



ulm university universität  
**uulm**

**Universität Ulm** | 89069 Ulm | Germany

**Fakultät für Ingenieurwissenschaften,  
Informatik und Psychologie**  
Institut für Medieninformatik

# God Rays und Volumetrische Lichtstreuung

Bachelorarbeit an der Universität Ulm

**Vorgelegt von:**

Marius Kircher

[marius.kircher@uni-ulm.de](mailto:marius.kircher@uni-ulm.de)

**Gutachter:**

Prof. Dr. Timo Ropinski

**Betreuer:**

Julian Kreiser, M.Sc.

**2016**



## **Zusammenfassung**

Bei der Beleuchtungsberechnung in der Computergrafik liegt der Fokus auf der plausiblen Darstellung von Oberflächen fester Objekte. Oft wird angenommen, dass zwischen den Objekten ein Vakuum liegt. Allerdings hängt das natürliche Erscheinungsbild einer realen Szene in den meisten Fällen auch von der Lichtinteraktion mit der Luft und darin enthaltenen Partikeln ab. Neben bloßem Realismus können räumliche Beleuchtungseffekte gewisse Stimmungen erzeugen, ästhetisch wirken oder die Aufmerksamkeit eines Betrachters lenken. Diese Arbeit setzt sich detailliert mit der Theorie von Lichtstreuung in gasförmigen Medien und der Implementierung von Renderingverfahren, die insbesondere den mit God Rays bezeichneten Effekt behandeln, auseinander. God Rays sind auffällige Lichtstrahlen, die zum Beispiel bei bestimmten dunstigen Wetterbedingungen auftreten können. Die Implementierung der Verfahren erfolgt in C++ und OpenGL. Den Abschluss bildet eine Evaluierung der umgesetzten Verfahren.



# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation . . . . .	2
1.2. Überblick . . . . .	2
<b>2. Hintergrund</b>	<b>5</b>
2.1. Physikalische Grundlagen . . . . .	5
2.1.1. Strahlung . . . . .	5
2.1.2. Licht und Lichtstreuung . . . . .	8
2.2. Computergrafik . . . . .	14
2.2.1. Physikalisch-basiertes Rendering . . . . .	16
2.2.2. Lichtstreuung in der Atmosphäre . . . . .	18
2.2.3. Shadow Mapping . . . . .	24
2.3. Literaturüberblick . . . . .	25
<b>3. Praktische Umsetzung</b>	<b>29</b>
3.1. Post-Processing . . . . .	29
3.2. Ray-Marching . . . . .	33
3.3. Implementierung . . . . .	34
3.3.1. Framework . . . . .	34
3.3.2. Post-Processing von Mitchell . . . . .	38
3.3.3. Post-Processing von Sousa . . . . .	42
3.3.4. Ray-Marching von Tóth und Umenhoffer . . . . .	46
3.4. Evaluierung . . . . .	49
3.4.1. Vergleich der implementierten Verfahren . . . . .	50
3.4.2. Limitierungen . . . . .	56
<b>4. Fazit</b>	<b>63</b>
4.1. Problemstellung und Lösung . . . . .	63
4.2. Ausblick . . . . .	64
<b>A. Anhang</b>	<b>65</b>



# 1. Einleitung

Gegenstände in der realen Welt können nicht direkt gesehen werden. In Wirklichkeit nehmen wir das Licht wahr, das von den Gegenständen in unser Auge gestreut wird. Lichtstreuung ist ein komplexes Thema, das vor allem in der Physik erforscht wird. Zunehmendes Interesse am Phänomen der Lichtstreuung hat seit einigen Jahrzehnten aber auch die Computergrafik als Teilgebiet der Informatik. Die vorliegende Arbeit betrachtet speziell die Lichtstreuung in gasförmigen Medien und einen damit verbundenen, oft als ästhetisch empfundenen visuellen Effekt: auffällige Lichtstrahlen, wie in Abbildung 1.1 zu sehen, die unter anderem God Rays genannt werden. Der praktische Teil dieser Arbeit stellt die Implementierung von Verfahren vor, die God Rays in einer beliebigen Szene erzeugen können.



Abbildung 1.1.: Lichtstreuung in gasförmigen Medien kann als ästhetisch wahrgenommene Phänomene in der Natur hervorrufen. Autor: Carlos Perez Couto, Lizenz: Creative Commons Attribution-Share Alike 3.0 Unported

## 1. EINLEITUNG

### 1.1. Motivation

Computergrafik hat heute breit gefächerte Anwendungsgebiete. Angefangen bei Visualisierungen für die Architektur oder den Maschinen- und Automobilbau, helfen zu Grafiken aufbereitete Daten auch in der Medizin, Biologie und Chemie. Nicht zuletzt werden in der Unterhaltungsbranche durch Computergrafik faszinierende Bilder und interaktive Erlebnisse kreiert.

Das übergreifende Ziel der Computergrafik ist, aus den Daten einer meist dreidimensionalen Szene eine zweidimensionale Projektion zu berechnen. Die Anforderungen an das dabei entstehende Bild variieren je nach Anwendungsgebiet. In der Architektur oder Filmindustrie sollen synthetische Bilder oft in Kombination oder als Ersatz für Abbildungen der realen Welt eingesetzt werden. In diesem Fall muss die Berechnung meist durch aufwändige Verfahren mit langer Rechenzeit durchgeführt werden. Im Maschinenbau oder der Medizin dienen die Bilder oft illustrativen Zwecken, z. B. der Veranschaulichung des Aufbaus eines Motors oder der Untersuchung des CT-Scans eines Patienten. In diesem Fall ist es sinnvoll, nicht fotorealistische, sondern interaktive Bilder zu berechnen, was die Erkundung einer Szene bzw. eines Modells durch freies Verändern der Perspektive erlaubt. In der Entwicklung von Computerspielen und interaktiver digitaler Kunst versucht man oft, einen ansprechenden Kompromiss zwischen einem natürlichen Erscheinungsbild und Interaktivität zu erreichen. Eine stetige Verbesserung des Kompromisses wird durch moderne Grafikhardware und Renderingverfahren ermöglicht.

Eine Voraussetzung für Interaktivität ist die Berechnung von Bildern in Echtzeit. Das natürliche Erscheinungsbild einer Szene hängt davon ab, wie genau das sichtbare Resultat des vielfach gestreuten Lichts berechnet wird. Eine nicht zu vernachlässigende Komponente ist der in gasförmigen Medien gestreute Anteil des Lichts, der insbesondere beim Auftreten von God Rays sichtbar wird. Die vorliegende Arbeit stellt eine Auswahl bestehender Echtzeitverfahren für God Rays vor.

### 1.2. Überblick

Lichtstreuung kann auf verschiedenen Ebenen und aus verschiedenen Perspektiven untersucht werden. Um ein breites Verständnis zu erhalten, betrachtet Abschnitt 2.1 Lichtstreuung ausgehend von einer rein physikalischen Perspektive. Es werden die grundlegenden Begriffe erläutert, die in der Physik zur Beschreibung des Lichts verwendet werden. Es werden außerdem die Welleneigenschaften des Lichts betrachtet. Schließlich wird auf dieser Basis eine Erklärung für das Entstehen von Farben in der Erdatmosphäre gegeben. Am Beispiel der Erdatmosphäre wird die Rolle unterschiedlicher Partikel in gasförmigen Medien erläutert.

Abschnitt 2.2 vergleicht die physikalische Perspektive mit der Sichtweise der Computergrafik und weist auf die grundsätzlich verschiedenen Interessen der jeweiligen Forschungsfelder hin. Im weiteren Verlauf wird ein Modell von Hoffman und Preetham [5] für das Rendering von Lichtstreuung in der Atmosphäre vorgestellt, das wie zunehmend viele andere Publikationen in der Computergrafik auf das Wissen der Physik zurückgreift. Dabei müssen Vorgänge auf einer mikroskopischen Ebene in andere Konzepte übersetzt werden, die im Hinblick auf Rechenzeit und Szenenmodellierung umsetzbar sind. Das Modell behandelt Effekte von Medien, aber nicht explizit das Rendering der als God Rays bezeichneten Lichtstrahlen. Abschnitt 2.3 gibt einen Überblick über Publikationen, die sich speziell auf die Erzeugung von God Rays beziehen. Darunter finden sich auch Verfahren ohne physikalische Basis, welche tendenziell eine einfachere Implementierung und kürzere Rechenzeit aufweisen.

Kapitel 3 stellt den praktisch orientierten Hauptteil der Arbeit dar. Das Kapitel enthält Erläuterungen zur Implementierung dreier ausgewählter Verfahren für das Rendering von God Rays sowie deren Evaluierung. In den Abschnitten 3.1 und 3.2 werden zwei bildbasierte und ein Ray-Marching-basiertes Verfahren analysiert. Der Fokus liegt dabei auf der Bedeutung der Parameter, die sich in den zwei Verfahrenstypen wesentlich unterscheidet. Eine Beschreibung der Implementierung in C++ und OpenGL erfolgt in Abschnitt 3.3 anhand von textuellen Erklärungen sowie Verweisen auf Programmausdrucke im Anhang. Anschließend werden die drei Verfahren in Abschnitt 3.4 im Hinblick auf Plausibilität und Rechenzeit evaluiert. Kapitel 4 fasst die Ergebnisse der Arbeit zusammen.



## 2. Hintergrund

### 2.1. Physikalische Grundlagen

Für den Zweck dieser Arbeit, die Implementierung und Bewertung von Verfahren für volumetrische Lichtstreuung, ist es hilfreich zu verstehen, was Licht ist und wie es entsteht. Dazu sollen im Folgenden die relevanten physikalischen Grundlagen zusammengefasst werden. Die Gleichungen und Erläuterungen zu den Gleichungen dieses Abschnitts sind an *Feynman Lectures of Physics* [4], insbesondere an Kapitel 32, angelehnt.

Licht lässt sich mit der sogenannten geometrischen Optik beschreiben. Diese betrachtet das Licht als Strahlen, die von einer Quelle ausgehen. Diese können reflektiert, absorbiert oder gebrochen werden. Irgendwann treffen sie das Auge eines Beobachters und erzeugen ein Bild. Durch Experimente kann man feststellen, dass bei der Reflexion der Ausfallswinkel gleich dem Einfallswinkel ist und dass der Brechungswinkel vom Material abhängt, in das der Lichtstrahl eintritt. Beobachtungen wie diese reichen bereits aus, um mittels Strahlensätzen einfache Bilder einer dreidimensionalen Szene zu erzeugen. Um zu verstehen, wie Licht an Partikeln im Raum gestreut wird, muss man auf neueres Wissen zurückgreifen. Nach der Theorie des Elektromagnetismus, die von Maxwell begründet wurde, ist es die Energie, die von elektrischer Ladung verursacht wird, wenn diese bewegt wird. Diese Energie wird als elektromagnetische Strahlung bezeichnet.

Im Folgenden wird das Thema Strahlung näher betrachtet. Zunächst werden in Abschnitt 2.1.1 die Größen und deren Zusammenhänge aufgezeigt, die man zur Beschreibung von elektromagnetischer Strahlung benötigt. Danach wird in Abschnitt 2.1.2 das Licht als Spezialfall von Strahlung betrachtet. Da das Licht die Eigenschaften von Wellen hat, ergibt sich das Phänomen der sogenannten Interferenz. Mithilfe der Interferenz lässt sich erst der entscheidende Unterschied von Lichtstreuung in verschiedenen dichten Medien verstehen.

#### 2.1.1. Strahlung

Viele grundlegende Phänomene in der Natur lassen sich mit dem physikalischen Begriff der Kraft erklären. Kraft ist etwas, das eine Geschwindigkeitsänderung (Beschleunigung) bewirken kann. Anschauliche Kraftwirkungen finden sich bei mechanischen Vorgän-

## 2. HINTERGRUND

gen. Tatsächlich hat man erkannt, dass auch Licht eine Kraftwirkung im subatomaren Maßstab ist. Dabei handelt es sich um elektrische Kraftwirkung. Elektrische Kraft wird durch Ladung verursacht. Die kleinste Ladungseinheit in der Natur ist ein Elektron. Eine Ladung kann auf sehr weite Entfernung Kraft auf eine andere Ladung ausüben, ohne dass dabei ein Medium als Träger im Raum zwischen den Ladungen vorhanden sein muss. Die Kraft pro Ladung, die an einem Punkt im Raum wirkt, heißt elektrisches Feld. Man kann sich also vorstellen, dass eine nicht-materielle Größe (Kraftfeld) den Raum um eine Ladung ausfüllt, die mit zunehmender Entfernung kleiner wird.

Es gibt zwei Zustände der Ladung, bei denen eine jeweils andere Kraft wirkt. Der erste Zustand ist die Ladung in Ruhe; dabei wird eine Kraft verursacht, die mit dem Quadrat der Entfernung abfällt, die Coulombkraft, die hier nicht interessant ist, da sie nur auf sehr kleinem Raum zum Tragen kommt. Der zweite Zustand ist die bewegte Ladung; dabei wird eine Kraft verursacht, die nur linear mit der Entfernung abfällt. Das ist elektromagnetische Strahlung, die im Folgenden betrachtet wird. Strahlung wird von einer beschleunigten Ladung emittiert und breitet sich mit Lichtgeschwindigkeit im Raum aus.

Abbildung 2.1 betrachtet das Feld  $E$ , also die Kraft die durch eine Ladung  $q$  mit der Beschleunigung  $a$  in einer bestimmten Entfernung  $r$  zur Zeit  $t$  in eine bestimmte Richtung  $\perp \theta$  auf eine andere Ladung wirkt, die sich in Richtung  $\theta$  befindet. Formal kann das Feld wie folgt formuliert werden:

$$\underbrace{E(t)}_{\substack{\text{Feld zum} \\ \text{Zeitpunkt } t}} = \frac{-q}{4\pi\epsilon_0 c^2 r} \underbrace{a \left( t - \frac{r}{c} \right)}_{\substack{\text{Beschleunigung} \\ \text{zum Zeitpunkt} \\ \text{der Emission}}} \sin\theta, \quad (2.1)$$

wobei  $\epsilon_0$  die elektrische Feldkonstante und  $c$  die Lichtgeschwindigkeit ist. Die Vektoren  $a$  und  $E$  und die Strecke  $r$  liegen in einer Ebene. Man sieht in der Gleichung, dass das Feld  $E$  umgekehrt proportional zur Entfernung  $r$  ist.

Wie bereits geschildert kann Kraft eine Beschleunigung bewirken. Daneben gibt es in der Physik den Begriff der Energie. Energie ist etwas, das Arbeit verrichten kann. Arbeit ist eine Kraftwirkung entlang eines Weges. Energie kann also eine bestimmte Wegstrecke lang eine bestimmte Kraft ausüben. Energie ist abstrakt, sodass man Systeme betrachtet, die Energie besitzen. Eine beschleunigte Ladung ist ein solches System. Die Ladung kann Energie aufnehmen und abgeben (wobei sie Arbeit verrichtet bzw. Arbeit an ihr verrichtet wird). Man kann im Falle der Beschleunigung die sogenannte kinetische Energie  $W_{kin}$  zu einem Zeitpunkt berechnen. Diese hängt von der Masse  $m$  und dem Quadrat der Geschwindigkeit  $v$  eines Systems zu diesem Zeitpunkt ab, sodass  $W_{kin} = \frac{1}{2}mv^2$ .

Man hat gemessen, dass sich eine Ladung in einem Feld mit einer Geschwindigkeit proportional zur Feldstärke bewegt. Daraus kann man mit obiger Formel für  $W_{kin}$

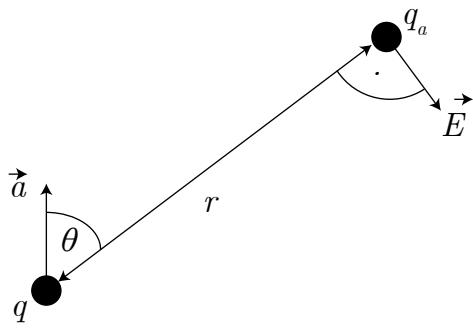


Abbildung 2.1.: Die Beschleunigung  $\vec{a}$  der Ladung  $q$  bewirkt eine Kraft auf Ladung  $q_a$ , die sich in einer Entfernung  $r$  in Richtung  $\theta$  befindet. Man sagt, Ladung  $q$  erzeugt ein elektrisches Feld. Das elektrische Feld wird mit  $\vec{E}$  bezeichnet. Alle Vektoren liegen in einer Ebene.

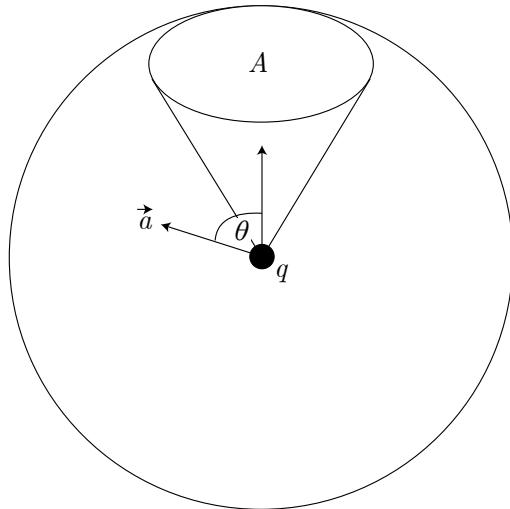


Abbildung 2.2.: Die Strahlung, die von Ladung  $q$  mit Beschleunigung  $\vec{a}$  in Richtung  $\theta$  emittiert wird, breitet sich kegelförmig aus. Der Energiefluss durch die Fläche  $A$  ist proportional zum Quadrat des elektrischen Feldes. Die Energie in dem Kegel bleibt erhalten. Entlang einer Geraden in Richtung  $\theta$  nimmt die Energie quadratisch ab (sog.  $1/r^2$ -Gesetz).  $\theta$  und  $\vec{a}$  liegen in einer Ebene.

ableiten, dass die kinetische Energie in der Ladung proportional zum Quadrat des Feldes sein muss. Kennt man das elektrische Feld, das von einer beschleunigten Ladung verursacht wird, kann man nun die Energie berechnen, die von der Ladung abgestrahlt wird. Betrachtet man die über eine Sekunde gemittelte Feldstärke  $E$ , dann ist  $W(\theta) = \epsilon_0 c E^2$  die Energie, die in dieser Sekunde durch eine Einheitsfläche in Richtung

## 2. HINTERGRUND

$\theta$  fließt (vgl. Abb. 2.2). Setzt man für  $E$  den Ausdruck (2.1) ein, erhält man

$$W(\theta) = q^2 a \left( t - \frac{r}{c} \right)^2 \sin^2 \theta \frac{1}{16\pi^2 \epsilon_0 r^2 c^3}. \quad (2.2)$$

Man sieht, dass die Energie, die in Richtung  $\theta$  abgestrahlt wird, mit dem Quadrat der Entfernung kleiner wird.

Integriert man (2.2) über die gesamte kugelförmige Umgebung der Ladung, erhält man schließlich

$$W = \frac{q^2}{6\pi\epsilon_0 c^3} a \left( t - \frac{r}{c} \right)^2 \quad (2.3)$$

für die Gesamtenergie, die pro Sekunde von der Ladung abgestrahlt wird.

### 2.1.2. Licht und Lichtstreuung

Licht entsteht durch eine besondere Beschleunigung von Ladung, nämlich durch die Oszillation von Elektronen. Abbildung 2.3 illustriert, wie sich das elektrische Feld wellenförmig ausbreitet. Wellen haben in der Physik eine besondere Bedeutung, da deren Eigenschaften das Verhalten des Lichts sehr gut erklären, aber auch sonst nahezu überall in der Natur auftreten, da alles aus Teilchen besteht, die sich ständig in Schwingung befinden.

Für Oszillationen gilt das Zeit-Beschleunigungs-Gesetz  $a(t) = -\omega^2 x_0 \sin(\omega t)$ , wobei  $\omega$  die Winkelgeschwindigkeit<sup>1</sup> und  $x_0$  die maximale Auslenkung<sup>2</sup> ist. Die Winkelgeschwindigkeit lässt sich auch schreiben als  $\omega = 2\pi f$ , wobei  $f$  die Frequenz des Lichts ist. Das menschliche Auge ist, genau wie eine Kamera, ein Sensor, der die Frequenzen des Lichts wahrnimmt und in ein Abbild der Welt umwandelt. Eine weitere Eigenschaft des Lichts ist die Wellenlänge  $\lambda = \frac{c}{f}$ , wobei  $c$  die Lichtgeschwindigkeit ist. Die Wellenlängen des sichtbaren Lichts liegen ungefähr zwischen 380 und 750 Nanometer und die Frequenz ungefähr zwischen 400 und 789 Kilohertz. Wellen mit kürzeren Wellen bzw. höherer Frequenz sind energiereicher als Wellen mit längeren Wellen bzw. niedriger Frequenz. Das sichtbare Licht mit der niedrigsten Energie erscheint uns rot; das andere Ende des Spektrums ist blau bzw. violett. Weißes Licht enthält alle Wellenlängen gleichermaßen.

Gleichung (2.3) hat die Energie des Lichts in Beziehung zur Beschleunigung der Ladung gesetzt. Der Zeitpunkt der Beschleunigung ist jedoch nicht der Zeitpunkt, zu dem die gesamte Energie auf einen Schlag frei wird. Einen solchen klar definierten Zeitpunkt gibt es im Allgemeinen nicht. Im Falle einer Oszillation, bei der die Beschleunigung am Anfang und Ende eines Zyklus Null ist, lässt sich aber sagen, dass die Energie pro

---

<sup>1</sup>Die Winkelgeschwindigkeit heißt auch Winkelfrequenz oder einfach nur Frequenz, da sie zu dieser direkt proportional ist.

<sup>2</sup>Die maximale Auslenkung heißt auch Amplitude.

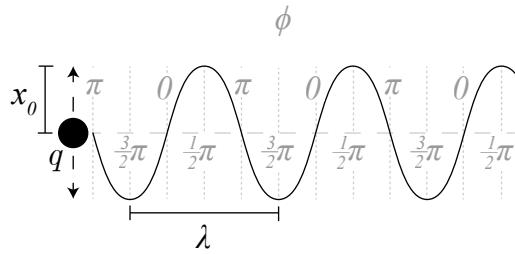


Abbildung 2.3.: Ein oszillierendes Elektron emittiert Licht. Die Feldstärke an einem Ort ändert sich periodisch sinusförmig. Daher wird Licht als Welle betrachtet. Grau eingezeichnet ist die sogenannte Phase  $\phi \in [0, 2\pi)$ .

Zyklus aus der mittleren Beschleunigung über einen Zyklus entsteht. Daher benutzt man den Mittelwert der quadrierten Beschleunigung aus dem Zeit-Beschleunigungs-Gesetz  $\langle a^2 \rangle = \frac{1}{2} \omega^4 x_0^2$ , wobei der Mittelwert der quadrierten Sinusfunktion  $\frac{1}{2}$  ist. Damit wird Gleichung (2.3) zu

$$W = \frac{q^2}{12\pi\epsilon_0 c^3} \omega^4 x_0^2. \quad (2.4)$$

Man hat die Energie nun statt in Abhängigkeit von der Beschleunigung, in Abhängigkeit von der Frequenz  $\omega$  und der Amplitude  $x_0$ .

Streuung nennt man den Effekt, wenn das elektrische Feld einfallender Lichtstrahlen Elektronen wieder zum Oszillieren bringt, wodurch diese wegen ihrer Beschleunigung erneut Licht ausstrahlen. Im Folgenden wird das Elektron gleichgesetzt mit dem Atom, in dem es sich befindet, da dieser Unterschied vorerst nicht relevant ist. Das Phänomen ist abhängig davon, ob man feste oder gasförmige Materie betrachtet.

In festen Materialien sind die Atome in einem regelmäßigen Muster angeordnet. Die hier stattfindenden Lichtinteraktionen sind Reflexion, Brechung und Absorption, wobei Absorption ein relativ einfacher Sonderfall ist. Absorbierte Lichtenergie wird als verloren angesehen und dessen Weg daher nicht weiter verfolgt. Details zur Berechnung der Absorption werden später besprochen. Was Reflexion und Brechung betrifft, sind dies Spezialfälle von kohärenter Streuung, was im Folgenden erläutert wird.

Streuung ist kohärent, wenn die vom Material gestreuten Wellen dieselbe Frequenz haben und bei ihrer Ausbreitung eine feste Phasendifferenz behalten. Je fester die Bindungen der Atome im Material desto mehr kohärente Streuung erhält man, da die meisten Atome in Phase schwingen. Viele Wellen werden in dieselbe Richtung gestreut, treffen irgendwann in das Auge eines Beobachters und überlagern sich dort. Wellen mit einer festen Phasendifferenz können sich bei Überlagerung gegenseitig verstärken

## 2. HINTERGRUND

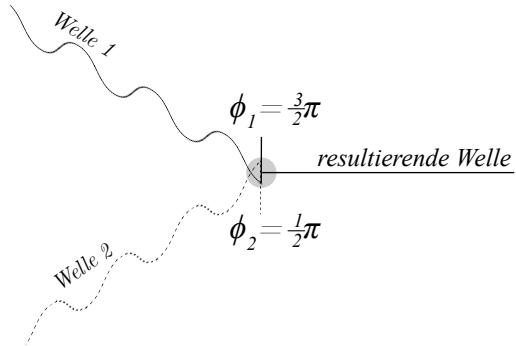


Abbildung 2.4.: Von links treffen zwei frequenzgleiche Wellen ein und interferieren. Dabei trifft hier beispielhaft ein Wellenberg auf ein Wellental (Phasendifferenz  $\phi_1 - \phi_2 = \pi$ ). Die Wellen löschen sich gegenseitig aus.

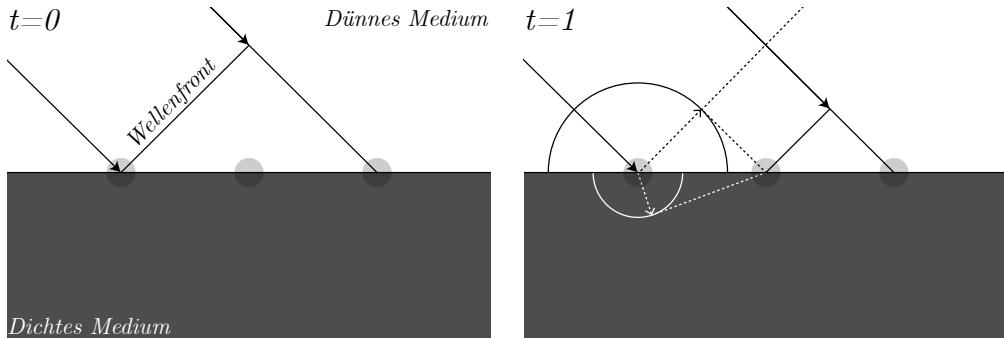
oder auslöschen. Dies wird Interferenz genannt. Abbildung 2.4 zeigt die Überlagerung zweier Wellen.

Noch interessanter ist die Betrachtung vieler interferierender Wellen. In Abbildung 2.5 wird eine Wellenfront gezeigt, die auf die Grenzfläche zweier Medien trifft. Das obere Medium sei dünn und das untere sei dicht. Im dichten Medium sitzen die Atome dichter und Lichtwellen breiten sich mit einer geringeren Geschwindigkeit aus. An der Grenzfläche sind beispielhaft drei Atome eingezeichnet, man muss sich aber das gesamte Medium dicht vorstellen. Beim Auftreffen der Wellenfront beginnen die Atome das Licht zu streuen. Jedes Atom wird zum Zentrum einer Kugelwelle. Durch die Interferenz der Kugelwellen erhält man nur in bestimmten Winkeln Licht. Dies sind gerade die Winkel für die Reflexion und Brechung. Die Darstellung geht auf das Huygens'sche Prinzip zurück.

Jetzt soll der Fall der Lichtstreuung in gasförmigen Medien betrachtet werden. Hier ist die Streuung wegen der schwachen Bindungen der Atome in Gasen größtenteils nicht kohärent. Die Atome sind regellos verteilt und in ständiger Bewegung. Sie verhalten sich wie unabhängige Quellen. Das hat entscheidende Auswirkungen darauf, wie ein Beobachter das gestreute Licht sieht, denn inkohärente Wellen interferieren nicht wie kohärente Wellen. Das lässt sich mit folgender Gleichung erklären. Wenn zwei Wellen mit den Amplituden  $A_1$  und  $A_2$  interferieren, dann gilt für die resultierende Amplitude

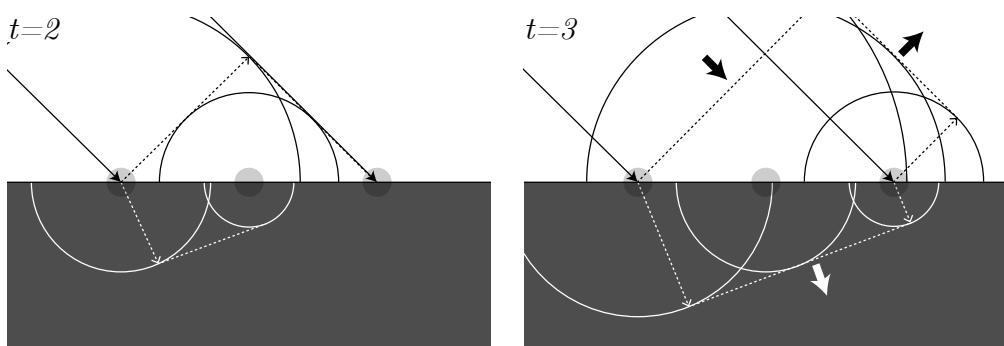
$$A_R^2 = A_1^2 + A_2^2 + 2A_1A_2 \cos(\phi_1 - \phi_2), \quad (2.5)$$

was man mit Trigonometrie zeigen kann. Bei inkohärenten Wellen ist die Phasendifferenz  $\phi_1 - \phi_2$  nicht fest. Folglich schwankt der Kosinusterm mehr oder weniger schnell zwischen den Werten  $-1$  und  $1$ . Mittelt man dies über einen ausreichend großen Zeitraum wird er zu Null und es bleibt nur noch die Summe der Amplituden  $A_1^2 + A_2^2$  übrig. Tatsächlich ist unser Auge bei weitem nicht in der Lage, die Interferenz bei gewöhnli-



(a) Zeitpunkt  $t = 0$ : Von links oben fällt ein Bündel Lichtwellen ein. Die Querlinie kennzeichnet die Wellenfront, an der die Wellen dieselbe Phase haben.

(b) Zeitpunkt  $t = 1$ : Die Atome des dichten Mediums (graue Fläche) streuen die Wellen kugelförmig in alle Richtungen. Die Kreise zeigen den Querschnitt der Kugeln, wo die gestreuten Wellen dieselbe Phase haben.



(c) Zeitpunkt  $t = 2$ : Die Atome beginnen zeitlich versetzt mit der Streuung. Die Wellen breiten sich im dichten Medium langsamer aus als im dünnen Medium.

(d) Zeitpunkt  $t = 3$ : Man erhält nur in den Richtungen eine neue Wellenfront, in denen sich die Wellen nicht durch ihre Phasendifferenz aufheben. Oberhalb des Mediums ist das genau die Richtung des reflektierten Lichts und innerhalb des Mediums die Richtung des gebrochenen Lichts (Beachte die gestrichelten Pfeile und die jeweilige Querlinie, die die neue Wellenfront und gleichzeitig die Einhüllende der Kreise darstellt).

Abbildung 2.5.: Huygen'sches Prinzip

## 2. HINTERGRUND

chen Lichtquellen wahrzunehmen, weil schon das Loch in der Pupille so groß ist, dass die Interferenzeffekte über einen im Vergleich zur Wellenlänge sehr großen Bereich gemittelt werden.

In Gleichung (2.4) wurde festgehalten, dass die gesamte ausgestrahlte Energie eines Elektrons proportional zum Quadrat der Amplitude ist. Damit folgt, dass die Energie des gestreuten Lichts in gasförmigen Medien die Summe der von jedem Atom gestreuten Energie ist. Man braucht also nur ein Atom zu betrachten und dessen gestreute Energie mit der Anzahl aller Atome zu multiplizieren. Vereinfachend wird weiterhin angenommen, dass es in einem Atom genau ein streuendes Elektron gibt, das eine ungedämpfte Schwingung ausführt.

Ein schwingungsfähiges System hat eine Bindung an die Ruhelage, um die es schwingt, oder anders gesagt, es gibt eine Kraft, die es stets in Richtung der Ruhelage beschleunigt. Von dieser Kraft hängt ab, mit welcher Frequenz das System schwingt, nachdem es einmalig aus der Ruhelage ausgelenkt wurde. Diese Frequenz heißt Eigenfrequenz  $\omega_0$ . Ein Elektron in einem Atom hat eine solche Eigenfrequenz, die einbezogen werden muss, wenn man die vom elektrischen Feld einfallender Lichtwellen angeregte Schwingung berechnet. Man kann die Amplitude  $x_0$  der angeregten Schwingung durch folgende Gleichung in Abhängigkeit von der Eigenfrequenz  $\omega_0$ , der anregenden Frequenz  $\omega$  und dem maximalen elektrischen Feld  $E_0$  berechnen:

$$x_0 = \frac{q_e E_0}{m_e (\omega^2 - \omega_0^2)}, \quad (2.6)$$

wobei  $q_e = \text{const.}$  die Elektronenladung und  $m_e = \text{const.}$  die Elektronenmasse ist.

Mit Gleichung (2.4) hatte man bereits die Lichtenergie eines schwingenden Elektrons abhängig von dessen Frequenz und Amplitude. Dies kann man jetzt benutzen, um die gestreute Energie zu berechnen. Durch Einsetzen von Gleichung (2.6) in (2.4) ergibt sich für die gesamte gestreute Energie pro Sekunde

$$W_s = \frac{\epsilon_0 c E_0^2}{2} \underbrace{\frac{8\pi r_e^2}{3} \frac{\omega^4}{(\omega^2 - \omega_0^2)^2}}_{\text{Streuquerschnitt}}, \quad (2.7)$$

wobei  $r_e$  der klassische Elektronenradius ist, der durch  $r_e = (q_e^2 / 4\pi\epsilon_0 m_e c^2) \approx 2.818 \cdot 10^{-15} \text{ m}$  definiert ist (siehe [13]). Der Term  $(8\pi r_e^2 / 3) \cdot (\omega^4 / (\omega^2 - \omega_0^2)^2)$  hat die Einheit  $\text{m}^2$  und ist daher eine Fläche. Diese wird als Streuquerschnitt  $\sigma_s$  bezeichnet und dient als Hilfsvariable dazu, die gestreute Energie in Abhängigkeit von der eintreffenden Energie zu betrachten, wie in Abbildung 2.6 dargestellt.

Atome in der Luft sind meistens Stickstoff- oder Sauerstoffatome, seltener auch Wassermoleküle, Kohlendioxid, Argon oder andere Gase, die aber so selten vorkommen, dass sie hier nicht beachtet werden. Die Eigenfrequenzen  $\omega_0$  der Elektronen in Stickstoff- und Sauerstoffatomen sind wesentlich höher als die Frequenzen des sichtbaren Lichts.

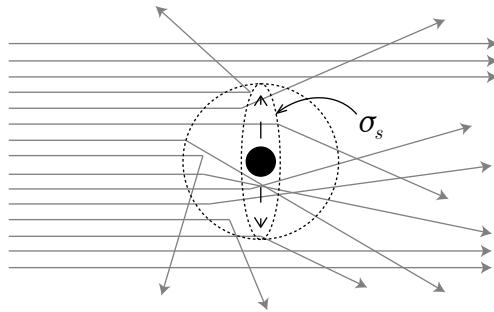


Abbildung 2.6.: Die einfallende Strahlung, die den sogenannten Streuquerschnitt  $\sigma_s$  trifft, wird in alle Richtungen gestreut. Die gestreute Energie ist proportional zu  $\sigma_s$ . Der Streuquerschnitt muss nicht dem Querschnitt des streuenden Teilchens entsprechen. Er dient nur als Hilfestellung zur Berechnung der gestreuten Energie.

Damit fällt im Streuquerschnitt in erster Näherung der Nenner von  $(\omega^4 / (\omega^2 - \omega_0^2)^2)$  weg, sodass die gestreute Energie proportional zu  $\omega^4$ , der vierten Potenz der Lichtfrequenz, ist. Das blaue Ende des Lichtspektrums hat ungefähr die doppelte Frequenz des roten Endes. Daraus folgt, dass blaues Licht  $2^4 = 16$  mal so intensiv gestreut wird wie rotes Licht. Der blaue Anteil des Lichts, das von der Sonne kommt, erreicht einen Beobachter auf der Erde, der nicht direkt in die Sonne schaut, folglich auch aus vielen anderen Richtungen, während der rote Anteil einen geraderen Weg nimmt. Das ist auch der Grund, warum der Himmel blau erscheint. Geht der Blick in Richtung Sonne, nimmt das Licht wieder eine weiße Farbe an, da der rote Anteil des Lichts zunimmt. Geht der Blick in Richtung des Horizonts, überwiegt der ausgestreute blaue Anteil, da das Licht, das vom Horizont kommt einen längeren Weg durch die streuende Atmosphäre hat, als das Licht das gerade von oben kommt. Steht die Sonne knapp über dem Horizont, sieht der Beobachter größtenteils rotes Licht, das wegen seiner geringeren Frequenz gegen Streuung unempfindlicher ist. Das ist auch der Grund, warum der Himmel am Horizont bei Sonnenauf- und untergang rot erscheint.

Die Erläuterungen des letzten Abschnittes gelten für Teilchen, deren Durchmesser in der Größenordnung von  $10^{-10}$  Metern liegt, was für einzelne Atome zutrifft. Im Verhältnis zu den Wellenlängen des sichtbaren Lichts, die zwischen  $10^{-7}$  und  $10^{-6}$  Meter liegen, sind diese Teilchen klein. Streuung an diesen relativ zur Wellenlänge kleinen Teilchen wird Rayleigh-Streuung genannt. Die Streuung an Teilchen mit Durchmesser ab einer Lichtwellenlänge heißt Mie-Streuung. Im Freien kommen diese Teilchen als kondensierte Wassertropfen in Dunst, Nebel und Wolken vor; in Innenräumen möglicherweise in der Form von Staub. Die Atome sitzen innerhalb dieser Teilchen enger als in reinen Gasen, sodass sie in Phase schwingen können. Ihr Verhalten ähnelt Festkörpern (siehe

## 2. HINTERGRUND

Abbildung 2.5). Das Teilchen streut kohärente Wellen, die interferieren können. In den Winkeln, in denen sich die Wellen nicht durch ihre Phasendifferenz aufheben, addieren sich die Amplituden auf. Da die Energie proportional zum Quadrat der Amplitude ist (siehe Gleichung 2.4), erhält man ein quadriertes Vielfaches der gestreuten Energie eines einzelnen Atoms. Die Lichtenergie wächst also quadratisch mit der Größe des streuenden Teilchens. Nun gilt es aber den folgenden Grenzfall zu betrachten. Wenn der Durchmesser des Teilchens die Wellenlänge des gestreuten Lichts überschreitet, löschen sich die Wellen, die an der Vorderseite und an der Hinterseite (vom Beobachter aus gesehen) des Teilchens gestreut werden, durch Interferenz gegenseitig aus. Das blaue Licht ist davon zuerst betroffen, da es kürzere Wellen hat. Folglich verschwindet bei größer werdenden Teilchen der blaue Anteil im Licht. Oft sind viele Teilchengrößen gleichzeitig vorhanden. Das ist auch der Grund, warum man trotz der blauen Rayleigh-Streuung weiße Wolken am Himmel sieht.

Zum Abschluss dieses Kapitels ist klarer geworden, was volumetrische Lichtstreuung bedeutet. Es wurde gezeigt, dass die Wahrnehmung des Lichts davon abhängt, welche Medien es durchquert hat. Es wurden zwei Arten von Partikeln bezüglich ihrer Interaktion mit dem Licht unterschieden, die sogenannten Rayleigh- und Mie-Partikel. Durch Streuung an Mie-Partikeln kann der sogenannte Tyndall-Effekt entstehen. Ein Spezialfall des Tyndall-Effekts sind God Rays. Während die Betrachtungen in diesem Kapitel eher qualitativ waren, folgt im nächsten Abschnitt eine Erklärung der Parameter, die man benutzt, um das in Medien gestreute Licht zu berechnen.

## 2.2. Computergrafik

Bei der Betrachtung des physikalischen Hintergrundes von Lichtstreuung fiel auf, dass die Ursache der Farben, die wir sehen, die Welleneigenschaften des Lichts sind. Die Welleneigenschaften erklären sich aus dem subatomaren Aufbau der Materie, wie im vorangehenden Abschnitt dargestellt wurde. Die große Anzahl subatomarer Teilchen in der realen Welt stellt die Grenze von Realismus in der Computergrafik dar. Vor diesem Hintergrund lässt sich auch der Unterschied zwischen der Computergrafik und wissenschaftlichem Rechnen in der Physik verstehen. Während in der Physik Grundlagenforschung betrieben wird, bedient sich die Computergrafik annähernden, auf einer makroskopischen Sicht der Welt basierenden Methoden, um die Ziele zu erreichen, die von ihren Anwendungsbereichen vorgegeben werden. Der hier geltende Maßstab ist stets die vergleichsweise grobe menschliche Wahrnehmung, wie sie ohne besondere Hilfsmittel gegeben ist. Zunächst können zwei Anwendungsbereiche unterschieden werden.

**Echtzeit-Rendering** Die Rechenzeit für ein Bild ist begrenzt. Ein für interaktive Anwendungen angemessenes Zeitbudget sind  $\frac{1}{60}s \approx 16,667ms$ . Die beim Echtzeit-Rendering verwendeten Techniken basieren heute im Allgemeinen auf einer Rendering-Pipeline bestehend aus den zwei Teilen Koordinatensystemtransformation und Rasterisierung. Die Eingabe der Rendering-Pipeline sind üblicherweise 3D-Koordinaten, welche die Transformation in den Sichtkegel einer Kamera durchlaufen. Zwischen den Koordinaten wird interpoliert, um Flächen zu erhalten, zum Beispiel Flächen von Dreiecken. Die Ausgabe sind diese Flächen, abgetastet an den Pixeln eines Rasterbildschirmes. Der Aufbau der Rendering-Pipeline sorgt für gute Parallelisierbarkeit. Die Rendering-Pipeline ist auf allen modernen GPUs implementiert, die durch Parallelprozessoren die Bilderzeugung entscheidend beschleunigen können. Der später in dieser Arbeit folgende praktische Teil nutzt die GPU über die standardisierte Schnittstelle OpenGL.

**Offline-Rendering** Darunter fallen alle Arten von Bilderzeugung, die keiner Echtzeitbedingung unterliegen. Die erzeugten Bilder haben im Allgemeinen höhere Detailtreue als bei Echtzeitanwendungen und können daher realistischer wirken. Die Rechenzeit ist entsprechend hoch und kann für ein einzelnes Bild mehrere Stunden bis Tage dauern. Der scheinbare Realismus wird durch Anwendung der geometrischen Optik erreicht. In der geometrischen Optik macht man die aus makroskopischer Sicht richtige Annahme, dass sich Licht auf geraden Strahlen fortbewegt. In der Computergrafik macht man außerdem die Annahme, dass Licht additiv ist, d. h., dass zwei Strahlen mit gleicher Energie bei Überlagerung die doppelte Energie haben. Die Annahme ist korrekt, da Licht im Allgemeinen nicht kohärent ist. Auf dieser Basis kann man mit Raytracing oder Pathtracing den Weg des Lichts durch die Szene finden und entlang der Strahlen über die Änderung der Energie integrieren, um den Wert für einen Pixel zu erhalten. Diese Techniken sind im Allgemeinen nicht mit der oben genannten Hardware zur Grafikbeschleunigung kompatibel. Offline-Rendering wird zum Beispiel in der Filmproduktion oder der Visualisierung von Architektur eingesetzt. In dieser Arbeit wird ein physikalisch-basierter Pathtracer zur Evaluierung verwendet.

In der Diskussion des physikalischen Hintergrundes wurde aufgezeigt, dass Lichtinteraktion mit fester und gasförmiger Materie durch eine einheitliche Theorie erklärt werden kann. Maßgeblich für die Computergrafik ist jedoch, dass feste Materie vom Menschen vorrangig wahrgenommen wird. Daher konzentriert sich die meiste Arbeit in der Computergrafik auf die plausible Darstellung von Oberflächen. Oft wird angenommen, dass sich das Licht zwischen den Objekten durch ein Vakuum bewegt. Da die Computergrafik statt Atomen nur Objektoberflächen kennt, werden Phänomene von

## 2. HINTERGRUND

Licht im Medium, das die Objekte umgibt, getrennt behandelt. Dafür wird der Begriff *partizipierende Medien* verwendet.

Im Folgenden wird ein Überblick über Methoden zur Beleuchtungsberechnung in der Computergrafik gegeben. Abschnitt 2.2.1 ordnet den Begriff des physikalisch-basierten Rendering, den man in der Literatur immer wieder findet, in einen Gesamtzusammenhang ein. Abschnitt 2.2.2 betrachtet diejenigen Variablen, die in der Computergrafik so oder in ähnlicher Form typischerweise für die Beleuchtung in Anwesenheit partizipierender Medien benutzt werden und setzt diese in Beziehung zueinander. Abschnitt 2.2.3 erklärt in knapper Form das sogenannte Shadow Mapping, eine weit verbreitete Methode zum Darstellen von Schatten, die sich insbesondere auch für volumetrische Schatten eignet. Abschnitt 2.3 diskutiert einige Publikationen, die spezielle Renderingverfahren für God Rays vorschlagen.

Drei der in Abschnitt 2.3 angesprochenen Verfahren werden im nächsten Kapitel, das den praktischen Teil dieser Arbeit darstellt, näher analysiert, implementiert und evaluiert. Der Fokus im praktischen Teil liegt auf der Benutzung der Rendering-Pipeline, insbesondere des Fragmentshaders.

### 2.2.1. Physikalisch-basiertes Rendering

Für die folgenden Aussagen über Computergrafik wurde auf *Computer Graphics: Principles and Practice (3rd Edition)* [7] zurückgegriffen.

Die Computergrafik hatte zunächst kein Interesse am Lichttransport zwischen Partikeln. Das ist historisch nachvollziehbar, denn zur Zeit der Entwicklung erster Beleuchtungsmodelle waren Rechenleistung und Speicherkapazität sehr begrenzt. Statt Lösungen für das Problem "Lichttransport" wurden Lösungen für das Problem "Streuung an einem Punkt auf einer Objektoberfläche" entwickelt. Daher fasst man die dabei entstandenen Methoden als lokale Beleuchtung zusammen. Eine lokale Betrachtung grenzt die Einflussfaktoren der Berechnung des wahrgenommenen Lichts stark ein; konkret auf die Ausrichtung eines Punktes zur Lichtquelle und zum Auge. In Wirklichkeit spielen aber weit mehr Faktoren eine Rolle, wie zum Beispiel Reflexionen oder Schatten von benachbarten Objekten. Lokale Modelle beschränken sich also auf eine empirische Beobachtung und haben keine theoretische Fundierung. Ein Beispiel für empirische Modelle ist die Phong-Beleuchtung (1975) [15]. Vorteile der Phong-Beleuchtung, die noch heute zutreffen, sind die einfache Implementierung in bestehenden Rendering-Pipelines wie OpenGL und die schnelle aber plausible Visualisierung einer 3D-Szene, vor allem für Studenten.

Während immer leistungsfähigere Rechner zur Verfügung standen, wuchs der Anspruch realistischere Bilder zu erzeugen. Man konnte nun beginnen, die Beleuchtung eines Punktes global zu berechnen, also unter Einbeziehung von Licht, das nicht direkt

von der Lichtquelle kommt. Die dazu entwickelten Modelle haben im Gegensatz zur lokalen Beleuchtung meist physikalische Grundlagen. Zunächst trennen die Modelle Objekte in Geometrie und Material, also in positionsabhängige und positionsunabhängige Informationen. Weiterhin klassifizieren sie Materialien danach, wie sie mit Licht interagieren. Es wurde eine Reihe von Materialeigenschaften identifiziert, die für markante Fälle von Lichtstreuung verantwortlich sind. Zwei Eigenschaften, die man getrennt voneinander betrachtet sind Reflexion und Transmission. Reflektierendes Material kann perfekt spiegelnd sein; transmittierendes Material kann Licht in genau eine Richtung brechen. Neben diesen Spezialfällen können sie unter anderem zu einem gewissen Grad spekular oder diffus sein. Die Unterscheidung zwischen spekular und diffus wäre nicht nötig, wenn man Materialien bis hin zur Mikrostruktur genau beschreiben könnte, da die für Menschen sichtbare Lichtinteraktion aus vielen Lichtinteraktionen unter der Oberfläche eines Materials erwächst. Um die Komplexität gering zu halten, wird in der Computergrafik ein Maßstab gewählt, der dem menschlichen Auge angemessen ist. Materialien mit einer Kombination der genannten Eigenschaften werden durch eine Funktion definiert, die zwei Richtungsvektoren erwartet und eine Farbe zurückgibt. Eine solche Funktion nennt man Bidirectional Scattering Distribution Function (BSDF). Gewissermaßen gibt diese Funktion an, wie ein Objekt einfallendes in ausgehendes Licht umwandelt. Durch diese Definition von Materialien kann man den Lichttransport exakt berechnen, was durch Raytracing und ähnliche Algorithmen realisiert wird. Unter anderem, da diese Algorithmen BSDFs mit statistischen Verfahren auswerten, sind sie zum aktuellen Stand nicht echtzeitfähig. Echtzeitanwendungen wie Spiele greifen oft auf grobe Vereinfachungen oder Tricks zurück, um den Eindruck von globaler Beleuchtung zu erwecken. Solche Tricks gibt es für eine große Zahl von Effekten, die man unter globaler Beleuchtung einordnen kann.

Partizipierende Medien stellen einen Aspekt von globaler Beleuchtung dar. Die Beleuchtung von Medien wird zwar getrennt von Oberflächen betrachtet, doch eigentlich ist eine Oberfläche nichts anderes als die Grenze zweier Medien. So gesehen betrachtet man bei partizipierenden Medien das Innere von Materialen bzw. den Bereich unter der Oberfläche. Es gibt in der Computergrafik noch ein anderes Thema, das sich mit Lichtinteraktion unter der Oberfläche beschäftigt und in der Literatur als Volumenstreuung (engl. *subsurface scattering*) bezeichnet wird. Dieser Begriff umfasst die Untersuchung und Darstellung von Materialien wie menschliche Haut, Milch, Marmor oder Granit. Das Thema ist nicht zu verwechseln mit partizipierenden Medien, die sich mit den Effekten in Gasen wie zum Beispiel Dampf, Rauch, Dunst, Nebel usw. beschäftigen. Im Folgenden wird sich diese Arbeit mit einer Auswahl von Verfahren für partizipierende Medien beschäftigen. Zu Beginn soll nun ein physikalisch basiertes Modell für Lichtstreuung in der Atmosphäre betrachtet werden.

### 2.2.2. Lichtstreuung in der Atmosphäre

In der Publikation *Rendering Outdoor Light Scattering in Real Time* [5] beschreiben Naty Hoffman und Arcot J. Preetham ein Modell für die Berechnung der Interaktionen des Lichts mit der Atmosphäre. Dieses bietet unter anderem eine akkurate Alternative zu Modellen, die für das Rendern von Nebel verwendet werden, aber enthält allgemeingültige Grundlagen zu partizipierenden Medien, die im Folgenden Abschnitt betrachtet werden. Nebel ist eine Wolke aus kondensierten Wassertröpfchen und interagiert daher mit dem Licht durch Mie-Streuung. Diese verursacht sehr helle weiße Streuung. Ein anderer Effekt, der als Luftperspektive bezeichnet wird, tritt dagegen als Folge von Rayleigh-Streuung auf. Dieser lässt Objekte in größerer Entfernung bläulich erscheinen, da in den Weg des Lichts zum Beobachter von dazwischen liegenden Atomen blaues Licht eingestreut wird. Das Modell ist abstrakt genug, um beide Effekte zu unterstützen.

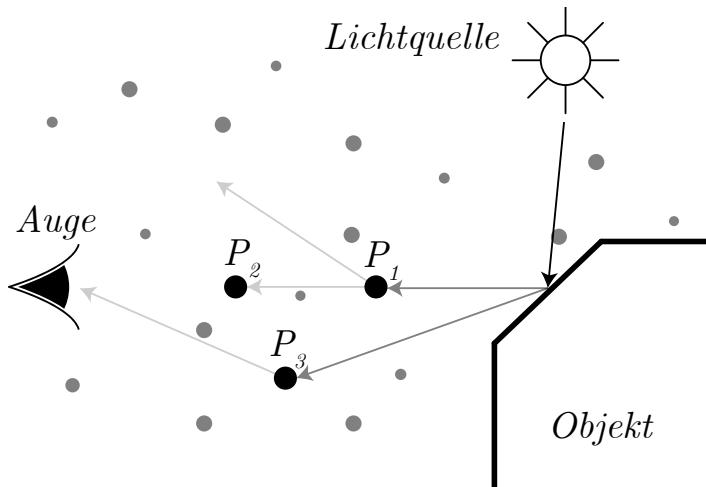


Abbildung 2.7.: Überblick über die 3 Phänomene von Lichtinteraktion in partizipierenden Medien. Es werden beispielhaft 3 Partikel betrachtet, an welchen Ausstreuung ( $P_1$ ), Absorption ( $P_2$ ) und Einstreuung ( $P_3$ ) stattfindet. Beachte, dass hier bereits Licht vom Objekt reflektiert wurde, was nicht zwingend der Fall sein muss. Die Interaktion des Lichts mit einem Partikel erfolgt mit der Wahrscheinlichkeit  $\beta^{(\lambda)} dx = \sigma^{(\lambda)} \rho dx$ , wobei  $\sigma^{(\lambda)}$  den von der Wellenlänge  $\lambda$  abhängigen Absorptions- bzw. Streuquerschnitt,  $\rho$  die Dichte des Mediums und  $dx$  den zurückgelegten Weg des Lichts angibt.

Eine wichtige Komponente des Modells stellt der Streuquerschnitt (siehe Abbildung 2.6) dar. Man unterscheidet den Einfluss atmosphärischer Partikel auf die Lichtintensität anhand dreier Phänomene, die im Folgenden erklärt und in Abbildung 2.7 gezeigt werden. Anstelle des englischen Wortes Radiance wird im Folgenden Lichtin-

tensität bzw. Lichtenergie verwendet. Für Irradiance wird einfallende Lichtenergie verwendet.

**Absorption** Partikel nehmen Lichtenergie auf und wandeln sie in Wärme um. Absorption findet statt, wenn die Frequenz der anregenden Lichtwelle im Bereich der Eigenfrequenz des Teilchens ist. Absorbierte Energie kann das Auge des Betrachters nicht mehr erreichen. Dies ist der erste Faktor der sogenannten Extinktion.

**Ausstreuung** Lichtenergie wird aus dem Pfad zum Auge des Beobachters abgelenkt. Das ist der zweite und letzte Faktor der Extinktion.

**Einstreuung** Lichtenergie wird in das Auge des Beobachters gestreut und addiert sich dort mit Lichtenergie aus anderen Richtungen auf.

Die Tatsache, dass ein Medium auch Licht emittieren kann, wird an dieser Stelle ignoriert. Folgende Variablen werden eingeführt, um obige Aspekte zu quantifizieren. Beachte, dass viele Variablen von der Lichtwellenlänge  $\lambda$  abhängen.

- Strecke  $x$  – Die Distanz, die das Licht durch ein Medium gewandert ist.
- Winkel  $\theta$  – Der Winkel zwischen urprünglicher Lichtrichtung und der Richtung, in der das Auge des Beobachters liegt.
- Lichtenergie  $L_0^{(\lambda)}$  – Lichtenergie am Startpunkt des Strahls, bevor dieser durch das Medium geht bzw. nachdem das Licht mit einem Objekt oder einem Partikel interagiert hat. Das Licht hat die Wellenlänge  $\lambda$ .
- Extinktionskoeffizient  $\beta_{ex}^{(\lambda)}$  – Stellt ein Maß für die Auslöschung von Licht mit der Wellenlänge  $\lambda$  auf dem Weg von der Lichtquelle zum Beobachter dar.
- Absorptionskoeffizient  $\beta_{ab}^{(\lambda)}$  – Der erste Teil des Extinktionskoeffizienten. Ein Maß für die Auslöschung von Licht mit der Wellenlänge  $\lambda$  durch Absorption. Für die Berechnung benötigt man
  - den Absorptionsquerschnitt  $\sigma_{ab}^{(\lambda)}$ , der analog zum Streuquerschnitt die Fläche bezeichnet, durch die Lichtenergie von Licht mit der Wellenlänge  $\lambda$  hindurchgehen muss, um von dem Partikel absorbiert zu werden (vgl. Abb. 2.6) und
  - die mittlere Dichte  $\rho_{ab}$  des absorbierenden Mediums.
- Streukoeffizient  $\beta_{sc}^{(\lambda)}$  – Stellt ein Maß für die durch einen Partikel (in alle Richtungen) gestreute Lichtenergie von Licht mit der Wellenlänge  $\lambda$  dar. Für die Berechnung benötigt man
  - den Streuquerschnitt  $\sigma_{sc}^{(\lambda)}$  (vgl. Abb. 2.6) und
  - die mittlere Dichte  $\rho_{sc}$  des streuenden Mediums. Meistens gilt  $\rho_{sc} = \rho_{ab}$ .

## 2. HINTERGRUND

- Winkelabhängiger Streukoeffizient  $\beta_{sc}^{(\lambda)}(\theta)$  – Stellt ein Maß für die durch einen Partikel in Richtung  $\theta$  gestreute Lichtenergie von Licht mit der Wellenlänge  $\lambda$  dar. Für die Berechnung benötigt man die Streuphasenfunktion  $\Phi(\theta)$ , die einer Richtung  $\theta$  eine Wahrscheinlichkeit zuordnet, mit der ein Partikel einstrahlendes Licht in diese Richtung streut.

Aus der Wellenlängenabhängigkeit ergibt sich das Problem, wie man die physikalische Größe der Lichtenergie in die für die Darstellung auf einem Monitor benötigten RGB-Werte umwandelt. Die Lichtenergie ist kontinuierlich über das Wellenlängenspektrum verteilt. Im physikalisch korrekten Offlinerendering verwendet man eine gewichtete Integration über alle Wellenlängen. Im Echtzeitrendering nimmt man drei Samples an drei Wellenlängen, um direkt und ohne weiteren Rechenaufwand, aber unter Einführung einer Vereinfachung, die RGB-Werte zu erhalten. Somit können alle wellenlängenabhängigen Variablen als RGB-Vektoren betrachtet werden. Nach dem Erhalten der RGB-Werte kann man zu große Helligkeitsunterschiede mit einer Gamma-Korrektur oder einem anderen Tonemapping ausgleichen.

Nun soll der Zusammenhang der Variablen erläutert werden, um den Lichttransport in partizipierenden Medien genauer zu verstehen. Es wird erläutert, dass das Problem keine ausreichend einfache Lösung für Echtzeitrendering hat. Daher werden Approximationen eingeführt, die für die Implementierung von Echtzeit-Methoden benötigt werden.

Betrachte zunächst das Phänomen der Absorption im Medium. Der Absorptionskoeffizient  $\beta_{ab}^{(\lambda)}$  ist definiert als Produkt aus Absorptionsquerschnitt  $\sigma_{ab}^{(\lambda)}$ , gemessen in  $m^2$ , und Dichte  $\rho_{ab}$  des Mediums, gemessen in  $1/m^3$ ; er hat daher die Einheit  $1/m$ . Die Bedeutung des Absorptionskoeffizienten lässt sich so verstehen: Es ist der Anteil  $dL^{(\lambda)}$ , den einfallendes Licht im Verhältnis zu seiner Energie  $L^{(\lambda)}$  pro Strecke  $dx$  durch Absorption in einem Medium verliert. Eine intuitivere Größe ist  $\beta_{ab}^{(\lambda)}dx$ , die die Wahrscheinlichkeit für die Absorption einer Energieeinheit auf der Strecke  $dx$  angibt. Der Absorptionskoeffizient ist abhängig von der Wellenlänge des einfallenden Lichts und der Eigenfrequenz der Atome des absorbierenden Partikels.

Der Streukoeffizient  $\beta_{sc}^{(\lambda)}$  ist dazu analog: Intuitiv ausgedrückt ist  $\beta_{sc}^{(\lambda)}dx$  die Wahrscheinlichkeit für ein Streuereignis. Zunächst betrachtet man nur die Lichtenergie, die aus dem Pfad zum Auge ausgestreut wird. Beide Vorgänge, Absorption und Ausstreuung, werden zusammengefasst zur Extinktion. Der Extinktionskoeffizient ist die Summe  $\beta_{ex}^{(\lambda)} = \beta_{ab}^{(\lambda)} + \beta_{sc}^{(\lambda)}$ . Da dieser angibt, wie viel Lichtenergie insgesamt pro Strecke verloren geht, kann man die Lichtenergie  $L^{(\lambda)}(x)$  nach Zurücklegen der Distanz  $x$  durch Lösen einer Differenzialgleichung wie folgt berechnen:

$$-\frac{dL^{(\lambda)}}{L^{(\lambda)}} = \beta_{ex}^{(\lambda)} \iff L^{(\lambda)}(x) = L_0^{(\lambda)} e^{-\beta_{ex}^{(\lambda)} x}, \quad (2.8)$$

wobei  $\lambda$  die Wellenlänge und  $L_0^{(\lambda)}$  die Lichtenergie am Startpunkt des Strahls ist, bevor dieser durch das Medium geht.

Es wurde vereinfachend angenommen, dass Dichte und Absorptions- bzw. Streuquerschnitt über das Medium hinweg konstant sind. Damit können nur idealisierte Aussagen gemacht werden, da Luft und andere Medien in Wirklichkeit nur annähernd homogen sind. Wegen der Schwerkraft der Erde sinkt die Dichte der Luft mit steigender Höhe. Die Koeffizienten für Luft und verschiedene Arten von Nebel bzw. Dunst in der Erdatmosphäre können in gemessenen Datenbeständen nachgeschlagen werden, die zum Teil in [16] oder [6] zusammengetragen wurden. Praktikable RGB-Koeffizienten für Luft sind nach [6]:  $\beta_{scAir}^R = 6,95 \cdot 10^{-6} m^{-1}$ ,  $\beta_{scAir}^G = 1,18 \cdot 10^{-5} m^{-1}$ ,  $\beta_{scAir}^B = 2,44 \cdot 10^{-5} m^{-1}$ . Beachte, dass der Koeffizient für rot eine Größenordnung kleiner ist als die Koeffizienten für grün und blau.

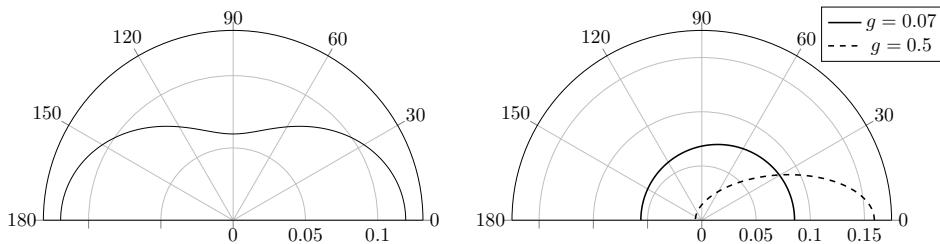


Abbildung 2.8.: Streuphasenfunktionen in Polarkoordinaten. Links: Wahrscheinlichkeit  $\Phi_R(\theta) = \frac{3}{16\pi}(1 + \cos^2\theta)$  für die Streuung von Licht an Rayleigh-Partikeln (bzw. in Luft) in Richtung  $\theta$ . Rayleigh-Partikel bewirken genauso viel Vorwärts- wie Rückwärtsstreuung und streuen bei  $\theta = 90^\circ$  am wenigsten. Rechts: Wahrscheinlichkeit  $\Phi_M(\theta) = \frac{(1-g)^2}{4\pi(1+g^2-2g\cos\theta)^{3/2}}$  für die Streuung von Licht an Mie-Partikeln (bzw. in Dunst oder Nebel). Mit  $g$  wächst die Vorwärtsstreuung und  $g$  wächst mit der Partikelgröße.  $g = 0.07$  beschreibt schwachen Dunst,  $g = 0.5$  beschreibt starken Nebel [6]. Beide Funktionen sind symmetrisch zur horizontalen Achse.

Gleichung 2.8 beschreibt den multiplikativen Effekt eines partizipierenden Mediums auf einfallendes Licht, bei dem die Lichtenergie durch Extinktion reduziert wird. Die Abschwächung wächst exponentiell mit der zurückgelegten Strecke. Gleichzeitig kommt aber Lichtenergie durch Einstreuung hinzu. Eingestraute Energie addiert sich auf die Energie des Strahls. Man kann berechnen, wie viel Lichtenergie in den Weg eines Strahls zum Auge eingestreut wird. Dazu benötigt man aber zunächst die Information, wie viel Lichtenergie in eine Richtung  $\theta$  gestreut wird. Dies gibt der Winkelabhängige Streukoeffizient  $\beta_{sc}^{(\lambda)}(\theta) = \beta_{sc}^{(\lambda)}\Phi(\theta)$  an, der je nach streuendem Partikel variiert. Abbildung 2.8 zeigt die Streuphasenfunktionen  $\Phi_R(\theta)$  und  $\Phi_M(\theta)$  für Rayleigh- und Mie-Partikel.

## 2. HINTERGRUND

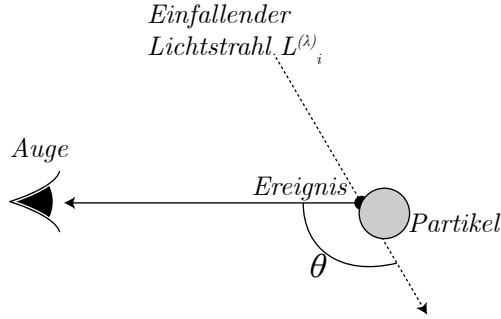


Abbildung 2.9.: Ein einzelnes Ereignis von Einstreuung. Der einfallende Lichtstrahl mit der Energie  $L^{(\lambda)}_i$  wird mit der Wahrscheinlichkeit  $\Phi(\theta)$  in das Auge des Beobachters gestreut. Hat das einfallende Licht die Wellenlänge  $\lambda$ , dann ist der winkelabhängige Streukoeffizient für diesen Partikel  $\beta_{sc}^{(\lambda)}(\theta) = \beta_{sc}^{(\lambda)}\Phi(\theta)$ . Angelehnt an Abbildung 4 in [5].

Betrachte ein einzelnes Ereignis von Einstreuung, das in Abbildung 2.9 gezeigt wird.  $L^{(\lambda)}_i(\theta) \cdot \Phi(\theta)$  ist die in das Auge eingestraute Lichtenergie aus der Richtung  $\theta$ . Da aus allen Richtungen Licht eingestreut werden kann, muss  $L^{(\lambda)}_i(\theta) \cdot \Phi(\theta)$  über eine Kugel integriert werden. Das Integral wird mit der Wahrscheinlichkeit für ein Streuereignis  $\beta_{sc}^{(\lambda)} dx$  multipliziert, um die gesamte eingestraute Lichtenergie auf einem Wegabschnitt  $dx$  zu erhalten. Zieht man  $\beta_{sc}^{(\lambda)}$  in das Integral und setzt  $\beta_{sc}^{(\lambda)}\Phi(\theta) = \beta_{sc}^{(\lambda)}(\theta)$  ein erhält man folgende Differenzialgleichung (in Kombination mit der Extinktion aus Gleichung 2.8):

$$\frac{dL^{(\lambda)}}{dx} = \underbrace{-\beta_{ex}^{(\lambda)} L^{(\lambda)}}_{\text{Extinktion}} + \underbrace{\int_{\Omega} L^{(\lambda)}_i(\theta) \beta_{sc}^{(\lambda)}(\theta) d\omega}_{\text{Einstreuung}}, \quad (2.9)$$

wobei  $\omega$  ein Abschnitt der Kugeloberfläche und  $\Omega$  die gesamte Kugeloberfläche ist. In [5] wird eine Lösung angegeben, die das Integral über Absorption, Ausstreuung und Einstreuung für infinitesimale Schritte  $dx$  vollständig angibt. Wichtig ist dabei, dass gestreutes Licht wieder Extinktion und Einstreuung durchläuft, was die Berechnung sehr aufwendig macht. Daher wird mehrfache Streuung in der Echtzeitgrafik meist ignoriert. An dieser Stelle soll nur festgehalten werden, dass man keine Auswertung erwarten kann, die gleichzeitig effizient und exakt ist.

Für Echtzeitanwendungen hat sich eine Annäherung als praktikabel erwiesen. Die Annäherung gilt für Einfachstreuung in homogenen Medien. Man ignoriert mehrfache Streuung, beachtet also nur Licht, das nach einem Streuereignis das Auge erreicht, und nimmt eine konstante Dichte für das Medium an. Dann genügt die folgende aus [5]

entnommene Gleichung für die Farbe eines Pixels:

$$L(s, \theta) = L_0 \underbrace{e^{-(\beta_R + \beta_M)s}}_{\text{Extinktion}} + \underbrace{\frac{\beta_R(\theta) + \beta_M(\theta)}{\beta_R + \beta_M} E_{\text{Lichtquelle}} (1 - e^{-(\beta_R + \beta_M)s})}_{\text{Einstreuung}}, \quad (2.10)$$

wobei  $s$  die Entfernung vom Objekt zum Auge ist,  $\theta$  der Winkel zwischen dem Strahl von der Lichtquelle und dem Strahl zum Auge,  $L_0$  die Farbe des vom Objekt reflektierten Lichts und  $E_{\text{Lichtquelle}}$  die (monochrome) Energie der Lichtquelle. Die Koeffizienten  $\beta_R$  und  $\beta_M$  für Rayleigh- und Mie-Partikel sind über das Medium hinweg konstant. Der Term für Einstreuung enthält einen Extinktionsfaktor, da auch eingestrautes Licht auf dem Weg zum Auge Extinktion durchläuft. An dieser Stelle sollte noch erwähnt werden, dass man oft andere Formulierungen der Gleichung findet. Der Faktor  $\frac{\beta_R(\theta) + \beta_M(\theta)}{\beta_R + \beta_M}$  wird auch als Albedo bzw. Rückstrahlvermögen bezeichnet und kann intuitiv als Helligkeit des Mediums verstanden werden. Es kann auch  $\sigma_s$  als einziger Streukoeffizient verwendet werden, wenn man Rayleigh- und Mie-Streuung nicht unterscheidet und die Dichte sowie die Phasenfunktion vom Koeffizienten entkoppelt. Als Extinktionskoeffizient kann auch  $\sigma_t$  und als Albedo  $\alpha = \sigma_s / \sigma_t$  benutzt werden [3].

Bei Lichtstreuung in der Atmosphäre ist die Lichtquelle typischerweise die Sonne. Die Teilstrecke von der Sonne zum Objekt ist durch die vereinfachte Gleichung, die nur für homogene Medien gilt, nicht zu berechnen, da das Licht auf dem Weg sowohl die Sonnenatmosphäre als auch die Erdatmosphäre, welche beide wegen der Schwerkraft in großer Höhe dünner sind und mit sinkender Höhe dichter werden, durchquert hat. Es besteht aber die Möglichkeit gemessene Werte des verbliebenen Lichts nach dieser Teilstrecke zu verwenden. Für die meisten Anwendungen ist die vereinfachte Gleichung somit angemessen, da sich die Objekte in vergleichsweise geringen Höhenunterschieden befinden, für die eine konstante Dichte der Atmosphäre angenommen werden kann. Damit hat man ein Modell für einfache Streuung in nicht-emittierenden homogenen Medien erhalten.

Eine zusätzliche Herausforderung ergibt sich durch die Anwesenheit von Schatten in partizipierenden Medien. Eine einzige Auswertung von Gleichung 2.10 an jedem Pixel eines Bildes bezieht keine Schatten mit ein, da Einstreuung auf der gesamten Strecke  $s$  berechnet wird. Daher müssen Verfahren, die Schatten einbeziehen,  $s$  an mehreren Punkten abtasten. Dies entspricht der Approximation von

$$L(s, \theta) = L_0 e^{-(\beta_R + \beta_M)s} + \int_0^s L_{\text{Einstreuung}}(dx, \theta) S(x) dx, \quad (2.11)$$

wobei  $L_{\text{Einstreuung}}(dx, \theta)$  der obige Einstreuungsterm für eine Teilstrecke  $dx$  ist.  $S(x)$  steht für die Sichtbarkeit der Lichtquelle am Punkt  $x$  bzw. gibt an, ob der Punkt im Schatten liegt oder nicht.

### 2.2.3. Shadow Mapping

Man kann zwei Fälle unterscheiden, bei denen ein Punkt im Schatten liegt. Im ersten Fall liegt der Punkt auf einer Objektoberfläche, deren Normale im Winkel von  $> 90^\circ$  von der Richtung der Lichtquelle weg zeigt. Dann kann ein lokales Beleuchtungsmodell bereits die Antwort geben, dass der Punkt im Schatten liegt. Zeigt die Normale andererseits nicht von der Lichtquelle weg, oder dem Punkt kann keine Normale zugeordnet werden, da er in einem partizipierenden Medium liegt, muss ein zweiter Fall geprüft werden, bei dem der Punkt von anderen Objekten beschattet wird. Daraus folgt, dass zur endgültigen Bestimmung von Schatten Informationen über die gesamte Szene benötigt werden. Aus diesem Grund sind Schatten im Allgemeinen ein Problem der globalen Beleuchtung.

In der Rendering-Pipeline stehen globale Informationen nicht zur Verfügung. Es gibt daher viel Forschungsarbeit, die sich mit der effizienten Bestimmung von Schatten beschäftigt. Eine Methode, die sich heute durchgesetzt hat, ist das Shadow Mapping [19]. Shadow Mapping basiert darauf, dass der Tiefenwert jedes Punktes aus der Sicht der Lichtquelle eine ausreichende Information für einen einfachen Schattentest darstellt. Um die Tiefenwerte zu erhalten, rendert man die Szene einmal zusätzlich. Die Weltkoordinaten werden in das Koordinatensystem der Lichtquelle transformiert. Der Framebuffer muss nicht komplett beschrieben werden, sondern nur der Z-Buffer, da nur die Tiefenwerte benötigt werden. Die Tiefenwerte werden anschließend in einen Zwischenspeicher, zum Beispiel eine Textur, kopiert. Dieser Zwischenspeicher wird Shadow Mapping genannt. Beim nächsten Durchgang, dem Rendern aus Sicht der Kamera, werden die Weltkoordinaten in beide Koordinatensysteme, das der Kamera und das der Lichtquelle transformiert. Die Tiefenwerte in der Shadow Mapping stellen nun den Vergleichswert für die im aktuellen Durchgang erhaltenen Koordinaten aus Sicht der Lichtquelle dar. Der Schattentest wird somit zu einem Vergleich zwischen zwei Tiefenwerten.

Die soeben beschriebene einfachste Form des Shadow Mapping hat verschiedene Limitierungen. Zunächst werden bei einem Renderdurchlauf aus Sicht der Lichtquelle nur Tiefenwerte im Sichtkegel erfasst. Der Sichtkegel der Lichtquelle muss somit den Sichtkegel der Kamera möglichst abdecken. Außerdem ist die Auflösung der Shadow Mapping begrenzt. Daher steht selbst bei einer guten Abdeckung des Sichtkegels nicht für jeden von der Kamera gesehenen Punkt ein Tiefenwert aus Sicht der Lichtquelle zur Verfügung. Die Folge davon ist Aliasing. Schließlich geht einfaches Shadow Mapping von einer unendlich kleinen (Punkt-)Lichtquelle aus. Der Schattentest ist binär. Somit können nur harte Schatten abgeleitet werden. Erweiterungen umfassen daher die Erstellung mehrerer Shadow Maps, die Einführung eines Schwellenwertes und Glättung der Schattenkanten.

Eine Glättung der Schattenkanten ermöglicht das scheinbare Vorhandensein von Halbschatten (Penumbra) sowie Anti-Aliasing. Die Glättung wird durch Anwenden von

Filtern erreicht. Ein üblicher Filter führt den Schattentest für mehrere Tiefenwerte in einer bestimmten Umgebung durch und bildet den Durchschnitt. Die Technik heißt *percentage closer filtering* (PCF). Das Ergebnis ist nicht mehr binär, sondern liegt im Intervall  $[0, 1]$ . Mit wachsender Umgebung erscheinen die Schattenkanten weicher, jedoch wächst auch die Anzahl teurer Texturzugriffe.

Ein Vorteil des Shadow Mapping ist dessen Effizienz durch die Ausnutzung des Z-Buffers und der Hardwarefunktionalität für den Tiefentest auf modernen GPUs.

## 2.3. Literaturüberblick

Lichtstreuung in Medien wird durch die Anwesenheit von Schatten als God Rays sichtbar. Verschiedene existierende Verfahren machen sich dies zunutze. Die Titel der meist englischen Publikationen verwenden neben dem Begriff *participating media* typischerweise noch die Schlagwörter *volumetric lighting*, *volumetric light scattering*, *volumetric shadows*, *atmospheric scattering* und *single scattering*; seltener findet man *light shafts*, *shafts of light*, *crepuscular rays*, *sun beams* und *god rays*. Es werden zunächst zwei bildbasierte Ansätze vorgestellt, die nicht physikalisch-basiert, jedoch einfach zu implementieren sind und trotz Problemen beim Realismus ansprechende Ergebnisse liefern. Andere Ansätze führen dagegen ein Ray-Marching auf den korrekten Sichtstrahlen im Sichtvolumen durch und benutzen physikalische Parameter.

### Bildbasierte Methoden

Bildbasierte Methoden finden sich im Bereich der Computerspiele, da hier künstlerische Anpassung an die Spielwelt Vorrang vor physikalischer Genauigkeit hat. Ein bildbasiertes Verfahren wurde von Kenny Mitchell publiziert [12]. Das Verfahren approximiert Gleichung 2.10. Die physikalischen Parameter werden durch eher praktische Parameter ersetzt. Zusätzlich werden volumetrische Schatten einbezogen, indem die Verdeckung der Lichtquelle geschätzt wird, ohne volumetrische Informationen zu verwenden. Es wird allerdings ein sogenanntes Verdeckungsbild benötigt, das alle potenziell verdeckenden Objekte einfarbig schwarz darstellt. Von der Position der Lichtquelle im Bildraum ausgehend, wird das Verdeckungsbild *verwischt* (der Prozess wird auch als Radial Blur bezeichnet). Das dabei entstandene Bild wird mit dem Bild der Szene kombiniert.

Eine ähnliche Methode wurde von Tiago Sousa auf der GDC 2008 [17] präsentiert. Anstelle dem Verdeckungsbild wird von Sousa eine Tiefenkarte benutzt. Die Anwendung des Radial Blur-Filters besteht pro Pixel aus  $n$ -maligem Abtasten des Bildes und der Summation der abgetasteten Werte. Dabei wird immer zu einem Zentrum hin abgetastet. Wie schon im Verfahren von Mitchell, wird auch von Sousa als Zentrum die Lichtquelle verwendet. Sousa benutzt eine feste Schrittweite während Mitchell  $n$  als fest definiert.

## 2. HINTERGRUND

Außerdem werden von Sousa die auffälligen harten Übergänge bei zu geringer Anzahl Samples durch verzahnte Abtastung (engl. *interleaved sampling*) [9] reduziert.

Das Verfahren von Mitchell benutzt reguläre Abtastung (engl. *regular sampling*). Eine mögliche Verbesserung ist irreguläre Abtastung (engl. *irregular sampling*). Die harten Übergänge, die bei zu wenigen Samples auftreten, werden dabei durch die irreguläre Abtastung maskiert, indem die Samples zufällig verteilt werden. Das stattdessen auftretende Rauschen (engl. *noise*) wird vom Menschen weniger wahrgenommen.

### Ray-Marching Methoden

Höhere Genauigkeit als bildbasierte Methoden bieten Ray-Marching-basierte Methoden, die Sichtstrahlen im dreidimensionalen Raum abtasten.

Eine Ray-Marching-basierte Methode wurde von Balázs Tóth und Tamás Umenhoffer [18] publiziert. Ähnlich wie im bildbasierten Verfahren von Mitchell werden Strahlen durch das Medium generiert und in  $n$  Schritten abgetastet. Allerdings arbeitet das Verfahren von Tóth korrekter, indem es Informationen über die Szene hinzuzieht, die im Bildraum nicht verfügbar sind. Zunächst wird die Position des Oberflächenpunktes benötigt, der durch einen Pixel sichtbar ist. Weiterhin wird die Position der Kamera benötigt, um die Richtung und Entfernung zum Oberflächenpunkt zu berechnen. Danach kann der erhaltene Sichtstrahl abgetastet werden. An jedem abgetasteten Punkt wird eine Shadow Mapping gelesen, um zu bestimmen, ob der Punkt im Schatten liegt. Somit wird schließlich das gesamte Sichtvolumen abgetastet.

Der Ray-Marching-Ansatz benötigt generell eine hohe Anzahl Samples  $n$ . Die Laufzeitkomplexität ist  $O(whn)$ , wobei  $wh$  die Bilddauflösung ist. Es wurden verschiedene Erweiterungen vorgeschlagen, die die Laufzeit verbessern. Eine ebenfalls in [18] zu findende Erweiterung ist die verzahnte Abtastung, bei der die  $n$  Samples auf einen Pixelblock aufgeteilt werden. Die Laufzeitkomplexität mit verzahnter Abtastung ist  $O(wh \frac{n}{M^2})$ , wobei  $M \times M$  die Größe eines Pixelblocks ist. Eine weitere mögliche Erweiterung ist die Benutzung einer festen Schrittweite anstatt einer festen Anzahl Samples. So kann eine unnötig hohe Abtastdichte bei Pixeln mit geringer Tiefe vermieden werden.

Baran et al. [1] beschleunigen die Abtastung, indem sie ausnutzen, dass Lichtstrahlen, die an einem Punkt geblockt wurden, zu allen nachfolgenden Samples, die sie sonst noch durchlaufen worden wären, keine Streuung mehr beitragen. Dies verhindert unnötige Schattentests und unnötige Berechnungen der Streuung an den nachfolgenden Samples. Die Laufzeitkomplexität wird auf  $O(w(h+n) \log n)$  reduziert. Der Ansatz erfordert aber den Aufbau einer Datenstruktur, die die Endpunkte der Lichtstrahlen speichert und bestimmten Samples zuordnet. Auf diese Datenstruktur muss während der Abtastung zugegriffen werden, was bei einer parallelen Abtastung zu Schwierigkeiten bei der Synchronisation führt. Diese Eigenschaft macht den Ansatz insbesondere für die Benut-

zung mit der Rendering-Pipeline inkompatibel. Eine Verbesserung der Datenstruktur zur Vereinfachung der Parallelisierbarkeit wurde von Chen et al. [2] publiziert.

Ein Verfahren von Klehm et. al [10] benutzt eine erweiterte Shadow Map, um das Ray-Marching komplett zu umgehen. Die Shadow Mapping wird rektifiziert, sodass eine einzelne Zeile der Schattenverteilung entlang eines Sichtstrahls entspricht. Danach wird ein Filter auf die Shadow Mapping angewandt, dessen Kernel die Texel einer Zeile benutzt, um das in den Sichtstrahl gestreute Licht in jedem Texel zu speichern. Während ein Ray-Marching-basiertes Verfahren an jedem Sample aus der Shadow Mapping liest, genügt mit dem Verfahren von Klehm et al. ein einziger Zugriff pro Pixel. Von Peters et al. [14] wurde eine Verbesserung publiziert, die mittels Moment Shadow Mapping bei gleichbleibender Bildqualität statt 256 Bit nur 64 Bit Speicherplatz pro Texel benötigt.



# 3. Praktische Umsetzung

Der Hauptteil dieser Arbeit wendet sich der praktischen Umsetzung zu. Für die Umsetzung wurden die Verfahren von Kenny Mitchell [12], Tiago Sousa [17] sowie Balázs Tóth und Tamás Umenhoffer [18] ausgewählt. Hierbei handelt es sich um zwei bildbasierte Verfahren (im Folgenden auch als Post-Processing bezeichnet) und ein Ray-Marching-basiertes Verfahren. In den Abschnitten 3.1 und 3.2 werden beide Verfahrenstypen näher analysiert. Abschnitt 3.3 enthält die Beschreibung der eigentlichen Implementierung. Abschnitt 3.4 evaluiert die implementierten Verfahren im Hinblick auf Plausibilität und Rechenzeit.

## 3.1. Post-Processing

Das von Kenny Mitchell in *Volumetric Light Scattering as a Post-Process* [12] publizierte Verfahren lässt sich auf zwei Arten interpretieren. Zunächst wird es von dem physikalischen Standpunkt betrachtet, der schon im Abschnitt 2.2.2 über partizipierende Medien eingenommen wurde und den auch Mitchell als Ausgangspunkt wählt. Danach wird das Verfahren aus einer empirischen Sicht betrachtet.

### Physikalische Interpretation

Vor dem Hintergrund der Theorie partizipierender Medien nähert das Verfahren Gleichung (2.11) an. Das Integral lässt sich näherungsweise durch eine Summe ausdrücken:

$$L(s, \theta) = L_0 + \sum_{i=0}^n \text{Extinktion}^{-i} L_{\text{Einstreuung}}(s_i, \theta_i) S(s_i) \quad (3.1)$$

Die Auswertung der Gleichung erfolgt durch eine Abtastung des Bildes in  $n$  Schritten. Somit ist die Laufzeitkomplexität bei Bildauflösung  $wh$   $O(whn)$ . Im Bildraum stehen keine volumetrischen Informationen zur Verfügung. Die Strecke  $s$  aus Gleichung (2.11) kann folglich nicht abgetastet werden, da diese genau auf einem Sichtstrahl liegt. Stattdessen wird für jeden Pixel die Strecke zur Position der Lichtquelle im Bildraum abgetastet. Die Samples werden zur Auswertung von  $L_{\text{Einstreuung}}(s_i, \theta_i)$  benutzt, obwohl sie keine ausreichende Information über die tatsächliche Einstreuung enthalten, was durch Abbildung 3.1 veranschaulicht wird. Gleichzeitig ermöglichen die Samples auf

### 3. PRAKTISCHE UMSETZUNG

der Strecke vom Pixel zur Lichtquelle im Bildraum eine Abschätzung von  $S(s_i)$ , wie in Abbildung 3.2 gezeigt wird. Beachte, dass  $\theta$  in Gleichung (3.1) vorkommt, aber mit einem bildbasierten Verfahren keine anisotrope Streuphasenfunktion benutzt werden kann. Diese Interpretation des Verfahrens zeigt, dass die Vereinfachungen für den Bildraum große Abweichungen vom Modell mit sich bringen.

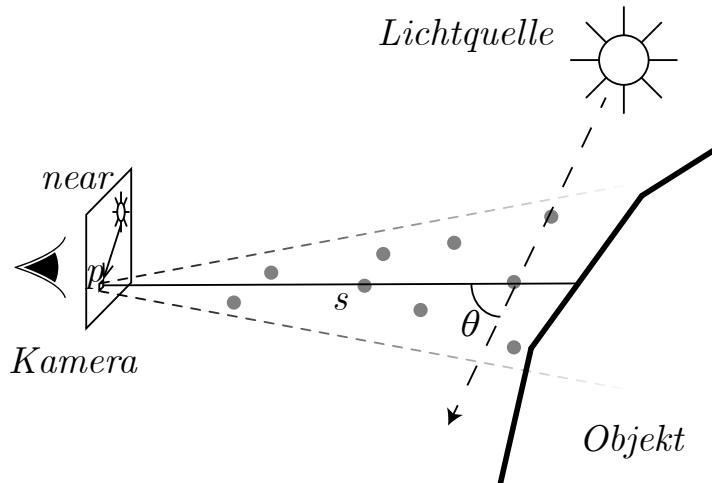


Abbildung 3.1.: Gezeigt wird ein Pixel  $p$  in der Bildebene und die Partikel des Mediums, die sich im Sichtvolumen von  $p$  befinden. Diese Partikel verursachen die Einstreuung, die man bei Einfachstreuung beachtet. Während eine Abtastung von  $s$  korrekt wäre, liefert auch eine Abtastung des eingezeichneten Vektors im Bildraum ein plausibles Ergebnis. Da  $\theta$  im Bildraum nicht bestimmt werden kann, muss isotrope Streuung angenommen werden.

#### Empirische Interpretation

In der empirischen Interpretation basiert das Verfahren auf der simplen Beobachtung, dass God Rays in einer zweidimensionalen Sicht einfach nur helle Streifen sind, die von einem gemeinsamen Zentrum, der Sonne, ausgehen. Auf einem Bild aus Wasserfarben könnte man diese Streifen einfach erzeugen, indem man von der Sonne aus beginnt, das Bild mit dem Finger zu verwischen. Das Verwischen bewirkt, dass Punkte in der Umgebung der Sonne einen Teil der Farbe der Sonne erhalten. Genau dieses Prinzip wird in einem Post-Process umgesetzt, der als Radial Blur bezeichnet wird. Auf einer abstrakten Ebene führt der Radial Blur-Algorithmus pro Pixel folgende 3 Schritte aus:

1. Laufe von der Lichtquelle zum Pixel.

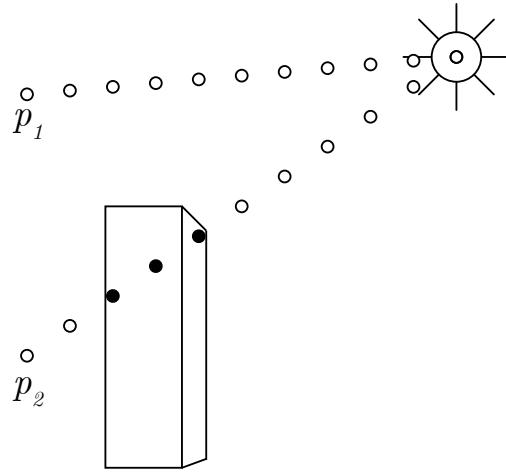


Abbildung 3.2.: Abschätzung von Schatten im Bildraum. Hier wird die Abtastung für zwei Pixel  $p_1$  und  $p_2$  gezeigt. Die Wahrscheinlichkeit der Sichtbarkeit der Lichtquelle an einem Punkt ergibt sich aus dem Anteil der Samples, die auf einem verdeckenden Objekt liegen:  $S(s_i) = \frac{n_v}{n}$ .

2. Nehme auf dem Weg in bestimmten Abständen die Farbe und multipliziere sie mit einer absteigenden Gewichtung.
3. Addiere die Summe der gewichteten Farben auf die Pixelfarbe.

Der Radial Blur-Filter ist die Basis des Verfahrens von Mitchell. Um nur die Farbe der Sonne (bzw. die Farbe der emittierenden Region, die auch den Himmel umfassen kann) in die erzeugten Streifen einzubeziehen, wird in Schritt 2 die Farbe eines nicht-emittierenden Objekts mit 0 gewichtet. Aus der empirischen Sicht liegt dem Verfahren keine physikalische, sondern eher eine künstlerische Motivation zugrunde.

Um den Effekt zu kontrollieren und an die Gegebenheiten verschiedener Szenen anpassen zu können, werden folgende Parameter eingeführt: *exposure* regelt die Gesamthelligkeit, *decay*  $\in [0, 1]$  den exponentiellen Abfall der Strahlen und *density*  $\in [1, \infty]$  die Schrittweite. Die parametrisierte Form von Gleichung (3.1) ist

$$L(s, \theta) = L_0 + \text{exposure} \sum_{i=0}^n (\text{decay})^i L_{\text{Einstreuung}}(s_i, \theta_i) S(s_i). \quad (3.2)$$

Die Schrittweite ergibt sich durch  $s / (n \cdot \text{density})$ , wobei  $s$  die Strecke vom Pixel zur Lichtquelle im Bildraum ist.

Das Verfahren von Mitchell ist grundsätzlich auch mit mehreren Lichtquellen kompatibel. Dabei muss für jede Lichtquelle ein separates Verdeckungsbild erzeugt werden, da eine Lichtquelle von einer anderen verdeckt werden kann. Der Radial Blur-Filter

### 3. PRAKTISCHE UMSETZUNG

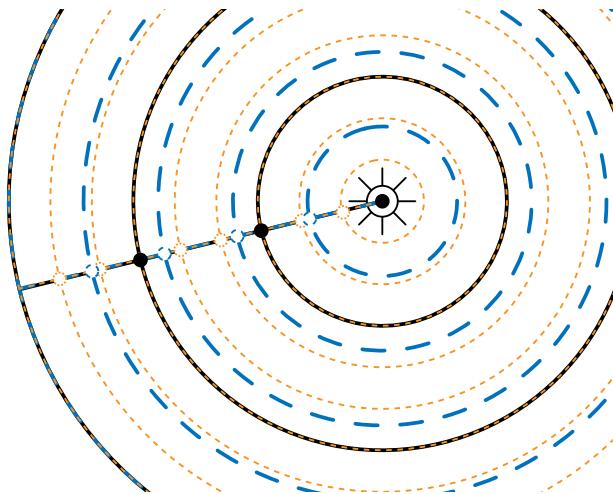


Abbildung 3.3.: Abtaststrategien in einem radialen Raster. Die Lichtquelle liegt im Zentrum. Das Verfahren von Sousa tastet nacheinander auf unterschiedlichen Rastern ab (verzahnte Abtastung); hier erst schwarz, dann blau, dann orange. Das Verfahren von Mitchell tastet nur auf einem Raster ab (reguläre Abtastung).

muss auf jedes Verdeckungsbild angewandt und die Ergebnisbilder addiert werden. Die Anzahl der Lichtquellen ist somit ein linearer Faktor für den Rechenaufwand.

#### Verfahren von Sousa

Ein ähnliches Verfahren von Tiago Sousa, das für die CryEngine entwickelt wurde [17], basiert ebenfalls auf dem Radial Blur-Filter, aber wendet eine andere Abtaststrategie an. Der Filter wird in  $N$  Durchgängen mit jeweils fester Schrittweite  $n^{-1}, n^{-2}, \dots, n^{-N}$  angewandt, wobei  $n$  die maximale Anzahl Samples pro Durchgang ist. Die Abtastung in einem Durchgang kann vor Erreichen von  $n$  Samples abgebrochen werden, wenn die Strecke zwischen Pixel und Lichtquelle verlassen wurde. Es genügt jedoch eine geringe Größe für  $n$ , sodass für die meisten Pixel nicht die gesamte Strecke abgetastet werden muss. Die Ausgabe jedes Durchgangs ist die Eingabe für den nächsten Durchgang. Die Abtaststrategie ist eine Form von verzahnter Abtastung [9] und wird in Abbildung 3.3 veranschaulicht. Durch verzahnte Abtastung werden auffällige harte Kanten bei niedriger Abtastrate verhindert. Insgesamt werden dadurch weniger Samples notwendig.

Bei mehreren Lichtquellen wird analog zum Verfahren von Mitchell vorgegangen. Der Radial Blur-Algorithmus wird für jede Lichtquelle separat durchlaufen. Die einzelnen Ergebnisbilder werden addiert.

## 3.2. Ray-Marching

Anwendungen mit bestimmten Anforderungen an physikalische Korrektheit berechnen volumetrische Streuung nicht im Bildraum. Die Idee der Verfahren, die Ray-Marching verwenden, ist die Abtastung des gesamten Sichtvolumens entlang der Sichtstrahlen, die von jedem Pixel ausgehen. Eine notwendige Voraussetzung für die meisten Echtzeitverfahren, die keine starken Einschränkungen der Szenenkomplexität erfordern, ist momentan das Ignorieren von mehrfacher Streuung. Betrachtet man ausschließlich Einfachstreuung, kann jeder Sichtstrahl unabhängig behandelt werden, was für eine parallele Berechnung wichtig ist. Für einen Sichtstrahl  $s$  muss Gleichung (2.11) ausgewertet werden. Im Folgenden wird eine Lösung von Balázs Tóth und Tamás Umenhoffer [18] betrachtet.

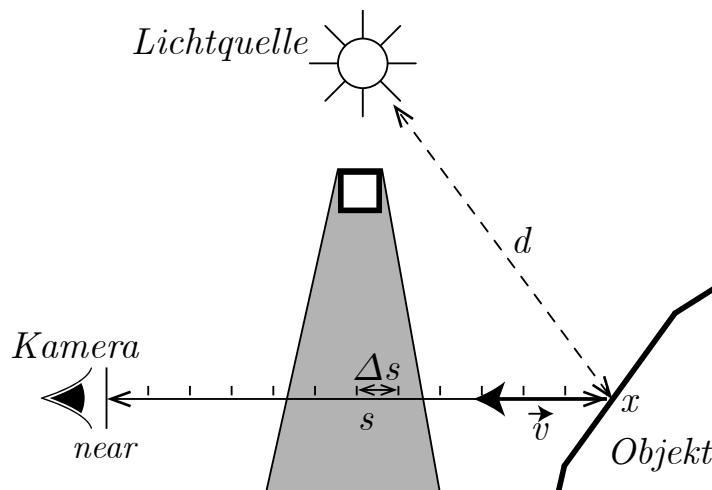


Abbildung 3.4.: Die Strecke  $s$  wird in  $n$  Schritte der Länge  $\Delta s$  unterteilt. Der Prozess des Ray-Marching bewegt den Punkt  $x$  an der Objektoberfläche schrittweise entlang des Vektors  $\vec{v}$ , der die Kamerarichtung angibt. Für jedes Schritt  $i$  wird der Einstreuungsterm ausgewertet und ein Schattentest durchgeführt, um den Wert von  $S(x_i) \in [0, 1]$  zu erhalten.  $d$  ist die Entfernung zwischen  $x$  und der Lichtquelle.

Das Verfahren formuliert den Lichttransport im partizipierenden Mediums ohne Streu- und Extinktionskoeffizienten, die in Abschnitt 2.2.2 benutzt wurden, sondern nutzt nur die Dichte  $\rho$  des Mediums. Das bedeutet, es können keine wellenlängenabhängigen Effekte berechnet werden. Es liegt also ein monochromes Modell zugrunde. Die Albedo wird mit  $\alpha$  bezeichnet.  $\Phi(\theta)$  ist die Streuphasenfunktion.

Das Integral in Gleichung (2.11) wird durch eine Riemann-Summe angenähert. Die Summe wird iterativ ausgewertet. Abbildung 3.4 stellt einen Überblick über den Prozess

### 3. PRAKTISCHE UMSETZUNG

dar. In mathematischer Notation ist

$$L(s, \theta) = L_0 e^{-\rho s} + \sum_{i=0}^n e^{-\rho(i \cdot \Delta s)} \underbrace{\Phi(\theta) \rho \alpha}_{\text{Einstreuung}} \frac{P}{4\pi d_i^2} e^{-\rho d_i} \Delta s S(x_i), \quad (3.3)$$

wobei  $\Delta s = s/n$  die Schrittweite ist. Der Ausdruck  $\frac{P}{4\pi d_i^2}$  gibt die von der Lichtquelle emittierte Leistung  $P$  nach Zurücklegen der Entfernung  $d_i$  zum Punkt  $x_i$  an. Es ist außerdem zu beachten, dass die Faktoren für die exponentielle Extinktion des Lichts im Medium hier nicht sukzessive ausgewertet werden. Das bedeutet, dass in der  $i$ -ten Iteration der Wert der  $(i-1)$ -ten Iteration nicht benötigt wird und die einzelnen Samples somit unabhängig sind. Dies kann in Optimierungen des Verfahrens für verschiedene Abtaststrategien ausgenutzt werden (z.B. verzahnte oder irreguläre Abtastung). Die Funktion  $S(x)$  gibt die Schattenverteilung entlang des Sichtstrahls an. In jeder Iteration wird  $S(x_i) \in [0, 1]$  ausgewertet. Hierfür wird Shadow Mapping (siehe Abschnitt 2.2.3) eingesetzt.

Erweitert man das Verfahren um verzahnte Abtastung, werden die ursprünglich für einen Pixel verwendeten  $n$  Samples auf einen Block von  $M \times M$  Pixeln periodisch aufgeteilt. Dies reduziert die Laufzeitkomplexität von  $O(whn)$  auf  $O(wh\frac{n}{M^2})$ . Unter der Annahme, dass die Samples im Block nah an einem Sichtstrahl liegen, erhält man eine ähnliche Lösung wie bei  $n$  Samples pro Pixel, da die Lichtstreuung bei benachbarten Pixeln tendenziell ähnlich ist.

## 3.3. Implementierung

Im Folgenden wird die praktische Umsetzung der in den vorangehenden Abschnitten 3.1 und 3.2 analysierten drei Verfahren erläutert. Für die Implementierung wurde ein Framework in C++ (Version 11) geschrieben, das die Rendering-Pipeline über OpenGL (Version 4.3) anspricht. Die Rendering-Pipeline und OpenGL werden als bekannt vorausgesetzt. In Abschnitt 3.3.1 folgt zunächst ein kurzer Überblick über das Framework. Es ist zu beachten, dass die Beschreibung sich auf relevante Teifunktionen beschränkt und keine vollständige Dokumentation ist. Abschnitt 3.3.2 beschreibt detailliert die Implementierung des Verfahrens von Mitchell im Framework. Abschnitt 3.3.3 fährt mit dem Verfahren von Sousa fort. Abschnitt 3.3.4 beschreibt sowohl die Implementierung des Verfahrens von Tóth und Umenhoffer, als auch des dazu benötigten Shadow Mapping.

### 3.3.1. Framework

Das Framework setzt sich aus Datenklassen, Modulklassen und Shadern zusammen. Durch die Datenklassen werden allgemein benötigte OpenGL-Funktionen gekapselt.

Die Modulklassen sorgen für die Erweiterbarkeit des Frameworks. Jedes der drei Verfahren wurde als Modul implementiert. Die Kontrolle über die Funktionen eines Moduls liegt beim Framework. Somit werden Module auf einen einheitlichen Programmfluss beschränkt. Shadercode wird in externen Dateien gespeichert und durch eine entsprechende Datenklasse in das Programm geladen. Es gibt folgende Datenklassen:

**Model** hält Vertexdaten in Rohform und Modellmatrizen.

**ModelType** hält Vertexattribute, Instanzattribute, Zeichenprimitiv und Shader. Wird benutzt, um Modelle mit demselben Speicherlayout zu gruppieren.

**VertexAttribute** hält das Speicherformat der Vertexdaten. Genauer: die Speicherposition eines Vertexattributes und dessen Variablenname im Shader; das Speicherlayout wird durch eine Liste von Vertexattributen beschrieben.

**InstanceAttribute** hält das Speicherformat eines Instanzattributs. Wird für sog. Geometry Instancing benutzt.

**Shader** kompiliert Shadercode und hält das OpenGL-Handle des kompilierten Programms.

**GPUBuffer** repräsentiert Daten auf der GPU, u.a. Vertexbuffer, Instanzbuffer, Uniform-Buffer, Shader-Storage-Blocks und Framebuffer.

**Uniform** repräsentiert globale Shadervariablen, sog. Uniforms. Bei Uniforms müssen regelmäßig Updates der Daten auf der GPU durchgeführt werden. Für Uniform-Updates verwendet das Framework anonyme Funktionen, die seit C++11 zur Verfügung stehen, als sog. Callbacks.

Die Klasse `World` stellt eine Art Controller-Klasse dar, die die Modelle und Shadervariablen verwaltet. Der Konstruktor von `World` erhält eine Liste aus Modellen und lädt die Vertexdaten von Modellen desselben Typs jeweils in einen OpenGL-Vertexbuffer. Die Funktion `void World::draw()` iteriert über die Vertexbuffer und startet für jedes Modell einen Durchlauf der Rendering-Pipeline.

Wie bereits erwähnt unterstützt das Framework ein Modulsystem. Die Klasse `Module` stellt eine Basisklasse hierfür dar. Module werden mit `void World::extend(Module)` in den Renderprozess eingefügt. `World` kommuniziert mit `Module` über bestimmte Funktionen. Durch Überschreiben der Funktion `Module::uniforms()`, die eine Liste von `Uniform`-Instanzen zurückgibt, können zum Beispiel Shadervariablen hinzugefügt werden. Es sind sowohl einfache Uniforms als auch Buffer und Shader-Storage-Blocks möglich. Die Klasse `World` fragt regelmäßig nach Änderungen und startet die entsprechenden Uniform-Updates durch Aufrufen der jeweiligen Callbackfunktion. Uniforms werden anhand ihres Bezeichners im Shader identifiziert und für jeden Shader gesetzt,

### 3. PRAKTISCHE UMSETZUNG

in dem sie definiert und benutzt werden. Die Klassen `Camera` und `Light` erben von `Module`, um die Kameramatrix und Daten der Lichtquellen als Uniforms an die GPU zu senden. Im Anhang der Arbeit befindet sich im Programmausdruck A.1 kommentierter Code, der den Einsatz der beschriebenen Klassen demonstriert, um eine Szene aus einer Wavefront/OBJ-Datei zu laden und zu rendern. In dieser Arbeit wurden die 3D-Modelle *Sibenik Cathedral* und *Chinese Dragon* [11] verwendet.

Die Klasse `Module` enthält neben der bereits erläuterten Funktion zum Hinzufügen von Shadervariablen noch weitere Funktionen, die überschrieben werden können. Diese ermöglichen es, zusätzliche vorgesetzte Renderdurchläufe (zum Beispiel zum Erstellen von Shadow Maps usw.) durchzuführen:

`int Module::num_passes()` gibt die Anzahl der zusätzlichen Renderdurchläufe für ein Modul zurück.

`GLuint Module::rendertarget(int pass)` gibt das OpenGL-Handle für einen Framebuffer zurück, in den das Ergebnis eines Durchlaufs geschrieben werden soll (Rendern in eine Textur ist durch Anhängen einer Textur an den Framebuffer über `glFramebufferTexture2D` möglich).

`void Module::on_pass(int pass)` wird vor einem Durchlauf aufgerufen.

`void Module::after_pass(int pass)` wird nach einem Durchlauf aufgerufen.

`GLuint Module::shader()` Gibt das OpenGL-Handle für einen Shader zurück, der für die zusätzlichen Durchläufe benutzt werden soll.

Abbildung 3.5 zeigt einen schematischen Überblick über den Rendervorgang.

Mit Post-Processing wird das nachträgliche Bearbeiten eines Bildes bezeichnet. Hinter der Realisierung von Post-Processing im Framework steht die Überlegung, dass die nachträgliche Bearbeitung eines Bildes der Szene eine Aufgabe der Kamera und nicht der Welt ist. Eine Objekt der Klasse `Camera` enthält genau eine Instanz von `Camera::PostProcessor`. Diese Instanz verwaltet einen separaten Shader. Ein Aufruf von `Camera::post_processor()` aktiviert Post-Processing und liefert einen Zeiger auf die Instanz zurück. Ist Post-Processing aktiviert, wird das nach einem vollständigen Renderdurchlauf, d.h. nach den Durchläufen aller Module sowie dem regulären Durchlauf, entstandene Bild in einer Textur gehalten und ein nachgeschalteter Renderdurchlauf durchgeführt. Der nachgeschaltete Renderdurchlauf nutzt den Shader von `Camera::PostProcessor`. Der Fragments shader erhält über Textureunit 0 wahlfreien Zugriff auf das Bild der Szene. Post-Processing wird im Fragments shader auf diesem Bild ausgeführt. Die Ausgabe des Fragments shaders wird verzerrungsfrei auf ein bildschirmfüllendes Rechteck projiziert.

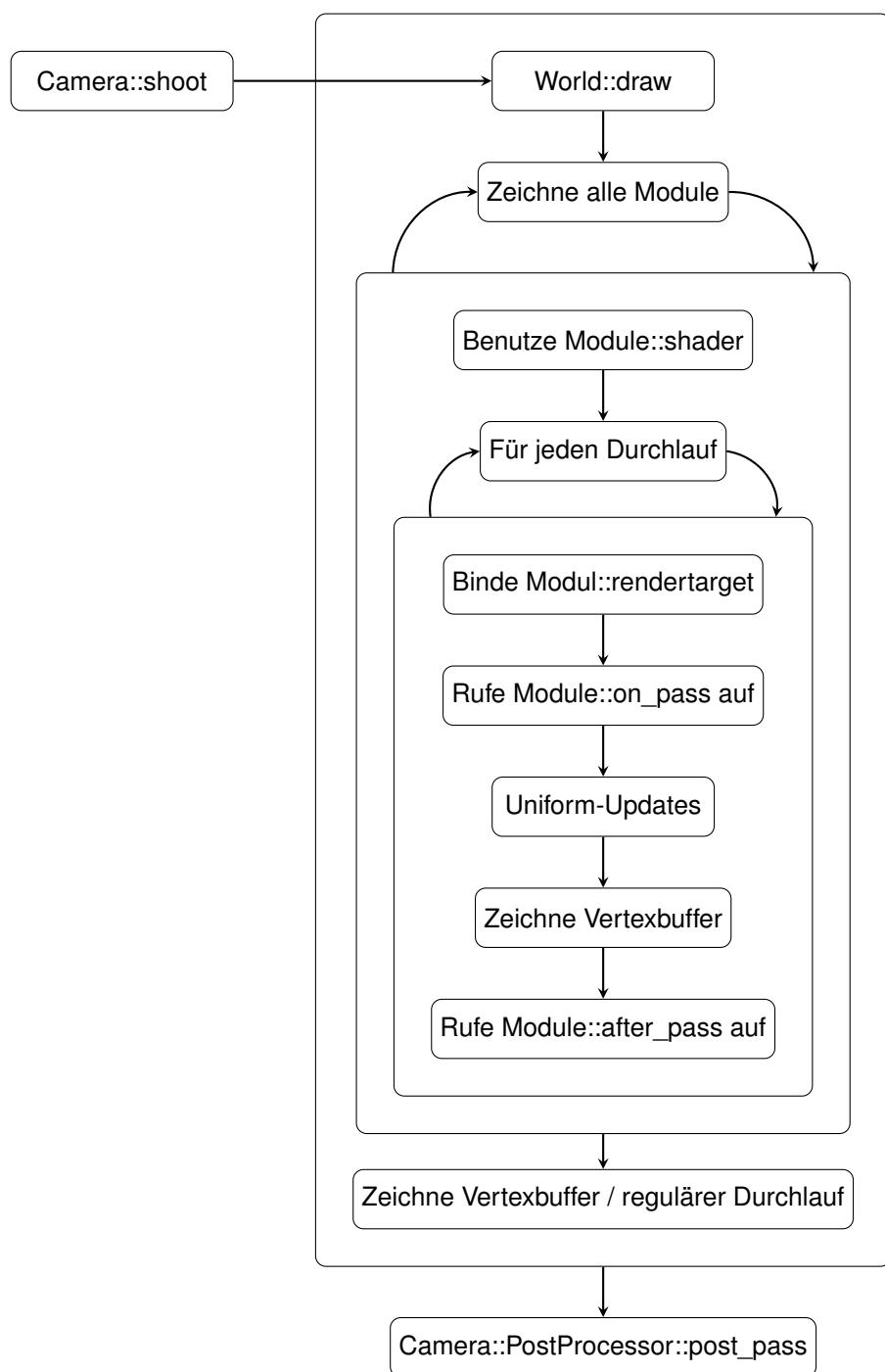


Abbildung 3.5.: Überblick über den Rendervorgang im Framework.

### 3. PRAKTISCHE UMSETZUNG

In den folgenden Abschnitten wird dargestellt, wie die in 3.1 und 3.2 betrachteten Verfahren im Framework implementiert wurden. Da der Fokus auf volumetrischer Lichtstreuung liegt, wird für Oberflächen ein sehr einfaches lokales Beleuchtungsmodell angewandt. Die Lichtquelle befindet sich in einem Punkt. Im Fragmentshader wird für jeden Oberflächenpunkt nur diffus reflektiertes Licht  $L_R$  nach dem Lambertschen Gesetz ausgewertet:

$$L_R = \begin{cases} R \cdot \cos(\theta) \cdot L, & \text{falls } \cos(\theta) > 0 \\ 0, & \text{sonst} \end{cases} \quad (3.4)$$

wobei  $R$  die Reflektanz der Oberfläche und  $L$  die Lichtfarbe als RGB-Vektoren sind.  $\theta$  ist der Winkel zwischen Normale  $\vec{n}$  und Lichtrichtung  $\vec{l}$  und es gilt  $\cos(\theta) \propto \text{dot}(\vec{n}, \vec{l})$ . Auf Texturierung der Objekte in den Testszenen wird verzichtet.

#### 3.3.2. Post-Processing von Mitchell

In diesem Abschnitt wird die Implementierung des bildbasierten Verfahrens von Mitchell [12] im Framework erläutert. Das Verfahren wurde als Modul implementiert. Es gibt eine Klasse, die von `Module` abgeleitet ist und den nötigen Zustand für das Verfahren als Attribute hält. Zum Zustand gehören ein Framebuffer mit angehängter Textur, ein Zeiger auf eine Instanz von `Camera::PostProcessor`, die Position der Lichtquelle und die zum Verfahren selbst gehörenden Parameter. Durch Überschreiben der entsprechenden Funktionen in `Module` dem Framework mitgeteilt, dass ein vorgeschalteter Renderdurchlauf benötigt wird und das Ergebnisbild in den eigenen Framebuffer geschrieben werden soll. Programmausdruck A.2 im Anhang zeigt den Aufbau des Moduls.

Es wird ein zusätzlicher Renderdurchlauf benötigt. Wie bereits erwähnt ist es im Bildraum nicht möglich genau zu ermitteln ob ein Punkt des Mediums im Schatten liegt oder nicht. Anhand des Verhältnisses von Samples auf verdeckenden Objekten zu emittierenden Regionen, wird die Verdeckung vom Algorithmus geschätzt; das setzt

- a) die Existenz einer emittierenden Region, deren Fläche ungleich Null ist, sowie
- b) die Kenntnis der Objekte, die die Lichtquelle verdecken könnten,

voraus. In der vorliegenden Implementierung wird, um a) zu behandeln, zwar eine Punktlichtquelle verwendet, die sich aber im Mittelpunkt eines Objekts befindet, das effektiv als emittierendes Objekt gesehen werden kann. Das Objekt wird in der Lichtfarbe gerendert, ist unschattiert und wirft keine Schatten auf andere Objekte. Außerdem wird der Himmel als emittierende Region angenommen. Um b) zu behandeln werden alle nicht-emittierenden Objekte als verdeckende Objekte gesehen. Verdeckende Objekte

müssen einen hohen Kontrast zur Farbe der emittierenden Region aufweisen, um die gewünschten volumetrischen Schatten zu erhalten. Im Allgemeinen weisen Objekte den nötigen Kontrast nicht auf. Daher wird ein vorgeschalteter Renderdurchlauf durchgeführt, bei dem alle verdeckenden Objekte einfarbig schwarz gerendert werden. Dieses Bild wird hier Verdeckungsbild bzw. im Code auf Englisch *occlusion image* genannt. Für den vorgeschalteten Renderdurchlauf wird der Framebuffer über die entsprechende Funktion der Klasse `Module` als Ziel gesetzt. In der vorliegenden Implementierung benötigt die angehängte Textur pro Texel RGB-Komponenten mit jeweils 8 Bit; es sind verschiedene Auflösungen möglich. Ein Verdeckungsbild wird in Abbildung 3.6(a) gezeigt.

Nach dem vorgeschalteten Renderdurchlauf wird von `World` der reguläre Renderdurchlauf durchgeführt, bei dem alle Objektoberflächen mit der Lambertschen Beleuchtung gerendert werden. Das in diesem Schritt entstandene Bild zeigt Abbildung 3.6(c). Dieses Bild wird ebenfalls in einer 8-Bit-RGB-Textur gespeichert. Danach wird `Camera::PostProcessor` aktiv. Diese Klasse rendert ein bildschirmfüllendes Rechteck mit Texturkoordinaten (0,0) in der linken unteren Ecke und (1,1) in der rechten oberen Ecke, sodass eine Textur verzerrungsfrei auf das Rechteck projiziert werden kann. Diese Technik ermöglicht die Manipulation beliebiger Texel in einem Fragmentshader, bevor ein Bild in den Bildschirm-Framebuffer geschrieben wird. Die Textur mit dem Bild des regulären Renderdurchlaufs wird von der Klasse `Camera::PostProcessor` an Textureunit 0 gebunden. Das Binden der Textur, die das Verdeckungsbild hält sowie der Parameter des Post-Processings wird vom Modul durchgeführt. Hierfür stellt die Klasse `Camera::PostProcessor`, die den Shader für Post-Processing verwaltet, entsprechende Funktionen bereit. Wie diese Funktionen in diesem Fall eingesetzt werden zeigt der Programmausdruck A.3 im Anhang. Es ist zu beachten, dass die Position der Lichtquelle von Weltkoordinaten in Texturkoordinaten transformiert werden muss.

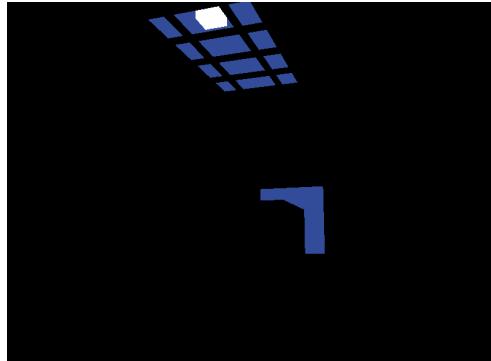
An dieser Stelle kann der eigentliche Algorithmus von Mitchell durchgeführt werden. Wie bereits in Abschnitt 3.1 erläutert wurde, beruht der Algorithmus auf einem Radial Blur-Filter. Der Filter muss auf das Verdeckungsbild angewandt werden. Somit tragen nur emittierende Regionen zur Summe aus dem Modell (siehe Gleichung 3.1) bei. Der Algorithmus wurde im Fragmentshader implementiert, der im nachgeschalteten Renderdurchlauf für Post-Processing benutzt wird. Der Fragmentshader erhält als Eingabe Texturkoordinaten *pixelcoords*; da diese über das bildschirmfüllende Rechteck interpoliert werden, entsprechen sie gerade den Koordinaten des aktuellen Pixels. Textureunit 0 enthält das Bild des regulären Renderdurchlaufs. Textureunit 1 enthält das Verdeckungsbild. Außerdem erhält der Fragmentshader über Uniforms Zugriff auf die Position der Lichtquelle in Texturkoordinaten *lightsource* sowie die Parameter des Verfahrens *n*, *density*, *decay* und *exposure* (siehe Gleichung 3.2). Der Algorithmus läuft wie folgt ab:

### 3. PRAKTISCHE UMSETZUNG

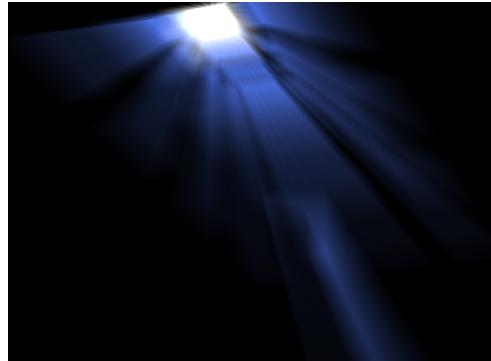
- Berechne einen vom aktuellen Pixel zur Position der Lichtquelle zeigenden Vektor  $s = pixelcoords - lightsource$ .
- Berechne die Schrittweite  $s = s / (n * density)$ .
- Lese die Farbe des aktuellen Pixels und speichere sie als Startwert in  $pixelcolor$ .
- Initialisiere den Extinktionsfaktor  $expDecay = 1$ .
- Speichere die aktuellen Pixelkoordinaten als Startwert in  $smplcoords$ .
- Für jeden Schritt 0 bis  $n$ :
  - Setze  $smplcoords = smplcoords - s$ .
  - Lese die Farbe des Pixels an der Stelle  $smplcoords$  und speichere sie in  $smplcolor$ .
  - Setze  $smplcolor = smplcolor * expDecay$ .
  - Setze  $pixelcolor = pixelcolor + smplcolor$ .
  - Setze  $expDecay = expDecay * decay$ .
- Setze  $pixelcolor* = exposure$ .
- Addiere  $pixelcolor$  auf die Farbe des Oberflächenpunktes an der Stelle des aktuellen Pixels.

Der Algorithmus ist in GLSL mit Kommentaren im Programmausdruck A.4 aufgeführt (siehe Anhang).

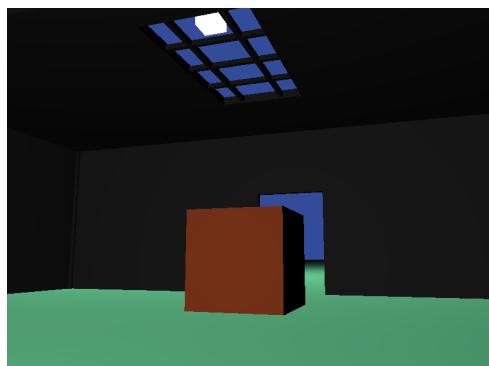
Es gibt 3 Parameter, die Abtastartefakte erzeugen können. Das ist zunächst die Anzahl Samples  $n$ . Die Rechenzeit pro Pixel wächst linear mit  $n$ . Ist  $n$  jedoch zu gering, tritt Aliasing auf. Um Aliasing auch bei nicht zu großen  $n$  gering zu halten, wird der Parameter  $density$  eingeführt. Eine Erhöhung von  $density$  verringert die Schrittweite und lässt die Lichtschäfte kürzer werden. Schließlich kann eine reduzierte Auflösung des Verdeckungsbildes gegenüber dem Bild aus dem regulären Renderdurchlauf nicht nur den Speicherbedarf verringern, sondern auch durch bilineare Texturfilterung Anti-Aliasing leisten. Die Auswirkungen der Parameter  $n$ ,  $density$  und der Auflösung des Verdeckungsbildes wird in der Evaluierung besprochen.



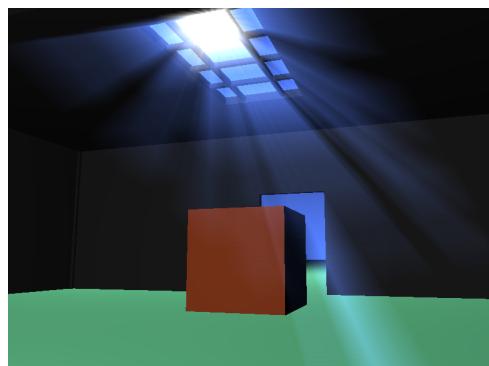
(a) Verdeckungsbild. Man sieht die schwarz gerenderten verdeckenden Objekte. Der weiße Würfel und die blaue Hintergrundfarbe ( $R = 0.2, G = 0.3, B = 0.6$ ) stellen die emittierende Region dar.



(b) Verdeckungsbild nach Post-Processing.



(c) Szene mit Lambertscher Beleuchtung der Oberflächen. Es gibt eine zweite Lichtquelle hinter der Kamera, die bei der volumetrischen Beleuchtung ignoriert wird.



(d) Szene nach Addition mit dem Verdeckungsbild.

Abbildung 3.6.: Ablauf des Verfahrens von Mitchell. Parameter:  $n = 100$ ,  $exposure = 0.01921$ ,  $density = 1.4$ ,  $decay = 1.0$ .

### 3. PRAKTISCHE UMSETZUNG

#### 3.3.3. Post-Processing von Sousa

In diesem Abschnitt wird die Implementierung des bildbasierten Verfahrens von Sousa [17] im Framework erläutert. Da dieses Verfahren einen Filter mehrmals hintereinander anwendet, sind mehr Zustandsvariablen als beim Verfahren von Mitchell nötig. Auch für das Verfahren von Sousa gibt es eine Klasse, die von `Module` abgeleitet wird. Die Attribute dieser Klasse umfassen drei Zeiger auf Instanzen der Klasse `Camera::PostProcessor`, zwei Framebuffer mit angehängter Textur, sowie die Position der Lichtquelle und die Parameter des Verfahrens selbst. Im Programmausdruck A.5 ist der Aufbau des Moduls mit erklärenden Kommentaren aufgeführt (siehe Anhang).

In der vorliegenden Implementierung werden insgesamt fünf Renderdurchläufe, davon ein Durchlauf zum Erstellen einer Tiefenkarte, drei Durchläufe mit Post-Processing und ein regulärer Durchlauf durchgeführt. Das Post-Processing wird auf einer Tiefenkarte durchgeführt, die aus Sicht der Kamera gerendert wird. Somit entfällt ein Problem, das beim Verfahren von Mitchell (siehe vorheriger Abschnitt 3.3.2) aufgetreten ist: Die Aufteilung der Szene in emittierende und verdeckende Objekte. Durch Verwenden der Tiefenkarte wird einfach angenommen, dass sich emittierende Teile der Szene weit im Hintergrund befinden, was für den Himmel bzw. die Sonne eine praktikable Annahme ist. Tatsächlich wurde der God-Ray-Effekt von Sousa in der CryEngine nur bei Sichtbarkeit der Sonne im Bildraum benutzt. Ein Szenenaufbau wie in Abbildung 3.8 und 3.15, bei dem die Lichtquelle eine viel geringere Tiefe als der Himmel hat, ist ein Beispiel, bei dem die Verwendung der Tiefenkarte einen Nachteil darstellt. Die Lichtschäfte werden zur Lichtquelle hin dunkler, was der natürlichen Erwartung widerspricht.

Um die Tiefenkarte zu erstellen, wird ein vorgeschalteter Renderdurchlauf durchgeführt. Es ist zu beachten, dass die Tiefenwerte wegen der perspektivischen Projektion nicht linear verteilt sind. Daher wird im vorgesetzten Renderdurchlauf ein Fragments-hader verwendet, der die Tiefenwerte in ein lineares Koordinatensystem transformiert (siehe Programmausdruck A.6 im Anhang). Die Tiefenkarte wird in eine Textur geschrieben, die einen Tiefenwert mit 8 Bit pro Texel hält. Während des gesamten Verfahrens werden drei Tiefenkarten mit jeweils 8 Bit pro Texel und eine RGB-Textur mit 8 Bit pro Kanal benötigt. Abbildung 3.7 zeigt einen schematischen Überblick über das Verfahren.

Die Anwendung des Radial Blur-Filters unterscheidet sich im Verfahren von Sousa zum Verfahren von Mitchell. Statt einer Aufteilung der Strecke vom Pixel zur Lichtquelle in eine feste Anzahl Samples  $n$ , wird eine feste Schrittweite verwendet und die Abtastung nach einer maximalen Anzahl Samples abgebrochen. Folglich wird nicht notwendigerweise für jeden Pixel die gesamte Strecke abgetastet. Ein weiterer Unterschied zum Verfahren von Mitchell ist die Monochromazität im Verfahren von Sousa, da statt dem Verdeckungsbild eine Tiefenkarte verwendet wird. Die Eingaben für den Algorithmus von Sousa sind neben der Tiefenkarte die maximale Anzahl Samples, die mit *TAPS* bezeichnet wird, die Koordinaten des aktuellen Pixels *pixelcoords*, die

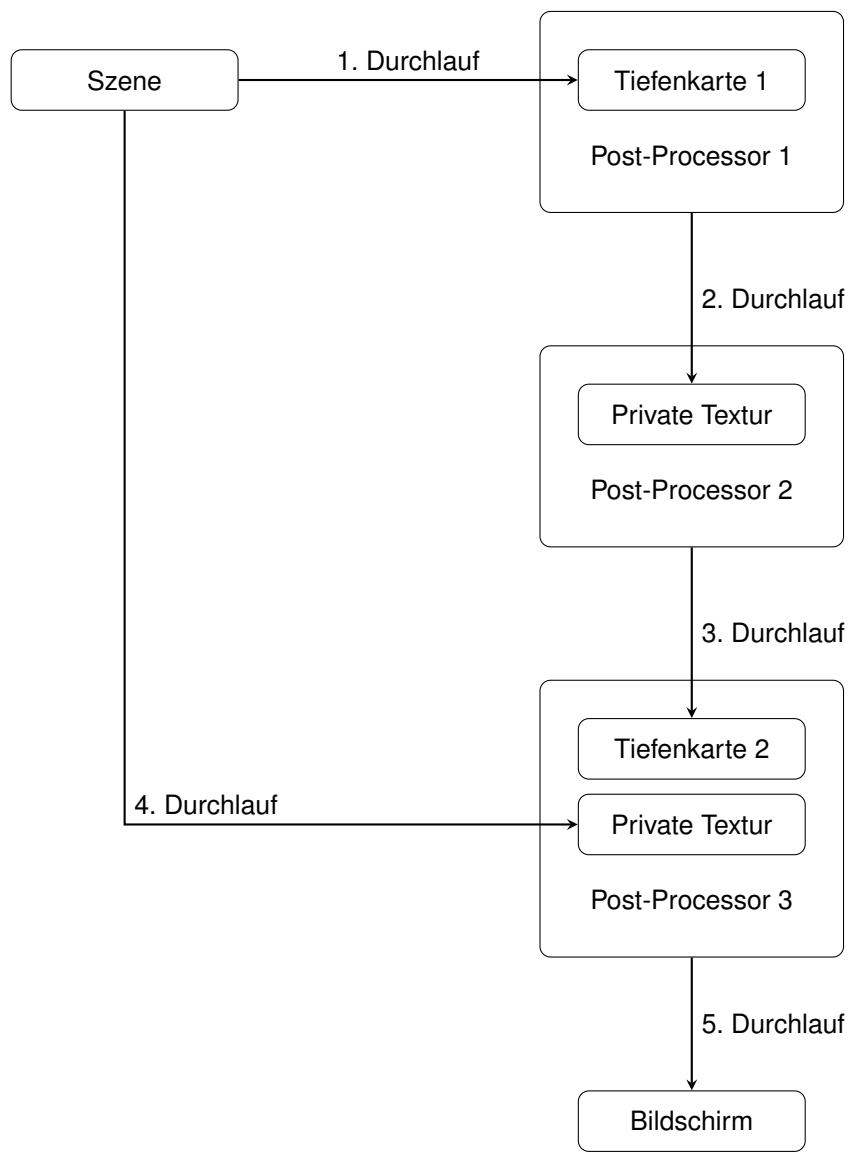


Abbildung 3.7.: Überblick über das Verfahren von Sousa. Im 1. Durchlauf wird die Tiefenkarte erstellt. Im 2. und 3. Durchlauf wird der Radial Blur-Filter angewandt. Im 4. Durchlauf werden die Oberflächen der Szene gerendert. Im 5. Durchlauf wird der Radial Blur-Filter erneut angewandt und das Ergebnis mit dem Bild des 4. Durchlaufs kombiniert. Die Benennung der Elemente entspricht den Variablennamen im Modul (siehe Programm ausdruck A.5), wobei die privaten Texturen keine Attribute des Moduls, sondern der jeweiligen Instanz von `Camera::PostProcessor` sind.

### 3. PRAKTISCHE UMSETZUNG

Koordinaten der Lichtquelle *lightsource*, die Schrittweite  $s$  und der Parameter *brightness*. Der Algorithmus läuft wie folgt ab:

- Berechne einen vom aktuellen Pixel zur Position der Lichtquelle zeigenden Vektor  $v = \text{pixelcoords} - \text{lightsource}$ .
- Speichere die Länge des Vektors in  $l$ .
- Berechne einen Vektor mit der Richtung von  $v$  und der Länge  $s$ :  $\text{step} = s * v / l$ .
- Initialisiere den Startwert für die Akkumulation der Tiefenwerte:  $\text{sum} = 0$ .
- Speichere die aktuellen Pixelkoordinaten als Startwert in  $p$ .
- Für jeden Schritt 0 bis  $TAPS$  und solange die Strecke zwischen Pixel und Lichtquelle nicht verlassen wurde:
  - Lese die Tiefekarte an den Koordinaten  $p$  und speichere die Tiefe in  $\text{smpldepth}$ .
  - Setze  $\text{sum} = \text{sum} + \text{smpldepth}$ .
  - Setze  $p = p + \text{step}$ .
- Setze  $\text{sum} *= \text{brightness}$ .
- Schreibe  $\text{sum}$  in die Tiefenkarte für den nächsten Durchgang oder addiere  $\text{sum}$  auf die Farbe des aktuellen Oberflächenpunktes, wenn alle Durchgänge ausgeführt wurden.

Der Algorithmus ist in GLSL mit Kommentaren im Programmausdruck A.7 aufgeführt (siehe Anhang). Der Parameter *brightness* für die Gesamthelligkeit liegt im Intervall  $[0, 1]$  und wird für jeden Pixel nach der Abtastung mit der akkumulierten Helligkeit multipliziert, um eine Überbelichtung zu vermeiden. Eine Anpassung der Gesamthelligkeit zur Laufzeit kann nötig werden, da sich die Tiefenkarte bei dynamischer Kamera ändert. Abbildung 3.8 zeigt alle Zwischenschritte im Verfahren von Sousa mit entsprechenden Bildern.

Wie bereits in Abschnitt 3.1 angesprochen, wird im Verfahren von Sousa verzahnte Abtastung verwendet. Die Schrittweite wird in jedem Durchgang, in dem der Radial Blur-Filter angewandt wird, geändert. In der vorliegenden Implementierung wird also mit 3 verschiedenen Schrittweiten  $s_1, s_2$  und  $s_3$  abgetastet. Es trägt erheblich zur Bildqualität bei, wenn die Schrittweiten der Folge  $s_1 = TAPS^{-1}, s_2 = TAPS^{-2}, s_3 = TAPS^{-3}$  entsprechen. Verhalten sich die Schrittweiten nicht in dieser Relation zueinander, treten Abtastartefakte auf, wie in der Evaluierung in Abbildung 3.15 gezeigt wird.

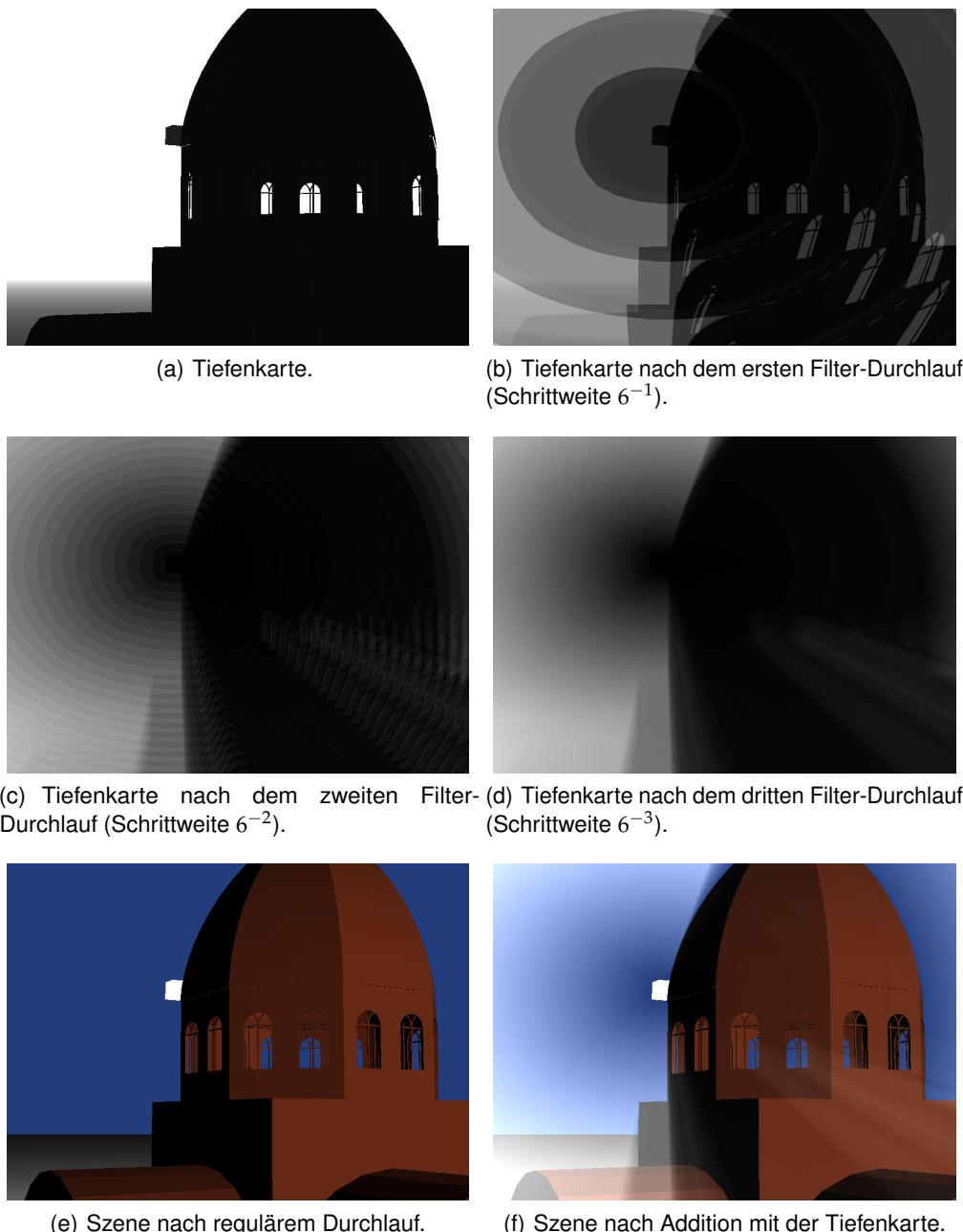


Abbildung 3.8.: Ablauf des Verfahrens von Sousa bei 3 Durchgängen.  $TAPS = 6$ , Schrittweiten:  $s_1 = 6^{-1}$ ,  $s_2 = 6^{-2}$ ,  $s_3 = 6^{-3}$ . Helligkeit: 0.19.

### 3. PRAKTISCHE UMSETZUNG

#### 3.3.4. Ray-Marching von Tóth und Umenhoffer

In diesem Abschnitt wird die Implementierung des Verfahrens von Tóth und Umenhoffer [18] im Framework erklärt. Im Unterschied zu den Verfahren von Mitchell und Sousa benötigt das Verfahren von Tóth und Umenhoffer kein Post-Processing. Nachdem in einem vorgesetzten Renderdurchlauf eine Tiefenkarte aus Sicht der Lichtquelle erzeugt wurde, kann der Prozess des Ray-Marching komplett im regulären Renderdurchlauf durchgeführt werden.

Das Verfahren greift auf Shadow Mapping zurück, welches im Framework als eigenständiges Modul implementiert wurde, sodass es auch mit anderen Verfahren kombiniert werden kann. Der Programmausdruck A.8 im Anhang zeigt den Aufbau des Moduls. Im Folgenden wird genauer auf die Realisierung von Shadow Mapping eingegangen.

Für Shadow Mapping muss ein vorgesetzter Renderdurchlauf durchgeführt werden. Dieser nutzt einen Vertexshader, der die Szene in ein Koordinatensystem transformiert, bei dem die Lichtquelle im Ursprung ist und das im Folgenden auch Lichtraum genannt wird. Genau wie bei einer Kamera wird der Lichtraum durch einen Sichtkegel in Weltkoordinaten definiert, der ein Seitenverhältnis, Öffnungswinkel sowie eine vordere und hintere Clippingebene hat. Diese Eigenschaften und die Ausrichtung des Sichtkegels müssen so gewählt werden, dass alle schattenwerfenden und schattenempfangenden Objekte im darin liegen. Für die Transformation in den Lichtraum wird eine Matrix erzeugt, die im folgenden Lichtraummatrix genannt wird. Im Vertexshader wird die Lichtraummatrix an die Vertexposition multipliziert. Der Fragmentshader schreibt nur einen Tiefenwert und keine Farbe. Als Ziel des vorgesetzten Renderdurchlaufs dient eine Textur mit 32 Bit pro Texel, die die sogenannte Shadow Map hält. Die Auflösung der Shadow Map bestimmt ist in der vorliegenden Implementierung  $1024 \times 1024$  Texel. Es wurde die einfachste Form von Shadow Mapping umgesetzt, bei der nur eine Lichtquelle unterstützt und für die Lichtquelle wird nur eine Shadow Map benutzt wird. Eine Filterung der Shadow Mapping mittels PCF (siehe Abschnitt 2.2.3) kann die Qualität verbessern, aber vervielfacht die Renderzeit, was in der Evaluierung in Abbildung 3.18 gezeigt wird. Für die Implementierung des Verfahrens von Tóth und Umenhoffer wurden Optimierungen von Shadow Mapping als nicht lohnenswert erachtet, da Probleme bei volumetrischen Schatten weit weniger sichtbar sind, als bei Schatten auf Oberflächen.

Für den regulären Renderdurchlauf wird die Shadow Map an Textureunit 0 gebunden und die Lichtraummatrix als Uniform gesetzt. Im Vertexshader werden die Weltkoordinaten in den Lichtraum transformiert und an den Fragmentshader weitergegeben (siehe Programmausdruck A.9 im Anhang). Im Fragmentshader gibt es eine Funktion für den Schattentest, die einen Punkt im Lichtraum erwartet und einen Booleschen Wert zurückgibt. Der Punkt im Lichtraum durchläuft die Transformation in normalisierte Gerätekoordinaten und anschließend in Texturkoordinaten, bevor dessen Tiefenwert mit dem Tiefenwert in der Shadow Map verglichen wird. Die Funktion für den Schattentest, wie

sie im Fragmentshader implementiert ist, wird im Programmausdruck A.10 im Anhang gezeigt. Im Folgenden wird erklärt, wie der Schattentest während des Ray-Marching Anwendung findet.

Das Verfahren von Tóth und Umenhoffer führt Ray-Marching im Lichtraum durch. Das hat die Vorteile, dass die Position der Lichtquelle implizit durch den Nullvektor gegeben ist und dass vor dem Schattentest keine Transformation mehr durchgeführt werden muss. Der Algorithmus benötigt die Kameraposition  $camPos$  und die interpolierte Vertexposition  $fragPos$  jeweils im Lichtraum. Außerdem werden die Parameter des Verfahrens selbst benötigt: die Anzahl Samples  $n$ , die Leistung der Lichtquelle  $P$  sowie die Albedo  $\alpha$  und Dichte  $\rho$  des Mediums. Mit  $L$  wird die Farbe eines Oberflächenpunktes bezeichnet. Um die Veränderung von  $L$  durch einfache Streuung auf dem Sichtstrahl auszuwerten, führt der Algorithmus von Thoth folgende Schritte aus:

- Berechne den in Richtung der Kamera zeigenden Sichtvektor im Lichtraum:  $viewDir = fragPos - camPos$ .
- Speichere die Länge des Sichtvektors in  $s$ .
- Normalisiere den Sichtvektor.
- Berechne die Schrittweite  $ds = s/n$ .
- Berechne die Extinktion des vom Oberflächenpunkt reflektierten Lichts auf dem Weg zur Kamera:  $L = L * e^{-s*\rho}$ .
- Initialisiere die Position für die Abtastung entlang des Sichtvektors mit der Position des Oberflächenpunktes:  $x = fragPos$ .
- Solange  $s \geq 0$ :
  - Setze  $s = s - ds$ .
  - Setze  $x = x + viewDir * ds$ .
  - Speichere das Ergebnis des Schattentests für  $x$  in  $S$ .
  - Speichere die Distanz von  $x$  zur Lichtquelle in  $d$  (im Lichtraum ist dies einfach die Länge des Vektors  $x$ ).
  - Berechne die Extinktion des direkten Lichts von der Lichtquelle auf dem Weg zum Punkt  $x$  und speichere diese in  $ext_1 = e^{-d*\rho}$ .
  - Berechne die Extinktion des eingestreuten Lichts vom Punkt  $x$  zur Kamera und speichere diese in  $ext_2 = e^{-s*\rho}$ .
  - Berechne die vom Medium unabhängige Reduzierung der Leistung  $P$  auf der Distanz  $d$  und speichere die verbliebene Leistung in  $P_r = P/(4 * \pi * d^2)$ .

### 3. PRAKTISCHE UMSETZUNG

- Berechne das Ergebnis der Streuphasenfunktion für den Winkel zwischen  $x$  und  $viewDir$  und speichere es in  $\Phi$ .
- Setze  $L = L + S * ext_1 * ext_2 * P_r * \Phi * \rho * \alpha$ .

Der Umsetzung des Algorithmus in GLSL ist mit Kommentaren im Programmausdruck A.12 im Anhang aufgeführt. Das Ergebnis des Schattentests wird als Sichtbarkeit der Lichtquelle im Wertebereich  $[0, 1]$  interpretiert, wobei ein Punkt im Schatten die Sichtbarkeit 0 und ein beleuchteter Punkt die Sichtbarkeit 1 hat. Somit lassen sich auch nicht-binäre Schattentests leicht einbinden. Die Streuphasenfunktion nimmt ebenfalls Werte im Intervall  $[0, 1]$  an, da sie die Wahrscheinlichkeit für Streuung in die Richtung der Kamera angibt. Hier können beispielsweise die Phasenfunktionen für Rayleigh- bzw. Mie-Partikel (siehe Abbildung 2.8) verwendet werden, um Luft bzw. Nebel zu simulieren. Eine entsprechende Implementierung der Funktionen zeigt der Programmausdruck A.11 im Anhang.

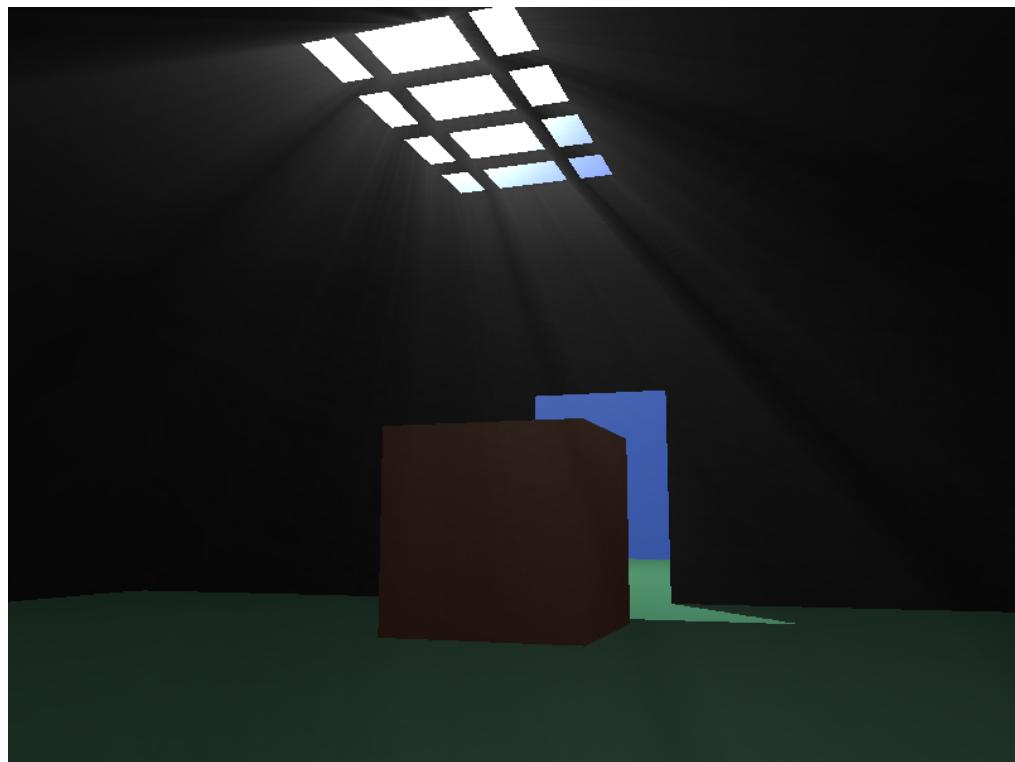


Abbildung 3.9.: Ein Ergebnis des Verfahrens von Tóth und Umenhoffer. Parameter:  $n = 100$ ,  $\rho = 0.01$ ,  $\alpha = 0.9$ ,  $P = 9000$ . Es wurde die Phasenfunktion für Rayleigh-Streuung verwendet.

Es werden für jeden Pixel die Koordinaten des sichtbaren Oberflächenpunktes benötigt. Daraus folgt, dass es keinen Pixel geben darf, an dem keine Oberfläche sichtbar ist. Es ist daher nicht möglich, wie im Verfahren von Mitchell eine Hintergrundfarbe zu verwenden. Stattdessen wird eine spezielle Geometrie zur Begrenzung der Szene benötigt, z.B. ein sogenannter Skydome oder eine Skybox. Diese Geometrie muss bei der Beleuchtungsberechnung gesondert behandelt werden. Die Szenen in den Abbildungen 3.18, 3.19 und 3.9 verwenden eine Skybox mit konstanter blauer Farbe  $RGB(0.278, 0.472, 0.976)$ . Die Farbe wurde so gewählt, um dem Verhältnis der  $RGB$ -Werte des in Abschnitt 2.2.2 erwähnten Streukoeffizienten für Luft in der Erdatmosphäre zu entsprechen.

Die Implementierung in einem einzigen Renderdurchlauf bringt den Nachteil mit sich, dass das gesamte Ray-Marching unter Umständen auch für Fragmente durchgeführt wird, die in der Ausgabe ohnehin nicht sichtbar sind. Vor allem bei Szenen mit einer hohen Tiefenkomplexität kann daher sogenanntes Differed Shading einen Vorteil bei der Laufzeit bringen. Bei Differed Shading wird die Verdeckung von Fragmenten in einem Geometrie-Durchlauf aufgelöst. Die für die Beleuchtung benötigten Parameter werden in einen sogenannten Geometry-Buffer geschrieben. Die Beleuchtung selbst wird in einem Post-Processing-Durchlauf berechnet. Der Differed Shading-Ansatz ist in der vorliegenden Implementierung nicht enthalten.

## 3.4. Evaluierung

In diesem Abschnitt werden die implementierten Verfahren im Hinblick auf Plausibilität und Fehler evaluiert. In den Verfahren liegen zwei grundsätzlich verschiedene Orientierungen vor. Auf der einen Seite stehen die bildbasierten Verfahren von Kenny Mitchell und Tiago Sousa, die eher eine künstlerisch-ästhetische Orientierung aufweisen. Im Falle des Verfahrens von Sousa ist dies klar so, da es als Teil einer Game-Engine entwickelt wurde. Auf der anderen Seite steht das Ray-Marching Verfahren von Tóth und Umenhoffer, das eine physikalische Orientierung aufweist. Ein direkter Vergleich ist insofern nicht möglich, dass man nicht dieselbe Parameterkonfiguration wählen und die Ergebnisbilder vergleichen kann. Die Parameter haben je nach Orientierung des Verfahrens eine grundsätzlich andere Bedeutung. Im Folgenden werden daher anhand von Screenshots die qualitativen Eigenschaften von Ergebnisbildern verglichen und ein Vergleich der Laufzeit durchgeführt (Abschnitt 3.4.1). Im Abschnitt 3.4.2 werden die Grenzen der implementierten Echtzeitverfahren anhand von Bildern des physikalisch-basierten Offlinerenderers Mitsuba [8] aufgezeigt.

### 3. PRAKTISCHE UMSETZUNG

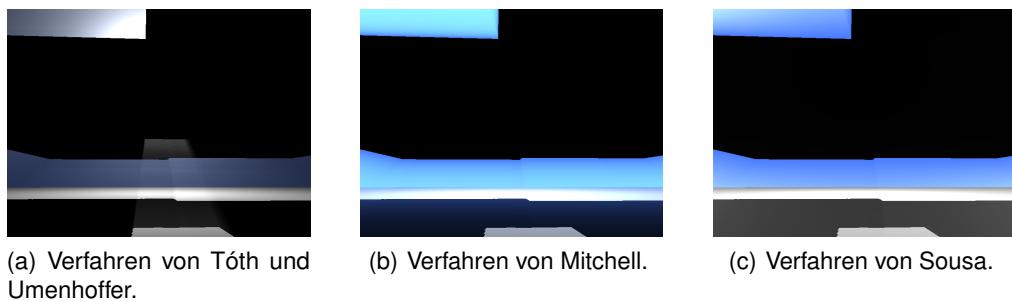
#### 3.4.1. Vergleich der implementierten Verfahren

Die Verfahren von Mitchell und Sousa sind vom physikalischen Standpunkt aus gesehen wenig akkurat. Sie legitimieren sich jedoch durch das visuell ansprechende Ergebnis im Verhältnis zum geringen Rechenaufwand, welcher unabhängig von der Szenenkomplexität ist. Beide Verfahren erzielen einen plausiblen Effekt, wenn die Kamera auf den Himmel gerichtet ist. Die Plausibilität geht in bestimmten Fällen verloren, da im Bildraum schlicht nicht genügend Informationen über die Szene zur Verfügung stehen. Ein solcher Fall ist gegeben, wenn die Lichtstrahlen auf Hindernisse treffen, da sie von diesen nicht korrekt blockiert werden (siehe Abbildung 3.10). Ein anderer Fall ist bei Geometrien mit im Bildraum nicht sichtbaren Löchern gegeben (siehe Abbildung 3.11). Schließlich kann der Fall auftreten, bei dem sich die Lichtquelle im Vordergrund eines Objekts befindet. Die bildbasierten Verfahren nehmen an, die Lichtquelle bzw. die emittierende Region, die meist durch die Sonne bzw. den Himmel dargestellt wird, befindet sich weit im Hintergrund. In Abbildung 3.12 wurde eine Szene konstruiert, bei der diese Annahme nicht zutrifft.



Abbildung 3.10.: Die Verfahren Mitchell, Sousa und Tóth und Umenhoffer im Vergleich bei einer Szene, in der sich ein Hindernis im Weg der Lichtstrahlen befindet.

Ein Aspekt, den alle Verfahren gemeinsam haben, ist die Abtastung eines Strahls und Akkumulation von Lichtintensitäten an jedem Pixel des Bildes. In der Theorie dient dieses Vorgehen der Annäherung eines infinitesimalen Integrals mittels einer finiten Summe. Da in der Praxis keine unendlich kleinen Schritte möglich sind, kommt es zu Abtastartefakten. Dies sind Bildfehler in Form von harten Übergängen an Stellen, an welchen ein Beobachter aus Erfahrungen mit der realen Welt keine solchen Abstufungen erwartet. Natürlich trägt eine wachsende Anzahl Samples  $n$  in jedem Fall zur Reduzierung von Abtastartefakten bei. Allerdings ist es unvermeidlich, dass mit  $n$  auch die Laufzeit eines Algorithmus wächst. In den implementierten Verfahren wurden

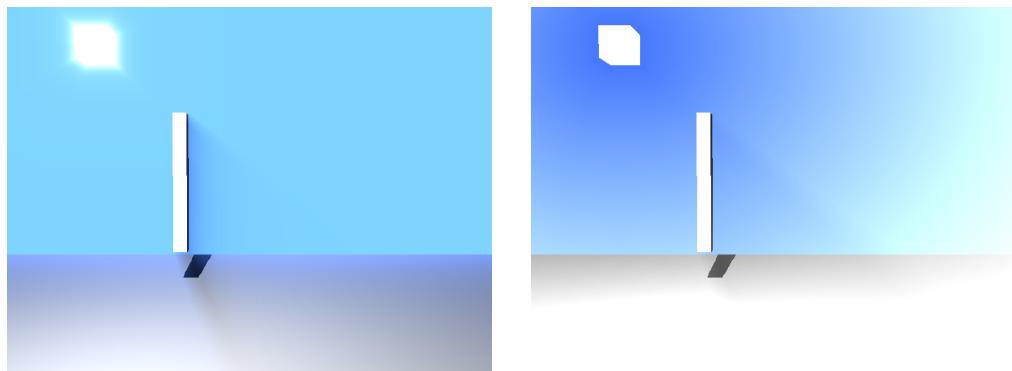


(a) Verfahren von Tóth und Umenhoffer.

(b) Verfahren von Mitchell.

(c) Verfahren von Sousa.

Abbildung 3.11.: Geometrie mit einem Loch, das im Bildraum aus der gezeigten Perspektive nicht sichtbar ist. Die bildbasierten Verfahren von Mitchell und Sousa stellen den einfallenden Lichtschaft nicht dar, das Ray-Marching-Verfahren von Tóth und Umenhoffer dagegen schon.



(a) Verfahren von Mitchell.

(b) Verfahren von Sousa.

Abbildung 3.12.: Szene mit Lichtquelle im Vordergrund. Die Verfahren von Mitchell und Sousa machen die globale Annahme, die Lichtquelle sei immer im Hintergrund. Der volumetrische Schatten geht in die falsche Richtung.

weitere Ansätze vorgeschlagen, um die Sichtbarkeit der Bildfehler, die durch zu niedrige Abtastrate entstehen, zu verringern. Im Verfahren von Mitchell gibt es zwei Ansätze. Zum einen ist dies die Einführung des Parameters *density*, mit dem die Schrittweite verkleinert werden kann. Zum anderen ist dies eine Reduzierung der Auflösung des Verdeckungsbildes, auf dem die Abtastung durchgeführt wird, und anschließende bilineare Filterung. Eine Erhöhung von *density* bewirkt, dass zwar nur ein Teil der Strecke von einem Pixel zur Lichtquelle einbezogen, aber dafür dichter abgetastet wird. Die Folge davon ist ein schnelleres Abfallen der Lichtschäfte, was eine Änderung ist, die eigentlich durch eine Änderung des Absorptionsverhaltens des partizipierenden Mediums verursacht werden müsste. Der Ansatz kann also eventuell zu einer Einschränkung

### 3. PRAKTISCHE UMSETZUNG

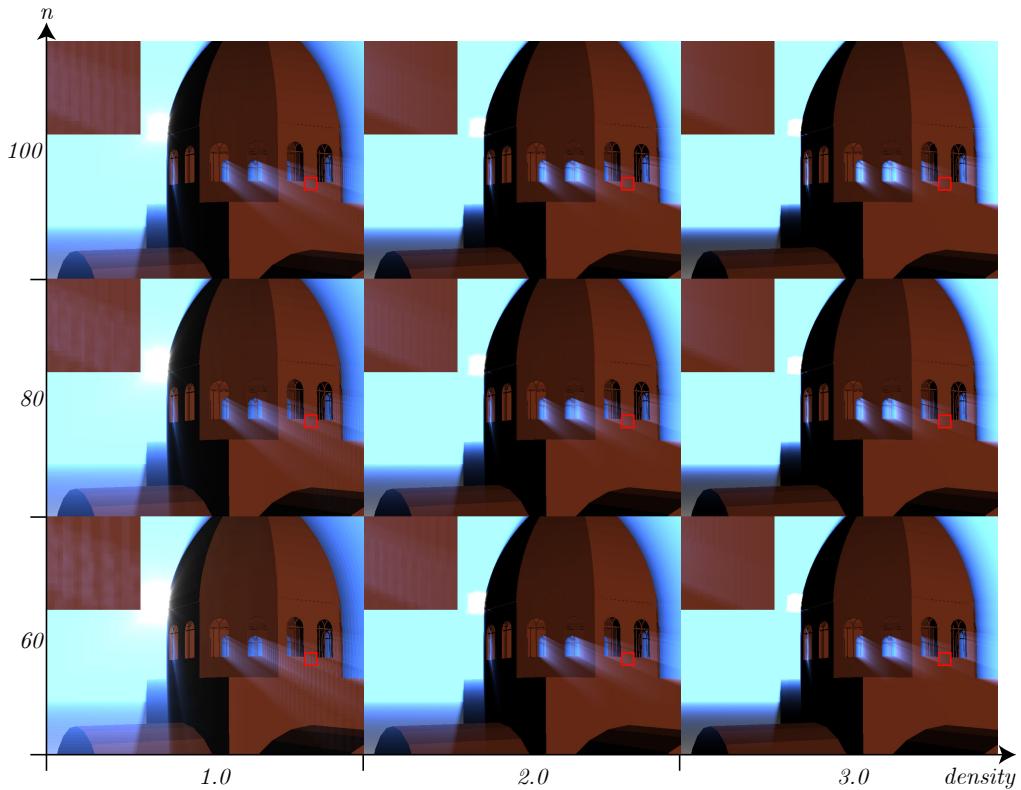


Abbildung 3.13.: Reduzierung von Abtastartefakten im Verfahren von Mitchell durch Erhöhung der Parameter  $n$  bzw.  $density$ . Auflösung des Verdeckungsbildes:  $800 \times 600$  Texel (entspricht der Fenstergröße).

der Plausibilität des Effekts führen. Der Ansatz, der eine bilineare Filterung des abgetasteten Bildes beinhaltet bewirkt letztlich, dass stets auch die Umgebung eines Pixels zur Abtastung mit beiträgt. Dieser Ansatz wirkt harten Übergängen entgegen, ohne andere sichtbare Veränderungen wie die Länge der Lichtschäfte nach sich zu ziehen. Abbildung 3.13 zeigt die Auswirkungen von  $density$  und Anzahl Samples  $n$  auf die Bildqualität bei der Fenstergröße entsprechender Auflösung des Verdeckungsbildes. In Abbildung 3.14 wurde die Auflösung auf  $1/4$  der Fenstergröße reduziert und bilineare Filterung aktiviert.

Im Verfahren von Sousa wird durch verzahnte Abtastung dem Hervortreten von harten Übergängen entgegengewirkt. Die harten Übergänge bestehen zwar noch, sind aber unregelmäßig verteilt und somit für den Menschen weniger auffällig. Bei der verzahnten Abtastung wird mehrmals mit unterschiedlichen Schrittweiten abgetastet. Es wurde beobachtet, dass im Verfahren von Sousa eine gewisse Abhängigkeit der Schrittweiten besteht (siehe Abbildung 3.15). Außerdem vermeidet das Verfahren von Sousa im

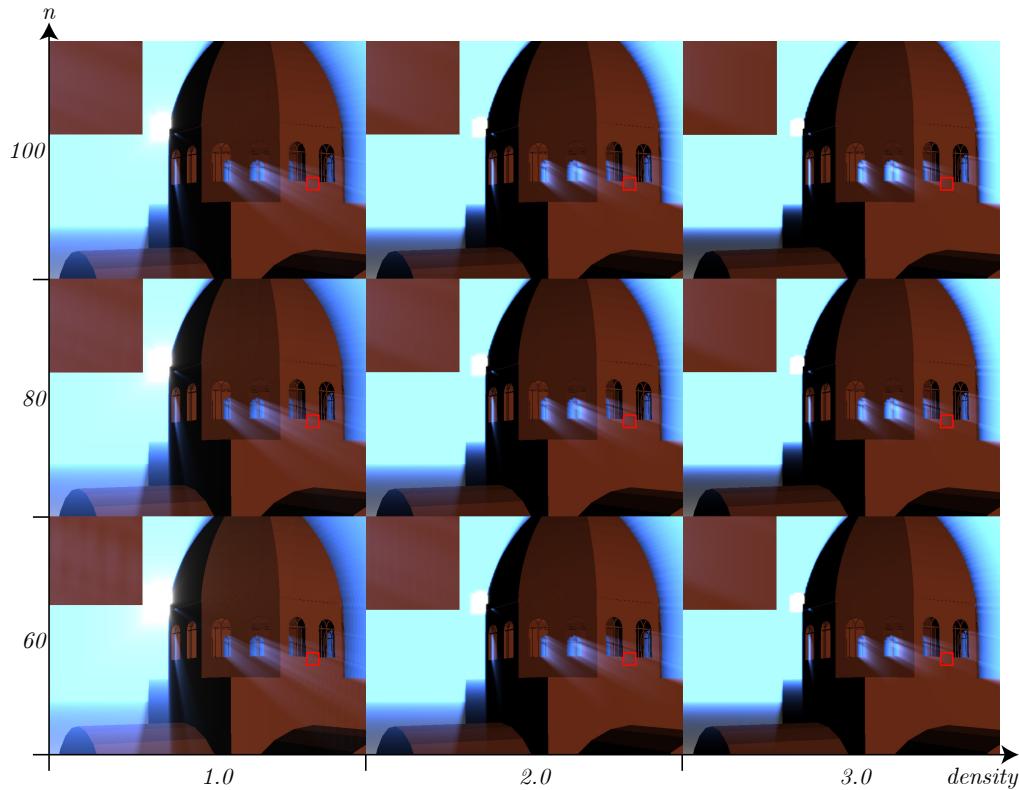


Abbildung 3.14.: Reduzierung von Abtastartefakten im Verfahren von Mitchell durch Erhöhung der Parameter  $n$  bzw.  $density$ . Auflösung des Verdeckungsbildes:  $200 \times 150$  Texel (entspricht  $1/4$  der Fenstergröße). Durch bilineare Filterung werden Abtastartefakte zusätzlich reduziert.

Gegensatz zu den Verfahren von Mitchell und Tóth und Umenhoffer eine unnötig hohe Abtastrate für Pixel in der Nähe der Lichtquelle, indem über das Bild hinweg eine konstante Schrittweite verwendet wird. Insgesamt werden vom Verfahren von Sousa am wenigsten Abtastartefakte erzeugt, was im Vergleich in Abbildung 3.17 beobachtbar ist. Im Verfahren von Sousa werden nur 18 Samples bzw.  $4.34ms$  Renderzeit für ein Bild benötigt, für das im Verfahren von Mitchell 60 Samples bzw.  $7.54ms$  und im Verfahren von Tóth und Umenhoffer 60 Samples bzw.  $28.33ms$  benötigt werden. Die Optimierung der Verfahren von Mitchell und Tóth und Umenhoffer mithilfe der angesprochenen Vorteile des Verfahrens von Sousa ist theoretisch möglich.

Ein Vergleich der Laufzeiten der umgesetzten Verfahren in Abhängigkeit der Anzahl Samples wird in Abbildung 3.16 gezeigt. Es ist zu beachten, dass die Renderzeit auf dem integrierten Grafikprozessor *Intel HD Graphics 4600* gemessen wurden und daher

### 3. PRAKTISCHE UMSETZUNG

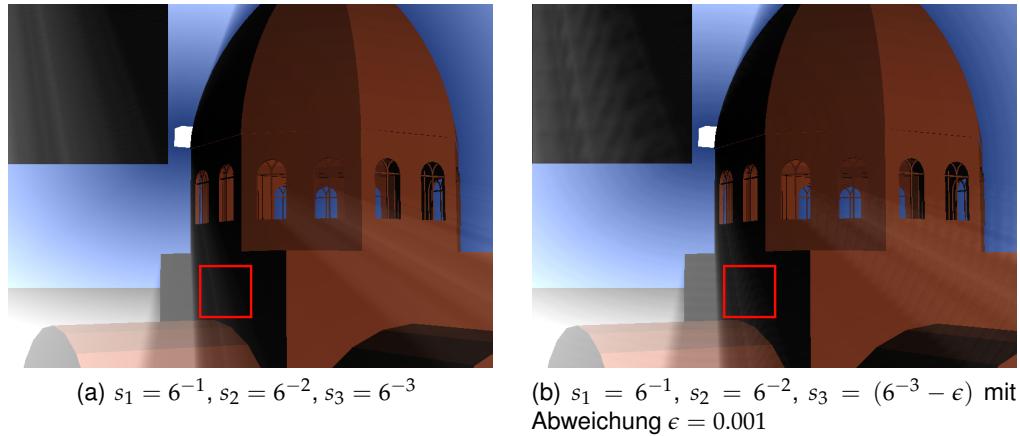


Abbildung 3.15.: Verfahren von Sousa: Verhalten sich die Schrittweiten nicht entsprechend der Folge  $s_1 = TAPS^{-1}, s_2 = TAPS^{-2}, s_3 = TAPS^{-3}$ , treten Abtastartefakte auf.  $TAPS$  bezeichnet die Anzahl der Samples, die für einen Pixel auf der Strecke zur Lichtquelle genommen werden, sofern die Strecke nicht verlassen wird. Details zu den Parametern finden sich in Abschnitt 3.3.3.

teilweise nur annähernd Echtzeit erreichen. Auf dedizierten Grafikprozessoren sind kürzere Renderzeiten zu erwarten.

Das Verfahren von Tóth und Umenhoffer greift auf Shadow Mapping zurück. Daher lassen sich Optimierungen für das Shadow Mapping auch in diesem Verfahren einsetzen. Für Abbildung 3.18 wurde der Einfluss von PCF auf die Lichtschäfte untersucht. Es lässt sich beobachten, dass die Glättung der Schattenkanten bei den Schatten auf Oberflächen einen weit größeren Einfluss als bei volumetrischen Schatten hat, was speziell die Bilder 3.18(c) und 3.18(c) zeigen. Weichere Übergänge in den Lichtschäften sind vorhanden, allerdings erst bei näherer Betrachtung zu erkennen. Eine sich lohnende Steigerung der Plausibilität der Bilder ist fraglich, zumal sich die Renderzeiten beim Einsatz von PCF rund verdreifachen. Es sollte an dieser Stelle erwähnt werden, dass PCF nicht die einzige Technik zur Glättung von Schattenkanten ist und eine Kombination des Verfahrens von Tóth und Umenhoffer mit anderen Varianten von Shadow Mapping theoretisch möglich ist.

Das Verfahren von Tóth und Umenhoffer unterstützt physikalische Konzepte, wie zum Beispiel beliebige Streuphasenfunktionen. Durch die Funktion von Henyey-Greenstein kann dünner bis dichter Nebel simuliert werden. Während von Nebel hauptsächlich Mie-Streuung verursacht wird, findet in reiner Luft Rayleigh-Streuung statt, deren Phasenfunktion ebenfalls verwendet werden kann. Die Streuphasenfunktionen wurden bereits in Abschnitt 2.2.2 erläutert. Charakteristisch für Rayleigh-Streuung ist, dass

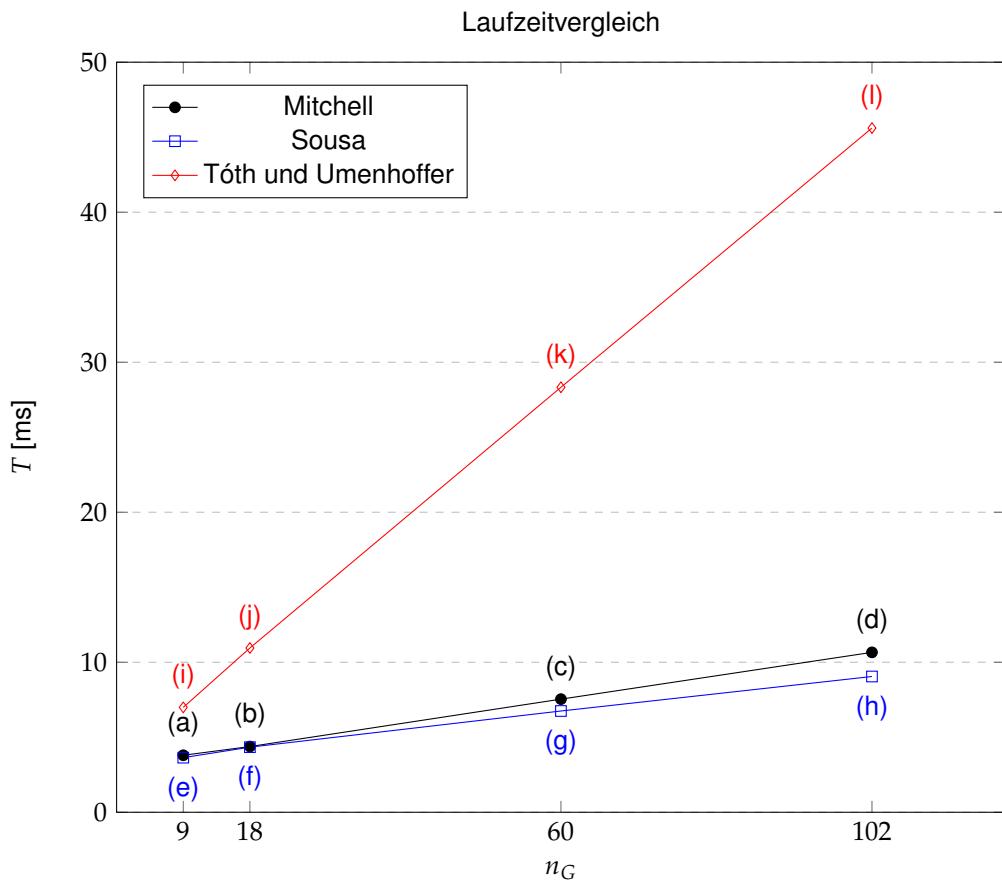


Abbildung 3.16.: Laufzeiten der Verfahren von Mitchell, Sousa und Tóth und Umenhoffer.

$T$  ist die Renderzeit für ein Bild in Millisekunden;  $n_G$  ist die Gesamtanzahl Samples, die für die Erzeugung eines Bildes benötigt werden.

Beachte, dass bei Sousa  $n_G = 3 \cdot n$  und bei Mitchell sowie Tóth und Umenhoffer  $n_G = n$  gilt. Die Buchstaben ordnen den Messungen das jeweilige Bild in Abbildung 3.17 zu.

starke Vorwärts- und Rückwärtsstreuung stattfindet, jedoch nur ein geringer Anteil in einem Winkel von  $90^\circ$  gestreut wird. Bei Mie-Streuung hängt dies vom Parameter  $g$  in der Phasenfunktion von Henyey-Greenstein ab. Bei kleinen  $g$  erhält man in allen Richtungen einen nahezu gleichen Anteil gestreuten Lichts (annähernd isotrope Streuung). Bei großen  $g$  erhält man viel Vorwärtsstreuung und wenig gestreutes Licht in anderen Richtungen. Die Auswirkungen der Streuphasenfunktionen werden in Abbildung 3.19 gezeigt und erklärt.

### 3. PRAKTISCHE UMSETZUNG

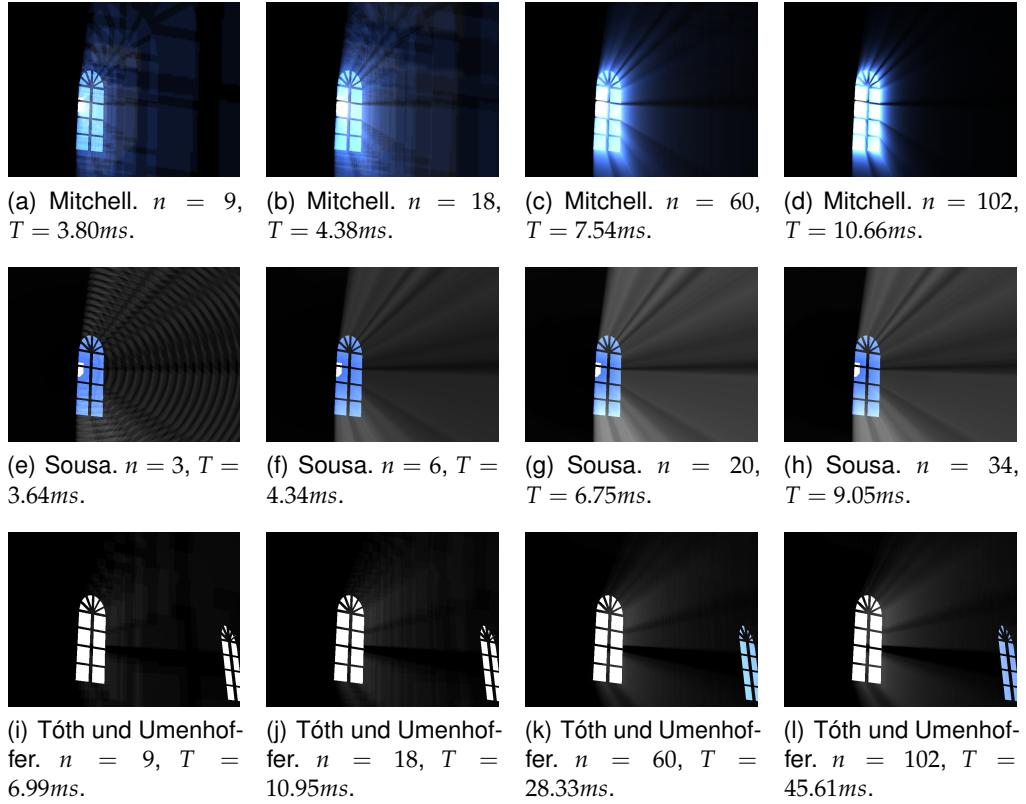
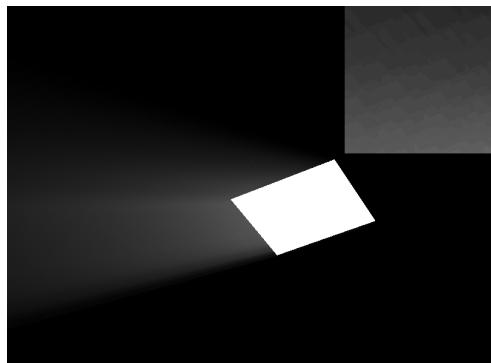


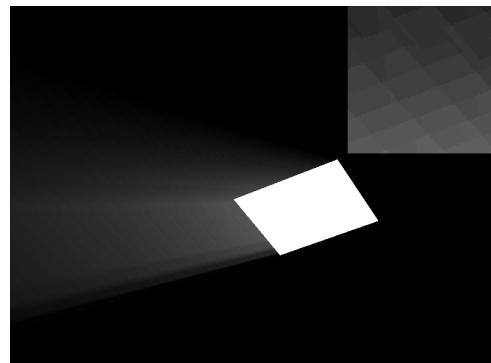
Abbildung 3.17.: Kathedrale von innen. Laufzeit und Bildqualität der Verfahren von Mitchell, Sousa und Tóth und Umenhoffer im Vergleich. Das Verfahren von Sousa liefert bereits bei 18 Samples eine akzeptable Bildqualität (Bild (f)), während von den Verfahren von Mitchell und Tóth und Umenhoffer bei der gleichen Anzahl Samples starke Abtastartefakte produziert werden (Bild (b) und Bild (j)).

#### 3.4.2. Limitierungen

Bezüglich der physikalischen Korrektheit haben die implementierten Verfahren einige offensichtliche Nachteile. Bei den bildbasierten Verfahren stehen nicht die nötigen Informationen zur Verfügung, um einfach eingestreutes Licht genau zu berechnen. Insbesondere fehlt die Länge der Sichtstrahlen. Außerdem lassen sich die Eigenschaften des Mediums nicht mittels physikalischer Parameter wie der Albedo oder Dichte konfigurieren. Eine weitere Limitierung, die für alle implementierten Verfahren gilt, ist das Ignorieren von mehrfacher Streuung. Es wird nur Licht beachtet, das nach einem einzigen Streuereignis im Medium die Kamera erreicht. Für die Abbildungen 3.20 und 3.21 wurde der physikalisch-basierte Renderer Mitsuba [8] als Referenz bezüglich



(a) PCF-Shadow Mapping. Die Größe des Filterkernels ist  $3 \times 3$  Texel. Renderzeit: 0.0802s.



(b) Shadow Mapping ohne Filter. Renderzeit: 0.026s.



(c) PCF-Shadow Mapping. Die Größe des Filterkernels ist  $3 \times 3$  Texel. Renderzeit: 0.140s.

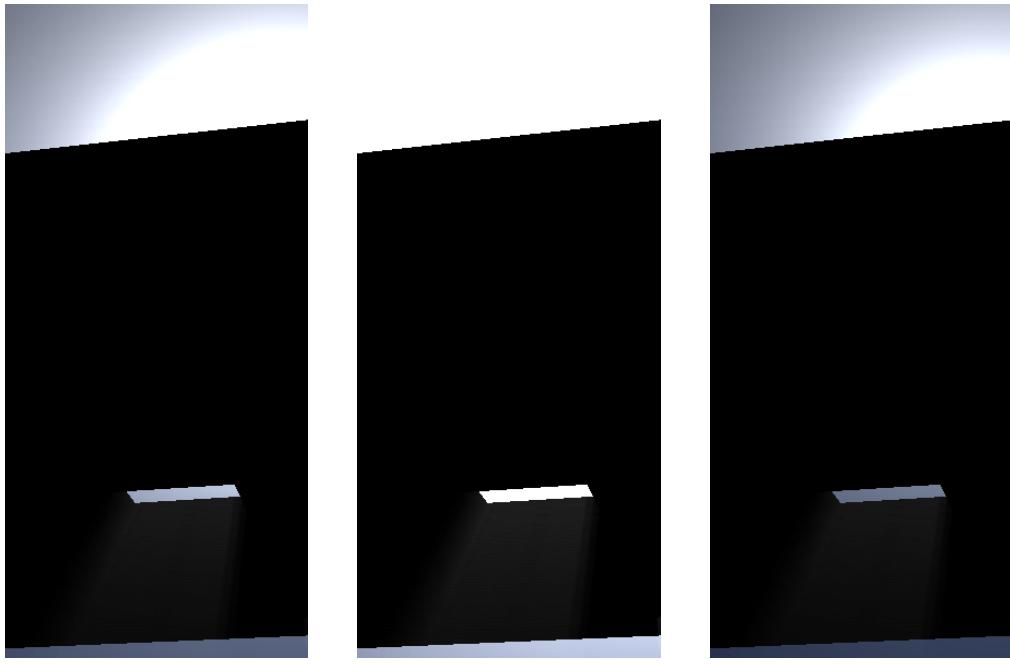


(d) Shadow Mapping ohne Filter. Renderzeit: 0.044s.

Abbildung 3.18.: Auswirkungen von Shadow Map-Filterung im Verfahren von Tóth und Umenhoffer. Die Bilder (a) und (b) zeigen, dass die Auswirkung auf die Lichtschäfte erst im Detail sichtbar wird. Die Auswirkung auf Schatten, die auf Oberflächen fallen, sind deutlicher zu erkennen, wie die Bilder (c) und (d) zeigen. Eine weitere Beobachtung ist der Anstieg der Renderzeit um ungefähr das Dreifache, wenn PCF aktiviert wird.

Genauigkeit der implementierten Echtzeitverfahren herangezogen. Der Mitsuba Renderer unterstützt Path Tracing. Dabei wird der Weg des Lichts von der Lichtquelle zu einem Pixel über mehrere Streuereignisse auf einem Material bzw. in einem Medium hinweg verfolgt. Die in diesem Abschnitt gezeigten Bilder, die mit dem Mitsuba Renderer erzeugt wurden, zeigen Licht nach maximal 3 Streuereignissen und kommen einem wahren Erscheinungsbild folglich näher als die Verfahren von Mitchell, Sousa und Tóth und Umenhoffer, die nur Einfachstreuung beachten. Die Konfiguration homogener Medien kann im Mitsuba Renderer unter anderem über den Extinktionskoeffizienten

### 3. PRAKTISCHE UMSETZUNG



(a) Rayleigh-Streuung. Starke Vorwärts- und Rückwärtsstreuung, aber wenig senkrechte Streuung.

(b) Mie-Streuung. Funktion von Henyey-Greenstein mit  $g = 0.07$ ; somit nahezu isotrope Streuung.

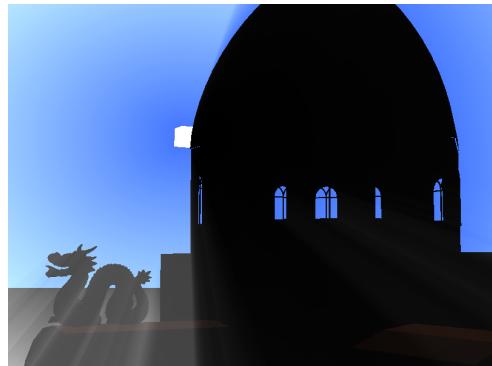
(c) Mie-Streuung. Funktion von Henyey-Greenstein mit  $g = 0.5$ ; somit starke Verwärtsstreuung und wenig Licht in anderen Richtungen.

Abbildung 3.19.: Auswirkungen verschiedener Streuphasenfunktionen im Verfahren von Tóth und Umenhoffer. Die Ergebnisse stimmen mit dem physikalisch-basierten Modell in Abschnitt 2.2.2 in der Hinsicht überein, dass Bild (b) den hellsten Lichtschaft aufweist, da die Sichtrichtung an der Stelle des Lichtschafts fast senkrecht zur Lichtrichtung ist.

in inversen Längeneinheiten, die Albedo sowie eine Phasenfunktion erfolgen. Die Bedeutung des Extinktionskoeffizienten entspricht der Dichte im Verfahren von Tóth und Umenhoffer, die auch in inversen Längeneinheiten gemessen wird; für diese beiden Parameter wurde in den Abbildungen 3.20(c), 3.20(d), 3.21(c) und 3.21(d) der Wert 0.0001 verwendet. Die Albedo hat jeweils den Wert 0.9. Als Phasenfunktion wurde die Henyey-Greenstein-Funktion mit  $g = 0.5$  benutzt.

Die implementierten Verfahren machen die vereinfachende Annahme, dass das streuende Medium homogen ist. Anders gesagt, die Verfahren nehmen an, es sei nur ein einziges Medium vorhanden. Dies ist eine starke Vereinfachung, die mit einigen Einschränkungen verbunden ist, da in der realen Welt oft Mischungen von Gasen vorkommen. Ein Beispiel von God Rays in einem inhomogenen Medium zeigt Ab-

bildung 3.22. Der Vorteil bei homogenen Medien ist, dass weder eine aufwendige Modellierung noch eine räumliche Datenstruktur zur Speicherung und Abfrage der Medienparameter erforderlich ist. Setzt man beides voraus, ist die Berechnung der Streuung in einem inhomogenen Medium durch ein Ray-Marching-basiertes Verfahren theoretisch möglich. Ob die Berechnung in Echtzeit durchgeführt werden könnte, ist eine andere Frage, der in dieser Arbeit nicht weiter nachgegangen wird.



(a) Verfahren von Sousa. Renderzeit: 0.023s.



(b) Verfahren von Mitchell. Renderzeit: 0.029s.



(c) Verfahren von Tóth und Umenhoffer. Renderzeit: 0.082s.



(d) Mitsuba Path Tracer. Renderzeit: > 8h.

Abbildung 3.20.: Kathedrale von außen. Die Verfahren Mitchell, Sousa und Tóth und Umenhoffer im Vergleich zu Renderings in Mitsuba. Die Konfiguration des Mediums ist in (c) und (d) gleich, allerdings wird in (d) bis zu dreifache Streuung beachtet, während (c) nur Einfachstreuung beachtet. Bei der Betrachtung von (a) und (b) fällt auf, dass die Plausibilität der bildbasierten Verfahren in dieser Szene mit den physikalisch-basierten Verfahren vergleichbar ist.

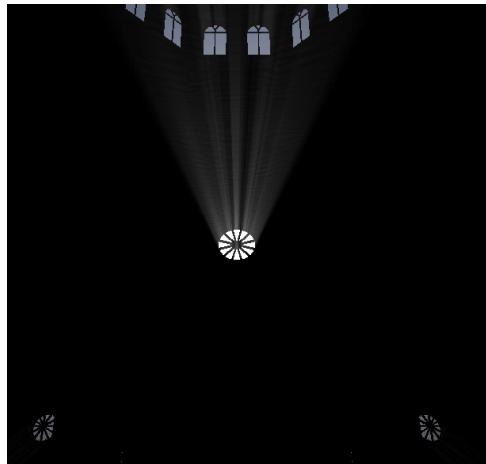
### 3. PRAKTISCHE UMSETZUNG



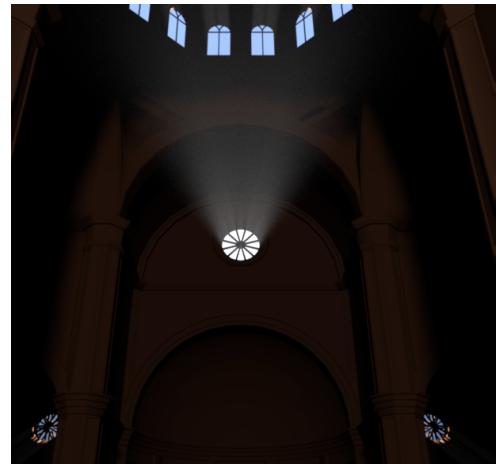
(a) Verfahren von Sousa. Renderzeit: 0.004s.



(b) Verfahren von Mitchell. In dieser Szene, in der nur kleine Teile des Himmels sichtbar sind, musste der Parameter *density* erhöht werden, was in sehr kurzen Lichtschläften und damit eingeschränkter Plausibilität resultiert. Außerdem kann die hier sehr kontrastreiche Färbung der Lichtschläge irritieren. Renderzeit: 0.023s.



(c) Verfahren von Tóth und Umenhoffer. Renderzeit: 0.058.



(d) Mitsuba Path Tracer. In diesem Bild, das bis zu dreifache Streuung zeigt, sind die beschatteten Regionen nicht so klar von den beleuchteten Regionen getrennt wie in den anderen Bildern, die nur Einfachstreuung zeigen. Renderzeit: > 8h.

Abbildung 3.21.: Kathedrale von innen. Die Medienparameter in (c) und (d) sind gleich. In (a), (b) und (d) existieren zwei Lichtquellen, während in (c) wegen Einschränkungen der vorliegenden Implementierung von Shadow Mapping nur eine Lichtquelle möglich war.



Abbildung 3.22.: Lichtstreuung in inhomogenen Medien, so wie in dieser Wolke über einer Thermalquelle, ist ein Phänomen, das in den implementierten Verfahren nicht unterstützt wird. Autor: Mila Zinkova, Lizenz: Creative Commons Attribution-Share Alike 3.0 Unported



# 4. Fazit

Zusammenfassend können einige allgemeine Punkte festgehalten werden. In computergenerierten Bildern können komplexe Beleuchtungseffekte in Echtzeit plausibel dargestellt werden. Um ein plausibles und natürliches Erscheinungsbild zu erhalten, greift man auf physikalisches Wissen zurück. Das heute sehr detaillierte Wissen der Physik wird nur bis zu einer gewissen Tiefe verfolgt. In der realistischen Computergrafik konzentriert man sich auf die Entwicklung und Optimierung von Algorithmen, die oft auf geschickt gewählten Vereinfachungen beruhen. Diese Forschungsanstrengungen ermöglichen unter anderem die kurzen Rechenzeiten, die in der Echtzeitgrafik benötigt werden.

## 4.1. Problemstellung und Lösung

Diese Arbeit beschäftigte sich mit der Theorie und praktischen Umsetzung der Visualisierung von God Rays bzw. volumetrischer Lichtstreuung. An die Umsetzung wurden spezielle Anforderungen gestellt, welche die Konzentration auf Echtzeitverfahren und die Programmierung in C++ und OpenGL umfassen. Die erzielten Ergebnisse sollten schließlich evaluiert werden.

Nach Aufarbeitung der physikalischen Grundlagen von Strahlung und Lichtstreuung wurde die Lichtstreuung in der Erdatmosphäre betrachtet. Für die Computer-, insbesondere die Echtzeitgrafik, müssen vereinfachende Modelle gefunden werden, um das Phänomen der volumetrischen Lichtstreuung in angemessener Rechenzeit visuell darstellen zu können. In diesem Zusammenhang wurde das Modell von Hoffmann und Preetham [5] vorgestellt.

Das Phänomen von God Rays konnte nun als Spezialfall von volumetrischer Lichtstreuung in Kombination mit Schatten gesehen werden. Die Literatur zum Thema God Rays verfolgt verschiedene Ansätze, um Lichtstreuung und Schatten im 3D-Raum auszuwerten. Ein Ansatz schätzt die Schatten anhand von Informationen aus dem zweidimensionalen Bildraum, und zeichnet God Rays in einem Nachbearbeitungsschritt. Ein anderer Ansatz führt eine Abtastung des gesamten Sichtvolumens durch und prüft an jedem abgetasteten Punkt die Sichtbarkeit der Lichtquelle. Für die Implementierung wurden zwei Vertreter des erstgenannten Ansatzes, die Verfahren von Mitchell [12] und

#### 4. FAZIT

Sousa [17], und ein Vertreter des letztgenannten Ansatzes, das Verfahren von Tóth und Umenhoffer [18], ausgewählt.

Die Verfahren von Mitchell und Sousa sind künstlerisch orientiert und verfolgen das Ziel, einen ästhetischen Effekt mit einer technisch nicht zu aufwendigen Implementierung zu erhalten. Hierbei traten im Zuge der Evaluierung Einschränkungen bei der Genauigkeit hervor. Insbesondere können God Rays nur dann erzeugt werden, wenn sich die Lichtquelle im Bildraum befindet. Im Gegensatz dazu weist das Verfahren von Tóth und Umenhoffer eine eher physikalische Orientierung auf. Die in vielen Fällen plausibleren Ergebnisse gehen jedoch mit längeren Rechenzeiten einher.

## 4.2. Ausblick

Die meisten publizierten Verfahren für God Rays machen zwei große Vereinfachungen. Die erste Vereinfachung ist das Ignorieren von Licht, das mehrfach im Medium gestreut wurde. Dies verhindert, dass alle Partikel im Medium miteinander in Abhängigkeit treten. Durch die Reduzierung der Abhängigkeiten ermöglicht man Parallelisierung. Das ist insbesondere bei einer Umsetzung mit Grafikhardware wichtig. Solange man die Partikel, die durch ein Pixel gesehen werden, unabhängig von den anderen Partikeln im Medium behandeln möchte, muss die Vereinfachung auf Einfachstreuung beibehalten werden. Die zweite Vereinfachung ist die Annahme eines homogenen Mediums. Die Schwierigkeit bei inhomogenen Medien besteht in der Modellierung und Zwischenspeicherung. Für die Modellierung sind gemessene Dichteverteilungen oder prozedurale Generierung denkbar. Als Datenstruktur zur Zwischenspeicherung kann eine dreidimensionale Textur verwendet werden. Dann kann insbesondere das implementierte Verfahren von Tóth und Umenhoffer erweitert werden, indem an jedem abgetasteten Punkt die dort bestehende Dichte des Mediums für die Berechnung benutzt wird.

Neue Ansätze in der Literatur [10] [14] schlagen eine Erweiterung von Shadow Mapping vor. Eine aufwendige Abtastung des Sichtvolumens ist nicht mehr nötig, wenn die benötigten Informationen für Streuung entlang der Sichtstrahlen in jedem Texel der Shadow Map zu finden sind. Die Ansätze erfordern eine Rektifizierung der Shadow Map, sodass die Zeilen entlang der Sichtstrahlen ausgerichtet sind und eine anschließende Akkumulation der eingestreuten Lichtintensität an jedem Texel. Diese Ansätze sind vielversprechend, da sie sowohl die Genauigkeitsprobleme der bildbasierten Verfahren, als auch die Rechenzeitprobleme der volumetrischen Verfahren vermeiden.

# A. Anhang

---

**Programmausdruck A.1:** Der Code demonstriert, wie das Framework benutzt wird, um eine Szene aus einer Wavefront/OBJ-Datei zu laden und zu rendern.

---

```
// 1. Erstelle Kamera (ezeugt Fenster und OpenGL Kontext)
Camera camera(
    glm::vec3(22, 6, -21), // Position
    glm::vec3(7, 9, 6),   // Ziel
    glm::vec3(0, 1, 0),   // Oben
    25,                  // Oeffnungswinkel
    glm::vec2(800, 600), // Bildaufloesung
    0.1f,                // Abstand vorderer Clippingebene
    800.0f);             // Abstand hinterer Clippingebene

// 2. Erstelle Punktlichtquelle
Light light({{
    {1.0f, 0.0f, 1.0f},           // Farbe im HSL-Farbraum
    {3.0f, 10.0f, -60.0f}}}); // Position

// 3. Lade und kompiliere Shader aus Datei
GLuint shader = Shader::link({
    VertexShader("triangle.vert"),
    FragmentShader("triangle.frag")});

// 4. Definiere Modelltyp
ModelType type(
    0,                         // ID
    GL_TRIANGLES,               // Zeichenprimitiv
    shader,                     // Shader
    {VertexAttribute("normal")}, // Liste der Vertexattribute
    {Material()});              // Liste der Instanzattribute
```

## A. ANHANG

```
// 5. OBJLoader kann Vertices aus OBJ-Dateien lesen
// und ein internes Format konvertieren
OBJLoader obj;

// 6. Erstelle Modell
Model scene(
    0,                                // ID
    &type,                            // Modelltyp
    obj.load("sibenik.obj")); // Lade Vertexdaten aus Datei

// 7. Erstelle Welt und füge ein Modell hinzu
World world({&sibenik}, [](Model* m) {
    // Callback für Debugging
    // wird aus der Renderschleife für jedes Modell aufgerufen
}, {
    // Liste für allgemeine Shadervariablen
});

// 8. Erweitere Welt durch Module
world.extend(&camera);
world.extend(&light);

// 9. Setze allgemeine OpenGL-Zustandsvariablen
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClearDepth(1.0f);
 glEnable(GL_CULL_FACE);
 glCullFace(GL_BACK);
 glFrontFace(GL_CW);

// 10. Starte die Renderschleife
while(camera.is_on()) {
    camera.shoot(&world);
}
```

---

---

**Programmausdruck A.2:** Der Aufbau des Moduls für das Verfahren von Mitchell [12].

---

```
class VolumetricLightScatteringMitchell : public Module {

    // Framebuffer als Ziel fuer das Verdeckungsbild
    GPUBuffer occlusion_framebuffer_;

    // Textur, die an den Framebuffer angehaengt wird
    // und als Zwischenspeicher fuer das Verdeckungsbild dient
    GLuint occlusion_texture_;

    // Zeiger auf eine Instanz von Camera::PostProcessor,
    // die den Shader fuer Post-Processing ausfuehrt.
    Camera::PostProcessor* post_processor_;

    // Position der Lichtquelle in Weltkoordinaten
    glm::vec3 lightsource_worldspace_;

    // Parameter des Post-Processing-Algorithmus von Mitchell
    int NUM_SAMPLES_;
    float EXPOSURE_;
    float DENSITY_;
    float DECAY_;

    [...]
};
```

---

**Programmausdruck A.3:** Senden der Variablen für das Verfahren von Mitchell[12] an die GPU und Setzen der Werte im Shader für Post-Processing. Für die Uniform-Updates werden von der Klasse Camera::PostProcessor sog. Callbacks verwendet, die vor jedem Renderdurchlauf aufgerufen werden.

---

```
std::vector<Uniform> Volumetric[...].uniforms() {

    // Binde die Textur mit dem Verdeckungsbild an Textureunit 1
    post_processor_->sampler(1, occlusion_texture_);

    // Transformiere Position der Lichtquelle in Texturkoordinaten
    // (Nach der Transformation wird diese Position im Post-Processing
    // als Zentrum fuer den Radial Blur-Filter verwendet)
    post_processor_->uniform("lightsource", [this] (GLint location) {
        glm::vec2 l =
            post_processor_->camera()->transform_screen_to_texturespace(
                post_processor_->camera()->transform_world_to_screenspace(
                    glm::vec4(this->lightsource_worldspace_, 1)));
        glUniform2f(location, l.x, l.y);
    });

    // Setze die fuer das Verfahren von Mitchell
    // benoetigten Parameter
    post_processor_->uniform("NUM_SAMPLES", [this] (GLint location) {
        glUniform1i(location, this->NUM_SAMPLES_);
    });
    post_processor_->uniform("EXPOSURE", [this] (GLint location) {
        glUniform1f(location, this->EXPOSURE_);
    });
    post_processor_->uniform("DENSITY", [this] (GLint location) {
        glUniform1f(location, this->DENSITY_);
    });
    post_processor_->uniform("DECAY", [this] (GLint location) {
        glUniform1f(location, this->DECAY_);
    });

    // An den Shader des regulaeren Durchlaufs
    // werden keine Uniforms uebergeben
}
```

```
    return {};  
}
```

---

---

**Programmausdruck A.4:** Post-Processing Fragmentshader für das Verfahren von Mitchell [12].

---

```
#version 430  
out vec4 outColor;  
  
// Texturkoordinaten, die stets den  
// aktuellen Pixelkoordinaten entsprechen  
in vec2 pixel;  
  
// Bild aus dem regulaeren Renderdurchlauf,  
// das die Farben der Objektoberflaechen enthaelt  
layout(binding = 0) uniform sampler2D world_image;  
  
// Verdeckungsbild aus dem vorgeschalteten Renderdurchlauf  
layout(binding = 1)  
    uniform sampler2D occlusion_pre_pass_image;  
  
// Position der Lichtquelle in Texturkoordinaten  
uniform vec2 lightsource;  
  
// Parameter des Verfahrens von Mitchell  
uniform int NUM_SAMPLES;  
uniform float EXPOSURE;  
uniform float DENSITY;  
uniform float DECAY;  
  
void main() {  
    // Berechne Vektor vom aktuellen Pixel zur Lichtquelle  
    vec2 s = pixel - lightsource;  
    // Berechne Schrittweite  
    s *= 1.0 / (NUM_SAMPLES * DENSITY);  
    // Lese Pixelfarbe  
    vec4 pixelcolor =  
        texture(occlusion_pre_pass_image, pixel);
```

## A. ANHANG

```
// Initialisiere Wert der exponentiellen Extinktion
float expDecay = 1.0;
// Initialisiere Koordinaten der Abtastung
vec2 smplcoords = pixel;
for(int i = 0; i < NUM_SAMPLES; i++) {
    // Reduziere Distanz zur Lichtquelle
    smplcoords -= s;
    // Abtastung der Pixelfarbe an aktuellen Koordinaten
    vec4 smplcolor = texture(occlusion_pre_pass_image, smplcoords);
    // Multipliziere mit Extinktionsfaktor
    smplcolor *= expDecay;
    // Akkumuliere Pixelfarbe
    pixelcolor += smplcolor;
    // Erhoehe Extinktionsfaktor
    expDecay *= DECAY;
}
// Reguliere Helligkeit und kombiniere mit Oberflaechenfarbe
outColor = pixelcolor * EXPOSURE +
texture(world_image, pixel);
}
```

---

---

### Programmausdruck A.5: Aufbau des Moduls für das Verfahren von Sousa [17].

---

```
class SunShaftsSousa : public Module {

    // Framebuffer und angehaengte Textur
    // als Ziel des ersten Renderdurchlaufs
    // (Der erste Durchlauf rendert eine Tiefenkarte
    // aus Sicht der Kamera)
    GPUBuffer framebuffer1_;
    GLuint depth_map1_;

    // Zeiger auf eine Instanz von Camera::PostProcessor,
    // welche den zweiten Renderdurchlauf ausfuehrt
    // (Der zweite Durchlauf bearbeitet die Tiefenkarte
    // ein erstes Mal)
    Camera::PostProcessor* post_processor_1_;
```

```

// Zeiger auf eine Instanz von Camera::PostProcessor,
// die als Ziel des zweiten Renderdurchlaufs dient,
// und den dritten Durchlauf ausfuehrt
// (Der dritte Durchlauf bearbeitet die Tiefenkarte
// ein zweites Mal)
Camera::PostProcessor* post_processor_2_;

// Framebuffer und angehaengte Textur
// als Ziel des dritten Renderdurchlaufs
GPUBuffer framebuffer2_;
GLuint depth_map2_;

// Zeiger auf eine Instanz von Camera::PostProcessor,
// die als Ziel des vierten/regulaeren Renderdurchlaufs dient,
// und den fuenften Durchlauf ausfuehrt.
// (Der fuenfte Durchlauf bearbeitet die Tiefenkarte
// ein drittes Mal und kombiniert diese
// mit dem Bild des regulaeren Durchlaufs)
Camera::PostProcessor* composition_post_processor_;

// Position der Lichtquelle in Weltkoordinaten
glm::vec3 lightsource_worldspace_;

// Parameter des Verfahrens von Sousa
float STEPSIZE_1_;
float STEPSIZE_2_;
float STEPSIZE_3_;
float stepsize_;
float brightness_;
int TAPS_;

[...]
};

```

---

## A. ANHANG

---

**Programmausdruck A.6:** Fragmentshader, der beim Verfahren von Sousa [17] für das Erstellen der Tiefenkarte ausgeführt wird. Die Tiefenwerte werden vom perspektivisch verzerrten in ein lineares Koordinatensystem transformiert.

---

```
#version 430
layout(location = 0) uniform float near;
layout(location = 1) uniform float far;

void main() {
    float tmp = (gl_FragCoord.z * 2.0 - 1.0) * (far - near);
    gl_FragDepth = (2.0 * near * far) / (far + near - tmp) / far;
}
```

---

---

**Programmausdruck A.7:** Post-Processing Fragmentshader für das Verfahren von Sousa [17]. Der Fragmentshader wird mehrmals mit jeweils unterschiedlicher Schrittweite ausgeführt.

---

```
// Texturkoordinaten, die stets den
// aktuellen Pixelkoordinaten entsprechen
in vec2 pixel;

// Tiefenkarte aus Sicht der Kamera
uniform sampler2D surface_map;

// Maximale Anzahl Samples
uniform int TAPS;

// Schrittweite des aktuellen Durchgangs
uniform float stepsize;

// Parameter zur Regulierung der Helligkeit
uniform float brightness;

vec4 SousaBlur() {
    // Berechne Vektor vom aktuellen Pixel zur Lichtquelle
    vec2 v = lightsource - pixel;
```

```

// Berechne die Laenge des Vektors
float l = length(v);
// Berechne den Schrittvektor, der in Richtung von v zeigt
// und die Laenge von stepsize hat
vec2 stp = stepsize * v / l;
// Berechne Schritte, die gemacht werden koennen,
// ohne Strecke zwischen Pixel und Lichtquelle zu verlassen
float MAX_STEPS = l / stepsize;
// Initialisiere Tiefenwert
float sum = 0.0;
// Initialisiere Koordinaten der Abtastung
vec2 p = pixel;
for(int i = 0; i < TAPS; i++) {
    // Breche Abtastung ab, wenn hinter Lichtquelle
    if(i >= MAX_STEPS) break;
    // Akkumuliere Tiefenwert
    sum += texture(surface_map, p).r;
    // Gehe Schritt Richtung Lichtquelle
    p += stp;
}
// Reguliere Helligkeit
return vec4(sum * brightness);
}

```

---

**Programmausdruck A.8:** Aufbau des Moduls für Shadow Mapping. Es wird eine Matrix benötigt, um die Szene in die Perspektive der Lichtquelle zu transformieren, aus der die Tiefenkarte gerendert wird. Für das Rendering der Tiefenkarte wird die Auflösung des OpenGL-Viewport geändert und muss danach wieder auf den ursprünglichen Wert gesetzt werden.

---

```

class ShadowMapping : public Module {

    // Auflösung der Tiefenkarte
    glm::vec2 depth_map_resolution_;

    // Framebuffer und angehaengte Textur als Ziel
    // fuer das Rendering der Tiefenkarte

```

## A. ANHANG

```
GPUBuffer depth_framebuffer_;
GLuint depth_texture_;

// Transformationsmatrix von Weltkoordinaten in
// den Lichtraum
glm::mat4 light_space_matrix_;

// Urspruengliche Aufloesung
glm::vec2 original_resolution_;

[...]
};
```

---

**Programmausdruck A.9:** Vertexshader für das Verfahren von Toth. Die Koordinaten des Oberflächenpunkts und die Kameraposition werden in den Lichtraum transformiert.

---

```
#version 430

// Transformation von Modell- in Weltkoordinaten
in mat4 ModelMatrix;

// Transformation von Welt- in Kamerakoordinaten
layout(std140) uniform Camera {
    mat4 view;
    mat4 proj;
};

// Transformation von Welt- in Lichtkoordinaten
uniform mat4 LightSpaceMatrix;

// Vertexposition
in vec3 position;

// Vertexposition in Weltkoordinaten
// (wird interpoliert)
out vec3 fragPosition;
```

```

// Vertexposition in Lichtkoordinaten
// (wird interpoliert)
out vec4 fragPosLightSpace;

// Kameraposition in Weltkoordinaten
uniform vec3 cameraPosWorldSpace;

// Kameraposition in Lichtkoordinaten
// (wird nicht interpoliert)
flat out vec4 fragCameraPosLightSpace;

[...]

void main() {
    // Transformiere Vertexposition in Weltkoordinaten
    fragPosition = vec3(ModelMatrix * vec4(position, 1));
    // Transformiere Vertexposition in Lichtkoordinaten
    fragPosLightSpace =
        LightSpaceMatrix * vec4(fragPosition, 1);
    // Transformiere Kameraposition in Lichtkoordinaten
    fragCameraPosLightSpace =
        LightSpaceMatrix * vec4(cameraPos, 1);
    [...]
}

```

---

**Programmausdruck A.10:** Funktion für einen einfachen binären Schattentest im Fragmentshader. Es wird geprüft, ob ein gegebener Punkt im Lichtraum von der Lichtquelle aus sichtbar ist oder nicht.

---

```

#version 430

layout(binding = 0) uniform sampler2D ShadowMap;

[...]

bool testShadow(vec4 positionLightSpace) {
    // Wende perspektivische Projektion an,
    // um normalisierte Geraetekoordinaten zu erhalten

```

## A. ANHANG

```
// (Beim Rendern der Shadow Map wurde dieser Schritt
// implizit durchgefuehrt)
vec3 ndc = positionLightSpace.xyz / positionLightSpace.w;

// Wenn der Punkt nicht im Sichtkegel der Lichtquell liegt,
// wird kein Schatten angenommen
if(ndc.x < -1 || ndc.x > 1 ||
    ndc.y < -1 || ndc.y > 1 ||
    ndc.z > 1 || ndc.z < 0) {
    return false;
}

// Transformiere normalisierte Geraetekoordinaten
// in Texturkoordinaten
ndc = ndc * 0.5 + 0.5;

// Lese Tiefenwert aus Shadow Map
float closestDepth = texture(ShadowMap, ndc.xy).r;

// Vergleiche die Tiefenwerte unter Anwendung eines
// Schwellenwertes, um Moireeffekt zu vermeiden
float currentDepth = ndc.z;
return closestDepth < (currentDepth-0.0001);
}

[...]
```

---

**Programmausdruck A.11:** Implementierung der Streuphasenfunktionen für Rayleigh- und Mie-Streuung im Fragmentshader.

---

```
#define M_PI 3.1415926535897932384626433832795

// Parameter g in der Henyey-Greenstein-Phasenfunktion
uniform float HG_g;

// Henyey-Greenstein-Phasenfunktion,
// mit der Streuung in Nebel simuliert werden kann
float HenyeyGreenstein(
```

```

vec3 particlePosLightSpace,
vec3 viewDirLightSpace)
{
    float cosine = dot(
        normalize(particlePosLightSpace),
        normalize(viewDirLightSpace));
    float term1 = pow(1.0 - HG_g, 2);
    float term2 = 1.0 + pow(HG_g, 2) - 2 * HG_g * cosine;
    float term3 = 4 * M_PI * pow(term2, 1.5);
    return term1/term2;
}

// Rayleigh-Phasenfunktion, mit der Streuung
// in Luft simuliert werden kann
float Rayleigh(
    vec3 particlePosLightSpace,
    vec3 viewDirLightSpace)
{
    float cosine = dot(
        normalize(particlePosLightSpace),
        normalize(viewDirLightSpace));
    return pow(cosine, 2);
}

```

---

**Programmausdruck A.12:** Ray-Marching Fragmentshader für das Verfahren von Toth [18]. Die Funktion erwartet als Eingabe die Farbe eines Oberflächenpunktes  $L$ . Die Ausgabe ist  $L$  nach Beeinflussung durch Extinktion und Streuung im Medium. Das Verfahren arbeitet mit Koordinaten im Lichtraum.

---

```

#define M_PI 3.1415926535897932384626433832795

// Anzahl der Samples
uniform int n;
// Leistung der Lichtquelle
uniform float P;
// Medienabhaengige Albedo
uniform float albedo

```

## A. ANHANG

```
// Dichte des Mediums
uniform float density

vec4 ParticipatingMediaToth(vec4 L) {
    // Berechne den Sichtvektor
    vec4 viewDirLightSpace =
        fragCameraPosLightSpace - fragPosLightSpace;
    // Berechne die Distanz des Oberflaechenpunktes zur Kamera
    float s = length(viewDirLightSpace);
    // Normalisiere den Sichtvektor
    viewDirLightSpace = normalize(viewDirLightSpace);
    // Berechne die Schrittweite
    float ds = s / float(n);
    // Berechne Extinktion
    L *= exp(-s * density);
    // Initialisiere Abtastpunkt mit Oberflaechenpunkt
    vec4 x = fragPosLightSpace;
    while(s >= 0) {
        // Reduziere die Distanz zur Kamera
        s -= ds;
        // Bewege Abtastpunkt in Richtung Kamera
        x += viewDirLightSpace * ds;
        // Werte Sichtbarkeit des Abtastpunktes von der Lichtquelle aus
        float S = testShadow(x) ? 0.0 : 1.0;
        // Berechne die Distanz des Abtastpunktes von der Lichtquelle
        float d = length(x);
        // Berechne resultierende Einstreuung
        L += S * albedo * density *
            P/4/M_PI/pow(d,2) *
            exp(-d * density) *
            exp(-s * density) *
            scatteringPhaseFunction(x.xyz, viewDirLightSpace.xyz);
    }
    return L;
}
```

---

# Literaturverzeichnis

- [1] I. Baran, J. Chen, J. Ragan-Kelley, F. Durand, and J. Lehtinen, “A hierarchical volumetric shadow algorithm for single scattering,” in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 6. ACM, 2010, p. 178.
- [2] J. Chen, I. Baran, F. Durand, and W. Jarosz, “Real-time volumetric shadows using 1d min-max mipmaps,” in *Symposium on Interactive 3D Graphics and Games*. ACM, 2011, pp. PAGE–7.
- [3] P. Dutre, K. Bala, P. Bekaert, and P. Shirley, *Advanced Global Illumination*. AK Peters Ltd, 2006.
- [4] R. P. Feynman, R. B. Leighton, and M. Sands, *The Feynman Lectures on Physics, Desktop Edition Volume I*. Basic books, 2013, vol. 1.
- [5] N. Hoffman and A. J. Preetham, “Rendering outdoor light scattering in real time,” in *Proceedings of Game Developer Conference*, vol. 2002, 2002, pp. 337–352.
- [6] ——, “Graphics programming methods,” J. Lander, Ed. Rockland, MA, USA: Charles River Media, Inc., 2003, ch. Real-time Light-atmosphere Interactions for Outdoor Scenes, pp. 337–352. [Online]. Available: <http://dl.acm.org/citation.cfm?id=957155.957189>
- [7] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley, *Computer graphics: principles and practice (3rd ed.)*. Boston, MA, USA: Addison-Wesley Professional, July 2013.
- [8] W. Jakob, “Mitsuba renderer,” 2010, <http://www.mitsuba-renderer.org>.
- [9] A. Keller and W. Heidrich, “Interleaved sampling,” in *Rendering Techniques 2001*. Springer, 2001, pp. 269–276.
- [10] O. Klehm, H.-P. Seidel, and E. Eisemann, “Prefiltered single scattering,” in *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2014, pp. 71–78.

## LITERATURVERZEICHNIS

- [11] M. McGuire, “Computer graphics archive,” August 2011, <http://graphics.cs.williams.edu/data>. [Online]. Available: <http://graphics.cs.williams.edu/data>
- [12] K. Mitchell, “Volumetric light scattering as a post-process,” in *GPU Gems 3*, H. Nguyen, Ed. Addison-Wesley, 2008, pp. 275–285.
- [13] N. I. of Standards and Technology, “Codata recommended values,” 2014. [Online]. Available: <http://physics.nist.gov/cgi-bin/cuu/Value?re>
- [14] C. Peters, C. Münstermann, N. Wetzstein, and R. Klein, “Beyond hard shadows: Moment shadow maps for single scattering, soft shadows and translucent occluders,” in *Proceedings of the 20th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ser. i3D ’16. New York, NY, USA: ACM, 2016, pp. 159–170. [Online]. Available: <http://dx.doi.org/10.1145/2856400.2856402>
- [15] B. T. Phong, “Illumination for computer generated pictures,” *Commun. ACM*, vol. 18, no. 6, pp. 311–317, Jun. 1975. [Online]. Available: <http://doi.acm.org/10.1145/360825.360839>
- [16] A. J. Preetham, P. Shirley, and B. Smits, “A practical analytic model for daylight,” in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 91–100.
- [17] T. Sousa, “Crysis next gen effects,” 2008. [Online]. Available: <http://www.gdcvault.com/play/247/CRYYSIS-Next-Gen>
- [18] B. Toth and T. Umenhoffer, “Real-time Volumetric Lighting in Participating Media,” in *Eurographics 2009 - Short Papers*, P. Alliez and M. Magnor, Eds. The Eurographics Association, 2009.
- [19] L. Williams, “Casting curved shadows on curved surfaces,” in *ACM Siggraph Computer Graphics*, vol. 12, no. 3. ACM, 1978, pp. 270–274.

### **Erklärung**

Ich, Marius Kircher, Matrikelnummer 846949, erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....   
Marius Kircher