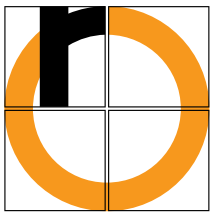
Hochschule **Rosenheim**
University of Applied Sciences



Fakultät für Informatik

Studiengang Informatik

Echtzeitgenerierung künstlicher Landschaften durch Simulation natürlicher Prozesse mit zellulären Automaten

Master Thesis

von

Tobias Gerteis

Datum der Abgabe: 8. März 2017
Erstprüfer: Prof. Dr. Jochen Schmidt
Zweitprüfer: Prof. Dr. Ludwig Frank

Erklärung

Ich versichere, dass ich diese Arbeit selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Rosenheim, den 8. März 2017

Tobias Gerteis

Kurzfassung

In dieser Arbeit wird ein neues Verfahren vorgestellt, um in Echtzeit mit circa 60 Bilder pro Sekunde realistische Höhenkarten zu generieren. Im Gegensatz zu bisherigen Verfahren, welche entweder manuell modelliert oder durch ein mathematisches Verfahren erzeugt werden, wird hierbei eine stark vereinfachte Simulation von realen natürlichen Prozessen verwendet. Simuliert werden Gestein, Sand, Wasser, Wind, Luftfeuchtigkeit, Temperatur und Eis, sowie eine daraus resultierende Klimaverteilung.
Die Simulation selbst basiert auf zellulären Automaten, welche den Vorteil haben, dass ein neues Ergebnis nur aus dem vorhergehenden Ergebnis berechnet wird. Dadurch lässt sich die Simulation leicht parallelisieren. Um die Echtzeitfähigkeit trotz der vielen Simulationen zu erreichen, wird diese mittels OpenGL auf der Grafikkarte berechnet. Das Resultat der Arbeit sind deutlich verbesserte und abwechslungsreiche Höhenkarten. Des Weiteren lässt sich die Software leicht durch weitere Simulationsansätze erweitern. Die aktuellste Version der Software befindet sich unter `https://github.com/tarTG/RTWG`.

Schlagworte: Simulation, zelluläre Automaten, Höhenkarten, OpenGL, Echtzeit

Inhaltsverzeichnis

1 Einführung

In vielen Computerspielen ist ein Element das Erkunden einer bisher noch unbekannten Welt. Um diese Welt zu erstellen, gibt es verschiedene Möglichkeiten, die hier im Folgenden vorgestellt werden.

1.1 Manuell erstellte Welten

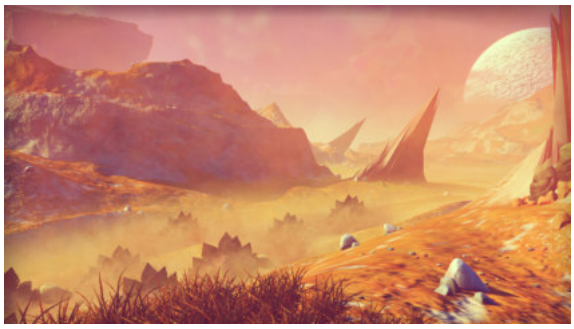
Das gängigste Verfahren, um Welten für Computerspiele zu erstellen, ist, diese von Hand zu modellieren. Der Vorteil hierbei ist, dass die Welt detailliert an die Vorstellungen der Designer angepasst werden kann. Je umfangreicher eine Welt gestaltet wird, desto mehr Aufwand muss investiert werden. Professionelle Entwicklungsumgebungen besitzen entsprechende Tools, die bei der Erstellung der Karten die Designer unterstützen [Eng]. Von Nachteil ist, dass alle Anpassungen manuell erfolgen müssen, was meist mit großen Zeitaufwand verbunden ist.

1.2 Weltengeneratoren

Um möglichst viele Karten in einem Spiel anzubieten, ohne diese manuell entwerfen zu müssen, gibt es die Möglichkeit, durch einen Algorithmus eine Spielwelt zu generieren. Diese Methode ist vor allem in Strategiespielen weit verbreitet, um bei jedem Spielstart eine neue Welt bereitzustellen. Dem Algorithmus werden entsprechende Regeln vorgelegt, damit nicht eine völlig zufällige, sondern eine faire und natürlich wirkende Spielwelt erzeugt wird. Dieses Verfahren kann auch zur Laufzeit neue Landschaften erzeugen und erweitern. Von Nachteil ist, dass sich Landschaften hierdurch immer ähneln und Spielern wenig Abwechslung bieten. In Abbildung 1.1a ist ein Beispiel aus dem Spiel *No Man Sky* aus dem Jahr 2016 zu sehen. Das Spiel generiert zur Laufzeit unterschiedliche Planeten inklusive Flora und Fauna. Jedoch ist die mangelnde Vielfalt schnell erkennbar [Lin16]. Ein zweites Beispiel ist in Abbildung 1.1b aus dem Spiel *Minecraft* zu sehen. Auch hier wird bei jedem neuen Spielstart ein andere Welt nach bestimmten Regeln erstellt.

1.3 Landschaftsveränderungen als Spielelement

In einigen Computerspielen wird das Manipulieren der Umgebung als aktives Spielelement genutzt. Eines der ältesten Beispiele hierfür ist *SidMaiers SimEarth* (Abbildung 1.2a). Hierbei werden durch die Manipulation von Umgebungsvariablen, wie der Sonneneinstrahlung oder der Anzahl von Vulkanausbrüchen, globale Veränderungen simuliert. Durch relativ simple Berechnungen werden hierbei verschiedene geologische, biologische und evolutionäre Mechaniken nachgestellt. Ein weiteres Beispiel ist *UbiSofts From Dust* (Abbildung 1.2b), bei dem durch direkte Manipulation von Lava, Wasser und Sand die Spielwelt verändert werden muss, um ein bestimmtes Ziel zu erreichen. Vor allem im Bereich Wassererosionen



(a) Beispiel aus No Man Sky.
Quelle: www.no-mans-sky.com

(b) Beispiel aus Minecraft
Quelle: www.wallpapercave.com

Abbildung 1.1 Beispiele für Fraktale Weltengeneratoren

(a) Beispiel aus Maxis SimEarth
Quelle: ©Wikimedia Commons / Fair Use

(b) Beispiel aus From Dust
Quelle: ubisoft.com

Abbildung 1.2 Beispiele für Landschaftsveränderungen in Spielen

und thermische Erosionen wurden einige Forschungsarbeiten angefertigt [MDH07], [JT11], [GGG+13], um die Effekte in Echtzeit zu erzeugen.

1.4 Ziel der Arbeit

Da die Welten für Computerspiele immer größer und umfangreicher werden, ist dies immer mit einem deutlich größeren Arbeitsaufwand verbunden. Daher entstand in dieser Arbeit die Idee, mittels vereinfachter Simulationen, wie sie in Computerspielen eingesetzt werden, erdähnlich Landschaften zu generieren und zur Laufzeit anpassbare Parameter bereitzustellen, um die Möglichkeit zu geben, die Welt anzupassen. Um die Effekte der Parameteranpassung schnell sichtbar zu machen, soll die Simulation in Echtzeit mit 30 bis 60 Aktualisierungen pro Sekunde ablaufen. Die Simulation wurde hierbei von *SidMaiers SimEarth* inspiriert, welche zelluläre Automaten für die Simulation verwenden [Oni09]. Dadurch wurde das Ziel dieser Arbeit formuliert: Echtzeitgenerierung künstlicher Landschaften durch Simulation natürlicher Prozesse mit zellulären Automaten.

2 Grundlagen

2.1 Höhenkarten

Höhenkarten sind zweidimensionale Grauwerttexturen, welche die Struktur einer Landschaft aus der Draufsicht beschreiben. Hierbei geben die Grauwerte an, wie hoch ein Punkt in der Landschaft ist. Je heller ein Punkt ist, desto höher liegt dieser. Der Vorteil dieser Technik ist, dass mit sehr geringem Speicheraufwand von einem Wert pro Pixel auch große Landschaftsprofile gespeichert werden können. Ein großer Nachteil von Höhenkarten ist, dass durch die Draufsicht keine Überhänge oder Höhlen abgebildet werden können. In Abbildung 2.1 ist eine Höhenkarte von Irland zu sehen. Hierbei erkennt man gut, wo sich Gebirgszüge, Küsten und Flusstäler befinden. Höhenkarten sind vor allem in Computerspielen aufgrund des geringen Speicherbedarfs weit verbreitet. Mittels *Displacement Mapping* [AMHH08] kann die Höhenkarte als dreidimensionale Landschaft gezeichnet werden. Fehlende Datenpunkte können leicht durch Interpolation der umgebenden Werte generiert werden.

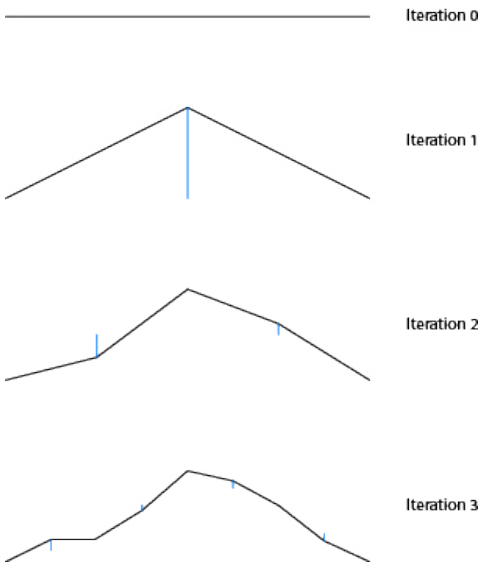
2.2 Algorithmen für die Erzeugung von Höhenkarten

Neben dem Erstellen der Höhenkarten aus Satellitendaten und dem manuellen Zeichnen hat sich schon relativ früh das Generieren von Höhenkarten durch Algorithmen entwickelt [Man77]. Hierbei spielen vor allem Fraktale und Selbstähnlichkeit eine wichtige Rolle, da diese mit simplen rekursiven Algorithmen Strukturen erzeugen können, welche Landschaften ähneln. In dieser Arbeit wird für die Ausgangslage (Unterabschnitt 3.2.2) und für die Windsimulation (Unterabschnitt 3.5.4) der Diamond-Square-Algorithmus [Mil86] verwendet. Dieser bietet in kurzer Laufzeit eine parametrierbare Landschaft.
Im zweidimensionalen Fall startet das Verfahren mit einer Geraden. Diese wird am Mittelpunkt um einem zufälligen Wert zwischen -1 und 1 in y-Richtung verschoben und die Linien mitgezogen. Die beiden neu entstehenden Geraden zwischen den Rändern und dem Mittelpunkt werden erneut am Mittelpunkt zufällig verschoben, jedoch werden die Maximalwerte um einen parametrierbaren Faktor reduziert. Dieser Vorgang wird so oft wiederholt, bis ein ansprechendes Ergebnis vorhanden ist. Ein Beispiel hierfür ist in Abbildung 2.2 zu sehen. Dieses Verfahren nennt sich Midpoint-Displacement.

Für die Generierung von dreidimensionalen Strukturen muss der Algorithmus anpasst werden. Zuerst erhöht man die vier Ecken der noch flachen Landschaft um einen zufälligen Wert. Anschließend wird der Mittelpunkt zwischen den vier Ecken bestimmt, durch den Schnittpunkt der Diagonalen. Dieser Punkt wird um den Mittelwert der vier Eckpunkte plus einen zufälligen Wert geändert. Diesen Teil des Algorithmus nennt man den Diamantschritt. Nun bilden sich aus den Eckpunkten und dem Mittelpunkt vier neue Rechtecke, indem die neuen Punkte mittig zwischen den Eckpunkten hinzugenommen werden. Diese neuen Punkte werden ebenfalls um den Mittelwert der Eckpunkte sowie den vorherigen Mittelpunkt

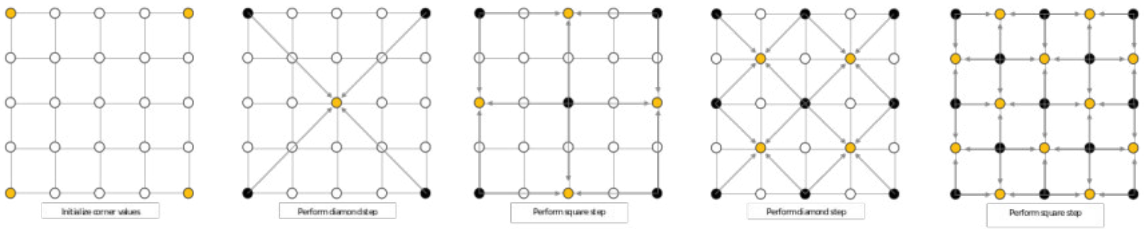
Quelle: imgur.com/qr7YY

Abbildung 2.1 Eine Höhenkarte von Irland.



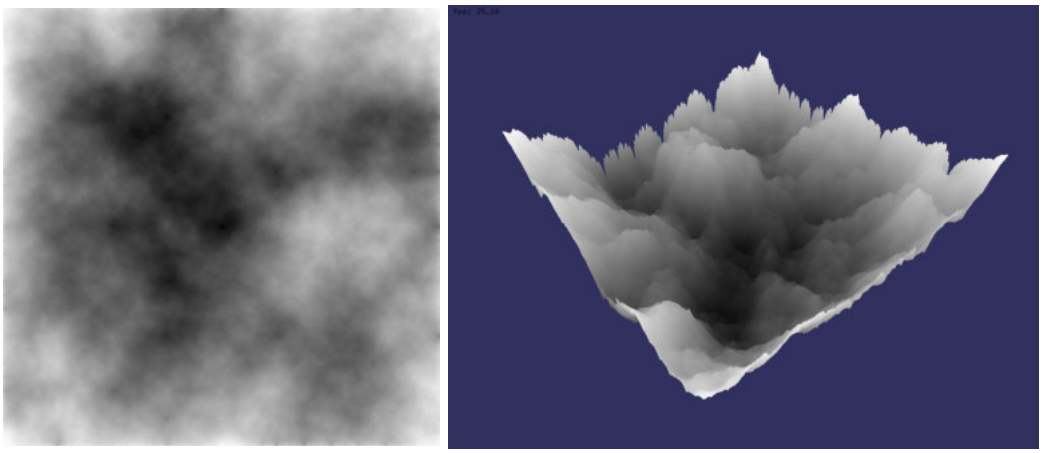
Quelle: [Asc14]

Abbildung 2.2 Midpoint-Displacement Algorithmus



Quelle: ©Christopher Ewin / Wikimedia Commons / CC BY-SA 4.0

Abbildung 2.3 Diamond-Square Algorithmus



(a) Ergebnis in 2D **(b)** Ergebnis in 3D

Quelle: [Bea10]

Abbildung 2.4 Diamond-Square Ergebnis

plus einen zufälligen Wert geändert. Dies ist der Viereckschritt. Diese beiden Schritte werden so lange wiederholt, bis alle Punkte der Textur einen Wert haben (siehe Abbildung 2.3). Nach jedem Schritt wird, wie zuvor bei Midpoint-Placement, der Maximalwert des Zufalls um einen parametrierbaren Faktor verringert. Das Ergebnis hängt hierbei von dem reduzierenden Faktor ab. Ist der Wert groß, verringern sich die Abweichungen schnell und es entsteht ein sehr flach ansteigendes Gelände. Ist der Wert gering, werden die Geländeänderungen steiler. Bei einem Wert von Null ist das Ergebnis ein Zufallsrauschen. In Abbildung 2.4 ist ein mögliches Ergebnis sowie eine 3D-Repräsentation für diesen Algorithmus zu sehen.

2.3 Zelluläre Automaten

Zelluläre Automaten wurden bereits 1951 von John von Neumann dazu verwendet, komplexe, selbst reproduzierende Probleme zu simulieren [von51]. Diese bestehen aus meist quadratisch aneinander liegenden Feldern, folgend Zellen genannt. Der Zustand einer Zelle im folgenden Simulationsschritt ist abhängig von seinem aktuellen eigenen Zustand, sowie der Zustand der umgebenden Zellen. Hierbei unterscheidet man zwischen der *Moor Nachbarschaft*, welche alle acht umgebenden Zellen berücksichtigt und der *von Neumann Nachbarschaft*, welche nur die vier direkt anliegenden Zellen betrachtet (siehe Abbildung 2.5). Die Regeln für den zellulären Automaten, welche die Kriterien für den nächsten Zustand

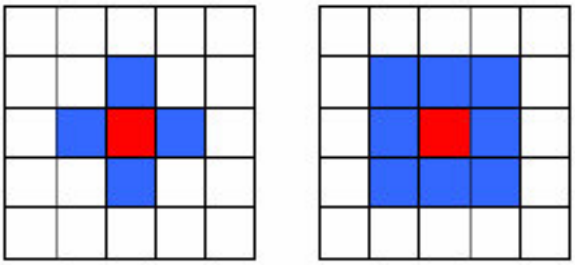


Abbildung 2.5 von Neumann (links) und Moor Nachbarschaft

bestimmen, variieren je nach Anwendungsfall. Die Zustände sind in den meisten Automaten binär, jedoch ist dies kein zwingendes Kriterium. Betrachtet müssen auch die Randbedingungen, welche festlegen, wie sich die Zellen an den Rändern verhalten, da diese keine vollständige Nachbarschaft beinhalten.
Zelluläre Automaten wurden schon für die Simulation verschiedenster Probleme, wie Flächenbrandentwicklung [TDR+11] oder Wolkendynamik [SG10] eingesetzt, unter anderem auch in *SidMaiers SimEarth* [Oni09]. Der bekannteste Vertreter von zellulären Automaten ist *Conways Spiel des Lebens*[1].

2.4 Tiefpassfilter

Der Tiefpassfilter ist ein Verfahren aus der Bildverarbeitung, um in einem Bild hohe Frequenzen zu entfernen. Das Bild wird dadurch weich gezeichnet und hat weniger abrupte Farbübergänge, wie in Abbildung 2.6 zu sehen ist. Hierfür werden, von einem Pixel ausgehend, alle umgebenden Pixel in einer Filtermaske verrechnet, um einen neuen Farbwert zu berechnen. Wie viele umgebende Pixel mit eingerechnet werden, ist vom Verfahren abhängig. Die zwei meist verwendeten Filtermasken sind der Mittelwertfilter, bei dem alle einbezogenen Werte gemittelt werden, und der Gaußfilter, bei dem eine Gewichtung entsprechend der Gaußverteilung angewendet wird.
In dieser Arbeit wird nur der 3x3 Mittelwertfilter, welcher alle direkt umgebenden Pixel verwendet. Für ein bestimmtes Pixel x ist der nachfolgende Wert wie folgt zu berechnen.

$$x = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.1}$$

Hierbei ist n die Anzahl der berücksichtigten Pixel.

1 de.wikipedia.org/wiki/Conways_Spiel_des_Lebens

Quelle: [mil]

Abbildung 2.6 Beispiel für einen 3x3 Mittelwertfilter. Links das ursprüngliche Bild. Rechts das gefilterte Bild

3 Simulation

3.1 Funktionsweise

Die Simulation besteht aus sieben Teilen: Gestein, Sand, Wasser, Wind, Luftfeuchtigkeit, Temperatur und Eis, sowie den daraus resultierenden Klimazonen. Diese Teile werden hier als Simulationsebenen bezeichnet. Zu Beginn wird für jede Ebene eine Ausgangslage generiert, die, außer beim Gestein, alle Werte auf Null setzt. In einem Simulationsschritt werden alle Simulationsebenen einmal aktualisiert. Für die Berechnung eines Simulationsschrittes werden das Ergebnis des vorgehenden Simulationsschrittes sowie zusätzliche Parameter verwendet. Da sich einige Ebenen auch untereinander beeinflussen, greifen diese auch auf die Daten aus anderen Ebenen zu. Aus dem Resultat eines Simulationsschrittes werden anschließend noch die Klimazonen generiert (siehe Abbildung 3.1). Da die Simulation zelluläre Automaten verwendet, müssen die Zellen klar definiert sein. Hierbei wird eine Bildgröße in Pixeln angegeben, wobei jedes Pixel in dem Bild einer Zelle entspricht. Die Einstellung der Parameter, mit welchen jede Simulationsebene angepasst werden kann, war eine der schwierigsten Aufgaben in dieser Arbeit. Da die Simulation etwa 30 bis 60 Mal in der Sekunde aktualisiert wird, bewirken zu große Werte, dass die Auswirkungen extrem schnell ablaufen und unkontrollierbar sind. Bei zu kleinen Werten dauert es hingegen sehr lange, bis die Auswirkungen beobachtbar sind. Die im finalen Programm verwendeten Parametergrenzen wurden durch viel Ausprobieren und Testen festgelegt.
Grundsätzlich ist die Simulation so aufgebaut, dass diese eine erdähnliche Welt generiert. Die oberen und unteren Bereiche der Textur sind als Polregionen zu interpretieren; die Mitte als Äquator. Durch Parameter können diese Bereiche verschoben, jedoch nicht grundsätzlich verändert werden.

3.1.1 Anpassung der zellulären Automaten

Da in dieser Arbeit zelluläre Automaten dazu verwendet werden, physikalische Prozesse auf globaler Ebene vereinfacht zu simulieren, wird das Verfahren hierfür erweitert. Um die Echtzeitfähigkeit zu erreichen, wird hier die von Neumann Nachbarschaft verwendet, da hierbei nur vier umgebende Zellen mit berechnet werden müssen. Die erste Erweiterung ist, dass für jede Simulationsebene ein eigener Automat mit eigenen Regeln definiert wird. Eine weitere Anpassung ist, dass eine Zelle nicht nur einen einzigen Wert, sondern bis zu drei Werte enthalten kann. Dadurch können auf einer Simulationsebene mehrere Daten gespeichert und verwendet werden. Um feinere Unterschiede zu repräsentieren, sind die Werte der Zellen nicht binär, sondern Gleitkommazahlen. Dies ermöglicht nicht nur, dass die konkreten Werte genauer angegeben, sondern auch Richtungsvektoren in negative Richtung gespeichert werden können. Standardmäßig wird in einem zellulären Automaten ausschließlich eine Ausgangslage definiert, bevor die Simulation startet. In dieser Arbeit wird zusätzlichen Input während der Simulation generiert, um auch Effekte wie zum Beispiel Sonneneinstrahlung darstellen zu können, welche in jedem Simulationsschritt auf die Zellen wirkt. Die Simulation wird an den Rändern umgebrochen, damit diese die Werte auf der

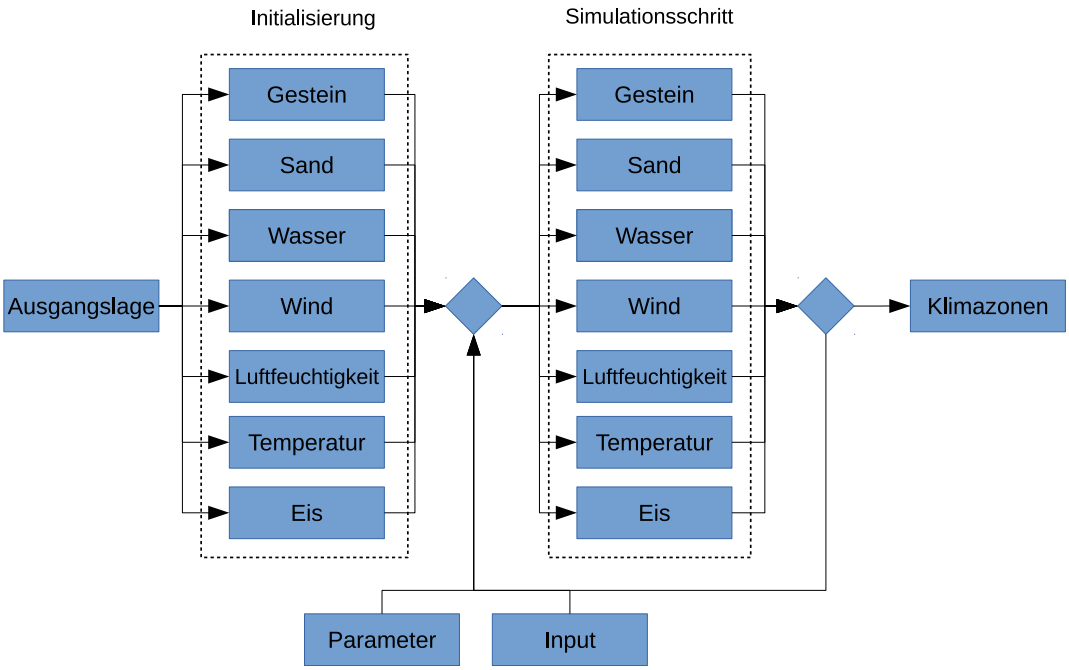


Abbildung 3.1 Ablaufdiagram der Simulation

gegenüberliegenden Seite verwendet. Dies ist vor allem für den Umbruch von unten und oben sehr unrealistisch für eine Weltsimulation, jedoch würde hier eine Sonderbehandlung deutlich mehr Rechenaufwand bedeuten. Mit Rücksicht auf die Echtzeitfähigkeit wird dieser Umstand hier vernachlässigt.

3.1.2 Mathematische Formulierung

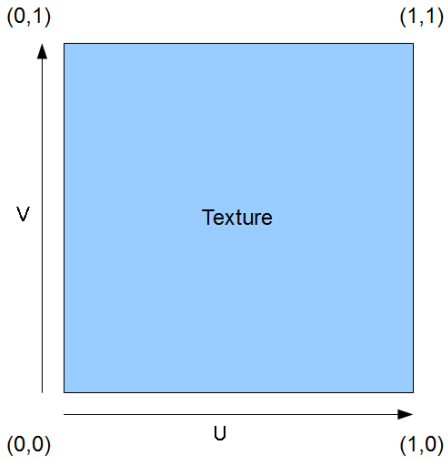
Um die Verfahren mathematisch zu beschreiben können, werden die Funktionen wie folgt dargestellt.

Variablennamen

Die Simulationsebenen beeinflussen sich oftmals untereinander. Um klarzustellen, aus welcher Ebene der verwendete Wert stammt, wird ein entsprechender Bezeichner in Klammern hinter die Variable gesetzt, z.B. $x_t(rock)$ ist der x-Wert in der Gesteinssimulationsebene. In Tabelle 3.1 ist beschrieben, welcher Bezeichner welche Ebene bezeichnet. Werte, die in einem Simulationsschritt gespeichert werden, werden mit x, y und z entsprechend dem ersten, zweiten und dritten Wert in der RGB-Textur benannt. Die Bedeutung dieser Werte wird zu Beginn jedes Kapitels erläutert.
Parameter, welche global für die Simulation gelten, werden mit p bezeichnet. Diese werden durch einen Namen unterschieden, der tiefgestellt unter dem p steht. Da die Werte global gelten, ist dies kein Konflikt zur nachfolgenden Schreibweise der Simulationsschritte.

Tabelle 3.1 Schlüsselwörter für die Simulationsebenen

Ebene	Bezeichner
Gestein	rock
Wasser	water
Sand	sand
Wind	wind
Temperatur	temp
Luftfeuchtigkeit	moist
Eis	ice



Quelle: http://www.learnopengles.com/

Abbildung 3.2 Textur Koordinaten System

Funktionsnamen

Um mehrere Funktionen innerhalb einer Simulationsebene unterscheiden zu können, werden diese mit f gekennzeichnet und mittels einem Namen beschrieben, der tiefgestellt ist. In einigen Fällen ist die Funktion abhängig von der Texturkoordinate. Dies wird gekennzeichnet durch (u, v) dargestellt, wobei das Koordinatensystem in der linken unteren Ecke mit $(0, 0)$ beginnt und in der rechten oberen Ecke mit $(1, 1)$ endet (siehe Abbildung 3.2).

Simulationsschritte

Da die Simulation nicht mit realen Zeitabschnitten sondern in abstrakten Schritten rechnet, wird der aktuelle Simulationsschritt als t und der nachfolgende als $t + 1$ bezeichnet. Wenn ein nächster Wert exakt dem aktuellen Wert entsprechen soll, lautet die Formel:

$$x_{t+1} = x_t \tag{3.1}$$

Tiefpassfilter

Ein mit einem 3x3 Mittelwertfilter (Abschnitt 2.4) verwendeter Wert aus dem vorherigen Simulationsschritt wird mit einem Strich über der Variable gekennzeichnet. Sprich wenn der

Wert x_t für die Berechnung von x_{t+1} gefiltert wird, sieht die Formel wie folgt aus.

$$x_{t+1} = \bar{x}_t \tag{3.2}$$

Zugriff im zellulären Automaten

Für die Zugriffe im zellulären Automaten wird normalerweise eine Koordinatenbeschreibung angegeben. Da in dieser Arbeit nur die von-Neuman Nachbarschaft (siehe Abschnitt 2.3) betrachtet wird, kann die aktuell betrachtete Zelle anstatt mit $x(u, v)$ nur als x bezeichnet. Die umliegenden Zellen werden mit hochgestelltem Buchstaben unterschieden. Diese lauten x^T, x^B, x^L, x^R für top, bottom, left und right. Wenn zum Beispiel ein Wert x aus der Summe aller umgebenden Werten berechnet wird, wird es folgendermaßen dargestellt.

	x^T	
x^L	x	x^R
	x^B	

$$x_{t+1} = \sum_{i=L,B,T,R} (x_t^i) \tag{3.3}$$

3.2 Gestein und Plattentektonik

Die erste Simulationsebene stellt festes Gestein bzw. den Erdboden dar. Dieser ist der thermischen Erosion $f_{thermerosion}$ und der Wassererosion $f_{watererosion}$ ausgesetzt. Außerdem verändert sich dieser durch tektonische Bewegungen. Das Ergebnis hierbei ist die Höhenkarte der Landschaft, weswegen auch die Werte nicht kleiner als Null werden dürfen, da diese sonst nicht mehr darstellbar wären. Das Gestein wird hierbei stark vereinfacht als eine aus einer Substanz bestehenden Schicht betrachtet. Auch die Veränderung durch Vulkane, Meteoriteneinschläge und biologische Änderungen werden in dieser Arbeit nicht berücksichtigt.

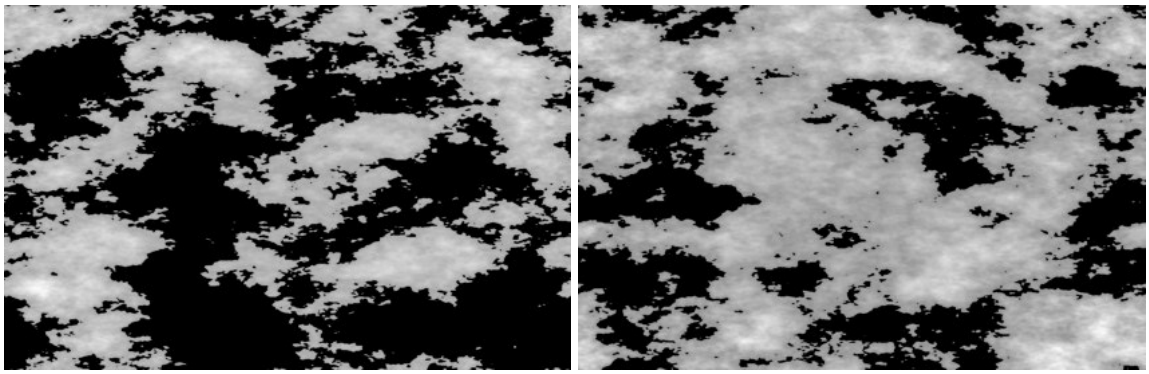
$$x_{t+1}(rock) = \max(0.0, x_t(rock) + f_{thermerosion} + f_{watererosion}) \tag{3.4}$$

3.2.1 Datenformat

Da das Ergebnis eine Höhenkarte ist, besitzt die Textur nur einen Wert: den Höhenwert der Landschaft $x(rock)$ (siehe Abschnitt 2.1). Der Wertebereich ist zwischen Null, was den tiefsten Punkt an einem Meeresboden beschreibt, und dem Maximalwert der Gleitkommazahl. Um die Textur speichern zu können, werden die Werte beim exportieren normiert.

3.2.2 Ausgangslage

Beim Start der Simulation wird mittels Diamond-Squares-Algorithmus (siehe Abschnitt 2.2) eine Landschaft generiert. Um für die Plattentektonik (Unterabschnitt 3.2.4) zwischen kontinentaler und ozeanischer Erdkruste unterscheiden zu können, wird bei der Generierung ein Faktor für den Prozentanteil der Meeresoberfläche mitgegeben. Alle kleinsten Werte,



(a) 66 Prozent Meeresboden **(b)** 30 Prozent Meeresboden

Abbildung 3.3 Startlandschaft mit verschiedenen Ozeananteilen

die sich unter diesem Prozentsatz finden, werden auf den Wert Null gesetzt und somit zu Meeresboden. Um den Umbruch an den Texturrändern in der Simulation zu entsprechen, ist die Landschaft an den Rändern so gestaltet, dass diese nahtlos übergeht. In Abbildung 3.3 sind zwei Startlandschaften mit unterschiedlich großem Meeresanteil zu sehen.

3.2.3 Erosion

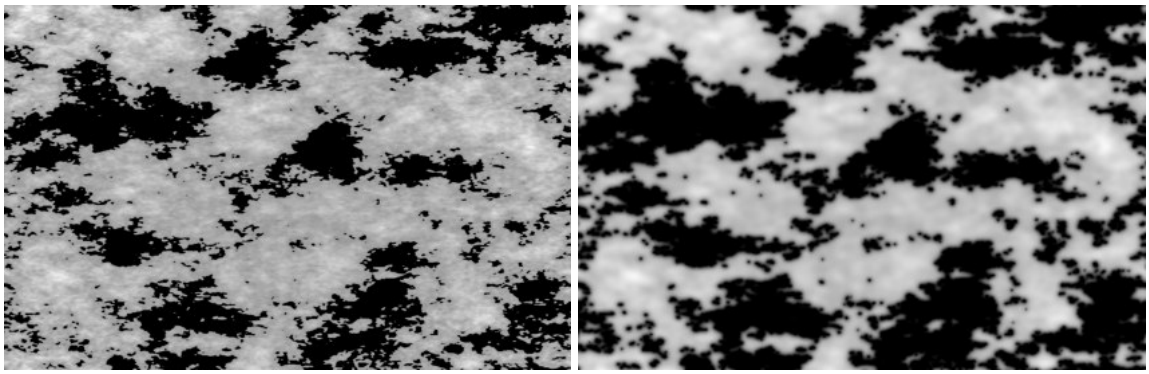
Thermische Erosion

Die thermische Erosion ist ein Verwitterungsprozess, bei dem Gestein durch wechselnde Temperatureinflüsse mit der Zeit zerbröckelt. Dieser Effekt entsteht, da bei Temperaturänderungen sich auch das Volumen des Gesteins dehnt und dadurch Spannungen in dem Gestein entstehen. Makroskopisch gesehen erodieren Klippen und steile Übergange langsam ab. Dieser Effekt wird hier mittels eines Tiefpassfilters (siehe Abschnitt 2.4) realisiert. Falls kleine Übergänge oder hohe Spitzen erhalten bleiben sollen, enthält die Simulation einen minimalen $p_{mintherm}$ und einen maximalen Höhenunterschied $p_{maxtherm}$, welche zur Laufzeit änderbar sind. Alle Höhenunterschiede, die sich außerhalb dieser Grenzwerte befinden, werden vom Filter nicht berücksichtigt. Da thermische Erosion nicht unter Wasser stattfindet, werden die Werte, welche von Wasser bedeckt sind, ebenfalls ignoriert. In Abbildung 3.4 ist gut zu sehen, wie die zunächst fein strukturierte Landschaft nach der thermischen Erosion weniger Strukturen aufweist.

$$f_{thermerrosion} = \sum_{i=L,R,T,B} \begin{cases} \bar{x}_t^i(rock) \cdot p_{thermfactor} & p_{mintherm} < x_t^i(rock) < p_{maxtherm} \\ & \cap\, x_t^i(water) < 0.05 \\ x_t^i(rock) \cdot p_{thermfactor} & sonst \end{cases} \tag{3.5}$$

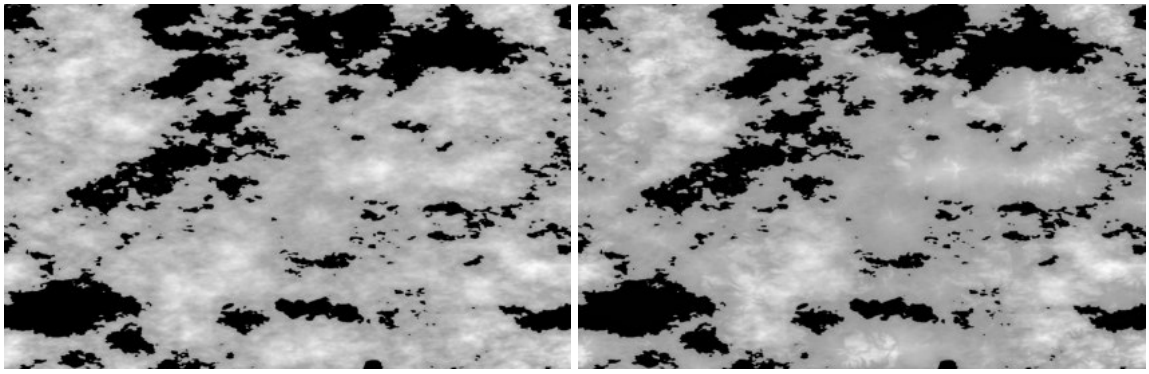
Wassererosion

Das durch Wasser erodierte Gestein wird zu Sand umgewandelt. Die Berechnung hierfür findet in der Sandsimulation statt (siehe Abschnitt 3.4). Diese speichert in $y_t(sand)$, wie viel Gestein abgetragen wurde. In der Gesteinssimulation wird dieser Wert von der aktuellen



(a) Ausgangslage **(b)** Nach der thermischen Erosion

Abbildung 3.4 Beispiel der thermischen Erosion



(a) Ausgangslage **(b)** Nach der Wassererosion

Abbildung 3.5 Beispiel der Wassererosion

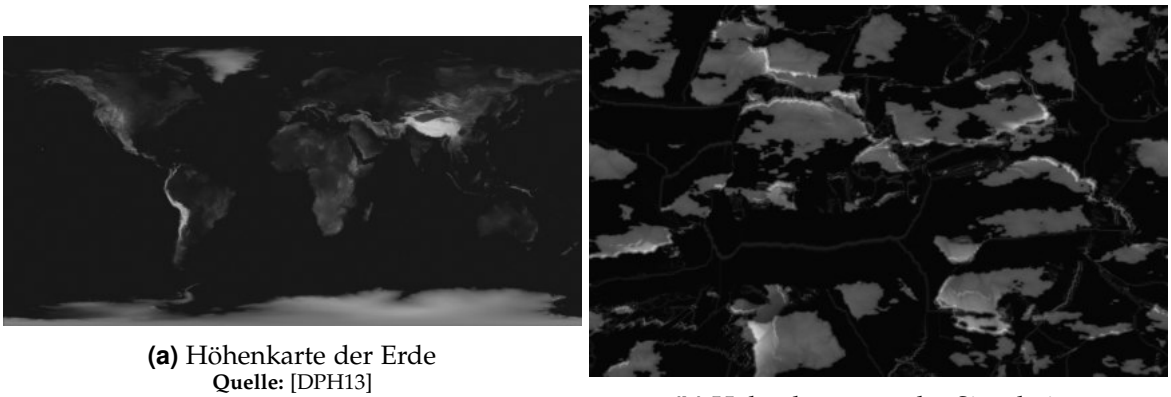
Gesteinshöhe $x_t(rock)$ subtrahiert. In Abbildung 3.5b ist mittig unten gut zu erkennen, wie sich Flussläufe durch das Gestein gegraben haben.

$$f_{watererrosion} = -y_t(soil) \tag{3.6}$$

3.2.4 Plattentektonik

Um Gebirgszüge zu generieren, enthält diese Arbeit eine Plattentektoniksimulation. Dafür wird die externe Bibliothek platec[1] verwendet. Details hierfür können in [Vii12] nachgelesen werden. Die Bibliothek wurde für die Arbeit erweitert, da diese ausschließlich die Plattentektonik simuliert. Hinzugefügt wurde die Möglichkeit, eigene Erosionsmodelle durch Übergabe einer aktualisierten Höhenkarte zu verwenden. Außerdem können die Parameter, z.B. ab wie viel Prozent Überschneidung zwei Platten zu einer Platte werden, nun während der Laufzeit manipuliert werden. Die Wassersimulation wird, solange die Plattentektonik aktiv ist, ausgeschaltet, da die Fließgeschwindigkeit mit der Bewegung der Platten nicht mithalten kann und dadurch Artefakte entstehen. In Abbildung 3.6b kann man gut erkennen, dass die Bibliothek sowohl Gebirgszüge als auch Inseln generieren kann. Dieser Mechanis-

1 https://github.com/tarTG/RTplatec



(a) Höhenkarte der Erde
Quelle: [DPH13]

(b) Höhenkarte aus der Simulation

Abbildung 3.6 Vergleich der Höhenkarten

mus ist separat und wird nicht mit zellulären Automaten berechnet, da es in dieser Arbeit nicht gelungen ist, damit ein zufriedenstellendes Ergebnis zu erreichen.

3.2.5 Resultat

Das Ergebnis der Gesteinssimulation ist trotz der simplen Ansätze eine abwechslungsreiche Landschaft mit Bergketten, Flusstälern und Hügellandschaften. Verglichen mit der Weltkarte in Abbildung 3.6a sind bereits einige Ähnlichkeiten sichtbar. Vor allem durch den Einsatz der Plattentektonik, die in der Generierung von Landschaften bisher kaum Verwendung findet, wirkt die Karte natürlicher. Um noch bessere Landschaften zu erreichen, müssen zusätzliche Effekte wie Vulkanismus oder Meteoriteneinschläge simuliert werden.

3.3 Wasser

Die Oberfläche der Erde wird maßgeblich durch die Verteilung und die Erosion von Wasser gestaltet. Neben der geografischen Unterteilung der Welt durch Ozean, Seen und Flüssen sind auch deren langfristigen Prägungen wichtig. Ehemalige Flussbette und Seen sind als tiefe Furchen in der Landschaft zu erkennen und oft als sandreiche Wüsten zurückgeblieben. Aus diesem Grund gibt es bereits eine Vielzahl an Arbeiten, die sich mit der Erosion von Landschaften durch Wasser beschäftigt haben [MDH07], [JT11], [GGG+13]. In dieser Arbeit wird auf das in [Boe11] entwickelte Verfahren zurückgegriffen, jedoch mit Rücksicht auf die Echtzeitfähigkeit an einigen Stellen vereinfacht.

3.3.1 Datenformat

Das Datenformat für die Wassersimulation hat drei Werte. Der erste $x(water)$ gibt den aktuellen Wasserstand an und kann Werte zwischen Null und dem maximalen Gleitkommazahlenwert annehmen. Der zweite $y(rock)$ und dritte Wert $z(rock)$ sind die Fließgeschwindigkeit in x- und y-Richtung. Der Wertebereich ist hier zwischen -1 und 1. Der Zweck der Begrenzung ist eine Vermeidung einer zu schnellen Erosion, die von der Fließgeschwindigkeit abhängt. Für eine bessere Sichtbarkeit in den folgenden Darstellungen werden die Werte für die Fließgeschwindigkeit normiert. Rot stellt den Wasserstand dar, Grün die Fließrichtung in x-Richtung und Blau die Fließrichtung in y-Richtung.

3.3.2 Ausgangslage

Zu Beginn der Wassersimulation wird eine Meereshöhe festgelegt. Die Differenz zwischen dem Gestein $x(rock)$ und der Meereshöhe $p_{sealevel}$ ist der Startwert für den Wasserstand. Die Fließgeschwindigkeiten sind vorerst alle auf Null.

$$x(water) = \max(0.0, p_{sealevel} - x(rock)) \tag{3.7}$$

3.3.3 Wasserstandänderungen

Die Änderung des Wasserstandes wird durch vier Funktionen beeinflusst: dem Regen f_{rain}, der Verdunstung $f_{evaporate}$, dem Schmelzwasser f_{melt} und durch Wassertransport $f_{transport}$. Limitiert wird die Höhe durch die Meeresspiegelhöhe $p_{sealevel}$. Nicht betrachtet werden in dieser Arbeit Versickerung, unterirdische Wasserverschiebungen und globale Meeresströmungen. Der Höhenstand wird zusätzlich noch gefiltert, um zu abrupte Höhenänderung im Wasser zu vermeiden.

$$x_{t+1}(water) = \max(p_{sealevel} - x_t(rock) - x_t(sand), \bar{x}_t + f_{evaporate} + f_{rain} + f_{melt} + f_{transport})$$

Regen

Der Regen ist ein Effekt, der den Wasserstand an einer bestimmten Stelle erhöht. Wie viel neues Wasser durch Regen generiert wird, hängt von zwei Parametern ab. Der Erste ist die aktuelle Luftfeuchtigkeit $x_t(moist)$ (siehe Abschnitt 3.7). Je höher die Luftfeuchtigkeit, desto mehr Regen wird erzeugt. Ein zusätzlicher Parameter p_{rain} bestimmt die Gewichtung der Luftfeuchtigkeit.

$$f_{rain} = x_t(moist) \cdot p_{rain}$$

Verdunstung

Der entgegengesetzte Effekt zum Regen ist die Verdunstung $f_{evaporate}$. Die Stärke der Verdunstung hängt von der aktuellen Temperatur $x_t(temp)$ ab (siehe Abschnitt 3.6). Auch hier gilt, je höher die Temperatur, desto größer ist die Verdunstung. Es wird hier ebenfalls ein zusätzlicher Faktor $p_{evaporate}$ für die Einflussstärke der Temperatur verwendet.

$$f_{evaporate} = x_t(temp) \cdot p_{evaporate} \cdot -1$$

Schmelzwasser

Der Wasserstand erhöht sich auch durch das Schmelzen von Eis f_{melt}. Aus der Eissimulation entsteht der Wert $y_t(ice)$, um den sich die Eismenge reduziert hat (siehe Abschnitt 3.8). Diesen Wert wird mit einem anpassbaren Faktor p_{melt} multipliziert, um den neuen Wasserstand zu berechnen.

$$f_{melt} = y_t(ice) \cdot p_{melt}$$

Wassertransport

Der Transport von Wasser orientiert sich ausschließlich an Höhenunterschiede. Die Fließgeschwindigkeit hat hier keinen Einfluss. Betrachtet werden die Summe der Höhen von Gestein $x_t(rock)$, Sand $x_t(sand)$ und Wasser $x_t(water)$. Ist die Differenz zwischen der aktuellen Zelle und einer benachbarten Zelle Null, ändert sich der Wasserstand nicht. Ansonsten wird die Hälfte der Differenz bei der Höheren subtrahiert und bei der niedrigeren Zelle addiert. Da alle umgebenden Zellen gleich gewichtet werden, wird die Summe aller Höhenunterschiede durch vier geteilt. In der Abbildung 3.7 ist zu erkennen, dass sich dadurch immer eine annähernd flache Wasseroberfläche bildet.

$$f_{transport} = \frac{1}{4} \cdot \sum_{i=L,R,T,B} \{\frac{1}{2} \cdot [(x_t(rock) + x_t(sand) + x_t(water)) - (x_t^i(rock) + x_t^i(sand) + x_t^i(water))]\}$$

Meeresspiegel

Die Simulation enthält auch einen anpassbaren Meeresspiegel $p_{sealevel}$. Wenn die Summe der Höhen von Gestein $x_t(rock)$, Sand $x_t(sand)$ und Wasser $x_t(water)$ nach den zuvor beschriebenen Schritten kleiner als der eingestellte Meeresspiegel ist, wird der Wasserstand entsprechend erhöht. Vorteil hierbei ist, dass sich dadurch der Meeresspiegel leicht anpassen lässt, nicht abhängig von den anderen Simulationsteilen ist und ein Ozean nicht durch falsche Parametereinstellungen verschwindet. Allerdings können sich auch keine Seen an Land unterhalb des Meeresspiegels bilden.

3.3.4 Fließrichtung und Fließgeschwindigkeit

Die Fließrichtung und die Fließgeschwindigkeit werden durch einen zweidimensionalen Vektor beschrieben. Die Berechnung der Fließrichtung funktioniert ähnlich wie beim Wassertransport (siehe Abschnitt 3.3.3). Betrachtet werden die Summe der Höhen von Gestein $x_t(rock)$, Sand $x_t(sand)$ und Wasser $x_t(water)$ aller direkt benachbarten Zellen. Die Differenz der aktuellen und einer benachbarten Zellen ergibt die Fließrichtung. Aus den vier Ergebnissen wird anschließend der Mittelwert gebildet. Zusätzlich wird das Ergebnis gefiltert, um abrupte Übergänge zu vermeiden. Falls die entstehenden Werte größer als 1 oder kleiner als -1 sind, werden diese reduziert.

$$y_{t+1}(water) = \bar{y}_t(water) + \frac{1}{2} \cdot \sum_{i=L,R} \{\frac{1}{2} \cdot [(x_t(rock) + x_t(water)) - (x_t^i(rock) + x_t^i(water))]\} \tag{3.8}$$

$$z_{t+1}(water) = \bar{z}_t(water) + \frac{1}{2} \cdot \sum_{i=T,B} \{\frac{1}{2} \cdot [(x_t(rock) + x_t(water)) - (x_t^i(rock) + x_t^i(water))]\} \tag{3.9}$$

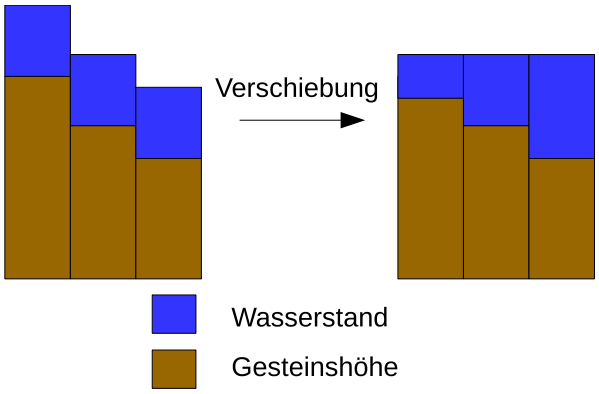


Abbildung 3.7 Darstellung der Wasserstandsänderung

3.3.5 Resultat

Durch die Einflüsse von Temperatur, Eis und Luftfeuchtigkeit auf die Verteilung der Wassermengen entstehen realistischere Flüsse und Seen. Auch dass sich diese über die Zeit in der Simulation und nicht durch einen vorgegebenen Algorithmus bilden, wirkt sich positiv aus. Jedoch kommt es in seltenen Fällen durch die starke Vereinfachung der Simulation vor allem bei steilen Abhängen zu Artefakten. Im Vergleich zur realen Weltkarte in Abbildung 3.8b sind sowohl stark mit Flüssen durchzogenen Gegenden, als auch Wüsten ohne Flüsse erkennbar. Dass die Flüsse nicht so fein verzweigt wie im realen Vorbild sind, liegt an der geringeren Auflösung der Textur.

3.4 Sand

In dieser Arbeit entsteht Sand alleinig durch Wassererosion. Hierbei wird das feste Gestein von Wasser abgetragen und durch Wasser $f_{watermovement}$ und Höhenunterschiede $f_{heigthmovement}$ transportiert. In der Realität bewegen sich große Mengen an Sand auch durch Windbewegungen[2]. Dieser Effekt wird hier nicht berücksichtigt. Für die Berechnung der Erdbodenhöhe wird die Summe aus Gesteinshöhe $x(rock)$ als auch die Sandhöhe $x(sand)$ verwendet.

$$x_{t+1}(sand) = \max(0.0, x_t(sand) + y_{t+1}(sand) + f_{watermovement} + f_{heigthmovement}) \tag{3.10}$$

3.4.1 Datenformat

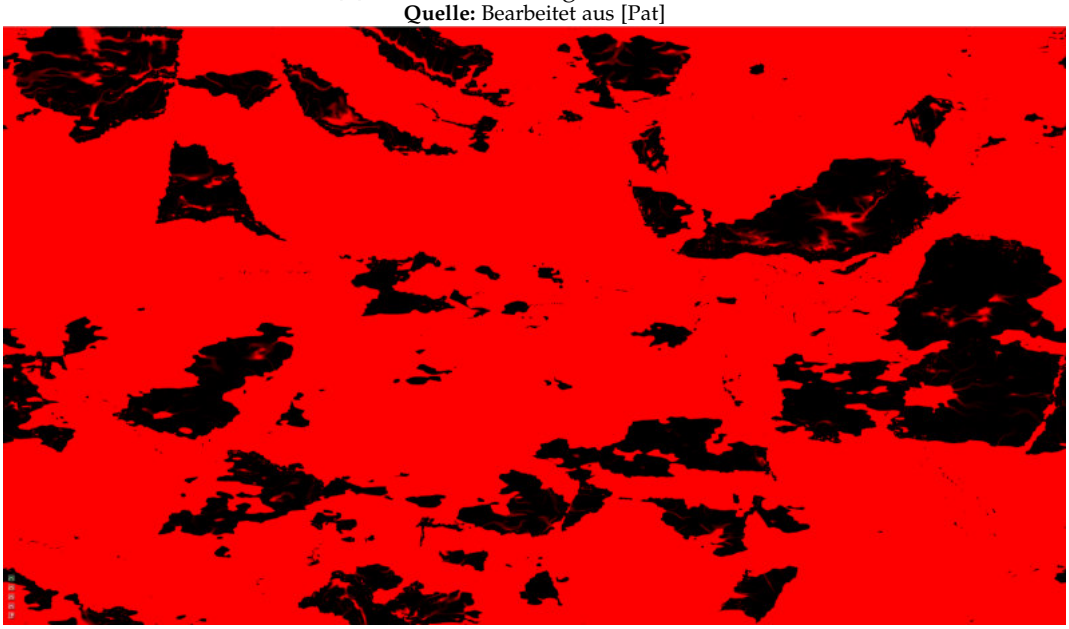
Das Datenformat für die Sandsimulation besteht aus zwei Werten. Der erste $x(Sand)$ gibt die aktuelle Sandhöhe an. Der zweite $y(Sand)$, wie viel neuer Sand im letzten Simulationsschritt generierte wurde. Die Wertebereich hierfür ist zwischen 0 und dem maximalen Gleitkommazahlwert.

3.4.2 Sanderosion

Die Sanderosion ist abhängig von der Wasserfließgeschwindigkeit ($y(water)$,$z(water)$), dem Wasserstand x_{wasser} und der aktuellen Sandhöhe x_{Sand}. Die Fließgeschwindigkeit wird mit

2 de.wikipedia.org/wiki/Saharastaub

(a) Wasserverteilung auf der Erde
Quelle: Bearbeitet aus [Pat]



(b) Beispiel Wassersimulation

Abbildung 3.8 Ergebnis aus der Wassersimulation.

einem Erosionsparameter $p_{wassererosion}$ multipliziert, um die Erosionsstärke anpassbar zu machen. Das Ergebnis wird durch die Summe aus Wasserstand und Sandhöhe geteilt. Der Sinn hierbei ist, dass wenn bereits Sand zwischen dem Gestein und dem Wasser existiert, weniger neuer Sand entsteht. Dies verhindert, dass alles Gestein zu Sand umgewandelt wird. Durch den Einfluss der Wasserhöhe erodiert ein flaches Gewässer, wie z.B. ein Fluss, mehr Sand als ein tiefes Gewässer.

$$y_{t+1}(sand) = \frac{\sqrt{y_t(water)^2 + z_t(water)^2} \cdot p_{watererrosion}}{x_t(sand) + x_t(water)} \tag{3.11}$$

3.4.3 Verschiebung durch Wasser

Die Sandmassen können durch die Bewegung des Wassers verschoben werden, wie in Abbildung 3.9 dargestellt. Die bewegte Sandmenge wird bestimmt durch das Verhältnis von Wasserfließgeschwindigkeit zum Wasserstand $x_t(wasser)$. Auch hierbei wird ein anpassbarer Faktor $p_{watermove}$ mit eingerechnet. Das Ergebnis wird mit der aktuellen Sandhöhe x_{Sand} multipliziert. Falls kein Sand vorhanden ist, beträgt somit auch der Wert der Verschiebung Null. Die Berechnung wird für die aktuelle Zelle, als auch für alle umgebenden Zellen verwendet. Das Ergebnis der aktuellen Zelle ist die Sandmasse, die aus der Zelle heraus transportiert wird. Bei den umgebenden Zellen wird der Richtungsvektor mit beachtet, um zu prüfen, ob die Sandmasse in die Zelle hinein transportiert wird. Das entstehende Sandbild ist in Abbildung 3.10a dargestellt. Hier sieht man sehr gut, dass die Sandmengen entlang der Flüsse entstehen. Wenn der Sand nur durch das Wasser verschoben wird, entstehen dabei schmale Flüsse, wie in Abbildung 3.10b zu sehen ist, da der Sand die Höhenunterschiede nicht ausgleicht.

$$\begin{aligned} f_{watermovement} = |\frac{y_t(water) + z_t(water)}{x_t(water)}| \cdot p_{watermove} - [\frac{y_t^L(water) \cdot p_{watermove}}{x_t^L(water)} \cdot x_t^L(sand) \\ + \frac{-y_t^R(water) \cdot p_{watermove}}{x_t^R(water)} \cdot x_t^R(sand) + \frac{z_t^B(water) \cdot p_{watermove}}{x_t^B(water)} \cdot x_t^B(sand) \\ + \frac{-z_t^T(water) \cdot p_{watermove}}{x_t^T(water)} \cdot x_t^T(sand)] \end{aligned}$$

3.4.4 Verschiebung durch Höhenunterschiede

Die zweite Art, mit der sich die Sandmassen verschieben, ist die Verschiebung durch Höhenunterschiede (Abbildung 3.11). Hierbei wird die Differenz zwischen den Gesteinshöhen betrachtet. Wenn die Höhendifferenz positiv ist, verschiebt sich Sand in die Zelle, ist sie negativ, reduziert sich die Sandmenge in der Zelle. Auch hierbei ist die Stärke der Verschiebung parametrierbar, um den Effekt anzupassen. Das Sandbild in Abbildung 3.12a ist hierbei deutlich breiter verteilt und nicht nur entlang der Flüsse. Die Flüsse hierbei werden ebenfalls deutlich verbreitert (Abbildung 3.12b), da der Sand die Höhenunterschiede ausgleicht.

$$f_{heigthmovement} = \frac{p_{sandmove}}{4} \cdot \sum_{i=L,R,T,B} \{\frac{1}{2} \cdot [(x_t(rock) + x_t(sand)) - (x_t^i(rock) + x_t^i(sand))]\} \tag{3.12}$$

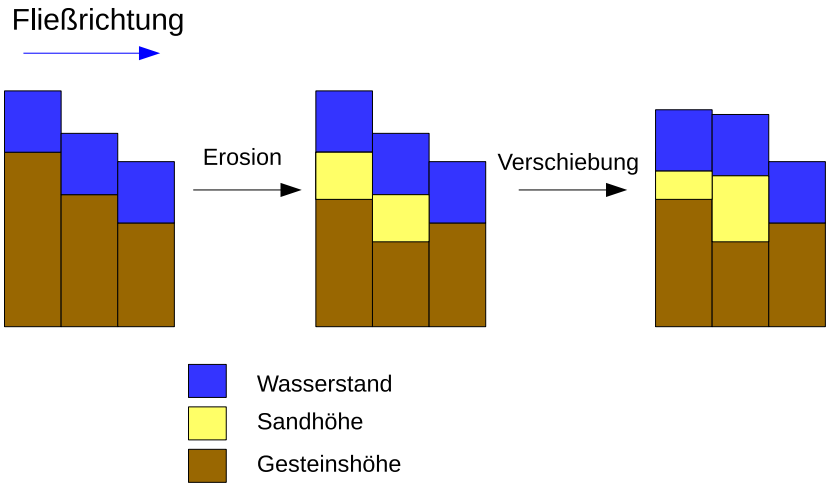
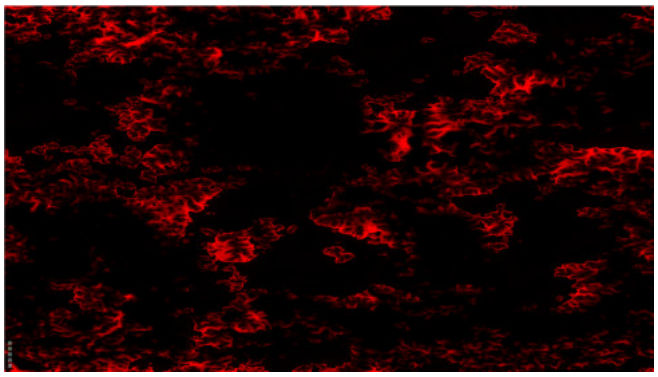
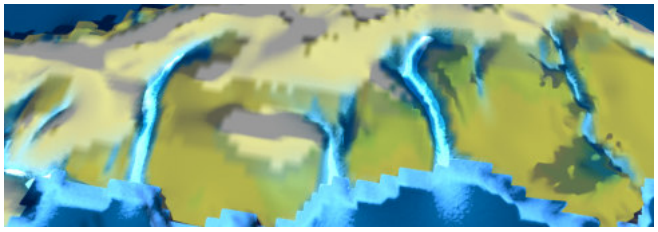


Abbildung 3.9 Darstellung von Sandverschiebung durch Wasser



(a) Sandverteilung



(b) 3-D-Darstellung der Flussbildung

Abbildung 3.10

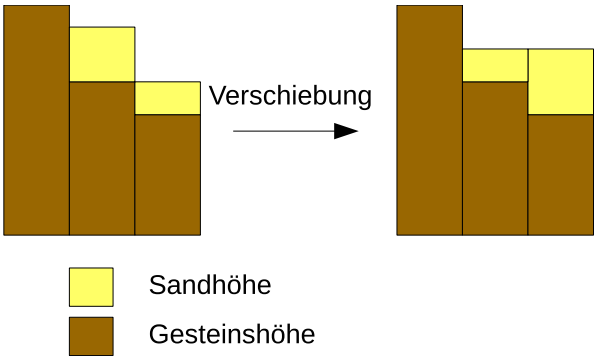
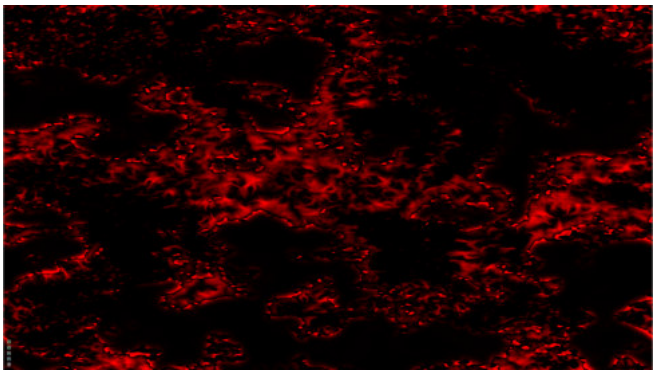
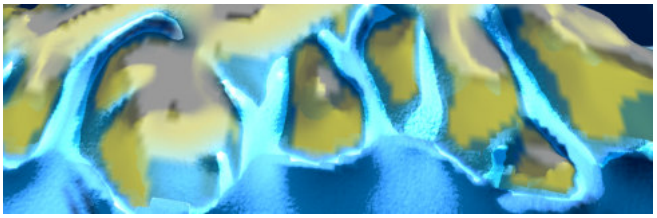


Abbildung 3.11 Darstellung von Sandverschiebung durch Höhen

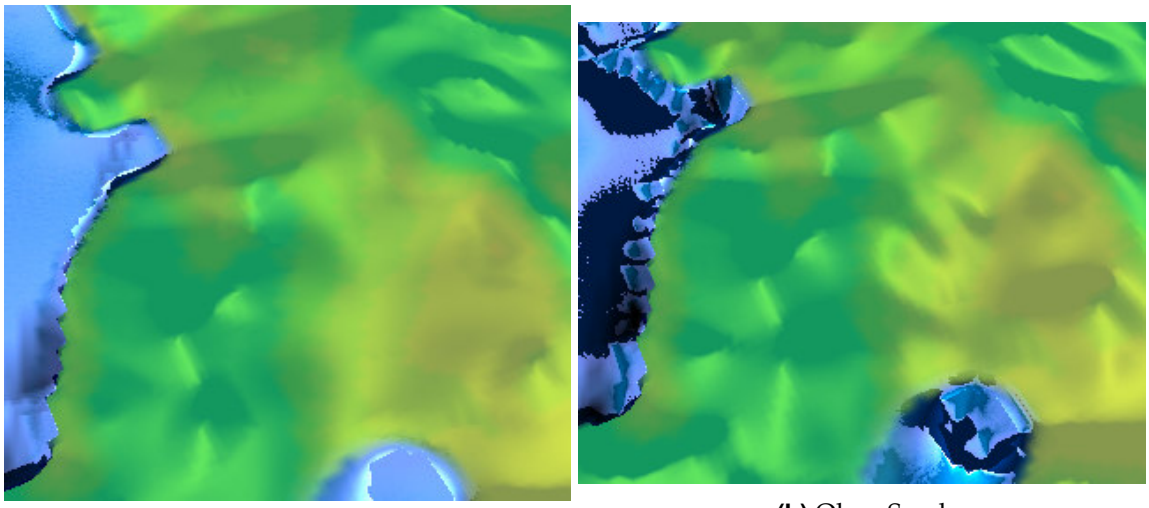


(a) Sandverteilung



(b) 3-D-Darstellung der Flussbildung

Abbildung 3.12 Darstellung der Sandeffekte bei Verschiebung durch Höhen



(a) Mit Sand

(b) Ohne Sand

Abbildung 3.13 Darstellung der Landschaftshöhe mit und ohne Sand

3.4.5 Resultat

Durch die parametrierbare Sandbildung kann die Veränderung der Landschaft leicht kontrolliert werden. Ob flache, breite oder schmale, tiefe Flüsse entstehen sollen, bestimmt dieser Teil der Simulation. Auch die daraus resultierende Höhenkarte wird massiv durch den Erosionsprozess verändert. Wenn die Sandhöhe in die Berechnung der Landschaftshöhe mit einbezogen wird, entstehen bei von Flüssen ausgewaschenen Gegenden eine flachere Landschaft. In Abbildung 3.13a ist zu sehen, dass der selbe Landschaftsabschnitt unter Einbezug der Sandhöhe flachere Übergänge hat.

3.5 Wind

Wind ist in dieser Arbeit das Transportmedium für Temperatur und Luftfeuchtigkeit. Berechnet wird die Windrichtung aus drei Faktoren: dem lokalen Luftdruck $f_{pressureDir}$, den globalen Windrichtungen $f_{globalDir}$ sowie einer zusätzlichen Textur $f_{noiseDir}$, jeweils in x- und y-Richtung. Ziel hierbei ist es, natürliche Phänomene, wie den Windschatten von Bergen oder globale Windbewegungen, nachzustellen. Vernachlässigt wird hierbei der Effekt von Temperatur und Luftfeuchtigkeit auf die Windrichtung. Ebenfalls nicht vorhanden sind große Hoch- und Tiefdruckgebiete. Diese werden durch die zusätzliche Textur angedeutet, jedoch nicht simuliert. Die Stärke drei Faktoren können über entsprechende Parameter eingestellt werden, um verschiedene Windbilder zu erzeugen. Zusätzlich wird das Ergebnis noch gefiltert, um die lokale Wechselwirkung der Windrichtung darzustellen.

$$x_{t+1}(wind) = \overline{f_{pressureDirX} \cdot p_{pressure} + f_{globalDirX} \cdot p_{globalWind} + f_{noiseDirX} \cdot p_{noise}} \tag{3.13}$$

$$y_{t+1}(wind) = \overline{f_{pressureDirY} \cdot p_{pressure} + f_{globalDirY} \cdot p_{globalWind} + f_{noiseDirY} \cdot p_{noise}} \tag{3.14}$$

3.5.1 Datenformat

Die Windrichtung wird angegeben als ein zweidimensionaler Vektor für die xy-Richtung ($x(wind)$,$y(wind)$). Die maximale Windstärke in jede Richtung ist 1 und -1, um Artefakte durch zu hohe Windgeschwindigkeiten zu vermeiden. Zusätzlich wird der aktuelle Luftdruck $z(wind)$ gespeichert. In den folgenden Bildern ist die x-Richtung in Rot und die y-Richtung in Grün angegeben. Somit kennzeichnen gelbe Bereiche eine Windrichtung nach rechts oben und schwarze Bereiche eine Windrichtung nach links unten. Der Farbe für den Luftdruck wird zur Übersichtlichkeit nicht mit angezeigt.

3.5.2 Luftdruck

Der aktuelle Luftdruck $z(wind)$ einer Zelle wird berechnet aus der Summe der Höhe von Wasser $x(water)$, Sand $x(sand)$ und Gestein $x(rock)$. Die Höhe hat inverses Verhältnis zum Luftdruck[3]. Je höher die Landschaft ist, desto niedriger ist der Luftdruck. Der Effekt durch die Höhenunterschiede bewirkt, dass sich der Wind von niedrigen Regionen in Richtung von höheren Regionen bewegt. Da sich bei Bergspitzen der Wind von beiden Seiten in Richtung Spitze bewegt, ermöglicht dies die Erzeugung von Regenschatten, da die durch den Wind transportierte Luftfeuchtigkeit an der Bergspitze hängen bleibt. Die Windrichtung durch den Luftdruck $f_{pressureDirx/y}$ wird anhand der Differenz des Luftdruckes der umgebenden Zellen berechnet. In Abbildung 3.14 lässt sich anhand der Windrichtung die Struktur der Landschaft erkennen.

$$z_{t+1}(wind) = \frac{1}{x_t(rock) + x_t(sand) + x_t(water)} \tag{3.15}$$

$$f_{pressureDirX} = (z_{t+1}(wind) - z_t^R(wind)) - (z_{t+1}(wind) - z_t^L(wind)) \tag{3.16}$$

$$f_{pressureDirY} = (z_{t+1}(wind) - z_t^T(wind)) - (z_{t+1}(wind) - z_t^B(wind)) \tag{3.17}$$

3.5.3 Globale Windrichtungen

Die durch verschiedenste Effekte[4] erzeugten globalen Windrichtungen sind hier als gestreckte Cosinuskurven entlang der y-Achse implementiert. Die Skalierungsfaktoren wurden so gewählt, dass die auf der Erde vorkommenden globalen Windrichtungen nachgebildet werden können. Hierbei werden die realen Druckunterschiede ignoriert und die Windrichtungen als feste Werte erzeugt. In Abbildung 3.15a ist die reale Windzirkulation dargestellt. Ignoriert wird ebenfalls die Zirkulation in verschiedenen Luftschichten, welche in den blauen Feldern dargestellt ist. Die Winde bewegen sich in der x-Achse in Richtung Äquator sowie die Polen. In y-Richtung wechselt die Richtung entlang des dreißigsten nördlichen und südlichen Breitengrades. In Abbildung 3.15b ist die Darstellung in der Simulation zu sehen.

$$f_{globalDirX}(u, v) = -\cos(3.5v \cdot \pi) \tag{3.18}$$

$$f_{globalDirY}(u, v) = -\cos(3v \cdot \pi) \tag{3.19}$$

3 de.wikipedia.org/wiki/Luftdruck
4 de.wikipedia.org/wiki/Planetarische_Zirkulation

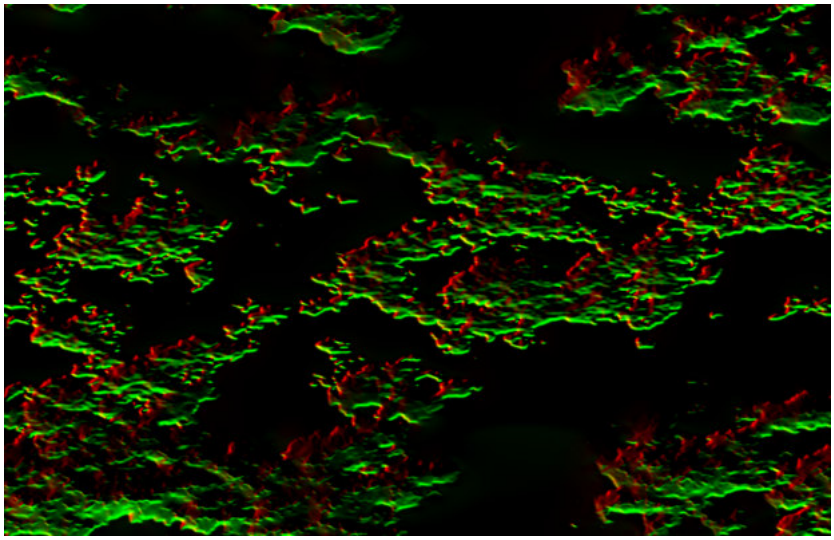
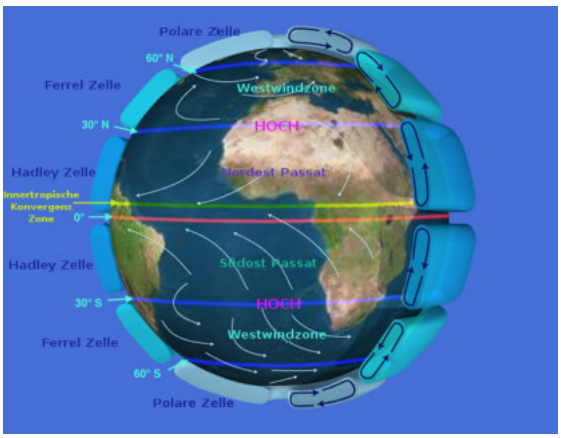


Abbildung 3.14 Windrichtung durch Höhenunterschiede



(a) Darstellung globaler Zirkulation
Quelle: ©Wikimedia Commons, NASA

(b) Globale Zirkulation

Abbildung 3.15 Globale Windrichtungen

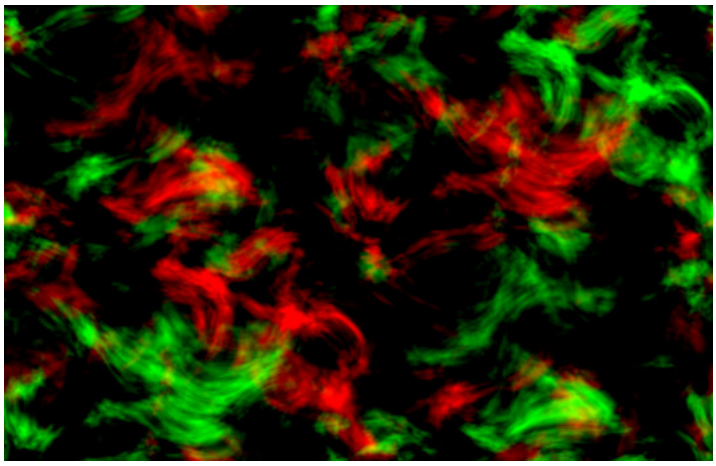


Abbildung 3.16 Wind Noisetextur

3.5.4 Noisetextur

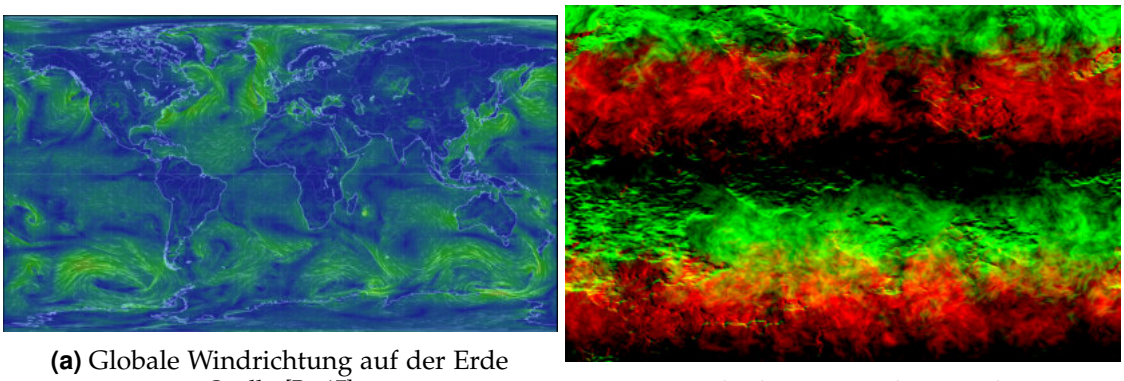
Da die Kombination aus lokalen Effekten und globalen Windrichtungen zu einem unbefriedigenden Ergebnis führten, wird die Windrichtung zusätzlich durch eine Textur verzerrt. Hierbei kommt der Diamond-Square-Algorithmus zum Einsatz, allerdings erweitert durch Turbulenzen. Dadurch entstehen Verzerrungen, welche Windbewegungen ähneln. Neben dem Parameter für die Noisetextur lässt sich auch die Stärke der Turbulenz anpassen. Um zusätzliche eine Verschiebung der Windzonen zu simulieren, kann man die Textur über die Zeit in eine Richtung verschieben. Ein Beispiel für eine solche Textur ist in Abbildung 3.16 zu sehen.

3.5.5 Resultat

Die finale Windrichtung ist eine Kombination der drei oben beschriebenen Methoden. Hierbei lässt sich durch jeweils einen Faktor $p_{globalWind}$, $p_{pressure}$ und p_{noise} einstellen, wie stark die Teile auf das Ergebnis wirken. Zuletzt wird das Ergebnis gefiltert, um weiche Windrichtungsänderungen zu generieren. Hierbei entsteht ein abwechslungsreiches und anpassbares Windverhalten mit Windschatten, Turbulenzen und globalen Windbewegungen. Leider sind die Ähnlichkeiten mit dem realen Vorbild aus Abbildung 3.17a aufgrund der Farbgebung schwer zu erkennen. Sichtbar sind die entstehenden Wirbel durch die Noise Textur sowie die auf dem Land stärker unterbrochenen Windströme. Fehlend sind Einflüsse wie Temperatur oder Luftfeuchtigkeit auf den Luftdruck sowie der daraus resultierenden Windänderung. Bei Versuchen, diese zu implementieren, entstand jedoch bisher kein verbessertes Ergebnis.

3.6 Temperatur

Die Temperatur beeinflusst drei andere Simulationsebenen: die Verdunstung von Wasser, die Entstehung von Eis und Luftfeuchtigkeit sowie die Verteilung der Klimazonen. Hauptsächlich wird hier die Erwärmung durch die Sonneneinstrahlung $f_{tempChange}$ generiert. Weitere Effekte hierbei sind die Höhenabkühlung, der Nachtschatten und die unterschiedliche Temperaturänderung über Wasser und Land. Zusätzlich wird die Temperatur noch durch den Wind bewegt $f_{tempMovement}$. Dadurch soll eine erdähnliche Temperaturverteilung generiert



(a) Globale Windrichtung auf der Erde
Quelle: [Bec17]

(b) Windrichtung aus der Simulation

Abbildung 3.17 Ergebnis aus der Windsimulation

werden, mit hohen Temperaturen am Äquator, niedrigen Temperaturen an den Polen, sowie eine abwechslungsreiche Verteilung dazwischen.

$$x_{t+1}(temp) = \bar{x}_t(temp) + f_{tempChange} + f_{tempMovement} \tag{3.20}$$

3.6.1 Datenformat

Die Temperatur wird bestimmt durch einen Temperaturwert $x(temp)$, der Werte zwischen -1 und 1 annehmen kann. Für eine bessere Visualisierung wird der Temperaturwert mit -1 multipliziert auf $z(temp)$ geschrieben und dadurch blau dargestellt, da negative Farbwerte nicht gezeichnet werden können. Somit sieht man die warmen Temperaturen in Rot und die kalten Temperaturen in Blau.

3.6.2 Sonneneinstrahlung

Die Erzeugung von Wärme wird hauptsächlich von der Sonneneinstrahlung bestimmt. Hierfür wird eine halbe Sinuskurve entlang der y-Achse des Bildes berechnet. Dadurch entstehen, wie in Abbildung 3.18 zu sehen ist, an den Äquatorregionen hohe Werte und an den Polregionen niedrige Werte. Um die Sonneneinstrahlung anpassen zu können, gibt es Parameter, um die Kurve zu verschieben. Durch eine Verschiebung des Graphen in x-Richtung ändert man den Winkel der Sonneneinstrahlung $p_{sunAngle}$. Eine Verschiebung in die y-Richtung erhöht die Stärke der Sonneneinstrahlung $p_{sunStrength}$.

$$f_{sun}(u, v) = \cos((v + p_{sunAngle}) \cdot 2\pi + \pi) + p_{sunStrength} \tag{3.21}$$

3.6.3 Nachtschatten

Da auf der sonnenabgewandten Seite eines Planeten keine Sonneneinstrahlung wirkt, wird zusätzlich ein sogenannter Nachtschatten erzeugt. Dieser wird stark vereinfacht als gestreckte Sinuskurve entlang der x-Achse des Bildes gezeichnet (siehe Abbildung 3.19). Diese Sinuskurve wird über die Zeit einstellbar verschoben, um den Temperaturwechsel zwischen Tag und Nacht zu erzeugen. Die Kurve lässt sich auch anhalten, um einen sonnensynchronen Orbit zu simulieren. Alle Koordinaten, die sich unterhalb der Kurve befinden, werden um

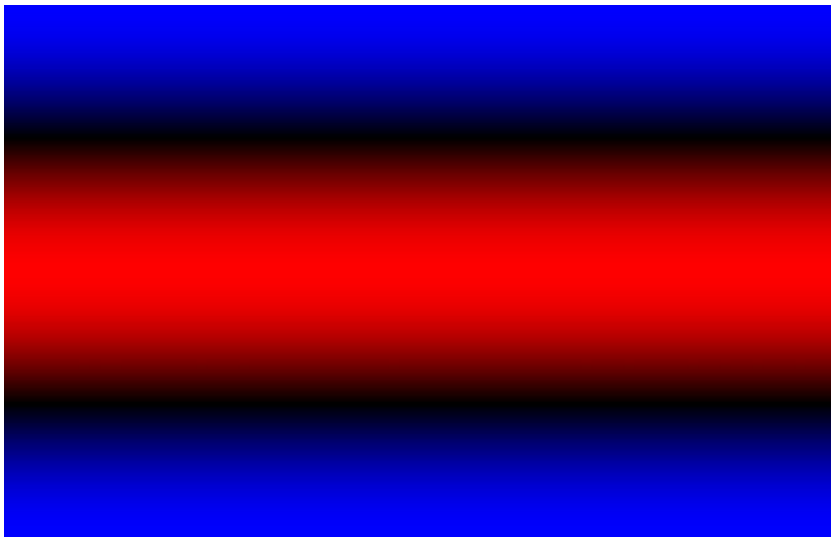


Abbildung 3.18 Globale Sonneneinstrahlung

einen parametrierbaren Wert $p_{nightTemp}$ abgekühlt. Durch die zeitliche Verschiebung des Nachtschattens entstehen in der Simulation dynamische Änderungen der Temperaturverteilung. Die in Gleichung 3.22 verwendeten Werte zur Berechnung der Sinuskurve wurden so gewählt, um den Nachschatten echt wirken zu lassen.

$$f_{night}(u,v) = \begin{cases} f_{sun}(u,v) & v + p_{sunAngle} < \sin(\frac{\pi}{2} \cdot \cos(u)) \cdot 0.7 + 0.5 \\ p_{nightTemp} & sonst \end{cases} \tag{3.22}$$

3.6.4 Abkühlung durch Höhen

Um die Temperaturveränderung in großen Höhen, zu simulieren, wird ab einem einstellbaren Schwellwert $p_{cooldown}$ die Temperatur reduziert. Je größer die Höhe über dem Schwellwert, desto stärker ist die Abkühlung. Dadurch entstehen um die Berge kältere Klimazonen und auf den Spitzen der Berge Eis. In Abbildung 3.20 ist durch die kalten, blau dargestellten Bereiche gut zu erkennen, wo sich auf der Karte die Berge befinden.

$$f_{heightCool} = \begin{cases} -(x_t(rock) - p_{cooldown}) & x_t(rock) > p_{cooldownLimit} \\ 0.0 & sonst \end{cases} \tag{3.23}$$

3.6.5 Temperaturänderung über Wasser und Land

Temperaturveränderungen passieren über dem Meer aufgrund der großen Wärmekapazität von Wasser deutlich langsamer[5]. Um dies nachzustellen, werden die in f_{night} und $f_{heightCool}$ berechneten Werte summiert und mit einem Faktor multipliziert, um die aktuelle Temperaturänderung zu erfassen. Der erwähnte Faktor ist, falls Wasser in der Zelle vorhanden ist, halb so groß wie die Änderung über Land.

5 https://de.wikipedia.org/w/index.php?title=Land-See-Windsystem&oldid=160712765

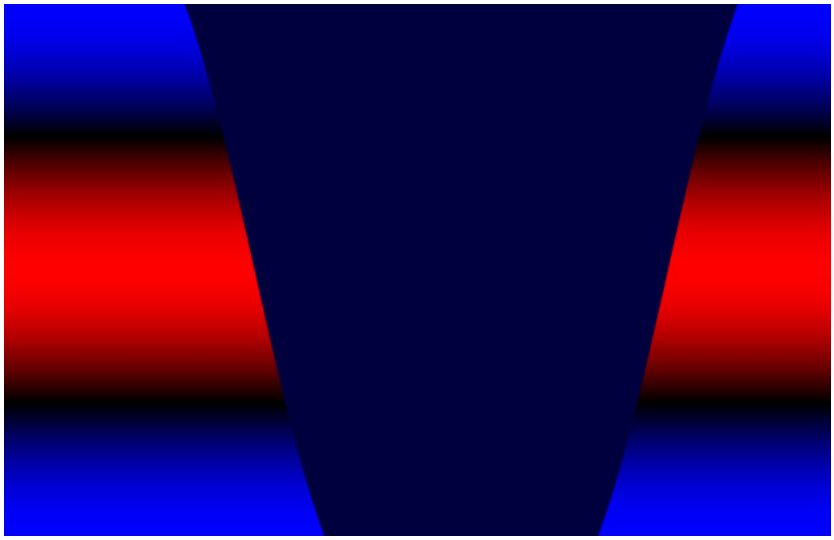


Abbildung 3.19 Nachtschatten

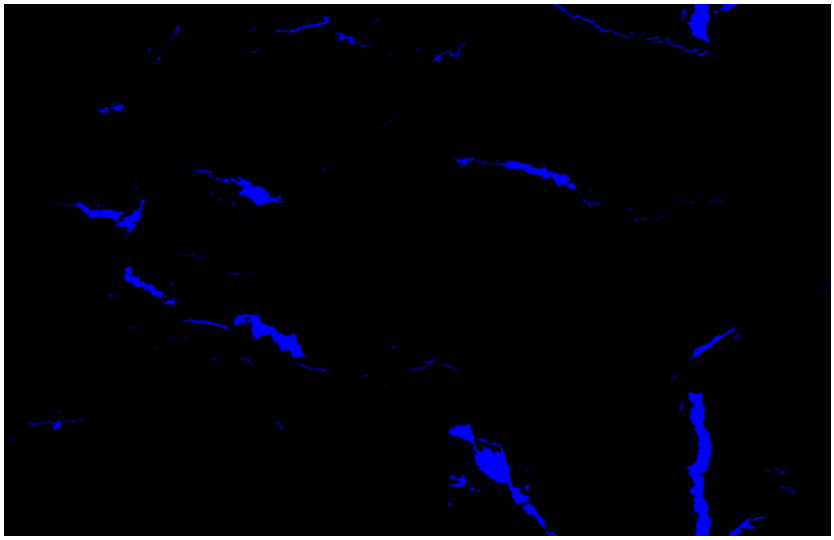


Abbildung 3.20 Abkühlung durch Höhen

$$f_{tempChange} = (f_{heightCool} + f_{night}) \cdot \begin{cases} 1 & x_t(water) > 0.1 \\ 2 & sonst \end{cases} \tag{3.24}$$

3.6.6 Verschiebung durch Wind

Die Temperatur wird auch durch die Windrichtung $f_{tempMovement}$ geändert. Das Verfahren hierbei ist ähnlich der Sandverschiebung durch Wasser (Unterabschnitt 3.4.3). Zuerst wird berechnet, wie viel Temperatur aus der Zelle heraus transportiert wird. Hierzu werden die Werte des Windrichtungsvektors $x_t(wind)$ und $y_t(wind)$ mit einem Faktor $p_{windMove}$ sowie der aktuellen Temperatur multipliziert. Das selbe wird für alle umgebenden Zellen berechnet, wobei nur die zur Zelle zeigende Achse des Vektors berücksichtigt wird. Die Ergebnisse der umgebenden Zellen werden summiert und die abtransportierte Temperatur davon abgezogen. Dadurch entsteht eine abwechslungsreiche Temperaturverteilung, abhängig von den eingestellten Windparametern.

$$\begin{aligned} f_{tempMovement} = |x_t(wind) + y_t(wind)| \cdot p_{windMove} \cdot x_t(temp) \\ +[-x_t^L(wind) \cdot p_{windMove} \cdot x_t^L(temp) \\ -x_t^R(wind) \cdot p_{windMove} \cdot x_t^R(temp) \\ +y_t^B(wind) \cdot p_{windMove} \cdot x_t^B(temp) \\ -y_t^T(wind) \cdot p_{windMove} \cdot x_t^T(temp)] \end{aligned}$$

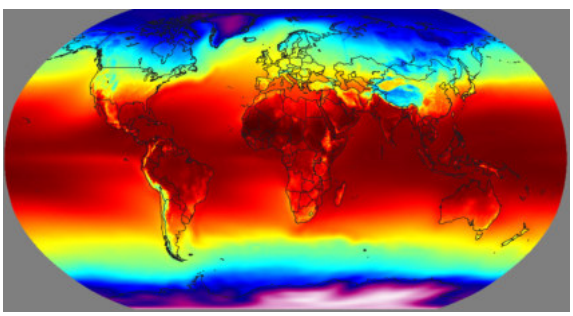
3.6.7 Resultat

Mit den oben beschriebenen Verfahren werden die wesentlichen Ziele im Vergleich mit Abbildung 3.21a für die Temperaturverteilung erreicht. Eine abwechslungsreiche und parametrierbare, zugleich realistisch erscheinende Verteilung der Temperatur. Durch den sich bewegenden Nachtschatten gewinnt die Simulation zusätzlich zur Laufzeit an Dynamik. Die Verteilung ist hauptsächlich geprägt durch die eingestellten Parameter für Wind und Sonneneinstrahlung. Allerdings machen sich auch fehlende Effekte bemerkbar, wie der Temperaturtransport durch Wasser[6], welche auch in polnahen Regionen gemäßigte Klimazonen erzeugt.

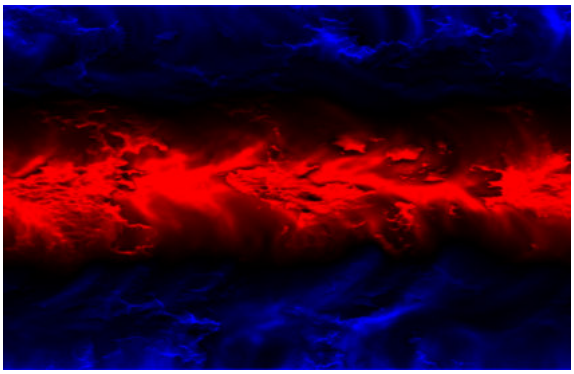
3.7 Luftfeuchtigkeit

In der realen Welt bewegt sich die Luftfeuchtigkeit auf verschiedenen Höhenebenen, wodurch z.B. Wolken und Nebel entstehen. Hier wird nur die Luftfeuchtigkeit in großen Höhen vereinfacht betrachtet. Die Luftfeuchtigkeit ist neben der Temperatur die Einflussgröße für die Verteilung der Klimazonen. Zusätzlich gibt diese an, wo und wie viel Wasser generiert werden soll. Erzeugt wird die Luftfeuchtigkeit über Wasser, was die Aufnahme von Wassermolekülen in die Luft darstellt. Über Land, Eis und bei hohen Temperaturen verringert sich die Luftfeuchtigkeit. Zusätzlich wird diese durch den Wind $f_{moistMovement}$ bewegt.

6 de.wikipedia.org/wiki/Meeresströmung#Klima



(a) Temperaturverteilung auf der Erde
Quelle: Bearbeitet, ©Robert A. Rohde / Wikimedia Commons / CC-BY-SA-3.0-migrated



(b) Temperaturverteilung aus der Simulation

Abbildung 3.21 Ergebnis aus der Temperatursimulation

$$x_{t+1}(moist) = \bar{x}_t(moist) + f_{moistChange} + f_{moistMovement} \tag{3.25}$$

3.7.1 Datenformat

Für die Luftfeuchtigkeit wird nur der Wert $x(moist)$ verwendet, der die aktuelle Luftfeuchtigkeit angibt. Der Wertebereich ist zwischen Null, für absolut trockene Luft, und Eins, für 100% Luftfeuchtigkeit.

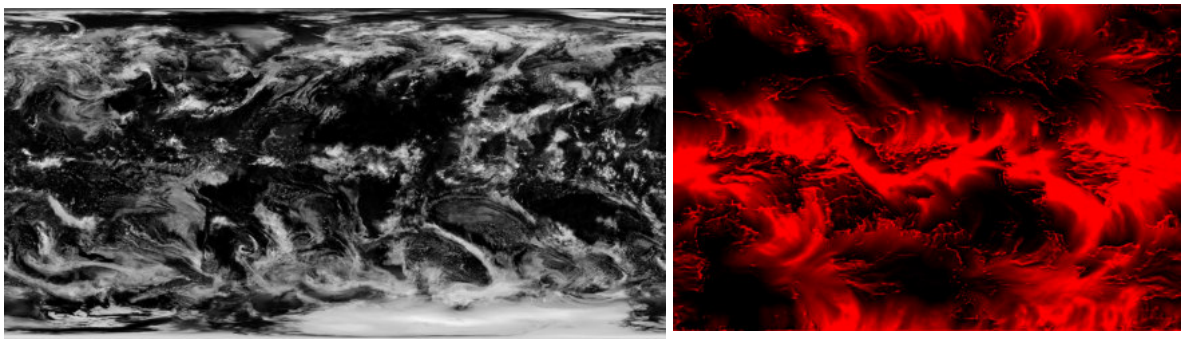
3.7.2 Änderung der Luftfeuchtigkeit

Die Luftfeuchtigkeit erhöht sich durch die Verdunstung von Wasser. Hierbei wird, falls sich in der Zelle Wasser befindet, die Luftfeuchtigkeit leicht erhöht. Die Temperatur hat hierbei keinen Einfluss, da ansonsten in kälteren Regionen der Wert zu schnell Null erreicht und dadurch sehr abwechslungsarm ist. Um wie viel der Wert pro Simulationsschritt ansteigt, ist durch den Parameter $p_{waterAdd}$ zur Laufzeit einstellbar. Wenn sich kein Wasser in der Zelle befindet, wird die Luftfeuchtigkeit leicht reduziert. Im Gegensatz zur Verdunstung bestimmt hier der Wert der Temperatur $x(temp)$, wie schnell die Luftfeuchtigkeit abnimmt. Je höher die Temperatur ist, desto schneller verringert sich die Luftfeuchtigkeit. Auch hierbei ist ein $p_{landRemove}$ Parameter vorhanden, der die Stärke des Effekts bestimmen kann. Um den vorher beschriebenen fehlenden Einfluss der Temperatur auf die Luftfeuchtigkeit auszugleichen, wird diese zusätzlich um den Wert $p_{iceRemove}$ reduziert, wenn sich Eis $x(ice)$ in der Zelle befindet. Dies simuliert das Frieren des Wassers in der Luft. Das Ergebnis aus den drei Einflüssen ist die aktuelle Änderung der Luftfeuchtigkeit $f_{moistChange}$.

$$f_{moistChange} = \begin{cases} p_{waterAdd} & x_t(water) > 0.1 \\ -p_{landRemove} & sonst \end{cases} - \begin{cases} p_{iceRemove} & x_t(ice) > 0.1 \\ 0.0 & sonst \end{cases} \tag{3.26}$$

3.7.3 Verschiebung durch Wind

Die Luftfeuchtigkeit wird durch die Windrichtung verschoben. Wie bereits bei der Temperatur (Unterabschnitt 3.6.6), wird die Luftfeuchtigkeit der aktuellen sowie den umgebend Zellen mit der Windstärke in x- und y-Richtung multipliziert und mit einem Parameter $p_{windMove}$



(a) Wolkenverteilung auf der Erde
Quelle: [NAS02]

(b) Luftfeuchtigkeitsverteilung in der Simulation

Abbildung 3.22 Ergebnis der Luftfeuchtigkeitssimulation

verrechnet. Von der Summe der umgebenden Zellen wird das Ergebnis der aktuellen Zelle subtrahiert.

$$\begin{aligned} f_{moistMovement} = |x_t(wind) + y_t(wind)| \cdot p_{windMove} \cdot x_t(moist) \\ +[-x_t^L(wind) \cdot p_{windMove} \cdot x_t^L(moist) \\ -x_t^R(wind) \cdot p_{windMove} \cdot x_t^R(moist) \\ +y_t^B(wind) \cdot p_{windMove} \cdot x_t^B(moist) \\ -y_t^T(wind) \cdot p_{windMove} \cdot x_t^T(moist)] \end{aligned}$$

3.7.4 Resultat

Wie in Abbildung 3.22b zu erkennen ist, entstehen in verschiedenen Regionen des Bildes trockene und feuchte Abschnitte. Die erkennbaren dünnen Linien ist die an Gebirgszügen hängen bleibende Luftfeuchtigkeit. Die schwarzen Bereiche dahinter sind als Regenschatten zu interpretieren. Auch die Transportwege der Luftfeuchtigkeit durch den Wind sind klar erkennbar. Abhängig von den Windparametern verändert sich die Verteilung. Auch im Vergleich zur realen Abbildung 3.22a ergibt sich ein sehr ähnliches Bild, auch wenn hier nur die Wolken und nicht die Luftfeuchtigkeitsverteilung als Referenz dient. Leider nimmt die Verteilung bei gleichbleibenden Parametern schnell einen stabilen Zustand ein, der sich nicht mehr ändert. Hier besteht noch Überarbeitungsbedarf, um die Simulationsebene noch dynamischer zu gestalten.

3.8 Eis

Eis beeinflusst die Beschaffung der Erde auf verschiedene Arten. So entstehen durch Schmelzwasser $f_{iceChange}$ weitreichende Flusssysteme und Meeresspiegeländerungen. Auch das Eis selbst erzeugt durch die Eigenbewegung ebenfalls Erosion. Die Bewegung der Eismassen $f_{transportIce}$ wird in dieser Arbeit simuliert, erzeugt jedoch keine Erosion. Die Gefrier- und Schmelzprozesse werden durch die Temperatur $x(temp)$ bestimmt.

$$x_{t+1}(ice) = \max(0.0, x_t(ice) + f_{iceChange} + f_{transportIce}) \tag{3.27}$$

3.8.1 Datenformat

Die aktuelle Eisdicke wird in $x(ice)$ gespeichert und ist in den folgenden Bildern Rot dargestellt. Die Maximalwerte hierfür sind zwischen Null für kein Eis und Eins. In $y(ice)$ wird gespeichert, wie viel Eis im letzten Simulationsschritt geschmolzen ist. Dies wird für die Erzeugung von Schmelzwasser (siehe Abschnitt 3.3.3) benötigt.

3.8.2 Eiswachstum und Eisschmelze

Die Änderung in der Eisdicke $f_{iceChange}$ ist abhängig von der aktuellen Temperatur $x(temp)$ und einer einstellbaren Gefriertemperatur $p_{freezeTemp}$. Wenn die Temperatur niedriger als die Gefriertemperatur ist, entsteht neues Eis. Ansonsten verringert sich die Eisdicke. Die geschmolzene Eismenge wird zusätzlich noch in $y(ice)$gespeichert.

$$f_{iceChange} = (x_t(temp) \cdot \begin{cases} 1 & x_t(temp) < p_{freezeTemp} \\ -1 & sonst \end{cases} \tag{3.28}$$

$$y_{t+1}(ice) = \max(0.0, |x_t(ice) - f_{iceChange}|) \tag{3.29}$$

3.8.3 Bewegung durch Höhenunterschiede

Ähnlich der Wasserverschiebung (siehe Unterabschnitt 3.3.4), verschieben sich die Eismassen durch Höhenunterschiede. Hierbei wird in den Zellen die Summe der Höhen von Eis $x(ice)$, Sand $x(sand)$ und Gestein $x(rock)$ berechnet und mit der aktuellen Eishöhe multipliziert. Ansonsten würde allein durch Höhenunterschiede Eis entstehen, ohne dass in der Zelle vorher welches vorhanden war. Die Differenz aus den Höhenunterschieden zwischen der aktuellen und einer benachbarten Zelle ergibt die transportierte Eismenge. Beachtet werden hierbei die vier direkt umgebenden Zellen. Die finale Änderung ist die Summe aus allen vier Berechnungen.

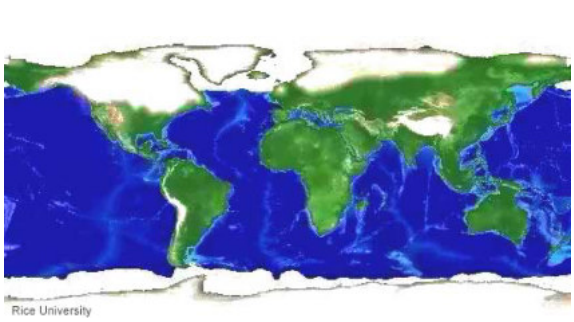
$$\begin{aligned} f_{transportIce} = \sum_{i=L,R,T,B} \{\frac{1}{2} \cdot [(x_t(rock) + x_t(soil) + x_t(ice)) \cdot x_t(ice) \\ -(x_t^i(rock) + x_t^i(soil) + x_t^i(ice)) \cdot x_t^i(ice)]\} \end{aligned}$$

3.8.4 Resultat

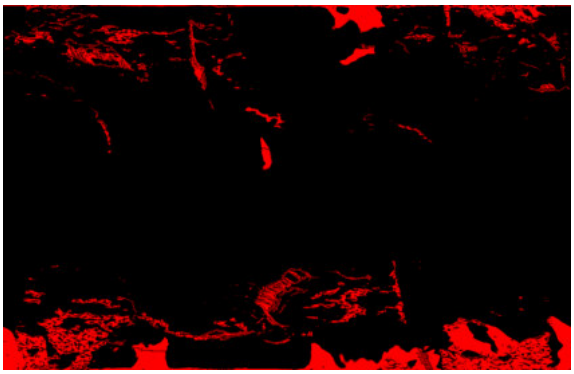
In Abbildung 3.23b sind zwei verschiedene Arten der Eisverteilung erkennbar. In den Bereichen, die durchgehend Rot sind, befinden sich Eisschilde, welche vor allem an den Polregionen entstehen. Die anderen, mit schwarzen Furchen durchzogenen Bereiche, sind Gletscher an Bergregionen. Die Furchen entstehen durch die Verschiebung bergab. Somit werden die zwei wichtigsten globalen Eiseffekte dargestellt. Auch die großflächige Verteilung ist dem realen Vorbild in Abbildung 3.23a durchaus ähnlich.

3.9 Klimazonen

Die Verteilung der Klimazonen ist kein direkter Teil der Simulation, der in die Berechnung mit einfließt. Vielmehr ist es eine Darstellung der Temperatur-, Luftfeuchtigkeit-, Eis- und



(a) Eisverteilung auf der Erde
Quelle: [NAS02]



(b) Eisverteilung in der Simulation

Abbildung 3.23 Ergebnis der Eissimulation

Wasserverteilung. Durch die Darstellung kann leichter erkannt werden, ob die eingestellten Parameter zu einem der gewünschten Ergebnisse führen. Des Weiteren kann es für folgende Arbeiten zur Generierung von verschieden Ansichten, wie z.B. der Vegetation dienen.

3.9.1 Holdridge life zones system

Das *Holdridge life zones system* ist eine Klassifizierung von Klimazonen, das von Leslie Holdridge 1947 entwickelt wurde [Hol47]. Es unterscheidet anhand von Luftfeuchtigkeit, Temperatur und Geländehöhe insgesamt 38 verschiedene Klimazonen (siehe Abbildung 3.24). Das System wurde in dieser Arbeit vereinfacht umgesetzt, da zwar die Werte für die Bestimmung der Klimazonen vorhanden sind, allerdings in abstrakten und nicht in physikalisch korrekten Werten. Zusätzlich werden in dieser Arbeit noch Gewässer nach Flüssen, Seen und Ozean unterschieden.

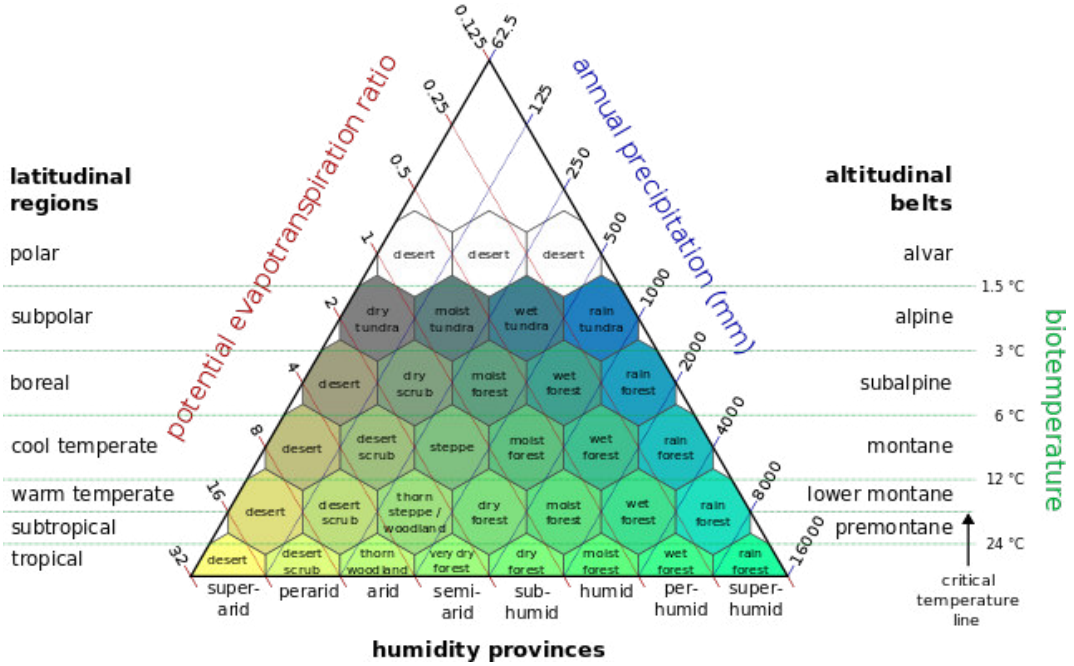
3.9.2 Datenformat

Für jede der 38 Klimazonen wurde eine RGB-Farbe definiert, die angenähert an die Farbe in Abbildung 3.24 ist. Damit sind die Klimazonen im resultierenden Bild farblich unterscheidbar (siehe Tabelle 3.2).

3.9.3 Berechnung

Eis und Gewässer

Als erstes wird geprüft, ob eine Zelle mit Eis bedeckt ist. Falls das der Fall ist, wird dieser die Klimazone *Polare Wüste* zugewiesen. Bei den Gewässern wird zwischen Flüssen, Seen und Ozeanen unterschieden. Wenn der Wasserstand eine bestimmte Höhe hat, wird anhand von der Geländehöhe unterschieden, ob das Gewässer ein See oder Ozean ist. Falls das Gelände einen Wert kleiner als 0.1 hat, was knapp über dem Meeresboden liegt, ist das Gewässer ein Ozean. Ansonsten ist es ein See. Flüsse werden anhand der Fließgeschwindigkeit definiert.



Quelle: ©Peter Halasz / Wikimedia Commons / CC-BY-SA-2.5

Abbildung 3.24 Holdridge life zones

Temperaturzonen

Um die Holdridge life zones nach zustellen, werden hier sechs verschiedene Temperaturzonen unterschieden. Diese sind anhand der Temperatur $x(temp)$ einer Zelle bestimmbar und gliedern sich wie folgt auf:

- alpin ($x(temp) \leq$ -0.8)
- boreal (-0.8 < $x(temp) \leq$ -0.3)
- kal$x(temp)$(-0.3 < $x(temp) \leq$ 0.0)
- warm (0.0 < $x(temp) \leq$ 0.25)
- subtropisch (0.25 < $x(temp) \leq$ 0.5)
- tropisch (0.5 < $x(temp)$)

Die Werte wurden so gewählt, dass diese eine regelmäßige Verteilung der Klimazonen erzeugt.

Humiditätszonen

Die Humiditätszonen werden anhand der Werte der Luftfeuchtigkeitssimulation $x(moist)$ in acht verschiedenen Zonen eingeteilt.

- superarid ($x(moist) \leq$ 0.1)
- perarid (0.1 < $x(moist) \leq$ 0.25)

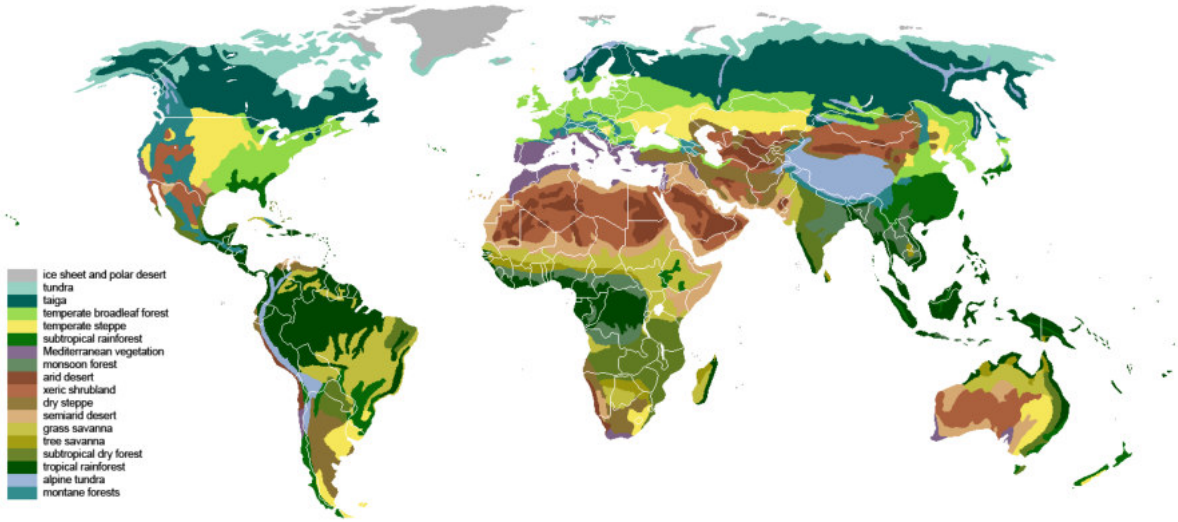
Tabelle 3.2 Farbdarstellung der Klimazonen. Die Zahlenwerte geben die RGB Farbe an

	superarid	perarid	arid	semiarid	humid	perhumid	superhumid
alpin	0.5 0.5 0.5	0.38 0.5 0.5	0.25 0.5 0.5	0.125 0.5 0.75	0.125 0.5 0.75	0.125 0.5 0.75	0.125 0.5 0.75
boreal	0.62 0.62 0.5	0.5 0.62 0.62	0.37 0.62 0.5	0.25 0.62 0.56	0.125 0.62 0.75	0.125 0.62 0.75	0.125 0.62 0.75
kalt	0.75 0.75 0.5	0.62 0.75 0.5	0.5 0.75 0.5	0.37 0.75 0.5	0.25 0.75 0.56	0.125 0.75 0.75	0.125 0.75 0.75
warm	0.87 0.8 0.5	0.81 0.8 0.5	0.62 0.8 0.5	0.5 0.8 0.5	0.37 0.8 0.5	0.25 0.8 0.56	0.125 0.8 0.75
subtropisch	0.92 0.95 0.5	0.82 0.95 0.5	0.69 0.95 0.5	0.5 0.95 0.5	0.37 0.95 0.5	0.25 0.95 0.56	0.125 0.95 0.69
tropisch	1.0 1.0 0.5	0.88 1.0 0.5	0.75 1.0 0.5	0.63 1.0 0.5	0.5 1.0 0.5	0.37 1.0 0.5	0.125 1.0 0.62

- arid ($0.25 < x(moist) \leq 0.35$)
- semiarid ($0.35 < x(moist) \leq 0.5$)
- humid ($0.5 < x(moist) \leq 0.65$)
- perhumid ($0.65 < x(moist) \leq 0.8$)
- superhumid ($0.8 < x(moist)$)

3.9.4 Resultat

Wie in Abbildung 3.26 zu sehen ist, ergibt sich durch die Aufteilung ein abwechslungsreiches Bild der Klimazonen. Auch verglichen mit den Klimazonen der Erde in Abbildung 3.25 sind Ähnlichkeiten erkennbar. Vor allem um den Äquator bilden sich ausgedehnte tropische Gebiete, sowie trockenen Wüsten nördlich und südlich davon. An den Polregionen wirken die Klimazonen der Simulation deutlich fragmentierter als die reale Vorlage, jedoch ist dies dem Umstand geschuldet, dass das Beispiel nur 18 Klimazonen unterscheidet.



Quelle: ©Ville Koistinen / Wikimedia Commons / CC-BY-SA-3.0-migrated

Abbildung 3.25 Klimazonen auf der Erde

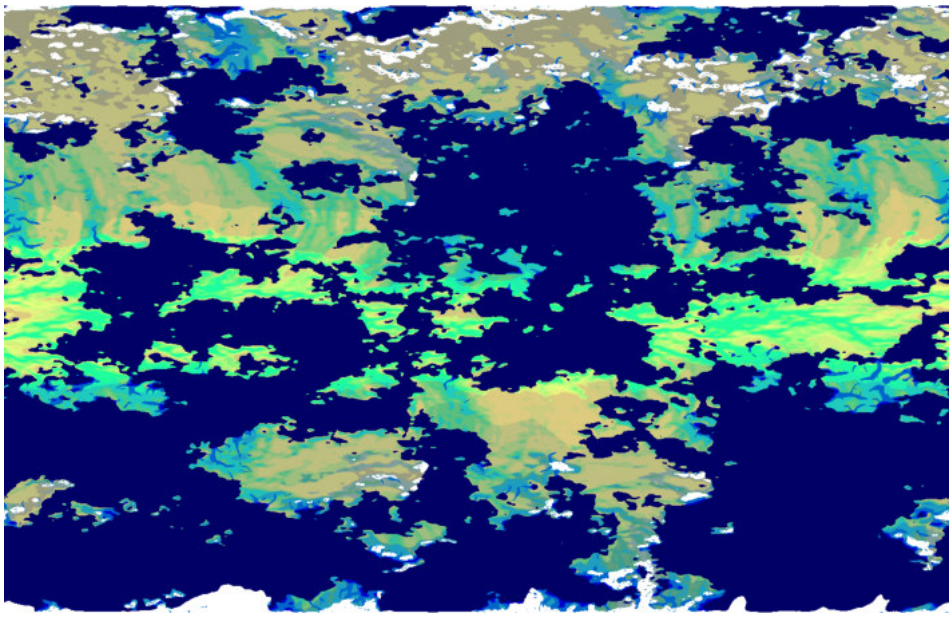


Abbildung 3.26 Klimazonen aus der Simulation

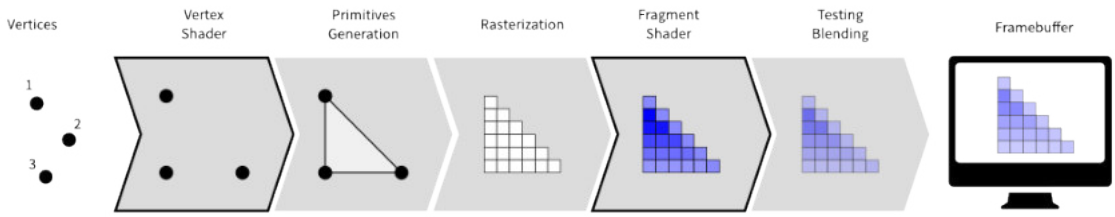
4 Implementierung

4.1 Zellulärer Automat auf der GPU

Ein Ziel dieser Arbeit ist, mit der Simulation Echtzeitfähigkeit von 60 Simulationsschritten pro Sekunde zu erreichen. Da diese auf zellulären Automaten basiert, in welchen der nächste Zustand einer jeden Zelle unabhängig von allen anderen Zellen berechnet werden kann, lässt sich dies einfach parallelisieren. Um die GPU für die Berechnungen zu verwenden, wurde auf die Programmierschnittstelle OpenGL zurückgegriffen. Diese bietet seit der Version 3.1 sogenannte Shader an [SA09]. Shader sind Programme für die GPU, welche für bestimmte Abschnitte beim 3D-Rendering zuständig sind. Diese sind frei programmierbar, haben jedoch als Ergebnis immer eine zweidimensionale Textur. Da die Datenformate in der Simulation so gewählt wurden, dass diese in einer RGB-Textur gespeichert werden können, kann das Ausgabebild des Renderings als Ergebnis der verschiedenen Simulationsebenen benutzt werden.

4.1.1 Renderingpipeline

Die OpenGL Renderingpipeline besteht aus fünf mit Shadern programmierbare Abschnitten. Die ersten vier Abschnitte sind für die Erstellung und Manipulation von dreidimensionalen Punkten und Polygonen zuständig. Diese sind in Abbildung 4.1 unter dem Abschnitt *Primitives Generation* zusammengefasst. Im ersten Schritt, dem *Vertexshader*, werden drei Punkte, Vertices genannt, im Raum verschoben und transformiert. In dem *Tesselation Control* und dem *Tesselation Evaluation Shader* wird das sogenannte Tesselation behandelt. Darunter versteht man die Unterteilung von einem Polygon, wie z.B. einem Dreieck, in mehrere kleine Polygone. Im *Geometryshader* kann die Geometrie der Polygone nachträglich manipuliert werden. Danach findet die sogenannte Rasterisierung statt. Hierbei wird die 3-D-Welt auf ein 2-D-Bild der Kamera projiziert. Im letzten Schritt, dem *Fragmentshader*, werden schließlich nur noch einzelne Pixel betrachtet. Diese können nach belieben eingefärbt werden. Die finale Textur, die der *Fragmentshader* erzeugt wird im *Framebuffer* gespeichert.



Quelle: glumpy.github.io

Abbildung 4.1 Die OpenGL Renderingpipeline

4.1.2 Offscreen Rendering

Das *Offscreen Rendering* ist ein 3-D-Renderingverfahren, bei dem ein gerendertes Bild aus dem Framebuffer nicht direkt auf den Bildschirm gezeichnet, sondern als Textur zwischengespeichert wird [SWH15]. Diese lässt sich in einem nachfolgenden Renderingschritt wiederverwerten. Bekannt wurde diese Technik vor allem für das Erstellen von Spiegelungen [Khr]. Hierbei wurde die Kamera für das *Offscreen Rendering* zuerst an die Position des Spiegels gestellt und aus dessen Sicht eine Textur erstellt. Diese wurde später im eigentlichen Renderingschritt auf den Spiegel angebracht. Da in den Texturen bis zu vier Gleitkommazahlen gespeichert werden können, wird diese Technik inzwischen immer mehr verwendet, um andere Daten in der Textur zu speichern. Beispiele hierfür sind Ray-Tracing oder Blur-Effekte [SWH15].

4.1.3 Shaderimplementierung

In dieser Arbeit wird des *Offscreen Rendering* verwendet, um die Simulation auf der GPU zu parallelisieren. Da sich die Simulation auf RGB-Texturen abspielt, ist der einzige interessante Schritt in der Renderingpipeline im Fragmentshader. Davor wird im Vertexshader lediglich ein 3-D Rechteck so gesteckt, dass nach der Rasterisierung ein Pixel der Simulationstextur genau auf einem Fragment im Fragmentshader landet. Normalerweise werden die Farbwerte der Texturen hier interpoliert, falls der Wert nicht exakt passt. Da jedoch die Textur als zellulärer Automat implementiert werden soll, sodass jedes Pixel einer Zelle entspricht, ist diese Streckung notwendig.
Damit auf die Texturen des vorherigen Simulationsschrittes zugegriffen werden kann, muss die entsprechende Texturkoordinate berechnet werden. OpenGL bietet bereits eine fertige Funktion *textur2D()* mit der Textur, auf die zugegriffen wird, sowie der Koordinate als Gleitkommawert zwischen Null und Eins, als Parameter. Für die Koordinatenberechnung erzeugt OpenGL im Fragmentshader automatisch die sogenannte Fragment-Koordinate *gl_FragCoord*. Diese gibt für den aktuellen Pixel die Koordinate innerhalb der Kamera. Wenn wie Kamera eine Auflösung von 800x600 Pixeln hat, ist die Koordinate unten links im Bild (0,0) und oben rechts (800,600). Um den Wert für die Funktion zu berechnen, muss die Fragmentkoordinate durch die Texturgröße geteilt werden. Auch für die Texturgröße gibt es bereits eine fertige OpenGL Funktion *textureSize()*. Da auch auf die benachbarten Zellen zugegriffen wird, muss die Fragmentkoordinate noch mit einem Vektor addieren werden. Mittels der *fract()* Funktion wird garantiert, dass die Werte an den Rändern umgebrochen werden. Somit können wir mittels

```
vec3 getTexValue(sampler2D tex, float x, float y)
{
    return texture2D(tex, fract((gl_FragCoord.xy+vec2(x,y))/textureSize(tex,0))).xyz;
}
```

auf die entsprechenden Zellen zugreifen.
Da OpenGL vorschreibt, dass eine Textur, die als Input in ein Shaderprogamm übergeben wird, schreibgeschützt ist, muss ein *Double-Buffering* implementiert werden. Hierbei gibt es zwei Texturen, eine als Quelle, eine als Ausgabe. Da die Ausgabe im nächsten Simulationsschritt als Quelle verwendet wird, tauschen die Texturen ihre Plätze. Die neue Ausgabetextur wird hierbei überschrieben, was aber nicht relevant ist, da die Simulation nur den direkten Vorgänger beachtet.

Die Renderingpipeline ist optimiert für die Berechnung von Farbwerten. Diese sind Standardmäßig als vorzeichenlose 16-Bit Gleitkommazahlen definiert, da der Wertebereich für die Farbdarstellung ausreichend ist. Jedoch ist für die Simulation notwendig, dass mit vorzeichenbehafteten 32-Bit Gleitkommazahlen gerechnet wird. Hierfür müssen drei spezielle Einstellungen, GL_CLAMP_VERTEX_COLOR, GL_CLAMP_READ_COLOR und GL_CLAMP_FRAGMENT_COLOR auf FALSE gesetzt werden. Ansonsten würden die Ergebnisse automatisch auf die 16-Bit Werte reduziert werden.
Die Anpassungen der Parameter in der Simulation wird über sogenannte Unifroms gemacht. Uniforms sind Variablen, die auf der CPU verändert werden können und als Input für ein Shaderprogramm dienen. Dadurch lassen sich die in der Simulation beschriebenen anpassbaren Parameter anpassen.

4.2 Softwaredesign

Die Software besteht aus fünf Teilen, welche in dem Klassendiagramm in Abbildung 4.2 zu sehen sind. Das Inputhandling kümmert sich um die Verarbeitung der Tastatur- und Mauseingaben, welche für die Parameteranpassung sowie der Kamerasteuerung benötigt werden. Für das Laden und Speichern von Texturen und Shadern gibt es jeweils eine entsprechende Klasse. Ein Softwareblock kümmert sich um die Initialisierung und dem Handling von OpenGL sowie dem Starten eines Simulationsschrittes. Zusätzlich wurde für eine bessere Visualisierung noch eine 3-D Ansicht eingebaut. Verwendet werden die Bibliotheken GLFW3[1] für das Erstellen des OpenGL Kontextes, glm[2] für Matrizenberechnungen, DevIL[3] für das Laden und Speichern von Texturen, libNoise[4] für die Erstellung von fraktalen Texturen und AntTweakBar[5] für die Bedienelemente.

4.2.1 Laden und Speichern

ShaderLoader

Die Klasse ShaderLoader ist für das laden, kompilieren, linken und überprüfen von Shadern zuständig. Die Shaderdateien müssen mit *.glsl* enden und davor ein Kürzel entsprechend dem Abschnitt in der Rendering Pipeline haben, z.B. ein Vertexshader muss die Dateiendung *.vs.glsl* verwenden. Darüber hinaus müssen Shaderdateien, die zu einem Programm zusammengehören, denselben Namen vor der Dateiendung besitzen. Die zwei Shaderdateien für die Wassersimulation heißen *water_sim.vs.glsl* und *water_sim.fs.glsl*. Falls die angegebenen Dateien vorhanden sind, werden diese kompiliert und zu einem Programm gelinkt. Falls hierbei ein Fehler auftritt, wird dieser auf der Konsole ausgegeben. Zuletzt wird das Programm auf die GPU übertragen und eine ID zurückgegeben, über die das Programm referenziert werden kann.

1 glfw.org
2 glm.g-truc.net
3 openil.sourceforge.net
4 libnoise.sourceforge.net
5 anttweakbar.sourceforge.net

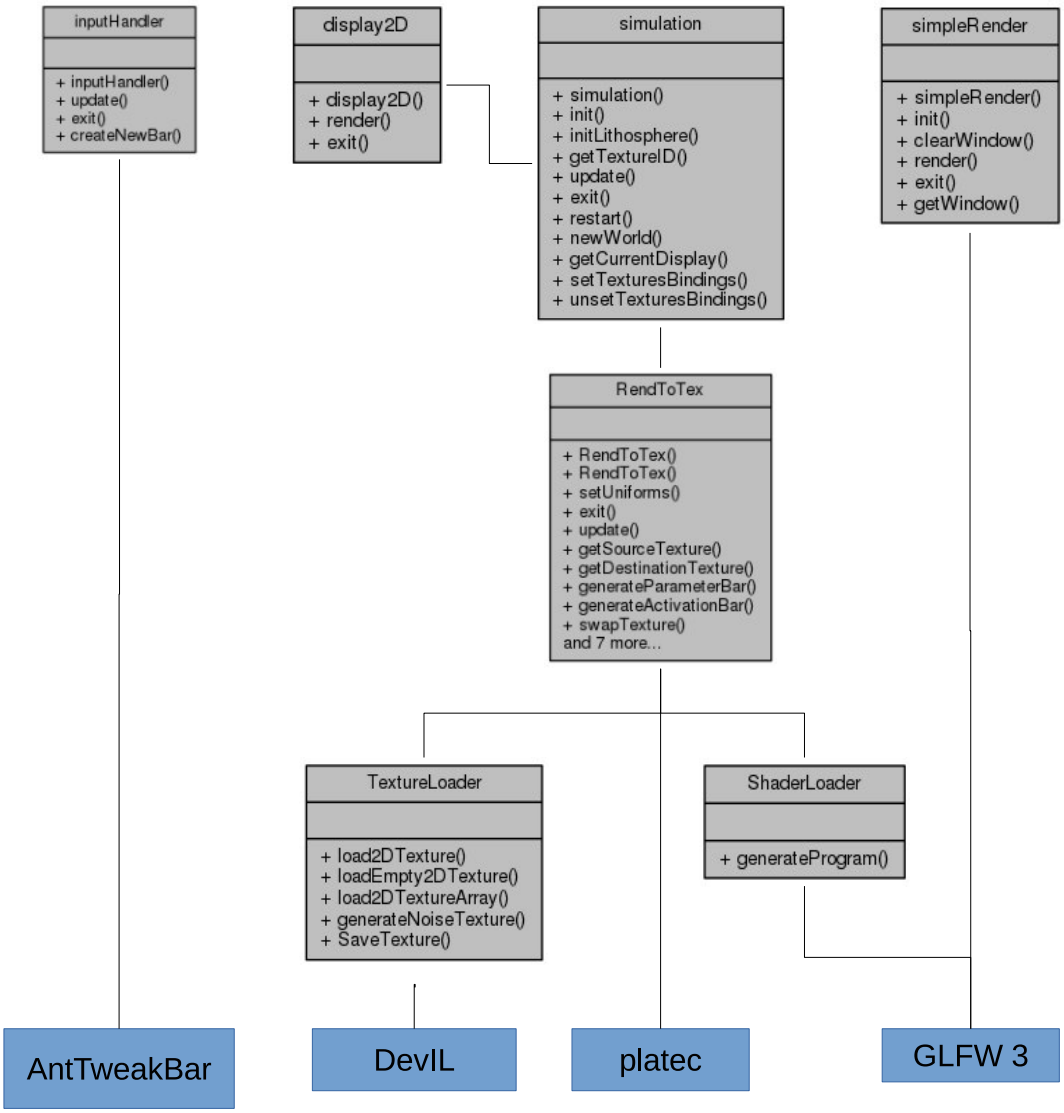


Abbildung 4.2 Klassendiagramm

TextureLoader

Die TextureLoader Klasse ist für das laden, speichern und generieren von Texturen zuständig. Generierte und geladene Texturen werden auf der GPU gespeichert und sind mittels einer ID von der CPU aus referenzierbar. Hier können auch bereits existierende Höhenkarten geladen und als Ausgangslage für die Simulation verwendet werden. Des Weiteren können leere, also komplett schwarze, Texturen generiert werden. Diese werden für den Start aller Simulationsteile benötigt. Auch die fraktalen Texturen für die Höhenkartengenerierung und die Windsimulation werden im TextureLoader implementiert. Hierfür wird auf die Bibliothek *libNoise* zurückgegriffen, welche umfangreiche Optionen für das Generieren von fraktalen Texturen bietet. Zuletzt können Texturen von der GPU zurück auf die Festplatte geschrieben werden, um einen aktuellen Zustand aus der Simulation zu exportieren. Hierbei wird die Ausgangstextur der Simulation so angepasst, dass diese Werte zwischen Null und Eins haben, damit sie in der Textur darstellbar sind.

4.2.2 OpenGL Handling

SimpleRender

Um die grundlegenden Elemente für das Rendering kümmert sich die SimpleRender-Klasse. Dabei initialisiert diese die Komponenten, welche für das Rendering notwendig sind, erzeugt ein leeres Fenster und zeichnet darin einen schwarzen Hintergrund. Als Bibliotheken für die Hardwareabstraktion dient GLFW3, welches die Erzeugung des OpenGL Kontextes sowie das Handling für das Fenster übernimmt, sowie GL3W, welche die OpenGL Shader Erweiterungen zur Verfügung stellt.

RendToTex

In der RendToTex Klasse ist das *Offscreen Rendering* (siehe Abschnitt 4.1.2) sowie das *Double-Buffering* (siehe Abschnitt 4.1.3) implementiert. Hierfür werden über den ShaderLoader und den TextureLoader die entsprechenden Daten generiert und die Schnittstellen für das *Offscreen Rendering* initialisiert. Des Weiteren können für die Simulation die anpassbaren Parameter übergeben werden. Durch die Updatefunktion wird das Shader-Programm ausgeführt und die Textur zwischengespeichert. Mittels entsprechenden get- und set-Funktionen können die Parameter und die Textur zur Laufzeit manipuliert werden.

Simulation

Die Simulationsklasse konfiguriert die Simulation. Hier werden alle Shader, Parameter, Texturen und GUI-Elemente initialisiert und zugeordnet. Auch die Funktionalität der Plattentektonik Bibliothek *platec* wird über die Klasse angesprochen. In der Update-Funktion wird ein vollständiger Simulationsschritt durchgeführt. Des Weiteren kann mit den *restart* und *newWorld* Funktionen die Simulation neu gestartet werden, wahlweise mit der bisherigen Welt oder einer neuen.

Display2D

Die Klasse Display2D ist für die Bildschirmausgabe der Texturen zuständig. Das entsprechende Shaderprogramm wird hier geladen und mittels einer Variable wird festgelegt, welche

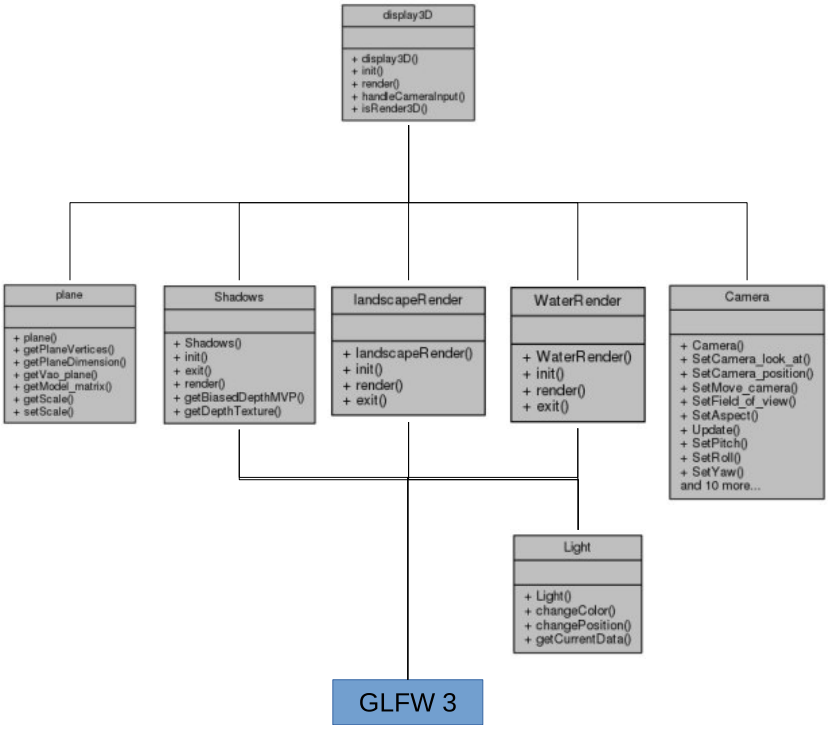


Abbildung 4.3 Klassendiagramm für 3D Darstellung

Simulationsebene gerade angezeigt werden soll. Der Shader ist so ausgelegt, dass man das Fenster beliebig skalieren kann und die Texturen entsprechend interpoliert werden. Dies ist nur für die Darstellung möglich, die Simulationsgröße wird hierbei nicht beeinflusst.

4.2.3 Inputhandling

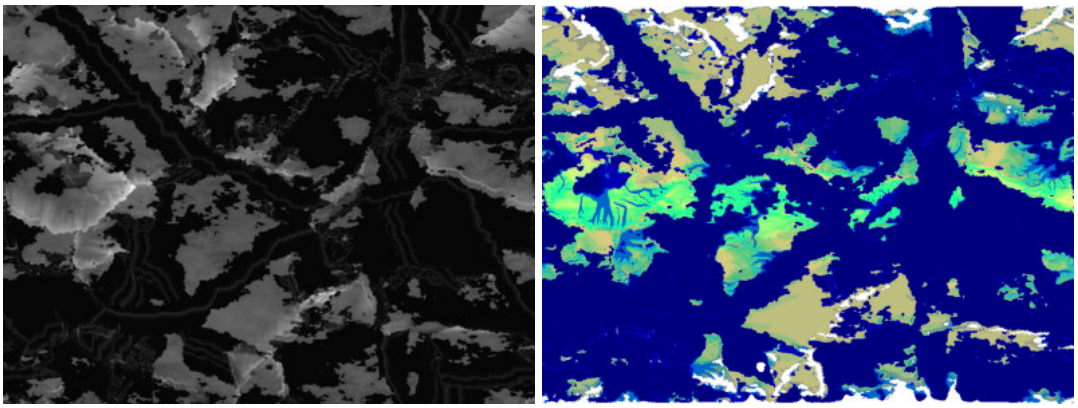
Im Inputhandling werden Tastatur- und Mauseingaben verarbeitet. Die verwendete Bibliothek GLFW3 bietet hierfür bereits passende Schnittstellen. Mittels Callbacks werden die Eingaben abgefangen und an die für GUI-Elemente zuständige Bibliothek *AntTweakBar* weitergeleitet. Auch die Kamerasteuerung wird hier behandelt. Sobald die Shift-Taste gedrückt ist, kann mittels der Maus und der Tastatur die Kamera für die 3-D-Ansicht bewegt werden.

4.2.4 3D Rendering

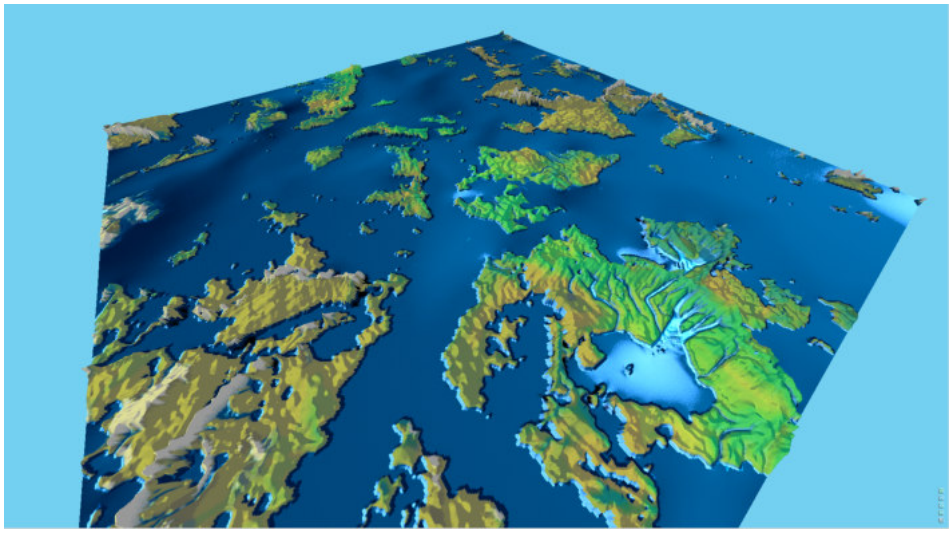
Für eine bessere Visualisierung der Simulation und der generierten Landschaft wurde eine einfache 3-D Ansicht implementiert. In Abbildung 4.3 ist das Klassendiagramm hierfür zu sehen. Zwischen der 2-D und der 3-D Ansicht kann zur Laufzeit gewechselt werden.

LandscapeRender

Die LandscapeRender Klasse kümmert sich um das Rendern der 3-D Landschaft. Hier werden die Shader für das *Displacement Mapping* [AMHH08] geladen. Bei dieser Technik wird eine Ebene in viele Polygone unterteilt und anschließend anhand eines Höhenwerts an der y-Achse verschoben. Die für die Darstellung notwendigen Parameter, wie die Schattentextur, Lichtposition, Kameraposition und einen einstellbaren Höhenparameter, werden in der



(a) Höhenkarte **(b)** Klimazonen



(c) 3D Visualisierung

Abbildung 4.4 Beispiel einer 3D Visualisierung

render-Funktion an die GPU übergeben. Die Farbdarstellung der Landschaft ist abhängig von der ausgewählten Simulationsebene. In der zusätzlich implementierten 3-D Ansicht wird die Klimazone als Farbwert übergeben. In Abbildung 4.4 ist ein Beispiel für die 3-D Darstellung. Anhand der Höhenkarte (Abbildung 4.4a) wird die Struktur der 3-D Landschaft generiert.

Schatten

Um Höhenunterschiede in der Landschaft besser sichtbar zu machen, wurde eine einfache Schattenberechnung implementiert. Die hier verwendete Technik ist das sogenannte *Shadow Mapping* [Wil78]. Hierbei wird die Landschaft einmal aus Sicht des Lichtes gezeichnet und die kleinsten Abstände zwischen einem Objekt und einem Pixel in der Kamera in einer Textur gespeichert. Diese Textur wird anschließend im eigentlichen Rendervorgang so transformiert, dass diese zur Kameraposition passt und in die Farbberechnung mit einbezogen. Zusätzlich zum *Shadow Mapping* wird zur Beleuchtung der Landschaft das Phong-Beleuchtungsmodell [Pho75] eingesetzt. In der Abbildung 4.5 sieht man deutlich, wie die vom Licht abgewandten Seite der Bergkette dunkler dargestellt wird, sowie die dazugehörigen Schatten auf dem

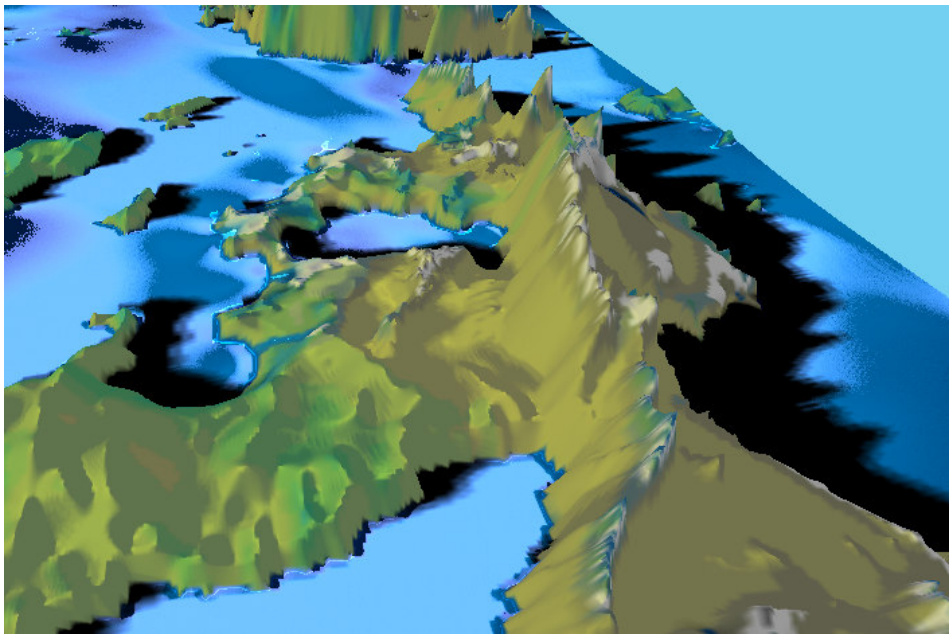


Abbildung 4.5 Darstellung von Schatten

Wasser. Außerdem lassen sich die Konturen der Landschaft deutlich erkennen.

WaterRender

Zusätzlich zur 3-D Landschaft wird auch das Wasser aus der Wassersimulation (Abschnitt 3.3) gerendert. Zuerst wird die Wasserhöhe ebenfalls mittels *Displacement Mapping* dargestellt. Die Höhe hierbei ist die Summe der aktuellen Höhen von Gestein, Sand und Wasser. Um Artefakte bei niedrigen Wasserständen zu vermeiden, bekommt die Wasserdarstellung einen zusätzlichen Alpha-Wert für die Transparenz. Unter einem bestimmten Wasserstand wird das Wasser komplett durchsichtig dargestellt, was die Anzahl der Artefakte deutlich verringert. Um die Fließrichtung des Wassers darzustellen, wird das Wasser zuerst mittels *Bump Mapping* beleuchtet. Hierbei verwendet man eine zusätzliche Textur, welche die Reflexionen der Wasseroberfläche so verändert, dass diese plastischer wirkt, wie in Abbildung 4.6 zu sehen ist. Die Textur für das *Bump Mapping* wird nun zeitlich in die Fließrichtung verschoben. Zuerst wurde versucht, zwischen den Datenpunkten die Fließrichtung zu interpolieren, was jedoch zu Alasing-Artefakten geführt hat und stellenweise so aussah, als würde sich das Wasser Bergauf fließen. Nun wird die Bump-Textur verkleinert und oft innerhalb eines Datenpunktes wiederholt. Bei genauerem hinsehen sind die unregelmäßigen Übergänge sichtbar, dafür ist die Fließrichtung klar zu erkennen.

Display3D

Die Display3D Klasse ist die Schnittstelle zum 3D Rendering. Die Klasse initialisiert und verwaltet alle für das 3D-Rendering verwendeten Klassen. Des Weiteren verändert sie die Kameraposition anhand der Maus- und Tastatureingaben.

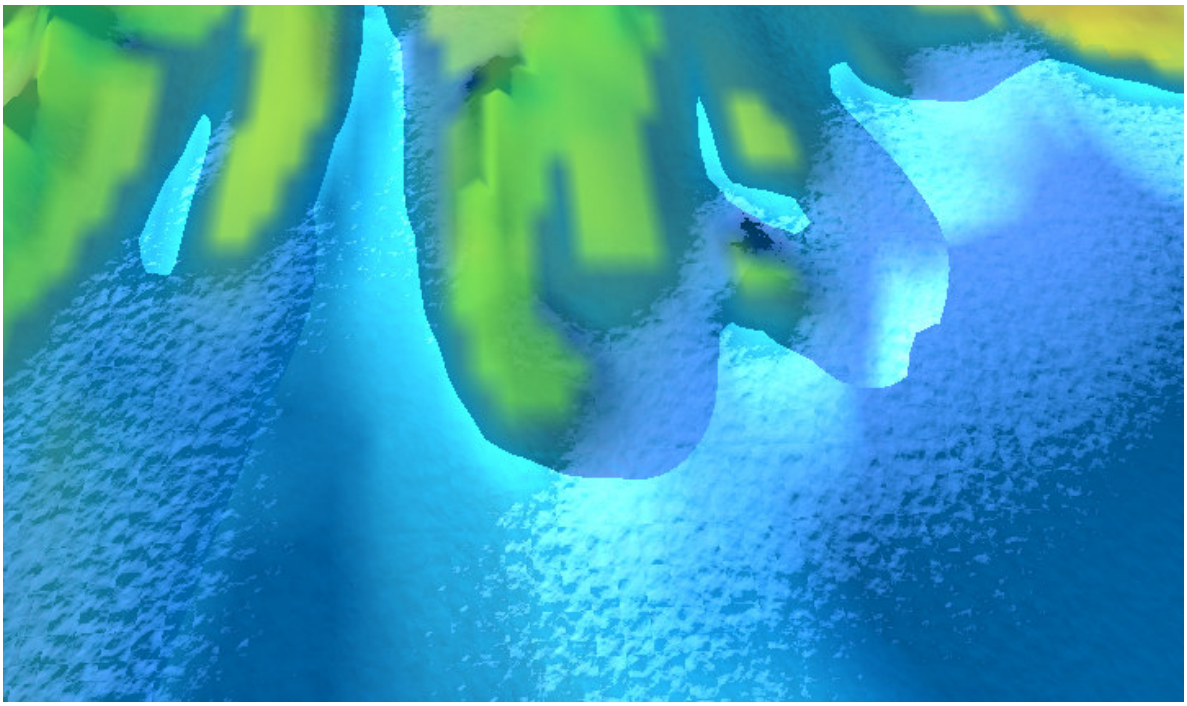


Abbildung 4.6 Darstellung von Wasser

Tabelle 4.1 Kenngrößen der Testhardware

	Quadro M1000M	GT 520	GTX 1060
Shaderkerne	96	48	1280
Taktung (MHz)	1124	810	1506
Texel fill Rate (GT/s)	11.2	6.5	82
Speicherbandbreite (GB/s)	80	14.4	192

4.3 Laufzeitverhalten

Da die Simulation, ausgenommen der Plattentektonik, ausschließlich auf der GPU berechnet wird, ist diese auch der ausschlaggebende Faktor für das Laufzeitverhalten. Die Geschwindigkeitsangaben sind hierbei in Bilder pro Sekunde angegeben, wobei die Erzeugung eines Bildes einem kompletten Simulationsschritt entspricht. Da die Berechnungen auf der GPU nur im *Fragmentshader* stattfinden, sind die beiden entscheidenden Leistungsmerkmales die Texel Fill Rate, wie viele Texturpixel pro Sekunde verarbeitet werden, sowie die Speicherbandbreite. Die Perfomancetests wurden auf drei verschiedenen Grafikkarten mit variablen Texturgrößen simuliert und die durchschnittliche Bildrate nach 3 Minuten erfasst. Ziel der Arbeit war es, die Echtzeifähigkeit mit optimal 60 Bildern pro Sekunde, aber mindesten 30 Bildern pro Sekunde zu erzeugen. Diese Schwellwerte sind in Abbildung 4.7 mit einer gelben und roten Linie gekennzeichnet.

Tabelle 4.2 Messergebnisse in FPS

Texturgröße	Quadro M1000M	GT 520	GTX 1060
800x600	111	105	333
1024x600	90	100	333
1200x800	71	89	250
1600x900	52	66	200
1920x1080	38	53	142
2880x1620	19	3,5	93
3840x2160	10	1,5	43

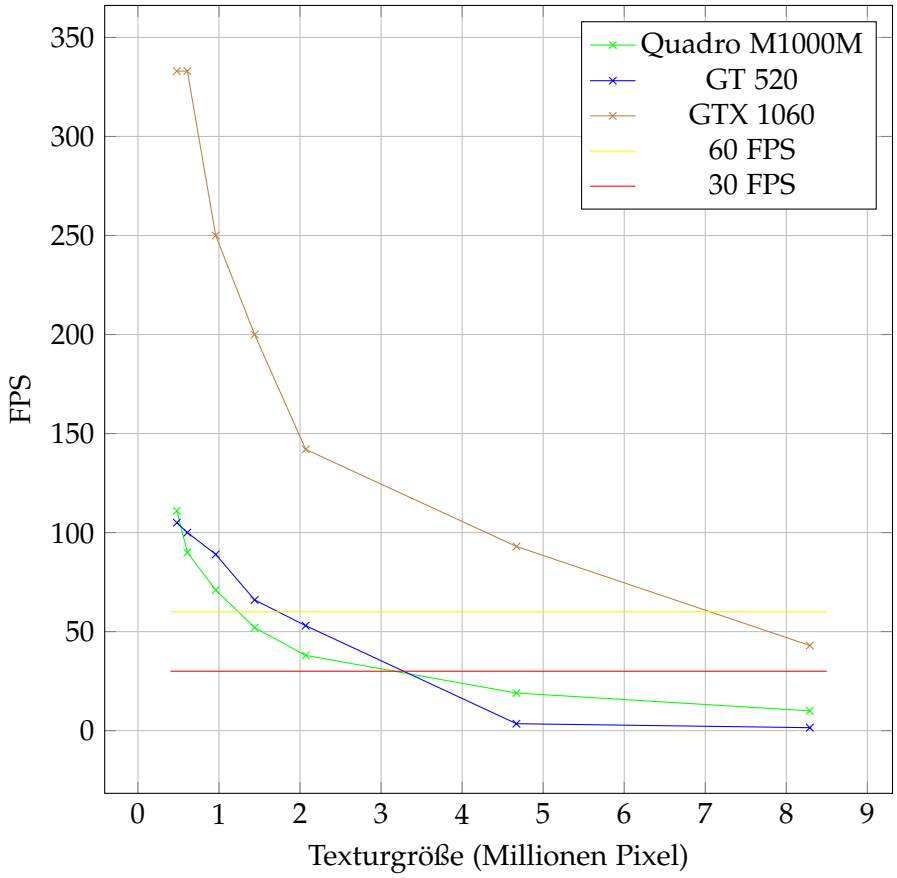


Abbildung 4.7 Darstellung der Messergebnisse

5 Bewertung der Arbeit

5.1 Pro

5.1.1 Abwechslungsreichere Landschaften

Durch das Verfahren, welches in dieser Arbeit entwickelt wurde, lassen sich abwechslungsreiche und anpassbare Landschaften generieren. Im Gegensatz zu den durch fraktale Algorithmen generierten Landschaften sind hierbei Flussbette, Klippen, lang gezogene Bergketten und realistische Klimazonen erzeugbar. An die Komplexität der manuell erstellten Landschaften kommt dieses Verfahren nicht heran. Jedoch kann es als Experimentierfeld, Inspirationsquelle oder Ausgangslage für Designer benutzt werden.

5.1.2 Echtzeitdynamik und Manipulation

Ein weiterer Vorteil dieser Arbeit ist, dass sich viele Parameter zur Laufzeit manipulieren lassen. Das Verhalten jeder Simulationsebene kann genau eingestellt werden und die Auswirkungen sind in kürzester Zeit sichtbar. Durch die Echtzeitdynamik ist es möglich, den Veränderungsprozess der Landschaft zu beobachten und zu beeinflussen. Berge entstehen und vergehen, Flüsse graben sich durch die Landschaft und trockenen aus, sogar Eiszeiten und geosynchrone Sonnenorbits sind darstellbar. Dies ist ein großer Vorteil gegenüber den bisherigen Generierungsverfahren, die meist mit einem nach dem Start unveränderbaren Parametersatz arbeiten.

5.1.3 Leicht erweiterbar

Die in dieser Arbeit vorgestellten Simulationsebenen stellen nur einen Bruchteil aller in der Natur vorkommenden Effekte dar. Die Simulation kann hierfür noch verfeinert und durch zusätzliche Ebenen, wie z.B. Stickstoffverteilung oder Wassertemperatur, erweitert werden. Durch das leicht erweiterbare Softwaredesign müssen für Veränderungen nur die Shader angepasst oder ergänzt werden, was keine Neukompilierung der Software erfordert. Bei dem Hinzufügen von neuen Ebenen muss bei dem hier vorliegenden Zustand die Software, zusätzlich zum Shader, noch etwas angepasst werden. Grundsätzlich könnte diese Erweiterungen auch über eine Konfigurationsdatei erfolgen, was jedoch aus zeitlichen Gründen nicht mehr implementiert wurde.

5.2 Kontra

5.2.1 Momentaufnahmen

Durch die Echtzeitdynamik dieser Arbeit sind sämtliche Zustände der Simulation Momentaufnahmen. Zwar lässt sich die Simulation pausieren, jedoch verhindert dies nicht, dass der vorhergehende Simulationsschritt nicht wiederherstellbar ist. Eine mögliche Lösung für

dieses Problem wäre, eine zusätzliche Aufnahmefunktion zu implementieren, welche den Verlauf der Simulation aufzeichnet.

5.2.2 Reproduzierbarkeit

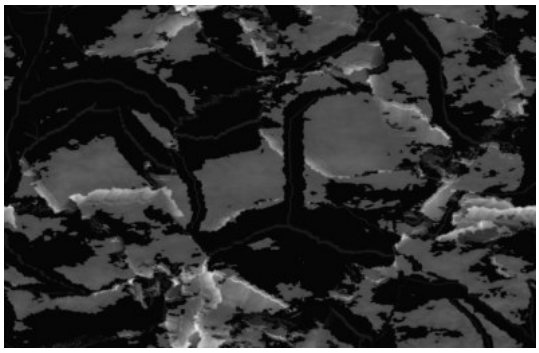
Ein weiterer Nachteil der Echtzeitdynamik ist, dass die Landschaften nicht reproduzierbar sind. Zwar kann die Simulation mit denselben Ausgangsparametern starten, jedoch bereits kleinste Rundungsfehler können Auswirkungen auf das Ergebnis der Simulation haben.

5.2.3 Variable Performance

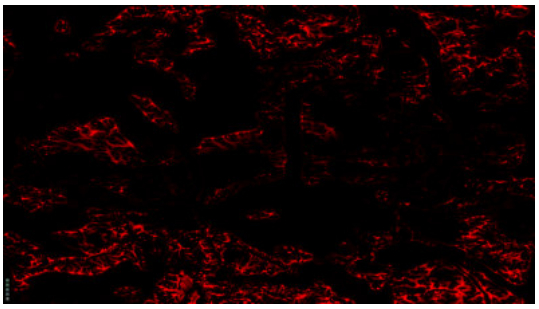
Die Geschwindigkeit der Simulation ist abhängig von der Grafikkarte. Wenn diese für die erwartete Texturgröße nicht genügend Leistung aufweist, läuft die Simulation entsprechend langsamer. Auch bei extrem hohen Bildwiederholraten spielt sich die Simulation kaum noch nachvollziehbar ab. Letzterem kann jedoch mittels einer Begrenzung der Bildwiederholrate abgeholfen werden.

5.3 Ergebnis der Arbeit

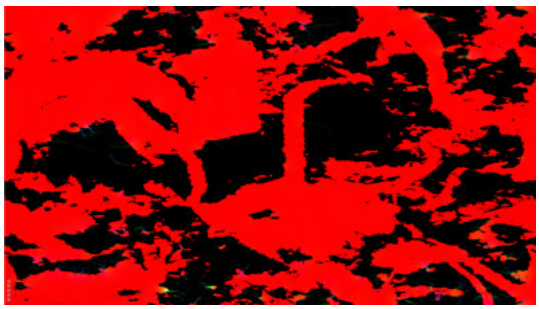
Das Ziel dieser Arbeit war, eine neue Methode für das generieren von Landschaften zu implementieren. Diese sollte realistische Landschaften durch vereinfachte Simulationen generieren und dabei in Echtzeit anpassbare Parameter zur Verfügung stellen.
Diese Ziele wurden alle erreicht. In Abbildung 5.1 und Abbildung 5.2 ist das Ergebnis einer Simulation für alle Simulationsebenen sowie die zugehörige 3-D Ansicht zu sehen. Die erstellten Höhenkarten weisen eine deutlich größere Abwechslung auf als jene, die mittels Diamond-Square-Algorithmus erstellt werden. Auch die Verteilung von Klimazonen, Gewässern und Berglandschaften sind deutlich realistischer, als bei fraktal erzeugten Welten. Die Echtzeitfähigkeit ist bei entsprechender Hardware und bestimmten Texturgrößen erreichbar. Natürlich müssen die Simulationen, deren Interaktionen und die Anzahl der Simulationen noch ausgebaut und verfeinert werden, um realitätsnähere Welten generieren zu können. Doch dient die Arbeit als Grundlage, auf die zukünftige Arbeiten aufbauen können.



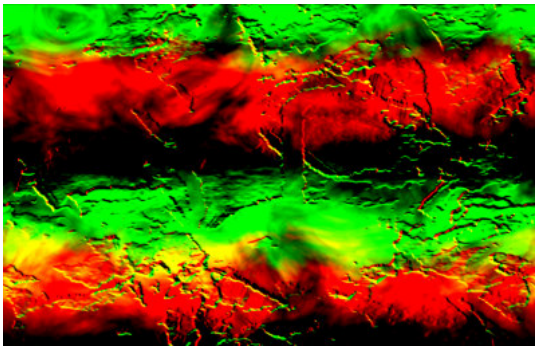
(a) Höhenkarte



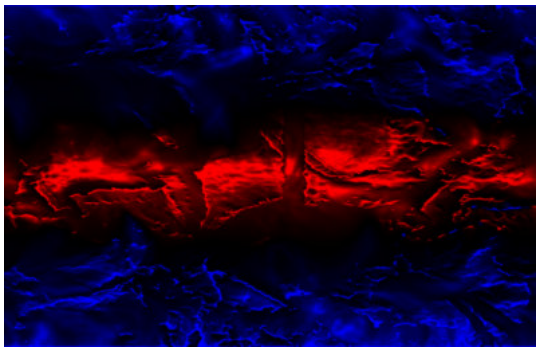
(b) Sand



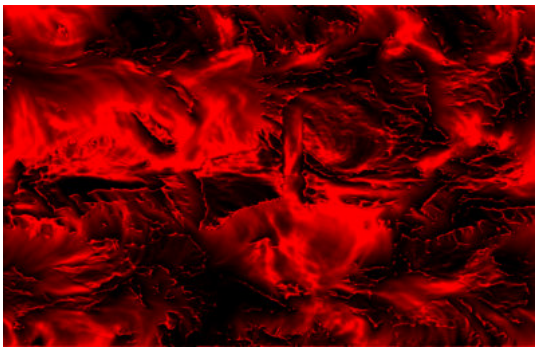
(c) Wasser



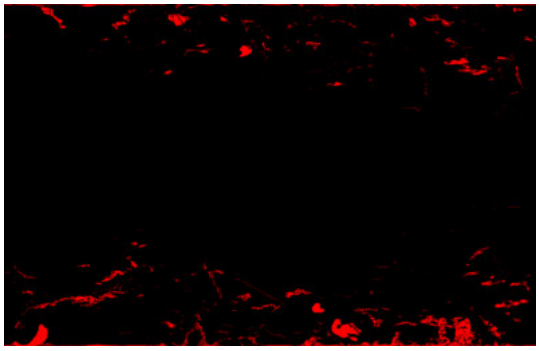
(d) Wind



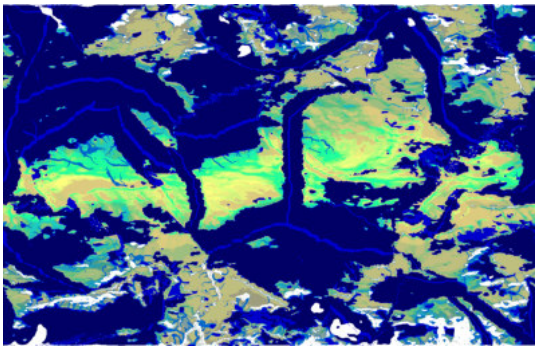
(e) Temperatur



(f) Luftfeuchtigkeit



(g) Eis



(h) Klimazonen

Abbildung 5.1 Ein Ergebnis der Arbeit

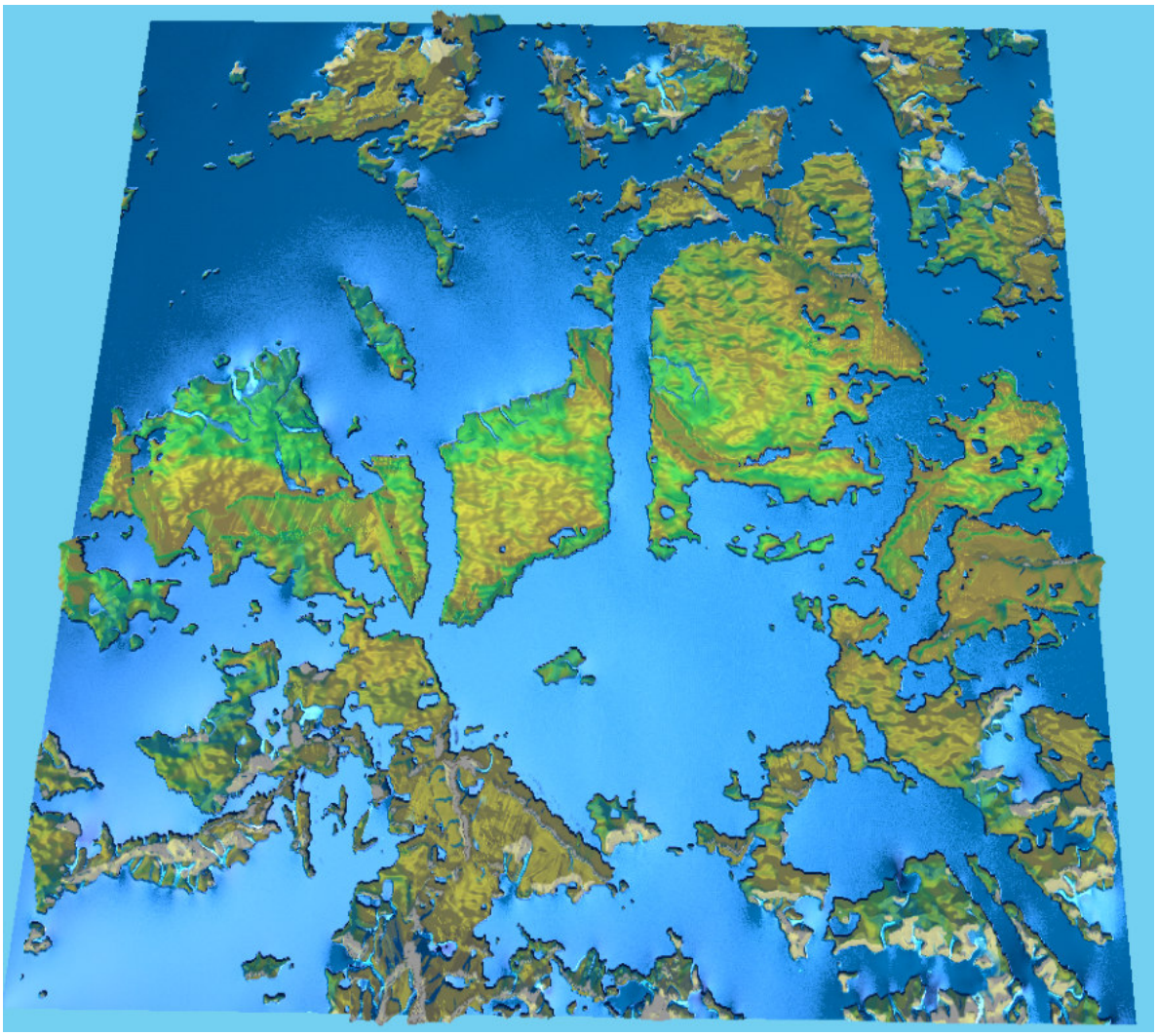


Abbildung 5.2 3D Darstellung des Ergebnisses

Literaturverzeichnis

[AMHH08] Akenine-Möller, Tomas ; Haines, Eric ; Hoffman, Naty: *Real-Time Rendering 3rd Edition*. Natick, MA, USA : A. K. Peters, Ltd., 2008. – ISBN 987–1–56881–424–7

[Asc14] Aschenbach, Nick: *2D Fractal Terrain Generation*. http://nick-aschenbach.github.io/blog/2014/07/06/2d-fractal-terrain/. Version: 2014. – Online, zugegriffen am 17.Februar 2017

[Bea10] Beard, Daniel: *Terrain Generation – Diamond Square Algorithm*. https://danielbeard.wordpress.com/2010/08/07/terrain-generation-and-smoothing/. Version: 2010. – Online, zugegriffen am 17.Februar 2017

[Bec17] Beccario, Cameron: *a visualization of global weather conditions*. https://earth.nullschool.net/#current/wind/surface/level/patterson. Version: 2017. – Online, zugegriffen am 12.Februar 2017

[Boe11] Boesch, Florian: *WebGL GPU Landscaping and Erosion*. http://codeflow.org/entries/2011/nov/10/webgl-gpu-landscaping-and-erosion/. Version: 2011. – Online, zugegriffen am 5.Februar 2017

[DPH13] Daniel P. Huffman, Tom P.: *The Design of Gray Earth: A Monochrome Terrain Dataset of the World*. http://cartographicperspectives.org/index.php/journal/article/view/cp74-huffman-patterson/623. Version: 2013. – Online, zugegriffen am 12.Februar 2017

[Eng] Engine, Unity: *Terrain Engine Guide: Height*. https://docs.unity3d.com/412/Documentation/Components/terrain-Height.html. – Online, zugegriffen am 2.Februar 2017

[GGG+13] Génevaux, Jean-David ; Galin, Éric ; Guérin, Eric ; Peytavie, Adrien ; Benes, Bedrich: Terrain Generation Using Procedural Models Based on Hydrology. In: *ACM Trans. Graph.* 32 (2013), Juli, Nr. 4, 143:1–143:13. http://dx.doi.org/10.1145/2461912.2461996. – DOI 10.1145/2461912.2461996. – ISSN 0730–0301

[Hol47] Holdridge, L. R.: Determination of World Plant Formations From Simple Climatic Data. In: *Science (New York, N.Y.)* 105 (1947), April, Nr. 2727, 367–368. http://dx.doi.org/10.1126/science.105.2727.367. – DOI 10.1126/science.105.2727.367. – ISSN 0036–8075

[JT11] Jákó, Balázs ; Tóth, Balázs: Fast Hydraulic and Thermal Erosion on GPU. In: Avis, N. (Hrsg.) ; Lefebvre, S. (Hrsg.): *Eurographics 2011 - Short Papers*, The Eurographics Association, 2011. – ISSN 1017–4656, S. 057–060

[Khr] KHRONOS: *Transformations*. https://www.opengl.org/archives/resources/faq/technical/transformations.htm#tran0170n/. – Online, zugegriffen am 5.Februar 2017

[Lin16] LINK, Andreas: *No Man's Sky zu mathematisch und deswegen langweilig - sagt Geoff Keighley*. http://www.pcgameshardware.de/No-Mans-Sky-Spiel-16108/News/ist-zu-mathematisch-und-deswegen-langweilig-1209581/. Version: 2016. – Online, zugegriffen am 15.Februar 2017

[Man77] MANDELBROT, Benoit B.: *Fractals: Form, chance, and dimension*. New York : W. H. Freedman and Co., 1977

[MDH07] MEI, Xing ; DECAUDIN, Philippe ; HU, Baogang: Fast Hydraulic Erosion Simulation and Visualization on GPU, 2007

[mil] MILL, Graphics: *Minimum, maximum and median filters*. https://www.graphicsmill.com/docs/gm/minimum-maximum-median-filters.htm. – Online, zugegriffen am 17.Februar 2017

[Mil86] MILLER, Gavin S P.: The Definition and Rendering of Terrain Maps. In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM, 1986 (SIGGRAPH '86). – ISBN 0–89791–196–2, Seite 39–48

[NAS02] NASA: *Blue Marble: Clouds*. http://visibleearth.nasa.gov/view.php?id=57747. Version: 2002. – Online, zugegriffen am 12.Februar 2017

[Oni09] ONISHI, A.: Integrated Global Models of Sustainable Development - Volume III:, EOLSS, 2009. – ISBN 9781905839193, S. 47–67

[Pat] PATTERSON, Tom: *Natural Earth III – Extra Data*. http://www.shadedrelief.com/natural3/pages/extra.html. – Online, zugegriffen am 12.Februar 2017

[Pho75] PHONG, Bui T.: Illumination for Computer Generated Pictures. In: *Commun. ACM* 18 (1975), Juni, Nr. 6, 311–317. http://dx.doi.org/10.1145/360825.360839. – DOI 10.1145/360825.360839. – ISSN 0001–0782

[SA09] SEGAL, Mark ; AKELEY, Kurt: The OpenGL Graphics System: A Specification (Version 3.1) / The Khronos Group Inc. 2009. – Forschungsbericht

[SG10] SILVA, Alisson R. ; GOUVÊA, Maury M. Jr.: Cloud Dynamics Simulation with Cellular Automata. In: *Proceedings of the 2010 Summer Computer Simulation Conference*. San Diego, CA, USA : Society for Computer Simulation International, 2010 (SCSC '10), 278–283

[SWH15] SELLERS, Graham ; WRIGHT, Richard S. ; HAEMEL, Nicholas: *OpenGL Superbible: Comprehensive Tutorial and Reference*. 7. Auflage. Addison-Wesley Professional, 2015. – ISBN 0672337479, 9780672337475

[TDR+11] Trunfio, Giuseppe A. ; D'Ambrosio, Donato ; Rongo, Rocco ; Spataro, William ; Di Gregorio, Salvatore: A New Algorithm for Simulating Wildfire Spread Through Cellular Automata. In: *ACM Trans. Model. Comput. Simul.* 22 (2011), Dezember, Nr. 1, 6:1–6:26. http://dx.doi.org/10.1145/2043635.2043641. – DOI 10.1145/2043635.2043641. – ISSN 1049–3301

[Vii12] Viitanen, Lauri: Physically Based Terrain Generation. (2012). http://www.theseus.fi/handle/10024/40422

[von51] von Neumann, John: The general and logical theory of automata. In: Jeffress, L. A. (Hrsg.): *Cerebral Mechanisms in Behavior – The Hixon Symposium*. John Wiley & Sons, 1951, S. 1–31

[Wil78] Williams, Lance: Casting Curved Shadows on Curved Surfaces. In: *SIGGRAPH Comput. Graph.* 12 (1978), August, Nr. 3, 270–274. http://dx.doi.org/10.1145/965139.807402. – DOI 10.1145/965139.807402. – ISSN 0097–8930

Abbildungsverzeichnis

Tabellenverzeichnis