UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Echtzeitrendering von volumetrischen Wolken

Bachelorarbeit

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

vorgelegt von

Jakob Fedorenko

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Bastian Krayer MSc. (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Oktober 2019

Erklärung

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

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☒	☐

Koblenz, 8.10.19 J. Fedorenko

(Ort, Datum) (Unterschrift)

Abstract

In dieser Arbeit wird das Echtzeitrendering von Wolken von der Theorie bis hin zur Entwicklung derselben behandelt. Dabei sollen die visuellen Eigenschaften der Wolken sowie die unterschiedliche Wolkentypen simuliert werden. Dabei ist die Berechnung der Beleuchtung essentiell für ein glaubwürdiges Ergebnis. Die Rendertechniken nutzen dabei unterschiedliche Noise-Texturen; für die Modulierung der Wolken sind es hauptsächlich Perlin- und Perlin-Worley-Texturen. Das Rendern der Wolken wird per Compute-Shader durchgeführt um die Echtzeitfähigkeit zu gewährleisten. Um die Performance zu steigern, werden Temporal Reprojektion und andere Optimierungstechniken angewendet.

This paper deals with the real-time rendering of clouds from theory to development. The aim is to simulate both the visual properties and the various cloud types. The calculation of the illumination is essential for a credible result. The rendering techniques use different noise textures for the modulation of the clouds , mainly Perlin and Perlin-Worley textures. In order to ensure the ability of real-time rendering, the rendering is performed by compute shaders. Temporal reprojection and other optimization techniques are used to improve the performance.

Vorwort

Zum Echtzeitrendering von Wolken haben mich zwei Punkte motiviert. Zum einen die Faszination des Computerspielmediums, dessen Vielfalt an Kunst und Geschichten in Kombination mit unterschiedlichen Grafikstilen es schafft mich zu erreichen. Zum anderen die unterschiedlichen Lehrveranstaltungen im Bereich der Computergrafik die mein Interesse geweckt haben, eben selbige Grafik aus Spielen selbst entwickeln können zu wollen.

Ich möchte mich noch in diesem Vorwort bei einigen Personen bedanken die mich während der Arbeit an der Thesis unterstützt haben.

Zu Beginn möchte ich mich bei Prof. Dr.-Ing. Stefan Müller und Bastian Krayer MSc. für die hervorragende Betreuung bedanken.
Danke an alle, die sich die Zeit genommen haben meine Arbeit Korrektur zu lesen.
Danke an Pascal Bendler und Sebastian Gaida, die mich zu Beginn tatkräftig beim Aufbau des Projekts unterstützt haben. Und vor allem an Sebastian für die Unterstützung beim Schreiben und bei Problemen.
Vielen Dank an Vanessa Schüller für die vielen hilfreichen Tipps die mich sehr Voran gebracht hatten.
Ein großes Dankeschön noch an Elena Lörsch für die großartige Unterstützung beim Schreiben der Thesis.
Vielen Dank an meine Familie die mich während der Zeit motiviert und hinter mir stand.

Inhaltsverzeichnis

1 Einleitung

Mit der steigenden Performanz von Hardware und dem Streben nach immer größerer Immersion werden CGI-Effekte und Computerspiele stetig realistischer. Szenarien, die mit einem freien Himmel arbeiten, wie beispielsweise Flugsimulatoren und Open-World Rollenspielen, verhelfen natürlich wirkende Wolken zu eben dieser Immersion. Vor allem in Cockpit-Situationen in Virtual-Reality wird dies verstärkt. Daher müssen die Wolken realistisch, dabei aber auch leistungsfähig berechnet werden. Ersteres bedeutet in diesem Fall die größtmögliche Annäherung an die Form, Beschaffenheit und Farbe echter Wolken. Zugleich muss dies leistungsfähig in Echtzeit ablaufen, da, aus Kostengründen, im CGI mit möglichst wenigen Frames gearbeitet werden und in Computerspielen für ein flüssiges Spielerlebnis eine Framerate von mindestens 30FPS, besser 60FPS oder mehr erreicht werden sollten.
Das Echtzeitrendering von Wolken ist bis heute eine herausfordernde Aufgabe, die komplexer wird, je dynamischer Himmel und Wetter sich verhalten sollen. Dadurch, dass es keine statische Skybox ist, ist es möglich, glaubwürdige visuelle Effekte, beispielsweise Crepuscular Rays (Wolkenstrahlen), Wolkenschatten und Silver-Lining (Silberstreifen) darzustellen. Betrachtet man Wolken als Objekte mit Volumen, so enthält dieses Volumen in der realen Welt Wassertröpfchen und andere Aerosole, welche die Lichtbrechung innerhalb der Wolken beeinflussen. Das mehrfache Brechen und Streuen von Licht innerhalb einer Wolke bezeichnet man als Multiple-Scattering. Die Berechnung davon ist komplex und äußerst rechenintensiv, weshalb sich dieser in dieser Thesis nur vereinfacht angenähert wird, um ein Echtzeitrendering von volumetrischen Wolken zu ermöglichen. Dabei werden verschiedene Optimierungsstrategien und Compute-Shader implementiert, um die, für die Darstellung realistischer Wolken, notwendigen Berechnungen durchzuführen.
Die gängigen Verfahren des Echtzeitrenderings werden im 2. Kapitel vorgestellt, während im Darauffolgenden 3. die physikalischen Grundlagen realer Wolken erläutert werden, die diesen Form und Farbe geben. Das 4. Kapitel beschreibt die Rendertechnik, auf der diese Thesis aufbaut, und die physikalische Basis der Berechnung der Radiance innerhalb der Wolken. Anschließend werden im 5. Teil der Arbeit die Konzeptionierung der Funktionen zum Rendern der Wolken, des Lichts und des Himmels definiert, deren Implementierung im 6. und Optimierung im Kapitel 7 behandelt wird. Die Resultate dieser werden im darauffolgenden Kapitel 8 beschrieben und dargestellt. Während im 9. Kapitel die Eindrücke aus dem Entstehungsprozess der Thesis eingebracht werden, soll das abschließende Kapitel als Ausblick mit Vorstellungen für künftige Arbeiten fungieren.

2 State of the Art

In diesem Kapitel wird ein Einblick in bisherige Ansätze und Theorien des Wolken-Renderings gegeben, da dieses die Computergrafik seit einigen Jahren beschäftigt und dabei immer wieder vor Probleme stellt. Unterschieden wird generell zwischen zwei Verfahren: den Nicht-Volumetrischen und den Volumetrischen.

Nicht-Volumetrische Ansätze:
Der erste Ansatz Wolken zu berechnen, stammt aus dem Jahr 1985 und wurde von Geoffrey Garner [Gar85] vorgestellt. Dieser verwendete mathematische Texturfunktionen, die eine Summe von Sinuswellen beinhalteten, und kombinierte diese mit Ellipsoiden, um angedachte Wolken zu modellieren. Elinas und Stürzleiner [ES00] konnten 2000 durch die fortschreitenden technischen Möglichkeiten Gardners Ansätze erstmalig in Echtzeit umsetzen. 2001 wurde von Harris und Lastra[HL01] deren Partikelsystem vorgestellt, in dem der Raum innerhalb der Wolken mit Partikel gefüllt und per Splatting gerendert wurde. Zur Optimierung der Performance wurde das Multiple-Forward-Scattering vorberechnet und die Partikel in Billboards gerendert. Eine typische Anwendung des Systems fand in Flugsimulatoren statt, da es hiermit möglich war durch Wolken durchzufliegen.
Bei all diesen Ansätzen wurde das anisotrope Streuen des Lichts kaum berücksichtigt.

Volumetrische Ansätze:
Eine der ersten volumetrischen Umsetzungen von Wolken war von Bouthors et al. [BNM+08], bei welchen diese am Rand Meshes besitzen. Dieser Rand wird dann anhand von prozeduralen, volumetrischen Hyper-Textures erodiert, um die Wolken final zu formen. Dabei wurde bei der Lichtberechnung von der Lorenz-Mie-Streuung Gebrauch gemacht. Andrew Schneider veröffentlichte 2015 [SV15] einen Ansatz, welches bei dem Spiel Horizon Zero Dawn angewendet und in GPU Pro 7 [Sch16] publiziert wurde. Bei diesem wurden ebenfalls prozedurale Texturen verwendet, aus denen die Form und Dichte der Wolken mittels Ray-Marching berechnet wurden. Als Noise-Texturen dienten Perlin- sowie Perlin-Worley-Noise. Für die Lichtberechnung der Wolken wurde die Henyey-Greenstein Funktion verwendet. Nur zwei Jahre später aktualisierte Schneider seine Implementation und präsentierte diese ebenfalls auf der SIGGRAPH 2017 [Sch17]. In der überarbeiteten Version sei, nach Schneider, die Basis der Wolken gleichbleibend, jedoch könnten diese durch weitere Parameter weiter angepasst werden. Unter anderem durch die aktualisierte Lichtberechnung wurden die Wolken interaktiver und realistischer.

Im selben Jahr veröffentlichte Disney [KMM+17] eine neue Technik unter dem Einsatz eines neuronalen Netzwerks sowie einer Monte Carlo Integration. Dabei werden die sichtbaren Punkte einer Wolke abgetastet und an einen 3-D Deskriptor weitergegeben. Dieser speist diese Informatio-

nen dann in das neurale Netzwerk und erhält von diesem die passende Shading-Konfiguration. Wie bei anderen neuralen Netzen wird die Genauigkeit und Geschwindigkeit der Ergebnisse mit der Anzahl der Testdaten besser.
Auf der SIGGRAPH 2016 präsentierte Sebastien Hillaire von Frostbite seine Kombination des Renderns von Atmosphäre und Himmel [Hil16]. Basierend auf Schneiders-Ansatz aus dem Jahr 2015 modifizierten sie diesen so, das der Himmel von anderen Planeten berechnet werden konnte. Unter dem Einsatz von vielen vorberechneten Look-up-Tables, kurz LUTs, zum Beispiel Scattering-LUTs, war diese Umsetzung echtzeitfähig. Dieser Ansatz wurde in Mirrors Edge Catalyst angewendet.
2019 im Rahmen der SIGGRAPH [FB19] präsentierte Rockstar die Implementation der Wettersimulation, somit auch des Wolken-Renderings, aus Red Dead Redemption 2. Dabei verfeinerten sie die Ansätze von Schneider, fügten weitere Lichteffekte wie Blitze und Anfänge für Back-Scattering hinzu und erhöhten die Anzahl unterschiedlicher Wolken.
Ziel dieser Thesis ist das Rendern volumetrischer Wolken auf Basis von Schneiders Nubis-System [Sch17], außerdem das Hinzufügen weiterer Verfahren und deren Prüfung.

3 Physikalische Grundlagen

In diesem Kapitel wird auf die verschiedenen Arten von Wolken und deren physikalischen Eigenschaften eingegangen, die beim Rendern weitgehend simuliert werden.

3.1 Wolkenarten

Eine Wolke besteht aus einer großen Anzahl von dispersiven Teilchen wie Wassertröpfchen, Eiskristallen sowie anderen kleinen Partikeln wie Seesalz. Diese werden allerdings erst sichtbar, wenn warmer Wasserdampf unter einer bestimmten Temperatur abkühlt und die Tröpfchen kondensieren [Hit11]. Die Form der Wolke wird grundlegend von der Temperatur und dem Wind beeinflusst. Da die Temperatur mit steigender Höhe abnimmt, steigt die Dichte der Wolken an, weswegen diese auch im oberen Bereich runder wirken. Sobald die Temperatur auf der Erdoberfläche unter der Wolke steigt, tritt der Effekt auf, dass die Wolken aufsteigen.

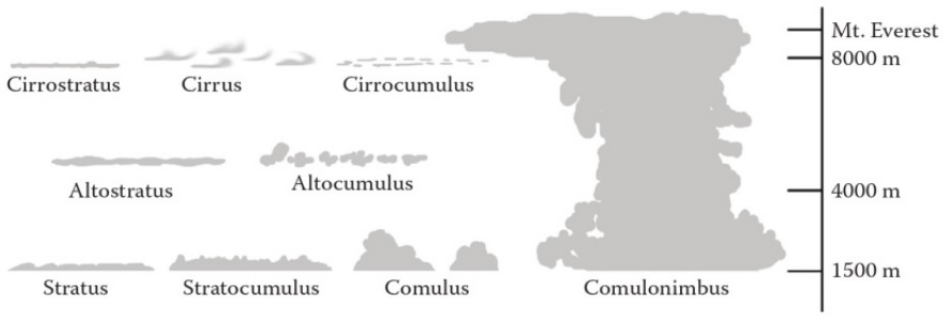

Abbildung 1: Wolkenarten und ihre Bezeichnungen

Anhand dieser Eigenschaften, unter anderem Höhe und Form, werden Wolken in vier Wolkenfamilien eingeteilt, aus denen sich zehn Wolkenarten ableiten [Nel18] lassen. In der Abbildung 1 werden die gängigsten Arten aufgeführt. Die Wolken auf der niedrigsten Höhe, abseits vom Nebel, sind die Cumulus, Stratocumulus und Stratuswolken. Die dichten, bauschigen und hellen Wolken werden als Cumulus bezeichnet. Das Aussehen der Stratuswolken ähnelt einer grauen bis fast weißen Schicht, die sich flach und merkmalslos in der Atmosphäre aufhält. Befinden sich Stratus- und Cumuluswolken auf mittlerer Höhe von 2 km bis 7 km, erhalten diese die Vorsilbe-Alto. In der höchsten Schicht ab ca. 8 km Höhe befinden sich die Cirrus, Cirrosstratus- und Cirrocumuluswolken. Diese sind sehr fein in ihrer Erscheinung und treten bei guten Wetterbedingungen und tiefen Winden auf. Aufgrund der Höhe bestehen diese hauptsächlich aus Eiskristallen. Auf allen Höhen gleichzeitig kann die Cumulonimbus auftreten. Erhält eine Wolke den Zusatz Nimbus oder den Vorsilbe Nimbo, handelt es sich um eine Regenwolke, da das Maximum an Sättigung an Wassertröpfchen für die Wolke erreicht ist. Einige der vorgestellten Wolken werden in den Abbildungen 2,3,4,5,6, und 7 dargestellt.

Abbildung 2: Cirrus-Wolke [Wik05c]

Abbildung 3: Altocumulus [Wik05a]

Abbildung 4: Altostratus aus [Wik05b]

Abbildung 5: Cumulus aus [Wik05d]

Abbildung 6: Stratus aus [Wik17]

Abbildung 7: Stratocumulus aus [Ton06]

3.2 Visuelle Eigenschaften

Um überzeugende Wolken rendern zu können, muss die Renderengine in der Lage sein, verschiedene, in der Natur auftretende, Effekte darzustellen. Die visuellen Eigenschaften dieser Effekte werden im folgenden Abschnitt beschrieben.

3.2.1 Silver-Lining

Einer der grundlegendsten Effekte ist das sogenannte Silver-Lining. Dieser beschreibt die Vorwärtsstreuung (Forward-Scattering) des Lichts innerhalb einer Wolke. Dadurch wirken die Außenkanten heller, wodurch die Wirkung entsteht, dass die Wolke von hinten beleuchtet wird.

3.2.2 Crepuscular Rays

Ein weiterer Effekt, der bei starkem Sonnenschein und bewölktem Himmel auftreten kann, sind die Strahlenbüschel, zu Englisch god oder crepuscular Rays. Wenn Sonnenstrahlen durch Öffnungen einer Wolkendecke scheinen und sich dahinter dunklere Flächen befinden, wie zum Beispiel die Wolkenunterseite einer Regenwolke, entsteht dieses Phänomen, bei dem eine Lichtsäule scheinbar die Wolkendecke durchbricht und ungebrochen bis zur Erdoberfläche reicht.

3.2.3 Puderzucker-Effekt

Darunter versteht man, vor allem bei Cumuluswolken, dass der zur Blickrichtung geneigte Wolkenrand dunkle Kanten aufweist. Dadurch werden bauschigen Formen selbst dann sichtbar, wenn diese nahezu weiß ist. Das tritt in Erscheinung, wenn die Blickrichtung sich der Lichtrichtung nähert. Als Ursache dient auch hier das In-Scattering der Lichtstrahlen, denn innerhalb der Wolke erhalten die Partikel mehr nach innen gestreute Lichtstrahlen als die Partikel an der Wolkenoberfläche. Insofern erscheinen Spal-

ten innerhalb einer Wolke heller als Kanten. Zur Veranschaulichung dient hierbei die Abbildung 8.

Abbildung 8: Wolken mit Puderzucker-Effekt

3.2.4 Wolkenfarbe

Die Erscheinungsweise der Wolken wird hauptsächlich durch die Menge des erhaltenen Lichts und dessen Farbe bestimmt. Sobald die Lichtstrahlen in die Wolken eindringen, werden diese von den Aerosolen, Wassertröpfchen oder Eiskristallen reflektiert und somit mehrfach gestreut. Da bei diesem Prozess die einzelnen Partikel gleichmäßig alle Wellenlängen ein wenig absorbieren, erscheinen die Wolken weiß. Je größer eine Wolke ist, desto mehr Lichtstrahlen werden in dieser reflektiert. Dadurch wird das Erreichen des Lichtes am Wolkenboden verhindert und dieser erscheint für den Betrachter nicht weiß, sondern grau. Ebenso bei einer Regenwolke, die mehr Energie absorbiert, wodurch die Partikel grau werden. Das Rayleigh-Scattering, welches beim Auf- und Untergehen der Sonne auftreten kann, führt wie in der Abbildung 9 zu einer Färbung der Wolken in verschiedenen Rottönen.

Abbildung 9: rötlich beleuchtete Wolken beim Sonnenuntergang

Zwar wird dieser Vorgang im späteren Verlauf der Arbeit genauer erläutert, folgend soll er dennoch zusammengefasst werden. Befindet

sich die Sonne in der Nähe des Horizont, müssen die Wellenlängen des Lichtes einen längeren Weg zurücklegen und die Blauen werden durch die Schichten der Atmosphäre zerstreut. Die roten und orangenen Wellenlängen dagegen werden von den Wolken reflektiert. Da die Sonne die ganze Umgebung beleuchtet, gibt es weitere potenzielle Lichtquellen, die das Licht zurückstrahlen.

4 Volume Rendering

Allgemein erfolgt die Lichtberechnung in der Computergrafik durch das Verfolgen von theoretischen Lichtstrahlen, die auf Objekte treffen. Dabei bliebt in einem luftleeren Raum die Radiance zwischen zwei Treffern identisch, außerhalb eines Vakuums dagegen interagieren die Lichtstrahlen auf ihrem Weg mit verschieden großen Partikeln. Dazu zählen auch Wolkenteilchen. Deshalb kann das Rendern der Wolken anders gehandhabt werden, da Wolken aufgrund ihrer Eigenschaften als Participating-Medium verwendet werden können. Die Technik wird als Volume-Rendering bezeichnet. Im Verlauf dieses Kapitels wird zuerst auf die Problematik des Participating-Medium eingegangen und wie dieses zu lösen ist. Damit lässt sich das Zusammenspiel zwischen Materie und Licht mathematisch mit der Volume-Rendering-Equation beschreiben und auf Basis von Novat et. al.[NGH+18] und Fong et. al.[FWKH17] simulieren.
Innerhalb eines Participating-Medium gibt es drei physikalische Prozesse, die auftreten können, wenn ein Lichstrahl dieses durchdringt. Die Absorption, Emission und Streuung, auch Scattering genannt, auf die jetzt im Detail eingegangen wird. Die Berechnungen für diese drei werden als Koeffizienten definiert, welche von Novak et. al. [NGH+18] beschrieben wurde.
Absorption:
Treffen Photonen eines Strahls auf absorbierende Partikel wird ein Teil der Radiance reduziert. Die Visualisierung des Vorgangs wird in Abbildung 10 dargestellt.

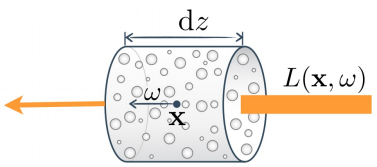

Abbildung 10: Absorption

Der Absorptionskoeffizient beschreibt die Wahrscheinlichkeit, dass die Dichte des Lichtstrahls über eine Unit-Distanz absorbiert wird. $L(x, w)$ ist die Strahlendichte a am Punkt x in Richtung w, so ist der Absorptionsko-

effizient μ_a und die Dichte dz. Somit kann man die Gleichung wie folgend aufstellen:

$$\frac{dL}{dz} = -\mu_a L(x, w) \tag{1}$$

$$\mu_a = Absorptionskoeffizient \tag{2}$$

Emission:
Anders als bei der Absorption wird die Radiance bei der Emission verstärkt. Dies resultiert aus der Freisetzung von Partikeln bei glühenden oder lumineszenten Prozessen. Die Emission ist in Abbildung 11 visualisiert.

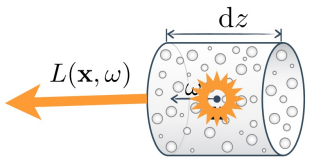

Abbildung 11: Emission

$$\frac{dL}{dz} = -\mu_a L_e(x, w) \tag{3}$$

$$L_e = abgegebene\, Radiance \tag{4}$$

Dabei kann die Emission nur existieren sofern das Material in der Lage ist, Energie aufzunehmen und als Photonen wieder abzugeben. Die L-Funktion wird dabei manuell definiert durch dass Messen realer Gegebenheiten oder Flüssigkeitssimulationen.

Scattering:
Unter Scattering versteht man zwei verschiedene Arten, das In-Scattering und Out-Scattering.
In-Scattering:
Wenn der Lichtstrahl in einer Richtung w durch ein Medium strahlt, wird es darin enthaltenen Partikel treffen. Veranschaulicht durch die Abbildung 12.

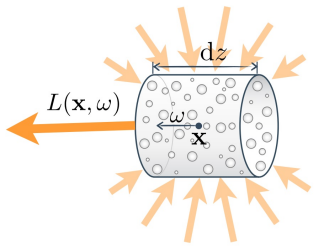

Abbildung 12: In-Scattering

In-Scattering bedeutet, dass Photonen in dieselbe Richtung w gestreut werden, die nicht von dem Ausgangspunkt von w, nämlich der Strahlenquelle, sondern auch von reflektierten Strahlen stammen.

$$\frac{dL}{ds} = -\mu_s L_s(x, w) \tag{5}$$

$$L_s = Scatteringkoeffizient \tag{6}$$

Die Radiance des In-Scattering wird hier über ein Integral jeder Richtung einer Kugel berechnet. Dabei wird die eingehende Radiance über eine Phasenfunktion f_P gewichtet.

Integral $L_s(y, w) = \int_{S^2} f_P(w, \bar{w}) L(y, \bar{w}) d\bar{w}$

Out-Scattering:
Out-Scattering entsteht durch die Tatsache, dass bei dem Kontakt eines Lichtstrahles mit einem Participating-Medium die Photonen wahrscheinlich auf Partikel treffen werden.

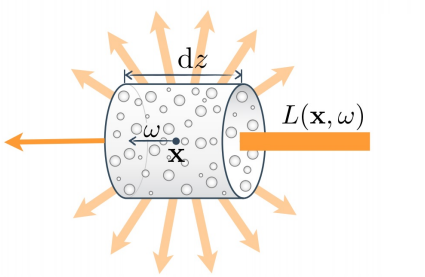

Abbildung 13: Out-Scattering

Anders aber werden die Licht-Photonen zerstreut und somit von ihrer eigentlichen Richtung w, wie in der Abbildung 13 zu erkennen, abgelenkt. Die Radiance des Lichts nimmt aus diesem Grund ab. Mit dem sogenannten Streuungs- oder Scatteringskoeffizienten μ_s, analog zur Asorption, wird die Wahrscheinlichkeit des Scatterings über eine Distanzeinheit ausgedrückt.

$$\frac{dL}{ds} = -\mu_s L(x, w) \tag{7}$$

$$\mu_s = Scatteringkoeffizient \tag{8}$$

4.1 Phasenfunktion

Die zuvor genannte Phasenfunktion $f_p(x, w, \bar{w})$ [FWKH17] beschreibt die Winkelverteilung des gestreuten Lichts. Diese kann als Wahrscheinlichkeitsdichtefunktion verstanden werden, da es die Wahrscheinlichkeit angibt, in welche Richtung ein Photon gelenkt wird. Diese wird als 1D-Funktion als Winkel zwischen zwei Richtungen definiert. Solche Funktionen werden über eine Kugel normiert, da sichergestellt werden muss, dass bei Streukollisionen keine Radiance hinzugefügt oder entfernt wird.

$$f_p(x, w, \bar{w})d\theta = 1 \tag{9}$$

Zusätzlich müssen diese die Wechselseitigkeit erfüllen, indem die beiden Richtungen, ein- und auswärts, getauscht werden können, aber dennoch ein äquivalentes Ergebnis hervorbringen. Beispiele hierfür wären isotropische Strahlung, Rayleigh-Scattering sowie die später vorgestellte Henyey-Greenstein-Funktion. Bei Ersterem ist die Streuung des Lichts, das Scattering, in alle Richtungen identisch. Die Gleichung 8 stellt die Phasenfunktion für isotropische Volumen nach [FWKH17] dar.

$$f_p(x, \theta) = \frac{1}{4\pi} \tag{10}$$

Da Wolken anisotropisches Verhalten zeigen, ist die Berechnung des Scatterings komplexer. Dabei müssen weitere Faktoren, wie die Form des Partikels, die Wellenlänge des Photons oder die Außentemperatur, berücksichtigt werden.
Eine Phasenfunktion in dieser Thesis ist die Rayleigh-Scattering-Funktion. Dieses erklärt die unterschiedlichen Färbungen am Himmel über den Tagesverlauf und wird von Alan Zucconi [Zuc17] vereinfacht dargestellt. Dieser Farbverlauf entsteht durch die Streuung des Lichtes, durch Partikel, die kleiner sind als die Wellenlängen des Lichts und deshalb je nach Position der Sonne den Himmel unterschiedlich färben.

$$f_p(x, \theta) = \frac{3}{16\pi} * (1 + cos^2\theta) \tag{11}$$

Eine vergleichbare Funktion ist das, nach Gustav Mie benannte, Mie-Scattering. Anders als beim Rayleigh-Scattering geht man hier von Partikeln in der Atmosphäre aus, die einen ähnlichen Durchmesser besitzen wie die Wellenlänge [FWKH17]. Da das MIE-Scattering eine komplexe Berechnung darstellt, ist die von Louis George Henyey entwickelte Henyey-Greenstein-Funktion eine günstigere Alternative:

$$f_p(x, \theta) = \frac{1}{4\pi} * \frac{1 - g^2}{(1 + g^2 - 2g * cos\theta)^{\frac{2}{3}}} \tag{12}$$

Die Asymmetrie der Funktion wird durch den Parameter $g = [-1, 1]$ kontrolliert, welcher sich gut zum modellieren von Scattering-Arten eignet. Damit ist es möglich unterschiedliche Streuarten zu Modellieren. Für Backward-Scattering muss g negativ gewählt werden. Forward-Scattering wäre bei $g > 0$ und bei isotropem Scattering ist $g = 0$. Die Verbindung mehrerer HG-Phasenfunktionen ermöglicht die Approximation komplizierterer Phasenfunktionen [FWKH17]. Dies wird unter anderem von Hillaire [Hil16] angewendet. Kombiniert werden Absorption, Emission und Scattering zu der Radiative Transfer Equation (RTE), auch als Strahlungstransportgleichung definiert, welche die Ausbreitung von Strahlung innerhalb eines Mediums beschreibt.

4.2 Radiative Transfer Equation

Diese Gleichung charakterisiert die Wahrscheinlichkeitsdichte, kurz PDF für Probability-Density-Function, von Kollisionen pro Entfernungseinheit innerhalb eines Mediums. Indem μ_t die Summe der Absorption μ_a und des Out-Scatterings μ_s definiert, kann die RTE wie folgt beschrieben werden: Da sowohl Absorptions- als auch Out-Scatteringskoeffizient die gleiche Auswirkung auf die Radiance haben, bildet man durch die Kombination beider in der RTE den Extinktionskoeffizienten μ_t [NGH+18].

$$Extinktionskoeffizient \quad \mu_t = \mu_a(x) + \mu_s(x) \quad als \quad Losses \tag{13}$$

$$\frac{dL(x,w)}{dz} = -\mu_t(x)L(x,w) + \mu_a(x)L(x,w) + \mu_s(x)L(x,w) \tag{14}$$

Die RTE umgeformt in Integralform:

$$L(x,w) = \int_0^z T(x,y)[\mu_a L_e(y,w) + \mu_s(y)L_s(y,w)]dy \tag{15}$$

Das $T(x, y)$ in der Gleichung repräsentiert in diesem Fall die Transmission-Funktion. Diese Transmission wird im späteren Verlauf der Arbeit auch mit dem englischen Begriff Transmittance bezeichnet. Dabei wird ausgeführt, wie viel Strahlung in Abhängigkeit mit der optischen Dichte $\tau = \int_0^y \mu_t(s)ds$ von einem Punkt x zu einem anderen y gelangt, woraus sich die ganze Funktion als $T(x, y) = e^{-\tau(x,y)}$ definieren lässt. Diese wiederum beschreibt jede Materie im Medium, die theoretisch mit dem Licht interagieren könnte.

4.3 Volume Rendering Equation

Wenn dem RTE-Term noch die ausgehende Radiance, der Oberfläche hinter dem Medium hinzugefügt wird und anschließend mit der Transmission des gesamten Strahls multipliziert, resultiert daraus die Volume Rendering Equation (VRE).

$$L(x,w) = \int_0^z T(x,y)[\mu_a L_e(y,w) + \mu_s(y) L_s(y,w)]dy + T(x,y)L_o(z,w) \quad (16)$$

Da die Lösung dieser Gleichung unter Berücksichtigung der Mehrfachstreuung berechnet werden muss, ist dieser Prozess sehr rechenintensiv. Es bietet sich an, diese Gleichung mit dem Einsatz von Monte-Carlo Methoden zu approximieren und die Mehrfachstreuung zu vereinfachen.

4.4 Volume Ray-Marching

Eine Technik, die in volumetrischen Renderingsystemen im medizinischen Kontext verwendet wird, ist das Ray-Marching, welches alternativ auch als Ray-Casting bezeichnet wird. Genutzt wird diese beispielsweise sowohl beim Rendern komplexer Geometrien wie etwa Fraktalen als auch bei der Erstellung von CT's oder MRT-Scans [DMPPA11, Wei07]. Bei diesen Anwendungsgebieten werden 2D-Bilder aus 3D-Daten generiert und visualisiert. Am Anfang wird, ähnlich dem Ray-Tracing, ein Strahl von jedem Pixel aus gesendet. Dabei wird in einer bestimmten Schrittlänge, je nach Implementation festdefiniert oder adaptiv, geprüft, ob der Strahl ein Objekt trifft. Bei einem Treffer werden an dieser Position die Daten für weitere Berechnungen aus dem Volumen herausgelesen. Dies wird solange wiederholt, bis das Volumen verlassen wird oder ein bestimmter Schwellwert erreicht wird. Sobald dies Fall der Fall ist, werden alle Werte aus den Trefferpositionen zu einem finalen Wert akkumuliert. Bei dieser Thesis wird dies im Compute-Shader genutzt, bei denen die Strahlen, in der Implementation als Rays bezeichnet, 3D-Noise-Texturen als Volumen verwenden.

4.5 Noise-Texturen

Allgemein wird mit Noise das Rauschen bezeichnet, welches bei verschiedenen Signalen, etwa Bildern und Tönen, unterschiedliche Störungen verursachen kann. Manche Arten von Noise können durch spezielle Filter reduziert oder retuschiert werden. In der Computergrafik dagegen haben sich Noises als geeignetes Mittel hervorgetan, um unerwünschte Effekte zu eliminieren, beispielsweise Banding-Effekte. Ken Perlin [Per85] entwickelte zum Rendern von Wolken oder Wasser 1983 eine Funktion namens Perlin-Noise. Dadurch war es möglich, Muster zu simulieren, die nicht computergeneriert oder einheitlich aussehen. Die Funktionsweise des

verbesserten Perlin-Noises wird verständlich und im Detail von Adrian Biagiolis Blog [Bia14] sowie in der Arbeit von Wilhelm Burger [Bur08] erläutert.
Eine weitere Noise-Funktion die innerhalb der Thesis verwendet wird, ist das sogenannte Worley-Noise, die 1996 von Steve Worley [Wor96] entwickelt wurde, um Wasseroberflächen oder Steinmuster zu simulieren. Der Algorithmus basiert auf den folgenden zwei Schritten: Zunächst werden zufällig Punkte in einem 2D- oder 3D-Raum verteilt. Danach berechnet die Noise-Funktion den Wert des Pixels auf der Basis der Distanz der Punkte zum aktuellen Pixel. Daraus resultiert eine zellenartige Struktur in der Noise-Textur. Mit diesen Noises werden die 3D-Noise Texturen erstellt und in verschiedenen Kanälen gespeichert.

4.6 Fractal Brownian Motion

Um die abweichenden Noises aus den Texturen zu kombinieren, wird fractal Brownian motion, abgekürzt fBm, verwendet. [SV15]. Damit lassen sich die verschiedenen Ebenen der Noises mit ihren unterschiedlichen Frequenzen und Amplituden zusammenfassen. Da sich daraus eine feinere Granularität und höhere Variation im Gegensatz zu der vorgegebenen Noise-Textur ergeben, eignet sich das fBm zum Modellieren von Wolken.

5 Konzeptionierung

In diesem Abschnitt werden die Konzepte vorgestellt, auf denen die Implementation basiert.
Die Basis der Konzepte und Implementierung beruhen auf den zuvor erwähnten Ansätzen von Andrew Schneider [SV15, Sch16, Sch17]. Damit ist es möglich, eine Großzahl an Wolken zu simulieren.

5.1 Wolkenmodellierung

Um die Form der Wolke zu bestimmen, ist der Algorithmus in zwei Phasen aufgeteilt. Im Ansatz von Schneider wird das zuvor vorgestellte Ray-Marching verwendet, um zwei 3D-Texturen zu samplen. Aus diesen ergeben sich Werte, wie etwas die Dichte, die als Paramter für die Generierung der Wolkenform dienen. Die Basisform wird in der ersten Phase anhand der ersten 3D-Textur erstellt, während mithilfe der zweiten in der folgenden Phase Details hinzugefügt werden.
Um die Textur für die Basisform zu erstellen, wird, aufgrund seiner Eigenschaften, das zuvor erläuterte Perlin-Noise-Verfahren verwendet. Um ein wolkenähnliches, bauschiges Muster zu erhalten, wurde dieses mit einem

niedrig frequenten invertierten Worley-Noise kombiniert, wobei jedoch alle zuvor genannten Charakteristika erhalten bleiben. In jedem Slice der 3D-Textur enthalten sind die RGBA-Kanäle unterschiedlicher Noises. Das kombinierte Perlin-Worley-Noise wird im roten Kanal gespeichert, während in den GBA-Kanälen invertierte Worley-Noises gespeichert werden. Die Frequenz der Worley-Noises nimmt zwischen den Kanälen zu und besitzt im Alpha-Kanal somit die höchste Frequenz.
Um zu verhindern, dass die Wolken eine glatte Oberfläche haben, wird die zweite 3D-Textur verwendet, um diesen mehr Details hinzuzufügen. Die Oberfläche wird erodiert, indem die zweite Textur hochfrequente Noises ihrer RGB-Kanäle nutzt. Verwendet wird dafür, wie in der ersten Textur, ein immer höher frequentes Worley Noise. Für die Erstellung dieser Texturen wurde von Schneider ein Houdini-Plugin [Sch17] entwickelt, der die Generierung beider 3D-Texturen mit unterschiedlichen Paramtern ermöglicht.

5.2 Density-Height-Funktion

Da erläutert wurde, dass unterschiedliche Wolkenarten heterogene Dichten und Höhen besitzen, wird eine Funktion benötigt, um dieses Verhalten auch zu simulieren. Diese Funktion interpoliert drei unterschiedliche Gradienten, die Stratus-, Stratocumulus- und Cumulus-Gradienten. Zusätzlich verwendet die Funktion die relative Höhe in der Atmosphäre und einen Gewichtungsparameter, um keine allzu einheitlichen Wolken zu erhalten. Dieser wird in einer 2D Textur gespeichert, die als Weather-Map bezeichnet wird. Um unterschiedliche Wolkenarten zu erhalten, wird das Ergebnis aus der Density-Height-Funktion anschließend mit der Dichte des zu berechnenden Punktes multipliziert.

5.3 Weather-Map

Die Weather-map enthält in ihren RGB-Kanälen jeweils unterschiedliche Parameter für den Renderprozess. Folgendes ist in den Kanälen enthalten:

Roter Kanal: Der Coverage-Wert
Auf diesem Kanal lässt sich der Grad der Deckkraft einstellen, welche der zu berechnende Punkt besitzen soll. Der Wert, welcher sich zwischen 1 und 0 einstellen lässt, wird dann mit der Dichte der Basis-Wolkenform multipliziert.

Grüner Kanal: der Prezipitation-Wert
Dieser Wert bestimmt die Höhe des Grades an Niederschlag, ergo ist mit diesem die Bildung von Regelwolken möglich. Erreicht wird dies durch die Berechnung des Wertes aus der Textur mit dem Absorptionskoeffizienten.

Für Niederschlag wird hier Prezipitation verwendet, wobei ein höherer Wert zu einer dunkleren Wolke führt.

Blauer Kanal: der Wolkentyp-Wert
Werte aus dem blauen Kanal werden in der Density-Height-Funktion verwendet, um die Wolkenarten zu bestimmen. Stratus-Wolken sind auf dem Wert 0.0 definiert, 0.5 sind Stratocumulus und 1.0 sind Cumuluswolken. Die Erstellung der Weather-Map kann manuell oder automatisch generiert werden. Damit lassen sich cineastische Wetterbedingungen oder unterschiedliche Wetterzonen erstellen. Auch hier lässt sich wiederum Perlin-Noise verwenden um harte Übergänge zwischen den Wolken zu vermeiden.

5.4 Cirrus Wolken

Anders als die Wolken im niedrigeren Höhen werden die Cirrus-Wolken, statt mit dem Ray March Algorithmus, als 2D-Texturen hinter den anderen Wolken am Himmel gerendert.

5.5 Wolkenbeleuchtung

Um eine realistische Beleuchtung zu erzeugen, müssen viele komplexe und aufwendige Gleichungen berechnet werden. In dieser Thesis wird aber auch die Echtzeitfähigkeit untersucht, somit wird die Beleuchtung approximiert werden müssen. Deshalb wird die Berechnung der Emission nicht durchgeführt, da in dieser Thesis keine Gewitter mit Blitzen simuliert werden. In diesem Fall kann die Berechnung sich auf die Absorption und Scattering konzentrieren.

Dabei wird die Beleuchtung in drei Teilberechnungen aufgeteilt: Das Directional-Scattering, das Out-Scattering, welches die Absorption inkludiert, sowie das In-Scattering. Das Directional-Scattering ist verantwortlich für das angesprochene Silver-Lining der Wolken. Dafür wird die Henyey-Greenstein-Funktion verwendet, welche sowohl für Forward und Backward-Scattering verwendet werden kann. Dies macht sie zur passenden Funktion zur Bestimmung des Scatterings, da Wolken eine sehr hohe Wahrscheinlichkeit für anisotrope Streuung besitzen. Daher wird jedem Berechnungspunkt $g > 0$ gesetzt, um Forward-Scattering zu erhalten und somit Wolken, die sich in der Nähe der Sonne befinden, stärker zu beleuchten.

Da Out-Scattering und Absorption die Radiance verringern, können diese beiden Teile der Lichtberechnung zusammengefasst werden, was sich später in der Umsetzung anbietet. Diese Funktion verwendet das Beer-Lambert-Gesetz, auch kurz Beer's Law. In der Physik beschreibt die Funktion die Abschwächung der Radiance bei dem Durchstrahlen

von Material. Im Kontext des Volume Rendering ermöglicht die optische Dicke eines Mediums die Berechnung einer Lichtmenge an dem zu bestimmenden Punkt. Dabei besteht eine Abhängigkeit der Transmission zur optischen Tiefe und somit zur Lichtenergie.

$$Beer's\,law \quad T = e^{-t} \tag{17}$$

Die Transmission nimmt mit optischer Tiefe des Mediums exponentiell ab. Somit erreicht, mit zunehmender Tiefe innerhalb der Wolke, immer weniger Lichtenergie den zu berechnenden Punkt.
Das In-Scattering berechnet zuletzt den Puderzucker-Effekt, welcher bei runden und dichten Regionen der Wolken auftritt. Da dieser Effekt das Gegenteil des Beers-Gesetzes verursacht, nämlich das Spalten heller als Ausbuchtungen wahrgenommen werden, kalkuliert die neue Funktion das Inverse des Beers-Gesetzes. Dadurch wird der In-Scatterkoeffizient berechnet, ergo wie sehr sich ein Punkt im In-Scattering beteiligt.

$$inverses\,Beer's\,law \quad T = 1 - e^{-t*2} \tag{18}$$

5.6 Regenwolken

Mit dem Wert aus dem blauen Kanal der Weather-Map werden die Regenwolken definiert. Um dies im Renderingprozess umzusetzen, wird der Prezipitationswert mit der berechneten Dichte multipliziert und das Ergebnis anschließend an die Lichtberechnungsfunktionen übergeben. Aus diesen Termen wird die finale Berechnung durchgeführt, welche die Lichtenergie eines Punktes berechnet. In der nachfolgenden Gleichung ist die Lichtenergie E, der Prezipitation-Wert sei p, dl steht für den Dichtewert zwischen dem Punkt und der Lichtquelle, der zuvor genannte Parameter g steht für die Eccentricity und θ für den Winkel der zwischen Lichtquelle und der Blickrichtung existiert.
Das Konzept der Lichtenergieberechnung beruht auf den Ansatz von Schneider [SV15] und wird in der Implementation optimiert.
Lichtenergie:

$$E = Beer's\,law * inverses\,Beer's\,law * Henyey - Greenstein - Funktion \tag{19}$$

$$E = 2.0 * e^{-dp} * \frac{1}{4\pi} * \frac{1 - g^2}{1 + g^2 - 2g * \cos(\theta)^{\frac{3}{2}}} \tag{20}$$

5.7 Himmel

Um den Himmel zu rendern, wurde das analytische Himmelsmodel von Preetham [PSS99] in vereinfachter Form [HP+02] angewendet. Um das atmosphärische Scattering zu berechnen, eignet sich, laut den Arbeiten von Hoffmann und Preetham, folgende Gleichung.

$$L_in(s,\theta) = \frac{\beta_{Ray}(\theta) + \beta_{Mie}(\theta)}{\beta_{Ray} + \beta_{Mie}} E_{Sun} * (1 - e^{-(\beta_{Ray}+\beta_{Mie})*s)}) \qquad (21)$$

Diese setzt sich aus Rayleigh- und Mie-Scattering zusammen, welche respektive die Funktionen β_{Ray} und β_{Mie} in der Gleichung haben. Dabei wird in der Mie-Funktion die Henyey-Greenstein verwendet, um das Mie-Scattering approximiert zu berechnen.
Der Wert E_{Sun} gilt hier als Konstante und beschreibt die Farbe der Sonne. Die Verwendung als Konstante vereinfacht die Rechnung, da dieser Faktor, welcher sich aus der zuvor genannten Absorption und dem Out-Scattering zusammensetzt, durch die Extinktion beeinflusst wird. Die β_{Ray} und β_{Mie} Konstanten beschreiben die Wellenlänge des Lichts auf Meereshöhe und den Brechungsindex der Luft. Da sich die Kamera in der Implementation auf Meereshöhe befindet, werden die Werte für diese Höhe verwendet [BN08]. Das Preetham ist das gängige Modell im Echtzeitrendering und es gibt viele Open-Source Ansätze zur Implementation. Als eine Alternative wäre hier das Hosek-Wilkie Verfahren [HW13] zu nennen .

6 Implementation

Nachdem die Konzepte erläutert wurden, wird nun die tatsächliche Implementation vorgestellt. Grundkenntnisse in OpenGL sollten für das Verständnis bekannt sein. In diesem Kapitel wird auf die Umsetzung der Struktur im System eingegangen. Anschließend werden die essentiellen Klassen behandelt, der Ray-Marching Alogrithmus sowie der Rendering Prozess.

6.1 Aufbau des Systems

Das Rendering System wurde durch die Verwendung von C++ und OpenGL umgesetzt. Neben den gängigen Klassen für die Kamera, Shader und Texturen gibt es weitere wie einen Noise-Generator. Es besteht die Möglichkeit die zwei essenziellen 3D-Texturen beim Start des Programms zu generieren oder als Dateien zu laden. Für den Fall der Generierung erstellt die Klasse GenNoise.cpp dementsprechend die Weather-Map (512×512), die Perlin-Worley-Textur $(128 \times 128 \times 128 \times 4)$ für die

Basis-Wolkenform und die Worley-Textur $(32 \times 32 \times 32 \times 3)$ für die Erosionstextur. In der Compute-Shader-Klasse werden die unterschiedlichen Compute-Shader erstellt und dispatched. Der Hauptanteil des Renderings befindet sich in der Cloudsystem-Klasse. In ihr werden die Texturen geladen, die Frame Buffer Objects (FBOs) und Shader initialisiert und ImGui [CI] aufgesetzt. Zusätzlich werden weitere wichtige Parameter für die jeweiligen Shader berechnet.

6.2 Renderprozess

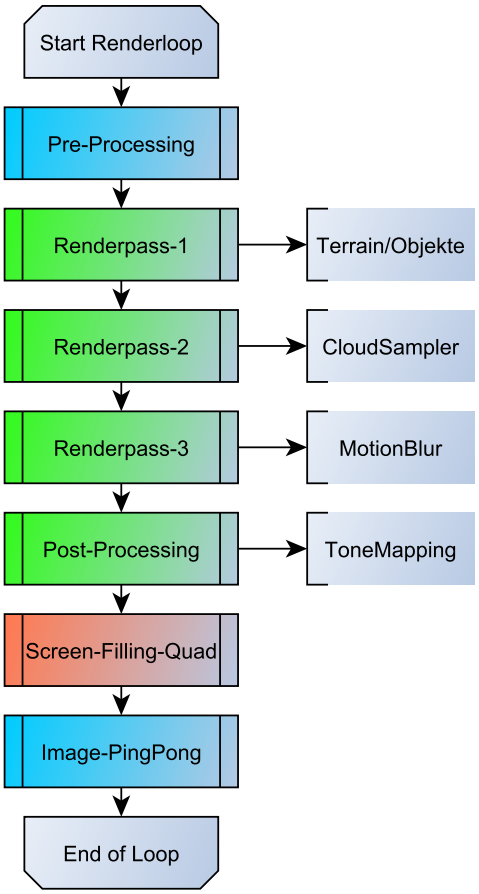

Abbildung 14: Renderprozess

Anhand der Abbildung 14 wird die Renderschleife verdeutlicht. Zu Beginn jeder Iteration werden alle nötigen Uniformberechnung an die Shader übermittelt. Dabei unterscheidet sich die erste Iteration von allen darauffolgenden, denn durch das Image-PingPong wird der MotionBlur-Shader übersprungen. Denn beim Image-PingPong wird im Shader das LastFrameImage verwendet, welches bei der ersten Iteration noch schwarz ist.

Das Ergebnis hiervon wäre ein schwarzes Endbild. Die weiteren Schritte im Renderprozess werden folgend näher erläutert.

6.2.1 Pre-Processing

Zu Beginn des Programms wird das Cloudsystem initialisiert und dabei die GLFW3 Bibliothek verwendet. Darin wird die Kamera erstellt und Variablen gesetzt, die für den Renderprozess initial benötigt werden. Die Parameter sind den Bereichen Wolken, Licht, Debugging und Rendering zuzuordnen, wobei diese zusätzlich über anpassbare Variablen wie die Wolkendichte verfügen, die anschließend im GUI zu ändern sind. Als Nächstes besteht die Möglichkeit der Generierung der benötigten Texturen, welche per Compute-Shader unter der Verwendung von Perlin- sowie Worley-Noise berechnet und als Textur hinterlegt werden. Unter den Abbildungen 15,16,17 und 18 sieht man eine generierte Textur der Basisform, sowie bei den Grafiken 19,20 und 21 der Erosion. Zu Demonstrationszwecken werden auch händisch erstellte Texturen verwendet. Die jeweiligen Shader und Compute-Shader werden attached und gelinkt, wobei auf diese folgend noch genauer eingegangen wird. Mit den Shadern wird ebenfalls das Render-FBO mit drei Colorattachments generiert, wobei ebenfalls für das später verwendete Textur-PingPong ein Array mit dem Texturindex gefüllt wird.

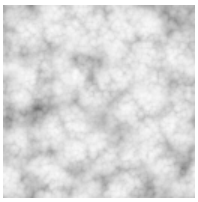

Abbildung 15: PerlWorley-Noise

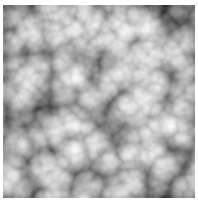

Abbildung 16: Worley-Noise im G-Kanal

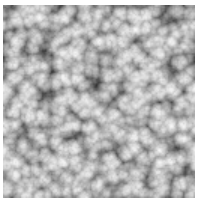

Abbildung 17: Worley-Noise im B-Kanal

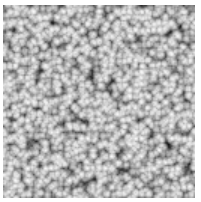

Abbildung 18: Worley-Noise im A-Kanal

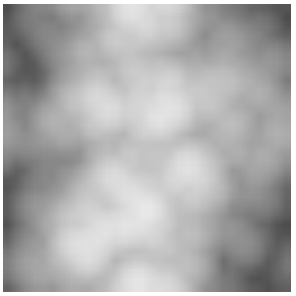

Abbildung 19: Worley-Noise im R-Kanal

Abbildung 20: Worley-Noise im G-Kanal

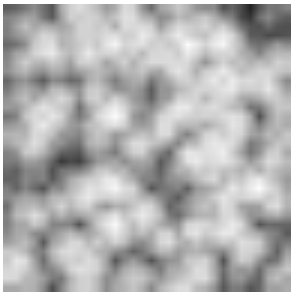

Abbildung 21: Worley-Noise im B-Kanal

Zuletzt wird eine Halton-Sequenz Rechnung [Hal62] durchgeführt und die Uniforms für die Shader gesetzt. Um im späteren Rendering mögliche Artefakte wie Undersampling zu umgehen, wird in der Computergrafik Jittering angewendet.

Beim Jittering werden Berechnungen, beispielsweise dem Berechnen des Treffpunkts, zufällige Werte hinzugefügt. Um diese Technik anwenden zu können, bedarf es einer Anzahl randomisierter Zahlen, wobei ein gängiges Beispiel die Verwendung einer BlueNoise-Textur [MG16] wäre. In dieser Thesis wird stattdessen die Halton-Sequenz verwendet [Ken67, CKK18]. Diese Vorgehensweise erweist sich als praktikabel in der Anwendung mit TXAA und kann somit auch für das Jittering [Ped16] der Hauptberechnungen verwendet werden. Die Berechnung der Zahlen findet auf der CPU statt und die Verwendung im jeweiligen Shader.

6.2.2 Renderpasses

Das Renderingsystem enthält drei Renderpasses, die für unterschiedliche Berechnungen eingeteilt werden. Im ersten Pass könnten Objekte und Terrain gerendert werden. In dieser Thesis liegt der Fokus auf dem Rendering von Wolken und somit wird vom ersten FBO eine schwarze Textur an den zweiten Renderpass gegeben. Im Zuge der Optimierung wurde dieser Schritt entfernt und nur ein FBO verwendet. Innerhalb des zweiten Renderpasses wird das im Pre-Processing genannte FBO und ein Compute-Shader für das Rendern der Wolken verwendet. Im nachfolgenden Renderpass wird das erechnete Bild zur temporalen Reprojektion an den zweiten Compute-Shader weitergegeben, welcher das Bild vervollständigt und es an das Post-Processing weiter reicht.

6.2.3 Post-Processing

Sobald die Reprojektion abgeschlossen ist, wird an dieser Textur das Uncharted Tonemapping [Hab10] durchgeführt. An dieser Stelle bietet sich die Implementierung von Crepusculur Rays und TXAA-Filtern an. Zum einen, um die finalen Wolken weicher wirken zu lassen ohne die Gefahr von Detailverlust und der Reduktion von Artefakten, zum anderen, um die Dynamic-Range des Bildes einzuschränken, da dieses sonst zu grell wirken würde. Dieses Bild wird dann per Screen-Filling-Quad auf den Bildschirm gerendert.

6.3 Wolkenberechnung

Für die Berechnung der Wolken wird, wie zuvor genannt, ein Compute-Shader, dessen Ablauf in der Abbildung 22 illustriert ist, verwendet. Dieser erhält per Uniformvariablen alle Sampler, zwei Image-Units

und weitere Parameter wie, unter anderem, die Kameradaten. Für das Rendering werden drei Kugeln benötigt, die Welt, auf der sich die Kamera sich befindet, die innere und die äußere Atmosphäre. Es wird dabei allerdings keine Geometrie verwendet, sondern einfache Radien. Daher befindet sich die Kamera immer auf Meereshöhe, wofür der Erdkern als $vec3(0.0, -earthRadius, 0.0)$ definiert wird. Zwischen den beiden Atmosphären befindet sich der Bereich, in dem der Ray-Marcher die volumetrischen Wolken rendert. Ebenfalls als Image-Unit sind die image2D-Units CurrentFrame, LastFrame und GodRayImage vorhanden. In diesem Compute-Shader wird allerdings hauptsächlich CurrentFrame gefüllt und in Sonderfällen auch das GodRayImage.

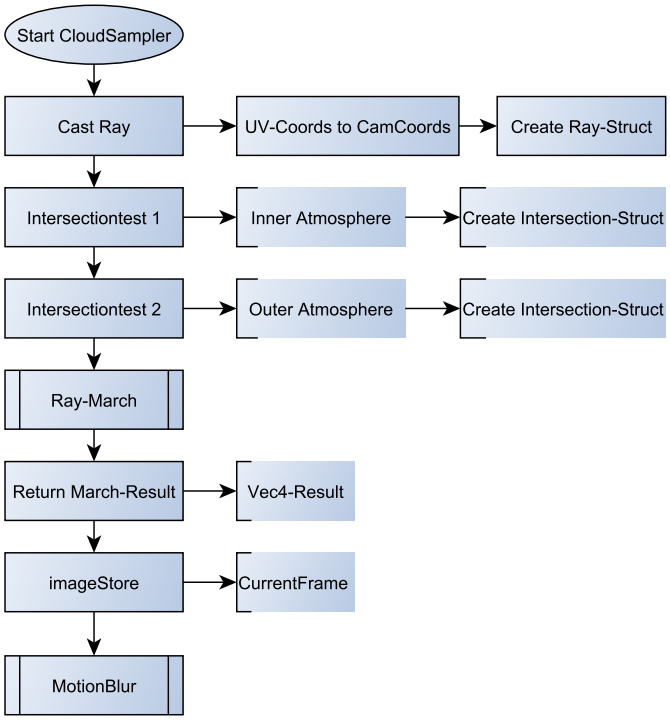

Abbildung 22: Compute-Shader für das Wolkenrendering

Im Compute-Shader wird zuerst der Pixel bestimmt, der zu der passenden UV-Koordinate des CurrentFrame-Images gehört. Daraus wird anschließend, in Abhängigkeit zu den Kameradaten, ein Punkt in Weltkoordinaten erstellt. Während zunächst aus der View-Matrix Kameravektoren extrahiert werden, erfolgt gleichzeitig die Berechnung der NDC-Raumpunkte der UV-Koordinaten, welche anschließend zu Kamera- und Weltkoordinaten konvertiert werden. Aus diesen ist die Erstellung eines Strahles möglich, der über Ursprung und Richtung verfügt, und mit dem, gemeinsam mit den zwei Atmosphären, zwei Intersection-Tests für Kugeln berechnet

werden. Eine Möglichkeit der Implementierung findet sich bei Schneider [Sch16]. Das Ergebnis dieser Methoden sind zwei Structs, die den Start und Endpunkt des Ray-Marchers bestimmen.

6.4 Ray-Marching

Im folgenden Programmablaufplan in der Abbildung 23 erkennt man die jeweiligen Schritte der Umsetzung ohne jegliche Optimierung.

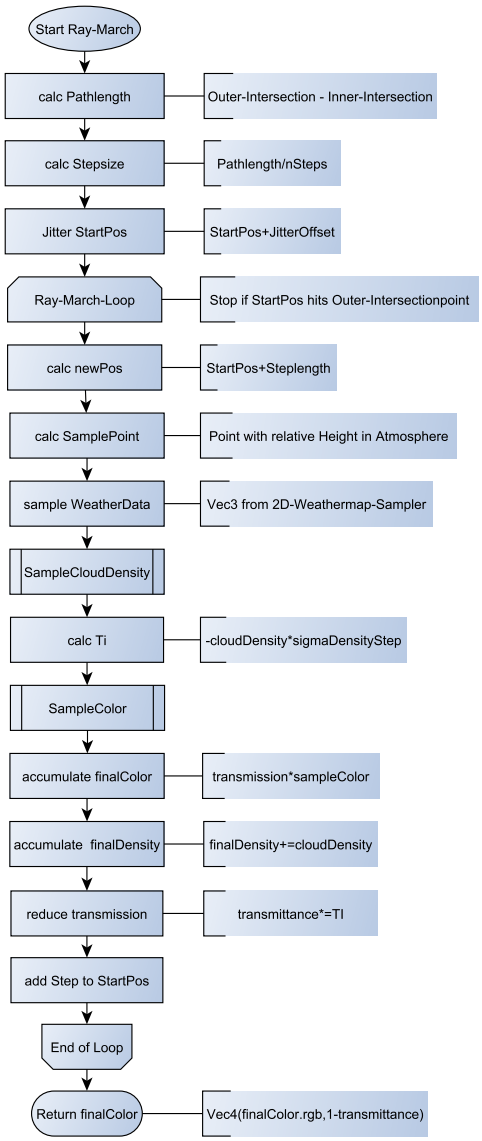

Abbildung 23: Ray-Marching-Ablauf

Die Eingabeparamter beschränken sich auf den Strahl und die beiden Intersections. Zu Beginn wird die Pfadlänge zwischen den beiden Intersectionspunkten berechnet, woraufhin die Schrittlänge ermittelt wird, indem die Pfadlänge durch die Anzahl der Tastschritte dividert wird. Anschließend werden für die weitere Berechnung Parameter initialisiert. Bevor es in die Ray-March-Schleife geht, wird die Lichtrichtung und der Blickwinkel zur Kamera berechnet, die später benötigt werden.

Anschließend wird ein Jitter-Offset mit der Halton-Sequenz berechnet, der auf die Startposition addiert wird. Zu Beginn der Schleife wird mithilfe der Startposition die neue Position mit folgender Rechnung kalkuliert:

```
pos = ray.origin + t * (ray.direction + vec3(jitterLocation.x, (jitterLocation.x + jitterLocation.y)*
jitterfactor, jitterLocation.y));
```

Abbildung 24: Jittering

Das t bildet hierbei die Länge des momentanen Strahls, welcher mit der angepassten ray.direction multipliziert und zur berechnenden Position hinzugefügt wird. Es ist ein $JitterFactor$ implementiert, mit welchem sich das Jittering der Y-Koordinate verstärken lässt, was zu besseren Ergebnissen bezüglich des Undersamplings führt. Momentan wird für jede Iteration ein Jittering durchgeführt.
Für die weiteren Berechnungen wird nun die relative Höhe der Position in der Atmosphäre errechnet und daraus ein neuer Punkt erstellt. Anhand dieses 3D-Punktes wird wiederum ein Punkt auf die 2D-Weather-Map projiziert. Somit enthält dieser berechnete Vektor die Wetterdaten für die weiteren Berechnungsschritte. Diese zwei Vektoren werden als Eingabeparameter für die SampleCloudDensity-Funktion übergeben, auf die später eingegangen wird. Als Ergebnis gibt die SampleCloudDensity-Funktion einen Float-Wert zurück, welcher zur Berechnung von Ti, also des Transmittance-Index, verwendet wird.
Die zuvor berechneten Werte werden zur Berechnung der Farbe mit SampleColor() des betrachtenden Punktes genutzt und zu einem Wert addiert, der, nachdem die Schleife beendet ist, als finale Farbe ausgegeben wird. Dabei wird diese mit der globalen $Transmittance$ multipliziert, wodurch die Gewichtung eines Punktes immer geringer wird, je tiefer sich dieser in der Wolke befindet. Die finale Farbe wird als Vec4 ausgegeben, wobei der Alphawert mit $1 - Transmittance$ definiert wird. Daraus resultiert das die Transparenz der Wolken an den Rändern steigt, da dort der Transmissionswert hoch ist.

6.5 SampleCloudDensity

Eine der essenziellen Funktionen des Ray-Marcher ist die SampleCloudDensity-Funktion. In folgender Abbildung 25 wird der Ablauf dargelegt und anschließend beschrieben.

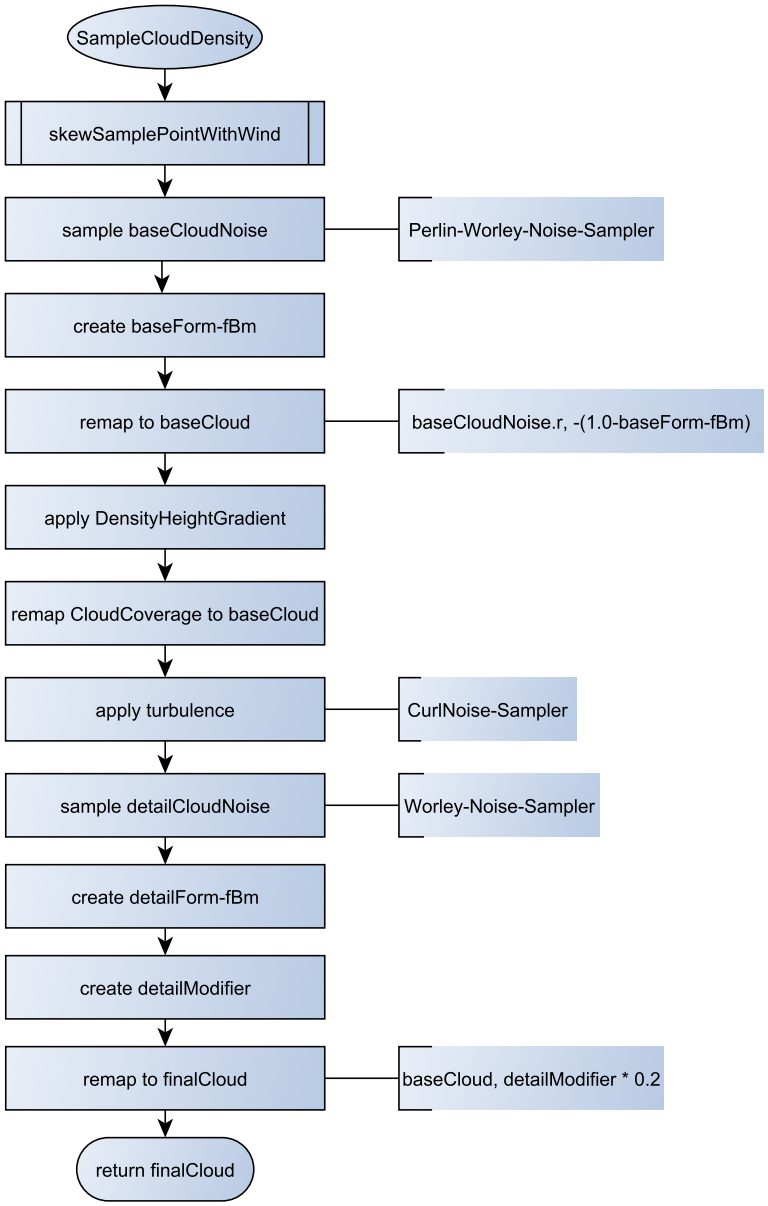

Abbildung 25: SampleCloudDensity-Ablauf

Als Eingangsparameter verwendet die Funktion einen 3D-Sample-Punkt, eine relative Höhe, den 3D-Wetterpunkt und ein Level-Of-Detail-

Wert als Integer. Als Sample-Punkt dient hier der 3D-Punkt, der für die Projektion auf die Weather-Map verwendet wurde. Zunächst wird der Sample-Punkt mit der Funktion skewSamplePointWithWind() in Richtung eines globalen Windes bewegt. In dieser Funktion wird dem Sample-Punkt ein simpler Offset in eine Windrichtung addiert. Zusätzlich wird beim Animieren des Windes ein weiterer kleiner Offset entlang der y-Achse addiert, damit dieser den optischen Eindruck des Aufsteigens hat.
Mit diesem verschobenen 3D-Punkt wird per TextureLod() auf der Basisform-Textur, also dem Perlin-Worley-Noise, ein Vec4 berechnet, mit dem die eigentliche Berechnung der Wolkendichte beginnt. Aus den GBA-Kanälen, welche die Worley-Noises der Basisform-Textur sind, wird ein Basisform-fBm erstellt, wobei die unterschiedlichen Kanäle verschieden stark gewichtet sind. Damit erhält die Basisform einige Details. Mit diesem und dem R-Kanal des 4er-Vektors wird mit der Remap-Funktion der Basis-Dichtewert berechnet.

```
199  float remap(in float value, in float original_min, in float original_max, in float new_min, in float new_max)
200  {
201      return new_min + (((value - original_min) / (original_max - original_min)) * (new_max - new_min));
202  }
```

Abbildung 26: Remap-Funktion

Mit dieser Remap-Funktion von Schneider [SV15] aus Abbildung 26 wird ein Wert insofern angepasst, dass dieser in Relation zu seinem alten Minimum und Maximum ein neues Minimum und Maximum erhält, dabei aber seine Relation zu diesen erhalten bleibt. Ebenfalls wird jetzt die zuvor genannte DensityHeightGradient-Funktion benötigt, welche mithilfe der relativen Höhe und dem 3D-Wetterpunkt den Gradienten berechnet, der mit dem Basis-Dichtwert multipliziert wird. Um die finale Basisdichte des Wolkenpunkts zu erhalten, wird der momentane Basisdichte-Wert mit der Wolkenabdeckung, welche sich im R-Kanal des 3D-Wetterpunktes befindet, kombiniert und anschließend mit dieser multipliziert. Nachdem somit die Berechnung der Basisform abgeschlossen ist, muss jetzt die zweite Noise-Textur verwendet werden, um der Wolkenoberfläche durch Erodierung weitere Details hinzufügen zu können.
Bis auf die Verwendung der hochfrequenten Worley-Textur ist die Vorgehensweise identisch mit der der Basisform. Zuvor wird dem Sample-Punkt eine Turbulenz hinzugefügt, wofür eine Curl-Noise-Textur mit TextureLod angewendet wird. Anschließend wird die 3D-Worley-Textur gesampled, mit der ein hochfrequentes fBm erstellt werden kann. Dieses wird zusammen mit seinem Inversem in einem Modifier linear interpoliert, damit der Wolkenboden flüchtige Kanten besitzt und der obere Teil der Wolken wellig bleibt. Das resultierende Ergebnis wird dann zwischen 0 und 1 geclampt und kann für die Berechnung der Farbe verwendet werden.

6.6 SampleColor

Um die Farbe des Punktes, wie in Abbildung 27, bestimmen zu können, wird von dieser Stelle des Programms aus eine Funktion aufgerufen, die einen Ray-Marching Algorithmus zur Lichtquelle durchführt. Dadurch erhält man die benötigte Dichte zwischen Lichtquelle und Sample-Punkt, wodurch es möglich ist die weiteren Lichtberechnungen durchzuführen. Diese und weitere Parameter werden in die GetLightEnergy-Methode von Schneider [Sch17] übergeben, dessen Visualisierung in der Abbildung 28 zu finden ist.

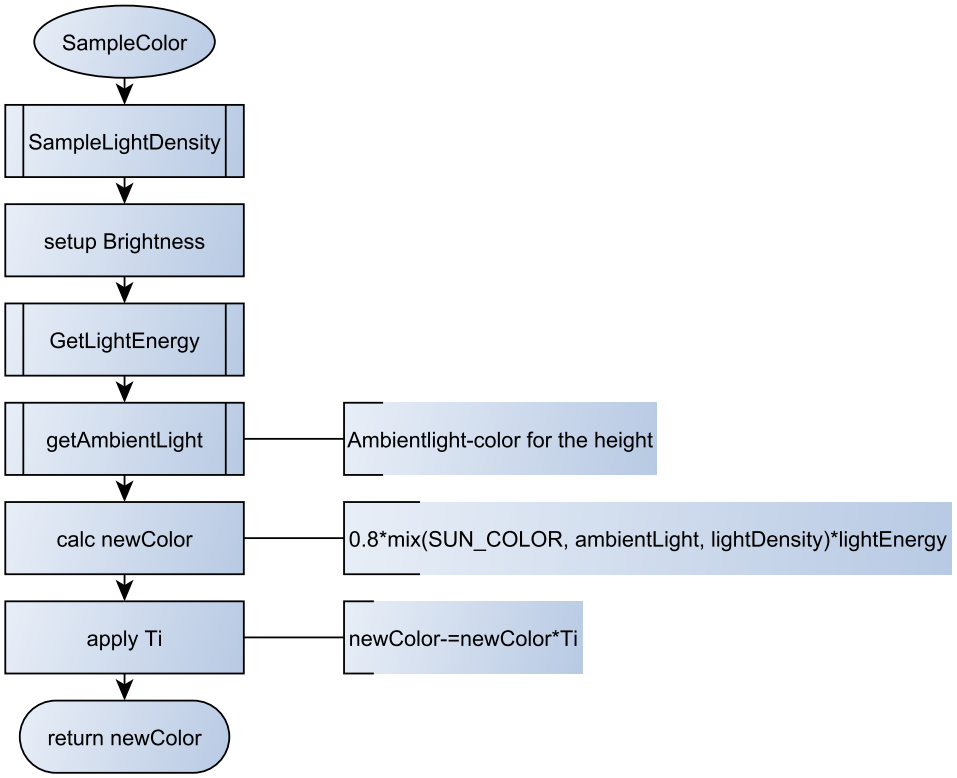

Abbildung 27: SampleColor-Ablauf

Darin wird, wie im vorherigen Kapitel erläutert, der G-Kanal des 3D-Wettervektors als Absorptionsfaktor verwendet, während die drei Berechnungskoeffizienten berechnet werden, welche am Ende die Lichtenergie ergeben. Mit dem Directional-Scattering-Koeffizienten wird das Silver-Lining kontrolliert. Dabei werden zwei Henyey-Greenstein-Funktionen angewendet, die mit einem max()-Operator kombiniert werden. Dadurch wird, laut Schneider, verhindert, dass Wolken, die in einem Winkel von 90° zur Sonne stehen, sich zu stark verdunkeln. Diesem artistischen

Ansatz wurden Parameter hinzugefügt, die die Kontrolle von Stärke sowie Reichweite des Effektes ermöglichen. Der Out-Scatter-Koeffizient oder auch Absorption-Koeffizient wiederum verwendet die vorgestellte Beer-Lambert-Funktion, kurz das Beers Law. Da in dieser Funktion nur die Abschwächung des Lichts berechnet wird, nicht aber der In-Scatter-Koeffizient, würde es die Wolken zu sehr verdunkeln. Auch hier werden wiederum zwei Beer-Lambert-Funktionen mit einem Max-Operator kombiniert und anschließend remapped.
Für die dritte Wahrscheinlichkeit, dem In-Scatter-Koeffizienten, wird wiederum eine Zwei-Term-Funktion angewendet. Der erste Teil der Funktion bezieht sich darauf, wie viel Licht potenziell zu dem Sample-Punkt gestreut werden kann. Aufgrund der Tatsache, dass ein tiefer in der Wolke gelegener Punkt über eine höhere Wahrscheinlichkeit für die Erhaltung gestreuten Lichtes verfügt als ein Punkt, welcher sich am Rand befindet, wird die, den Punkt umgebene, Dichte berechnet.
Mit Hilfe eines selbstdefinierten Höhengradienten sorgt der zweite Term der Funktion dafür, dass das In-Scattering mit steigender Höhe verringert wird. Beide Terme, sowie das Ergebnis daraus werden mit den anderen Koeffizienten multipliziert, welches zu der Lichtenergie resultiert.

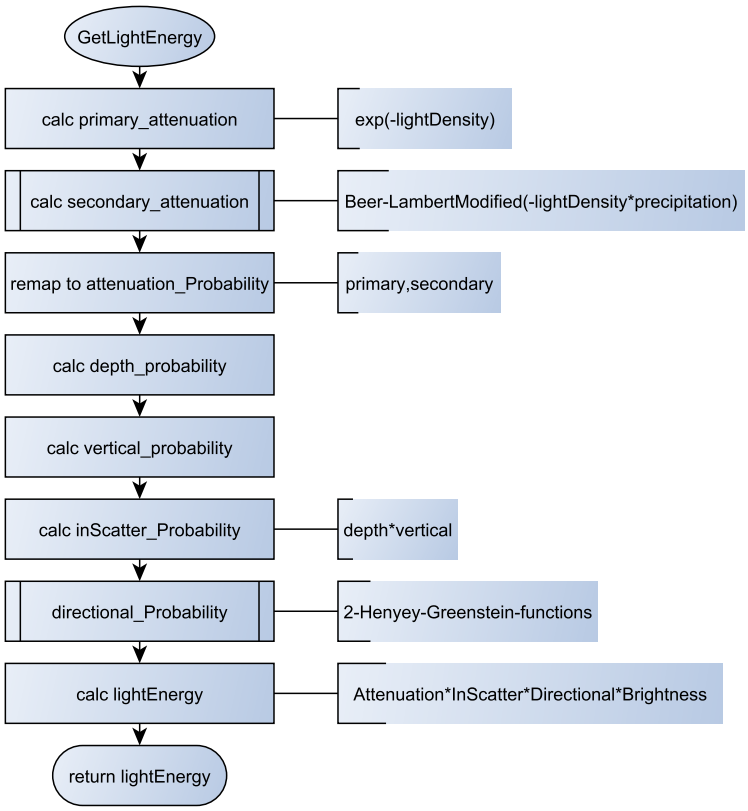

Abbildung 28: GetLightEnergy-Ablauf

Die Lichtenergie wird mit dem resultierenden 3D-Vektor multipliziert, der sich aus dem ambienten Licht, der Lichtdichte und der Sonnenfarbe ergibt. Dabei steht hier die Lichtdichte im Vordergrund, denn je niedriger diese ist, desto stärker der Einfluss auf die Farbe der Sonne. Das Ergebnis wird anschließend ein weiteres Mal abgeschwächt, indem es mit der Transmission multipliziert wird.

6.7 SampleLightDensity

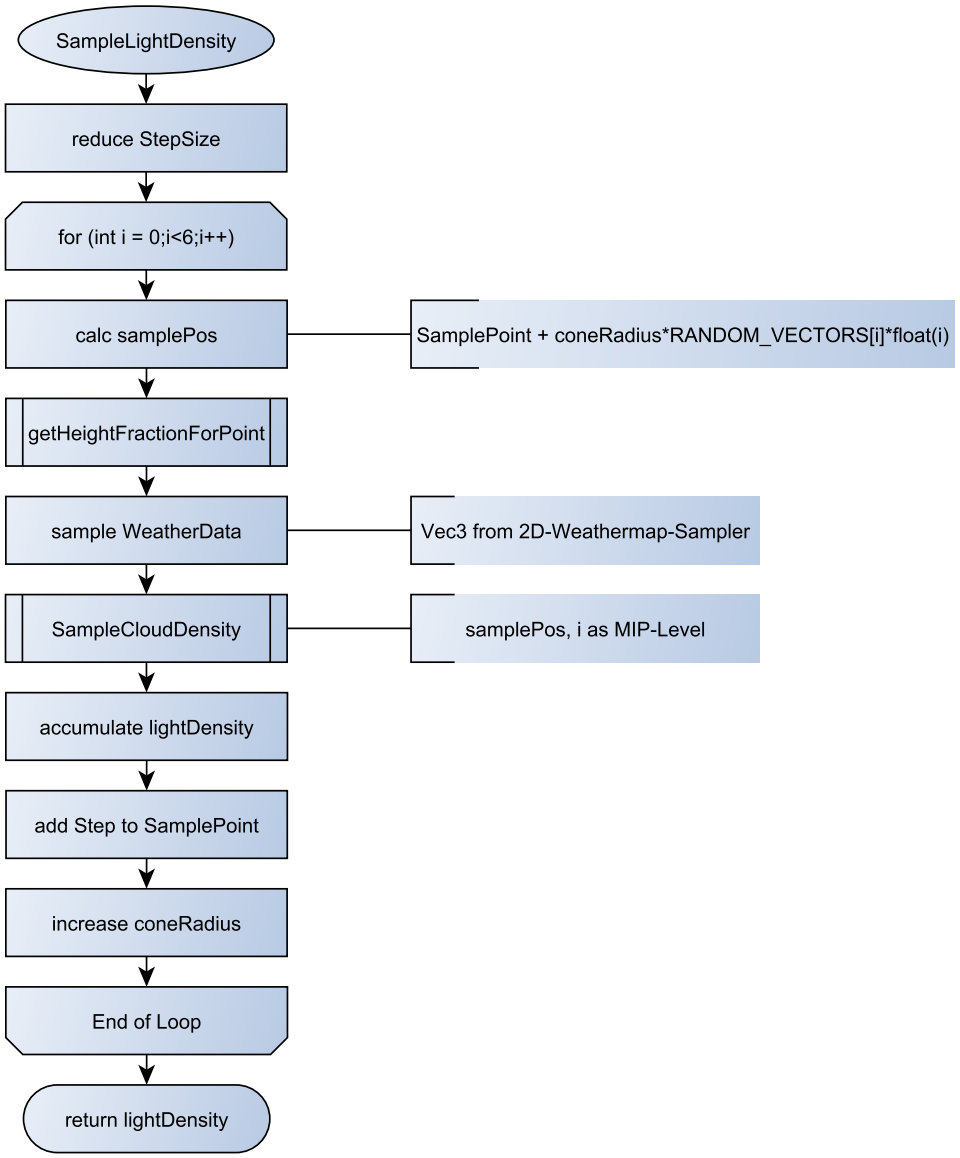

Abbildung 29: SampleLightDensity-Ablauf

Diese Funktion dient zur Bestimmung der Menge an Licht, die den Sample Punkt erreicht. Dafür wird ein Trichter in Richtung der Lichtquelle aufgespannt, in dem sich sechs zufällige Punkte befinden. Um diesen Trichter zu erstellen, wird ein Noise-Kernel genutzt, welcher zusätzlich mit dem steigenden Index multipliziert wird, um den Trichterradius zu erhöhen. In einer Schleife wird für jeden Punkt die Dichte mit der zuvor erläuternden SampleCloudDensity() berechnet und akkumuliert. Da die Samples auf eine Anzahl von sechs beschränkt werden, besteht die Möglichkeit der Entstehung von Artefakten. Diese können durch ein Senken des Mip-Levels mit fortschreitendem Index umgangen werden.

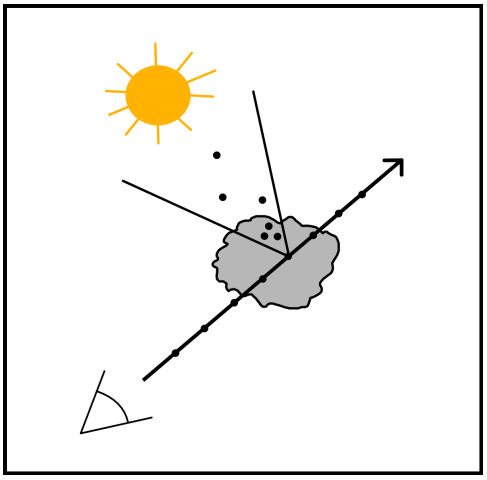

Abbildung 30: Adaptive Stepsizes

7 Optimierung

Für die Implementierung wurden verschiedene Optimierungen hinzugefügt, welche in diesem Kapitel vorgestellt werden.

7.1 Early Ray Termination

Im Normalfall wird der Ray-March Algorithmus bis zu seiner vollen Länge ausgeführt und erst dann wird die akkumulierte Dichte zurückgegeben. Befindet sich der Strahl in einer sehr langen oder dichten Wolke erreicht die akkumulierte Dichte jedoch schon vor dem Erreichen des Endpunktes die Maximaldichte. Tritt dies ein konvergiert in der Implementation die Transmission gegen null, was bedeutet, dass es nicht möglich wäre weiter in die Wolke hineinzusehen.

```
if (accumColor.a <=1.0 || transmittance > 0.01)
```

Abbildung 31: Early-Ray-Termination

Um überflüssige Rechenschritte zu vermeiden, wird in dem Ray-March-Algorithmus eine Abbruch-Kondition 31 hinzugefügt. In jedem Schritt der Schleife wird zu Beginn geprüft, ob die akkumulierte Dichte kleiner 1.0 oder der Transmissionswert kleiner als 0.01 ist. Ist dies nicht der Fall, wird die Schleife beendet und die berechnete Farbe wird zurückgegeben. Ebenfalls kommt es zu keinem Ray-Marching, sofern das Skalarprodukt der ray.direction und dem Up-Vektor negativ ist, somit würde die ray.direction unter den Horizont zeigen. Gemäß diesem Falle erhalten diese Pixel ein Ozeanblau. Alle anderen erhalten, bevor die Wolken gerendert werden, ihre Farbe per Preetham-Implementation.

7.2 Adaptive Steps and Stepsize

Die Qualität des Ray-Marching Algorithmus hängt sehr stark von der Anzahl und der Größe der jeweiligen Abtastschritte ab. Mehr Schritte führen zu einem realistischeren Erlebnis, benötigen dafür aber auch viel Rechenleistung. Da in dieser Thesis das Echtzeitrendering als Ziel gesetzt wurde, muss Wert auf eine effiziente Rechnung gelegt werden. Wie auf der Grafik 32 zu erkennen, ist die logische Konsequenz eine Festlegung der Schrittanzahl im Zusammenhang mit der Länge des Rays. Denn um eine äquivalente Qualität zu erhalten, benötigt ein kürzerer Strahl weniger Abtastungen als ein längerer. Somit wird zwischen den minimalen und maximalen möglichen Schritten linear interpoliert und angewendet.

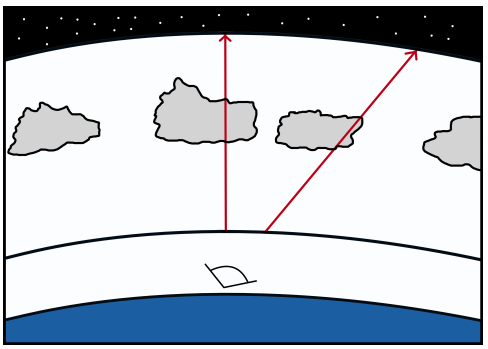

Abbildung 32: Adaptive Stepsizes

Zusätzlich lassen sich weitere Abtastungen vermeiden, indem zuerst geprüft wird, ob sich an dem Abtastpunkt überhaupt eine Wolke befindet. Wurde keine Wolke getroffen, ist der Cloudtest folglich gleich 0, wird die

Schrittgröße temporär verdoppelt und somit die Abtastungen eines Strahles, der auf keine Wolke trifft, halbieren. Wird jedoch in einem Schritt eine Wolke getroffen, wird der Cloudtest nach den Berechnungen der Dichte auf 0 gesetzt und der Abtastpunkt um einen Schritt zurückgesetzt, um gegebenfalls übersprungene Wolken zu rendern.

7.3 Cheap Sampling

Wie bereits zuvor erwähnt werden die Abtastschritte getestet, indem dort die Dichte der Wolke berechnet wird. Dieser Vorgang lässt sich dahingehend optimieren, dass die Berechnung nach der Basisdichte der Wolken per Flag kontrolliert wird. Somit kann die Verwendung der hochfrequenten Worley-Textur gezielt verwendet werden, sobald sie auch erforderlich ist. Die Umsetzung geschieht durch die Abfrage in der Abbildung 33 des Boolschen Werts $expensive$ und $finalCloud$.

```
561     float finalCloud = baseCloudWithCoverage;
562     //Details
563     if (expensive==true && finalCloud>0){
```

Abbildung 33: Cheap Sampling

7.4 Cheap Lightning

Ähnlich wie beim Cheap Sampling können auch einige Rechenschritte bei der Lichtberechnung eingespart werden. Da die Lichtberechnung die Voraussetzung der Farbberechnung ist, wird die Optimierung im Ray-March-Algorithmus angewendet, wie in Abbildung 34 zu sehen ist. Denn das Licht muss nur berechnet werden sofern der Punkt sich in einer Wolke befindet. Somit wird erst mittels Cheap Sampling geprüft, ob der Punkt eine Dichte besitzt.

```
753     cloudTest = SampleCloudDensity(skewedSamplePoint, relativeHeight, false, weatherData, 0);
754     if (cloudTest > 0.0f)
```

Abbildung 34: Cheap Lightning

Ist die Dichte nun größer 0 wird das teure Sampling angewendet und die Licht- sowie Farbberechnung wird ausgeführt. Andernfalls wird die Berechnung übersprungen.

7.5 Benutzer-Oberfläche

Die Anwendung wurde mit einer Oberfläche realisiert, um für Testzwecke bestimmte Uniform-Parameter in Echtzeit verändern zu können. Da-

für wurde ImGui, welches die Parameter in unterschiedliche Bereiche unterteilt, in die Anwendung implementiert. Wie zuvor genannt gehört dazu der Bereich Debugging-, Licht- und Wolkenparameter, die respektive in der Abbildung 35 zu sehen sind. Mit den Debuggingparametern lässt sich in der Renderszene erkennen, ob die Bindings der Texturen korrekt sind, ob die Intersections korrekt kalkuliert werden, sowie lassen sich weitere Tests durchführen. Im Bereich der Lichtparameter lassen sich die drei Hauptparameter der Lichtberechnung anpassen: die Eccentricity, die Silver-Intensity und der Silver-Spread. Ebenfalls ist es möglich, verschiedene Faktoren, wie beispielsweise die Coverage oder Basisdichte, anzupassen.

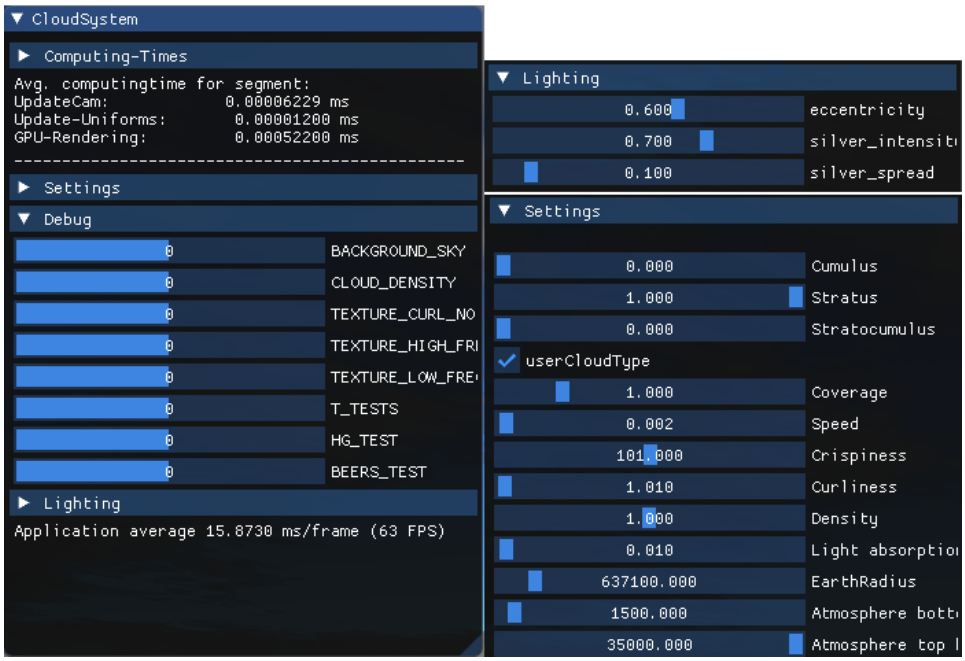

Abbildung 35: imGui-Settings

8 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Thesis behandelt. Zunächst einmal wird geprüft, ob dabei das Ziel von volumetrischen Wolken erreicht wurde und wie echtzeitfähig dieses System ist.

8.1 Silver-Lining

Das Silver-Lining wurde mit einer Kombination aus zwei Henyey-Greenstein-Funktionen berechnet. Die Abbildung 36 zeigt ein Beispiel einer echten Wolke, die Grafik 37 dagegen die aus dem Renderingsystem.

Abbildung 36: Cumuluswolke mit Silver-Lining

Vergleicht man die beiden Bilder miteinander, erkennt man eine Ähnlichkeit. Bei der Wolke aus 36 handelt sich um eine Wolke mit unterschiedlichen Dichteregionen. Dabei ist zu erkennen, dass das Silver-Lining im weniger dichten, oberen Teil der Wolke wesentlich breiter ist, als bei dem sehr dichten, rechten Bereich der Wolke. Dieses Phänomen erschließt sich aus dem Umstand, dass das Scattering mit der Dichte zunimmt. Um eine größere Übereinstimmung bei dichten Wolken zu simulieren, reicht ein Single-Scattering nicht aus, da das Silver-Lining aus mehrfachen Scattering resultiert. Dennoch hat der Ansatz des Directional-Scatterings befriedigende Ergebnisse für weniger dichte Wolken geliefert. Innerhalb der beiden Henyey-Greenstein-Funktionen kann das Silver-Lining über den Parameter g gesteuert und variabel gehalten werden. In den Grafiken 37, 38 und 39 wird veranschaulicht, wie die Änderung von g das Forward-Scattering beeinflusst.

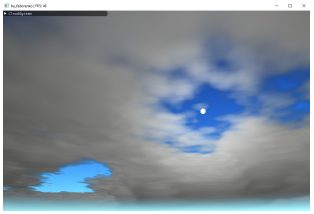

Abbildung 37: $g = 0{,}6$

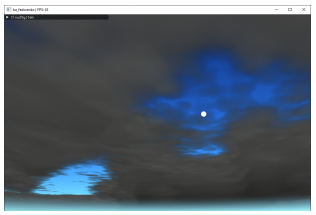

Abbildung 38: $g = 0$

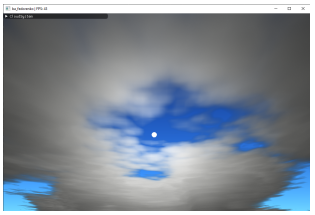

Abbildung 39: $g = 0{,}99$

8.2 Temporal Reprojektion

Damit das System weiterhin echtzeitfähig bleibt, wurden beim zweiten Renderpass immer nur ein Pixel innerhalb eines vierer Blocks gerendert, während die übrigen Pixel aus dem vorherigen Bild stammen. Dieses Vorgehen kann ein Ghosting der Wolken verursachen, welches mittels MotionBlur oder anderen Filtern umgangen werden kann. In den Grafiken 40 und 41 kann man das selbe Szenario ohne und mit der Reprojektion begutachten.

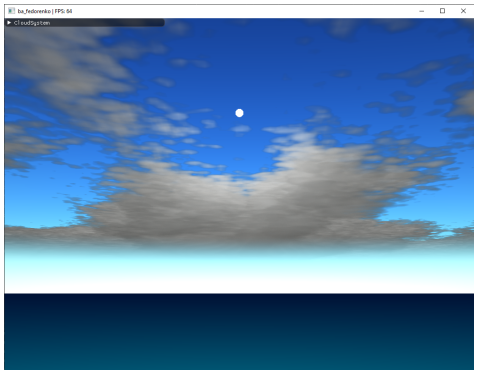

Abbildung 40: Rendering ohne Reprojektion

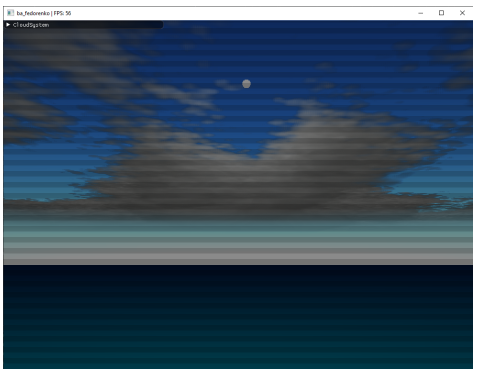

Abbildung 41: Rendering mit Reprojektion

8.3 Halton-Jittering

Mithilfe des Jitterings wird das Undersampling der Wolken reduziert, aber in der Implementation sind stattdessen andere Artefakte zum Vorschein gekommen. Jedenfalls konnten diese unerwünschten Effekte verringert werden, sofern man die Reprojektion aktiviert. Des Weiteren wurden die Wolken dadurch weicher.

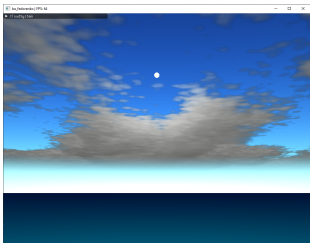

Abbildung 42: Rendering ohne Jittering

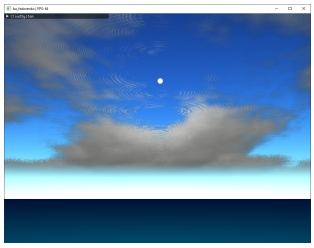

Abbildung 43: Jittering ohne Reprojektion

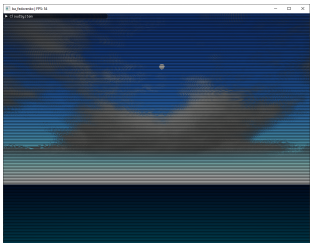

Abbildung 44: Reprojektion und Jittering

8.4 Optimierungs Ergebnisse

Das System, auf welchem die Thesis implementiert wurde, verwendet eine NVIDIA 770 GTX 4GB Gainward Phantom und einen i7-4790 mit 3,6GHz. Die Testdaten wurden jeweils für drei verschiedene Anwendungsfälle aufgzeichnet: Als erstes Blickrichtung zum Horizont, als zweites zur Sonne und zuletzt mit $90°$.

Zu Beginn die Testdaten für den Blick mit Richtung Horizont, da hier nur die Hälfte des Frames auf Wolken getestet werden muss, dafür aber die Rays in der Nähe des Horizonts die größte Länge erreichen werden.

Keine Optimierung	40 FPS	40,00 ms
Cheap Sampling	33 FPS	30,30 ms
Cheap Lightning	92 FPS	10,99 ms
Temporal Reprojektion (TR)	14 FPS	71,40 ms
Adaptive Stepsize/length	26 FPS	38,46 ms
Early Ray Termination	26 FPS	38,46 ms
Alle Optimierungen, ohne TR	91 FPS	10,90 ms
Alle Kombiniert	59 FPS	16,95 ms

Abbildung 45: Testergebnisse mit Blickrichtung Horizont

In der Tabelle 45 erkennt man schon deutlich, wie teuer die Lichtberechnung ist, denn sobald das Cheap Lightning eingebaut ist wird zwar die Qualität der Wolken schlechter, dafür verdoppeln sich die FPS.

Beim zweiten Test erhöht sich die Anzahl der zu berechnenden Pixel um 50% und somit der anspruchsvollste der drei Tests, da sich hier ebenfalls die längsten Rays im unteren Bereich befinden.

Keine Optimierung	13 FPS	76,90 ms
Cheap Sampling	16 FPS	62,50 ms
Cheap Lightning	53 FPS	18,86 ms
Temporal Reprojektion (TR)	12 FPS	83,30 ms
Adaptive Stepsize/length	13 FPS	76,92 ms
Early Ray Termination	13 FPS	76,92 ms
Alle Optimierungen, ohne TR	49 FPS	20,40 ms
Alle Kombiniert	51 FPS	19,60 ms

Abbildung 46: Testergebnisse bei $40°$C Neigung

An den Resultaten aus der Tabelle 46 lässt sich erkennen, das sich bei allen Implementationen die FPS deutlich verringern und keine 60 FPS mehr erreicht werden. Einzig das Cheap Lightning und die letzten beiden Kombinationsarten vermögen es auf über 30 FPS zu kommen.

Keine Optimierung	12 FPS	83,33 ms
Cheap Sampling	16 FPS	62,50 ms
Cheap Lightning	70 FPS	14,28 ms
Temporal Reprojektion (TR)	12 FPS	83,30 ms
Adaptive Stepsize/length	16 FPS	62,50 ms
Early Ray Termination	12 FPS	83,33 ms
Alle Optimierungen, ohne TR	55 FPS	18,18 ms
Alle Kombiniert	60 FPS	16,39 ms

Abbildung 47: Testergebnisse bei $90°$C Neigung

Bei den letzten Testresultaten aus der Tabelle 47 werden zwar auch alle Pixel auf Wolken überprüft und berechnet, dafür ist die Länge der Rays deutlich kürzer als bei den vorherigen Tests. Dadurch erreichen auch die einzelnen Strategien wieder höhere FPS, beziehungsweise niedrigere Millisekunden. An den Tests lässt sich eine eindeutige Leistungssteigerungen durch die Optimierungen erkennen, nur manche bieten mehr Raum zur Überarbeitung als andere. Die Berechnung des zu berechnenden Pixel für die Reprojektion scheint nicht ausgereift zu sein und sollte angepasst werden. Die wahre Stärke der Optimierung kommt aber erst zur Geltung, wenn alle, sofern man die Reprojektion auslässt, kombiniert werden, da hierdurch das beste optische sowie performante Ergebnis erreicht wird.

9 Fazit

Die vorliegende Studienarbeit behandelt das Echtzeitrendering volumetrischer Wolken. Um einen Überblick über dieses Thema zu verschaffen, wurden anfangs die verschiedenen Wolkentypen vorgestellt sowie die visuellen Eigenschaften dieser erläutert. Anschließend wurden die Theorie des Volume Renderings gemeinsam mit den physikalischen Formen, welche zur Berechnung von Licht und Himmeln dienen, ausgeführt, wobei auch auf die drei grundlegenden Terme der VRE eingegangen wurde, die in den nachfolgenden Kapiteln in Funktionen konzeptioniert wurden. Essenzielle Grundlage für diese Konzepte war hierbei die Umsetzung des Volume Renderings mit dem Ray-Marching Algorithmus. Dieser durchschritt zwei Noise-Texturen, um daraus die Beschaffenheit der Wolken erstellen zu können.
Die Implementation dieser Konzepte resultierte in einem Renderingsystem mit grafischer Oberfläche, welches volumetrische Wolken mit den vorher angesprochenen visuellen Eigenschaften darstellen kann. Dabei können auf dem vorgestellten System 30 FPS erreicht werden. Somit ist es möglich, bei leistungsfähigeren Systemen weit über 60 FPS zu erzielen. Da die Kernelemente dieses Systems nur wenige Texturen, Funktionen und den Ray-Marching-Algorithmus sind, sind diese simpel in andere Renderengines zu implementieren, sowie weitere Features hinzuzufügen, die es aus zeitlichen Gründen nicht in diese Thesis geschafft haben. Die Ansätze der Crepuscular Rays sowie des TXAA-Filter sind vorhanden und müssen nur noch ausgebessert werden. Im Falle von MotionBlur wurde dieser Shader deaktiviert, da dieser noch Fehler enthielt. Insgesamt hat es sich als möglich erwiesen, in dem Rahmen dieser Arbeit der Aufgabenstellung nachzukommen. Durch die ausgiebige Beschäftigung haben sich Techniken gefestigt, aber auch die Arbeitsweise an sich hat einen Lernprozess ergeben.

10 Zukünftige Arbeiten

Um mit weiteren komplexeren Lichtberechnungen echtzeitfähig zu bleiben, bieten sich zusätzliche Optimierungen an. Sofern diese Wolken mit einem Terrain und anderen Objekten gerendert werden, bietet sich Depth-Culling an, um schon früh im Compute-Shader erkennen zu können, ob das jeweilige Pixel berechnet werden muss. Zusätzlich zum temporal Reprojektion kann auch der TXAA-Filter angewendet werden, um Artefakte, die in den Wolken auftreten können, zu verhindern und um die Wolken weicher wirken zu lassen.
Wie bereits vorgestellt gibt es viele Arten von Wolken, die nicht alle in dieser Thesis implementiert wurden. Daher sollte angestrebt werden auch diese zu verwirklichen. Eine Möglichkeit dies umzusetzen hat Rockstar auf der SIGGRAPH 2019 [FB19] vorgestellt, indem anhand der Weather-Map eine 1x128 RGBA8 Textur erstellt wird, die alle Höhen-Gradienten für die Altocumulus- und Stratuswolken enthält und als LUT dient. Dadurch erreichen sie eine hohe Varietät ihrer Wolken.
Um dem User noch mehr Interaktionsspielraum zu geben, könnte das Wetter dynamisch geändert werden. Es könnten für verschiedene Wetterarten wie Gewitter, Orkane oder ähnliche, Textur-Templates hinterlegt werden, die sich über Zeit mit der momentanen Weather-Map einblenden und diese dem gewünschten Effekt anpassen. Ganz ähnlich dem Prinzip von Schneider, nachdem die Klimazonen von Horizon Zero Dawn [Sch17] entwickelt wurden.
Zusätzlich könnte die Implementation von Blitzen in den Wolken in Betracht gezogen werden. Ob diese aber in einem echtzeitfähigen System möglich sind, muss getestet werden, da die Wegfindung des Blitzes und das zusätzliche Scattering in den Wolken mit in die Berechnungen einfließen müssen. Der vorgestellte Ansatz von Rockstar sei grob implementiert und verwendet ein Array mit Point-Lights und die Berechnungen dafür fließen wiederum in die Scattering-Berechnungen ein. Dadurch könnten aber Szenarien wie authentische Gewitter sowie bei Naturkatastrophen wie Vulkanausbrüche mit Eruptionsgewitter oder Hurrikane dargestellt werden. Ebenso könnte statt dem Rayleigh-Scattering auch das Hosek-Wilkie Verfahren verwendet werden, welches realistischere Ergebnisse hervorbringen soll. Insgesamt gibt es viele Wege, Wolken noch lebensnaher zu generieren und somit bietet das Feld ausreichend Platz für weitere Forschungen.

Abbildungsverzeichnis

Literatur

[Bia14] BIAGIOLI, Adrian: *Understanding Perlin Noise*. https://flafla2.github.io/2014/08/09/perlinnoise.html. Version: 2014. – [Online; Stand 04. Oktober 2019]

[BN08] BRUNETON, Eric ; NEYRET, Fabrice: Precomputed atmospheric scattering. In: *Computer graphics forum* Bd. 27 Wiley Online Library, 2008, S. 1079–1086

[BNM+08] BOUTHORS, Antoine ; NEYRET, Fabrice ; MAX, Nelson ; BRUNETON, Eric ; CRASSIN, Cyril: Interactive multiple anisotropic scattering in clouds. In: *Proceedings of the 2008 symposium on Interactive 3D graphics and games* ACM, 2008, S. 173–182

[Bur08] BURGER, Wilhelm: Gradientenbasierte Rauschfunktionen und Perlin Noise / School of Informatics, Communications and Media, Upper Austria University of Applied Sciences. Version: November 2008. http://staff.fh-hagenberg.at/burger/. Hagenberg, Austria, November 2008 (HGBTR08-02). – Forschungsbericht

[CI] CORNUT, O ; IMGUI, Dear: *Bloat-free Immediate Mode Graphical User Interface for C++ with minimal dependencies*

[CKK18] CHRISTENSEN, Per ; KENSLER, Andrew ; KILPATRICK, Charlie: Progressive Multi-Jittered Sample Sequences. In: *Computer Graphics Forum* Bd. 37 Wiley Online Library, 2018, S. 21–33

[DMPPA11] DODIN, Pierre ; MARTEL-PELLETIER, Johanne ; PELLETIER, Jean-Pierre ; ABRAM, François: A fully automated human knee 3D MRI bone segmentation using the ray casting technique. In: *Medical & Biological Engineering & Computing* 49 (2011), Dec, Nr. 12, 1413–1424. http://dx.doi.org/10.1007/s11517-011-0838-8. – DOI 10.1007/s11517–011–0838–8. – ISSN 1741–0444

[ES00] ELINAS, Pantelis ; STÜRZLINGER, Wolfgang: Real-time rendering of 3d clouds. In: *Journal of Graphics Tools* 5 (2000), Nr. 4, S. 33–45

[FB19] FABIAN BAUER, Rockstar: *Creating the atmospheric world of Red Dead Redemption 2*. http://advances.realtimerendering.com/s2019/slides_public_release.pptx. Version: 2019

[FWKH17] FONG, Julian ; WRENNINGE, Magnus ; KULLA, Christopher ; HABEL, Ralf: Production Volume Rendering: SIGGRAPH 2017 Course. In: *ACM SIGGRAPH 2017 Courses*. New York, NY, USA : ACM, 2017 (SIGGRAPH '17). – ISBN 978–1–4503–5014–3, 2:1–2:79

[Gar85] GARDNER, Geoffrey Y.: Visual Simulation of Clouds. In: *SIGGRAPH Comput. Graph.* 19 (1985), Juli, Nr. 3, 297–304. http://dx.doi.org/10.1145/325165.325248. – DOI 10.1145/325165.325248. – ISSN 0097–8930

[Hab10] HABLE, John: *Uncharted 2: HDR Lighting*. https://www.gdcvault.com/play/1012351/Uncharted-2-HDR. Version: 2010. – [Online; Stand 04. Oktober 2019]

[Hal62] HALTON, John H.: Sequential monte carlo. In: *Mathematical Proceedings of the Cambridge Philosophical Society* Bd. 58 Cambridge University Press, 1962, S. 57–78

[Hil16] HILLAIRE, S: Physically based sky, atmosphere and cloud rendering in frostbite. In: *Physically Based Shading in Theory and Practice, SIGGRAPH Courses* (2016)

[Hit11] HITT, David: *What Are Clouds?* https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-are-clouds-58.html. Version: 2011. – [Online; Stand 04. Oktober 2019]

[HL01] HARRIS, Mark J. ; LASTRA, Anselmo: Real-time cloud rendering. In: *Computer Graphics Forum* Bd. 20 Wiley Online Library, 2001, S. 76–85

[HP+02] HOFFMAN, Nathaniel ; PREETHAM, Arcot J. u. a.: Rendering outdoor light scattering in real time. In: *Game developers conference*, 2002

[HW13] HOŠEKHOŠEK, Lukáš ; WILKIE, Alexander: Adding a solar-radiance function to the hošek-wilkie skylight model. In: *IEEE computer graphics and applications* 33 (2013), Nr. 3, S. 44–52

[Ken67] KENSLER, Andrew: Correlated multi-jittered sampling. In: *Mathematical Physics and applied mathematics* 7 (1967), S. 86–112

[KMM+17] KALLWEIT, Simon ; MÜLLER, Thomas ; MCWILLIAMS, Brian ; GROSS, Markus ; NOVÁK, Jan: Deep Scattering: Rendering Atmospheric Clouds with Radiance-predicting Neural Networks. In: *ACM Trans. Graph.* 36 (2017), November, Nr. 6,

231:1–231:11. http://dx.doi.org/10.1145/3130800.3130880. – DOI 10.1145/3130800.3130880. – ISSN 0730–0301

[MG16] MIKKEL GJOEL, Mikkel S.: *Rendering of Inside, High Fidelity, Low Complexity*. https://www.gdcvault.com/play/1023002/. Version: 2016. – [Online; Stand 04. Oktober 2019]

[Nel18] NELSON, Daniel: *Types of clouds in the sky*. https://sciencetrends.com/many-different-types-clouds-sky/. Version: 2018. – [Online; Stand 04. Oktober 2019]

[NGH+18] NOVÁK, Jan ; GEORGIEV, Iliyan ; HANIKA, Johannes ; KŘIVÁNEK, Jaroslav ; JAROSZ, Wojciech: Monte Carlo Methods for Physically Based Volume Rendering. In: *ACM SIGGRAPH 2018 Courses*. New York, NY, USA : ACM, 2018 (SIGGRAPH '18). – ISBN 978–1–4503–5809–5, 14:1–14:1

[Ped16] PEDERSEN, Lasse Jon F.: *Temporal Reprojection Anti-Aliasing in INSIDE*. https://www.gdcvault.com/play/1022970/Temporal-Reprojection-Anti-Aliasing-in. Version: 2016. – [Online; Stand 04. Oktober 2019]

[Per85] PERLIN, Ken: An Image Synthesizer. In: *SIGGRAPH Comput. Graph.* 19 (1985), Juli, Nr. 3, 287–296. http://dx.doi.org/10.1145/325165.325247. – DOI 10.1145/325165.325247. – ISSN 0097–8930

[PSS99] PREETHAM, A. J. ; SHIRLEY, Peter ; SMITS, Brian: A Practical Analytic Model for Daylight. 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, 91–100

[Sch16] SCHNEIDER, Andrew: Real-time volumetric cloudscapes. In: *GPU Pro 7: Advanced Rendering Techniques* 97 (2016)

[Sch17] SCHNEIDER, A: *Nubis, Authoring Real-Time Volumetric Cloudscapes with the Decima Engine*. 2017

[SV15] SCHNEIDER, Andrew ; VOS, Nathan: The real-time volumetric cloudscapes of horizon: Zero dawn. In: *Advances in Real-Time Rendering in Games, ACM SIGGRAPH* (2015)

[Ton06] TONELLI, Nicholas A.: *Flickr - Nicholas T - Sheared.jpg*. ByNicholasA.TonellifromPennsylvania,

USA-Sheared,CCBY2.0,https://commons.wikimedia.org/w/index.php?curid=21219478. Version: 2006. – [Online; Stand 04. Oktober 2019]

[Wei07] WEISKOPF, Daniel: *GPU-based interactive visualization techniques*. Springer, 2007

[Wik05a] WIKIMEDIA: *Altocumulus1.jpg*. CCBY-SA3.0,https://commons.wikimedia.org/w/index.php?curid=121025. Version: 2005. – [Online; Stand 04. Oktober 2019]

[Wik05b] WIKIMEDIA: *As 1.jpg*. CCBY-SA3.0,https://commons.wikimedia.org/w/index.php?curid=243704. Version: 2005. – [Online; Stand 04. Oktober 2019]

[Wik05c] WIKIMEDIA: *Cirrus over Warsaw*. CCBY-SA2.0,https://commons.wikimedia.org/w/index.php?curid=225666. Version: 2005. – [Online; Stand 04. Oktober 2019]

[Wik05d] WIKIMEDIA, Glg: *Cumulus cloud above Lechtaler Alps at tannheim, Austria.jpg*. VonGlg-phototakenbyGlg,CCBY-SA2.0de,https://commons.wikimedia.org/w/index.php?curid=172807. Version: 2005. – [Online; Stand 04. Oktober 2019]

[Wik17] WIKIMEDIA: *Hardangervidda.JPG*. PublicDomain,https://commons.wikimedia.org/w/index.php?curid=434175. Version: 2017. – [Online; Stand 04. Oktober 2019]

[Wor96] WORLEY, Steven: A cellular texture basis function. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* ACM, 1996, S. 291–294

[Zuc17] ZUCCONI, Alan: *The Mathematics of Rayleigh Scattering*. https://www.alanzucconi.com/2017/10/10/atmospheric-scattering-3/. Version: 2017. – [Online; Stand 04. Oktober 2019]