

**Großer Beleg**

# **Dynamische Streckengenerierung und deren Einbettung in ein Terrain**

Michael Franke

März 2011

Technische Universität Dresden  
Fakultät Informatik  
Institut für Multimediatechnik  
Professur Computergrafik

verantwortlicher Hochschullehrer: Prof. Dr. rer. nat. Stefan Gumhold  
Betreuer: Dipl.-Inf. Daniel Höhne, Dipl.-Medieninf. Sören König



## **Selbstständigkeitserklärung**

Hiermit erkläre ich, Michael Franke, dass ich diese Belegarbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

---

Dresden, März 2011



## **Abstract**

Die vorliegende Arbeit beschreibt die Konzeption und Umsetzung eines Fahrbahn simulators. Das Hauptaugenmerk liegt dabei auf der Realisierung einer dynamischen Streckenerzeugung, die sich durch die Variation entsprechender Parameter während der Laufzeit der Simulation anpassen lässt. Zuerst werden Überlegungen vorgestellt, wie eine solche dynamische Anpassung vorgenommen werden kann und die bevorzugte Variante vorgestellt. Danach wird der theoretische Teil der Arbeit besprochen - von der Vorstellung der zugrunde liegenden Konzepte von Strecken- und Terraingenerierung bis zur konkreten Umsetzung. Im Anschluss daran wird das verwendete Grafikframework vorgestellt und auf den Aufbau sowie bestimmte Details der begleitend erstellten Applikation eingangen. Das letzte Kapitel befasst sich mit der Auswertung und Analyse der erzielten Resultate. Ein Ausblick beschließt dann diese Arbeit.



# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>9</b>
1.1 Motivation und Kontext dieser Arbeit . . . . .	9
1.2 Der Aufbau dieser Arbeit . . . . .	9
<b>2 Die Vorstellung der Aufgabe</b>	<b>9</b>
<b>3 Verwandte Arbeiten</b>	<b>10</b>
<b>4 Vorüberlegungen</b>	<b>10</b>
4.1 Die Auswahl der Terraingenerierungsmethode . . . . .	11
4.2 Die Einbettung der Straße . . . . .	12
4.3 Streckenverlauf und Rasterung des Terrains . . . . .	12
<b>5 Die Vorstellung der Arbeit</b>	<b>14</b>
5.1 Grundlegende Konzepte . . . . .	15
5.1.1 Der TriangleEdge-Algorithmus . . . . .	15
5.1.2 Der TriangleEdge-Algorithmus mit Constraints . . . . .	17
5.1.3 Modellierung des Straßenverlaufs . . . . .	19
5.1.4 Parameter für die Streckenanpassung . . . . .	25
5.2 Generierung der (Meta-)Strecke . . . . .	26
5.3 Ablauf der Zellgenerierung . . . . .	29
5.3.1 Straßenverlauf als Bézierkurve . . . . .	29
5.3.2 Einbettung der Straße in die Zelle . . . . .	30
5.3.3 Generierung des Terrains . . . . .	31
5.3.4 Generierung der Gimmicks . . . . .	32
<b>6 Umsetzung und Implementierung</b>	<b>33</b>
6.1 Die Auswahl der Grafikengine . . . . .	34
6.2 Die OGRE-Engine allgemein . . . . .	34
6.2.1 Vorstellung des OGRE-Frameworks . . . . .	34
6.2.2 Entwicklung einer Anwendung mit OGRE . . . . .	35
6.3 Aufbau der Belegapplikation . . . . .	36
6.3.1 Vorstellung der Module . . . . .	38
6.3.2 Verwendete Fremd-Plugins . . . . .	40
<b>7 Eine Analyse der Arbeit</b>	<b>41</b>
7.1 Benchmark der Applikation . . . . .	41
7.2 Analyse der veränderbaren Parameter . . . . .	44
7.3 Fazit . . . . .	46
<b>8 Ausblick und Erweiterungsmöglichkeiten</b>	<b>47</b>
<b>Literatur</b>	<b>49</b>

## Abbildungsverzeichnis

1	Computergeneriertes Terrain (Bild aus der Beleg-Simulation) . . . . .	11
2	Schematische Darstellung der Meshes von Straße und Terrain . . . . .	13
3	Sichtbereiche - das rote Kreuz markiert die Fahrerposition . . . . .	14
4	Punktgitter - Draufsicht und 3D-Ansicht . . . . .	15
5	Der TriangleEdge-Algorithmus . . . . .	16
6	Verschiedene Artefakte . . . . .	17
7	Angepasste Formel des TriangleEdge-Verfahrens (aus Belhadj (2007)) . . . . .	18
8	Schematische Darstellung des MDBU PROzesses (aus Belhadj (2007)) . . . . .	19
9	Schematische Darstellung einer Klothoide mitsamt Krümmungsband (aus TU Dresden (2010)) . . . . .	21
10	Schematische Darstellung des Straßenverlaufs als Bézierkurve (Draufsicht) . .	22
11	Zwei mögliche „Anomalitäten“ bei Bézierkurven . . . . .	23
12	Schematische Darstellung einer Schlaufe bei der Wegfindung . . . . .	27
13	Beispiel einer Zeilen- und Spaltentabelle vor und nach Einfügen der Zelle (X:4, Y:3) . . . . .	28
14	Schematische Darstellung der verzögerten Generierung von Zellen . . . . .	29
15	Schematische Darstellung des Straßenmeshes entlang der Bézierkurve . . . .	30
16	Einbettung des Straßenmeshes in das Zellgitter (inklusive Darstellung des Strassenmeshes) . . . . .	31
17	In das Terrain eingebetteter Straßenverlauf . . . . .	32
18	Szene aus der fertigen Applikation . . . . .	33
19	Zeiten zur Generierung und Speicherung von Zellen (Detailgrad 8 - Initialisierungsphase - Werte in Sekunden) . . . . .	42
20	Zeiten zur Generierung und Speicherung von Zellen (Detailgrad 8 - Laufzeit - Werte in Sekunden) . . . . .	42
21	Zeiten zur Generierung und Speicherung von Zellen (Detailgrad 9 - Initialisierungsphase - Werte in Sekunden) . . . . .	43
22	Zeiten zur Generierung und Speicherung von Zellen (Detailgrad 9 - Laufzeit - Werte in Sekunden) . . . . .	43
23	Zeiten der einzelnen Phasen der Generierung (Detailgrad 8 - Laufzeit - Werte in Sekunden) . . . . .	43
24	Zeiten der einzelnen Phasen der Generierung (Detailgrad 9 - Laufzeit - Werte in Sekunden) . . . . .	43
25	Höhenverlauf für verschiedene Parameter bei mehreren Testläufen - Straßenhöhenwerte an den Zellgrenzen für jeweils fünf Testläufe zu zehn Zellen mit verschiedenen Erwartungswerten . . . . .	45

# 1 Einführung

## 1.1 Motivation und Kontext dieser Arbeit

Sowohl im Entertainment-Bereich als auch im Bereich wissenschaftlicher Forschung wurden und werden Fahrsimulationen häufig umgesetzt und untersucht. Neben reiner Unterhaltungssoftware, also vielfältigen Computerspielen, finden sich auch semiprofessionelle oder professionelle Lösungen, beispielsweise als Ergometer zur Trainingsunterstützung von Radrennfahrern (vgl. dazu bspw. RacerMate (2006)). Auch wissenschaftliche Untersuchungen im Kontext der Streckengenerierung gibt es einige, als Beispiel sei hier auf Kaußner u. a. (2003) hingewiesen, wo eine Streckengenerierung unter fahrpsychologischen Aspekten entworfen wurde. Diese Untersuchungen hat Kaußner (2003) in seiner Dissertation fortgesetzt und dort ein skriptgesteuertes Modell vorgestellt, mit dem sich Straßennetzwerke erstellen und dynamisch anpassen lassen.

Die vorliegende Arbeit beleuchtet das Problemfeld unter dem Aspekt einer einzelnen, zu untersuchenden Straße und rückt deren dynamische und automatische Anpassung in den Mittelpunkt. Den Schwerpunkt bildet dabei die Erzeugung der Streckenführung an sich (vgl. Kapitel 5.1.3 und 5.2). Den zweiten Schwerpunkt bildet die optische Präsentation einer möglichst abwechslungsreichen virtuellen Umgebung. Dabei wird besonders die Einbettung der Strecke in ein iterativ zufallsgeneriertes Terrain nach dem Ansatz von Belhadj (2007) herausgestellt. Ergänzend wird eine Schnittstelle vorgestellt, mit deren Hilfe die Simulation von „außerhalb“ gesteuert werden kann.

## 1.2 Der Aufbau dieser Arbeit

Die vorliegende Belegarbeit soll den Entstehungsprozess der begleitend erstellten Simulation verdeutlichen. Dabei soll zuerst die Aufgabenstellung vorgestellt werden (Abschn. 2). Aus dieser resultieren dann gewisse Vorüberlegungen (Abschn. 4). Danach soll gezeigt werden, wie diese Vorüberlegungen in konkrete Konzepte umgesetzt wurden, die die theoretische Grundlage der Belegarbeit bilden (Abschn. 5). Im Anschluss daran wird die Implementierung und Umsetzung dieser Konzepte in Quellcode vorgestellt, sowie auf einige Probleme und deren implementatorische Lösung eingegangen werden (Abschn. 6). Der letzte große Teilbereich beschäftigt sich mit einer Analyse der Performanz und der Qualität der Applikation beziehungsweise der verwendeten Algorithmen (Abschn. 7). Ein Ausblick beschließt dann diese Ausarbeitung (Abschn. 8).

# 2 Die Vorstellung der Aufgabe

Das Ziel der Belegarbeit ist die Entwicklung eines Streckengenerators. Dieser soll in der Lage sein, eine Strecke zu erzeugen und deren Verlauf anhand dynamisch während der Simulation veränderbarer Parameter anpassen zu können. Diese Simulation soll interaktiv angelegt sein, das heißt, dass ein „Spieler“ oder „Fahrer“ in die Lage versetzt werden soll, die erzeugte Strecke virtuell abzufahren. Dazu sind verschiedene Teilaufgaben zu realisieren:

- Eine geeignete Literaturrecherche, beispielsweise bezüglich automatisiert generierter Parcours.

## 4 Vorüberlegungen

- Die Entwicklung des Streckengenerators.
- Die Implementierung des interaktiven Simulators zum Abfahren der generierten Strecken.
- Die stereoskopische Visualisierung auf der Powerwall des Lehrstuhls.
- Die Analyse der Arbeit hinsichtlich der erreichten Qualität.

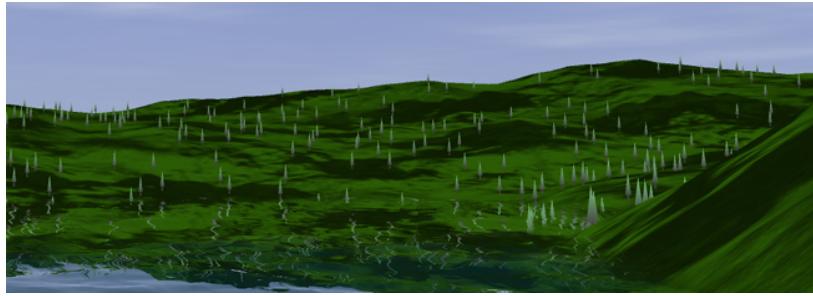
Zusätzlich zu diesen Mindestanforderungen wurden noch diverse optionale Punkte benannt, um die Qualität des Programms steigern zu können, wie beispielsweise die Anreicherung der Welt mit Effekten und „Gimmicks“ (Wind, Wetter, Bäume etc.), die Entwicklung einer Schnittstelle für die Steuerung der Software „von außen“ oder auch die Implementierung von Events, um den Wiederspielwert und die Motivation zu erhöhen.

## 3 Verwandte Arbeiten

Die Generierung von Fahrbahnen und Strecken sowie deren Einbettung in ein Terrain stehen meist im Kontext von Fahrsimulatoren beziehungsweise Simulationen. Beispielhaft für einen solchen Simulator soll hier der Fahrsimulator Universität Würzburg (2011) genannt werden. Dabei stehen im wissenschaftlichen Bereich häufig Probleme des Fahrers und des Fahrens im Mittelpunkt (beispielsweise Aufmerksamkeitsuntersuchungen beim Fahren, siehe Metz (2009)). In Kaußner u. a. (2003) wird eine Streckensimulation zur Bewältigung bestimmter Fahrsituationen vorgestellt. Die Auswahl der Streckenabschnitte erfolgt dabei anhand der konkreten Fahraufgabe durch einen Übungsleiter oder ein vorher festgelegtes Schema. Kaußner (2003) stellt ein allgemeineres Modell einer dynamischen Streckensimulation vor, mit dessen Hilfe sich auch komplexe Straßennetze und Verkehrssituationen modular zusammensetzen lassen. Die verwendeten Streckenmodule sind dabei allerdings bereits im Vorfeld der Simulation zu definieren. Im Bereich der Terraingenerierung sticht vor allem die Arbeit von Farès Belhadj und Pierre Audibert (Belhadj und Audibert (2005a), Belhadj und Audibert (2005b)) hervor, die ein Modell entwickelt haben, wie in ein zufallsgeneriertes Terrain beliebige Objekte eingebettet werden können. Dieses Verfahren wurde in der vorliegenden Arbeit übernommen.

## 4 Vorüberlegungen

Aus der Aufgabenstellung heraus ergeben sich zwei hauptsächliche Anforderungen an das Programm: einerseits die Modellierung eines dynamisch anpassbaren Straßenverlaufes, der in „Endlosschleife“ gefahren werden kann. Zum Zweiten ist es für die Motivation des „Fahrers“ beziehungsweise „Spieler“ notwendig, eine geeignete optische Präsentation zu geben, die Streckenführung also in den Kontext eines geeigneten Terrains zu stellen. Verschiedene Überlegungen liegen den einzelnen Entscheidungen bezüglich Design und Architektur im Hinblick auf diese zwei Anforderungen zugrunde. Diese wurden in die drei Bereiche „Die Auswahl der Terraingenerierungsmethode“, „Die Einbettung der Straße“ sowie „Streckenverlauf und Rasterung des Terrains“ gegliedert und sollen im Folgenden kurz vorgestellt werden, bevor im nachfolgenden Kapitel dann auf die Umsetzung der verwendeten Konzepte genauer eingegangen wird.



**Abbildung 1:** Computergeneriertes Terrain (Bild aus der Beleg-Simulation)

## 4.1 Die Auswahl der Terraingenerierungsmethode

Die Darstellung eines geeigneten Terrains stellt eine Kernkomponente der vorliegenden Arbeit dar. Es existieren dabei zwei grundlegende Möglichkeiten, diese Landschaften zu erstellen: im Vorfeld („per Hand“) oder „on the fly“ durch einen entsprechenden mathematischen Algorithmus. Beide Varianten haben Ihre Vor- und Nachteile:

Die Generierung „per Hand“ erfolgt im Vorfeld der Simulation. Dazu sind oft komplexe Editoren notwendig, um die entsprechenden Geometrien zu generieren. Diese können dann von nahezu beliebiger Komplexität<sup>1</sup> sein. Diese Art der Erstellung bietet dem Entwickler die größtmögliche Kontrolle über die gestaltete Landschaft, ist dabei allerdings enorm zeitintensiv. Darüber hinaus belegen die erzeugten Geometrien eine gewisse Menge Speicherplatz<sup>2</sup>. Damit Wiederholungen in der Landschaft minimiert werden, ist es erforderlich, eine sehr große Menge an Terrain im Vorfeld zu erzeugen, was einen enormen Aufwand bedeutet. Der wichtigste Punkt aber, der gegen diese Methode spricht, ist die mangelnde Flexibilität: wird das Terrain komplett im Vorfeld erzeugt, ist es nur sehr schwer möglich, im Nachhinein einen zufälligen Straßenverlauf darin zu platzieren, der auch noch dynamischen Parametern unterworfen sein soll.

Aus diesen Gründen wurde der zweiten Methode, der Generierung per Algorithmus, der Vorzug gegeben. Diese bietet eine schnelle, ressourcensparende Möglichkeit, wiederholungsfreie Landschaften zu erzeugen und gleichzeitig die notwendige Flexibilität bereitzustellen. Dem gegenüber steht allerdings eine verminderte Glaubwürdigkeit der Landschaften beim Betrachter<sup>3</sup> und - zumindest in der hier verwendeten Form - die Unmöglichkeit, mehrere „Terrainschichten“ übereinander darzustellen - Überhänge, Brücken, Höhlen oder Tunnel können mit der hier vorgestellten Software also nicht erstellt werden.

Für die algorithmische Erstellung von Terrains gibt es mehrere verschiedene Ansätze. Ein

<sup>1</sup> „Komplexität“ bedeutet hier, dass beispielsweise mehrere „Terrainschichten“ übereinander modelliert werden können (Tunnel, Brücken, etc.)

<sup>2</sup> Um so mehr, je komplexer die Geometrien werden.

<sup>3</sup> Natürlich existieren auch sehr ausgereifte Algorithmen, die künstliches Terrain einer Qualität erzeugen, das nur noch schwer bis gar nicht mehr von natürlicher Landschaft unterschieden werden kann (vgl. dazu z.B. PlanetSide (2010)) Die Entwicklung solcher Software ist allerdings hochkomplex und nicht Gegenstand der vorliegenden Arbeit. Des Weiteren erzeugen diese Generatoren im Normalfall keine Geometrie „on the fly“, sondern benötigen eine gewisse Erstellungszeit - Zeit, die in dieser Simulation nicht zur Verfügung stehen würde.

## 4 Vorüberlegungen

Vergleich der Algorithmen `TriangleEdge`, `DiamondSquare` und `Square-Square` findet sich bei Miller (1986). Wie von Martz (1996) ausgeführt, ist aber zumindest dessen Darstellung des `DiamondSquare`-Algorithmus fehlerhaft. Für diese Arbeit hat sich der `TriangleEdge`-Algorithmus als praktikabelste Variante erwiesen, da zwischen den einzelnen Iterationen keine Informationen ausgetauscht werden, der Algorithmus performant und sehr kompakt ist und die Unterteilung des Terrains in Dreiecke bereits eine Vorarbeit zur Generierung der Geometrie auf der Grafikkarte darstellt.

### 4.2 Die Einbettung der Straße

Es gibt grundsätzlich zwei Möglichkeiten, einen Straßenverlauf in ein Zufallsterrain einzubinden: entweder im Vorfeld der Terraingenerierung, oder erst im Nachhinein. Beide Varianten haben ihre Vor- und Nachteile:

Wird zuerst der Straßenverlauf generiert und danach das Terrain „herumgebaut“, ist die vollständige Kontrolle über die generierte Strecke gegeben. Das Terrain wird dabei lediglich als „Beiwerk“ betrachtet, welches sich in jedem Punkt dem Straßenverlauf anpassen soll. Im Gegensatz dazu steht die Einbettung der Straße in ein bereits vorher generiertes Terrain, wo der Straßenverlauf sich dem Terrain anpassen muss. Dabei ist allerdings zu beachten, dass eine Straße ja eine kontinuierliche Fläche darstellt, deren Parameter (Kurven, Steigung, Neigung etc.) wesentlich stärkeren Einschränkungen unterliegen, als mit der Zufallsgenerierung überhaupt berücksichtigt werden können. Dadurch würde eine, möglicherweise gegenseitige, nachträgliche Anpassung des Terrains an den Straßenverlauf (oder umgekehrt) dringend erforderlich. Andererseits könnte man auf diese Art auch Daten real existierender Landschaften einbinden.

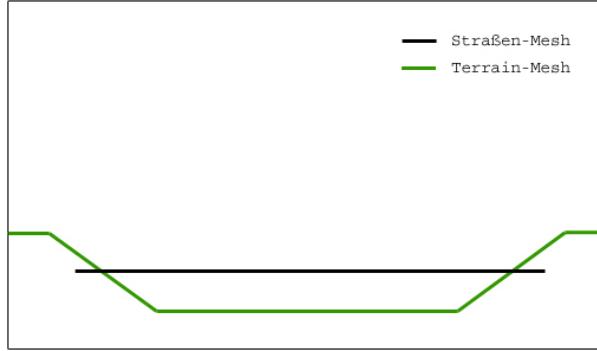
Diese Arbeit gibt der Generierung des Terrains „um die Straße herum“ den Vorzug. Diese Lösung ist aufgrund der exakten Trennung in zwei Phasen (Streckengenerierung vs. Terrain-generierung) erstrebenswert. Auf die Möglichkeit, real existierende Landschaften benutzen zu können, wurde explizit verzichtet. Des Weiteren wird sowohl die Implementation und Umsetzung vereinfacht, als auch eine Aufteilung der Landschaft in Zellen, wie sie im Folgenden vorgestellt wird, überhaupt erst ermöglicht.

Einfache Terraingenerierungsalgorithmen, wie der hier verwendete `TriangleEdge`-Algorithmus, arbeiten stets auf einem quadratischen, äquidistanten Punktgitter (für Details siehe Abschn. 5.1.1). Die Kurven des Straßenverlaufs können aus diesem Grund nicht direkt auf das Terrain-Mesh übertragen werden, da für eine Vermeidung eines stark sichtbaren Alias-Effektes eine extrem hohe Auflösung des Terrains erforderlich wäre. Statt dessen wird der Straßenverlauf als eigenes Mesh ein Stück „über“ das Terrain gelegt. Die Ränder der Straßeführung werden durch das Terrain-Mesh „hindurchgesteckt“, um so die Illusion eines einzelnen Blocks zu erzeugen (vgl. Abb. 2). Diese Lösung ist in der Computergrafik weit verbreitet, da sie schnell und einfach gute Resultate erzielt.

### 4.3 Streckenverlauf und Rasterung des Terrains

Ziel der Arbeit ist es, einen sich an bestimmte dynamisch veränderbare Parameter anpassenden Streckenzug zu generieren. Dabei soll der grundlegende Verlauf der Strecke zufällig erzeugt werden. Es ist offensichtlich, dass diese Anforderungen eine Generierung des kom-

### 4.3 Streckenverlauf und Rasterung des Terrains



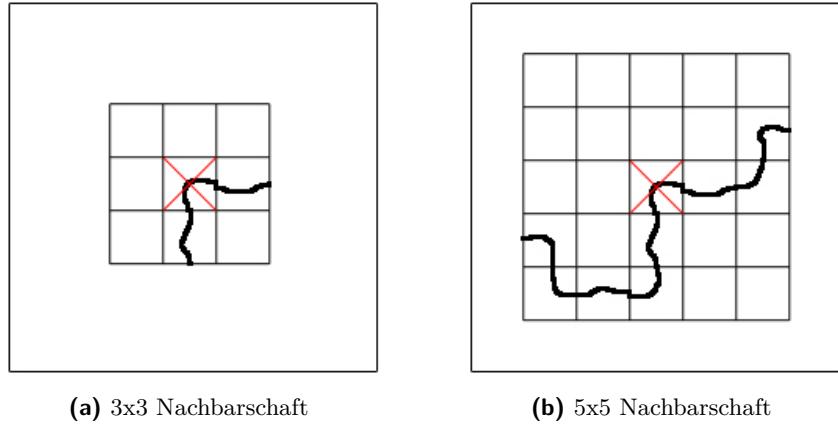
**Abbildung 2:** Schematische Darstellung der Meshes von Straße und Terrain

pletten Weges im Vorfeld unmöglich machen. Aus diesem Grund können jeweils nur die Teilabschnitte des Kurses erstellt werden, die gerade vom Fahrer in der Simulation befahren werden. Eine wichtige Einschränkung der Arbeit ist die Konzentration auf den Streckenverlauf an sich: da die Straße und ihr Verlauf im Mittelpunkt der Arbeit stehen, ist es nicht erforderlich, ein frei befahrbares Terrain umzusetzen.

Der hier verwendete Ansatz zur Umsetzung der Anforderungen sieht vor, das gesamte Terrain zu „kacheln“, also in schachbrettähnlich angeordnete, quadratische Zellen gleicher Größe zu unterteilen. Dabei wird zwischen „Terrainzellen“, also Zellen, in denen sich lediglich Terrain befindet, und „Straßenzellen“, also Zellen, in denen die Strecke verläuft, unterschieden. Die Generierung einer solchen Kachel ist sehr rechenintensiv und kann deshalb nicht in Echtzeit erfolgen (siehe Kapitel 7.1). Daher muss diese im Hintergrund parallel zur laufenden Simulation bewältigt werden. Die auf diese Art generierten Zellen werden dann so lange unsichtbar „in Reserve“ gehalten, bis sie das Sichtfeld des Fahrers betreten. Der Entwurf sieht weiterhin vor, dass der Fahrer sich lediglich entlang der Straße in den Straßenzellen bewegen darf. Auf beiden Seiten der Straße werden jeweils Terrainzellen dargestellt, um ein größeres Sichtfeld zu ermöglichen und damit das Trugbild einer durchgehenden Welt zu erzeugen. Es ist offensichtlich, dass diese Illusion um so überzeugender gelingt, um so mehr Terrain dabei gezeigt wird. Je mehr Fläche allerdings auf diese Art dem Sichtfeld des Fahrers hinzugefügt wird, desto mehr Zellen müssen auch im Vorfeld generiert werden. Dieses Verhalten widerspricht aber der Forderung nach einer möglichst dynamischen Anpassung der Strecke an bestimmte Parameter. Ein Kompromiss aus Sichtbereich und Anpassbarkeit des Streckenverlaufes muss also gefunden werden.

Das folgende Beispiel soll diesem Zusammenhang noch einmal verdeutlichen. Dabei wird von der Voraussetzung ausgegangen, dass eine Zelle des Terrains in etwa eine Kantenlänge von einem Kilometer besitzt. Abbildung 3 demonstriert jeweils zwei unterschiedliche Sichtbereiche des Fahrers (mit X markiert): links eine Sichtweite von einer Zelle, rechts von zwei Zellen in jeder Richtung. Somit ergibt sich zum einen eine 3x3 Nachbarschaft aus Zellen, zum anderen eine 5x5 Nachbarschaft. Der Streckenverlauf wurde jeweils willkürlich gewählt, richtet sich allerdings schon nach den in Abschnitt 5.2 näher beschriebenen Regeln.

Alle Zellen im Sichtbereich müssen bereits generiert worden sein, bevor sie aufgedeckt werden können. Für die 3x3 Nachbarschaft ergibt sich in dem Fall, dass in Fahrtrichtung der Verlauf von bereits zwei Straßenzellen vorbestimmt ist. Im Fall der 5x5 Nachbarschaft sind es hier bereits X Straßenzellen. Darüber hinaus müssen weitere Zellen im Vorfeld generiert



**Abbildung 3:** Sichtbereiche - das rote Kreuz markiert die Fahrerposition

werden, um ein nahtloses Umschalten bei Überschreitung der Zellgrenzen durch den Fahrer zu ermöglichen. Es ist offensichtlich, dass im Fall der 5x5 Nachbarschaft kaum noch von einer „dynamischen Anpassung“ gesprochen werden kann, da sich Änderungen an den Streckengenerierungsparametern erst sehr spät auf die generierte Strecke auswirken können. Aus diesem Grund wurde in dieser Arbeit der 3x3 Nachbarschaft der Vorzug gegeben, da sie den besten Kompromiss zwischen Sichtweite und Anpassbarkeit der Streckenparameter darstellt.

Dabei ist zu beachten, dass zu keinem Zeitpunkt geprüft wird, ob ein bestimmtes Feld der 3x3-Nachbarschaft überhaupt aus Fahrerperspektive sichtbar ist. Einerseits kann es nur in Ausnahmefällen dazu kommen<sup>4</sup>, dass ein bestimmtes Feld einer 3x3-Nachbarschaft von keinem Punkt der Fahrbahn aus gesehen werden kann. Zusätzlich dazu würde die Prüfung auf dieses Kriterium nicht unerheblich Rechenzeit benötigen<sup>5</sup>. Außerdem kann ein auf diese Weise „ausgeschlossenes“ Feld möglicherweise später doch durch die zufällige Streckengenerierung in den sichtbaren Nachbarschaftsbereich eines „späteren“ Straßenfeldes gelangen. Der Zeitgewinn, der durch die Nicht-Generierung der Zellgeometrie entstehen würde, steht dazu in keinem Verhältnis.

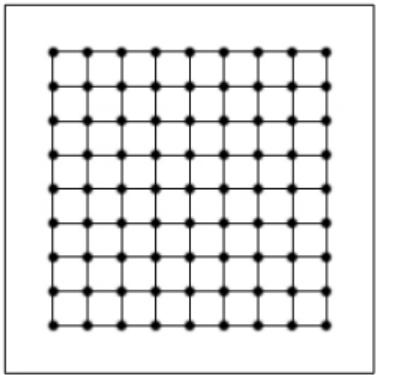
## 5 Die Vorstellung der Arbeit

Nachdem im vorangegangenen Kapitel auf bestimmte Vorüberlegungen eingegangen wurde, soll in diesem Abschnitt die Umsetzung dieser Überlegungen in konkrete Algorithmen besprochen werden. Zunächst werden dazu einige grundlegende Konzepte erläutert, bevor dann konkret der Ablauf der Straßen- und Zellgenerierung besprochen wird.

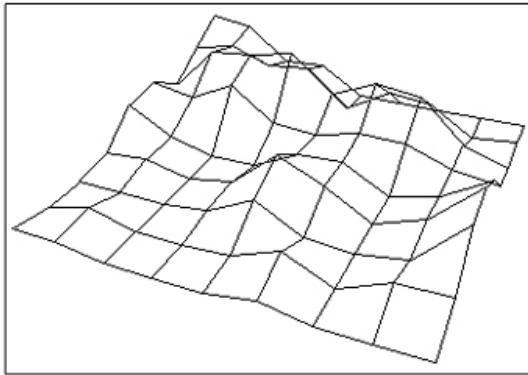
---

<sup>4</sup>Die Fahrt in einem mehrere Kacheln überspannenden Talkessel wäre eine solche Ausnahme.

<sup>5</sup>Für jeden Punkt der Straße müsste eine entsprechende „Sichtbarkeitsprüfung“ für alle Punkte der angrenzenden Felder durchgeführt werden.



(a) 9x9 Punktgitter in Draufsicht



(b) 3D-Ansicht eines 9x9 Punktgitters

**Abbildung 4:** Punktgitter - Draufsicht und 3D-Ansicht

## 5.1 Grundlegende Konzepte

Der Generierung von Straßen und Zellen liegen einige grundlegende Konzepte zu Grunde. Das Verständnis dieser Konzepte und ihrer Vor- und Nachteile ist essentiell, um im Anschluss daran eine genaue Betrachtung der Straßen- und Zellgenerierung durchführen zu können.

### 5.1.1 Der TriangleEdge-Algorithmus

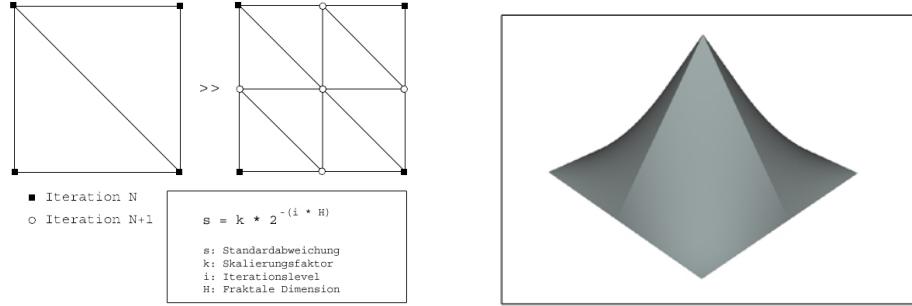
TriangleEdge ist ein seit mehreren Jahrzehnten bekannter, einfacher, rekursiver und fraktaler Terraingenerierungsalgorithmus. Er erzeugt, wie alle verwandten Algorithmen, ein gleichmäßiges, äquidistantes Punktgitter, dessen Ausmaße von der Rekursionstiefe abhängen. Die Punkte dieses Gitters bilden dabei die zu erzeugende Oberfläche und sind damit im eigentlichen Sinne Koordinaten im dreidimensionalen Darstellungsraum, also der Szene. Der Algorithmus bestimmt dabei lediglich die Y-Komponente, also die Höhe, des Punktes, während die X-Z-Koordinaten unangetastet bleiben. Das Aussehen eines solchen Gitters wird in Abbildung 4 dargestellt.

Der TriangleEdge-Algorithmus ist ein iterativer Algorithmus, der durch eine Erhöhung der Iterationsstufen in der Lage ist, eine immer feinere Auflösung des Terrains zu erzeugen. Als Input benötigt er lediglich die vier Eckpunkte der quadratischen Terrainfläche und deren initiale Höhen. Anhand dieser Informationen wird die Fläche in jedem Schritt in weitere Vierecke unterteilt, bis die gewünschte Iterationstiefe erreicht wird. Abbildung 5a zeigt diesen Vorgang.

Die Koordinaten der im jeweiligen Iterationsschritt neu erzeugten Punkte werden dabei lediglich aus der jeweiligen direkten „Elterngeneration“ gebildet. Die X- und Z-Komponente entstehen durch einfache „Halbierung“ der entsprechenden Koordinatenkomponenten der Eltern, während die Y-Komponente eine Interpolation der Höhen der Eltern darstellt. Diese Interpolation findet wie folgt statt:

Punkte auf den Kanten des neu entstehenden Teilvierecks werden durch die jeweiligen, ebenfalls auf dieser Kante liegenden Elternpunkte interpoliert. Der Mittelpunkt hingegen

## 5 Die Vorstellung der Arbeit



(a) Schematische Darstellung der Funktion des TriangleEdge-Algorithmus (vgl. Miller (1986))  
(b) TriangleEdge-Verfahren mit initial gesetztem Mittelpunkt

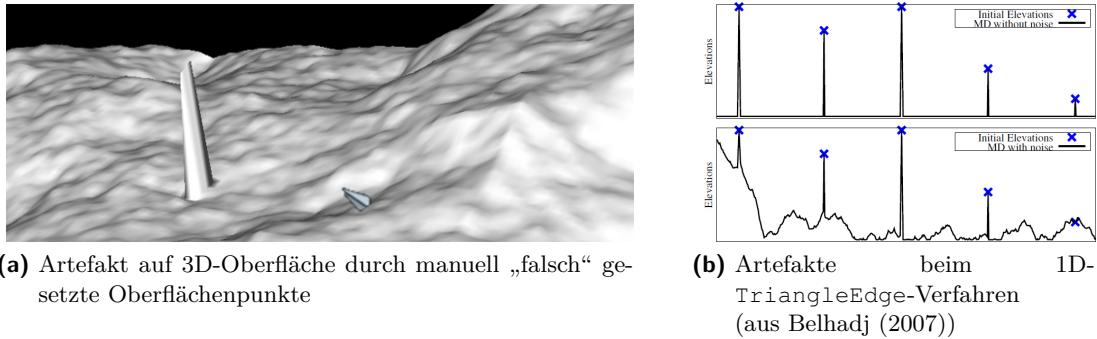
**Abbildung 5:** Der TriangleEdge-Algorithmus

entsteht durch Interpolation aller vier Eckpunkte des Elternvierecks<sup>6</sup>. Findet dabei eine reine Interpolation ohne Einfluss einer weiteren Größe statt, entstehen vollständig regelmäßige Oberflächen. Diese haben allerdings noch nichts mit fraktalen Terrains zu tun. Um diese zu erzeugen, wird ein zusätzliches Displacement eingeführt. Dieses Displacement bezieht sowohl die fraktale Dimension als auch die Iterationstiefe mit ein (vergleiche die Formel in Abb. 5a).

Zusätzlich dazu wird noch ein Zufallswert eingeführt, mit dessen Hilfe das Terrain seine Regelmäßigkeit verliert. Durch die fraktale Natur des Algorithmus wird erreicht, dass die Zufallswerte in frühen Iterationsstufen zu einem starken Displacement führen, in späteren hingegen nur noch geringen Einfluss haben. Somit wird das Terrain abhängig von der Parameterwahl global in Senken und Höhen unterteilt, während im lokalen Bereich um einen Punkt herum sich relativ gleichmäßige gestaltete, „glatte“ Flächen ergeben. Dieses Prinzip könnte auch als „Zufall im Globalen, Interpolation im Lokalen“ bezeichnet werden. Es führt dazu, dass die entstehenden Terrains überhaupt erst als „glaubwürdig“ angesehen werden können. Dabei ist allerdings zu beachten, dass auf diese Art und Weise lediglich unter bestimmten Bedingungen Landschaften ohne Artefakte entstehen können. „Artefakte“ bezeichnet hier alle optisch wahrnehmbaren „Störungen“ einer ansonsten „glaubwürdigen“ Oberfläche (vgl. Grafik 6a). Es ist offensichtlich, dass zum Start der Interpolationen mindestens die vier Eckpunkte als initiale „Constraints“, also Initialbedingungen, angegeben werden müssen. Wird, wie auch in Abbildung 5b zu sehen, auch der Mittelpunkt auf einen Initialwert gesetzt, entstehen ebenfalls artefaktfreie Oberflächen. Anders sieht es aus, wenn beliebige, zusätzliche Punkte initial gesetzt werden. Abbildung 6b, entnommen aus Belhadj (2007), verdeutlicht dieses Problem für den eindimensionalen Fall.

Belhadj (2007) kommt zu dem Schluss, dass immer genau dann Artefakte entstehen, wenn die Höhen von Kindpunkten bekannt sind, die ihrer erzeugenden Elternpunkte hingegen nicht Zur Verdeutlichung: das initiale Setzen der Höhe des Mittelpunktes der Landschaft ist problemlos möglich, da die Höhen aller erzeugenden Elternpunkte (die Eckpunkte) ebenfalls

<sup>6</sup>Dabei wird jeweils linear auf den Kanten der Diagonalen interpoliert und von beiden Ergebnissen der Mittelwert berechnet.



**Abbildung 6:** Verschiedene Artefakte

initial bekannt sind. Das Setzen der Höhen anderer Punkte wird genau dann zum Problem, wenn nicht die Höhen aller Elternpunkte ebenfalls gesetzt werden.

Für diese Belegarbeit ist es allerdings von essentieller Wichtigkeit, dass zu Beginn der Terraingenerierung beliebige Constraints gesetzt werden können, da unter anderem ja die Straße in des Terrain eingebettet werden muss. Im folgenden Abschnitt soll eine Lösung dieses Problems aufgezeigt werden.

### 5.1.2 Der TriangleEdge-Algorithmus mit Constraints

Wie bereits im letzten Abschnitt besprochen, funktioniert die Generierung von Zufallsterains mittels des normalen TriangleEdge-Verfahrens nur dann artefaktfrei, wenn die initialen Startwerte sehr strengen Bedingungen unterliegen. Da in dieser Belegarbeit aber unter anderem der Straßengrund als Menge an Constraints übergeben werden soll, muss zwingend eine Lösung gefunden werden, die ein zufallsgeneriertes Terrain mit beliebigen Vorbedingungen kombiniert. In Belhadj (2007) und Belhadj und Audibert (2005a) beziehungsweise Belhadj und Audibert (2005b) wurde eine solche Lösung präsentiert, die in leicht abgewandelter Form auch für diese Arbeit übernommen wurde.

Die Überlegungen in Belhadj (2007) beruhen dabei auf der Erkenntnis, dass Constraints immer dann artefaktfrei benutzt werden können, wenn die jeweiligen Elternpunkte ebenfalls bekannt sind, also ebenfalls in der Menge der Vorbedingungen enthalten sind. Dieser Fall tritt allerdings unter normalen Bedingungen aufgrund der Arbeitsweise des TriangleEdge-Algorithmus eher selten auf. Die Idee von Belhadj (2007) ist es nun, aus den vorhandenen Constraint-Punkten in einer Vorverarbeitung alle erzeugenden Elternpunkte zu bestimmen und der Menge an Vorbedingungen hinzuzufügen. Diese derart erweiterte Constraint-Menge kann dann problemlos dem klassischen TriangleEdge-Verfahren übergeben werden, welches dann die größtmögliche Artefaktfreiheit garantiert<sup>7</sup>.

<sup>7</sup>Natürlich können immernoch Artefakte produziert werden, indem einfach Constraints eingegeben werden, die von dem Algorithmus zu keiner „glatten“ Oberfläche mehr verarbeitet werden können. Zwei benachbarte Punkte auf sehr unterschiedliche Höhen festzulegen wäre ein Beispiel für dieses Verhalten. In diesem Fall würde allerdings jeder mögliche iterative Algorithmus versagen und entsprechende Artefakte bilden.

## 5 Die Vorstellung der Arbeit

Dieser vorverarbeitende Prozess wird bei Belhadj (2007) MDBU - Midpoint Displacement Bottom Up - genannt. Bevor näher auf den Prozess als solches eingegangen wird, soll zuerst eine andere Anpassung beschrieben werden, die Belhadj (2007) eingeführt hat. Die Interpolation eines Kindpunktes aus seinen Elternpunkten erfolgt im klassischen TriangleEdge-Verfahren als die Bestimmung des arithmetischen Mittels der Höhen der Elternpunkte, plus eines eventuellen randomisierten Displacements (siehe Abschnitt 5.1.1). Belhadj (2007) verwendet nun statt des arithmetischen Mittels eine gewichtete Funktion, bei der das Ergebnis der Berechnung von der Distanz zwischen Kind- und Elternpunkt abhängt (siehe Formel in Abb. 7)

$$\Delta(e, d) = e \times (1 - \sigma(I) \times (1 - (1 - \frac{d}{d_{max}})^{|I|}))$$

$$\sigma(I) = \begin{cases} 1 & I \geq 0 \\ -1 & I < 0 \end{cases}$$

**Abbildung 7:** Angepasste Formel des TriangleEdge-Verfahrens (aus Belhadj (2007))

Dabei bezeichnet  $\Delta(e, d)$  die gewichtete Höhe abhängig von der Distanz  $d$  zwischen Kind- und Elternpunkt und der Höhe  $e$  der Elternzelle.  $d_{max}$  bezeichnet die Diagonale der Zelle an sich. Zusätzlich wurde eine Variable  $I$  eingeführt, die zur Anpassung der Interpolationskurve genutzt werden kann:

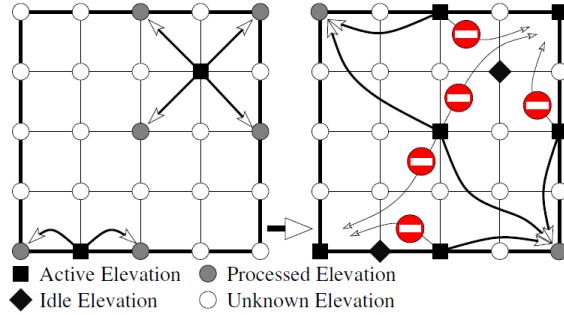
„Here [...]  $I$  is used to tune the interpolation curve; depending on  $I$  we have:  
 $\Delta(e, d) = e$ , when  $I = 0$ , otherwise  $\Delta(e, d)$  respectively decreases or increases according to the distance  $d$  when  $I > 0$  or  $I < 0$ ; it respectively follows a bell curve, a line or a ridged curve when  $0 < |I| < 1$ ,  $|I| = 1$  or  $|I| > 1$ .“  
(Zitat aus Belhadj (2007) zur Verwendung des Parameters  $I$ )

Dieser Algorithmus führt zur besseren Kontrolle des Terrains und der Erhöhung der „Natürlichkeit“ des Terrains und somit dessen Glaubwürdigkeit. Er bietet zudem den Vorteil, dass auch Aussagen über die Höhen von Eltern- bzw. Kindpunkten gemacht werden kann, wenn nur jeweils ein Elternpunkt betrachtet wird bzw. bekannt ist. Diese Eigenschaft spielt für den MDBU-Prozess eine große Rolle. Dieser Prozess basiert, wie bereits ausgeführt, auf der Idee, unbekannte Elternpunkthöhen durch bekannte Kindpunkthöhen „rückwärts“ zu interpolieren. Die für jeden Punkt relevanten Informationen sind dabei seine Höhe, also seine Y-Komponente<sup>8</sup>, sowie sein Status(„unbekannt“ beziehungsweise „bekannt“).

Der Prozess gliedert sich in zwei Phasen. Abbildung 8 aus Belhadj (2007) verdeutlicht diesen Prozess. In der ersten Phase werden alle bekannten Constraints als Kindpunkte betrachtet und ihre jeweiligen Elternpunkte identifiziert. Alle Elternpunkte, deren Status „bekannt“ ist, werden in eine gesonderte Liste eingetragen. Die zweite Phase verarbeitet

---

<sup>8</sup>Die X und Z-Komponenten sind ja durch seine Lage im Zellgitter fest vorgegeben und dienen nur seiner eindeutigen Identifikation im Gitter.



**Abbildung 8:** Schematische Darstellung des MDBU Prozesses (aus Belhadj (2007))

nun die Punkte in dieser Liste. Dabei werden abhängig von der Menge der bekannten Kindpunkte die Höhen der Elternpunkte entsprechend der Abbildung 7 interpoliert. Bei allen so erhaltenen Punkten wird der Status auf „bekannt“ gesetzt und sie werden in die Liste der Constraints eingetragen. Mit dieser Liste wird der MDBU-Prozess nun erneut gestartet und so oft wiederholt, bis keine neuen Elternpunkte mehr identifiziert werden können. Das ist genau dann der Fall, wenn die vier Eckpunkte als Eltern identifiziert werden. Da diese auf jeden Fall eine bekannte Höhe aufweisen und keine weiteren Elternpunkte mehr besitzen, bricht der Algorithmus in jedem Fall nach endlicher Zeit ab. Aus den anfänglich nur zwei gegebenen Constraints sind zehn geworden. Diese garantieren nun die größtmögliche Artefaktfreiheit bei dem sich anschließenden TriangleEdge-Verfahren.

Diese Belegarbeit verwendet diese Lösung von Belhadj (2007) mit zwei Abwandlungen. Einerseits wird die gewichtete Interpolation nur im BottomUp-Prozess benutzt. Das anschließende TriangleEdge-Verfahren benutzt dabei den klassischen Interpolationsalgorithmus. Zum anderen wird im MDBU-Prozess kein Displacement verwendet, da dieses bereits im normalen TriangleEdge-Algorithmus zur Anwendung kommt und dort zur Generierung glaubhafter, zufälliger Terrains vollständig ausreicht.

Im folgenden Abschnitt soll nun darauf eingegangen werden, wie der Streckenverlauf an sich erstellt wird.

### 5.1.3 Modellierung des Straßenverlaufs

Der Streckenverlauf mitsamt seiner dynamisch anpassbaren Parameter bildet das Kernstück der Belegarbeit. Es konnten folgende Anforderungen identifiziert werden, welche die Generierung des Straßenverlaufs erfüllen muss:

- Die Generierung muss abschnittsweise bzw. in Teilstrecken erfolgen können.
- Es müssen Parameter einstellbar sein, die das Vorhandensein von Kurven, Steigungen etc. beeinflussen.
- Die abschnittsweise Erstellung muss mit dem Konzept der Terrainzellen zusammenarbeiten.
- Der erstellte Streckenzug soll so natürlich wie möglich ausfallen.

## *5 Die Vorstellung der Arbeit*

Drei verschiedene Möglichkeiten wurden dazu im Laufe der Arbeit untersucht bzw. teilweise umgesetzt:

- a) Die Verwendung von per Hand vormodellierten Teilstücken
- b) Die Umsetzung von in der Realität benutzten Straßenbauvorschriften
- c) Die Modellierung des Streckenzuges als Bézierkurve

Es soll nun kurz auf die Vor- und Nachteile dieser Lösungen eingegangen werden.

### **a) Die Verwendung von per Hand vormodellierten Teilstücken**

Die Benutzung von vormodellierten Teilstücken bietet auf den ersten Blick scheinbar viele Vorteile: sie ist einfach zu implementieren, es ist kein Rechenaufwand zur Berechnung des Streckenzuges nötig, die Modellierung per Hand gibt größtmögliche Freiheiten und diese Lösung passt perfekt mit der Idee der schachbrettartig angeordneten Terrainzellen zusammen. Die Speicherung und das Laden einer so zusammengebauten Szene ist ebenfalls recht trivial, da theoretisch nur die Folge der verwendeten Teilstücke gespeichert werden muss.

Im Hinblick auf zukünftige Erweiterungen könnte ein recht einfach zu bedienender Editor erstellt werden, mit dem Teilzüge oder auch komplett Streckenverläufe vordefiniert werden könnten. Editoren dieser Art und Funktionalität sind beispielsweise auf dem Computerspielsektor schon seit sehr langer Zeit bekannt.

Andererseits ist diese Lösung auch extrem unflexibel, da im Vorfeld in jedem Fall nur ein endlich großer Pool an Teilstücken vorgefertigt werden kann. Dies führt zu starken Einschränkungen sowohl bei dem erzeugten Realismusgrad (durch Wiederholung bzw. Langeweile) wie auch zu einem relativ großen Speicherplatzbedarf, da jedes Teilstück als eigenes Modell in einer Bibliothek vorrätig gehalten werden muss.

Diese mangelnde Flexibilität ist allerdings im Hinblick auf die größtmögliche Dynamik und Anpassbarkeit des Streckenverlaufs als zu stark einschränkend bewertet worden. Das Konzept wurde deshalb verworfen.

### **b) Die Umsetzung von in der Realität benutzten Straßenbauvorschriften**

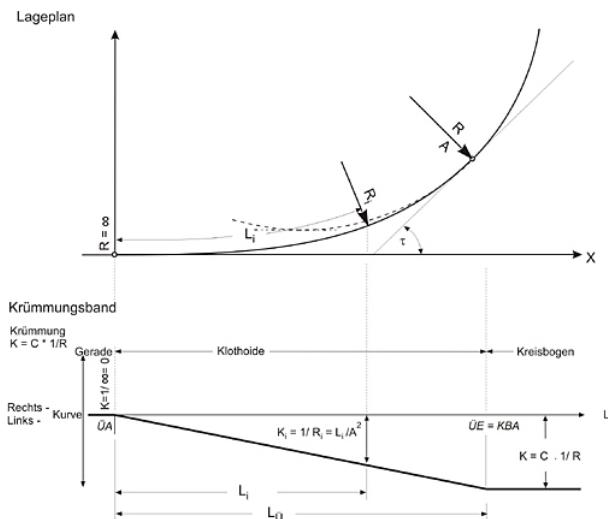
Die naheliegendste Möglichkeit, als realistisch empfundene Straßenzüge zu generieren, besteht wohl in der Verwendung von Bauvorschriften, wie sie in der Realität verwendet werden. Eine Vielzahl an Normen und Vorschriften sind dazu in Deutschland erlassen worden. Einen kurzen Überblick dazu bietet WIKIPEDIA (2010), ohne Anspruch auf Vollständigkeit zu erheben.

Auch wenn alle für die Belegarbeit irrelevanten Vorschriften ausgeklammert werden, verbleiben immer noch weit zu viele, um sie im Rahmen eines Beleges korrekt umsetzen zu

können. Deshalb lohnt es sich, nur einige wenige relevante Aspekte des Straßenbaus an sich zu isolieren:

Da in dieser Arbeit weder auf Gegenverkehr, noch auf Abzweige, Einmündungen oder Ähnliches eingegangen werden soll, verbleiben lediglich zwei Komponenten, die einen Straßenzug ausmachen: gerade Teilstücke sowie Kurven beliebiger Krümmung. Die Umsetzung der geraden Teilstücke ist dabei trivial, die Gestaltung von Kurvensegmenten erfordert allerdings eine genauere Betrachtung.

Die einfachste Lösung wäre sicherlich das „Anstecken“ einer Kurve bestimmter Krümmung an ein Geradensegment. Leider entsteht dadurch ein unsauberer Übergang zwischen beiden Segmenten, da sich die Krümmung abrupt ändert. Dieses Verhalten würde vom Fahrer verlangen, dass er beim Überfahren der Segmentgrenze seinen Lenker sofort (idealerweise ohne Zeitverzug) auf die für das Durchfahren der Kurve „korrekte“ Krümmung hin einschlägt. Das ist in der Realität natürlich nicht umsetzbar. Zusätzlich zu der in dem Fall ebenfalls abrupt auftretenden Querbeschleunigung würde ein „Ausdriften“ an den äußeren Straßenrand stattfinden. Deshalb wurde als Verbindungsstück zwischen Gerade und Kurve im Straßenbau die Verwendung von sogenannten Klohoiden (vgl. Abb. 9) eingeführt. Eine Klohoide ist ein Straßensegment, dessen Krümmung linear von Null zur gewünschten Krümmung, beispielweise der anschließenden Kurve, ansteigt. Dadurch wird gewährleistet, dass der Übergang zwischen Gerade und Kurve möglichst „sanft“ verläuft und sowohl Lenkeinschlag als auch Querbeschleunigung ebenfalls nur linear, statt abrupt erhöht werden.



**Abbildung 9:** Schematische Darstellung einer Klohoide mitsamt Krümmungsband (aus TU Dresden (2010))

Es ist leicht ersichtlich, dass für jede Kurvenkrümmung verschiedene Klohoidenradien erforderlich sind. Auch ist der Übergang von Klohoide bzw. Kurve zu Gerade wieder nur über eine zweite Klohoide gegenläufiger Krümmung möglich. Diese Einschränkungen stehen in starkem Kontrast zu der Forderung, dass der Straßenverlauf zum „Terrain-Schachbrett“ kompatibel sein soll, da im Vorfeld nicht klar abgegrenzt werden kann, ob die Strecke noch

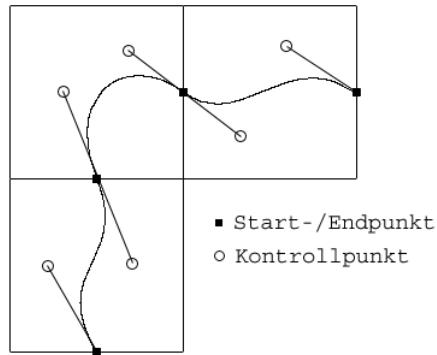
## 5 Die Vorstellung der Arbeit

innerhalb der vorgesehenen Zellen verläuft oder andere, nicht vorgesehene Zellgrenzen überschreitet. Deshalb wurde auch diese Lösung zugunsten der im Folgenden besprochenen Bézierkurve verworfen.

### c) Die Modellierung des Streckenzuges als Bézierkurve

Die letzte untersuchte Möglichkeit besteht in der Verwendung mathematischer Funktionen zur Modellierung der Kurve. Diese bieten den Vorteil, einfach, schnell und robust berechnet und implementiert werden zu können. Da das Terrain lokal aus Zellen aufgebaut wird, können folgende Teilprobleme identifiziert werden:

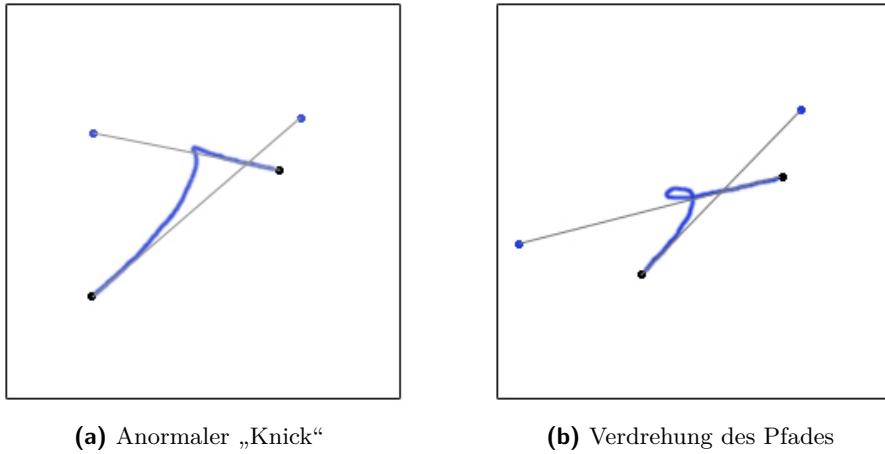
- Wie kann innerhalb der Straßenzelle eine Kurve definiert werden?
- Wie sehen die Übergänge zwischen zwei Straßenzellen aus?
- Wie können die geforderten, dynamisch anpassbaren Parameter einbezogen werden?



**Abbildung 10:** Schematische Darstellung des Straßenverlaufs als Bézierkurve (Draufsicht)

Diese Anwendung benutzt zur Definition der Streckenabschnitte Bézierkurven dritten Grades. Diese stellt die „Mittellinie“ der Straße dar, die sich dann auf einer gewissen Breite erstreckt. Es existiert auf zwei jeweils verschiedenen Seiten einer Straßenzelle ein Eintritts- beziehungsweise Austrittspunkt sowie zwei Kontrollpunkte innerhalb der Zelle. (vgl. Abb. 10)

Die Eigenschaften der Bézierkurve bringen es mit sich, dass Ein- und Austrittspunkt interpoliert werden, also auf der Straße liegen. Im Gegensatz dazu liegen die Kontrollpunkte in der Regel nicht darauf. Die Lage der Punkte kann dabei nicht beliebig zufällig gewählt werden. Liegen Ein- oder Austrittspunkt zu nahe an den Zellrändern, dann ragt der Straßenkörper möglicherweise ein Stück weit in eine benachbarte Terrainzelle hinein. Dieses Verhalten würde die klare Trennung zwischen Straßen- und Terrainzellen verwischen und ist deshalb zu vermeiden. Weitere Probleme können sich bei ungünstiger Wahl der Lage der Kontrollpunkte relativ zueinander und relativ zu dem Endpunkten ergeben. Dann entstehen unnormal scharfe Kurven, Verdrehungen oder ähnliche unerwünschte Effekte (siehe Grafik 11)



**Abbildung 11:** Zwei mögliche „Anomalitäten“ bei Bézierskurven

Es ist äußerst wichtig, dass die Straße über Zellgrenzen hinweg nahtlos weitergeführt werden kann. Dabei soll dem Anwender gar nicht bewusst werden können, dass er gerade eine solche Grenze überfahren hat. Dazu ist es notwendig, einige Randbedingungen an die Bézierskurven zweier benachbarter Zellen zu stellen:

- Der Austrittspunkt der einen Zelle muss gleichzeitig der Eintrittspunkt der nachfolgenden sein.
- Der Anstieg der Tangente an Aus- und Eintrittspunkt muss übereinstimmen. Dadurch sind beide Straßen gleich orientiert.
- Die Länge der Strecke vom zweiten Kontrollpunkt zum Austrittspunkt der ersten Zelle muss gleich der Länge der Strecke vom Eintrittspunkt zum ersten Kontrollpunkt der Folgezelle sein. Auf diese Art haben beide Straßen an der Schnittstelle die gleiche Krümmung.

Diese drei Bedingungen müssen erfüllt sein, damit ein nahtloser Übergang zwischen zwei Zellen erfolgen kann. Dabei wird deutlich, dass die beiden Kontrollpunkte nicht beliebig in der Zelle verteilt werden können, sondern immer die Lage des ersten Kontrollpunktes einer Zelle von der Lage des letzten Kontrollpunktes der Vorgängerzelle bestimmt wird. Es sind dabei Szenarien zu vermeiden, bei denen die Kontrollpunkte aufgrund dieser Abhängigkeit außerhalb der zugehörigen Straßenzelle gesetzt werden müssen.

Aufgrund dieser Rahmenbedingungen wurden mehrere Einschränkungen bezüglich der Lage der Rand- und Kontrollpunkte getroffen. Die Lage der Start- und Endpunkte innerhalb einer Straßenzelle wird immer auf den Mittelpunkt des jeweiligen Zellrandes gelegt. Auf diese Art und Weise müssen keine Informationen bezüglich dessen Lage zwischen zwei benachbarten Zellen ausgetauscht werden. Außerdem sorgt diese Anordnung in einem ersten Schritt dafür, dass der Streckenzug die Zellgrenzen nicht verlässt, da Anfangs- und

## 5 Die Vorstellung der Arbeit

Endstücke dieser Strecke zwingend wieder in die Zellmitte zurückgeführt werden<sup>9</sup>.

Auf der anderen Seite wurde auch die Lage der Kontrollpunkte stark eingeschränkt. Die Berechnung des zweiten Kontrollpunkts<sup>10</sup> folgt folgender Vorschrift:

Es wird ein Zufallswert zwischen 0.0 und 1.0 mittels einer Normalverteilung berechnet, deren Erwartungswert bei 0.5 liegt und deren Streuung die Kurvigkeitsparameter (vgl. Abschn. 5.1.4) darstellt. Der so erhaltene Wert mit mit einem Viertel der gesamten Zellbreite multipliziert und auf der Linie Zellmittelpunkt-Endpunkt vom ein Viertel Zellenbreite vom Mittelpunkt weg abgetragen. Der so erhaltene Punkt wird um einen Winkel  $\alpha$  um den Endpunkt gedreht. Dieser Winkel liegt im Intervall von  $-30^\circ$  bis  $+30^\circ$  und wird ebenfalls zufällig bestimmt. Dazu wird ein Zufallswert aus einer Normalverteilung (Erwartungswert: 0.0, Streuung: Kurvigkeitsparameter) in den Grenzen -1.0 bis +1.0 ermittelt, der mit den  $30^\circ$  verrechnet wird. Das Ganze nochmal etwas kompakter gefasst:

```
berechne maximale Verschiebung: Viertel der Zellbreite;  
berechne Zufallszahl: Normalverteilung (Grenzen 0 bis 1, Erwartungswert 0.5,  
Streuung Kurvigkeitsparameter);  
berechne Verschiebung: maximale Verschiebung * Zufallszahl + maximale  
Verschiebung;  
berechne Zwischenpunkt: Endpunkt + Verschiebung;  
berechne Drehung: Normalverteilung (Grenzen  $-30^\circ$  bis  $+30^\circ$ , Erwartungswert  $0^\circ$ ,  
Streuung Kurvigkeitsparameter *  $30.0$ );  
berechne Kontrollpunkt: drehe Vektor Endpunkt->Zwischenpunkt;
```

**Listing 1:** Anweisungen zur Erstellung des zweiten Kontrollpunktes

Bis jetzt wurden die Verschiebungen der Punkte lediglich für die XZ-Ebene betrachtet. Sowohl für den End- als auch für den zweiten Kontrollpunkt können aber die Y-, also Höhenwerte angepasst werden. Zwei Parameter stehen dafür zur Verfügung: die Hügeligkeit und die Steilheit (vgl. Abschn. 5.1.4). Die Steilheit hat Einfluss auf die Höhenverschiebung des Endpunktes: der userdefinierte Steilheitsfaktor wird mit einem Zufallswert multipliziert. Auch dieser wird über eine Normalverteilung (Erwartungswert: Steilheitsparameter, Streuung: 0.1) in den Grenzen -1.25 bis +1.25 gewonnen und stellt die Verschiebung des Endpunktes gegenüber der Höhe des Startpunktes dar.

Die Hügeligkeit hingegen ist für den Höhenversatz des zweiten Kontrollpunktes zuständig. Er wird analog zur Endpunkthöhe aus einer Normalverteilung (Erwartungswert: Hügeligkeitsparameter, Streuung: 0.1) in den Grenzen von -1.25 bis +1.25 gewonnen.

Diese Einschränkungen beziehungsweise die Werte für die Zufallsverteilungen wurden mehr oder weniger willkürlich getroffen, sind aber dennoch aus einer Reihe Tests mit verschiedenen Werten als günstig hervorgegangen. Sie gewährleisten einerseits verdrehungs- und artefaktfreie Streckenverläufe sowie „glaubwürdige“ Kurven, bieten aber andererseits auch ein breites Spektrum an möglichen Straßenverläufen. Über die Vielzahl einstellbarer

<sup>9</sup>Aufgrund der Anfangs- und Endpunktinterpolation bei Bézierkurven.

<sup>10</sup>Wie bereits gezeigt wurde, ergibt sich die Lage des ersten Kontrollpunktes ja stets aus dem zweiten Kontrollpunkt der vorhergehenden Zelle.

Parameter ist darüberhinaus die dynamische Anpassung des Streckenverlaufs sichergestellt. Von den ursprünglich zwölf Freiheitsgraden<sup>11</sup> sind also am Ende vier übrig geblieben: die Lage des zweiten Kontrollpunktes sowie die Höhenkomponente des Endpunktes.

Eine ähnliche, grundlegend gleichwertige Alternative würde die Verwendung von Splines oder B-Splines darstellen. Da im Gegensatz zu Bézierkurven bei Splines allerdings jeder Punkt interpoliert wird, müssten gänzlich andere Einschränkungen bezüglich der Punktlagen innerhalb der Zelle getroffen werden. Die einfache Handhabung und Zeichenbarkeit von Bézierkurven hat gegenüber einer Lösung mit Splines den Ausschlag gegeben.

An dieser Stelle existiert jetzt lediglich eine idealisierte Streckenführung innerhalb der Straßenzelle, repräsentiert durch die End- und Kontrollpunkte der entsprechenden Bézierkurve. In Abschnitt 5.3.2 wird später erläutert, wie diese Kurve dann auf das konkrete Zellgitter abgebildet wird. Nachdem allerdings bereits einige der benutzerdefinierten beziehungsweise dynamisch anpassbaren Parameter angesprochen wurden, soll im folgenden Abschnitt zuerst einmal eine Übersicht gegeben werden, welche Parameter existieren und wozu sie verwendet werden.

#### 5.1.4 Parameter für die Streckenanpassung

Aus den vier verbliebenen Freiheitsgraden bei der Verwendung einer Bézierkurve mit den beschriebenen Beschränkungen wurden drei verschiedene Parameter erzeugt, um die Streckengenerierung sowohl statisch<sup>12</sup> als auch dynamisch anzupassen:

**Steilheit** Die Steilheit bestimmt den Höhenversatz zwischen dem Start- und Endpunkt einer Zelle. Sie ist dabei ein *relativer* Wert und wird als Erwartungswert einer Normalverteilung betrachtet, deren Streuung auf 0.1 festgelegt wurde. Auf diese Art und Weise lässt sich auf lange Sicht eine recht zuverlässige Prognose über den globalen Höhenverlauf erstellen, für die konkrete Zelle an sich kann es allerdings dennoch zu gewissen zufälligen Abweichungen kommen (vgl. Abschn. 7.2). Die *absolute* Steilheit einer gegebenen Zelle wird zusätzlich davon durch den statischen Parameter **Steilheitsfaktor** bestimmt.

**Hügeligkeit** Die Hügeligkeit bestimmt den Höhenversatz des zweiten Kontrollpunktes gegenüber des linearen An- bzw. Abstiegs von Start- zu Endpunkt einer Zelle. Je höher dieser Wert ausfällt, um so „welliger“ erscheint das generierte Terrain. Auch dieser Wert ist, analog zur Steilheit, lediglich der Erwartungswert einer entsprechenden Normalverteilung, also ein *relativer* Wert. Der absolute Versatz wird zusätzlich durch den statischen Parameter **Hügeligkeitsfaktor** bestimmt.

**Kurvigkeit** Die Kurvigkeitswerte werden an mehreren Stellen im Programm genutzt, um den Streckenverlauf zu bestimmen. Als einziger der Parameter findet er nicht nur Anwendung im

---

<sup>11</sup>Die Raumkoordinaten aller vier Punkte der Bézierkurve.

<sup>12</sup>„statisch“ bedeutet hier, dass eine Anpassung der Parameter im Vorfeld erfolgt, die während der laufenden Simulation nicht mehr geändert werden.

## 5 Die Vorstellung der Arbeit

`TerrainGenerator`, sondern wird auch im `TerrainManager` zur Generierung der globalen Streckenführung herangezogen (vgl. Abschnitt 5.2). Im `TerrainGenerator` wird über zwei Zufallsberechnungen (vgl. Abschn. 5.1.3) der Verlauf der Bézierkurve bestimmt, die den Straßenverlauf modelliert. Je größer dabei der Parameter gewählt wird, desto weiter liegen die beiden Kontrollpunkte potentiell auseinander<sup>13</sup>. Im `TerrainManager` bestimmt die Kurvigkeitsfaktor hingegen die Häufigkeit von Kurven im globalen Straßenverlauf.

Neben diesen drei dynamischen Parametern existieren noch zwei statische:

**Steilheitsfaktor** Dieser Parameter bestimmt den *absoluten* Anteil beim Höhenversatz zwischen Start- und Endpunkt der Bézierkurve einer Zelle. Er wird in *Punkten* angegeben, wobei zehn Punkte in etwa einem Meter realer Höhe entsprechen. Diese Relation ist allerdings willkürlich gewählt und kann über bestimmte Parameter der `config.ini`-Datei angepasst werden.

**Hügeligkeitsfaktor** Dieser Faktor bestimmt den *absoluten* Anteil beim Höhenversatz des zweiten Kontrollpunktes. Er wird ebenfalls in *Punkten* angegeben und kann in der `config.ini` angepasst werden. Da die Höhe des ersten Kontrollpunkts ebenfalls Einfluss auf den Höhenverlauf einer Straßenzelle hat, aber im Gegensatz zur Höhe des zweiten Kontrollpunkts von der Vorgängerzelle abhängt, ist es ersichtlich, dass ein durchgehend „gerader“ An- bzw. Abstieg von Start- zu Endpunkt einer Straßenzelle auch dann extrem unwahrscheinlich ist, wenn sowohl die Hügeligkeit als auch der Hügeligkeitsfaktor auf 0.0 gesetzt werden.

Alle diese Parameter können in der Datei `config.ini` initial gesetzt werden. Die dynamischen Parameter können mittels des Ziffernblocks bzw. über die externe Steuerung während der laufenden Simulation angepasst werden. Es ist dabei zu beachten, dass eine Änderung eines Parameters sich erst auf zukünftige Zellen auswirkt, also erst verzögert wahrgenommen werden kann. Wie diese verzögerte Auswirkung zustande kommt und auf welche Art die globale Streckenführung erstellt wird, soll nun Gegenstand des folgenden Kapitels sein.

### 5.2 Generierung der (Meta-)Strecke

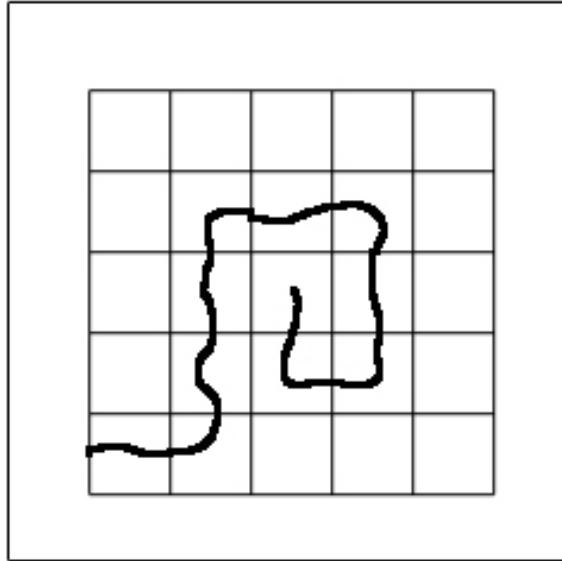
Im vorangegangenen Kapitel wurde die Generierung der Streckenführung innerhalb einer Straßenzelle besprochen. Im Folgenden soll nun dargestellt werden, wie diese Generierung auf der globalen Ebene funktioniert.

Jede Straßenzelle besitzt vier mögliche Ein- bzw. Austrittspunkte, an denen die Straßenteile miteinander verbunden werden können. Da der Ein- und der Austrittspunkt stets verschieden sind, gibt es also drei mögliche Alternativen für den „globalen Verlauf“ der Strecke: geradeaus und links bzw. rechts abbiegend (vom Eintrittspunkt aus gesehen). Da die Streckenführung nur abschnittsweise generiert wird, kann zu einem bestimmten Zeitpunkt keine Aussage über den zukünftigen Verlauf der Strecke getroffen werden<sup>14</sup>. Es muss

---

<sup>13</sup>Je weiter die beiden Kontrollpunkte einer Straßenzelle auseinanderliegen, desto „stärker“ ausgeprägt sind dadurch generierte Kurven.

<sup>14</sup>Durch die Notwendigkeit, die Streckenführung etwas im Voraus zu puffern (vgl. Abschn. 4.3) kennt man den zukünftigen Verlauf allerdings doch zumindest ein wenig, das ändert aber an dem grundlegenden



**Abbildung 12:** Schematische Darstellung einer Schlaufe bei der Wegfindung

daher gewährleistet werden können, dass der spätere Verlauf der Straße nicht durch Zufall wieder durch Zellen führt, die bereits generiert worden sind - denn dann hätte es entweder eine Straßenkreuzung geben müssen oder die Straße hätte zumindest in der Nachbarzelle schon gesehen werden können. Unter der Maßgabe eines 3x3 Sichtfeldes muss also zu jedem Zeitpunkt zwischen zwei Straßenzellen, die nicht zum gleichen Streckenzug gehören, ein Abstand von mindestens einer Terrainzelle existieren. Des Weiteren muss gewährleistet werden, dass die Strecke nicht in „Schlaufen“ hineinläuft, aus denen es keinen Ausweg mehr gibt (siehe Abbildung 12)

Um diesen Anforderungen zu genügen, wird eine Art konvexe Hülle des Straßenverlaufs angelegt. Es existieren dazu zwei Tabellen: eine Spalten- und eine Zeilentabelle. Die Einträge in diesen Tabellen beziehen sich auf die globalen Koordinaten des „Schachbretts“ der Metastreckengenerierung. Da dieses potentiell unendlich groß in jede Richtung werden kann, werden in den Tabellen nur Einträge abgelegt, an deren Koordinaten auch eine konkrete Straßenzelle liegt. Jeder Eintrag beinhaltet dabei die Zeilen- beziehungsweise Spaltenkoordinate sowie einen Minimumwert und einen Maximumwert. Diese beiden Werte entsprechen Zeilen bei der Spaltentabelle und Spalten bei der Zeilentabelle.

Abbildung 13 verdeutlicht die Arbeitsweise dieser Tabellen. Es wird bei der Streckengenerierung beispielsweise die Straßenzelle (X: 4, Y: 3) angelegt. Zu diesen Koordinaten enthält die Zeilentabelle bereits den Eintrag (Zeile: 3, Min: 1, Max 3), die Spaltentabelle ist für die Spalte „4“ hingegen noch unbelegt. Das heißt, das in der Zeile „3“ die am weitesten „links“ liegende Straßenzelle in der Spalte „1“ liegt, die bisher am weitesten „rechts“ liegende Straßenzelle dagegen in der Spalte „3“ liegt. Die Generierung der Zelle (X: 4, Y: 3) ändert den

---

Problem nichts.

## 5 Die Vorstellung der Arbeit

Neue Zelle: (X:4 , Y:3)					
Zeilentabelle (vor Einfügen)			Spaltentabelle (vor Einfügen)		
Zeile	Min	Max	Spalte	Min	Max
1	1	2	1	1	2
2	2	2	2	2	2
3	1	3	3	0	2

Zeilentabelle (nach Einfügen)			Spaltentabelle (nach Einfügen)		
Zeile	Min	Max	Spalte	Min	Max
1	1	2	1	1	2
2	2	2	2	2	2
3	1	4	3	0	2
			4	3	3

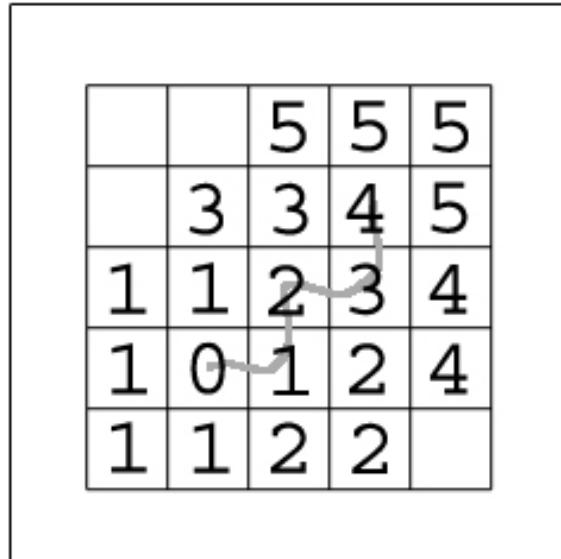
**Abbildung 13:** Beispiel einer Zeilentabelle und Spaltentabelle vor und nach Einfügen der Zelle (X:4, Y:3)

Eintrag der Zeilentabelle in (Zeile: 3, Min: 1, Max: 4), in der Spaltentabelle wird ein Eintrag (Spalte: 4, Min: 3, Max: 3) angelegt.

Die im Beispiel genannte Zelle (X: 4, Y: 3) kann in dieser Konstellation problemlos angelegt werden, sie liegt quasi „am Rand“ der bisherigen konvexen Hülle (es gab noch keinen Spalteneintrag) und ihr Anlegen ist „ungefährlich“, führt also nicht zu den erwähnten Schlaufen. Wäre statt dieser Zelle versucht worden, die Zelle (X: 2, Y: 3) anzulegen, hätte das zu einem Fehler geführt, da der X-Wert zwischen Minimum- und Maximum-Wert des entsprechenden Eintrags in der Zeilentabelle gelegen hätte. In diesem Fall würde der Streckenverlauf zwischen zwei bereits gesetzten Straßenzellen „hindurchführen“. Dieses Verhalten kann potentiell zu den genannten „Sackgassen“ führen und wird deshalb unterbunden. Das hier verwendete Verfahren stellt unter den speziellen Bedingungen eine effiziente und schnelle Lösung zur Berechnung der konvexen Hülle dar. Da allerdings das komplette Hüll-Polygon für den weiteren Straßenverlauf gesperrt wird, ist die Streckenführung in gewissen Bereichen eingeschränkt, in dem zum Beispiel keine „Doppelschlaufen“ angelegt werden können.<sup>15</sup>

Es existiert noch eine letzte Regel, die bei der Erstellung der Streckenführung von Bedeutung ist. Wie schon erwähnt, müssen Zellen im Vorfeld generiert werden, um genügend Puffer zum Umschalten zu besitzen (vgl. Abschn. 4.3). Dabei werden angrenzende Terrainzellen mit einem Schritt Verspätung als „fertig“ markiert, können also in weiteren Generierungsschritten nicht mehr für Straßenzellen genutzt werden. Diese Maßnahme sorgt für den notwendigen Mindestabstand von einer Terrainzelle zwischen zwei Straßenzügen. Dabei markiert eine Zahl in der jeweiligen Zelle, in welchem „Generierungsschritt“ sie angelegt wurde. Abbildung 14 verdeutlicht diesen Vorgang noch einmal. Man kann erkennen, dass bei der Erzeugung einer Straßenzelle alle die Nachbarzellen des 3x3-Sichtradius des vorgehenden Straßenabschnittes „belegt“ werden, die nicht direkt von der aktuellen Zelle heraus erreicht werden können.

<sup>15</sup>Diese Einschränkung ist allerdings für das Fahrempfinden irrelevant, da einerseits eh ein Mindestabstand zwischen zwei Straßenzügen eingehalten werden muss, andererseits auch aus der subjektiven Perspektive des Fahrers über den globalen Verlauf der Strecke keine Aussage getroffen werden kann.



**Abbildung 14:** Schematische Darstellung der verzögerten Generierung von Zellen

### 5.3 Ablauf der Zellgenerierung

Nachdem im vorangegangenen Abschnitt die Generierung der globalen Streckenführung besprochen wurde, soll nun auf die Generierung der einzelnen Zellen näher eingegangen werden. Diese erfolgt in mehreren Abschnitten, die im Folgenden vorgestellt werden sollen. Davor soll jedoch noch auf eine grundsätzliche Unterscheidung hingewiesen werden: es kann entweder eine Terrain- oder eine Straßenzelle angefordert worden sein. Diese Unterscheidung ist in mehrererlei Hinsicht wichtig, die wichtigste jedoch besteht in der simplen Tatsache, dass in einer Terrainzelle sich niemals eine Straße befinden wird. In der Beschreibung des Generierungsbalaufes wird daher davon ausgegangen, dass eine Straßenzelle angefordert wurde. Die Erstellung einer Terrainzelle erfolgt analog, auf die Unterschiede wird an entsprechender Stelle hingewiesen.

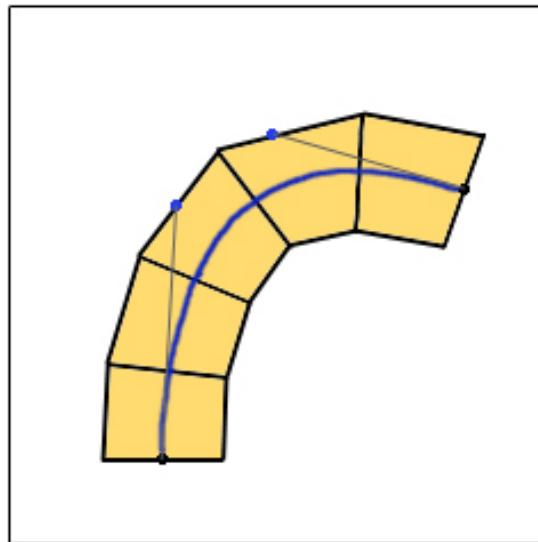
#### 5.3.1 Straßenverlauf als Bézierkurve

Es wurde bereits dargelegt, dass jede Zelle um die in ihr liegende Straße „herumgebaut“ werden muss<sup>16</sup> (vgl. Abschn. 4.2). Daher besteht der erste Schritt darin, die entsprechende Bézierkurve aufzustellen (vgl. Abschn. 5.1.3). Dabei werden die Anforderungen der globalen Streckenführung einbezogen. Diese Kurve entspricht natürlich nur dem Verlauf des „Mittelstreifens“ und stellt noch keine richtige Straße dar. Deshalb wird nun ein entsprechendes Straßenmesh erzeugt. Dazu werden mittels des *de Casteljau-Algorithmus* Punkte auf dem Kurvenverlauf ermittelt<sup>17</sup>. Dann werden zu jedem so ermittelten Kurvenpunkt die „Straßen-

<sup>16</sup>Dieser Schritt entfällt natürlich für reine Terrainzellen.

<sup>17</sup>Die Menge dieser Punkte bestimmt die „Genauigkeit“, mit der der Kurvenverlauf approximiert wird. Je größer die Anzahl der Punkte ist, desto „runder“ wirkt die Straße auf den Betrachter. Eine größere Anzahl an Punkten führt dabei allerdings auch zu einer längeren Berechnungszeit und größeren Meshes, ein Kompromiss aus Genauigkeit und Geschwindigkeit ist also notwendig. Der Wert kann dabei vom Nutzer

seitpunkte“ ermittelt, also die Punkte, die jeweils im rechten Winkel zur Kurventangente im ermittelten Punkt im der Straßenbreite entsprechendem Abstand<sup>18</sup> liegen. (vgl. dazu Grafik 15) Auf das so erhaltene Mesh wird eine rechteckige Straßengrundtextur gemappt. Die dabei auftretenden Verzerrungen und Aliaseffekte sind bei einer ausreichend hohen Anzahl Punkte auf der Bézierkurve vernachlässigbar gering.



**Abbildung 15:** Schematische Darstellung des Straßenmeshes entlang der Bézierkurve

### 5.3.2 Einbettung der Straße in die Zelle

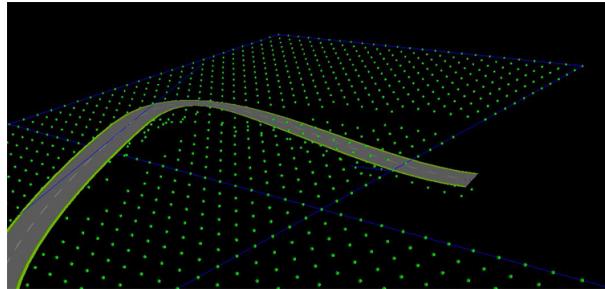
Nachdem die Bézierkurve erstellt wurde, muss sie natürlich innerhalb des Terrains verankert werden. Wie bereits vorgestellt, wird an dieser Stelle eigentlich der Straßenuntergrund generiert, während das eigentliche Straßenmesh in gewissem Abstand darübergelegt wird (vgl. Abschn. 4.2). Dazu müssen allerdings alle Punkte des Zellgitters identifiziert werden, die „unterhalb“ bzw. „direkt neben“ der Straße liegen, um deren Höhe entsprechend anzupassen. Da die Punkte des Zellgitters allesamt regelmäßig angeordnet sind, kann die Lösung dieses Problems im zweidimensionalen Raum erfolgen: direkt „von oben“<sup>19</sup> betrachtet, ergibt sich eine solche Straßenzelle als quadratisches, regelmäßig angeordnetes Punktmuster. Darüber wird jetzt das „Skelett“ des Straßenmeshes gelegt (vgl. Abbildung 2). An dieser Stelle kann ein Straßenzug durch eine Zelle also als eine Aneinanderreihung von Vierecken betrachtet werden, deren Größe von der Auflösung des Straßenmeshes bestimmt ist (vgl. Abschn. 5.3.1). Um nun die Punkte zu ermitteln, die innerhalb der Grenzen dieser Polygone liegen, wurde der SCANLINE–Algorithmus verwendet. Dieser ist bereits seit viele Jahrzehnten bekannt und wird beispielsweise dazu benutzt, Rasterungen von Polygonzügen auf Pixelrastern von beispielsweise Monitoren zu erzeugen (siehe beispielsweise Bouknight (1970)). Dabei wird jede Zeile des Punktgitters „abgelaufen“ und für jeden Punkt bestimmt,

---

in der config.ini-Datei eingestellt werden.

<sup>18</sup>Diese kann ebenfalls in der config.ini-Datei eingestellt werden.

<sup>19</sup>Also in negativer Y-Achsenrichtung.



**Abbildung 16:** Einbettung des Straßenmeshes in das Zellgitter (inklusive Darstellung des Straßenmeshes)

ob er „innerhalb“ oder „außerhalb“ des Polygonzuges liegt, sowie in welchem Polygon des Straßenzuges er sich konkret befindet<sup>20</sup>. Im Anschluss daran wird für jedes Polygon des Straßenzuges die Ebenengleichung aufgestellt<sup>21</sup> und die Höhenkomponente jedes Zellgitterpunktes unterhalb eines solchen Segments auf einen gewissen Abstand von diesem festgelegt (vgl. Abbildung 16).

Im Anschluss daran werden noch die Punkte neben der Straße identifiziert. Diese werden dann auf eine gewisse Höhe über dem Straßengrund festgelegt, um dem Straßenrand eine Profilform zu geben, wie sie in Abbildung 2 zu sehen ist. Die auf diese Art ermittelten und in der Höhe festgelegten Punkte werden als `Constraints` an die nächste Phase, die Terraingenerierung, übergeben.

### 5.3.3 Generierung des Terrains

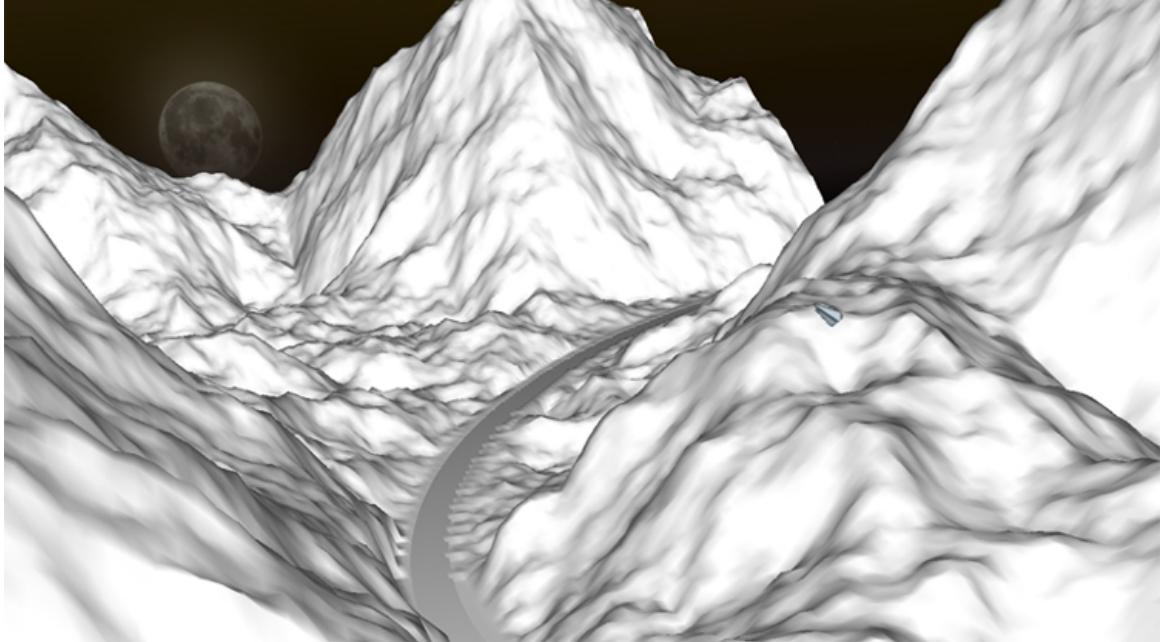
Die dritte Phase der Zellgenerierung erstellt nun aus den bisherigen Rohdaten das Geländemesh. Diese Rohdaten beinhalten einerseits die Zellpunkte unterhalb und neben der Straße, deren Identifikation im vorigen Abschnitt beschrieben wurde<sup>22</sup>. Andererseits gibt es allerdings mit den Zellrändern noch eine Reihe anderer Constraints, die beachtet werden müssen: da Zellen (Terrain- wie auch Straßenzellen) nahtlos aneinandergrenzen sollen, ist es erforderlich, die Zellrandpunkte auf die entsprechenden Höhen der bereits generierten Nachbarzellen zu setzen. Zusätzlich muss auch gewährleistet sein, dass die in den Randpunkten jeweils vorhandenen Anstiege des Meshes der vorhergehenden Zelle gewahrt bleiben, um einen „glatten“ Übergang zu gewährleisten. Je nach Lage der zu generierenden Zelle sind nun also zwischen null und allen vier Rändern der Zelle in ihrer Höhe determiniert, zählen also zur Menge der `Constraints`<sup>23</sup>. Auf diese Menge wird nun der `MDBU`-Schritt des Terraingenerierungsverfahrens angewendet (vgl. Abschn. 5.1.2), um die Menge der erzeugenden Elternknoten in die Menge der `Constraints` aufzunehmen. Danach wird ein normaler

<sup>20</sup>Diese werden dazu durchnummeriert.

<sup>21</sup>Aufgrund der Vereinfachung, dass die Straßenseitpunkte stets auf der gleichen Höhe wie die zugehörigen Punkte der Bézierkurve liegen, bildet jedes der Polygone des Straßenzuges auch im dreidimensionalen Raum stets eine Ebene.

<sup>22</sup>Es ist natürlich leicht ersichtlich, dass im Fall der Generierung einer reinen Geländezeile dieser Abschnitt der erste ist, der bearbeitet wird - und dass es dann natürlich entsprechend auch keine Constraints gibt, die den Straßenuntergrund darstellen.

<sup>23</sup>„Null“ determinierte Ränder hat dabei nur die Startzelle, „vier“ kann nur bei sehr wenigen Streckenverläufen bei einer Terrainzelle erreicht werden.



**Abbildung 17:** In das Terrain eingebetteter Straßenverlauf

`TriangleEdge`-Algorithmus ausgeführt, wobei alle die Punkte unverändert bleiben, die als Constraint festgelegt wurden. Dadurch erhält man dann das Mesh der jeweiligen Zelle als zufälliges Terrain, wobei alle Vorgaben (Ränder, eventuell eine durchführende Straße) eingearbeitet wurden. Eine anschließende Glättung des Terrains wäre nun möglich<sup>24</sup>. In der vorliegenden Arbeit wurde allerdings darauf verzichtet, da eine Erhöhung der Rekursionstiefe des `TriangleEdge`-Verfahrens ein ähnliches Ergebnis liefern kann.

#### 5.3.4 Generierung der Gimmicks

Den Abschluss der Generierung bildet die Einbindung sogenannter „Gimmicks“. Damit sollen alle (optischen) Anreicherungen der Szene gemeint sein, die keinen Einfluß auf die Strecke und das Terrain-Mesh besitzen<sup>25</sup>. Die vorliegende Arbeit benutzt Bäume zur Demonstration dieses Features. Userseitig können dabei in der `config.ini`-Datei Angaben gemacht werden, wieviele Bäume pro Zelle maximal gesetzt werden sollen und wie „flach“ oder „steil“ der Untergrund dafür sein darf. Für jeden (zufällig) ausgewählten möglichen Terrainpunkt, an dem sich ein Baum befinden könnte, wird dann der maximale Anstieg der umliegenden Punkte ermittelt. Ist dieser Anstieg kleiner als der festgelegte Schwellwert, ist das Gelände hinreichend „flach“ und der Baum wird gesetzt<sup>26</sup>. Zusätzlich dazu bietet das Programm

<sup>24</sup>Dabei müssten allerdings die Constraints natürlich ausgeschlossen werden.

<sup>25</sup>Natürlich ist auch die Einbindung von Gimmicks möglich, die Einfluß auf das Terrain haben würden (ein Haus würde beispielsweise einen planen Untergrund benötigen). Dieses Vorgehen würde einen weiteren Schritt vor der Terraingenerierung benötigen, bei der weitere Constraints erzeugt werden müssten. Auf diese Erweiterung des Konzeptes wurde aber im vorliegenden Entwurf verzichtet, um eine Konzentration auf das Thema „Straße“ zu erreichen.

<sup>26</sup>Die im Programm verwendeten Bäume sind selbstverständlich nur Dummy-Objekte mit einer sehr kleinen Polygonanzahl, um einen möglichst geringen Einfluss auf die Gesamtperformance zu nehmen. Die Entwick-



**Abbildung 18:** Szene aus der fertigen Applikation

die Möglichkeit, Wasser darzustellen<sup>27</sup>. Diese Option könnte auch als Gimmick umgesetzt werden, das für jede Zelle einzeln gesetzt wird. In der vorliegenden Arbeit wurde es allerdings als globale Wasserfläche implementiert, sodass es kein „Gimmick“ im hier diskutierten Sinne darstellt. Die Generierung der Zelle ist an dieser Stelle abgeschlossen. Das Mesh und alle auf der Zelle platzierten Gimmicks werden nun in den Grafikspeicher geladen und die entsprechenden Physikkörper für die Kollisionserkennung werden erzeugt. Die so erzeugte Zelle „ruht“ nun unsichtbar im Speicher, bis sie in das 3x3-Sichtfeld des Spielers gerät, und wird dann sichtbar geschaltet.

Nachdem es in den vorangegangenen Abschnitten eher um allgemeinere Konzepte ging, soll im folgenden Kapitel nun konkret auf einige Details der Implementierung eingegangen werden.

## 6 Umsetzung und Implementierung

Die vorangegangenen Abschnitte haben sich mit dem theoretischen Fundament der Beleg-Applikation beschäftigt und den algorithmischen Rahmen der Straßen- und Zellgenerierung

lung eines ausgereiften Algorithmus zur effizienten Darstellung von (glaubwürdigen) Bäumen ist alles andere als trivial, da auf jeden Fall LevelOfDetail-Algorithmen, Verdeckungsrechnungen, gegebenenfalls entsprechende (Wind-)Physik und andere komplexe Funktionen einbezogen werden müssen. Das war aber nicht Gegenstand dieser Arbeit, sodass an dieser Stelle lediglich ein Ansatz gegeben wird, der gegebenenfalls ausgebaut werden könnte.

<sup>27</sup>Diese Option ist allerdings defaultmäßig deaktiviert, da gerade im Zusammenspiel mit den Bäumen ein enormer Performanceeinbruch zu verzeichnen ist. Zur Aktivierung bitte die entsprechenden Parameter in der config.ini-Datei setzen.

## *6 Umsetzung und Implementierung*

vorgegeben. Das folgende Kapitel soll nun auf die konkrete Umsetzung dieser Vorgaben in Quellcode eingehen, das Programm vorstellen und auf einige Besonderheiten näher eingehen.

### **6.1 Die Auswahl der Grafikengine**

Die erste wichtige Entscheidung, die zu treffen war, betraf die Wahl einer geeigneten Grafikengine beziehungsweise eines Grafikframeworks, innerhalb dessen die Applikation geschrieben werden sollte. Eine Eigenentwicklung wurde nicht in Betracht gezogen, da die verfügbare Zeit besser in die Umsetzung der Aufgabenstellung fließen sollte. Es existieren mehrere OpenSource-Engines zur Darstellung von 3D-Daten. Für die Umsetzung wurden die beiden Engines Irrlicht (2010) sowie OGRE 3D (2010) evaluiert. Am Ende wurde sich für die OGRE-Engine entschieden, da sie sich deutlich einsteigerfreundlicher und flexibler sowohl im Einsatz als auch in der Programmierung darstellte. Des Weiteren existiert zu OGRE eine zahmäig weit größere Entwicklergemeinde, die Dokumentation ist deutlich umfangreicher und es existiert die Möglichkeit, über ein Plugin-System weitere Features nahtlos zu integrieren.

In den folgenden Abschnitten soll das OGRE-Framework kurz vorgestellt werden sowie der grundlegende Aufbau einer OGRE-Applikation erläutert werden.

### **6.2 Die OGRE-Engine allgemein**

Das OGRE-Framework bietet eine vielseitige und flexible Grafikengine, die sich nahezu nahtlos in eine Applikation integrieren lässt.

#### **6.2.1 Vorstellung des OGRE-Frameworks**

Das für die Realisierung der Belegarbeit verwendete Grafik-Framework ist die OpenSource Software OGRE. Diese stellt eine plattformunabhängige Grafik-Engine zur Verfügung, die sowohl OpenGL als auch Direct3D sowie aktuelle Shadersprachen unterstützt. Sie wird fortlaufend weiterentwickelt. Die für den Beleg verwendete Version ist 1.6.5. Eine Portierung auf die derzeit neueste Version 1.7 wurde aufgrund vieler neuer Features (wesentlich bessere Threadunterstützung, neue Terrainverwaltungsmöglichkeiten etc.) in Betracht gezogen, aber aufgrund der nötigen Portierungszeit wieder verworfen. OGRE ist modular aufgebaut, Erweiterungen können in Form der schon angesprochenen Plugins integriert werden. Einige dieser Plugins werden für das Funktionieren benötigt (Render-Plugins, Scene-Manager), andere werden optional mit dem SDK mitgeliefert (ODE als Implementation einer Physik-Engine oder CEGUI als integrierbares GUI-System). Dabei wird tatsächlich „nur“ eine Grafik-Engine zur Verfügung gestellt, um dem Programmierer keine Entscheidungen bezüglich anderer möglicherweise verwendeter Komponenten (Sound, Physik etc.) aufzuzwingen. Die dritte Form von Plugins werden durch die Community bereitgestellt und reichen von Wrappern für diverse Physik-Engines bis zu eigenständigen Erweiterungen der Grafikengine oder Portierungen von OGRE auf andere Programmierumgebungen.

### 6.2.2 Entwicklung einer Anwendung mit OGRE

OGRE bietet zwei Möglichkeiten des Programmablaufs: einerseits kann das Rendern eines einzelnen Frames direkt aufgerufen werden. Die eigene Software kann dann zwischen zwei Aufrufen beliebigen Quelltext ausführen. Dieses Vorgehen ermöglicht die Integration von OGRE in eine beliebige Applikation, die bereits eine eigene MAIN-Schleife ausweist. Andererseits kann OGRE auch mit der Durchführung einer kompletten Render-Schleife beauftragt werden, der eigene Quellcode wird dann in Form von FrameListenern jedes Mal vor beziehungsweise nach dem Rendern eines Einzelbildes aufgerufen. Diese Methode bietet den Vorteil, die Verwaltung der MAIN-Schleife komplett OGRE zu übertragen und die einzelnen Teile der eigenen Applikation logisch getrennt auf verschiedene FrameListener zu verteilen. Die Grafikengine läuft dabei in einem eigenen Thread, der in der Regel auch der Main-thread jeder Anwendung sein sollte. Der Zugriff auf Komponenten des Grafikframeworks über Thread- oder gar Prozessgrenzen hinaus ist nur in eingeschränktem Maße möglich - eine Technik, die diesen Zugriff realisiert, wird im Verlaufe dieses Kapitels noch vorgestellt werden. Abgesehen von dieser Einschränkung läuft OGRE aber autonom und bietet mit beiden Programmablauffunktionen genügend Möglichkeiten, eine Anwendung problemlos an die Grafikengine anbinden zu können.

Im ersten Schritt sollte eine minimale OGRE-Anwendung erstellt werden. Der Beitrag TidalWare (2009) zeigt den Ablauf der dazu notwendigen Schritte am Beispiel von Visual Studio 2008. Darauf aufbauend kann dann die Implementierung der eigentlichen Anwendung erfolgen.

OGRE benutzt zur Verwaltung der Grafikobjekte das Konzept der *Szenen*. Jede Szene ist eine Komposition aus Knoten, den *SceneNodes*, welche sich baumartig unterhalb des *root-Nodes* erstellen lassen. An diese Nodes können Grafikobjekte, sogenannte *Entities*, angehängt werden. Der Knoten fungiert dabei als Container beliebig vieler Entities. Lineare Transformationen werden direkt auf die einzelnen Nodes angewandt und nur mit dem root-Node verbundene Nodes werden bei der Renderung berücksichtigt. Zur Verwaltung der Knoten und Entities steht der *SceneManager* zur Verfügung. Pro Szene kann es nur einen SceneManager geben, es stehen allerdings mehrere Prototypen zur Verfügung. Diese sind für ihren jeweiligen Einsatzzweck optimiert. Neben dem Standard-SceneManager *GENERIC* existieren beispielsweise mehrere Manager speziell für das Rendering von Terrains bzw. großen Outdoor-Szenen.

Die Verwaltung von Materialien und Texturen übernimmt der *MaterialManager*. Dieser ermöglicht den Zugriff auf Texturen und Materialien einer Szene zur Laufzeit des Programms. Die meisten Objekte bekommen ihre Textur- und Materialeigenschaften allerdings über das externe Materialverwaltungssystem. In mehreren Textdateien können Materialzuweisungen und Shaderprogrammierung durchgeführt werden, auch ist darüber beispielsweise eine automatische LevelOfDetail Behandlung auf Materialebene möglich. Für eine ausführliche Dokumentation der Materialverwaltung sei an dieser Stelle aus Platzgründen auf die OGRE-Dokumentation verwiesen.

Eine Verwaltung einzelner Szenen ist in OGRE nicht implementiert, für den Wechsel und etwaige „Aufräumarbeiten“ zwischen mehreren Szenen hat der Entwickler selbst zu sorgen.

## 6 Umsetzung und Implementierung

Auch die Organisation des Programmablaufs mittels Zuständen oder ähnliche Aufgaben, die nicht direkt die Grafikausgabe betreffen, werden von OGRE nicht nativ unterstützt. Dennoch existieren in der OGRE-Community viele Lösungen und Vorschläge für konkrete Probleme. Bei Bedarf sollten diese im Forum, im Wiki oder in den Cookbooks nachgeschlagen werden.

### 6.3 Aufbau der Belegapplikation

Die Software basiert grundsätzlich auf vier verschiedenen Modulen:

- grafikparalleler Teil (Physik, Steuerung)
- TerrainManagement
- TerrainGenerator
- Communication Server

Diesen vier Komponenten ist gemeinsam, dass sie jeweils in einem eigenen Thread laufen, somit weitgehend unabhängig voneinander agieren können. Diese Trennung ist notwendig, da die tatsächliche Generierung des Terrains mehrere Sekunden Zeit in Anspruch nimmt. Würde die Generierung nun im grafikparallelen Teil der Software durchgeführt, kommt dazu grundsätzlich nur der Zeitraum zwischen zwei Frames in Frage. Dabei würde die Generierungszeit den Anwendungsfluss derart unterbrechen, dass kein sinnvoller Programmablauf zustande kommt. Eine „Stückelung“ der Generierung in viele kleinere Teile von wenigen ms Länge wäre zwar denkbar, aber nicht zweckmäßig in der Umsetzung. Die Trennung des Terrain-Managements sowohl vom grafikparallelen Teil als auch vom Terrain-Generator folgt ähnlichen Überlegungen: einerseits soll eine saubere Trennung von der Grafikengine durchgeführt werden, um diese nicht durch zu zeitintensive Berechnungen zwischen zwei Frames „auszubremsen“, andererseits soll das Terrain-Management auch dann ansprechbar sein, wenn der Generierungsprozess einer einzelnen Zelle gerade im Gange ist.

Bevor im Folgenden die einzelnen Komponenten näher vorgestellt werden, soll allerdings erst das Konzept der Zellpuffer näher erläutert werden:

**Zellpuffer** Wie schon näher erläutert, muss im vorliegenden Fall die Generierung des Terrains in einem nachgelagerten Thread erfolgen. Nachdem die Daten generiert wurden, müssen sie allerdings noch in ein für OGRE darstellbares Format, also in die Form eines Entities gebracht werden. OGRE ermöglicht dazu die manuelle Generierung von Entities, sogenannter *ManualObjects*, indem die benötigten Daten (Vertices, Indices, Texture Coordinates etc.) direkt während der Laufzeit zusammengestellt werden. Dafür existieren prinzipiell zwei Möglichkeiten: einmal die Generierung der ManualObjects durch einfache Übergabe der entsprechenden Datenstrukturen, OGRE kümmert sich dann selbst um die Organisation und den Upload der Daten auf der Grafikkarte. Zum Anderen kann diese Organisation auch manuell übernommen werden, OGRE stellt dann bei Bedarf lediglich Zeiger auf Grafikspeicherbereiche zur Verfügung, die von der Anwendung sinnvoll mit den entsprechenden Daten befüllt werden müssen.

Die Organisation der ManualObjects durch OGRE kann nur innerhalb des Grafikthreads durchgeführt werden, da Zugriffe aus anderen Threads heraus zu undefiniertem Verhalten führen würden<sup>28</sup>. Bei dieser Variante würde also lediglich die Struktur des Terrains im Hintergrund erstellt werden können, der Upload auf die Grafikkarte müsste zwischen zwei Frames durch OGRE erledigt werden. Dieser Upload dauert allerdings ebenfalls zu lange, um einen ruckelfreien Programmablauf gewährleisten zu können. Aus diesem Grund kam für das Programm nur die Variante in Frage, bei der OGRE lediglich Zeiger auf die entsprechenden Speicherbereiche zur Verfügung stellt. Die Befüllung dieser Speicherbereiche kann nämlich durchaus auch in einem nachgelagerten Thread durchgeführt werden, solange die Anwendung sicherstellt, dass die entsprechenden Zeiger gültig bleiben und die reservierten Speicherbereiche nicht anderweitig verwendet werden.

Die Umsetzung dieser Variante erfordert einen großen Aufwand an Kooperation zwischen den drei beteiligten Threads<sup>29</sup>. Diese Kooperation sieht wie folgt aus: Zu Programmstart werden einige „Dummy-Entities“ erstellt und in der Szene platziert<sup>30</sup>. Diese enthalten vorerst keinen Inhalt. Es wird allerdings dadurch so viel Speicher angefordert, dass jedes Entity die entsprechende Menge an Daten speichern kann. Diese Speicherbereiche werden gesperrt und die Zeiger dem Terrainmanager zur Verfügung gestellt. Dieser legt für jeden Zelltyp nun einen Pool mit diesen Zeigern an. Dieser Pool enthält zusätzliche Informationen darüber, welche Zellen gerade benutzt werden und welche Zelle konkret gerade gespeichert wird (XY Wert des Rasters). Der TerrainManager sorgt jetzt über einen zweiten Puffer dafür, dass freie oder zum Überschreiben freigegebene Entities (bzw. deren Zeiger) bei Bedarf zur Erstellung der nächsten benötigten Terrainzellen herangezogen werden. Dazu wird dann der freie Zeiger zusammen mit Informationen über die zu erzeugende Zelle dem TerrainGenerator übergeben. Dieser beschreibt die durch den Zeiger markierten Speicherbereiche mit den Daten der neuen Zelle, sodass diese dann, sobald sie in den sichtbaren 3x3 Umkreis des Fahrers rückt, im Vordergrundthread sichtbar geschaltet werden kann.

Die Verwendung einer festen Zahl an Entities wäre nicht zwingend notwendig, es könnte auch für jede neue Zelle ein neues Entity angelegt werden. Dabei ist allerdings zu bedenken, dass die Terrainzellen eine bestimmte, userseitig konfigurierbare Menge an Daten enthalten und der Speicher auf der Grafikkarte nur endlich zur Verfügung steht. In diesem Kontext ist die Vollpufferlösung vorzuziehen, da sie nur noch die Konfiguration der Zellgrößen als Unbekannte bei der Speichernutzung zulässt und ansonsten einen konstanten Speicherbedarf bedeutet.

Im Folgenden sollen die einzelnen Komponenten noch einmal einzeln betrachtet werden und auf ihre spezifischen Aufgaben eingegangen werden.

---

<sup>28</sup>Wie schon erwähnt, gibt es keine Möglichkeit, auf Befehle des OGRE-Frameworks außerhalb des Grafikthreads zuzugreifen.

<sup>29</sup>TerrainGenerator, TerrainManager und der grafikparallele Teil sind an der Umsetzung beteiligt.

<sup>30</sup>Es werden sowohl Entities für Terrainzellen als auch für Straßenzellen angefordert.

### 6.3.1 Vorstellung der Module

**a) grafikparalleler Teil der Anwendung** Es gibt verschiedene Aufgaben der Simulation, die in Echtzeit erledigt werden müssen. Die Bewegung der Figur selbst steht dabei natürlich an erster Stelle. Die Steuerungskomponente bekommt ihre Information entweder asynchron durch die Abfrage von Maus und Tastatur oder ebenfalls asynchron durch den Communicator zugespielt. In beiden Fällen werden die Eingaben dazu benutzt, Lage und Orientierung der Figur im Raum für jeden Frame neu zu berechnen und sie entsprechend zu bewegen. Falls sich die Figur dabei über Zellgrenzen hinwegbewegen sollte, wird eine Nachricht an das TerrainManagement gesendet. Im Kooperation mit dem TerrainManager werden dann die Sichtbarkeiten des die Fahrerfigur umgebenden 3x3 Zellrasters neu angepasst und die frei werdenden Zellpuffer entsprechend markiert und dem Manager gemeldet. In diesem Teil der Anwendung werden darüber hinaus die Physik-Objekte angelegt, die zur Navigation in der Szene erforderlich sind: also die Kollisionsobjekte der Strasse und die „Wände“, die den Fahrer auf Kurs halten. Da diese Objekte bereits von TerrainGenerator angelegt wurden, werden sie hier lediglich verknüpft und der Simulation bekanntgemacht. Zuletzt ist in dieser Komponente die Verwaltung der anderen Komponenten beheimatet. Diese werden hier gestartet und entsprechend beendet, sodass an dieser Stelle auch das Speichern und Laden von Skripten, Configdateien sowie Strecken angesiedelt ist.

**b) TerrainManager** Der TerrainManager ist das „Herzstück“ der Anwendung. Dieser Teil der Software steuert das Nachladen von neu ins Blickfeld gekommenen Zellen wie auch die Anforderung neuer Zellen beim TerrainGenerator. Des Weiteren wird hier der Verlauf der globalen Route festgelegt und die dynamische Zuordnung der Puffer organisiert. Direkt nach Programmstart werden die Puffer entsprechend vorgefüllt, damit die Simulation beginnen kann. Dazu werden fünf Straßenzellen sowie die jeweils angrenzenden Terrainzellen angelegt und beim Generator angefordert. Der Algorithmus garantiert, dass die vierte Strassenzelle auf jeden Fall außer Sichtweite der aktuellen Zelle, also außerhalb des 3x3 Rasters, generiert wird. Somit kann in jedem Fall bei Überfahrung der Zellgrenze auf ein bereits generiertes Feld umgeschaltet werden (für Details Algorithmus vgl. Abschn. 5.2). Der Manager stellt überdies sicher, dass bereits generierte Terrainzellen, die sowohl in der Umgebung einer bereits abgefahrenen Straßenzelle als auch in der Umgebung einer im weiteren Verlauf erscheinenden Straßenzelle liegen, nicht neu erstellt werden sondern korrekt aus dem Speicher zurückgelesen werden. Sollten die Zellen noch im Puffer liegen, aber als verwerfbar markiert worden sein, wird sie reaktiviert, ohne neu eingelesen werden zu müssen. Dabei ist das Reaktivieren aus dem Puffer natürlich wesentlich effizienter als das erneute Einlesen über den TerrainGenerator. Die frei einstellbare Puffergröße hat an dieser Stelle maßgeblichen Einfluss auf den möglichen Performance-Gewinn. Eine Puffergröße von mindestens 14 Terrainzellen wird benötigt, um genügend Zellen für die Umschaltung bei Zellgrenzüberschreitung bereitzuhalten<sup>31</sup>. Dabei ist zu beachten, dass sehr große Terrainpuffer abhängig von der gewählten Auflösung der Zellen entsprechend viel Speicherplatz auf der Grafikkarte benötigen. Der reale Performancegewinn durch eine größere Anzahl an Terrainpuffern ist allerdings vollständig vom Output des Streckengenerierungsalgorithmus abhängig und somit generell vergleichweise gering.

---

<sup>31</sup>Da der Streckenverlauf dem Zufall unterliegt, ist diese Zahl für den „worst case“ notwendig, also den Streckenverlauf, bei dem zur Terrainumschaltung die meisten Terrainzellen im Puffer liegen müssen.

**c) TerrainGenerator** Der TerrainGenerator dient dazu, die Vorgaben des TerrainManagers in konkrete Zellgeometrien umzusetzen. Dabei wird von TerrainManager eine bestimmte Menge an Zellen angefordert, welche der Reihe nach abgearbeitet werden (der Algorithmus wird im Detail in Abschn. 5.3 besprochen). Nachdem die Geometrien in die entsprechenden Speicherbereiche geschrieben wurden, werden bei Straßenzellen zusätzlich die Straßengeometrieobjekte erzeugt und auf der Grafikkarte hinterlegt, wie auch deren Physikkörper erzeugt und für die Aktivierung durch den grafikparallelen Teil der Anwendung vorbereitet. Der TerrainGenerator hat ebenfalls die Aufgabe, gegebenenfalls zur angeforderten Zelle vorhandene Skriptinformationen einzuarbeiten beziehungsweise die Geometrie aus der Sicherung zu laden, falls eine gespeicherte Strecke neu abgefahren werden soll. Dazu gehört natürlich auch die Einarbeitung der als „Gimmicks“ bezeichneten Landschaftselemente. Die Simulation liefert eine rudimentäre Unterstützung für Bäume mit, Erweiterungen müssten gegebenenfalls in den Generierungsprozess an entsprechender Stelle integriert werden. Dabei sollten alle Gimmicks, die Änderungen in der Zellgemometrie an sich benötigen (beispielsweise plane Grundflächen für Häuser) als Constraints vor den eigentlichen Generierungsprozess gestellt werden. Alle Gimmicks, die hingegen auf der vorhandenen Zellgeometrie aufbauen ohne diese ändern zu müssen, sollten hingegen erst im Anschluss an die Geometrieerzeugung eingebaut werden.

**d) Communication Server** Optional besteht die Möglichkeit, innerhalb der Anwendung einen Communication-Server zu starten, der ebenfalls einen eigenen Thread belegt. Grundlage des Servers sind `NamedPipes`. Der Server nimmt auf seiner Pipe `.\|pipe|mynamedpipe` Nachrichten eines bestimmten Formats entgegen und reicht sie dann an die anderen Teile der Anwendung weiter. Der Communicator stellt damit die Schnittstelle dar, mit der eine externe Anwendung oder ein Eingabegerät die Steuerung der Simulation übernehmen könnte. Der mitgelieferte Communication-Server ist dabei lediglich eine Beispielanwendung, wie die Komponente in der entsprechenden externen Anwendung implementiert werden könnte. Zu Demonstrationszwecken erfolgt deshalb eine zufällige Auswahl der verfügbaren Befehle, deren Umsetzung in der Beleg-Applikation beobachtet werden kann.

Es wurde bereits erwähnt, dass alle Module der Software in eigenen Threads ablaufen. Dadurch entsteht die Notwendigkeit einer synchronisierten Kommunikation der Module untereinander. Zu diesem Zweck wurde ein entsprechendes Event-Management-System (EMS) entworfen und implementiert, mit dessen Hilfe beliebige Daten zwischen den jeweiligen Threads ausgetauscht werden können. Das EMS selbst belegt dabei keinen „eigenen“ Thread, sondern stellt quasi einen geschützten Datenbereich für alle Threads gleichzeitig dar. Der `EventManager` ist dabei eine Implementierung des *Singleton-Entwurfsmusters*. Jedes entsprechend konfigurierte Objekt kann sich dort als `EventListener` für bestimmte `Events` registrieren. Diese `EventListener` erhalten dann die durch die `EventSources` generierten Ereignisse, sofern sie diese abonniert haben. Die Abholung der Nachrichten durch die Listener erfolgt dabei durch Polling des jeweiligen „Postkastens“.

Es wurde darauf verzichtet, ein bereits durch vorgefertigte Bibliotheken EMS zu verwenden<sup>32</sup>, da die Eigenentwicklung die maximale Anpassung des EMS an die Bedürfnisse der Applikation ermöglicht.

---

<sup>32</sup>Beispielsweise durch Nutzung von *Boost.Signals*.

### 6.3.2 Verwendete Fremd-Plugins

Die Anwendung nutzt mehrere Fremd-Plugins, um zusätzliche Funktionalität bereitzustellen. Folgende Plugins wurden genutzt und sollen kurz vorgestellt werden:

**SkyX** SkyX ist ein OGRE-Plugin von Xavier González. Es bietet vielfältige Möglichkeiten zur Visualisierung von Himmel und Wolken, atmosphärischen Effekten sowie Himmelskörpern. Zusätzlich ist eine Tagesablauf-Komponente integriert, die einen wechselnden Tageszyklus ermöglicht. Anhand der Tageszeit wandern die entsprechenden Himmelskörper (Sterne, Sonne, Mond) über den virtuellen Himmel. Zusätzlich bietet das Plugin Möglichkeiten, auf die Lichtstreuung in der Atmosphäre (Rayleigh- bzw. Mie-Streuung) Einfluss zu nehmen und über Shader die Szene entsprechend einzufärben. Die Anwendung benutzt lediglich die Darstellung des Himmels und der Sonne zu einer festen Tageszeit. Die atmosphärische Streuung wurde aufgrund einer Inkompatibilität entfernt. Die Nutzung des Zeitverlaufs für eine realistischer Simulation oder für besondere Situationen (Dämmerung, Nachtfahrten, ...) könnte gegebenenfalls als Erweiterung der Arbeit vorgenommen werden.

**HydraX** Dieses Plugin stammt ebenfalls von Xavier González. Es dient der realistischen Darstellung von Wasserflächen sowie darauf auftreffenden Reflexionen oder Lichtbrechungen. Dabei kann es mit dem Sonnenstand von SkyX kombiniert werden. Da es sich in einer frühen Entwicklungsphase befindet, wird lediglich die Erstellung einer unendlich großen Ozeanfläche mit festgelegter „Höhe“ unterstützt - deshalb wird bei Aktivierung des Wassers in der config.ini damit gleichzeitig eine Minimalhöhe für die Straße festgelegt, die nicht mehr unterschritten werden kann. Aus Performancegründen ist die Darstellung des Wassers standardmäßig deaktiviert.

**StereoManager** Das StereoManager-Plugin von „Thieum“<sup>33</sup> dient der Visualisierung der Grafik auf der PowerWall des Lehrstuhles. Zu diesem Zweck werden zwei ViewPorts in Ogre erstellt und an das Plugin übergeben. Für die Darstellung der Szene wird dann aus beiden Augpositionen heraus entsprechend der (einstellbaren) Parameter Augabstand und Fokuspunkt ein Bild gerendert und nebeneinander in den ViewPort geschrieben. Die Verwendung dieses Plugins kann innerhalb der config.ini-Datei festgelegt werden, zusätzlich existiert auch noch ein eigenes Konfigurationsfile<sup>34</sup>. In diesem kann beispielsweise auch ein Anaglyph-Modus<sup>35</sup> aktiviert werden.

**Newton Physics** Newton ist eine eigenständige, vollwertige Physikengine. Das Hauptaugenmerk dieser Entwicklung wurde laut dem Entwickler darauf gelegt, den höchstmöglichen Grad an Realismus zu bieten, den man bei Verwendung eines linearen Läasers für die physikalischen Gleichungen erreichen kann und gleichzeitig eine akzeptable Performanz zu erzielen. Für die Belegarbeit wurden nur zwei Komponenten der Physikengine genutzt: einerseits der

<sup>33</sup>Der Name des Erstellers in den OGRE User Foren.

<sup>34</sup>Die Datei stereo.cfg

<sup>35</sup>Das separate Rendern der Szene aus den beiden Augpositionen in den roten berziehungsweise blauen Farbkanal.

CharacterController, mit dem die Figur gesteuert wird. Zum Zweiten wurde um die Straße ein „Käfig“ aus Kollisionsobjekten gelegt, um den Fahrer auf der Spur zu halten. Diese Beschränkung der Freiheit ist aus zwei Gründen wichtig:

- a) Performanz

Die Erstellung der Kollisionsobjekte dauert je nach Größe der Objekte eine gewisse Zeit. Wird jeweils eine komplette Zelle als Kollisionsobjekt angelegt, sodass der Fahrer also auch quer fahren könnte, dauert die Generierung (nur des Physikkörpers) auf dem Testsystem bei einer Strassenzellgröße von  $2^8+1$  Punkten etwa 40 Sekunden. Das ist ein Vielfaches der Zeit, die die Zellgenerierung an sich benötigt. Dieser Umstand wurde als nicht akzeptabel bewertet.

- b) Streckenverlauf

Ist es dem Fahrer möglich, die vorgegebene Strecke zu verlassen, kann er theoretisch die Zellen an nicht dafür vorgesehenen Stellen verlassen und beispielsweise in eine der angrenzenden TerrainZellen fahren. Dieses Vorgehen würde allerdings unweigerlich zum Absturz führen, da sich die Terraingenerierung generell streng an den Straßenzellen orientiert.

Zusätzlich dazu stellt sich die Frage nach der Sinnhaftigkeit frei befahrbarer Zellen - im Hinblick auf die Verwendung als Ergometer ist der Fahrer notwendigerweise an die Straße gebunden.

Zusammenfassend soll nun im folgenden Kapitel die Performance der Applikation an sich sowie eine Beurteilung der verwendeten Algorithmen vorgenommen werden.

## 7 Eine Analyse der Arbeit

In diesem Kapitel soll die Qualität der erstellten Applikation sowie der gewählten Algorithmen untersucht werden. Dazu wird zuerst ein kurzes Benchmark der Terraingenerierung als „Herzstück“ der Anwendung vorgenommen. Danach soll die Effizienz der dynamisch anpassbaren Parameter beurteilt werden. Die daraus gewonnenen Erkenntnisse bilden danach das Fazit, welches das Kapitel beschließt.

### 7.1 Benchmark der Applikation

Die vorgestellte Software soll sowohl die Streckengenerierung als auch die eigentlich Fahrsimulation in Form einer Echtzeitanwendung realisieren. Verschiedene Anstrengungen wurden unternommen, um die Performance der Anwendung zu steigern sowie die Qualität des erzeugten Terrains zu erhöhen. Der deutlichste Performancegewinn wird dabei dadurch erzielt, dass Terrainzellen einen niedrigeren *Detailgrad* aufweisen als Straßenzellen. Der Detailgrad ist dabei die Anzahl der für den TriangleEdge-Algorithmus notwendigen Iterationsschritte<sup>36</sup>. Dieser Detailgrad kann in der config.ini-Datei eingestellt werden.

Die folgenden Benchmark-Tabellen gehen von einem Detailgrad von 8 bzw. 9 aus und stellen den Zeitbedarf dar, der für die Generierung von Zellen an sich benötigt wird. Des

---

<sup>36</sup>Bei einer Menge an Punkten pro Kante einer Straßenzelle von  $2^8+1$  Punkten ist der Detailgrad also „8“, bei einer „Kantenlänge“ von  $2^9+1$  Punkten wäre er „9“.

## 7 Eine Analyse der Arbeit

Straßenzelle			Terrainzelle	
Generierung der Straße	Generierung des Terrains	Schreiben in den Grafikspeicher	Generierung des Terrains	Schreiben in den Grafikspeicher
0,0147206	0,0786630	0,0176423	0,0197266	0,0012549
0,0131355	0,0838246	0,0188269	0,0203137	0,0011923
0,0134584	0,0812991	0,0172664	0,0220520	0,0011493
0,0140071	0,0822925	0,0189295	0,0199190	0,0012579
0,0143554	0,0770606	0,0176903	0,0217283	0,0010733
0,0138451	0,0862491	0,0174939	0,0206017	0,0012307
0,0125745	0,0848982	0,0175882	0,0198069	0,0011332
0,0117324	0,0805254	0,0175813	0,0217164	0,0011946
0,0157573	0,0784541	0,0177164	0,0202204	0,0012975
0,0125453	0,0859930	0,0176508	0,0206148	0,0011911
Durchschnitt:	0,0126132	0,0819260	0,0178386	0,0206700
Gesamtzeit:	0,1133777			0,0218674

**Abbildung 19:** Zeiten zur Generierung und Speicherung von Zellen (Detailgrad 8 - Initialisierungsphase - Werte in Sekunden)

Straßenzelle			Terrainzelle	
Generierung der Straße	Generierung des Terrains	Schreiben in den Grafikspeicher	Generierung des Terrains	Schreiben in den Grafikspeicher
0,0160714	0,1071100	0,0205802	0,0234233	0,0010990
0,0140601	0,0971655	0,0204055	0,0234340	0,0011005
0,0160146	0,0959732	0,0195500	0,0224741	0,0011078
0,0135298	0,0920799	0,0181300	0,0227248	0,0011796
0,0142229	0,1021350	0,0183727	0,0235849	0,0015290
0,0141818	0,0862409	0,0185117	0,0260862	0,001245
0,0149552	0,0854815	0,020976	0,0258101	0,0015118
0,0162456	0,0876533	0,0174995	0,0208479	0,0013017
0,0160088	0,0880653	0,0171320	0,0233856	0,0012111
0,0148907	0,0947018	0,0185965	0,0224668	0,0012806
Durchschnitt:	0,0150181	0,0935606	0,0189776	0,0234238
Gesamtzeit:	0,1275563			0,0245683

**Abbildung 20:** Zeiten zur Generierung und Speicherung von Zellen (Detailgrad 8 - Laufzeit - Werte in Sekunden)

Weiteren werden die einzelnen Schritte der Zellgenerierung genauer im Hinblick auf ihren Zeitbedarf unter die Lupe genommen. Vorher soll allerdings das Computer-System vorgestellt werden, auf dem alle Benchmarks vorgenommen wurden:

- Prozessor: Intel Core<sup>2</sup> Duo E6750 2x2,66GHz
- Hauptspeicher: 4096MB DDR2-800 (Taktung 5-5-15)
- Grafikkarte: Nvidia GeForce 8800 GTS
- Betriebssystem: Windows 7 Professional 64

Abbildung 19 zeigt, welche Zeit zur Zellgenerierung mit dem Detailgrad 8 während der Initialisierungsphase benötigt wird. Initialisierung bezeichnet hierbei die Phase, bevor die Echtzeitsimulation startet. Im Gegensatz dazu zeigt Abbildung 20 die zur Laufzeit auftretenden Zeiten. Die Abbildungen 21 und 22 stellen die Generierungszeiten für den Detailgrad 9 dar.

Es ist deutlich zu sehen, dass während der Laufzeit eine etwa 15% höhere Generierungszeit in allen Phasen benötigt wird. Der Mehrbedarf ist durch die Parallelisierung der Aufgaben natürlich erwartet worden und fällt relativ moderat aus. Da pro Zellwechsel im Durchschnitt eine neue Straßen- und drei neue Terrainzellen generiert werden müssen, ergeben sich also

## 7.1 Benchmark der Applikation

Straßenzelle			Terrainzelle		
Generierung der Straße	Generierung des Terrains	Schreiben in den Grafikspeicher	Generierung des Terrains	Schreiben in den Grafikspeicher	
0,0639459	0,3109880	0,0421319	0,0852665	0,0065362	
0,0696297	0,3259870	0,0429306	0,0845960	0,0051481	
0,0705006	0,3290290	0,0434118	0,0863574	0,0052219	
0,0598791	0,3175290	0,0435116	0,0906903	0,0057068	
0,0655087	0,3114860	0,0432912	0,0899235	0,0056849	
0,0750819	0,3253910	0,0431050	0,0852354	0,0052238	
0,0664414	0,3245190	0,0432897	0,0830497	0,0055026	
0,0688064	0,3225820	0,0426872	0,0813782	0,0058036	
0,0651309	0,3163080	0,0435796	0,0825478	0,0058977	
0,0712183	0,3261230	0,0438080	0,0832378	0,0054826	
Durchschnitt:	0,0676143	0,3209942	0,0431747	0,0852283	0,0056208
Gesamtzeit:	0,4317832			0,0908491	

**Abbildung 21:** Zeiten zur Generierung und Speicherung von Zellen (Detailgrad 9 - Initialisierungsphase - Werte in Sekunden)

Straßenzelle			Terrainzelle		
Generierung der Straße	Generierung des Terrains	Schreiben in den Grafikspeicher	Generierung des Terrains	Schreiben in den Grafikspeicher	
0,0764800	0,3943810	0,0618574	0,0872098	0,0075454	
0,0806862	0,4154950	0,0621177	0,0932788	0,0087319	
0,0691747	0,3894620	0,0500517	0,0967604	0,0076752	
0,0785701	0,3703470	0,0511614	0,0901435	0,0060225	
0,0791127	0,3665030	0,0436886	0,0935230	0,0066069	
0,0776919	0,3611820	0,0632655	0,0936378	0,0062221	
0,0807243	0,3766540	0,0724766	0,0915401	0,0065837	
0,0761291	0,3495540	0,0447011	0,0925181	0,0057007	
0,0740928	0,3467890	0,0527565	0,0953968	0,0079128	
0,0747736	0,3484940	0,0456216	0,0934468	0,0061542	
Durchschnitt:	0,0767435	0,3718861	0,0547698	0,0927455	0,0069155
Gesamtzeit:	0,5033994			0,0996610	

**Abbildung 22:** Zeiten zur Generierung und Speicherung von Zellen (Detailgrad 9 - Laufzeit - Werte in Sekunden)

Straßenzelle					Terrainzelle					
Initiale Punkte setzen	MDBU	MD	Normalen	Bäume	Initiale Punkte setzen	MDBU	MD	Normalen	Bäume	
0,0001486	0,0024049	0,0487695	0,0162890	0,0024084	0,0002193	0,0018647	0,0112359	0,0048502	0,0008663	
0,0003041	0,0049312	0,0532968	0,0231679	0,0020044	0,0001636	0,00099220	0,0134841	0,0046152	0,0007319	
0,0003383	0,0037792	0,0531608	0,0212852	0,0016346	0,0001344	0,0011335	0,0119009	0,0050279	0,0011355	
0,0001575	0,0041538	0,0531444	0,0213776	0,0016340	0,0001919	0,0010603	0,0112744	0,0049644	0,0011354	
0,0001559	0,0025409	0,0440315	0,0202113	0,0016355	0,0001682	0,0008839	0,0156038	0,0059045	0,0009569	
0,0003298	0,0041402	0,0453659	0,0175129	0,0020376	0,0002319	0,0019065	0,0111525	0,0049884	0,0008936	
0,0001536	0,0023896	0,0471080	0,0178305	0,0022997	0,0001763	0,0018101	0,0128108	0,0045391	0,0009085	
0,0003441	0,0037408	0,0461339	0,0192612	0,0021910	0,0002599	0,0018516	0,0113369	0,0045718	0,0005476	
0,0001494	0,0023316	0,0567696	0,0202373	0,0017295	0,0002285	0,0018739	0,0112724	0,0047304	0,0007522	
0,0004082	0,0054411	0,0485069	0,0208429	0,0024180	0,0001946	0,0009212	0,0110627	0,0060920	0,0012334	
Durchschnitt:	0,0002489	0,0034596	0,0488833	0,0197966	0,0020723	0,0002054	0,0016232	0,0123139	0,0049234	0,0008851
Gesamtzeit:	0,0744607				0,0199511					

**Abbildung 23:** Zeiten der einzelnen Phasen der Generierung (Detailgrad 8 - Laufzeit - Werte in Sekunden)

Straßenzelle					Terrainzelle					
Initiale Punkte setzen	MDBU	MD	Normalen	Bäume	Initiale Punkte setzen	MDBU	MD	Normalen	Bäume	
0,0004623	0,0211762	0,2273220	0,0020419	0,0022857	0,0004205	0,0049108	0,0757639	0,0194021	0,0027532	
0,0006002	0,0216988	0,2260080	0,0020728	0,0041711	0,0002116	0,0017468	0,0520488	0,0208167	0,0025777	
0,0004854	0,0151238	0,2310760	0,0817756	0,0041786	0,0003467	0,0040688	0,0497741	0,0180981	0,0017122	
0,0004930	0,0154170	0,2630460	0,0817610	0,0022640	0,0003548	0,0044458	0,0488394	0,0195312	0,0021718	
0,0006009	0,0161621	0,2601270	0,0895506	0,0040173	0,0004497	0,0039405	0,0654426	0,0221899	0,0015678	
0,0002968	0,0116298	0,2425490	0,0930503	0,0029003	0,0002987	0,0041759	0,0502640	0,0215417	0,0011961	
0,0007077	0,0171105	0,2613470	0,0859688	0,0033372	0,0003394	0,0019146	0,0465286	0,0200598	0,0013612	
0,0006985	0,0166040	0,2842180	0,0860579	0,0038698	0,0003752	0,0017771	0,0773429	0,0186726	0,0020490	
0,0003521	0,0113246	0,2493920	0,0835957	0,0052679	0,0004285	0,0039681	0,0514010	0,0180951	0,0017022	
0,0007146	0,0164650	0,2342860	0,0901174	0,0021860	0,0005902	0,0019860	0,0578785	0,0179180	0,0016458	
Durchschnitt:	0,0005432	0,0162712	0,2480061	0,0854792	0,0035327	0,0003815	0,0032932	0,0576285	0,0196327	0,0018717
Gesamtzeit:	0,3538323				0,0828077					

**Abbildung 24:** Zeiten der einzelnen Phasen der Generierung (Detailgrad 9 - Laufzeit - Werte in Sekunden)

## 7 Eine Analyse der Arbeit

Generierungszeiten von  $0,127\text{s} + 3 * 0,024\text{s} = \mathbf{0,2\text{s}}$  für den Detailgrad 8 und  $0,503\text{s} + 3 * 0,099\text{s} = \mathbf{0,8\text{s}}$  für den Detailgrad 9. Es ist offensichtlich, dass diese Generierungszeiten niemals im grafikparallelen Teil der Anwendung untergebracht werden könnten. Während der Generierungszeit tritt bei höheren Detailgraden ein deutlich wahrnehmbares „Ruckeln“ auf, was deren Nutzung quasi unmöglich macht. Möglicherweise könnte durch eine bessere Programmierung der Threads und der nötigen Algorithmen an dieser Stelle eine größere „Stabilität“ für den Simulationsteil erreicht werden und auf diese Art noch höhere Detailgrade „spielbar“ gemacht werden.

Da der Anteil der reinen Terrain-Generierung etwa 75% der gesamt benötigten Generierungszeit einnimmt, soll diese Phase noch einmal genauer betrachtet werden. Sie wurde deshalb für die Laufzeitbetrachtung in fünf Teile zerlegt:

- 1. Initialisierungs-Phase (Setzen der initialen Punkte)
- 2. MDBU-Phase (Durchführung des MDBU-Prozesses)
- 3. MD-Phase (Durchführung des MD-Prozesses, hier des TriangleEdge-Verfahrens)
- 4. Normalen-Phase (Berechnung der Oberflächennormalen)
- 5. Bäume & Gimmicks (Setzen von Landschaftsverschönerungen)

Die Abbildungen 23 und 24 zeigen die Zeiten der einzelnen Phasen während der Laufzeit der Simulation. Es ist leicht erkennbar, dass die Gesamtsumme aller Phasen eine deutlich kleinere Ausführungszeit ergibt als die Zeitmessung für die Generierung der gesamten Zelle<sup>37</sup>. Diese auf den ersten Blick paradox anmutende Tatsache beruht auf der Zeit, die der TerrainGenerator zum „Aufräumen“ der jeweiligen temporären Daten benötigt. Möglicherweise könnte durch effizientere Programmierung an dieser Stelle Zeit gespart werden.

Es zeigt sich, dass die Entscheidung für verschiedene Detailgrade von Straßen- beziehungsweise Terrainzellen einen enormen Einfluss auf die Performance der Simulation hat. Angenommen, es werden in einem Schritt neben einer Straßenzelle noch vier Terrainzellen angefordert. Hätten beide Zellformen den gleichen Detailgrad, müssten dafür 0,58s (Detailgrad 8:  $0,12755\text{s} + 4 * 0,1125\text{s}$ ) beziehungsweise 2,21s (Detailgrad 9:  $0,50339\text{s} + 4 * 0,42665\text{s}$ ) aufgebracht werden. Die Verringerung des Terraindetailgrades um eins ergibt dann allerdings jeweils 0,23s (Detailgrad der Straßenzelle 8) und 0,9s (Detailgrad der Straße 9). Das bedeutet in beiden Fällen eine Reduzierung der Generierungszeit um rund 60%! Der Verlust an optischer Qualität durch die verringerte Auflösung des Terrains ist dabei vernachlässigbar, da der Fahrer stets etwa eine halbe Zellenbreite Abstand dazu hat.

### 7.2 Analyse der veränderbaren Parameter

Nachdem im vorangegangenen Abschnitt die Performance der Generierungsprozesse beleuchtet wurde, soll jetzt im Anschluss eine Bewertung der Algorithmen vorgenommen werden, die für den Straßenverlauf verantwortlich zeichnen: die Steilheitsfunktion, die Hügeligkeitsfunktion und die Kurvigkeitsfunktion (siehe Abschnitt 5.1.3). Alle drei Algorithmen

<sup>37</sup>Für eine Straßenzelle des Detailgrades 8 beträgt die Gesamtsumme aller Generierungsphasen nur etwa 80% der gesamt benötigten Zeit ( $0,074\text{s} / 0,094\text{s} = 0,796$ ).

## 7.2 Analyse der veränderbaren Parameter

Werte: Faktor 1000, Erwartungswert: 0.0, Standard-Abweichung: 0,1											
10	-66,34	47,57	-13,51	-37,67	-150,23	-192	-345,98	-274,13	-406,51	-526,1	
10	-125,01	-157,05	-162,81	-17	82,96	61,77	24,44	73,17	70,39	25,28	
10	31,8	154,04	244,25	40,91	84,54	-79,68	-254,09	-193,86	-153,65	-84,54	
10	57,26	40,91	206,28	174,84	152,22	197,55	147,82	234,75	223,82	125	
10	13,46	22,25	-67,1	-112,57	-18,31	-206,32	-253,56	-109,69	-288,65	-258,53	
<b>Werte in Punkten:</b>											
10	-17,77	21,54	41,42	9,7	30,24	-43,74	-136,27	-53,95	-110,92	-143,78	
<b>Werte in Metern:</b>											
1	-1,78	2,15	4,14	0,97	3,02	-4,37	-13,63	-5,4	-11,09	-14,38	
<b>Höhenmeter gesamt:</b> -15,38											
Werte: Faktor 1000, Erwartungswert: -0,3, Standard-Abweichung: 0,1											
10	-219,97	-517,33	-776,71	-1065,75	-1256,34	-1565,04	-1939,77	-2255,73	-2788	-3114,67	
10	-389,32	-668,79	-1106,46	-1386,94	-1633,27	-2158,66	-2403,12	-2665,52	-2949,64	-3396,08	
10	-374,15	-707,25	-914,83	-1096,39	-1333,86	-1635,52	-1960,89	-2112,81	-2564,66	-2943,38	
10	-232,99	-416,3	-811,39	-1034,48	-1244,45	-1427,98	-1649,1	-1969,93	-2106,16	-2576,5	
10	-225,16	-509,81	-796,68	-1044,44	-1466,61	-1726,93	-1938,27	-2105,54	-2457,77	-2744,87	
<b>Werte in Punkten:</b>											
10	-288,32	-563,89	-881,21	-1125,6	-1386,91	-1702,83	-1978,23	-2221,91	-2573,25	-2955,1	
<b>Werte in Metern:</b>											
1	-28,83	-56,39	-88,12	-112,56	-138,69	-170,28	-197,82	-222,19	-257,32	-295,51	
<b>Höhenmeter gesamt:</b> -296,51											

**Abbildung 25:** Höhenverlauf für verschiedene Parameter bei mehreren Testläufen - Straßenhöhenwerte an den Zellgrenzen für jeweils fünf Testläufe zu zehn Zellen mit verschiedenen Erwartungswerten

basieren dabei auf Wahrscheinlichkeitsfunktionen. Dabei wurde bewusst auf „Globalität“<sup>38</sup> verzichtet, um dem Zufallsprinzip größeren Spielraum zu geben.

**Die Steilheitsfunktion** Diese Funktion ist hauptsächlich für die Menge überwundener Höhenmeter im Simulationsverlauf zuständig. Die Streuung der zugrunde liegenden Normalverteilung wurde bewusst sehr niedrig angesetzt, um über die Verschiebung des Erwartungswertes innerhalb des zulässigen Intervalls eine statistisch stabile Änderung des Höhenverlaufes erzeugen zu können. Die Tabelle in Abbildung 25 zeigt für zwei Verschiedene Erwartungswerte dieser Verhalten sehr eindrucksvoll: alle Testläufe beginnen bei einer Höhe von 1m über Null. Obwohl alle Testläufe im einzelnen sehr unterschiedliche Verläufe nehmen, stimmen die Durchschnittswerte aller Testläufe am Ende mit dem erwarteten Ergebnis überein.

**Die Hügeligkeitsfunktion** Diese Funktion bestimmt die „Unruhe“ des Höhenverlaufes der Strecke und ist daher maßgeblich daran beteiligt, wie der Fahrer den Weg subjektiv wahr-

<sup>38</sup>Globalität meint hier: die Festsetzung beispielweise einer bestimmten Menge Höhenmeter, die innerhalb einer gewissen Distanz überwunden werden muss.

nimmt. Auch hier ist wieder eine Normalverteilung implementiert worden, die über die Verschiebung des Erwartungswertes bei geringer Streuung den Einfluss auf die Strecke bestimmt. Geringe Werte bedeuten hierbei einen als „stetig“ wahrgenommenen Streckenverlauf, höhere Werte führen zu „Hügeln“ und damit zu Abwechslung. Möglicherweise kann in einer fertigen Simulation die Festlegung des Erwartungswertes ebenfalls zufällig erfolgen, um eine größtmögliche Abwechslung zu garantieren. Es muss dabei nur bedacht werden, dass, auch wenn dieser Parameter keinen Einfluss auf die Differenz der Höhenmeter der Strecke hat, er dennoch durch lokale Steilheitsänderungen (die „Hügel“) sehr wohl einen großen Einfluss auf das Fahrverhalten und die subjektive Wahrnehmung der Strecke beim Fahrer hat.

**Die Kurvigkeitsfunktion** Die dritte wichtige Funktion ist für die Bestimmung der Kurven des Kurses zuständig. Dieser Parameter ist auch der einzige der drei, der über die aktuelle Zelle hinaus Bedeutung besitzt: er wird sowohl für die Kurvigkeitsinnerhalb der Zelle als auch die Bestimmung des globalen Streckenverlaufes herangezogen. Die gitterartige Anordnung der einzelnen Zellen bringt es allerdings mit sich, dass die Kurvigkeitsinnerhalb einer Zelle weitaus weniger stark wahrgenommen wird als der Kurvenverlauf der Gesamtstrecke an sich<sup>39</sup>.

### 7.3 Fazit

Zusammenfassend lässt sich festhalten, dass die Simulation eine schnelle und stabile Generierung eines glaubhaften und stetigen, wenn auch nicht 100% realistischen Straßenverlaufes erlaubt. Die drei veränderbaren Parameter bieten genügend Spielraum, um die Strecke den Erfordernissen des Fahrers beziehungsweise der Simulation anpassen zu können. Die Verwendung einer Verkettung „lokaler Zufälle“ bietet hierfür gegenüber globaleren Methoden den deutlichen Vorteil geringerer Vorhersagbarkeit und damit „Langweiligkeit“. Dabei wurden die Werte der Zufallsfunktionen allerdings so weit angepasst, dass die anpassbaren Simulationsparameter vorzüglich zur Bestimmung von „Tendenzen“ genutzt werden können. Diese Tendenzen schlagen sich dann auch in einer sehr guten Vorhersagbarkeit der Durchschnittswerte nieder. Man könnte also durchaus von einem „eng gesteuerten Zufall“ sprechen, der an dieser Stelle genutzt wird. Es ist ebenfalls deutlich geworden, dass die hier vorgestellte Lösung der gepufferten Erzeugung der Zellen im Hintergrund eine Detailliertheit und Auflösung des Terrains ermöglichen, die niemals durch die Erzeugung des Terrains parallel zur Grafikbearbeitung erreicht werden könnte. Der deutlich sichtbare Nachteil dabei ist die Zerlegung des Terrains in Zellen fester Form und Größe, die beim Übergang zwischen zwei Zellen recht unschön sichtbar „umgeschaltet“ werden. Die Integrierung einer Sichtfeldbegrenzung würde an dieser Stelle durchaus eine deutliche optische Verbesserung darstellen.

Die Vorverarbeitung der Terraingenerierung mittels des MDBU-Algorithmus hat sich dabei als sehr einfache, effiziente und praktische Methode erwiesen, um die Flexibilität und Geschwindigkeit von computergeneriertem Zufallsterrain mit den Anforderungen einer konkreten Anwendung in Form von Constraints zu verbinden.

<sup>39</sup>Am deutlichsten wird der Einfluss dieses Parameters innerhalb einer Straßenzelle bei „Geradeausfahrten“, also wenn global gesehen keine Kurve vorliegt. Dann ergeben sich auf der Strecke nämlich kleine Abweichungen der idealen Geradeauslinie und erhöhen somit einerseits die „Glaubwürdigkeit“ des vorliegenden Straßenverlaufes wie auch die Aufmerksamkeit beim Fahrer.

## 8 Ausblick und Erweiterungsmöglichkeiten

Nachdem das Hauptziel der Belegarbeit - die dynamische Streckengenerierung - erfolgreich umgesetzt wurde, könnte nun ein Ausbau der Simulation erfolgen. Verschiedene vorstellbare Erweiterungsmöglichkeiten sollen im Folgenden kurz vorgestellt werden:

**Ein Fahrrad als Eingabegerät** Am Lehrstuhl Computergrafik wurde ein stationäres Fahrrad installiert, welches als Eingabegerät für die Simulation erschlossen werden könnte. Dazu müsste die Bewegungskomponente der Simulation entsprechend angepasst werden. Dazu müsste ein externes Programm geschrieben werden, das die bereits vorhandene Kommunikations- und Steuerungskomponente der Belegarbeit ansteuert. Auf diese Art und Weise könnte ein höherer Realismusgrad beim Fahrer erzeugt werden. Diese Art der Steuerung könnte möglicherweise mit einem Headtracking verbunden werden, um das Sichtfeld des Fahrers unabhängig von der Ausrichtung des Fahrrads selbst ändern zu können. Aufgrund der Einschränkungen der Projektion auf die Powerwall sollte an dieser Stelle aber über die Nutzung einer VR-Brille nachgedacht werden.

**Eine physikalisch korrekte Fahrsimulation** Ein im Vorfeld der Belegarbeit angesprochener, aufgrund der Komplexität aber verworfener Punkt ist die Umsetzung eines physikalisch korrekten Fahrmodells eines Fahrrads. Die zugrunde liegende Physik ist hochkomplex und mit größter Wahrscheinlichkeit nicht vom durch die verwendete Physikengine integrierten linearen Löser zu handhaben. Es müsste also das komplette physikalische Modell erstellt werden sowie geeignete Vereinfachungen getroffen werden. Diese werden mit hoher Wahrscheinlichkeit nötig werden, da die Ergebnisse einer Physiksimulation grundsätzlich parallel zur Grafik bereitgestellt werden müssen und somit nur relativ wenig Rechenzeit dafür zur Verfügung steht. Möglicherweise könnte ein Teil der Simulation durch die Wahl einer geeigneten Grafikkarte mit Physikbeschleunigung übernommen werden.

**Eine realistische Vegetationskomponente** Die Umsetzung der Bäume in der Belegarbeit ist wenig mehr als die grundsätzliche Demonstration der Machbarkeit an sich. Die Darstellung genügt keinerlei optischen Ansprüchen - doch das sollte sie auch gar nicht. Die Integration beziehungsweise Entwicklung einer Vegetationskomponente ist eine hochkomplexe Materie. Für die Umsetzung einer solchen Komponente müsste wahrscheinlich ein großer Teil des Programmcodes umgeschrieben werden.

**Der Einbau vorgenerierter Zellen** Die Terraingenerierung erfolgt momentan ausschließlich computergesteuert. Wie bereits vorgestellt, bieten solche Modelle allerdings nicht nur Vorteile, sie schränken auch die erzeugte Geometrie in vielerlei Hinsicht ein<sup>40</sup>. Diese Einschränkungen könnte man zumindest teilweise umgehen, indem entsprechende Abschnitte oder Zellen von Hand vormodelliert werden und lediglich an entsprechender Stelle in das Programm eingebunden werden. In Verbindung mit der Scripting-Erweiterung ergäben sich hier eine Menge neuer Möglichkeiten, das dargestellte Terrain sowie die Fahrsimulation an sich „interessanter“ zu gestalten.

<sup>40</sup>Die Unmöglichkeit, mittels des MD-Algorithmus so einfache Formationen wie Überhänge oder Brücken zu realisieren sei hier beispielhaft genannt.

**Ein echtes Scripting** Für die Belegarbeit war die Umsetzung eines „echten“ Scriptings angedacht gewesen, doch wurde sie letztlich nicht umgesetzt. Sie bietet dennoch eine attraktive Möglichkeit, sowohl im Vorfeld Einfluss auf die Zufallsgenerierung des Terrains zu nehmen, als auch für Abwechslung während der Fahrten zu sorgen.

**Die Erweiterung der Simulation zum Ergometer** Sobald ein reales Fahrrad als Eingabegeärt realisiert wurde, kann die Simulation auch zum voll funktionsfähigen Ergometer erweitert werden. Dazu müsste entweder in das Programm eine Auswertung der Vitalwerte des Fahrers integriert werden, oder diese Auswertung erfolgt in einem externen Programm und es wird lediglich die Kommunikationskomponente der Simulation genutzt, um die Parameter entsprechend einzustellen.

**Weitere optische Verschönerungen** Der Einbau von optischen Effekten wie Beleuchtungsmodelle für Nachtfahrten, die Umsetzung von Wind- und Wettereffekten und ähnliche Verschönerungen würden zu einer deutlich höheren Motivation beim Fahrer führen und den „Wiederspielwert“ erhöhen. Dabei bieten gerade die Wind- und Wettereffekte die Möglichkeit, auch physikalischen Einfluss auf die Simulation der Fahrverhältnisse nehmen zu können.

Es ist deutlich zu erkennen, dass in der Software einiges an Potential steckt. Insbesondere die Erweiterung zum Ergometer bietet eine interessante Perspektive, da die Anpassung des Streckenverlaufes an die Vitalwerte des Fahrers die Möglichkeit eines optimalen Trainings bieten.

## Literatur

- [Belhadj 2007] BELHADJ, Farès: *Terrain Modeling: A Constrained Fractal Model*. ACM Press : Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa, Grahamstown, South Africa, p. 197 - 204, 2007
- [Belhadj und Audibert 2005a] BELHADJ, Farès ; AUDIBERT, Pierre: *Modeling Landscapes with Ridges and Rivers*. ACM Press : VRST '05: Proceedings of the ACM symposium on Virtual reality software and technology, p. 151 - 154, 2005
- [Belhadj und Audibert 2005b] BELHADJ, Farès ; AUDIBERT, Pierre: *Modeling Landscapes with Ridges and Rivers: bottom up approach*. ACM Press : GRAPHITE '05: Proceedings of the 3rd international conference on Computergraphics and interactive techniques in Australasia and SouthEast Asia, p. 447 - 450, 2005
- [Bouknight 1970] BOUKNIGHT, Jack: *A procedure for generation of three-dimensional half-toned computer graphics presentations*. ACM Press : Communications of the ACM 13, 9 p. 527 - 536, 1970
- [Fahrssimulator Universität Würzburg 2011] FAHRSIMULATOR UNIVERSITÄT WÜRZBURG: *Der Fahrssimulator der Universität Würzburg*. Website. 2011. – URL <http://www.psychologie.uni-wuerzburg.de/methoden/forschung/technik/fahrssimulator.php.de>; zuletzt besucht am 09.03.2011.
- [Irrlicht 2010] IRRLICHT: *Irrlicht Grafikengine*. Website. 2010. – URL <http://irrlicht.sourceforge.net/>; zuletzt besucht am 24.10.2010.
- [Kaußner 2003] KAÜSSNER, A.: *Dissertation: Dynamische Szenerien in der Fahrssimulation*. Bayrische Julius-Maximilians-Universität Würzburg, 2003
- [Kaußner u. a. 2003] KAÜSSNER, A. u. a.: *Fahrssimulator-Datenbasen mit dynamisch veränderbaren Straßennetzwerken*. VDI Verlang : VDI-Berichte Nr. 1745. Simulation und Simulatoren - Mobilität virtuell gestalten., 2003
- [Martz 1996] MARTZ, Paul: *Generating Random Fractal Terrain*. Website. 1996. – URL <http://www.gameprogrammer.com/fractal.html>; zuletzt besucht am 24.10.2010.
- [Metz 2009] METZ, B.: *Dissertation: Worauf achtet der Fahrer? - Steuerung der Aufmerksamkeit beim Fahren mit visuellen Nebenaufgaben*. Bayrische Julius-Maximilians-Universität Würzburg, 2009
- [Miller 1986] MILLER, Gavin S.: *The Definition and Rendering of Terrain Maps*. ACM Press : SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques, p. 39 - 48, 1986
- [OGRE 3D 2010] OGRE 3D: *OGRE 3D Grafikframework*. Website. 2010. – URL <http://www.ogre3d.org>; zuletzt besucht am 24.10.2010.
- [PlanetSide 2010] PLANETSIDE: *TerraGen*. Website. 2010. – URL <http://www.planetside.co.uk/>; zuletzt besucht am 24.10.2010.

## Literatur

- [RacerMate 2006] RACERMATE: *CompuTrainer*. Website. 2006. – URL <http://www.racermateinc.com/computrainer.asp>; zuletzt besucht am 24.10.2010.
- [TidalWare 2009] TIDALWARE: *TidalWare Blog - Ogre and Visual Studio Beginners FAQ*. Website. 2009. – URL <http://blog.tidalware.com/2009/06/ogre-and-visual-studio-beginners-faq/>; zuletzt besucht am 24.10.2010.
- [TU Dresden 2010] TU DRESDEN: *Online-Kompendium Straßenentwurf*. Website. 2010. – URL [http://strassenentwurf.elcms.de/content/index\\_ger.html](http://strassenentwurf.elcms.de/content/index_ger.html); zuletzt besucht am 24.10.2010.
- [WIKPEDIA 2010] WIKPEDIA: *Liste der technischen Regelwerke und amtlichen Bestimmungen für das Straßenwesen*. Website. 2010. – URL [http://de.wikipedia.org/wiki/Liste\\_der\\_technischen\\_Regelwerke\\_und\\_amtlichen\\_Bestimmungen\\_für\\_das\\_Straßenwesen](http://de.wikipedia.org/wiki/Liste_der_technischen_Regelwerke_und_amtlichen_Bestimmungen_für_das_Straßenwesen); zuletzt besucht am 02.10.2010.