UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Simulation und Visualisierung von Wasseroberflächen

Bachelorarbeit

zur Erlangung des Grades einer Bachelor of Science (B.Sc.)
im Studiengang Informatik

vorgelegt von
Vera Christ

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergrafik)
Zweitgutachter:	Anna Katharina Hebborn, M.Sc. (Institut für Computervisualistik, AG Computergrafik)

Koblenz, im Juli 2014

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Abstract

Tiny waves driven by wind, shallow, long waves, head overlapping sea, all of these waves occur in every ocean and even in small lakes. The surface of water ist one of the most versatile phenomenas of nature. Not only the movement of waves, but also the reflection of sky, sun and coastline makes the surface of water unique. Exactly this complexity is what brings its own challenges to the simulation of water surfaces. That is why simulation of water occupies mathematicians with a challenge for nearly 400 years now.

In the last fifty years this challenge has more and more shifted to computer science. Computer graphic designers have tried to visualise water in a realistic manner for centuries. Science in this field expends from simple noise filters to mathematically complex solutions like Fourier Transformation.

In the following work historical background of todays wave theories, as well as mathematical fundamentals are given. The focus of this work is set on the implementation of these methods in OpenGL 3.3.

Zusammenfassung

Winzige vom Wind getriebene Wellen, flache, lange Wellen, steile sich überlagernde Wellen oder stürmische Brecher, sie alle kommen in jedem der Weltmeere und sogar in kleinen Seen vor. Die Wasseroberfläche ist eines der vielfältigsten Naturphänomene. Nicht nur die Bewegung der Wellen, auch die Spiegelung von Himmel, Sonne und Küste machen die Meeresoberfläche einzigartig. Gerade diese einzigartige Komplexität stellt ihre ganz eigenen Herausforderungen an die Nachahmung solcher Phänomene. Deshalb stellt die Berechnung von Wellen schon seit gut 400 Jahren Mathematiker vor eine große Aufgabe.

In den letzten fünfzig Jahren hat sich diese Herausforderung immer mehr in den Bereich der Informatik verschoben. Computergrafiker versuchen seit Jahrzehnten Wasser realistisch darzustellen. Die Forschung auf diesem Gebiet reicht mittlerweile von einfachen Ansätzen wie Rauschfiltern bis zu mathematisch hoch komplexen Ansätzen, wie der Fourier Transformation.

In der nachfolgenden Arbeit wird sowohl auf die geschichtliche Entwicklung der heutigen Wellentheorien, als auch auf die mathematischen Grundlagen dieser eingegangen. Schwerpunkt der Arbeit ist es, diese Methoden in OpenGL 3.3 zu implementieren.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Mit dem Wachstum der Computerspieleindustrie hat auch die Notwendigkeit, die Natur um uns herum realistisch darstellen zu können, stark zugenommen. Von aufwändigen 3D Spielen, in denen dem Spieler ein möglichst realistisches Umfeld geschaffen werden soll bis hin zu Simulationen für biologische oder geographische Anwendungen, z.B. in der Klimaforschung, reicht das Feld der computergrafischen Simulationen von Naturphänomenen.

Diese Arbeit beschäftigt sich mit der Darstellung von Wasseroberflächen. Im speziellen geht es darum große Wasserflächen, wie beispielsweise einen Ozean oder einen großen See, darzustellen. Dabei müssen sowohl komplizierte Wellenbewegungen, die von Wind oder Strömung verursacht werden, als auch Reflexionen der Sonne, des Himmel und anderer Objekte betrachtet werden.

Naturgemäß sind komplexe Phänomene, wie beispielsweise die Bewegung von Wasserteilchen unter Einfluss von zahlreichen Faktoren, wie Gravitation und Interaktion mit andern Medien, beispielsweise Luft, für die Computergrafik schwierig zu beschreiben. Überlagerung von Wellen mit Wellenlängen und -amplituden, die zwischen mehreren Kilometern und nur wenigen Millimetern schwanken können, erschweren die Berechnung.

Zwischen der Simulation von Wasseroberflächen und der Simulation von Wasser im Allgemeinen gibt es einen entscheidenden Unterschied: die Bewegung an Wasseroberflächen ist in hohem Maße von der Oberflächenspannung und der Gravitation abhängig.

In der Regel werden zur physikalischen Wassersimulation die Navier-Stokes-Gleichungen verwendet. Diese bieten eine sehr gute Beschreibung der natürlichen Bewegung von Wasser, unter dem Einfluss der Schwerkraft. Der Nachteil an dieser Methode ist, dass sie für kleine Mengen von Partikeln zwar sehr gut geeignet ist, sobald man aber eine große Fläche, wie zum Beispiel ein Meer simulieren möchte, wird der Rechenaufwand zu hoch. Dementsprechend müssen hier andere Lösungen gefunden werden.

Es gibt zahllose Möglichkeiten Wellen realistisch darzustellen. Dabei schwanken diese zwischen dem reinen realistischen Aussehen und der physikalischen Korrektheit. Mit verrauschten Filtern können, zumindest unter Annahme von ruhigen Wetterbedingungen, gute Ergebnisse erzielt werden. Wenn man physikalisch korrekt arbeiten muss, beispielsweise um Hochwasser oder Sturmfluten zu simulieren, liegt es nahe einen Ansatz zu wählen der Parameter, die in der Realität gemessen werden, berücksichtigen kann. Hier hat sich vor allem die Fast Fourier Transformation etabliert.

1.2 Verwandte Arbeiten und geschichtliche Herangehensweise

In diesem Unterkapitel wird sich vor allem auf das Buch *The Science of Ocean Waves* [1] und den Artikel *The origins of water wave theory* [2] bezogen.

Isaac Newton war der Erste, der sich mit der physikalischen Bewegung von Wasser beschäftigte. Es gelang ihm 1687 in seinem Buch *Philosophiae Naturalis Prinzipia Mathematica* eine Formel zur Frequenz von Wasserwellen zu veröffentlichen. Allerdings galt diese, wie er selbst sagte, nur unter der Annahme, dass die Bewegung eines Wasserteilchens ausschließlich nach oben und unten (also linear) verläuft. Er selbst nahm bereits in der dritten Auflage seines Buches an, dass diese Vorraussetzung falsch sei, da sich die Teilchen seiner Meinung nach eher auf Kreisbahnen bewegten. 1757 veröffentlichet Leonhard Euler in seinem Artikel *Principes généraux du mouvement des fluides* das erste partielle Differentialgleichungssystem zur Beschreibung von Strömungen in reibungsfreien Fluiden, die später als Eulersche Gleichungen bekannt wurden. Diese Gleichungen sind nicht-linear hyperbolisch. Hyperbolisch bedeutet, dass die Lösungen dieser Gleichungen (in diesem Fall Wellen) nur sehr wenig gedämpft werden und sich somit auch über weite Strecken ausbreiten können. Die Eulerschen Gleichungen beschreiben dabei sogar das Brechen von Wellen, wodurch sich Schockwellen bilden, die sich mit hoher Geschwindigkeit bewegen können. Die Entstehung von Schockwellen, zeigt die Nicht-Liniearität der Lösung. Die Eulergleichungen werden häufig in Form des Impulssatzes geschrieben:

$$\frac{\delta v}{\delta t} + (v * \nabla) * v + \frac{1}{\rho}\nabla p = 0 \tag{1.1}$$

Hierbei ist v der Geschwindigkeitsvektor, ρ die Dichte und p der Druck. Durch den Napla-Operator erhält man im dreidimensionalen Fall somit drei Gleichungen, da der Napla-Operator dann die partiellen Ableitungen in alle drei Raumrichtungen darstellt.

Im achtzehnten Jahrhundert versuchten viele französische Mathematiker Wasserwellen unter bestimmten einschränkenden Bedingungen zu beschreiben. Um das Jahr 1786 gelang Joseph-Louis Lagrange eine Beschreibung unter der Annahme von langen Wellen mit kleinen Amplituden in flachem Wasser. Damit konnte er zum ersten Mal beweisen, dass Wasserteilchen sich in flachem Wasser auf elliptischen Bahnen bewegen. Je flacher das Wasser, desto „plattgedrückter“ werden die Ellipsen. In sehr tiefem Wasser bewegen sich die Teilchen hingegen auf Kreisbahnen. Dieses Thema wird im Abschnitt *Theoretische Grundlagen* dieser Arbeit noch ausführlich behandelt.

Auch war ab dem achtzehnten Jahrhundert klar, dass sich Flachwasserwellen physikalisch sehr unterschiedlich zu Tiefwasserwellen verhalten. Man spricht dann von Flachwasserwellen, wenn die Wassertiefe geringer

ist als die doppelte Wellenlänge.

1802 veröffentlichte der tschechische Wissenschaftler Jozef Gerstner seine Theory zur Bewegung von Wasserwellen. Seine Lösung war die erste exakte, nicht-lineare Beschreibung von Wasserwellen.

Dem britischen Astronom und Mathematiker George Biddell Airy gelang 1841 schließlich eine bis dato mathematisch unerreichte Beschreibung von Schwerewellen. Die verschiedenen Wellenformen werden im Detaill im Kapitel *Unterschiedliche Wellenarten* beschrieben. An dieser Stelle genügt es zu wissen, dass fast alle Wasserwellen Schwerewellen sind.

Airy konnte Newtons Ergebnis, dass die Wellenlänge entsprechend dem Quadrat der Wellenperiode wächst bestätigen. Außerdem wies er nach, dass auch die Wellengeschwindigkeit einen proportionalen Zusammenhang mit der Wellenperiode besitzt. Daraus resultiert, dass sich Wellen mit einer großen Wellenlänge deutlich schneller fortbewegen, als solche mit einer kleinen Wellenlänge.

Eine Weiterentwicklung des Ansatzes von Gerstner war die Fast Fourier Transformation, die Tessendorf 2001 in seinem Paper vorstellte [3]. Diese Technik basiert im Gegensatz zu den Gerstner-Wellen, allerdings auf stochastischen Methoden, die empirisch bewiesen werden konnten.

Alle diese Theorien genügen allerdings nicht, um das Verhalten von Wasserwellen, die auf einen Strand auflaufen, zu beschreiben. Für eine solche Beschreibung ist entscheidend, dass sich beim Auflaufen auf den Strand alle Wellenparameter ändern, bis auf die Wellenperiode. Diese bleibt immer konstant. In den beiden Artikeln [4] und [5] werden verschiedene Ansätze zur Lösung des Problems vorgestellt. Es werden verschiedenen Abwandlungen der Gerstner-Wellen verwendet und mit Wellen, die durch das Auftreffen auf den festen Untergrund ihre Richtung ändern, kombiniert.

Auch für Wellen im Flachwasser gibt es andere Vorgehensweisen, die hier nicht behandelt werden. Eine Lösung der so genannten *Shallow Water Equation* bietet zum Beispiel das Paper von Lee und Sullivan [6].

Eine Zusammenstellung über Bücher und Internetseiten, die sich mit der Darstellung von Wasser auf Computern beschäftigen, sowie eine Liste von Grafik-Engines mit integrierter Wassersimulation bietet diese Seite [7].

1.3 Ozeanographie

Die physikalische Ozeanographie ist ein Teilgebiet der allgemeinen Meereskunde. In diesen Bereich fallen abgesehen von der hier relevanten Erforschung des Seegangs (Wellenbewegung) auch noch die Erforschung von Turbulenzen, Strömungen, Lichtdurchlässigkeit und einigen weiteren Aspekten.

Das Ziel der Ozeanographie ist die systematische Beschreibung der physikalischen Vorgänge im Meer unter Berücksichtig verschiedener Parame-

ter, wie z.B. Temperatur, Druck und Strömung. Auch die Wechselwirkung zwischen ozeanischen und atmosphärischen Strömungen, zur Wetterforschung sind Teilgebiet der Ozeanographie.

Methoden wie die Fourier Transformation, die von Tessendorf auf Wasserwellen angewandt wurde, beruhen teilweise auf Erkenntnissen der Ozeanographie. So ist zum Beispiel durch empirische Messungen und Beobachtungen in der Ozeanographie bekannt, dass das so genannte Phillips Spektrum eine sehr gute Näherung für das Verhalten von Meereswellen darstellt.

Die Ozeanographie hilft der Informatik nicht nur bei den physikalischen Grundlagen zur Darstellung von Wasser, sondern sie profitiert umgekehrt auch von dieser. Durch physikalisch korrekte, sehr komplexe Simulationen, unter Berücksichtigung zahlloser Faktoren, können Einflüsse von Strömungen erforscht und Prognosen für die Zukunft erstellt werden.

2 Wellentheoretische Grundlagen

2.1 Wellenbewegung im Allgemeinen

In der Mechanik bezeichnet eine Welle eine räumliche und zeitliche Zustandsänderung physikalischer Größen. Eine Welle entsteht, wenn einer Kette von Oszillatoren (Schwingkörpern) periodisch Energie zugeführt wird und die miteinander gekoppelten Oszillationen dadurch nacheinander in Schwingungen versetzt werden [8]. Wellen können sich sowohl quer als auch längs ausbreiten. Abbildung 1 zeigt die beiden Fälle.

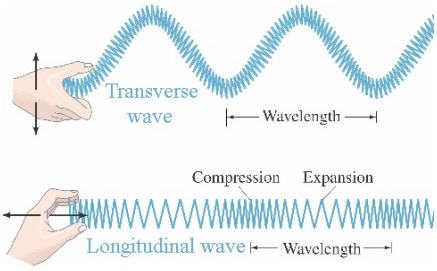

Abbildung 1: Darstellung von Transversal- und Longitudinalwellen anhand einer Feder. Quelle: [9]

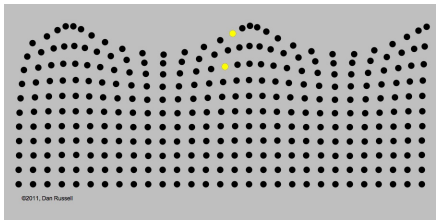

Abbildung 2: Wellenbewegung am Beispiel von Wasserpartikeln. Zu sehen ist die longitudinale und die transversale Bewegungsrichtung. Quelle: [10]

Eine Welle wird dann transversal genannt, wenn der Schwingungsvektor parallel zur Ausbreitungsrichtung steht. Im Gegensatz dazu wird eine Welle longitudinal genannt, wenn der Schwingunsvektor senkrecht zur Ausbreitungsrichtung steht. Intuitiver ist die Vorstellung, dass über eine Transversalwelle Wellenberge und -täler laufen, während es bei einer Longitudinalwelle Verdichtungen und Verdünnungen sind.

Allgemeine physikalische Größen zur Beschreibung von Wellen sind:

- Amplitude A: maximale Auslenkung der Welle von der Position der Null-Linie aus gemessen (höchster Wellenberg bzw. niedrigstes Wellental)
- Wellenlänge λ: kürzeste Distanz zwischen zwei Wellenbergen/-tälern (räumliche Periode)
- Schwingungsdauer T: Zeit in Sekunden für genau einen Wellendurchgang (zeitliche Periode)
- Frequenz $f = 1/T$
- Phasengeschwindigkeit $v_{Ph} = \lambda/T$

Bei Wasserwellen müssen sowohl transversale als auch longitudinale Komponenten berücksichtigt werden. Die Grafik 2 zeigt, wie sich die Partikel bewegen. Es wird eine waagerechte und eine senkrechte Bewegungsrichtung deutlich. Das bedeutet, dass für die Wellenanimation nicht nur der Höhenwert eine Rolle spielt.

2.2 Unterschiedliche Wellenarten

Die meisten Wellenformen entstehen durch den Einfluss von Wind. Dabei wird die Energie des Windes auf die Welle übertragen. Die Welle selbst besteht also aus Energie. Natürlich gibt es auch andere Faktoren, die für die Entstehung von Wellen verantwortlich sind. Man kann verschiedene Wellensorten durch ihre Entstehung klassifizieren. Andererseits können auch Klassifizierungen aufgrund der Wassertiefe oder der Art und Weise wie eine Welle bricht vorgenommen werden. Es gibt viele verschiedene Arten von Wellen. Hier werden nur die wichtigsten aufgezählt.

Schwerewellen bzw. Windwellen sind Wellen, die über länger Zeiträume erhalten bleiben und sich somit über den offenen Ozean viele Kilometer weit bewegen können. Sie entstehen in Folge der Schwerkraft und der Massenträgheit.

Wasserwellen sind meist externe Schwerewellen, das bedeutet, Schwerewellen die sich entlang der Grenzfläche zweier Fluide ausbreiten. Die beiden Fluide sind in diesem Fall dann Wasser und Luft. Schwerewellen sind windinduziert, werden also aufgrund des Windes erzeugt. Dieser Mechanismus wird als Kelvin-Helmholtz-Instabilität bezeichnet.

Um 1871 entdeckten Kelvin und Helmholtz unabhängig voneinander die Instabilität, die zwischen zwei bewegten Fluiden herrscht. Kelvin erklärte seine Theorie so: einen flachen, wellenlosen Ozean bezeichnete er als stabiles System. Ohne äußere Einflüsse wird dieses System auch immer stabil bleiben. Durch die Wirkung verschiedener Kräfte kann aus dem stabilen ein instabiles System werden.

Kelvin ging als Startpunkt von einer infinitisimal kleinen Sinuswelle aus, die sich frei über einen Ozean bewegt. Bewegen sich nun Luftteilchen nah am Wasser, müssen diese sich über die Wellenberge hinweg bewegen, befinden sich also zumindest teilweise auf einer Kreisbahn. Körper, die sich auf Kreisbahnen bewegen unterliegen der Zentrifugalkraft, die sie nach außen drückt. Die Teilchen werden also durch diese Kraft nach oben gedrückt. Dadurch entsteht direkt über der Welle ein Unterdruck. Da jedes System einen Druckausgleich anstrebt, wird eine Kraft entstehen, die die Wasserteilchen im Wellenberg nach oben zieht um den Druck auszugleichen. So wächst die zunächst kleine Welle immer weiter an.

Im Wellental tritt exakt der gleiche Effekt auf. Die Zentrifugalkraft drückt die Luftteilchen nach unten und durch den steigenden Luftdruck wird das

Wasser nach unten gedrückt [1].

Mittlerweile gibt es weitreichendere Erkenntnisse darüber, wie durch Wind Wellen entstehen können. Die Ergebnisse von Kelvin und Helmholtz reichen jedoch nicht aus, um die Wellenentstehung bei geringen Windgeschwindigkeiten zu erklären. Kelvin nahm zwar bereits an, dass die Berücksichtigung von Windturbulenzen dieses Problem lösen könnte, allerdings fehlten ihm zu seiner Zeit die mathematischen Mittel zu deren Berechnung.

Die wichtigsten Größen zur Berechnung von Schwerewellen sind die Windgeschwindigkeit, die Zeit die der Wind auf das Meer wirkt und die Länge der Strecke (Fetch) auf die der Wind Einfluss nimmt [11].

Kapillarwellen werden hauptsächlich durch die Oberflächenspannung bestimmt. Diese Kraft wird bei den großen Schwerewellen vernachlässigt. Bis zu einer Wellenlänge von wenigen Zentimetern werden Wellen fast ausschließlich durch die Oberflächenspannung charakterisiert. Je größer die Wellenlänge wird, desto mehr gehen Kapillarwellen in Schwerewellen über, bei denen die Oberflächenspannung vernachlässigt werden kann und die Schwerkraft eine immer größere Rolle spielt. Der Übergang von Kapillar- zu Schwerewellen, wird im nächsten Kapitel ausführlich behandelt.

Flachwasserwellen können aus Schwerewellen entstehen, wenn diese in flacheres Wasser gelangen. Flachwasserwellen treten allerdings, auch wenn der Name etwas anderes vermuten lässt, nicht ausschließlich in flachem Wasser auf. Viel mehr bezeichnet Flachwasserwelle die Beziehung zwischen der Wellenlänge und der Wassertiefe. Auch auf diese Wellenform wird im nächsten Kapitel im Detail eingegangen.

Gezeitenwellen werden von der Anziehungskraft zwischen Erde, Mond und Sonne bestimmt. Im Gegensatz zu Schwerewellen, die eine Periodendauer von wenigen Sekunden haben, liegt die Periodendauer bei Gezeitenwellen bei mehreren Stunden. Wenn man bedenkt, dass die Flut circa zwei Mal pro Tag eintritt, entspricht die Wellenlänge einer normalen Gezeitenwelle dem halben Erdumfang (d.h. circa 20.000 km). Wie im folgenden Abschnitt erklärt, gehören Gezeitenwellen (wenn man von einer durchschnittlichen Wassertiefe von 4 km ausgeht) also zu den Flachwasserwellen [11], obwohl sie nicht ausschließlich im flachen Wasser auftreten ($4km < 3183km = \frac{20000km}{2\pi}$).

Tsunamiwellen werden durch Erdbeben, tektonische Verschiebungen oder Vulkanausbrüche ausgelöst. Mit Periodendauern im Minutenbereich und Wellenlängen von bis zu 500km, gehören auch sie zu den Flachwasserwellen ($4km < 80km = \frac{500km}{2\pi}$).

Seiche treten nur in ruhenden Gewässern wie Seen oder Hafenbecken auf. Sie entstehen durch die Reflexion der Wellen an den Rändern des Gewässers. Seiche sind stehende Wellen.

2.3 Oberflächenwellen

Als Ausgangspunkt werden folgende Annahmen getroffen: Bei einer Welle, die nicht bricht bewegen sich die Wasserteilchen annähernd auf Kreisbahnen. Die Geschwindigkeit v dieser Teilchen hat also nichts mit der Phasengeschwindigkeit c zu tun, mit der sich die Welle fortbewegt. Es lässt sich beobachten, dass sich die Teilchen im Wellental ein wenig schneller bewegen, als die Teilchen, die sich auf einem Wellenberg befinden.

Das kann folgendermaßen erklärt werden: In der Physik gilt immer das Gesetz der Energieerhaltung. In diesem Fall bedeutet es, dass ein Wasserteilchen immer die gleiche Energie hat, ob es sich nun oben oder unten auf der Welle befindet. In einer Formel ausgedrückt bedeutet das:

$$E_{berg} = E_{tal} \tag{2.1}$$

$$E_{pot,berg} + E_{kin,berg} = E_{pot,tal} + E_{kin,tal} \tag{2.2}$$

$$E_{pot,berg} - E_{pot,tal} = E_{kin,tal} - E_{kin,berg} \tag{2.3}$$

$$\Delta E_{pot} = E_{kin,tal} - E_{kin,berg} \tag{2.4}$$

Energie setzt sich immer aus der potentiellen und der kinetischen Energie zusammen. Die potentielle Energie ist die Lageenergie. Hier kommt es meist auf die Höhe an, in der sich ein Körper befindet. Die allgemeine Formel der potentiellen Energie lautet:

$$E_{pot} = mgh \tag{2.5}$$

wobei m die Masse eines Körpers ist (hier: Masse eines Teilchens), $g = 9,80665\frac{m}{s^2}$ die Erdbeschleunigung und h die Höhe, in der sich ein Körper befindet. Wenn jetzt h der Amplitude entspricht (Höhe eines Wellenbergs im Bezug auf den normalen Wasserspiegel), dann gewinnt ein Teilchen, dass die Höhendifferenz $2h$ durchläuft (vom Berg zum Tal), eine potentielle Energie von:

$$\Delta E_{pot} = E_{pot,berg} - E_{pot,tal} = mg2h. \tag{2.6}$$

Die kinetische Energie ist die Bewegungsenergie. Sie ist maßgeblich von der Geschwindigkeit abhängig mit der sich ein Körper bewegt. Die Formel lautet:

$$E_{kin} = \frac{1}{2}mv^2 \tag{2.7}$$

Geht man jetzt noch davon aus, dass sich die Teilchen im Uhrzeigersinn auf Kreisbahnen bewegen, wenn die Welle sich von links nach rechts bewegt,

dann weiß man, dass die Geschwindigkeit auf dem Wellenberg $c + v$ und im Wellental $c - v$ beträgt. Jetzt werden diese Erkenntnisse, zusammen mit Formel 2.6 und 2.7 in Formel 2.4 eingesetzt:

$$mg2h = \frac{1}{2}m(c-v)^2 - \frac{1}{2}m(c+v)^2 \tag{2.8}$$

$$gh = cv \tag{2.9}$$

$$v = \frac{gh}{c} \tag{2.10}$$

Anschließend setzten wir die allgemein geltenden Formeln $v = r\omega$ ein, wobei w für die Winkelgeschwindigkeit steht. Der Radius r ist hier gleich der Wellenamplitude h und wird gekürzt. Wir erhalten:

$$\omega = \frac{g}{c} \tag{2.11}$$

Aus den aus der Physik bekannten Formeln $\omega = 2\pi f$ und $c = \lambda f$ zusammen mit dieser Formel erhalten wir für die Wellengeschwindikeit c:

$$c = \sqrt{\frac{g\lambda}{2\pi}} \tag{2.12}$$

Was bedeutet diese Formel nun?

Man kann sehen, dass die Wellenlänge λ im Zähler des Bruches steht. Daraus lässt sich schließen, dass lange Wellen (große Wellenlängen) deutlich schneller laufen als kurze Wellen (geringe Wellenlängen). Werden die Wellenlängen noch kürzer (Kapillarwellen) spielt die potentielle Energie der Wasserteilchen im Schwerefeld der Erde keine große Rolle mehr. Wichtiger wird hier die Oberflächenspannung wie in Kapitel 2.2 bereits erwähnt. Zusammen mit der Formel für den Schweredruck

$$p = g\rho h, \tag{2.13}$$

in der ρ für die Meerwasserdichte $10257\frac{kg}{m^3}$ steht, kann man aus der Formel 2.6 (durch umformen nach gh) die Formel

$$E = 2\frac{mp}{\rho} \tag{2.14}$$

erhalten.

Da jetzt nicht mehr der Schweredruck, sondern der Oberflächendruck die entscheidende Größe ist, wird der Schweredruck durch folgende Formel ersetzt:

$$p = \frac{\sigma}{r} = \sigma\frac{4\pi^2}{\lambda^2}h \tag{2.15}$$

σ steht hier für die spezifische Oberflächenenergie, allgemein auch Oberflächenspannung genannt. Die Herleitung dieser Formel ist hier nicht relevant, funktioniert aber über das invertierte Maximum der zweiten Ableitung einer normalen Schwingungsfunktion.

Der Oberflächendruck wird in Gleichung 2.14 eingesetzt und das Ergebnis anstelle von $mg2h$ in Gleichung 2.8 eingesetzt. Danach wird, wie oben beschrieben, weitergerechnet und als Ergebnis erhält man:

$$c_{kap} = \sqrt{\frac{2\pi\sigma}{\rho\lambda}} \tag{2.16}$$

Im Vergleich zu der Geschwindigkeit für Schwerewellen (Formel 2.12) fällt sofort auf, dass c_{kap} mit der Zunahme von λ nicht etwa auch zunimmt, wie es für die Geschwindigkeit der Schwerewelle galt. Hier gilt die umgekehrte Zuordnung: je größer die Wellenlänge, desto geringer die Geschwindigkeit. Mit zunehmender Wellenlänge werden die Kapillarwellen nicht nur langsamer, sondern an einem gewissen Punkt gehen sie in Schwerewellen über. Dieser Punkt ist dann erreicht, wenn die beiden Geschwindigkeiten gleich sind, sich also die Kurven der jeweiligen Geschwindigkeitsverläufe treffen:

$$c_{schwere} = c_{kap} \tag{2.17}$$

$$\lambda = 2\pi\sqrt{\frac{\sigma}{\rho g}} \tag{2.18}$$

Bei Wasser gehen Kapillarwellen in Schwerwellen über bei $\lambda = 0,017m$.

Was hier deutlich wird, ist das Prinzip der Dispersion. Dispersion beschreibt die Abhängigkeit der Phasengeschwindigkeit von der Wellenlänge, wie wir es hier sehen. Man spricht von normaler Dispersion, wenn, wie bei Schwerewellen, λ mit der Geschwindigkeit zunimmt und im anderen Fall, wie bei Kapillarwellen, von anormaler Dispersion. Ein anderes Kriterium für Dispersion hat mit den verschiedenen Geschwindigkeiten der Wellen zu tun. Wenn sich mehrere Wellen überlagern und man die Intensitäten betrachtet, kann durch konstruktive Interferenz ein Intensitätsmaximum entstehen.

Konstruktive Interferenz bedeutet dabei, dass sich die Wellen so überlagern können, dass ihre Maxima an der selben oder beinahe an der selben Stelle liegen und sie sich somit gegenseitig verstärken. Ihre Intensitäten werden aufaddiert. Das Maximum, das entsteht, zeichnet eine Wellengruppe aus. Wellengruppen bestehen in der Regel aus Wellen mit sehr ähnlichen Wellenlängen und somit sehr ähnlichen Geschwindigkeiten, so dass sie sich nahezu gemeinsam fortbewegen.

Hätten alle diese Teilwellen die gleiche Geschwindigkeit, läge keine Dispersion vor. Nur wenn sich die einzelnen Phasengeschwindigkeiten von der Gruppengeschwindigkeit unterscheiden, spricht man von Dispersion. Bei Schwerewellen ist die Gruppengeschwindigkeit beispielsweise kleiner als die Phasengeschwindigkeit, wohingegen bei Kapillarwellen der entgegengesetzte Fall eintritt.

Im Flachwasser liegt der Fall noch einmal etwas anders: Die Annahme, dass sich die Wasserteilchen auf Kreisbahnen bewegen, gilt hier nicht mehr. Je flacher das Wasser und je näher die Teilchen am Meeresgrund, umso flacher werden auch die Kreisbahnen auf denen sie sich bewegen. Die Kreise werden zu Ellipsen. Diese werden immer weiter abgeflacht und ähneln am Meeresboden mehr Linien als Ellipsen, da sich die Teilchen nur parallel zum Boden bewegen können. Eine Veranschaulichung bietet folgende Abbildung:

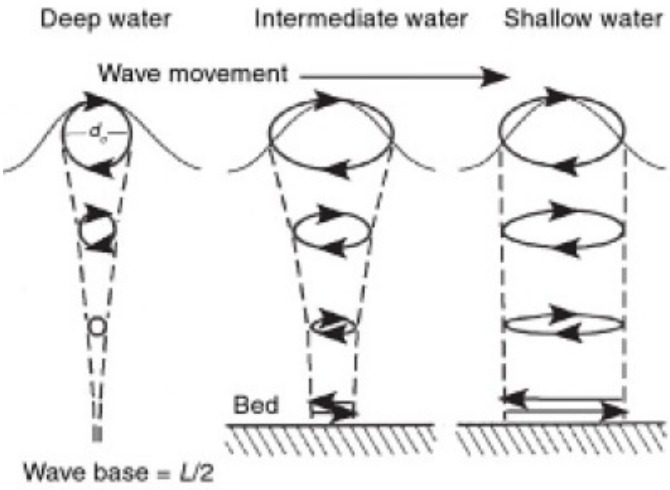

Abbildung 3: Links: im Tiefwasser ($\frac{\lambda}{2} < 2$) bewegen sich die Wasserteilchen auf kreisförmigen Bahnen. Mitte und Rechts: Je flacher das Wasser wird, umso flacher und länglicher werden die Ellipsen, auf denen sich die Wasserteilchen bewegen. Quelle: [11]

Ab einer Wassertiefe H, kleiner als $\frac{\lambda}{2\pi}$, tritt diese Störung auf. Deshalb muss man ab diesem Grenzwert die Geschwindigkeit nicht mehr von der Wellenlänge abhängig machen, sondern ausschließlich von der Wassertiefe. In der Formel 2.12 ersetzen wir deshalb $\frac{\lambda}{2\pi}$ durch H und erhalten:

$$c = \sqrt{gH}. \tag{2.19}$$

Der Umstand, dass c nun nicht mehr von λ abhängt bedeutet natürlich auch, dass hier keine Dispersion auftritt. Flachwasserwellen kommen nicht nur in seichten Gebieten nahe der Küste vor, sondern auch auf Seen und Innenmeeren.

Zusammenfassend wurde in diesem Abschnitt also gezeigt, dass sehr kleine Kapillarwellen, die durch die Oberflächenspannung entstehen zu Beginn sehr schnell sind. Wächst ihre Wellenlänge, werden sie immer langsamer, bis sie im Bereich von circa einem Zentimeter in Schwerwellen übergehen. Das ist genau die Wellenlänge mit der minimalen, bei Wasserwel-

len vorkommenden Geschwindigkeit. Wenn die Wellenlänge nun weiter wächst, wird die Welle wieder beschleunigt. Eine Ausnahme bilden hier die Flachwasserwellen. Ihre Geschwindigkeit hängt von der Wassertiefe ab.

Als Quelle für dieses Unterkapitel wurde Kapitel 4.6 des *Gerthsen Physik* [12] herangezogen.

2.4 Die verschiedenen Wellentheorien

In diesem Unterkapitel werden verschiedene Wellentheorien vorgestellt. Als Einleitung wird erklärt, wie bereits durch die einfache Überlagerung von Sinuswellen erstaunlich gute Ergebnisse erzielt werden können. Danach werden zwei Verfahren vorgestellt, die zumindest grundlegend auf diesem Ansatz aufbauen. Als Quellen für dieses Kapitel wurden die Paper von Tessendorf [3] und Nvidia [13] genutzt.

2.4.1 Summe von Sinuswellen

Die sicherlich einfachste Methode Wellen darzustellen, ist die Verwendung der trigonometrischen Funktionen Sinus und Kosinus. Natürlich sind Wasserwellen sehr viel komplexer als eine einfache Sinuskurve. In Grafik 4 sind einige mögliche Sinuskurven, die sich in Frequenz, Wellenlänge und Amplitude unterscheiden, aufgezeichnet.

Legt man diese, wie in Abbildung 5 gezeigt übereinander, so entsteht ein Muster, dass entfernt an den Querschnitt einer Welle erinnert. Im 3D Raum sehen die Sinuskurven dann eher wie in Grafik 6 aus. Hier haben die einzelnen Sinuskurven Richtungsvektoren, welche die Wellenrichtung darstellen. Überlagern sich diese wie im 2D auch, so entsteht eine Wellenoberfläche wie in Abbildung 7, die erstaunlich realistisch wirkt. Je mehr solcher Wellen man addiert, umso realistischer das Ergebnis.

Die Formel für die Höhe an der Postion $\begin{pmatrix} x \\ z \end{pmatrix}$ zum Zeitpunkt t lautet daher:

$$H(x, z, t) = \sum_{i=1}^{N} A \; sin(\omega \; \vec{k} * \begin{pmatrix} x \\ z \end{pmatrix} + t \; \phi) \tag{2.20}$$

Hierbei ist N die Anzahl der Wellen, die sich überlagern. A ist die Amplitude und λ die Wellenlänge wie in 2.1 erklärt. $\vec{k}$ ist der Richtungsvektor der Welle. Er wird über das Skalarprodukt mit dem aktuellen Ortsvektor multipliziert.

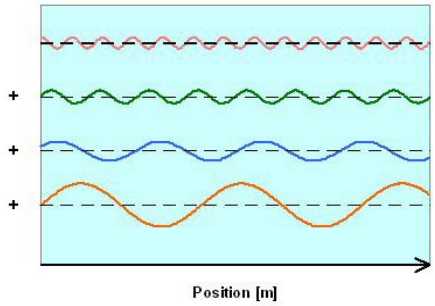

Abbildung 4: Vier Sinuskurven mit unterschiedlichen Frequenzen, Wellenlängen und Amplituden. Quelle: [10]

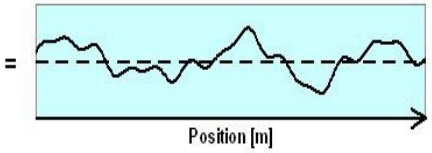

Abbildung 5: Überlagerung der vier Sinuswellen. Quelle: [10]

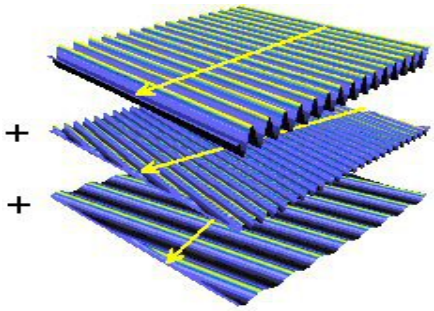

Abbildung 6: Drei Sinuskurven mit unterschiedlichen Frequenzen, Wellenlängen und Amplituden in einer Ebene. Quelle: [10]

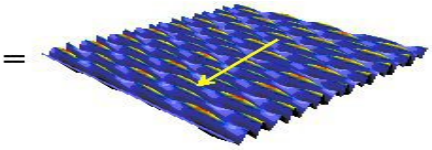

Abbildung 7: Überlagerung der drei Sinuswellen in der Ebene. Quelle: [10]

Für ω und ϕ gelten die Formeln:

$$\omega = \frac{2\pi}{\lambda} \tag{2.21}$$

$$\phi = v\,\frac{2\pi}{\lambda} \tag{2.22}$$

wobei v die Fortschrittsgeschwindigkeit der Welle ist. ϕ nennt man Gangunterschied. Dieser Faktor ist für den Versatz der Wellen verantwortlich. Das bedeutet, dass sich eine Welle anhand ihrer Geschwindigkeit schneller fortbewegen kann, als eine andere Welle.

Zu beachten ist bei diesem Ansatz, dass hier nur der Höhenwert verändert wird. Die Werte der horizontalen x, z-Ebene bleiben bestehen. Eine solche Veränderung der x- und z-Werte wird erst im nächsten Abschnitt bei den Gerstner-Wellen behandelt. Daher gilt auch: $P(x, z, t) = (x, H(x, z, t), z)$.

Wenn man diese Fläche aus einer geringen Entfernung betrachtet, fällt die Regelmäßigkeit des Wellenmusters kaum auf, vor allem dann nicht, wenn man kleine Amplituden wählt, also ein Meer bei ruhigen Wetterbedingungen simuliert. Problematisch wird es, wenn die Wellen höher werden. Man stellt schnell fest, dass die Wellenberge, abgerundet durch die Sinuskurven, für stürmisches Wetter zu rund sind. Das ist ein Problem, dass sich durch den Ansatz von Gerstner, wie wir im nächsten Kapitel sehen werden, lösen lässt.

Ebenfalls schwierig wird es, wenn man die Meerebene von weiter weg betrachtet: dann kann man eindeutig erkennen, dass jede Wellengruppe gleich aussieht. Dieses Problem wird auch bei den Gerstner-Wellen im folgenden Kapitel auftreten.

In Kapitel 3 wird erklärt, dass die Oberflächennormalen zur Berechnung von Licht und Reflexion benötigt werden. Deshalb müssen an dieser Stelle bereits die Normalen berechnet werden. Aus der Schule ist bekannt, dass die erste Ableitung einer Funktion an einem Punkt, der Tangente an diesen Punkt entspricht. Eine Funktion im dreidimensionalen Raum, die von zwei Parametern abhängt, braucht dementsprechend eine Tangente in die eine und eine in die andere Richtung. Wir bezeichnen die erste partielle Ableitung in x-Richtung als Binormale. In y-Richtung bleibt der Name der ersten partiellen Ableitung als Tangente bestehen.

Binormale:

$$B(x,z) = (\frac{\delta x}{\delta x}, \frac{\delta H(x,z,t)}{\delta x}, \frac{\delta z}{\delta x}) = \begin{pmatrix} 1 \\ \sum_{i=1}^{N} \omega\ k_x\ A\ cos(\omega\ \vec{k} * \begin{pmatrix} x \\ z \end{pmatrix} + t\ \phi) \\ 0 \end{pmatrix} \tag{2.23}$$

Tangente:

$$T(x,z) = (\frac{\delta x}{\delta z}, \frac{\delta H(x,z,t)}{\delta z}, \frac{\delta z}{\delta z}) = \begin{pmatrix} 1 \\ \sum_{i=1}^{N} \omega\ k_y\ A\ cos(\omega\ \vec{k} * \begin{pmatrix} x \\ z \end{pmatrix} + t\ \phi) \\ 0 \end{pmatrix} \tag{2.24}$$

Die Normale steht nun senkrecht, auf der Tangente und der Binormale, lässt sich also durch das Vektorprodukt der beiden berechnen. Somit lautet die fertige Lösung für die Normale:

$$N(x,z) = B(x,z) \times T(x,z) = \begin{pmatrix} \sum_{i=1}^{N} \omega\ k_x\ A\ cos(\omega\ \vec{k} * \begin{pmatrix} x \\ z \end{pmatrix} + t\ \phi) \\ -1 \\ \sum_{i=1}^{N} \omega\ k_y\ A\ cos(\omega\ \vec{k} * \begin{pmatrix} x \\ z \end{pmatrix} + t\ \phi) \end{pmatrix} \tag{2.25}$$

2.4.2 Gerstner-Wellen

Die natürliche Form von Wellen wurde bereits 1802 von dem tschechischen Wissenschaftler Jozef Gerstner beschrieben. Erst 1986 wurde dieses Konzept dann von Fournier und Reeves für die Computergrafik umgesetzt [14].

Gerstner beschreibt die Bewegung der Wasseroberfläche mit Hilfe einer trochoidalen Wellenform. Ein Trochoid beschreibt die Linie, die von einem

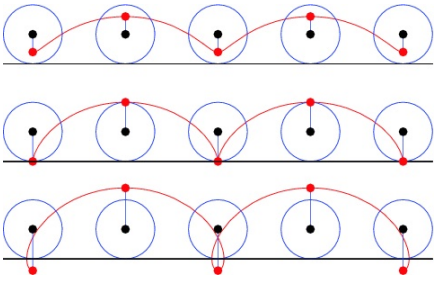

Abbildung 8: Beispiele für Trochoide in Abhängigkeit von der Distanz zum Kreismittelpunkt. Quelle: [15]

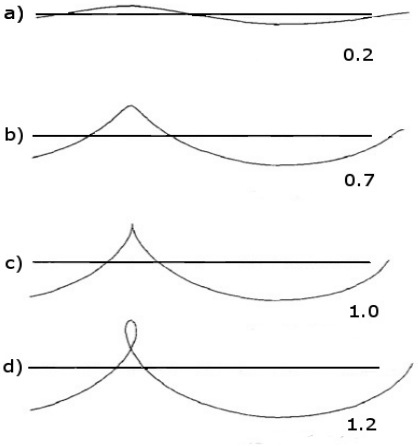

Abbildung 9: Bei bestimmter Parameterwahl, verändert sich die Steilheit der Wellenberge. Quelle: [14]

Punkt im Kreis, mit dem Abstand d zum Kreismittelpunkt gezeichnet wird, wenn dieser an einer geraden Linie entlang rollt. In Grafik 8 werden drei Beispiele für mögliche Trochoide gegeben.

Gerstners Theorie basiert auf der Orbitalbewegung, die in Abbildung 3 gezeigt wurde. Auch andere Wellentheorien, wie zum Beispiel die von Airy-Laplace stützen sich auf diese Annahme. Stokes hingegen ging von einer zusätzlichen seitlichen Verschiebung in Form von einer Driftgeschwindigkeit aus. Diese ist jedoch für die Simulation zu vernachlässigen und auch in der Originalformel von Gerstner nicht enthalten.

Gerstner berücksichtigte in seinen Ausarbeitungen bereits das Abflachen der Kreisbahnen zu Ellipsen bei Annäherung zum Meeresboden.

Die allgemeine Formel von Gerstner zur Beschreibung einer trochoiden Wellenform lautet:

$$x = x_0 - \sum_{i=1}^{N} \frac{\vec{k_i}}{|\vec{k_i}|} A_i \; sin(\vec{k_i} * \begin{pmatrix} x \\ z \end{pmatrix} - \omega_i \; t + \phi_i) \tag{2.26}$$

$$y = \sum_{i=1}^{N} A_i \; cos(\vec{k_i} * \begin{pmatrix} x \\ z \end{pmatrix} - \omega_i * t + \phi_i) \tag{2.27}$$

$$z = z_0 - \sum_{i=1}^{N} \frac{\vec{k_i}}{|\vec{k_i}|} A_i \; sin(\vec{k_i} * \begin{pmatrix} x \\ z \end{pmatrix} - \omega_i \; t + \phi_i) \tag{2.28}$$

Dabei ist zu beachten, dass der Vektor $\begin{pmatrix} x \\ z \end{pmatrix}$ die Grundfläche der Wasserebene darstellt. x_0 und z_0 sind die Startwerte von x und z, entsprechen also

der normalen Gitterebene. A ist die Amplitude, also die maximale Auslenkung, t die Zeit und ϕ die Phasenverschiebung. $\vec{k}$ ist der zweidimensionale, horizontale Richtungsvektor der Wellen. $|\vec{k}|$ entspricht dem Betrag (also der Länge) des Vektors. Diese Länge wird allgemein auch als Kreiswellenzahl bezeichnet und steht wie folgt mit der Wellenlänge λ in Verbindung.

$$\lambda = \frac{2\pi}{|\vec{k}|} \tag{2.29}$$

Für die Kreisfrequenz ω gilt allgemein die Dispersionsrelation:

$$\omega = \sqrt{g\,|\vec{k}|} \tag{2.30}$$

wobei $g = 9,80665\frac{m}{s^2}$ die Erdbeschleunigung ist.

Wie in Kapitel 2.3 beschrieben, tritt Dispersion dann auf, wenn sich Wellen mit unterschiedlicher Wellenlänge und Geschwindigkeit als Gruppe zusammen bewegen. Die Dispersionsrelation beschreibt den Zusammenhang zwischen der Kreisfrequenz ω und dem Wellenvektor $\vec{k}$. Allgemein lautet die Formel für die Dispersionsrelation:

$$\omega = v\,|\vec{k}| \tag{2.31}$$

Normale Wellen sind mehr durch ihre Phasengeschwindigkeit v charakterisiert. Da für Wasserwellen die Schwerkraft eine entscheidendere Rolle spielt, git für diese der oben genannte Spezialfall.

Wenn man die Gerstner Formeln ohne die Summen implementiert, erhält man eine einzige Wellenfront, die sich optisch kaum von einer Sinuswelle unterscheidet. Das besondere an dieser Wellenfront im Gegensatz zu einer einfachen Sinuswelle ist jedoch die trochoidale Form. Das bedeutet, dass bei bestimmter Parameterwahl die Steilheit der Wellenberge steigt.

Wählt man die Parameter $\vec{k}$ und A so, dass $|\vec{k}|\ A > 1$ gilt, dann entsteht am oberen Ende der Welle ein „Looping“ wie im untersten Beispiel der Abbildung 9 gezeigt. Ein solcher Effekt ist natürlich nicht wünschenswert, da er jeglichem Realismus entbehrt. Deshalb muss bei der Wahl der Parameter darauf geachtet werden, dass gilt: $0 < |\vec{k}|\ A < 1$.

Je näher dieser Wert an der Null liegt, umso geringer wird, wie man ebenfalls in Abbildung 9 sehen kann, die Steilheit der Welle.

In der Natur kommen steile Wellen überwiegend bei stürmischer See vor. Das ist der Grund warum bei ruhigem Wetter der Sinusansatz ausreichen kann, für rauere Bedingungen aber nicht genügt. Allerdings zeigen Beobachtungen, dass auch bei ruhiger See Wellen mit kleinen Amplituden durchaus steiler werden können, als es für den Sinus möglich ist. Dementsprechend bietet der Ansatz von Gerstner eine gute Erweiterung des Sinusansatzes und für das geübte Auge auch hier eine bessere Beschreibung.

Um die Normale der Gerstner Wellen zu erhalten wird der gleiche Ansatz verfolgt wie im vorherigen Kapitel: zuerst werden Binormale und Tangente als partielle Ableitungen in x- und y-Richtung berechnet und danach das Kreuzprodukt gebildet. Die Formel für die Normalen der Gerstner Gleichungen lautet also:

$$N_g(x,z) = \begin{pmatrix} \sum_{i=1}^{N} k_x \; A_i \; sin(\vec{k_i} * \begin{pmatrix} x \\ z \end{pmatrix} - \omega_i \; t + \phi_i) \\ 1 - (\sum_{i=1}^{N} \quad A_i \; cos(\vec{k_i} * \begin{pmatrix} x \\ z \end{pmatrix} - \omega_i \; t + \phi_i) \\ \sum_{i=1}^{N} k_y \; A_i \; sin(\vec{k_i} * \begin{pmatrix} x \\ z \end{pmatrix} - \omega_i \; t + \phi_i) \end{pmatrix} \tag{2.32}$$

2.4.3 Fast Fourier Transformation

In diesem Unterkapitel wird noch eine weitere Berechnungsmethode für Wellen vorgestellt, die sowohl physikalische, als auch mathematische Korrektheit bietet. Fourier-Transformationen spielen in fast jedem Bereich der Physik, so wie in der Mathematik und in den Ingenieurwissenschaften eine große Rolle.

Der Namensgeber der Fourier-Transformation war Jean Baptiste Joseph Fourier, der 1822 in seiner *Théorie analytique de la chaleur* zeigte, dass sich ein Vorgang, der sich nach einer Periode T wiederholt aus harmonischen Schwingungen aufbauen lässt. Solche T-periodischen Vorgänge können durch Fourier-Reihen dargestellt werden. Dabei werden sie in ein diskretes Frequenzspektrum übertragen. Auch Vorgänge die nicht periodisch ablaufen (aperiodisch) können durch eine (kontinuierliche) Fourier Transformation auf ein Frequenzspektrum abgebildet werden. Hier kommen allerdings Fourier-Integrale zum Einsatz.

In der Informatik spielt eher die diskrete Fourier Transformation (DFT), oder ihre Weiterentwicklung, die Fast Fourier Transformation (FFT) eine Rolle, denn oftmals liegen keine kontinuierlichen sondern nur diskrete Ausgangswerte vor. Das Ergebnis der DFT sowie der FFT befindet sich im diskreten Frequenzspektrum. Der Unterschied zwischen DFT und FFT ist, dass bei der Fast Fourier Transformation Zwischenergebnisse, die bereits berechnet wurden verwendet werden und nicht jedes mal neu berechnet werden müssen. Für die allgemeine FFT gilt die Formel:

$$X(n) = \sum_{k=0}^{N-1} x_0(k) \; e^{i\frac{2\pi}{N} \; kn} \quad mit \quad n = 0, 1, ..., N-1 \quad und \tag{2.33}$$

$$x_0(k) = \sum_{n=0}^{N-1} X(n) \; e^{-i\frac{2\pi}{N} \; kn} \quad mit \quad n = 0, 1, ..., N-1 \tag{2.34}$$

Hier gibt es allerdings sehr viele unterschiedliche Formeln für verschiedene Anwendungsgebiete und auch die Symbolik ist in der Literatur alles andere als einheitlich. $X(n)$ ist hierbei eine, von den diskreten Werten n abhängige Funktion, die im Frequenzspektrum dargestellt werden soll. Grafisch ergibt diese Formel ein Linienspektrum.

Um Wasser darzustellen, wird eine leicht abgewandelte Form der FFT verwendet, wie sie von Tessendorf 2001 beschrieben wurde. Die betrachtete Funktion nennen wir jetzt h, für die Höhe der Wellen, da wir zunächst nur auf Höhenveränderungen achten wollen. Diese ist zusätzlich von der Position $\vec{x} = \begin{pmatrix} x \\ z \end{pmatrix}$ und auch von der Zeit t abhängig.

Mit dieser Formel ist es möglich, Werte, die im Frequenzspektrum berechnet wurden, ins Zeitspektrum zurückzurechnen - also handelt es sich genau genommen um eine inverse FFT.

$$h(\vec{x}, t) = \sum_{\vec{k}} \tilde{h}(\vec{k}, t)\, e^{i\vec{k}\vec{x}} \tag{2.35}$$

Betrachtet man diese Formel genauer fällt auf, dass $e^{i\vec{k}\vec{x}}$ eine komplexe Zahl ist, die nach der Eulerschen Formel auch als $cos(\vec{k}\vec{x}) + isin(i\vec{k}\vec{x})$ dargestellt werden kann. Also haben wir auch hier, wie in den vorherigen beiden Lösungsansätzen, wieder eine Kombination aus Sinus und Kosinus Wellen. Der wichtige Unterschied ist die Amplitude.

Für die Amplitude $\tilde{h}(\vec{k}, t)$ kann jetzt folgende Formel eingesetzt werden, die maßgeblich für die Oberflächenstruktur verantwortlich ist:

$$\tilde{h}(\vec{k}, t) = \tilde{h_0}(\vec{k})\, e^{i\omega(k)t} + \tilde{h_0}^*(\vec{k})\, e^{-i\omega(k)t} \tag{2.36}$$

Dabei steht $\omega(k)$, wie auch bei den Gerstner-Wellen für die Dispersionsrelation, die mit der Formel 2.30 ausgedrückt wird. Die Komplexe Zahl $\tilde{h_0}(\vec{k})$ muss noch bestimmt werden. $\tilde{h_0}^*(\vec{k})$ ist das komplex Konjugierte dieser Zahl. Das komplex Konjugierte einer komplexen Zahl erhält man durch Vertauschen des Vorzeichens des Imaginärteils, was einer Spiegelung an der reellen Achse entspricht.

Das Ziel ist es jetzt eine zufällige Menge von Amplituden zu finden, die den Wellenbewegungen auf einem Ozean möglichst nahe kommen. Forscher der Ozeanographie konnten durch Analyse von Messwerten, die unter realen Bedingungen im Meer aufgenommen wurden feststellen, dass die Amplituden einer statistisch unabhängigen Gauß-Verteilung entsprechen. Zusammen mit der komplexen gaußschen Zufallszahl $\xi = \xi_{re} + i\xi_{im}$ können die Amplituden mit Hilfe des Phillipsspektrums beschrieben werden. Dieses bietet ein gutes Model für Wasserwellen, die größer als Kapillarwellen sind.

$$P_h(\vec{k}) = A \frac{exp(-\frac{1}{(|\vec{k}|\lambda)^6})}{|\vec{k}|^4} |\vec{k} * \vec{w}|^2 \tag{2.37}$$

Hierbei bezeichnet A eine frei wählbare Konstante, die meist sehr klein ist. Nicht zu verwechseln sind die Dispersionsrelation ω und der Windvektor w. Der Windvektor ist wie bei den Gestner-Wellen auch horizontal gerichtet. Dieser sagt nicht nur etwas über die Richtung des Windes aus, sondern der Betrag dieses Vektors entspricht auch der Windgeschwindigkeit v ($v = |\vec{w}|$). Die maximale Wellenlänge λ wird dann erreicht, wenn gilt:

$$\lambda = \frac{v^2}{g} = \frac{|\vec{w}|^2}{g} \tag{2.38}$$

Zu beachten ist in Gleichung 2.37 auch, dass der Faktor $|\vec{k} * \vec{w}|$ durch das Skalarprodukt 0 wird, sollte der Windvektor w senkrecht auf dem Wellenvektor k stehen. In einer solchen Situation hätte die Welle keinen Einfluss auf die Gesamtsituation und das Phillipsspektrum würde dementsprechend 0. Um Wellen die zu klein sind zu unterdrücken, kann das Phillipsspektrum noch mit dem Term

$$e^{-|\vec{k}|^2 l^2} \tag{2.39}$$

multipliziert werden. Dabei wird l so gewählt, dass es deutlich kleiner ist als λ.

Um das Phillipsspektrum jetzt mit der gesuchten Zahl $\tilde{h_0}(\vec{k})$ zu verknüpfen, kommt folgende Formel zum Einsatz, die die vorhin erwähnte gaußsche Zufallszahl ξ enthält:

$$\tilde{h_0}(\vec{k}) = \frac{1}{\sqrt{2}} \xi \sqrt{P_h(\vec{k})} \tag{2.40}$$

Das Ergebnis sieht so aus wie in Abbildung 10 zu sehen ist. Da das Muster einem Schmetterling ähnelt, wird der Algorithmus manchmal auch *Butterfly Algorithmus* genannt.

Die einzige Variable, die bis jetzt noch nicht erklärt wurde, ist der Wellenvektor $\vec{k}$. Dieser kann jetzt nicht mehr, wie bei den Gerstner-Wellen, frei gewählt werden, sondern hat eine bestimmte vorgeschriebene Form:

$$\vec{k} = \begin{pmatrix} k_x \\ k_y \end{pmatrix} = 2\pi \begin{pmatrix} \frac{n}{L_x} \\ \frac{m}{L_y} \end{pmatrix} \tag{2.41}$$

Hierbei gilt:

$$-\frac{N}{2} \leq n < \frac{N}{2} \qquad und \qquad -\frac{M}{2} \leq n < \frac{M}{2} \tag{2.42}$$

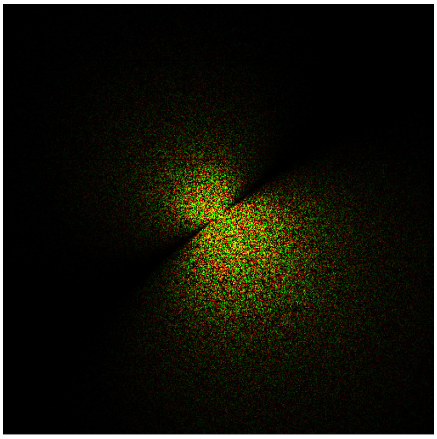

Abbildung 10: Verrauschte *Butterfly Textur* als gemeinsames Ergebnis des Phillipsspektrums und des Gauß-Filters. Quelle: [16]

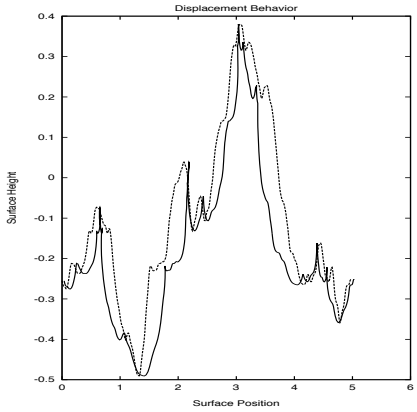

Abbildung 11: Die gestrichelte Linie zeigt das Ergebnis der FFT ohne die Veränderung der x- und z-Werte. Die durchgezogene Linie hingegen entspricht der zusätzlichen Anwendungen der Formel 2.44. Quelle: [3]

Die Wahl von N und M entscheidet über die Ausmaße der Wasserfläche. In der Implementation legen diese Werte später die Größe des Gitters fest. Sie sollten zwischen 64 und 512 liegen und immer Zweierpotenzen sein. L_x und L_y müssen so gewählt werden, dass gilt:

$$dx, dz > 2 \qquad mit \quad dx = \frac{L_x}{M} \quad und \quad dz = \frac{L_z}{N} \tag{2.43}$$

Bis jetzt wurde nur die Veränderung des Höhenwertes betrachtet. Im vorherigen Unterkapitel haben wir bereits festgestellt, dass Wellenberge, um realistisch zu wirken, nicht rund sein dürfen, sondern durchaus relativ spitzt zulaufen. Um diesen Effekt zu erzielen, werden jetzt zusätzlich zur Höhe auch die x- und z-Werte verändert, und zwar entsprechend folgender Formel:

$$D(\vec{x}, t) = \sum_k -i\frac{\vec{k}}{|\vec{k}|}\tilde{h}(\vec{k}, t)e^{i\vec{k}\vec{x}} \tag{2.44}$$

Die Ähnlichkeit zur Formel 2.35 ist deutlich zu erkennen. Der einzige Unterschied besteht darin, dass die x- und z-Werte zusätzlich mit dem Faktor $-i\frac{\vec{k}}{|\vec{k}|}$ multipliziert werden. Dadurch werden Gitterpunkte in der Nähe von Wellenbergen näher zusammen gedrückt und Punkte in Wellentälern werden auseinandergezogen. Grafik 11 zeigt den Vergleich. Außerdem bedeutet die Multiplikation mit i, dass nur der Imaginärteil verwendet wird.

Wie immer werden auch hier Normalen für die später Beleuchtung be-

nötigt. ϵ ist die Ableitung von h in alle drei Raumrichtungen:

$$\epsilon(\vec{x},t) = \nabla h(\vec{x},t) = \sum_k -i\vec{k}\tilde{h}(\vec{k},t)e^{i\vec{k}\vec{x}} \tag{2.45}$$

Die zugehörige Normale lautet dann:

$$\vec{N} = \begin{pmatrix} -\epsilon_x(\vec{x},t) \\ 1 \\ -\epsilon_z(\vec{x},t) \end{pmatrix} \tag{2.46}$$

In Kapitel 5 werden wir die Gerstner-Wellen mit den Ergebnissen der FFT vergleichen. Auch wenn die FFT mathematisch sehr komplex ist und damit längst nicht so intuitiv wie beispielsweise die Gerstner-Wellen, sind die mit dieser Methode erzielten Ergebnisse den Gerstner-Wellen um einiges Voraus.

In diesem Kapitel wurden die Quellen [3] und [12] verwendet.

3 Optische Aspekte

Nicht nur die Bewegung der Wellen macht Wasseroberflächen so interessant und einzigartig. Die Brechung von Sonnenstrahlen im Wasser, die Reflexion der Umgebung und des Himmels: das ist es was Wasser für uns ausmacht.

Die Optik ist natürlich nicht in jedem Bereich der Wassersimulationen von gleicher Bedeutung. In der Forschung und Technik steht physikalische Genauigkeit im Vordergrund. In Simulationen, Filmen und Computerspielen kommet es hingegen sehr stark auf die Optik an. Wasser, das nicht reflektiert und dessen Farbe unnatürlich wirkt, wird auch mit noch so tollen Wellenbewegungen niemals als realistisch angesehen werden.

Ein Lichtstrahl, der auf die Wasseroberfläche trifft, spaltet sich an dieser in zwei Teile. Ein Teil wird reflektiert, bleibt also oberhalb des Wassers, ein anderer Teil wird an der Grenzschicht gebrochen und verläuft auf veränderter Bahn im Wasser weiter.

Im Folgenden wird der Begriff Welle nicht mehr ausschließlich für Wasserwellen verwendet, sondern allgemein für elektromagnetische Wellen, wie z.B. Licht.

In den nachstehenden Unterkapiteln werden zunächst Reflexion, Brechung und deren Bezug zueinander, sowie deren Berechnung behandelt.

3.1 Reflexion

Im physikalischen Sinne bezeichnet Reflexion die Richtungsänderung einer Welle beim Auftreffen auf eine Grenzfläche zwischen zwei Medien, sodass diese Welle in das Medium aus der sie gekommen ist zurückgeworfen wird. Die Geschwindigkeit der Welle verändert sich dabei, anders als bei der Brechung, nicht.

Durch das Huygens'sche Prinzip ergibt sich das Reflexionsgesetz: Einfallswinkel = Ausfallswinkel. Das Huygens'sche Prinzip besagt, dass jeder Punkt einer Wellenfront als Ausgangspunkt einer Elementarwelle angesehen werden kann, die sich mit der gleichen Frequenz und Phasengeschwindigkeit wie die ursprüngliche Welle ausbreitet.

In Grafik 12 ist ein einfallender Strahl dargestellt, der teilweise gebrochen und teilweise reflektiert wird. Ein- und Ausfallswinkel werden im Bezug zur Flächennormale gemessen.

Für die Implementation bedeutet das, dass der Vektor zwischen der Kamera und jedem Punkt der Wasseroberfläche berechnet und an der jeweiligen Normale in diesem Punkt gespiegelt werden muss. Der Punkt, an dem dieser gespiegelte Vektor nun zum ersten mal auf ein Objekt trifft, ist derjenige, der die Farbe des Punktes auf der Wasseroberfläche bestimmt.

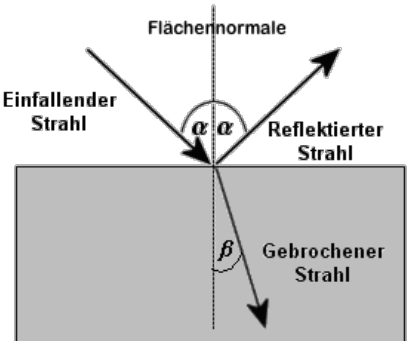

Abbildung 12: Reflexion und Brechung an der Grenzfläche zweier Medien. Quelle: [17]

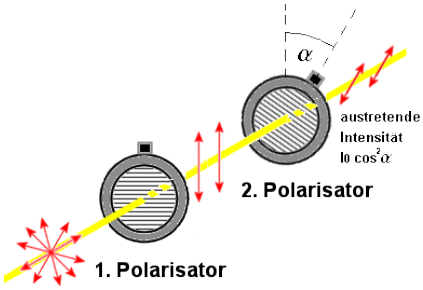

Abbildung 13: Ein paralleles Lichtbündel, dessen Schwingungsvektor zunächst, wie es bei natürlichem Licht er Fall ist, seine Richtung sehr schnell ändert, fällt nacheinander auf zwei Filter und wird dabei polarisiert. Quelle: [17]

3.2 Brechung

Beim Auftreffen des Lichts auf die Wasseroberfläche wird normalerweise nur ein Teil der Energie reflektiert. Der restliche Teil breitet sich im zweiten Medium, dem Wasser, aus. Hierbei handelt es sich um Transmission. Dieser Anteil wird durch den veränderten Wiederstand im Wasser an der Oberfläche gebrochen, sodass sich die Richtung und Geschwindigkeit der elektromagnetischen Welle ändert. Die Art und Weise wie eine solche Brechung stattfindet, wird durch das Snell'sche Brechungsgesetz beschrieben, dass lautet:

$$\frac{sin\alpha}{sin\beta} = \frac{c_1}{c_2} = \frac{n_1}{n_2} \tag{3.1}$$

Wie in Abbildung 12 zu sehen, ist α der Einfallswinkel und β der Ausfallswinkel. c_1 stellt die Ausbreitungsgeschwindigkeit des Lichts im Medium 1, also der Luft, c_2 die Lichtgeschwindigkeit im Medium 2, dem Wasser, dar. Da die Lichtgeschwindigkeit in der Luft circa $0,28\%$ geringer als im Vakuum, in Wasser aber sogar 25% geringer ist, folgt aus der obigen Formel unmittelbar, dass der Winkel zwischen Flächennormale und gebrochenem Strahl im Wasser kleiner sein muss als in der Luft ($\beta < \alpha$). Der Strahl wird also zur Normale hin gebrochen. n_1 und n_2 sind die verschiedenen Brechzahlen der Medien. Das Snell'sche Brechungsgesetz kann, durch die Annahme einer konstanten Frequenz aus den Maxwellgleichungen, die den Zusammenhang zwischen elektromagnetischen Feldern beschreiben, hergeleitet werden.

3.3 Fresnelsche Formeln

Wie in den vorherigen Unterkapiteln erklärt, findet an der Wasseroberfläche sowohl Reflexion als auch Brechung statt. Aber wie viel Prozent der Lichtwelle wird reflektiert und wie viel wird gebrochen?

Um diese Frage zu beantworten kommen die Fresnelschen Formeln zur Anwendung. Die Arbeit des französischen Physikers Augustin Jean Fresnel zu Beginn des 19. Jahrhunderts, wird heute als eine der wichtigsten Grundlagen der modernen Wellentheorie des Lichts angesehen.

Fresnel beschäftigte sich sowohl theoretisch, als auch experimentell mit der Polarisation von Licht. Natürliches Licht besteht aus elektromagnetischen Transversalwellen und ist zunächst unpolarisiert, schwingt also, wie in Abbildung 13 zu sehen, in alle Richtungen. Filter stellen eine Möglichkeit dar Licht zu polarisieren. Dabei muss man sich den Lichtvektor vor Augen halten. Steht dieser parallel zu den Stäben des Filters, so kann er den Filter ungehindert durchqueren. Steht er senkrecht kann das Licht nicht passieren. Dementsprechend wird bei der Polarisation nur der Teil des Lichtes durchgelassen, dessen Schwingungsvektor einen parallelen Anteil zum Filter hat. Der Anteil des Lichtes nach dem ersten, sowie nach dem zweiten Filter ist also linear polarisiert.

Polarisation durch Filter ist das klassische Beispiel, aber es gibt auch andere Methoden Licht zu polarisieren: zum Beispiel durch Reflexion. Hier kann allerdings nicht ganz so leicht eine lineare Polarisation erreicht werden. Nach Brewster ist dies nur für den Spezialfall, dass reflektierter und gebrochener Strahl im $90°$ Winkel zueinander stehen, möglich [8]. Durch die Reflexion und Brechung an der Wasseroberfläche wird das Licht also polarisiert. Das ist deshalb von Nutzen, da dies die Möglichkeit bietet das Verhältnis zwischen der Intensität des einfallenden und des reflektierten bzw. gebrochenen Strahls zu errechnen. Nach Fresnel gilt für diese Verhältnisse:

$$\frac{E_r}{E_e} = -\frac{sin(\alpha - \beta)}{sin(\alpha + \beta)} \tag{3.2}$$

$$\frac{E_g}{E_e} = -\frac{tan(\alpha - \beta)}{tan(\alpha + \beta)} \tag{3.3}$$

Dabei ist E der Betrag des elektrischen Feldvektors, für den einfallenden (E_e), reflektierten (E_r) und gebrochenen (E_g) Strahl [12]. Die Winkel entsprechen der vorherigen Abbildung.

Um die Fresnel Gleichungen effektiv in der Computergrafik nutzen zu können, gibt es eine geeignete Näherung von Christophe Schlick. Die Ergebnisse dieser Näherung ergeben optisch genau den selben Eindruck wie die Fresnel Gleichungen, sind aber deutlich einfacher zu berechnen. Der

Koeffizient zur Bestimmung des Verhältnisses zwischen Brechung und Reflexion lautet nach Schlick folgendermaßen:

$$F = F_0 + (1 - F_0)(1 - H * V)^5 \tag{3.4}$$

mit

$$F_0 = (\frac{n_1 - 2}{n_1 - n_2})^2 \tag{3.5}$$

Für die Implementation entspricht V der Blickrichtung, also dem Abstand zwischen Kamera und Punkt auf dem Wasser, und H entspricht:

```
1 vec3 H  = normalize(eye + normal);
```

also der normalisierten Addition der Blickrichtung und der Normale dieses Punktes. Die Vektoren H und V werden mit dem Skalarprodukt verbunden.

Mit den Brechungsindices $n_1 \approx 1$ für Luft und $n_2 = 1,33$ für Wasser, ergibt sich für $F_0 = 0,02$. Mit diesen Informationen kann durch einfaches Anwenden der Formel der Fresnel Term näherungsweise berechnet werden [18].

4 Implementation

4.1 OpenGL Shader und GPU

Im Rahmen dieser Arbeit wurde ein Programm geschrieben, dass die theoretischen Aspekte der vorherigen Kapitel in OpenGL 3.3 umsetzt. OpenGL ist eine API (Application Programming Interface), die eine Abstraktionsschicht zwischen der Applikation und dem zugrundeliegenden grafischen System darstellt. Dadurch können leicht plattformunabhängige Programme geschrieben werden, ohne dass dem Entwickler genaue Einzelheiten der jeweiligen GPU (Graphic Processing Unit) bekannt sein müssen. OpenGL setzt zur Performancesteigerung auf ein Pipelinesystem. Dabei werden die Daten an die GPU gesendet, die dann die graphische Berechnung effizient vornehmen kann.

Aufgrund der Tatsache, dass eine GPU aus tausenden kleinen Kernen, so genannten *shader cores* besteht, können die ankommenden Daten hier parallel verarbeitet werden. Diese Parallelität ist der Grund, warum in der heutigen Grafikprogrammierung die Berechnungen weitgehend auf der GPU ausgeführt werden. Abgesehen davon, dass die CPU meist mit Hintergrundoperationen, wie beispielsweise der Allokierung von Speicherplatz beschäftigt ist, ist die CPU im Gegensatz zur GPU auch nicht auf Parallelität ausgelegt, sondern auf serielle Datenverarbeitung. Moderne OpenGL Programme machen sich diese Parallelität mit Hilfe von Shader Programmen zunutze. Jeder *shader core* der GPU kann einen Shader ausführen. Transformationen müssen beispielsweise für jeden Punkt einzeln durchgeführt werden, dabei sind diese jedoch nicht voneinander abhängig. Genauso wenig ist die Farbe eines Fragments von der eines anderen Fragments abhängig. Das bedeutet, dass die Bearbeitung aller Vertices oder Fragmente gleichzeitig, also parallel von jeweils einem *shader core* ausgeführt werden, was zu einer erheblichen Performancesteigerung führt.

Nicht alle Schritte in der OpenGL Pipeline sind programmierbar. Diese *fixed-function* Elemente werden von OpenGL zur Verfügung gestellt. Die Shader hingegen können in GLSL (OpenGL Shadering Language) programmiert werden. Dabei gibt es verschiedene Arten von Shadern. Für diese Implementation wurden nur Vertex und Fragment Shader und zu debugging Zwecken ein Geometry Shader verwendet.

Als Nachschlagewerk für die Implementation in OpenGL wurde die *OpenGL Super Bible* [19] verwendet.

4.2 Umgebung

Abgesehen von der Kamerasteuerung mit Maus und Tastatur um eine Bewegung in der Szene zu ermöglichen, wurde zu Beginn dieser Arbeit zuerst die Umgebung für die Wassersimulation erstellt.

Abbildung 14: Mit Terragen 2 erstellte Cube Map Textur. Die untere Hälfte der Textur wird von der Wasserfläche verdeckt, stellt also den Meeresboden dar.

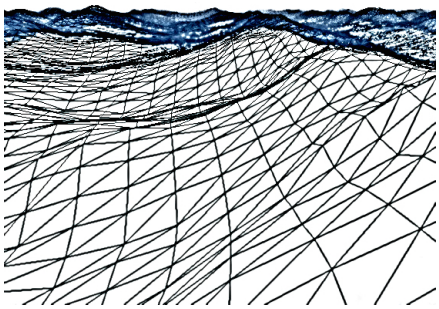

Abbildung 15: Das implementierte 2D Gitter zur Darstellung der Wasseroberfläche, mit einer darauf angewandten Höhenfunktion (hier: Gerstner-Wellen).

Zur Darstellung des Himmels im Hintergrund wurde, wie es üblich ist *Environment Mapping* verwendet. Dabei gibt es die beiden gängigen Varianten *Sphärisches Environment Mapping* und *Cube Mapping*. Da für diese Implementation *Cube Mapping* verwendet wurde, wird auch nur dieses im Folgenden erklärt. Eine *Cube Map* besteht aus sechs einzelnen Texturen (die zusammen als eine Textur behandelt werden) die so angelegt sind, dass sie nahtlos ineinander übergehen. Die für die Implementation verwendete Textur wurde mit Terragen 2 erstellte und wird in Abbildung 14 gezeigt. Die Textur kann so zu einem Würfel gefaltet werden, dass von innen der Eindruck eines Himmels entsteht.

Diese Texturen werden auf die sechs Seiten eines Würfels abgebildet. Dabei befindet sich der Ausgangspunkt im Zentrum; Die gesamte Szene ist also von diesem Würfel umgeben. Beim Erstellen solcher *Cube Maps* ist besonders darauf zu achten, dass nicht nur alle sechs Texturen quadratisch sind und die gleiche Größe haben, sondern, dass diese auch der *Power of Two* Regel entsprechen. Diese Forderung gilt für Texturen im Allgemeinen und besagt, dass deren Längen bzw. Breiten immer 2^x entsprechen müssen. Diese Regel muss zwar nicht eingehalten werden, gehört aber zum guten Programmierstil. Außerdem kann es vorkommen, dass Texturen ansonsten verzerrt dargestellt oder fehlende Pixel interpoliert werden, was zu Unschärfe führen kann.

Ein weiterer interessanter Punkt im Bezug auf *Cube Maps* sind die UV-Koordinaten. Für normale Texturen geht man von zweidimensionalen UV-Koordinaten aus. Möchte man diese Texturen in den Shadern verwenden, wird dazu der *Sampler2D* Datentyp benötigt. Für Cube Maps gibt es speziell den *SamplerCube* Datentyp. Dieser kann allerdings nur zusammen mit 3D Texturkoordinaten verwendet werden. Darum gibt es hier die Besonderheit, dass die R, S und T Koordinaten einer *Cube Map* einen Vektor vom

Zentrum des Würfels nach außen definieren. Der Schnittpunkt dieses Vektors mit einer der Würfelflächen entspricht der UV-Koordinate.

Praktisch an dieser Art der Berechnung ist, dass, wenn der Würfelmittelpunkt und der Mittelpunkt des Bezugssystems auf den selben Punkt fallen, die Positionsdaten des Würfels als UV-Koordinaten verwendet werden können [19].

Als Abschluss der Vorarbeiten wurde die Wasserebene erstellt. Verschiedenen Möglichkeiten, sowie deren Vor- und Nachteile werden im folgenden Abschnitt diskutiert.

Für die Implementationen in diesem und teilweise auch in den folgenden Abschnitten waren die OpenGL Tutorials von Anton Gerdelan [20] und Nvidia GPUGems hilfreich.

4.3 Das Wassergitter

Jetzt haben wir einen Himmel. Aber was ist mit dem Wasser? Wie können wir dieses visualisieren?

Üblich ist es, Wasser entweder als Gitter oder durch ein Partikelsystem zu simulieren. Der entscheidende Nachteil von Gittern im Allgemeinen ist, dass für jeden x-, z-Wert immer nur genau ein Höhenwert existieren kann. Daher ist es mit dieser Methode nicht möglich, brechende Wellen zu simulieren. Möchte man beispielsweise das Brechen von Wellen beim Auflaufen auf den Strand simulieren, dann wäre an dieser Stelle ein so genanntes Partikelsystem zu favorisieren.

Dieser Ansatz kommt der Vorstellung, dass Wasser aus winzig kleinen Molekülen besteht sehr nahe und ist daher intuitiver. Die Partikel stellen die Wassermoleküle dar, auf die Größen wie die Schwerkraft, Druck oder die Viskosität einer Flüssigkeit wirken. Unter den Einschränkungen dieser Größen kann sich ein Partikel ansonsten frei im Raum bewegen. Bei geeigneter Wahl dieser Größen entsteht unter Anwendung der richtigen Formel das Verhalten von Wasser.

Möchte man Wasser in einer einschränkenden Umgebung, z.B. in einem Glas oder einer Wanne simulieren und die Interaktion des Wassers mit Gegenständen oder seine Reaktion auf Bewegung darstellen, so ist die Partikelmethode zusammen mit den Navier-Stokes-Gleichungen sicherlich eine geeignete Wahl.

Das Problem bei dieser Methode ist, dass für jeden Partikel nicht nur die Wirkung sämtlicher physikalischer Kräfte berechnet werden muss, sondern auch seine Wechselwirkung mit den Partikeln in seiner Nachbarschaft. Für einen großen Wasserbereich wie in einem Meer, wäre der Rechenaufwand dementsprechend zu hoch.

Außerdem ist ein Partikelsystem zwar für die Simulation der Wellen geeignet, da man aber die einzelnen Partikel sieht und nicht die Wasseroberfläche wie sie sein soll, müssten für die Visualisierung der optischen

Aspekte noch zusätzliche Maßnahmen ergriffen werden.

Für eine Simulation von großen Wasseroberflächen unter Vernachlässigung des Brechens von Wellen ist ein Gitter also die bevorzugte Wahl, weil sich hier der Rechenaufwand gering halten lässt. Da der Schwerpunkt dieser Arbeit nicht nur die Simulation sondern auch die Visualisierung ist, wurde hier auch eben diese Methode angewandt.

Es besteht weiterhin die Möglichkeit 2D Gitter oder 3D Gitter zu erstellen. Der Vorteil von 3D Gittern ist, dass für jeden Punkt im Wasser alle physikalischen Kräfte, die auf diesen wirken, ausgewertet werden können. Das führt natürlich zu hoher Genauigkeit und zu physikalischer Korrektheit. Allerdings ist diese Methode auch sehr rechenaufwendig. Für kleinere Wasserbereiche ist ein solches Vorgehen sinnvoll; um ein gesamtes Meer zu simulieren ist der Rechenaufwand ebenfalls zu hoch.

Da sich die meisten Punkte unter der Oberfläche befinden, ist ein solcher Aufwand oftmals auch nicht nötig. Vor allem in Computerspielen, wo Performance eine zentrale Rolle spielt, werden daher oft 2D Gitter verwendet, die nur die Oberfläche darstellen.

Im Endeffekt fiel die Entscheidung also auf ein 2D Gitter, wie es im Screenshot der Implementation (Abbildung 15) zu sehen ist. Dazu wird bei einer Gitterbreite und -länge von N über zwei verschachtelte For-Schleifen jeder der NxN Gitterpunkte erstellt. Mit Hilfe einer Indexliste, werden diese auf die GPU hochgeladen.

4.4 Water Vertex Shader

Vertex Shader können auf jedem Vertex Operationen ausführen, sie also beispielsweise transformieren. Dabei ist es in Vertex Shadern allerdings nicht möglich auf benachbarte Vertices zuzugreifen, Vertices zu zerstören und neu zu erstellen. Vertex Shader bekommen als Input sowohl die Daten der auf Applikationsseite angelegten Vertex Buffer Objects (VBOs), sowie uniform Variablen, die ebenfalls von der Applikation übergeben werden. Die Ausgabe der Vertex Shader wird, falls Tessellation oder Geometry Shader aktiv sind, an diese weitergegeben, dann rasterisiert und schließlich an den Fragment Shader gegeben.

In diesem Unterkapiteln wird die Implementation der Gerstner-Wellen, sowie einige Kleinigkeiten die zur Steigerung des Realismus dienen, erklärt. Im Wasser Vertex Shader werden zudem noch Sinuswellen erstellt. Da die Implementation dieser sich allerdings nur in der Formel und der Festlegung einiger Variablen von den Gerstner-Wellen unterscheidet, wird dieser Teil hier nicht näher erläutert.

Im Vertex Shader wird z.B. der Nebel berechnet und an den Fragment Shader übergeben. Die Berechnung des Nebels findet deshalb im Vertex Shader statt, weil eben diese Funktion auch dazu verwendet wird, die Wellen, die weit entfernt sind, immer weiter abzuflachen. Hier wird sich die

Tatsache, dass die menschliche Wahrnehmung in der Ferne abnimmt zunutze gemacht. Würde man darauf verzichten, wäre die Horizontlinie nicht mehr klar zu erkennen, sondern von Wellenbergen verdeckt, die ein Mensch auf diese Entfernung eigentlich gar nicht mehr wahrnehmen kann.

Um einen realistischen Eindruck in der Ferne zu erhalten wird im Vertex Shader folgende Zeile eingefügt:

```
1   float passFogFac = exp(-pow(0.1*length(newPos), 2.0));
```

Hier wird die Formel des *GL_EXP2* Modus verwendet, um einen Faktor zu finden mit dem im Folgenden die Höhenwerte und später im Fragment Shader die Farbwerte interpoliert werden. Anhand der Position und mit Hilfe eines Dichtefaktors wird ermittelt, wie stark der Anteil der Überlagerung durch den Nebel ist.

```
1   newPos.y = passFogFac * newPos.y + (1-passFogFac) * 0;
```

In dieser Zeile des Quelltextes wird der Höhenwert der neuen Position mit dem eben berechneten Fog Faktor so interpoliert, dass weit entfernte Höhen sich immer mehr der Null annähern. Dabei dient der zweite Summand hier nur zur Verdeutlichung der Interpolation, fällt aber weg, da er Null ergibt.

4.4.1 Gerstner-Wellen Implementation

Für die Implementation der Gerstner-Wellen werden folgende Variablen verwendet:

- $amplitude$: Wellenamplitude, die hier Werte zwischen 0,001 und 0,01 annehmen können,
- D: Wellenvektor, einem zweidimenisonalen Vektor dessen x und y Werte zwischen -10 und 10 liegen,
- k: Betrag (bzw. Länge) des Wellenvektors,
- ω: Dispersionsrelation, die ebenfalls anhand des Wellenvektors errechnet wird,
- ϕ: Gangunterschied, im Bereich von 0,1 bis 1,
- t: Zeit, die durch die periodische Inkrementation einer Zählervariable von der Applikation bereitgestellt wird.

Diese werden in den folgenden Zeilen Quellcode so umgesetzt, wie in Kapitel 2.4.2 beschrieben. Dabei entspricht die erste For-Schleife der Implementation der eigentlichen Wellen wie in Formel 2.28 beschrieben und in der zweiten Schleife findet die Normalenberechnung wie in Formel 2.32 statt.

```
for(int i = 0; i< 5; i++) {
  k = sqrt(D[i].x*D[i].x + D[i].y*D[i].y);
  w = sqrt(g* k);

  calc.x -= (D[i].x/float(k) * amplitude[i] *
             sin(dot(D[i], calc.xz) - w*t + phi[i]));
  calc.y += (amplitude[i] *
             cos(dot(D[i], calc.xz) - w*t + phi[i]));
  calc.z -= (D[i].y/float(k) * amplitude[i] *
             sin(dot(D[i], calc.xz) - w*t + phi[i]));
}
newPos += calc;

for(int i = 0; i< 5; i++) {
  k = sqrt(D[i].x*D[i].x + D[i].y*D[i].y);
  w = sqrt(g* k);

  newNormal.x += (D[i].x * amplitude[i] *
                  sin(dot(D[i], calc.xz) - w*t + phi[i]));
  newNormal.y -= (D[i].x*D[i].x/float(k) * amplitude[i] *
                  cos(dot(D[i], calc.xz) - w*t + phi[i])
                +D[i].y*D[i].y/float(k) * amplitude[i] *
                  cos(dot(D[i], calc.xz) - w*t + phi[i]));
  newNormal.z += (D[i].y * amplitude[i] *
                  sin(dot(D[i], calc.xz) - w*t + phi[i]));
}
```

Da es mitunter recht schwierig sein kann durch bloßes Hinsehen einzuschätzen, ob die Reflexionen auf der Wasseroberfläche korrekt sind, wurde zusätzlich ein Geometry Shader verwendet, der per Tasterturabfrage zugeschaltet werden kann. Das Ergebnis dieses Geometry Shader ist in Abbildung 16 zu sehen. Die Normalen werden hier in grün und rot dargestellt, wobei die grüne Seite die Richtung, in die die Normalenvektoren zeigen, markiert.

Hierfür ist es nötig das Wasserobjekt ein zweites Mal zu rendern. Bei diesem zweiten Renderschritt werden dann zusätzliche Vertex-, Geometry- und Fragment Shader verwendet. Das ist zwar ineffizient, da alle Berechnungen doppelt gemacht werden müssen, da diese Option aber nur zu debugging Zwecken dient, stellt das kein Problem dar. Der hier verwendete Vertex Shader ist eine simple Kopie des Wasser Vertex Shaders, bis auf die Tatsache, dass er außer der Position und der Normale keinen Output hat. Außerdem werden weder Normale noch Position irgendeiner Transformation unterzogen. Diese findet im Geometry Shader statt. An dieser Stelle ist es wichtig, dass die Transformation hier mit der vollständigen Model-ViewProjection Matrix vorgenommen wird und nicht nur, wie es im Water Vertex Shader der Fall ist, zur weiteren Berechnung ins Kamerakoordinatensystem transformiert wird.

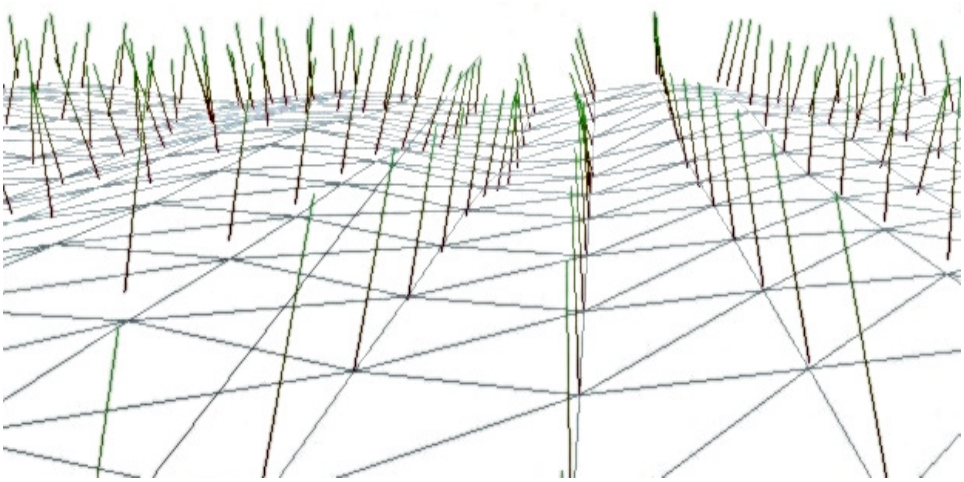

Abbildung 16: Darstellung der Normalen mit Hilfe eines Geometry Shaders.

Die restliche Berechnung im Geometry Shader besteht im Grunde nur daraus, aus den Inputvariablen der Normale und Position die Start und Endwerte der Linie zu berechnen, die als Normale gezeichnet werden soll. Der Startwert entspricht einfach der Position und der Endwert ist die Postion plus Normale.

Wie in Abbildung 16 deutlich zu erkennen ist, bewegen sich die Normalen der einzelnen Vertices in die Richtung, die man erwarten würde. So zeigen Normalen im Wellental oder auf einem Wellenberg senkrecht nach oben, während Normalen an den Seiten der Wellen schief geneigt sind, sodass sie weiterhin senkrecht auf der nun schiefen Fläche stehen. Hier wurde extra eine geringe Auflösung des Gitters gewählt, damit die einzelnen Normen klar zu erkennen sind.

4.4.2 Rauschtextur

Zusätzlich zu den Gerstner-Wellen werden die Höhenwerte in der fertigen Implementation auch noch von zwei verschiedenen verrauschten Texturen beeinflusst, die wie folgt implementiert sind:

```
1 mix(
2  texture(noise1Texture, uvCoord*vec2(d,d)).x /10,
3  texture(noise2Texture, uvCoord*vec2(d,d)).x /10,
4  float(sin(0.6*timer*pi) +1)/2
5 );
```

Die beiden Texturen werden so interpoliert (mit Hilfe die Methode *mix()*), dass eine zeitabhängige Animation zwischen den beiden Texturen entsteht. Die Variable d beeinflusst dabei, in welcher Skalierung die Textur auf die Gitterpunkte gemappt wird und stellt somit die Detailgenauigkeit dar. Durch

die Anwendung des Sinus auf den Timer, entwickelt eine schwingungsartige Animation. Erst in Kombination mit den Gerstner-Wellen führt dies zu dem gewünschten Ergebnis.

Es wird deshalb nur auf den x Wert der Texturen zugegriffen, weil diese dem Rot-Kanal entspricht. Da die Rauschtexturen als Grauwertbilder vorliegen haben sie nur einen Kanal. Die Grauwerte werden also als verschiedene Rottöne interpretiert, was jedoch keine Rolle spielt, da nur diese Werte ausgelesen und auf die Höhe angewendet werden, die Textur aber niemals farblich dargestellt wird.

4.5 FFT Implementation

Für die hier beschriebene Implementation wurden als Quellen das Tutorial *Ocean Simulation* von Keith Lantz [21], sowie die Masterarbeit von Sanjay Sen [22] verwendet. Da es nötig ist auf Werte des vorausgegangen Renderschrittes zurückzugreifen, um die Berechnung durchzuführen, wurde die FFT nicht im Vertex Shader implementiert, sondern auf Applikationseite.

Komplexe Zahlen werden nicht standardmäßig von C++ unterstützt, deshalb wurde eine Klasse Complex geschrieben, die simple Rechenoperationen wie Addition, Subtraktion, Multiplikation und Division auf komplexen Zahlen, sowie die Berechnung des komplex Konjugierten enthält.

Die erste Aufgabe ist es jetzt das Phillipsspektrum und zwei gaußsche Zufallszahlen zu berechnen und diese mit $\tilde{h}_0$ zu verknüpfen. Die komplexe gaußsche Zufallszahl wird mit dem Box-Müller Algorithmus erzeugt:

```
1  Complex Ocean::gauss() {
2      float u1 = float(rand()) / float(RAND_MAX);
3      float u2 = float(rand()) / float(RAND_MAX);
4
5      float v1 = float(rand()) /RAND_MAX;
6      float v2 = float(rand()) /RAND_MAX;
7
8      float firstRandom = sqrt(-2.0f * log(u1)) * cos(2* pi
           * u2);
9      float secondRandom = sqrt(-2.0f * log(v1)) * cos(2* pi
           * v2);
10
11     return Complex(firstRandom, secondRandom);
12 }
```

Das Phillipsspektrum sieht wie folgt aus:

```
1  float Ocean::phillips(int n, int m) {
2      //k
3      glm::vec2 k = glm::vec2(pi * (2 * n - N) / length, pi
           * (2 * m - N) / length);
4      float kLen  = sqrt(k.x*k.x + k.y*k.y);
5      if (kLen < 0.000001) return 0.0;
```

```
6       float kLen4 = kLen * kLen * kLen * kLen;
7
8       //w
9       float wLen = sqrt(w.x *w.x + w.y *w.y);
10
11      //k and w
12      float kDdotW = glm::dot(glm::normalize(k), glm::
            normalize(w));
13      float kDotW6 = kDdotW * kDdotW * kDdotW * kDdotW *
            kDdotW * kDdotW;
14
15      //lambda
16      float L = wLen * wLen / g;
17
18      //suppression of too small waves
19      float l = 0.001 * 0.001 *L* L;
20
21      return A * exp(-1.0f / (kLen*kLen * L*L)) / kLen4 *
            kDotW6 * exp(- kLen*kLen * l);
22  }
```

Die Kombination erfolgt durch:

```
1   Complex Ocean::hTilde_0(int n, int m) {
2       Complex r = gauss();
3       return r * sqrt(phillips(n, m) / 2.0f);
4   }
```

Auffallend ist, dass bis zu diesem Schritt noch keine Abhängigkeit von der zeitlichen Komponente besteht. Im Klartext bedeutet das, dass die Aufrufe dieser Methoden aus dem Konstruktor der Wasserklasse heraus erfolgen können, da sie nur einmal zu Beginn berechnet werden müssen. Im Konstruktor wird das Gitter erstellt. Deshalb müssen jetzt auch für jeden Gitterpunkt folgende Werte berechnet werden: $\tilde{h_0}(\vec{k})$ und $\tilde{h_0}^*(\vec{k})$. Dazu wird die Methode *hTilde_0* aus dem Konstruktor heraus zweimal aufgerufen. Eines der Ergebnisse wird komplex konjugiert. Die Real- und Imaginäranteile der beiden Ergebnisse werden in einen Pointer vom Typ *glm::vec4* gespeichert.

Zur Laufzeit werden diese Ergebnisse nun benötigt um zusammen mit der zeitabhängigen Dispersion ω die komplexe Zahl $\tilde{h}(\vec{k}, t)$ zu berechnen:

```
1   Complex Ocean::hTilde(float t, int n, int m) {
2       int i = m * (N+1) + n;
3
4       //take values calculated in constructor
5       Complex htilde0(fftHandle[i].x,  fftHandle[i].y);
6       Complex htilde0conjunct(fftHandle[i].z, fftHandle[i].w
            );
7
8       //calculate dispersion
```

```
9     glm::vec2 k = glm::vec2(pi * (2 * n - N) / length, pi
          * (2 * m - N) / length);
10    float dispersion =  sqrt(g * sqrt(k.x*k.x + k.y*k.y));
11
12    //calulate complex values using euler's formula
13    Complex exp0(cos(dispersion * t),  sin(dispersion * t)
          );
14    Complex exp1(cos(dispersion * t), -sin(dispersion * t)
          );
15
16    return htilde0 * exp0 + htilde0conjunct*exp1;
17 }
```

Die bis hierhin berechneten Amplituden werden mit Hilfe der inversen Fourier Transformation wieder in den Zeitbereich zurück transformiert. Der transformierte Wert ist immer noch eine komplexe Zahl, deren Realteil die Höhe darstellt. Der Imaginärteil wird zur Berechnung der horizontalen Verschiebung der x, z-Werte verwendet, sowie zur Normalenberechnung:

```
1  glm::vec3 Ocean::calcFourier(glm::vec2 x, float t) {
2     //inti values
3     Complex h = Complex(0.0f, 0.0f);
4     glm::vec2 D = glm::vec2(0.0f, 0.0f);
5     glm::vec3 normal = glm::vec3(0.0f, 0.0f, 0.0f);
6     Complex ex, height;
7     float kx, kz, kLen, kDotX;
8     glm::vec2 k;
9
10    for (int m = 0; m < N; m++) {
11       kz = 2.0f * M_PI * (m - N / 2.0f) / length;
12       for (int n = 0; n < N; n++) {
13          kx = 2.0f * M_PI * (n - N / 2.0f) / length;
14
15          //k
16          k = glm::vec2(kx, kz);
17          kLen = sqrt(k.x*k.x + k.y*k.y);
18
19          //k and x
20          kDotX = glm::dot(k, x);
21
22          //calulate complex e^(k*x) with euler's formula
23          ex = Complex(cos(kDotX), sin(kDotX));
24
25          //combine random phillips amplitude with complex
                number e^(k*x)
26          height = hTilde(t, n, m) * ex;
27
28          //calculate height
29          h = h + height;
30
```

```
31          //calculate horizontal displacement in x and z
                direction using imaginary part
32          if (kLen > 0.000001)
33                  D = D + glm::vec2(kx / kLen * height.imag
                        , kz / kLen * height.imag);
34
35          //caluclate normal using imaginary part as well
36          normal = normal + glm::vec3(-kx * height.imag,
                0.0f, -kz * height.imag);
37        }
38    }
39    //calculate final normalized normal
40    normal = glm::normalize(glm::vec3(0.0, 1.0, 0.0) -
          normal);
41
42    //assign normal to pointer of normals for every point
43    normalHelper = normal;
44
45    //return complete displacement vector
46    return glm::vec3(D.x, h.real, D.y);
47  }
```

Die errechneten Normalen werden in der Pointerliste *normalHelper* gespeichert und die horizontale und vertikale Verschiebung wird als 3D Vektor zurückgegeben. Der letzte Schritt besteht nun darin, in einer doppelten For-Schleife über alle Punkte zu laufen und diesen, jeweils abhängig von der Position, die neuen Werte zuzuweisen:

```
1  newPosition = calcFourier(x, t);
2
3  verticesNew[i].x = verticesOld[i].x - newPosition.x;
4  verticesNew[i].y = newPosition.y;
5  verticesNew[i].z = verticesOld[i].z - newPosition.z;
6
7  normals[i] = normalHelper;
```

Zusätzlich müssen noch ein paar Randfälle abgefangen werden.

Mit dem Befehl

```
1  glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(glm::vec3) * (
       N+1) * (N+1), verticesNew);
```

können nun in jeder Renderschleife die neuen Postions- und Normalenwerte auf die GPU hochgeladen werden.

Die Ergebnisse der FFT Implementation werden in Kapitel 5 dargestellt. In selbigem Kapitel wird das Verfahren auch mit den Ergebnissen der Gerstner Implementation verglichen.

4.6 Water Fragment Shader

Der Fragment Shader wird für jedes Fragment bzw. Pixel, das durch die Rasterisierung entstanden ist, aufgerufen. Da gerade bei hohen Auflösungen deutlich mehr Pixel als Vertices existieren, wird der Fragment Shader öfter aufgerufen als der Vertex Shader. Auch dem Fragment Shader können uniform Variablen aus der Applikation übergeben werden. Diese Sorte Shader wird normalerweise zur Beleuchtung der Szene und zum setzen verschiedener Umgebungswerte, wie z.B. Nebel, verwendet. Die Ausgabe des Fragment Shaders ist die *fragmentColor*, die Farbe eines Pixels.

In diesem Projekt wurde ein Fragment Shader für das Wasser und einer für die Skybox verwendet, wobei der Shader für die Skybox hier nicht weiter erklärt wird, da seine einzige Funktion die Anwendung der *Cube Map* Textur ist.

Im Wasser Vertex Shader werden, abgesehen von der Phong Beleuchtung, auch die Reflexion und Brechung, die in Kapitel 3 beschrieben wurden, implementiert. Dazu wird der Vektor der aktuellen Blickrichtung an der Normalen jedes Punktes gespiegelt. Dies ist mit dem folgenden Kommando möglich.

```
1   vec3 reflected = reflect(eye, normal);
```

Der resultierende reflektierte Vektor stellt jetzt sozusagen die UV-Koordinate der *Cube Map* Textur dar, somit kann dies über den reflektierten Vektor mithilfe der Methode *texture()* auf die Wasseroberfläche projiziert werden.

Auf eben diese Weise kann auch die Brechung berechnet werden, indem man das Kommando *refract()* verwendet.

Um die finale Pixelfarbe zu erhalten wird jetzt zwischen der Phong Farbe, der Textur, der Reflexion und der Brechung derart interpoliert, dass ein realistischer Wassereindruck entsteht. Dabei wird das Verhältnis zwischen Reflexion und Brechung mit der in Kapitel 3.3 vorgestellten Näherung für den Fresnel Term bestimmt.

Wie bereits im vorherigen Kapitel erwähnt, wird auf die endgültige Pixelfarbe noch ein interpolierter Nebelwert aufaddiert, der die Farben im Hintergrund verblassen lässt und somit die Endlichkeit des Models verhüllt.

5 Ergebnisse und Auswertung

In diesem Kapitel wird der Vergleich zwischen den beiden Verfahren

- Gerstner-Wellen in Kombination mit einer Rauschtextur und
- Fourier Transformation mit Hilfe des Phillips Spektrums

gezogen.

Gerstner-Wellen bieten eine effiziente und recht einfache Möglichkeit Wasseroberflächen darzustellen. Für ein photorealistisches, physikalisch akkurates Modell reichen Gerstner Wellen jedoch nicht aus. Daher kommen sie beispielsweise in Computerspielen zum Einsatz, bei denen der Fokus auf Effizienz, statt auf physikalischer Genauigkeit liegt. Die FFT Methode von Tessendorf hingegen bietet physikalische Korrektheit und stellt durch das Phillipsspektrum eine Verbindung zu empirisch ermittelten, realen Messungen im Meer dar. Dadurch, dass diese Methode auf gaußschen Zufallszahlen basiert, besteht nicht die Gefahr, dass die Wellen zu regelmäßig erscheinen, wie es bei der Methode von Gerstner der Fall ist.

Abbildung 17 zeigt das Ergebnis der Kombination: Gerstner Wellen plus Rauschtextur. Hier wurden die Ergebnisse der Unterkapitel 4.4.2 und 4.4.1 kombiniert.

Wie gut erkennbar ist sind die Gerstner-Wellen viel zu glatt und unnatürlich. Um diese ebenmäßige Form ein wenig willkürlicher zu gestalten wurden verschiedene Rauschtexturen, wie auf der linken Seite gezeigt, auf den Höhenwert aufaddiert.

Im Vergleich dazu sieht man direkt darunter in Abbildung 18 das Gitternetz der Fourier Implementation. Es ist ein deutlich anderes Wellenmuster erkennbar. Die Regelmäßigkeit der Gerster-Wellen ist durch die Rauschtextur zwar nicht mehr so deutlich sichtbar, aber vor allem bei einer weit entfernten Kamerapostion doch zu sehen. Durch die Zufallswerte in Kombination mit dem Phillipsspektrum leidet die Implementation der Fourier Transformation nicht unter diesem Problem.

Bei den Gerstner-Wellen war es ohne Probleme möglich, ein Gitter mit 512x512 Punkten zu wählen. Dadurch, dass einzelne Berechnungen auf der GPU stattfinden können, erreicht man hier sehr gute Performancewerte. Für die Implementation der FFT war es gerade einmal möglich ein Gitter mit 16x16 Gitterpunkten zu wählen. Bereits bei 32x32 Punkten ruckeln die Wellenbewegungen schon sichtbar. Entsprechend dieser geringen Auflösung leidet die Optik natürlich. Die hier gezeigten Screenshots wurden mit einem 128x128 Gitter aufgenommen, allerdings muss man hier bereits Wartezeiten im Minutenbereich einkalkulieren. Das ist natürlich für ein Computerspiel untragbar. Daher gibt es Möglichkeiten die Fourier Transformation von Tessendorf auch auf der GPU auszuführen; Diese werden z.B. in

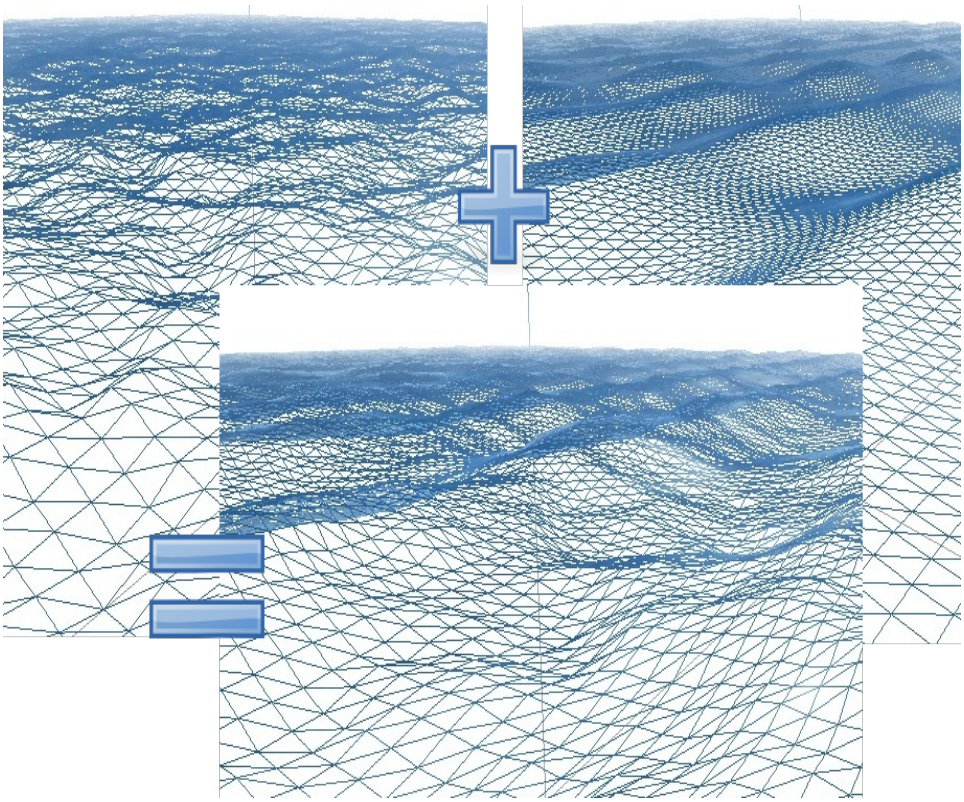

Abbildung 17: Rauschtextur + Gerstner-Wellen = realistisches Wellenbild.

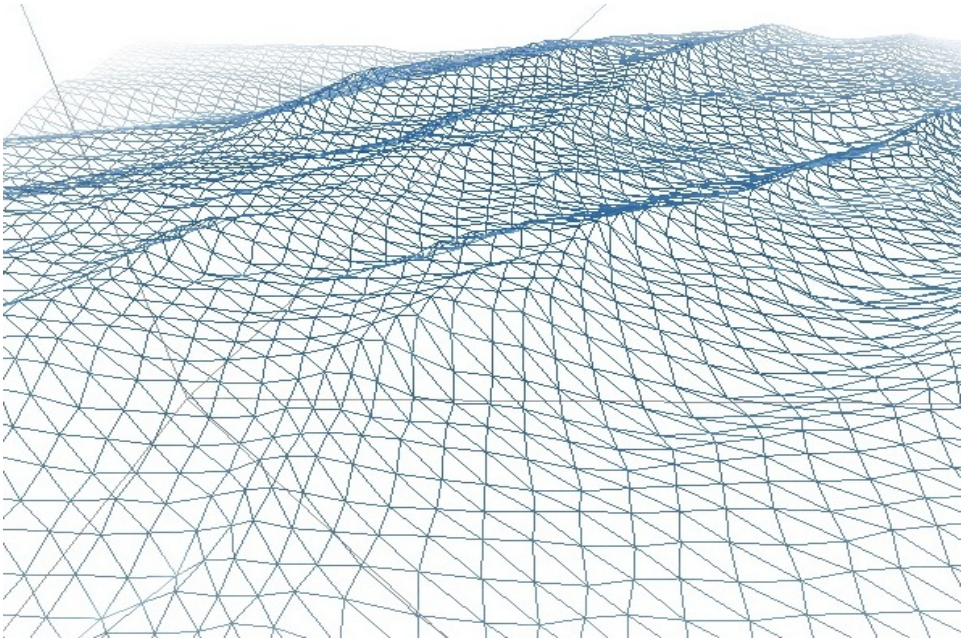

Abbildung 18: Ergebnis der Fourier Transformation

folgenden Artikeln und Internetseiten behandelt: [16], [23]. Da diese Verfahren sehr fortgeschrittene Techniken beinhalten und der Schwerpunkt dieser Arbeit auf der Implementation und dem Vergleich der Verfahren und nicht auf einer Simulation in Echtzeit liegt, wurden diese nicht implementiert.

Nachdem der Animationsteil damit abgedeckt ist werden im Folgenden Bilder der verschiedenen optischen Aspekte der Implementation gezeigt.

Die Abbildungen 19 und 20 zeigen Reflexion und Brechung. Die Abbildung der Brechung zeigt die charakteristische Verzerrung an den Wellenkämmen. Was man sieht ist eine durch Brechung verzerrte Bodentextur. Die Reflexion des Himmels stellt das Wasser bereits realistisch dar.

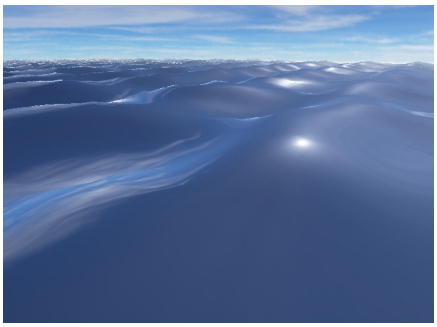

Abbildung 19: Die Implementation der Gerstner-Wellen nur mit den Reflexionen des Himmels.

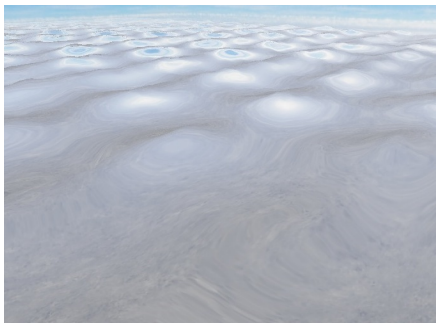

Abbildung 20: Die Implementation der Gerstner-Wellen nur mit Brechung.

Jetzt werden diese noch mit dem Fresnel Term kombiniert und eine Phong Beleuchtung hinzugefügt. Die Ergebnisse mit sowohl Brechung, Reflexion und Phong Beleuchtung für die Gerstner und Fourier Wellen zeigen Abbildungen 21 und 22.

Die beiden Bilder 23 und 24 zeigen die Verfahen mit Reflexion, Brechung, Phong Beleuchtung und einer Wassertextur.

Jetzt sind optische Eindrücke natürlich Geschmacksache. Abbildung 21 wirkt noch plastisch, wohingegen die Fourier Wellen in 22, mit den selben optischen Eigenschaften das beste Ergebnis, mit sehr realistischer Wirkung, erzielen. Bei den Fourier Wellen verbessert die Textur die Wirkung kaum, bei den Gerstner-Wellen hingegen enorm.

Abbildung 21: Gerstner-Wellen mit Reflexion, Brechung und Phong Beleuchtung.

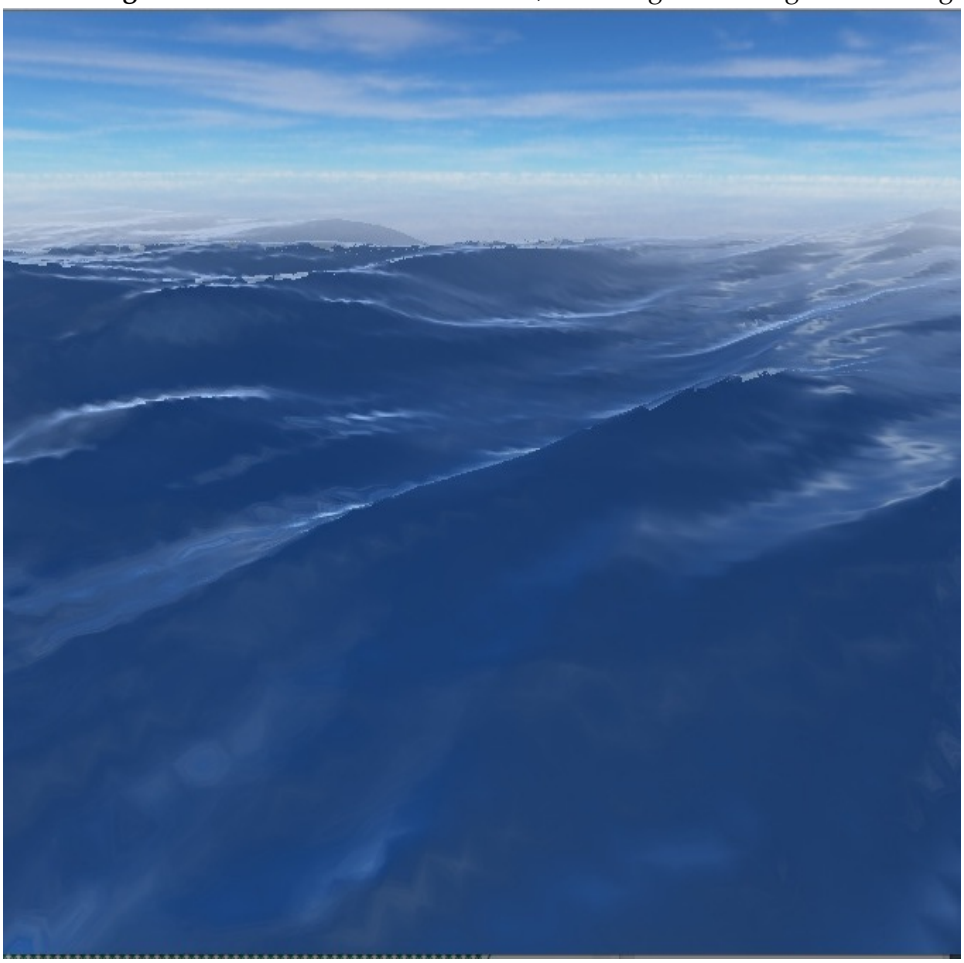

Abbildung 22: Fourier Wellen mit Reflexion, Brechung und Phong Beleuchtung.

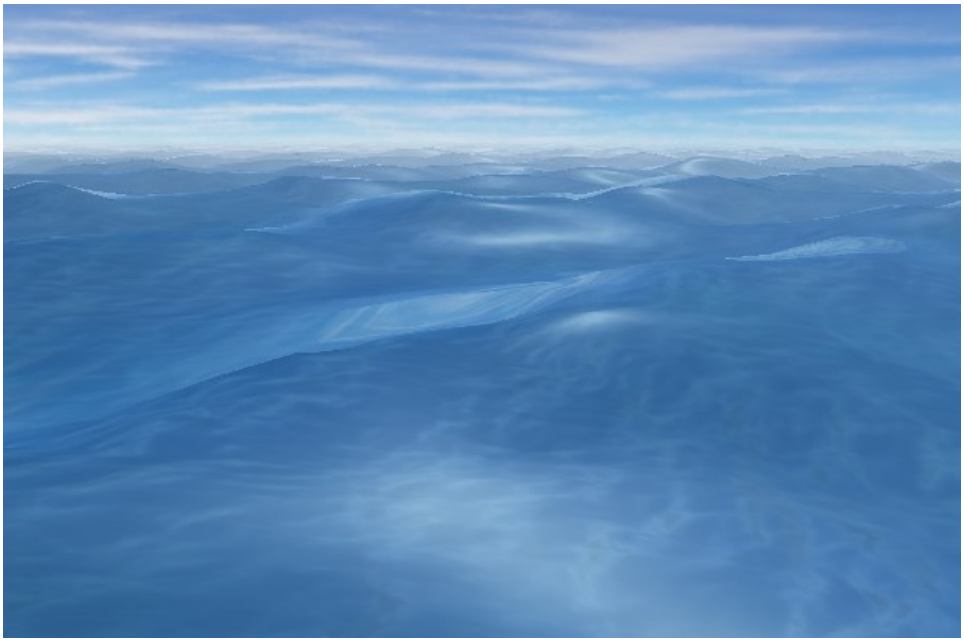

Abbildung 23: Gerstner-Wellen mit Reflexion, Brechung, Phong Beleuchtung und einer Wassertextur.

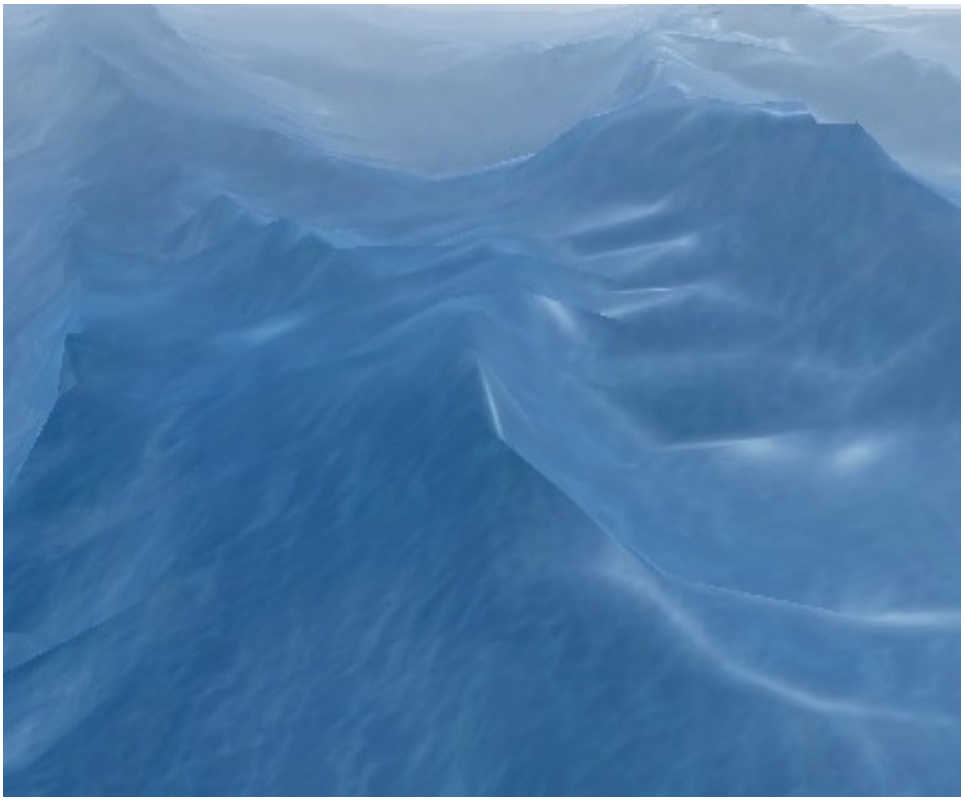

Abbildung 24: Fourier Wellen mit Reflexion, Brechung, Phong Beleuchtung und einer Wassertextur.

6 Fazit und Ausblick

Das Ziel dieser Arbeit ist die Animation und Visualisierung von Wasseroberflächen. Im Bereich der Animation wurden dazu zwei Verfahren zur Darstellung von Wellen mathematisch beschrieben und implementiert. Zum einen das Verfahren von Gerstner, welches auf der Kombination von Sinuswellen beruht und mit einer Rauschtextur verbunden wurde. Zum anderen die Fast Fourier Transformation mit Hilfe von Zufallszahlen und dem Phillipsspektrum zur Generierung der Amplituden. Im vorausgehenden Kapitel wurden diese beiden Verfahren verglichen. Im Bereich der Visualisierung wurden Reflexion und Brechung, sowie deren Verhältnis zueinander betrachtet und implementiert.

Natürlich gibt es noch zahllose Möglichkeiten die Implementation zu verbessern. Abgesehen von einer performancegünstigeren Implementation der FFT gibt es aber auch noch kleine Verbesserungen die zur Performancesteigerungen beitragen könnten. Zum Beispiel besteht die Möglichkeit das Gitter so anzulegen, dass nahe dem Betrachter die Gitterpunkte dicht zusammen liegen und in weiterer Entfernung nur noch im Verhältnis wenige Punkte existieren, da in großen Entfernungen die Details nicht mehr erkannt werden können. Dieser, dem Level of Detail (LOD) Schema folgenden Ansatz, ist allerdings nur dann sinnvoll, wenn sich eine performancegünstige Lösung findet das Gitter an Veränderungen der Kameraposition zur Laufzeit anzupassen.

Außerdem wurden im Bereich „Optik“ dieser Arbeit keine Mehrfachreflexionen behandelt. Wenn also ein reflektierter Strahl von einem Punkt im Wellental, bevor er auf die Himmeltextur trifft eigentlich noch eine andere Welle (z.B. einen sehr hohen Wellenberg) schneiden würde, so wird das hier nicht berücksichtigt. Dem Betrachter würde eine solche falsche Reflexion allerdings erst dann auffallen, wenn andere Objekte betroffen wären. Triebe zum Beispiel ein Boot auf dem Wasser, würde dieses ebenfalls nicht reflektiert werden. Solche Reflexionen sind schwierig und sprengen den Rahmen dieser Arbeit, da sie nicht mehr mit dem in OpenGL möglichen Texturzugriff zu bewältigen sind. Hier müssten entweder Raytracing, oder wenn man Wert auf Echtzeit legt, andere physikalisch zwar nicht korrekt, aber gute Ergebnisse liefernde Algorithmen angewandt werden.

Abgesehen von großen Erweiterungen, wie der Kollisionserkennung mit z.B. Schiffen, die auf dem Wasser treiben, Felsen oder ganzen Küstenabschnitten, die natürlich auch noch möglich sind, allerdings fortgeschrittenere Verfahren erfordern würden, besteht auch noch die Möglichkeit einer weiteren kleinen Erweiterung: Die Generierung der böigen Wellen durch die horizontale Verschiebung, wie in Grafik 11 gezeigt, hat einen bis jetzt noch nicht erwähnten Nachteil. Betrachtet man die Abbildung, kann z.B.

an der Spitze des ganz linken Wellenberges erahnt werden, dass die Oberfläche sich hier mit sich selbst schneidet und dadurch die Unterseite der Fläche sichtbar werden lässt. Das liegt daran, dass die generierten Amplituden teilweise sehr groß werden können.

Allerdings kann dieser vermeintliche Nachteil auch ausgenutzt werden: Möchte man beispielsweise Schaumkronen erstellen, ist dies ein gutes Kriterium dafür, ob Partikel gesprüht werden sollen oder nicht. Aufgrund der Tatsache, dass die Funktion die die y-Werte auf die Höhe abbildet nicht mehr eindeutig ist, kann mit Hilfe der so genannten Jacobi Matrix getestet werden, ob eine Überschneidung der Fläche mit sich selbst stattfindet.

Ein Algorithmus für Schaumkronen würde sich also einen zufälligen Punkt auf der Wasseroberfläche suchen, mit der Jacobi Matrix testen ob eine Überschneidung stattgefunden hat und falls das der Fall ist, an dieser Stelle Partikel sprühen. Die Geschwindigkeit und Richtung dieser Partikel würden zuerst von der Jacobi Matrix und der Oberflächennormale abhängen, später dann immer stärker von der Gravitation und dem Wind beeinflusst werden.

Literatur

[1] ZIRKER, Jack B.: *The Science of Ocean Waves: Ripples, Tsunamis, and Stormy Seas*. JHU Press, 2013

[2] CRAIK, Alex D.: The origins of water wave theory. In: *Annu. Rev. Fluid Mech.* 36 (2004), S. 1–28

[3] TESSENDORF, Jerry u. a.: Simulating ocean water. In: *Simulating Nature: Realistic and Interactive Techniques. SIGGRAPH* 1 (2001)

[4] STUHLMEIER, Raphael: On edge waves in stratified water along a sloping beach. In: *Journal of Nonlinear Mathematical Physics* 18 (2011), Nr. 01, S. 127–137

[5] WHITHAM, G. B.: *Lectures on wave propagation*. Bd. 61. Tata Institute of Fundamental Research Bombay, 1979

[6] LEE, Richard ; O'SULLIVAN, Carol: A Fast and Compact Solver for the Shallow Water Equations. In: *VRIPHYS*, 2007, S. 51–57

[7] DIGITALRUNE-TEAM: *Water Rendering Resources*. March 2014. – Ressource: `http://www.digitalrune.com/Support/Blog/tabid/719/EntryId/208/Water-Rendering-Resources.aspx`; visited at July 1th 2014

[8] GREHN, Joachim ; KRAUSE, Joachim: *Metzler Physik*. Bildungshaus Schulbuchverlag, 2008

[9] VUNDI, David: *General Wave Properties*. – Ressource: `http://ap-physics.david-s.org/general-wave-properties/`; visited at April 30th 2014

[10] HABIB: *Water Mathematics*. April 2014. – Ressource: `http://habibs.wordpress.com/water-mathematics/`; visited at Mai 15th 2014

[11] BRIDGE, John ; DEMICCO, Robert: *Earth Surface Processes, Landforms and Sedimant Deposits*. 2008

[12] MESCHEDE, Dieter: *Gerthsen Physik*. 21. Springer Verlag, 2002. – ISBN 3–540–42024–X

[13] FINCH, Mark ; WORLDS, Cyan: *Effective Water Simulation from Physical Models*. – Ressource: `http://http.developer.nvidia.com/GPUGems/gpugems_ch01.html`; visited at Mai 20th 2014

[14] FOURNIER, Alain ; REEVES, William T.: A simple model of ocean waves. In: *ACM Siggraph Computer Graphics* Bd. 20 ACM, 1986, S. 75–84

[15] WEISSTEIN, Eric W.: *Trochoid*. – From MathWorld - A Wolfram Web Ressource. http://mathworld.wolfram.com/Trochoid.html; visited at April 30th 2014

[16] LIU, Shiqiu: *Simulating Ocean Waves with FFT on GPU*. November 2011. – Ressource: http://www.edxgraphics.com/blog/simulating-ocean-waves-with-fft-on-gpu; visited at June 20th 2014

[17] WILEY-INFORMATION-SERVICES-GMBH: *Phänomen: Polarisation durch Reflexion, Polarisation von Licht*

[18] OLANO, Marc: *Graphics Trick: Schlick Approximation*. September 2010

[19] SELLERS, Graham ; WRIGHT, Richard ; HAMMEL, Nicholas: *OpenGL Super Bible*. Sixth Edition. Addison-Wesley, 2014. – ISBN 978–0–321–90294–8

[20] GERDELAN, Anton: *OpenGL 4 Tutorials*. March 2014

[21] LANTZ, Keith: *Ocean Simulation Part Two: Using the Fast Fourier Transformation*. November 2011

[22] SEN, Sanjay: Methods of Deep Ocean Simulation. (2013)

[23] CRAITOIU, Sergiu: *Compute Shader FFT of size 32*. November 2013. – Ressource: http://in2gpu.com/2013/11/01/compute-shader-fft-of-size-32/; visited at July 1th 2014