UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TECHNISCHE INFORMATIK

Verbesserte Wassereffekte für die Simulationsumgebung MARS

Improved water effects for the simulation environment MARS

Bachelorarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
John Paul Jonte

ausgegeben und betreut von
Prof. Dr.-Ing. Erik Maehle

mit Unterstützung von
Dipl.-Inf. Thomas Tosik

Lübeck, den 7. Oktober 2014

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Lübeck, den 7. Oktober 2014

Kurzfassung

In dieser Arbeit wurden grafische Effekte implementiert, die die Darstellung des Wassers in der Simulationsumgebung MARS verbessern. Die Wasseroberfläche und die Unterwasseransicht haben mehrere glaubwürdige grafische Darstellungen erhalten. In der Unterwasseransicht werden Partikel angezeigt, die im Wasser schwimmen und z.B. Algen oder andere Pflanzen darstellen. Fährt ein Fahrzeug an der Wasseroberfläche, so hinterlässt es eine kurze Schaumspur. Kommt ein Fahrzeug in die Nähe des Bodens, wird Sediment aufgewirbelt.

Abstract

In this thesis graphical effects were implemented in order to improve the rendering of water in the MARS simulator. The water surface and underwater view were enhanced with multiple effects. The underwater view contains particles suspended in the water. These particles represent plant matter and sediment. Vehicles traveling along the water surface leave a short foam trail. Vehicles coming close to the ocean floor cause clouds of sediment to rise.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Zielsetzung 1
1.2 Gliederung 1

2 Grundlagen **3**
2.1 Verwendete Technologien 3
2.1.1 jMonkeyEngine 3 3
2.1.2 MARS 5
2.1.3 Grafikpipeline und OpenGL Shading Language 6
2.1.4 Simplex Noise 7
2.2 Verwandte Arbeiten 8
2.2.1 osgOcean 8
2.2.2 OGRE 10
2.2.3 Triton 10

3 Umsetzung und Implementierung **11**
3.1 Verbindung von Projected Grid und Advanced Water Filter 11
3.2 Partikel im Wasser 13
3.3 Schaumspuren von Fahrzeugen 16
3.4 Sedimentaufwirbelung 18
3.5 Integration in MARS 19

4 Evaluation **21**
4.1 Testaufbau 21
4.2 Testszenario 21
4.3 Testergebnisse 23

5 Zusammenfassung und Ausblick **29**

Abbildungsverzeichnis **31**

Tabellenverzeichnis **33**

Abkürzungsverzeichnis **35**

Literaturverzeichnis **37**

1 Einleitung

Für die Entwicklung autonomer Unterwasserroboter (Autonomous Underwater Vehicle, AUV) ist Simulation ein wichtiges Werkzeug. Mit einem Simulator lässt sich die Software des Roboters z.B. ohne vollständige Hardware testen. Speziell bei der Entwicklung von Unterwasserrobotern muss man kein Gewässer aufsuchen, um die Software zu testen. Auch besteht keine Gefahr, dass die Hardware beschädigt wird. Damit die Simulation ihren Zweck erfüllt, ist es wichtig, dass die Umwelt des Roboters auch glaubwürdig für die simulierte Kamera dargestellt wird. Über die Kamera können mit Hilfe von Bildalgorithmen viele Informationen über die Umwelt gesammelt werden, darunter auch Informationen über Bioindikatoren des Gewässers [1]. Die grafische Ansicht sollte auch die physikalischen Gegebenheiten des Wassers widerspiegeln. Daher spielt die Darstellung des Wassers eine wichtige Rolle in der Simulation von AUVs.

1.1 Zielsetzung

Ziel dieser Arbeit ist es, die grafische Darstellung des Wassers in der Simulationsumgebung MARS um einige Effekte zu erweitern, sodass eine glaubwürdige Darstellung entsteht. Die vorhandenen Darstellungen der Wasseroberfläche, *Projected Grid* und *Advanced Water Filter*, sollen so verbunden werden, dass die Informationen über die Wellen von anderen Komponenten der Simulationsumgebung verwendet werden können. Die Darstellung des Wassers soll sowohl über als auch unter Wasser realistisch aussehen. Unter Wasser sollen Partikel zu sehen sein, die im Wasser schwimmen und die Sicht der Kamera teilweise verdecken [2]. An der Oberfläche fahrende Fahrzeuge sollen einen Streifen aus Schaum hinter sich her ziehen. Wenn ein Fahrzeug auf den Grund des Gewässers stößt, soll es Sediment aufwirbeln, welches die Sicht vermindert.

1.2 Gliederung

Zuerst werden die Grundlagen der Arbeit beschrieben (siehe Kap. 2). Dazu gehören die verwendeten Technologien (siehe Kap. 2.1) und verwandte Arbeiten (siehe Kap. 2.2). Als nächstes wird auf die Umsetzung der Effekte eingegangen (siehe Kap. 3). Darauf folgt eine Evaluation der implementierten Effekte (siehe Kap. 4). Das letzte Kapitel besteht aus einer Zusammenfassung und einem Ausblick (siehe Kap. 5).

2 Grundlagen

In diesem Kapitel werden die für die Wassereffekte verwendeten Technologien beschrieben. Dazu gehört die jMonkeyEngine 3, die die Basis für die Simulationsumgebung MARS ist, MARS selber, die OpenGL Shading Language und das in einem Effekt verwendete Simplex Noise. Als Letztes wird auf verwandte Arbeiten eingegangen.

2.1 Verwendete Technologien

2.1.1 jMonkeyEngine 3

Die jMonkeyEngine ist eine Open-Source Engine für 3D-Spiele, die in Java geschrieben ist [3]. Die aktuelle Version der jMonkeyEngine ist 3.1. Der Aufbau einer Szene wird durch den Szenengraphen bestimmt, der hierarchisch die in der Szene enthaltenen Objekte beschreibt. Die Engine ist stark Shader-basiert. Grafische Effekte werden komplett mit Shadern realisiert. Die grafische Darstellung von Objekten wird durch ein Materialsystem bestimmt. Mit dem Asset-System werden Ressourcen wie Materiale, Modelle und Texturen verwaltet. Darüber hinaus lassen sich leicht Spezialeffekte mit dem Partikelsystem oder Postprocessing-Filtern erzeugen. Physikalische Simulationen werden durch die Bullet-Engine[1] ermöglicht.

Die jMonkeyEngine bietet zwei Komponenten, die in dieser Arbeit für die Wasseroberfläche und die Unterwasseransicht verwendet wurden. Diese Komponenten sind das Projected Grid und der Advanced Water Filter.

Projected Grid

Das Projected Grid ist ein aus der älteren jMonkeyEngine 2 stammender Effekt, der auf Basis eines Rasters von Punkten mit zufällig generierter Höhe eine Wasseroberfläche mit Wellen erzeugt (siehe Abb. 2.1). Aus dem Raster wird eine Geometrie erzeugt, die die Wasseroberfläche darstellt. Auf die Geometrie wird dann ein Shader angewendet, der Transparenz sowie Brechungs- und Reflektionseffekte erzeugt. Nachteil des Projected Grid ist, dass es nur die Wasseroberfläche darstellt. Unter Wasser bietet es keine Darstellung.

[1] `http://bulletphysics.org/`

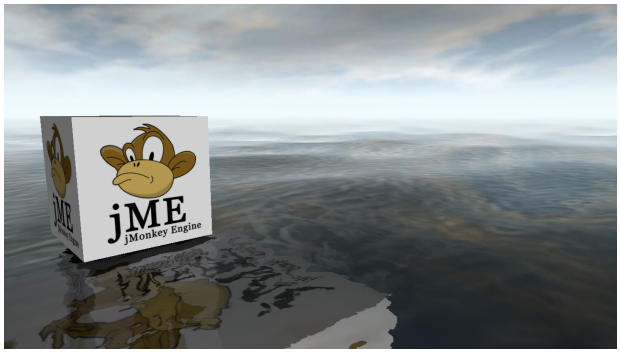


Abbildung 2.1: Das Projected Grid im Einsatz. Dieser Filter bietet nur die gezeigte Oberflächendarstellung mit Reflektion, Lichtbrechung und Wellen. Eine Unterwasseransicht ist nicht vorhanden.

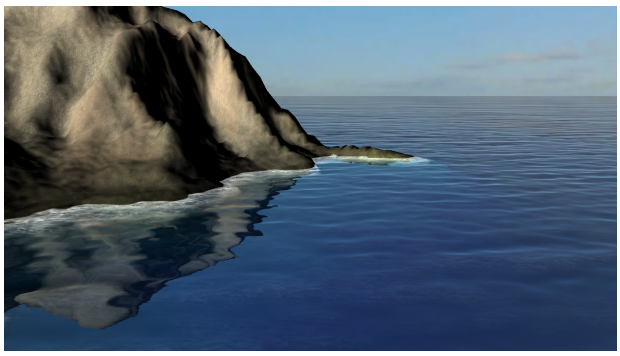


Abbildung 2.2: Der Advanced Water Filter im Einsatz. Zu sehen sind Reflektion, Lichtbrechung, Wellen und Schaum. Darüber hinaus bietet der Filter noch eine Unterwasseransicht (siehe Abb. 2.3).

Advanced Water Filter

Der Advanced Water Filter simuliert eine Wasseroberfläche mit Wellen und Schaum (siehe Abb. 2.2) sowie eine Unterwasseransicht mit Kaustik-Effekten (siehe Abb. 2.3). Im Gegensatz zum Projected Grid wird beim Advanced Water Filter der Effekt komplett im Shader-Code erzeugt. Daher können die Informationen über die Wellen nicht in anderen Komponenten, z.B. für die Auftriebsberechnung, verwendet werden. Mit Hilfe des Depth-Buffers wird die Position eines Pixels berechnet und basierend darauf entschieden, ob dieser Pixel über oder unter Wasser ist. Anschließend wird aus verschiedenen Parametern und vorgegebenen Texturen die endgültige Pixelfarbe berechnet.

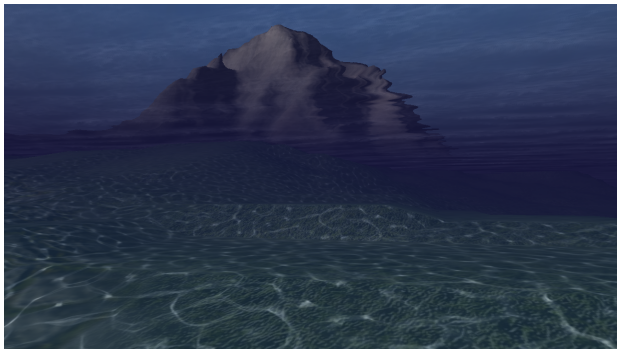


Abbildung 2.3: Die Unterwasseransicht des Advanced Water Filters mit Kaustik-Effekten und Lichtbrechung.

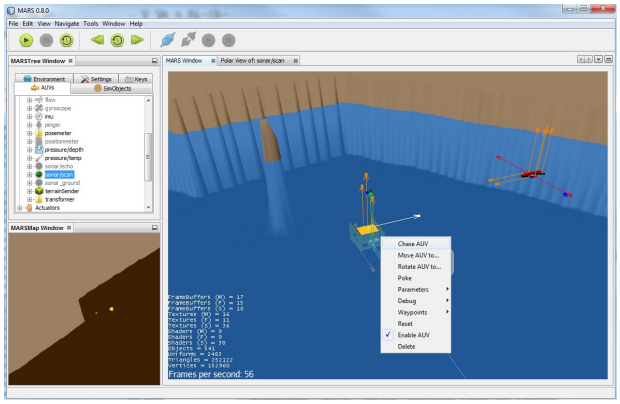


Abbildung 2.4: Die Simulationsumgebung MARS. Zu sehen sind zwei AUVs in der simulierten Umgebung sowie eine Baumansicht der Aktoren und Sensoren der AUVs. Links unten befindet sich eine Karte der Umgebung aus der Vogelperspektive.

2.1.2 MARS

MARS (MArine Robotics Simulator) ist eine am Institut für Technische Informatik entwickelte echtzeitfähige Simulationsumgebung für AUVs [4]. MARS wurde in Java mit Hilfe der jMonkeyEngine programmiert. Es baut auf der NetBeans Platform[2] auf und kann deshalb leicht mit Plugins erweitert werden.

MARS unterstützt die gleichzeitige Simulation mehrerer AUVs (siehe Abb. 2.4). Über ROS[3] (Robot Operating System) kommuniziert es mit der Software eines AUVs. Es unterstützt eine Vielzahl an Sensoren und Aktoren, wie z.B. Motoren, Kameras und Sonar.

[2] `https://netbeans.org/features/platform/`
[3] `http://www.ros.org/`

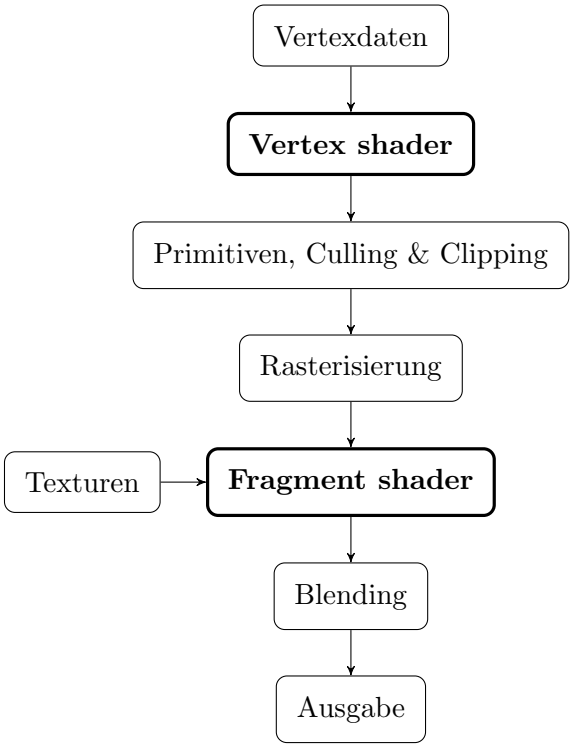


Abbildung 2.5: Eine vereinfachte Darstellung der Grafikpipeline. Die Grafikpipeline bestimmt, wie aus einer dreidimensionalen Beschreibung einer Szene ein zweidimensionales Bild berechnet wird. Die programmierbaren Stufen, der Vertex Shader und der Fragment Shader, sind dick umrandet.

2.1.3 Grafikpipeline und OpenGL Shading Language

Die meisten der in dieser Arbeit implementierten Effekte wurden mit der OpenGL Shading Language realisiert. Die OpenGL Shading Language ist eine Programmiersprache für Programme (Shader), die auf einer GPU (Graphics Processing Unit) ausgeführt werden, um grafische Effekte zu erzeugen [5]. Die Syntax der OpenGL Shading Language basiert auf der Programmiersprache C. Shader kommen an verschiedenen Stellen in der Grafikpipeline zum Einsatz, daher wird zunächst diese vereinfacht beschrieben.

Die Grafikpipeline beschreibt die Schritte, die nötig sind, um aus der dreidimensionalen Beschreibung einer Szene die zweidimensionale Darstellung dieser Szene auf einem Bildschirm zu erhalten (siehe Abb. 2.5).

Die erste Stufe der programmierbaren Grafikpipeline ist der Vertex Shader. Die Aufgabe des Vertex Shaders ist, die Projektion der Eckpunkte (Vertices) der Geometrien von Weltkoordinaten in Bildschirmkoordinaten durchzuführen. Der Vertex Shader kann mit der OpenGL Shading Language frei programmiert werden und erlaubt somit grafische Effekte, die auf der Veränderung der Geometrie basieren.

Als Nächstes werden aus den transformierten Vertices Primitiven (meist Dreiecke) erzeugt und Culling und Clipping durchgeführt. Beim Culling werden Flächen, die von der Kamera aus nicht sichtbar sind, verworfen, damit diese nicht weiter verarbeitet werden müssen. Beim Clipping werden Linien und Teilflächen entfernt, die aus dem Sichtbereich der Kamera herausragen. Bei der anschließenden Rasterisierung wird bestimmt, welche Pixel zu einer Primitive gehören. Die Pixel, die aus der Rasterisierung entstehen, werden Fragmente genannt. Sie stimmen nicht unbedingt mit den Pixeln auf dem Bildschirm überein, da es bei überlappenden Primitiven mehrere Fragmente pro Pixel geben kann, die später im Blending vermischt werden.

Die Fragmente werden vom Fragment Shader weiterverarbeitet. Der Fragment Shader weist jedem Fragment basierend auf Texturen, Lichtquellen und anderen Informationen einen Farbwert zu. Auch diese Stufe kann mit der OpenGL Shading Language frei programmiert werden und erlaubt eine Vielzahl an Effekten.

Der letzte Schritt vor der Ausgabe auf dem Bildschirm ist das Blending. Beim Blending werden die Farbwerte der vom Fragment Shader bearbeiteten Fragmente vermischt und einem Bildschirmpixel zugewiesen.

Die OpenGL Pipeline enthält mittlerweile weitere programmierbare Stufen: *Tesselation Control Shader*, *Tesselation Evaluation Shader* und *Geometry Shader*. In dieser Arbeit wurden diese Stufen nicht verwendet. Daher wird darauf nicht weiter eingegangen.

2.1.4 Simplex Noise

Simplex Noise ist eine von Ken Perlin entwickelte Rauschfunktion, die eine verbesserte Variante seines ursprünglichen Perlin Noise ist [6]. Im Folgenden wird nur das zweidimensionale Simplex Noise beschrieben. Das Verfahren lässt sich jedoch leicht auf drei oder mehr Dimensionen erweitern. So lässt sich z.B. für drei Dimensionen und die Zeit ein Rauschen generieren.

Die Grundlage des Simplex Noise ist eine Aufteilung des Raums in Simplices. Im Zweidimensionalen sind dies gleichseitige Dreiecke. Jedem Eckpunkt eines Simplex wird ein zufällig gewählter Gradient, also ein Richtungsvektor, zugewiesen (siehe Abb. 2.6). Da das Rauschen für feste Parameter immer gleich sein soll, erfolgt diese Zuweisung pseudozufällig, d.h. derselbe Eckpunkt erhält auch immer denselben Gradienten.

Um für einen Punkt (x, y) den Funktionswert zu berechnen, wird zunächst der Simplex bestimmt, in dem der Punkt liegt. Zur Bestimmung des Simplex werden die Koordinaten so geschert, dass aus jeweils zwei Simplices ein Quadrat entsteht. Im resultierenden quadratischen Raster lässt sich der zugehörige Simplex leicht bestimmen (siehe Abb. 2.7).

Der Funktionswert ergibt sich aus einer von der Entfernung von den Eckpunkten abhängigen gewichteten Summierung der Gradienten. In dieser Arbeit wurde das Simplex Noise in Form eines OpenGL-Shaders verwendet.

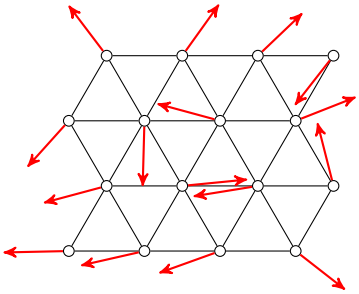


Abbildung 2.6: In Simplices aufgeteilte Fläche mit zufällig zugewiesenen Gradienten (rot). Das Rauschen wird aus einer Interpolation der Gradienten erzeugt.

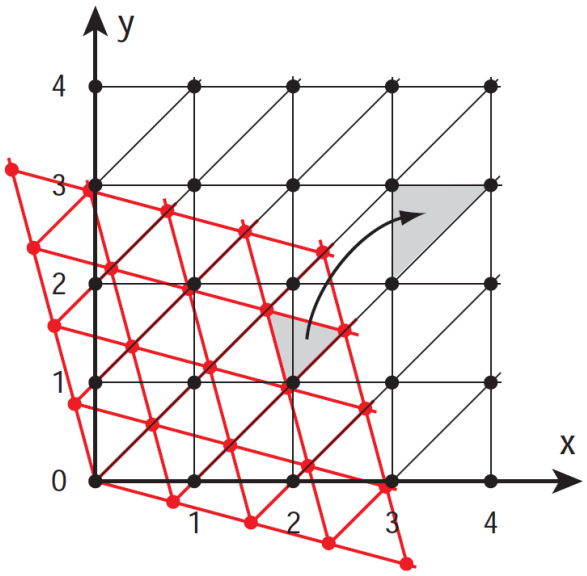


Abbildung 2.7: Die Scherung der Koordinaten bewirkt, dass ein einfach zu handhabendes, quadratisches Raster entsteht [6].

2.2 Verwandte Arbeiten

2.2.1 osgOcean

osgOcean ist eine grafische Wassersimulation auf Basis von OpenSceneGraph[4], die als Teil eines von der EU geförderten Projektes entwickelt wurde [7]. OpenSceneGraph ist ein in C++ geschriebenes Toolkit für 3D-Anwendungen. osgOcean ist ein Open-Source-Projekt und simuliert eine Wasseroberfläche mit Wellen, Schaum, Lichtbrechung,

[4] `http://www.openscenegraph.org/`

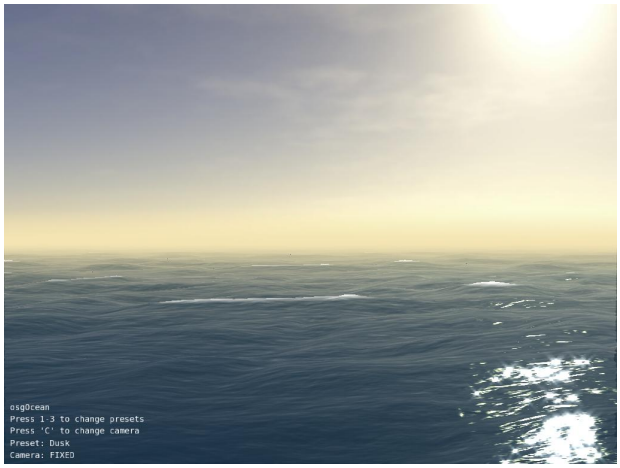


Abbildung 2.8: Die Wasseroberfläche in osgOcean mit Wellen, Reflektionen, Lichtbrechung und Schaum. Darüber hinaus bietet es eine Unterwasseransicht.

Abbildung 2.9: Die Wasseroberfläche in OGRE/Hydrax. Zu sehen sind Wellen, Reflektionen, Lichtbrechung, Kaustik-Effekte und Schaum. Hydrax bietet auch eine Unterwasseransicht.

Reflektionen und anderen Effekten sowie eine Unterwasseransicht (siehe Abb. 2.8). Die Simulation basiert auf Algorithmen, die in einem Paper von Jerry Tessendorf beschrieben wurden [8].

Da osgOcean in C++ geschrieben wurde, ist es für die Nutzung im Java-basierten MARS ungeeignet. Darüber hinaus ist die LGPL-Lizenz von osgOcean restriktiv im Hinblick auf die Verwendung in anderen Projekten. Daher wird osgOcean nicht in MARS verwendet.

Abbildung 2.10: Die Wasseroberfläche in Triton mit Wellen, Reflektionen, Lichtbrechung und Schaum. Zusätzlich dazu gibt es eine Unterwasseransicht.

2.2.2 OGRE

OGRE[5] ist eine in C++ geschriebene cross-platform 3D-Engine und ein Open-Source-Projekt mit der MIT-Lizenz. OGRE bietet eine Wassersimulation in Form der Bibliothek Hydrax[6]. Hydrax simuliert eine Wasseroberfläche und eine Unterwasseransicht mit Reflektionen und Lichtbrechung sowie Tiefen-, Kaustik-, Licht- und Schaumeffekten (siehe Abb. 2.9).

Da OGRE in C++ geschrieben ist, lässt sich die Wassersimulation mit Hydrax nicht in MARS verwenden.

2.2.3 Triton

Triton[7] ist ein kommerzielles SDK zur 3D-Simulation von Wasser. Es bietet u.a. eine Vielzahl an Wellenformen, Reflektionen, Lichtbrechung, durch Objekte erzeugten Schaum und Spritzwasser und eine Unterwasseransicht (siehe Abb. 2.10).

Triton kann nur mit C++ und C# verwendet werden und ist daher für die Benutzung in MARS ungeeignet. Darüber hinaus sind die Lizenzkosten für Triton mit mehreren Tausend US-Dollar sehr hoch.

[5] http://www.ogre3d.org/
[6] http://www.ogre3d.org/tikiwiki/Hydrax
[7] http://sundog-soft.com/sds/features/ocean-and-water-rendering-with-triton/

3 Umsetzung und Implementierung

In diesem Kapitel wird die Implementierung der einzelnen Effekte und die Integration in MARS beschrieben. Der Zusammenhang der implementierten Effekte und deren Abhängigkeiten sind in Abbildung 3.1 zu sehen.

In dieser Arbeit wurden die Java-Klassen WaterGridFilter, WaterParticleFilter und SedimentEmitter sowie die Shader WaterGridShader und WaterParticleShader implementiert. Die Java-Klassen sind nur von Klassen aus der jMonkeyEngine abhängig. Die Effekte werden in der Klasse WaterState zu einem Modul vereint.

Die jMonkeyEngine bietet den Advanced Water Filter, der in der Klasse WaterFilter implementiert ist. In dieser Arbeit wurde eine modifizierte Version dieses Filters in der Klasse WaterGridFilter implementiert. Diese Klasse verwendet zusätzlich den von der jMonkeyEngine gestellten ProjectedGrid. Sie bereitet die Effekte vor und gibt die entsprechenden Texturen und Parameter an den WaterGridShader weiter. Dieser berechnet den eigentlichen Effekt.

In der Klasse WaterParticleFilter wurden Partikel implementiert, die im Wasser schweben. WaterParticleFilter erbt von der Klasse Filter, die von der jMonkeyEngine gestellt wird, und übergibt Texturen und Parameter an den WaterParticleShader, der den Effekt erzeugt. WaterParticleShader benutzt zusätzlich eine Bibliothek, die Simplex-Noise implementiert[1].

Die Klasse SedimentEmitter ist für die Sedimentaufwirbelung zuständig. Sie benutzt dafür die jMonkeyEngine-Klasse ParticleEmitter.

3.1 Verbindung von Projected Grid und Advanced Water Filter

Das Projected Grid bietet durch die Wellengeometrie die Möglichkeit, die Höhe des Wassers in der physikalischen Simulation zu verwenden. Dies ist besonders für die Berechnung des Auftriebs wichtig. Der Advanced Water Filter lässt dies nicht zu, hat dafür aber eine Unterwasseransicht. Um die Vorteile beider Effekte ausnutzen zu können, wurden sie vereint.

In einem ersten Ansatz wurden die Höhendaten der Wellen des Projected Grid in Form eines Arrays an den Advanced Water Filter übergeben. Hier stellte sich das Problem,

[1] `https://github.com/ashima/webgl-noise`

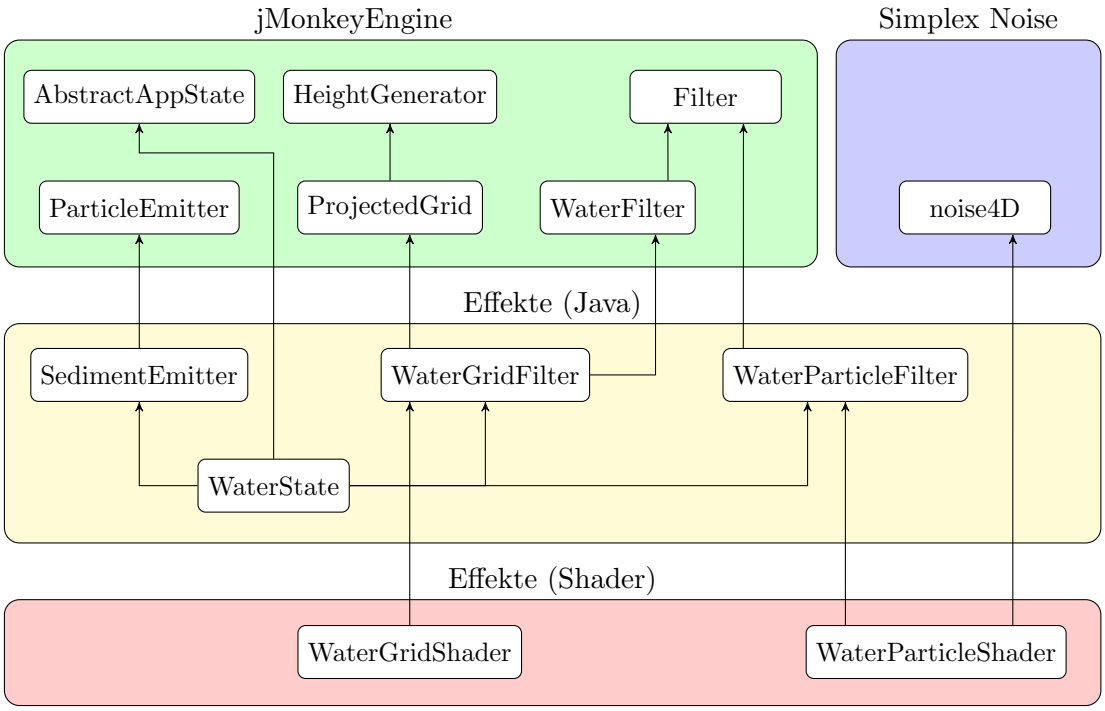


Abbildung 3.1: Zusammenhang und Abhängigkeiten der implementierten Effekte. In dieser Arbeit wurden die Abschnitte „Effekte (Java)“ und „Effekte (Shader)“ implementiert. Die Effekte sind hauptsächlich von Komponenten der jMonkeyEngine abhängig.

die für den aktuellen Pixel relevanten Höhenvektoren zu bestimmen und zu interpolieren. Der vom Projected Grid gestellte Array ist so sortiert, dass eine Suche nach den nächsten Vektoren umständlich ist. Darüber hinaus ist der Array sehr groß, was zu einer erheblichen Verlangsamung des Shaders führt. Da diese Methode sich als zu aufwendig herausstellte, wurde sie verworfen.

Im vereinten Filter, der in der Klasse WaterGridFilter und im Shader WaterGridShader implementiert ist, werden die Höhendaten über eine Textur an den Advanced Water Filter übergeben (siehe Abb. 3.2, 3.3). Dazu wird auf die Geometrie des Projected Grid ein Material mit einem speziellen Shader angewendet. Der Shader codiert die Höhe der Wasseroberfläche in die Farbe der Pixel (siehe Abb. 3.3b). Somit wird dem Advanced Water Filter nur eine Textur in der Größe des Bildschirms übergeben. Der Advanced Water Filter wurde so abgeändert, dass die Höhe des Wassers nicht aus dem aktuellen Pixel und einiger Parameter berechnet, sondern aus der neu gerenderten Textur ausgelesen wird. Danach wird der Advanced Water Filter normal ausgeführt.

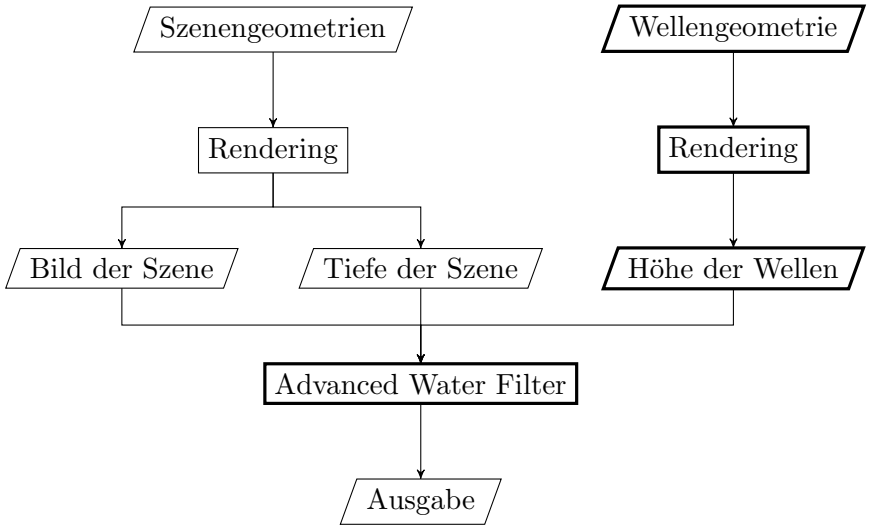


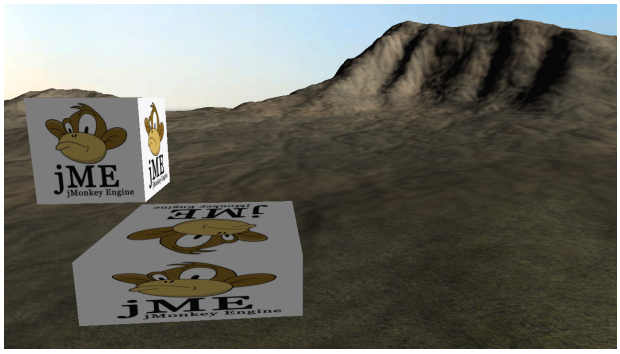
Abbildung 3.2: Der Ablauf des abgeänderten Advanced Water Filter im Detail. Neue oder modifizierte Schritte sind dick umrandet. Hinzugekommen ist das Rendering der Höhe der Wellengeometrie. Diese wird im abgeänderten Advanced Water Filter an Stelle einer festen Höhe zur Berechnung der Wassereffekte verwendet.

3.2 Partikel im Wasser

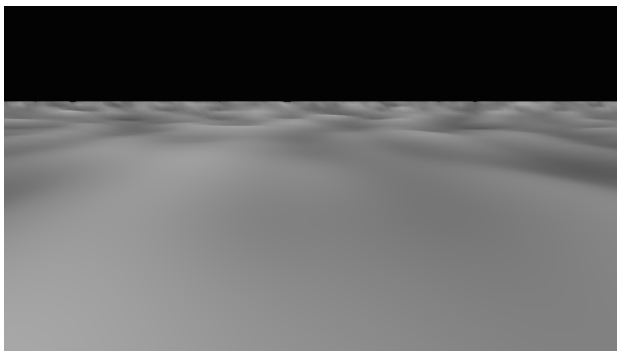
Für im Wasser schwebende Partikel, wie z.B. Pflanzen oder Sediment (siehe Abb. 3.4), wurde ein eigener Filter entwickelt. Der Filter wurde mit Simplex Noise realisiert. Die Idee hierbei ist, dass für jeden Pixel im Wasser die Koordinaten als Parameter für die Noise-Funktion verwendet werden können. Da die Funktion für die selben Parameter immer das gleiche Ergebnis liefert, entsteht so der Eindruck, dass an bestimmten Stellen im Wasser Partikel schweben. Durch die Verwendung der Zeit als vierten Parameter lässt sich eine leichte Variation einbauen, wodurch die Partikel nicht komplett statisch wirken.

Der Partikelfilter wird nur eingeschaltet, wenn sich die Kamera unter Wasser befindet. Dazu werden die x- und z-Koordinaten der Kamera an die Funktion übergeben, die im Projected Grid die Höhe des Wassers berechnet. Der resultierende Wert ist die Höhe der Wasseroberfläche genau über bzw. unter der Kamera. Diese wird mit der Position der Kamera verglichen und basierend darauf der Filter ein- bzw. ausgeschaltet.

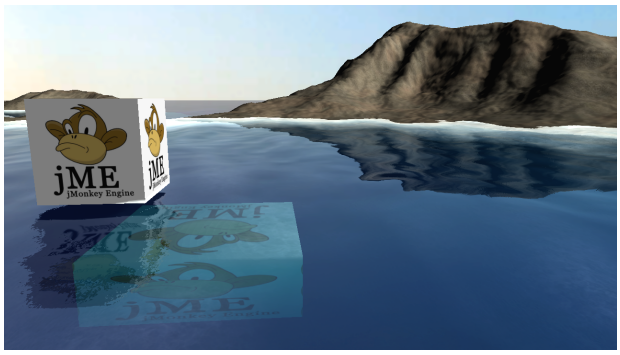
Um die exakte Position eines Pixels unter Wasser zu bestimmen, werden die Bildschirmkoordinaten des Pixels mit Hilfe der inversen Transformationsmatrix der Kamera transformiert. Danach liegen die Koordinaten des Pixels im Weltkoordinatensystem vor. Die Koordinaten sowie die aktuelle Zeit werden an die Simplex Noise-Funktion übergeben. Um mehr Partikel zu erhalten, wird das Rauschen über mehrere Iterationen oder „Oktaven“ berechnet [9]. Mit jeder Iteration werden die Parameter der Noise-Funktion verdoppelt und die erhaltenen Werte summiert. Damit das Rauschen eher punktförmig wird



(a) Die Szene, wie sie der Advanced Water Filter erhält. Außer dieser Textur erhält der Filter nur die Entfernung jedes Pixels von der Kamera.



(b) Die Wasseroberfläche des Projected Grid mit kodierter Höheninformation. Schwarz stellt einen tiefen Punkt dar, weiß einen hohen Punkt.



(c) Das Ergebnis des kombinierten Filters ist eine mit dem Advanced Water Filter gerenderte Wasseroberfläche mit Höheninformationen aus dem Projected Grid.

Abbildung 3.3: Der Ablauf des abgeänderten Advanced Water Filters anhand der verwendeten Texturen.

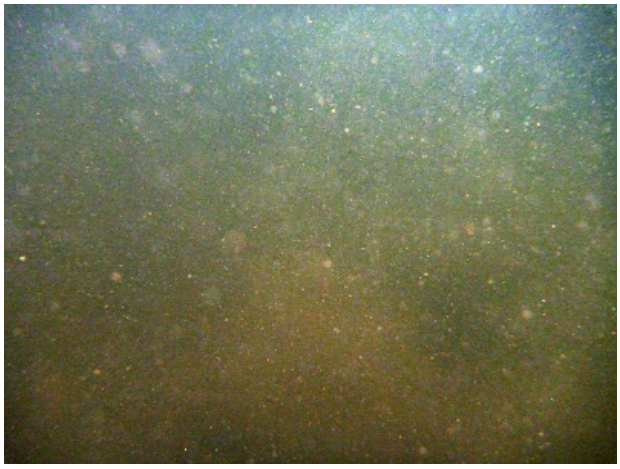


Abbildung 3.4: Eine echte Aufnahme von im Wasser schwebenden Partikeln zum Vergleich.[2]

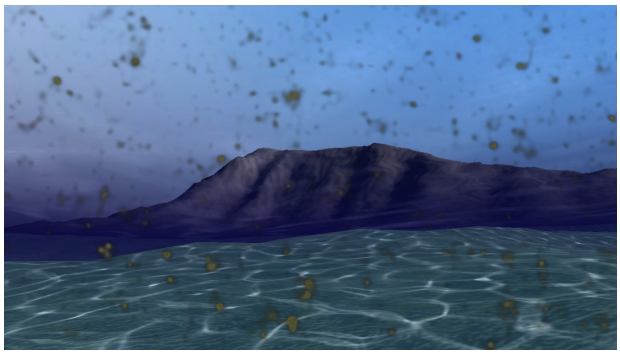


Abbildung 3.5: Das durch den Partikelfilter erzeugte Bild. Mit Simplex Noise generierte Partikel werden der Szene überlagert. Die Partikel stellen z.B. Pflanzen und Sediment dar.

und somit mehr wie eine Menge an Partikeln wirkt, wird der berechnete Wert mit einem festen Parameter potenziert. Werte gegen Null werden somit kleiner, während ein Wert von Eins bestehen bleibt. Das Rauschen nähert sich dadurch punktförmigen Bereichen an. Dieser Wert wird mit der Farbe der Partikel multipliziert, um den Rauschen eine sedimentähnliche Farbe zu geben. Um große, einfarbige Flächen zu vermeiden, wird die Farbe noch mit einem Farbwert aus einer Textur multipliziert. Dadurch erhalten die Partikel eine farbliche Variation. Zuletzt wird das bisherige Ergebnis mit der Farbe des aktuellen Pixels vermischt, sodass die Partikel auf die Szene überlagert werden (siehe Abb. 3.5).

[2] http://whitemountainsojourn.blogspot.com/2010/07/rainy-day-at-carter-notch-7-19-10-in.html

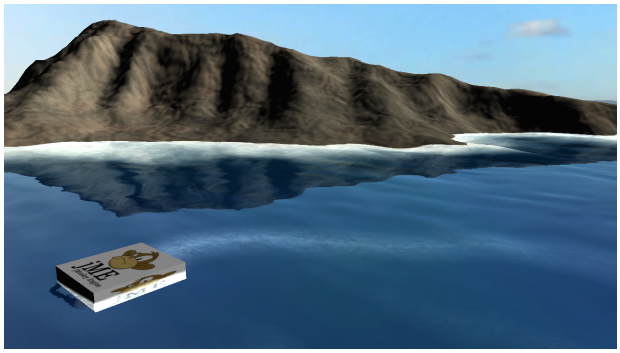


Abbildung 3.6: Fahrzeuge ziehen eine Schaumspur hinter sich her.

Abbildung 3.7: Zum Vergleich ein Bild einer echten Schaumspur.[3]

3.3 Schaumspuren von Fahrzeugen

Fahrzeuge, die an der Wasseroberfläche fahren, sollen eine Schaumspur hinter sich herziehen, die durch die Bewegung durch das Wasser entsteht (siehe Abb. 3.6, 3.7). Da der Advanced Water Filter bereits Schaum am Rand der Wasseroberfläche erzeugt, wurden die Schaumspuren dort integriert.

Um Schaumspuren zu erzeugen, muss ein Fahrzeug beim Advanced Water Filter registriert werden. Der Filter merkt sich fortlaufend die Position des Fahrzeugs und erzeugt daraus und aus der Höhe des Wassers eine Geometrie, die hinter dem Fahrzeug her schwimmt. Zur Konstruktion der Geometrie werden für jede gemerkte Position zwei Punkte eingefügt (siehe Abb. 3.8). Die Punkte werden mit Abstand parallel zur Fahrtrichtung gesetzt und erhalten als Höhe die aktuelle Höhe des Wassers. Zu Beginn der Schaumspur entspricht der Abstand der Breite des Fahrzeugs, die über seine Bounding Box ermittelt wird. Eine Bounding Box ist ein Quader, der ein Objekt komplett beinhaltet und somit die ungefähren Maße des Objektes besitzt. Der Abstand der Punkte wird zum Ende der Schaumspur größer, wodurch die Schaumspur breiter wird.

Die Geometrie wird in eine Textur mit einem Shader gerendert, der in der Mitte der

[3] http://makezine.com/2010/08/19/zipper-boat/

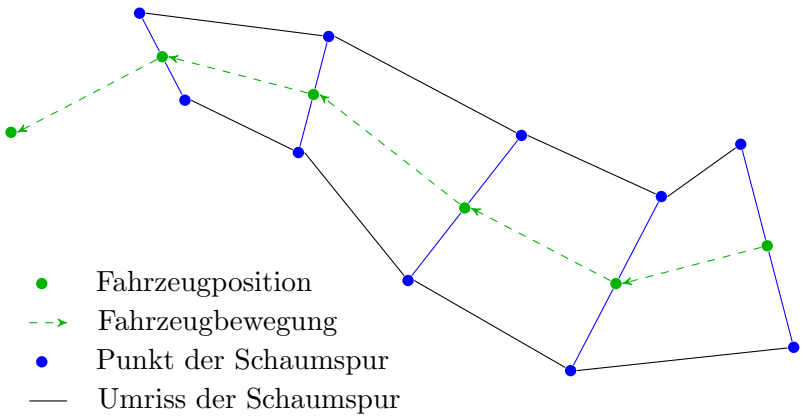
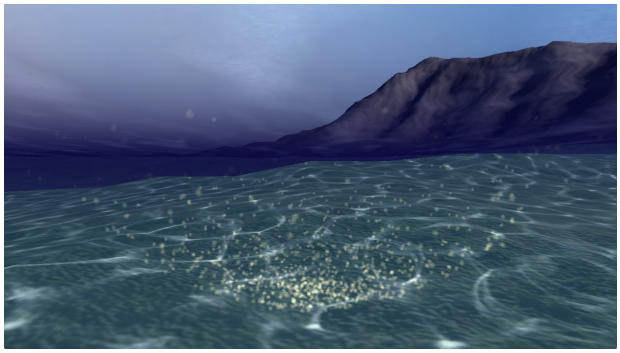


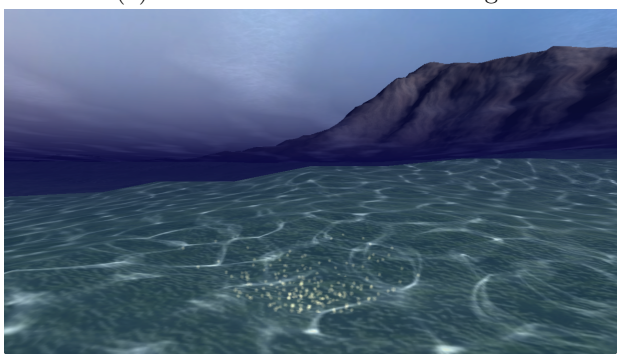
Abbildung 3.8: Die Konstruktion der Schaumspur. Für jeden Punkt, den das Fahrzeug passiert hat, werden parallel zur Fahrtrichtung zwei Punkte erzeugt. Die Punkte werden zu der Schaumspur verbunden. Nach hinten wird der Abstand der Punkte größer, sodass die Schaumspur breiter wird.

Abbildung 3.9: Die aus den gespeicherten Positionsdaten des Fahrzeugs erzeugte Schaumgeometrie. Anhand dieser wird eine Schaumtextur auf das Bild überlagert, es entsteht eine Schaumspur hinter dem Fahrzeug (siehe Abb. 3.6).

Schaumspur einen Wert von 1 und am Rand einen Wert von 0 ausgibt (siehe Abb. 3.9). In den Bereichen dazwischen werden die Werte interpoliert. Die Textur wird anschließend an den eigentlichen Advanced Water Filter übergeben. Dieser berechnet bereits für jeden Pixel anhand der Tiefe des Wassers einen Wert für den Schaum und mischt mit diesem Wert eine Schaumtextur in die Pixelfarbe. Steht in der übergebenen Textur ein Wert größer als 0, wird dieser mit dem bisher berechneten Schaumwert verrechnet und somit in die Schaumgenerierung des Filters integriert.



(a) Starke Sedimentaufwirbelung.



(b) Schwache Sedimentaufwirbelung.

Abbildung 3.10: Die simulierte Sedimentaufwirbelung. Das Sediment wird durch einen Stoß oder durch den Motor eines Fahrzeugs seitlich aufgewirbelt und bewegt sich nach oben.

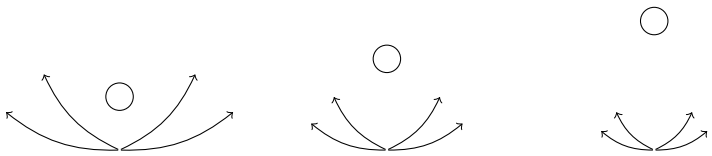


Abbildung 3.11: Die Stärke der Sedimentaufwirbelung ist von der Distanz der Quelle zum Boden abhängig. Wird die Entfernung größer, wird die Aufwirbelung schwächer.

3.4 Sedimentaufwirbelung

Die Aufwirbelung von Sediment wurde mit Hilfe des Partikelsystems der jMonkeyEngine realisiert (siehe Abb. 3.10). Das Sediment wird durch erdfarbene Partikel dargestellt, die kontinuierlich generiert werden. Sie werden durch einen *RadialParticleInfluencer* von einem Punkt ausgehend kreisförmig ausgestoßen. Damit es so erscheint, dass das Sediment auf den Boden aufprallt und wieder nach oben schwimmt, erhalten die Partikel eine negative Gravitation. Entfernt sich die Quelle der Aufwirbelung vom Boden, wird

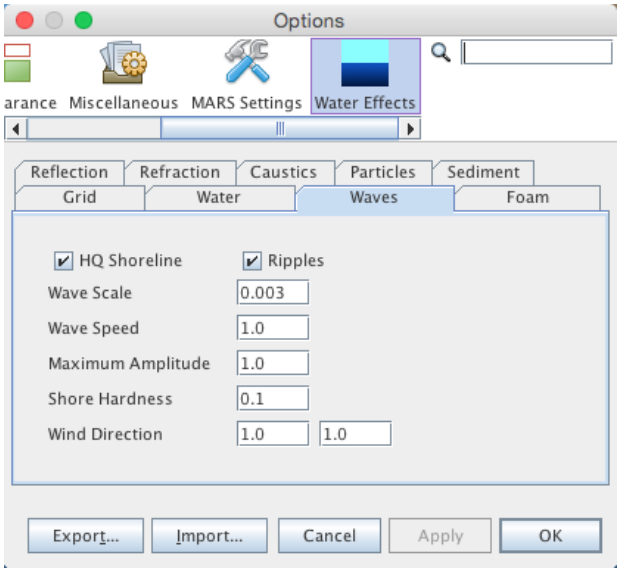


Abbildung 3.12: Das für die Effekte erstellte OptionsPanel. Für jeden Effekt gibt es ein Tab, in dem die Einstellungen des Effekts geändert werden können.

der Effekt schwächer (siehe Abb. 3.11).

Da die Partikel keine Objekte im Sinne der jMonkeyEngine darstellen, schreiben sie keinen Wert in den Depth-Buffer. Der Advanced Water Filter erkennt jedoch anhand dieses Buffers, ob sich in der Szene ein Objekt befindet, das beachtet werden muss. Daher muss für die Partikel ein spezieller „Blend Mode“ eingestellt werden, da sie sonst vom Advanced Water Filter überschrieben werden.

3.5 Integration in MARS

Zur Integration in MARS wurden die Effekte in ein eigenständiges AppState namens WaterState integriert. Ein AppState stellt in der jMonkeyEngine eine Komponente eines Programms dar, die eigenständig läuft und sich zur Laufzeit ein- und ausschalten lässt. MARS verfügt über einige AppStates, die bestimmte Aspekte der Simulation bearbeiten.

Das WaterState initialisiert alle Effekte und fügt sie der Szene hinzu. Es verfügt über einige Methoden, mit denen sich die Effekte konfigurieren lassen.

Um die einfache Konfiguration der Effekte zu ermöglichen, wurde ein OptionsPanel entwickelt und in MARS integriert (siehe Abb. 3.12). Ein OptionsPanel ist eine Komponente der NetBeans Platform, die es erlaubt, eigene Einstellungsfenster für ein Programm zu erstellen.

Das OptionsPanel enthält Tabs für verschiedene Aspekte der Wasseroberfläche und der

Unterwasseransicht, sowie eigene Tabs für den Partikelfilter und die Sedimentaufwirbelung. Es lassen sich Parameter und Texturen für die Wasseroberfläche, Wellen, Schaum, Reflektionen, Lichtbrechung und Kaustik-Effekte einstellen. Optionen für die Schaumspuren wurden in das „Foam“-Tab integriert.

4 Evaluation

In diesem Kapitel wird die Evaluation der Leistung der Wassereffekte auf unterschiedlichen Systemen beschrieben. Zur Evaluation wurde ein Testszenario entwickelt, mit dem die Effekte einzeln und als Ganzes getestet und bewertet wurden.

4.1 Testaufbau

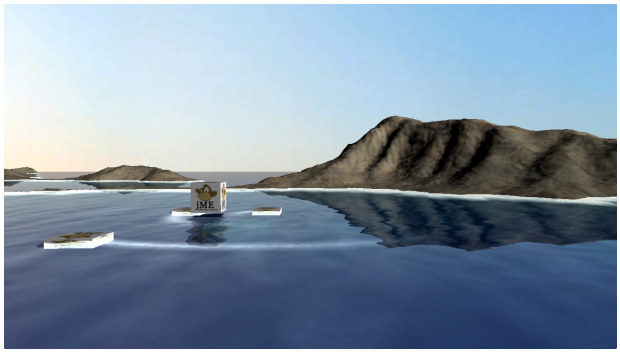
Die Wassereffekte wurden auf mehreren Systemen getestet (siehe Tabelle 4.1). Zur Bewertung der Leistung wurden dabei die FPS, d.h. wie oft pro Sekunde die Szene gerendert werden konnte, herangezogen. Diese wurden beim Durchlauf der Szenarien gemessen und die Minimal-, Maximal- und Durchschnittswerte ermittelt.

System	Betriebssystem	RAM	Prozessor	Grafikkarte
1	Mac OS X 10.10	8 GB	Intel Core i5-520M 2x2,4 GHz	Intel HD Graphics
2	Windows 8.1	8 GB	Intel Core i5-520M 2x2,4 GHz	NVIDIA GeForce GT 330M
3	Windows 7	8 GB	Intel Core i7-2620M 4x2,7 GHz	NVIDIA GeForce NVS 4200M
4	Windows 7	8 GB	Intel Core i7-3770 4x3,4 GHz	NVIDIA GeForce GTX 760
5	Windows 8.1	8 GB	Intel Core i7-Q740 4x1,7 GHz	NVIDIA GeForce GTX 460M

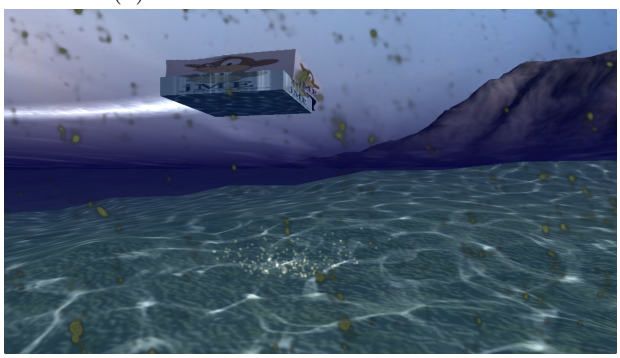
Tabelle 4.1: Liste der Systeme, auf denen die Wassereffekte evaluiert wurden.

4.2 Testszenario

Das Testszenario besteht aus einem zufällig generierten Terrain mit fünf Fahrzeugen, die an der Wasseroberfläche fahren, und fünf Sedimentaufwirblern (siehe Abb 4.1, 4.2). Um alle Effekte abzudecken, wurde eine Kamerafahrt eingestellt, bei der die Kamera zu gleichen Teilen unter und über Wasser ist. Die Effekte wurden einzeln und zusammen getestet. Zusätzlich wurde der ursprüngliche Advanced Water Filter aus der jMonkeyEngine getestet. Die Wasseroberfläche und die Schaumspuren wurden zusammen getestet,



(a) Das Testszenario über Wasser.



(b) Das Testszenario unter Wasser.

Abbildung 4.1: Das Testszenario über und unter Wasser.

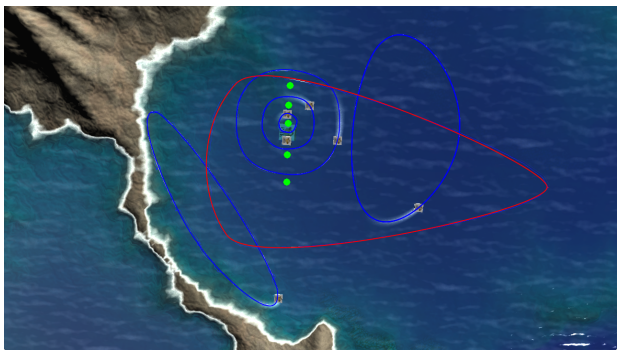


Abbildung 4.2: Eine Übersicht des Testszenarios. Zu sehen sind die Bewegungen der Fahrzeuge in Blau, die Bewegung der Kamera in Rot und die Standorte der Sedimentaufwirbler in Grün.

da die Schaumspuren in das Rendering der Wasseroberfläche integriert wurden. Die Tests wurden jeweils 60 Sekunden lang ausgeführt. Dabei wurde die minimale, maximale und durchschnittliche Anzahl an Bildern pro Sekunde gemessen. Eine Auflistung der Tests und eine kurze Beschreibung ist in Tabelle 4.2 zu sehen.

Testname	Beschreibung
jME Water	Der ursprüngliche Advanced Water Filter aus der jMonkeyEngine.
Water Filter	Der modifizierte Advanced Water Filter mit Projected Grid.
Water Filter mit Schaum	Der modifizierte Advanced Water Filter mit Projected Grid und Schaumspuren.
Partikelfilter	Filter, der im Wasser schwebende Partikel erzeugt.
Sedimentaufwirbler	Der Sedimentaufwirbelungs-Effekt.
Alle	Alle Effekte ohne den ursprünglichen Advanced Water Filter.

Tabelle 4.2: Die durchgeführten Tests mit einer kurzen Beschreibung.

4.3 Testergebnisse

Test	min. FPS	durchschn. FPS	max. FPS
jME Water	1.5	14.1	23.3
Water Filter	1.9	14.5	22.7
Water Filter mit Schaum	5.1	13.6	21.5
Partikelfilter	3.4	8.3	9.8
Sedimentaufwirbler	3.4	24.5	32.1
Alle	3.8	8.6	18.3

Tabelle 4.3: Die Ergebnisse der Evaluation auf dem ersten System.

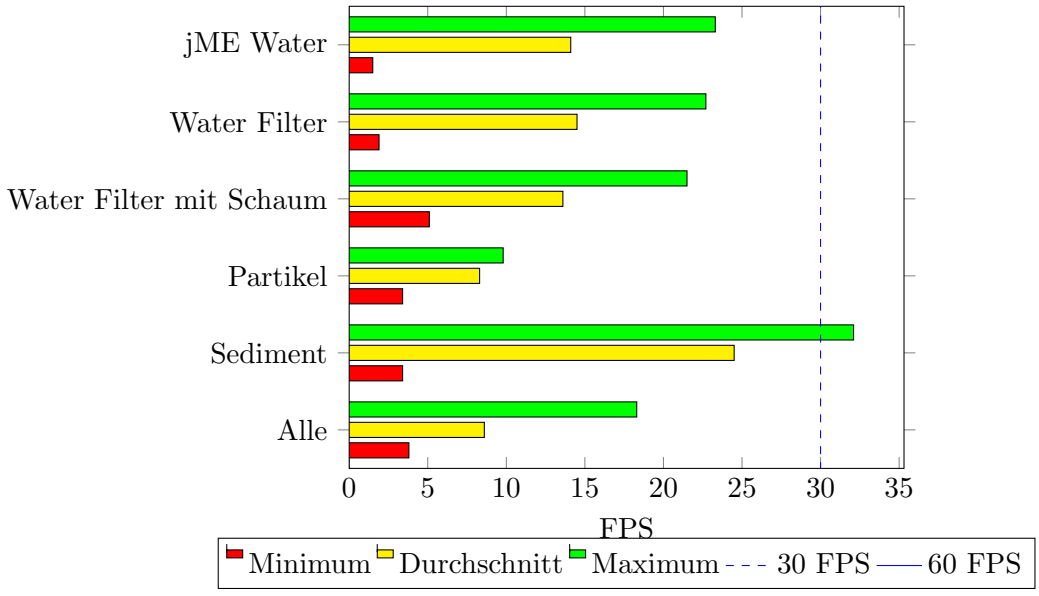


Abbildung 4.3: Die Ergebnisse der Evaluation auf dem ersten System.

Das erste System hat eine schwache Grafikkarte, wodurch die gemessenen Bildraten niedrig sind (siehe Tabelle 4.3, Abb. 4.3). Besonders zu Beginn der Tests wurde eine sehr niedrige Bildrate gemessen. Nur im Test der Sedimentaufwirbler erreicht das System eine flüssige durchschnittliche Bildrate mit ca. 24 FPS. Dies liegt vermutlich daran, dass die Sedimentaufwirbelung aufgrund der hohen Zahl zu verwaltender Partikel mehr von der CPU als von der Grafikkarte abhängig ist.

Test	min. FPS	durchschn. FPS	max. FPS
jME Water	3.3	61.6	464.2
Water Filter	26.2	66.4	472.3
Water Filter mit Schaum	18.4	60.8	311.9
Partikelfilter	11.1	26.5	419.7
Sedimentaufwirbler	4.0	77.0	122.5
Alle	12.1	31.3	88.6

Tabelle 4.4: Die Ergebnisse der Evaluation auf dem zweiten System.

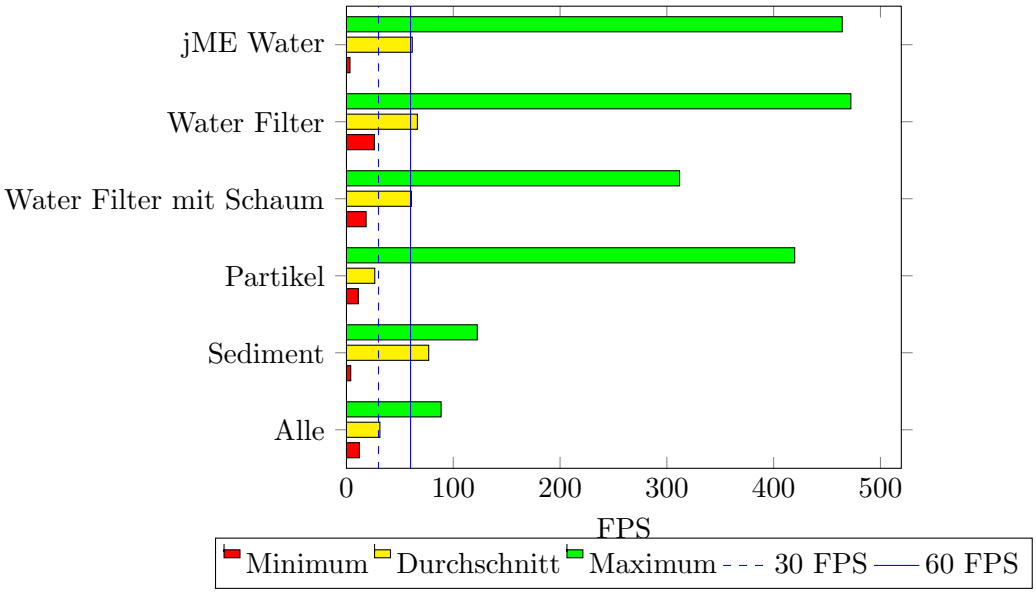


Abbildung 4.4: Die Ergebnisse der Evaluation auf dem zweiten System.

Das zweite System besitzt eine deutlich leistungsfähigere Grafikkarte und erzielt in allen Tests flüssige durchschnittliche Bildraten (siehe Tabelle 4.4, Abb. 4.4). In allen Tests ohne den Partikelfilter war die durchschnittliche Bildrate über 60 FPS. Mit dem Partikelfilter konnte eine durchschnittliche Bildrate von ca. 26 FPS erzielt werden.

Test	min. FPS	durchschn. FPS	max. FPS
jME Water	2.5	48.2	518.3
Water Filter	34.5	48.8	73.2
Water Filter mit Schaum	26.0	45.3	72.6
Partikelfilter	25.3	29.8	36.4
Sedimentaufwirbler	6.5	82.6	177.4
Alle	19.8	29.9	80.0

Tabelle 4.5: Die Ergebnisse der Evaluation auf dem dritten System.

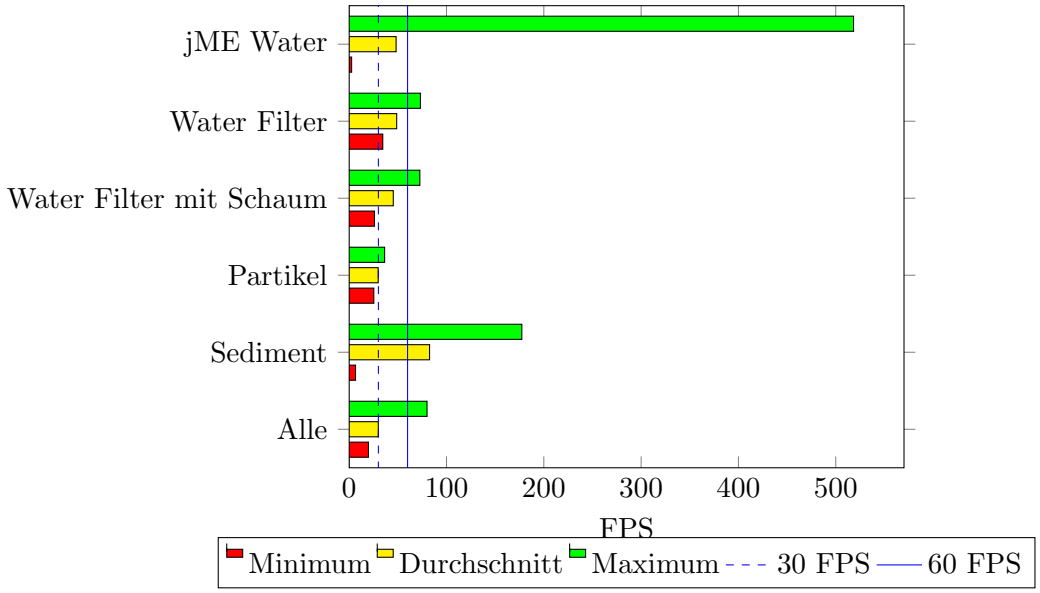


Abbildung 4.5: Die Ergebnisse der Evaluation auf dem dritten System.

Das dritte System erzielt ähnliche Ergebnisse wie das zweite System (siehe Tabelle 4.5, Abb. 4.5). Auch hier erzielt der Partikelfilter die geringste und die Sedimentaufwirbelung die höchste durchschnittliche Bildrate.

Test	min. FPS	durchschn. FPS	max. FPS
jME Water	11.7	888.6	1534.4
Water Filter	24.2	870.4	1410.5
Water Filter mit Schaum	18.4	682.7	916.7
Partikelfilter	29.8	399.4	1469.2
Sedimentaufwirbler	16.5	255.2	458.6
Alle	57.2	196.7	261.5

Tabelle 4.6: Die Ergebnisse der Evaluation auf dem vierten System.

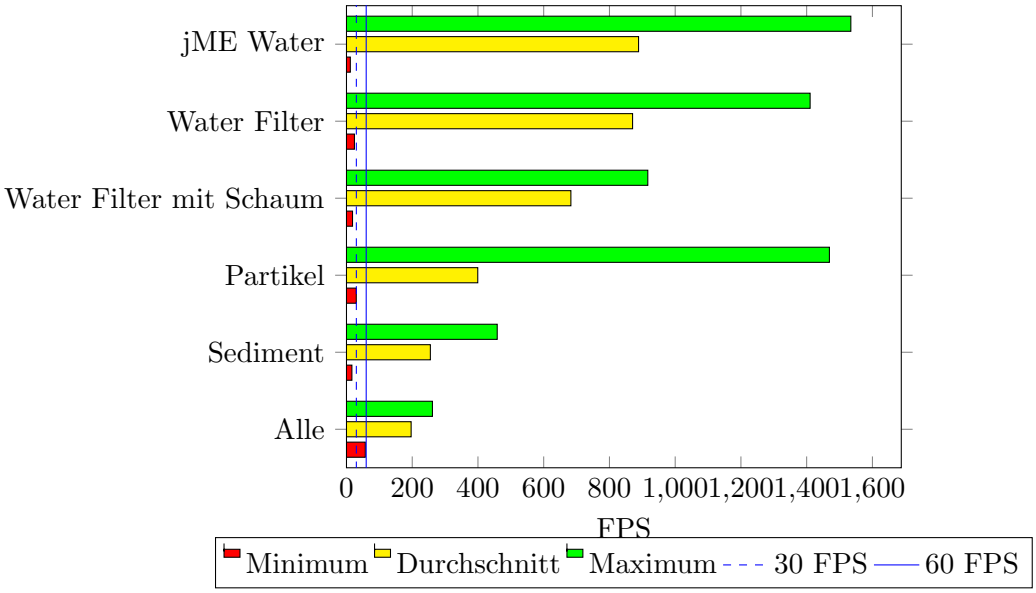


Abbildung 4.6: Die Ergebnisse der Evaluation auf dem vierten System.

Das vierte System hat eine sehr leistungsfähige Grafikkarte und erzielt in allen Tests im Durchschnitt sehr hohe Bildraten (siehe Tabelle 4.6, Abb. 4.6). Die geringste durchschnittliche Bildrate wurde im Gegensatz zu den anderen Systemen beim Test aller Effekte erzielt. Mit nahezu 200 FPS laufen die Effekte trotzdem sehr flüssig.

Test	min. FPS	durchschn. FPS	max. FPS
jME Water	45.7	193.1	476.0
Water Filter	50.1	182.6	429.2
Water Filter mit Schaum	28.4	167.4	302.6
Partikelfilter	100.1	112.5	140.9
Sedimentaufwirbler	7.4	86.9	153.4
Alle	31.7	70.8	99.3

Tabelle 4.7: Die Ergebnisse der Evaluation auf dem fünften System.

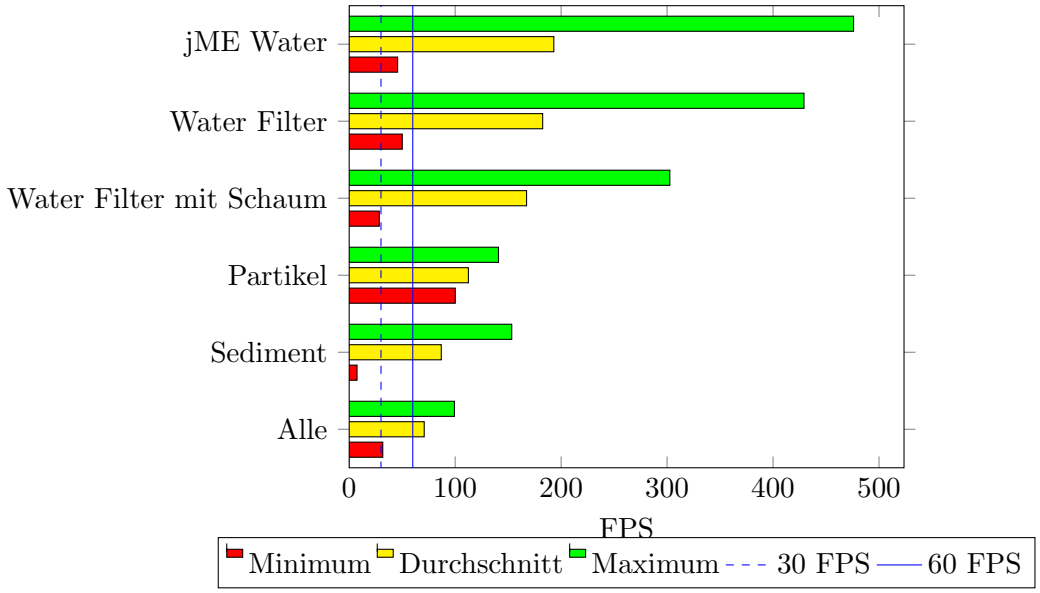


Abbildung 4.7: Die Ergebnisse der Evaluation auf dem fünften System.

Das fünfte System erzielt auch sehr gute Ergebnisse (siehe Tabelle 4.7, Abb. 4.7). Die geringste durchschnittliche Bildrate ist mit 70 FPS immer noch hoch. Wie beim vorigen System wurde diese beim gemeinsamen Test aller Effekte erreicht.

Insgesamt ist die Leistung der Effekte gut, sofern keine sehr schwache Grafikkarte verwendet wird. Die Modifikationen des Advanced Water Filters haben keine messbare Auswirkung auf die Leistung des Filters. Das Hinzufügen der Schaumspuren führt nur zu einer kleinen Verringerung der Bildrate. Der Partikelfilter benötigt die meiste Rechenleistung und zieht die durchschnittliche Bildrate stark nach unten. Die Sedimentaufwirbelung erzielt die beste Bildrate, da sie hauptsächlich von der Leistung der CPU abhängig ist.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurden verschiedene Wassereffekte entwickelt und implementiert. Die Effekte wurden als Plugin in die Simulationsumgebung MARS integriert. Über ein Optionsmenü können Effekte ein- und ausgeschaltet und Parameter sowie Texturen eingestellt werden.

Die Wasseroberfläche zeigt Wellen, Reflektionen, Lichtbrechung und Schaum. Die Unterwasseransicht bietet eine realistische Absorption des Lichts und Kaustik-Effekte. Dafür wurden von der jMonkeyEngine bereitgestellte Effekte verwendet. Die Höhe des Wassers kann in der Simulationsumgebung für die Auftriebsberechnung verwendet werden.

Fahrzeuge, die entlang der Wasseroberfläche fahren, ziehen eine Schaumspur hinter sich her. Die Schaumspur hat zu Beginn die Breite des Fahrzeugs und wird zum Ende hin breiter. Sie passt sich an die Höhe der Wellen an, sodass sie auf der Wasseroberfläche schwimmt.

Unter Wasser wird die Sicht durch Partikel wie Pflanzen oder Sediment vermindert. Die Partikel haben eine feste Stelle im Raum, es kann aber eine zeitliche Variation eingestellt werden. Die Farbe, die Dichte und die Textur der Partikel können angepasst werden.

Kommt ein Fahrzeug in die Nähe des Bodens, wird Sediment aufgewirbelt. Die Sedimentpartikel breiten sich kreisförmig vom Ursprung aus und bewegen sich nach oben. Der Benutzer kann die Anzahl, Farbe und Bewegungsgeschwindigkeit der Partikel einstellen.

Die Leistung der Effekte wurde auf verschiedenen Systemen mit einem Testszenario sowohl einzeln als auch als Ganzes getestet. Dabei wurde die erreichte Bildrate bei der Ausführung der Effekte gemessen. Die Leistung der Effekte ist gut, jedoch erzielt der Partikelfilter mit großem Abstand die geringste Bildrate der Effekte, weshalb er noch optimiert werden sollte. Da der Filter für jeden Pixel mehrere Iterationen mit rechenintensiven Operationen durchführt, sollte die Optimierung darauf fokussiert werden. Alternativ könnte der Effekt mit vorgegebenen Rauschtexturen und Blending nachgeahmt werden.

Die aktuellen Implementationen der Effekte beinhalten in Grenzfällen Darstellungsfehler. Durch die externe Berechnung der Wellen entstehen bei den Reflektionen Artefakte und die Schaumspuren werden nicht richtig dargestellt, wenn sie einander überlappen. Eine Verbesserung der Unterwasseransicht wäre, für jeden Pixel zu bestimmen, ob er sich unter oder über der Wasserlinie befindet. Bisher wird nur festgestellt, ob die Kamera sich unter Wasser befindet und für das gesamte Bild die Berechnung der Unterwassersicht

durchgeführt. In Verbindung mit den Schaumspuren könnten Fahrzeuge während der Fahrt Wassertropfen versprühen. Um die Simulation realistischer zu machen, könnten physikalische Simulationen in die Effekte integriert werden, wie z.B. Strömungs- oder Windsimulationen.

Abbildungsverzeichnis

2.1 Projected Grid 4
2.2 Advanced Water Filter, über Wasser 4
2.3 Advanced Water Filter, Unterwasser 5
2.4 Die Simulationsumgebung MARS 5
2.5 Die vereinfachte Grafikpipeline 6
2.6 Raster aus Simplices mit Gradienten 8
2.7 Bestimmung des zugehörigen Simplex 8
2.8 osgOcean 9
2.9 OGRE/Hydrax 9
2.10 Triton 10

3.1 Zusammenhang und Abhängigkeiten der implementierten Effekte 12
3.2 Der Ablauf des abgeänderten Advanced Water Filters im Detail 13
3.3 Texturen des abgeänderten Advanced Water Filters 14
3.4 Vergleichsbild für Partikel im Wasser 15
3.5 Ergebnis des Partikelfilters 15
3.6 Schaumspuren 16
3.7 Vergleichsbild für Schaumspuren 16
3.8 Konstruktion der Schaumspur 17
3.9 Resultierende Schaumgeometrie 17
3.10 Sedimentaufwirbelung 18
3.11 Abhängigkeit der Sedimentaufwirbelung von der Distanz 18
3.12 Das für die Effekte erstellte OptionsPanel 19

4.1 Das Testszenario 22
4.2 Übersicht über das Testszenario 22
4.3 Testergebnisse auf System 1 23
4.4 Testergebnisse auf System 2 24
4.5 Testergebnisse auf System 3 25
4.6 Testergebnisse auf System 4 26
4.7 Testergebnisse auf System 5 27

Tabellenverzeichnis

4.1 Liste der Evaluationssysteme 21
4.2 Durchgeführte Tests und Beschreibung 23
4.3 Testergebnisse auf System 1 23
4.4 Testergebnisse auf System 2 24
4.5 Testergebnisse auf System 3 25
4.6 Testergebnisse auf System 4 26
4.7 Testergebnisse auf System 5 27

Abkürzungsverzeichnis

FPS ..... Frames per Second

GPU ..... Graphics Processing Unit

MARS ..... MArine Robotics Simulator

ROS ..... Robot Operating System

SDK ..... Software Development Kit

Literaturverzeichnis

[1] R. Wagner, "Wasserqualität und Gewässerqualität." ftp://ftp.gwdg.de/pub/mpil-schlitz/Wagner/limnologie/Wasserqualit%E4t%20und%20Gew%E4sserqualit%E4t.pdf. Zuletzt aufgerufen am 05.09.2014.

[2] R. Pott and D. Remy, *Gewässer des Binnenlandes*. Ulmer Stuttgart, 2000.

[3] "jMonkeyEngine." http://jmonkeyengine.org/. Zuletzt aufgerufen am 08.07.2014.

[4] T. Tosik and E. Maehle, "MARS: A Simulation Environment For Marine Robotics," *OCEANS 14 MTS/IEEE St. Johns*, 2014.

[5] E. Angel and D. Shreiner, "An Introduction to OpenGL Programming," in *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, (New York, NY, USA), pp. 3:1–3:110, ACM, 2013.

[6] S. Gustavson, "Simplex noise demystified." http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf, 2005. Zuletzt aufgerufen am 08.07.2014.

[7] "osgOcean." https://code.google.com/p/osgocean/. Zuletzt aufgerufen am 08.07.2014.

[8] J. Tessendorf, "Simulating Ocean Water," *Simulating Nature: Realistic and Interactive Techniques. SIGGRAPH*, vol. 1, 2001.

[9] J. R. Frisvad and G. Wyvill, "Fast High-Quality Noise," in *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, GRAPHITE '07, (New York, NY, USA), pp. 243–248, ACM, 2007.