Hochschule Darmstadt
- Fachbereich Informatik -

Simulation von geologischen Alterungs- und Erosionsprozessen am Beispiel fraktaler Landschaften

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

vorgelegt von

Marco Münch B.Sc.

Referent: Prof. Dr. Wolf-Dieter Groch

Korreferentin: Prof. Dr. Elke Hergenröther

Ausgabedatum: 30.09.2010
Abgabedatum: 30.03.2011

Erklärung

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

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 3. April 2011

Marco Münch

Abstrakt

Ziel dieser Arbeit ist es, einen Überblick über die vorhandenen Verfahren der Erzeugung von fraktalen Landschaften zu geben. Auf diesen erzeugten Landschaften sollen Erosionsprozesse einwirken, um sie natürlicher Aussehen zu lassen.

Die in dieser Arbeit vorgestellten Erosionsprozesse sind die Simulation von fließendem und stehendem Wasser und das Abrutschen von Gesteinsmaterial durch die sogenannte thermale Erosion. Diese werden mit den entsprechenden Algorithmen vorgestellt und einige davon erweitert.

Als Alterungsprozess wurde für diese Arbeit der geologische Gesteinskreislauf vereinfacht. Damit ist es möglich, das Altern der Gesteine zu simulieren, was zur Folge hat, dass die Erosionsprozesse anders wirken.

Des Weiteren wurde ein Generator entwickelt, mit dessen Hilfe es möglich ist, eine Landschaft zu erzeugen, die schon über mehrere Gesteinsschichten verfügt.

Auf Grund diesen erarbeiteten Algorithmen wurde ein Konzept zur Umsetzung in das an der Hochschule Darmstadt entwickelte Framework GLAB erarbeitet und umgesetzt.

Inhaltsverzeichnis

Until a few years ago, the topics in my Ph.D. were unfashionable, but they are very popular today.

Benoît Mandelbrot

Kapitel 1

Einleitung

Benoît Mandelbrot wird als Vater der fraktalen Mathematik angesehen. In seinem 1982 erschienen Buch „The Fractal Geometry of Nature“ [Man82] beschrieb er zum ersten Mal den Begriff Fraktal, welches sich aus dem lateinischen Wort „fractus“ (zu Deutsch: „gebrochen“) ableitet. Darin beschreibt er Körper, die sich aus kleineren Versionen ihrer selbst zusammensetzen. Aber schon bevor der Begriff Fraktal definiert wurde, sind Forscher auf fraktale Gebilde gestoßen. Ein bekanntes Fraktal ist das 1915 vom Mathematiker Waclaw Sierpiński beschriebene Sierpinski-Dreieck, welches in Abbildung 1.1 zu sehen ist. Das Dreieck besteht aus drei kleineren Versionen seiner selbst: einmal oben in der Mitte, einmal links unten und rechts unten (blaue Kästchen). Diese kleineren Versionen bestehen aber wiederum aus kleineren Versionen (rote Kästchen). Diese Abbildung kann im Prinzip bis ins Unendliche fortgesetzt werden.

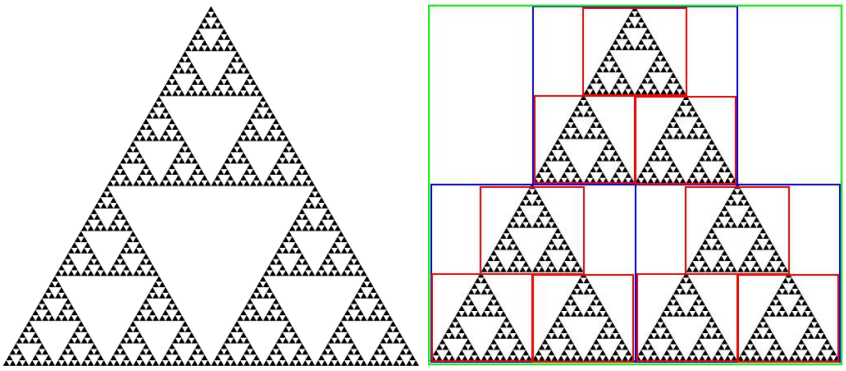

Abbildung 1.1: Sierpinski-Dreieck[1]

Nicht nur reine Mathematiker sind auf die fraktalen Körper gestoßen, sondern auch Forscher anderer Fachgebiete. Edward N. Lorenz ist am Anfang der 1960er bei der Simulation von Wetterdaten auf den Lorenz-Attraktor gestoßen. 1973 hat der deutsche Biochemiker Otto E. Rössler den nach ihm benannten Rössler-Attraktor entdeckt. Es wurden noch viele weitere fraktale Gebilde erzeugt, die schließlich alle unter dem von Mandelbrot beschriebenen Begriff Fraktal zusammengefasst werden. Ein Teilgebiet der fraktalen Mathematik, das keine einmaligen Gebilde generiert, sind die fraktalen Landschaften.

1.1 Fraktale Landschaften

Die Haupteigenschaft eines Fraktals ist seine Selbstähnlichkeit, dessen Ursprung in der Natur zu finden ist. Wie in Abbildung 1.2 zu sehen ist, scheint ein Romanesco (Unterart des Blumenkohls) oder ein Farn immer aus kleineren Versionen seiner selbst zu bestehen.

Genau dieses Prinzip machen sich fraktale Landschaften zunutze. So sieht seine Felswand aus der Ferne einem Sandhaufen aus der Nähe ähnlich. Genau dieser Vergleich wird in der Abbildung 1.3 verdeutlicht. Während im linken Bild eine Sandsteinwand aus dem Spessart dargestellt wird, ist im

[1] http://www.wikipedia.de/Sierpinski-Dreieck

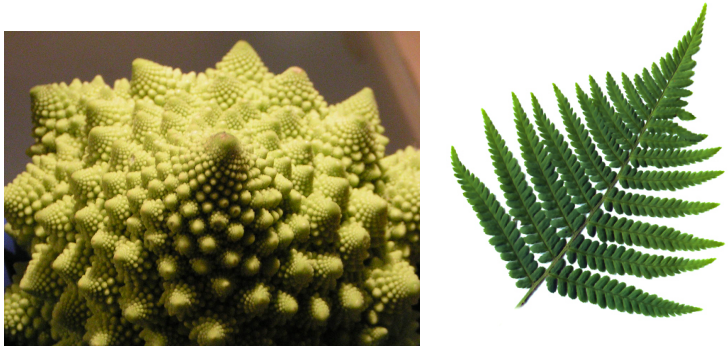

Abbildung 1.2: Selbstähnlichkeit in der Natur[2]

rechten Bild eine Nahaufnahme eines Sandhaufens zu sehen.

Abbildung 1.3: Selbstähnlichkeit bei Landschaften

Die ersten Ansätze zur Generierung von künstlichen fraktalen Landschaften wurden Ende der 1970er von Benoît Mandelbrot beschrieben. Auf den Ansätzen der englischen Übersetzung seines Essays *Les objets fractals, forme, hasard et dimension* [Man75] [Man77] von 1975 generierte Richard F. Voss 1977 eines der ersten Bilder einer fraktalen Landschaft (siehe Bild 1.4).

Nachdem dieses Bild veröffentlicht wurde arbeitete Richard F. Voss weiterhin an der visuellen Darstellung fraktaler Landschaften, wobei weitere Bilderreihen generiert wurden. So wie in Abbildung 1.5 zu sehen, in der die gleiche Urlandschaft mit verschiedenen Parametern zu sehen ist.

Benoît Mandelbrot war so begeistert von diesen Bildern, dass er sie

[2]http://www.waldoberschule.de, http://www.aromatisches-blog.de/
[3]http://classes.yale.edu/fractals/panorama/Art/MountainsSim/Classical/Classical.html

Abbildung 1.4: Planet Rise von Richard F. Voss [Man82]

Abbildung 1.5: Mountain Simulation von Richard F. Voss[3]

1982 in seinem bekanntesten Werk *The Fractal Geometry of Nature* [Man82] abdrucken und neue Bilder generieren ließ (Abbildung 1.6).

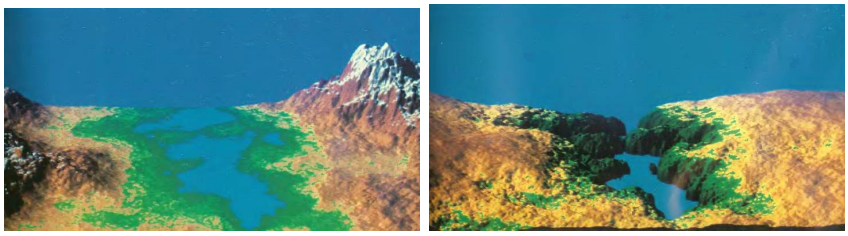

Abbildung 1.6: Fluss von Richard F. Voss [Man82]

Zwei Jahre zuvor sorgte der Kurzfilm *Vol Libre* von Loren Carpenter auf der SIGGRAPH[4] computer graphics conference für Furore. In diesem Film wurde zum ersten Mal die Generierung einer Landschaft als fortlaufender Film gezeigt. Ein Ausschnitt der Generierung wird in Abbildung 1.7 gezeigt.

Das Video wurde über die Konferenz hinaus sehr bekannt. Loren Carpenter durfte 1982 für den Film „Star Trek 2 - Der Zorn des Khan“ die erste computeranimierte Sequenz der Filmgeschichte generieren. In dieser Szene wurde ein Planet künstlich erschaffen.

Seit diesen Anfängen wurden verschiedene Algorithmen zur Generierung von Landschaften entwickelt. Vor allem ist der technische Fortschritt bei der Visualisierung dieser Landschaften zu sehen, weniger bei den fraktalen Ansätzen.

Heutzutage liegt der Einsatz der Generierung von realistisch aussehenden Landschaften vor allem in der Medienbranche, der Computer-Spiele-Branche oder der Branche für Flugsimulationen zur Pilotenausbildung.

[4] **S**pecial **I**nterest **G**roup on **GRAPH**ics and Interactive Techniques

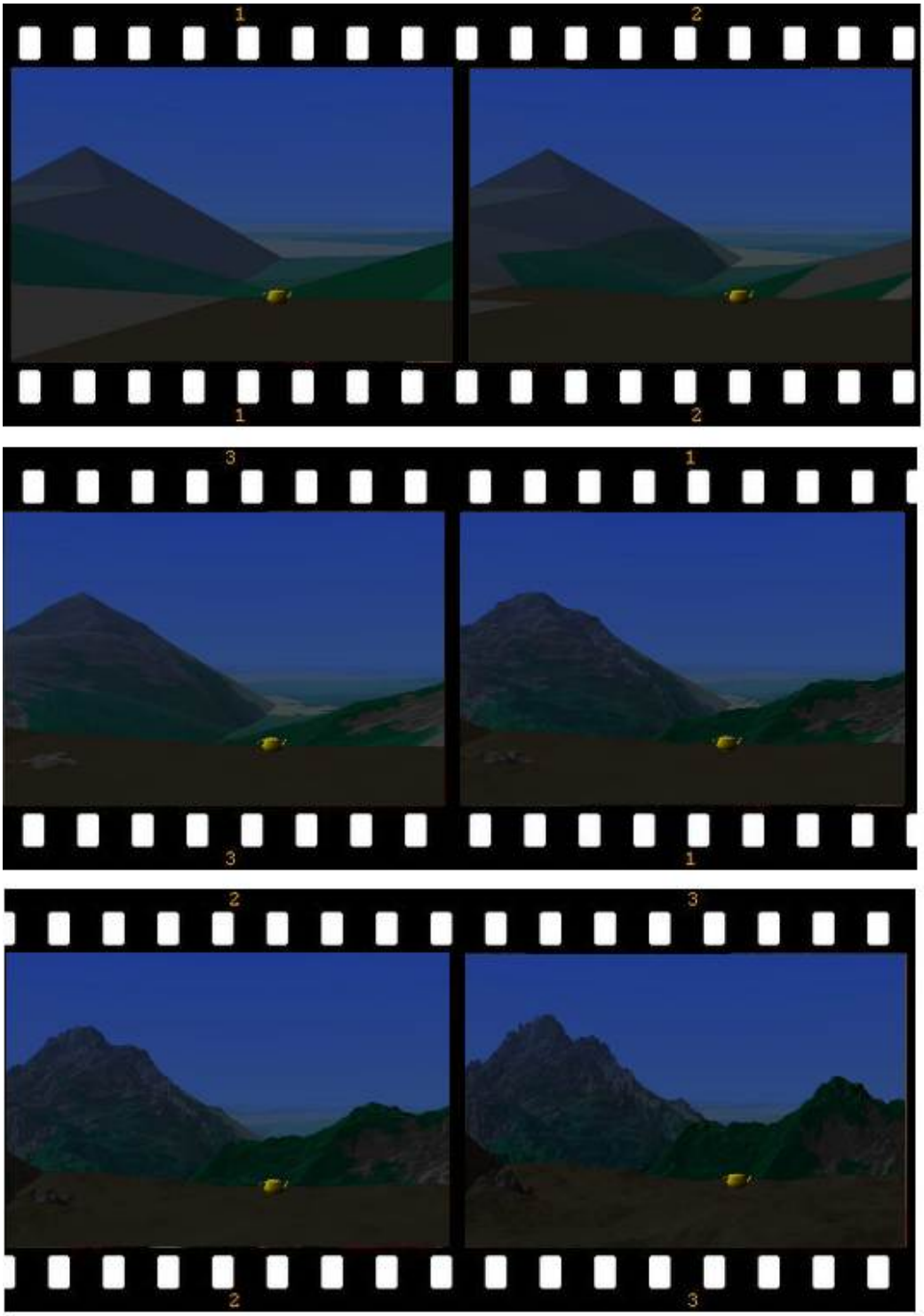

Abbildung 1.7: Ausschnitt aus Vol Libre

1.2 Frühere Arbeiten

Die erste Arbeit, die sich mit der Erzeugung von Landschaften mit Hilfe der fraktalen Mathematik beschäftigt hat, war das Essay *Les objets fractals, forme, hasard et dimension* [Man75] welches 1975 von Benoît Mandelbrot veröffentlicht und 1977 als *Fractals: Form, Chance and Dimension* [Man77] von ihm ins Englische übersetzt wurde.

Die Ergänzungen dieses Essays kamen 1982 als Buch heraus: *The Fractal Geometry of Nature* [Man82] von Benoît Mandelbrot. In diesem Buch beschäftigte sich Mandelbrot mit Fraktalen im Allgemeinen, aber vor allem wie sich die Natur aus Fraktalen zusammensetzt und wie das Wissen darüber genutzt werden kann, um eine künstliche, aber natürlich wirkende Umgebung im Computer zu generieren.

Die Ideen in diesem Buch wurden in den folgenden Jahren erweitert und verbessert. Ken Perlin veröffentlichte 1985 den Beitrag *An image synthesizer* [Per85], in dem er das sogenannte Perlin-Noise-Verfahren vorstellte, das dazu genutzt wird um synthetische Texturen zu erzeugen. Wie sich herausstellte, konnte dieses Verfahren dazu verwendet werden, um aus einer Textur eine fraktale Landschaft zu generieren.

Ebenfalls 1982 wurde der Artikel *Computer rendering of stochastic models* [FFC82] von Alain Fournier auf der SIGGRAPH vorgestellt. Darin wurde das Verfahren Midpoint-Displacement vorgestellt, welches ebenfalls wie Perlin-Noise eine Rauschtextur erzeugt. Beide Verfahren werden in Kapitel 2.2 ausführlich vorgestellt.

Diese Generatoren erzeugten natürlich aussehende Landschaften, aber andere Forscher wollten noch mehr Natürlichkeit, da es sich immer nur um eine Urlandschaft[5] handelt. Es flossen nun physikalische Prozesse in die Generierung der Landschaften mit ein, um den Landschaften ein noch natürlicheres Aussehen zu geben.

[5] fraktale Landschaft, ohne das Erosion auf sie eingewirkt hat

Im Jahr 1988 befasste sich eine Arbeit mit Erosions-Modellen. In *Terrain simulation using a model of stream erosion* [KMN88] von Alex D. Kelley et al.. Darin wurde ein Modell aufgezeigt, wie sich eine fraktale Landschaft durch einen Erosions-Algorithmus verändern lässt.

1989 wurde der Artikel *The synthesis and rendering of eroded fractal terrains* [MKM89] von F. Kenton Musgrave et al. auf der SIGGRAPH'89 vorgetragen. Darin wurden physikalische Modelle vorgestellt, welche die Erosion durch Wasserläufe (hydraulic erosion) und das Abrutschen von aufgeweichten Sedimentschichten (thermal weathering) simulieren. Diese Modelle sind nicht physikalisch exakt, bieten aber eine für Computer optimierte Annäherung der echten physikalischen Prozesse. Deshalb ist dieser Artikel die Grundlage für viele weitere Paper und Arbeiten geworden.

Vier Jahre später schrieb F. Kenton Musgrave seine Dissertation *Methods for Realistic Landscape Imaging* [Mus93] an der Yale University, in der er die Verfahren aus seinem Paper weiter verfeinerte.

Ein ähnliches Erosionsmodell wurde 1990 von Norishige Chiba et al. in der Arbeit *Terrain Simulation Based on the Recursive Refinement of Ridgelines Considering Erosion Processes* [CMSM90] vorgestellt. Da diese Arbeit nur in japanischer Sprache veröffentlicht worden ist, wurden die Ergebnisse in vielen weiteren Arbeiten kaum beachtet. Dennoch soll sie der Vollständigkeit halber erwähnt werden.

Da bei der Erosion auf fraktalen Landschaften immer nur eine einzige Gesteinsschicht beachtet wurde, simulierten P. Roudier et al. in der Publikation *Landscapes Synthesis Achieved through Erosion and Deposition Process Simulation* [RP93] weitere Gesteinsschichten. Jede dieser Schichten besitzt nun unterschiedliche Eigenschaften und reagiert unterschiedlich auf die Erosionsprozesse.

Die physikalischen Modelle, welche von Kelley et al., Musgrave et al. und Roudier et al. entwickelt wurden, sind Hauptbestandteil des Artikels *Computer generation of eroded valley and mountain terrains* [Nag97] von

Kenji Nagashima. In diesem 1997 erschienenen Artikel wurden diese Modelle so verfeinert, dass auch Flussläufe simuliert werden können.

Norishige Chiba et al. veröffentlichen 1998 ein neues Paper mit dem Titel *An erosion model based on velocity fields for the visual simulation of mountain scenery* [CMF98]. Darin beschrieben sie eine neue Datenstuktur, genannt Velocity Fields. Dies ist ein anderes Modell der Simulation von fließendem Wasser und wird im Kapitel 2.3.1 genauer behandelt.

Diese Arbeit wiederum diente 2001 dem Paper *Layered Data Representation for Visual Simulation of Terrain Erosion* [BF01] von Bedrichals Benes et al. als Grundlage. Es wurde darin eine neue Repräsentation der Landschaft entwickelt. Sie ermöglicht es, ähnlich wie die Arbeit von P. Roudier et al., unterschiedliche Gesteinsschichten zu simulieren, dies aber mit Hilfe der Velocity Fields.

Ein weiteres Jahr später wurde das Paper *Visual Simulation of Hydraulic Erosion* [FB02] von Rafael Forsbach et al. veröffentlicht, welches einen schnelleren und einfacheren Algorithmus basierend auf den Arbeiten von Musgrave et al., Chiba et al. und Nagashima vorstellte.

In den folgenden Jahren nahm das Forschungsinteresse im Bezug auf die Erzeugung fraktaler Landschaften ab. Die meisten der folgenden Arbeiten beschränkten sich darauf die bekannten physikalischen Modelle auf moderne Computersysteme effizient umzusetzen.

Die Arbeit *Fast Hydraulic Erosion Simulation and Visualization on GPU* von Xing Mei, et al. beschäftigte sich damit, die Berechnung der physikalischen Modelle von Musgrave et al., Chiba et al., Kelley et al. und Forsbach et al. auf die schnelleren Grafikkarten auszulagern.

Als derzeit letzte große Arbeit zum Thema Erosion ist das 2008 veröffentlichte Paper *Interactive terrain modeling using hydraulic erosion* [BvBK08] von Ondřej St'ava et al. zu nennen. Auch hier ging es darum, Teile der physikalischen Berechnung auf die Grafikkarte auszulagern.

1.3 Ziel dieser Arbeit

Ziel dieser Arbeit ist es, einen Überblick über die vorhandenen Verfahren der Erzeugung von fraktalen Landschaften zu geben, um daraufhin ein eigenes Konzept auszuarbeiten. Dieses Konzept wird um eigene Ansätze erweitert, um es anschließend in das an der Hochschule Darmstadt entwickeltes Framework GLAB implementiert.

Im Implementierungsteil dieser Arbeit soll nun, neben den Klassen für die Vorlesung „Chaos und Fraktale“, eine Schnittstelle für die Generierung eigener fraktaler Landschaften geschaffen. Auf diese Landschaften sollen unterschiedliche Erosionsprozesse einwirken um dadurch den zeitlichen Verlauf der Veränderungen an der Landschaft simulieren zu können.

Die Anzeige soll nicht nur eine statische Landschaft erzeugen, sondern die Erosions- und Alterungsprozesse in einem Zeitraffer darstellen können. Über verschiedene Parameter kann Einfluss auf die Veränderung der Landschaft genommen werden.

Das Hauptaugenmerk dieser Arbeit liegt auf den zu Grunde liegenden Algorithmen der Generierung und der physikalischen Veränderung und nicht auf der grafisch „hochwertigen“ Darstellung dieser Landschaften.

Order doesn't come by itself.

Benoît Mandelbrot

Kapitel 2

Grundlagen

Dieses Kapitel soll die Grundlagen für diese Arbeit liefern. Es wird zuerst auf die grundlegenden Datenstrukturen eingegangen und eine visuelle Darstellungsmethode für fraktale Landschaften vorgestellt, die in dieser Arbeit verwendet wird. Anschließend werden Generatoren und die Erosionsprozesse vorgestellt. Wie diese beiden Elemente zusammenhängen, wird in der Abbildung 2.1 dargestellt.

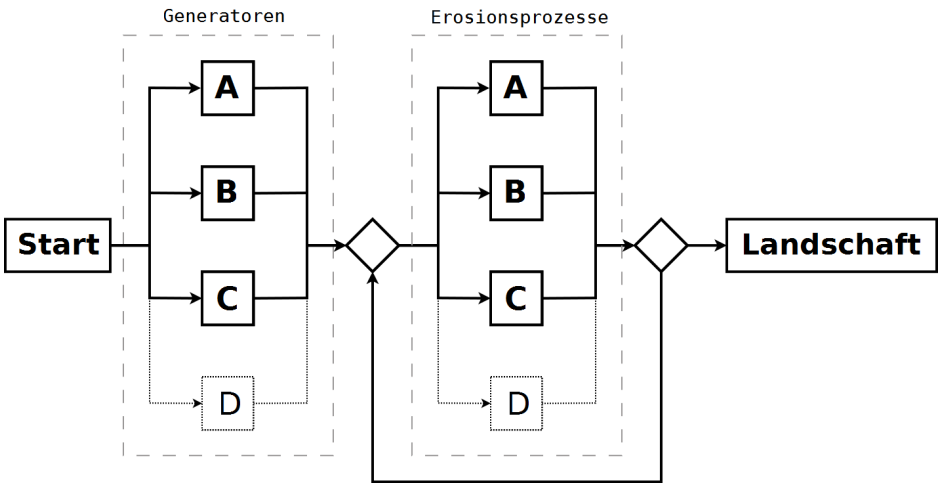

Abbildung 2.1: Ablaufplan zur Erzeugung fraktaler Landschaften

Es gibt viele unterschiedliche Generatoren, einer davon erzeugt eine Ur-

landschaft. Zwei der bekanntesten Generatoren werden in Kapitel 2.2.1 und 2.2.2 ausführlich vorgestellt. Die Urlandschaft, welche von keinen natürlichen Einflüssen berührt wurde, wird im zweiten Schritt durch einen iterativen Prozess verändert. Diese Veränderungen werden durch die verschiedenen Erosionsprozesse hervorgerufen. Die unterschiedlichen physikalischen Verlauf notwendigen Formeln Die verschiedenen Formeln, die für den physikalischen Verlauf notwendig sind, werden hierbei unterschiedlich häufig angewandt. So muss die Erosion durch Wasser (Kapitel 2.3.1) häufiger angewendet werden, als die der Gesteinveränderung (Kapitel 2.3.4).

Das abschließende Kapitel 2.4 zeigt den aktuellen Stand dieses Forschungsgebietes.

2.1 Datenstrukturen

Damit die einzelnen Generatoren und Erosionsprozesse erklärt werden können, muss zuerst eine Datenstruktur vorhanden sein, die im folgenden Kapitel erklärt wird.

2.1.1 Heightmap

Eine Sonderstellung unter den Datenstrukturen nimmt die Heigthmap ein. Sie dient zum einen der Speicherung einer fraktalen Landschaft, zum anderen wird sie auch als Darstellungsform benutzt. Die in Abbildung 2.2 dargestellte Landschaft soll möglichst effizient gespeichert werden.

Um dies zu ermöglichen, werden in die Zellen einer 2D-Matrix (Abbildung 2.3) die Höhen von den entsprechenden Positionen auf der Landschaft eingetragen.

Um die Höhenwerte von der Landschaft zu erhalten wird die 2D-Matrix

Abbildung 2.2: Fraktale Landschaft

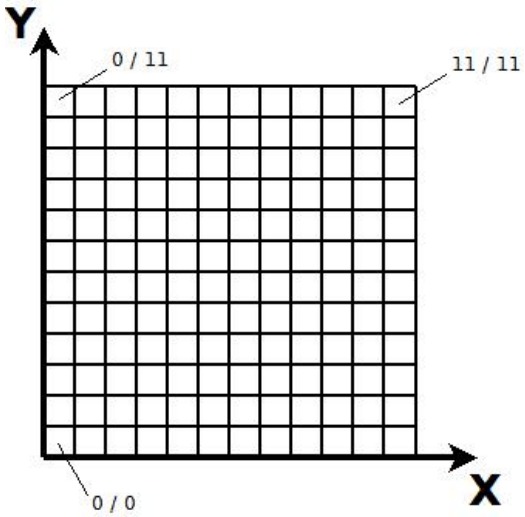

Abbildung 2.3: 2D-Matrix

„virtuell“ auf die Landschaft gelegt und anschließend die entsprechenden Höhenwerte eingetragen. Dies wurde im Bild 2.4 verdeutlicht.

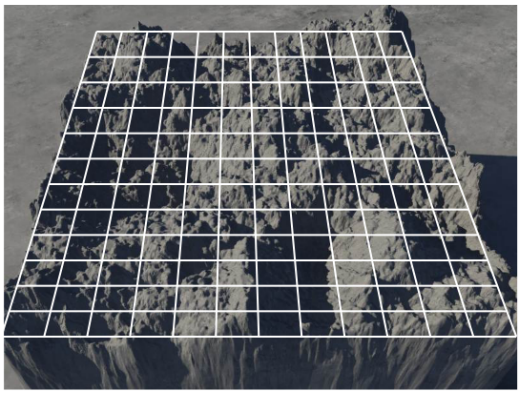

Abbildung 2.4: 2D-Matrix auf die Landschaft

Wenn die Spalten- und Zeilenanzahl sehr groß gewählt wird können feinere Höhenunterschiede gespeichert werden. Diese können als 8 Bit Grauwerttextur angezeigt werden. Dies wird umgangssprachlich Heigthmap genannt und als Raster Bild 2.5 dargestellt.

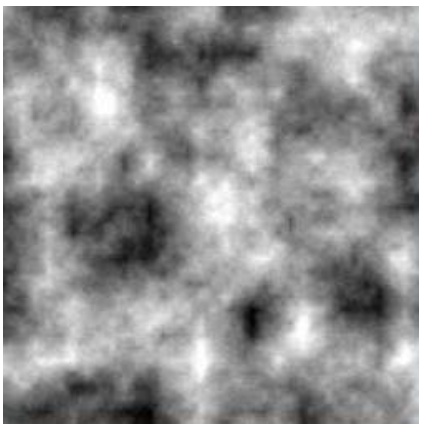

Abbildung 2.5: Heightmap

Die höchsten Höhen werden als helle Pixel dargestellt und äquivalent dazu die niedrigsten Punkte dunkel. Historisch bedingt wurde die Heightmap als 8bit (= 256 Farbwerte) Graustufenbild dargestellt. Die Umwandlung erfolgte nach folgender Formel:

$$Value_{x,y} = INT(\frac{h_{x,y}}{h_{max}} * 255) \tag{2.1}$$

$h_{x,y}$ ist hierbei die Höhe an der Stelle x/y, h_{max} die maximale Höhe, die eine solche Landschaft erreichen kann und $Value_{x,y}$ ist der Höhenwert an der Stelle x/y. Dieser Wert wird jedem RGB-Kanal eines Bildes zugewiesen. Das Ergebnis liegt nun zwischen 0 (=schwarz) und 255 (=weiß). Den berechneten Werten können auch andere Farben zugewiesen werden, sodass ein Falschfarbenbild (indiziertes Bild) entsteht. Damit ist es für den Menschen teilweise einfacher, die Höhenunterschiede festzustellen. Das Bild 2.5 mit einer topologischen Falschfarbenpalette ist repräsentativ zur Abbildung 2.6.

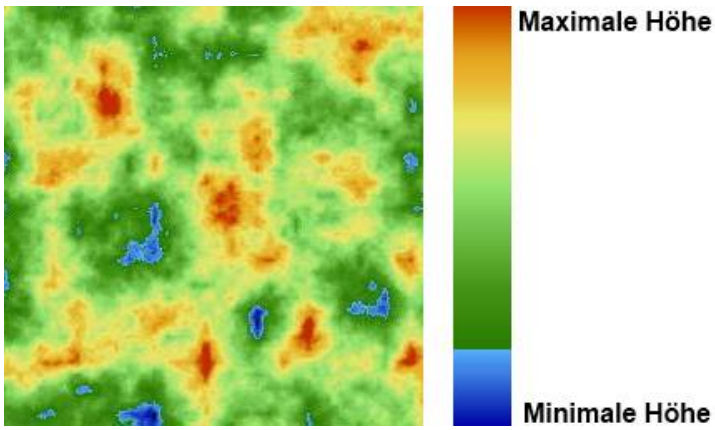

Abbildung 2.6: Heightmap mit indizierten Farbwerten

In dieser Arbeit werden zur Darstellung sowohl die Heightmap als auch gelegentlich, zur besseren Übersicht, das Bild mit indizierten Farbwerten verwendet.

Die Seitenlänge N einer Heightmap sollte dabei immer eine Potenz von Zwei sein. Das macht bei einer Seitenlänge von N = 2^{10} = 1024 insgesamt N^2 = 2^{10^2} = 2^{20} = 1.048.576 Bildpunkte. Diese Zweierpotenz und damit auch die quadratische Form der Heightmap ist durch die einzelnen Landschaftgeneratoren bestimmt. Bei allen wichtigen Generatoren werden immer wieder

Strecken durch Zwei geteilt, wodurch die daraus generierten Landschaften quadratisch sind. Für manche Generatoren, wie zum Beispiel den in Kapitel 2.2.1 vorgestellten Midpoint-Displacement-Generator ist eine Heightmap von 2^n+1 erforderlich, da hier zwingend ein eindeutiger Mittelpunkt benötigt wird.

Die Grauwerte bestehen aus 256 unterschiedlichen Abstufungen, die sich somit in 8 Bit je Zelle abspeichern lassen können. Das sind bei einer Heightmap mit N = 256 nur 524.288 Bit (64 Kilobyte) Speicherplatz. In den 1970er und 1980er Jahren, als die Entwicklung der fraktalen Landschaften begann, war Speicherplatz in den Rechnern sehr rar und damit kostbar. Daher wurde das Ausgabeformat gleichzeitig als Speicherformat verwendet.

Als nun Erosionsprozesse auf fraktale Landschaften angewendet wurden, ist man zu den Schluss gekommen, das sich die Heightmap dafür nur mäßig eignet. Gründe dafür sind vor allem die mangelnde Genauigkeit der ganzzahligen Grauwerte bei der Berechnung der Erosionsprozesse. Wie schon in dem vorangegangen Kapitel 2.3 zu sehen ist, werden sehr viele Gleitkommaberechnungen durchgeführt, deren Ergebnisse sich nicht auf die 256 Ganzzahlen abbilden lassen. Das Hauptproblem war es die zusätzlichen Schichten, wie Regolith oder Wasser, zu verwalten.

Daher wird die 8 Bit Heightmap nur noch für die Darstellung einer computergenerierten Landschaft verwendet.

Heutzutage kann auch eine größere Farbpalette verwendet werden, sodass der Zahlenbereich der Höhen größer wird.

2.1.2 Voxel

Ein Voxel ist ein Element, das sich aus den englischen Begriffen *Volumetric* und *Pixel* zusammensetzt. Es ist also genauer gesagt ein räumliches Pixel, welches im Gegensatz zu den x- und y-Positionen eines normalen Bildpunktes

über eine zusätzliche z-Position verfügt. Die schematische Darstellung eines Voxels innerhalb eine Volumens wird in Abbildung 2.7 gezeigt.

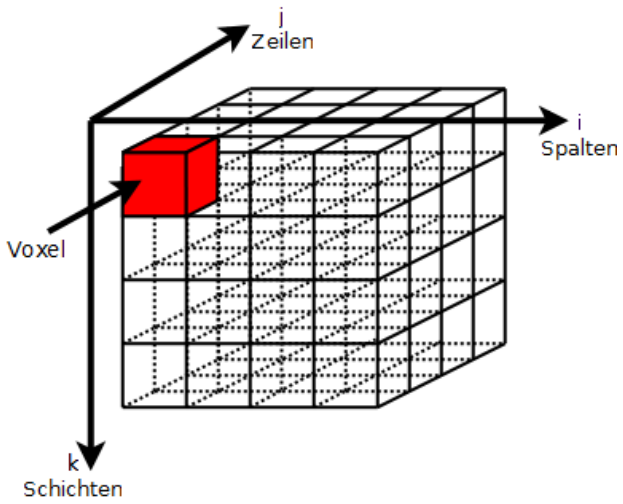

Abbildung 2.7: Voxel in einem Volumenbild

Viele Voxel ergeben in einem dreidimensionalen Raum eine Datenstruktur. Ein Voxel enthält jeweils die Eigenschaften, die an dieser Position stehen sollen. Bei der Anwendung in der Computergrafik enthält der Voxel normalerweise die Farben, in unserem Fall vor allem physikalische Eigenschaften.

Phyxel

Weil der Voxel für die folgenden Modelle hauptsächlich physikalische Eigenschaften besitzen soll wird er gelegentlich auch Phyxel, also physikalisches Pixel, genannt [Ste08] [JE06a] [MKN+04].

Für fraktale Landschaften müssen die Phyxel Daten speichern, wie zum Beispiel, welche Gesteinsart sich an der Position x/y/z befindet. Damit können verschiedene Schichten simuliert werden, ohne das auf zusätzliche Datenstrukturen zugegriffen werden muss. Aber leider steigt dadurch der benötigte Speicherplatz. Wenn nun angenommen wird, dass genau der gleiche Inhalt wie eine Heightmap mit N = 256 zu speichern ist, wird genau ein Bit (Gestein = 0, Luft = 1) für ein Phyxel benötigt. Es wird angenommen, dass ein

Phyxel genau einer Einheit der Seitenlänge N entspricht. Somit ergibt 256 x 256 x 256 x 1 Bit = 16.384 Bit (2.048 KB) und damit knapp 30 Mal soviel Speicherplatz . Damit die Landschaft mit den Erosionsformeln umgehen kann, würde erstens der Speicherplatz pro Phyxel steigen (mehrere Schichten, velocity vector) und, damit eine feinere Landschaft erzeugt werden kann, muss zweitens auch noch die Anzahl der Phxyel pro Einheit der Seitenlänge N erhöht werden. Beides führt dazu, dass der Speicherbedarf exponentiell steigt.

Dieser Nachteil wurde schon frühzeitig erkannt, sodass Phyxel eher für kleine Objekte, wie Steine oder Statuen [DEJ+99] [IN12] verwendet werden.

2.1.3 Layered Data Representation

Um den Speicheraufwand von Phyxeln deutlich zu verringern, wurde eine neue Datenstruktur, genannt Layered Data Representation, entwickelt [BF01]. Grundlage dafür ist das Schichtenmodel, welches in Kapitel 2.3.4 vorgestellt wird.

Wie bei einer Heightmap wird davon ausgegangen, dass die Landschaften aus Säulen besteht (Siehe 2.22). Eine Säule, deren Position durch eine X- und eine Y-Position definiert ist, wird in Schichten unterteilt. Jede Schicht hat die physikalischen Eigenschaften gespeichert und zusätzlich noch jeweilige Schichthöhe.

In der Gegenüberstellung der einzelnen Datenstrukturen, welche in Bild 2.8 zu sehen ist, sind die einzelnen Vorzüge zu sehen. Die Heightmap kann nur eine einzige Schicht verwalten, eine Wasserschicht (blau) muss extra verwaltet werden. Voxel können mit vielen Schichten umgehen, müssen aber redundante Eigenschaften verwalten, wodurch ein großer Verwaltungs- und Speicheraufwand betrieben werden muss. Bei der Layered Data Representation hingegen können mehrere Schichten simuliert werden. Der Verwaltungs- und Speicheraufwand wird, im direkten Vergleich zu Voxeln, deutlich mini-

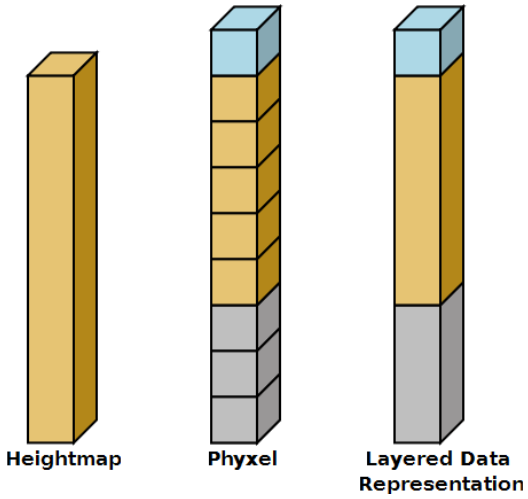

Abbildung 2.8: Datenstrukturen im Vergleich

miert.

2.2 Generatoren fraktaler Landschaften

Die Grundlagen zur Erzeugung fraktaler Landschaften gehen schon in die Anfänge des 19. Jahrhunderts zurück. Genauer bis in das Jahr 1827 als der schottische Botaniker Robert Brown die Bewegungsmuster von Pollen im ruhenden Wasser untersuchte.[1] Er stellte dabei fest, dass die kleinen Wasserteilchen die größeren Pollen unkontrollierbar durch die Flüssigkeit stoßen. Dadurch ergeben sich nicht vorhersagbare „Zuckungen“, die heute als Brown'sche (Molekular-)Bewegung bekannt sind. Wenn diese Bewegung auf ein zweidimensionales Diagramm abgebildet werden würde, könnte es wie in Abbildung 2.9 aussehen. [Mar07]

Wenn bei dieser zweidimensionalen Bewegung nur eine Dimension im Zeitverlauf betrachtet wird, entsteht ein Diagramm, welches aus zufälligen Schwankungen besteht. Dieses Diagramm sieht aus wie der Querschnitt einer

[1] Ein ähnliches Experiment (Holzkohlestaub in Alkohol) wurde schon 1785 von Jan Ingenhousz ausgeführt, welches zum gleichen Ergebnis führte.

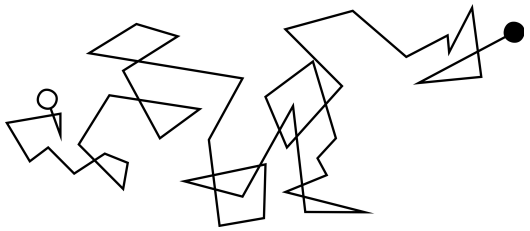

Abbildung 2.9: Brown'sche (Molekular-)Bewegung

Gebirgskette, wie in Bild 2.10 zu sehen.

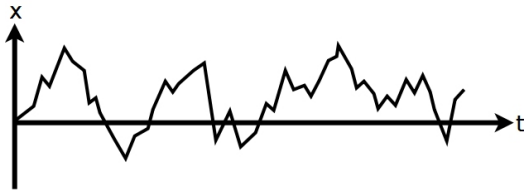

Abbildung 2.10: Brown'sche Bewegung im Zeitverlauf

Dieses Diagramm besitzt nun auch die Eigenschaften eines Signals. Daher werden im weiteren Verlauf die Bezeichnungen von Signalen verwendet. Diese können aus der Abbildung 2.11 entnommen werden.

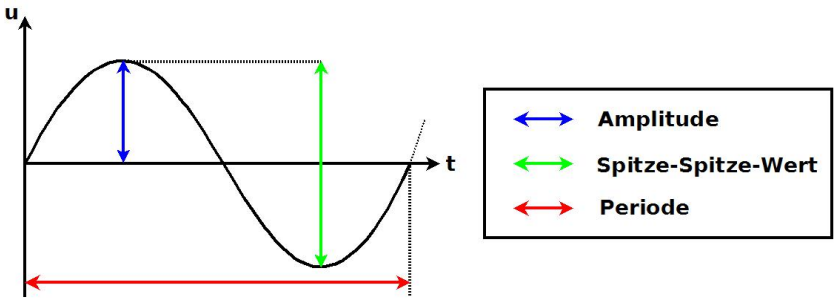

Abbildung 2.11: Darstellung eines Signals

Gelegentlich wird auch die Frequenz benötigt, die sich als Bild schlecht darstellen lässt. Nach DIN[2] ist die Frequenz definiert durch [LP06]:

[2] **D**eutsches **I**nstitut für *N*ormung

$$Frequenz = \frac{1}{T} \tag{2.2}$$

Sie ist gleichbedeutend mit der Anzahl der Perioden ΔP durch die Zeit Δt:

$$Frequenz = \frac{\Delta P}{\Delta t} \tag{2.3}$$

Die Brown'sche Bewegung des Partikels kann mit Hilfe einfachster Vorschriften simuliert werden, wie zusätzlich in Abbildung 2.12 zu sehen.

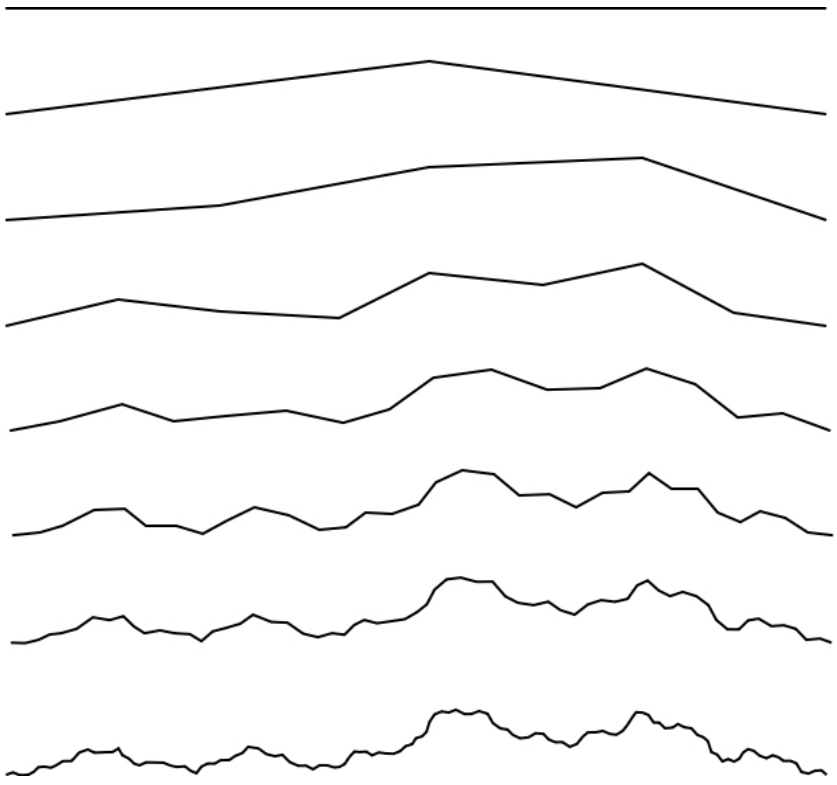

Abbildung 2.12: Simulation Brown'scher Bewegung

1. Am Anfang wird eine waagerechte Strecke erzeugt.
2. Der Mittelpunkt dieser Strecke wird um einen zufälligen Wert nach oben oder unten verschoben.
3. Von den zwei Teilstrecken wird jeweils der Mittelpunkt um einen zufälligen Höhenwert verschoben. Dieses Mal aber nur um maximal den halben Wert vom vorhergehenden Zyklus.
4. Punkt drei wird nun so oft wiederholt, bis die gewünschte Feinheit erreicht wurde.

Diese Mittelpunktverschiebung ist Grundlage vieler Generatoren für fraktale Landschaften. Ein Generator, der dieses Verfahren anwendet, ist der Midpoint-Displacement-Generator.

2.2.1 Midpoint-Displacement

Dieses Verfahren wurde 1982 von Alain Fournier et al. vorgestellt [FFC82] und von Gavin S. P. Miller [Mil86] 1986 weitervertieft. Hierbei wird eine Heightmap Textur generiert, die durch die in Abbildung 2.12 aufgezeigten Höhenänderungen entsteht. Zur Veranschaulichung wird dieses 2.13 dargestellt.

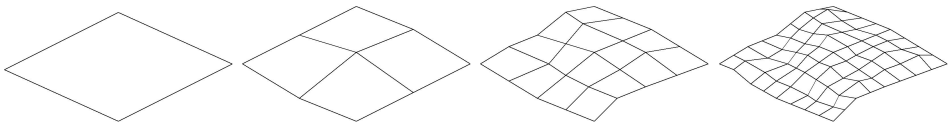

Abbildung 2.13: Midpoint-Displacement als Wireframe

Der von Fournier entwickelte Algorithmus funktioniert wie folgt:

Zuerst werden die Höhen an den Eckpunkten h_{E1} - h_{E4} einer quadratischen Grundfläche durch zufällige Höhenwerte initialisiert. Anschließend wird diese Grundfläche in vier weitere Quadrate eingeteilt. Die Höhenwerte für diese Quadrate werden anhand einer 3x3 Matrix beispielhaft erläutert.

$$\begin{matrix} h_{E1} & 0 & h_{E2} \\ 0 & (h_{E1} + h_{E2} + h_{E3} + h_{E4})/4 \pm \lambda & 0 \\ h_{E4} & 0 & h_{E3} \end{matrix}$$

Die Höhe h_{Mi} am Mittelpunkt wird aus den Höhen h_{E1} - h_{E4} der vier äußeren Ecken berechnet und um einen zufälligen Wert λ erhöht oder erniedrigt. Wichtig ist dabei, dass der Zufallswert λ in jedem Iterationsschritt geringer ausfällt. Im nächsten Schritt wird die Höhe des berechneten Mittelpunkts h_{Mi} auf die Höhen der Seiten des Quadrates verrechnet:

$$\begin{matrix} h_{E1} & (h_{E1} + h_{E2} + h_{Mi})/3 \pm \lambda & h_{E2} \\ (h_{E1} + h_{E4} + h_{Mi})/3 \pm \lambda & h_{Mi} & (h_{E2} + h_{E3} + h_{Mi})/3 \pm \lambda \\ h_{E4} & (h_{E3} + h_{E4} + h_{Mi})/3 \pm \lambda & h_{E3} \end{matrix}$$

Die nun erzeugte Matrix wird wieder in vier gleichgroße Quadrate aufgeteilt und nach der oberen beschriebenen Matrix berechnet. Damit ist es möglich, eine immer feiner werdende Landschaft zu generieren. Der erste und der zweite Iterationsschritt werden in Abbildung 2.14 verdeutlicht.

Dieser Algorithmus wird auch Diamond-Square-Algorithmus genannt, da zuerst die Mitte erzeugt wird (Diamond) und danach ein Quadrat (Square).

An diesem Verfahren wird deutlich, dass die Brown'sche Bewegung wichtig für diese Erzeugung ist. Es wird immer die Hälfte von einer Strecke genommen und um einen zufälligen Wert nach oben oder nach unten korrigiert. Dieses Mal aber im dreidimensionalen Raum.

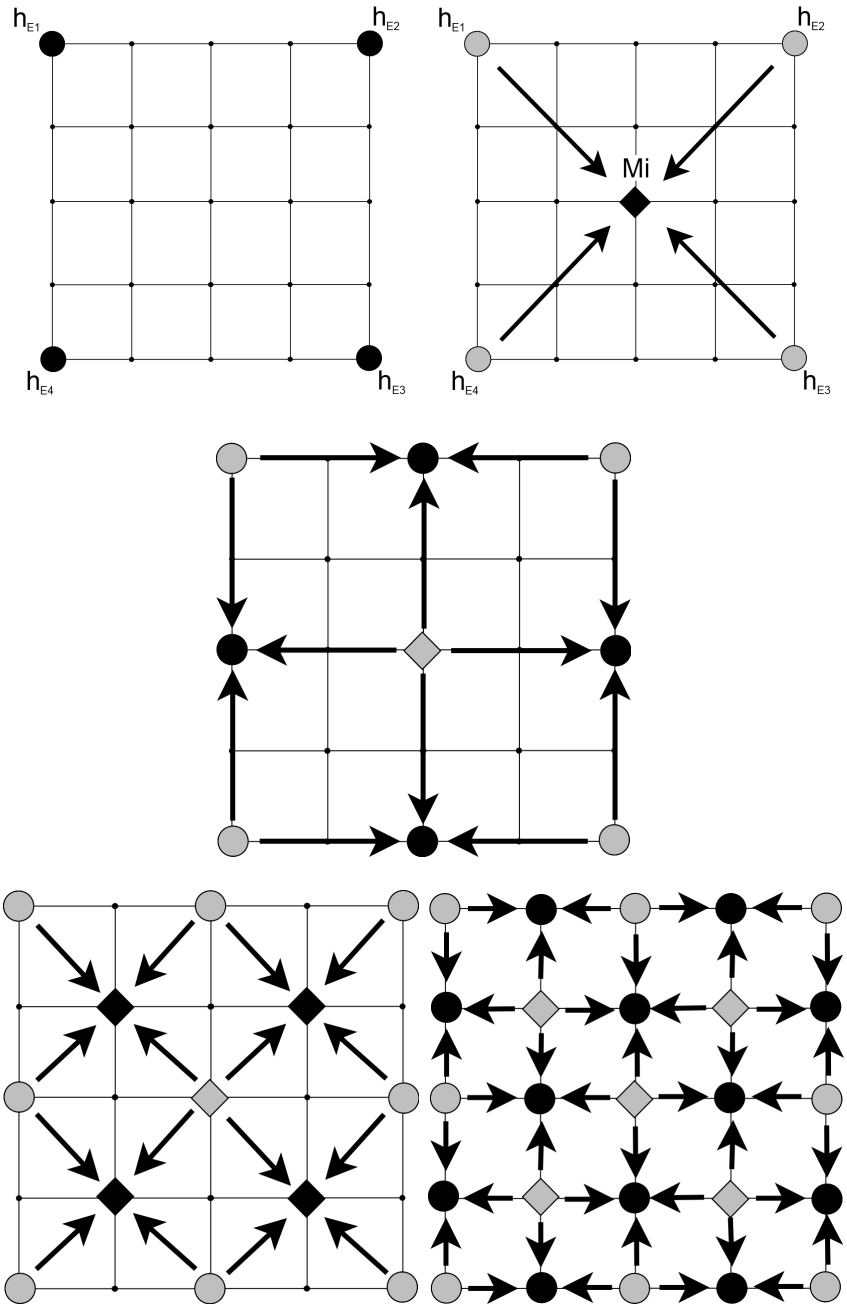

Abbildung 2.14: Diamond-Square-Algorithmus

2.2.2 Perlin-Noise

Das Verfahren Perlin-Noise ist nach seinem Erfinder Ken Perlin benannt. Dieses Verfahren ist eine synthetische Texturerzeugung, das heißt es können Texturen mit wenigen Parametern zufällig erzeugt werden. Die Grundlagen für dieses Verfahren wurden 1982 von ihm für den Film „TRON“ entwickelt und schließlich 1985 in seiner Arbeit *An image synthesizer* [Per85] der Öffentlichkeit vorgestellt. Perlin-Noise kann dazu benutzt werden um natürlich wirkende Texturen zu erzeugen, wie zum Beispiel für Wolken, Feuer, Holzmaserungen und eben zur Erzeugung fraktaler Landschaften.

Die Idee hinter Perlin-Noise ist das Aufaddieren verschiedener Rauschfunktionen. Wie in der Bilderreihe 2.15 zu sehen ist, werden die Funktionen über die Amplitude h und einer Frequenz v bestimmt. Die Frequenz v bestimmt die Anzahl der Schwingungsänderungen innerhalb einer bestimmten Zeit t, die Amplitude h beschreibt die Stärke der Schwingung. Die Amplitude h entspricht hierbei der Höhe der generierenden Landschaft. Beide Parameter sind Zweierpotenzen. Dies für den Algorithmus von Vorteil, weil die Frequenz v verdoppelt und die Amplitude h halbiert wird. Wenn alle Werte berechnet wurden, können die jeweiligen Höhen der einzelnen Funktionen zusammen addiert werden. Das Ergebnis dieser Addition wird in Bild 2.16 dargestellt. Es hat Ähnlichkeiten mit einem Gebirgszug.

Die Rauschfunktionen werden durch Generierung von Zufallszahlen erzeugt. In dem vorgestellten Beispiel wurden die Zufallszahlen jeweils im Bereich $2 * h$ erzeugt, damit sie komplett im positiven Bereich liegen. Es werden $v + 1$ Stützstellen ermittelt, an denen die Zufallszahlen erzeugt werden. Die Werte zwischen zwei Stützstellen wurden in dem Beispiel zwischen den Punkten h_{x_1,y_1} und h_{x_2,y_2} linear interpoliert. Die Höhe h von x an der Stelle i wird wie folgt berechnet:

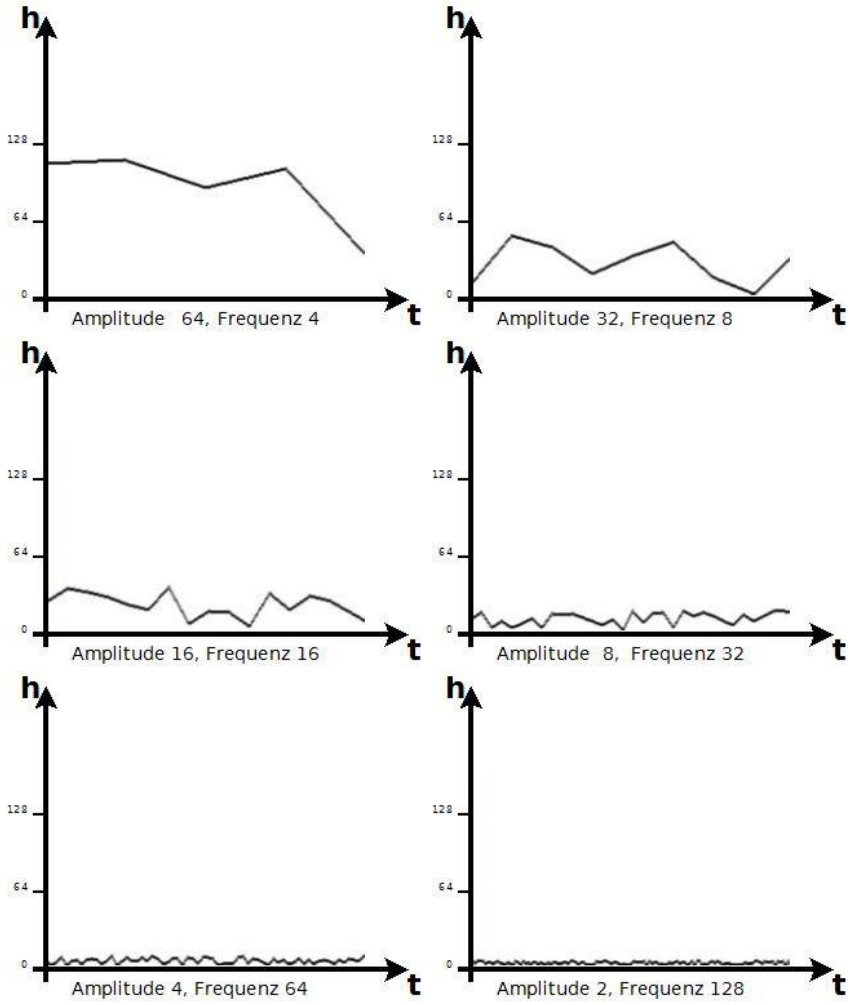

Abbildung 2.15: Rauschfunktionen unterschiedlicher Intensität

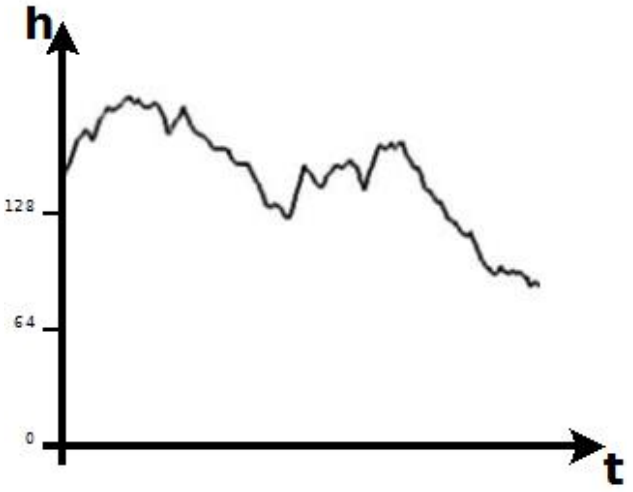

Abbildung 2.16: Summe der einzelnen Rauschfunktionen

$$n = x_2 - x_1, \ mit \ x_1 < x_2 \tag{2.4}$$

$$m = \frac{1}{n} \tag{2.5}$$

$$\Delta x = m * (x_i - x_1), \ mit \ x_1 < x_i < x_2 \tag{2.6}$$

$$h_{x_i} = h_{x_1} * (1 - \Delta x) + h_{x_2} * \Delta x \tag{2.7}$$

Die Lineare Interpolation ist zwar einfach zu berechnen, hat aber den Nachteil, dass sie extreme Richtungsänderungen aufweist. Da eine natürlich wirkende Landschaft aber nie über derartig extreme Richtungsänderung verfügt, kann die Formel (2.7) durch die Cosinus-Interpolation ersetzt werden.

$$h_{x_i} = h_{x_1} * (1 - \frac{1 - cos(\Delta x * \pi)}{2}) + h_{x_2} * \frac{1 - cos(\Delta x * \pi)}{2} \tag{2.8}$$

Dadurch wird ein Graph erzeugt, der über natürlich wirkende Übergänge verfügt, wie in Abbildung 2.17 zu sehen ist.

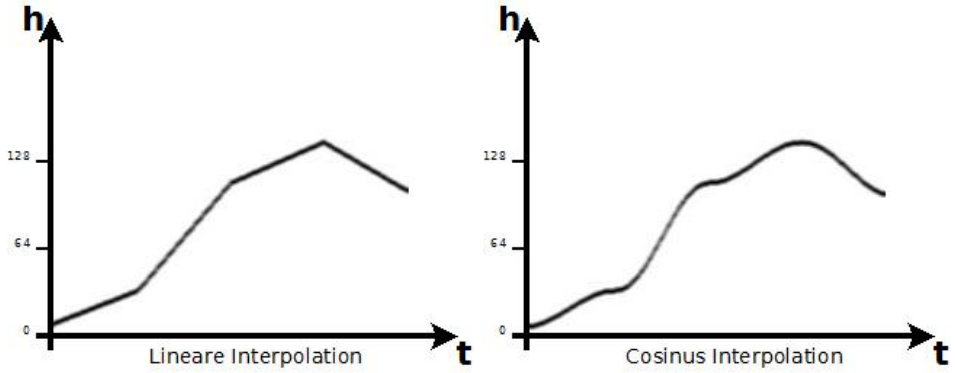

Abbildung 2.17: Lineare und Cosinus-Interpolation

Es gibt darüber hinaus noch weitere Interpolationsverfahren, wie z. B. das kubische Interpolationsverfahren. Diese sind deutlich aufwendiger zu berechnen ohne optisch bessere Resultate zu liefern.

Bis jetzt wurde die Idee von Perlin-Noise nur anhand einer zweidimensionalen Funktion erklärt. Bei der Verwendung von Perlin-Noise für fraktale Landschaften wird das Verfahren um eine weitere Dimension erweitert. Auch hier werden zufällige Höhenwerte erzeugt und in einem zweidimensionalen Raum an bestimmten Stützstellen verteilt. Die Abstände dieser Stützstellen orientieren sich wieder anhand der Frequenz; je niedriger die Frequenz, desto größer ist der Abstand. Die Höhenwerte zwischen diesen Stützstellen werden anhand einer Interpolationsberechnung, am besten der Cosinus-Interpolation, ermittelt. Die 2D-Matrix, die daraus entsteht, kann als Heightmap dargestellt werden. In Abbildung 2.18 sind fünf Texturen abgebildet, die sich jeweils in der Amplitude und der Frequenz unterscheiden.

Die unterschiedlichen Texturen werden, wie schon im anfänglichen Beispiel gezeigt, aufaddiert, sodass eine Heightmap entsteht, die für eine fraktale Landschaft verwendet werden kann. Die aus den fünf vorhergehenden Texturen erzeugte Heightmap wird in Abbildung 2.19 dargestellt.

Die Veränderung von Amplitude und Frequenz in jedem Schritt kann auch mit Hilfe eines anderen Parameters beeinflusst werden: der Persistenz.

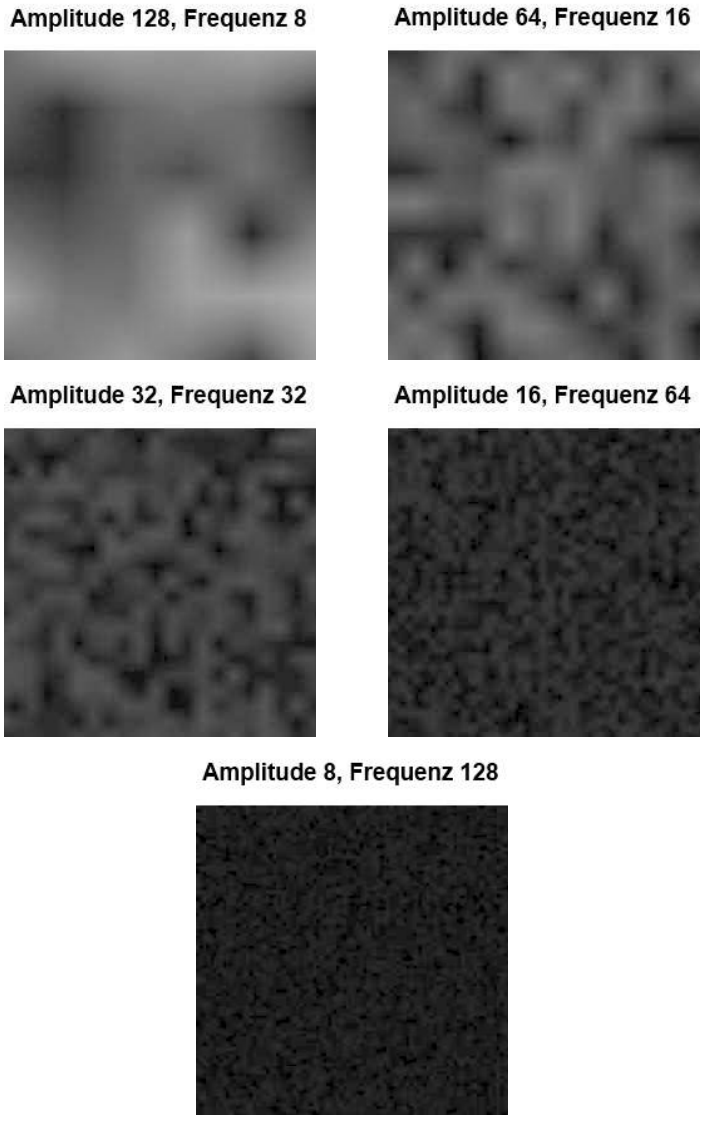

Abbildung 2.18: Perlin-Noise Texturen mit unterschiedlichen Parametern

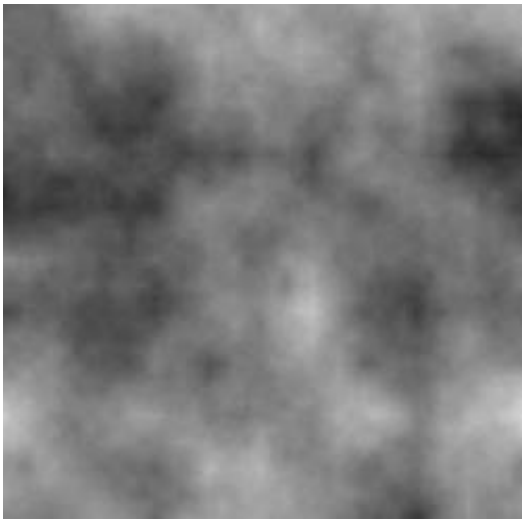

Abbildung 2.19: Summe der einzelnen Texturen

Sie ist wie folgt definiert [Hom06] :

$$Frequenz = 2^i \tag{2.9}$$

$$Amplitude = Persistenz^i \tag{2.10}$$

i steht dabei für den i-ten Durchlauf (eine Oktave) der Funktion. Bei einer Persistenz von 1, bleibt die Amplitude immer gleich, es ändert sich mit jeder Oktave nur die Frequenz. Wenn eine Persistenz größer als 1 verwendet wird, so wird die Amplitude mit jeder Oktave größer. Für die Generierung fraktaler Landschaften sollte eine Persistenz kleiner 1 gewählt werden, damit bei höherer Frequenz eine niedrigere Amplitude erreicht wird. Das wird so angewendet, weil dadurch die Ausschläge des Signal zahlreicher, aber auch weniger werden.

2.2.3 Weitere Generatoren

Midpoint-Displacement und Perlin-Noise sind zwei der bekanntesten Generatoren zur Erzeugung einer fraktalen Landschaft. Darüber hinaus gibt es noch zahlreiche weitere, ein paar davon werden diesem Kapitel kurz vorgestellt.

Einer der ersten Landschaftsgeneratoren wurde von Benoît Mandelbrot und Richard F. Voss entworfen und nannten es *poisson faulting*. In diesem Verfahren wird die Landschaft durch eine besondere Art der Brown'scher Bewegung erstellt. Dabei handelt es sich um das Signal von einem weißen Rauschen. Dieses Rauschen sieht als Signal ähnlich wie Abbildung 2.10, nur mit der besonderen Eigenschaft, dass die Höhen der Signalamplituden nach der Gauß'schen Glockenkurve verteilt sind. Die Höhenwerte innerhalb der 2D-Matrix werden nun nach diesem Rauschen erzeugt. Es hat Ähnlichkeit mit dem Perlin-Noise-Verfahren, nur werden hier keine unterschiedlich stark ausgeprägten Graphen aufaddiert. [Man82]

Ein weiteres Verfahren ist das *successive random additions* [Mus93]. Bei diesem Verfahren wird in einer begrenzten Fläche eine zufällige Position x/y ausgewählt und der Höhenwert an dieser Position um einen zufälligen Wert λ erhöht. Die Auswahl der zufälligen Position x/y wird ermittelt, indem die Strecke zwischen zwei zufälligen Positionen x_1/y_1 und x_2/y_2 berechnet wird. Auf dieser Strecke wird nun eine beliebige Position zufällig ausgewählt und der Höhenwert an dieser Stelle mit einen zufälligem Wert λ addiert. Dieses Vorgehen wird nun permanent wiederholt bis die gewünschte Feinheit erzeugt wurde. Dieses Verfahren ist besonders gut geeignet um den Detailreichtum einer Landschaft in einem bestimmten Bereich zu erhöhen. Anwendungsbeispiel hierfür ist eine Zoom-Funktion innerhalb dieser Landschaft. Dieser Algorithmus wird anhand der Bildreihe 2.20 verdeutlicht. Als Beispiel dient dafür eine dreieckige Grundfläche.

Der sogenannte Circle-Algorithmus geht einen etwas anderen Weg, als die oben genannten Verfahren. Hierbei werden Körper, in diesem Fall eine Art Halbkugel, von unten oder von oben in eine flache Ebene „gedrückt“. Zuerst wenige, aber im Durchmesser große Halbkugeln, dann mehrere aber kleinere Halbkugeln. Als Grundlage für die Höhe kann die Cosinus Schwingung zwischen $-\pi$ und $+\pi$ genommen werden, mit Null als Zentrum. Diese Art der Schwingung wird deshalb genommen, da sie in unteren Bereich langsamer abflacht. Da die Schwingung einen Spitze-Spitze-Wert (Siehe Abbildung 2.11) von -1 und 1 besitzt, muss sie um Eins erhöht werden, danach liegen sie im Positiven Bereich zwischen 0 und 2 (siehe Abbildung 2.21).

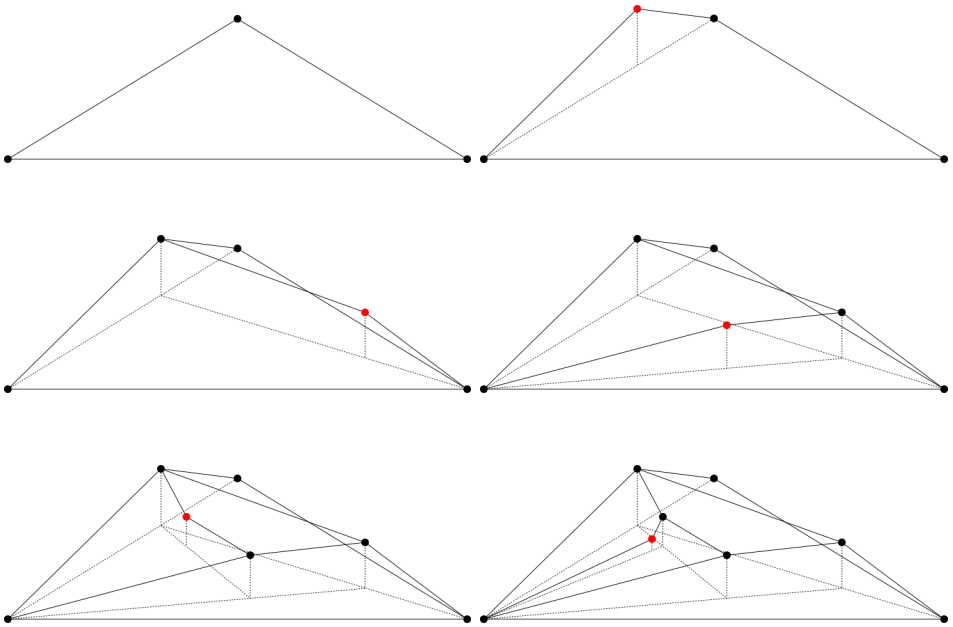

Abbildung 2.20: Successive Random Additions

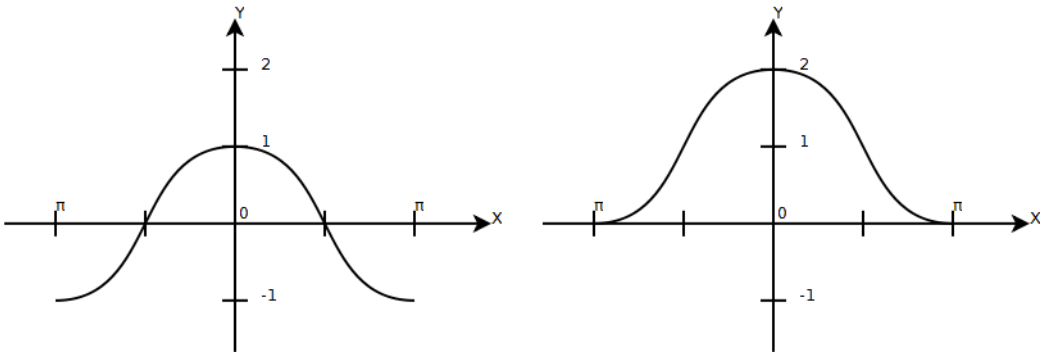

Abbildung 2.21: Circle-Algorithmus

Wenn sich nun diese Teilschwingung um das Zentrum (x = 0) dreht, wird eine Art Halbkugel erzeugt. Der Algorithmus funktioniert so, dass eine zufällige Position in einer 2D-Matrix ausgewählt wird. Diese dient als Zentrum der Cosinus Schwingung. Die Höhen werden nach der Formel

$$h_{x,y} = h_{x,y} + ((cos(Distanz/Radius * \pi) + 1) * Höhenfaktor) \qquad (2.11)$$

im Halbkreis um das Zentrum herum in alle berührenden Zellen eingetragen, damit ein kugelförmiges Objekt entsteht. Es werden dabei zuerst sowohl der Radius als auch der Höhenfaktor groß gewählt, um wenige große Halbkugeln zu erzeugen. Beide Parameter werden verringert um kleine Halbkugeln zu erzeugen, dafür aber eine größere Anzahl.

Bei allen aufgeführten Generatoren besteht das Problem, dass sie nur eine Urlandschaft erzeugen. Auf den ersten Blick sehen sie natürlich aus, auf den zweiten Blick zeichnen sich Unregelmäßigkeiten ab. Es kommt vor, dass Felsformationen unnatürliche Übergänge aufweisen oder dass diese Landschaften keinerlei Anzeichen von Wasserläufen besitzen.
Damit die fraktalen Landschaften natürlicher wirken, werden Erosions- und Alterungsprozesse eingesetzt, die diese Landschaften dahingehend verändern.

2.3 Erosion- und Alterungsprozesse

Erosionen auf fraktalen Landschaften werden seit Mitte der 1980er Jahre in der Forschung behandelt. Ziel dieser Arbeiten war es die Landschaften realistischer aussehen zu lassen. Der Hauptfokus der meisten Arbeiten lag auf der Erosion durch Wasser. Das ist dadurch zu erklären, dass diese Art der Erosionen am deutlichsten und vor allem am schnellsten in einer Landschaft zu sehen ist. So kann sich innerhalb weniger Jahre ein kleines Flussbett in einen felsigen Boden graben, vorausgesetzt es gibt einen stetigen Wasserfluss. An-

dere Faktoren, wie beispielsweise Wind, benötigen dagegen einige (virtuelle) hundert oder tausend Jahre bis eine sichtbare Veränderung auftritt.

Für diese Arbeit sollen die Erosions- und Alterungsprozesse über simulierte zehntausend Jahre realisiert werden. In den weiteren Unterkapiteln werden die einzelnen Erosions- und Alterungsprozesse im Detail besprochen.

Für die folgenden Erosionsprozesse werden diese Höhen an Position x/y als Säulen (eng. column) angesehen, wie in Bild 2.22 dargestellt.

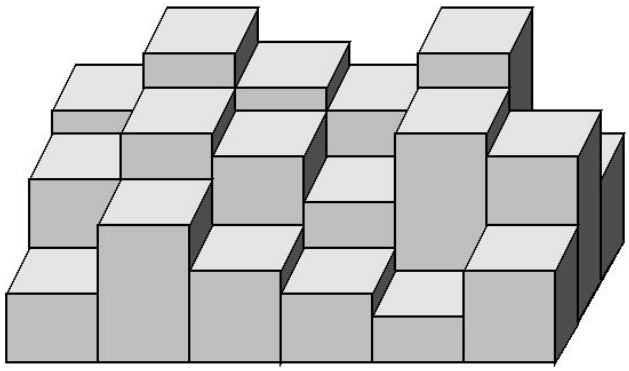

Abbildung 2.22: Landschaft als Säulengrafik

Bei fast allen Erosionsprozessen werden die Höhen der benachbarter Positionen miteinander verglichen. Es existieren dabei zwei mögliche Nachbarschaftsverhältnisse. Bei der sogenannten Von-Neumann-Nachbarschaft, auch N4-Nachbarschaft genannt, werden nur die vier Nachbarn ober- und unterhalb, links und rechts berücksichtigt (Siehe Abbildung 2.23). Bei der Moore-Nachbarschaft (N8-Nachbarschaft) werden zusätzlich noch alle vier diagonalen Nachbarn näher betrachtet, wie in Abbildung 2.24 zu sehen.

In vielen Fällen wird die N8-Nachbarschaft verwendet, da diese für die Erosionsprozesse genauer ist. Dabei müssen allerdings doppelt so viele Vergleiche durchgeführt werden, wie bei der N4-Nachbarschaft. Für die folgenden Modelle wird, wenn nicht anders angegeben, die N8-Nachbarschaftsbeziehung verwendet.

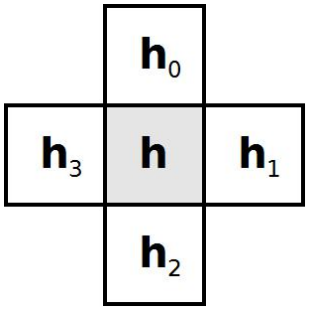

Abbildung 2.23: Von Neumann Nachbarschaft (N4)

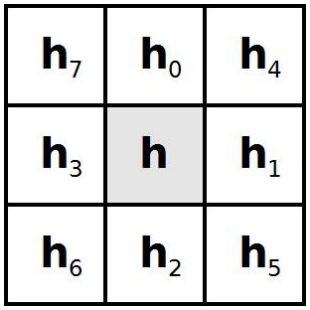

Abbildung 2.24: Moore Nachbarschaft (N8)

2.3.1 Erosion durch fließende Gewässer

Die Erosion durch Wasser ist in der Computergraphik, im Hinblick auf fraktale Landschaften, das am meisten erforschte Thema. [KMN88], [MKM89], [Mus93], [CMSM90], [Nag97], [BTHB06], [BA05], [Ols04], [FB02], [BvBK08]

Die erste Arbeit auf diesem Gebiet wurde in den 1990er Jahren von Forest Kenton Musgrave et al. veröffentlicht. Das Modell, welches dabei entwickelt wurde, dient als Grundlage für einige folgende Projekte verwendet. Weitere wichtige Arbeiten aus den 2000er Jahren stammen von Bedřich Beneš et al..

Eines der ersten und gleichzeitig auch wichtigsten Modelle ist das sogenannte Hydraulic Erosion Modell. Hierin wird fließendes Gewässer simuliert. Dies kann ein Flusslauf sein, der Gestein aufweicht und mittransportiert, aber auch Regen, der im Grunde nichts anderes ist, wie ein sehr kleiner, lokaler Flusslauf. Das aufgeweichte Gestein heißt Regolith[3].

Wie schon in den vorherigen Kapiteln erklärt, besteht die computergenerierte Landschaft aus einer 2D-Matrix mit den jeweiligen Höhenwerten. Die Höhen der Landschaft setzen sich aus den Schichten $s_0 - s_n$ an der Position (x/y) zusammen. Die Landschaft, die aus den Generatoren in Kapitel 2.2

[3]lockeres Gestein bestehend aus u. a. Sand, Staub, Kiesel

erzeugt wurde, besitzt nur eine einzige Schicht s_0. Auf dieser Schicht s wird nun durch eine Quelle das Wasser w an die Stelle (x/y) aufaddiert. Daraus ergibt sich die Höhe h. Falls es, wie schon in Kapitel 2.1.3 angedeutet, mehrere Schichten gibt, so wird die Höhe der einzelnen Schichten aufsummiert.

$$h_{x,y} = \sum_{x=0}^{c_{x,y}} s_{x,y} + w_{x,y} \tag{2.12}$$

Für dieses Modell wird öfters der Höhenunterschied eines Nachbarn benötigt. Für diese Differenz $d_{x,y}$ wird die Höhe $h_{x,y}$ vom Nachbarn h_{x_n,y_n} abgezogen.

$$d_{x,y} = h_{x,y} - h_{x_n,y_n} \tag{2.13}$$

In speziellen Fällen wird die höchste Höhendifferenz an der Position x/y benötigt. Dann werden alle Differenzen um diese Position berechnet, entweder als N4- oder N8-Nachbarschaft. Für dieses Erosions-Modell wird die N4-Nachbarschaft verwendet, da sie nach [MDH07] ähnlich gute Ergebnisse liefert als die N8-Nachbarschaft. Dieser Vergleich wird in dem folgenden Bild 2.25 verdeutlicht.

Nachdem diese Formeln bekannt sind, kann das Hydraulic Erosions Modell erklärt werden. Dieses Modell kann in fünf Teilschritte gegliedert werden. Diese welche immer wieder wiederholt werden:

1. Wasser erscheint als Regen oder als Flussquelle
2. Das Wasser fließt zum lokalen Minimum
3. Umwandlung der Landschaftsoberfläche in Regolith

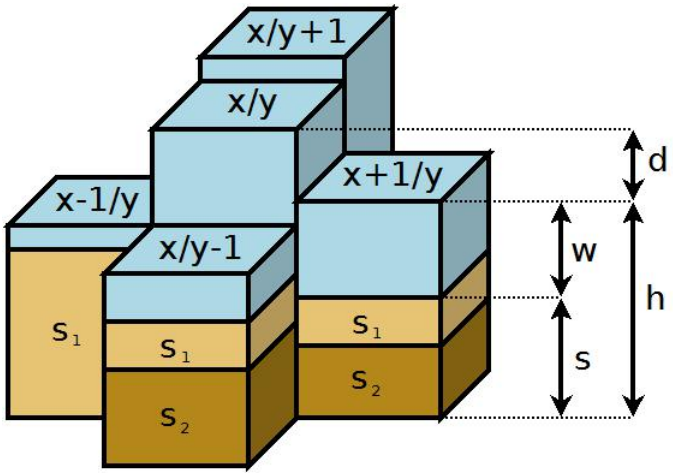

Abbildung 2.25: Parameter der Hydraulic Erosion

4. Aufgeweichte Erde wird mit dem Wasser transportiert
5. Wasser verdunsten und/oder versickert in die Erde

Wie das Modell im einzelnen berechnet wird, folgt auf den nächsten Seiten.

1. Wassererzeugung

Im ersten Schritt wird Wasser auf der Landschaft erzeugt. Dies kann entweder in Form einer Wasserquelle oder als Regen sein. Eine Wasserquelle ist regional auf die Position x/y begrenzt. Es wird also die Wasserschicht $w_{x,y}$ um die Menge an Wasser w_q erhöht. Die zugeführte Wassermenge ist zusätzlich abhängig von der Zeit Δt:

$$w^{x,y}_{\Delta t+1} = w^{x,y}_{\Delta t} + \Delta t * w^{q_{x,y}}_{\Delta t} \tag{2.14}$$

Regen wird dadurch simuliert, dass an einer zufälligen Position x/y eine

kleine Wassermenge $w_{r_{x,y}}$ zu einer eventuell vorhandenen Wassermenge hinzu addiert wird. Diesen Vorgang wird je nach Regenstärke mehrfach durchgeführt, damit über die komplette Landschaft viele lokale Wasserquellen entstehen.

$$w^{x,y}_{\Delta t+1} = w^{x,y}_{\Delta t} + \Delta t * w^{r_{x,y}}_{\Delta t} \tag{2.15}$$

Mit jeder dieser beiden Varianten befindet sich eine gewisse Menge Wasser auf der Landschaft.

2. Simulation von fließendem Wasser

Der nächste Schritt besteht aus der Simulation des Wasserflusses. Für diese Simulation wird das sogenannte Pipe-Modell [OH95] verwendet. Bei diesem Modell wird der Wasserverlauf durch virtuelle Rohre (eng. Pipe) zwischen den Nachbarn realisiert. Eine schematische Darstellung wird in Bild 2.26 gezeigt.

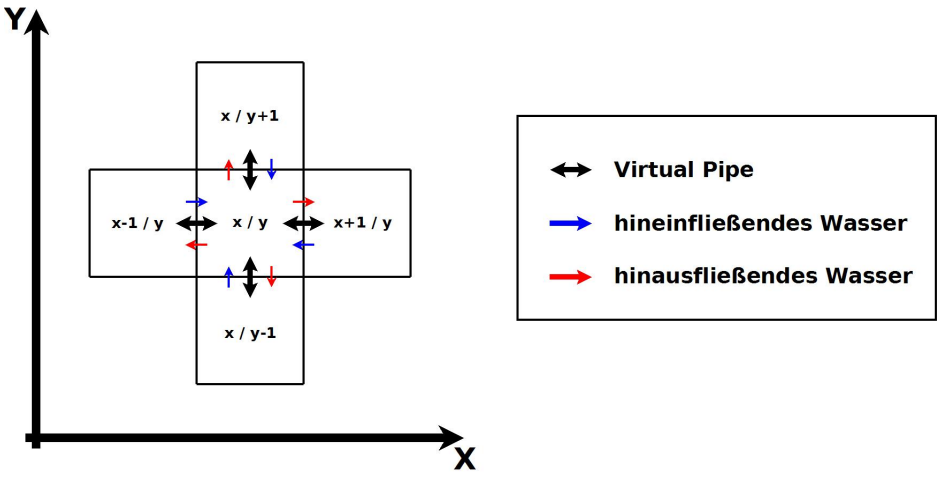

Abbildung 2.26: Pipe-Modell

Die Menge, die durch eine virtuellen Röhren fließen kann, ist durch den Durchfluss f (eng. flux) begrenzt. Die Begrenzung dieser Röhre kommt durch die physikalischen Eigenschaften von Wasser, insbesondere der Trägheit zustande. Würde eine andere Art Flüssigkeit simuliert werden, zum Beispiel eine zähflüssigere Substanz als Wasser, so wäre der maximale Durchfluss kleiner bei Wasser.

In jedem Durchgang wird der Durchfluss f zu der Position x/y berechnet. Dazu muss der Durchfluss in allen vier angrenzenden Zellen betrachtet werden. In der nachfolgenden Formel werden die Durchfluss-Werte mit $f^L = f_{x-1,y}$, $f^R = f_{x+1,y}$, $f^B = f_{x,y-1}$ und $f^T = f_{x,y+1}$ angegeben.

Zunächst wird die Menge des ausfließenden Wassers der Position x/y bestimmt. Dazu wird für alle vier virtuellen Röhren f^L, f^R, f^B und f^T der Ausfluss berechnet:

$$f^i = max(0, f^i + \Delta t * A * \frac{g * d_{x,y}}{l}),\ wobei\ i = L, R, B, T \tag{2.16}$$

Es wird dabei zuerst die Differenz $d_{x,y}$ zu dem zu vergleichenden Nachbarn berechnet und anschließend mit einem Gravitationsfaktor g multipliziert. Das soll die abfließende Menge Wasser simulieren. Anschließend wird das Ergebnis wird durch die Länge l geteilt, welcher die Entfernung zwischen den beiden Nachbarn angibt, und schließlich mit dem Zeitfaktor Δt und der Fläche A multipliziert. Die Fläche A ergibt sich aus der Differenz $d_{x,y}$ und der jeweilige Länge l in der x- oder der y-Richtung.

$$A = d_{x,y} * l_x \tag{2.17a}$$
$$A = d_{x,y} * l_y \tag{2.17b}$$

Die Fläche A wird in Abbildung 2.27 dargestellt.

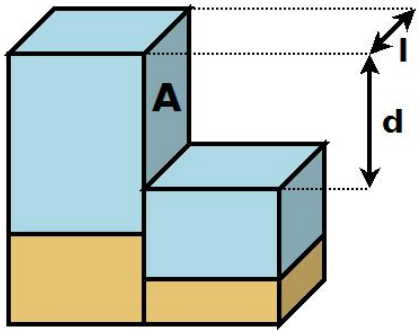

Abbildung 2.27: Berechnung der Fläche A zwischen zwei Zellen

Nun wurde der Ausfluss f^i für jede Richtung bestimmt. Es kann nun unter Umständen vorkommen, dass der Ausfluss größer als die vorhandene Wassermenge ist, sodass die Menge durch einen Skalierungsfaktor $K^f_{x,y}$ begrenzt werden muss:

$$K^f_{x,y} = min(1, \frac{w_{x,y} * l_x * l_y}{(f^L + f^R + f^T + f^B) * \Delta t}) \tag{2.18}$$

Um den endgültigen Ausfluss zu bestimmen, wird der errechnete Ausfluss mit dem Faktor $K^f_{x,y}$ multipliziert:

$$f^i = K^f_{x,y} * f^i, \ wobei \ i = L, R, B, T \tag{2.19}$$

Diese Berechnungen müssen mit allen Zellen der 2D-Matrix durchgeführt werden. Die Zellen am Rand (z. B. (x,0), (0,y)) und in den vier Ecken besitzen nicht überall Nachbarn, sodass der Ausfluss nicht berechnet werden kann (Siehe Abbildung 2.28).

Durch verschiedene Strategien können diese Randzellen trotzdem be-

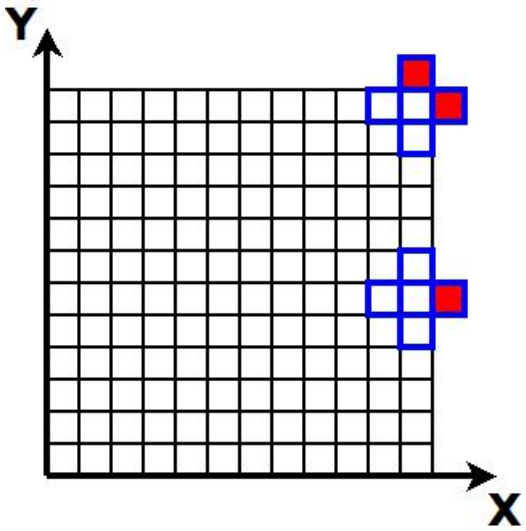

Abbildung 2.28: Probleme der Ausfluss-Berechnung der Randzellen

rechnet werden. Eine Möglichkeit wäre, alle Zellen, die den Rand der 2D-Matrix berühren, wegzulassen. Das hätte den Nachteil, dass Wasser nicht mehr aus der Landschaft fließen kann, da es dort keinen Fluchtweg hat. Ein weitere Möglichkeit wäre es die Landschaft als Torus abzubilden. Das heißt, die Höhe der außen liegenden Randzellen wird jeweils von der gegenüberliegenden Seite genommen. Ein anderer geographischer Körper, die Kugel, könnte dadurch gebildet werden, indem nur der linke und der rechte Rand als Übergang dienen. Der bessere Ansatz ist es, für die Höhe $h_{x,y}$ der außenliegenden Randelemente den Höhenwert Null anzunehmen. Die Höhe des Wassers $w_{x,y}$ ist an dieser Stelle mit dem Wert Null anzunehmen. Damit wird erreicht, dass es einen stetigen Abfluss des Wassers aus der Landschaft gibt, sobald das Wasser den Rand berührt. Dieser Ansatz ist daher zu empfehlen.

Nachdem für jede Zelle der Ausfluss-Wert f errechnet wurde, kann nun für jede Zelle der Wasserstand neu ermittelt werden. Er ergibt sich aus der Menge des Ausflusses der umliegenden Zellen abgezogen vom eigenen Ausfluss:

$$\Delta V_{x,y} = \Delta t * (f^{in}_{x,y} - f^{out}_{x,y}) \tag{2.20}$$

Der komplette Ausfluss $f^{out}_{x,y}$ ergibt aus der Summierung aller $f^{i}_{x,y}$, auch zu sehen in der Abbildung 2.29:

$$f^{out}_{x,y} = f^{L} + f^{R} + f^{B} + f^{T} \tag{2.21}$$

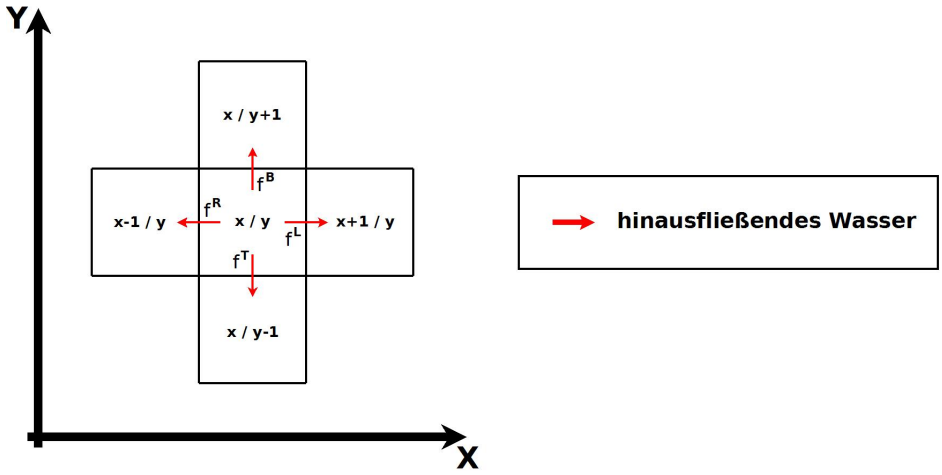

Abbildung 2.29: Outflow-flux

Das einfließende Wasser f_{in} ergibt sich aus der Summe des ausfließenden Wassers der Nachbarn. Also vom linken Nachbarn an Position $(x-1, y)$ muss der Ausfluss nach rechts f^R addiert werden. Somit ergibt sich folgende Formel:

$$f^{in}_{x,y} = f^{L}_{x+1,y} + f^{R}_{x-1,y} + f^{B}_{x,y+1} + f^{T}_{x,y-1} \tag{2.22}$$

Zu Verdeutlichung wird f_{in} in der Abbildung 2.30 veranschaulicht.

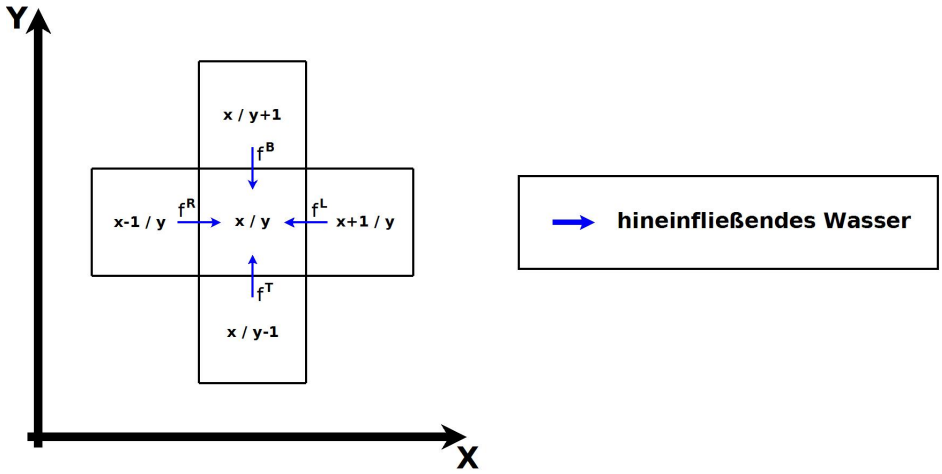

Abbildung 2.30: Input-flux

Da nun die gesamte Wasserveränderung in der Zelle berechnet wurde, kann die neue Wasserhöhe $w^{x,y}_{\Delta t+1}$ bestimmt werden:

$$w^{x,y}_{\Delta t+1} = w^{x,y}_{\Delta t} + \frac{\Delta V_{x,y}}{l_x l_y} \tag{2.23}$$

Mit diesen ganzen Berechnungen ist nun die Simulation eines Durchgangs der Wasserbewegung auf der Landschaft fertig. Für die beiden nächsten Teile des Modells, die Umwandlung in weiches Sedimentgestein und den anschließenden Transport dieses Gesteins, wird noch der sogenannte velocity vector $\vec{v}_{x,y}$, die Strömungsgeschwindigkeit des Wassers, benötigt. Er gibt an, wie viel Wasser in einer Zeiteinheit Δt die Zelle x/y durchläuft. Die Strömungsgeschwindigkeit muss sowohl für die x-Richtung, als auch die y-

Richtung der Landschaft berechnet werden.

$$\Delta W_x = \frac{f^R_{x-1,y} + f^R_{x,y} - f^L_{x,y} - f^L_{x+1,y}}{2} \tag{2.24a}$$

$$\Delta W_y = \frac{f^B_{x,y+1} + f^B_{x,y} - f^T_{x,y} - f^T_{x,y-1}}{2} \tag{2.24b}$$

Bei dieser Berechnung werden die Wasserströme miteinander verrechnet.

Um nun die beiden Elemente v_x und v_y des velocity vector $\vec{v}_{x,y}$ zu berechnen wird folgende Formel verwendet:

$$v_x = \frac{\Delta W_x}{l_x * \overline{w}} \tag{2.25a}$$

$$v_y = \frac{\Delta W_y}{l_y * \overline{w}} \tag{2.25b}$$

$\overline{w}$ stellt dabei die durchschnittliche Wasserhöhe zwischen den beiden Schritten Δt und $\Delta t + 1$ dar:

$$\overline{w} = \frac{w^{x,y}_{\Delta t+1} w^{x,y}_{\Delta t}}{2} \tag{2.26}$$

Damit ist nun auch der velocity vector $\vec{v}_{x,y}$ berechnet.

3. Sedimenterzeugung

Der nächste Schritt dieses Erosions-Algorithmus ist nun der Erosionsprozess, der sich sichtbar auf die Landschaft auswirkt. Im nächsten Schritt wird nun simuliert, dass Wasser das Gestein aufweicht und anschließend in ein Tal (=lokales Minimum) transportiert wird. Wie viel Gestein mittransportiert werden kann, ist abhängig von der Geschwindigkeit des Wassers (velocity vector $\vec{v}_{x,y}$) , der Steilheit an der Position x/y (Sinus Winkel a), und der maximale Transport-Kapazität des Wassers (Sediment-Transport-Kapazität-Konstante K_c). Der Sinus Winkel a wird hierbei durch den Höhenunterschied zum Nachbarn bestimmt (siehe Abbildung 2.31), mit dem die Stelle verglichen wird. Der Höhenunterschied in der Zeichnung wird als Talus-Winkel T bezeichnet, auf dessen Beschreibung im Kapitel 2.3.3 näher eingegangen wird.

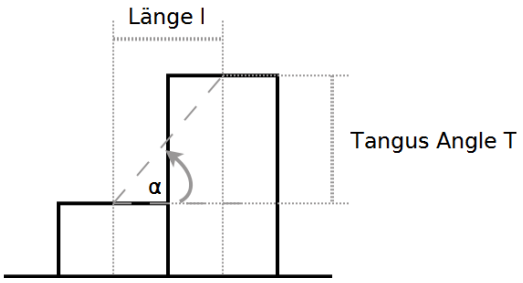

Abbildung 2.31: Sinus a in der Berechnung zum Nachbarn

Daraus wird folgende Formel abgeleitet:

$$C_{x,y} = K_c * sin(a_{x,y}) * |\vec{v}_{x,y}| \tag{2.27}$$

Als Ergebnis kommt die maximale Transportkapazität des Sediments $C_{x,y}$ heraus. Das ist die Menge an Gestein, welche die Wasserströmung maximal mittransportieren kann. Dieses Ergebnis wird benötigt, um zu berechnen, wie viel Gesteinsmaterial von der Position x/y noch abtransportiert werden kann. Das durch das Wasser fort geschwemmte Matrial wird als $n_{x,y}$ bezeich-

net. Eine Überprüfung ob $C_{x,y} > r_{x,y}$ zeigt, ob noch weiteres Gesteinsmaterial mitgeschwemmt werden kann. Falls dies zutrifft, wird nun um einen konstanten Wert K_s etwa Material von der eigentlichen Landschaft $s_{x,y}$ abgezogen und um den gleichen Wert zu dem mitgeschwemmten Material $n_{x,y}$ addiert.

$$s^{x,y}_{\Delta t+1} = s^{x,y}_{\Delta t} - K_s(C_{x,y} - n^{x,y}_{\Delta t}) \tag{2.28a}$$
$$n^{x,y}_{\Delta t+1} = n^{x,y}_{\Delta t} + K_s(C_{x,y} - n^{x,y}_{\Delta t}) \tag{2.28b}$$

Es kann auch vorkommen, dass $C_{x,y} < r_{x,y}$ ist. Das heißt, es befindet sich mehr Material im Wasser, als dieses transportieren kann. Dies kann vorkommen, wenn das Gelände flacher wird oder wenn die Strömung $\vec{v}_{x,y}$ schwächer wird. In diesem Fall setzt sich das mitgeschwemmte Material $g_{x,y}$ in Form von Regolith ab. Für die Formel wird $s_{x,y}$ als allgemeine Bezeichnung von $r_{x,y}$ verwendet.

$$s^{x,y}_{\Delta t+1} = s^{x,y}_{\Delta t} + K_s(n^{x,y}_{\Delta t} - C_{x,y}) \tag{2.29a}$$
$$n^{x,y}_{\Delta t+1} = n^{x,y}_{\Delta t} - K_s(n^{x,y}_{\Delta t} - C_{x,y}) \tag{2.29b}$$

Wenn nun das komplette mitgeschwemmten Material g der Landschaft berechnet wurde, kann der eigentliche Transport des Gesteins stattfinden.

4. Sediment-Transport

Das mitgeschwemmte Gestein kann nur mit der Strömung fließen, daher ist der velocity vector $\vec{v}_{x,y}$ wichtig für das folgende Modell von [Sta99a]. Es sagt aus, dass sich die Richtung einer Strömung nur wenig ändert, daher hat sich das mitgeschwemmte Material zu einem früheren Zeitpunkt entgegen der Strömung an einem höheren Punkt befunden. Bei dem Sediment-Transport-

Algorithmus wurde die frühere Position des mitgeschwemmten Materials $n_{x,y}$ mit einer Formel (2.30) berechnet.

$$n^{x,y}_{\Delta t+1} = n^{x-\vec{v}_x*\Delta t, y-\vec{v}_y*\Delta t}_{\Delta t} \tag{2.30}$$

Wenn sich die Position des mitgeschwemmten Materials außerhalb der 2D-Matrix befindet, so wurde definiert, dass der Durchschnitt des mitgeschwemmten Materials $n_{x,y}$ von den vier Nachbarn der N4-Nachbarschaft aufgenommen wird.

$$n^{x,y}_{\Delta t+1} = \frac{n^{x-1,y}_{\Delta t} + n^{x+1,y}_{\Delta t} + n^{x,y-1}_{\Delta t} + n^{x,y+1}_{\Delta t}}{4} \tag{2.31}$$

Dieses Modell des Sediment-Transports hat sich als einfach und schnell herausgestellt. Eine eigene Erweiterung dieses Algorithmus wird in einem späteren Kapitel bearbeitet.

5. Verdunsten

Nach dem bisherigen Modell kann Wasser aus der Landschaft hinauslaufen, wenn es die Landschaftsgrenzen erreicht. Darüber hinaus soll das Wasser auch innerhalb der Landschaft verdunsten können. In der Simulationen wird davon ausgegangen, dass die Temperatur immer gleich ist. Um die Verdunstung zu simulieren, wird am Ende einer Iteration in Abhängigkeit der Verdunstungskonstante K_V etwas Wasser entfernt.

$$w_{x,y} = w_{x,y} * (1 - K_V * \Delta t) \tag{2.32}$$

Damit ist der komplette Kreislauf der Wassererosion abgeschlossen. Er muss nun so oft wiederholt werden, bis die gewünschten Veränderungen eingetreten sind.

Überblick über die Wassererosion

Das komplette Modell ist sehr komplex und weist sehr viele mathematische Formeln auf. Aus diesem Grund gibt es einen Überblick über die Reihenfolge der einzelnen Schritte. $t+1$ weißt darauf hin, dass dieser Parameter für diesen Iterationsschritt fertig berechnet wurde und im nächsten Iterationsschritt wiederum als Input t dient.

$$
\begin{aligned}
w_1 &\longleftarrow Wassererzeugung(w_{\Delta t}) \\
w_2, f_{\Delta t+1}, \vec{v}_{\Delta t+1} &\longleftarrow Wasserfluss(w_1, s_{\Delta t}, f_{\Delta t}) \\
s_{\Delta t+1}, r_1 &\longleftarrow Sedimenterzeugung(\vec{v}_{\Delta t+1}, s_{\Delta t}, r_{\Delta t}) \\
r_{\Delta t+1} &\longleftarrow Sediment - Transport(\vec{v}_{\Delta t+1}, s_{\Delta t+1}) \\
w_{\Delta t+1} &\longleftarrow Verdunsten(w_2)
\end{aligned}
$$

Das gesamte Modell basiert auf der N4-Nachbarschaft könnte aber auch auf die N8-Nachbarschaft erweitert werden. Versuche [MDH07] zeigten, dass sich allerdings kaum sichtbare Verbesserungen ergaben, die den doppelten Berechnungsaufwand rechtfertigen würden.

2.3.2 Erosion durch stehende Gewässer

In diesem auch Dissolution-based Algorithm genannten Modell wirkt stehendes Gewässer auf die darunter liegenden Sedimentsschichten. Wie in dem in Kapitel 2.2 gezeigten Beispiel der Brown'sche Bewegung besitzt stehen-

des Wasser eine permanente, stetige Bewegung. Dies sorgt dafür, dass über einen längeren Zeitraum, die obere Gesteinsschicht zu feinem Regolith zerrieben wird. Diese Bewegung hat des Weiteren zur Folge, dass sich der Boden immer mehr angleicht, also nahezu eben wird. Wenn nun das Wasser langsam verdunstet, versickert oder abfließt, bleibt von diesem Schlick immer etwas zurück. Dies ist sehr gut in der Abbildung 2.32 zu sehen. Dieser Schlick härtet danach aus und verbindet sich mit der darunter liegenden Gesteinsschicht (Näheres dazu in Kapitel 2.3.4).

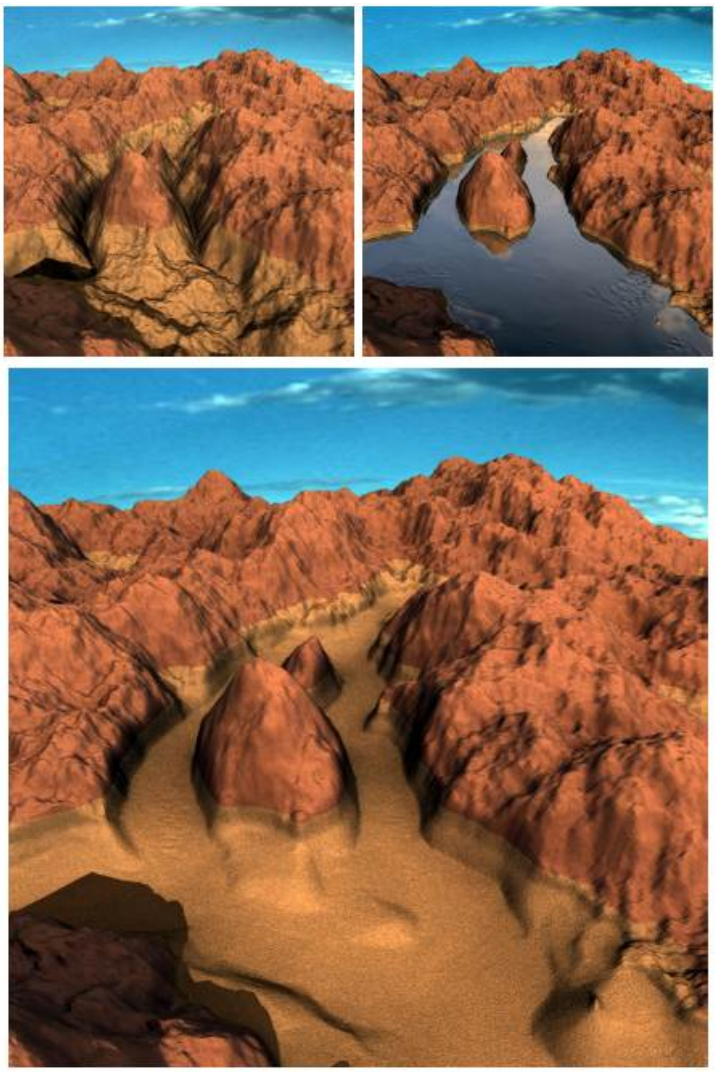

Abbildung 2.32: Erosion durch stehende Gewässer [BvBK08]

Das oben genannte Modell basiert auf der Arbeit von Michael Kass et al. [KM90], wurde dann von Bedřich Beneš erweitert [Ben07], um anschließend auf das Pipe-Modell, welches im vorherigen Kapitel 2.3.1 vorgestellt wurde, angepasst zu werden [BvBK08].

Für das Modell werden die Bezeichnungen aus dem vorhergehenden Kapitel verwendet. Sie können auch im Anhang auf der Seite 125 nachgeschlagen werden.

Diese neue Schicht $r_{x,y}$ wird erzeugt, indem das Einsickern von Wasser in diese Schicht simuliert wird. Dazu wird die Konstante K_s für die Schicht s definiert, die aufzeigt, wie tief das Wasser in die Schicht eindringt, um die oberste Sedimentschicht in Regolith umzuwandeln. Die Umwandlung erfolgt durch folgende Vorschrift:

$$r_{x,y} = \begin{cases} w_{x,y} > K_s & : w_{x,y} \\ w_{x,y} \leq K_s & : K_s \end{cases} \tag{2.33}$$

Die entsprechende Wassermenge $w_{x,y}$ muss anschließend noch entfernt werden, da sie eingesickert ist.

$$w_{x,y} = w_{x,y} - r_{x,y} \tag{2.34}$$

Durch diese Umwandlung kann die maximale Höhe der Regolith-Schicht den Wert der Konstante K_s annehmen. Wenn das Wasser verdunstet oder komplett eingesickert ist, kann der Algorithmus der Thermalerosion (Kapitel 2.3.3) auf den Untergrund einwirken.

2.3.3 Thermalerosion

Eine weitere physikalische Größe, die sich abtragend auf eine Landschaft auswirken kann, ist die wärmebedingte oder auch thermale Erosion (eng. Thermal Weathering). [MKM89] [Mus93] [Ols04]
Diese Erosion entsteht, wenn Sonne den Boden ausdorrt und diese Gesteinsschichten gelegentlich abrutschen. Dadurch entwickelt sich eine gleichmäßig abfallende Region an losem Gestein (eng. Talus Slops). In natura sieht das aus, wie in Abbildung 2.33.

Abbildung 2.33: Abgerutschtes Gestein durch Termalerosion[4]

Das erste Modell, das diesen Vorgang abbildete, wurde 1989 von Forest Kenton Musgrave [MKM89] entwickelt. Hierbei wird die Differenz der Höhen zwischen den in der N4-Nachbarschaft liegenden Zellen verglichen. Die Formel für die Differenzberechnung ist identisch mit Formel (2.13), die auf Seite 36 eingeführt wurde.

Der von Forest Kenton Musgrave entwickelte Vergleich lautet wie folgt:

$$s_{\Delta t+1}^{x_n,y_n} = \begin{cases} d_{\Delta t}^{x,y} > T_A & : s_{\Delta t}^{x_n,y_n} + K_{\Delta t}(d_{\Delta t}^{x,y} - T_A) \\ d_{\Delta t}^{x,y} \leq T_A & : s_{\Delta t}^{x,y} \end{cases} \tag{2.35}$$

Bei diesem Modell wird zu jedem Zeitschritt $\Delta t + 1$ überprüft, wie sich

[4] http://www.itecuk.com

die Höhe der Sedimentsschicht s an der Stelle x, y zu der Höhe s des Nachbarn x_n, y_n verhält. Die Differenz wird mit dem Talus Winkel (eng. talus angle) T_A verglichen. Der Talus-Winkel kann berechnet werden, indem die Differenz der Höhe aus dem gewünschten Gefälle berechnet wird. Das wird in Abbildung 2.34 verdeutlicht.

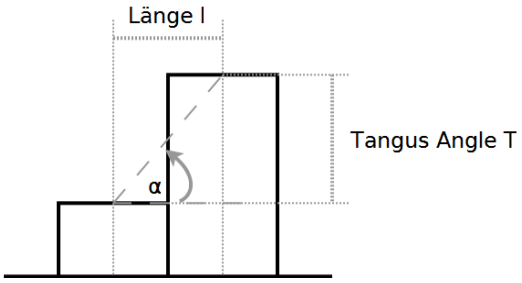

Abbildung 2.34: Talus Angle

Daraus ergibt sich folgende Formel:

$$T_A = tan(\ \alpha\ ) * l \tag{2.36}$$

Der Talus-Winkel kann je nach Bodenbeschaffenheit unterschiedlich groß sein. Physikalisch korrekte Werte sind in keiner Arbeit angegeben und müssen erst durch Versuchsreihen ausgetestet werden (näheres dazu in Kapitel 3.1.2).

Wenn nun die Differenz $d_{\Delta t}^{x,y}$ durch andere Erosionsarten größer als T_A wird, so wird die Differenz mit einem festen Prozentsatz $K_{\Delta t}$ multipliziert und auf den Nachbarn mit der niedrigeren Höhe addiert. Dieses $K_{\Delta t}$ gibt an wie viel Prozent Gesteinsmaterial in die nächste Zelle der Matrix rutscht. Zeitgleich muss diese Differenz $K_{\Delta t}(d_{\Delta t}^{x,y} - T_A)$ vom höheren Nachbarn abgezogen werden:

$$s_{\Delta t+1}^{v} = s_{\Delta t}^{v} - K_{\Delta t}(d_{\Delta t}^{x,y} - T_A) \tag{2.37}$$

Dieses Vorgehen wird in Abbildung 2.35 verdeutlicht. Im ersten Iterationsschritt t_0 ist die Differenz d_2 größer als T_A; es rutscht Gestein nach unten. In Iterationsschritt t_1 wurde die Höhe bei s_0 verringert und bei s_2 hinzu addiert.

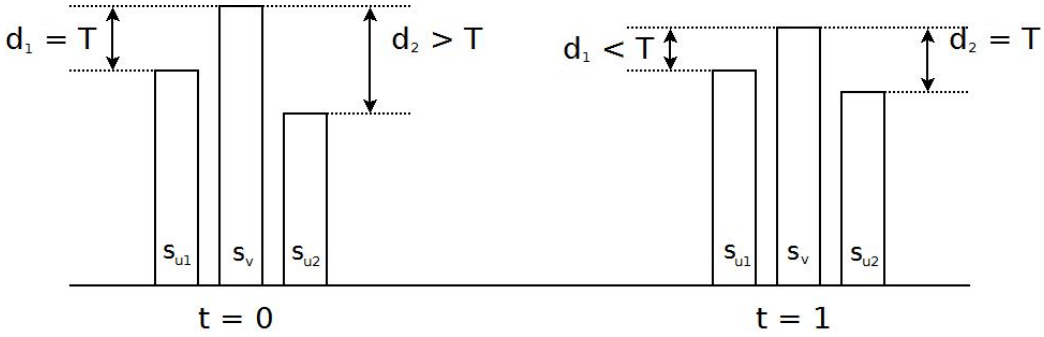

Abbildung 2.35: Beispiel der thermalen Erosion

In diesem Beispiel wurden nur jeweils der linke und der rechte Nachbar betrachtet. In der Anwendung müssen aber alle vier möglichen (N4-Nachbarschaft) Kandidaten ausgewertet werden. Es kann nun dabei vorkommen, dass es mehrere Kandidaten gibt, das heißt, es gibt mehrere Nachbarn für die $d_{\Delta t}^{x_n,y_n} > T_A$ zutrifft. In diesem Fall wird das Sediment anteilig auf die neue Höhen der Kandidaten addiert. Nach dem Verfahren von Jacob Olsen [Ols04] wird nun wie folgt berechnet:

$$s_{\Delta t+1}^{x_n,y_n} = s_{\Delta t}^{x_n,y_n} + K_{\Delta t}(d_{\Delta t}^{max} - T_A) * \frac{d_{\Delta t}^{x_n,y_n}}{d_{\Delta t}^{total}} \tag{2.38}$$

$d_{\Delta t}^{max}$ ist der größte Differenzwert der Werte, die größer T_A sind, $d_{\Delta t}^{x_n,y_n}$ der Differenzwert für die Position u und $d_{\Delta t}^{total}$ die Summe aller Differenzwerte die größer als T sind. Zum Schluss muss nur noch die abgerutschte Sedimentsschicht vom Mittelpunkt abgezogen werden.

$$s_{\Delta t+1}^{v} = s_{\Delta t}^{v} - K_{\Delta t}(s_{\Delta t}^{max} - T_A) \tag{2.39}$$

Damit ist die thermale Erosion abgeschlossen.

2.3.4 Gesteinsveränderung

Ein für das menschliche Auge kaum wahrnehmbarer Alterungsprozess ist die Gesteinsveränderung, die durch Druck über einen längeren Zeitraum entstehen kann.

Wie in den vorherigen Kapiteln vermittelt, werden für die gesamten Erosionsprozesse verschiedene Gesteinsschichten benötigt. In den meisten bisher beschriebenen Erosionsprozessen spielt die Art des Gesteins kaum eine Rolle, da die Prozesse nur auf die oberen Gesteinsschichten wirken. Das einzige Modell, welches zwei unterschiedliche Schichten unterscheidet, jenes der Hydraulic Erosion (Kapitel 2.3.1). Da die Oberfläche der Erde aus Gesteinsschichten besteht, sollten diese nicht unberücksichtigt bleiben. In Abbildung 2.36 ist eine schematische Darstellung einer Bohrprobe zu sehen.

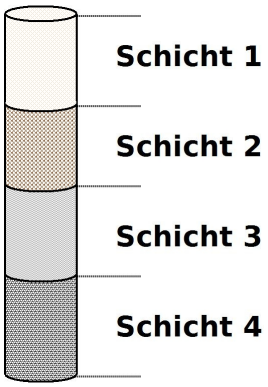

Abbildung 2.36: Schichtenmodell

Daran ist zu erkennen, dass diese Schichten sehr streng voneinander getrennt sind. Diese Schichten besitzen jeweils verschiedene physikalische Eigenschaften, die sich durch Druck über einen längeren Zeitraum ändern können.

In dem Paper *Layered Data Representation for Visual Simulation of Ter-*

rain Erosion [BF01] von Bedřich Beneš et al. wurde dieses Schichtenmodell zum ersten Mal beschrieben und umgesetzt. Zur Anwendung kam es allerdings nur bei der thermalen Erosion (Kapitel 2.3.3).

Die Veränderung der Gesteinsschichten oder die Verschmelzung gleichartiger Schichten wurde aber von Bedřich Beneš, et al. nicht beachtet. Dieses Phänomen wurde erst sieben Jahre später in einem sehr kurzen Abschnitt im Paper von Ondřej St'ava et al. [BvBK08] aufgegriffen, das aber auf keinen geologisch fundierten Arbeiten basiert.

Bei Ondřej St'ava et al. lagen die Unterschiede der einzelnen Schichten nur in der Härte. Dies hatte zur Folge, dass eine weichere Schicht durch Wassererosion leichter abgetragen werden konnte, als eine härtere.

Sie definierten, dass sich immer eine weichere Schicht s über einer härteren Schicht befindet und wenn sich über eine Zeitspanne keine Änderung in der weicheren Schicht ergibt, so wird diese ein Teil der härteren Schicht. Die Veränderungen einer Schicht kann das Abrutschen durch thermale Erosion (Kapitel 2.3.3) oder das Abschwemmen durch Wasser (Kapitel 2.3.1) sein.

Ihr Ansatz war es, für jede Schicht einen Zähler zu definieren, der in jedem Iterationsschritt hochgezählt wurde, wenn auf der Schicht keine Veränderung vorgenommen wurde. Falls es eine Änderung gab, wurde der Zähler auf Null zurückgesetzt. Wenn der Zähler einen bestimmten Wert erreicht hatte, wurde diese Schicht mit der daunterliegenden Schicht vereint.

2.4 State of the Art

Seit den Anfängen Ende der 1970er wurden sehr viele Bereiche bei fraktalen Landschaften untersucht. Sei es bei der Generierung der Landschaften, den anschließenden Erosions- und Alterungsprozessen oder der visuellen Darstellung.

Bei der Generierung von Landschaften müssen die Generatoren in den Kapiteln 2.2.1 und 2.2.2 als Quasi-Standard angesehen werden. In dieser Richtung wird aktuell weniger geforscht.

Es gibt vereinzelte Bestrebungen, die Generierung von Landschaften mit physikalischen Prozessen zu vereinen. In der Arbeit *Creating Landscapes with Simulated Colliding Plates* von 2006 [JE06a] wurde ein Generator vorgestellt, der Landschaften erzeugen konnte, indem er die Kollision zweier Kontinentalplatten simulierte. Als Ergebnis kamen Landschaften heraus, die über physikalisch korrekte Gebirgszüge verfügten. Die Berechnung dieser Landschaften dauerte deutlich länger als bei den bekannten Generatoren um eine Urlandschaft zu erzeugen. Auf diese durch Kollisionen erzeugten Gebirge müssen am Ende trotzdem Erosions- und Alterungsprozesse angewandt werden, damit sie „natürlich“ aussehen. Es wurde also nichts verbessert.

Dieser Teilbereich der Forschung ist fast komplett zum erliegen gekommen. Die bekannten Generatoren liefern fast durchweg in Sekundenschnelle eine fertige Urlandschaft, die schon sehr „natürlich“ aussieht.

Ähnlich sieht es aktuell auch bei der Forschung nach neuen Erosions- und Alterungsmodellen aus. Die in Kapitel 2.3 vorgestellten Modelle sind der aktuelle Forschungsstand. Die gegenwärtigen Arbeiten, die sich mit Erosions- und Alterungsmodellen befassen, beschäftigen sich damit, die bekannten Modelle in Echtzeit auf dem Computer darzustellen. Eine Arbeit von 2006 an der TU München [JS06] beschäftigte sich mit der Darstellung und der Veränderung von Landschaften in Echtzeit. Bei diesem Forschungsprojekt spielten Erosions- und Alterungsmodelle noch eine untergeordnete Rolle. Ein Jahr später wurde die Erosion durch fließendes Gewässer (Kapitel 2.3.1) auf dem Grafikprozessor simuliert [MDH07]. Das Ergebnis wurde anschließend als Video[5] veröffentlicht. 2008 nahm sich Bedřich Beneš ebenfalls dieses Themas an [Ben07]. Er war jahrelang sehr aktiv bei der Forschung von physikalischen Prozessen auf fraktalen Landschaften. Aktuell liegt der Schwerpunkt seiner Forschungsarbeit aber bei der Generierung von Städten.

[5] http://www.youtube.com/watch?v=9kLoZShDr0o

Bis auf die Konvertierung der Erosions- und Alterungsmodelle auf die schnellen Grafikprozessoren ist in den letzten vier Jahren keine mir bekannte Arbeit erschienen, die ein neues Erosionsmodell vorgestellt hat.

Der Hauptschwerpunkt ist die visuelle Darstellung der Landschaften. Hier wird eigentlich immer sehr viel Forschung betrieben, auch wenn sie eher von der Wirtschaft vorangetrieben wird. Daher gibt es hier kaum wissenschaftliche Arbeiten.

Eine große Firma, die immer wieder neue Technologien erforscht, ist NVidia. Sie ist eine der wenigen Firmen die eine andere Veröffentlichungstaktik verfolgen. Viele dieser Technologien werden anschließend von ihren neuesten Grafikkarten unterstützt und in den *GPU Gems* Büchern vorgestellt [6]. Hierbei sind es Elemente der Shader-Technologie, die verbessert werden.

Auch im Spielemarkt werden immer wieder neue Technologien entwickelt. Eine Firma, die immer wieder graphische Neuerungen erfindet, ist id Software. Sie hat 2009 eine neue Technologie, genannt Megatexture, vorgestellt, um hochauflösende Texturen auf Landschaften zu mappen. Damit ist es nun möglich bis zu 128000*128000 Pixel große Texturen zu verwenden, ohne Performanceeinbußen hinzunehmen. Über diese neue Technologie wurde von J.M.P. van Waveren auf der SIGGRAPH 2009 eine Präsentation[7] gehalten.

2010 stellte die australische Firma Euclideon ihre Grafiktechnologie Unlimited Detail Technology[8] vor. Die auf Voxeln basierende Grafikengine soll in Echtzeit gigantische Landschaften bis auf kleinste Steinchen darstellen können. Die Technik wird Point Cloud Data System genannt. Hierbei können sie extrem effizient bestimmte Voxel anzeigen oder ausblenden. Kombiniert mit einem regional begrenzten Generator, wie dem Successive Random Additions Algorithmus, kann beliebig nah an die Landschaft heranzoomt werden.

[6] http://http.developer.nvidia.com/GPUGems3/gpugems3_ch01.html
[7] http://s09.idav.ucdavis.edu/talks/05-JP_id_Tech_5_Challenges.pdf
[8] http://unlimiteddetailtechnology.com/

Eine Firma, die sich nur mit der visuellen Darstellung von fraktalen Landschaften beschäftigt, ist Planetside Software. Sie hat die Software Terragen© entwickelt, die zwar auch mit Voxeln arbeitet, aber nicht echtzeitfähig ist. Das Bild 2.2 wurde beispielsweise mit dieser Software erstellt. Viele Spieleherstel-ler benutzten Terragen um Texturen zu erzeugen und auch Hollywood benutzt gelegentlich das Tool. Aktuell sind Grafiken, die damit erzeugt wurden, in TRON: Legacy [9] zu sehen. Einige Beispielbilder, die im hinteren Teil zu finden sind, wurden mit dieser Software gerendert.

Es ist zu vermuten, das sich dieser Trend weiter fortsetzt. Das Hauptaugenmerk wird vermutlich weiterhin bei der graphischen Umsetzung fraktaler Landschaften liegen.

[9] http://www.planetside.co.uk/content/view/62/100/

A cloud is made of billows upon billows upon billows that look like clouds. As you come closer to a cloud you don't get something smooth, but irregularities at a smaller scale.

Benoît Mandelbrot

Kapitel 3

Konzeption

Im folgenden werden nun die im vorausgegangenen Kapitel vorgestellten Modelle zu einem Einzgen vereint. Der erste Schritt dazu ist es, die in Kapitel 2 vorgestellten Elemente aufeinander anzupassen. Anschließend werden notwendige Änderungen als auch Verbesserungen besprochen.

3.1 Zusammenführen der vorhandenen Modelle

Der erste Schritt, der für diese Ausarbeitung wichtig ist, ist die Wahl einer geeigneten Datenstruktur.

3.1.1 Auswahl einer geeigneten Datenstruktur

Wie schon im Kapitel 2.1 angesprochen besitzen Heightmap, Phyxel und Layered Data Representation unterschiedliche Eigenschaften, die nicht unbedingt für die Erosions- und Alterungsprozesse geeignet sind. Damit eine

Aussage getroffen werden kann, welche Datenstruktur für die Anwendung der Erosions- und Alterungsprozesse am besten geeignet ist, müssen die Anforderungen analysiert werden.

Die Datenstruktur soll in der Lage sein einen Höhenwert innerhalb einer 2D-Matrix zu speichern. Des Weiteren soll sie, wie in Kapitel 2.3.4 vermittelt, mehrere Schichten speichern können. Jede dieser Schichten kann jeweils unterschiedliche Eigenschaften aufweisen, zum Beispiel wie viel Wasser in Sediment umgewandelt wird (Formel (2.33) auf Seite 50) oder der Talus Angle (Formel (2.35) auf Seite 51).

Die Heightmap als Datenstruktur für diese Arbeit scheitert daran, dass sie nur eine einzige Schicht verwalten kann. Um zusätzliche Schichten verwalten zu können, muss zusätzlicher Aufwand betrieben werden. Denkbar wäre eine Hightmap für jede Schicht oder gar eine andere Datenstruktur.

Von den beiden verbliebenen Datenstrukturen können alle mehrere Schichten speichern. Ebenso sind sie in der Lage unterschiedliche physikalische Eigenschaften zu verwalten. Daher kann eine entsprechende Auswahl nur über den Speicherverbrauch und den Aufwand für die Verwaltung bestimmt werden.

Für die Testlandschaft wird festgelegt, dass die Variable N = 256 und pro Zeile in der Matrix acht Gesteinsschichten besitzt. Der Speicherverbrauch liegt bei Phyxel durch die Berechnung

$$256 * 256 * 256 = 16777216 * 32\ Bit = 536870912\ Bit$$

Die Anzahl der Schichten bleibt bei Phyxel unberücksichtigt, da jede Zelle im Raum einem bestimmten Gesteinstyp entspricht. Bei der Layered Data Representation hingegen spielt die Anzahl eine wichtige Rolle, da hier jede Schicht nur einmal existiert. Dafür muss hier noch zusätzlich die Höhe jeder einzelnen Schicht gespeichert werden. Bei der maximalen Höhe von

256 ergibt sich ein zusätzlicher Speicherverbrauch von 8 Bit, also 40 Bit pro physikalischer Einheit.

$$256 * 256 * 8 = 524288 * 40\ Bit = 20971520\ Bit$$

Das entspricht in etwa 3.9% der Phyxel-Darstellung. Somit liegt hier ein deutlicher Vorteil bei der Layered Data Representation. Dieser wird auch solange bestehen bleiben, bis die Anzahl der Schichten größer werden sollte als den Speicheraufwand der Phyxel. Bei der Speicheraufwand der Phyxel beträgt nur (32 Bit / 40 Bit * 100 =) 80 % im vergleich zu der Layered Data Representation. 80 % von 256 möglichen Erdschichten sind 205. Ab dieser Anzahl übersteigt der Speicherwaufwand der Layered Data Representation dem der Phyxel.

Ein anderer Punkt wäre der Aufwand, der für die Verwaltung betrieben werden muss, um die Landschaft zu speichern. Da sowohl bei Phyxel als auch bei der Layered Data Representation die Grundfläche von 256 x 256 identisch ist, kann ein möglicher Unterschied höchstens in der Verwaltung der Schichten liegen. Bei Phyxel müssen immer 256 Elemente gespeichert bleiben, bei der Layered Data Representation bestenfalls eine Schicht im schlimmsten Fall 256. 256 kann trotz acht unterschiedlichen physikalischen Elementen vorkommen, da mehrere Schichten gleichen Typs von Anderen getrennt werden können. Der Aufwand ist in diesem Fall auch nicht höher als die der Phyxel.

Mit all diesen Überlegungen wird das gleiche Ergebnis erreicht wie die Arbeit von Bedřich Beneš und Rafael Forsbach [BF01]. Mit der Simulation von Erosionsprozessen auf fraktale Landschaften die Layered Data Representation den voxelbasierten Phyxel vorzuziehen.

3.1.2 Entwicklung der physikalischen Schichten

Der nächste Schritt, der gemacht werden muss, ist eine Zusammenstellung der einzelnen Schichten. In keiner Arbeit wurde auf die näheren physikalischen Gegebenheiten der einzelnen Schichten eingegangen, sodass dieser Punkt ebenfalls von mir bearbeitet werden muss.

Die in dieser Arbeit vorgestellten physikalischen Formeln sind eine angenäherte Darstellungsweise der Wirklichkeit. Daher auch in diesem Fall nicht alle möglichen Gesteinsschichten simuliert, sondern nur einige, wenige, repräsentative Materialien.

Zur Auswahl wurde der Gesteinskreislauf [Hau08] genommen, der auch in der Abbildung 3.1 zu sehen ist. Magma steigt auf und erstarrt zu magmatischem Gestein (Erstarrungsgesteine). Dieses wandelt sich durch Druck und Temperatur in metamorphe Gesteine (Umwandlungsgesteine) und durch Erosion in Sedimentgesteine (Ablagerungsgesteine). Jede Gesteinsart wird irgendwann aufgeschmolzen und wieder zu Magma werden.

Das magmatische Gestein ist die härteste Gesteinsart. Es kommt im Boden in sehr großen Tiefen vor. Erst durch entsprechende Alterungsprozesse kann magmatische Gestein an die Oberfläche wandern. In dieser Arbeit sollen die Erosions- und Alterungsprozesse einer Urlandschaft simuliert werden, sodass diese Gesteinsart eher in den tieferen Schichten zu finden sein wird. Es wird daher Granit als Repräsentant für dieses Erstarrungsgestein ausgewählt.

Die Gesteinsarten, welche sich aus dem Magma entwickelt und noch nicht mit denn Erosionsprozessen reagiert haben, ist das metamorphem Gesteine. Es entsteht durch Druck und hohe Temperaturen und ist nach den Erstarrungsgestein das nächst härteste Gestein. Für die Simulation dieser Gesteinsart wurde Marmor gewählt.

Ein Gestein, welches zwischen metamorphen Stein und Sedimentgestein liegt, ist Schiefer. Auch dieses Gestein wurde für die Simulation verwendet.

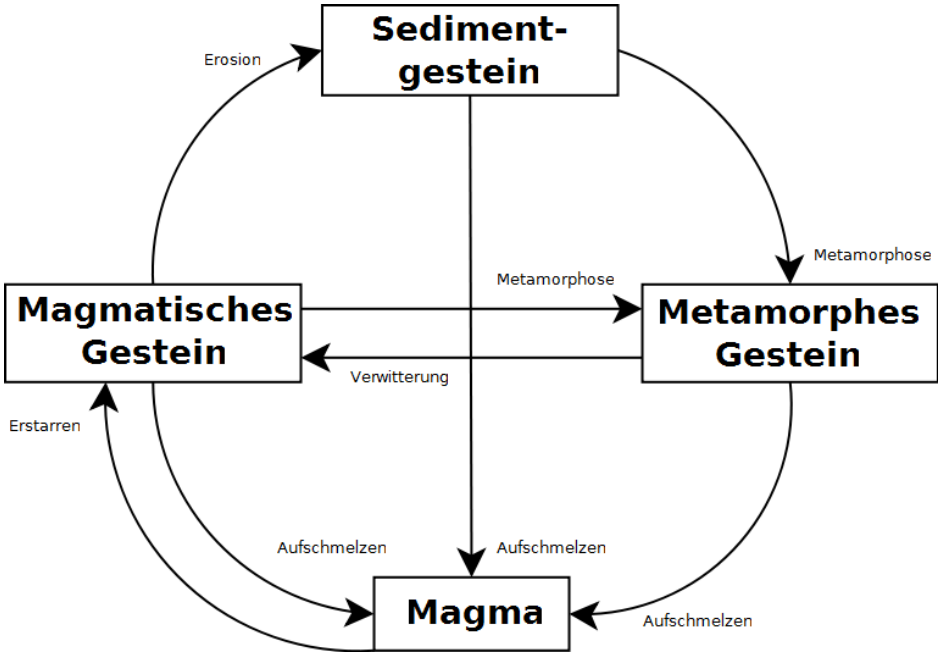

Abbildung 3.1: Gesteinskreislauf

Die wichtigeren Schichten sind jene, die direkt mit den Erosionsprozessen in Berührung kommen. Das wären die Sedimentgesteine. Aus diesem Grund wird mehr Sedimentgesteine simuliert als bei den anderen Gesteinsarten. Diese Art kann noch einmal in zwei Unterkategorien eingeteilt werden: die terrestrischen Sedimentgesteine und die Lockersedimente. Der Unterschied zwischen diesen beiden ist der Zusammenhalt. Terrestrische Sedimentgesteine sind fester und weisen eine klare Schichtenbildungen auf. Darunter fallen Arten wie Sand- oder Kalkstein, die auch für diese Arbeit verwendet werden sollen. Lockersedimente besitzen keinen klaren Zusammenhalt, da sie aus kleinen Steinchen bestehen. Ein Vertreter dieser Art ist Sand, der auch für diese Arbeit verwendet wird.

Da nun soweit alle Gesteinsschichten über ein oder zwei Repräsentanten verfügen, fehlen nur noch zwei weitere Elemente, die abgebildet werden müssen. Das Erste davon ist Regolith, welches durch die bekannten Erosionsformeln entsteht. Das zweite Element, welches abgebildet werden muss, ist Wasser. Damit stehen nun alle Elemente zur Verfügung, um das Schichtenmodell anzuwenden:

- Magmatisches Gestein
  - Granit
- Metamorphes Gestein
  - Marmor
- Magmatisches + metamorphes Gestein
  - Schiefer
- Sedimentgestein
  - Kalkstein
  - Sandstein
  - Sand
- andere Elemente
  - Regolith
  - Wasser

Wegen fehlender Angaben in der Quellenliteratur mussten alle Parameter, die für die Modelle nötig sind, selbst entwickelt werden. Da es sich hierbei um keine physikalisch korrekten Modelle handelt, wurden sie anhand von Versuchsreihen ermittelt.

Name	Talus-Winkel	K_s
Granit	70	0.05
Marmor	65	0.1
Schiefer	60	0.30
Kalkstein	60	0.65
Sandstein	50	0.70
Sand	45	0.90
Regolith	20	1

Tabelle 3.1: Parameter der einzelnen Schichten

Ein weiterer Aspekt, der betrachtet werden muss, ist der Übergang der einzelnen Schichten in einen festeren Zustand. Bei dem in der Arbeit [BvBK08] angesprochenen Vorgang wurde das weiche Regolith, zur darunterliegenden Schicht transformiert, wenn nach einer gewissen Zeit kein Wasser oder neues Regolith dazukam. Die Zeitspanne dafür wurde vom Härtegrad der darunter liegenden Schicht definiert.

Das Regolith muss in den Gesteinskreislauf eingegliedert werden. Das sehr lockere Regolith verwandelt sich nach einer kleinen Zeitspanne in lockeren Sand. Dieser wiederum in Sandstein. Dies kann so lange fortgeführt werden, bis sich das Regolith in Granit verwandelt hat.

Es wurde anhand des Gesteinskreislaufs und der physikalischen Dichte (nach der Liste von SiMetric [Sim11]) ein Ablauf generiert, der dem natürlichen Kreislauf angenähert ist. Dieser Ablauf wird in Bild 3.2 dargestellt.

In der Abbildung ist nun zu sehen, dass sich jedes Material, nach dem in der Tabelle 3.1 definierten Parametern in Regolith verwandelt. Anschließend durchläuft das Regolith eine Metamorphose, wobei hier die Zeitspanne, in der das Material den Zustand ändert, eine Potenzreihe zur Basis vier ist. Gleichzeitig wird die Menge der Schicht halbiert, um damit den Druck der Schicht zu simulieren.

Diese Werte haben sich in den Simulationen als gut herausgestellt.

Nachdem nun sowohl die komplette Datenstruktur ausgewählt ist, als auch die einzelnen Schichten simuliert werden, können die Modelle angepasst werden.

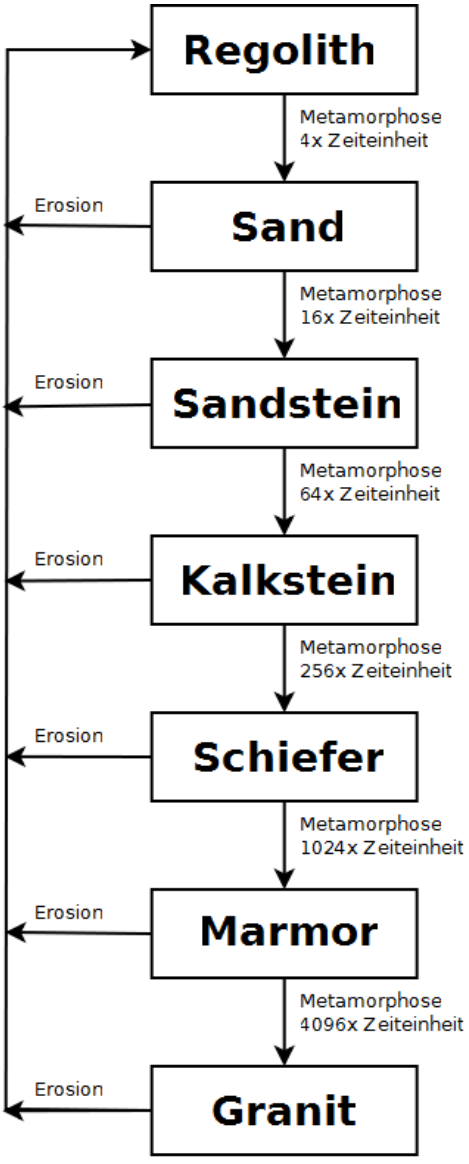

Abbildung 3.2: Entwickelter Gesteinskreislauf

3.1.3 Erweiterung der Erosion durch fließendes Gewässer

Die Umsetzung des Modells der Erosion durch fließendes Wasser ist insgesamt der umfangreichste Teil, da einige Berechnungen aus dem Kapitel 2.3.1 leicht angepasst wurden. Grund dafür sind Optimierungen der Geschwindigkeit, da die Formeln, wie schon in Kapitel 2.3.1 erwähnt, zur Verwendung auf der GPU optimiert sind. Die Zeiten für die Berechnung werden im späteren Kapitel 4 besprochen.

Somit Variablen und dementsprechend überflüssige Berechnungen eingespart werden können, müssen diese erst identifiziert werden. Die Landschaft besteht aus Zellen, die der 2D-Matrix entsprechen. Der Abstand dieser Zellen wurde für die Simulation immer mit einem Meter angegeben. Daher besteht auch jeweils die Länge l_x und l_y aus dem Wert eins. Dieser kann insgesamt in den Formeln (2.16), (2.17), (2.18), (2.23) und (2.25) eingespart werden.

Eine weitere Variable, die identifiziert wurde ist die Zeitvariable Δt. Da auch hier immer mit der Zeiteinheit eins gerechnet wurde kann sie in den Formeln (2.18), (2.20) und (2.32) weggelassen werden.

Indem die Länge eingespart wird, kann auch auf die Formel (2.17) zur Berechnung der Verbindungsfläche zwischen zwei Zellen komplett verzichtet werden. Aus der Formel $A = d_{x,y} * l$ wird dadurch die Formel $A = d_{x,y}$. Da A die Fläche angibt, durch die das Gesteinsmaterial zwischen den Zellen ausgetauscht werden kann, wird angenommen, das immer die komplette Differenz $d_{x,y}$ ausgetauscht werden kann.

Damit wird die Formel (2.16), die den Ausfluss einer Zelle berechnet, durch folgende Formel ersetzt:

$$f^i = max(0, f^i + g * d_{x,y}),\ wobei\ i = L, R, B, T \tag{3.1}$$

Die Einsparung beträgt somit die komplette Berechnung von A. Das sind vier Multiplikationen für eine Zelle und jeweils vier Multiplikationen und vier Divisionen für die Berechnung von f^i.

In bestimmten Fällen ist es nötig den Ausfluss auf mehrere Nachbarzellen zu verteilen. Die in der Formel (2.18) berechnete Skalierung benötigt in der originalen Formel jeweils die zwei Längen l_x und l_y, aber auch die Zeitvariable Δt. Sie werden in insgesamt drei Multiplikationen benötigt, die eingespart werden können.

$$K^f_{x,y} = min(1, \frac{w_{x,y}}{(f^L + f^R + f^T + f^B)}) \tag{3.2}$$

Die Zeitvariable Δt kann in der Berechnung (2.20) und somit pro Zelle in einer Multiplikation eingespart werden.

$$\Delta V_{x,y} = f^{in}_{x,y} - f^{out}_{x,y} \tag{3.3}$$

Ebenfalls durch die Entfernung von l_x und l_y kann bei der Berechnung der neuen Wasserhöhe (2.23) für jede Zelle eine Multiplikation und eine Divisionen eingespart werden. Daraus wird die folgende Formel generiert:

$$w^{x,y}_{\Delta t+1} = w^{x,y}_{\Delta t} + \Delta V_{x,y} \tag{3.4}$$

Bei der Berechnung des velocity vector (2.25), der für jede Zelle und jedem Iterationsschritt einmal benötigt wird, können jeweils zwei Multiplikationen eingespart werden. Damit werden die beiden Richtungsvektoren v_x

und v_y nach der neuen Formel berechnet:

$$v_x = \frac{\Delta W_x}{\overline{w}} \tag{3.5a}$$

$$v_y = \frac{\Delta W_y}{\overline{w}} \tag{3.5b}$$

Zu Letzt kann in der Formel (2.32), die das Verdunsten simuliert, die Zeitvariable Δt und somit eine Multiplikation eingespart werden. Dieser Fall tritt aber nur dann auf, wenn Wasser vorhanden ist, für das gilt:

$$w_{x,y} = w_{x,y} * (1 - K_V) \tag{3.6}$$

Das ergibt insgesamt eine Einsparung von acht Multiplikationen und vier Divisionen für jede Zelle und jedem Iterationsschritt. Wenn noch die beiden Fälle Skalierung dar Wasserhöhe (3.2) und Verdunstung (3.6) zutreffen, so werden abermals vier Multiplikationen weniger benötigt. Das ergibt bei einer Heightmap mit N=256 eine Einsparung von 786.432 bis 1.048.576 Rechenoperationen.

Erweiterung des Sedimenttransportes

Es wurde im Verlauf der Testreihen (siehe Kapitel 5.3.1 ab Seite 102) festgestellt, dass sich mit den gegebenen Formeln keine Flussläufe in die Landschaft gegraben haben. Eine Erhöhung der Sediment-Transport-Kapazität-Konstante K_c sollte dazu führen, dass mehr Material mit dem Wasser mittransportiert wird.

Dies führte dazu, dass sich unnatürliche Gesteinsablagerung gebildet

haben. Daher wurde eine neue Formel des Sediment-Transportes gesucht, die eine hohe Sediment-Transport-Kapazität-Konstante K_c verträgt, aber sich nicht unnatürlich auf die Landschaft auswirkt.

Die Idee war es, dass das aufgelöste Material mit dem Wasser aus der Zelle fließt. Hierfür wurde die Summe des Ausfluss $f_{x,y}^{out}$ von der Formel (2.21) genommen und mit dem im Wasser mitgeschwemmten Material $n_{x,y}$ verrechnet. Es wird der Skalierungsfaktor $z_{x,y}$ nach folgender Formel berechnet:

$$z_{x,y} = n_{\Delta t}^{x,y} / f_{x,y}^{out} \tag{3.7}$$

Damit kann exakt die Menge an mitgeschwemmten Material an die angrenzenden Zellen berechnet werden, indem der Skalierungsfaktor $z_{x,y}$ mit dem entsprechenden Ausfluss f verrechnet wird.

$$n_{\Delta t+1}^{x-1,y} = f_{x,y}^{L} * z_{x,y} n_{\Delta t+1}^{x+1,y} = f_{x,y}^{R} * z_{x,y} n_{\Delta t+1}^{x,y-1} = f_{x,y}^{B} * z_{x,y} n_{\Delta t+1}^{x,y+1} = f_{x,y}^{T} * z_{x,y} \tag{3.8}$$

Die Ergebnisse zu diesem Ansatz wird in Kapitel 5.3.1 besprochen.

3.1.4 Erweiterung der Erosion durch stehendes Gewässer

Das Modell der Erosion durch stehende Gewässer muss nur marginal angepasst werden. Das stehende Gewässer dringt in die jeweils oberste Sedimentschicht ein und wandelt diese in Regolith um.

Der in 2.3.2 besprochene Algorithmus ging immer davon aus, dass das

Wasser in Form eines Teichs, Sees oder Meeres steht. Diese Vorgabe ist mit dem vorhergehenden Modell nicht vereinbar, da hier fließendes Gewässer vorausgesetzt wird.

Es muss ein Unterscheidungskriterium gefunden werden, wann Wasser steht oder fließt. Dies kann mit Hilfe des vorangegangenen Modells des fließenden Wassers herausgefunden werden. Hierfür müssen die beiden Parameter f_{out} und f_{in} jeweils null sein, damit das Wasser nicht fließt. Nach einigen Tests wurde herausgefunden, dass die beiden Werte selten genau einen Nullwert annehmen. Das kommt zum einen durch Rechenungenauigkeiten oder durch einen minimalen Wasserfluss. Daher wird ein kleiner Schwellenwert von 0.1 angenommen.

In der Formel (2.20) auf Seite 42 wird der Ausfluss ΔV berechnet, der sich aus dem hinein fließenden Wasser f_{in} und dem hinaus fließenden Wasser f_{out} ergibt. Wenn ΔV einen Ausfluss nahe null angibt, so müssen Wasser f_{in} und f_{out} fast identisch sein.

Daraus lässt sich wiederum ein Struktogramm entwickeln, welches in Abbildung 3.3 zu sehen ist.

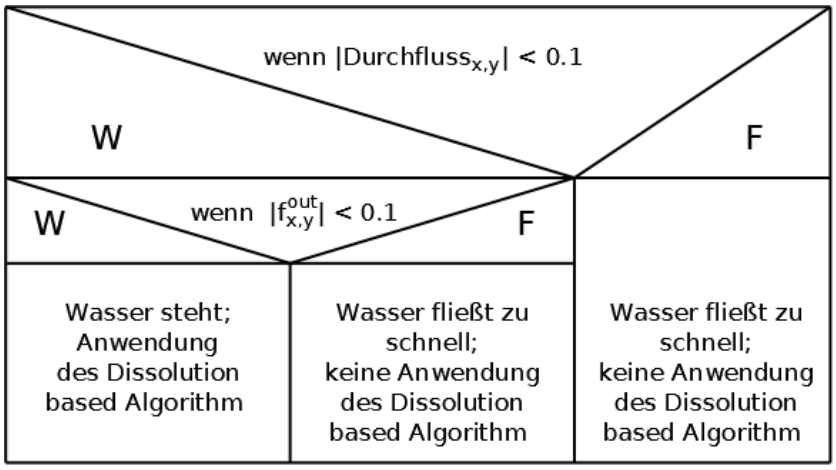

Abbildung 3.3: Struktogramm zur Ermittlung der Wassergeschwindigkeit

Dieser Algorithmus kommt aufgrund dieser Überprüfung nur sehr selten zum Einsatz.

3.1.5 Erweiterung der thermalen Erosion

An den reinen mathematischen Formeln von (2.35) - (2.39) muss keine inhaltliche Veränderung vorgenommen werden. Es wird wie bei diesem Modell üblich die Differenz zu den Nachbarn errechnet, mit dem Talus-Winkel verglichen und gegebenenfalls auf die Nachbarn verteilt. Ein Punkt, der geklärt werden musste, ist, von welcher Gesteinsschicht die Parameter genommen werden.

Wie Abbildung 3.4 aufzeigt, kann es unter Umständen vorkommen, dass eine härtere Sedimentsschicht auf eine weichere fällt.

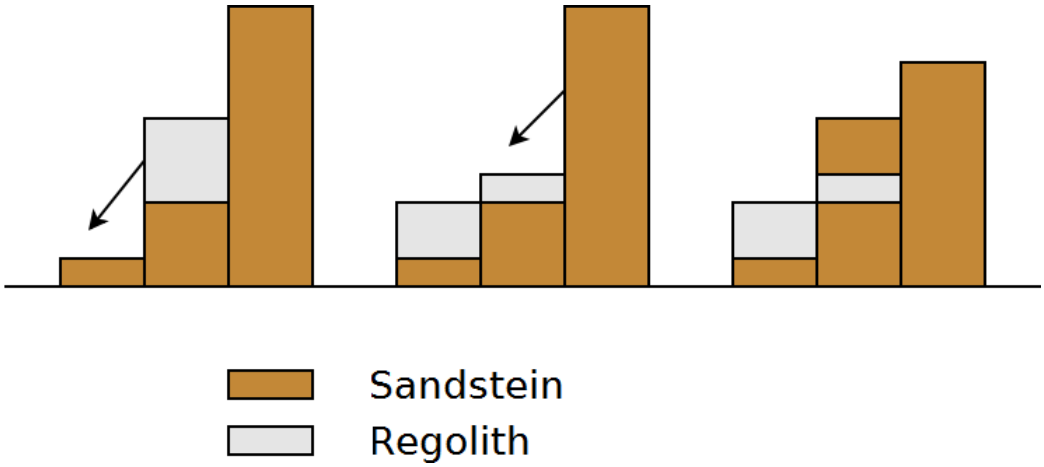

Abbildung 3.4: Abrutschen von härterem auf weicheres Sediment

Wie Abbildung 3.5 zeigt, kann es unter Umständen vorkommen, dass der Talus-Winkel der Regolith-Schicht kleiner als der der darüber liegenden Sandstein-Schicht ist. Nun gibt es drei Fälle, wie es behandelt werden könnte: Im ersten Fall bleibt alles so, wie es ist, da die schwerere Schicht auf die weiche Schicht drückt und somit keine Material mehr in den nächsten Bereich rutschen kann. Im zweiten Fall rieselt das weiche Material aus dem Zwischenraum heraus in den nächsten Bereich. Das geschieht so lange, bis das Material vollkommen transferiert ist. Im letzten Fall rutscht das gesamte Material in die nächste Zelle.

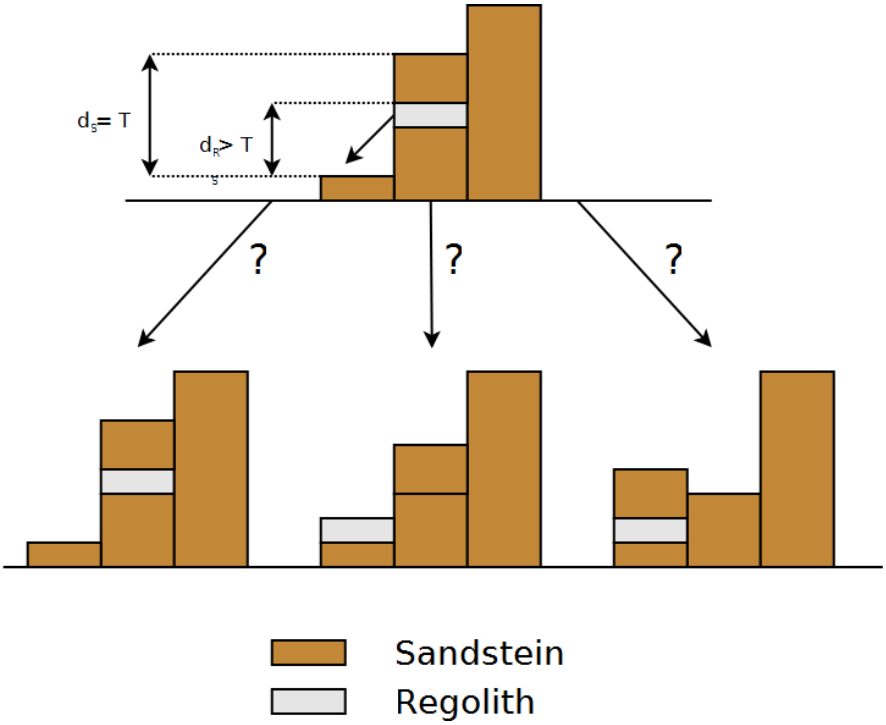

Abbildung 3.5: Problem der thermalen Erosion durch die Einführung des Schichtenmodells

Genau diese drei Fälle sollen abgebildet werden. Der zweite und dritte Fall kann nur unter der Voraussetzung auftreten, dass die darunterliegende Sedimentschicht keine starke Verbindung aufweist. Es müssen also Lockersedimente, im Fall dieser Anwendung, also Regolith und Sand sein, auf welche die obere Schicht abrutschen kann. Das ist schon das erste Unterscheidungsmerkmal zum ersten Fall. Wenn eine darunterliegende Schicht dichter als Sandstein (siehe Abbildung 3.2 auf Seite 66) ist, so wird sie nie auf eine andere Zelle rutschen.

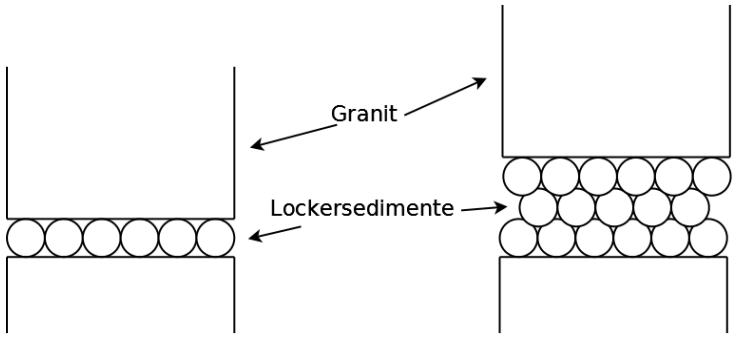

Abbildung 3.6: Abrutschen im Detail

Eine andere Möglichkeit, um die anderen Fälle zu unterscheiden ist die Menge der Lockersedimente. Wenn zu wenig Material vorhanden ist, kann das schwerere Material nicht abrutschen. In Abbildung 3.6 soll das deutlich gemacht werden. Im linken Bild muss eine Kraft von der Seite einwirken, damit die obere Schicht zum Nachbarn transportiert werden kann. Im rechten Bild hingegen kann sich, physikalisch gesehen, durch den Druck der schweren Schicht von oben das komplette Gesteinsmaterial zur tieferen Nachbarzelle bewegen.

Druck p entsteht, indem eine Kraft F auf die Grundfläche A einwirkt [1]. Für die Fläche A gilt hierbei Länge * Breite.

$$p = \frac{|F|}{A} \tag{3.9}$$

[1] http://en.wikipedia.org/wiki/Pressure

Im Fall der fraktalen Landschaft kann es sich bei der Kraft F nur um die Anziehungskraft der Erde handeln, also der Gravitation. Die Gravitationsenergie F ist dadurch gegeben, dass die Masse mit der Gravitation multipliziert wird[2]:

$$F = m * g \tag{3.10}$$

Für die Simulation wurde dann die folgende Formel für eine Schicht abgeleitet:

$$p = \frac{s_{x,y} * m * g}{A} \tag{3.11}$$

Es wurde anschließend bestimmt, dass die obere Gesteinsschicht nur abrutschen kann, wenn der Druck p_{i-1} der unteren Lockersedimentschicht größer ist als der Druck p_i der darüberliegenden Schichten. Da im folgenden Vergleich die Grundfläche A immer identisch ist, kann diese zur Verbesserung der Geschwindigkeit weggelassen werden.

$$p = s_{x,y} * m * g \tag{3.12}$$

Aufgrund dieser Formel wurde folgendes Struktogramm 3.7 entwickelt werden.

Nachdem alle Modelle angepasst wurden, kann theoretisch die Simulation beginnen. Jedoch besitzen die Landschaften mit den Generatoren aus Kapitel 2.2 nur eine einzige Schicht. Daher wurde ein neuer Landschaftsge-

[2]http://de.wikipedia.org/wiki/Gravitation#Gravitation_auf_der_Erde

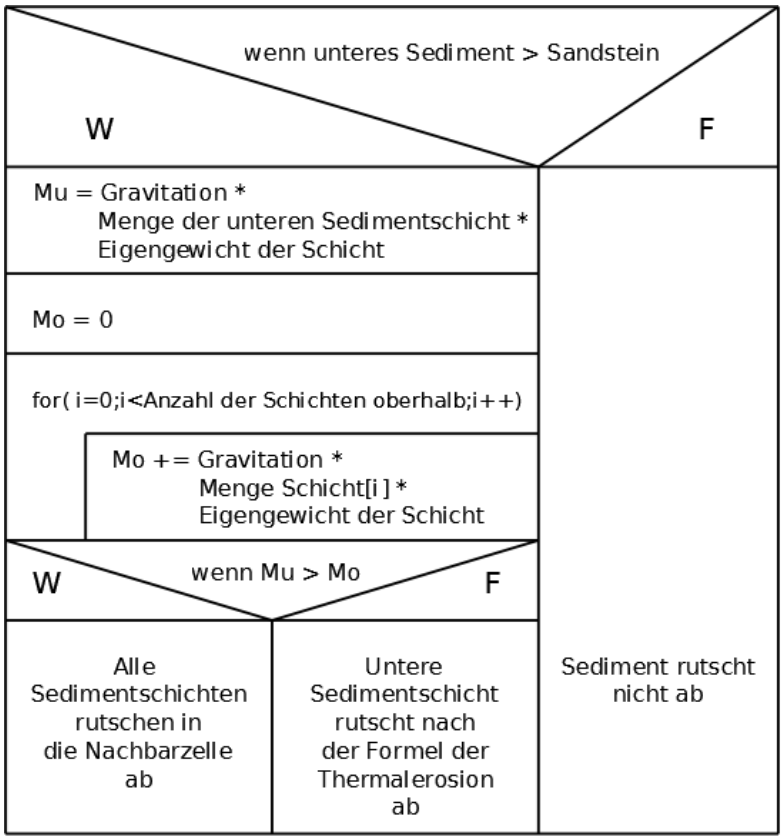

Abbildung 3.7: Stuktogramm für Thermalerosion in Verbindung zum Schichtenmodell

nerator entwickelt, der eine Urlandschaft erzeugt, die schon über ein Schichtenmodell verfügt.

3.2 Multi-Layer-Generator

Nachdem die Modelle für das Programm angepasst wurden, kann theoretisch die Simulation gestartet werden. Da die Urlandschaft schon über einige Schichten verfügen soll, wurde ein neuartiger Generator entwickelt. Bei dem sogenannten Multi-Layer-Generator wird eine solche Landschaft erzeugt.

Der Multi-Layer-Generator basiert darauf, dass Schichten immer zusammenhängend sind. Daher können die Schichten jeweils mit einem Generator, wie dem Midpoint-Displacement oder dem Perlin-Noise-Generator, erzeugt werden. Die einzelnen Schichten werden dabei mit jeweils unterschiedlichen Parametern erzeugt und getrennt gespeichert.

Die unterschiedlichen Parameter dienen dazu, den einzelnen Schichten unterschiedliche Eigenschaften zu geben. Die tiefer liegenden Schichten weisen hierbei nur geringe Höhenunterschiede auf, während die oben liegenden Schichten über stärkere Höhenunterschiede verfügen.

Dafür eignet sich besonders gut der Perlin-Noise-Generator, der sich gut über die Parameter Amplitude und Frequenz beeinflussen lässt. Dies ist mit dem Midpoint-Displacement-Generator nicht so einfach möglich.

In der Abbildung 3.8 ist ein derartiger Aufbau einer Landschaft mit drei Schichten zu sehen. Während Granit über eine hohe Amplitude und eine niedrigen Frequenz die unterste Schicht markiert, so besitzt die höher liegende Sandstein-Schicht zwar eine niedrige Amplitude, aber eine hohe Frequenz.

In den einzelnen Bildern, jeweils als Heightmap dargestellt, sind jedes Mal die unterschiedlichen Höhen der einzelnen Schichten zu sehen. Die auf-

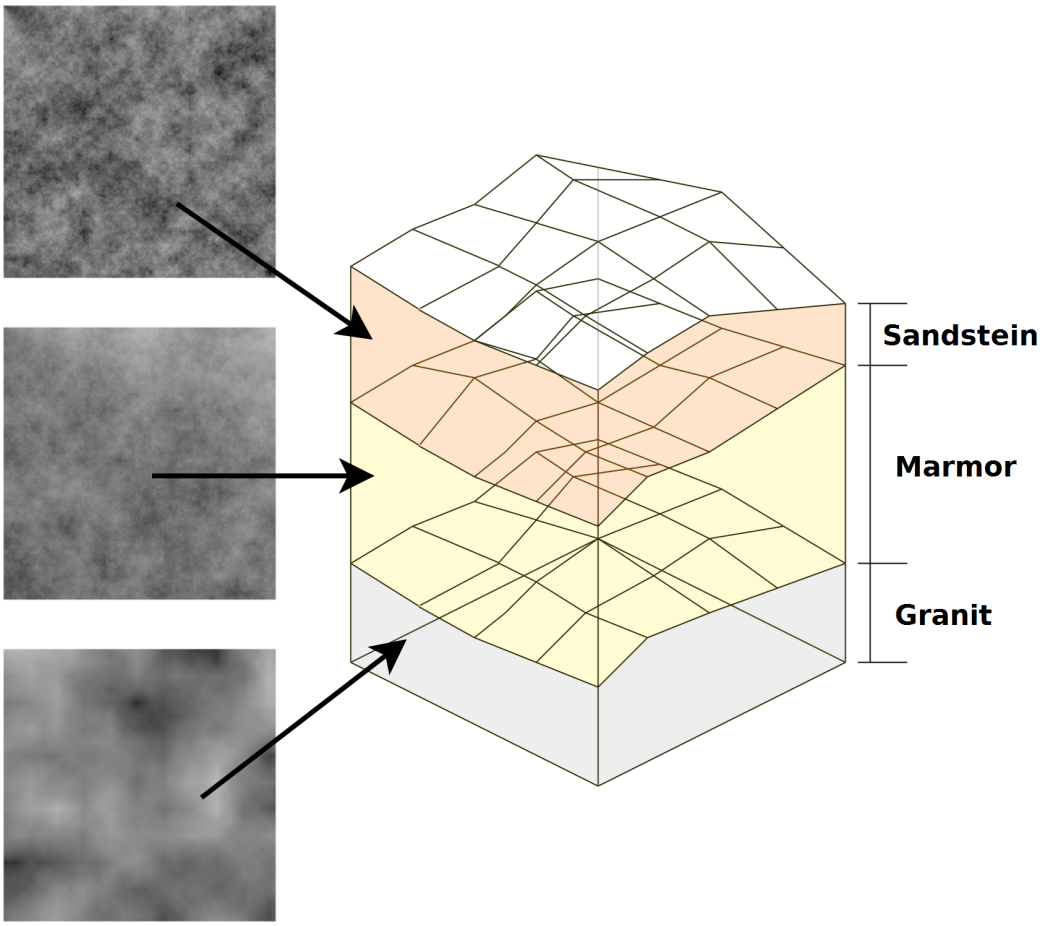

Abbildung 3.8: Multi-Layer-Generator

summierte Höhe der einzelnen Schichten ergibt die Gesamthöhe dieser Landschaft. Sie ist in der Abbildung 3.9 zu sehen.

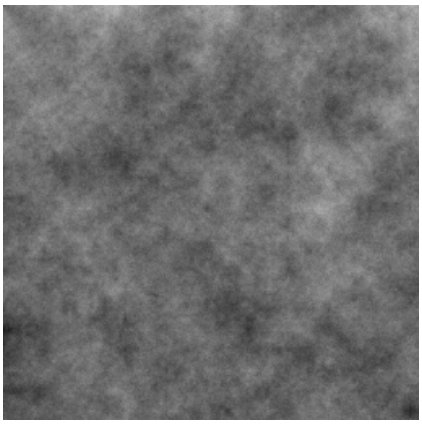

Abbildung 3.9: Die aus dem Bild 3.8 erzeugte Landschaft

Dieser neue Ansatz ermöglicht eine schnelle Generierung einer Urlandschaft, die schon über unterschiedliche Schichten verfügt.

Anhand der Ausarbeitung dieses Konzeptes wurde die Implementierung in C# vorgenommen, die im nächsten Kapitel besprochen wird.

There is a joke that your hammer will always find nails to hit. I find that perfectly acceptable.

Benoît Mandelbrot

Kapitel 4

Implementierung

In diesem Kapitel wird die Implementierung der aus dem vorangegangenen Kapitel bekannten Landschaftsgeneratoren sowie den Erosions- und Alterungsprozessen beschrieben. Das komplette Projekt ist in C# geschrieben, da sich die Integration in das an der Hochschule Darmstadt neu entwickelte Framework GLAB anbot.

4.1 Entwicklung der Klassenstruktur

Das für die Hochschule entwickelte Framework GLAB soll für die Praktikumsaufgaben von verschiedenen grafischen Vorlesungen verwendet werden. Dazu gehört auch die Veranstaltung Chaos und Fraktale zu dessen Teilgebiet auch die Erzeugung fraktaler Landschaften gehört. Daher können und sollen die verschiedenen Klassen und Methoden so abstrakt wie möglich gehalten werden, damit sie gegebenenfalls auch wiederverwendet werden können.

Aus diesem Grund wurde das Projekt in zwei Projektteile gegliedert: Zum einen in GLab.Terrain, das alle in dieser Arbeit vorgestellten Generatoren und Erosionsprozesse enthält, zum anderen Example.Terrain, welches

die grafische Anwendung beinhaltet. Einen ersten Überblick über die Klassenstruktur zeigt Bild 4.1.

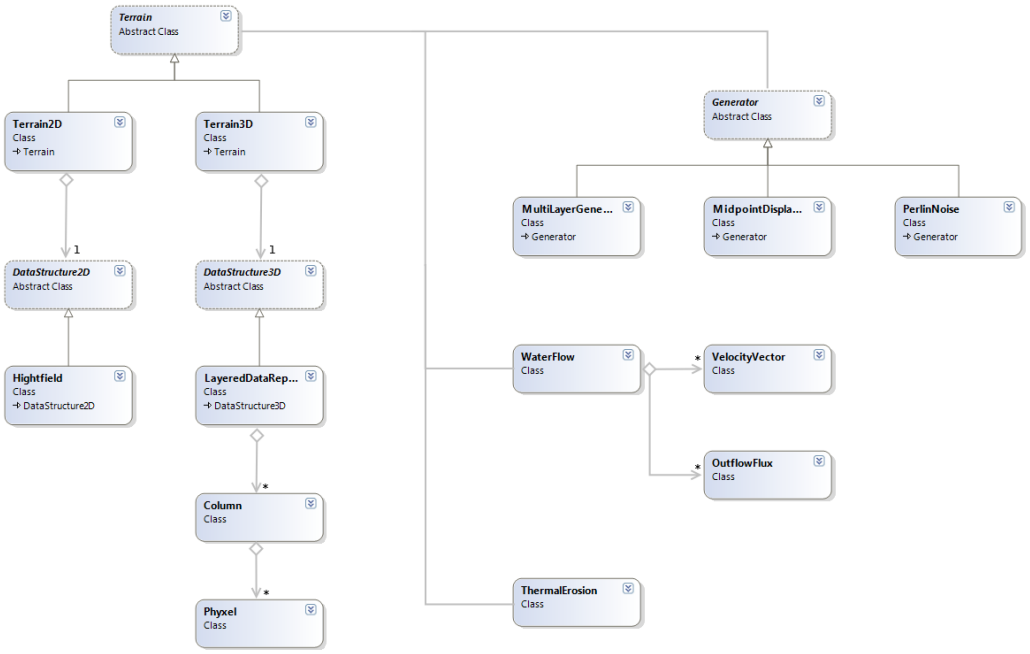

Abbildung 4.1: Klassendiagramm der kompletten Anwendung

Diese Klassenstruktur wird nun einzeln erklärt.

Der erste Teil, der erläutert wird, ist die Speicherung und die Verwaltung des Geländes. Dafür wurde die abstrakte Klasse *Terrain* erzeugt. Sie besitzt zum einen sämtliche globalen Parameter, wie Gravitationskonstante g, Sediment-Transport-Kapazität-Konstante K_c oder die Verdunstungskonstante K_V. Des Weiteren werden die für die Verwaltung wichtigen Methoden zur Verfügung gestellt. Die können aus der Abbildung 4.2 entnommen werden.

Davon können zwei Klassen abgeleitet werden, *Terrain2D* und *Terrain3D*, die intern mit unterschiedlichen Datenstrukturen arbeiten.

Terrain2D arbeitet mit einer abstrakten Klasse *Datastructure2D* und *Terrain3D* mit einer abstrakten Klasse *Datastructure3D*. Die Idee war, dass

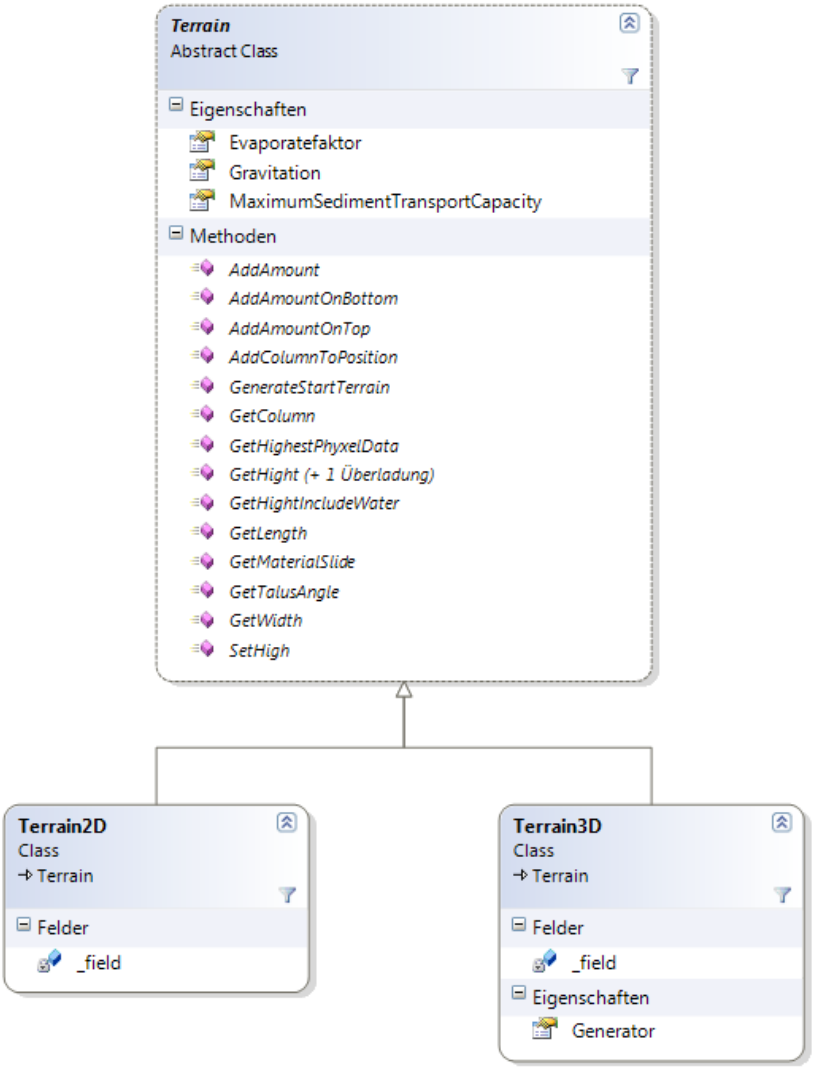

Abbildung 4.2: Klassendiagramm der Terrain-Klassen

die 2D-Datenstruktur aus einer 2D-Matrix mit den Höhenwerten besteht, während die Landschaft mit der 3D-Datenstruktur aus einer 2D-Matrix besteht, welche die Schichten als dritte Dimension speichert.

Für die abstrakte Klasse *Datastructure2D* wurde im speziellen die Klasse *Heightmap* erzeugt, welche genau die Eigenschaften der Heightmap aus Kapitel 2.1.1 enthält. Das Klassendiagramm ist in Abbildung 4.3 zu sehen. Die Oberklasse stellt nur die entsprechenden Methoden zur Verfügung, damit Terrain2D damit arbeiten kann. Jede abgeleitete Klasse, in diesem Fall Heightfield, kann über eine eigene gewünschte Datenstruktur verfügen. Somit sind auch für besonders große Landschaften effektive Zugriffsmethoden möglich.

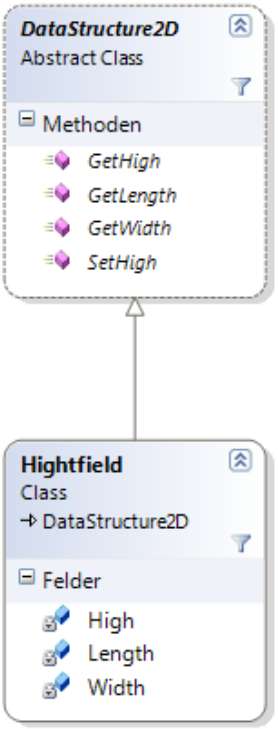

Abbildung 4.3: Klassendiagramm für die 2D-Datenstruktur

Die Klasse *Datastructure3D* ist ähnlich aufgebaut wie *Datastructure2D*. Sie enthält die Methoden, welche die abgeleitete Klasse implementieren muss, damit sie von der Klasse Terrain3D verwendet werden kann. Auch hier ist es

möglich, eigene abgeleitete Klassen, wie zum Beispiel eine rein voxelbasierte Klasse, zu entwickeln.

Für die Anwendung wurde die aus Kapitel 2.1.3 bekannte Layered Data Representation in der Klasse *LayeredDataRepresentation* implementiert. Als Datenstruktur wird auch hier eine 2D-Matrix verwendet, allerdings mit der Klasse *Column* anstatt den Höhenwerten. Die *Column* Klasse besitzt eine verkettete Liste von *Phyxel*. Jedes Phyxel repräsentiert eine eigene Schicht mit den entsprechenden physikalischen Eigenschaften. Die Klassenansicht dazu ist in 4.4 abgebildet.

Die Klasse LayeredDataRepresentaion übernimmt auch die Verwaltung des in Kapitel 3.1.2 vorgestellten Gesteinskreislaufes.

Das Klassendiagramm für die im Kapitel 2.2 bzw. 3.2 erklärten Generatoren wird in Abbildung 4.5 dargestellt. Es wurde die abstrakte Klasse Generator eingeführt, welche für alle abgeleiteten Klassen die beiden Methoden *GenerateTerrain(Datastructure2D)* und *GenerateTerrain(Datastructure3D)* zur Verfügung stellt. Die beiden Methoden werden von den abgeleiteten Klassen mit ihrem Programmcode ausgeführt.

Der Grund für die überladene Methode *GenerateTerrain* ist der, wie schon im Kapitel 2.2 besprochen, dass die Generatoren nur bestimmte Ergebnisse liefern. Bei dem Midpoint-Displacement-Generator und beim Perlin-Noise-Generator werden nur Höhenwerte erzeugt, wodurch sie sich nur für eine 2D-Datenstruktur eignen, während der Multi-Layer-Generator für eine 3D-Datenstruktur ausgelegt ist.

Bei jedem Generator wurden beide Methoden implementiert. Bei dem Midpoint-Displacement-Generator und beim Perlin-Noise-Generator wurde für die Implementierung der 3D Landschaft nur eine einzige Schicht erzeugt. Bei dem Multi-Layer-Generator wird eine mehrschichtige Landschaft erzeugt und anschließend nur die Gesamthöhe in die Klasse *Datastructure2D* geschrieben.

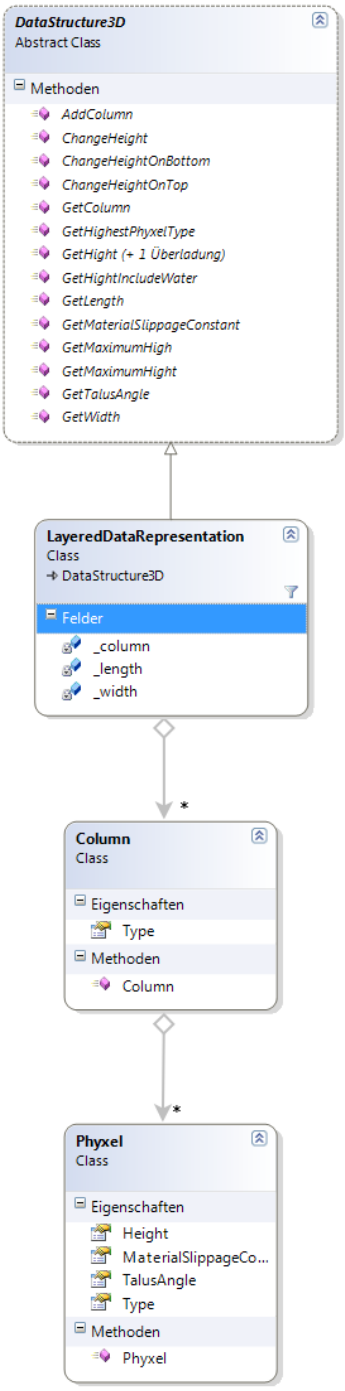

Abbildung 4.4: Klassendiagramm für die 3D-Datenstruktur

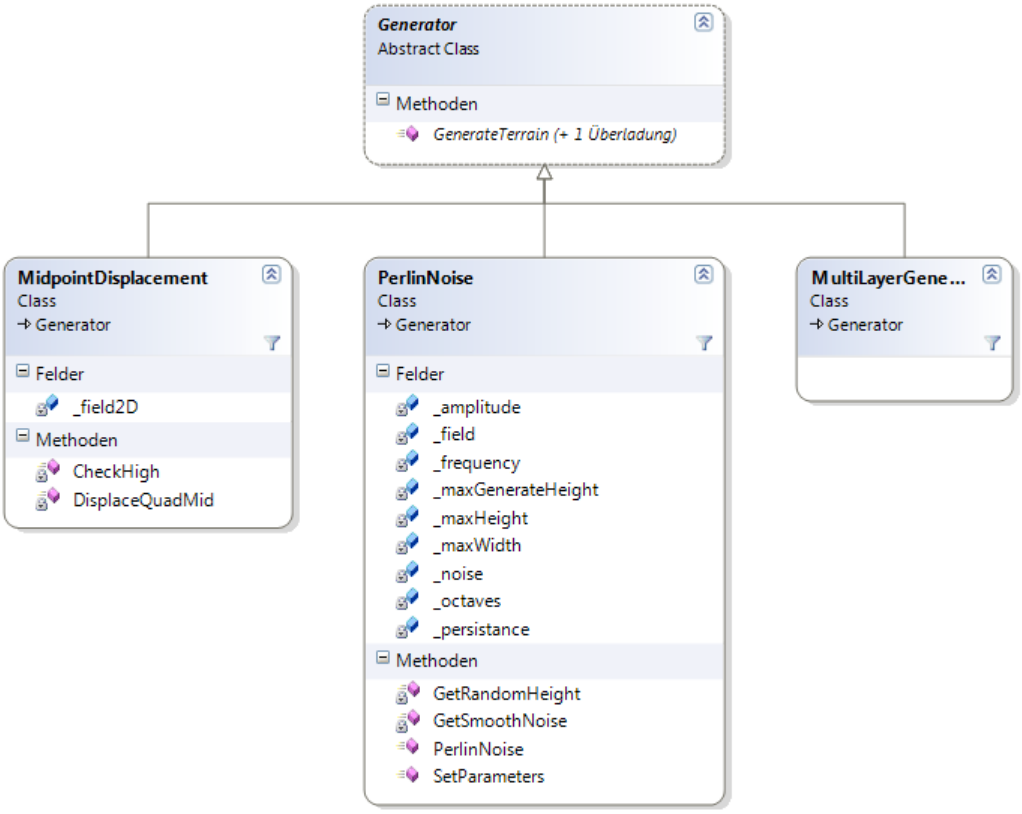

Abbildung 4.5: Klassendiagramm für die Generatoren

Die eigentliche Simulation der Erosions- und Alterungsprozesse wird in zwei Klassen realisiert. Zum einen in der Klasse *WaterFlow*, welche den gesamten Wasserfluss berechnet, zum anderen in der Klasse *ThermalErosion*, die die thermale Erosion berechnet.

Die Klasse *Waterflow* (siehe Abbildung 4.6) simuliert sowohl das Wasser das fließende als auch das stehende Gewässer, welche in den beiden Kapiteln 2.3.1 und 2.3.2 erklärt ist. Wie im Kapitel 3.1.4 erarbeitet wurde, benötigen die Formeln für das stehende Gewässer Elemente von den Formeln des fließenden Gewässers. Um Berechnungsaufwand zu sparen, wurden sie in einer Klasse vereint.

Um die Klasse zu verwenden, müssen zuerst die die Parameter für das Wasser gesetzt werden. Dies ist zum einen die Position einer Wasserquelle in der x/y Position (*SourceOfSpring*) und die Menge an Wasser, die diese Quelle in jedem Iterationsschritt hervorbringt (*QuantityOfSpring*). Des Weiteren kann die Stärke des Regens eingegeben werden (*StrengthRain*), die angibt, an wie vielen zufälligen x/y Positionen eine gewisse Menge Wasser (*QuantityOfRain*) addiert wird.

Mit der Methode *DoWaterflow* wird eine Iteration der Wassersimulation ausgeführt.

Die letzte Erosionsart, die aus Kapitel 2.3.3 bzw. 3.1.5 beschriebene thermale Erosion, wird in der Klasse ThermalErosion implementiert. Der Aufruf der Methode Erode berechnet die komplette thermale Erosion. Sie wird in Abbildung 4.7 dargestellt.

4.2 Anwendung

Für die Simulation kann nun auf die eben beschriebenen Klassen zugegriffen werden. Um den Ablauf, wie er in der Abbildung 2.1 auf Seite 11 dargestellt ist, zu realisieren, wird der Ablauf der Erosionsprozesse als Sequenzdiagramm

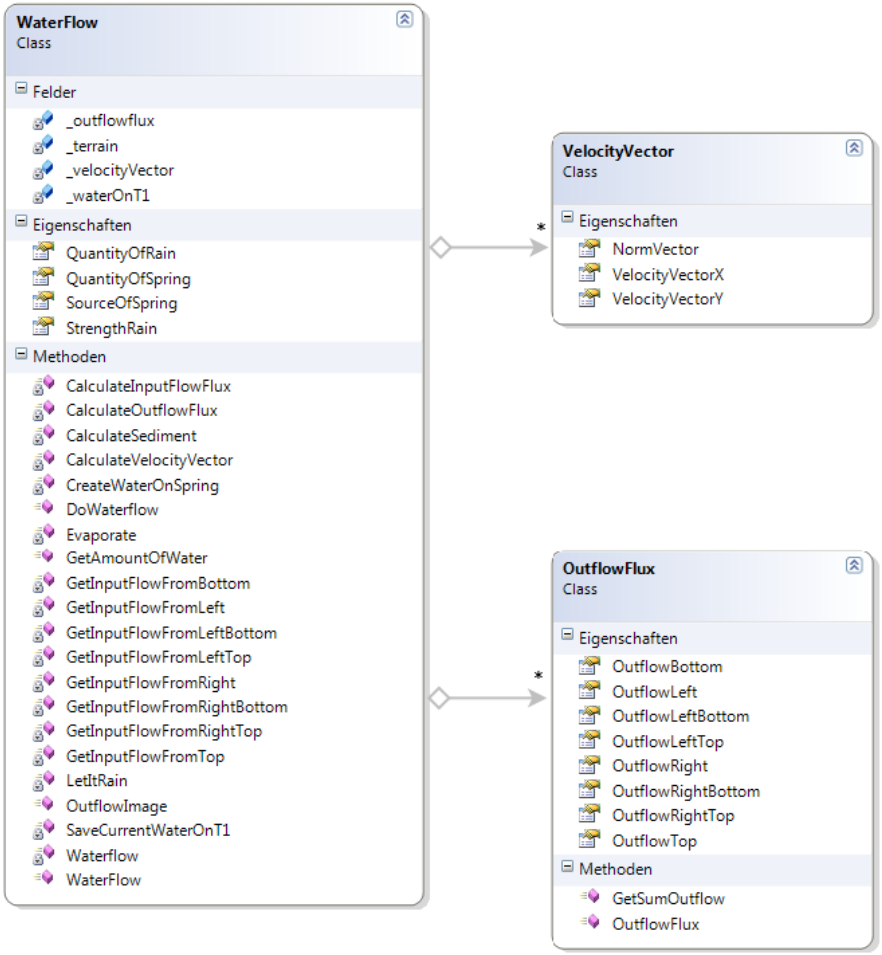

Abbildung 4.6: Klassendiagramm für den Wasserfluss

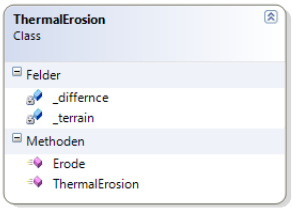

Abbildung 4.7: Klassendiagramm der Thermal Erosion

dargestellt (zu sehen in Abbildung 4.8).

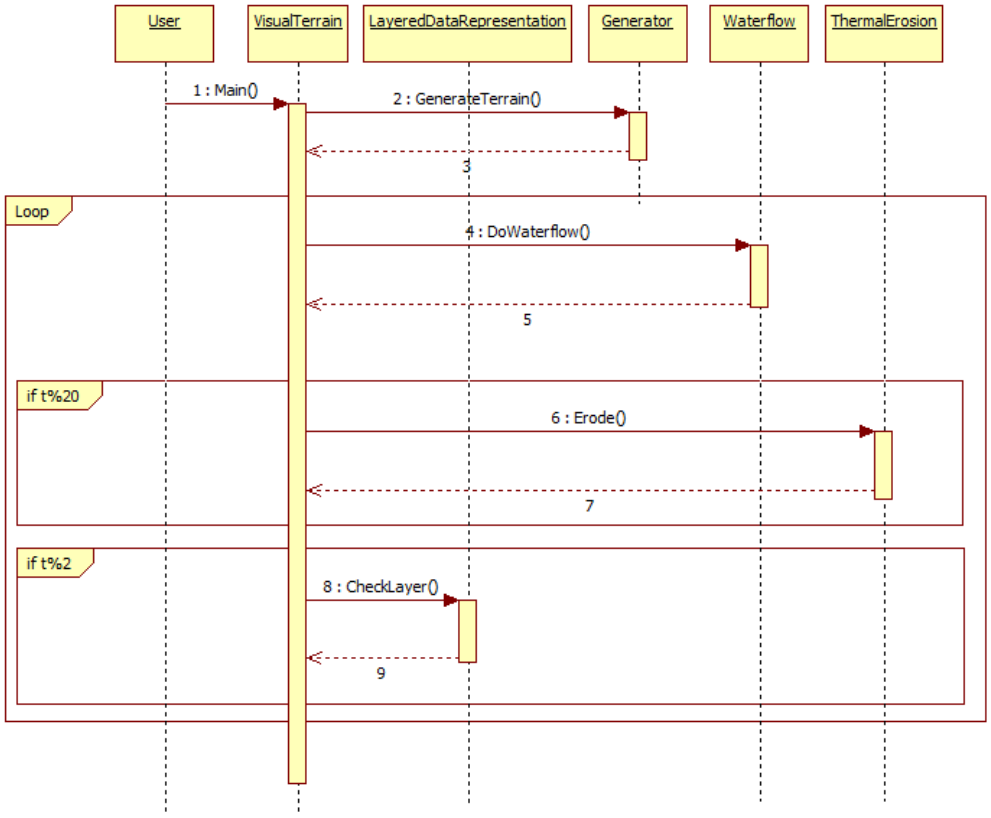

Abbildung 4.8: Sequenzdiagramm

Die Klasse *VisualTerrain* stellt die Main-Methode zur Verfügung. In ihr wird zuerst die Datenstruktur für die Landschaft erzeugt, anschließend die Parameter für die einzelnen Erosionsklassen und noch eine Zeitvariable t auf null gesetzt. Diese sind zwecks Übersicht nicht im Sequenzdiagramm zu sehen. Danach wird durch den Aufruf *GenerateTerrain* einer speziellen Generatorklasse die Urlandschaft erzeugt.

Nun werden in einer Schleife die einzelnen Erosionsklassen nacheinander aufgerufen. Die Klasse *Waterflow* wird in jedem Iterationsschritt aufgerufen, da sich das Wasser permanent bewegt. Die Methode *Erode* der Klasse *ThermalErosion* wird nur jeden zwanzigsten Iterationsschritt aufgerufen. Die Veränderung der Gesteinsschicht wird in jedem zweiten Schritt aufgerufen wird,

da dass, wie in Kapitel 3.1.2 bearbeitet wurde, die kleinste Zeiteinheit ist, in der sich eine Schicht zu einer anderen Schicht vereinen kann.

Im folgenden Kapitel wird nun die Implementierung getestet und bewertet.

For much of my life there was no place where the things I wanted to investigate were of interest to anyone.

Benoît Mandelbrot

Kapitel 5

Ergebnis

In diesem Kapitel werden die Ergebnisse der Arbeit besprochen. Zuerst werden einige von den Generatoren erzeugte Landschaften gezeigt. Anschließend werden die Zeiten ermittelt, welche die Generatoren benötigen, um eine Landschaft zu generieren. Anschließend wird die Stärke der Erosionsprozesse ermittelt und dazu die Berechnungszeit. Damit soll die Effizienz der Erosionsprozesse bewertet werden.

5.1 Varianten der Generator Parameter

In diesem Kapitel werden einige erzeugte Landschaften dargestellt. Die Abbildung 5.1 zeigt eine Landschaft, die mit dem Midpoint-Displacement-Generator erzeugt wurde. Auf die Landschaft selbst kann nur wenig Einfluss genommen werden. Parameter dafür wären die Wahrscheinlichkeit, dass sich der Boden hebt oder senkt, oder die allgemeine Stärke der Höhe.

Diese Parameter müssen gut aufeinander abgestimmt sein, sonst bilden sich diagonale oder vertikale Linien. Beispiele dafür sind in Abbildung 5.2 zu sehen.

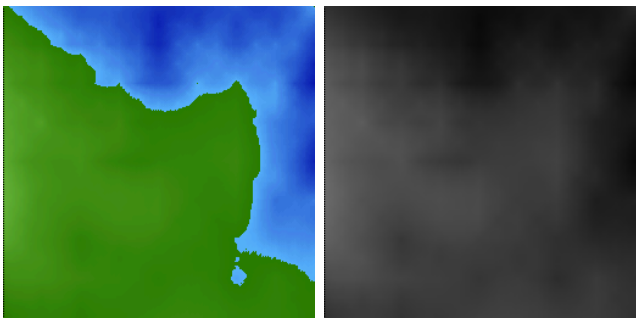

Abbildung 5.1: Midpoint-Displacement-Generator

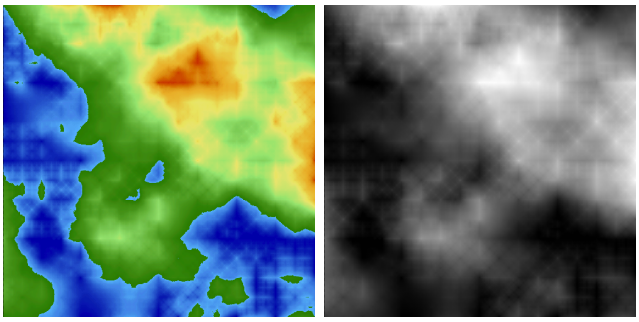

Abbildung 5.2: Beispiel eines schlecht abgestimmten Midpoint-Displacement-Generators

Bei den nächsten Beispielen handelt es sich um Landschaften, die mit Perlin-Noise erzeugt wurden. Der Generator lässt sich mit den Parametern Frequenz, Amplitude und Persistenz verändern. Die ersten drei Bilderreihen 5.3 - 5.5 zeigen eine Landschaft mit jeweils unterschiedlicher Frequenz. Bei zu niedriger Frequenz sind die erzeugten Landschaften flach, bei zu hoher Frequenz ist es keine natürliche Landschaft mehr. Für die folgenden Bilder liegt die Amplitude bei 0.75 und die Persistenz bei 0.55.

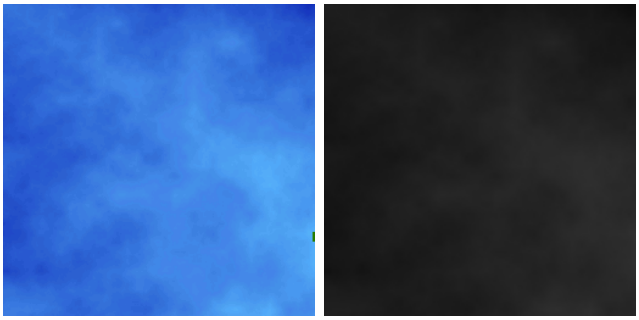

Abbildung 5.3: Perlin-Noise mit Frequenz 0.001

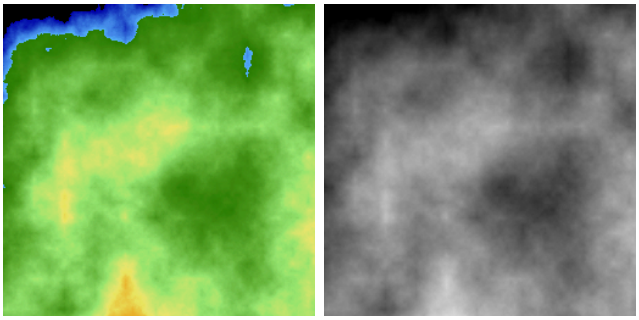

Abbildung 5.4: Perlin-Noise mit Frequenz 0.01

Bei den nächsten drei Bildern 5.6 - 5.8 ändert sich nur die Amplitude, während die Frequenz bei 0.325 und Persistenz bei 0.55 liegt. Auch hier zeigt sich bei zu niedrigen Amplituden eine zu flache, bei zu hohen Amplituden eine zu hügelige Landschaft.

Die nächsten drei Bilder 5.9 - 5.11 zeigen die Veränderung der Landschaft bei einer unterschiedlichen Persistenz. Die Frequenz liegt bei 0.325 und die Amplitude bei 0.75.

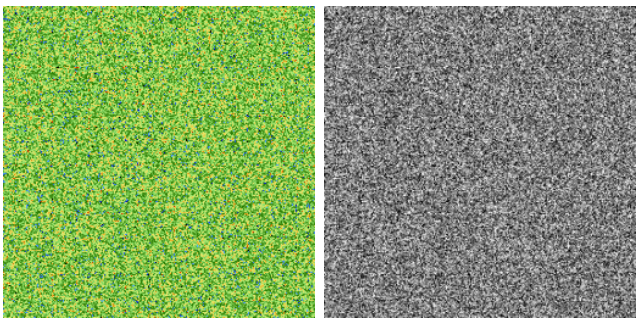

Abbildung 5.5: Perlin-Noise mit Frequenz 0.1

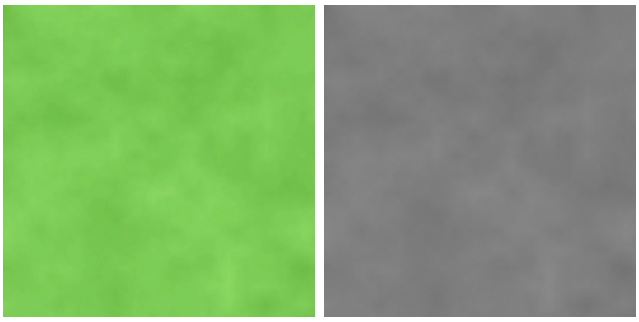

Abbildung 5.6: Perlin-Noise mit Amplitude 0.05

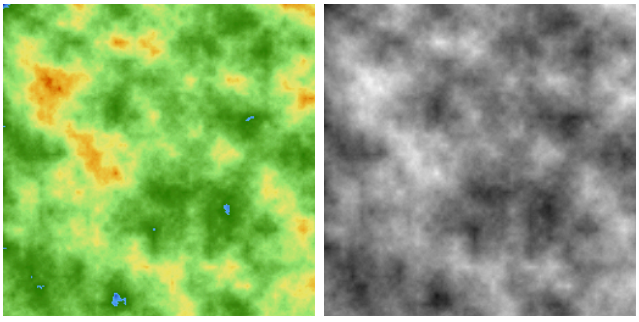

Abbildung 5.7: Perlin-Noise mit Amplitude 0.55

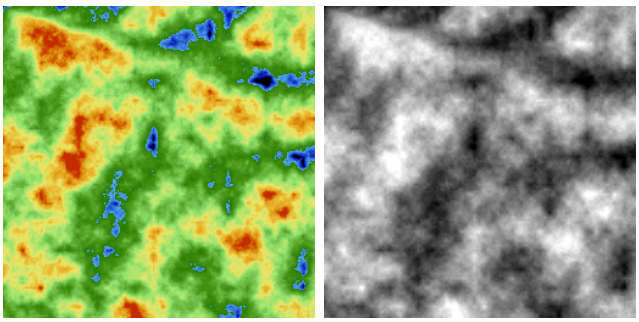

Abbildung 5.8: Perlin-Noise mit Amplitude 0.95

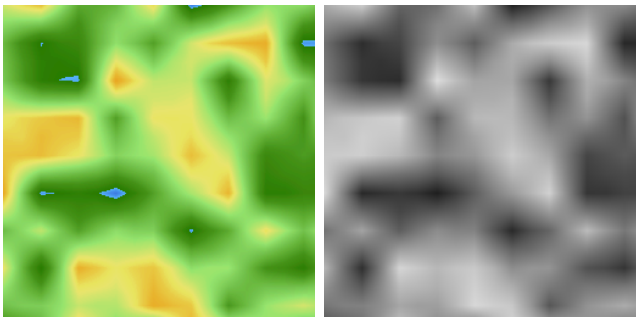

Abbildung 5.9: Perlin-Noise mit Persistenz 0.05

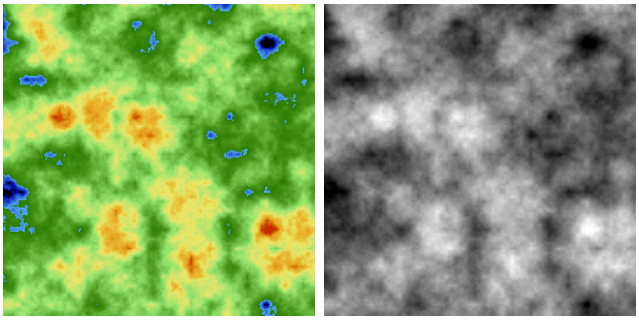

Abbildung 5.10: Perlin-Noise mit Persistenz 0.55

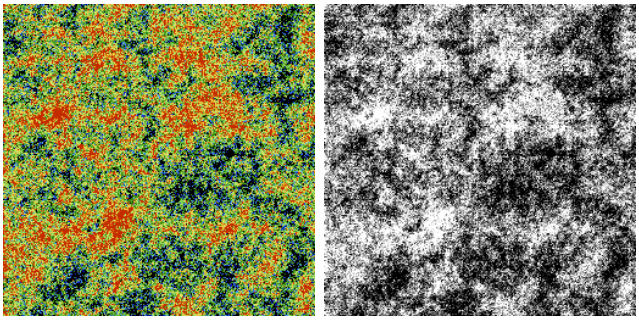

Abbildung 5.11: Perlin-Noise mit Persistenz 0.95

Zu guter Letzt wird eine Landschaft mit der Verwendung des Multi-Layer-Generators in Bild 5.12 gezeigt. Die einzelnen Schichten werden dabei von dem Perlin-Noise-Generator erzeugt, da sich die Schichten individueller beeinflussen lassen. Die erzeugte Landschaft besteht aus vier Schichten: einer Granit-Schicht, einer Marmor-Schicht, einer Schicht aus Kalk und einer aus Sandstein. Die Granit-Schicht entspricht den Parametern von Abbildung 5.9, die Marmor Schicht der von Abbildung 5.4 und die Abbildung 5.8 der Kalk und auch der Sandstein Schicht.

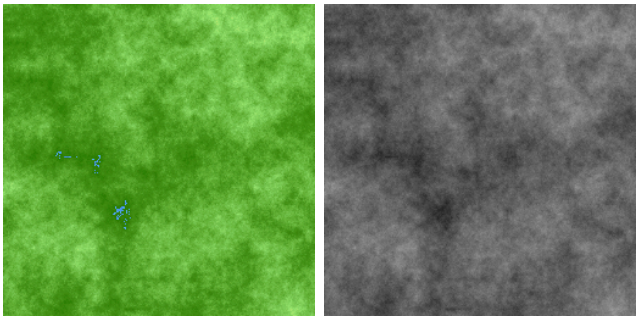

Abbildung 5.12: Multi-Layer-Generator

Diese einzelnen Generatoren wurden anschließend einigen Laufzeittests unterzogen.

5.2 Berechnungszeit der Generatoren

Für die Berechnungszeit wurden zwei verschiedene Computersysteme verwendet:

1. AMD Athlon 2.1GHz, 32bit, 1.5GB RAM, Windows XP
2. AMD Athlon X2 3.0GHz, 64bit, 2GB RAM, Windows 7

Es wurden mehrere Testreihen durchgeführt. In jeder Testreihe wurde die Zeit berechnet wie lange ein Generator zum Erzeugen einer Landschaft benötigt. Es wurden 1000 Wiederholungen durchgeführt und davon der Durchschnitt errechnet.

Die Ergebnisse werden in der folgenden Tabelle 5.1 aufgeführt:

Landschaftsgröße	Rechner	Midpoint Disp.	Perlin-Noise	Multi-layer
256x256	1	434ms	432ms	2100ms
256x256	2	313ms	305ms	1643ms
512x512	1	1010ms	1652ms	7779ms
512x512	2	775ms	1201ms	5163ms
1024x1024	1	4232ms	9467ms	33532ms
1024x1024	2	2863ms	3298ms	22532ms
2048x2048	1	23791ms	32428ms	216941ms
2048x2048	2	15932ms	21532ms	144983ms
4096x4096	1	87728ms	131510ms	-
4096x4096	2	58203ms	87903ms	-

Tabelle 5.1: Berechnungszeit der Generatoren

Diese Ergebnisse dieser Tabelle werden in Abbildung 5.13 grafisch dargestellt.

An diesen Werten ist zu sehen, dass sich der Aufwand zum Berechnen der Landschaften bei dem Midpoint-Displacement-Generator also auch bei

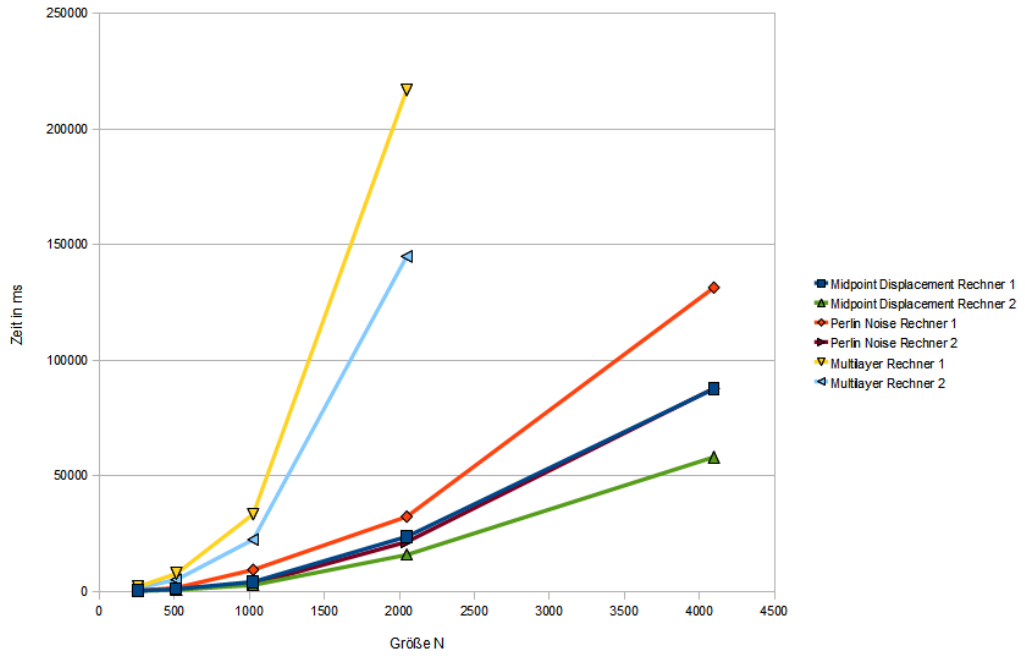

Abbildung 5.13: Diagramm der Generator Testwerte

Perlin-Noise-Generator jeweils um den Faktor vier erhöht, wenn die Seitenlänge N verdoppelt wird. Dies ist entspricht auch den Erwartungen, da sich die Fläche der Landschaft bei einer Verdoppelung vervierfacht.

Ein wenig anderes sieht das Ergebnis bei dem Multi-Layer-Generator aus. Wie schon am Anfang erwähnt, werden insgesamt drei Schichten mit Hilfe des Perlin-Noise-Generators erzeugt. Also der dreifache Aufwand gegenüber des Perlin-Noise-Generators. In Wirklichkeit beträgt der Aufwand bei N=256 etwa das Fünffache bin zum Siebenfache bei N=2048. Ursache dafür ist die vielfache Erzeugung von Objekten der Klasse *Column* und der Klasse *Phyxel*.

Das ist auch die Ursache, dass keine Ergebnisse bei N = 4096 möglich waren. Die verwendete Datenstruktur der Column, eine von C# verwaltete verkettet Liste, war nicht mehr in der Lage die über 50 Millionen (4096*4096*3 Schichten) Phyxel zu verwalten und brachte einen Speicher Zugrifffehler.

Bei der Testreihe mit Perlin-Noise wurde eine weitere Überprüfung vorgenommen, inwieweit sich die Laufzeit durch Veränderung der Parameter verhält. Dabei wurde einige Parameter zufällig bestimmt und dazu die Zeit gemessen.

Rechner	Frequenz	Amplitude	Persistenz	Berechnungszeit
1	0.001	0.9	0.9	9076ms
2	0.001	0.9	0.9	6070ms
1	0.1	0.1	0.1	8261ms
2	0.1	0.1	0.1	5683ms
1	0.1	0.9	0.9	7475ms
2	0.1	0.9	0.9	5023ms
1	0.0001	0.0001	0.0001	9386ms
2	0.0001	0.0001	0.0001	6473ms

Tabelle 5.2: Berechnungszeit unterschiedlicher Perlin-Noise Parameter bei N=512

Diese Ergebnisse dieser Tabelle werden in Abbildung 5.14 grafisch dargestellt.

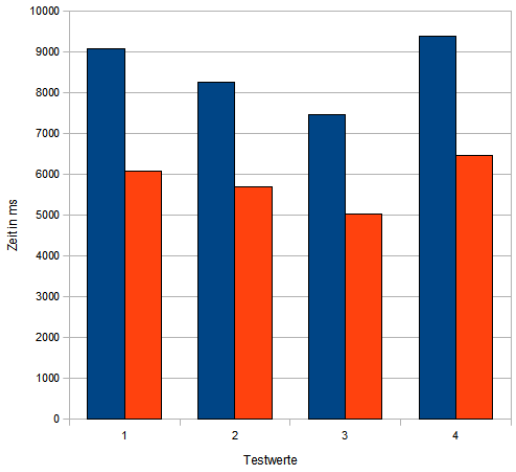

Abbildung 5.14: Diagramm der unterschiedlichen Parlin Noise Parameter

An diesen Werten ist zu sehen, dass es eine leichte Abhängigkeit zwischen der Berechnungszeit und den Parametern gibt.

5.3 Stärke der Erosionsprozesse

Damit eine Aussage getroffen werden kann, wie stark sich ein Erosionsmodell auf die Landschaft ausgewirkt hat, wurde eine Erosions-Score-Formel entworfen [Ols04].

Zuerst wird eine 2D-Matrix mit den maximalen Höhenunterschieden (eng. Slope map) zu den N4-Nachbarn erzeugt. Diese Matrix hat die gleichen Maße der Heightmap, also N^2. Die Höhe h an der Stelle x/y ist hierbei definiert als $h_{x,y}$. Der maximale Höhenunterschied $d_{max}^{x,y}$ an der Stelle x/y definiert wie folgt:

$$d_{max}^{x,y} = max(\,|h_{x,y} - h_{x-1,j}|\,, \quad |h_{x,y} - h_{x+1,j}|\,, |h_{x,y} - h_{x,j-1}|\,, \quad |h_{x,y} - h_{x,j+1}|) \tag{5.1}$$

Alle Werte d_{max} aus dieser 2D-Matrix werden aufsummiert und anschließend durch die Anzahl der Elemente N^2 geteilt. Das Ergebnis $\overline{d}$ gibt den Erwartungswert des maximalen Höhenunterschieds an:

$$\overline{d} = \frac{1}{N^2} \sum_{x=1}^{N-2} \sum_{y=1}^{N-2} d_{max}^{x,y} \tag{5.2}$$

Anschließend wird die Varianz σ_d zu dem Erwartungswert $\overline{s}$ berechnet:

$$\sigma_d = \sqrt{\frac{1}{N^2} \sum_{x=1}^{N-2} \sum_{y=1}^{N-2} (d_{max}^{x,y} - \overline{d})^2} \tag{5.3}$$

Mit der Varianz σ_d kann eine Aussage über die Beschaffenheit der Landschaft getroffen werden. Wenn die Varianz σ_d klein ist, so weißt die Landschaft nur wenige große Unterschiede der Höhe auf; das heißt es gibt nur

wenige Hügel und Berge. Der Fall ist umgekehrt, wenn die Varianz σ_d sehr groß ist. Es gibt mehr Hügel und Berge. Den Score, den diese Landschaft nun besitzt, wird errechnet indem σ_d durch $\overline{d}$ geteilt wird.

$$\epsilon = \frac{\sigma_d}{\overline{d}} \tag{5.4}$$

Dieser Score gibt nun an, wie stark die Veränderung der Landschaft durch einen Erosionsprozess ist. Dazu werden zwei Score-Werte vor und nach dem Erosionsprozesss verglichen.

Die Testreihe in den nächsten Kapiteln wird anhand drei unterschiedlichen Landschaftstypen ermittelt. Die Typen werden mit dem Perlin-Noise-Generator erzeugt die folgende Parameter besitzt.

1. Frequenz 0.0325, Amplitude 0.05, Persistenz 0.55
2. Frequenz 0.0325, Amplitude 0.55, Persistenz 0.55
3. Frequenz 0.1, Amplitude 0.55, Persistenz 0.55

Diese drei Landschaften entsprechen drei möglichen Landschaften: Flache, ein wenig zerklüftet und eine mit (unnatürlichen) Gebirgszügen.

Die für die Testreihe verwendeten Landschaften werden in der Abbildung 5.15 gezeigt.

Die Landschaften bestehen aus jeweils 60% Granit, 20% Marmor, 15% Kalk und 5% Sandstein. Es wird für den Test 100 Iterationen durchgeführt und von jeden zehnten Schritt den Score ermittelt.

Interessant sind dafür nur der Wasserfluss und die thermale Erosion, den durch die Alterung von Gestein ändert sich nichts außer der Zuordnung der

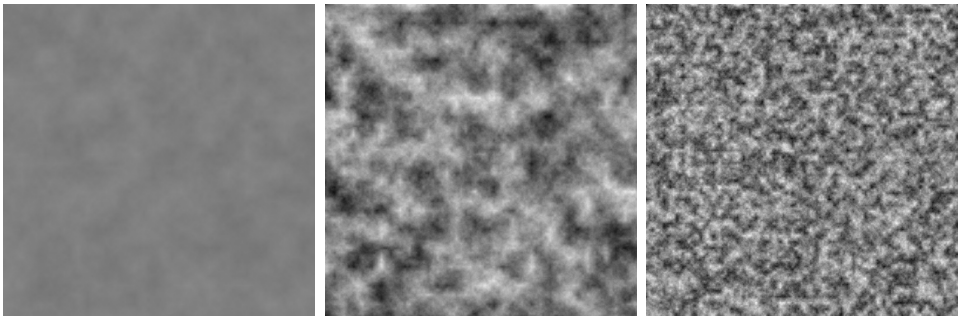

Abbildung 5.15: Testlandschaften 1-3 (von links nach rechts) mit N = 512

einzelnen Schichten. Daher ergeben sich keine Veränderungen an der Struktur der Landschaft selbst.

5.3.1 Erosionswirkung des Wasserflusses

Bei der Erosionswirkung des Wasserflusses wurden einige interessante, aber schlussendlich für die eigentliche Anwendung negative Ergebnisse herausgefunden.

Test mit den originalen Formeln

Die erste Testreihe wurde mit einer Wasserquelle, die pro Iteration 20 Einheiten Wasser liefert, eine Regenstärke von 10 (also an zehn Stellen wird zufällig Wasser an einer Position erzeugt), die jeweils mit 20 Einheiten Wasser gefüllt werden. Als maximale globale Sediment-Transport-Kapazität-Konstante K_c wurde 0.5 angenommen. Die Gravitation g liegt bei 0.9 und der Verdunstungsfaktor K_V bei 0.01. Die restlichen Werte können der Tabelle 3.1 auf Seite 64 entnommen werden.

Das Ergebnis dieses Testlaufes ist den beiden folgenden Tabellen zu entnehmen.

Landschaft	Anfangsscore	10	20	30	40	50
1	0.0245	0.0245	0.0245	0.0245	0.0245	0.0245
2	0.2525	0.2525	0.2525	0.2524	0.2524	0.2524
3	0.2598	0.2598	0.2598	0.2597	0.2597	0.2597

Tabelle 5.3: Erosionswirkung des Wasserflusses (Test 1) 1

Landschaft	Anfangsscore	60	70	80	90	100
1	0.0245	0.0245	0.0245	0.0245	0.0245	0.0245
2	0.2525	0.2523	0.2523	0.2523	0.2522	0.2522
3	0.2598	0.2597	0.2596	0.2596	0.2596	0.2596

Tabelle 5.4: Erosionswirkung des Wasserflusses (Test 1) 2

An den Werten kann entnommen werden, dass sich die Landschaft nur sehr wenig verändert haben kann, da der Score sehr nahe beieinander liegt. Die erste Idee zu Verbesserung des Scores lag darin, die Iterationsschritte drastisch zu erhöhen. Denn das Wasser wirkt sich nur lokal auf die Landschaft aus, und damit eine deutliche Veränderung an der Landschaft zu sehen ist, muss das Wasser auf einen größeren Bereich wirken.

Landschaft	Anfangsscore	500	1000	1500	2000	2500
1	0.0245	0.0240	0.0233	0.0231	0.0232	0.0232
2	0.2525	0.2473	0.2430	0.2396	0.2370	0.2329
3	0.2598	0.2564	0.2532	0.2504	0.2481	0.2460

Tabelle 5.5: Erosionswirkung des Wasserflusses (Test 1) - Erweiterung der Iterationen

An den Werten ist zu sehen, dass sich der Score mit Erhöhung der Iterationsschritte deutlich steigert. Das hat natürlich Auswirkung auf die Laufzeit (mehr dazu in Kapitel 5.4).

Um die Stärke der Veränderungen zu erhöhen, wurde die maximale globale Sediment-Transport-Kapazität-Konstante K_c wurde 2.5 angehoben und der Regenparameter auf null gesetzt, sodass nur eine Wasserquelle vorhanden war.

Schon bei der ersten Landschaft wurde der anschließende Testlauf abge-

brochen, da sich die Landschaft unnatürlich verändert hat, wie im Bild 5.15 zu sehen.

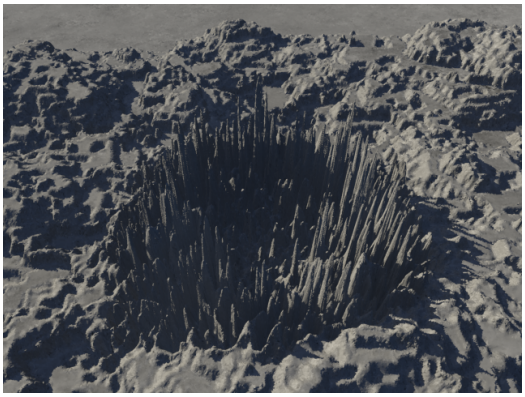

Abbildung 5.16: Unnatürlicher Erosionsprozess durch Verwendung $K_c = 1.5$ (Visualisierung mit Terragen©)

Ursache dafür liegt an der Formel, wie der Sediment-Transport berechnet wird. Wie im Kapitel 2.3.1 nachzulesen ist, wird das aufgeweichte Sediment von einer Stelle entgegen der Strömungsgeschwindigkeit, die als Vektor $\vec{v}_{x,y}$ gespeichert wird, genommen. Da nun das Wasser über eine sehr hohe Sediment-Transport-Kapazität verfügt, wird viel Material auf die neue Position transportiert (siehe sehr vereinfachte Abbildung 5.17). Da nun viel mehr Material vorhanden ist fließt das Wasser wieder zurück, nur um mit noch mehr Wasser und mehr Sediment zurück zu kommen. Wenn dies geschieht schaukelt sich die Landschaft lokal auf.

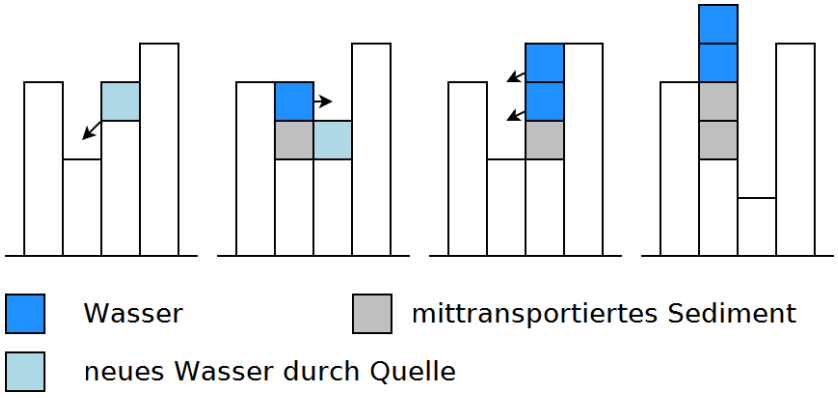

Abbildung 5.17: Aufschaukeln des Sedimenttransported

Test mit den eigens entwickelten Formeln

Es wurde der Sediment-Transport wie in Kapitel 3.1.3 umgeschrieben und der gleiche Testlauf mit den gleichen Parametern wie in Kapitel 5.3.1 durchgeführt.

Landschaft	Anfangsscore	10	20	30	40	50
1	0.0245		0.0245	0.0245	0.0245	0.0245
2	0.2525	0.2524	0.2524	0.2523	0.2522	0.2521
3	0.2598	0.2598	0.2597	0.2596	0.2596	0.2596

Tabelle 5.6: Erosionswirkung des Wasserflusses (Test 2) 1

Landschaft	Anfangsscore	60	70	80	90	100
1	0.0245	0.0245	0.0244	0.0244	0.0243	0.0243
2	0.2525	0.2521	0.2520	0.2519	0.2518	0.2517
3	0.2598	0.2595	0.255	0.2594	0.2594	0.2593

Tabelle 5.7: Erosionswirkung des Wasserflusses (Test 2) 2

Der Score ist schlechter geworden. Nach hundert Iterationen hat sich weniger in der Landschaft geändert, als ohne die Veränderung.

Der Hauptgrund, weshalb die Formel verändert wurde, die Sediment-Transport-Kapazität-Konstante K_c zu erhöhen um mit weniger Iterationsschritten mehr Sediment anzutragen führte zu einem ähnlichen Ergebnis wie in der Abbildung 5.16 zu sehen war, wenn auch nicht so stark.

Test mit erhöhtem Wasservorkommen

Da die Erhöhung der Sediment-Transport-Kapazität-Konstante K_c keine Verbesserung brachte wurde die Wassermenge erhöht, die auf die Landschaft einwirkt. Es wurde die Menge an der Wasserquelle um 30 auf 50 Einheiten pro Iteration angehoben und die Anzahl der zufälligen Regenpositionen auf 100. Die ermittelten Werte dieses Durchlaufes ist in den Tabellen 5.8 und 5.9 angegeben.

Landschaft	Anfangsscore	10	20	30	40	50
1	0.0245	0.0245	0.0246	0.0246	0.0246	0.0246
2	0.2525	0.2524	0.2524	0.2524	0.2523	0.2522
3	0.2598	0.2598	0.2597	0.2597	0.2596	0.2594

Tabelle 5.8: Erosionswirkung des Wasserflusses (Test 3) 1

Landschaft	Anfangsscore	60	70	80	90	100
1	0.0245	0.0246	0.0247	0.0247	0.0247	0.0247
2	0.2525	0.2521	0.2520	0.2519	0.2518	0.2516
3	0.2598	0.2593	0.2592	0.2590	0.2588	0.2587

Tabelle 5.9: Erosionswirkung des Wasserflusses (Test 3) 2

Wenn die Ergebnisse mit den Werten aus den Tabellen 5.3 und 5.5 verglichen werden, ist zu sehen, dass sich der Score bei den Landschaften 2 und 3 etwas stärker verändert, es wird mehr Sediment abgetragen. Bei Landschaft 1 hingegen steigt der Score minimal an. Das kommt davon, dass das Wasser auf eine fast ebene Fläche trifft, Material aus der Landschaft herauslöst und durch die Verteilung des Wassers das Material verteilt. Dadurch entstehen kleine Verwerfungen und der Score steigt.

Eine weitere Testreihe, die in den beiden Tabellen 5.10 und 5.11 zu sehen ist, zeigt die Stärke der Erosion bei einer weiteren Veränderung der Wasserparameter. Hier wurde die Wasserquelle auf 100 Einheiten gesetzt und 20 Einheiten Regenwasser bei 400 zufälligen Positionen pro Iteration.

Landschaft	Anfangsscore	10	20	30	40	50
1	0.0245	0.0245	0.0246	0.0246	0.0246	0.0246
2	0.2525	0.2524	0.2524	0.2522	0.2521	0.2519
3	0.2598	0.2598	0.2597	0.2595	0.2593	0.2591

Tabelle 5.10: Erosionswirkung des Wasserflusses (Test 4) 1

Die Reihe zeigt, dass, wie zu erwarten, der Erosionseffekt stärker wird.

In weiteren Testreihen, die hier nicht im Einzelnen aufgeführt werden, konnte die Verstärkung der Erosion bis zu einer gewissen Wassermenge fortgeführt werden. Irgendwann ist ein Punkt erreicht, an dem mehr Wasser

Landschaft	Anfangsscore	60	70	80	90	100
1	0.0245	0.0247	0.0247	0.0247	0.0248	0.0248
2	0.2525	0.2518	0.2516	0.2515	0.2513	0.2512
3	0.2598	0.2589	0.2586	0.2583	0.2580	0.2578

Tabelle 5.11: Erosionswirkung des Wasserflusses (Test 4) 2

keine Auswirkungen mehr auf den Score hat.

Insgesamt konnte die Erosion großflächig auf die Landschaft angewendet werden, der erhoffte Effekt, dass sich das Wasser als Flusslauf in die Landschaft hinein gräbt, konnte nicht simuliert werden.

Es wurden noch viele weitere Testreihen, dessen einzelne Parameterkonstellationen nicht näher aufgeführt werden, durchgeführt, aber mit keinem konnte ein Flusslauf simuliert werden. Da in dem originalen Paper [MDH07], beziehungsweise in den Arbeiten auf dem dieses Paper aufbaut, nur sehr wenige Parameter explizit angegeben waren, könnte dieser Fehler auf eine falsche Einstellung der Parameter zurückzuführen sein. Daher wurden E-Mails an die ursprünglichen Entwickler des Modells geschickt, die bis zum heutigen Tag unbeantwortet blieben.

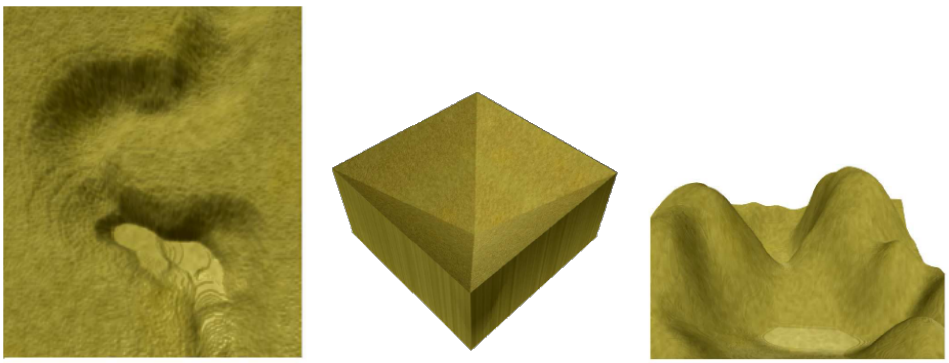

Abbildung 5.18: Landschaften die nach dem Modell im Paper [MDH07] verwendet wurden

Eine weitere mögliche Erklärung kann sein, dass sich die erzeugten Land-

schaften nicht besonders gut für dieses Modell eignen. Die Landschaften, die im ursprünglichen Paper verwendet wurden (siehe Abbildung 5.18), weisen nur wenige sprunghafte Höhenunterschiede auf, sodass das Wasser gezwungen ist, in eine Richtung zu fließen. Es sind quasi nur winzige Ausschnitte aus einer Landschaft mit einer zentraler Anhöhe. Die Landschaften, die in dieser Arbeit verwendet wurden, weisen sehr viele Gebirge und Täler auf. Daher sind die Höhenunterschiede der anliegenden Zellen zu sprunghaft, das sich das Wasser innerhalb dieser Landschaft nicht gut verteilen kann, das heißt nicht richtig fließen kann.

5.3.2 Erosionswirkung der thermalen Erosion

In den beiden Tabellen 5.12 und 5.13 werden die Werte der Wirkung der thermalen Erosion gezeigt.

Landschaft	Anfangsscore	10	20	30	40	50
1	0.0245	0.0245	0.0245	0.0245	0.0245	0.0245
2	0.2525	0.2488	0.2468	0.2453	0.2439	0.2427
3	0.2598	0.2345	0.2226	0.2136	0.2065	0.2005

Tabelle 5.12: Erosionswirkung der thermalen Erosion 1

Landschaft	Anfangsscore	60	70	80	90	100
1	0.0245	0.0245	0.0245	0.0245	0.0245	0.0245
2	0.2525	0.2416	0.2406	0.2397	0.2388	0.2380
3	0.2598	0.1954	0.191	0.1871	0.1837	0.1806

Tabelle 5.13: Erosionswirkung der thermalen Erosion 2

Hier wird nach dem oben erwähnten Testverlauf abermals der Score alle 10 Iterationen ermittelt. Dieser Score wird in der Abbildung 5.19 grafisch aufbereitet.

An dieser Grafik ist zu erkennen, dass sich der Score immer einem nicht näher bestimmbaren Minimum annähert. Bei der Landschaft . gab es keinerlei Hügel, sodass die Erosionswirkung der thermalen Erosion bei null liegt.

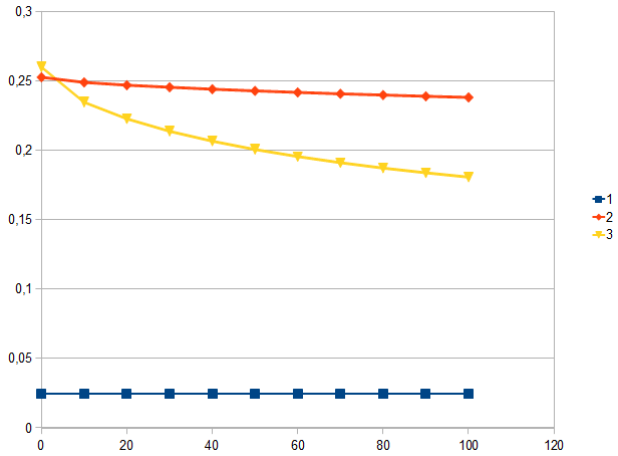

Abbildung 5.19: Diagramm des Erosions Score bei Thermaler Erosion

Bei einer Landschaft mit wenigen Hügeln, wie die Landschaft 2, sind, nähern sich die Veränderungen mit fortschreitender Zeit einem Minimum an. Genau das ist auch bei der Testlandschaft 3 zu sehen, aber etwas stärker ausgeprägt, da hier mehr Geröll abrutschen kann. Dadurch ist die Veränderung der Landschaft am Anfang sehr stark, gegen Ende ist sie abgeschwächt.

Irgendwann gibt es keine Erosionswirkung mehr, da sämtliches Material abgerutscht ist. Dann ist das Minimum erreicht, das heißt der Score verändert sich nicht mehr weiter. Dies wurde anhand längeren Testläufen bestätigt.

5.4 Berechnungszeit der Erosionsprozesse

In den folgenden Unterkapiteln werden die Zeiten für die Berechnungszeit der einzelnen Erosionsarten aufgezeigt.

Für die Testläufe werden wieder die beiden Testrechner aus dem vorherigen Kapitel verwendet. Die Ergebnisse des Erosionsscore wurden zeitgleich mit Zeitmessungen durchgeführt, damit sie zusammenpassen. Für die Zeitmessung wurde nur die jeweilige Methode zum Ausführen des Erosionspro-

zesses gemessen, sodass keine unnötigen Berechnungen mit einfließen.

5.4.1 Berechnungszeit des Wasserflusses

In den Tabellen 5.14 und 5.15 sind die Ergebnisse der Zeitmessung für den Testlauf 1 (siehe Kapitel 5.3.1) angegeben.

Landschaft	Rechner	10	20	30	40	50
1	1	1350ms	2686ms	3840ms	5055ms	6330ms
1	2	912ms	1765ms	2602ms	3495ms	4196ms
2	1	1010ms	2007ms	3042ms	4138ms	5184ms
2	2	732ms	1423ms	2078ms	2874ms	3435ms
3	1	1180ms	1959ms	2964ms	3976ms	4946ms
3	2	793ms	1385ms	2034ms	2625ms	3323ms

Tabelle 5.14: Zeitmessung des Wasserflusses 1

Landschaft	Rechner	60	70	80	90	100
1	1	9674ms	10740ms	11714ms	12594ms	13369ms
1	2	5185ms	6126ms	6998ms	7923ms	8843ms
2	1	9772ms	10863ms	11853ms	12726ms	13558ms
2	2	4194ms	5034ms	5673ms	6532ms	7307ms
3	1	5853ms	6806ms	7792ms	8642ms	9643ms
3	2	3867ms	4507ms	5223ms	5790ms	6478ms

Tabelle 5.15: Zeitmessung des Wasserflusses 2

Es ist zu sehen, dass die Berechnungszeit mit zunehmenden Fortschritt steigt. Um einen Vergleich zu erhalten, wie sich die Berechnungszeit mit mehr Iterationen verhält, wurde ein Testlauf auf Rechner 2 mit 1000 Iterationen gestartet. Der Durchschnitt dieser Zeit ist in der Tabelle 5.22 zu sehen.

Das Ergebnis des Laufes ist, dass die durchschnittliche Berechnungszeit des Rechners 2 bei N = 512 in etwa so groß ist, wie die Rechenzeit nach 100 Iterationen. Die liegt daran, dass sich die Anzahl der Zellen, in der sich Wasser befindet, angleicht. Den es müssen nur die Zellen berechnet werden, die mit dem Wasser zu tun haben. Dies erklärt auch, dass die Berechnungszeit bei

N	∅ Zeit nach 1000 Iterationen
32	45ms
64	347ms
128	1308ms
256	4344ms
512	8307ms
1024	15847ms

Tabelle 5.16: Zeitmessung der Wasserflusses in Abhänigkeit von N

der flachen Landschaft 1 deutlich höher ist. Denn hier verteilt sich das Wasser stärker über die Fläche, während sich bei den anderen beiden Landschaften das Wasser in den Tälern sammelt.

5.4.2 Berechnungszeit der thermalen Erosion

Die Werte der Zeitmessung der thermalen Erosion kann den Tabellen 5.17 und 5.18 entnommen werden.

Landschaft	Rechner	10	20	30	40	50
1	1	2803ms	2831ms	2827ms	2845ms	2827ms
1	2	1847ms	1883ms	1912ms	1844ms	1850ms
2	1	2923ms	2839ms	2847ms	2820ms	2823ms
2	2	1938ms	1923ms	1878ms	1891ms	1912ms
3	1	2898ms	2837ms	2831ms	2865ms	2837ms
3	2	1923ms	1843ms	1921ms	1928ms	1893ms

Tabelle 5.17: Zeitmessung der thermalen Erosion 1

Landschaft	Rechner	60	70	80	90	100
1	1	2802ms	2794ms	2808ms	2810ms	2836ms
1	2	1824ms	1793ms	1796ms	1783ms	1802ms
2	1	2823ms	2823ms	2848ms	2831ms	2893ms
2	2	1887ms	1832ms	1874ms	1884ms	1823ms
3	1	2837ms	2834ms	2850ms	2818ms	2893ms
3	2	1897ms	1864ms	1894ms	1902ms	1893ms

Tabelle 5.18: Zeitmessung der thermalen Erosion 2

Aus den Messungen lässt sich folgendes Diagramm 5.20 für Rechner 2 erstellen.

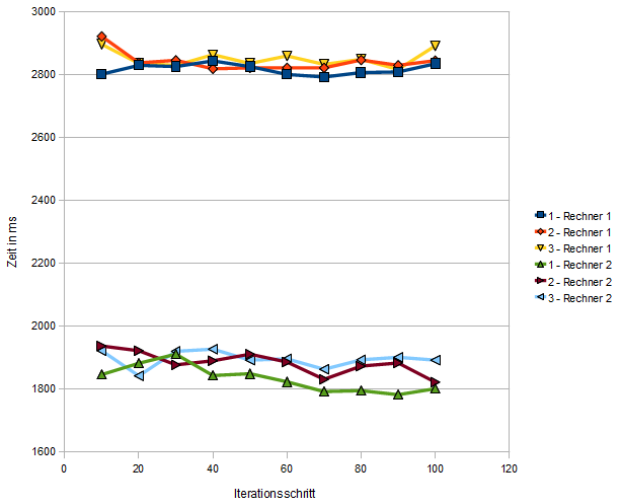

Abbildung 5.20: Diagramm der Zeitmessung der thermalen Erosion

Die Messunterschiede liegen zwischen 2% und 5% und können zu einem Teil auf die nicht vermeidbare Hintergrundaktivität des Betriebssystems zurückgeführt werden.

Trotzdem können einige Folgerungen gezogen werden. Die relativ flache Landschaft 1 benötigt fast durchweg die gleiche Zeit, in einem Ermessensspielraum von 2%, als die beiden anderen Landschaften. Die Landschaft 3 benötigt im Durchschnitt mehr Zeit als die anderen Beiden.

Hier kann der Schluss gezogen werden, dass eine zerklüftete Landschaft insgesamt länger bei der Berechnung der thermalen Erosion benötigt als eine weniger zerklüftete Landschaft. Die Zeiten liegen allerdings jeweils in einem Bereich von 5%.

Für den Rechner 2 wurde eine weitere Testreihe vorgenommen, wie sich die Zeit durch eine Veränderung der Landschaftgröße verändert. Die Ergebnisse sind in der Tabelle 5.19 zu finden.

An diesen Werten der Tabelle 5.19 ist zu sehen, das sich mit einer Ver-

N	∅ Zeit nach 1000 Iterationen
32	7ms
64	25ms
128	121ms
256	417ms
512	1787ms
1024	7083ms

Tabelle 5.19: Zeitmessung der thermalen Erosion in Abhänigkeit von N

dopplung der Seitenlänge die Zeiten in etwa Vervierfachen. Die Zeiten steigen linear.

5.4.3 Berechnungszeit des Gesteinskreislaufes

Bei der Berechnungszeit des Gesteinskreislaufes wird die Überprüfung der einzelnen Schichten, ob sie zu einer härteren Schicht umgewandelt wird oder nicht, gemessen. Der Grund hierfür liegt, wie bei der Abbildung 3.2 auf Seite 66 nachgeschlagen werden kann, dass die Umwandlung eines Teils der Schicht in Regolith von der jeweiligen Erosionsmethode vorgenommen wird.

Die Ergebnisse dieser Überprüfung sind den Tabellen 5.20 und 5.21 zu entnehmen.

Landschaft	Rechner	10	20	30	40	50
1	1	4510ms	4460ms	4667ms	4478ms	4502ms
1	2	3001ms	2953ms	3110ms	3012ms	3102ms
2	1	4691ms	4675ms	4744ms	4655ms	4511ms
2	2	3148ms	3114ms	3168ms	3194ms	3010ms
3	1	4483ms	4471ms	4466ms	4439ms	4316ms
3	2	2954ms	2987ms	3047ms	3073ms	2932ms

Tabelle 5.20: Zeitmessung des Gesteinskreislaufes 1

Auf diesen Daten beruhend stellt das Diagramm 5.21 den zeitlichen Verlauf dar.

Landschaft	Rechner	60	70	80	90	100
1	1	4497ms	3878ms	3205ms	3230ms	3213ms
1	2	3023ms	2589ms	2094ms	2120ms	2179ms
2	1	4692ms	3895ms	3207ms	3247ms	3193ms
2	2	3103ms	2684ms	2228ms	2123ms	2103ms
3	1	4293ms	3761ms	3116ms	3219ms	3176ms
3	2	2843ms	2632ms	2234ms	2185ms	2074ms

Tabelle 5.21: Zeitmessung des Gesteinskreislaufes 2

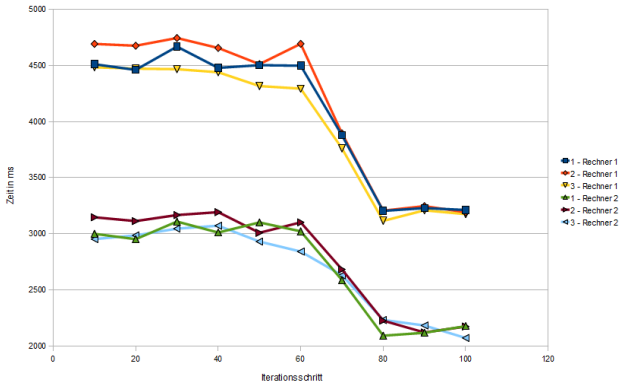

Abbildung 5.21: Diagramm der Zeitmessung des Gesteinskreislaufs

An dem Diagramm ist sehr gut zu erkennen, dass der Berechnungsaufwand, und damit die Zeit, nach 60 - 80 Iterationsschritten deutlich geringer wird. Der Grund hierfür liegt darin, dass sich die Anzahl der Schichten von vier auf drei verringert hat. Das kommt davon, dass sich nach 64 Iterationen die Sandsteinschicht in die Kalksteinschicht, mit deren Eigenschaften umwandelt.

Hier kann also festgestellt werden, das sich der Rechenaufwand mit der Anzahl der Schichten erhöht.

Ein Testlauf in Abhängigkeit der Landschaftsgröße N ist in der folgenden Tabelle 5.22 zu sehen.

N	∅ Zeit nach 1000 Iterationen
32	9ms
64	65ms
128	141ms
256	568ms
512	2195ms
1024	17944ms

Tabelle 5.22: Zeitmessung der Gesteinskreislaufes in Abhänigkeit von N

Im Gegensatz zu thermalen Erosion ist hier der Verlauf potenziell. Der Grund hier für liegt darin, dass sich durch Verdoppelung der Seitenlänge nicht nur Fläche vervierfacht, sondern gleichzeitig auch damit die Anzahl der Schichten vervielfacht.

5.5 Erzeugte Landschaften

In diesem Kapitel werden einige Landschaften, vor und nachdem auf sie ein Erosionsprozess eingewirkt hat, visuell dargestellt.

Die folgenden Bilder wurden zur besseren Übersicht mit dem Programm Terragen© gerendert.

Das erste Bilderpaar 5.23 zeigt eine selbst erstellte Landschaft, die nach den Modellen aus dem Paper [MDH07] gefertigt wurde, auf die Regen gefallen ist. Links daneben sind die Originalbilder aus dem Paper.

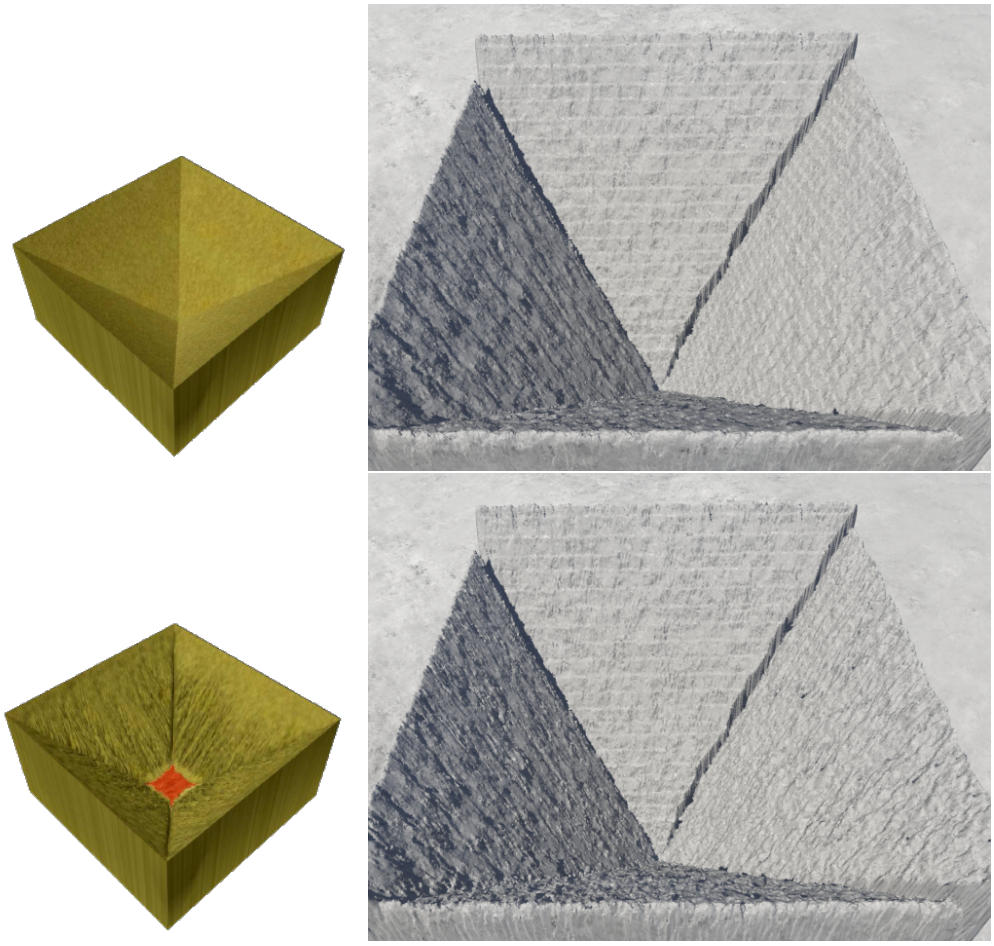

Abbildung 5.22: Selbsterstellte Landschaft auf die Regen gefallen ist

Der Treppcheneffekt, der innerhalb der Landschaft zu sehen ist, kommt von dem grafischen Interpolator des Grafikprogramms, mit dem die Heightmap erzeugt wurde. Das untere Bild zeigt eine Landschaft nach 1000 Iterationen der Wassererosion.

Für die nächsten beiden Bildern in 5.23 wurde eine schräge Fläche erzeugt, auf der eine Wasserquelle im oberen Bereich einwirkt. Der Wassermenge war gering gewählt (3 Einheiten pro Iteration) wodurch sich nur ein

wenig der Wasserlauf herausbildet. Die auf der Landschaft vertikalen Streifen stammen wieder von dem Bildbearbeitungsprogramm.

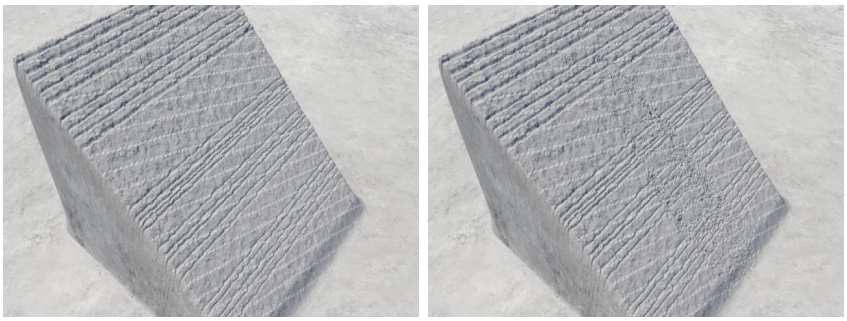

Abbildung 5.23: Selbsterstellte Landschaft mit Wasserquelle

Zu guter Letzt ist noch ein Bilderpaar 5.24 mit der Auswirkung der thermalen Erosion auf die Landschaft.

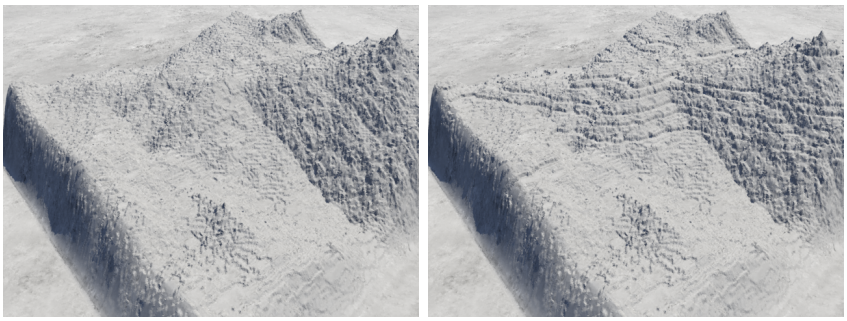

Abbildung 5.24: Landschaftsveränderung durch die thermale Erosion

Damit wurden alle möglichen Erosionsarten mit je einem Bild vorgestellt.

I don't seek power and do not run around.

Benoît Mandelbrot

Kapitel 6

Zusammenfassung und Ausblick

Die im Rahmen dieser Masterarbeit entwickelte Teilkomponente für GLAB ist in der Lage eine fraktale Landschaft auf mehrere Arten zu erzeugen. Entweder als 2D-Textur in Form einer Heightmap oder als 3D-Modell, welches auf Schichten aufgebaut ist. Auf jede dieser Urlandschaften lassen sich verschiedene Erosionsprozesse anwenden und durch verschiedene Parameter beeinflussen.

Die Klassenstruktur wurde so entworfen, dass es für den späteren Nutzer einfach ist eine eigene Datenstruktur für die Landschaft zu schreiben, einen Generator oder gegebenenfalls einen neuen Erosionsprozess zu implementieren.

Es wurden insgesamt drei Erosionsmodelle erarbeitet: die thermale Erosion und Erosion durch stehendes und fließendes Wasser.

Die thermale Erosion wirkt nur bei hügeligen Landschaften und dort sehr stark. Da sie in der Natur ein langwieriger Prozess ist, muss die Methode nicht dauernd aufgerufen werden, sondern nur gelegentlich. Die Berechnungszeit

ist hier bei nur abhängig von der Landschaftsgröße und dem Prozessor. Bei einem 3GHz Rechner und einer Landschaft mit den Maßen 512x512 benötigt die Berechnung einer Iteration in etwa zwei Sekunden.

Die nächsten beiden Erosionsmodelle, stehendes und fließendes Wasser, wurden vereint, da sie teilweise die gleichen Berechnungen benötigten und somit Berechnungszeit eingespart werden kann. Das Modell sollte es ermöglichen sich in die Landschaft zu graben, damit es aussieht als wenn ein Fluss durchgelaufen ist. Im Verlauf dieser Arbeit stellte sich jedoch heraus, dass dieses Modell ungeeignet ist einen Flusslauf zu simulieren. Für Regen, also lokale Erosion, wirkt das Modell hingegen gut.

Dieses Modell sehr häufig durchgeführt werden, um den Wasserfluss zu berechnen. Die Umgesetzte Variante benötigt für eine Iteration, bei einem 3GHz Rechner und einer Landschaft mit den Maßen 512x512, in etwa acht Sekunden. Es hat sich herausgestellt, das sich die Berechnungszeit stark von der Anzahl der Zellen abhängt, die mit Wasser bedeckt sind, da hier umfangreiche Berechnungen durchgeführt werden müssen. Würde es einen richtigen Flusslauf geben, der vom oberen Bereich der 512x512 großen Landschaft zum unteren Bereich fließen würde, so benötigt allein diese Berechnung (512 Zellen * 8 Sekunden = 4096 Sekunden) etwa 68 Minuten.

Als Alterungsprozess wurde der Gesteinskreislauf realisiert. Es wurden insgesamt acht verschiedene Gesteinstypen definiert und unterschiedlichen Eigenschaften zugewiesen, die für die Erosionsmodelle benötigt werden. Periodisch wird überprüft, ob sich ein Gesteinstyp durch virtuellen Druck und Wärme in einen anderen Typ mit anderen Eigenschaften umwandelt. Sie hat also keine optische Beeinflussung. Die Berechnungszeit ist abhängig von der Anzahl der Schichten, der Größe der Landschaft und der CPU. Sie benötigt in etwa zwischen 1 bis 16 Sekunden, je nachdem, wie viele Schichten es gibt.

Bei den vorgestellten Erosionsalgorithmen ist das Hauptproblem die Zeit, die benötigt wird, um diese zu berechnen.

Hier wäre ein Ansatzpunkt die Algorithmen zu parallelisieren, da sich die

Berechnungen immer nur lokal auf einen Bereich der Landschaft begrenzen. Geeignet dafür wäre die Aufteilung auf in vier Threads, die auf den einzelnen Kernen eines Quadcores verteilt werden. Jeder dieser vier Threads kann dann eine quadratische Teilfläche der Landschaft für sich berechnen.

Eine weitere Überlegung wäre es die Algorithmen für die Anwendung auf einer Grafikkarte aufzubereiten um sie noch schneller berechnen zu können.

Die Überprüfung der Schichten für den Gesteinskreislauf ist sehr zeitintensiv. Auf die strikte Einteilung in die unterschiedlichen Gesteinsarten kann eventuell verzichtet werden, wenn nur noch die physikalischen Eigenschaften bearbeitet werden würden. Also wenn eine Schicht nach hundert Iterationen nicht verändert hat, wird zum Beispiel die Sedimentskonstante K_s um 10% veringert und der Talus-Winkel um 10% erhöht. So würden hier sämtliche Bedingungen entfallen.

Das Modell der thermalen Erosion kann uneingeschränkt empfohlen werden, die Erosionswirkung als auch die Berechnungszeit liegen in einem vernünftigen Rahmen.

Das Erosionsmodell für den Wasserlauf ist dagegen problematischer. Es benötigt sehr lange, da die Berechnungen immer wieder hintereinander durchgeführt werden müssen, erst so kann der Wasserlauf simuliert werden. Ein richtiger Wasserfluss, der sich in die Landschaft gräbt, ist damit nicht realisierbar. Gut geeignet ist es, um Regen zu simulieren.

Das neu entwickelte Schichtenmodell kann sich sehr gut an diese Modelle anpassen, wobei der Aufwand für die Berechnung je nach Anzahl der Schichten enorm sein kann. Hier könnte eine Alternative zu dem Modell erarbeitet werden, um sie schneller zu machen.

Abbildungsverzeichnis

Tabellenverzeichnis

Übersicht der mathematischen Abkürzungen

Dieser Anhang bietet eine Übersicht der verwendeten mathematischen Bezeichnungen.

x, y	Position x,y
x_n, y_n	Position eines Nachbarn der Position x,y

$h_{x,y}$	die Höhe an der Stelle x/y, definiert in (2.12)
h_{max}	maximale Höhe in einer 2D-Matrix
N	Seitenlänge einer Fläche

T	Zeiteinheit
Δt	Teilintervall
ΔP	Anzahl der Perioden
u	Signalstärke
v	Frequenz

h_{E1}	die Höhe in einer Ecke1 einer 2D-Matrix
h_{Mi}	die Höhe in der Mitte einer 2D-Matrix
i	Anzahl der Iterationen

$s_{x,y}$	Menge aller Sedimentsschichten an der Position x/y
$r_{x,y}$	Regolith, als besondere Schicht $s_{x,y}$
$n_{x,y}$	Das im Wasser befindliche gelöstes Gesteinsmaterial lagert sich anschließend als Regolith $r_{x,y}$ ab
$w_{x,y}$	Wassermenge an der Stelle x/y
$d_{x,y}$	eine Differenz der Höhe von x/y und dem Nachbarn x_n, y_n
$d_{max}^{x,y}$	die höchste Differenz der Höhe um die Position x/y
$\overline{d}$	Erwartungswert der Höhe
σ_d	Varianz der Höhe
ϵ	Score der Landschaft
$c_{x,y}$	Anzahl der Schichten an der Position x/y
$f_{x,y}$	kompletter Wasser-Ausfluss aus der Zelle an der Position x/y
f^R	Wasser-Ausfluss in die rechte Nachbarzelle
f^L	Wasser-Ausfluss in die linke Nachbarzelle
f^B	Wasser-Ausfluss in die untere Nachbarzelle
f^T	Wasser-Ausfluss in die obere Nachbarzelle
$f_{x-1,y}^R$	Wasser-Ausfluss in die rechte Nachbarzelle von der linken Nachbarzelle (=x-1,y)
A	Fläche zwischen zwei Wasserfeldern
g	globale Gravitation
l	Länge einer Zelle
K_f	lokaler Skalierungsfaktor
ΔV	Differenz zwischen Inputflow und Outputflow
ΔW_x	Differenz der Flussgeschwindigkeit in x Richtung
$\vec{v}_{x,y}$	Veocity Vektor an Poistion x/y
$\overline{w}$	durchschnittlicher Wasserstand
$C_{x,y}$	Sediment-Transport-Kapazität an Position x/y
K_c	maximale globale Sediment-Transport-Kapazität-Konstante
K_s	lokale Sedimentskonstante, abhängig von der Gesteinsart
K_V	globale Verdunstungskonstante
T_a	Talus Angle
K_T	Lokale Talus Angle Konstante, abhängig von der Gesteinsart

m_i	Masse der i-ten Schicht

p	Druck
F_g	Energie, im speziellen Gravitationsenergie
$z_{x,y}$	Skalierungsfaktor, zum berechnen des im Wasser gelösten Material

Literaturverzeichnis

[BA05] Beneš, Bedřich ; Arriaga, X.: Table Mountains by Virtual Erosion, 2005

[Ben07] Beneš, Bedřich: Real-Time Erosion Using Shallow Water Simulation. In: *VRIPHYS 2007* 4 (2007)

[BF01] Beneš, Bedřich ; Forsbach, Rafael: Layered Data Representation for Visual Simulation of Terrain Erosion. In: *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics*. Washington, DC, USA : IEEE Computer Society, 2001. – ISBN 0–7695–1215–1, S. 80

[BTHB06] Beneš, Bedřich ; Těšínský, Václav ; Hornyš, Jan ; Bhatia, Sanjiv K.: Hydraulic erosion: Research Articles. In: *Comput. Animat. Virtual Worlds* 17 (2006), May, 99–108. http://portal.acm.org/citation.cfm?id=1133115.1133120. – ISSN 1546–4261

[BvBK08] Beneš, Bedřich ; Šťava, Ondřej ; Brisbin, Matthew ; Křivánek, Jaroslav: Interactive terrain modeling using hydraulic erosion. In: *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2008. – ISBN 978–3–905674–10–1, S. 201–210

[CMF98] Chiba, Norishige ; Muraoka, Kazunobu ; Fujita, K.: An erosion model based on velocity fields for the visual simulation of mountain scenery. In: *Journal of Visualization and Computer Animation* 9 (1998), S. 185–194

[CMSM90] Chiba, Norishige ; Muraoka, Kazunobu ; Sinya, Endo ; Mamoru, Miura: Terrain Simulation Based on the Recursive Refinement of Ridge-lines Considering Erosion Processes. In: *ITEJ Technical Report* 14 (1990), Nr. 73, 7-12. http://ci.nii.ac.jp/naid/110003706906/en/. – ISSN 03864227

[DEJ+99] Dorsey, Julie ; Edelman, Alan ; Jensen, Henrik W. ; Legakis, Justin ; Pedersen, Hans K.: Modeling and rendering of weathered stone. In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1999 (SIGGRAPH '99). – ISBN 0–201–48560–5, 225–234

[FB02] Forsbach, Rafael ; Beneš, Bedřich: Visual Simulation of Hydraulic Erosion. In: *Journal of WSCG* 10 (2002), S. 2002

[FFC82] Fournier, Alain ; Fussell, Don ; Carpenter, Loren: Computer rendering of stochastic models. In: *Commun. ACM* 25 (1982), June, 371–384. http://doi.acm.org/10.1145/358523.358553. – ISSN 0001–0782

[FM96] Foster, Nick ; Metaxas, Dimitri: Realistic animation of liquids. In: *Graph. Models Image Process.* 58 (1996), September, 471–483. http://portal.acm.org/citation.cfm?id=244304.244315. – ISSN 1077–3169

[FQZ+03] Feng, Huamin Q. ; Qiu, Feng ; Zhang, Nan ; Kaufman, Arie ; Wan, Ming: Ray Tracing Height Fields. In: *Procedings of Computer Graphics International*, 2003, S. 202–207

[GR87] Greengard, L. ; Rokhlin, V.: *A Fast Algorithm for Particle Simulations*. 1987

[Hau08] Hausdoerfer, A.: Wenn Steine erzaehlen ... In: *Praxis Geographie 5/2008* (2008)

[HHGG94] Horn, Jeffrey ; Horn, Jeffrey ; Goldberg, David E. ; Goldberg, David E.: Genetic Algorithm Difficulty and the Modality

of Fitness Landscapes. In: *Foundations of Genetic Algorithms 3*, Morgan Kaufmann, 1994, S. 243–269

[Hom06] Homann, Jörg: *Fraktale Planetengenerierung*, Johann Wolfgang Goethe-Universität, Diplomarbeit, 2006

[HW04] Holmberg, Nathan ; Wünsche, Burkhard C.: Efficient modeling and rendering of turbulent water over natural terrain. In: *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. New York, NY, USA : ACM, 2004. – ISBN 1–58113–883–0, S. 15–22

[IN12] Issei, Fujishiro ; Nao, Ozawa: A Morphological Approach to Volume Synthesis of Weathered Stones. In: *Natural science report of the Ochanomizu University* 50 (1999-12), Nr. 2, 21-30. http://ci.nii.ac.jp/naid/110005944209/en/. – ISSN 00298190

[JE06a] Jarocha-Ernst, Alex: *Creating Landscapes with Simulated Colliding Plates*, Rochester Institute of Technology Department of Computer Science, Masterarbeit, 2006

[JE06b] Jarocha-Ernst, Alex: *Creating Landscapes with Simulated Colliding Plates*. 2006

[JS06] Jens Schneider, Rüdiger W. Tobias Boldte B. Tobias Boldte: Real-Time Editing, Synthesis, and Rendering of Innite Landscapes on GPUs, 2006

[KM90] Kass, Michael ; Miller, Gavin: Rapid, stable fluid dynamics for computer graphics. In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1990 (SIGGRAPH '90). – ISBN 0–89791–344–2, 49–57

[KMN88] Kelley, Alex D. ; Malin, Michael C. ; Nielson, Gregory M.: Terrain simulation using a model of stream erosion. In: *SIGGRAPH Comput. Graph.* 22 (1988), June, 263–268. http://doi.acm.org/10.1145/378456.378519. – ISSN 0097–8930

[LM93] Li, Xin ; Moshell, J. M.: Modeling Soil: Realtime Dynamic Models for Soil Slippage and Manipulation. In: *In Computer Graphics Proceedings, Annual Conference Series*, 1993, S. 361–368

[LP06] Lüders, Klaus ; Pohl, Robert O.: *Pohls Einführung in die Physik*. http://ebooks.ub.uni-muenchen.de/8160/. Version: 2006

[Man75] Mandelbrot, Benoît B.: *Les objets fractals, forme, hasard et dimension*. 1975

[Man77] Mandelbrot, Benoît B.: *Fractals: Form, Chance and Dimension*. 1977

[Man82] Mandelbrot, Benoît B.: *The Fractal Geometry of Nature*. San Francisco : W. H. Freedman and Co., 1982

[Mar07] Marjanovic, David: *Fraktale Stadtgenerierung*, Johann Wolfgang Goethe-Universität, Diplomarbeit, 2007

[MCT99] Mantle Convection, Three dimensional ; Tackley, Paul J.: *Self-Consistent Generation of Tectonic Plates in Time-Dependent, Three-Dimensional Mantle Convection Simulations, Part 1: Pseudo-Plastic Yielding*. 1999

[MDH07] Mei, Xing ; Decaudin, Philippe ; Hu, Bao-Gang: Fast Hydraulic Erosion Simulation and Visualization on GPU. In: *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–3009–5, S. 47–56

[MFC06] Maes, Marcelo M. ; Fujimoto, Tadahiro ; Chiba, Norishige: Efficient animation of water flow on irregular terrains. In: *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*. New York, NY, USA : ACM, 2006 (GRAPHITE '06). – ISBN 1–59593–564–9, 107–115

[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, 39–48

[MKM89] Musgrave, Forest K. ; Kolb, C. E. ; Mace, R. S.: The synthesis and rendering of eroded fractal terrains. In: *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1989. – ISBN 0–89791–312–4, S. 41–50

[MKN+04] Müller, M. ; Keiser, R. ; Nealen, A. ; Pauly, M. ; Gross, M. ; Alexa, M.: Point based animation of elastic, plastic and melting objects. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2004 (SCA '04). – ISBN 3–905673–14–2, 141–151

[MM] Moresi, Louis ; Mühlhaus, Hans-Bernd: *Multi Phase Flows and Computational Aspects of Plate Tectonics*

[Mus93] Musgrave, Forest K.: *Methods for realistic landscape imaging*, Yale University, Diss., 1993

[Nag97] Nagashima, Kenji: Computer generation of eroded valley and mountain terrains. In: *The Visual Computer 13*, 1997, S. 456–464

[OH95] O'Brien, J. F. ; Hodgins, J. K.: Dynamic simulation of splashing fluids. In: *Proceedings of the Computer Animation*. Washington, DC, USA : IEEE Computer Society, 1995 (CA '95). – ISBN 0–8186–7062–2, 198–

[Ols04] Olsen, Jacob: Realtime Procedural Terrain Generation, 2004

[Per85] Perlin, Ken: An image synthesizer. In: *SIGGRAPH Comput. Graph.* 19 (1985), July, 287–296. http://doi.acm.org/10.1145/325165.325247. – ISSN 0097–8930

[PJ] Pabst, Joost Van Lawick V. ; Jense, Hans: *Dynamic Terrain Generation Based on Multifractal Techniques*

[RL01] Rusinkiewicz, Szymon ; Levoy, Marc: Efficient Variants of the ICP Algorithm. In: *INTERNATIONAL CONFERENCE ON 3-D DIGITAL IMAGING AND MODELING*, 2001

[RP93] Roudier, B. P. P. P. Peroche ; Perrin, M.: Landscapes Synthesis Achieved through Erosion and Deposition Process Simulation. In: *Computer Graphics Forum* 12 (1993), S. 375–383

[SBW06] Schneider, Jens ; Boldte, Tobias ; Westermann, Ruediger: Real-Time Editing, Synthesis, and Rendering of Infinite Landscapes on GPUs. In: *Vision, Modeling and Visualization 2006*, 2006

[Sim11] Simetric: *Density of materials*. 2011. – Online verfügbar unter http://www.simetric.co.uk/si_materials.htm, zuletzt besucht am 09.01.2011

[SS05] Stachniak, S. ; Stuerzlinger, W.: An Algorithm for Automated Fractal Terrain Deformation. In: *In Proceedings of Computer Graphics and Artificial Intelligence*, 2005, S. 64–76

[Sta99a] Stam, Jos: Stable fluids. In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1999 (SIGGRAPH '99). – ISBN 0–201–48560–5, 121–128

[Sta99b] Stam, Jos: Stable Fluids, 1999, S. 121–128

[Ste08] Stegmayr, Christofer: *Procedural deformation and destruction in real-time*, -, Masterarbeit, 2008

[Teo08] Teoh, Soon T.: River and Coastal Action in Automatic Terrain Generation. In: *CGVR*, 2008, S. 3–9