HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Bachelorthesis

im Studiengang

Medieninformatik Bachelor

Prozedurale Generierung von Wolken in Offline-Renderengines

mit Fokus auf Anwenderfreundlichkeit

Referent	:	Prof. Christoph Müller
Koreferent	:	Rainer Duda
Vorgelegt am	:	31.08.2019
Vorgelegt von	:	Simon Storl-Schulke Matrikelnummer: 251669 Baumannstraße 7, 78120 Furtwangen simon.storl-schulke@hs-furtwangen.de

Abstract

Die vorliegende Arbeit beschreibt Ansätze zur Generierung von Wolken zur Anwendung in der digitalen Bildsynthese. Dabei werden dem Nutzer Parameter zur Verfügung gestellt, um die häufigsten Wolkenarten mithilfe automatisch generierte Texturen darzustellen. Spezieller Fokus liegt hierbei auf der Optimierung der Anwenderfreundlichkeit, durch den Ausgleich von Komplexität und Einfachheit. Diese Bachelorarbeit ist für Studierende im Bereich Medieninformatik interessant, da sie die Themenfelder Gestaltung und Informatik vereint.

Inhaltsverzeichnis

Abstract I
Inhaltsverzeichnis IV
Abbildungsverzeichnis V
1 Einleitung 3
1.1 Motivation 3
1.2 Ziel der Arbeit 4
1.3 Eingrenzung 4
1.4 Beobachtung realer Wolken 5
1.4.1 Powdered Suggar Look 7
2 Stand der Technik 9
2.1 Rendering von volumetrischen Materialien in Path Tracing Renderengines 9
2.1.1 Path Tracing 9
2.1.2 Volume Scatter 12
2.2 Techniken zur Generierung von Wolken 13
2.2.1 Simulation 14
2.2.2 Point-Cloud-Data Generierung 14
2.2.3 Mesh Displacement 14
2.2.4 Mesh Proximity 15
2.2.5 Volume Displacement 17
3 Implementierung mit OSL 19
3.1 Grundform 19
3.1.1 Ellipsoid 20
3.1.2 Formgradient mit Exponentialfunktion 22
3.1.3 Formgradient mit Gradationskurven 23
3.2 Verzerrungsmethode 23
3.3 Verzerrungstextur 25

3.3.1 Simplex/Perlin Noise 26
3.3.2 Fractal Brownian Motion 28
3.3.3 Worley Noise 30
3.3.4 Perlin-Worley Noise 32
3.3.5 Sinustextur 32
4 Benutzeroberfläche 35
4.1 Modularität oder einzelner Node? 35
4.2 Parameterliste 35
4.3 Animation 39
4.4 Interaktive Wolkenvorschau 40
5 Evaluation 41
5.1 Vor- und Nachteile der Implementierung in OSL 41
5.1.1 Performance 41
5.2 Zusammenfassung und Ausblick 43
Literaturverzeichnis 45

Abbildungsverzeichnis

Abbildung 1: Verschiedene Wolkenarten
Quelle: de.wikipedia.org/wiki/Datei:Cloud_types_de.svg
Urheber: Valentin de Bruyn
Lizenz: CC BY-SA 3.0 creativecommons.org/licenses/by-sa/3.0/deed.de 5

Abbildung 2: Blumenkohlartige sich aufbauschende Strukturen bei Cumuluswolken
Quelle: https://www.flickr.com/photos/gedankenstuecke/96292108
Urheber: Bastian Greshake Tzovaras
Lizenz: CC BY-SA 3.0 6

Abbildung 3: Feine Strukturen bei Stratuswolken und an den Rändern von Cumuluswolken 6

Abbildung 4: Wellenartige Strukturen bei Cirruswolken
Foto Links:
Quelle:commons.wikimedia.org/wiki/File:Cirrus_Clounds.jpg
Urheber: Nissim Angdembay
Lizenz: CC BY-SA 3.0
Foto rechts:
Quelle: en.wikipedia.org/wiki/File:Cirrus_clouds2.jpg Lizenz: CC BY-SA 3. 7

Abbildung 5: Schaubild Path Tracing 9

Abbildung 6: Path Tracing - Material mit unterschiedlichen Roughness Werten 10

Abbildung 7: Path Tracing mit variierender Anzahl an GI Bounces . . . 10

Abbildung 8: Path Tracing mit variierender Anzahl an Volume Bounces 11

Abbildung 9: Path Tracing mit variierender Anzahl an Iterationen/Samples 11

Abbildung 10: Schaubild Volume Scatter 12

Abbildung 11: Volumetrisches Material mit variierender dichte 12

Abbildung 12: Verhalten der Strahlen bei unterschiedlichen Anisotropiewerten 13

Abbildung 13: Wolkengenerierung durch prozedurale Verzerrung eines Grundmeshes (hier Perlin FBM Noise 3.3.2) 15

Abbildung 14: Mesh Proximity - Querschnitt durch Modell aus (Abb. 13 16

Abbildung 15: Wolke mit quadrierter Mesh Proximity als Multiplikator für die Dichte. 16

Abbildung 16: Einfaches OSL Script und dessen Repräsentation als Shadernode in Blender 19

Abbildung 17: Generierte Texturkoordinaten. X,Y,Z Werte als Rot, Grün, Blau dargestellt. 20

Abbildung 18: Veränderte generierte Texturkoordinaten. X,Y,Z Werte als Rot, Grün, Blau dargestellt. 21

Abbildung 19: Grundformfunktion in Blender mit Cycles Nodes für die Erstellung eines Prototyps 22

Abbildung 20: Formvariation mit verschiedenen Exponenten für T_z . . . 22

Abbildung 21: Gradationskurven werden genutzt, um Rotationskörper zu definieren 23

Abbildung 22: Implementierung der Grundformfunktion mit OSL 25

Abbildung 23: Perlin/Simplex Noise 26

Abbildung 24: Mithilfe von Simplex-Noise rein prozedural generierte und texturierte Landschaft 27

Abbildung 25: Wolke mit Perlin Noise als Verzerrungstextur 27

Abbildung 26: FBM Noise mit verschiedenen Werten für Lacunarity und Gain bei 5 Oktaven 28

Abbildung 27: FBM Noise mit OSL 29

Abbildung 28: Wolke mit FBM Simplex Noise als Verzerrungstextur . . 29

Abbildung 29: Implementierung von Voronoi Textur (Distanz zur nächsten Zelle) in OSL nach [Kes] 30

Abbildung 30: *links:* Voronoi Diagramm mit zufällig eingefärbten Zellen.
rechts: Abstand der Zellen des selben Voronoi Diagramms als Luminanz dargestellt. 30

Abbildung 31: Worley Noise mit 1, 2 und 5 FBM Oktaven bei 2.0 Lacunarity und 0.5 Gain 31

Abbildung 32: Wolke mit FBM Worley Noise als Verzerrungstextur . . . 31

Abbildung 33: Wolke mit Perlin-Worley Noise als Verzerrungstextur . . 32

Abbildung 34: Cirruswolke mit wellenartiger Struktur durch Anwendung der Sinustextur 33

Abbildung 35: Benutzeroberfläche der OSL Nodes in Blender 36

Abbildung 36: Einfluss der Parameter auf das Erscheinungsbild der Wolken 38

Abbildung 37: Einfluss der Parameter auf das Erscheinungsbild der Wolken 39

Abbildung 38: Zweidimensionale annähernde Repräsentation (links) der aus den Parametern resultierenden Wolkenform (rechts) 40

Abbildung 39: Vergleich verschiedener Renderingmethoden 43

Glossar

BSDF Bidirectional scattering distribution function - Funktion die beschreibt, wie Licht sich auf einer Oberfläche verhält.. 19

CGI Computer Generated Images - engl. Computergenerierte Bilder. 3

CUDA API zur Programmierung parallelen Prozesse auf NVidia GPUs. 41

GPU Graphics Processing Unit - Grafikprozessor einer Grafikkarte. 1, 41

HDRI High Dynamic Range Image - engl. Bild mit großem Dynamikumfang. In der Computergrafik oft zur natürlichen Beleuchtung von Szenen genutzt.. 43

Nodegroup Ansamlung von verknüpften Shaderbausteinen zur Zusammenfassung komplexer Materialien oder Texturen. 36

OpenCL Quelloffene API zur Programmierung parallelen Prozesse auf GPUs. 41

Path Tracing Spezielle, iterative Form des Raytracings, die realistischere Beleuchtung ermöglicht. 3

Raytracing Rendertechnik, bei der von der Kamera aus Strahlen (Rays) simuliert werden, um die Szene darzustellen. 4, 9

Renderengine Software, die drei dimensionale Daten in zweidimendionale Bilder umwandelt. 3

VDB Dateiformat zum speichern volumetrischer Daten. 14, 43

VFX Visual Effects – Computergenerierte Effekte in Filmen (z.b. Explosionen). 3

1 Einleitung

1.1 Motivation

In modernen Filmen und Serien, sowie im Werbesektor werden immer häufiger realistisch per Computergrafik dargestellte Wolken benötigt. Mit dem immer weiter steigenden Realitätsstandard bei CGI und VFX generell, steigen somit auch für diesen Bereich die Erwartungen des Rezipienten an die computergenerierten Bilder. Die Toleranzgrenze unserer geschulten Augen sinkt immer weiter ab und was vor zehn Jahren noch als Revolution der Filmtechnik gefeiert worden wäre, wird 2019 als schlechtes CGI abgewunken. Wolken sind hierbei eine besondere Herausforderung für 3D-Software und Künstler, da diese speziellen Regeln der Natur folgen und in ihrer Komplexität nicht einfach und nur sehr zeitaufwändig von Hand modelliert werden können. In den letzten Jahren wurden bereits enorme Fortschritte in der Darstellung von volumetrischen Materialien gemacht - nicht zuletzt dank der Adaption von Path Tracing in moderne Studio Pipelines, wodurch die realistische Simulation von Licht in komplexen Situationen stark verbessert wurde. Moderne Path Tracing Renderengines Renderengine , wie zum Beispiel Pixars RenderMan, Octane, Arnold, Cycles oder Corona, können mit Leichtigkeit die komplizierte Brechung von Licht in millionen Staubpartikeln oder Wassertropfen simulieren. Die großen Schritte in der Hardwareentwicklung des letzten Jahrzehnts, sowie der einfachere Zugang zu entsprechender Software, eröffnen nun auch Privatnutzern und kleineren Studios den Zugang zu diesen Technologien. Während die Darstellung der Materialien an sich also immer schneller, besser und einfacher in der Anwendung wird, ist bei der Darstellung von Wolken auch Form, Struktur und Verhalten in Luftströmungen sehr wichtig. Während es hierfür bereits sehr mächtige Werkzeuge - z.b. mit der 3D VFX Software "Houdini"gibt, ist die Anwendung dieser Werkzeuge selbst für technisch versierte Benutzer recht aufwändig.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, ein einheitliches System zur Erstellung einzelner Wolken zu entwickeln, das mithilfe von wenigen einstellbaren Parametern die meisten typischen Wolkenarten abdeckt. Somit soll es möglich werden, innerhalb von wenigen Minuten große Szenen mit Wolken zu bereichern. Kern der Arbeit ist die Generierung der volumetrischen Daten, die dann von einer Renderengine zur Darstellung von Wolken genutzt werden können. Ein besonderer Fokus liegt auf der Anwenderfreundlichkeit. Das System soll auch für weniger technisch orientierte Nutzer intuitiv verständlich sein.

Die Umsetzung des Systems soll vor allem mit sogen. prozeduralen Texturen erfolgen, die in Kombination miteinander sehr komplexe natürliche Strukturen bilden können. Die Beschränkung auf wichtige Parameter vereinfacht hierbei auch die Animation. Mit einfachen Veränderungen der Werte können so zum Beispiel die Bewegung der Wolke im Wind oder die Bildung von Gewitterwolke dargestellt werden. Zur Erstellung eines Prototyps wird die Renderengine Cycles der Open-Source Software Blender verwendet. Mithilfe der daraus gewonnenen Erkenntnisse soll daraufhin die technische Implementierung mit der Programmiersprache OSL (Open Shading Language) erfolgen, die von den meisten gängigen Renderengines unterstützt wird.

1.3 Eingrenzung

Nur oberflächlich wird in dieser Arbeit der tatsächliche Rendervorgang von volumetrischen Materialien erläutert. Dieser Prozess wird von der jeweiligen Renderengine übernommen. Nicht behandelt wird die Darstellung von Wolken in Echtzeitanwendungen wie z.b. Unity3D oder der Unreal Engine, da sich die hier verwendete Technik noch Grundsätzlich von der in Path Tracern verwendeten Technologie unterscheidet - wobei sich das mit Echtzeit Raytracing (z.b. Nvidia RTX) und immer schneller werdenden Rauschentfernungsalgorithmen für Path Tracer, in absehbarer Zukunft ändern könnte.

Mit prozeduralen Texturen generierte Wolken können in keinem Fall den

Komplexität- und Realitätsgrad von physikalisch simulierten Wolken erreichen. Realistisches Verhalten von sich auftürmenden und von Wind und Luftströmungen geformten Wolken kann nur mit komplexen Simulationen erreicht werden. Rein mit prozeduralen Texturen erzeugte Wolken können daher nur eine ästhetisch möglichst überzeugende Annäherung an die komplexe Realität sein.

1.4 Beobachtung realer Wolken

Zur Generierung von prozeduralen Wolken ist es zunächst wichtig, die Eigenschaften realer Wolken zu beobachten und den zu behandelnden Wolkenarten (Abb. 1) zuzuordnen.

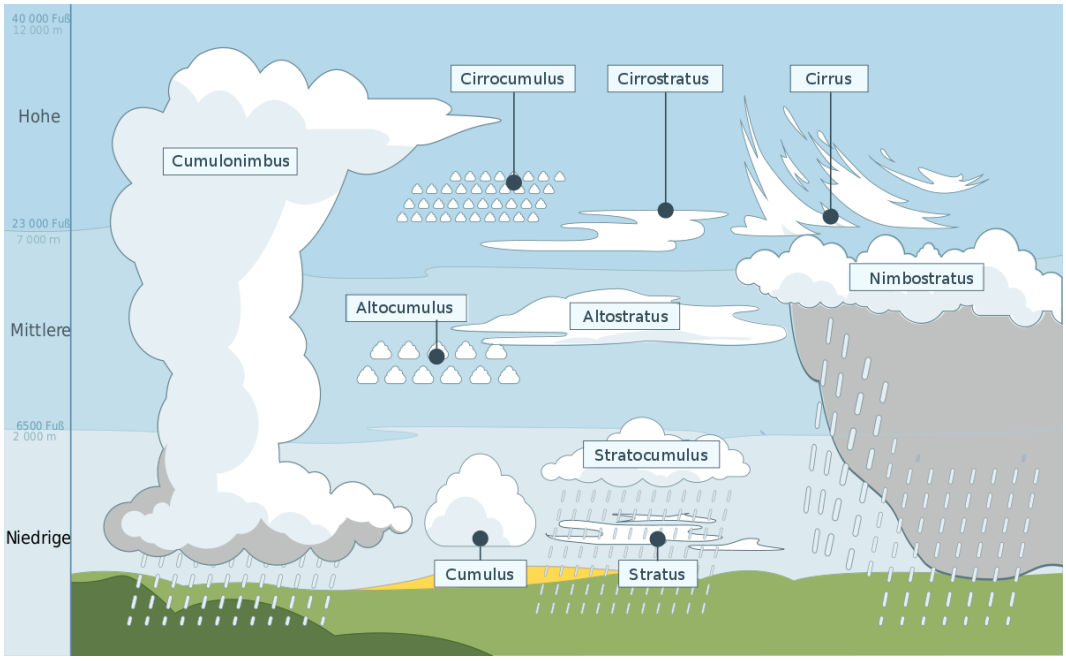

Abbildung 1: Verschiedene Wolkenarten[1]

Die unterschiedlichen Wolkenarten unterscheiden sich sowohl in Dichte, als auch in primären und sekundären Detailmerkmalen. Anhand von Referenzbildern lassen sich drei wichtige übergeordnete Merkmale im Erschei-

[1] Quelle: `de.wikipedia.org/wiki/Datei:Cloud_types_de.svg`
Urheber: Valentin de Bruyn
Lizenz: CC BY-SA 3.0 `creativecommons.org/licenses/by-sa/3.0/deed.de`

nungsbild feststellen, deren unterschiedlich starke Ausprägung die Wolkenformen unterscheidet.

1. **1. Aufbauschende Strukturen:** Sich ausbauschende und auftürmende Strukturen, die vor allem bei den große Wolkenarten entstehen und in ihren Formen Blumenkohl ähneln [NV15, 28] (Abb. 2).

Abbildung 2: Blumenkohlartige sich aufbauschende Strukturen bei Cumuluswolken[2]

2. **Fein detailierte Strukturen:** Bei Wolkenformen mit geringerer Dichte und an den Rändern von großen Wolkenformen auftretende feingliedrige Strukturen (Abb. 3).

Abbildung 3: Feine Strukturen bei Stratuswolken und an den Rändern von Cumuluswolken

3. **Wellenartige Strukturen:** Bei Cirruswolken auftretende Wellen- oder Haarartige Strukturen (Abb. 4).

[2]Quelle: `https://www.flickr.com/photos/gedankenstuecke/96292108`
Urheber: Bastian Greshake Tzovaras
Lizenz: CC BY-SA 3.0

Abbildung 4: Wellenartige Strukturen bei Cirruswolken[3]

1.4.1 Powdered Suggar Look

Zu beachten sind die dunklen Ränder der Wolkenformen, die sich vor dahinter liegenden Wolken und dem Hintergrund abheben (Abb. 2). Im der Präsentation *«The Real-Time Volumetric Cloudscapes of Horizon: Zero Dawn«* von Guerrilla Games wird dieser Effekt als Powdered Suggar Look bezeichnet, da bei Puderzucker ein ähnlicher Effekt auftritt [NV15, 59].

[3] Foto Links:
Quelle: commons.wikimedia.org/wiki/File:Cirrus_Clounds.jpg
Urheber: Nissim Angdembay
Lizenz: CC BY-SA 3.0
Foto rechts:
Quelle: en.wikipedia.org/wiki/File:Cirrus_clouds2.jpg Lizenz: CC BY-SA 3.

2 Stand der Technik

2.1 Rendering von volumetrischen Materialien in Path Tracing Renderengines

2.1.1 Path Tracing

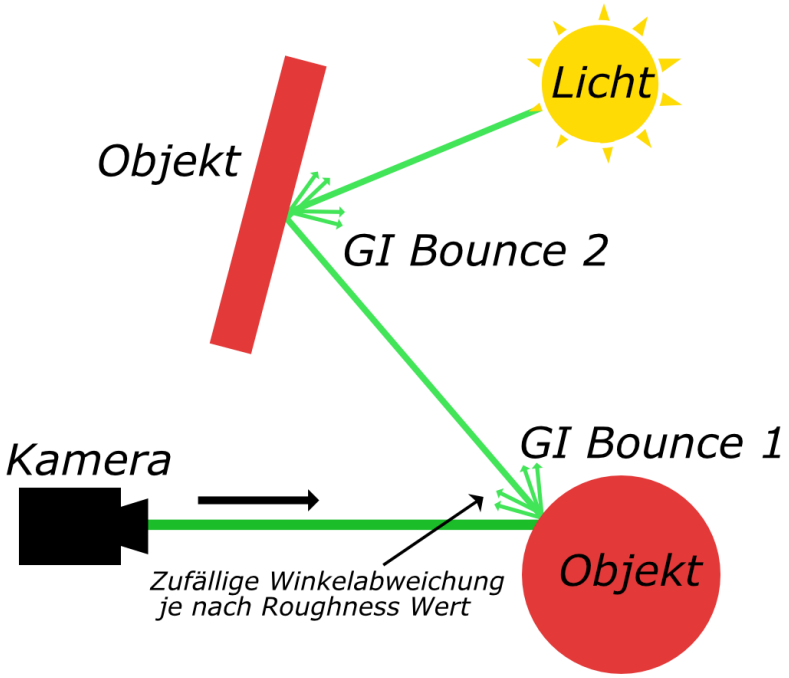

Abbildung 5: Path Tracing

Die hier behandelte und in den meisten modernen Renderengines genutzte Rendering Methode ist das sogenannte Path Tracing - einer speziellen Form von Raytracing, die 1986 zum ersten mal von James Kajiya [Kaj86] vorgestellt wurde. Bei dem am häufigsten verwendeten unidirektionalen Path Tracing, werden von der Kamera aus Strahlen (Rays) in die computergenerierte 3D Szene gesendet (Abb. 5). Sobald die Strahlen auf eine Oberfläche treffen, wird dessen Material evaluiert. Trifft der Strahl zum Beispiel auf eine raue metallische Oberfläche, wird ein weiterer Strahl in Richtung von dessen

reflektierten Ausfallswinkel gesendet. Abhängig von dessen Rauheitswert (Roughness), wird der Ausfallswinkel zufällig variiert (Abb. 6).

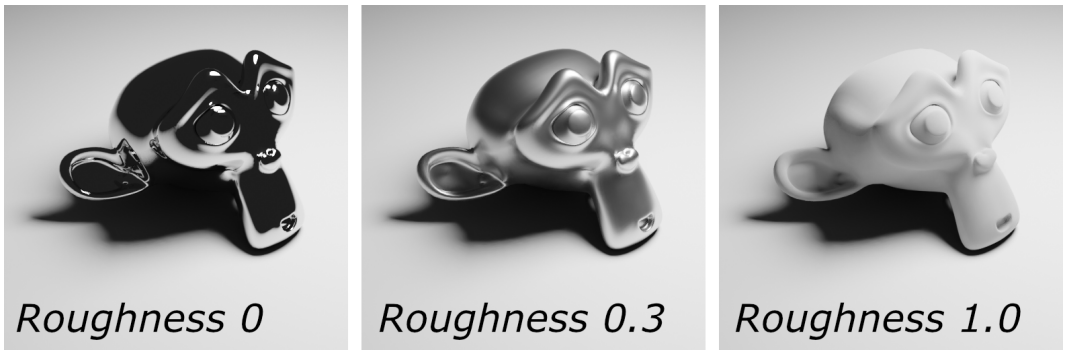

Abbildung 6: Path Tracing - Material mit unterschiedlichen Roughness Werten

Der resultierende Strahl wird nun wieder verfolgt, bis er auf eine Oberfläche trifft. Die Anzahl der sogenannten GI Bounces bestimmt hierbei, wie oft der Strahl weitergesendet wird. Die Kombination aus einem Strahl und dessen bei Treffen einer Oberfläche gesendeten "Kindstrahlen", repräsentiert einen Path (engl. Pfad), daher wird die Methode Path Tracing genannt. Auch wenn eine Lichtquelle getroffen oder ins Unendliche geht ohne eine Oberfläche zu treffen, endet der Pfad. Desto öfter die Strahlen in der Szene springen, desto eher nährt sich die indirekte Beleuchtung an das physikalisch korrekte Ergebnis an (Abb. 7).

Abbildung 7: Path Tracing mit variierender Anzahl an GI Bounces

Oft wird dabei je nach Art des Materials unterteilt und Volume Bounces (volumetrische Materialien) sind beispielsweise separat zu Glossy Bounces (Lichtreflektionen auf Oberflächen) einstellbar. (Abb. 8)

Abbildung 8: Path Tracing mit variierender Anzahl an Volume Bounces

Pro Pixel werden diese Schritte nun je nach Anzahl der eingestellten Iterationen wiederholt und das Ergebnis, abhängig von Helligkeit der getroffenen Lichtquellen und der Materialeigenchaften, interpoliert. So kann schon nach kurzer Renderzeit ein annährendes, wenn auch verrauschtes, Ergebnis erzielt werden, was Path Tracing geeignet für schnelles Arbeiten macht (Abb. 9).

Abbildung 9: Path Tracing mit variierender Anzahl an Iterationen/Samples

2.1.2 Volume Scatter

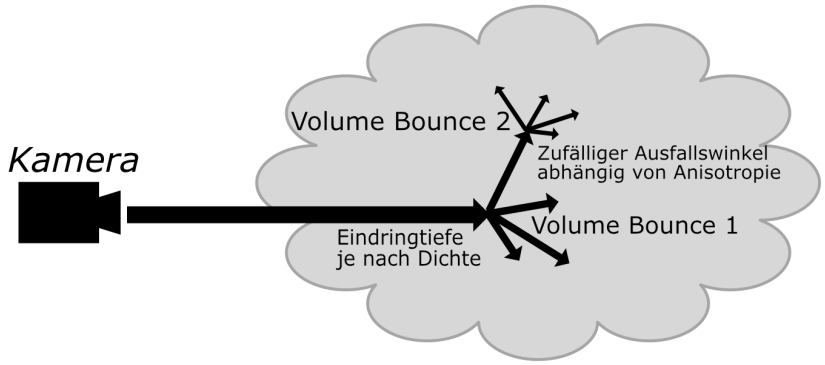

Abbildung 10: Volume Scatter

Zur Darstellung von volumetrischen Materialien, um beispielsweise Rauch, Wolken oder trübes Wasser zu simulieren, wird Volume Scattering (volumetrische Streuung) verwendet. Dabei dringen die Strahlen in das Volumen von Objekten ein, anstatt schon an der Oberfläche reflektiert zu werden. Wie weit die Strahlen eindringen können, hängt von der vorgegebenen Dichte des Materials ab (Abb. 11, 10). Innerhalb des Volumens springen die Strahlen dann je nach Anzahl der vorgegebenen Volume Bounces (Abb. 2.1.1) hin und her. Je nach Wert der Anisotropie Variablen, weicht der Winkel der springenden Strahlen dabei zum Einfallswinkel ab (Abb. 10).

Abbildung 11: Volumetrisches Material mit variierender dichte

Bei positiven Anisotropiewerten springen die Strahlen in Richtung des ankommenden Strahls weiter, bei Anisotropie 0 springen sie vollständig zu-

fällig in alle Richtungen und bei negativen Anisotropiewerten springen sie entgegengesetzt des ankommenden Strahls (Abb12).

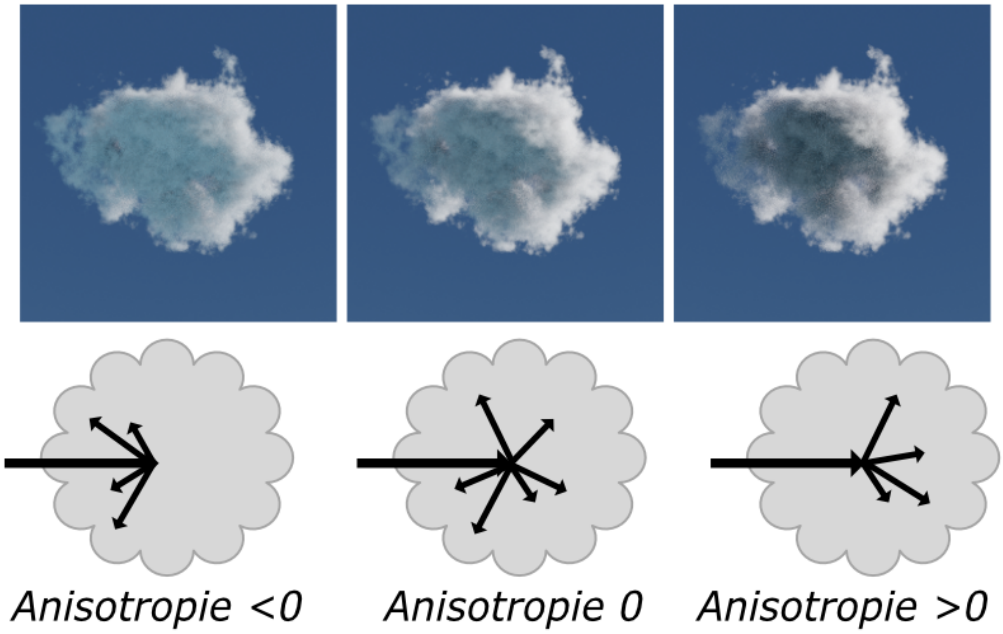

Abbildung 12: Verhalten der Strahlen bei unterschiedlichen Anisotropiewerten

Mit Renderman ist es möglich, Anisotropiewerte für den ersten Volume Bounce separat zu den folgenden einzustellen. Für physikalisch korrekekte Wolken eignen sich hier wohl die Werte -0.550 für den ersten- und 0.824 für die folgenden Volume Bounces [ren]. Leider steht diese Funktionalität in der für diese Arbeit benutzten Renderengine Cycles nicht - und generell nur in wenigen Renderengines zur Verfügung. Physikalisch realistische Beleuchtung von Wolken, die zum Beispiel den Powdered Suggar Effekt (1.4.1) erzeugt ist demnach besser in Renderengines umzusetzen, die diese Funktionalität unterstützen.

2.2 Techniken zur Generierung von Wolken

Zur Generierung von Wolken in der Computergrafik gibt es verschiedene Ansätze für unterschiedliche Anwendungsgebiete.

2.2.1 Simulation

Der physikalisch korrekteste Ansatz zur Generierung von Wolken ist es, simulierte Rauch- und Windsimulationen zu verwenden, um Wolken natürlich entstehen zu lassen und zu animieren. Die entstandenen volumetrischen Daten werden dann als VDB Datei ausgegeben und lassen sich dann in allen Renderengines mit VDB Unterstützung rendern. Diese Methode kann zum Beispiel in der 3D-Software Houdini von SideFX eingesetzt werden. Sie liefert qualitativ die hochwertigsten Ergebnisse, ist jedoch auch aufwendig umzusetzen und benötigt viel Rechenleistung.

2.2.2 Point-Cloud-Data Generierung

Eine andere Möglichkeit ist es, die Grundform der Wolken als Drahtgittermodell zu modellieren und daraus volumetrische Daten, d.h. eine Ansammlung von Punkten im Raum zu generieren. Die generierte Point-Cloud-Daten können z.b. als VDB Datei gespeichert werden und in Renderengines mit VDB Unterstützung gerendert werden. Diese Methode kann zum Beispiel in Houdini oder mit der Open-Source-Software Blender umgesetzt werden. Häufig dabei ist die Verwendung der prozeduralen sogen. Perlin-Noise Textur – die auch für diese Arbeit eine wichtige Rolle spielen wird – um die Dichte des volumetrischen Materials im Raum zu kontrollieren und somit wolkenähnliche Strukturen zu erzeugen. Diese Methode ist schnell umsetzbar, der Nachteil ist jedoch die fehlende Kontrolle über die Wolkenform.

2.2.3 Mesh Displacement

Eine in Online-Ressourcen oft verwendete Methode zur prozeduralen Generierung von Wolken ist die Anwendung einer prozeduralen Textur zur Verzerrung eines Grundmeshes. Dann wird dem Objekt ein homogenes volumetrisches Material (volumen mit konstanter Dichte) gegeben. Zwar einfach anzuwenden, bietet diese Methode jedoch einige Probleme. Zunächst ist mit dieser Technik nur eine Verzerrung des Meshes in Richtung der Normalen

der Vertices möglich. Somit sind Überhänge und sich abspaltende Wolkenfetzen nicht möglich. Überhänge ließen sich mit einer dreikanäligen Displacement Map und Vector Displacement umsetzen. Dabei wird jedem Kanal jeweils X, Y und Z zugewiesen und die Vertices entsprechend der Werte relativ zur Normale verschoben. Die Technik erfordert ein hoch aufgelöstes Mesh und ist an dessen Topologie gebunden. Adaptive Subdivision, d.h. die unterschiedlich starke Unterteilung des Meshes abhängig vom Abstand zur Kamera, kann beim letzten Punkt Abhilfe verschaffen. Zudem entsteht durch das homogene Volumen ein unnatürlich harter Übergang zwischen Wolke und Umgebung (siehe Abb. 13). Dieses Problem könnte jedoch mit Mesh Proximity (2.2.4) gelöst werden.

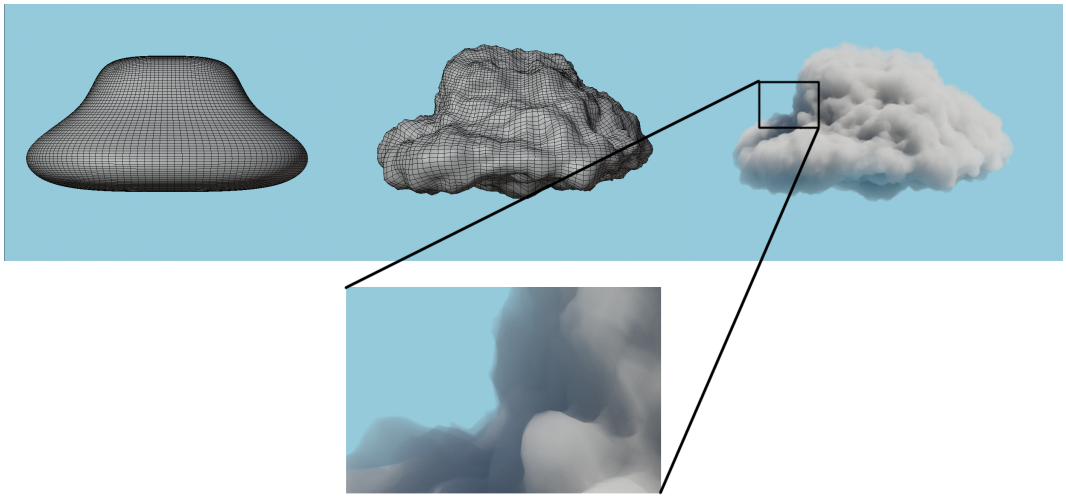

Abbildung 13: Wolkengenerierung durch prozedurale Verzerrung eines Grundmeshes (hier Perlin FBM Noise 3.3.2)

2.2.4 Mesh Proximity

Mesh Proximity [RS] bezeichnet hier den Abstand eines Punktes innerhalb eines Volumens zum nächsten Punkt auf der Oberfläche (Abb. 14). Mit OSL werden dazu von jedem, vom Path Tracer getroffenen, Punkt aus innerhalb eines Volumens mehrere Rays (je nach Anzahl der iterationen) in zufällige Richtungen gesendet. Für jeden Ray wird die zurückgelegte Distanz zur Oberfläche gemessen und mit dem niedrigsten Wert so annähernd der Abstand zur jener bestimmt.

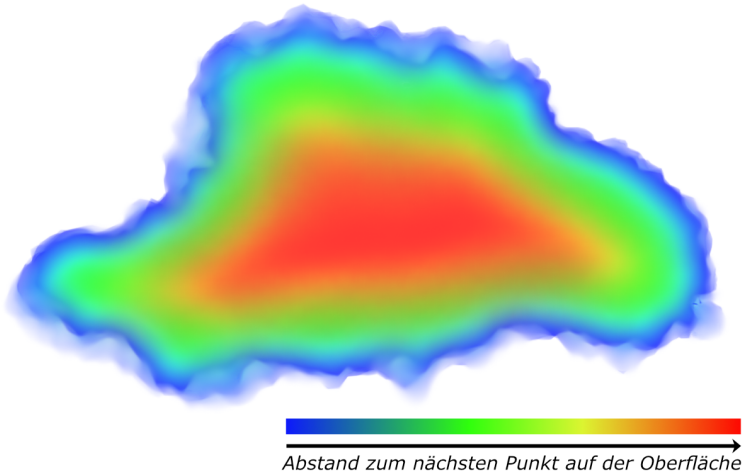

Abbildung 14: Mesh Proximity - Querschnitt durch Modell aus (Abb. 13)

Wird der resultierende Wert als Multiplikator für die Dichte verwendet, kann damit das mit Mesh Displacement (2.2.3) auftretende Problem des harten Übergangs zur Umgebung der Wolken gelöst werden. Quadriert anstatt linear angewendet ergibt sich ein weicherer Dichteabfall. Da so viele Rays gesendet werden müssen, ist diese Methode jedoch sehr langsam und nicht praktikabel für diese Arbeit. Schon bei nur 3 Iterationen stieg die Renderzeit bei sonst identischen Einstellungen von 4:12 Minuten auf 43:20 Minuten um das mehr als zehnfache an (Abb. 15).

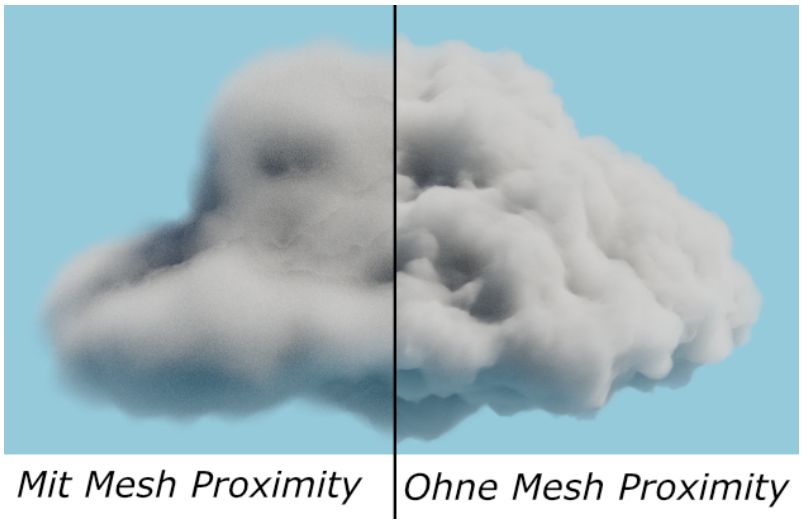

Abbildung 15: Wolke aus Abb. 13 mit quadrierter Mesh Proximity als Multiplikator für die Dichte. *AMD Ryzen 5 1600, Cycles Renderer, Auflösung 800*500, 1000 Samples, 2 Volume Bounces, Volume Step Size 0.15*

2.2.5 Volume Displacement

Eine mit der Arnold Renderengine mögliche Technik ist sogenanntes Volume Displacement [arnb], bei dem die Dichteinformationen eines VDB Volumens abhängig von einer Textur im Raum versetzt werden. Diese Methode ist sehr vielversprechend, da jedes vorher in ein VDB Volumen umgewandelte Objekt mit jeder beliebigen Textur angepasst werden kann. Nur mit OLS lässt sich diese Technik jedoch leider nicht umsetzen.

3 Implementierung mit OSL

OSL (Open Shading Language) ist eine Programmiersprache von Sony Pictures Imageworks mit C ähnlicher Syntax zur Programmierung von Shadern in Renderengines. OSL eignet sich neben der Programmierung von BSDF Closures auch hervorragend zur Generierung von prozeduralen Mustern [Gri18, 1]. In Pathtracing Renderengines wird dabei für jeden geworfenen Ray (2.1.1) der Shader an der getroffenen Stelle auf der Oberfläche des Materials evaluiert. Bei volumetrischen Materialien wird der Shader am Punkt der getroffen Stelle innerhalb des Volumens evaluiert. OSL Scripte lassen sich in Renderengines einbinden, indem eine Liste an Ein- und Ausgabewerten definiert wird, die dann auf einem Shadernode dargestellt wird (Abb. 16)

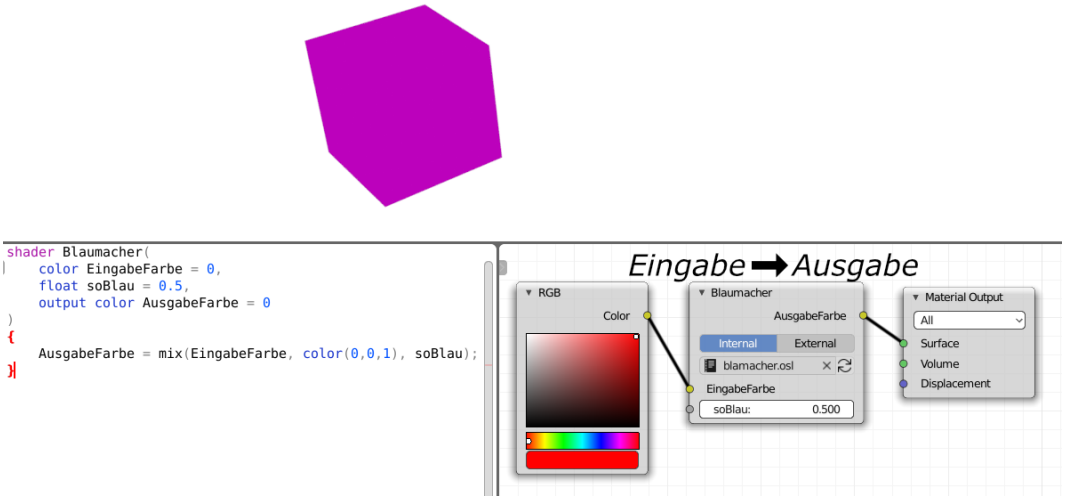

Abbildung 16: Einfaches OSL Script und dessen Repräsentation als Shadernode in Blender

3.1 Grundform

Da die Bestimmung der Wolkenform abhängig vom Mesh mit OSL aufgrund der Performanceschwierigkeiten mit Mesh Proximity ausgeschlossen wurde

(2.2.4), muss auch die Grundform der Wolke prozedural generiert werden. Als Grundobjekt wird hier ein einfacher Quader verwendet, der Höhe, Breite und Tiefe der Wolke eingrenzt. Daraus muss die Grundform der Wolke definiert werden, mit der sich alle typischen Wolkenarten generieren lassen.

3.1.1 Ellipsoid

Als Startpunkt soll eine Kugel innerhalb des Volumens angegeben werden, innerhalb der die Dichte des Volumens größer 0 ist. Am besten eignet sich hierfür die Berechnung des invertierten Abstands zum Objektmittelpunkt als Dichte. Da die Form bei Änderung der Quadermaße auch zum Ellipsoid werden soll, werden hier nicht die Weltkoordinaten verwendet, sondern die mit OSL aufrufbaren automatisch generierten Texturkoordinaten. Diese verwenden die äußeren Grenzen des Objekts, um Texturkoordinaten zu generieren, die unabhängig von dessen Skalierung immer von $T_x = T_y = T_z = 0$ in der Ecke unten links, bis zu $T_x = T_y = T_z = 1$ in der Ecke oben rechts des Grundobjekts reichen (Abb. 17).

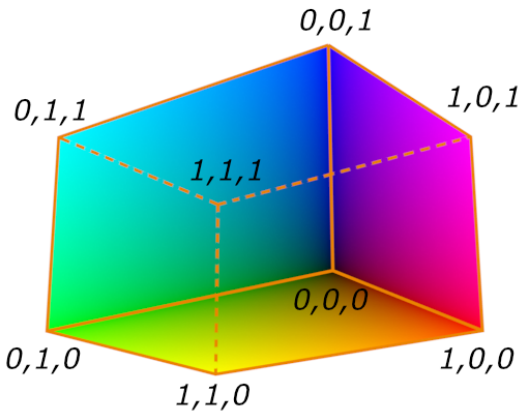

Abbildung 17: Generierte Texturkoordinaten. X,Y,Z Werte als Rot, Grün, Blau dargestellt.

Für die Berechnung des Abstands zur Objektmitte, muss zunächst der Ursprung des Koordinatensystems in die Selbige verschoben werden. Dazu werden die Texturkoordinaten T am Punkt P auf allen Achsen mit 2 multipli-

ziert und 1 subtrahiert: $T'_{(P)} = 2T_{(P)} - (1, 1, 1)$.

Es ergibt sich ein Koordinatensystem von $T_x = T_y = T_z = -1$ in der Ecke unten links, bis zu $T_x = T_y = T_z = 1$ in der Ecke oben rechts. Der Ursprung $T_x = T_y = T_z = 0$ liegt im Objektmittelpunkt (Abb. 18).

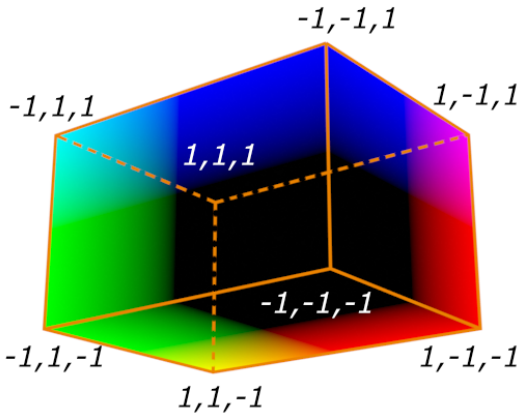

Abbildung 18: Veränderte generierte Texturkoordinaten. X,Y,Z Werte als Rot, Grün, Blau dargestellt.

Im Anschluss werden die Texturkoodinaten mit einem Skalierungsfaktor multipliziert. Für später folgende Schritte ist es wichtig, dass die Größe der Wolke nicht nur über die Skalierung des Grundobjektes beeinflusst werden kann: $T' = T * s$

Somit ergibt sich für die Dichte D an jedem Punkt P innerhalb des Grundobjekts der invertierte Summenwert der drei quadrierten Kanäle der generierten Texturkoordinaten T. Da der genaue Abstand nicht benötigt wird, sondern nur ein Gradient vom Objektmittelpunkt aus, ist es nicht erforderlich die Wurzel des Summenwerts zu berechnen.

$$D_{(P)} = 1 - (T_x^2 + T_y^2 + T_z^2) \tag{3.1}$$

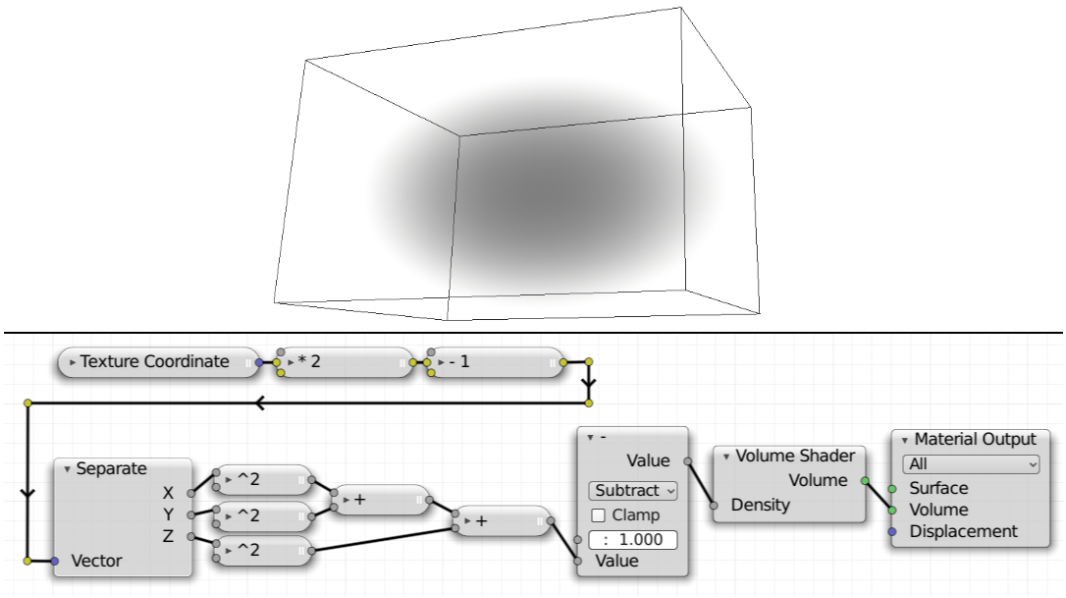

Abbildung 19: Grundformfunktion in Blender mit Cycles Nodes für die Erstellung eines Prototyps

3.1.2 Formgradient mit Exponentialfunktion

Im nächsten Schritt soll es möglich sein, die Form der Wolke entsprechend des Wolkentyps anzupassen. Eine einfache Möglichkeit hierzu ist die Manipulation der Z Komponente innerhalb der Ellipsoidfunktion (3.1.1). Einige Variationen lassen sich schon durch eine einfache Exponentialfunktion mit Z und einer einstellbaren Variable erreichen. Dazu wird vor der Anpassung des Koordinatensystems in 3.1.1 die Z Komponente T_z um einen Exponenten S Shape (engl. Form) potenziert: $T'_z = T_z^S$

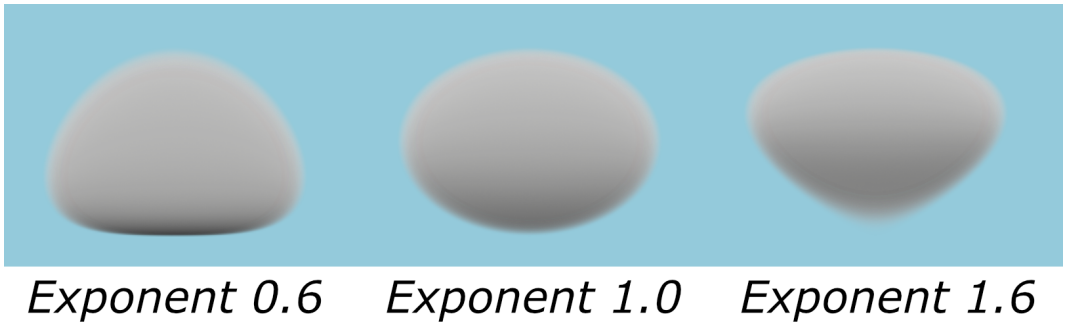

Abbildung 20: Formvariation mit verschiedenen Exponenten für T_z

Diese Methode hat den Vorteil, dass sie einfach zu bedienen ist, da nur

ein einziger Parameter benötigt wird. Komplexere Formen, die z.b. bei Cumulonimbuswolken (1.4) auftreten, lassen sich damit jedoch nicht umsetzen.

3.1.3 Formgradient mit Gradationskurven

Um die Wolkenform genauer anzupassen soll es möglich sein, mit einer Gradationskurve einen Querschnitt für den Rotationskörper zu definieren. Hierfür wird T_z als Eingabe für eine Gradationskurve genutzt und dessen Ausgabe S_{T_z} folgendermaßen in der Ellipsoidfunktion (3.1.1) anstelle von T_z^2 eingesetzt:

$$D_{(P)} = 1 - (T_x^2 + T_y^2 - S_{T_z} + 1) \tag{3.2}$$

Nun kann der Nutzer die Wolkenform mithilfe einer Gradationskurve die Wolkenform genau kontrollieren (Abb. 21). OSL bietet keine Funktion, Gradationskurven als Benutzeroberfläche in der entsprechenden 3D Anwendung darzustellen. Hierzu müssen die Möglichkeiten der jeweiligen Renderengine genutzt werden. Alternativ kann für Renderengines, die diese Funktionalität nicht aufweisen, ein OSL Script implementiert werden, dass für die relevanten Wolkenarten entsprechende Gradienten ausgibt oder eine Bildtextur verwendet werden.

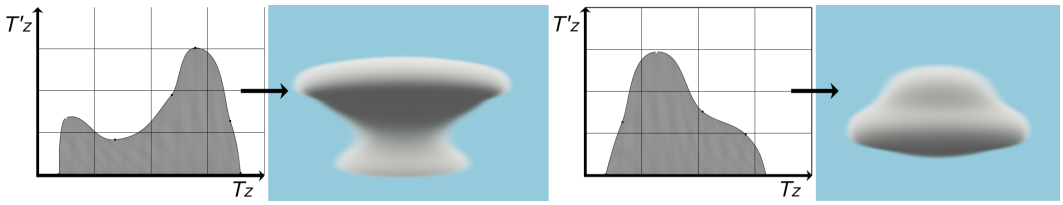

Abbildung 21: Gradationskurven werden genutzt, um Rotationskörper zu definieren

3.2 Verzerrungsmethode

Nun da die Grundform der Wolke generiert wurde, werden prozedural generierte Details hinzugefügt. Zunächst muss hierfür die in 3.1.3 definierte

Grundformfunktion um eine Variable O Offset (engl. Versatz) erweitert werden, die die Verzerrung dieser anhand einer Graustufentextur erlaubt. Dafür wird wird der Verzerrungsfaktor O am Punkt P in die Grundformfunktion an Stelle der 1 am Ende der Funktion eingesetzt. Zu beachten hierbei ist, dass die Verzerrungstextur O am Punkt P - also am Weltkoordinatensystem angegeben wird, während die generierten und in 3.1.1 angepassten Texturkoordinaten T die Form des Rotationskörpers kontrollieren. Dies ist notwendig, damit die prozedurale Textur durch Skalierung des Grundobjektes nicht so wie der Rotationskörper in die Länge gezogen wird.

$$D_{(P)} = 1 - (T_x^2 + T_y^2 - S_{T_z} + O_{(P)}) \tag{3.3}$$

Damit der Verzerrungsfaktor auch ohne Manipulation der Textur kontrolliert werden kann, wird eine Mixfunktion eingebaut, die je nach Wert der Variablen F Factor zwischen 1.0 und der Verzerrungstextur O mischt:
$O_{(P)} * F + 1 - F)$

Eingesetzt in die Grundformfunktion und gekürzt ergibt sich daraus schließlich folgende Formel für die Dichte der Wolke:

$$D_{(P)} = -O_{(P)} * F + S_{T_z} + F - T_x^2 - T_y^2 \tag{3.4}$$

In OSL lässt sich diese Formel sehr einfach als Funktion implementieren (Abb. 22).

```
1 float shape(
      point T,
3     float shapeGradient,
      float offsetMap,
5     float offsetFac)
  {
7     T = 2*T - 1;
      float density = -offsetMap * offsetFac + shapeGradient
9                     + offsetFac - T[0]*T[0] - T[1]*T[1];
      return density;
11 }
```

Abbildung 22: Implementierung der Grundformfunktion mit OSL

3.3 Verzerrungstextur

Für die Verzerrungstextur O wird nun eine Textur gesucht, die das tatsächliche Erscheinungsbild von Wolken widerspiegelt. Die Textur soll für jeden Punkt P innerhalb des Volumens variieren und daher dreidimensional sein. Da in der Grundformfunktion (3.1) Ursprünglich an der Stelle von $O_{(P)}$ eine 1.0 stand, sollte Textur gleichermaßen um den Wert 1.0 variieren und beispielsweise Werte von ca. 0.5 bis 1.5 ausgeben. So soll sichergestellt werden, dass die Verzerrungstextur die insgesamte Skalierung der Wolke nicht beeinflusst. Um die Nutzererfahrung zu verbessern, kann jedoch auch eine Textur in der Luminanzreichweite von 0.0 bis 1.0, zu der intern 0.5 addiert wird, verwendet werden.

3.3.1 Simplex/Perlin Noise

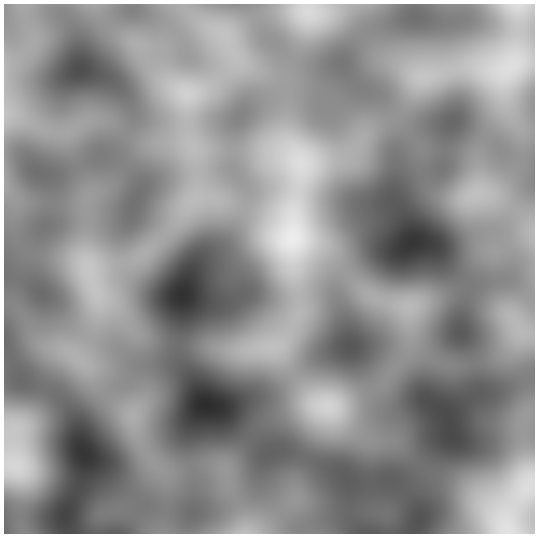

Abbildung 23: Perlin/Simplex Noise

Ein offensichtlicher Kandidat für die Detailstrukturen der Wolken ist die wohl bekannteste prozedurale Textur in der Computergrafik - Perlin-Noise (Abb. 23). Dieser Algorithmus wurde 1997 von Ken Perlin für die Computergenerierten Bilder im Film Tron entwickelt und gewann dafür einen Oscar. Die 2001 von Ken Perlin veröffentlichte verbesserte Version des Algorithmus erhielt den Namen Simplex Noise [Per01] - die beiden Begriffe werden heutzutage jedoch häufig austauschbar benutzt und meist ist mit Perlin Noise auch die verbesserte Version von 2001 gemeint. Simplex-Noise kann sehr gut überall angewendet werden, wo organische, ungeordnet erscheinende Materialien, wie z.b. Stein oder Gras benötigt werden. Materialien können mithilfe von Simplex-Noise zufällig und doch natürlich erscheinende Variationen erhalten. Auch bei prozedural generierten Landschaften findet Simplex-Noise Anwendung (Abb. 24).

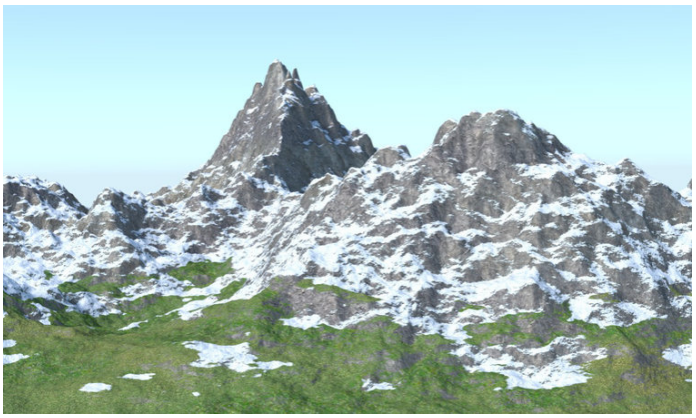

Abbildung 24: Mithilfe von Simplex-Noise rein prozedural generierte und texturierte Landschaft

Eine für diese Arbeit sehr wichtige Eigenschaft von Simplex Noise ist, dass sich der Algorithmus auch auf mehr als zwei Dimensionen anwenden lässt und somit für volumetrische Texturen geeignet ist. In OSL ist eine vierdimensionale Implementierung von Simplex-Noise bereits enthalten und muss daher nicht manuell implementiert werden. Die vierte Dimension dient hierbei meistens als Zeitvariable.

Wird reines Perlin Noise als Variable für OffsetMap O in die Grundformfunktion (22) eingesetzt, erhält man nun folgendes Ergebnis (Abb. 25):

Abbildung 25: Wolke mit Perlin Noise als Verzerrungstextur

Wie man sehen kann, erschein die Wolke noch nicht sehr detailiert. Um

dieses Problem zu lösen, kommt im nächsten Schritt (3.3.2) Fractal Brwonian Motion zum Einsatz.

3.3.2 Fractal Brownian Motion

Um der Simplex-Noise Funktion mehr Detail zu verleihen, wird Fractal Brownian Motion (FBM) oder auch Gebrochene Brownsche Bewegung angewendet. Dabei werden je nach Anzahl der Iterationen oder auch Oktaven mehrere Ebenen von Simplex Noise addiert. Skalierung und Amplitude (Luminanz) wird dabei von Ebene zu Ebene variiert. Der Multiplikator, um den die Skalierung der Textur dabei je weiterer Ebene verändert wird, wird Lacunarity genannt. Der Multiplikator, um den die Luminanz je Ebene Verändert wird als Gain bezeichnet. Standardmäßig werden dabei meistens die Werte Lacunarity=2 und Gain=0.5 verwendet.

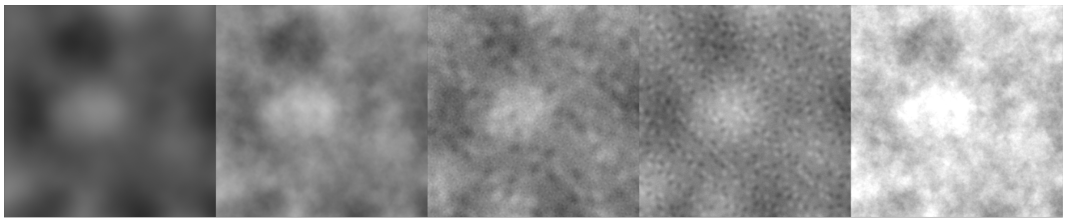

Abbildung 26: FBM Noise mit verschiedenen Werten für Lacunarity und Gain bei 5 Oktaven

Für die Implementierung in OSL wird dabei in einer For-Schleife pro Oktave eine Noise Textur dem Rückgabewert v hinzugefügt. Dabei werden in der Schleife Textukoordinaten und Amplitude entsprechend mit Lacunarity und Gain multipliziert. Der Parameter t eintspricht hier der vierten Dimension der Simplex-Noise Funktion (3.3.1) und kann als Zeitvariable behandelt werden.

```
float fbm(point texCoord, int octaves, float lacunarity, float
    gain, float t) {
    float v = 0;
    float amplitude = 0.5;

    for (int i = 0; i < octaves; i++) {
        v += amplitude * noise("simplex", texCoord, t);
        texCoord *= lacunarity;
        amplitude *= gain;
    }
    return v;
}
```

Abbildung 27: FBM Noise mit OSL

Wird FBM Simplex Noise nun als Variable für OffsetMap O in die Grundformfunktion 22 eingesetzt, erhält man folgendes Ergebnis (Abb. 28):

Abbildung 28: Wolke mit FBM Simplex Noise als Verzerrungstextur

Die in 1.4 behandelten feinen Strukturen der Wolke sind schon gut erkennbar. Die Form wirkt jedoch noch zu symmetrisch und es fehlen jedoch noch die für z.b. Cumuluswolken typischen Blumenkohlartigen Strukturen. Für diese bietet sich die Nutzung einer Worley- Noise Textur an.

3.3.3 Worley Noise

Für die Erzeugung der sogenannten Worley-Noise Textur, muss zunächst ein Voronoi Diagramm erzeugt werden. Für die Erzeugung des Voronoi Diagramms wird zunächst an jedem Punkt P durch Abrundung eine Position angegeben und diese dann auf allen drei Achsen um einen zufälligen Faktor versetzt. So entsteht ein Muster aus zufällig angeordneten Zellen (Abb. 30) Worley-Noise entsteht nun durch die Messung des Abstandes zur nächsten Zelle eines Voronoi Diagramms für jeden gegebenen Punkt.

```
float voronoi(point tx) {
    point thiscell = floor(tx);
    float dist2nearest = 1000;
    int i,j,k;
    for(i = -1; i <= 1; i += 1)
       for(j = -1; j <= 1; j += 1)
           for(k = -1; k <= 1; k += 1) {
              point testcell = thiscell + vector(i,j,k);
              point pos = testcell + noise("cell", testcell) -
                  0.5;
              float dist = distance(pos, tx);
              if(dist < dist2nearest)
                 dist2nearest = dist;
              }
    return dist2nearest;
}
```

Abbildung 29: Implementierung von Voronoi Textur (Distanz zur nächsten Zelle) in OSL nach [Kes]

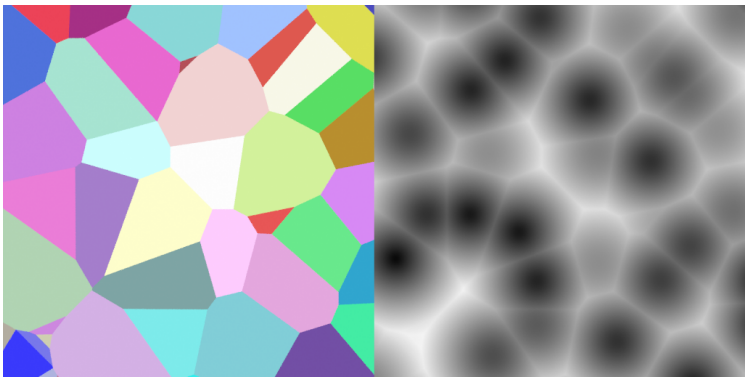

Abbildung 30: *links:* Voronoi Diagramm mit zufällig eingefärbten Zellen. *rechts:* Abstand der Zellen des selben Voronoi Diagramms als Luminanz dargestellt.

Der gemessene Abstand wird invertiert und mit der smoothstep Funkti-

on weicher dargestellt. `smoothstep(min, max, v)` ist eine in Shadersprachen wie OSL, RSL (Renderman Shading Language) und GLSL (OpenGL Shading Language) vordefinierte Funktion, die einen Eingabewert `v` auf eine Reichweite zwischen `min` und `max` weich abbilden kann. Wie bei Perlin Noise, kann auch zu Worley Noise mehr Detail hinzugefügt werden, indem mehrere Ebenen mit dem Fractal-Brownian-Motion Ansatz (3.3.2) übereinandergelagert werden (Abb. 31). Auch hier gehen zusätzliche Ebenen jedoch mit längeren Renderzeiten einher, die sich vor allem bei volumetrischen Texturen aufgrund der vielen Shaderevaluationen bemerkbar macht.

Abbildung 31: Worley Noise mit 1, 2 und 5 FBM Oktaven bei 2.0 Lacunarity und 0.5 Gain

Als Verzerrungstextur für die Grundform angewendet, ergibt sich folgende Wolke (Abb. 32)

Abbildung 32: Wolke mit FBM Worley Noise als Verzerrungstextur

3.3.4 Perlin-Worley Noise

Wie in [NV15, P 29] beschrieben, können nun Eine Simplex FBM und eine Worley FBM Textur miteinander multipliziert werden. Über Mixfunktionen im OSL Script kann der Einflussfaktor jeder Textur separat eingestellt werden. Somit lassen sich nun auch unförmigere, aufgebauschte Wolken (1.4) generieren (Abb. 33)

Abbildung 33: Wolke mit Perlin-Worley Noise als Verzerrungstextur

3.3.5 Sinustextur

Nun soll die Darstellung der wellenartigen Strukturen, die zum Beispiel in Cirruswolken auftreten (1.4) ermöglicht werden. Hierzu wird mit der in OSL eingebauten `sin(x)` Funktion der Sinus der X Komponente der Texturkoordinaten T_x berechnet. Letztere wird zuvor mit einer Simplex-Noise Funktion multipliziert, um der Textur Variation zu geben. Über die Variable Wave wird zwischen Worley-Noise und der Sinustextur gemischt (Abb. 34).

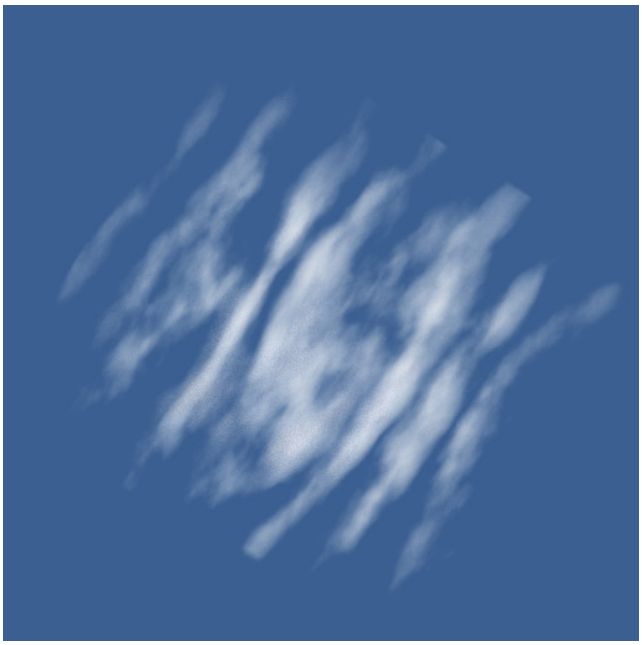

Abbildung 34: Cirruswolke mit wellenartiger Struktur durch Anwendung der Sinustextur

4 Benutzeroberfläche

4.1 Modularität oder einzelner Node?

Der offensichtlichste Ansatz zur Gestaltung der Benutzeroberfläche des Systems ist es, die komplette Funktionalität des Wolkengenerators in ein einzelnes OSL Script zu packen und als lange Liste von Parametern auf der Benutzeroberfläche des Nodes anzuzeigen. Es hat jedoch Vorteile, Shader in mehrere Nodes nach Funktionalität geordnet aufzuteilen. Der hier verwendete Ansatz ist es, den Shader in zwei Nodes aufzuteilen. Ein Node für die Grundform (3.1) und eine Node für die Textur. Zum Einen ist es dem Nutzer so möglich, die Texturen vor Verbindung des Nodes mit dem Grundform-Node noch mit benutzerdefinierten Nodes zu manipulieren, zum Anderen ist es so auch möglich, native Texturen der jeweiligen Renderengine als Verzerrungstextur zu nutzen.

4.2 Parameterliste

Das entwickelte System hat sehr viele Variablen, die das Erscheinungsbild der resultierenden Wolke verändern. Nicht alle davon müssen jedoch vom Anwender selbst kontrolliert werden können. Um die Komplexität der Benutzeroberfläche des Shaders übersichtlich und Anwenderfreundlich zu gestalten, sollte die Anzahl der vom Nutzer kontrollierbarer Parameter so niedrig wie möglich, jedoch so hoch wie nötig gehalten werden. Hierzu muss evaluiert werden, welche Parameter des Shaders dem Benutzer als Regler zur Verfügung gestellt werden und welche für Werte ein konstanter Wert oder ein aus anderen Parametern errechneter Wert hinreichend ist. Beispielsweise kann der Lacunarity Wert der FBM-Simplex-Noise Funktion fest auf 2.0 gesetzt werden, da Veränderung dieses Wertes beim Test keinen hinreichenden Mehrwert bot. Die Oktaven der FBM-Worley-Noise Funktion wer-

den fest auf 2 gesetzt, da für feinere Details die Simplex-Noise Funktion sorgt.

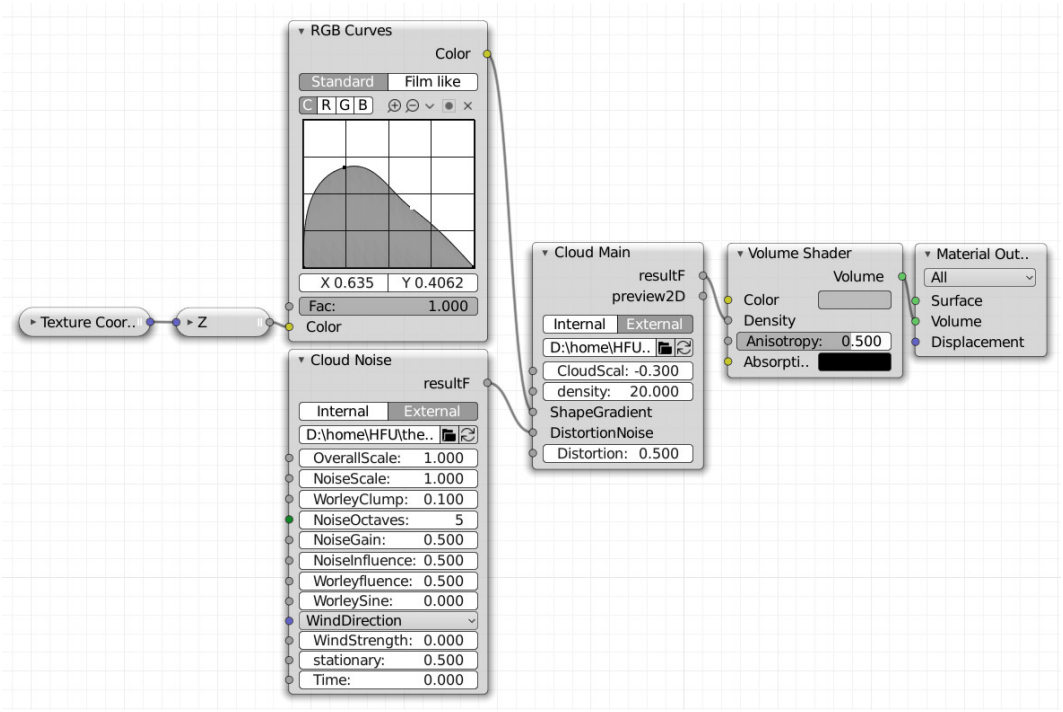

Abbildung 35: Benutzeroberfläche der OSL Nodes in Blender

Um die Nutzererfahrung zu verbessern, sollten möglichst viele Parameter auf einer Reichweite von 0.0 bis 1.0 dargestellt werden. Bei Parametern, die keine Grenze nach oben haben, soll die manuelle Eingabe von Werten über 1.0 zum erreichen zwar unnatürlicher, aber eventuell künstlerisch erwünschter Effekte trotzdem möglich sein. Neben der konsistenteren Nutzererfahrung, ermöglicht die Projektion der Werte auf einer Reichweite von 0-1 auch die einfache Kontrolle der Parameter mit Graustufentexturen ohne vorherige Umrechnung. Die OSL Implementierung von Cycles ermöglicht bisher noch keine Festlegung von Maximal- und Minimalwerten zur Darstellung eines Parameters als Schieberegler. In diesem Beispiel muss diese Funktionalität daher innerhalb von Blender mithilfe einer Nodegroup implementiert werden. Andere Renderengines wie z.b. Arnold erlauben es, diese Funktionalität direkt in einem OSL Script zu implementieren. Kondensiert auf die wichtigen Parameter ergibt sich nun folgende Liste an Parametern für den Cloud Noise Node:

- **Overall Scale:** Skaliert die Textur als Ganzes.

- **Worley Influence:** Kontrolliert, wie stark der Einfluss der Worley-Noise (3.3.3) Komponente auf die Textur ist.

- **Worley Clump:** Umbenannter Gain Faktor der FBM-Worley Funktion - bauscht die Wolke auf.

- **Wave:** Mischt die Sinus Textur für einen Wellenartigen Effekt ein.

- **Noise Influence:** Kontrolliert den Einfluss der FBM-Simplex-Noise Textur (3.3.2) auf die Textur.

- **Noise Scale:** Multiplikator für die Skalierung der FBM-Simplex-Noise Textur.

- **Noise Detail:** Oktaven der FBM-Simplex-Noise Textur.

- **Noise Whisp:** Umbenannter Gain Wert der FBM-Simplex-Noise Textur.

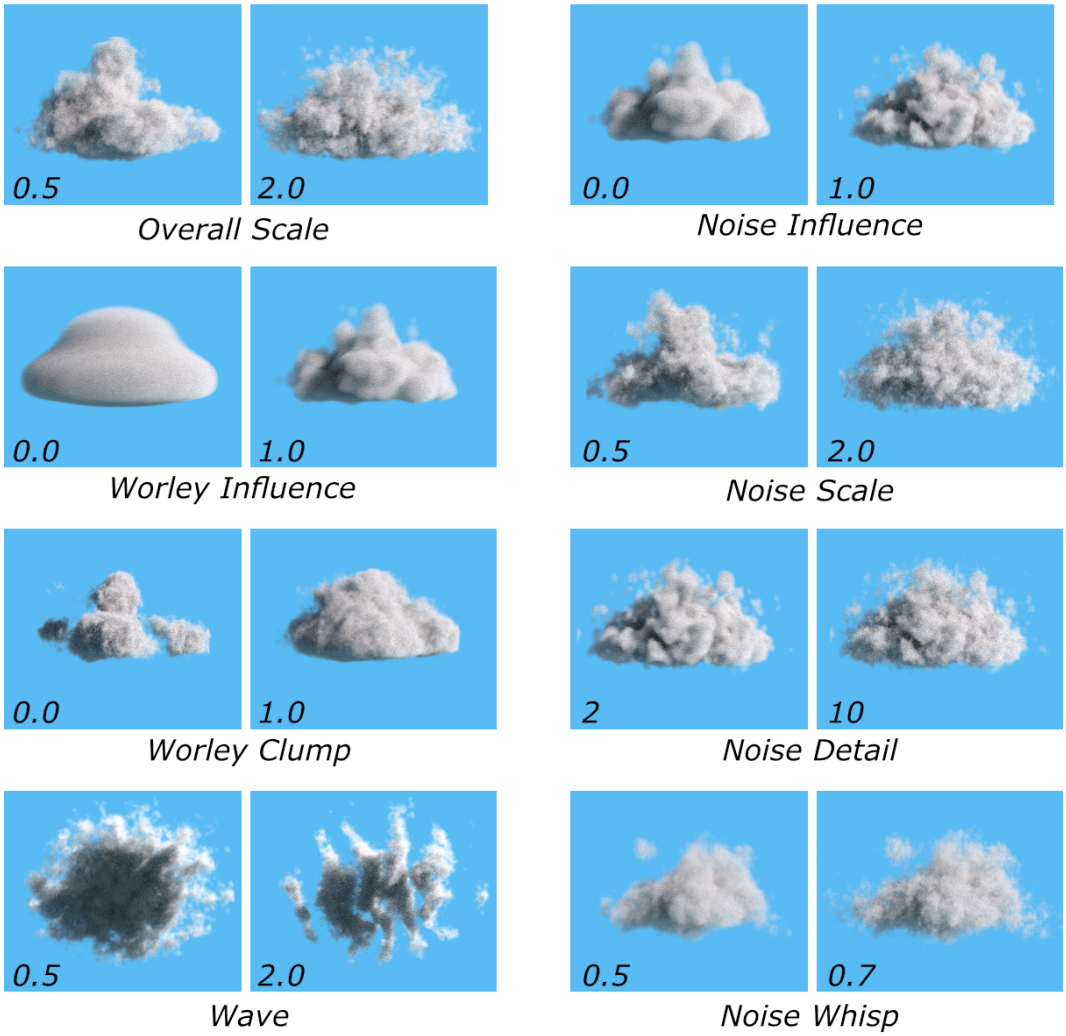

Abbildung 36: Einfluss der Parameter auf das Erscheinungsbild der Wolken

Für den Cloud-Main Node ergeben sich folgende Parameter:

- **Cloud Scale:** Skalierung der Wolke innerhalb des Grundobjekts.
- **Density:** Multiplikator für die dem Volume Shader übergebene Dichte der Wolke.
- **Shape Gradient:** Eingabesockel für den Formgradienten (3.1.3).
- **Distortion Noise:** Eingabesockel für den Cloud-Noise Node (3.3).
- **Distortion:** Grad der Verzerrung durch die Verzerrungstextur.

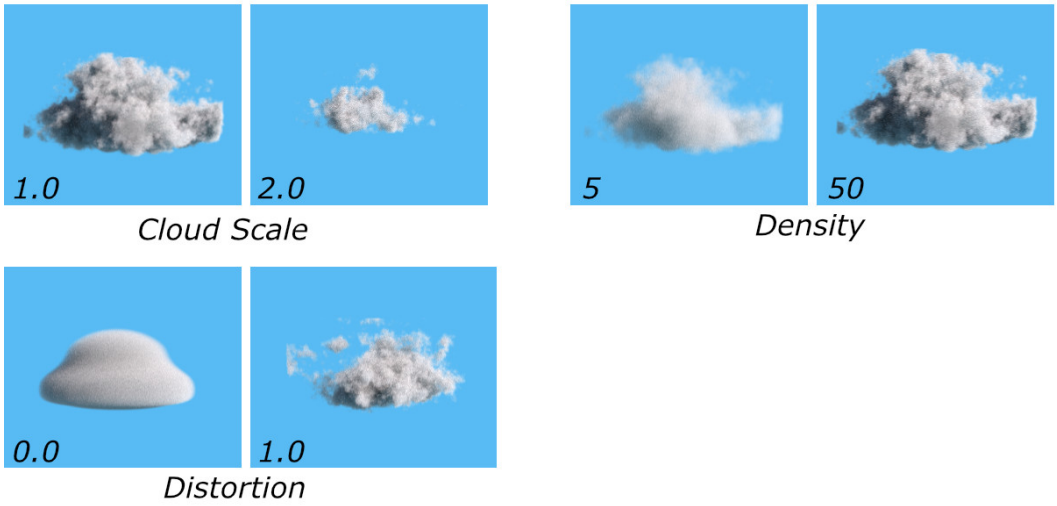

Abbildung 37: Einfluss der Parameter auf das Erscheinungsbild der Wolken

4.3 Animation

Um die dargestellten Wolken auch animierbar zu machen, müssen auf der Benutzeroberfläche Parameter zur Kontrolle der Animation zur Verfügung gestellt werden. Um das Gleichgewicht von Anwenderfreundlichkeit und Flexibilität zu wahren, werden jedoch nicht alle zu animierenden Werte als Parameter in der Benutzeroberfläche des Shaders angezeigt. Stattdessen werden folgende Parameter hervorgehoben, die die Animation der Wolke einfacher machen:

- **Wind Strengt:** Geschwindigkeit mit der die Textur auf dem Weltkoordinatensystem unabhängig von der Bewegung des Grundobjekts verschoben wird. Desto niedriger die Windgeschwindigkeit, desto weniger direktional erscheint die Verformung der Wolke. Für die nichtdirektionale Wolkenbewegung wird die vierte Dimension der FBM-Simplex-Noise (3.3.2) Textur genutzt.
- **Wind Direction:** Dreidimensionaler, normalisierter Vektor, in dessen Richtung die die Textur je nach Windstärke verschoben wird.
- **Stationary:** Multiplikator, um den sich die Textur auf den Weltkoordinaten mit der Bewegung des Grundobjekts mitbewegt.

- **Time:** Zeit in Sekunden.

4.4 Interaktive Wolkenvorschau

Die meisten modernen Renderengines bieten eine interaktive Vorschau des Renderergebnisses, die bei jeder Szenenveränderung aktualisiert wird. Um eine interaktiver Nutzererfahrung auch auf mittelklassiger Hardware zu ermöglichen, ist es sinnvoll neben der Anzeige des endgültigen volumetrischen Renderergebnis auch eine schneller renderbare Annäherung an das Endergebnis zu ermöglichen. Hierzu wird ein Querschnitt aus er Mitte der volumetrischen Textur als zweidimensionale Repräsentation der Wolke auf den Seiten des Grundmeshes dargestellt, indem der X Wert der Texturkoordinaten fest auf 0 gesetzt (Abb. 38). Um auch die volumetrische Vorschau interaktiv zu halten, ist es empfohlen die Einstellungen der Renderengine entsprechend anzupassen.

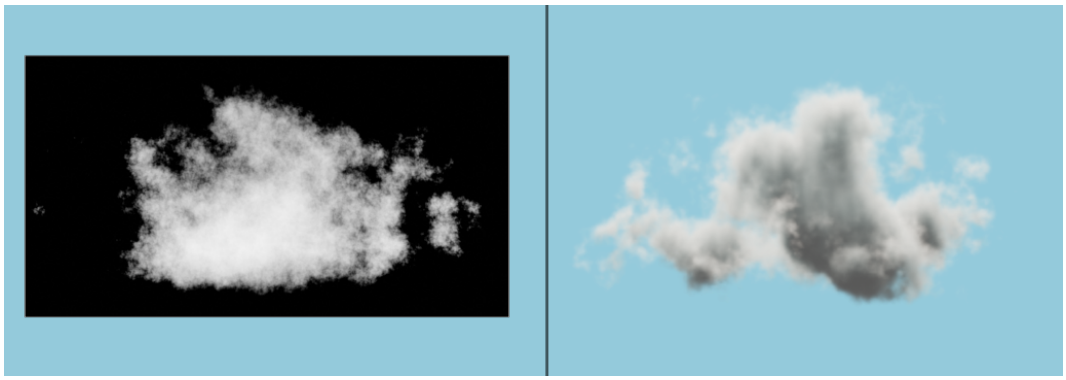

Abbildung 38: Zweidimensionale annähernde Repräsentation (links) der aus den Parametern resultierenden Wolkenform (rechts)

5 Evaluation

5.1 Vor- und Nachteile der Implementierung in OSL

Die Umsetzung des Wolkenshaders in OSL bietet sowohl Vor- als auch Nachteile. Der größte Vorteil von OSL, im Vergleich zu Nodebasierten Lösungen der verschiedenen Renderengines, ist die Kompatibilität. Die meisten modernen Renderengines unterstützen OSL (siehe Tabelle 1). Die unterstützen Sprachfeatures variieren jedoch zwischen den Renderengines.

Am Beispiel von Cycles bietet OSL zudem einfachen Zugriff auf vierdimensionale Noisefunktionen (3.3.1), der mit Shading Nodes nicht so einfach möglich ist. Der Nachteil von OSL für dieses Projekt ist die schlechtere Performance im Vergleich zu nativen Shadernodes der Renderengine. (5.1.1).

5.1.1 Performance

Bisher bietet keine Renderengine eine vollständige Implementierung von OSL mit CUDA oder OpenCL und somit ist das Beschleunigen des Renderings per GPU (Grafikprozessor) nicht möglich. Octane als komplett GPU basierte Renderengine unterstützt OSL zwar teilweise, die für diese Arbeit essentielle 4D Noise Funktion wird jedoch nicht unterstützt [4]. Für Arnold und Redshift ist die OSL Unterstützung für GPU Rendering für zukünftige Versionen angekündigt. Die Echtzeit- OpenGL Renderengine Eevee von Blender, hat ebenfalls keine Unterstützung von OSL. Es existiert ein Addon für Blender 'OSLPY'[5], mit dem es möglich ist, einfache OSL Shader zu Nodegroups zu konvertieren und somit auch auf dem GPU lauffähig zu machen. Die davon unterstützen Features sind allerdings geringer als bei der nativen

[4] docs.otoy.com/osl/features/index.html
[5] https://github.com/LazyDodo/oslpy

Renderengine	OSL Unterstützung
Appleseed	CPU
Arnold	CPU, GPU in Entwicklung [a]
Clarisse iFX	CPU
Cinema4D (Physical Renderer)	Nein
Corona	Nein
Cycles	CPU, GPU stark eingeschränkt durch OSLPY Plugin [b]
Houdini (Mantra)	Nein
Indigo	Nein
Iray	Nein
LuxCoreRender	Nein
Maxwell	Nein
Octane	GPU - Nur Texturen
Radeon ProRender	Nein
Redshift	GPU in Entwicklung [c]
Renderman	CPU
V-Ray	CPU

[a] answers.arnoldrenderer.com/questions/16493/osl-on-gpu-support.html
[b] github.com/LazyDodo/oslpy
[c] cgchannel.com/2019/05/redshift-rendering-technologies-unveils-redshift-3-0/

Tabelle 1: Liste bekannter Renderengines mit und ohne OSL unterstützung

CPU Implementierung und somit fehlen auch hier die für diese Arbeit benötigten Noise Funktionen. Ein weiterer Nachteil der Implementierung von prozeduralen Wolken auf Shaderebene generell, sind die hohen Performancekosten. Die Generierung Komplexer prozeduraler Texturen zur Renderzeit ist langsamer, als zum Beispiel das strikte Auslesen von vorher generierten VDB Daten. Für jeden gesendeten Ray muss der Pathtracer die Textur neu berechnen. Für volumetrische Materialien fällt das besonders ins Gewicht, da bei diesen der Shader öfter evaluiert werden muss als bei reinen Oberflächenmaterialien [arna] (da nicht nur Punkte an der Oberfläche, sondern auch innerhalb des Volumens berechnet werden müssen). Da Lookedevelopment ein sehr iterativer Prozess ist, bei dem es für Künstler wichtig ist schnelle Vorschaubilder zu rendern und Anpassungen zu machen, verlangsamt schlechte Performance nicht nur den finalen Renderingprozess, sondern auch den Weg dorthin.

Abbildung 39 vergleicht die Renderzeiten verschiedener Ansätze des Ren-

derings von volumetrischen Materialien mit Cycles. Die beiden Objekte in der Mitte wurden dabei mit einem dem OSL Shader ähnlichem Materialnodesystem umgesetzt. Auffällig hierbei ist der große Ersparnis an Renderzeit, der durch Nutzung der GPU entsteht. Die Umsetzung mit OSL in Cycles ist zudem wesentlich langsamer, als die Verwendung von nativen Material Nodes der Renderengine.

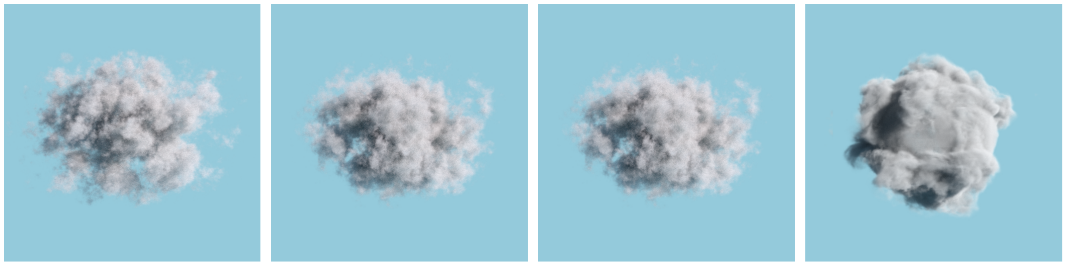

Abbildung 39: Verschiedene Rendering Methoden konstant 512x512 Pixel, 300 Samples, 2 Volume Bounces und 1.5 Step Size *CPU: AMD Ryzen 5 1600, GPU: GTX 1080*

5.2 Zusammenfassung und Ausblick

Grundsätzlich konnten in dieser Arbeit viele Techniken zur Generierung von Wolken gesammelt werden. Die Praktikabilität der Generierung dieser zur Renderzeit mit OSL, wird jedoch von den mit der vergleichsweise schlechten Rendering Performance einhergehenden Problemen (5.1.1) eingeschränkt. Zukünftig kann die Leistung andere Renderengines im Umgang mit OSL und prozeduralen volumetrischen Wolken evaluiert werden. Vor allem Redshift und Arnold könnten mit deren zukünftiger GPU Implementierung von OSL vielversprechend sein. Viele Erkenntnisse lassen sich voraussichtlich jedoch auch auf andere technische Grundlagen anwenden. So ließen sich die Grundform und deren Verzerrung durch prozedurale Texturen zum Beispiel auch als Programm, welches VDB Dateien erzeugt umsetzen. Auch eine zweidimensionale und dementsprechend performantere Variante zur prozedurale Generierung von in der Computergrafik oft benötigten 360° HDRI Umgebungstexturen ist denkbar. Für weiterführende Auseinanderset-

zungen in diesem Themenbereich bieten sich also einige Möglichkeiten an.

Literaturverzeichnis

[arna] *Standard Volume - Arnold for Maya Userguide 5 - Arnold Renderer*. docs.arnoldrenderer.com/display/A5AFMUG/Standard+Volume. – Aufgerufen am 24.08.2019

[arnb] *Volume-Displacement - Arnold for Maya Userguide 5 - Arnold Renderer*. docs.arnoldrenderer.com/display/A5AFMUG/Volume+-+Displacement. – Aufgerufen am 21.08.2019

[Bla] BLAIR, Stephen: *Foreneintrag: OSL on GPU support?* answers.arnoldrenderer.com/questions/16493/osl-on-gpu-support.html. – Aufgerufen am 24.08.2019

[Gri18] GRITZ, Larry: Open Shading Language 1.10 Language Specification, Sony Pictures Imageworks, 2018

[JL] JEN LOWE, Patricio Gonzalez V.: *The Book of Shaders: Fractal Brownian Motion*. thebookofshaders.com/13/. – Aufgerufen am 21.07.2019

[Kaj86] KAJIYA, James: The rendering equation, ACM SIGGRAPH Computer Graphics Paper, 1986. – ISSN 0097–8930

[Kes] KESSON, Malcolm: *OSL Cellnoise*. fundza.com/rfm/osl/cellnoise/index.html. – Aufgerufen am 26.08.2019

[NV15] NATHAN VOS, Andrew S.: The Real-time Volumetric Cloudscapes of Horizon: Zero Dawn, Guerrilla Games, 2015

[Per01] PERLIN, Ken: Chapter 2: Noise Hardware, 2001

[ren] *Renderman 21 Documentation - PxrVolume*. rmanwiki.pixar.com/display/REN/PxrVolume#PxrVolume-Clouds. – Aufgerufen am 27.08.2019

[RS] RICH SEDMAN, Stackexchange user: *Stackexchange: "Distance from the surface as an input node for use with volumetric materials"*. blender.stackexchange.com/questions/89202/distance-from-the-surface-as-an-input-node-for-use-with-volumetric-materials. – Aufgerufen am 20.08.2019

[Tha] THACKER, Jim: *Check out the new features coming up in Redshift 3.0*. www.cgchannel.com/2019/05/redshift-rendering-technologies-unveils-redshift-3-0/. – Aufgerufen am 24.08.2019

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.
Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.
Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

Furtwangen, den 31.08.2019 Simon Storl-Schulke