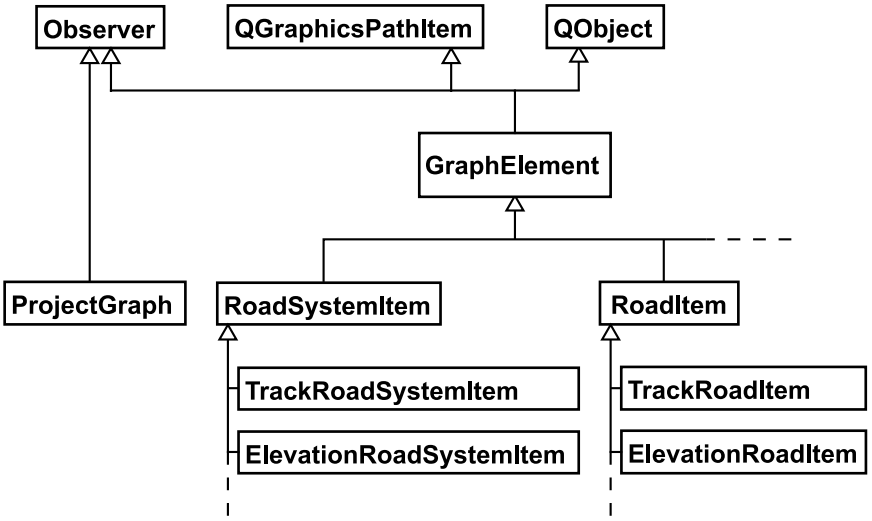


Abbildung 3.18: Vererbungshierarchie der Klassen der Präsentation.

Dieser Mechanismus wurde verwendet, um den Aktualisierungsvorgang der Präsentation zu gestalten. Die erforderliche Basisklasse ist die in Kapitel 3.1.4 vorgestellte Klasse `Observer`. Eintrittspunkt in die Kette ist die virtuelle Funktion updateObserver(). Strenggenommen entspricht diese Umsetzung nicht dem ursprünglichen Entwurfsmuster, da dieses eine Kommunikation zwischen verschiedenen Objekten beschreibt, hier jedoch (bisher) ausschließlich virtuelle Funktionen verwendet werden. Das zugrundeliegende Prinzip des Weiterreichens einer Anfrage und die damit einhergehenden Vorteile sind jedoch dieselben.

Abbildung 3.19 zeigt ein Beispiel eines Aktualisierungsvorgangs. Ein Item vom Typ `TrackElementItem` wird über seine Basisklasse `Observer` über eine Änderung im Modell benachrichtigt. Hat sich etwas an der Form, Länge oder Startkoordinate des visualisierten `TrackElement` geändert, so wird der Pfad des Items neu erstellt. Danach wird die Information über die Änderung an die Basisklasse `TrackComponentItem` weiter gereicht. Diese prüft, ob sich die Position oder Drehung des Segments geändert hat und aktualisiert gegebenenfalls die Transformation des Items. Zuletzt ist das `GraphElement` an der Reihe, das überprüft, ob das Element selektiert wurde, was eine optische Hervorhebung des Items bewirken soll oder ob das Element entfernt wurde, worauf es veranlasst, dass das Item ebenfalls gelöscht wird.

Es kann manchmal auch von Vorteil sein, die Kette in anderer Richtung zu

durchlaufen, beispielsweise um gleich zu Beginn überprüfen zu lassen, ob das Element gelöscht wurde und nur wenn dies nicht der Fall ist, weitere Aktualisierungen vorzunehmen. Ansonsten lässt sich die Kette auch abbrechen.

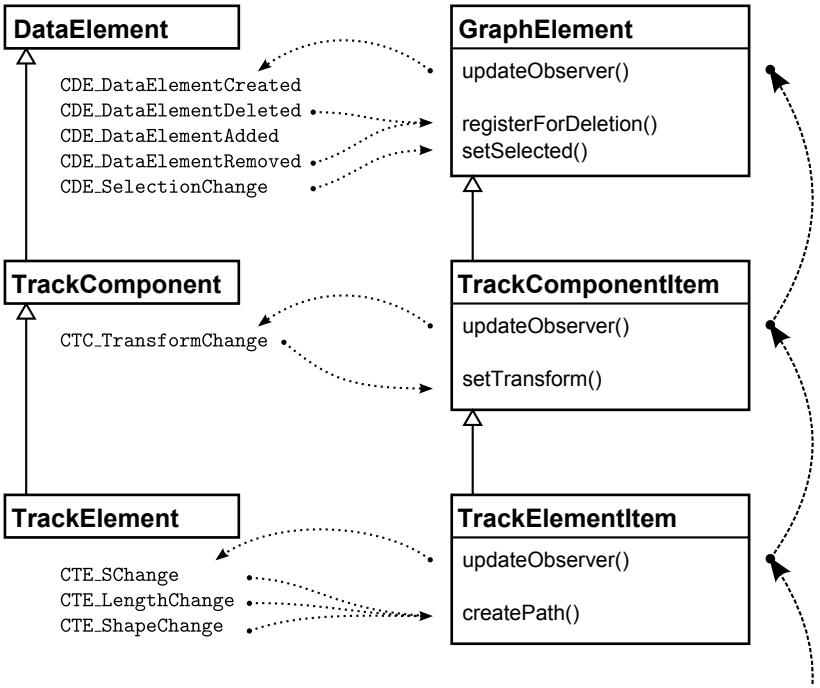


Abbildung 3.19: Beispiel eines Aktualisierungsdurchlaufs von Items der Präsentation.

Kapitel 4

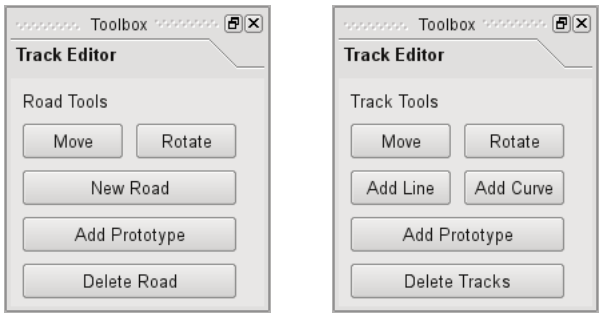
Implementierung

In diesem Kapitel werden Details zu einigen ausgewählten Funktionen der einzelnen Editoren vorgestellt. Dabei sollen jedoch mehr die mathematischen und logischen Hintergründe erläutert und weniger eine Bedienungsanleitung gegeben werden. Begonnen wird in Kapitel 4.1 mit dem TrackEditor, mit einleitenden Worten zu den Koordinatensystemen von OpenDRIVE und Qt sowie mit der Beschreibung der Geometrieelemente. Anschließend wird in Kapitel 4.2 der RoadTypeEditor vorgestellt. Hierbei liegt die numerische Berechnung von Straßenkoordinaten aus globalen Koordinaten im Vordergrund. Die Erstellung des Höhenplans und die Definition der Querneigung werden in Kapitel 4.3 betrachtet. Abschließend wird in Kapitel 4.4 beschrieben, wie Satellitenbilder und Straßenkarten eingeblendet werden können, um reale Straßenzüge nachzubilden.

4.1 TrackEditor

Der TrackEditor ist der umfangreichste der Editoren. Mit ihm wird der Verlauf der Straßenachse festgelegt, wozu Geradenstücke und Verbundkurven verwendet werden. Verbundkurven setzen sich aus den Grundelementen Klothoide-Kreisbogen-Klothoide zusammen und werden in Kapitel 4.1.5 ausführlich betrachtet.

Eine Übersicht der vorhandenen Werkzeuge ist in Abbildung 4.1 dargestellt. Dabei sind die Werkzeuge in die Bereiche **Road Tools** und **Track Tools** aufgeteilt. Mit den Road Tools können neue Straßen erstellt werden (**New Road**). Bestehende Straßen können als Ganzes verschoben (**Move**), ge-



(a) Road Tools (b) Track Tools

Abbildung 4.1: Werkzeugleiste des TrackEditor.

dreht (**Rotate**) und gelöscht werden (**Delete Road**). Desweiteren lassen sich komplette Straßenzüge, beispielsweise Kreuzungen, als Prototyp importieren (**Add Prototype**) und so einfach wiederverwenden.

Sehr ähnlich funktionieren die Track Tools. Mit diesen lassen sich an bestehende Straßen neue Elemente hinzufügen und bestehende Elemente verschieben, drehen oder löschen. Eine Besonderheit ist dabei das **Move**-Werkzeug, das in Abbildung 4.2 dargestellt ist. Zwischen den Geometrieelementen werden quadratische *Move Handles* eingeblendet. Grüne Handles begrenzen Verbundkurven und können frei verschoben werden. Gelbe Handles begrenzen Geradenstücke und besitzen nur einen Freiheitsgrad. Das heißt, sie können nur entlang der Geraden verschoben werden. Kreisbögen und Klothoiden werden durch rote Handles begrenzt, die nicht verschiebbar sind, weshalb stattdessen ausschließlich Verbundkurven mit zwei Freiheitsgraden und Geraden mit einem Freiheitsgrad verwendet werden sollten. Werden mehrere Handles gleichzeitig markiert, wird geprüft, ob sich die Handles in ein oder zwei Richtungen verschieben lassen und die Bewegung gegebenenfalls eingeschränkt.

Wird eines der drei Werkzeuge zum Hinzufügen von Elementen aktiviert, wird an den Straßenenden jeweils ein *Add Handle* eingeblendet. Wird dieser markiert, lassen sich nun je nach Werkzeug Geradenstücke (**Add Line**), Verbundkurven (**Add Curve**) oder Prototypen (**Add Prototype**) anfügen. Der Arbeitsprozess wird dadurch erleichtert, dass bei diesen drei Werkzeugen bereits Prototypen der anderen Straßensegmenttypen mit an die Straße angefügt werden. Dies kann beispielsweise der Querschnittsprototyp *RQ 10,5* sein, der Fahrspuren und Fahrbahnmarkierungen einer 10,5 Meter breiten Außerortsstraße repräsentiert und zweckmäßig dazu ein *rural*-Prototyp für die Straßenart.

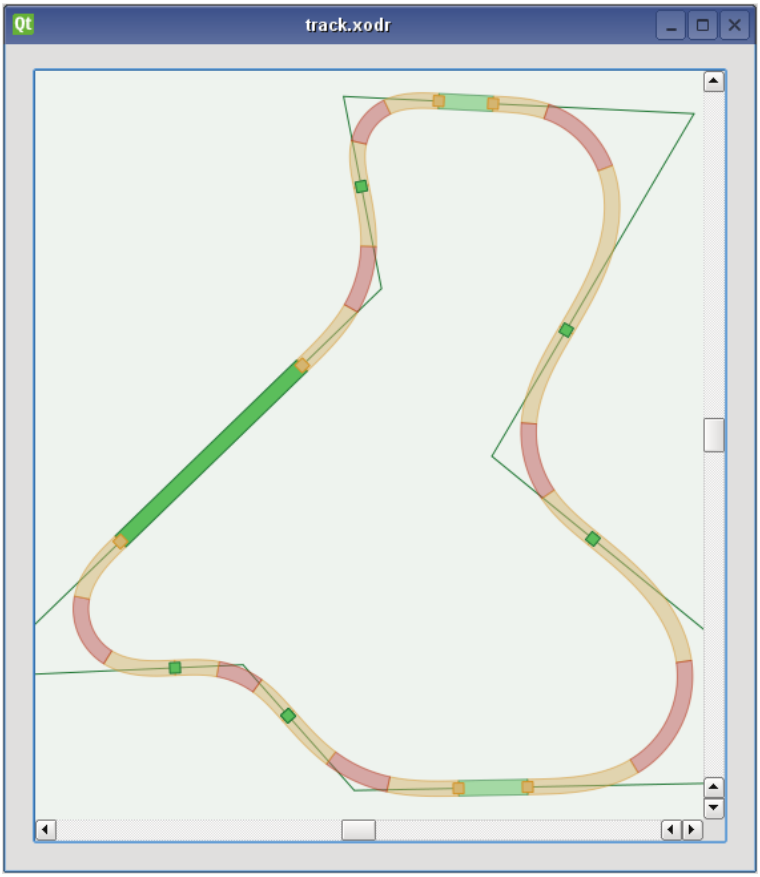
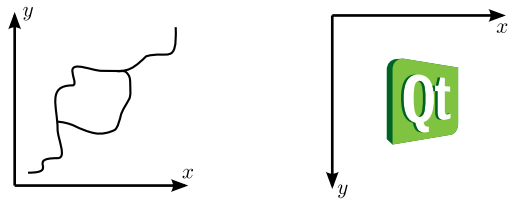


Abbildung 4.2: TrackEditor mit erstelltem Rundkurs.

4.1.1 Koordinatensystem

Der Betrachter einer Landkarte erwartet in der heutigen Zeit[1], dass Norden zum oberen Kartenrand zeigt und Osten nach rechts. Wie in Abbildung 4.3a dargestellt, ist das Koordinatensystem von OpenDRIVE so aufgebaut, dass die x-Achse in Richtung Osten zeigt und die y-Achse in Richtung Norden. Da das Koordinatensystem des Qt Graphics View Framework die y-Achse jedoch in die entgegengesetzte Richtung definiert, müssen die Koordinaten der OpenDRIVE Daten transformiert werden, bevor sie am Bildschirm angezeigt werden. Dazu muss das `GraphView`-Objekt 180° um die x-Achse transformiert werden. Die eingelesenen OpenDRIVE Daten im Modell und in der `GraphScene` bleiben davon unberührt. Lediglich die Ansicht wird transformiert.



(a) OpenDRIVE (b) Qt

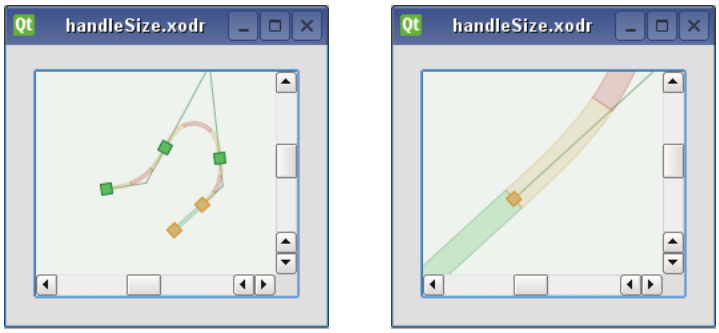
Abbildung 4.3: Koordinatensysteme von OpenDRIVE und Qt.

Die Transformation der Ansicht zieht jedoch auch Komplikationen nach sich, da Handles (dt. *Ziehpunkte*), mit denen beispielsweise Straßenstücke verschoben oder gedreht werden können, genau diese Transformation ignorieren, um unabhängig vom Zoomfaktor stets die gleiche Größe zu haben (Abbildung 4.4). Alle Handles besitzen deshalb die gemeinsame Basisklasse `Handle`, welche die ignorierte Transformation der Ansicht durch eine eigene Transformation nachbildet.

Für die Betrachtung der Geometrieelemente sind im wesentlichen drei Koordinatensysteme relevant. Die **natürlichen Koordinaten**[2] beschreiben dabei die Grundelemente Gerade, Kreisbogen und Klothoide in ihrem jeweils eigenen Koordinatensystem. Die **lokalen Koordinaten** beschreiben ein Geome-

[1] Früher war es durchaus üblich, Karten in Richtung Sonnenaufgang, also Osten (lat. *oriens*) auszurichten - daher auch das Wort Orientierung.

[2] Der Begriff der natürlichen Koordinaten ist an die Verwendung im Rahmen der *Finite Elemente Methode* angelehnt und beschreibt ein Koordinatensystem, in dem ein Element möglichst einfach dargestellt werden kann.



(a) (b)

Abbildung 4.4: Handles haben stets die gleiche Größe.

trieelement relativ zum jeweils übergeordneten Element. Dies kann beispielsweise eine Verbundkurve sein (Kapitel 4.1.5). Die **globalen Koordinaten** schließlich entsprechen den absoluten Koordinaten, sie sind also relativ zum Inertialsystem. In dieser Form werden die Geometrieelemente auf dem Bildschirm dargestellt oder ins OpenDRIVE Format exportiert.

In den folgenden Abschnitten wird die mathematische Beschreibung der Geometrieelemente in den natürlichen und lokalen Koordinatensystemen dargestellt.

4.1.2 Berechnung von Geradenstücken

In Abbildung 4.5 ist eine Gerade in ihrem **natürlichen Koordinatensystem** dargestellt. Der Ortsvektor

$$\mathbf{r}(s,d) = \begin{bmatrix} x(s) \\ y(d) \end{bmatrix} = \begin{bmatrix} s \\ d \end{bmatrix} \tag{4.1}$$

eines beliebigen Punktes der Straße ist dabei von der Straßenkoordinate s und dem Abstand d zur Straßenachse abhängig. Es sei angemerkt, dass sich die Straßenkoordinate s hier auf das aktuelle Segment bezieht, also dem Abstand zu dessen Startkoordinate entspricht.

Der Tangentialwinkel des Geradenstücks

$$\tau = 0.0 \tag{4.2}$$

ist ebenso wie die Krümmung

$$\kappa = 0.0 \tag{4.3}$$

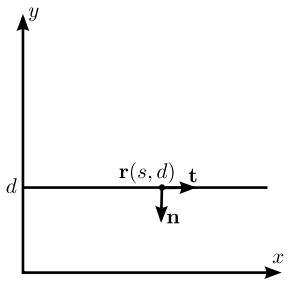


Abbildung 4.5: Natürliches Koordinatensystem eines Geradenstücks.

konstant und gleich Null. Auch der Tangentenvektor

$$\mathbf{t} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4.4}$$

und der Normalenvektor

$$\mathbf{n} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{4.5}$$

sind konstant.

Wird das Element relativ zum übergeordneten Element wie in Abbildung 4.6 dargestellt, um einen Vektor $\mathbf{r_0}$ verschoben und um einen Winkel τ_0 gedreht, so wird dies mit dem **lokalen Koordinatensystem** ausgedrückt. Die Drehung lässt sich dabei mit der Matrix

$$\mathbf{S_0} = \begin{bmatrix} \cos\tau_0 & -\sin\tau_0 \\ \sin\tau_0 & \cos\tau_0 \end{bmatrix} \tag{4.6}$$

darstellen.

Damit erhält man den Ortsvektor

$$\mathbf{r_{loc}}(s,d) = \mathbf{r_0} + \mathbf{S_0} \cdot \mathbf{r}(s,d) = \mathbf{r_0} + \begin{bmatrix} s\ \cos\tau_0 - d\ \sin\tau_0 \\ s\ \sin\tau_0 + d\ \cos\tau_0 \end{bmatrix} \tag{4.7}$$

im lokalen Koordinatensystem. Analog dazu können Tangentenvektor

$$\mathbf{t_{loc}} = \mathbf{S_0} \cdot \mathbf{t} = \begin{bmatrix} \cos\tau_0 \\ \sin\tau_0 \end{bmatrix} \tag{4.8}$$

und Normalenvektor

$$\mathbf{n_{loc}} = \mathbf{S_0} \cdot \mathbf{n} = \begin{bmatrix} \sin\tau_0 \\ -\cos\tau_0 \end{bmatrix} \tag{4.9}$$

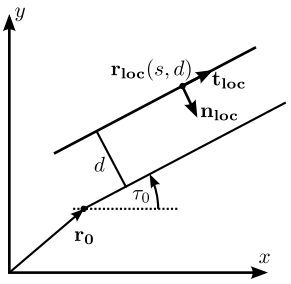


Abbildung 4.6: Lokales Koordinatensystem eines Geradenstücks.

aufgestellt werden. Der Tangentialwinkel

$$\tau_{loc} = \tau_0 \tag{4.10}$$

entspricht im lokalen Koordinatensystem dem Drehwinkel.

4.1.3 Berechnung von Kreisbögen

In Abbildung 4.7 ist ein Kreisbogen mit dem Radius r in seinem **natürlichen Koordinatensystem** dargestellt. Bei Linkskurven ist der Radius r positiv, bei Rechtskurven werden negative Radien verwendet. Der reziproke Wert des Radius ist die Krümmung

$$\kappa = \frac{1}{r}, \tag{4.11}$$

die bei Kreisbögen konstant ist.

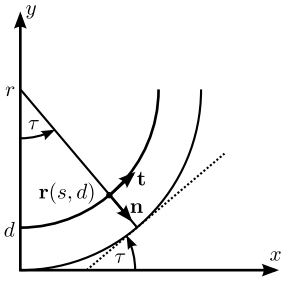


Abbildung 4.7: Natürliches Koordinatensystem eines Kreisbogens.

Mithilfe trigonometrischer Funktionen lässt sich nun der Ortsvektor

$$\mathbf{r}(s,d) = \begin{bmatrix} x(s,d) \\ y(s,d) \end{bmatrix} = \begin{bmatrix} (r-d)\,\sin\frac{s}{r} \\ r-(r-d)\,\cos\frac{s}{r} \end{bmatrix} \tag{4.12}$$

wieder in Abhängigkeit von der Straßenkoordinate s und dem Abstand d zur Straßenachse aufstellen. Der Tangentialwinkel

$$\tau(s) = \frac{s}{r} \tag{4.13}$$

berechnet sich über das Bogenmaß. Damit lassen sich nun der Tangentenvektor

$$\mathbf{t}(s) = \begin{bmatrix} \cos\tau(s) \\ \sin\tau(s) \end{bmatrix} \tag{4.14}$$

und der Normalenvektor

$$\mathbf{n}(s) = \mathrm{sign}(\kappa) \begin{bmatrix} \sin\tau(s) \\ -\cos\tau(s) \end{bmatrix} \tag{4.15}$$

bestimmen. Die Richtung des Normalenvektors ist dabei vom Vorzeichen der Krümmung abhängig. Es sei hier definiert, dass der Normalenvektor einer Kurve stets nach außen zeigt.

Analog zu den Geradenstücken entsteht das **lokale Koordinatensystem** wieder aus einer Verschiebung um den Vektor $\mathbf{r_0}$ und einer Verdrehung um den Winkel τ_0 (Abbildung 4.8).

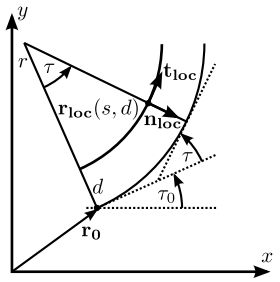


Abbildung 4.8: Lokales Koordinatensystem eines Kreisbogens.

So erhält man den Ortsvektor

$$\mathbf{r_{loc}}(s,d) = \mathbf{r_0} + \mathbf{S_0} \cdot \mathbf{r}(s,d) \tag{4.16}$$

und den Tangentialwinkel

$$\tau_{loc}(s) = \tau_0 + \tau(s). \tag{4.17}$$

Abschließend können wieder Formeln für den Tangentenvektor

$$\mathbf{t_{loc}}(s) = \mathbf{S_0} \cdot \mathbf{t}(s) = \begin{bmatrix} \cos \tau_{loc}(s) \\ \sin \tau_{loc}(s) \end{bmatrix} \tag{4.18}$$

und den Normalenvektor

$$\mathbf{n_{loc}}(s) = \mathbf{S_0} \cdot \mathbf{n}(s) = \text{sign}(\kappa) \begin{bmatrix} \sin \tau_{loc}(s) \\ -\cos \tau_{loc}(s) \end{bmatrix} \tag{4.19}$$

aufgestellt werden.

4.1.4 Berechnung von Klothoiden

Seit Mitte des 20. Jahrhunderts ist die Klothoide (engl. *Cornu Spiral* oder *Euler Spiral*) im Straßen- und Gleisbau der bevorzugte Übergangsbogen zwischen Geraden und Kreisbögen. Die vorallem im englischen Sprachgebrauch gängige Bezeichnung Cornu-Spirale geht auf den französischen Physiker Marie Alfred Cornu (1841 - 1902) zurück, der die Kurve 1874 umfangreich untersuchte, um damit die Beugungserscheinungen des Lichts zu berechnen. In Abbildung 4.9 ist eine Klothoide in ihrem natürlichen Koordinatensystem dargestellt.

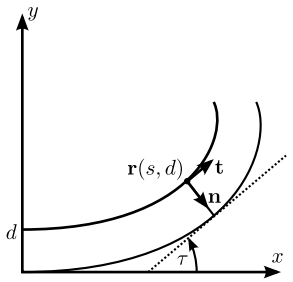


Abbildung 4.9: Natürliches Koordinatensystem einer Klothoide.

Klothoiden zeichnen sich dadurch aus, dass ihre Krümmung $\kappa(s)$ proportional mit der Bogenlänge s zunimmt. Der Proportionalitätsfaktor

$$A^2 = \frac{s}{\kappa(s)} \tag{4.20}$$

wird auch Klothoidenparameter genannt. Mit dem Radius

$$r(s) = \frac{1}{\kappa(s)} \tag{4.21}$$

an der Stelle s ergibt sich das Bildungsgesetz der Klothoide

$$A^2 = s\, r(s). \tag{4.22}$$

Ändert man den Klothoidenparameter A um einen gewissen Faktor, so ändern sich auch alle Längenwerte mit dem gleichen Faktor. Alle Winkel und die Verhältnisse $\frac{r(s)}{A}$ und $\frac{s}{A}$ bleiben jedoch an den Kennstellen konstant [10]. Diese geometrische Ähnlichkeit machte die Klothoiden einst für den Straßenbau interessant, da sich alle Klothoiden auf die Einheitsklothoide mit dem Parameter $A = 1$ zurückführen lassen.

Aus der Krümmung lässt sich durch Integration der Tangentialwinkel

$$\tau(s) = \int_0^s \kappa(\bar{s})\mathrm{d}\bar{s} = \frac{s^2}{2A^2} \tag{4.23}$$

berechnen. Will man nun durch erneute Integration die Koordinaten x und y eines Punktes an der Stelle s bestimmen, erhält man die Fresnelschen Integrale

$$x = \int_0^s \cos\tau(\bar{s})\mathrm{d}\bar{s} = \int_0^s \cos\frac{\bar{s}^2}{2A^2}\mathrm{d}\bar{s} \tag{4.24}$$

und

$$y = \int_0^s \sin\tau(\bar{s})\mathrm{d}\bar{s} = \int_0^s \sin\frac{\bar{s}^2}{2A^2}\mathrm{d}\bar{s}, \tag{4.25}$$

die nicht mehr analytisch lösbar sind.

Approximieren lassen sich die nach dem französischen Physiker Augustin Jean Fresnel (1788 - 1827) benannten Integrale allerdings durch die Potenzreihenentwicklungen der trigonometrischen Funktionen

$$\cos t^2 = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} t^{4n} \tag{4.26}$$

und

$$\sin t^2 = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} t^{4n+2}. \tag{4.27}$$

Durch die Substitution $\bar{s} = \sqrt{2}\,A\,\bar{t}$ werden die Gleichungen 4.24 und 4.25 zunächst auf die Form

$$x = \int_0^s \cos\frac{\bar{s}^2}{2A^2}\mathrm{d}\bar{s} = \sqrt{2}\,A\int_0^t \cos\bar{t}^2\mathrm{d}\bar{t} \tag{4.28}$$

$$y = \int_0^s \sin\frac{\bar{s}^2}{2A^2}\mathrm{d}\bar{s} = \sqrt{2}\,A\int_0^t \sin\bar{t}^2\mathrm{d}\bar{t} \tag{4.29}$$

gebracht, wobei $t = \frac{s}{\sqrt{2}\,A}$ die neue obere Integrationsgrenze ist. Nun lassen sich die Potenzreihen aus den Gleichungen 4.26 und 4.27 einsetzen. Nach der Integration erhält man durch Umformen die Gleichungen

$$x = A\sum_{n=0}^{\infty}\frac{(-1)^n}{4^n\,(4n+1)\,(2n)!}\left(\frac{s}{A}\right)^{4n+1}, \tag{4.30}$$

$$y = A\,\frac{1}{2}\sum_{n=0}^{\infty}\frac{(-1)^n}{4^n\,(4n+3)\,(2n+1)!}\left(\frac{s}{A}\right)^{4n+3}, \tag{4.31}$$

die schon mit wenigen Gliedern eine Genauigkeit im Bereich von Millimetern liefern. Mit dem Verhältnis

$$l = \frac{s}{A} = \sqrt{2\,\tau} \tag{4.32}$$

erhält man schlussendlich die Annäherungen

$$x \approx A\left(l - \frac{l^5}{40} + \frac{l^9}{3456} - \frac{l^{13}}{599040} + \frac{l^{17}}{175472640}\right), \tag{4.33}$$

$$y \approx A\left(\frac{l^3}{6} - \frac{l^7}{336} + \frac{l^{11}}{42240} - \frac{l^{15}}{9676800}\right), \tag{4.34}$$

wie sie auch in den *Richtlinien für die Anlage von Straßen - Teil: Linienführung (RAS-L)*[8] verwendet werden, beziehungsweise die später benötigte Kurzschreibweise

$$x \approx A\,\bar{x}(l), \tag{4.35}$$

$$y \approx A\,\bar{y}(l). \tag{4.36}$$

Im OpenDRIVE Format werden Klothoiden durch die Krümmungen an ihrem Anfang und Ende beschrieben. Mit Gleichung 4.20 kann daraus der Klothoidenparameter

$$A = \sqrt{\frac{s_{end} - s_{start}}{|\kappa_{end} - \kappa_{start}|}} \tag{4.37}$$

berechnet werden.

Positive Krümmungen beschreiben eine Linkskurve. Um eine Rechtskurven darzustellen muss das Vorzeichen der y-Koordinate vertauscht werden. Dies lässt sich mathematisch leicht in die oben genannten Formeln integrieren, indem man den Klothoidenparameter

$$A^2 = |a_x\, a_y|$$

in zwei Teile auftrennt, wobei für eine Linkskurve

$$\begin{aligned} a_x &= A \\ a_y &= A \end{aligned}$$

und für eine Rechtskurve

$$\begin{aligned} a_x &= A \\ a_y &= -A \end{aligned}$$

gilt.

Die endgültigen Formeln für die Berechnung der x- und y-Koordinate eines Punktes auf der Klothoide lauten demnach

$$x \approx a_x \left(l - \frac{l^5}{40} + \frac{l^9}{3456} - \frac{l^{13}}{599040} + \frac{l^{17}}{175472640} \right), \tag{4.38}$$

$$y \approx a_y \left(\frac{l^3}{6} - \frac{l^7}{336} + \frac{l^{11}}{42240} - \frac{l^{15}}{9676800} \right). \tag{4.39}$$

4.1.5 Berechnung von Verbundkurven

Ein wesentliches Ziel dieser Arbeit war es, dem Benutzer das Anlegen von Straßen so einfach wie möglich zu machen. Um dem Benutzer also das Abstimmen von Klothoidenparametern, Krümmungen und Kreisbogenlängen zu ersparen, wurde eine Verbundkurve aus einer Klothoide, einem Kreisbogen und einer weiteren Klothoide eingeführt. Die Verbundkurve repräsentiert eine gesamte Links- beziehungsweise Rechtskurve, startet und endet also mit der Krümmung $\kappa = 0$. Da sie nur als Ganzes bearbeitet werden kann, reduzieren sich die Freiheitsgrade auf die wesentlichen Eigenschaften einer Kurve. Dazu gehören die Start- und Endpunkte sowie die Richtungen in diesen und ein weiterer Faktor, der das Längenverhältnis von Kreisbogen zu Klothoiden beeinflusst.

Walton und Meek beschreiben in ihrer Veröffentlichung [12] Vorgehensweisen zur Berechnung von symmetrischen und asymmetrischen Klothoidenpaaren mit diesen Randbedingungen. Außerdem zeigen sie, wie zwischen die Klothoiden ein Kreisbogen eingefügt werden kann, wie es im Straßenbau üblich ist. Walton und Meek verwenden jedoch eine Beschreibung der Klothoiden, die vom Tangentialwinkel und nicht von der Bogenlänge abhängt. Im Folgenden werden deshalb die Formeln in der hier verwendeten Schreibweise hergeleitet.

Symmetrisches Klothoidenpaar

Abbildung 4.10 zeigt ein symmetrisches Klothoidenpaar zwischen den Punkten $\mathbf{P_0}$ und $\mathbf{P_1}$. Die Klothoiden treffen im Punkt $\mathbf{P}$ zusammen, ihre Tangenten schneiden sich im Punkt $\mathbf{V}$ mit dem Winkel α und besitzen die Länge g. Da die Klothoiden symmetrisch sind, wird nur die untere Klothoide betrachtet. Das Koordinatensystem der unteren Klothoide wird von der Tangente $\mathbf{T_0}$ und der Normalen $\mathbf{N_0}$ des Punktes $\mathbf{P_0}$ aufgespannt. In diesem Koordinatensystem gilt $\mathbf{P} = [x_p, y_p]^T$ und $\mathbf{V} = [g, 0]^T$.

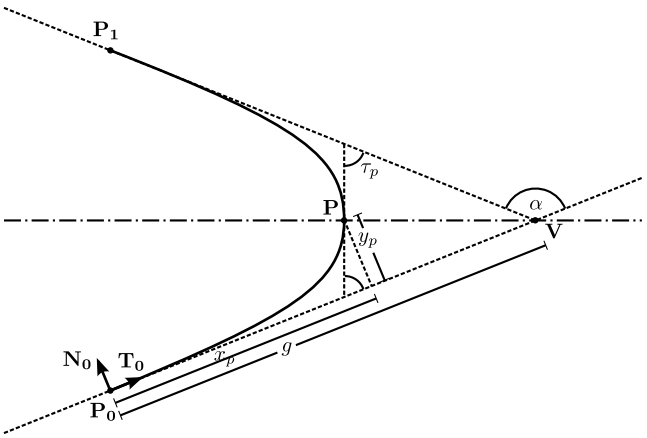


Abbildung 4.10: Symmetrisches Klothoidenpaar.

Um die relevanten Größen des Klothoidenpaars rechnerisch zu bestimmen, müssen zwei Bedingungen erfüllt werden. Zuerst wird die **Stetigkeit des Tangentialwinkels** im Punkt $\mathbf{P}$ betrachtet. Aus Symmetriegründen ist dann auch die **Krümmung** stetig. Für den Tangentialwinkel muss demnach

$$\tau_p = \frac{\alpha}{2} \tag{4.40}$$

gelten. Zudem muss die **Stetigkeit der Lage** im Punkt **P** gewährleistet sein. Dies ist der Fall wenn die Gleichung

$$g = x_p + y_p \tan\frac{\alpha}{2} \tag{4.41}$$

erfüllt ist. Durch Einsetzen von 4.35 und 4.36 erhält man

$$g = A\,\bar{x}_p(l_p) + A\,\bar{y}_p(l_p) \tan\frac{\alpha}{2}. \tag{4.42}$$

Aus den Gleichungen 4.32 und 4.40 ergibt sich

$$l_p = \sqrt{2\,\tau_p} = \sqrt{\alpha} \tag{4.43}$$

und man kann schließlich nach

$$A = \frac{g}{\bar{x}_p(\sqrt{\alpha}) + \bar{y}_p(\sqrt{\alpha}) \tan\frac{\alpha}{2}} \tag{4.44}$$

auflösen. Mit dem Klothoidenparameter A lassen sich nun sehr einfach alle weiteren benötigten Parameter bestimmen, wie beispielsweise s_p und κ_p.

Symmetrisches Klothoidenpaar mit Kreisbogen

Das vorherige Beispiel wird nun durch einen Kreisbogen erweitert, der sich zwischen den beiden Klothoiden befindet, wie in Abbildung 4.11 zu sehen ist. Der Mittelpunkt des Kreises liegt dabei im Punkt $\mathbf{M} = [g-\xi, \eta]^T$, der Radius des Kreises ist r. Punkt $\mathbf{P} = [x_p, y_p]^T$ befindet sich jetzt im Schnittpunkt von Klothoide und Kreisbogen, alle anderen Angaben sind die gleichen wie im Abschnitt zuvor.

Wieder muss die Stetigkeit der Krümmung, des Tangentialwinkels und der Lage gewährleistet werden, wobei auch hier nur die untere Klothoide betrachtet wird. Im Punkt **P** muss die Krümmung des Kreises

$$\kappa_{Kreis} = \kappa_p \tag{4.45}$$

mit der Krümmung der Klothoide übereinstimmen. Mit den Gleichungen 4.21, 4.22 und 4.23 lässt sich daraus eine Formel für den Klothoidenparameter

$$A = r\,\sqrt{2\,\tau_p} \tag{4.46}$$

in Abhängigkeit vom Kreisradius r und dem Tangentialwinkel τ_p herleiten.

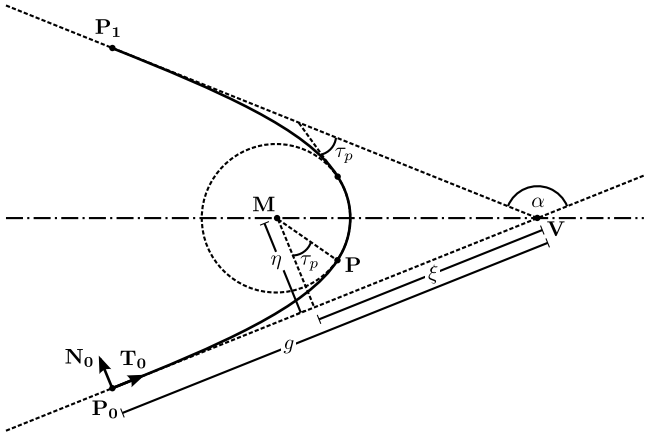


Abbildung 4.11: Symmetrisches Klothoidenpaar mit Kreisbogen.

Um die **Stetigkeit der Lage** im Punkt **P** zu garantieren müssen die Gleichungen

$$x_p = g - \xi + r \sin\tau \tag{4.47}$$
$$y_p = \eta - r \cos\tau \tag{4.48}$$

erfüllt sein. Zusammen mit der Bedingung

$$\frac{\eta}{\xi} = \cot\frac{\alpha}{2} \tag{4.49}$$

die besagt, dass **M** auf der Winkelhalbierenden liegt und den Gleichungen 4.35, 4.36 und 4.46 lässt sich daraus der Radius des Kreises

$$r = \left[\sqrt{2\,\tau_p}\left(\bar{x}(l_p) + \bar{y}(l_p)\,\tan\frac{\alpha}{2}\right) + \cos\tau_p \tan\frac{\alpha}{2} - \sin\tau_p\right]^{-1} \tag{4.50}$$

in Abhängigkeit vom Tangentialwinkel τ_p bestimmen, mit

$$l_p = \sqrt{2\,\tau_p}. \tag{4.51}$$

Der Tangentialwinkel $0 < \tau_p \leq \frac{\alpha}{2}$ bleibt als Freiheitsgrad. Für $\tau_p \approx 0$ werden die Klothoiden verschwindend kurz und der Kreisbogen maximal. Umgekehrt verschwindet der Kreisbogen für $\tau_p = \frac{\alpha}{2}$. Dies entspricht dann dem oben vorgestellten Fall des symmetrischen Klothoidenpaars.

Beim Hinzufügen von Verbundkurven an ein Straßenstück wird dieses Verfahren angewendet. Startpunkt und -richtung sind durch das bestehende Straßenstück gegeben, der Endpunkt wird vom Benutzer per Mausklick gewählt. Die Richtung der Verbundkurve im Endpunkt wird so berechnet, dass eine symmetrische Verbundkurve, wie sie hier vorgestellt wurde, eingefügt werden kann. Für den Tangentialwinkel wird ein Standardwert verwendet, beispielsweise $\tau_p = \frac{\alpha}{4}$.

Entscheidet sich der Benutzer, bereits bestehende Straßenstücke zu modifizieren, kann es dazu kommen, dass die Verbundkurven nicht mehr symmetrisch sind. Diese Fälle werden im nächsten Abschnitt betrachtet.

Asymmetrisches Klothoidenpaar

Zunächst soll wie zuvor der Fall ohne Kreisbogen betrachtet werden. In Abbildung 4.12 sind zwei Klothoiden dargestellt, die sich wieder im Punkt **P** schneiden. Ohne Beschränkung der Allgemeinheit soll die Tangente der unteren Klothoide länger sein. Man erhält den Quotienten

$$k = \frac{g}{h} > 1 \tag{4.52}$$

mit den Tangentenlängen g und h der unteren beziehungsweise der oberen Tangente.

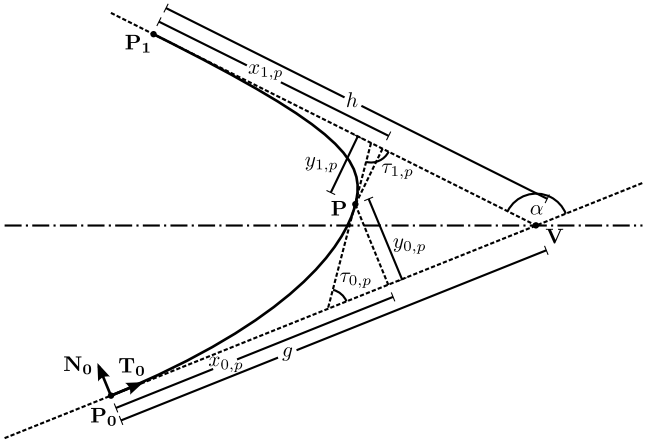


Abbildung 4.12: Asymmetrisches Klothoidenpaar.

Die Tangentialwinkel im Punkt **P** werden als $\tau_{0,p}$ und $\tau_{1,p}$ bezeichnet. Die Tangenten sind parallel wenn

$$\tau_{0,p} + \tau_{1,p} = \alpha \tag{4.53}$$

gilt.

Aus der Bedingung zur Stetigkeit der Krümmung im Punkt **P**

$$\kappa_{0,p} = \kappa_{1,p} \tag{4.54}$$

lässt sich mit den Gleichungen 4.20 und 4.23 das Verhältnis der Klothoidenparameter

$$A_1 = A_0 \sqrt{\frac{\tau_{1,p}}{\tau_{0,p}}} = A_0 \sqrt{\frac{\alpha - \tau_{0,p}}{\tau_{0,p}}} \tag{4.55}$$

bestimmen.

Die geforderte Punktstetigkeit führt zu den Gleichungen

$$x_{0,p} + x_{1,p} \cos\alpha + y_{1,p} \sin\alpha = g + h \cos\alpha \tag{4.56}$$

und

$$y_{0,p} + x_{1,p} \sin\alpha - y_{1,p} \cos\alpha = h \sin\alpha, \tag{4.57}$$

die mithilfe der Gleichungen 4.35, 4.36 und 4.55 auf die Form

$$A_0 \left(\bar{x}_{0,p} + \sqrt{\frac{\alpha - \tau_{0,p}}{\tau_{0,p}}} \left(\bar{x}_{1,p} \cos\alpha + \bar{y}_{1,p} \sin\alpha \right) \right) = g + h \cos\alpha \tag{4.58}$$

$$A_0 \left(\bar{y}_{0,p} + \sqrt{\frac{\alpha - \tau_{0,p}}{\tau_{0,p}}} \left(\bar{x}_{1,p} \sin\alpha - \bar{y}_{1,p} \cos\alpha \right) \right) = h \sin\alpha \tag{4.59}$$

gebracht werden können. Nun kann A_0 eliminiert werden und man erhält die Gleichung

$$\begin{aligned} f(\tau_{0,p}) &= \sqrt{\tau_{0,p}} \left[\bar{x}_{0,p} \sin\alpha - \bar{y}_{0,p} \left(k + \cos\alpha \right) \right] \\ &\quad + \sqrt{\alpha - \tau_{0,p}} \left[\bar{y}_{1,p} \left(1 + k \cos\alpha \right) - \bar{x}_{1,p} k \sin\alpha \right] = 0 \end{aligned} \tag{4.60}$$

in Abhängigkeit vom Tangentialwinkel $\tau_{0,p}$ mit der Ableitung

$$\begin{aligned} f'(\tau_{0,p}) &= \frac{\bar{y}_{0,p} \sin\alpha}{2\sqrt{\tau_{0,p}}} \left(\frac{\bar{x}_{0,p}}{\bar{y}_{0,p}} - \frac{k + \cos\alpha}{\sin\alpha} \right) \\ &\quad + \frac{\bar{y}_{1,p} k \sin\alpha}{2\sqrt{\alpha - \tau_{0,p}}} \left(\frac{\bar{x}_{1,p}}{\bar{y}_{1,p}} - \frac{1 + k \cos\alpha}{k \sin\alpha} \right) \end{aligned} \tag{4.61}$$

und

$$\begin{aligned}
\bar{x}_{0,p} &= \bar{x}(l_{0,p}) = \bar{x}(\sqrt{2\,\tau_{0,p}}) \\
\bar{y}_{0,p} &= \bar{y}(l_{0,p}) = \bar{y}(\sqrt{2\,\tau_{0,p}}) \\
\bar{x}_{1,p} &= \bar{x}(l_{1,p}) = \bar{x}(\sqrt{2\,\alpha - \tau_{0,p}}) \\
\bar{y}_{1,p} &= \bar{y}(l_{1,p}) = \bar{y}(\sqrt{2\,\alpha - \tau_{0,p}}).
\end{aligned}$$

Der gesuchte Tangentialwinkel $\tau_{0,p}$ kann nun durch eine iterative Nullstellensuche mit dem Newton-Raphson-Verfahren ermittelt werden. Es lässt sich zeigen, dass $f(\tau_{0,p})$ in $0 < \tau_{0,p} < \alpha < \pi$ genau eine Nullstelle besitzt [12].

Asymmetrisches Klothoidenpaar mit Kreisbogen

Abbildung 4.13 zeigt eine Verbundkurve aus zwei asymmetrischen Klothoiden und einem Kreisbogen. Die untere Klothoide schneidet den Kreis im Punkt **P** mit dem Tangentialwinkel $\tau_{0,p}$, die obere Klothoide schneidet den Kreis im Punkt **Q** mit dem Tangentialwinkel $\tau_{1,q}$.

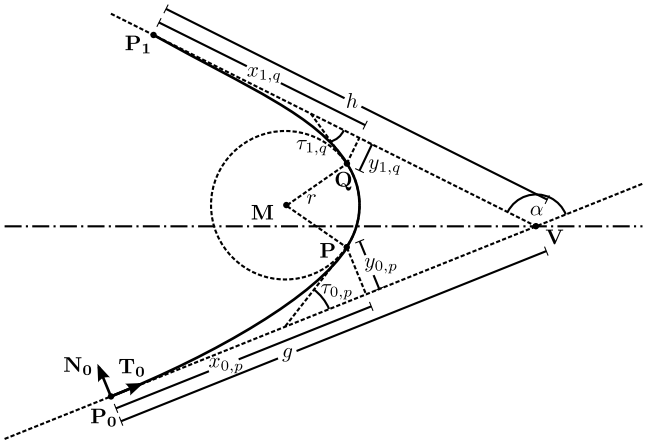


Abbildung 4.13: Asymmetrisches Klothoidenpaar mit Kreisbogen.

Die Stetigkeit der Krümmung fordert, dass die Krümmungen in den Punkten **P** und **Q** gleich der Krümmung des Kreises ist. Es muss also

$$\kappa_{0,p} = \kappa_{Kreis} = \kappa_{1,q} \tag{4.62}$$

gelten. Daraus lässt sich wieder das Verhältnis der Klothoidenparameter

$$A_1 = A_0 \sqrt{\frac{\tau_{1,q}}{\tau_{0,p}}} \tag{4.63}$$

herleiten.

Die Stetigkeit der Lage ist gegeben, wenn die Gleichungen

$$x_{0,p} - r\ \sin\tau_{0,p} + r\ \sin(\alpha - \tau_{1,q}) + x_{1,q}\ \cos\alpha + y_{1,q}\ \sin\alpha = g + h\ \cos\alpha \tag{4.64}$$

und

$$y_{0,p} + r\ \cos\tau_{0,p} - r\ \cos(\alpha - \tau_{1,q}) + x_{1,q}\ \sin\alpha - y_{1,q}\ \cos\alpha = h\ \sin\alpha \tag{4.65}$$

erfüllt sind, mit dem Radius

$$r = \frac{1}{\kappa_{0,p}} = \frac{A_0}{\sqrt{2\,\tau_{0,p}}} \tag{4.66}$$

des Kreises.

Nun lässt sich wieder der Klothoidenparameter A_0 freistellen

$$A_0 \left(\bar{x}_{0,p} - \frac{\sin\tau_{0,p}}{\sqrt{2\,\tau_{0,p}}} + \frac{\sin(\alpha - \tau_{1,q})}{\sqrt{2\,\tau_{0,p}}} + \sqrt{\frac{\tau_{1,q}}{\tau_{0,p}}}\,(\bar{x}_{1,q}\cos\alpha + \bar{y}_{1,q}\sin\alpha) \right) = g + h\ \cos\alpha \tag{4.67}$$

$$A_0 \left(\bar{y}_{0,p} + \frac{\cos\tau_{0,p}}{\sqrt{2\,\tau_{0,p}}} - \frac{\cos(\alpha - \tau_{1,q})}{\sqrt{2\,\tau_{0,p}}} + \sqrt{\frac{\tau_{1,q}}{\tau_{0,p}}}\,(\bar{x}_{1,q}\sin\alpha - \bar{y}_{1,q}\cos\alpha) \right) = h\ \sin\alpha \tag{4.68}$$

und man erhält nach dessen Eliminierung eine Gleichung

$$\begin{aligned} q(\tau_{0,p}, \tau_{1,q}) \ = \ & \sqrt{2\,\tau_{0,p}}\,[\bar{x}_{0,p}\ \sin\alpha - \bar{y}_{0,p}\,(k + \cos\alpha)] \\ & + \sqrt{\tau_{1,q}}\,[\bar{x}_{1,q}\,k\ \sin\alpha - \bar{y}_{1,q}\ (1 + k\ \cos\alpha)] \\ & - \cos(\alpha - \tau_{0,p}) - k\ \cos\tau_{0,p} \\ & + \cos\tau_{1,q} + k\ \cos(\alpha - \tau_{1,q}) = 0 \end{aligned} \tag{4.69}$$

in Abhängigkeit von den beiden Tangentialwinkeln $\tau_{0,p}$ und $\tau_{1,p}$ mit der Ableitung

$$q'(\tau_{0,p}) = \frac{\bar{y}_{0,p}\ \sin\alpha}{\sqrt{2\,\tau_{0,p}}} \left(\frac{\bar{x}_{0,p}}{\bar{y}_{0,p}} - \frac{k + \cos\alpha}{\sin\alpha} \right) \tag{4.70}$$

und

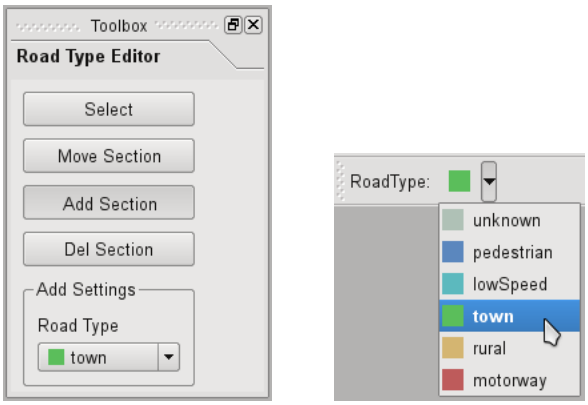
$$\begin{aligned}
\bar{x}_{0,p} &= \bar{x}(l_{0,p}) = \bar{x}(\sqrt{2\,\tau_{0,p}}) \\
\bar{y}_{0,p} &= \bar{y}(l_{0,p}) = \bar{y}(\sqrt{2\,\tau_{0,p}}) \\
\bar{x}_{1,q} &= \bar{x}(l_{1,q}) = \bar{x}(\sqrt{2\,\tau_{1,q}}) \\
\bar{y}_{1,q} &= \bar{y}(l_{1,q}) = \bar{y}(\sqrt{2\,\tau_{1,q}}).
\end{aligned}$$

Der Tangentialwinkeln $\tau_{1,q}$ kann frei gewählt werden. Als Maximalwert kommt dabei der berechnete Tangentialwinkel des asymmetrischen Falls ohne Kreisbogen in Frage, wobei der Kreisbogen dann verschwindet. Für $\tau_{1,q} \approx 0$ hingegen verschwindet die zweite Klothoide und der Kreisbogen wird maximal.

Für ein gewähltes $\tau_{1,q}$ in diesem Bereich besitzt $q(\tau_{0,p}, \tau_{1,q})$ genau eine Nullstelle, die erneut mit dem Newton-Raphson-Verfahren bestimmt werden kann.

4.2 RoadTypeEditor

Mit dem RoadTypeEditor können für jede Straße Segmente definiert werden, die den Straßentyp angeben. Dabei stehen fünf Arten von der Fußgängerzone bis zur Autobahn zur Auswahl. In Abbildung 4.14a ist die Werkzeugleiste des RoadTypeEditor dargestellt. Mit **Add Section** lassen sich neue Segmente hinzufügen. Dazu klickt man auf ein bestehendes Segment, das dann an der aktuellen Mausposition geteilt wird. Um die Straßenkoordinate der aktuellen Mausposition zu berechnen, ist ein numerisches Verfahren nötig, das in Abschnitt 4.2.1 vorgestellt wird. Das neue Segment erhält den Straßentyp, der derzeit bei den **Add Settings** ausgewählt ist. Bestehende Segmente können mit dem Werkzeug **Move Section** verschoben und mit **Del Section** gelöscht werden. Auch hierfür muss jeweils wieder die Straßenkoordinate der Mausposition berechnet werden. Der Straßentyp eines Segments kann geändert werden, indem man es mit dem Werkzeug **Select** markiert und in der Menüleiste den gewünschten Straßentyp auswählt, wie in Abbildung 4.14b zu sehen ist.



(a) Werkzeugleiste (b) Menüleiste

Abbildung 4.14: RoadTypeEditor.

4.2.1 Kürzester Abstand eines Punktes zur Straßenachse

Gesucht ist die Straßenkoordinate s_r desjenigen Punktes $\mathbf{r}$ der Straßenachse, der zu einem gegebenen Punkt $\mathbf{p}$ den kürzesten Abstand $\|\mathbf{d}\|$ hat. Da die Straßenkoordinate iterativ gesucht wird, ist es deshalb sinnvoll, von einem möglichst genauen Startwert s_0 und einem möglichst kleinen Bereich

$$s_s \leq s_0 \leq s_e \tag{4.71}$$

auszugehen. Der Bereich lässt sich auf das Segment der Straße einschränken, das unter der aktuellen Mausposition liegt. Für einen Startwert wird eine grobe Annäherung der Straße verwendet, die im nächsten Abschnitt vorgestellt wird [7].

Berechnung eines Startwertes

Die Straßenachse wird im untersuchten Intervall $[s_s, s_e]$ zunächst durch n Geradenstücke angenähert, die von den $n+1$ Stützstellen $r_{St,0}, r_{St,1}, ...r_{St,n}$ begrenzt werden. Ein Beispiel dazu ist in Abbildung 4.15 zu sehen.

Die Stützstellen werden dabei gleichmäßig über die Straße verteilt und befinden sich demnach an den Straßenkoordinaten

$$s_{St,i} = s_s + i\frac{s_e - s_s}{n}. \tag{4.72}$$

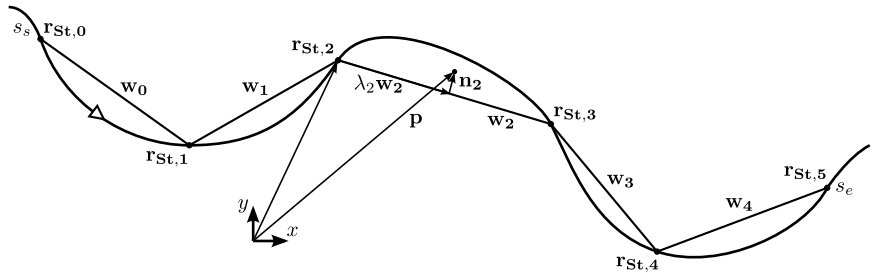


Abbildung 4.15: Suche eines geeigneten Startwerts.

Für die Gerade i kann die Parameterform

$$g_i : \mathbf{r_i} = \mathbf{r_{St,i}} + \lambda_i \mathbf{w_i} \tag{4.73}$$

mit

$$\mathbf{w_i} = \mathbf{r_{St,i+1}} - \mathbf{r_{St,i}}. \tag{4.74}$$

aufgestellt werden.

Für jedes Straßenstück wird der Normalenvektor

$$\begin{aligned} \mathbf{n_i} &= \mathbf{p} - \mathbf{r_i} \\ &= \mathbf{p} - \mathbf{r_{St,i}} - \lambda_i \mathbf{w_i}. \end{aligned} \tag{4.75}$$

berechnet, der auf den Punkt $\mathbf{p}$ zeigt. Durch Multiplikation mit $\mathbf{w_i}$ und der Bedingung für die Orthogonalität

$$\mathbf{n_i} \cdot \mathbf{w_i} = 0 \tag{4.76}$$

erhält man den noch fehlenden Faktor

$$\lambda_i = \frac{\mathbf{w_i}}{\|\mathbf{w_i}\|} \left(\mathbf{p} - \mathbf{r_{St,i}}\right). \tag{4.77}$$

Bestimmt man nun das Geradenstück m, das den kürzesten Normalenvektor besitzt, lässt sich eine erste Näherung

$$s_0 = s_{St,m} + \lambda_m \left(s_{St,m+1} - s_{St,m}\right) \tag{4.78}$$

für die Straßenkoordinate herleiten.

Berechnung der Straßenkoordinate durch Iteration

Die genaue Straßenkoordinate kann jetzt durch ein iteratives Verfahren berechnet werden. In Abbildung 4.16 ist Schritt i dargestellt. Die aktuelle Annäherung der Straßenkoordinate ist s_i, dazu gehören der Punkt $\mathbf{r_i}$, die Tangente $\mathbf{t_i}$ und die Normale $\mathbf{n_i}$.

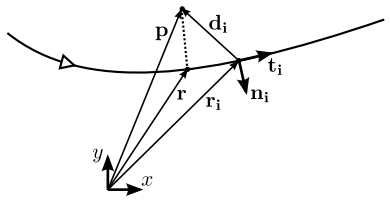


Abbildung 4.16: Schritt i der Iteration.

Für den Punkt mit minimalem Abstand muss die Bedinung

$$\mathbf{d} \cdot \mathbf{t} = 0 \tag{4.79}$$

erfüllt sein, das heißt Abstandsvektor $\mathbf{d}$ und Tangentenvektor $\mathbf{t}$ müssen orthogonal zueinander sein. Vorallem bei längeren Straßen können mehrere solcher Punkte existieren, weshalb ein gut gewählter Startwert wichtig ist.

Eine Nullstelle der Funktion

$$f(s) = \mathbf{d}(s) \cdot \mathbf{t}(s) \tag{4.80}$$

lässt sich mit dem Newton-Raphson-Verfahren ersten Grades [13] finden, für das die Iterationsvorschrift

$$s_{i+1} = s_i - \frac{f(s_i)}{f'(s_i)} \tag{4.81}$$

gilt.

Durch Ableiten der Funktion 4.80 mit der Produktregel erhält man

$$f'(s) = \frac{d}{ds}(\mathbf{d}(s) \cdot \mathbf{t}(s)) \tag{4.82}$$

$$= \frac{d}{ds}\mathbf{d}(s) \cdot \mathbf{t}(s) + \mathbf{d}(s) \cdot \frac{d}{ds}\mathbf{t}(s). \tag{4.83}$$

Mit

$$\mathbf{d}(s) = \mathbf{p} - \mathbf{r}(s) \tag{4.84}$$

lässt sich die Ableitung des Abstandsvektors

$$\frac{d}{ds}\mathbf{d}(s) = -\frac{d}{ds}\mathbf{r}(s) \tag{4.85}$$

bestimmen, wobei

$$\frac{d}{ds}\mathbf{r}(s) = \mathbf{t}(s) \tag{4.86}$$

dem Tangentenvektor entspricht.

Ebenso benötigt man die Ableitung des Tangentenvektors

$$\frac{d}{ds}\mathbf{t}(s) = -\kappa(s)\,\mathbf{n}(s) \tag{4.87}$$

mit der Krümmung $\kappa(s)$ und dem Normalenvektor $\mathbf{n}(s)$. Das negative Vorzeichen kommt dabei daher, dass der Normalenvektor hier nach außen zeigt. In der Literatur wird er jedoch auch oft zum Kurveninneren eingezeichnet.

Schlussendlich erhält man für die Ableitung

$$\begin{aligned} f'(s) &= -\mathbf{t}(s)^2 - \kappa(s)\,\mathbf{n}(s) \\ &= -1 - \kappa(s)\,\mathbf{n}(s) \end{aligned} \tag{4.88}$$

und damit für die Iterationsvorschrift

$$s_{i+1} = s_i - \frac{\mathbf{d}(s_i)\cdot\mathbf{t}(s_i)}{-1 - \kappa(s_i)\,\mathbf{n}(s_i)}. \tag{4.89}$$

Die Iteration wird ausgeführt, bis das Abbruchkriterium

$$|s_{i+1} - s_i| < \epsilon \tag{4.90}$$

erreicht ist. Für ϵ bietet sich bereits ein Wert im Zentimeterbereich an.

4.3 Elevation-, Superelevation- und CrossfallEditor

Mit dem ElevationEditor können Straßensegmente erstellt werden, welche die Höhe entlang der Straßenachse definieren. Analog dazu werden mit dem

SuperelevationEditor und dem CrossfallEditor Straßensegmente zur Definition des Neigungswinkels erstellt. In den Kapiteln 2.3.4 und 2.3.5 wurden der Höhenplan und die Querneigung bereits vorgestellt.

Ein Screenshot des ElevationEditor mit Draufsicht und Profilansicht einer Straße ist in Abbildung 4.17 zu sehen. Abbildung 4.18 zeigt die Werkzeugleiste des ElevationEditor.

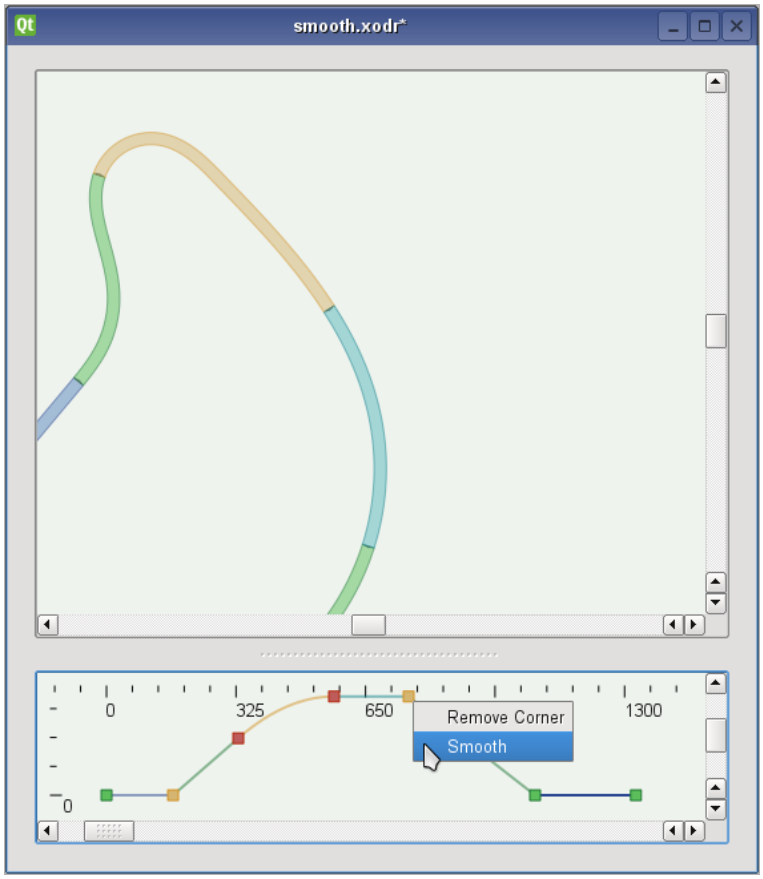


Abbildung 4.17: ElevationEditor mit Profilansicht.

Die Werkzeuge **Select**, **Add Section** und **Del Section** funktionieren analog zu denen des RoadTypeEditor und wurden in Kapitel 4.2 erläutert. Neu sind die Funktionen der Profilansicht. Diese erscheint im unteren Bildrand, wenn mit **Select** eine Straße auswählt wird. Die Straße wird dort mit steigender Straßenkoordinate von links nach rechts angezeigt. Die einzelnen Straßensegmente werden durch quadratische Handles getrennt, mit denen die Segmente verschoben werden können. Eine horizontale Verschiebung bedeutet

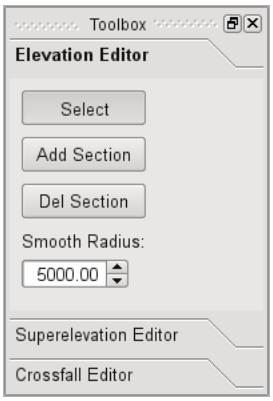


Abbildung 4.18: ElevationEditor: Werkzeugleiste.

dabei eine Veränderung der Startkoordinate eines Segments, eine vertikale Verschiebung definiert die Höhe beziehungsweise den Neigungswinkel. Die Handles werden in den Farben grün, gelb und rot dargestellt. Grüne Handles begrenzen Segmente ersten Grades und sind frei verschiebbar. Rote Handles begrenzen ein Segment mit einem Polynom zweiten oder dritten Grades und können nicht verschoben werden. Ein Segment ersten Grades, das einem Segment höheren Grades folgt, wird mit einem gelben Handle begrenzt, der ebenfalls nicht verschiebbar ist.

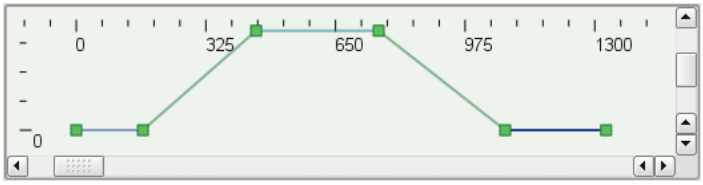
Es empfiehlt sich deshalb folgender Ablauf, um einen Höhenplan oder die Querneigung zu definieren. Zunächst wird ein grobes Profil aus Geradenstücken erstellt (Abbildung 4.19a). Durch einen Rechtsklick auf einen Handle öffnet sich ein Kontextmenü, mit dem weitere Funktionen angeboten werden. Hier lässt sich der Übergang zwischen zwei Geraden durch ein Polynom zweiten Grades glätten, wie in Abbildung 4.19b dargestellt ist. Nach dem Glätten aller Ecken erhält man einen fließenden Verlauf des Profils (Abbildung 4.19c).

Die mathematische Betrachtung des Glättungsvorgangs wird im folgenden Abschnitt vorgenommen.

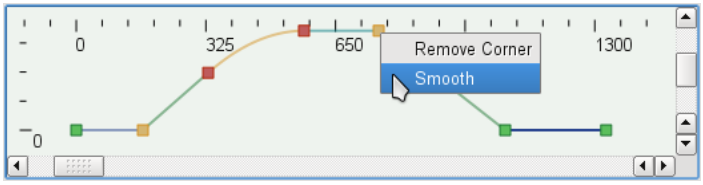
4.3.1 Glättung des Profils

Die in Abbildung 4.20 dargestellten Geradenstücke

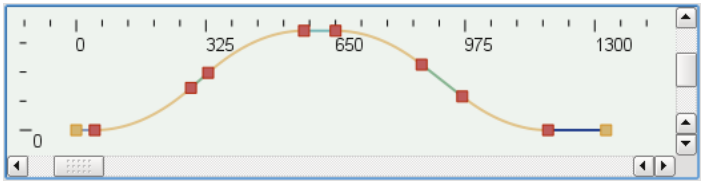
$$g(s) = a_g + b_g\, s \tag{4.91}$$



(a) Grobe Definition des Profils mit Geraden.



(b) Glätten des Profils.



(c) Vollständig geglättetes Profil.

Abbildung 4.19: Vorgang beim Erstellen eines Profils.

und

$$k(s) = a_k + b_k\, s \tag{4.92}$$

sollen durch das Polynom zweiten Grades

$$f(s) = a + b\, s + c\, s^2 \tag{4.93}$$

so verbunden werden, dass ein weicher, stetiger Übergang gewährleistet ist.

Für die Ermittlung der Parameter a, b und c werden die Ableitungen

$$f'(s) = b + 2\, c\, s \tag{4.94}$$
$$f''(s) = 2\, c \tag{4.95}$$

und die Randbedingungen an der Stelle $s = 0$ benötigt. Man sieht, dass Höhe und Steigung

$$f(0) = a = a_g \tag{4.96}$$
$$f'(0) = b = b_g \tag{4.97}$$

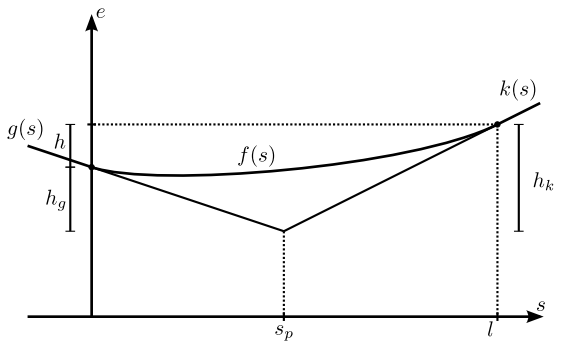


Abbildung 4.20: Glättung zweier Geraden durch ein Polynom zweiten Grades.

direkt vom ersten Geradenstück übernommen werden können. Mit der Krümmung an der Stelle $s = 0$

$$\kappa(0) = \frac{f''(0)}{(1 + f'(0)^2)^{\frac{3}{2}}} = \frac{2\,c}{(1 + b^2)^{\frac{3}{2}}} = \frac{1}{r} \tag{4.98}$$

und einem vom Benutzer festgelegten Radius r kann schließlich der dritte Parameter

$$c = \frac{1}{2\,r}\left(1 + b^2\right)^{\frac{3}{2}} \tag{4.99}$$

bestimmt werden. Durch den Glättungsradius r kann der Benutzer also vorgeben, wie stark der Übergang geglättet werden soll.

Um den Abstand zum Schnittpunkt s_p der beiden Geraden zu berechnen, werden zunächst die Längen l und h benötigt. Aus der Randbedingung

$$f'(l) = b + 2\,c\,l = b_k \tag{4.100}$$

für die Ableitung an der Stelle $s = l$ ergibt sich direkt die Länge

$$l = \frac{b_k - b}{2\,c}. \tag{4.101}$$

Die Höhendifferenz

$$h = f(l) - f(0) = b\,l + c\,l^2 \tag{4.102}$$

kann dann in Abhängigkeit von l dargestellt werden. Für die Höhendifferenz gilt außerdem

$$h = h_k - h_g = b_k\ (l - s_p) + b_g\,s_p, \tag{4.103}$$

woraus letztendlich der gesuchte Abstand zum Geradenschnittpunkt

$$s_p = \frac{h - b_k\, l}{b_g - b_k} \tag{4.104}$$

ermittelt werden kann. Bevor die Geradenstücke geglättet werden, muss noch geprüft werden, ob die Geradenstücke lang genug sind, um dazwischen das Polynom einfügen zu können. Ist dies nicht der Fall, kann der Benutzer den Vorgang mit einem geringeren Radius wiederholen und der Übergang wird dementsprechend weniger stark geglättet.

4.4 Satellitenbilder und Straßenkarten

Das Erstellen von Straßennetzen die reale Strecken nachempfinden, kann durch das Einblenden von Satellitenbildern und Straßenkarten erheblich vereinfacht werden. In das in Abbildung 4.21 dargestellte Projekt wurden beispielsweise zwei Straßenkarten des Kreuz Stuttgart eingefügt und es wurde begonnen, die Autobahn nachzubilden.

Mit der Funktion **Load Map** der in Abbildung 4.22 dargestellten Menüleiste lassen sich Bilder vieler gängiger Dateiformate wie PNG, JPEG und GIF laden. Diese können dann über die Eingabe von x- und y-Koordinaten platziert oder direkt mit der Maus verschoben werden. Zwei weitere Felder erlauben das Einstellen einer gewünschten Breite oder Höhe, um sie an den Maßstab anzupassen.

Zuletzt kann eine Deckkraft (**Opacity**) vorgegeben werden. Vorallem bei sehr farbintensiven Satellitenbildern bietet es sich an, die Deckkraft zu senken, um so die Sichtbarkeit der Straßensegmente zu erhöhen. Nicht zuletzt lassen sich hiermit auch Karten verschiedener Zoomstufen überblenden. Mit **Delete Map** können geladene Karten wieder gelöscht werden.

Um den Arbeitsfluss weiter zu vereinfachen wurde die Funktion **Lock Maps** implementiert, mit der die Bearbeitung der Karten gesperrt werden kann. So werden die Karten nach dem Einstellen der Position vor unabsichtlichen Änderungen geschützt.

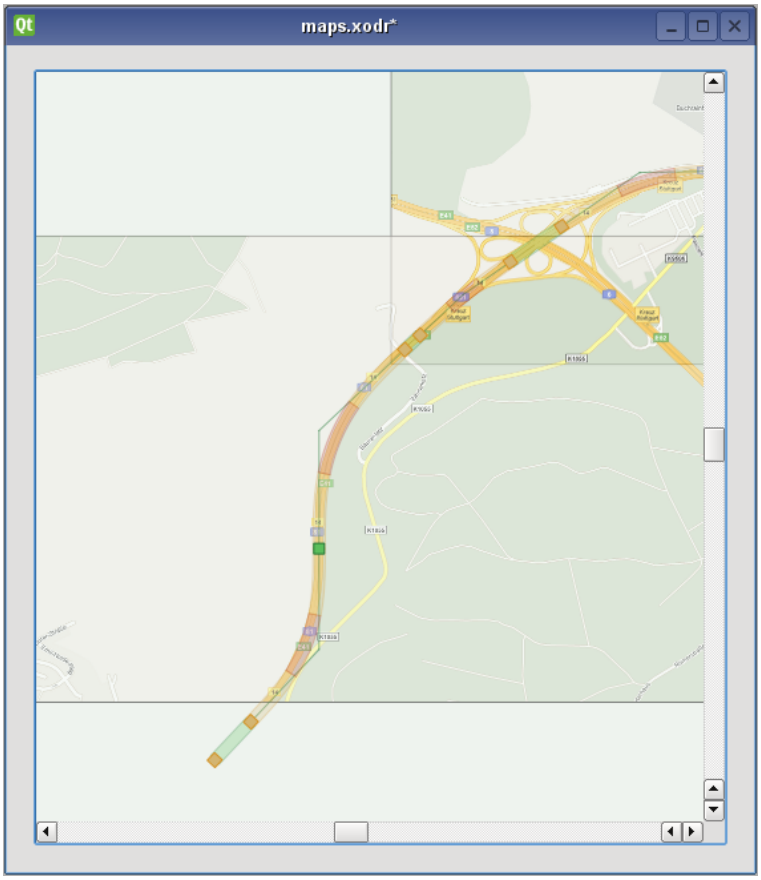
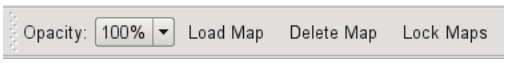


Abbildung 4.21: Einblenden von Satellitenbildern und Straßenkarten.



(a)

(b)

Abbildung 4.22: Menüleiste zum Einblenden von Straßenkarten.

Kapitel 5

Ausblick

In diesem Kapitel werden einige Ideen zusammengefasst, die während des Erstellens der Arbeit aufgekommen sind und für die Weiterentwicklung des Programms interessant sein können.

Derzeit können Straßen nur aus Geradenstücken und Verbundkurven aufgebaut werden. Dies reicht in den meisten Fällen aus und vereinfacht das Erstellen von Straßennetzen, indem es dem Benutzer die Freiheit (oder die Belastung) nimmt, sich beispielsweise mit Klothoidenparametern und Krümmungen auseinander zu setzen. Dennoch ist es wünschenswert, erfahrenen Benutzern die Möglichkeit zu geben, genau dies zu tun. Das Einfügen und Editieren von einzelnen Kreisbögen und Klothoiden oder gar Wende- und Eiklothoiden würde so das Arbeiten mit Spezialfällen erleichtern. Momentan ist dies nur über Prototypen möglich. Auch für den TrackEditor könnte eine Profilansicht hilfreich sein, die den Krümmungsverlauf darstellt.

Im Fokus der Arbeit stand die Erstellung einer grafischen Benutzeroberfläche, die es dem Benutzer erlaubt, mit wenigen Mausklicks ein Straßennetz zu erschaffen. Dies wurde mit den beiden Präsentationen `ProjectGraph` für die Draufsicht und `ProfileGraph` für die Profilansicht gelöst. Wünschenswert wäre aber auch eine dritte Präsentation, die `TextView`, mit der ein Benutzer durch die Eingabe von Zahlenwerten die Szene modifizieren kann. So würde das Erstellen von Strecken vereinfacht, für die exakte Maßangaben existieren. Durch die flexible Programmstruktur sollte es problemlos möglich sein, die zusätzliche Präsentation in das Programm zu integrieren.

Auch wäre es von Interesse, zu erörtern, inwieweit das vom HLRS und der Porsche AG verwendete Visualisierungsprogramm OpenCOVER als dreidimensionale Vorschau verwendet werden kann. Da mit dieser Software auch

die eigentliche Fahrsimulation durchgeführt wird, wäre es dadurch theoretisch möglich, Strecken zur Laufzeit des Programms zu verändern. Dazu müsste aber auch die komplette Fremdfahrzeugsimulation in das Observer-Pattern integriert werden.

Da die im OpenDRIVE Format spezifizierten Verkehrsschilder, Ampeln und Objekte am Fahrbahnrand in der Fremdfahrzeugsimulation nicht berücksichtigt werden, wurden sie auch im Rahmen dieser Arbeit nicht umgesetzt. Hier wäre es interessant, das Verhalten der Fremdfahrzeuge vorallem an Kreuzungen mit Verkehrsregeln und Ampeln zu verbessern. OpenDRIVE gibt insbesondere für Ampelschaltungen keine Formatvorgaben. Diese könnten in einer Nachfolgearbeit definiert werden. Es bietet sich dann an, einen weiteren Editor in das Programm zu integrieren, mit dem die Ampelschaltungen editiert werden können.

Am 17. August 2010 wurde OpenDRIVE in der leicht erweiterten Version 1.3 Rev. D veröffentlicht. Diese Änderungen müssen noch in das Programm und die Fremdfahrzeugsimulation aufgenommen werden.

Kapitel 6

Zusammenfassung

In der vorliegenden Arbeit wurde ein Programm konzipiert und erstellt, mit dem Straßennetze für Fahrsimulationen erschaffen werden können. Um eine einfache und effiziente Bedienung zu ermöglichen wurde mithilfe der Klassenbibliothek Qt eine grafische Benutzeroberfläche mit Multiple Document Interface programmiert.

Die Programmstruktur basiert auf dem Model-View-Controller Architekturmuster. Die darin beschriebene strikte Trennung von Datenmodell und Visualisierung erleichtert Änderungen im Programmcode durch verringerte Abhängigkeiten und reduziert dadurch auch die Fehleranfälligkeit. Der Aufbau des Modells orientiert sich am OpenDRIVE Format, das vom HLRS und der Porsche AG für die Fahrsimulation und das Simulieren von Fremdfahrzeugen verwendet wird. Das erstellte Programm kann Daten in diesem Dateiformat lesen und schreiben.

In der Konzeptphase wurde auf bewährte Entwurfsmuster der Softwareentwicklung zurückgegriffen. Insbesondere das Command-Pattern, mit dem das Rückgängigmachen und Wiederherstellen von einzelnen Arbeitsschritten ermöglicht wird und das Observer-Pattern, über das die Objekte der Präsentation von Änderungen im Modell informiert werden, sind elementare Bestandteile des Programms. Durch diese konnten die Grundprinzipien der Model-View-Controller Architektur umgesetzt und eine flexible und stabile Programmstruktur erschaffen werden.

Die Präsentationen wurden mit dem Qt Graphics View Framework umgesetzt. In der Draufsicht kann mit dem TrackEditor der Verlauf der Straße, die sogenannte Straßenachse, definiert werden. Die Straßen werden aus Geraden und Verbundkurven zusammengesetzt. Besonders für Klothoiden die

zusammen mit einem Kreisbogen die Verbundkurven bilden, sind komplexe numerische Berechnungen nötig, die hier detailliert vorgestellt wurden. Das Laden von Prototypen ermöglicht das Wiederverwenden von einfachen Strecken bis hin zu komplexen Kreuzungen. Das Nachbilden realer Straßenverläufe wird durch das Einblenden von Satellitenbildern und Straßenkarten vereinfacht.

Mit dem RoadTypeEditor kann einer Straße ein Straßentyp von der Fußgängerzone bis zur Autobahn zugewiesen werden. In drei weiteren Editoren wird eine zusätzliche Profilansicht eingeblendet, um einen Höhenplan zu erstellen oder um eine Querneigung der Straße anzugeben.

Anhang A

Programmierstil

Bei der Implementierung der vorliegenden Arbeit wurde viel Wert auf sauberen Programmcode gelegt, um ihn flexibel und nachvollziehbar zu gestalten. Dazu wurden zahlreiche Richtlinien und Empfehlungen von Scott [2] sowie Henricson und Nyquist [3] umgesetzt.

Insbesondere wurde auf Datenkapselung geachtet, das heißt stets private member-Variablen verwendet. Dies erleichtert Änderungen innerhalb von Klassen und verhindert durch eine klar definierte Schnittstelle zudem undurchsichtige und fehleranfällige Interaktionen, was den leicht erhöhten Programmieraufwand und eventuelle Geschwindigkeitseinbußen rechtfertigt. Außerdem sollte das Schlüsselwort `const` so weit möglich und sinnvoll eingesetzt werden, um so die Interfaces der Klassen deutlicher zu machen. Dadurch lassen sich auch `const` Objekte verwenden, was wiederum die Fehleranfälligkeit verringert. Variablen sollten über eine Initialisierungsliste bereits im Konstruktorrumpf Werte zugewiesen werden, insbesondere Zeigern.

Um die Lesbarkeit zu verbessern sollten member-Variablen stets mit einem Unterstrich enden, beispielsweise `meineVariable_`. Klassen werden groß geschrieben, Variablen klein, die Namen sollten dabei ausgeschrieben werden und möglichst keine Abkürzungen enthalten.

Jede Datei sollte maximal eine Klasse enthalten, es sei denn mehrere Klassen werden stets zusammen verwendet wie beispielsweise Commands. In Header-Dateien sollte nach Möglichkeit auf Include-Files verzichtet werden und diese stattdessen durch Forward Declarations ersetzt werden. Durch diese Maßnahmen kann die Zeit zum Kompilieren des Programms nach lokalen Änderungen deutlich beschleunigt werden, da die Abhängigkeiten sinken.

Anhang B

OpenDrive Erweiterungen

Wie in Kapitel 4.4 vorgestellt wurde, ist es möglich Satellitenbilder und Straßenkarten einzublenden. Diese sollten ebenfalls gespeichert werden, um nach einem erneuten Laden des Projekts die Einstellungen nicht erneut vornehmen zu müssen.

In [7] wurden bereits einige Erweiterungen des OpenDRIVE Formats vorgenommen, unter Anderem die Elemente `<scenery>` und `<environment>`. Die Straßenkarten werden im Element `<scenery>` gespeichert und besitzen folgende Parameter:

`<map>`

- `x:` x-Koordinate des linken oberen Bildrandes in [m].
- `y:` y-Koordinate des linken oberen Bildrandes in [m].
- `width:` Breite des Bildes in [m] (nicht Pixel).
- `height:` Höhe des Bildes in [m].
- `opacity:` Deckkraft im Bereich [0.0, 1.0].
- `id:` Einzigartige ID des Bildes.
- `filename:` Pfad und Dateiname des Bildes.

Desweiteren wird empfohlen, das `<environment>` Element ebenfalls in das `<scenery>` Element zu integrieren und das ursprüngliche `<scenery>` Element

wie folgt umzustellen. Hierdurch wird ein einheitliches `<scenery>` Element geschaffen, welches das SceneSystem repräsentiert.

`<file>`

- `filename` Pfad und Dateiname der Geometriedatei.
- `id` Einzigartige ID.

`<tesselation>`

- `active` Automatische Tesselierung der Straße aktivieren (`true` oder `false`).

Abbildungsverzeichnis

Literaturverzeichnis

[1] Blanchette, J.; Summerfield, M.: C++ GUI Programming with Qt 4. 1st ed. *Prentice Hall Open Source Software Development Series*, 2006.

[2] Meyers, S.: Effective C++: 55 Specific Ways to Improve Your Programs and Designs. 3rd ed. *Addison-Wesley Professional*, 2005.

[3] Henricson, M.; Nyquist, E.: Programming in C++, Rules and Recommendations. *Ellemtel Telecommunication Systems Laboratories*, 1992.

[4] Reenskaug, T.: The Model-View-Controller (MVC). Its Past and Present. *JavaZONE, Oslo*, 2003. *JAOO, Aarhus*, 2003.

[5] Dupuis, M.: OpenDRIVE®Format Specification. Rev. 1.2. *Rosenheim: VIRES Simulationstechnologie GmbH*, 2008.

[6] Dupuis, M.: ROD User Manual. *Rosenheim: VIRES Simulationstechnologie GmbH*, 2007.

[7] Seybold, F.: Simulation von Fremdfahrzeugverhalten im Einsatzfeld des virtuellen Fahrerplatzes. *Diplomarbeit an der Universität Stuttgart*, 2008.

[8] Richtlinien für die Anlage von Straßen - Teil: Linienführung (RAS-L). *Forschungsgesellschaft für Straßen- und Verkehrswesen e. V. (FGSV)*, 1995.

[9] Richtlinien für die Anlage von Straßen - Teil: Querschnitt (RAS-Q). *Forschungsgesellschaft für Straßen- und Verkehrswesen e. V. (FGSV)*, 1982.

[10] Ressel, W.: Vorlesungsskript: Einführung in die Verkehrsplanung - Teil: Straßenwesen. *Institut für Straßen und Verkehrswesen, Universität Stuttgart*, 2009.

[11] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software. 1st ed. *Addison-Wesley Professional*, 1994.

[12] Walton, D.J.; Meek D.S.: A controlled clothoid spline. *Computers and Graphics*, 29(3):353-363, 2005.

[13] Freund, R.W.; Hoppe R.H.W.: Stoer/Bulirsch: Numerische Mathematik 1. 10. Auflage. *Berlin, Heidelberg: Springer-Verlag*, 2007.