Physikalisch korrektes Rendern von Wasser als Image-Effekt

Eidesstattliche Erklärung

Hiermit versichere ich, Jannik Sonntag, dass die vorliegende Bachelorarbeit von mir selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

_____

Jannik Sonntag Köln, den 31. Juli 2015

Agenda

1 Einleitung

1.1 Begründung der Themenwahl

Gerade in letzter Zeit ist die Performance von PCs, vor allem im Bereich der Grafik, stark angestiegen, wodurch bei der Visualisierung in Computerspielen mehr Leistung genutzt werden kann. Aktuelle Hardware ist dabei optimiert auf die grafische Darstellung von Modellen als Ansammlung mehrerer Dreiecke. Mehr Leistung in der Hardware bedeutet also vor allem mehr Dreiecke, was wiederum zu einem erhöhten Detailgrad der dargestellten Modelle führt.

Gerade flüssige Oberflächen, wie zum Beispiel Wasser, leiden aber unter diesem Verfahren. Wasser besitzt an keiner Stelle eine wirklich gerade Oberfläche, was dazu führt, dass Dreiecke nur eine vage Annäherung an die Oberfläche des Wassers darstellen können. Solange man nicht eine übermäßig große Anzahl an Dreiecken verwendet, sieht die Oberfläche des Wassers unnatürlich aus und man erkennt direkt die Illusion.

Daher beschäftigt sich diese Arbeit mit einem alternativen Ansatz für die Visualisierung von Wasser, welcher komplett auf Polygone verzichtet und das Wasser als Image-Effekt mithilfe von Raytracing darstellt.

1.2 These

Die Arbeit beschäftigt sich mit dem Thema „Physikalisch korrektes Rendern von Wasser als Image-Effekt“ und setzt dabei eine Implementation dieses Effektes um, um ihn mit den herkömmlichen Methoden zu Vergleichen. Dabei wird sowohl Wert gelegt auf messbare Faktoren, wie Performance, aber auch auf visuelle Qualität, welche durch eine Umfrage abgedeckt wird.

1.3 Zielsetzung

Ziel der Arbeit ist es, einen Image Effekt zu schaffen, welcher das physikalisch korrekte Verhalten von Wasser als Grundlage hat und genutzt werden kann, um große Wasserflächen, wie zum Beispiel einen Fluss oder einen Ozean, in einem Spiel zu visualisieren.

Der Vergleich zwischen dem Ergebnis und aktuell genutzten Verfahren zur Visualisierung von Wasser soll dann Vor- und Nachteile in Bereichen Performance, Qualität und Flexibilität feststellen, wofür eine Umfrage mithilfe der Live-Demo geplant ist. Zielgruppe der Umfrage sind Studenten des SAE Institutes Cologne und weitere Interessenten.

2 Grundlagen

2.1 Physikalische Grundlagen von Wasser

Für die physikalisch korrekte Darstellung von Wasser gibt es zwei wichtige Eigenschaften. Zum einen trägt die Bewegung der Flüssigkeit unter Einfluss von Kräften wie Wind und Gefälle maßgeblich zur Darstellung bei. Da Wasser sehr flüssig ist, lässt es sich sehr leicht von Wind beeinflussen, weswegen schon bei kleinen Windeinflüssen auf offener See Wellen entstehen. Diese Wellen sind sinusförmig und laufen parallel in eine Richtung. Die einzelnen Wassermoleküle bewegen sich dabei hauptsächlich hoch und runter und nur leicht in Windrichtung (vgl. Florida Center for Instructional Technology 05). Diese Erkenntnis ist wichtig, um zu verstehen, dass sich auf der Wasseroberfläche liegende Gegenstände wie Blätter nicht mit der gleichen Geschwindigkeit bewegen wie die Wellenkämme, sondern um ein vielfaches langsamer.

Zum anderen trägt das Zusammenspiel zwischen Wasser und Licht einen Großteil zum visuellen Erscheinungsbild bei. Ab einer gewissen Tiefe „verschluckt“ das Wasser Licht, allerdings ist dieser Effekt nicht bei jeder Lichtfarbe gleich, so wird zum Beispiel das rote Licht zuerst absorbiert und das blaue Licht zuletzt (vgl. Braun & Smirnov 93), weswegen man Wasser eine blaue Farbe zuschreibt. Hier ist daher zu beachten, dass sehr flaches Wasser noch eine weiße Farbe hat, und erst sehr tiefes Wasser den bekannten kräftigen Blauton vom Meer besitzt. Bei sehr großen Tiefen wird Wasser theoretisch schwarz, allerdings schwimmen in den oberen Schichten des Wassers immer einige lichtreflektierende Partikel, weswegen das Wasser immer mindestens einen leichten Blauton behält, unabhängig von der Tiefe.

2.2 Visualisierung in Computerspielen

Da die meisten Spiele versuchen, eine realistische Welt darzustellen, ist es auch häufig notwendig, Wasser innerhalb des Spieles darzustellen, da dieses Teil der Welt ist. Manchmal ist das Wasser auch ein Element, welches von einem Spiel für die ein oder andere Mechanik genutzt wird (wie zum Beispiel, um Feuer zu löschen, oder generell bei Spielen mit Schiffen) und somit ist es meistens unumgänglich, sich darum zu kümmern, innerhalb des Spieles Wasser darzustellen. Spiele wie etwa „Assasin's Creed IV: Black Flag“[1] oder „Warcraft III: The Frozen Throne“[2] besitzen Seeschlachten als Teil des Spieles und sind somit auf Wasser angewiesen.

Da Wasser aber an sich im Vergleich zu anderen Materialien sehr komplexe Eigenschaften aufweist, ist es meistens sehr aufwendig, Wasser detailgetreu darzustellen. Um die Performance des Spieles nicht allzu sehr zu beeinflussen, wird bei der Darstellung von Wasser deswegen auf viele Tricks zurückgegriffen, um die Darstellung realistisch zu halten, aber die Performance dennoch in akzeptablem Rahmen zu halten.

1 Black Flag: Ubisoft, 2013, Action-Adventure, ca. 11 Millionen Exemplare weltweit verkauft
2 Warcraft III: Blizzard Entertainment, 2003, RTS

2.3 Traditionelle Wasservisualisierung

Normalerweise wird für die Visualisierung von Wasser eine sehr naheliegende Form der Visualisierung genutzt. Dabei wird das Wasser im Spiel im Grunde genommen dargestellt als eine große blaue Fläche, die horizontal in der Welt liegt und Wasser darstellt. Eine einfache blaue Fläche ist dabei natürlich nicht sehr überzeugend, weswegen weitere Tricks und Effekte genutzt werden, um den Eindruck von Wasser besser zu vermitteln. Jeder einzelne dieser Effekte verbessert das Aussehen von dem Wasser und in Verbindung ergeben alle Effekte zusammen eine sehr realistische Darstellung von Wasser. Diese weisen zwar meistens einige physikalische Ungenauigkeiten auf, aber diese fallen dem Nutzer nur bei sehr genauem Hinsehen und physikalischen Kenntnissen von Wasser auf.

Folgende Effekte werden häufig verwendet, um die Anschauung des Wassers zu verbessern:

Transparenz: Wasser ist eine durchsichtige Flüssigkeit und sollte dementsprechend in Spielen auch genau so dargestellt werden. Die Ebene, die das Wasser darstellt, ist deswegen nicht direkt blau, sondern blau und leicht durchsichtig, was den Effekt hat, das man zum Beispiel den Grund eines Baches noch sehen kann, trotz des Wassers, welches im Bach fließt. Dieser Effekt wird „Refraktion“ genannt (vgl. Elert 98), beinhaltet aber gleich noch einen weiteren Aspekt. Wenn Wasser Wellen schlägt, sieht man den Grund des Wassers nicht klar und deutlich, sondern leicht verzehrt, da die Wellen die Brechung (daher „Refraktion“) des Lichtes beeinflussen und somit der Grund des Wassers teilweise verzerrt erscheint.

Textur: Die Oberfläche von Wasser ist natürlich nicht überall gleichmäßig blau. Deswegen wird häufig eine Textur verwendet, um in die Wasseroberfläche etwas Varianz hineinzubringen. Diese Textur enthält zum einen Fremdkörper im Wasser (also zum Beispiel schwimmende Blätter oder Äste) und zum anderen aber auch Schaumkronen und eine wellenartige Oberfläche. Diese Oberfläche ist zwar physikalisch inkorrekt, da Wasser selbst wirklich nur eine blaue Farbe besitzt, allerdings ist die Berechnung für die einzelnen Faktoren für die genaue Farbinformation viel zu aufwendig.

Reflexion: Die Wasseroberfläche reflektiert Licht und dieser Effekt ist sehr wichtig für die visuelle Erscheinung von Wasser. Der wichtige Effekt ist aber nicht, dass sich die gesamte Landschaft, wie zum Beispiel Berge in der Ferne oder überhängende Palmen, im Wasser spiegelt, sondern die Reflexion der Sonne. Aufgrund des Wellenwurfes und der ständigen Bewegung von Wasser ist es fast immer möglich, auf einer Wasseroberfläche die Reflexion der Sonne zu sehen, solange die Sonne am Himmel steht. Man sieht sie in Form von kleineren weiß blitzenden Punkten auf der Oberfläche des Wassers. Dieses Phänomen ermöglicht es dem Gehirn, die Dreidimensionalität von Wasser besser zu begreifen, wodurch es nicht mehr so flach erscheint, wie es eigentlich ist.

Displacement: Es ist möglich, die einzelnen Punkte, aus denen die Plane aufgebaut ist, mithilfe der Grafikkarte während dem laufenden Spiel anzupassen. Dadurch ist es möglich, dass sich die Wellenbewegung, welche sehr typisch für Wasser ist, auch in der Plane widerspiegelt, um so den Realismus des Effektes zu erhöhen. Dadurch entstehen vor allem Konturen, wenn das Wasser am Ufer brandet und man erkennt auch die Kontur der Wellen am Horizont. Dieser Effekt ist relativ günstig und wird auf neuerer Grafikhardware durch die Tesselation[3] (vgl. Bonaventura 11) nochmal um einiges verbessert, indem die Verschiebung der Punkte nur an den wichtigen Stellen vorgenommen wird, die für den Betrachter sichtbar sind, beziehungsweise sich in seiner Nähe befinden.

3 Tesselation: Verfahren zur automatischen Verbesserung von Details in der Nähe der Kamera

3 Methodik

3.1 Begründung der Methodenwahl

Kern dieser Arbeit ist der praktische Teil, die Erstellung eines Image-Effektes zur Visualisierung von Wasser. Dieser Image-Effekt arbeitet auf eine spezielle Art und Weise und weicht von der normalen Umsetzung von Wassereffekten ab. Dennoch ist es das Ziel dieses Image-Effektes, mit der traditionellen Umsetzung in den meisten Bereichen mithalten zu können und in einigen Bereichen sogar einen Vorteil gegenüber herkömmlichen Verfahren zu bieten.

Der Image-Effekt selbst wurde mithilfe von Visual Studio, einem c++ Compiler[4] von Microsoft, entwickelt. Dabei wird vor allem die DirectX11[5] Bibliothek verwendet, um mithilfe der Shadersprache HLSL[6] den letztendlichen Effekt umzusetzen.
Die Umsetzung der Arbeit erfolgt daher zum Großteil durch Schreiben von Shader- und c++ Quellcode. Zusätzlich dazu werden auch einige Textur-Ressourcen verwendet. Die Texturen für das Terrain wurden online von CG Textures (http://www.cgtextures.com) erworben, wohingegen die Textur für das Wasser prozedural generiert wurde.

Im Bereich des Image-Effektes sind für das Endergebnis vor allem zwei Aspekte wichtig, die Performance des Image-Effekts (was der Image-Effekt an Rechenzeit „kostet“) und das visuelle Ergebnis (was der Image-Effekt „leistet“). Damit dieser Image-Effekt in einer realen Softwarelösung eingesetzt wird, muss das Verhältnis zwischen Rechenzeit und visuellem Ergebnis besser sein als ein „Konkurrenz“-Image-Effekt.

4 Compiler: Tool zur Umwandlung von Quellcode in eigenstehende Programme
5 DirectX11: Programmier-Bibliothek für die Visualisierung von 3D-Objekten
6 HLSL: High Level Shader Language, spezifische Programmiersprache für 3D-Visualisierung

3.2 Performance Tests

Eine Grafikkarte ist ein sehr komplexes Stück Hardware und Performance lässt sich nicht direkt ohne genaueren Einblick in die unterliegenden Vorgänge bestimmen (vgl. Akenine-Möler, Haines & Hoffman 08).

Die Grafikkarte agiert in mehreren Schritten und jeder dieser Schritte kann das „Bottleneck“ für die Performance sein.

Der erste Schritt in der Pipeline ist der Vertexshader. Auf jeden einzelnen Punkt des Models werden Instruktionen angewendet, die im Vertexshader beschrieben sind (vgl. McShaffry & Graham 13). Je mehr Punkte ein Model[7] hat oder je mehr Instruktionen der Vertexshader hat, desto langsamer läuft das Programm.

Um zu überprüfen, ob das Bottleneck im Vertexshader liegt, kann man künstlich die Anzahl an Punkten erhöhen, und somit die Last des Vertexshaders erhöhen, ohne die anderen Teile des Image-Effektes zu beeinflussen. Ist der Vertexshader das Bottleneck, so wird sich die gesamte Frametime dadurch erhöhen, ansonsten wird die erhöhte Anzahl an Punkten die Frametime nicht verändern.

Der zweite Schritt ist die Rasterisierung. Hier werden die mithilfe der Punkte aufgespannten Dreiecke in eine zweidimensionale Darstellung auf dem Bildschirm umgewandelt, wobei für jeden einzelnen Pixel überprüft wird, ob er innerhalb des Models ist oder nicht (vgl. Akenine-Möller, Haines & Hoffman 08). Durch Vergrößerung des Models durch bewegen der Kamera lässt sich die Anzahl an zu rasterisierenden Pixeln verringern und somit kann überprüft werden, ob das Bottleneck im Rasterizer liegt.

7 Model: Dreidimensionale Darstellung eines Objektes durch mehrere einzelne Dreiecke

Der dritte Schritt ist der Pixelshader, welcher direkt nach der Rasterisierung auf jeden ausgewählten Pixel angewendet wird. Der Pixelshader wird auf jeden Pixel, der von dem aktuellen Objekt bedeckt wird, angewandt und bestimmt die letztendliche Farbe an dieser Stelle (vgl. McShaffry & Graham 13).

Die Performance des Pixelshaders hängt davon ab, wie viele Pixel vom Objekt bedeckt sind und wie viele Instruktionen der Pixelshader enthält. Um zu überprüfen, ob das Bottleneck im Pixelshader liegt, macht es keinen Sinn die Anzahl an bedeckten Pixeln zu ändern, da dies gleichzeitig die Performance der Rasterisierung ändert. Der einzig sinnvolle Schritt ist also die Erhöhung der Anzahl an Instruktionen. Die Erhöhung der Instruktionsanzahl verändert zwar den Image-Effekt komplett und liefert somit andere Ergebnisse, allerdings lässt sich aus diesen Ergebnissen zumindest feststellen, ob der Pixelshader das Bottleneck ist. Der in dieser Arbeit genutzte Effekt ermöglicht aber auch eine weitere Möglichkeit zur Überprüfung des Bottlenecks im Pixelshader. Ändert man den Winkel, in dem die Kamera steht, so bedeckt das Wasser mehr oder weniger Platz (je nachdem wo der Horizont sichtbar ist). Die Rasterisierung wird hierbei noch auf das gesamte Bild angewandt, aber der Pixelshader muss nur in den Bereichen, in denen Wasser ist, wirklich arbeiten, wohingegen er in den anderen Bereichen direkt erkennt, dass hier kein Wasser ist. Ändert der Kamerawinkel die Frametime, so ist also der Pixelshader das Bottleneck.

Dieser Schritt sollte außerdem vor der Überprüfung des zweiten Schrittes geschehen, da man dann einfach nur die Anzahl an geschadeten Pixeln ändern kann, um festzustellen, ob der Rasterizer das Problem ist.

Die übrigen Schritte (GeometryShader, HullShader und DomainShader) sind für die Bachelorarbeit irrelevant, da diese Stufen der Pipeline nicht genutzt werden.

Die Bestimmung des Bottlenecks ist nicht nur für die Optimierung des Image-Effektes nötig (man sollte immer zuerst da optimieren, wo das Bottleneck liegt), sondern auch für die Erfassung der Performance. Wenn ein Spiel zum Beispiel in dem Großteil seiner Shader den Vertexshader auslastet (zum Beispiel Minecraft[8], mit sehr einfachem Shading aber einer sehr großen Anzahl an sichtbaren Dreiecken), dann ist ein zusätzlicher Shader, der hauptsächlich den Pixelshader auslastet, sehr kostengünstig, da dieser sowieso den Großteil der Zeit nichts zu tun hat.

Ein Beispiel: Auch wenn der Image-Effekt 5 ms in Anspruch nehmen würde, kostet dies vielleicht keine 5 ms der Zeit für einen Frame[9], da der Pixelshader so oder so 10 ms lang pro Frame nichts zu tun hatte, in denen er jetzt das Wasser berechnet.

Der letzte Punkt ist die Menge an Speicher, die verbraucht wird. Da dieser Image-Effekt keine speziellen Rendertargets verwendet, ist unter diesem Punkt nur die Anzahl an benötigten Texturen relevant. Aber auch in diesem Unterbereich bezieht sich der Image-Effekt nur auf eine Noise-Textur[10] für das Wasser selbst, was nichts direkt zu tun hat mit der Verwendung eines auf Raytracing basierenden Verfahren. Aufgrund dessen wird dieser Teil in dem Vergleich zu anderen Verfahren weggelassen.

Zu bestimmen ist also zum einen die generelle Zeit, die benötigt wird, um einen einzelnen Frame darzustellen, und zum anderen, wo das genaue Bottleneck des Image-Effektes liegt.

Etwas komplexer gestaltet sich die Auswertung der visuellen Qualität des Image-Effektes. Da das visuelle Endergebnis gerade für den Kunden der späteren Software relevant ist, wird für dieses Gebiet eine Umfrage gewählt.

8 Minecraft: Mojang, 2011, Sandbox, ca. 18 Millionen Exemplare weltweit verkauft
9 Frame: Ein Bild, welches gerendert wird und danach auf dem Monitor angezeigt wird
10 Noise-Textur: Textur gefüllt mit Zufälligen Graustufen

3.3 Strukturierung der Umfrage

Um die Ergebnisse der Umfrage einfacher auswerten zu können, wird die Umfrage in zwei Teile gegliedert. Im ersten Teil werden Fragen gestellt, um sich einen generellen Gesamteindruck über die Qualität der Arbeit zu verschaffen, wohingegen im zweiten Teil die eigene Arbeit mit einer bereits bestehenden Arbeit aus der Industrie verglichen wird.

3.3.1 Vergleichbarkeit

Als Arbeit aus der Industrie wurde die „Nvidia Island Demo“[11] gewählt, eine Demonstration der mit DirectX 11 neu eingeführten Technologie, „Tesselation“. Diese Arbeit wurde anhand mehrerer Kriterien ausgewählt:

High-End: Nvidia ist Grafikkarten- und Treiberhersteller und publiziert viele Tech-Demos, um die Features der neuen Grafikkarten zu präsentieren. Diese Demos spiegeln daher immer den aktuellsten Stand der Technik wider und sind qualitativ auf einem sehr hohen Niveau umgesetzt.

Vergleichbarkeit: Offensichtlich handelt es sich bei der Demo um Wasser. Der viel interessantere Punkt ist aber die Nutzung von Tesselation. Tesselation erhöht die Qualität der Wasseroberfläche, je näher sie an dem Betrachter liegt und erzeugt damit einen Kompromiss zwischen Qualität und Performance. Da die Arbeit der Bachelorarbeit das Wasser für jeden Pixel des Bildschirmes einzeln berechnet, ist die Qualität in der Nähe des Betrachters auch besser als in der Ferne (mehr sichtbare Pixel/m² Wasseroberfläche)

11 Nvidia Island Demo: Nvidia, April 2010, Tech-Demo

Verfügbarkeit: Um die Performance der einzelnen Arbeiten vergleichen zu können (siehe vorangehender Abschnitt), ist es nötig, einige Änderungen an dem Projekt zu machen, um die jeweiligen Metriken bestimmen zu können. Da die Nvidia Demo online verfügbar ist als uncompiliertes[12] Projekt, lassen sich beliebige Änderungen an der Quelle vornehmen, um die Metriken zu berechnen.

3.3.2 Zielgruppe

Das Thema der Umfrage ist sehr spezifisch, da die Visualisierung von Wasser hauptsächlich in Spielen ein wichtiges Thema ist. Allerdings geht es bei der Bewertung um die visuelle Qualität und den Realismusgrad. Diese Faktoren kann theoretisch jeder aus eigener Erfahrung bewerten.

Die Zielgruppe ist also an sich nicht so spezifisch wie das Thema und die Meinung jeder Person ist für die Auswertung relevant. Dennoch wird der erste Teil der Umfrage sich mit der Segmentierung der Ergebnisse beschäftigen und Fragen stellen zur Kategorisierung des Teilnehmers in verschiedene Gruppen, um ein detaillierteres Ergebnis zu liefern. Dadurch wird die Zielgruppe im Endeffekt in zwei große Gruppen eingeteilt: Diejenigen, die viel Zeit am Computer verbringen und bei der praktischen Demo der Arbeit Referenzen aus Spielen und eigenen Projekten haben und denjenigen, die eher wenig Zeit am Computer verbringen und als Referenz für die praktische Demo reales Wasser heranziehen.

12 Uncompiliert: Der originale Quellcode ist vorhanden und somit modifizierbar

3.3.3 Inhalt

Der erste Teil der Umfrage dient zur Unterteilung in die beiden oben beschriebenen Zielgruppen. Zur Einordnung ist zum einen die aktuelle Anstellung relevant, beziehungsweise, ob diese im Bereich der Gamesbranche liegt und zum anderen die Anzahl an Stunden, die pro Woche am Computer verbracht wird, um Computerspiele zu spielen.

Darüber hinaus erfolgt noch eine Einordnung in Altersgruppen und Geschlecht, um eventuell die Zielgruppen weiter einzugrenzen und spezifischer auswerten zu können.

Da der praktische Teil dieser Arbeit ein nicht gängiges Verfahren zur Umsetzung von Wasser verwendet, sind die Fragen der Umfrage hauptsächlich auf die Bereiche gerichtet, die von der Nutzung dieses nicht gängigen Verfahrens beeinflusst werden.

Folgende Fragen wurden im Rahmen der Umfrage gestellt:

Optischer Gesamteindruck / Gesamteindruck: Wie realistisch sieht das Wasser aus?

Zu Beginn der Umfrage wird zuerst der generelle Gesamteindruck abgefragt, bevor spezifischer auf die einzelnen Teilbereiche des Wassers eingegangen wird, um einen einfachen Vergleich zwischen den beiden Arbeiten vorab zu ermöglichen.

Oberfläche: Wie realistisch ist die Wellenbewegung des Wassers? / Oberfläche: Wie realistisch ist das Zusammenspiel zwischen Wasseroberfläche und Umgebung?

Der Hauptaspekt ist, dass die Wellen nicht direkt durch Geometrie repräsentiert werden, sondern nur durch Raytracing angenähert werden. Eine sehr wichtige Frage ist daher, wie die Form der Wellen in der Demo auf den Nutzer wirkt, was sich zum einen in der Wellenform/Bewegung widerspiegelt und zum anderen in der Übergangskante zwischen Wasser und Ufer.

Farbe: Wie realistisch sieht die Oberflächenfarbe aus? / Wie realistisch sieht die Farbabsorption des Wassers in der Tiefe aus? / Verleiht das Wasser einen realistischen Eindruck von Tiefe?

Durch das Raytracing hat der Image-Effekt genaue Informationen über die Tiefe des Wassers und kann Effekte wie Farbabsorption sehr physikalisch korrekt umsetzen, wohingegen andere Verfahren auf eine einheitliche Wasserfarbe ausweichen müssen.

Lichtbrechung: Sieht die Spiegelung des Wassers realistisch aus? / Sehen die Caustics realistisch aus?

Diese Fragen haben nichts direkt mit der Verwendung von Raytracing als grundlegendem Algorithmus zu tun, sondern dienen eher der Einstufung der generellen visuellen Qualität des Image-Effektes.

Diese Fragen werden im Laufe der Umfrage zwei Mal gestellt. Zuerst bei der direkten Bewertung des praktischen Teils der Bachelorarbeit und dann erneut für den Vergleich zwischen der Nvidia Island Demo und der Bachelor Arbeit.

Dadurch wird zum einen die Frage beantwortet, ob der praktische Teil der Arbeit einen hohen Qualitätsstandart erreicht, und zum anderen wie der praktische Teil im Vergleich mit anderen Lösungen aus der Branche steht.

4 Durchführung

4.1 Visualisierung von Objekten als Image-Effekt

Ein Post-Process-Effect ist ein Effekt, welcher nach dem Rendern der Szene auf das fertige Bild angewendet wird. Zu diesem Zeitpunkt ist keine komplette dreidimensionale Darstellung der Szene vorhanden. Die einzigen Daten, die für jeden Pixel zur Verfügung stehen, sind seine Position auf dem Bildschirm, seine Tiefeninformation (wie weit dieser Pixel von der Kamera entfernt ist) und die Farbe des jeweiligen Pixels (vgl. Ki 11). Des Weiteren besteht Zugang zu allen globalen Informationen, wie zum Beispiel der Position der Kamera oder die Position der Sonne beziehungsweise die Einfallrichtung des Sonnenlichtes.

Durch Kombination aller Informationen über die Kamera (Position, Blickrichtung, etc.) und der Position des Pixels in Bildschirmkoordinaten lässt sich die Position des Pixels in der dreidimensionalen Welt wiederherstellen. Mithilfe dieser Position kann man die simpelste Darstellung von Wasser einfach umsetzen, indem alle Pixel, deren Positionen in der Welt tiefer sind als die Oberfläche des Wassers, blau eingefärbt werden. Dieses einfache Verfahren erzeugt zwar weder realistische Wellen, noch optisch ansprechende Konturen auf der Oberfläche des Wassers, aber es veranschaulicht die grundlegende Arbeitsweise des Verfahrens.

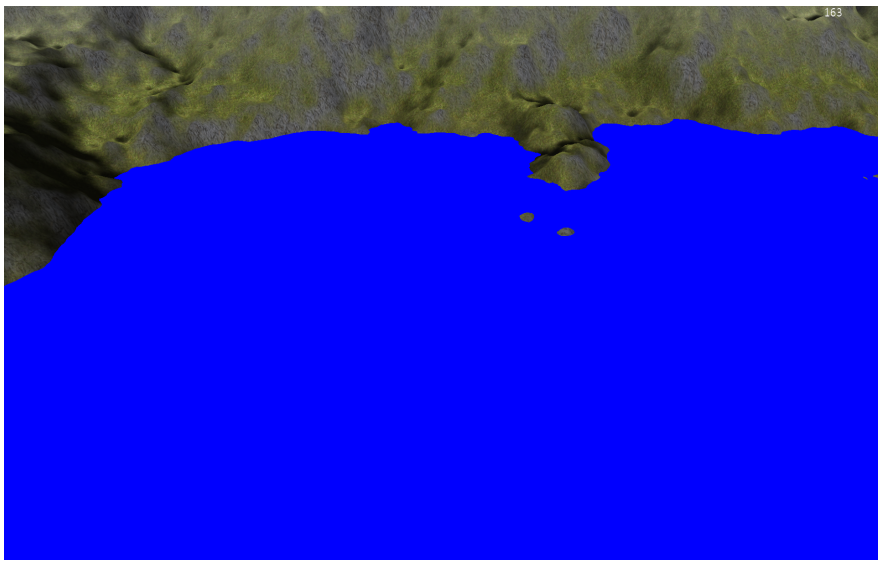


Illustration 1: Einfachste Darstellung als reines Blau

Mithilfe von Raytracing lässt sich die Position der Wasseroberfläche an der Stelle des aktuellen Pixels besser berechnen, sodass auch Wellen möglich sind. Raytracing bezeichnet eine Vielzahl von Algorithmen, welche einen Strahl (Ray) entlang gehen und für jeden Punkt auf diesem Strahl überprüfen, ob er ein bestimmtes Kriterium erfüllt (tracing) (vgl. Atwood 08). In Falle des Wassers wird für jeden Punkt zwischen Kamera und Pixel überprüft, ob dieser Punkt im Wasser ist, oder ob er über dem Wasser liegt. Der eine Punkt, welcher genau auf der Oberfläche des Wassers liegt, ist dann der Punkt, der für alle weiteren Effekte des Wassers genutzt wird.

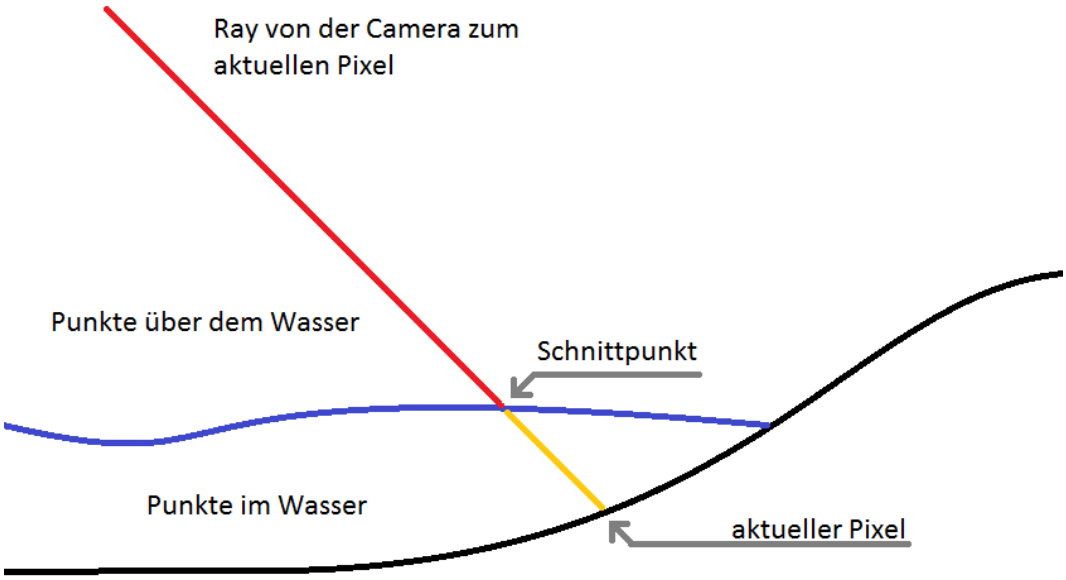


Illustration 2: Raytracing

Da es auf dem Strahl unendlich viele Punkte gibt, welche unmöglich alle einzeln überprüft werden können, müssen verschiedene Algorithmen zur Hilfe genommen werden. Diese Algorithmen helfen, den einen gesuchten Punkt (auf der Wasseroberfläche) schneller zu finden, sind dafür aber nicht immer 100% genau. Der einfachste Algorithmus, der verwendet werden kann, um den Schnittpunkt zwischen Wasseroberfläche und Strahl zu finden, ist eine einfache Schnittpunktberechnung durch Gleichsetzen. Ist das Wasser durch eine mathematische Funktion beschreibbar, so lässt sich durch Gleichsetzen mit der Funktion des Strahles ein genauer Schnittpunkt berechnen.

Beispielhafte Funktion des Wassers: $f(x)=\sin(x)$

Funktion des Strahles: $f(x)=RayOrig+x*RayDir$

Gleichsetzen liefert den X-Wert, an dem der Strahl die Wasseroberfläche schneidet, welcher dann wieder in eine der beiden Funktionen eingesetzt werden kann, um die Position des Punktes im Raum zu bestimmen.

Dieses Verfahren hat für die praktische Umsetzung im spezifischen Fall allerdings einige Probleme. Zum einen lässt sich die Wasseroberfläche von realem Wasser nicht durch eine Funktion beschreiben, zum anderen haben Shader-Sprachen wie HLSL nur einen begrenzten Umfang von mathematischen Funktionen, was das Bestimmen von Schnittpunkten zwar nicht unmöglich macht, die Umsetzung und Performance allerdings negativ beeinflusst.

Eine weitere Möglichkeit, die Position der Oberfläche zu bestimmen, ist eine Annäherung an die Position ähnlich der Binären Suche (vgl. Helmich 09). Angefangen wird mit 2 Punkten, die Position der Kamera und die Position des Pixels. Die Kamera ist außerhalb des Wassers und der Pixel ist innerhalb des Wassers. Dann wird ein neuer Punkt genau in der Mitte der beiden Punkte überprüft. Ist dieser Punkt innerhalb des Wassers, wird der Algorithmus fortgeführt mit diesem Punkt und der Position der Kamera, ansonsten wird der Algorithmus fortgeführt mit diesem Punkt und der Position des Pixels.

Rekursiv nähert sich dieser Algorithmus dadurch immer näher an die wirkliche Wasseroberfläche an, erreicht sie aber nie genau. Aber je mehr Rekursionen durchlaufen werden, desto genauer wird das Ergebnis.

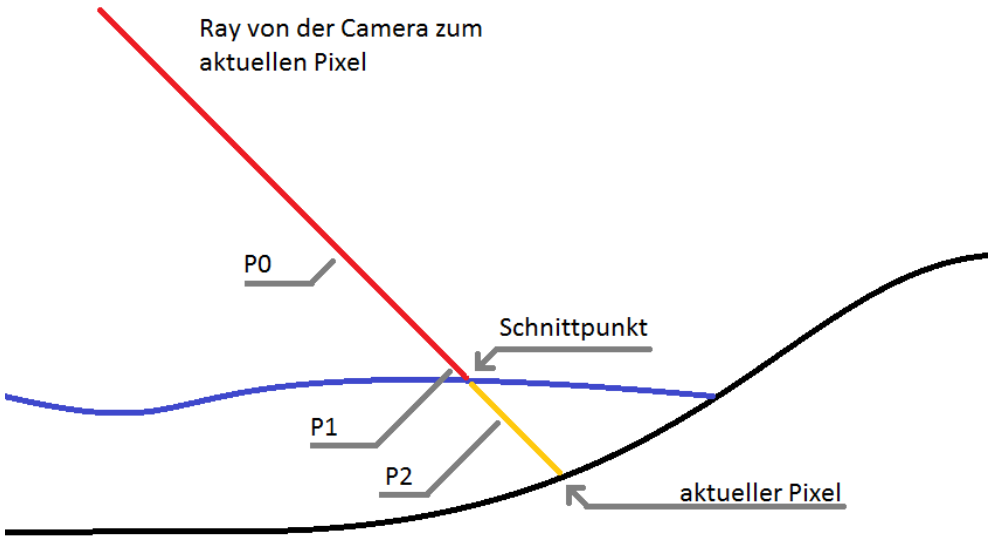


Illustration 3: Binäre Suche

Die Grundlage dieses Algorithmus ist sehr effektiv und funktioniert für jegliche Form des Wassers, solange man für einen beliebigen Punkt X bestimmen kann, ob dieser innerhalb des Wassers liegt oder nicht. Allerdings lässt sich die Präzision des Algorithmus noch verbessern durch die Nutzung einer Heuristik[13].

Wenn ein Punkt innerhalb des Wassers liegt, wird zunächst ein Punkt überprüft, der näher am Betrachter liegt. Der neue Punkt wird bei der Binären Suche durch das Unterteilen des aktuellen Bereiches in 2 neue Bereiche bestimmt. Das heißt, wenn der aktuelle Punkt im Wasser liegt, aber nur sehr knapp unter der Wasseroberfläche, dann kann es sein, das der nächste überprüfte Punkt sehr weit außerhalb der Wasseroberfläche liegt. Stattdessen wäre es sinnvoll, die Information „Wie nah ist der Punkt an der Wasseroberfläche“ für die Bestimmung des nächsten Punktes zu nutzen, der überprüft werden soll. Je näher der Punkt an der Wasseroberfläche ist, desto näher sollte der nächste Punkt an dem aktuellen liegen und umgekehrt.

13 Heuristik: Schätzung eines Wertes aufgrund bekannter Faktoren

Im Endeffekt ist die im Image-Effekt benutzte Herangehensweise eine Kombination der beiden oben beschriebenen.

Im ersten Schritt wird die Wasseroberfläche als gerade Fläche betrachtet und mithilfe von Gleichsetzung wird die Position des Punktes bestimmt, wo der Strahl mit der geraden Fläche zusammentrifft:

Wasser: $f(x) = WasserHöhe$

Strahl: $f(x) = CamPos + x * CamDir$

Durch Gleichsetzung und Vereinfachung ergibt dies eine einfache Formel, welche auch im Image-Effekt ohne Probleme errechnet werden kann:

$$CamPos.y + x * CamDir.y = Wasserhöhe$$
$$x * CamDir.y = Wasserhöhe - CamPos.y$$
$$x = (Wasserhöhe - CamPos.y) / CamDir.y$$

Nach der Berechnung des Schnittpunktes mit der angenäherten Wasserebene wird dann mithilfe von Linearer Suche der ungefähre Punkt bestimmt, an dem sich die Wasseroberfläche befindet. Mithilfe dieses Punktes lassen sich dann folgende Werte berechnen, die für den späteren Verlauf des Image-Effektes benötigt werden:

CurrentTracingPosition: (ungefähre) Position der Wasseroberfläche

DiagonalPenetration: Entfernung zwischen Oberflächenposition und Pixel

VerticalPenetration: Entfernung zwischen Pixel und Wasseroberfläche direkt über dem Pixel

4.2 Physikalische Eigenschaften der Visualisierung

Nachdem mithilfe von Raytracing genauere Daten über das Wasser an der Stelle des aktuellen Pixels bestimmt sind, lassen sich jetzt die einzelnen Komponenten errechnen, welche am Ende das Gesamtbild ergeben.

4.2.1 Reflexion und Refraktion

Wenn man auf ein halbtransparentes Objekt schaut, wie zum Beispiel Glas oder Wasser, dann setzt sich das, was man sieht, aus 2 Teilen zusammen. Zum einen kann man durch das Objekt hindurchschauen und sieht alle Objekte dahinter (Refraktion). Zum anderen spiegelt sich auf der Oberfläche des halbtransparenten Objektes alles (Reflexion). Dabei ist zu beachten, dass die Zusammensetzung aus Reflexion und Refraktion nicht immer 50/50 ist, sondern sich ändert, je nachdem aus welchem Winkel auf das halbtransparente Objekt geschaut wird (vgl. Lengyel 12). Schaut man direkt auf eine Glasscheibe, so ist alles hinter der Glasscheibe klar erkennbar, wohingegen auf der Oberfläche des Glases fast nur die Reflexion erkennbar ist, wenn man in einem sehr flachem Winkel auf das Glas schaut.

Die Refraktion für das Wasser an dem aktuell berechneten Punkt zu bestimmen ist trivial, da direkten Zugriff auf die ursprüngliche Farbe des Pixels an der entsprechenden Stelle möglich ist, welcher die Farbe des Wassergrundes darstellt und somit gleich der Refraktions-Farbe ist.

Die Reflexion ist etwas komplizierter und benötigt einen zweiten Rendering-Pass, welcher vor dem Image-Effekt ausgeführt werden muss. Bevor die gesamte Szene aus Sicht der Kamera gerendert wird, muss zuerst die „Reflection Map“[14] erzeugt werden. Diese Map ist im Grunde nichts anderes, als die gesamte Szene aus der Sicht einer anderen Kameraposition. Diese Kameraposition befindet sich unterhalb der Wasseroberfläche und schaut nach oben, wodurch sie beim Rendern genau das Bild erzeugt, welches sich in der Wasseroberfläche spiegelt. Um die genaue Position der neuen Kamera zu berechnen, werden die Position und die Blickrichtung der Kamera entlang der Wasserebene gespiegelt.

Der letzte Punkt ist die Mischung dieser beiden Komponenten zu einer Farbe für das Shading des Wassers. Dafür werden die Fresnel-Formeln verwendet, welche das Verhältnis zwischen Reflexion und Refraction beim Übergang eines Mediums in ein anderes (in diesem Fall von Luft zu Wasser) beschreiben (vgl. Lengyel 12). Für Computerspiele werden die Fresnel-Formeln nicht exakt berechnet, sondern nur mithilfe von folgender Formel angenähert, um eine entsprechende Effizienz zu gewährleisten:

$$fresnel = R + (1 - R) * (1 - \cos(BlickWinkel))^4$$

$$R = \frac{R0 - R1}{R1 + R0}$$

Dabei sind R0 und R1 die Refraktionsindices[15] von Wasser und Luft, welche aus Tabellen abgelesen werden können. R0 (Wasser) ~= 1.33 und R1 (Luft) = 1. Das Ergebnis der Formel gibt den Fresnel-Factor an bzw. die Stärke der Reflexion für einen bestimmten Winkel. Die Gesamtfarbe lässt sich also durch folgende Formel bestimmen:

$$FinalColor = ReflectionColor * fresnel + RefractionColor * (1 - fresnel)$$

14 Reflection Map: Gerendertes Bild der Szene aus einer spezifisch angepassten Kameraposition
15 Refraktionsindex: Brechungsindex, physikalische Eigenschaft, welche die Lichtbrechung beeinflusst

4.2.2 Lichtabsorption

Die Refraktion ist, wie oben beschrieben, der Teil des Image-Effektes, in dem man den Grund des Wassers sehen kann. Allerdings ist der Grund des Wassers nicht immer klar und deutlich erkennbar. Vor allem in tiefen Seen oder im Meer lässt sich der Boden nicht erkennen, stattdessen sieht man nur einen tiefen Blauton. Obwohl das Wasser transparent ist, scheint es eine eigene Farbe zu haben. In Wahrheit liegt dieser Effekt aber eher daran, dass Wasser das Licht absorbiert. Diese Lichtabsorption passiert je nach Wellenlänge des Lichtes in unterschiedlichen Tiefen des Wassers[2].

So wird zum Beispiel rotes Licht, welches eine sehr hohe Wellenlänge (etwa 780nm) hat, sehr früh absorbiert und blaues Licht, welches eine sehr geringe Wellenlänge (etwa 380nm) hat, sehr spät absorbiert. Dadurch erscheint tiefes Wasser in seinem natürlichen dunklen Blauton, da alle anderen Farben bereits absorbiert wurden und der Blauton als einziges durchscheint.

Im Image-Effekt ist dieser Effekt einfach zu reproduzieren, da Farben im Image-Effekt immer aus den drei Komponenten Rot, Grün und Blau bestehen. Für jede dieser Komponenten lässt sich mithilfe der Wassertiefe berechnen, wie stark diese Komponente bereits absorbiert wurde. Ab einer bestimmten Tiefe sind die jeweiligen Komponenten dann komplett Null, bzw. komplett absorbiert.

Als Werte für die Absorptionstiefen werden Werte ausgewählt, die ein realistisch aussehendes Wasser schaffen (Rot : 4,5 Meter Green : 75 Meter Blau : 300 Meter). Diese Werte sind ändern sich je nach Zusammensetzung des Wassers (vgl. Braun & Smirnov 93).

Für die Berechnung der Absorption wird eine lineare Funktion verwendet. Dies ist zwar nicht 100% physikalisch akkurat, liegt aber sehr nahe an der Realität und lässt sich sehr effizient umsetzen. Daraus ergibt sich für jeden Kanal folgende Formel:

$$Farbstärke = 1 - MaximaleAbsorptionsTiefe / Wassertiefe$$

Die Wassertiefe ist der gesamte Weg, den das Licht durch das Wasser zurücklegt. Dieser wird aus den Werten des Raytracings berechnet und zwar durch Addition der „VerticalPenetrationDepth“ (Weg, den das Licht zurücklegt von der Wasseroberfläche bis zum Grund) und der „DiagonalPenetrationDepth“ (Weg, den das Licht zurück legt auf dem Weg vom Grund zum Betrachter).

Die aus dieser Berechnung resultierenden Faktoren werden auf die Farbe der Refraktion angewandt und erschaffen so die für Wasser natürliche Farbabsorption und einen Eindruck von Wassertiefe.

4.2.3 Caustics

Wenn das Sonnenlicht ins Wasser einfällt, wird es beim Übergang ins Wasser gebrochen. Anhand der Wellenform ergeben sich deswegen auf dem Meeresgrund mit den Wellen tanzende Linien. Dieser Effekt ist an sich ein simples natürliches Phänomen, bei dem die Wasseroberfläche durch die Wellen ähnlich wie eine Lupe agiert, und die eintreffenden Sonnenstrahlen an bestimmten Stellen auf dem Grund des Wassers fokussiert (vgl. Guardado & Sánchez-Crespo 07).

Diesen Effekt realistisch im Image-Effekt widerzuspiegeln wäre immenser Aufwand. Für jeden Pixel des Bodens müsste berechnet werden, wie viele Sonnenstrahlen genau an diesen Punkt gebrochen werden, wodurch wiederum mehrere hunderte Sonnenstrahlen berechnet werden müssten, um ein ansehnliches Ergebnis zu erzielen.

Daher wurde für die Caustics ein Verfahren gewählt, welches zwar keinerlei physikalischen Bezug hat, aber dennoch die tanzenden Muster auf dem Boden des Wassers erzeugt. Für den menschlichen Betrachter ist es wahrscheinlich unmöglich festzustellen, dass diese Linien an der falschen Stelle sind.

Um die Linien zu berechnen, wird ein Querschnitt durch das Wasser auf ungefähr halber Wellenhöhe gemacht. Dieser Querschnitt zeigt dann Flächen an denen das Wasser höher ist und Flächen an denen das Wasser tiefer ist. Die Linien zwischen diesen Flächen werden als Caustics verwendet und auf dem Meeresgrund in einem etwas helleren Licht dargestellt.

Da sich die Wellen selbst im Laufe der Zeit bewegen und ihre Form verändern, tanzen auch die Caustics auf dem Boden des Wassers.

4.2.4 Wellenbrechung

Ein wichtiger Aspekt, damit das oben beschriebene System für das Raytracing funktioniert, ist die Möglichkeit, für einen beliebigen Punkt im Raum zu bestimmen, ob dieser Punkt innerhalb des Wassers liegt oder außerhalb. Um die oben beschriebene Optimierung der linearen Suche anzuwenden, muss darüber hinaus bestimmt werden können, wie weit unter oder über der Wasseroberfläche sich der Punkt befindet. Beide Probleme lassen sich durch eine Funktion beheben, welche für jeden beliebigen Punkt die Wasserhöhe an der entsprechenden Stelle berechnet. Die Differenz aus der Wasserhöhe minus der Höhe des Punktes ergibt dann den Abstand von der Wasseroberfläche. Ist dieser positiv, so befindet sich der Punkt im Wasser, ansonsten befindet er sich über dem Wasser.

Um die Höhenfunktion für das Wasser möglichst realistisch zu halten, wurden die Charakteristiken von realem Wasser auf die Funktion übertragen. Wellenbewegung im Wasser ist nichts anderes als sehr viele einzelne Wellen, die in verschiedenen Richtungen über die Wasseroberfläche wandern und sich zum Gesamtergebnis der Wasserhöhe zusammen addieren.

Um die Form einer einzelnen Welle zu generieren, könnte man auf die Form einer Sinuskurve zurückgreifen. Diese Form sieht aber im letztendlichen Abbild des Wassers sehr unrealistisch aus, da echtes Wasser nie solche perfekten Formen erreicht. Reales Wasser besteht zwar aus Sinuskurven, aber es sind mehrere Tausende davon übereinander gelagert, sodass man die Kurven selbst nicht mehr erkennen kann.

Um solche immensen Rechenzeiten zu umgehen, wird anstelle der Sinuswelle eine künstlich erzeugte Welle aus einer Textur geladen, welche selbst schon die Überlagerung mehrerer Wellen ist, sodass insgesamt nicht mehrere tausend Ebenen generiert werden müssen, sondern nur ein paar wenige.

Die Verlaufsrichtung der einzelnen Ebenen ist immer abwechselnd nach links und nach rechts und die ersten Ebenen sind dominanter als die späteren, wodurch sich insgesamt ein sehr realitätsnahes Wellenbild ergibt, welches sehr effizient mithilfe einiger Texture-Lookups[16] umgesetzt werden kann.

16 Texture-Lookup: Lesezugriff auf die Daten aus einer Textur

4.3 Durchführung der Umfrage

Im Rahmen der Umfrage wurde ein Video erstellt, welches die praktische Arbeit der Bachelorarbeit widerspiegelt. Dieses Video enthält mehrere Szenen, die jeweils ein bestimmtes Feature der Demo zeigen, und wurde direkt aus der Demo heraus gerendert und zusammengeschnitten. Es ist auf YouTube verfügbar als nicht gelistetes Video, so dass der Link (welcher natürlich in der Umfrage enthalten ist) benötigt wird, um das Video anschauen zu können.

Die Umfrage selbst wurde mithilfe von www.umfrageonline.com erstellt und hatte eine befristete Laufzeit von einem Monat (27.06.15 – 27.07.15). umfragenonline.com bietet einfache Erstellung von professionellen Umfragen, welche sich zu einem sehr hohen Grad anpassen lassen und daher für die genutzte Umfrage eine gute Wahl darstellte.

Die Verteilung der Umfrage an die Teilnehmer lief über direkte Mund zu Mund Propaganda, was sich vor allem ermöglichte, da die Umfrage eine sehr große Zielgruppe hat. Gerade die Studenten an der SAE-Köln trugen viel zu den Ergebnissen aus der Umfrage bei.

Auch wenn die Umfrage direkt nicht mehr abrufbar ist, wird das Video abrufbar bleiben unter:

https://www.youtube.com/watch?v=dhZl9tQc2a0&feature=youtu.be

5 Ergebnisse

5.1 Performance Ergebnisse

Bei der Berechnung der durchschnittlichen Frametime[17] wird für alle im Folgenden repräsentierten Ergebnisse ein Zeitfenster von 10 Sekunden angesetzt. Während diesem Zeitfenster wird gezählt, wie viele Frames gerendert werden, um die Frametime zu bestimmen als der Quotient vom Zeitfenster durch Anzahl an gerenderten Frames.

Für alle folgenden Messungen wurde ein High-End Gaming Rechner genutzt, mit folgender Hardware (nur die Relevante Hardware ist spezifisch gelistet):

Prozessor:	Intel Core i7-4770K @ ~3,5 GHz (8 CPUs)
Grafikkarte:	NVIDIA GeForce GTX 780
Arbeitsspeicher:	16 GB
Motherboard:	ASUS Z87-K

Zusätzlich dazu Mouse, Keyboard, 2 Monitore, eine SSD, ein HDD und ein DVD Laufwerk.

Auch wenn die generelle Frametime ohne genauere Bestimmung der Bottlenecks keine präzise Performance-Information liefert, wird zuerst die generelle Frametime berechnet um eine grobe Abschätzung für die Performance des Effektes zu erhalten. Da der Effekt für jeden Pixel des Bildes berechnet wird, weicht die Performance ab, je nachdem wie viel Fläche des Bildes mit Wasser bedeckt ist. Daher werden für verschiedene Szenarien verschiedene Daten erhoben.

17 Frametime: Zeit die benötigt wird um ein einzelnes Bild darzustellen

Frametime ist die Zeit, die benötigt wird, um die gesamte Szene zu rendern, da hier aber nicht die Performance der gesamten Szene betrachtet werden soll, sondern die Performance des Effektes, wird noch die reine Frametime berechnet. Diese reine Frametime ist die Zeit, die für die gesamte Szene (inkl. Wasser) benötigt wird, abzüglich der Zeit, die für die gesamte Szene ohne Wasser benötigt wird, wodurch nur noch die Zeit des Wassers übrig bleibt.

Wenn der Wassereffekt nicht gerendert wird, dann werden innerhalb des 10 sekündigen Zeitfensters insgesamt 5112 Frames gerendert, was einer Frametime von ~1,96 ms entspricht (10 s / 5112).

Die Bedeckung des Bildschirmes wird erreicht durch entsprechendes Neigen der Kamera, sodass der Horizont sich auf dem Bildschirm weiter oben oder unten befindet. Dadurch bedeckt das Wasser einen größeren oder kleineren Teil des Bildes.

Bedeckung des Bildschirmes	**Gerenderte Frames**	**Frametime**	**Reine Frametime**
100%	2580	3,88 ms	1,92 ms
75%	2887	3,46 ms	1,50 ms
50%	4533	2,21 ms	0,53 ms
0%	5076	1,97 ms	0,01 ms
100% (Camera Down)	4882	2,05 ms	0,09 ms

Um die obigen Werte in Perspektive zu setzen, wird von einer Anwendung ausgegangen, welche in Full-HD Auflösung läuft (1920*1080) und auf einem Standard-Monitor mit 60 Hz läuft. Um die Refreshrate des Monitors (60 Hz) voll auszunutzen werden 60 Frames pro Sekunde benötigt, was bedeutet, dass jedes einzelne Bild 1/60 Sekunde Renderzeit nutzen darf, oder genauer 16,67 ms.

Das Wasser benötigt also sogar im Worst-Case-Scenario (das gesamtes Bild ist Wasser) nur etwa ein Achtel der gesamten Frametime. Im Regelfall wird eine Anwendung allerdings nicht mehr als 50% des Bildes mit Wasser bedeckt haben (auch wenn es Ausnahmen geben mag). In diesen Fällen belegt das Wasser sogar nur etwa ein Achtzehntel der Frametime, was ein exzellentes Ergebnis ist.

Die obigen Ergebnisse zeigen sogar direkt, wo in dem Shader das Bottleneck liegt. Da die Frametime drastisch sinkt, sobald ein kleinerer Teil des Bildes mit Wasser bedeckt ist, liegt das Bottleneck definitiv im Pixelshader. Der Schritt des Rasterizer in der Pipeline wird nämlich trotzdem auf den gesamten Bildschirm angewendet, erst im Pixelshader wird entschieden, dass bestimmte Pixel des Bildes nicht gerendert werden müssen. Der Pixelshader ist bei dieser Entscheidung, ob ein Pixel gerendert werden muss oder nicht, sehr performant. Ist kein Pixel im gesamten Bild mit Wasser bedeckt, so benötigt der Effekt insgesamt nur ~0,01 ms.

Dieses Bottleneck im Pixelshader ist zu erwarten, da man dem Quellcode des Image-Effektes direkt ansieht, dass der Pixelshader einen Großteil der Performance benötigt. Der Vertexshader ist 7 Zeilen lang, wohingegen der Pixelshader sogar ohne die Funktionen 85 Zeilen lang ist. Dazu kommen noch die jeweiligen Funktionen und die Tatsache, dass der Code innerhalb einer Schleife mehrmals ausgeführt wird, obwohl er nur einmal zu der gesamten Anzahl an Zeilen beiträgt. Zusätzlich dazu sind Textur-Operationen immer eine sehr langsame Operation, wovon der Pixelshader 39 Stück verwendet und der Vertexshader keine einzige.

Was bei der Betrachtung der Daten auffällt, ist eine sehr geringe Frametime, wenn man die Kamera direkt nach unten neigt und nicht Richtung Horizont. Dieses Ergebnis tanzt aus der Reihe, da in diesem Fall das Wasser 100% des Bildschirmes belegt und trotzdem nur eine sehr geringe Frametime benötigt wird. Im Vergleich zu der 100% Bedeckung bei Betrachtung des Horizontes ist der Unterschied 1,83 ms. In beiden Fällen werden für jeden Pixel genau die gleichen Anweisungen ausgeführt, der einzige Unterschied sind die Werte der entsprechenden Variablen und somit die Stellen an denen die entsprechenden Texturen gelesen werden müssen. Die Grafikkarte ist in diesem Punkt sehr effektiv gestaltet, sodass das Lesen aus einer Textur sehr schnell geht, wenn man mehrmals an der gleichen Stelle liest (vgl. Doggett 12). Schaut die Kamera direkt nach unten, so wird bei der Bestimmung der Wasserhöhe sehr häufig an derselben Stelle (oder an einer sehr nahen anderen Stelle) die Textur gelesen, was sehr performant ist. Dieser Effekt zeigt auch direkt das genaue Bottleneck des Effektes, nämlich das Lesen aus der Textur.

Dieses Lesen der Textur macht mehr als 90% des Effektes aus (~1,83 ms / 1,92 ms).

Nutzt man das Shader tool von DirectX 11 (fxc.exe), so wird der Shader compiliert mit insgesamt 39 Texture Instructions, von denen jede die Nutzung einer Textur widerspiegelt. Diese liegen zum Großteil in der Bestimmung der Wasserhöhe: Der Effekt bestimmt 6-mal die Wasserhöhe, was jeweils 5-mal Texturdaten anfordert. Allein dadurch muss 30-mal eine Textur gelesen werden.

Im Vergleich dazu ist die Performance des Nvidia Effektes etwas komplizierter zu bestimmen. Dies liegt zum einen daran, dass es sich nicht um eigenen Code handelt und somit etwas Einarbeitungszeit von Nöten war, um den Code zur Performancebestimmung anzupassen, zum anderen lassen sich in der Demo viele Werte einstellen, welche die Performance der Demo beeinflussen. Diese Werte wurden allerdings einfachheitshalber auf den Standard Einstellungen gelassen, da der visuelle Vergleich zwischen den beiden Arbeiten später anhand der Videos stattfindet, und auch das Video der Nvidia Demo die Standard Einstellungen nutzt.

Wenn der Wassereffekt nicht gerendert wird, dann werden innerhalb des 10 sekündigen Zeitfensters insgesamt 3448 Frames gerendert, was einer Frametime von ~2,90ms entspricht (10s/3448).

Bedeckung des Bildschirmes	**Gerenderte Frames**	**Frametime**	**Reine Frametime**
100%	1074	9,31 ms	6,41 ms
75%	1138	8,79 ms	5,89 ms
50%	1334	7,50 ms	4,60 ms
0%	2163	4,62 ms	1,72 ms
100% (Camera Down)	1227	8,15 ms	5,25 ms

Insgesamt sieht man direkt, dass die Performance der Nvidia Demo schlechter ist. Es fällt allerdings auch auf, dass die Menge an Wasser, die sichtbar ist, nicht so stark in die Performance einfließt. Auch diese Zahlen waren allerdings zu erwarten, da mehr Operationen gemacht werden müssen, wenn das Wasser sichtbar ist. Dennoch ist ein Großteil der Performance von Nöten, selbst wenn das Wasser nicht sichtbar ist, da die Tesselation auf die Geometrie der Fläche immer angewendet wird (vgl. Bonaventura 11). Dennoch wird die Tesselation zu einem höheren Grad angewendet, wenn das Wasser sichtbar ist. Das Bottleneck liegt hier zum einen im Vertexshader (bzw. DomainShader/HullShader) und zum anderen im Pixelshader.

Im Vergleich liefert die Wasserdemo also einen generellen starken Performancegewinn. Zu beachten ist allerdings, dass in Grenzfällen die Tesselation-Demo trotzdem performanter sein kann, wenn die Software in allen anderen Bereichen nur den Pixelshader nutzt.

5.2 Ergebnisse aus der Umfragen

5.2.1 Zielgruppe

Das Durchschnittsalter der Teilnehmer liegt bei ungefähr 32 Jahren. Ein Großteil der Teilnehmer liegt entweder zwischen 22 und 26 Jahren oder etwa um 50 Jahre herum. Der Grund hierfür ist, dass viele Umfrageergebnisse von den Studenten der SAE kommen, wohingegen die anderen Ergebnisse aus dem eigenen Bekanntenkreis stammen.

Am Berufsbild erkennt man, dass diese Aufteilung ausgeglichen ist und ungefähr die Hälfte der Teilnehmer von der SAE kommt.

Auch die Tatsache, dass ein Großteil der Teilnehmer männlich ist, wundert nicht, da an der SAE hauptsächlich männliche Studenten sind.

Auch wenn nicht alle Teilnehmer in der Gamesbranche studieren, so verbringen sie dennoch zum Großteil viel Zeit am PC. Nur 27% der Teilnehmer spielen gar kein PC und etwa die Hälfte aller Teilnehmer (52,21%) spielt mehr als 6 Stunden pro Woche.

Gerade die ausgewogene Mischung zwischen Gamern und Nicht-Gamern macht es möglich, eine repräsentative Aussage aus der Umfrage zu gewinnen.

5.2.2 Direkte Bewertung der Qualität

Für die Auswertung der jeweiligen Daten wurden zuerst alle Ergebnisse jeweils in Mittelwert und Standartabweichung zusammengefasst, um genaue Informationen über die durchschnittliche Meinung der Teilnehmer und die Streuung der Ergebnisse zu erhalten.

	Mittelwert	Standardabweichung
Gesamteindruck	79,08	15,78
Realismusgrad	75,14	18,00
Wellenbewegung	76,38	17,49
Zusammenspiel mit Umgebung	71,24	18,14
Oberflächenfarbe	71,52	15,88
Farbabsorption	72,77	17,36
Eindruck von Tiefe	74,65	17,33
Reflexion	74,62	16,40
Caustics	68,24	17,86

Bei allen obigen Werten steht ein Wert von 0 für „Total unrealistisch / unschön“ und ein Wert von 100 für „sehr realistisch / schön“. Ein Wert von 50 spiegelt daher eine neutrale Meinung wieder und alle Werte über 50 eine positive.

Gerade durch die beiden hohen Bewertungen in den Bereichen Gesamteindruck und Realismusgrad lässt sich die generelle Tendenz der Teilnehmer der Umfrage erkennen. Das generelle Feedback ist, dass die Demo bzw. der Wassereffekt innerhalb der Demo gut gelungen ist und einem hohen Standard entspricht. Auch die Ergebnisse in allen anderen Bereichen belegen diesen Gesamteindruck mit jeweils guten Bewertungen.

Der negativ auffallende Wert ist hier die Bewertung der Caustics mit einem Durchschnittswert von etwa 68. Das ist an sich zwar nicht schlecht, ist aber von allen einzel Ergebnissen das niedrigste. Da die Caustics aber wirklich nicht auf die beste Art umgesetzt wurden und teilweise nur sehr schwer sichtbar sind, war eine etwas schlechtere Bewertung in diesem Bereich ersichtlich und die Tatsache, dass hier nicht so hohe Ergebnisse erzielt wurden zeigt, dass ehrlich geantwortet wurde.

Gerade wichtig für die Arbeit sind die Punkte Wellenbewegung und Zusammenspiel mit der Umgebung, da diese maßgeblich von dem Raytracing-Verfahren beeinflusst werden. Den Ergebnissen ist in diesem Berreich zu entnehmen, dass das angewandte Verfahren einen insgesamt positiven Eindruck verleiht und die Repräsentation des Wassers unter der Nutzung des Verfahrens nicht leidet.

Die hier angegebenen Werte sind alleinstehend allerdings nicht allzu Aussagekräftig, da die Antworten der Teilnehmer in keinerlei Relation stehen, sondern jeder Teilnehmer in Relation zu seinen eigenen Erfahrungen bewertet. Im nächsten Abschnitt werden daher die Werte in Relation zur Nvidia Demo ausgewertet.

5.2.3 Vergleich zur Referenz

	Mittelwert	Standardabweichung
Gesamteindruck	41,08	22,40
Realismusgrad	41,52	20,03
Wellenbewegung	43,87	22,19
Zusammenspiel mit Umgebung	43,20	20,88
Oberflächenfarbe	44,41	20,22
Farbabsorption	42,16	20,75
Eindruck von Tiefe	41,02	19,80
Caustics	54,57	21,63

Bei allen obigen Werten steht ein Wert von 0 dafür, dass die Bachelor-Arbeit weitaus besser war als die Nvidia Demo und eine 100 steht dafür, dass die Nvidia Demo wesentlich besser ist als die Bachelor Arbeit. Ein Wert von 50 entspricht einem Gleichstand. Alle Werte unter 50 sprechen daher für die eigene Arbeit.

Die Demo der Bachelor-Arbeit schneidet in fast allen Punkten etwas besser ab, als die Nvidia Island Demo. Einziger Ausreißer ist hier wieder die Qualität der Caustics, welche zugunsten der Nvidia Demo bewertet wurde. Dieses Ergebnis war aber, wie oben bereits beschrieben, zu erwarten.

Die Bachelor-Arbeit wurde nicht um Längen besser bewertet als die Nvidia Demo, aber wenn man bedenkt, dass die Nvidia Demo eine highend Tech Demo aus der Industrie ist, ist eine Bewertung zugunsten der eigenen Arbeit ein enormer Qualitätsbeweis.

6 Zusammenfassung

Das Ziel dieser Arbeit war die Erstellung eines Wasser-Effektes, der komplett ohne Geometrie auskommt, und somit Wasser darstellen kann, ohne dass man einzelne Flächen erkennen kann. Diese Flächen fallen gerade beim Zusammenspiel des Wassers mit dem Ufer auf und sind somit einer der wichtigen Punkte, die bei zukünftigen Fortschritten im Bereich der Darstellung von Wasser verbessert werden müssen.

Der für diese Arbeit erarbeitete und präsentierte Effekt löst dieses Problem mit der Technologie des Raytracings. Raytracing ist zwar für die Darstellung von 3D Objekten sehr selten genutzt, erfüllt im Rahmen dieser Arbeit aber genau die Kriterien, die gestellt wurden. Nach ersten Tests wurde diese Technologie als eine Möglichkeit zur Umsetzung gewählt und dann im Laufe der Arbeit immer weiter verbessert und angepasst, um zum Endergebnis zu gelangen.

Um die praktische Nutzbarkeit dieses Effektes darzustellen, wurde er auf zwei Arten bewertet. Zum einen wurde eine detaillierte Analyse der Performance dieses Effektes erstellt und mit ähnlichen Effekten verglichen. Hier wurde vor allem Wert gelegt auf die Bewertung der Performance im Anwendungsgebiet von Computerspielen, was der Hauptanwendungsbereich von Wasser-Visualisierung ist.

Zum anderen wurde die visuelle Qualität des Effektes durch eine Umfrage in Relation zu bereits vorhandenen Visualisierungen von Wasser gestellt. Dabei wurde vor allem Wert gelegt auf die Aspekte des Effektes, die hauptsächlich beeinflusst werden durch das Abhanden sein von Geometrie, beziehungsweise durch die Nutzung von Raytracing.

Wie man bereits durch die Ergebnisse der Arbeit erkennen kann, stellt der durch diese Arbeit präsentierte Effekt in beiden oben genannten Bereichen eine gute Alternative zu den zurzeit genutzten Technologien dar.

Im Bereich der Performance benötigt der Effekt zwar etwas mehr Zeit im Pixelshader, da hier das Raytracing umgesetzt ist, dafür wird aber quasi gar keine Zeit im Vertexshader verbracht, wodurch der Effekt insgesamt weniger Zeit benötigt, als vergleichbare Alternativen, die eine ähnlich detaillierte Wasseroberfläche mithilfe von Geometrie und Tesselation erzeugen. Je nach Anwendung ist diese Ersparnis im Vertexshader im Tausch für mehr Zeit im Pixelshader ein Vorteil. Dadurch hat dieser Effekt, beziehungsweise die Nutzung von Raytracing im Bereich der Wasserdarstellung, definitiv Bereiche, in denen er allen Alternativen überlegen ist.

Im Bereich der Qualität zeigt der Effekt keine Defizite gegenüber der Referenz-Umsetzung, welche als Tech-Demo von Nvidia auf einem sehr hohen Qualitäts-Standard angesiedelt ist. Sogar ganz im Gegenteil, der Effekt erzielt sogar überwiegend bessere Ergebnisse in den Bereichen, die durch das Raytracing beeinflusst sind. Die einzigen Ergebnisse, die hier negativ auffallen, sind die Qualität der Lichtbrechung auf dem Meeresgrund. Diese hängen aber nicht in Zusammenhang mit dem Verzicht auf Geometrie und können unabhängig von der grundlegenden Technologie des Effektes verbessert werden.

Nun zu meinem persönlichen Fazit. Ich hatte schon vor einiger Zeit mit der Visualisierung durch Raytracing experimentiert. Obwohl es damals schon interessante Ergebnisse geliefert hatte, fehlte mir die Zeit, weiter in diesem Bereich zu experimentieren. Im Rahmen der Bachelor Arbeit hatte ich mir das Ziel gesetzt, diesen Effekt umzusetzen und ihn auf ein Qualitäts-Niveau zu bringen, dass mit aktuellen alternativen Technologien mithalten kann und sie in einigen Bereichen sogar überholt. Dadurch könnte dieser Effekt Anwendungsbereiche finden, in denen seine Spezifikationen optimaler, sind als die der „Konkurrenz“.

Gleichzeitig wollte ich meine Programmierkenntnisse im Bereich von Shadern ausweiten. Gerade die physikalisch korrekte Visualisierung von natürlichen Phänomenen hat mich hier immer interessiert, da PBR (Physically based Rendering) die Königsdisziplin im Bereich der Shader ist, da keine Effekte realer sind als die „Physikalische Realität“.

Beide meiner persönlichen Ziele wurden im Rahmen dieser Bachelorarbeit zu genüge erfüllt. Der entwickelte Effekt kann mit dem Qualitäts-Niveau mithalten, auf dem sich Tech-Demos wie die „Nvidia Island Demo“ befinden und übertrifft diese in einzelnen Bereichen sogar. Vor allem die Vorteile im Bereich der Performance sind für mich eine hervorragende Leistung, vor allem wenn man bedenkt, dass die visuelle Qualität unter der Nutzung des Raytracings nicht leidet, sondern im Gegenteil sogar Vorteile darstellt gegenüber Geometrie-basierten Verfahren.

Auch meine eigenen Programmierkenntnisse konnte ich durch die Entwicklung der praktischen Demo stark ausweiten. Da innerhalb des Studiums die Programmierung von Shadern einen eher kleinen Teil dargestellt hat und ich mir vieles selber angeeignet habe, war diese Arbeit eine sehr lohnenswerte Erfahrung. Ich habe viel neues Wissen in den Bereichen Shaderprogrammierung und der Umsetzung von physikalischen Phänomenen in Shadern erlangt.

Zukunftsausblick

Auch wenn viel Zeit für die Umsetzung dieses Effektes genutzt wurde, ist er natürlich nicht perfekt und es gibt einige Aspekte, die in Zukunft noch verbessert werden können.

Nutzbarkeit

Zur einfacheren Umsetzung besitzt der Effekt zurzeit noch zwei größere Einschränkungen, die allerdings in der Präsentation der Demo nicht auffallen, da sie für diese nicht relevant waren.

Zum einen lässt sich der Effekt zurzeit nicht einschränken. Das heißt, er wird immer als eine unendliche Fläche in alle Richtungen gerendert. Es gibt keine Möglichkeit, etwas wie zum Beispiel einen Stausee darzustellen, welcher aus einer höhergelegenen und einer niedrig gelegenen Wasserfläche besteht. Die obere Wasserfläche würde einfach das gesamte Tal ausfüllen, auch nach der Staumauer. Den Effekt anzupassen, um dies zu ermöglichen, ist definitiv möglich und wahrscheinlich nicht allzu viel Arbeit, war aber im Rahmen der Demo dieser Arbeit nicht von Nöten und wurde daher ausgelassen.

Zum anderen ist das Wasser nicht dazu ausgelegt, von innerhalb der Wasseroberfläche betrachtet zu werden. Geht man mit der Kamera von oben durch die Wasseroberfläche hindurch, zerbricht sofort die Illusion von dem Wasser und das Wasser wird total falsch dargestellt. Hier ist der Effekt anzupassen, um zu unterscheiden, ob die Kamera sich über oder unter der Wasseroberfläche befindet, um gegebenenfalls einen „Unterwasser“-Effekt zu erzeugen.

Performance

Der Algorithmus zur Bestimmung des Punktes an der Wasseroberfläche kann bestimmt noch auf viele Arten optimiert werden. Da ein Großteil der Performancekosten durch die Texturen kommt, und diese hauptsächlich im Raytracing Algorythmus genutzt werden, ist dies der Punkt, wo man am wichtigsten optimieren muss. Gerade durch die Verringerung der Anzahl Schleifendurchläufen lässt sich hier viel Performance sparen. Man könnte zum Beispiel eine bessere Schätzung einbauen, um den nächsten Punkt zu suchen, der überprüft wird. Dadurch käme man mit weniger Durchläufen zum gleichen Ziel und hätte damit insgesamt weniger Textur-Nutzung. Im Tausch hätte man natürlich mehr mathematische Operationen, um die Schätzung genauer zu bestimmen, aber diese Operationen sind wesentlich kostengünstiger.

Auch in den anderen Bereichen des Image-Effektes kann bestimmt noch einiges an Performance rausgeholt werden, auch wenn die Gewinne in diesen Bereichen wohl nicht so groß werden, wie im Bereich des Raytracings.

Qualität

Auch wenn der Effekt in der Umfrage mit einem guten Ergebnisse abgeschlossen hat, gibt es noch einige Bereiche, in denen andere Wasserumsetzungen besser sind, wie zum Beispiel die Caustics bei der Nvidia Island Demo. Auch wenn eine höhere Qualität der Umsetzung in diesem Bereich eventuell etwas mehr Performance kostet, könnte das deutlich bessere visuelle Ergebnis in diesem Bereich es wert sein.

Für jedes der Features (Caustics, Reflection, Farbverschluckung, Wellenform etc.) gibt es mit großer Wahrscheinlichkeit eine andere Wasserdemo, die in diesem Bereich einen besseren Algorithmus verwendet und damit ein besseres visuelles Ergebnis erzielt. Theoretisch könnte also in jedem Einzelbereich des Image-Effektes noch ergänzende Arbeit geleistet werden um den Effekt in dem entsprechenden Gebiet zu optimieren.

Abschließend ist zu sagen, dass das Thema Wasser und seine visuelle Darstellung so vielfältig und komplex ist, dass es auch auf längere Sicht hin gesehen eine besondere Herausforderung bleiben wird.

7 Anhang

7.1 Produktionslogbuch

Das Produktions Logbuch befasst sich mit der Funktionsweise der Demo, welche als praktischer Teil der Bachelorarbeit umgesetzt wurde. Zur Übersichtlichkeit geht die Beschreibung in Chronologischer Reihenfolge durch die einzelnen Schritte, in der Reihenfolge, wie sie beim Rendern eines einzelnen Frames ausgeführt werden.

08.01.2015 Darstellung der Umgebung

Normalerweise werden Objekte beim Rendern direkt auf den Bildschirm übertragen. Für die Umsetzung des Wassers ist es allerdings von Nöten, dass die Umgebung zuerst in einer Textur zwischengespeichert wird. Es wird also eine Textur erstellt, welche die gleichen Maße besitzt wie der Monitor (im Regelfall 1920x1080 Pixel) und alle Objekte werden in diese Textur gerendert, anstatt direkt auf dem Bildschirm angezeigt zu werden.

Gleichzeitig wird die Tiefen-Information der Szene in einem Extraobjekt auf der Grafikkarte gespeichert, da diese später im Laufe des Image-Effektes benötigt wird, um die Position eines jeden Pixels wiederherzustellen.

Die Szene selbst besteht eigentlich nur aus 3 Objekten:

Das Terrain

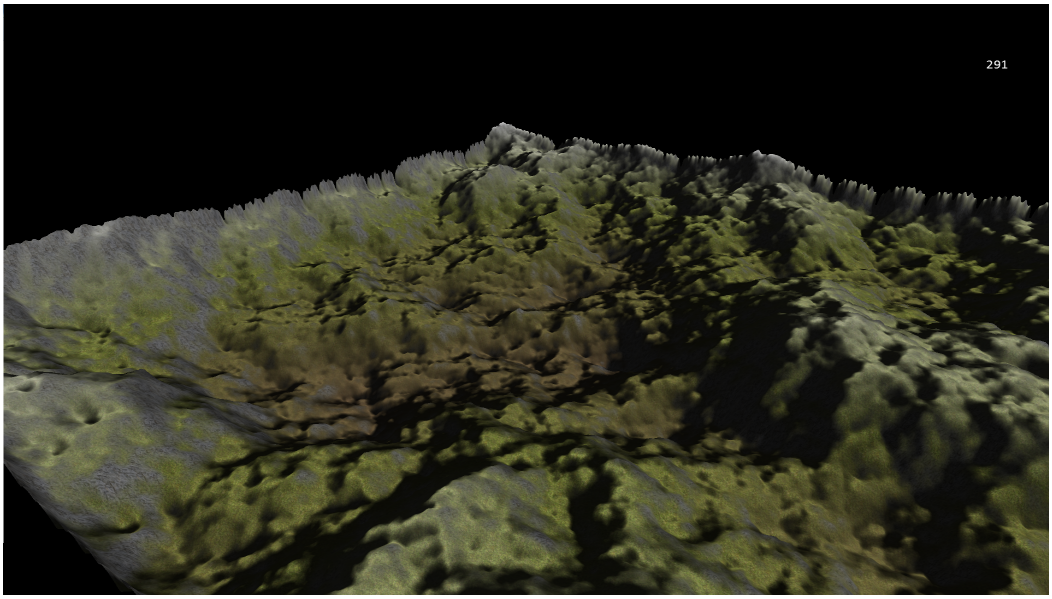


Illustration 4: Das Terrain

Das Terrain wird mit einem Shader gerendert, welcher die Informationen aus einer Textur nutzt, um dem Terrain Höhen und Tiefen zu geben. Außerdem wird das Terrain dabei je nach Höhe mit einer anderen Textur versehen. So erhalten zum Beispiel hohe Stellen auf dem Terrain eine Schnee Textur, wohingegen die niedrigeren Regionen mit Grass gefüllt werden. Darüber hinaus werden alle Flächen, die ein sehr starkes Gefälle aufweisen, mit Stein texturiert.

Texturierter Würfel

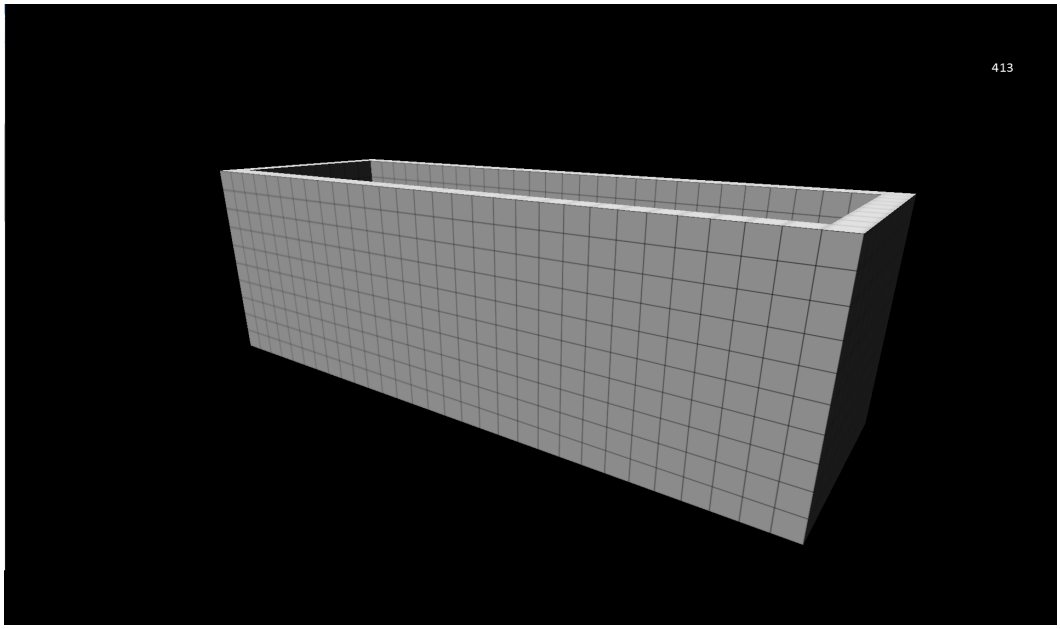


Illustration 5: Der Würfel

Im mittleren Teil des Videos sieht man auch noch einen weißen Würfel mit schwarzen Linien, welche immer genau einen Meter voneinander entfernt sind. Dieses Model wurde aus einem selbsterstellten .Obj File geladen und wird in der Szene mit einem einfachen Shader gerendert, welcher das Model inklusive Textur darstellt.

Himmel

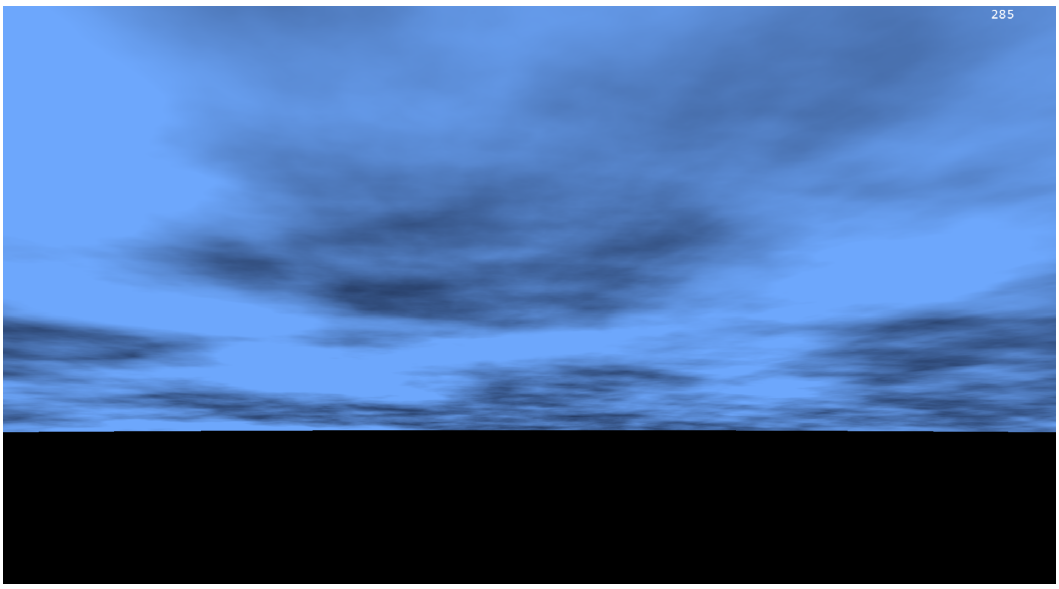


Illustration 6: Der Himmel

Auch wenn man ihn nicht direkt als Objekt wahrnimmt, ist der Himmel das dritte Objekt. Dieser wurde realisiert mithilfe einer Großen Halbkugel, die sich über den gesamten Horizont spannt. Die gesamte Textur des Himmels wird dynamisch in jedem Frame berechnet und besteht aus mehreren Perlin-Noise-Funktionen, welche übereinander gelagert die Wolken ergeben. Die Hintergrundfarbe des Himmels (zum Beispiel der Übergang zur roten Farbe, wenn die Sonne niedrig steht) wird aus einer Textur gelesen und je nach Sonnenstand leicht angepasst.

Die Visualisierung der Umgebung war insgesamt eine sehr einfache Aufgabe, da die hier erstellten Shader sehr ähnlich sind im Vergleich zu bisherigen eigenen Projekten. Einzig der Himmel sticht hier etwas hervor, da er komplett prozedural generiert ist und eine relativ komplexe Interaktion zwischen Horizont und Sonne besitzt. Hier habe ich einige weitere Fähigkeiten in dem Bereich der prozeduralen Textur-Erstellung erlangt, auch wenn der Teil selbst nur etwa eine Woche in Anspruch genommen hat. Gerade in diesem Bereich könnte man in der Zukunft die Demo zwar verschönern, da dies aber nicht der Fokus der Arbeit war wurde dieser Teil daher eher vernachlässigt.

20.01.2015 Rendern der Reflexion

Im zweiten Schritt wird die gesamte Szene erneut aus der Sicht der Reflektierten Kamera gerendert (siehe 4.2.1). Dafür wird die Kamera entlang der Y-Achse auf Höhe der Wasserebene gespiegelt.

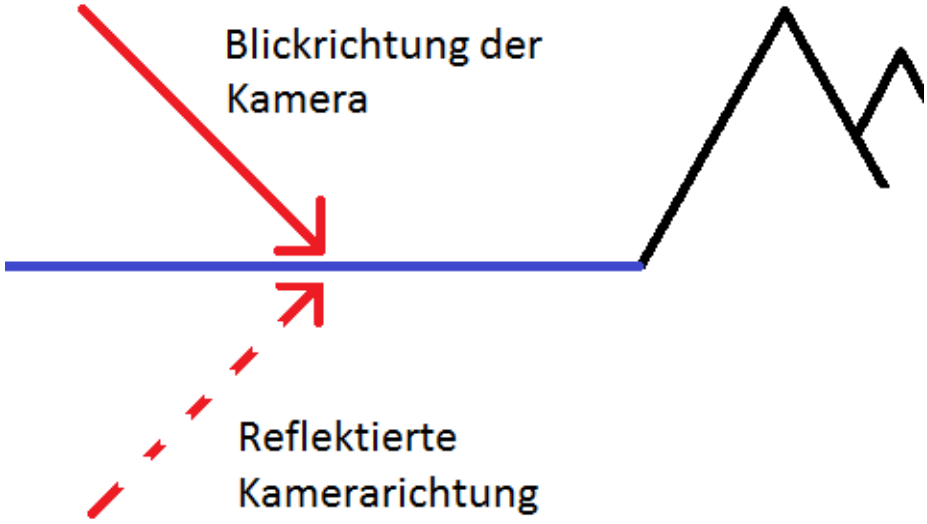


Illustration 7: Reflexion der Kamera

Ein weiterer wichtiger Punkt für die Reflexion ist, dass keine Objekte unterhalb der Wasserebene gerendert werden. Diese Objekte können zum einen in der Reflexion nicht gesehen werden, da man in der Wasserreflexion nur Objekte über der Wasseroberfläche sieht und zum anderen verdecken diese Objekte eventuell die Sicht auf die Objekte über der Wasseroberfläche.

Die praktische Umsetzung der Demo lässt nicht nur alle Objekte unterhalb der Wasseroberfläche aus, sondern schneidet auch alle anderen Objekte unterhalb der Wasseroberfläche ab, wodurch zum Beispiel das Terrain zwar sichtbar ist (da einige Teile oberhalb der Wasseroberfläche liegen), aber alle Pixel unterhalb der Wasseroberfläche nicht gerendert werden.

Auch wenn dieser Teil der Demo kleiner erscheint als der vorherige, wurde hier in etwa gleich viel Zeit benötigt. Gerade die Reflexion der Kamera entlang einer bestimmten Fläche hat aufgrund der mathematischen Operationen im Hintergrund einige Zeit gekostet. Nach erneuter Auffrischung in den Bereichen der Matrizen-Mathematik lies sich aber auch dieses Problem im Endeffekt sehr gut Lösen. Im Nachhinein gesehen wurde relativ viel Zeit verschwendet, da erst darauf los probiert wurde. Hätte man sich zuerst mit den Matrizen Grundlagen auseinandergesetzt, wäre dieser gesamte Schritt innerhalb eines Tages möglich gewesen.

30.02.2015 Rekonstruktion der Position

```
// GBuffer Sample
float Depth = GBuffer_Depth.Sample( gSampler, p_Input.UV ).rgb;

// Position Reconstruction
float depth = ( InvMVP._m23 + ( InvMVP._m22 * Depth ) ) / ( InvMVP._m33 +
InvMVP._m32 * Depth );
float4 _WorldPosition = p_Input.RayDir * depth + CameraPosition;
```

Die im ersten Schritt gespeicherte Tiefeninformation wird hier wieder für jeden einzelnen Pixel geladen und in der Variable „Depth“ gespeichert. Da die Tiefeninformation vorher mithilfe der MVP Matrix transformiert wurde, wird sie hier wieder mithilfe der inversen MVP („InvMVP“) zurück in die normalen Weltkoordinaten transformiert. Diese Matrixmultiplikation ist normalerweise etwas komplizierter, da hier aber nur die Tiefen-Information transformiert werden muss, können 3 von 4 Komponenten weggelassen werden. Dadurch werden nur 2 Multiplikationen, 2 Additionen und eine Division benötigt.

Da die Position der Kamera („CameraPosition“) und die Blickrichtung (p_Input.RayDir) bekannt sind, lässt sich mithilfe der Entfernung des Pixels vom Spieler die genaue Position des Pixels bestimmen mit der Formel Welt-Position = Kamera-Position + Blickrichtung * Tiefen-Information.

Auch in diesem Schritt waren wieder sehr viele komplexe mathematische Operationen von Nöten. Obwohl ich etwas ähnliches bereits in einer anderen Programmier-Sprache umgesetzt hatte, bereitete es mir in c++ mit DirectX 11 erneut große Probleme. Die mathematischen Operationen hängen direkt in Verbindung mit den Einstellungen innerhalb DirectX und teilweise war ich mehrere Tage ohne Fortschritte dabei, den gleichen Fehler zu suchen.

Entsprechend der großen Schwierigkeit war hier aber auch der Lerneffekt groß und das hier umgesetzte System kann ich von nun an für meine weiteren Projekte verwenden.

24.04.2015 Berechnung der Wasserhöhe

Bevor die Position der Wasseroberfläche genau Berechnet wird, wird sie erst mal angenähert durch die Annahme, dass das Wasser eine gerade Fläche ist.

```
float3 CurrentTracingPosition = _WorldPosition.xyz;
float Diff = WaterHeight - CurrentTracingPosition.y;
CurrentTracingPosition += EyeToTarget * Diff / EyeToTarget.y;
```

„CurrentTracingPosition“ soll die Variable werden, welche die Stelle der Wasseroberfläche speichert. Zu Beginn wird diese Variable aber mit der Untergrund-Position initialisiert. Um den Punkt auf der angenäherten Ebene zu finden, wird die Höhendifferenz zu der Fläche berechnet. Da es sich hierbei allerdings um die vertikale Höhe handelt und nicht die Diagonale, muss die Höhe durch den Y-Wert des Blickrichtungs-Vektors geteilt werden. Danach wird dann noch mit dem Richtungsvektor addiert und die neue Position durch Addition der beiden Komponenten berechnet.

```
for ( int x = 0; x < 5; x++ )
{
	float Diff = GetWaterHeightAt( CurrentTracingPosition.xz,
MagnitudeScaling, gTime )
		+ WaterHeight - CurrentTracingPosition.y;
	CurrentTracingPosition += EyeToTarget * Diff / EyeToTarget.y * 0.5f;
}
```

Für die Annäherung an die genauere Wellenform wird der Algorithmus aus dem ersten Schritt in einer Schleife mehrfach ausgeführt. Die Höhe der „Fläche“ wird dieses mal aber aus der Funktion gelesen, welche für jeden Punkt die Höheninformation des Wassers errechnet. Außerdem wird die neu berechnete Position nicht direkt als Ergebnis genommen, sondern die Stelle, die auf der Hälfte zwischen dem neuen Ergebnis und dem Alten liegt wird benutzt, da beide Ergebnisse nur Annäherungen sind und keine Präzisen Berechnungen.

Um die Wasserhöhe an einer bestimmten Position zu berechnen, wurde eine Funktion geschrieben, welche als Parameter die Position und die aktuelle Zeit nimmt, was für die Bestimmung der Wasserhöhe ausreicht. Zusätzlich dazu wurde noch eine WaterMagnitude Variable übergeben, welche als Faktor für die Wellenhöhe dient, um die Wellenhöhe bei sehr flachen Blickwinkeln auf das Wasser zu verringern, sodass das Raytracing keine Artefakte erzeugt.

```
float GetWaterHeightAt(float2 PositionXY, float WaterMagnitude, float Time)
{
	float Height = 0;

	float TexCoordScale = gWaterOffsetBase;
	float HeightScale = 1;
	float2 Variance = float2(1, 1);
	float MaxHeight = 0;

	for (int i = 0; i < 5; i++)
	{
		Height += Water_Height.SampleLevel( gSamplerTex, 0.5*
( PositionXY.rg  * TexCoordScale + Time * gWaterMove1 * Variance ), 1 ).g *
HeightScale;
		Variance.x *= -1;
		MaxHeight += HeightScale;
		TexCoordScale *= gWaterOffsetScaling;
		HeightScale *= gWaterHeightScaling;
	}

	return Height * gWaterBaseHeight * WaterMagnitude / MaxHeight;
}
```

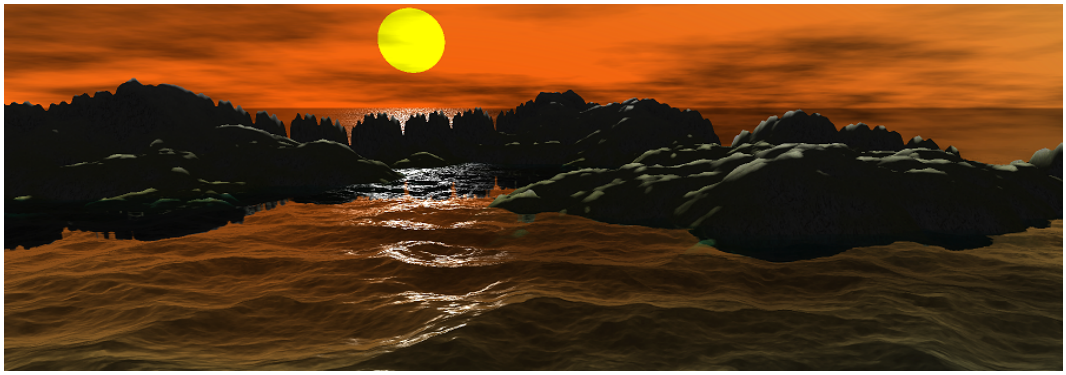
Die Höhe des Wassers an der entsprechenden Stelle wird auf 0 initialisiert. Danach wird fünf mal die Schleife durchlaufen, welche in jedem Durchlauf die Wellentextur ließt und die Höhe der Wellentextur zum Endergebnis hinzufügt. Da nach jedem Durchlauf der Schleife am Ende die HeightScale verringert wird, haben die späteren Durchläufe der Schleife weniger Einfluss auf das Gesamtergebnis. Gleichzeitig wird aber auch die Skalierung der Textur verringert, sodass die späteren Durchläufe zwar sehr kleine Wellen erzeugen, aber auch sehr kurze Wellen, wodurch die kleineren Wellen im Ozean simuliert werden.

Jeder Durchlauf der Schleife ändert außerdem die Richtung, in die sich die Wellen bewegen („Variance.x“), um den Eindruck einer stehenden Welle zu erschaffen. Da die Flussrichtung in jedem Durchlauf gewechselt wird, sind Durchläufe 1, 3 und 5 in die eine Richtung, und die anderen Durchläufe in die entgegengesetzte Richtung.

Am Ende wird das Wasser mit der WaterMagnitude multipliziert, um die Wellen bei flachen Blickwinkeln kleiner zu machen.

Da die Form des Wassers für das letztendliche Aussehen essentiell ist, wurde in diesen Bereich sehr viel Arbeit gesteckt, damit er so gut wie möglich ist. Allerdings fallen auch jetzt noch Defizite bei der Form des Wassers auf. Zum Beispiel flacht das Wasser zum Horizont hin merkbar ab. Durch besseres Ray-tracing in Kombination mit besserer Höhenberechnung lassen sich hier also eventuell noch Fortschritte machen. Gerade dieser Part ist das Kernstück des Image-Effektes und der Unterschied zu allen derzeit verwendeten Verfahren. Daher war es hier nicht möglich sich auf bereits vorhandenes Wissen zu beziehen und alles musste selbst erarbeitet werden. Das kostete zwar eine Menge an Zeit (Insgesamt fast 2 Monate) brachte mir aber auch reichlich Erfahrung im Bereich der Visualisierung durch Ray-tracing. Gerade das direkte herausarbeiten der Vor- und Nachteile von Ray-tracing im Vorhinein hat mir hier wohl viel Zeit erspart.

01.07.2015 Finaler Image-Effekt



Nach der Bestimmung der Wellenform mussten in diesem Schritt im Endeffekt nur noch alle einzelnen Faktoren, die die Farbe von Wasser ausmachen, zusammengeführt werden (genauer beschrieben im Teil der Durchführung). Hier wurde zusätzlich noch etwas an den vorherigen Schritten angepasst, da teilweise einzelne Faktoren im Zusammenspiel mit den anderen etwas abgeschwächt oder verstärkt werden mussten. Dabei wurde sehr viel neues Wissen erlangt im Bereich der Visualisierung von Wasser. Gerade die Erkenntnis, welche Faktoren für das realistische Aussehen von Wasser am Ausschlaggebendsten sind, habe ich sehr schnell erlangt und konnte somit meinen Hauptfokus auf diese Bereiche lenken.

Die Umsetzung des Effektes hat insgesamt zwar nur etwa 2-3 Wochen gedauert, aber darauf folgte etwa ein Monat in dem ich immer wieder einzelne Teile der Demo angepasst habe um das Gesamtergebnis zu verbessern.

Vor allem in diesem Bereich kann noch viel verbessert werden, da viele Effekte, wie zum Beispiel Lichtbrechung in den oberen Wasserschichten (Sub-Surface-Scattering), vollkommen ignoriert wurden aus Gründen der Zeit und ihrer Komplexität.

7.2 Literaturverzeichnis

Akenine-Möller, T & Haines, E & Hoffman, N 2008, *Real-Time Rendering Third Edition*, A K Peters, Natick.

Atwood, J 2008, *Real-Time Raytracing* [Online]
Available at: **http://blog.codinghorror.com/real-time-raytracing/**
[Zugriff am 20 Februar 2015].

Bonaventura, X 2011, *Terrain and Ocean Rendering with Hardware Tessellation*, A K Peters, Natick.

Braun, C & Smirnov, S 1993, *WHY IS WATER BLUE* [Online]
Available at: **http://www.dartmouth.edu/~etrnsfer/water.htm**
[Zugriff am 14 Januar 2015].

Doggett, M 2012, *Texture Caches* [Online]
Available at:
http://fileadmin.cs.lth.se/cs/Personal/Michael_Doggett/pubs/doggett12-tc.pdf
[Zugriff am 08 Juni 2015].

Elert, G 1998, *The Physics Hypertextbook* [Online]
Available at: **http://physics.info/refraction/**
[Zugriff am 14 Januar 2015].

Florida Center for Instructional Technology 2005, *Beach Profiles - Response to Oceanic Conditions* [Online]
Available at:
http://fcit.usf.edu/florida/teacher/science/mod2/beach.profiles.html
[Zugriff am 16 März 2015].

Guardado, J & Sánchez-Crespo, D 2007, *Rendering Water Caustics* [Online]
Available at: **http://http.developer.nvidia.com/GPUGems/gpugems_ch02.html**
[Zugriff am 24 März 2015].

Ki, H 2011, *Multi-Resolutioon Deferred Shading*, Course Technology, Boston.

Lengyel, E 2012, *Mathematics for 3D Game Programming and Computer Graphics Third Edition*, Course Technology, Boston.

McShaffry, M & Graham, D 2013, *Game Coding Complete fourth edition*, Course Technology, Boston.

Helmich, U 2009, *Lineare und binäre Suche* [Online]
Available at: **https://www.coga.tu-berlin.de/fileadmin/i26/download/AG_DiskAlg/FG_KombOptGraphAlg/CoMaWiSe2012/E-Kreide/binaere_suche_26112012.pdf**
[Zugriff am 22 Februar 2015].

7.3 Abbildungsverzeichnis

Alle verwendeten Abbildungen wurden selbst erstellt und erfordern daher keine weitere Quellenbeschreibung

7.4 Digitaler Anhang

Im digitalen Anhang in der Form einer CD befinden sich außerdem noch folgende Daten:

- Die reinen Daten au der Umfrage (Umfrage.csv / Umfrage.xls)

- Das Projekt selbst (Ordner: Wasser Demo)

- Das Demo Video (Showcase.avi)