UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Path Tracing und dessen Optimierung durch Sampling

Bachelorarbeit

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

vorgelegt von

Jonas Honsdorf

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Gerrit Lochmann, M.Sc. (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im November 2013

Erklärung

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

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Bachelorarbeit
Jonas Honsdorf
(Mat. Nr. 209 210 132)

Thema: Path Tracing und dessen Optimierung durch Sampling

Die Simulation globaler Beleuchtung ist ein Verfahren, das vor allem in der photorealistischen Computergrafik eine große Rolle spielt. Path Tracing ist in der Lage, mit Hilfe von Zufallsexperimenten die Rendergleichung zu lösen. Dadurch können physikalisch korrekte Bilder generiert werden.

Ansatz dieser Arbeit ist die Implementierung und Optimierung von Path Tracing. Ziel der Arbeit ist es, verschiedene Samplingstrategien miteinander zu vergleichen und Effekte wie beispielsweise Depth of Field zu visualisieren.

Insbesondere soll die Frage beantwortet werden, ob sich durch Sampling qualitativ bessere Ergebnisse erzielen lassen.

Die inhaltlichen Schwerpunkte der Arbeit sind:

1. Recherche über Path Tracing, Monte-Carlo Integration und Sampling
2. Implementierung von Path Tracing
3. Optimierung durch Sampling
4. Vergleich der Samplingstrategien
5. Bewertung der Ergebnisse

Koblenz, 03.06.2013

Jonas Honsdorf

S. Müller

– Prof. Dr. Stefan Müller –

Zusammenfassung

Die vorliegende Arbeit stellt Path Tracing zum Rendern von Bildern mit globaler Beleuchtung vor. Durch die Berechnung der Rendergleichung, mithilfe von Zufallsexperimenten, ist das Verfahren physikalisch plausibel. Entscheidend für die Qualität der Ergebnisse ist Sampling. Der Schwerpunkt der Arbeit ist die Untersuchung verschiedener Samplingstrategien. Dazu werden die Ergebnisse unterschiedlicher Dichtefunktionen verglichen und die Methoden bewertet. Außerdem werden Effekte, wie beispielsweise Depth of Field, mittels Sampling visualisiert.

Abstract

The present work introduce Path Tracing as an algorithm to render pictures with global illumination. The rendering equation ist solved with random experiments. The procedure creates images in a physically correct way. Important for high quality results is sampling. In this thesis the focus is to research about various sampling strategies. Therefore the achievements of different density functions will be compared and evaluated to each other. In addition this work present effects such as depth of field.

Inhaltsverzeichnis

1 Einleitung

Einige Zeit hat es gedauert, bis die Menschen das Phänomen Licht verstanden haben. Die Griechen entwickelten früher eine Theorie zu ihrer Vorstellung vom Sehen. Sie glaubten, ihr Auge sende Strahlen aus, die Objekte in der Umgebung abtasten. Dadurch erfährt das Auge Tiefe und Farbe der Umwelt. Heutzutage weiß man, jedes Bild im menschlichen Auge entsteht durch das Auftreffen von Licht auf der Netzhaut. Welche Farbe zum Vorschein kommt, wird durch die eingefangene Wellenlänge bestimmt. Kurzwelliges Licht erscheint uns blau, langwelliges rot. Treten alle Wellenlängen als Gemisch auf, empfinden wir es als weiß. Ein Beispiel dafür ist das Sonnenlicht. Trifft pures Sonnenlicht auf die Erde, so wird es mehrfach reflektiert. Dadurch werden Bereiche, die nicht im direkten Blickkontakt zum Licht stehen, trotzdem indirekt beleuchtet. Diese indirekte Beleuchtung ist ein Grundprizip von globaler Beleuchtung.
Die Simulation globaler Beleuchtung ist ein Verfahren, das vor allem in der photorealistischen Computergraphik eine große Rolle spielt. Ziel ist es, realistische Bilder zu erzeugen, welche aus der physikalisch korrekten Simulation von Licht in einer Szene hervorgehen. Algorithmen wie Path Tracing [Kaj86] oder Photon Mapping [Jen01] sind globale Renderingverfahren und Erweiterungen von klassischem Raytracing [Gla89]. Mit ihnen lassen sich photorealistische Bilder mithilfe stochastischer Methoden berechnen. Diese physikalisch basierten Renderingverfahren beruhen auf dem Versuch, die Rendergleichung von James Kajiya [Kaj86] zu lösen oder vielmehr anzunähern. Dadurch lassen sich aus einer gegebenen mathematischen Beschreibung einer Szene realistisch wirkende Bilder wie in Abbildung1[1] erzeugen.

Abbildung 1: Heutzutage sehen gerenderte Bilder erstaunlich realistisch aus. Ein Grund dafür ist die globale Beleuchtung.

[1] Grafiken aus Renderengine von Peter Kutz, abgerufen am 29. 11. 2013, auf http://www.peterkutz.com/.

2 Motivation

Photorealistisches Rendern von Bildern ist heutzutage in verschiedensten Bereichen im Einsatz. Ob Film und Visual Effects, Architektur oder in der Automobilindustrie, überall spielen physikalisch basierte Renderingverfahren eine Rolle. Die visuelle Realität war schon immer eine Motivation für Forschung in dem Gebiet und ist ein Verkaufsargument für viele graphik-basierte Anwendungen. Es ist zu erwarten, dass der Trend von kommerziell erfolgreichen Produkten in den nächsten Jahren andauert und die Generierung von photorealistischen Bildern eines der wichtigsten Gebiete im Bereich Rendering bleibt. [DBBS06]
Ansatz dieser Arbeit ist daher die Implementierung von Path Tracing. Neben dem globalen Beleuchtungsmodell liegt der Fokus auf dem Sampling. Zu diesem Zweck soll gezeigt werden, welche Vorteile die Wahl der jeweiligen Samplingstrategie hat. Zusätzlich werden Effekte wie beispielsweise Depth of Field vorgestellt, welche mithilfe von Sampling visualisiert werden können. Einen Überblick über die behandelten Bereiche gibt Tabelle 1. Die Verbesserungen der Ergebnisse gegenüber naivem Path Tracing werden verglichen und die Wahl der Strategie bewertet.

Sampling	**Nutzen**
Lichtquelle	Direkte Beleuchtung - Soft Shadows
Hemisphäre	Indirekte Beleuchtung
BRDF	Glossy Reflections & Refractions
Pixel	Anti-Aliasing
Blendenöffnung	Depth of Field
Zeit	Motion Blur

Tabelle 1: Übersicht Sampling.

3 Grundlagen

Path Tracing ist ein stochastisches Raytracing Verfahren, basierend auf dem Versuch, die Rendergleichung zu lösen. Um den Wert der Gleichung zu schätzen, wird die Monte-Carlo-Integration verwendet. Grundlage dafür ist das Sampling. Im Folgenden werden die Begriffe eingeführt.

3.1 Sampling

Sampling ist ein Begriff aus der Stochastik und basiert auf Zufallsexperimenten. In der Wahrscheinlichkeitsrechnung fasst man Messwerte als eine Ergebnismenge Ω zusammen. Beim zweimaligen Werfen eines Würfels, erhält man beispielsweise alle Paarungen der Augenzahlen:

$$\Omega = \{(1,1),(1,2),(2,1),...,(6,6)\}$$

Mit Mitteln der Statistik kann die Verteilung der Wahrscheinlichkeit eines Versuchsausgangs mathematisch angegeben werden. Zufällig bestimmte Messgrößen werden Zufallsvariablen X genannt. Die Zufallsvariable des Würfelexperiments ist eine Abbildung, die jedem möglichen Versuchsausgang eine reelle Zahl zuordnet. Eine mögliche Zufallsvariable wäre die Augensumme beider Würfe:

$$X(10) = \{(5,5),(6,4),(4,6)\} = 3$$

Jedem der Ereignisse kann eindeutig eine Wahrscheinlichkeit zugeordnet werden, mit dem es eintritt. Die aufgetragenen Wahrscheinlichkeiten in Tabelle 2 können durch Abzählen bestimmt werden:

$$P(X = 10) = \frac{3}{36}$$

(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)
(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)
(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)
(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	**(4, 6)**
(5, 1)	(5, 2)	(5, 3)	(5, 4)	**(5, 5)**	(5, 6)
(6, 1)	(6, 2)	(6, 3)	**(6, 4)**	(6, 5)	(6, 6)

Tabelle 2: Ereignisse beim zweifachen Würfeln.

Um einen Bereich zu bestimmen, in dem X liegt, wird die *Verteilungsfunktion* $F(x)$ verwendet:

$$F(x) = P(X \leq x), x \in \Re$$

Es lassen sich auch die Wahrscheinlichkeiten für beliebige Intervalle $(a, b]$ von X bestimmen:

$$P(a < X \leq b) = F(b) - F(a)$$

Abbildung 2 zeigt die Verteilungsfunktion des Würfelexperiments. Sie ist definiert im Bereich $(-\infty, \infty)$ und nimmt Werte zwischen 0 und 1 an. Die Wahrscheinlichkeit, dass ein Ereignis eintritt, ist immer positiv. Ein Ereignis hat die Wahrscheinlichkeit 1, wenn es eingetroffen ist, 0 wenn nie.

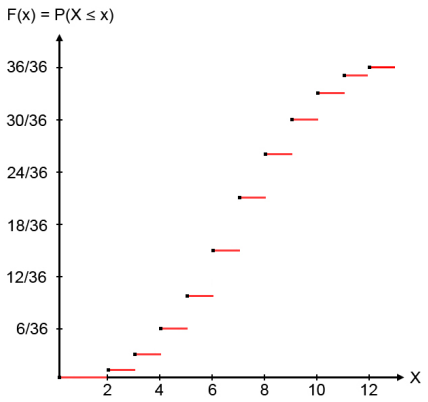


Abbildung 2: Verteilungsfunktion des Würfelexperiments.

Das Würfelexperiment besitzt diskrete Wahrscheinlichkeiten. Jedem Ereignis konnte eine eindeutige Wahrscheinlichkeit zugeordnet werden. Bei stetigen Verteilungen ist das anders. Eine Zufallsvariable heißt stetig, wenn sie unendlich viele Werte annehmen kann. Die Wahrscheinlichkeit für ein bestimmtes Ereignis ist Null. Bei stetig verteilten Zufallsvariablen ist daher die Verteilungsfunktion erforderlich. Zu diesem Zweck wird ein neuer Begriff eingeführt, der auf die Verteilungsfunktion zurückgreift.
Die *Dichtefunktion* $p(x)$ beschreibt eine stetige Wahrscheinlichkeitsverteilung. Sie besitzt nur positive Werte, kann aber im Gegensatz zu Wahrscheinlichkeiten beliebig große Werte annehmen. Die Fläche unter einer Dichtefunktion besitzt den Inhalt 1:

$$\int_{-\infty}^{\infty} p(x)dx = 1$$

Die Verteilungsfunktion bei stetig verteilten Zufallsvariablen ergibt sich durch Integration der Dichtefunktion im Bereich von $-\infty$ bis zu einem beliebigen Wert x:

$$F(x) = \int_{-\infty}^{x} p(x')dx' \tag{1}$$

Ziel ist es nun, Zufallsvariablen nach einer beliebigen Dichte zu erzeugen. Diese Werte werden *Samples* genannt. Sie sind die Grundlage von jedem stochastischen Raytracer und werden mit einem Zufallsgenerator erstellt.

In Abbildung 3 wurden auf der x-Achse Samples mit der Dichtefunktion $p(x) = x^2$ generiert. Die y-Achse enthält gleichverteilte Zufallswerte im Bereich $[0, 1]$. Dort, wo die Dichtefunktion hohe Werte annimmt, werden Samples mit höherer Wahrscheinlichkeit generiert.

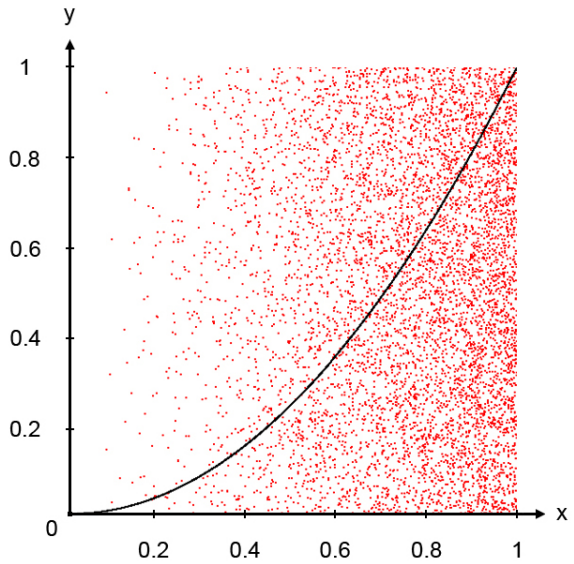


Abbildung 3: Verteilung der Dichte $p(x) = x^2, x \in [0, 1]$.

Die Zufallsvariablen ergeben sich mittels inverser CDF-Methode, wobei CDF im Englischen die Verteilungsfunktion ist (engl. *cumulative distribution function*). Unter der Annahme, dass X eine Zufallsvariable mit Eigenschaften einer Dichte p und eine Verteilung F besitzt, liefert $F(X)$ eine Zufallsvariable im Bereich $[0, 1]$. Solche Zufallsvariablen werden als ξ bezeichnet:

$$\xi = F(X)$$

Die Umkehrung ergibt folglich Zufallsvariablen, die nach Dichte p verteilt sind:

$$X = F^{-1}(\xi)$$

Gegeben ist das Beispiel aus Abbildung 3 mit Dichte $p(x) = x^2, x \in [0, 1]$. Das Vorgehen ist wie folgt: Die Verteilungsfunktion ergibt sich durch Inte-

gration der unteren Grenze bis x:

$$\begin{aligned}\xi = F(x) &= \int_0^x p(x')dx' \\ &= \int_0^x x'^2 dx' \\ &= \left[\frac{x'^3}{3}\right]_0^x \\ &= \frac{x^3}{3}\end{aligned}$$

Umstellen nach x liefert eine Zufallsvariable mit Dichte p und Verteilung F:

$$\xi = \frac{x^3}{3} \Rightarrow x = \sqrt[3]{3\xi}, \xi \in [0,1]$$

In C++ lassen sich Zufallswerte mit der Funktion $rand()$ erzeugen. Teilt man das Ergebnis durch RAND_MAX, erhält man eine $[0,1]$-verteilte Zufallsvariable.
Das Beispiel der Dichte $p(x) = x^2$ benötigte 1-dimensionale Zufallsvariablen. Um beispielsweise auf einer Kreisscheibe gleichverteilte Samples zu erzeugen, werden 2-dimensionale Zufallsvariablen benötigt. Aus Gleichung 1 folgt:

$$F(x,y) = \int_{-\infty}^{y}\int_{-\infty}^{x} p(x',y')dx'dy' \tag{2}$$

Mithilfe der Umrechnung eines Flächenelements in Polarkoordinaten,

$$dA = dx \cdot dy = r \cdot dr \cdot d\varphi,$$

erhält man die Verteilungsfunktion einer Kreisscheibe [AHJ+01]:

$$F(r,\varphi) = \int_0^{\varphi}\int_0^{r} p(r',\varphi')r' \cdot dr' \cdot d\varphi'$$

Die Samples müssen diesmal anhand einer 2-dimensionalen Dichte errechnet werden. Dabei stehen zwei $[0,1]$-verteilte Zufallsvariablen ξ_1 und ξ_2 zur Verfügung. Diese ergeben sich aus den Bedingungen [AHJ+01]:

$$\begin{aligned}\xi_1 &= F(r,\varphi_{max}) \\ \xi_2 &= \frac{F(\xi_1,\varphi)}{F(\xi_1,\varphi_{max})}\end{aligned}$$

Die gleichverteilte Dichte der Kreisscheibe ergibt sich aus der Invertierung der Kreisfläche [Kir92]:

$$p(r,\varphi) = \frac{1}{\pi \cdot R^2}, r \in [0,R], \varphi \in [0,2\pi]$$

Die Verteilungsfunktion für dieses Beispiel lautet somit:

$$\begin{aligned} F(r,\varphi) &= \int_0^{\varphi}\int_0^{r} p(r',\varphi')\cdot r'\cdot dr'\cdot d\varphi' \\ &= \int_0^{\varphi}\int_0^{r} \frac{r'}{\pi\cdot R^2}\cdot dr'\cdot d\varphi' \\ &= \int_0^{\varphi} \frac{r^2}{2\pi\cdot R^2}\cdot d\varphi' \\ &= \frac{r^2\cdot\varphi}{2\pi\cdot R^2} \end{aligned}$$

ξ_1 und ξ_2 ergeben sich als Nächstes durch Einsetzen in die Bedingungen:

$$\begin{aligned} \xi_1 &= F(r,\varphi_{max}) = F(r,2\pi) = \frac{r^2\cdot 2\pi}{2\pi\cdot R^2} = \frac{r^2}{R^2} \\ \xi_2 &= \frac{F(\xi_1,\varphi)}{F(\xi_1,\varphi_{max})} = \frac{\xi_1^2\cdot\varphi}{2\pi\cdot R^2}\cdot\frac{2\pi\cdot R^2}{\xi_1^2\cdot 2\pi} = \frac{\varphi}{2\pi} \end{aligned}$$

Durch die Umstellung nach r und φ können Samples auf einer Kreisscheibe generiert werden:

$$\begin{bmatrix} r \\ \varphi \end{bmatrix} = \begin{bmatrix} R\sqrt{\xi_1} \\ 2\pi\cdot\xi_2 \end{bmatrix} \tag{3}$$

Als letzten Schritt müssen die Polarkoordinaten in kartesische Koordinaten umgerechnet werden:

$$\begin{aligned} x &= r\cos\varphi \\ y &= r\sin\varphi \end{aligned}$$

Abbildung 4 zeigt einen Vergleich von Sampling einer Kreisscheibe mit gleichverteilten Zufallszahlen und nach gegebener Dichte $p(r,\varphi) = \frac{1}{\pi\cdot R^2}$.

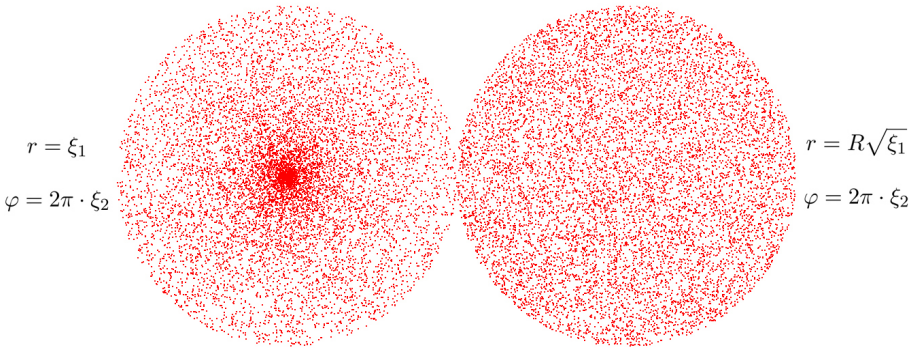


Abbildung 4: Links: Gleichverteilte Zufallszahlen. Rechts: Zufallszahlen mit Dichte $p(r,\varphi) = \frac{1}{\pi\cdot R^2}$.

Ein weiteres Beispiel für mehrdimensionale Zufallsvariablen ist das Sampling einer Kugel. Als Dichtefunktion wird die Oberfläche einer Kugel invertiert [Kir92]:

$$p(\theta, \varphi) = \frac{1}{4\pi r^2}$$

Das Flächenelement einer Kugel mit Radius 1 ist gegeben [DBBS06] durch:

$$dA = \sin\theta \cdot d\theta \cdot d\varphi$$

Eingesetzt in die Verteilungsfunktion ergibt:

$$\begin{aligned} F(\theta, \varphi) &= \int_0^\theta \int_0^\varphi p(\theta', \varphi') \sin\theta' \cdot d\theta' \cdot d\varphi' \\ &= \int_0^\theta \int_0^\varphi \frac{1}{4\pi} \sin\theta' \cdot d\theta' \cdot d\varphi' \\ &= \int_0^\theta \frac{1}{4\pi} \cdot \varphi \cdot \sin\theta' \cdot d\theta' \\ &= \frac{\varphi \cdot (1 - \cos\theta)}{4\pi} \end{aligned}$$

ξ_1 und ξ_2 erhält man wieder durch Einsetzen in die Bedingungen:

$$\begin{aligned} \xi_1 &= F(r, \varphi_{max}) = F(r, 2\pi) = \frac{1 - cos\theta}{2} \\ \xi_2 &= \frac{F(\xi_1, \varphi)}{F(\xi_1, \varphi_{max})} = \frac{\varphi(1 - \cos\xi_1)}{4\pi} \cdot \frac{4\pi}{2\pi(1 - \cos\xi_1)} = \frac{\varphi}{2\pi} \end{aligned}$$

Durch die Invertierung der Verteilungsfunktion ergeben sich die sphärischen Winkel θ und φ:

$$\begin{bmatrix} \theta \\ \varphi \end{bmatrix} = \begin{bmatrix} \arccos(1 - 2\xi_1) \\ 2\pi\xi_2 \end{bmatrix} \tag{4}$$

Die Kugelkoordinaten werden als letzten Schritt auf die entsprechenden kartesischen Koordinaten abgebildet:

$$\begin{aligned} x &= r \cdot \sin\theta \cdot \cos\varphi \\ y &= r \cdot \sin\theta \cdot \sin\varphi \\ z &= r \cdot \cos\theta \end{aligned}$$

Abbildung 5 veranschaulicht das Sampling einer Kugel mit gegebener Dichte $p(\theta, \varphi) = \frac{1}{4\pi r^2}$ und Radius 1. Die Kugel wurde gleichmäßig abgetastet.

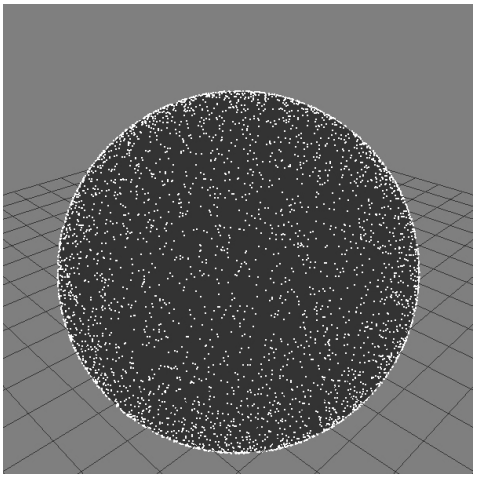


Abbildung 5: Gleichmäßige Abtastung einer Kugel mit Dichte $p(\theta, \varphi) = \frac{1}{4\pi r^2}$ und Radius 1.

3.2 Monte-Carlo-Integration

Sampling ist die Grundlage bei Path Tracing um beispielsweise Lichtquellen oder die Hemisphäre abzutasten. Nächster Ansatz ist das Lösen von Integralen mittels Sampling.
Der *Erwartungswert* $E(X)$ einer Zufallsvariablen beschreibt das im Mittel angenommene Ergebnis von X. Bei stetig verteilten Zufallsvariablen mit einer Dichtefunktion $p(x)$ ist der Erwartungswert definiert [DH04] als:

$$E(X) = \int_{-\infty}^{\infty} x \cdot p(x) dx$$

Jeder annehmbare Wert von x wird demzufolge mit seiner Wahrscheinlichkeitsdichte gewichtet. Das Beispiel der Dichte $p(x) = x^2$, mit $x \in [0, 1]$, ergibt folgenden Erwartungswert:

$$\begin{aligned} E(X) &= \int_0^1 x \cdot x^2 dx = \int_0^1 x^3 dx \\ &= \left[\frac{x^4}{4}\right]_0^1 = \frac{1}{4} \end{aligned}$$

Eine andere Herangehensweise ist ein Zufallsexperiment N-mal durchzuführen. Wenn X eine Zufallsvariable mit Dichtefunktion $p(x)$ ist, dann lässt sich ihr Erwartungswert annähern [SWZ96] durch:

$$E(X) = E(h(x)) = \int_a^b h(x)p(x)dx \approx \frac{1}{N} \sum_{i=1}^{N} h(x_i)$$

Dabei werden Zufallszahlen x_i mit der Dichte p generiert und gemittelt. Die Umformung $f(x) = h(x) \cdot p(x)$ ergibt folgende Darstellung:

$$\int_a^b f(x)dx \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \tag{5}$$

Folglich lässt sich ein bestimmtes Integral mit Zufallsvariablen x_i, mit einer beliebigen Dichtefunktion berechnen. Diese Methode wird *Monte-Carlo-Integration* genannt.
Aus vorherigem Beispiel ist die Lösung des Integrals

$$\int_0^1 x^3 dx = \frac{1}{4}$$

bekannt. Nun soll es mit einem Zufallsexperiment gelöst werden. Bei einer konstanten Dichte $p(x) = 1$ gilt:

$$\xi = \int_0^x p(x')dx' = \int_0^x 1dx' = x$$

Für beispielsweise $N = 3$ werden drei gleichverteilte Zufallsvariablen $\xi_i = \{0.166, 0.5, 0.833\}$ erzeugt. Aus Gleichung 5 folgt:

$$\int_0^1 x^3 dx \approx \frac{1}{3} \sum_{i=1}^{3} \frac{x_i^3}{1} = \frac{1}{3}(0.166^3 + 0.5^3 + 0.833^3) = 0.235$$

Die Abweichung zur Lösung beträgt 0.015. Je mehr Samples bei der Monte-Carlo-Methode einfließen, desto weniger weicht das Ergebnis dem genauen Wert des Integrals ab. Laut dem Gesetz der großen Zahlen lässt sich das auch beweisen [AHJ+01]. Wenn die Summe gegen unendlich geht, konvergiert das Ergebnis zum Erwartungswert:

$$P\left(\left|\frac{1}{N} \sum_{i=1}^{N\to\infty} x_i - E(X)\right| \leq \epsilon\right) \to 1$$

Das Zufallsexperiment ist somit für hinreichend große N eine erwartungstreue Schätzung.
Wie stark die Werte x_i im Durchschnitt vom Erwartungswert abweichen, wird durch die *Varianz* s^2 und die *Standardabweichung* σ beschrieben, wobei die Varianz die quadratische Abweichung angibt [Geo09]:

$$\begin{aligned} s^2(X) &= \frac{1}{N} \sum_{i=1}^{N} (x_i - E(X))^2 \\ \sigma &= \sqrt{s^2} \end{aligned}$$

Bei der Varianz wird durch das Quadrieren erreicht, dass sich positive und negative Werte nicht gegenseitig aufheben. Die Standardabweichung bei der Monte-Carlo-Integration ist:

$$\sigma \approx \frac{1}{\sqrt{N}}$$

Wählt man zum Beispiel 25, 100 und 400 Samples, beträgt der durchschnittliche Fehler:

$$\begin{aligned} \sigma &= \frac{1}{\sqrt{25}} = 0,2 \\ \sigma &= \frac{1}{\sqrt{100}} = 0,1 \\ \sigma &= \frac{1}{\sqrt{400}} = 0,05 \end{aligned}$$

Das bedeutet, es müssen viermal so viele Samples erzeugt werden, damit sich der Fehler halbiert.
Die Wahl der Dichtefunktion ist beliebig, hat jedoch Einfluss wie schnell das Zufallsexperiment zum Erwartungswert konvergiert und demzufolge auf die Anzahl der benötigten Samples. Wählt man für das vorherige Beispiel die Dichte $p(x) = 2x$, dann ergeben sich die Zufallszahlen wie folgt:

$$\xi = \int_0^x p(x')dx' = \int_0^x 2x \cdot dx' = x^2 \Rightarrow x = \sqrt{\xi}$$

Die selben gleichverteilten Zufallszahlen $\xi_i = \{0.166, 0.5, 0.833\}$ ergeben die nach p verteilten Samples $x_i = \{0.407, 0.707, 0.913\}$. Einsetzen in Gleichung 5 mit $N = 3$ ergibt:

$$\int_0^1 x^3 dx \approx \frac{1}{3}\sum_{i=1}^{3}\frac{x_i^3}{2x_i} = \frac{1}{3}\sum_{i=1}^{3}\frac{x_i^2}{2} = \frac{1}{6}(0.407^2 + 0.707^2 + 0.913^2) = 0.249$$

Die Abweichung zur korrekten Lösung beträgt nur noch 0.001. Die Wahl der Dichte ist daher entscheidend für die Effizienz der Monte-Carlo-Integration[2].
Die Wahl der Dichte ist schwierig und von Integral zu Integral verschieden. Beim *Importance Sampling* [Laf96] wird die Dichte so gewählt, dass sie $f(x)$ ähnlich ist. Das bedeutet, in Bereichen, wo die Funktion f hohe Werte besitzt, werden mehr Samples generiert. Später wird anhand von Beispielen gezeigt, welche Vorteile Importance Sampling hat.

[2]Für einen Vergleich der beiden Schätzungen, siehe Anhang B, Abbildung 43.

4 Globale Beleuchtung

Das herkömmliche Raytracing Verfahren ermöglicht keine globalen Beleuchtungseffekte. Das lokale Beleuchtungsmodell ignoriert die indirekte Lichteinwirkung. Diese wird mit einer ambienten Konstante angenähert und so das Bild aufgehellt. Dieses ambiente Licht hat nichts mit den Lichtquellen zu tun und hat auch keine Entsprechung in der Realität. Es ist ein Trick um die Szene aufzuhellen. Um physikalisch korrekte Bilder zu rendern, werden globale Beleuchtungsmodelle benötigt. Die indirekte Lichtreflexion spielt dabei eine wichtige Rolle. Im Folgenden wird das Reflexionsverhalten von Materialien erklärt und im Anschluss die Rendergleichung eingeführt.

4.1 Reflexionsverhalten

Objekte werden überhaupt erst sichtbar, weil sie Licht ins Auge reflektieren. Man unterscheidet zwischen zwei Grundformen von Reflexionen. Wenn das Licht auf kleine Unregelmäßigkeiten der Oberfläche trifft, erscheint es als matt. Dieses Phänomen wird als diffuse Reflexion bezeichnet. Gemäß dem Lambertschen Gesetz [PH10] hängt die Stärke der Reflexion vom Einfallswinkel des Lichtes ab, jedoch nicht vom Betrachterstandort. Beispiele für solche Materialien sind Papier oder Kreide. Das Licht auf diffusen Oberflächen wird gleichmäßig in alle Richtungen gestreut, wie Abbildung 6 in 2D zeigt.

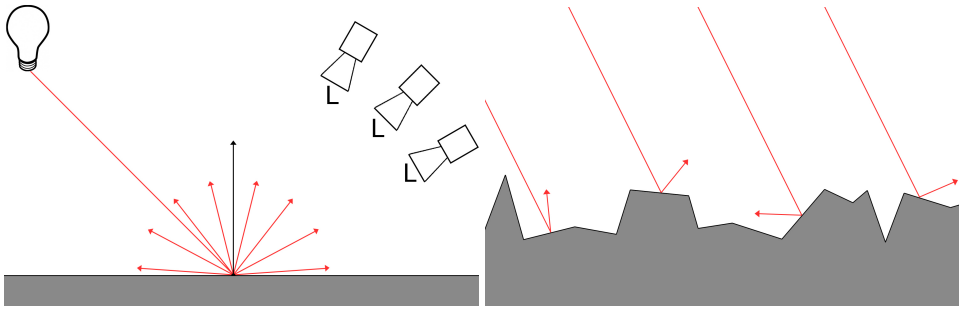


Abbildung 6: Die Helligkeit diffuser Flächen ist unabhängig vom Betrachter. Die Oberfläche besteht aus vielen kleinen Spiegeln. Die reflektierte Richtung ist Zufall.

Hingegen wirkt die Oberfläche bei der spekularen Reflexion als glänzend. Das liegt daran, dass das Licht im Idealfall nur in eine einzige Richtung reflektiert wird. Je rauer das Material ist, desto breiter wird das Licht in diese Richtung gefächert (vgl. Abbildung 7). Besteht die Oberfläche des Materials wie in Abbildung 8 aus einem perfekten Spiegel, wird das Licht nur in eine einzige Richtung reflektiert. Die Oberfläche solcher Materialien

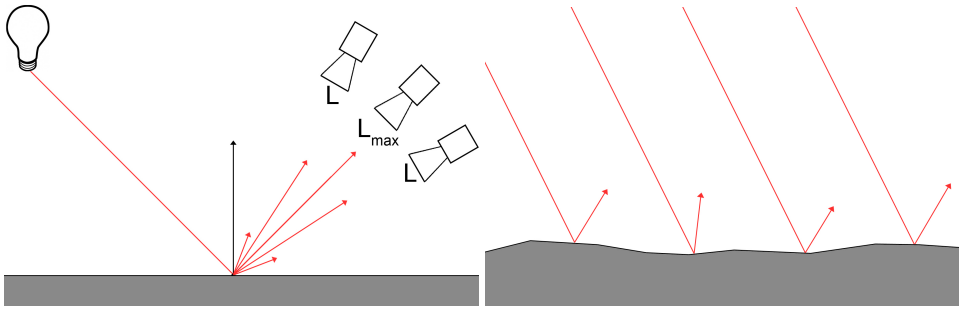


Abbildung 7: Das eintreffende Licht wird an der Oberfläche gespiegelt. Je rauer das Material, desto breiter wird das Licht gefächert. Die reflektierte Richtung besitzt eine Vorzugsrichtung. Ergebnis ist ein verschwommenes Spiegelbild.

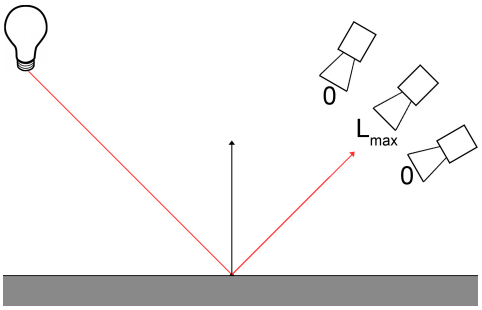


Abbildung 8: Bei einem perfekten Spiegel ist die Einfallsrichtung gleich der Ausfallsrichtung. In alle anderen Richtungen wird kein Licht reflektiert.

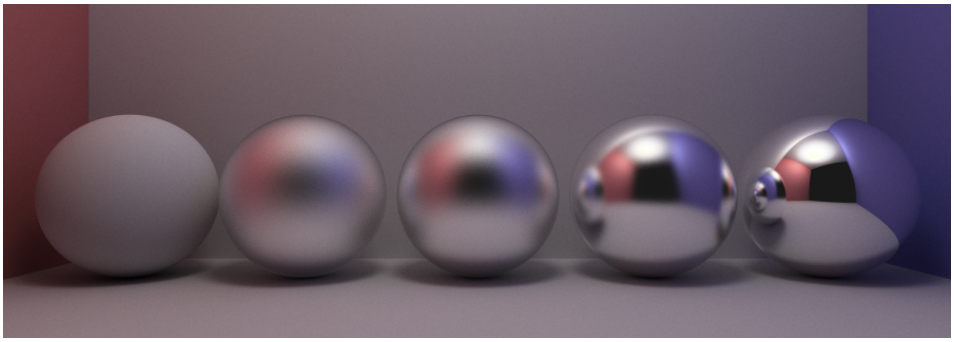


Abbildung 9: Unterschiedliche Charakteristiken der Reflexionen von diffus bis spekular.

besitzt keine Unregelmäßigkeiten und ist glatt. Die unterschiedlichen Charakteristiken von Reflexionen ergeben in einem Renderer die in Abbildung 9 zu sehenden Materialien.
In der Realität sind Kombinationen von diffuser und spekularer Reflexion möglich. Wenn Materialien gerendert werden, muss das Reflexionsverhalten bekannt sein. Die *Bidirektionale Reflexionsverteilungsfunktion* (BRDF) gibt das Verhältnis von reflektiertem zu einfallendem Licht eines Materials an. Die BRDF ist abhängig von dem Einfallswinkel des Lichts $\vec{\omega}_i$ und der betrachteten Reflexionsrichtung $\vec{\omega}_r$. Sind diese vorgegeben, kann das reflektierte Licht L_r an einem Oberflächenpunkt x in Richtung $\vec{\omega}_r$ berechnet werden:

$$L_r(x, \vec{\omega}_r) = f_r(x, \vec{\omega}_i, \vec{\omega}_r) \cdot L_i(x, \vec{\omega}_i) \cdot \cos\theta$$

Die rechte Seite der Gleichung entspricht der BRDF f_r multipliziert mit dem einfallenden Licht $L_i(x, \vec{\omega}_i)$ und dem Kosinus des Einfallswinkels θ. Um jedes einfallende Licht auf der Oberfläche zu berücksichtigen, wird das Integral über alle Einfallsrichtungen oberhalb von x gebildet und man erhält die Reflexionsgleichung:

$$L_r(x, \vec{\omega}_r) = \int_\Omega f_r(x, \vec{\omega}_i, \vec{\omega}_r) \cdot L_i(x, \vec{\omega}_i) \cdot \cos\theta \cdot d\vec{\omega}_i \tag{6}$$

4.2 Rendergleichung

Ziel einer Renderengine ist die Darstellung eines Bildes, wahrgenommen durch eine virtuelle Kamera. Dafür ist die Energie erforderlich, welche die Kamera, bzw. einen Pixel erreicht. Die Rendergleichung beschreibt die komplette Lichtausbreitung in einer Szene und ist Grundlage um photorealistische Bilder zu erzeugen:

$$L_o(x, \vec{\omega}_r) = L_e(x, \vec{\omega}_r) + \int_\Omega f_r(x, \vec{\omega}_i, \vec{\omega}_r) \cdot L_i(x, \vec{\omega}_i) \cdot \cos\theta \cdot d\vec{\omega}_i \tag{7}$$

Sie wurde 1986 von James Kajiya erstmals in [Kaj86] vorgestellt. Die Gleichung ermittelt den Energiefluss L_o in Richtung $\vec{\omega}_r$, ausgehend von einem Oberflächenpunkt x. Genau genommen ist der Punkt x ein winzig kleines Flächenelement. Ein Flächenelement besitzt im Gegensatz zu einem Punkt eine Normale und ist notwendig für die weitere Berechnung. L_e ist das von der Fläche selbst emittierte Licht in Richtung $\vec{\omega}_r$, vorausgesetzt es handelt sich bei ihr selbst um eine Lichtquelle. Das Integral ist die Reflexionsgleichung aus Gleichung 6. Vereinfacht lässt sich die Formel schreiben als:

$$L_o(x, \vec{\omega}_r) = L_e(x, \vec{\omega}_r) + L_r(x, \vec{\omega}_r)$$

Das insgesamt ausgestrahlte Licht L_o von einem Oberflächenpunkt x in eine Richtung $\vec{\omega}_r$, setzt sich zusammen aus dem von der Fläche selbst emittierten Licht L_e und dem an dieser Fläche reflektierten Licht L_r.

Die Rendergleichung ist eine rekursive Gleichung. Um $L_i(x, \vec{\omega}_i)$ zu bestimmen, muss die selbe Gleichung erneut angewendet werden. Aus diesem Grund ist die Berechnung des Integrals zu komplex und ihre Lösung analytisch nicht möglich. Dies führt zu einer der am weit verbreitetsten praktischen Methoden das Problem zu lösen. Dabei wird das Licht durch einen Strahl von einer Bildebene aus, in die umgekehrte Richtung der Lichtausbreitung eingesammelt. Entlang des Pfades wird der Strahl von Objekt zu Objekt reflektiert. Das folgende Kapitel beschreibt wie Path Tracing die Rendergleichung mithilfe der Monte-Carlo-Integration löst.

4.3 Path Tracing

Path Tracing ist im Allgemeinen eine Erweiterung zum klassischen Raytracing Algorithmus [Gla89]. Ein Unterschied liegt darin, dass Path Tracing die diffuse Wechselwirkung in einer Szene miteinbezieht und mittels Monte-Carlo-Integration versucht, die Rendergleichung aus Gleichung 7 zu lösen. Um die Intensität des Lichts an einem Punkt x einzufangen, werden Strahlen von einer virtuellen Kamera in die Szene geschickt. Diese werden verfolgt und ermitteln entlang des Pfades die abgegebene Energie. Erreicht ein Strahl eine Lichtquelle oder verlässt er die Szene und trifft den Hintergrund, wird die gesammelte Energie aufsummiert.
Ein weiterer Unterschied zu Raytracing liegt darin, dass an jedem Schnittpunkt nur ein Strahl weiterverfolgt wird. Dafür werden insgesamt N verschiedene Strahlen pro Pixel verschickt und gemittelt. Abbildung 10 verdeutlicht das Vorgehen von Path Tracing.

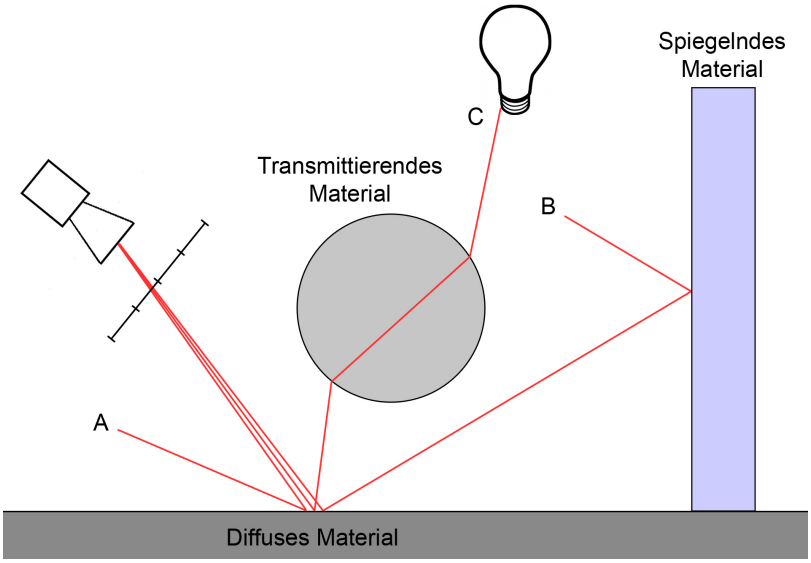
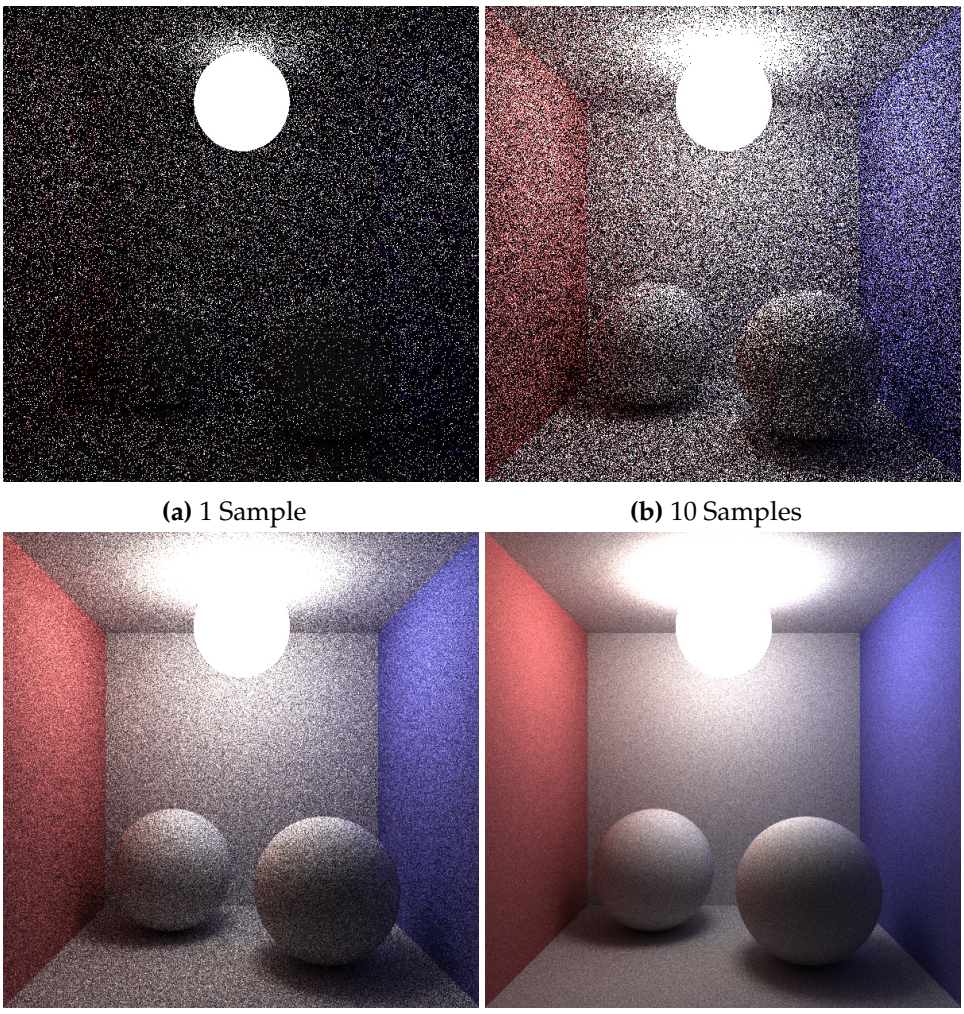


Abbildung 10: Das Beispiel zeigt drei Strahlen durch ein Pixel. Die gesammelte Energie wird gemittelt.

Welchen Pfad der Strahl wählt, entscheidet das Material und somit die BRDF am Schnittpunkt. Bei einem perfekten Spiegel wird der Strahl zum Beispiel nur in eine einzige Richtung weiterverfolgt. Die BRDF würde in alle anderen Richtungen Null ergeben.
Der zuvor beschriebene Algorithmus wurde ursprünglich in [Kaj86] vorgestellt. Zu Beginn ist aus der Rendergleichung einzig die Strahlrichtung $\vec{\omega}_r$ von der Kameraposition in die Szene bekannt. Jeder Kamerastrahl spiegelt einen Pixel des Bildschirms wieder. Der erste Schnittpunkt x muss bestimmt werden. Wie unterschiedliche Objekte geschnitten werden, findet man in [PH10] und [MT97]. Der Emissionsanteil $L_e(x, \vec{\omega}_r)$ wird ausgewertet, falls das geschnittene Objekt am Punkt x eine Lichtquelle darstellt. Das reflektierte Licht L_r kann durch Monte-Carlo-Integration geschätzt werden:

$$\begin{aligned} L_r(x, \vec{\omega}_r) &= \int_\Omega f_r(x, \vec{\omega}_i, \vec{\omega}_r) \cdot L_i(x, \vec{\omega}_i) \cdot \cos\theta \cdot d\vec{\omega}_i \\ &\approx \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(x, \vec{\omega}_i, \vec{\omega}_r) \cdot L_i(x, \vec{\omega}_i) \cdot \cos\theta_i}{p(\vec{\omega}_i)} \end{aligned} \tag{8}$$

Die Dichtefunktion $p(\vec{\omega}_i)$ ist beliebig wählbar. Das Reflexionsverhalten $f_r(x, \vec{\omega}_i, \vec{\omega}_r)$ kann mithilfe der BRDF ausgewertet werden. Der Kosinusterm $\cos\theta_i$ ist der Winkel zwischen einfallender Richtung und Normale am Schnittpunkt. Er ist das Ergebnis des Skalarprodukts von umgedrehter Richtung zum Licht und Normale. Die noch fehlende Funktion ist $L_i(x, \vec{\omega}_i)$. Gelöst wird das Problem durch erneutes Anwenden der Rendergleichung, diesmal für Richtung $\vec{\omega}_i$. Als Ergebnis erhält man einen rekursiven Prozess, der erst endet, sobald eine Lichtquelle oder der Hintergrund getroffen wurde. Zusätzlich sollte in einem Path Tracer ein Abbruchkriterium existieren, da ein Strahl möglicherweise erst nach vielen Indirektionen auf eine Lichtquelle trifft. Die Energie solcher Pfade ist ohnehin sehr schwach und trägt kaum noch zum Ergebnis bei. Daher wird beispielsweise nach vier Indirektionen abgebrochen.
Abbildung 11 zeigt den beschriebenen Algorithmus mit diffusen Materialien in der Cornell-Box Testszene. Problematisch ist die benötigte Anzahl an Samples, um zum Ergebnis zu konvergieren. Trifft ein Pfad die Lichtquelle nach gegebener Anzahl an Indirektionen nicht, dann ist die zurückgegebene Energie gleich Null. Die Wahrscheinlichkeit in einer zufälligen Richtung eine Lichtquelle zu treffen ist gering. Für Punktlichtquellen ist die Wahrscheinlichkeit sogar Null. Dies ist der Grund, weswegen Path Tracing ausschließlich flächige Lichtquellen verwendet. Ein weiteres Merkmal aus Abbildung 11 sind die verrauschten Ergebnisse bei zu geringer Anzahl an Samples. Die Fehler der Näherung des Integrals entsprechen der Varianz und sind Ursache für das Bildrauschen.



(a) 1 Sample **(b)** 10 Samples

(c) 100 Samples **(d)** 1000 Samples

Abbildung 11: Ergebnisse eines naiven Path Tracers mit unterschiedlicher Anzahl an Samples. Je mehr Samples, desto besser das Ergebnis.

4.4 Next Event Estimation

Der folgende Ansatz teilt das Integral der Rendergleichung in direkt und indirekt einfallendes Licht auf [SW92]. Diese Methode wird *Stratified Sampling* genannt. Das Integral wird zerlegt und als Summe der Teilintegrale berechnet:

$$L_r(x, \vec{\omega}_r) = L_{direct} + L_{indirect}$$

Es wird nicht mehr dem Zufall überlassen, ob ein Pfad eine Lichtquelle trifft. Bei jedem Schnittpunkt wird direkt die Oberfläche der Lichtquelle abgetastet. Dieses Vorgehen findet man in der Literatur unter *Next Event Estimation* [DHM+01], [Laf96]. Dazu wird die Rendergleichung abgeändert. Anstatt über alle Raumwinkel zu integrieren, genügt bei der direkten Beleuchtung die Integration über die Oberfläche einer Lichtquelle. Abbildung 12 veranschaulicht die direkte Beziehung zweier Flächen.

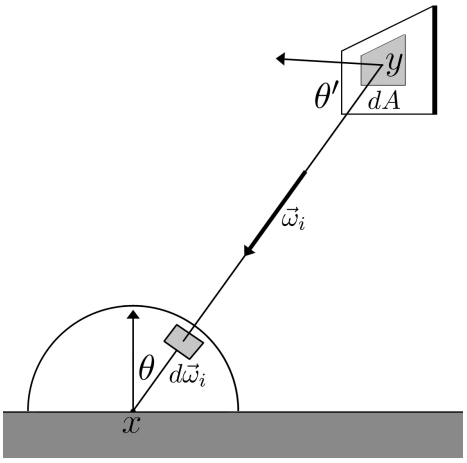


Abbildung 12: Beziehung zweier Flächen.

In [DBBS06] ergibt sich daraus folgende Gleichung:

$$d\vec{\omega} = \frac{dA \cos\theta'}{||\overrightarrow{y - x}||^2} \tag{9}$$

Einsetzen in die Rendergleichung und anpassen der Integrationsbereiche ergibt das Integral über die Fläche einer Lichtquelle:

$$L_r(x, \vec{\omega}_r) = \int_{y \in light} f_r(x, \vec{\omega}_i, \vec{\omega}_r) \cdot L_i(y, \overrightarrow{xy}) \cdot \frac{\cos\theta' \cdot \cos\theta}{||\overrightarrow{y - x}||^2} \cdot dA_y \tag{10}$$

Zusätzlich muss geprüft werden, ob die Punkte x und y sichtbar zueinander sind. Mithilfe einer Visibilitätsfunktion $V(x, y)$ wird dies entschieden.

Diese Funktion ist entweder Null, wenn der Punkt x im Schatten liegt oder Eins falls nicht:

$$\forall(x,y) : V(x,y) = \begin{cases} 1, & \text{x und y sichtbar zueinander} \\ 0, & \text{x und y nicht sichtbar zueinander} \end{cases}$$

In der Literatur [VG95] findet man häufig eine Geometriefunktion $G(x,y)$, die das Verhältnis der Flächen darstellt:

$$G(x,y) = G(y,x) = \frac{\cos\theta' \cdot \cos\theta}{||\overrightarrow{y-x}||^2}$$

Durch Einsetzen in Gleichung 10 folgt die Rendergleichung für eine flächige Lichtquelle:

$$L_r(x,\vec{\omega}_r) = \int_{y \in light} f_r(x,\vec{\omega}_i,\vec{\omega}_r) \cdot L_i(y,\overrightarrow{xy}) \cdot V(x,y) \cdot G(x,y) \cdot dA_y \qquad (11)$$

Für das Sampling der Gleichung 11 wird ein zufälliger Punkt y auf der Oberfläche der Lichtquelle mit einer Dichte $p(y)$ durch Monte-Carlo-Integration gewählt:

$$L_r(x,\vec{\omega}_r) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(x,\vec{\omega}_i,\vec{\omega}_r) \cdot L_i(y,\overrightarrow{xy_i}) \cdot V(x,y_i) \cdot G(x,y_i)}{p(y_i)}.$$

Nun können Lichtquellen bei jedem Schnittpunkt direkt mit beliebiger Dichtefunktion abgetastet werden. Die vollständige Rendergleichung mit direkter und indirekter Beleuchtung ist damit gegeben durch:

$$\begin{aligned} L_o(x,\vec{\omega}_r) &= L_e(x,\vec{\omega}_r) + \\ &\quad \int_{y \in light} f_r(x,\vec{\omega}_i,\vec{\omega}_r) \cdot L_i(y,\overrightarrow{xy}) \cdot V(x,y) \cdot G(x,y) \cdot dA_y + \\ &\quad \int_{\Omega} f_r(x,\vec{\omega}_i,\vec{\omega}_r) \cdot L_i(x,\vec{\omega}_i) \cdot \cos\theta \cdot d\vec{\omega}_i \\ &= L_e(x,\vec{\omega}_r) + L_{direct} + L_{indirect} \end{aligned}$$

Man erhält ein Integral über die Hemisphäre und ein weiteres über die Fläche einer Lichtquelle. Daraus folgen zwei seperate Monte-Carlo-Schätzungen und somit zwei Zufallsexperimente mit jeweils unterschiedlicher Dichtefunktion und Samplingstrategie:

$$\begin{aligned} L_o(x,\vec{\omega}_o) &\approx L_e(x,\vec{\omega}_o) + \\ &\quad \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(x,\vec{\omega}_i,\vec{\omega}_r) \cdot L_i(y,\overrightarrow{xy_i}) \cdot V(x,y_i) \cdot G(x,y_i)}{p(y_i)} + \qquad (12) \\ &\quad \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(x,\vec{\omega}_i,\vec{\omega}_r) \cdot L_i(x,\vec{\omega}_i) \cdot \cos\theta_i}{p(\vec{\omega}_i)} \qquad (13) \end{aligned}$$

Darauf zu achten ist, falls ein indirekter Strahl eine Lichtquelle trifft, dass dieser verworfen wird. Durch die direkte Abtastung des Lichts wurde bereits die Energie der Lichtquelle beigetragen. Abbildung 13 verdeutlicht den Unterschied zwischen direktem und indirektem Sampling.

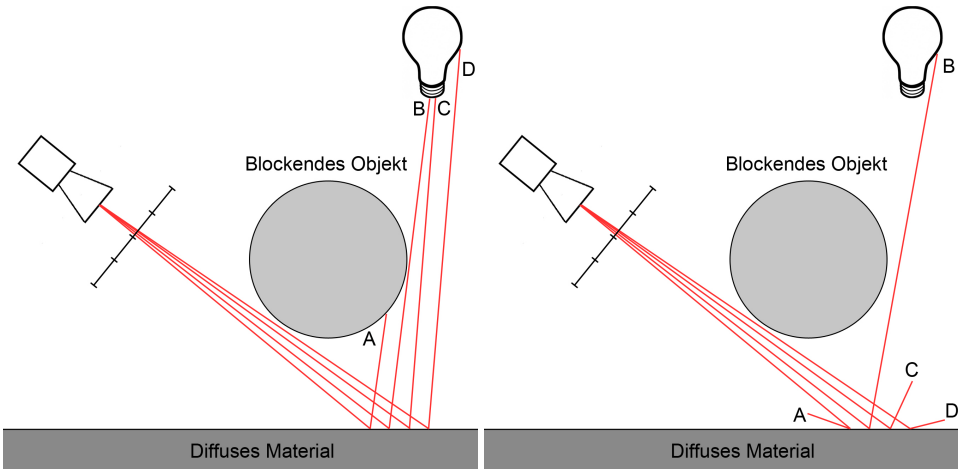
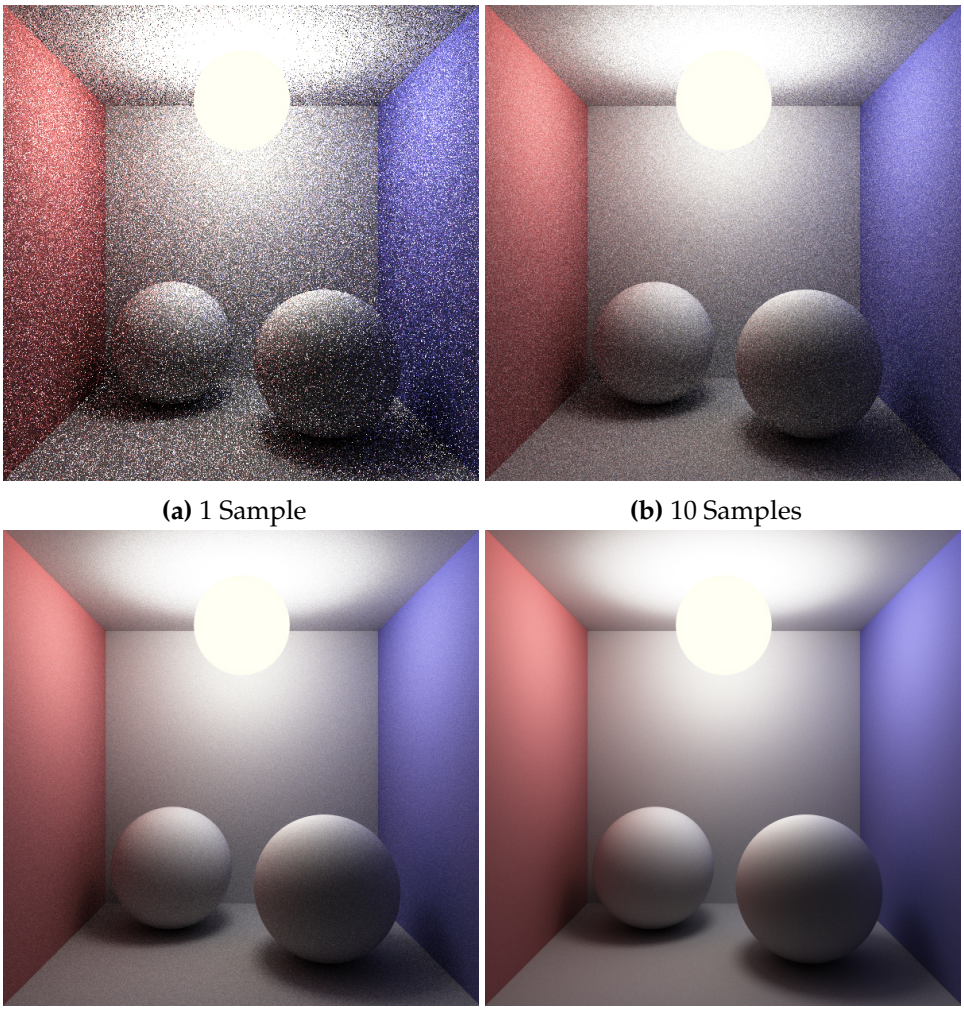


Abbildung 13: Links: Für jeden Schnittpunkt wird die Lichtquelle direkt abgetastet. Die Strahlen B, C und D tragen ihre Energie bei. Strahl A trägt keine Energie bei. Rechts: Für jeden Schnittpunkt wird außerdem das indirekte Licht bestimmt. Die Strahlen A, C und D werden verfolgt. Strahl B wird verworfen, da seine Energie bereits durch die direkte Abtastung vorliegt.

Der Vorteil von Next Event Estimation wird in Abbildung 14 deutlich. Durch die explizite Abtastung der Lichtquelle konvergieren die Bilder auf diffusen Oberflächen schneller zum Ergebnis. Abbildung 15 zeigt einen Vergleich mit und ohne direktem Sampling[3].
Die Szene kann durch spiegelnde und transmittierende Materialien erweitert werden (vgl. Abbildung 16a). Die Lichtquelle wird bei solchen Materialien üblicherweise nicht extra abgetastet. Die BRDF würde in den meisten Fällen ohnehin Null ergeben, es sei denn, die Lichtquelle liegt in exakter reflektierter oder gebrochener Strahlrichtung.
Materialien die sowohl einen diffusen, spiegelnden oder transmittierenden Anteil besitzen, sind ebenfalls möglich. Path Tracing verfolgt an jedem Schnittpunkt genau einen Strahl. Aus diesem Grund muss die Strahlrichtung bei kombinierten Materialien aus diffus, spiegelnd oder transmittierend gewählt werden. Das Vorgehen zum Auswählen des Pfades wird *Russian Roulette* genannt. Dazu wird eine gleichverteilte Zufallszahl ξ zwischen 0 und 1 erstellt. Besteht ein Material zu 60% aus einem diffusen, 20% aus einem spiegelnden und 20% aus einem transmittierenden Material, dann

[3]Für ein dokumentiertes Codebeispiel zu Path Tracing mit Next Event Estimation, siehe Anhang A, Codebeispiele 1, 2 und 3.

entscheidet die Zufallszahl ξ den weiterführenden Pfad. Liegt ξ im Intervall $[0, 0.6]$, wird ein diffuser Strahl generiert, usw.[4] Abbildung 16b zeigt kombinierte Materialien in der Testszene.

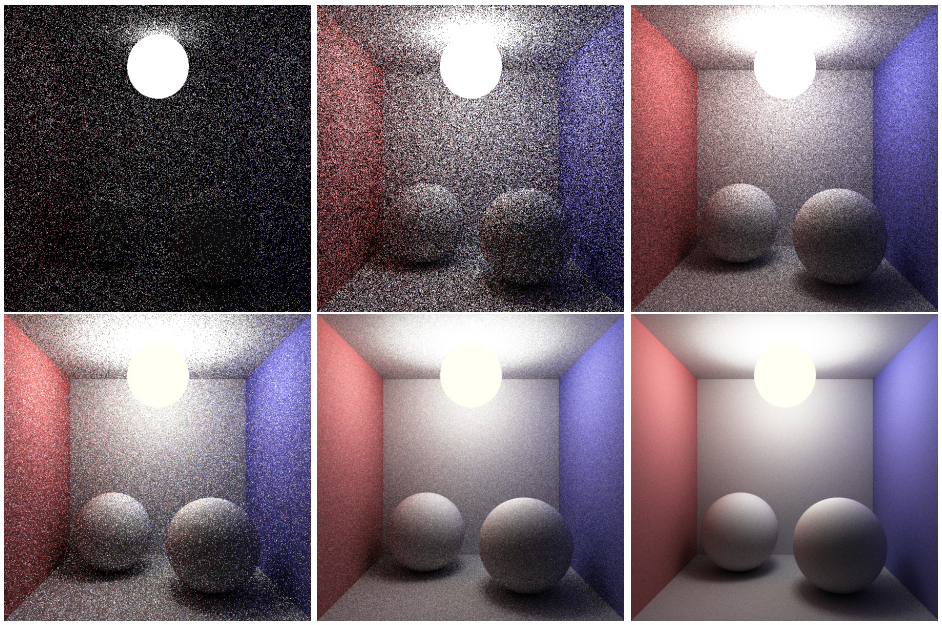


(a) 1 Sample **(b)** 10 Samples

(c) 100 Samples **(d)** 1000 Samples

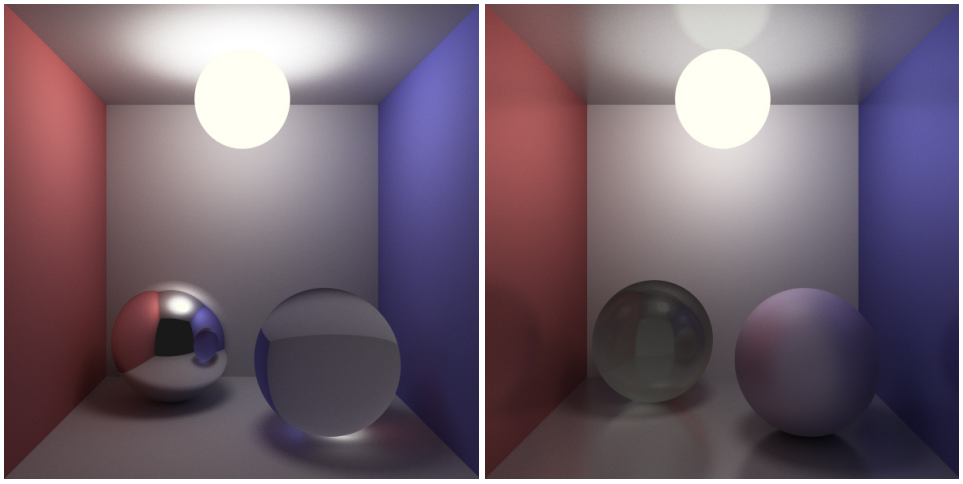
Abbildung 14: Das Bild konvergiert auf diffusen Flächen schneller zum Ergebnis mit weniger Bildrauschen. Dadurch wurde die Varianz vermindert.

[4] Für ein Codebeispiel zu Russian Roulette, siehe Anhang A, Codebeispiel 4.



(a) 1 Sample **(b)** 10 Samples **(c)** 100 Samples

Abbildung 15: Vergleich mit (unten) und ohne (oben) Next Event Estimation.



(a) **(b)**

Abbildung 16: (a) Spiegelnde und transmittierende Materialien. (b) Kombinationen von Materialien.

5 Optimierung durch Sampling und Bewertung der Strategie

Bisher wurde der Path Tracing Algorithmus und dessen Optimierung durch Next Event Estimation vorgestellt. In diesem Kapitel werden zuerst unterschiedliche Dichtefunktionen für das Sampling der direkten und indirekten Beleuchtung eingeführt und verglichen. Anschließend wird gezeigt, welche Erweiterungen mithilfe von Sampling vorgenommen werden können.

5.1 Direkte Beleuchtung & Soft Shadows

In einem klassischen Raytracer sind Schatten meistens scharf und die Folge von Punktlichtquellen in der Szene. In der Realität gibt es keine Punktlichtquellen und daher auch keine harten Schatten. Um weiche Schatten (engl. *soft shadows*) zu erzeugen, werden flächige Lichtquellen benötigt (vgl. Abbildung 17).

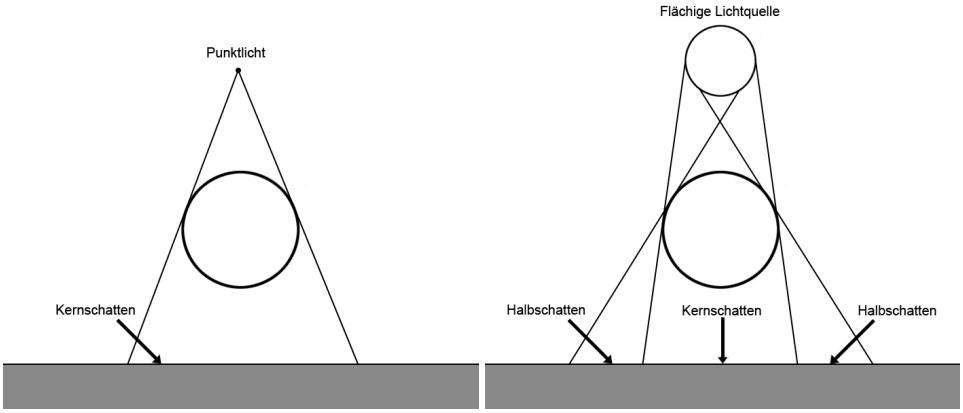


Abbildung 17: Links: Harte Schatten. Rechts: Halbschatten.

In einem Path Tracer mit Next Event Estimation wird an jedem Schnittpunkt die direkte und indirekte Beleuchtung berechnet. Durch Sampling der flächigen Lichtquelle werden weiche Schatten ermöglicht. Ein erster einfacher Ansatz wählt die Dichte als Inverse der Oberfläche [AHJ+01]:

$$p(x) = \frac{1}{A}$$

Daraus ergeben sich gleichverteilte Punkte auf der Lichtoberfläche. Eingesetzt in die Monte-Carlo-Integration aus Gleichung 13 ergibt:

$$L_{direct} = \frac{A}{N} \sum_{i=1}^{N} f_r(x, \vec{\omega}_i, \vec{\omega}_r) \cdot L_i(y, \overrightarrow{xy_i}) \cdot V(x, y_i) \cdot G(x, y_i)$$

In dieser Arbeit bestehen Lichtquellen ausschließlich aus Kugeln und sind Lambert-Strahler. Eine Lichtquelle wird Lambert-Strahler genannt, falls sie, unabhängig von der Betrachtungsrichtung aus, gleich hell erscheint. Für die BRDF solcher diffuser Strahler gilt nach [Laf96]:

$$f_r = \frac{\rho_x}{\pi} = const \tag{14}$$

Der Reflexionsgrad ρ_x definiert den Anteil des reflektierten Lichts zum einfallenden Licht von einem Oberflächenpunkt x. Er legt die Farbe des Materials fest und liegt im Bereich von 0 bis 1.
Die gleichverteilte Dichte einer kugelförmigen Lichtquelle mit Radius 1, sowie die Samples zum Abtasten einer Kugel sind aus Kapitel 3.1 bekannt:

$$p(\theta, \varphi) = \frac{1}{4\pi r^2}, \left[\ \theta, \varphi\ \right] = \begin{bmatrix} \arccos(1 - 2\xi_1) \\ 2\pi\xi_2 \end{bmatrix}$$

Nach der Konvertierung der Kugelkoordinaten in kartesische Koordinaten, werden die zufällig generierten Punkte auf den Mittelpunkt der Lichtquelle addiert. Schickt man nun Strahlen vom Schnittpunkt zu den generierten Punkten auf der Lichtquelle, werden durch diese einfache Methode weiche Schatten erzeugt (vgl. Abbildung 18).

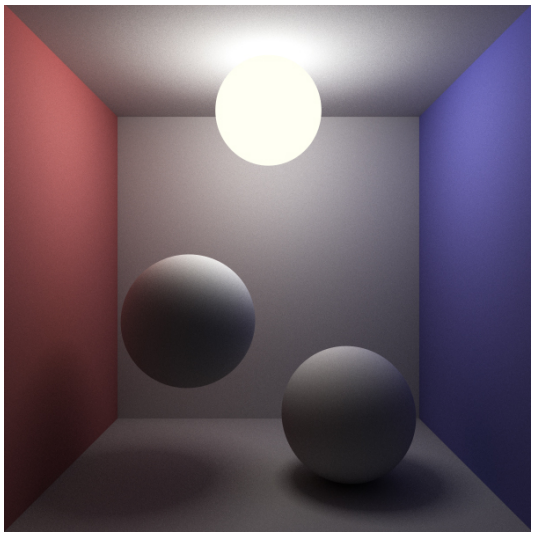


Abbildung 18: Weiche Schatten entstehen beispielsweise durch die gleichmäßige Abtastung einer Lichtquelle. Benötigte Anzahl an Samples: 500.

Die gewählte Dichte benötigt viele Samples, da die Varianz im Schatten groß ist. Die Folge von zu wenigen Samples führt zu verrauschten Ergebnissen im Schatten. Der Grund dafür ist die Generierung von Samples auch auf der abgewandten Rückseite der Kugel. Diese Punkte sind nicht sichtbar von einem Schnittpunkt x aus und führen durch die längeren Distanzen zu

einer Unterbelichtung der Szene. Durch einen Schnittpunkttest können die sichtbaren Punkte auf der Lichtquelle bestimmt werden. Es ergeben sich gleichverteilte Samples auf der sichtbaren Lichtoberfläche (vgl. Abbildung 19). Eine bessere Dichte kann durch Importance Sampling gewählt werden.

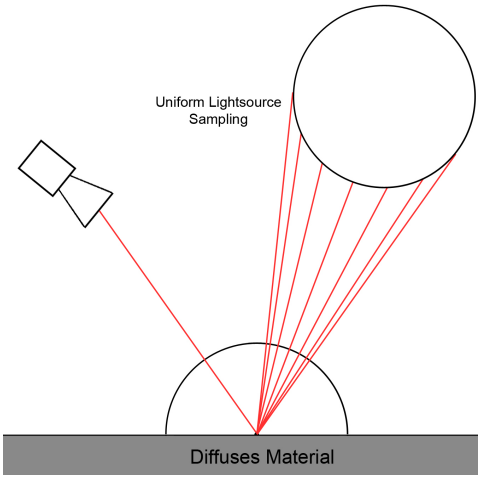


Abbildung 19: Gleichverteilte Abtastung der sichtbaren Lichtoberfläche.

Um das Rauschen und damit die Varianz zu reduzieren, sollen Samples nur in dem Bereich generiert werden, der von einem Schnittpunkt aus sichtbar ist. Der aufgespannte Kegel in Abbildung 20a wird Raumwinkel genannt. Es sollen nun gleichverteilte Samples auf der Einheitskugel im Inneren des Raumwinkels erzeugt werden (vgl. Abbildung 20b).

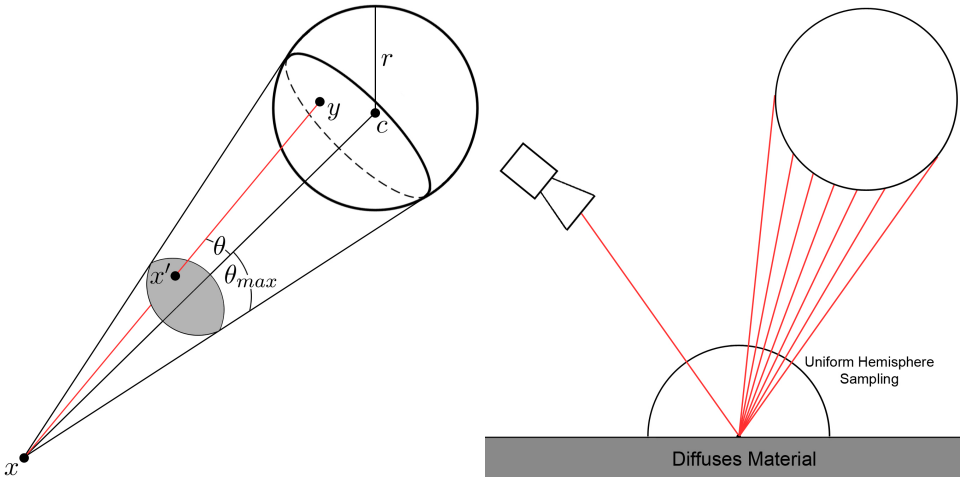


Abbildung 20: (a) Raumwinkel, (b) Gleichverteilte Abtastung des Raumwinkels.

Da die Lichtquelle als Lambert-Strahler vorliegt, sollte dies eine effizientere Sampling Methode der Lichtquelle sein. [SWZ96]
Zuerst wird der maximalen Öffnungswinkel θ_{max} bestimmt:

$$\theta_{max} = \arcsin(\frac{r}{||\overrightarrow{x-c}||}) = \arccos\sqrt{1-\left(\frac{r}{||\overrightarrow{x-c}||}\right)^2}$$

Die gleichverteilte Dichte für Samples innerhalb des Raumwinkels auf einer Einheitskugel, das bedeutet Radius 1, ist in [Kir92] gegeben:

$$p(x') = p(\theta,\varphi) = \frac{1}{(\varphi_2-\varphi_1)(\cos\theta_1-\cos\theta_2)}, \theta \in [\theta_1,\theta_2], \varphi \in [\varphi_1,\varphi_2]$$

Der Winkel φ läuft im Intervall $[0, 2\pi]$. θ hingegen läuft von 0 bis θ_{max}. Setzt man die gegebenen Werte ein, erhält man die Dichtefunktion für gleichverteilte Samples auf der Einheitskugel innerhalb des Raumwinkels:

$$\begin{aligned} p(x') = p(\theta,\varphi) &= \frac{1}{(2\pi-0)(\cos 0-\cos\theta_{max})} \\ &= \frac{1}{2\pi(1-\cos\theta_{max})} \\ &= \frac{1}{2\pi\left(1-\sqrt{1-\left(\frac{r}{||\overrightarrow{x-c}||}\right)^2}\right)} \end{aligned}$$

Mit inverser CDF-Methode ergeben sich θ und φ:

$$\begin{bmatrix} \theta \\ \varphi \end{bmatrix} = \begin{bmatrix} \arccos\left(1-\xi_1+\xi_1\sqrt{1-\left(\frac{r}{||\overrightarrow{x-c}||}\right)^2}\right) \\ 2\pi\xi_2 \end{bmatrix} \tag{15}$$

Nach der Transformation in kartesische Koordinaten erhält man einen Punkt ψ auf der Einheitskugel inmitten des Raumwinkels:

$$\psi = (\sin\theta, \cos\varphi, \sin\theta\sin\varphi, \cos\theta)$$

Abbildung 21 veranschaulicht die Samples auf einer Einheitskugel mit unterschiedlich weit entfernten Lichtquellen.
Danach muss das lokale Koordinatensystem $(\vec{u},\vec{v},\vec{w})$ bestimmt werden. Dabei entspricht die w-Achse dem Vektor vom Schnittpunkt x zum Lichtmittelpunkt c:

$$\vec{w} = \frac{\overrightarrow{c-x}}{||\overrightarrow{c-x}||}$$

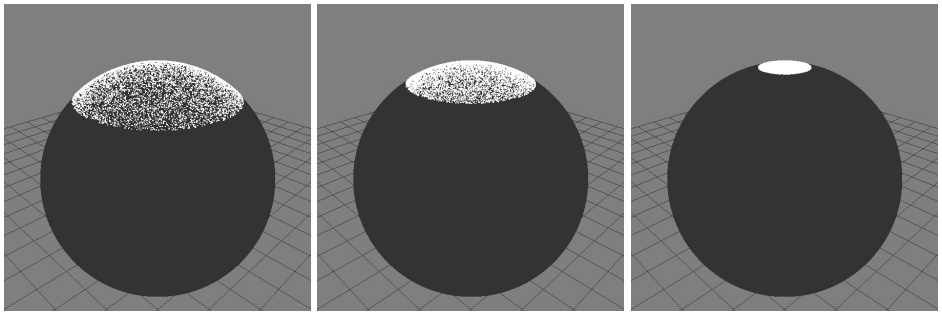


Abbildung 21: Gleichverteilte Samples auf einem Sektor der Einheitskugel. Die Distanz zur Lichtquelle von links nach rechts beträgt 1.5, 2 und 5. Je weiter die Lichtquelle entfernt ist, desto kleiner wird der Raumwinkel und damit der Bereich für Samples auf der Oberfläche.

In einem Rechtssystem ergeben sich die fehlenden Achsen $\vec{u}$ und $\vec{v}$ durch das Kreuzprodukt mit einem willkürlichen Hilfsvektor $\vec{t}$, der nicht mit $\pm\vec{w}$ übereinstimmt:

$$\begin{aligned} \vec{u} &= \vec{w} \times \vec{t} \\ \vec{v} &= \vec{w} \times \vec{u} \end{aligned}$$

Beide Achsen sind Einheitsvektoren. Durch Rotation von ψ um das neue Koordinatensystem erhält man den tatsächlichen Punkt x' aus Abbildung 20a auf der Einheitskugel innerhalb des Raumwinkels:

$$x' = \vec{u} \cdot \psi.x + \vec{v} \cdot \psi.y + \vec{w} \cdot \psi.z$$

Als Nächstes wird ein Strahl vom Schnittpunkt x zu x' geschickt und mit der Lichtquelle getestet. Mit dem vorderste Schnittpunkt y ist ein Punkt auf der Lichtquelle gefunden, der im Inneren des Raumwinkels liegt und diesen gleichverteilt abtastet.
Als letzten Schritt muss die Dichtefunktion von y mithilfe von $p(x')$ berechnet werden. Durch Gleichung 9 aus Kapitel 4.4 kann die Dichte $p(x')$ in Relation zur Fläche der Lichtquelle gebracht werden:

$$\begin{aligned} p(y) &= \frac{p(x') \cdot \cos\theta'}{||\overrightarrow{y-x}||^2} \\ &= \frac{\cos\theta'}{2\pi\left(1 - \sqrt{1 - \left(\frac{r}{||\overrightarrow{x-c}||}\right)^2}\right) \cdot ||\overrightarrow{y-x}||^2} \\ &= \frac{\cos\theta'}{2\pi\left(1 - \cos\theta_{max}\right) \cdot ||\overrightarrow{y-x}||^2} \end{aligned}$$

Nun kann $p(y)$ in die Monte-Carlo-Schätzung für die direkte Beleuchtung aus Gleichung 12 eingesetzt und mithilfe der BRDF für diffuse Oberflächen aus Gleichung 14 vereinfacht werden:

$$\begin{aligned} L_{direct} &= \frac{1}{N}\sum_{i=1}^{N}\frac{f_r(x,\vec{\omega}_i,\vec{\omega}_r)\cdot L_i(y,\overrightarrow{xy})\cdot G(x,y_i)\cdot V(x,y_i)}{p(y_i)} \\ &= \frac{\rho_x}{\pi N}\sum_{i=1}^{N}\frac{L_i(y,\overrightarrow{xy})\cdot G(x,y_i)\cdot V(x,y_i)2\pi\,(1-\cos\theta_{max})\,||\overrightarrow{y_i-x}||^2}{\cos\theta_i'} \\ &= \frac{\rho_x}{N}\sum_{i=1}^{N}L_i(y,\overrightarrow{xy})\cdot\cos\theta_i\cdot V(x,y_i)\cdot 2\cdot(1-\cos\theta_{max}) \end{aligned} \tag{16}$$

Durch Sampling der Lichtquelle nach dem vorgestellten Schema[5] entstehen die in Abbildung 22 zu sehenden Soft Shadows. Je größer die Lichtquelle, desto weicher werden die Schatten.
Die Unterschiede der vorgestellten Methoden sind gering. Der Vergleich in Abbildung 23 zeigt, dass die Varianz im Schatten in Abbildung 23d am geringsten ist. Im Gegensatz zu einer Punktlichtquelle liefern alle Vorgehensweisen ein deutlich realistischeres Resultat.

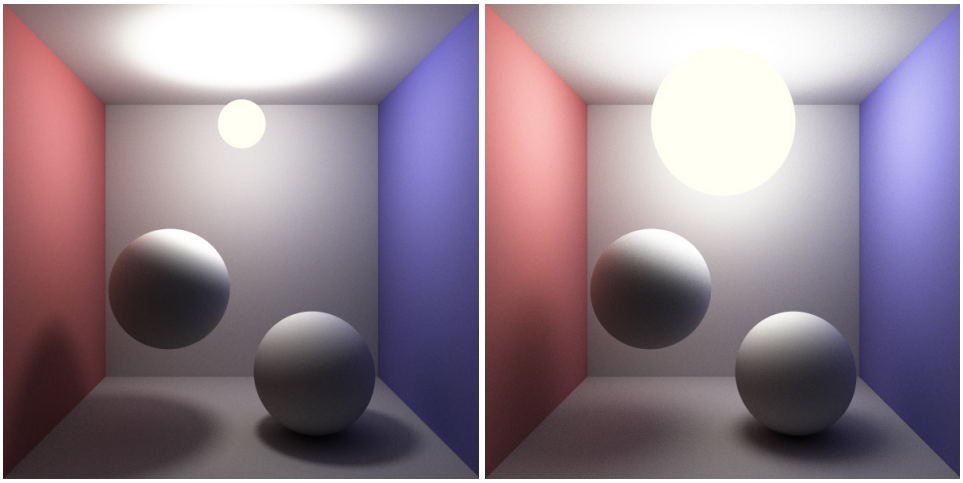
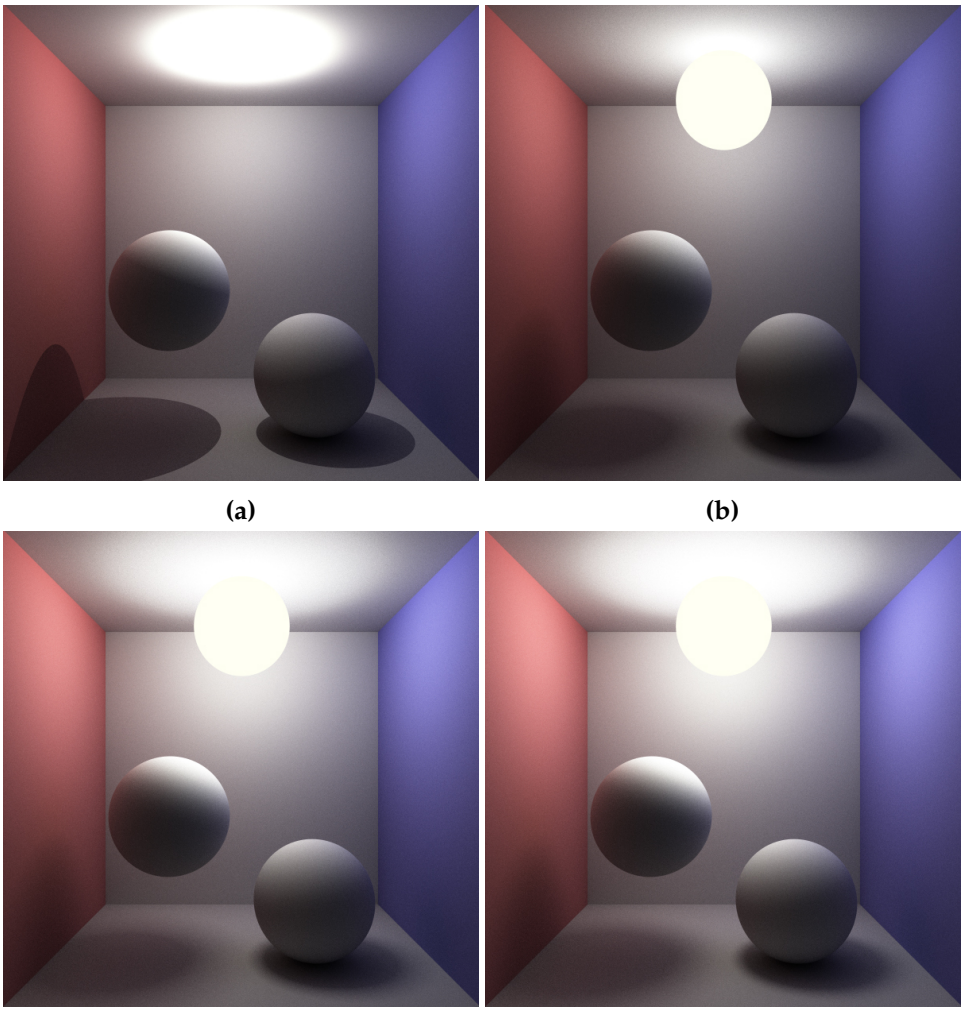


Abbildung 22: An jedem Schnittpunkt wird die sichtbare Lichtquelle innerhalb des Raumwinkels gleichmäßig abgetastet. Die entstandenen Soft Shadows variieren mit der Größe der Lichtquelle. Anzahl an Samples: 500.

[5]Für ein Codebeispiel zu direkter Beleuchung mit Samples innerhalb des Raumwinkels, siehe Anhang A, Codebeispiel 5.



(a) **(b)**

(c) **(d)**

Abbildung 23: Ergebnisse nach 500 Samples: (a) Punktlichtquelle, (b) gleichverteilte Samples auf der gesamten Lichtquelle, (c) gleichverteilte Samples auf der sichtbaren Lichtquelle, (d) gleichverteilte Samples innerhalb des Raumwinkels.

5.2 Indirekte Beleuchtung

Das globale Beleuchtungsmodell von Path Tracing berücksichtigt das indirekt reflektierte Licht von Oberflächen an einem Schnittpunkt. Dadurch sind die Bilder näher an der Realität und physikalisch plausibel. Die Monte-Carlo-Schätzung für die indirekte Beleuchtung wurde in Gleichung 13 eingeführt. Die Wahl der Dichte ist abhängig von dem Material am Schnittpunkt. Für diffuse Oberflächen ist die BRDF konstant. Nun soll eine Dichte gefunden werden, die die Hemisphäre bestmöglich abtastet.
Eine Möglichkeit die Hemisphäre gleichmäßig abzutasten bietet Uniform Sampling [Kir92]. Hierzu wird folgende Dichte gewählt:

$$p(\vec{\omega}_i) = \frac{1}{2\pi}$$

Einsetzen in die Monte-Carlo-Integration für die indirekte Beleuchtung aus Gleichung 13 und Vorziehen der BRDF ergibt:

$$\begin{aligned} L_{indirect} &= \frac{1}{N}\sum_{i=1}^{N}\frac{f_r(x,\vec{\omega}_i,\vec{\omega}_o)\cdot L_i(x,\vec{\omega}_i)\cdot\cos\theta_i}{p(\vec{\omega}_i)} \\ &= \frac{\rho_x}{\pi}\cdot\frac{1}{N}\sum_{i=1}^{N}\frac{L_i(x,\vec{\omega}_i)\cdot\cos\theta_i}{\frac{1}{2\pi}} \\ &= \rho_x\cdot\frac{2}{N}\sum_{i=1}^{N}L_i(x,\vec{\omega}_i)\cdot\cos\theta_i \end{aligned} \tag{17}$$

Um in einem Path Tracer Punkte nach der gegebenen Dichte zu generieren, wird eine Kreisscheibe auf die Hemisphäre projiziert [PH10]:

$$\begin{bmatrix} r \\ \varphi \end{bmatrix} = \begin{bmatrix} \sqrt{1-\xi_1^2} \\ 2\pi\xi_2 \end{bmatrix} \tag{18}$$

Die Samples werden auf kartesische Koordinaten abgebildet, wobei die y-Achse die gleichverteilten Werte ξ_1 zwischen 0 und 1 annimmt:

$$\begin{aligned} x &= \cos(\varphi)\cdot r \\ y &= \xi_1 \\ z &= \sin(\varphi)\cdot r \end{aligned}$$

Man erhält eine gleichmäßige Abtastung der Hemisphäre (vgl. Abbildung 24) . Nun muss der Punkt wieder in das lokale Koordinatensystem (u, v, w) transformiert werden. Dabei entspricht die v-Achse dem Normalenvektor am Schnittpunkt:

$$\vec{v} = normal$$

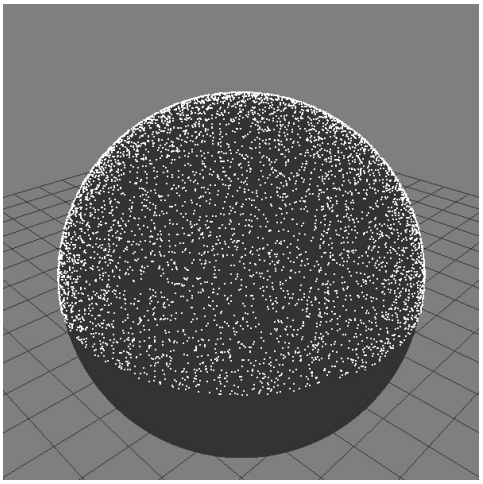


Abbildung 24: Uniform Sampling: Gleichverteilte Samples auf der oberen Hälfte einer Kugel.

Durch den willkürlichen Hilfsvektor t werden die beiden orthogonal zueinander liegenden Achsen bestimmt:

$$\begin{aligned} \vec{u} &= \vec{t} \times \vec{v} \\ \vec{w} &= \vec{u} \times \vec{v} \end{aligned}$$

Uniform Sampling skaliert das einfallende Licht in Gleichung 17 mit einem Kosinusfaktor. Somit tragen Strahlen am Boden der Hemisphäre wenig zum Gesamtbeitrag bei. Eine bessere Dichte kann unter Berücksichtigung von Importance Sampling gewählt werden, die weniger Strahlen am Boden der Hemisphäre generiert [DHM+01]:

$$p(\vec{\omega}_i) = p(\theta, \varphi) = \frac{\cos\theta}{\pi}$$

Die Monte-Carlo-Integration für die indirekte Beleuchtung aus Gleichung 13 vereinfacht sich zu:

$$\begin{aligned} L_{indirect} &= \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(x, \vec{\omega}_i, \vec{\omega}_o) \cdot L_i(x, \vec{\omega}_i) \cdot \cos\theta}{p(\vec{\omega}_i)} \\ &= \frac{1}{N} \sum_{i=1}^{N} \frac{\frac{\rho_x}{\pi} \cdot L_i(x, \vec{\omega}_i) \cdot \cos\theta}{\frac{\cos\theta}{\pi}} \\ &= \frac{1}{N} \sum_{i=1}^{N} \rho_x \cdot L_i(x, \vec{\omega}_i) \end{aligned} \tag{19}$$

Die beiden sphärischen Winkel werden mittels inverser CDF-Methode bestimmt:

$$\begin{bmatrix} \theta \\ \varphi \end{bmatrix} = \begin{bmatrix} \arccos(1 - \xi_1) \\ 2\pi\xi_2 \end{bmatrix} \tag{20}$$

Da ξ_1 eine gleichverteilte Zufallsvariable zwischen 0 und 1 ist, ergibt $1 - \xi_1$ ebenfalls eine gleichverteilte Zufallsvariable [PH10]:

$$\theta = \arccos(\xi_1)$$

Nach der Transformation in lokale Koordinaten ergibt sich eine kosinusgewichtete Verteilung der Samples auf einer Kugel (vgl. Abbildung 25).

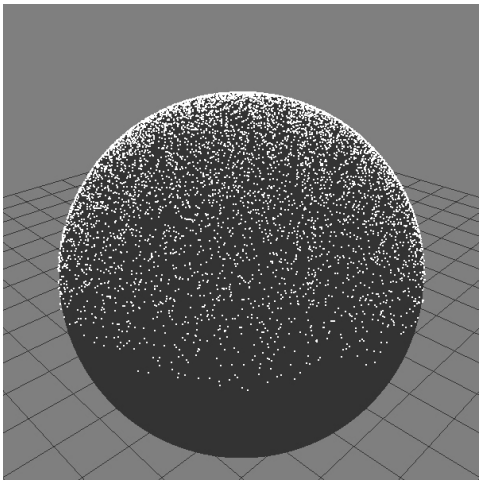


Abbildung 25: Importance Sampling: Keine Gleichverteilung der Samples. Es werden weniger Punkte am Boden der Hemisphäre generiert.

Die Unterschiede von Uniform und Importance Sampling sind gering. Allerdings liefert Importance Sampling[6] weniger Bildrauschen und konvergiert auf diffusen Oberflächen schneller zum Ergebnis (vgl. Abbildung 26).

5.3 Glossy Reflections & Refractions

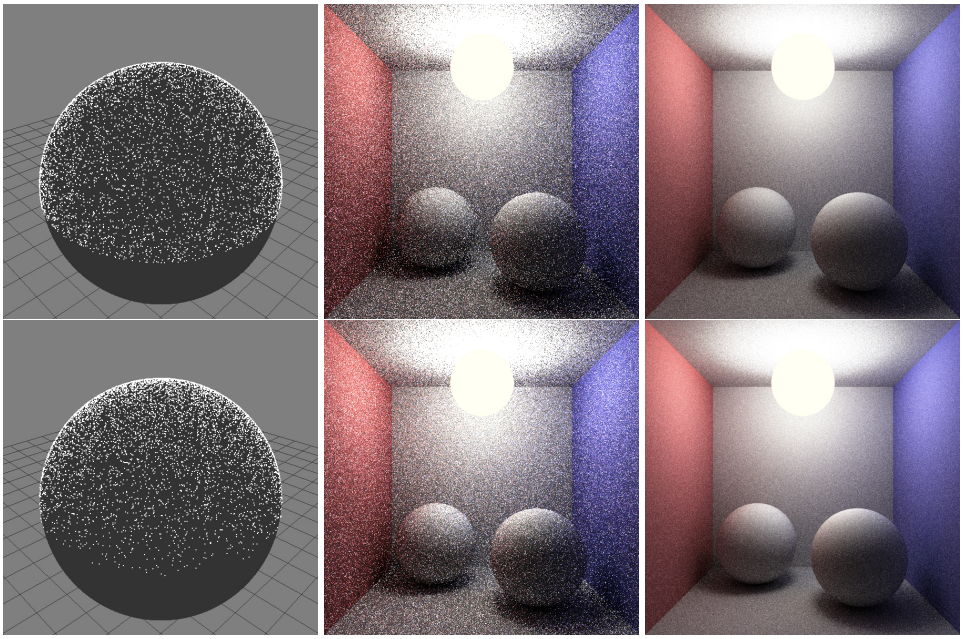
Das Auswählen zufälliger Strahlen oberhalb der Hemisphäre ist sinnvoll für diffuse Oberflächen. Für ein glänzendes Material gilt nach dem Phong-Beleuchtungsmodell [DHM+01]:

$$f_r = \rho_s \cdot \frac{n+2}{2\pi} \cdot \cos^n \theta$$

In [Kir92] werden Strahlen mit folgender Dichte gewählt:

$$p(\vec{\omega}_i) = p(\theta, \varphi) = \frac{n+1}{2\pi} \cos^n \theta$$

[6] Für ein Codebeispiel zu Importance Sampling der Hemisphäre, siehe Anhang A, Codebeispiel 6.



(a) 1 Sample **(b)** 16 Samples

Abbildung 26: Vergleich Uniform Sampling (oben) und Importance Sampling (unten) der Hemisphäre.

Einsetzen in das Integral der Rendergleichung ergibt:

$$\begin{aligned} L_r(x,\vec{\omega}_r) &= \int_\Omega f_r(x,\vec{\omega}_i,\vec{\omega}_r)\cdot L_i(x,\vec{\omega}_i)\cdot\cos\theta\cdot d\vec{\omega}_i \\ &= \rho_x\cdot\int_\Omega \frac{n+2}{2\pi}\cdot\cos^n\theta\cdot L_i(x,\vec{\omega}_i)\cdot\cos\theta\cdot d\vec{\omega}_i \\ &\approx \rho_x\cdot\frac{1}{N}\sum_{i=1}^{N}\frac{\frac{n+2}{2\pi}\cdot\cos^n\theta_i\cdot L_i(x,\vec{\omega}_i)\cdot\cos\theta_i}{p(\vec{\omega}_i)} \\ &\approx \rho_x\cdot\frac{1}{N}\sum_{i=1}^{N}\frac{n+2}{n+1}\cdot L_i(x,\vec{\omega}_i)\cdot\cos\theta_i \end{aligned} \tag{21}$$

In der Praxis kann der Faktor $\cos\theta_i$ weggelassen werden. Er verursacht beim Phong-Modell dunkle Ränder (vgl. Abbildung 27).
Um Strahlen mit der gegebenen Dichte zu generieren, bestimmt man die sphärischen Winkel θ und φ wieder mittels inverser CDF-Methode:

$$\begin{bmatrix}\theta\\ \varphi\end{bmatrix} = \begin{bmatrix}\arccos((1-\xi_1)^{\frac{1}{n+1}})\\ 2\pi\xi_2\end{bmatrix} \tag{22}$$

ξ_1 ist eine gleichverteilte Zufallsvariable zwischen 0 und 1, daher verkürzt

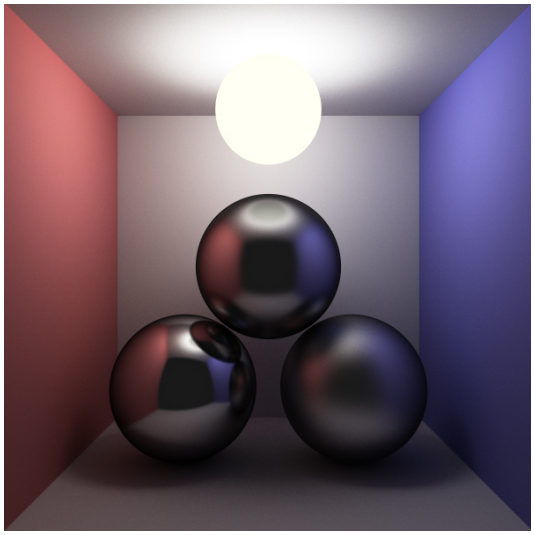


Abbildung 27: Der Faktor $\cos\theta_i$ verursacht beim Phong-Modell dunkle Ränder und kann vernachlässigt werden.

sich θ zu:

$$\theta = \arccos(\xi_1^{\frac{1}{n+1}})$$

Die Kugelkoordinaten werden wiederum auf die entsprechenden kartesischen Koordinaten abgebildet. Abbildung 28 veranschaulicht das Sampling einer Halbkugel mit unterschiedlichen Phong-Exponenten. Dabei entspricht ein Phong-Exponent von $n = 1$ Importance Sampling und $n = 0$ Uniform Sampling der Hemisphäre [DHM+01].

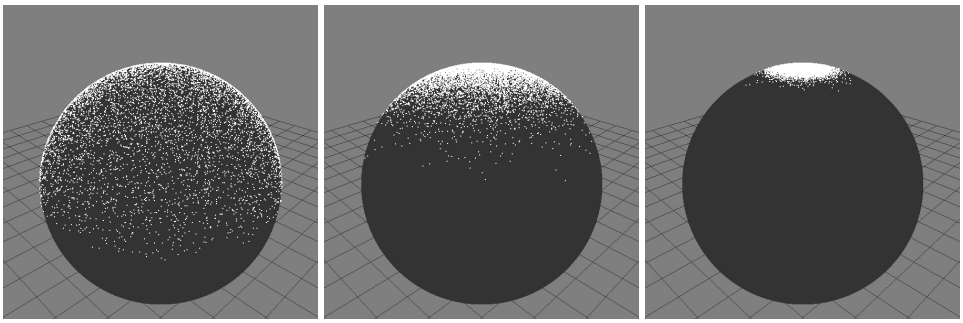


Abbildung 28: Von links nach rechts: Phong-Beleuchtungsmodell mit $n = 1, 10,$ und 100.

Nächster Schritt ist die Transformation in lokale Koordinaten. Für glänzende Oberflächen entspricht die v-Achse dem reflektierten Vektor. Die beiden anderen Achsen liegen orthogonal dazu. Anschließend muss auf den Fehler getestet werden, ob die Richtung des Strahls in den negativen Halbraum zeigt. Gegebenenfalls muss ein neuer Strahl generiert werden.

Beim Sampling der BRDF nach dem Phong-Beleuchtungsmodell[7] entspricht der Phong-Exponent n der Rauheit des Materials. Dadurch entstehen die in Abbildung 29 zu sehenden *Glossy Reflections* mit unterschiedlichen Phong-Exponenten. Das selbe Vorgehen, angewandt auf transmittierende Materialien, ergeben *Glossy Refractions* (vgl. Abbildung 30).

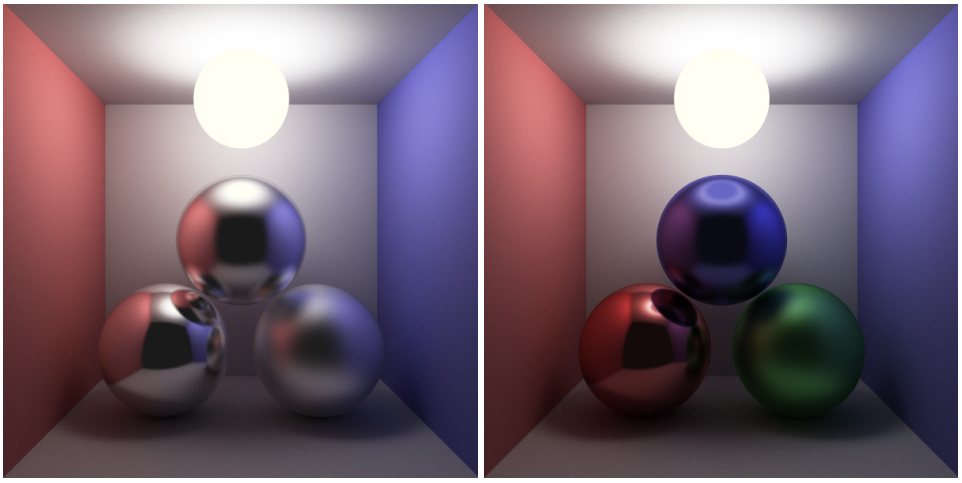


Abbildung 29: Glossy Reflections: Glänzende Materialien mit unterschiedlichen Phong-Exponenten. Von links nach rechts: $n = 1000, 100, 10$.

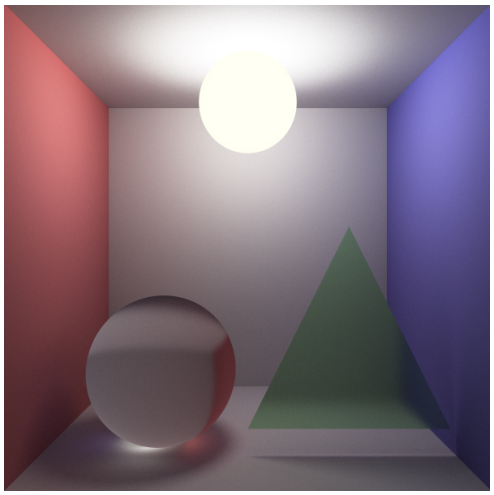


Abbildung 30: Glossy Refractions: Transmittierende Materialien mit Brechungsindex von Wasser und unterschiedlichen Phong-Exponenten (Kugel: $n = 500$ Dreieck: $n = 100$).

[7] Für ein Codebeispiel zu Sampling der BRDF nach dem Phong-Modell, siehe Anhang A, Codebeispiel 7.

5.4 Anti-Aliasing

Aliasing Artefakte entstehen durch das begrenzt aufgelöste Pixelraster. Die unerwünschten Effekte machen sich im Bild als kleine Treppenstufen bemerkbar (vgl. Abbildung 31).

Abbildung 31: Aliasing führt zu unerwünschten Artefakten.

Bei Path Tracing werden N Strahlen durch ein Pixel geschickt. Wählt man für jeden Strahl eine zufällige Verteilung innerhalb des Pixels, kann Aliasing vermindert werden. Eine Möglichkeit bietet *Random Sampling*. Zu diesem Zweck werden zwei gleichverteilte Zufallszahlen im Bereich $[-0.5, 0.5]$ generiert und auf die Pixelmitte in x- und y-Richtung addiert. In Abbildung 32 ist Random Sampling veranschaulicht.

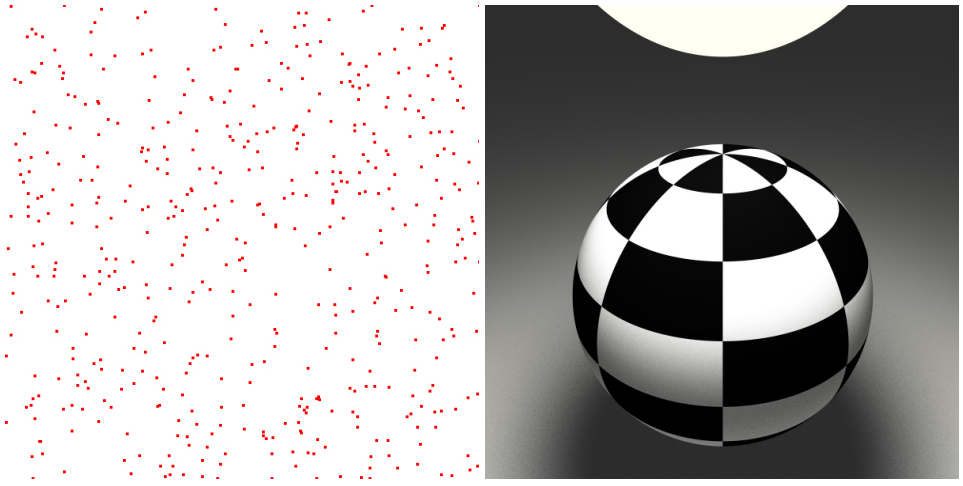


Abbildung 32: Random Sampling unterdrückt Aliasing.

Dabei kann es sein, dass Löcher oder Anhäufungen entstehen, die sich negativ auf die Bildqualität auswirken. Ein weiterer Versuch für Anti-Aliasing ist *Stratified Sampling*. Hierbei wird ein Gitter über das Pixel gelegt und innerhalb des Rasters zufällig eine Position gewählt. Zur Implementierung wird das Pixel zuerst in vier Quadranten unterteilt. Mithilfe der Modulo-Operation wird ein Quadrant ausgewählt. Das erste Sample wird im oberen linken Viertel generiert, das zweite im oberen rechten usw. Bei dem fünften Sample wird das obere linke Viertel wiederum in vier gleiche Quadranten unterteilt und ein Sample im oberen linken Viertel generiert. Abbildung 33 verdeutlicht den rekursiven Algorithmus zur Unterteilung des Pixels.

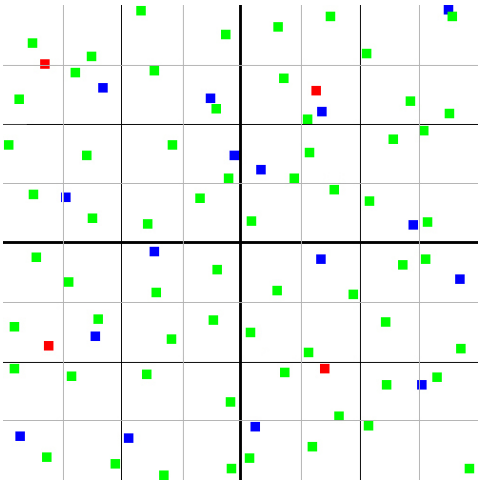


Abbildung 33: Stratified Sampling: Zuerst wurde das Pixel in vier gleich große Bereiche unterteilt. Anschließend wurden innerhalb der Quadranten vier rote Samples generiert. Der nächste Durchlauf zerlegt die Quadranten wiederum in vier Zellen und generiert die 16 blauen Samples. Die 64 grünen Samples entstehen im darauffolgenden Durchlauf.

Durch die Unterteilung in vier gleich große Quadranten ist Stratified Sampling am effektivsten bei folgender Anzahl an Wiederholungen:

$$N = 4, 20, 84, 340, ..., \sum_{i=1}^{N} 4^n$$

Das Verfahren bietet gegenüber Random Sampling eine bessere Verteilung der Stichproben (vgl. Abbildung 34). Im schlimmsten Fall kann es jedoch immer noch zu Häufungen an den Gitterkanten kommen. [PH10]
Random Sampling und Stratified Sampling basieren jeweils auf Zufallsvariablen. Sie bilden eine zufällige Verteilung im jeweiligen Pixel. Demgegenüber gibt es die sogenannten *Quasi-Random Numbers*. Ihr Vorteil liegt

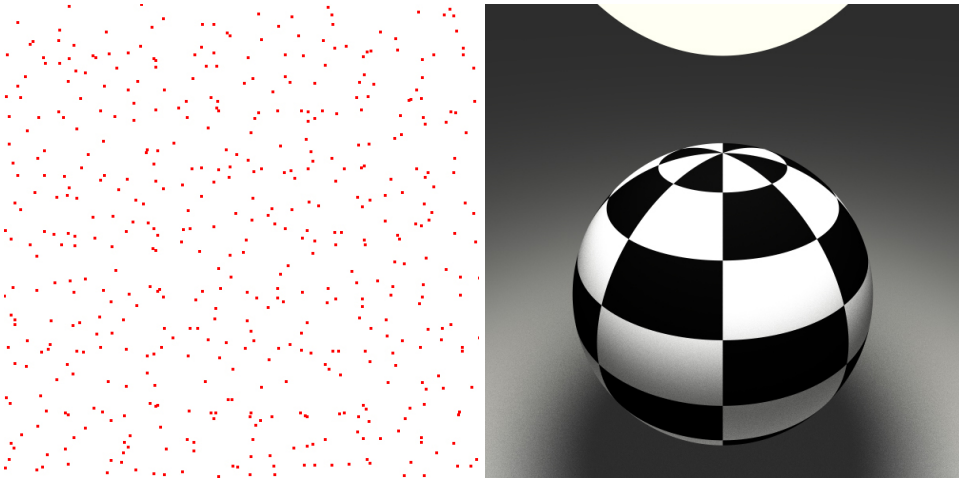


Abbildung 34: Stratified Sampling liefert gegenüber Random Sampling eine bessere Verteilung im Pixel.

darin, dass die generierten Werte eine gewisse Abweichung voneinander haben und es zu weniger Häufungen kommt. Die *Halton Sequenz* ist ein Entwurfsmuster aus dem Bereich der Quasi-Random Numbers und mindert das Aliasing Problem mit einer fest vorgegebenen Verteilung. Als Eingabe bekommt die Halton Sequenz eine Primzahl als Basis.
Gegeben ist das Intervall $[0, 1]$ und die Basis 2. Ausgehend von der Basis wird das Intervall nun halbiert, geviertelt, geachtelt usw. Bei der Basis 3 wird das Intervall entsprechend gedrittelt, geneuntelt, usw. Tabelle 3 zeigt die ersten 9 Komponenten der Basen $2, 3$ und 5. Dabei ist zu beachten, dass zuerst die niedrigeren Brüche aller Intervalle eingefügt werden.[8]

n	base 2	base 3	base 5
1	$1/2$	$1/3$	$1/5$
2	$1/4$	$2/3$	$2/5$
3	$3/4$	$1/9$	$3/5$
4	$1/8$	$4/9$	$4/5$
5	$5/8$	$7/9$	$1/25$
6	$3/8$	$2/9$	$6/25$
7	$7/8$	$5/9$	$11/25$
8	$1/16$	$8/9$	$16/25$
9	$9/16$	$1/27$	$21/25$
...	...	...	...

Tabelle 3: Halton Sequenz der Basen 2, 3 und 5.

[8] Für ein Codebeispiel zur Halton Sequenz, siehe Anhang A, Codebeispiel 8.

Ordnet man beispielsweise die Basen 2 und 3 aus dem Intervall $[0, 1]$ paarweise an, erhält man x- und y-Koordinate in einem Quadrat. Pixel Sampling mittels Halton Sequenz ergibt eine bessere Verteilung im Pixel als Stratified Sampling, wie Abbildung 35 zeigt. [PH10]

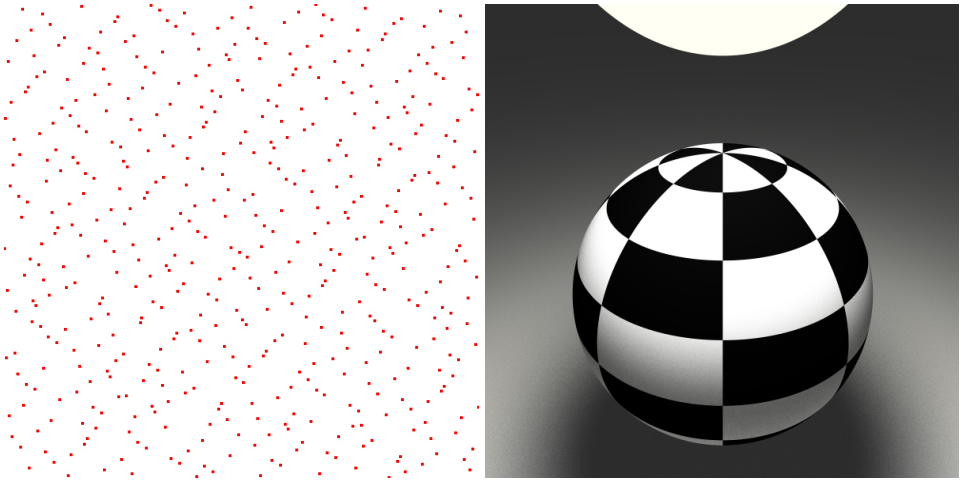


Abbildung 35: Halton Sequenz mit Basen 2 und 3.

Die Unterschiede der Ergebnisse der vorgestellten Methoden um Aliasing Artefakte zu vermindern sind gering. In Abbildung 36 ist ein Vergleich der Sampling Methoden mit 16 Samples aufgetragen. Stratified Sampling und Halton Sequenz liefern gegenüber Random Sampling schon zu Beginn annehmbare Ergebnisse. Die beiden Varianten sind bei steigender Anzahl an Samples kaum noch zu unterscheiden. Jedoch bietet die Halton Sequenz theoretisch die beste Verteilung um die unerwünschten Effekte zu beseitigen, da weniger Häufungen als bei Stratified Sampling entstehen.

5.5 Depth of Field & Motion Blur

In der Realität besitzen Kameras eine Brennweite und nicht alle Objekte der Szene liegen im Fokus. Dadurch entstehen verschwommene Bereiche in Bildern. Dieser Effekt wird *Depth of Field* genannt. Die virtuelle Kamera in einem Path Tracer besteht üblicherweise aus einer Lochkamera (engl. *pinhole camera*). Die Strahlen entstehen alle in einem Punkt und werden durch die Bildschirmebene geschickt. Aus diesem Grund sind alle Objekte scharf (vgl. Abbildung 37).
Um den Depth of Field Effekt zu implementieren wird Licht nicht in einem Punkt gebündelt, sondern durch eine Blendenöffnung eingefangen. Die Öffnung wird in einem Path Tracer durch Sampling dieser Blende angenommen. Dazu wird die Kameraposition und damit der Ursprung der Strahlen bei jedem Durchlauf senkrecht zur Strahlrichtung verschoben. An-

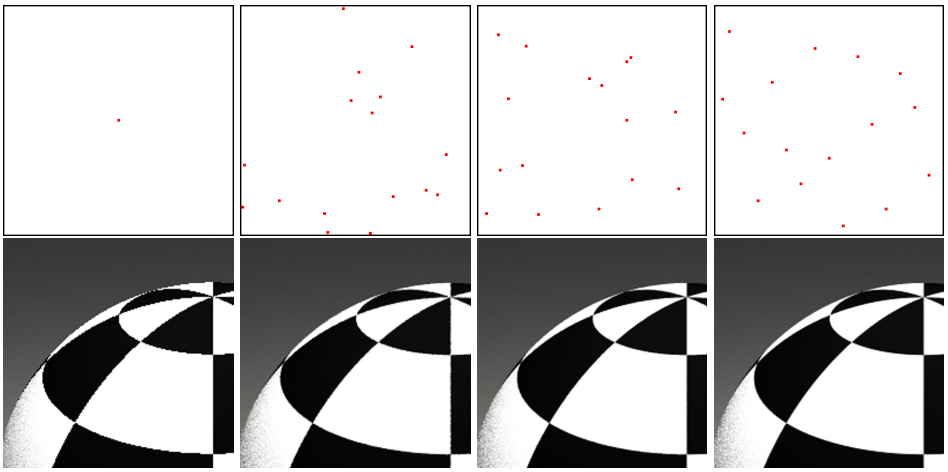


Abbildung 36: (a) kein Sampling, (b) Random Sampling, (c) Stratified Sampling, (d) Halton Sequenz.

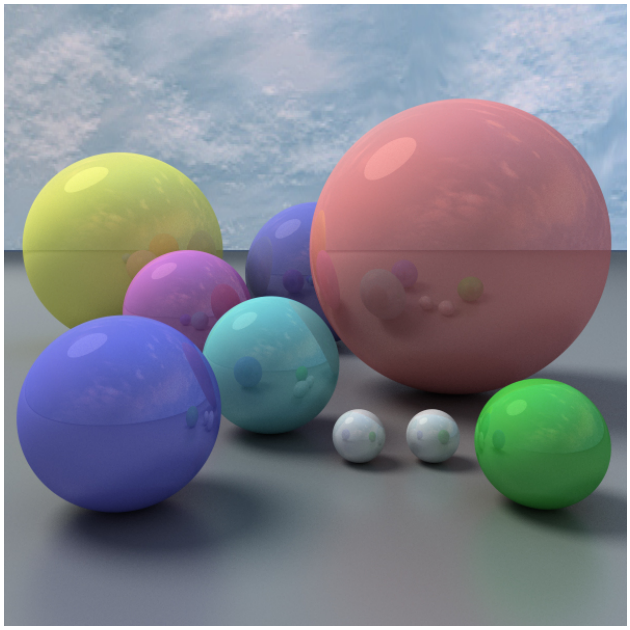
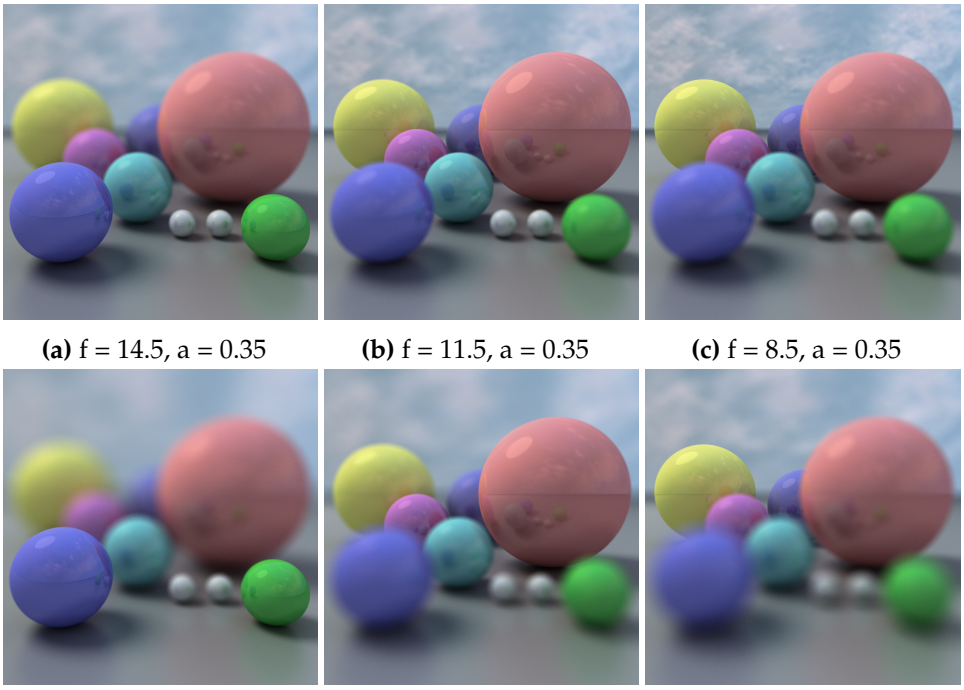


Abbildung 37: Der Kameraursprung liegt in einen festen Punkt. Alle Bereiche sind scharf.

schließend wird ein Brennpunkt definiert, der die Entfernung, in der Objekte im Fokus liegen, angibt. Außerhalb des Bereichs werden die Objekte verschwommen dargestellt. Wie stark die Unschärfe ist, gibt die Größe der Blendenöffnung an. Sie definiert das Ausmaß, in denen die Zufallszahlen liegen. Der neue Strahl hat seinen Ursprung in der verschobenen Kameraposition. Seine Richtung zeigt auf den Brennpunkt. Abbildung 38 zeigt unterschiedliche Einstellungen der Größe der Blende (engl. *aperture size*) und Entfernung des Brennpunkts (engl. *focal length*).



(a) f = 14.5, a = 0.35 **(b)** f = 11.5, a = 0.35 **(c)** f = 8.5, a = 0.35

(d) f = 14.5, a = 1.0 **(e)** f = 11.5, a = 1.0 **(f)** f = 8.5, a = 1.0

Abbildung 38: Ergebnisse nach 1000 Samples. f = Brennweite, a = Blendenöffnung.

Durch Sampling der Kameraposition kann die Form der Blende individuell angepasst werden. Die Form aus Abbildung 38 ging durch das zufällige Verschieben in x- und y-Richtung der Kamera hervor. Dadurch wurde wie beim Anti-Aliasing ein Quadrat abgetastet. Dementsprechend gibt es wieder die Möglichkeit Stratified Sampling oder die Halton Sequenz für eine gleichmäßigere Verteilung anzuwenden.
In der Realität besitzen Kameras jedoch meistens eine kreisförmige Blende (vgl. Abbildung 39). Aus Kapitel 3.1 ist die Dichtefunktion einer Kreisscheibe mit gleichmäßiger Verteilung und dessen Samples bekannt:

$$p(r,\varphi) = \frac{1}{\pi \cdot R^2}, \begin{bmatrix} r \\ \varphi \end{bmatrix} = \begin{bmatrix} R\sqrt{\xi_1} \\ 2\pi \cdot \xi_2 \end{bmatrix}$$

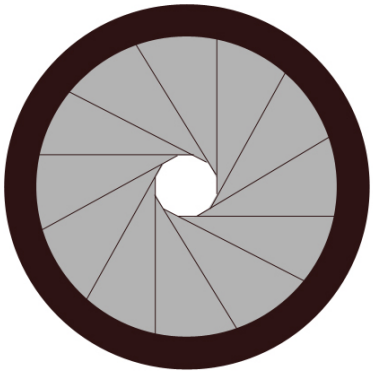


Abbildung 39: Die kreisörmige Blendenöffnung einer Kamera.

Der Radius R definiert die Größe der Kreisscheibe und ist ausschlaggebend für die Stärke der Unschärfe. Das Verschieben der Kameraposition innerhalb einer Kreisscheibe führt zu einer realistischeren Darstellung der Tiefenschärfe. Abbildung 40 zeigt den Unterschied zwischen quadratischer und kreisförmiger Blende[9].
Motion Blur ist eine weitere Möglichkeit, Bilder ästhetischer wirken zu lassen. Dieser Effekt ist Folge von bewegten Objekten während der Belichtungszeit. Dadurch wird dem Motiv eine gewisse Bewegungsunschärfe verliehen und der Eindruck von Bewegung entsteht. Beim Path Tracing wird ein Bild belichtet, solange die maximale Anzahl an Samples noch nicht erreicht wurde. Werden Objekte während dieser Zeit transliert oder rotiert, entsteht der Motion Blur Effekt [Kir92].
Zu Beginn wird die ursprüngliche Position eines Objekts gespeichert. In der darauffolgenden Schleife über alle Pixel wird die Position pro Durchlauf anhand von gleichverteilten Zufallszahlen verändert und der Strahl verfolgt. Dadurch befindet sich das Objekt bei jedem verfolgten Kamerastrahl an einer anderen Position. Am Ende der Schleife wird wieder die alte Position gesetzt. Je weiter ein Objekt zu seinem Ursprung verschoben wird, desto ausgeprägter ist der Motion Blur Effekt. In Abbildung 41 sind zwei Beispiele mit jeweils unterschiedlichen Einstellungen aufgetragen.

Depth of Field und Motion Blur sind Effekte, die mit wenig Aufwand in einen Path Tracer implementiert werden können. Im Gegensatz zu klassischem Raytracing sind keine mehrfachen Renderingschritte mit anschließender Mittelung notwendig [Kir92]. Der Vorteil, dass Path Tracing N Strahlen durch ein Pixel schickt, lässt sich ausnutzen, um beide Effekte zu realisieren.

[9] Für ein Codebeispiel zu Depth of Field mit kreisförmiger Blende, siehe Anhang A, Codebeispiel 9.

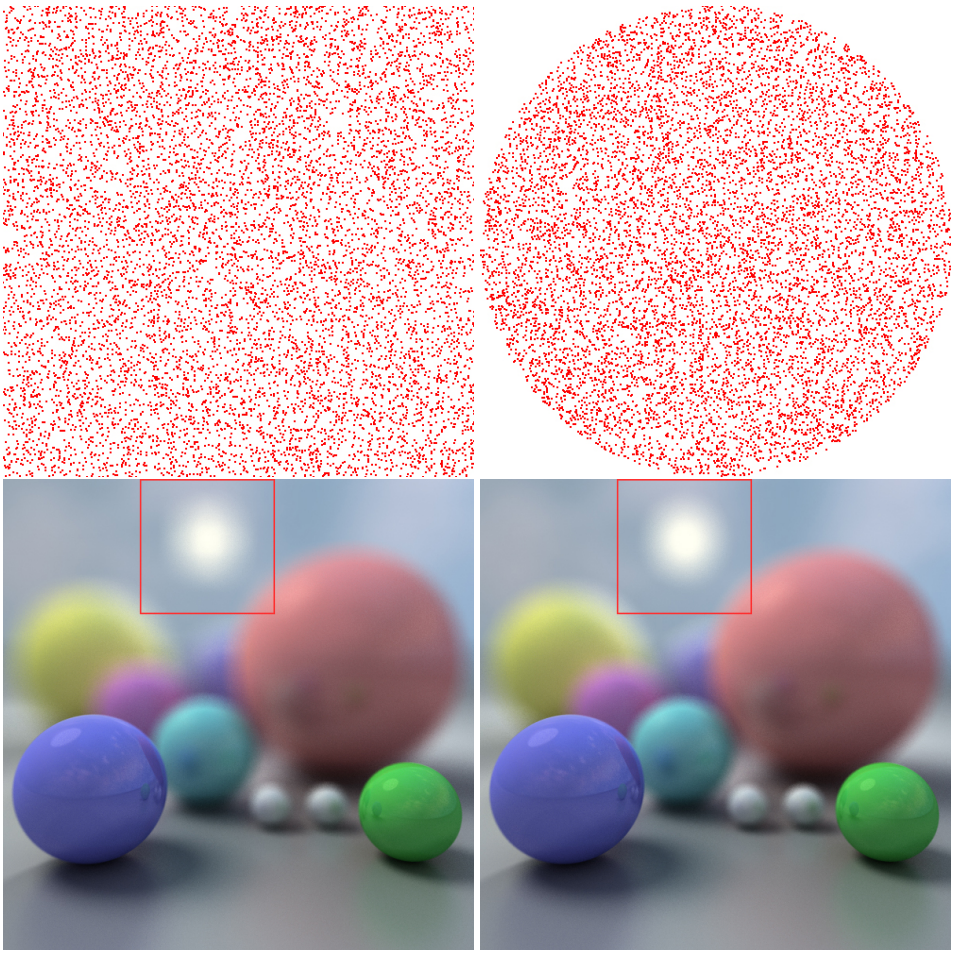


Abbildung 40: Links: Quadratische Blendenöffnung. Rechts: Kreisförmige Blendenöffnung.

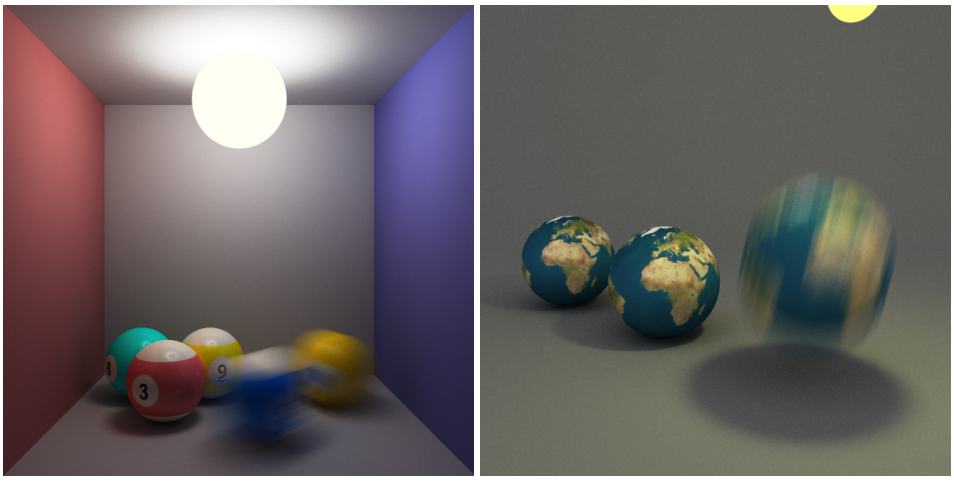
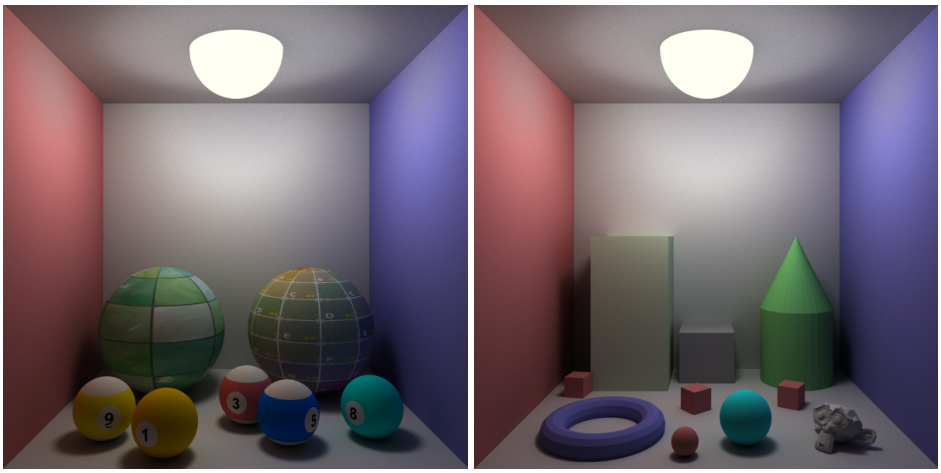


Abbildung 41: Motion Blur in verschiedene Richtungen.

6 Erweiterungen

Der implementierte Path Tracing Algorithmus ist erweiterbar, durch beispielsweise Texturen oder das Laden weiterer Objekte. Trifft ein Strahl auf ein Material mit einer Textur, wird zuerst die Texturfarbe am Schnittpunkt ausgewertet. Anschließend wird wieder die direkte und indirekte Beleuchtung berechnet (vgl. Abbildung 42a).
Aus Dreiecken zusammengesetzte Objekte können über einen Objectloader in die Szene geladen werden. Beim Laden der Objekte wird zusätzlich eine Bounding Box berechnet. Um Rechenzeit zu sparen, wird beim Schnittpunkttest zunächst die Bounding Box getestet. Wird die Box nicht geschnitten, müssen die darunter liegenden Dreiecke nicht getestet werden. Abbildung 42b zeigt verschiedene zusammengesetzte Objekte in der Cornell-Box.



(a) **(b)**

Abbildung 42: (a) Szene mit verschiedenen Texturen. (b) Zusammengesetzte Objekte aus Dreiecken sind über einen Objectloader möglich.

Dies ist ein einfacher Ansatz, um komplexere Meshes zu integrieren und den Path Tracer zu erweitern. Weiterführend können Datenstrukturen, wie eine Bounding Volume Hierarchy oder ein kd-Tree [PH10], implementiert werden, um den Rechenaufwand bei größerer Anzahl an Meshes zu verringern.
Seit Beginn des 21. Jahrhunderts ist das Interesse in der Forschung an echtzeitfähigen Raytracing Verfahren gewachsen. Heutzutage ist Raytracing bereits in der Lage, komplexe dynamische Szenen mit interaktiven Bildwiederholraten zu visualisieren. Der Grund dafür ist die Parallelisierung mithilfe der GPU und die Unterteilung der Szene durch geschickte Datenstrukturen. Ob es zukünftig auch echtzeitfähige Anwendungen mit Path Tracing geben wird, bleibt abzuwarten. [Abe09]

7 Fazit

Path Tracing ist ein globales Renderingverfahren, basierend auf Zufallsexperimenten. Mithilfe der Monte-Carlo-Integration kann die Rendergleichung, mit verschiedenen Dichtefunktionen für das direkte und indirekte Licht, berechnet werden. Es wurde gezeigt, dass mit unterschiedlichen Strategien beim Sampling[10] qualitativ bessere Ergebnisse erzielt werden können. Sampling nach gegebener Dichte ist Grundlage für die Zufallsvariablen. Diese haben Auswirkungen auf die Varianz und demzufolge auf das Bildrauschen.
Sampling einer Lichtquelle inmitten des Raumwinkels liefert gegenüber gleichverteilten Samples auf der Lichtoberfläche realistischere Schatten. Importance Sampling der Hemisphäre verringert die Varianz auf diffusen Oberflächen und konvergiert schneller zum Ergebnis. Durch Sampling nach dem Phong-Beleuchtungsmodell können zusätzlich glänzende Materialien visualisiert werden.
Path Tracing liefert außerdem qualitativ bessere Resultate durch Pixel Sampling. Es wurde gezeigt, dass unerwünschte Artefakte unterdrückt werden können. Depth of Field ist ein Effekt, der durch Sampling einer Blende realistische Schärfentiefe hervorruft. Dadurch wirken Bilder näher an der Realität. Mithilfe von Motion Blur kann der Eindruck von Bewegung im Bild, durch Veränderung der Position während der Strahlverfolgung, entstehen.
Mit Bezug auf Moore's law[11] besteht die Annahme, dass die Hardwareleistung in den nächsten Jahren fortlaufend steigt. Aus diesem Grund könnten zukünftig echtzeitfähige Anwendungen in der Lage sein, auf Path Tracing zurück zu greifen. Die parallele Berechnung mithilfe der GPU und die Unterteilung der Szene sind zu diesem Zweck nötig. Mit dieser Arbeit wurde die Grundlage für einen Renderer entwickelt, der die Anforderungen zum Rendern von physikalisch korrekten Bildern erfüllt.

[10] Für einen Überblick der behandelten Dichten, siehe Anhang B, Tabellen 4, 5 und 6.

[11] Moore's law geht auf Gordon Moore, Mitbegründer von Intel, zurück. Er prognostizierte in den sechziger Jahren, dass sich die Zahl der Transistoren von integrierten Schaltungen jährlich verdoppelt. http://www.britannica.com/EBchecked/topic/705881/Moores-law, abgerufen am 26. 11. 2013.

Anhang

A Dokumentierte Codebeispiele

```
#define SAMPLES 1000 // Maximale Anzahl an Samples
#define HEIGHT 600 // Höhe
#define WIDTH 900 // Breite

GLfloat *pixels, *tempPixels; // Speicher für Pixel

void display(){
    static int sample = 0;
    srand(sample++);

    // Schleife über alle Pixel des Bildschirms
    for (int y = 0; y < HEIGHT; y++){
        for (int x = 0; x < WIDTH; x++){

            // Berechne Kamerastrahl für das jeweilige Pixel
            Ray primaryRay(camPos, camDir, x, y);
            // Verfolge Kamerastrahl
            Color pixelColor = trace(primaryRay, 0);

            // Addiere Komponenten von pixelColor auf pixels
            pixels[(y * WIDTH + x) * 3] += pixelColor.r();
            pixels[(y * WIDTH + x) * 3 + 1] += pixelColor.g();
            pixels[(y * WIDTH + x) * 3 + 2] += pixelColor.b();
        }
    }

    // Pixelfarbe durch die Anzahl an Samples gewichten
    for (int s = 0; s < HEIGHT * WIDTH * 3; s++)
        tempPixels[s] = pixels[s] / (float)sample;

    // Zeichne Pixel im Framebuffer
    glDrawPixels(WIDTH, HEIGHT, GL_RGB, GL_FLOAT, tempPixels);
    glutSwapBuffers();

    // Solange wiederholen bis max. Samples erreicht wurden
    if (sample < SAMPLES)
        // Rufe display() erneut auf
        glutPostRedisplay();
    else
        // Framebuffer als Bitmap abspeichern
        snapshot(WIDTH, HEIGHT, "result.bmp");
}
```

Codebeispiel 1: display-Routine: Wird solange aufgerufen, bis maximale Anzahl an Samples erreicht wurde.

```
#define TRACE_DEPTH 5 // Rekursionstiefe
vector<Object*> objects; // vector für Objekte der Szene
Object *closestObject; // Objectpointer

Color trace(Ray ray, int depth){
    // Abbruch bei Überschreitung der Rekursionstiefe
    if (depth < TRACE_DEPTH){
        float tMin = FLT_MAX;
        closestObject = 0; // Nullpointer

        // berechne vordersten Schnittpunkt mit allen
        //Objekten der Szene
        for int o = 0; o < objects.size(); o++)
            if (objects.at(o)->findBBoxIntersect(ray)){
                float t = objects.at(o)->findIntersection(ray);
                if (t > 0 && t < tMin){
                    tMin = t;
                    // Objekt am nächsten zur Kamera
                    closestObject = objects.at(o);
                }
            }

        // kein Objekt getroffen oder
        //Objekt liegt hinter der Kamera
        if (closestObject == 0)
            return Color(0.1, 0.1, 0.1); // Hintergrundfarbe
            // alternativ Environment Map auswerten

        // Lichtquelle getroffen
        if (closestObject->getObjectMaterial().isLight())
            // Fehlerbehebung (Doppelauswertung)
            if (lightTwiceTest == false)
                return closestObject->getObjectColor();
            else{
                lightTwiceTest = false;
                return Color(0.0, 0.0, 0.0);
            }

        // Objekt getroffen, berechne Pixelfarbe
        else{
            // Position und Normale am Schnittpunkt berechnen
            Vector iPos(ray.getOrigin() + ray.getDir() * tMin);
            Vector iNormal(closestObject->getNormal(iPos));
            return shade(ray, depth, iPos, iNormal);
        }
    }else
        return Color(0.0, 0.0, 0.0);
}
```

Codebeispiel 2: trace-Routine: Kümmert sich um Schnittpunktberechnung.

```
Color shade(Ray ray, int depth, Vector iPos, Vector iNormal){
    lightTwiceTest = false;

    // Materialeigenschaften und Objektfarbe holen
    Material material(closestObject->getObjectMaterial());
    Color color(closestObject->getObjectColor());

    // vollkommen diffuses Material
    if (material.getDiffValue() == 1){
        // direkte Beleuchtung und Schatten auswerten
        Color directLight = compDirectLight(iPos, iNormal);

        Ray diffuseRay(calcDiffuseRay(intersectPos, iNormal));
        lightTwiceTest = true;

        // indirekte Beleuchtung
        Color indirectLight = trace(diffuseRay, depth++) * color;
        return directLight.colorAdd(indirectLight);
    }

    // vollkommen spiegelndes Material
    if (material.getSpecValue() == 1){
        Ray specRay(calcSpecRay(ray, iPos, iNormal, material));
        return trace(specRay, depth++) * color;
    }

    // vollkommen transmittierendes Material + Brechung
    if (material.getTransValue() == 1){
        Ray transRay(calcTransRay(ray, iPos, iNormal, material));
        return trace(transRay, depth++) * color;
    }

    // zusätzlich texturiertes Material
    if (material.isTextured()){
        // Texturfarbe an Schnittpunkt bestimmen
        Color texColor = compTextureColor(iPos);

        // direkte Beleuchtung mit Texturfarbe auswerten
        Color directLight = compDLTex(iPos, iNormal, texColor);

        Ray texRay(calcDiffuseRay(iPos, iNormal));
        lightTwiceTest = true;

        // indirekte Beleuchtung
        Color indirectLight = trace(texRay, depth++) * texColor;
        return directLight + indirectLight;
    }
}
```

Codebeispiel 3: shade-Routine: Wertet Material und Beleuchtung aus.

```
Color shade(Ray ray, int depth, Vector iPos, Vector iNormal){

    ...

    // kombiniertes Material -> Russian Roulette,
    //funktioniert nur, wenn diff + spec + trans <= 1 ist
    if ((material.getDiffValue() +
         material.getSpecValue() +
         material.getTransValue()) <= 1){

        float random = RANDOM1;

        if (random >= 0 &&
            random <= material.getDiffValue())
            // diffuser Pfad

        else if (random > material.getDiffValue() &&
                 random <= (material.getDiffValue() +
                            material.getSpecValue()))
            // spiegelnder Pfad

        else if (random > (material.getDiffValue() +
                           material.getSpecValue()) &&
                 random <= (material.getDiffValue() +
                            material.getSpecValue() +
                            material.getTransValue()))
            // transmittierender Pfad
    }
}
```

Codebeispiel 4: shade-Routine mit Russian Roulette.

```
Color compDirectLight(Vector iPos, Vector iNormal){
    Color directLight(0, 0, 0);

    // Schleife über alle Lichtquellen
    for (int l = 0; l < objects.size(); l++){
        if (objects.at(l)->isLight()){

            // Zufallszahlen zwischen 0 und 1
            float Xi1 = RANDOM1;
            float Xi2 = RANDOM1;

            Vector lightCenter(objects.at(l)->getCenter());
            float lightRadius(objects.at(l)->getRadius());

            float dist = (lightCenter - iPos).length();

            // maximaler Öffnungswinkel
            float thetaMax = asin(lightRadius / dist);

            // mit inverser CDF Methode bestimmt man die beiden
            //sphärischen Winkel
            float theta = acos(1.0 - Xi1 + Xi1 * cos(thetaMax));
            float phi = 2.0 * M_PI * Xi2;

            // Konvertierung in kartesische Koordinaten
            Vector psi(sin(theta) * cos(phi),
                       sin(theta) * sin(phi),
                       cos(theta));

            // die lokalen Koordinaten müssen bestimmt werden,
            //v-Achse entspricht dem Vektor vom
            //Schnittpunkt zum Lichtmittelpunkt
            Vector w(lightCenter - iPos);
            w = w.normalize();

            Vector t(0.0, 0.0, 0.0); // Hilfsvektor t
            if(fabs(w.getX()) >= fabs(w.getY()))
                t = Vector(0.0, 1.0, 0.0);
            else
                t = Vector(1.0, 0.0, 0.0);

            // die beiden anderen Achsen liegen senkrecht zu w
            Vector u = (w.crossProduct(t)).normalize();
            Vector v = (w.crossProduct(u)).normalize();

            // psi = x' in Bachelor Arbeit
            psi = u * psi.getX() +
                  v * psi.getY() +
                  w * psi.getZ();
```

```
psi = psi.normalize();

Ray r(iPos + psi * TRACE_BIAS, psi);

// berechne Schnittpunkt auf der Lichtquelle
float tMin2 = FLT_MAX;
float tL = objects.at(l)->findIntersection(r);
if (tL > 0 && tL < tMin2)
    tMin2 = tL;

// Punkt auf der Lichtquelle innerhalb
//des Raumwinkels gefunden
Vector pointOnLight(iPos + psi * tMin2);
Vector lightDir(pointOnLight - iPos);

float lengthSq = pow(lightDir.length(), 2.0);

lightDir = lightDir.normalize();

// Kosinus vom Winkel zwischen der Normalen
//und der Lichteinfallsrichtung
float cosinus = iNormal.dotProduct(lightDir);

// Skalarprodukt zwischen Normale und
//Einfallsrichtung <= 0, dann keine Beleuchtung
if (cosinus > 0.0){
    // Schattenstrahl bestimmen
    Ray shadowRay(iPos+lightDir * TRACE_BIAS, lightDir);

    // Test für Schatten
    bool isInShadow = false;
    float tMin = FLT_MAX;

    // berechne Schnittpunkte des Schattenstrahls mit
    //allen Objekten der Szene, bis auf Lichtquellen
    for (int o = 0; o < objects.size(); o++){
        if (!objects.at(o)->isLight()){
            if (objects.at(o)->
                    findBBoxIntersect(shadowRay)){
                float t = objects.at(o)->
                    findIntersection(shadowRay);
                if (t > 0 && t < tMin){
                    tMin = t;
                    if (tMin * tMin < lengthSq){
                        isInShadow = true;
                        // sobald Schatten festgestellt,
                        //for-Schleife abbrechen
                        break;
                    }
```

```
                    }
                }

            if (!isInShadow){
                Color objColor(closestObject->getObjColor());
                Color lightCol(objects.at(l)->getObjColor());
                Color I = lightCol * objects.at(l)->getCand();
                Color E = I * cosinus / lengthSq);

                directLight += objColor * I * cosinus * 2.0 *
                               (1.0 - cos(thetaMax));
            }
          }
        }
    }
    return directLight;
}
```

Codebeispiel 5: compdirectLight-Routine: Kümmert sich um Sampling der Lichtquelle. Samples werden auf der Oberfläche der Lichtquelle inmitten des Raumwinkels generiert.

```
Vector importanceSampling(Vector normal){

    // Zufallszahlen zwischen 0 und 1
    float Xi1 = RANDOM1;
    float Xi2 = RANDOM1;

    // mit inverser CDF Methode bestimmt man die beiden
    //sphärischen Winkel
    float theta = acos(sqrt(Xi1));
    float phi = 2.0 * M_PI * Xi2;

    // Konvertierung in kartesische Koordinaten
    float x = sin(theta) * cos(phi);
    float y = cos(theta);
    float z = sin(theta) * sin(phi);

    // die lokalen Koordinaten müssen bestimmt werden,
    //v-Achse entspricht dem Normalenvektor
    Vector v = normal;

    // Hilfsvektor t
    Vector t = v;

    if (fabs(t.getX()) <= fabs(t.getY()) &&
        fabs(t.getX()) <= fabs(t.getZ()))
        t = Vector(1.0, t.getY(), t.getZ());
    else if (fabs(t.getY()) <= fabs(t.getX()) &&
             fabs(t.getY()) <= fabs(t.getZ()))
        t = Vector(t.getX(), 1.0, t.getZ());
    else
        t = Vector(t.getX(), t.getY(), 1.0);

    // die beiden anderen Achsen liegen senkrecht zu v
    Vector u = (t.crossProduct(v)).normalize();
    Vector w = (u.crossProduct(v)).normalize();

    // Rotation um lokales Koordinatensystem
    return Vector(u * x + v * y + w * z).normalize();
}
```

Codebeispiel 6: importanceSampling-Routine: Sampling der Hemisphäre mit Kosinus Verteilung.

```
Vector glossyReflection(Vector reflectionVector,
                        Vector normal,
                        float phongExp){

    // Zufallszahlen zwischen 0 und 1
    float Xi1 = RANDOM1;
    float Xi2 = RANDOM1;

    // mit inverser CDF Methode bestimmt man die beiden
    //sphärischen Winkel
    float theta = acos(pow(Xi1, 1.0 / (phongExp + 1.0)));
    float phi = 2.0 * M_PI * Xi2;

    // Konvertierung in kartesische Koordinaten
    float x = sin(theta) * cos(phi);
    float y = cos(theta);
    float z = sin(theta) * sin(phi);

    // die lokalen Koordinaten müssen bestimmt werden,
    //v-Achse entspricht dem reflektierten Vektor
    Vector v = reflectionVector;
    Vector t = v;
    // Hilfsvektor t bestimmen
    if (fabs(t.getX()) <= fabs(t.getY()) &&
        fabs(t.getX()) <= fabs(t.getZ()))
        t = Vector(1.0, t.getY(), t.getZ());
    else if (fabs(t.getY()) <= fabs(t.getX()) &&
             fabs(t.getY()) <= fabs(t.getZ()))
        t = Vector(t.getX(), 1.0, t.getZ());
    else
        t = Vector(t.getX(), t.getY(), 1.0);

    // die beiden anderen Achsen liegen senkrecht zu v
    Vector u = (t.crossProduct(v)).normalize();
    Vector w = (u.crossProduct(v)).normalize();

    Vector randomDir(u * x + v * y + w * z);
    randomDir = randomDir.normalize();

    // Test ob Richtung in negativem Halbraum
    if (randomDir.dotProduct(normal) < 0.0)
        return randomDir.negative();
    else
        return randomDir;
}
```

Codebeispiel 7: glossyReflection-Routine: Sampling der BRDF nach dem Phong-Beleuchtungsmodell. Der Phong-Exponent entspricht der Rauheit des Materials.

```
float Camera::haltonSequence(int sampleIndex, int base){
    float result = 0;
    float f = 1.0 / base;
    int i = sampleIndex;

    while (i > 0){
        result = result + f * (i % base);

        // "floorf" rundet auf den nächst kleineren Wert
        i = floorf(i / base);
        f = f / base;
    }

    // die Halton Sequenz liefert Werte zwischen 0 und 1,
    //da auf die Pixelmitte addiert wird, muss das
    //Ergebnis zwischen -0.5 und 0.5 liegen
    return (result - 0.5);
}
```

Codebeispiel 8: haltonSequence-Routine: Erstellt Quasi-Random Numbers.

```
void display(){

    ...

    // Schleife über alle Pixel des Bildschirms
    for (int y = 0; y < HEIGHT; y++){
        for (int x = 0; x < WIDTH; x++){

            // Berechne Kamerastrahl für das jeweilige Pixel
            Ray primaryRay(camPos, camDir, x, y);

            // gibt die Tiefe der Fokusfläche an,
            //dort liegen die Objekte im Fokus
            float focalLength = 8.5;

            // gibt die Stärke der Unschärfe von
            //Objekten außerhalb der Fokusfläche an
            float apertureSize = 0.35;

            // Samples für eine Kreisscheibe
            float r = sqrt(RANDOM1) * apertureSize;
            float phi = 2 * M_PI * RANDOM1;

            // Brennpunkt
            Vector focalPoint(camPos + primaryRay.getRayDir()
                              * focalLength);

            // kreisfoermige Blendenoeffnung
            Vector jitterCirclePos(camPos +
                                   camLeft * cos(phi) * r +
                                   camDown * sin(phi) * r);

            Vector newDir = focalPoint - jitterCirclePos;
            Ray primaryRayDOF(jitterCirclePos, newDir);

            // Verfolge Strahl mit Depth of Field Effekt
            Color pixelColor = trace(primaryRayDOF, 0);

            ...

        }
    }
}
```

Codebeispiel 9: display-Routine mit Depth of Field.

B Zusätzliche Tabellen & Abbildungen

Sampling	Dichte
Kreisscheibe	$p(r,\varphi) = \frac{1}{\pi \cdot R^2}$
Kugel	$p(\theta,\varphi) = \frac{1}{4\pi r^2}$
Raumwinkel	$p(\theta,\varphi) = \frac{1}{2\pi(1-\cos\theta_{max})}$
Hemisphäre Uniform	$p(r,\varphi) = \frac{1}{2\pi}$
Hemisphäre Importance	$p(\theta,\varphi) = \frac{\cos\theta}{\pi}$
Hemisphäre Phong	$p(\theta,\varphi) = \frac{n+1}{2\pi}\cos^n\theta$

Tabelle 4: Übersicht Dichtefunktionen.

Sampling	Samples
Kreisscheibe	$r = R\sqrt{\xi_1}$ $\varphi = 2\pi \cdot \xi_2$
Kugel	$\theta = \arccos(1-2\xi_1)$ $\varphi = 2\pi\xi_2$
Raumwinkel	$\theta = \arccos\left(1-\xi_1+\xi_1\sqrt{1-\left(\frac{r}{\|x-c\|}\right)^2}\right)$ $\varphi = 2\pi\xi_2$
Hemisphäre Uniform	$r = \sqrt{1-\xi_1^2}$ $\varphi = 2\pi\xi_2$
Hemisphäre Importance	$\theta = \arccos(\xi_1)$ $\varphi = 2\pi\xi_2$
Hemisphäre Phong	$\theta = \arccos((\xi_1)^{\frac{1}{n+1}})$ $\varphi = 2\pi\xi_2$

Tabelle 5: Übersicht Samples.

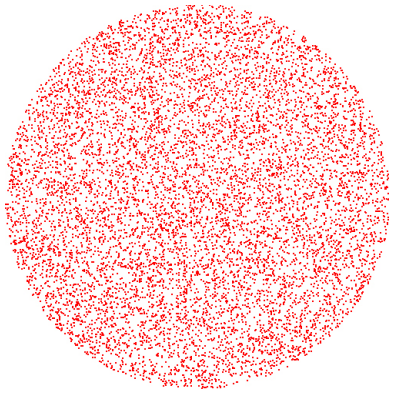
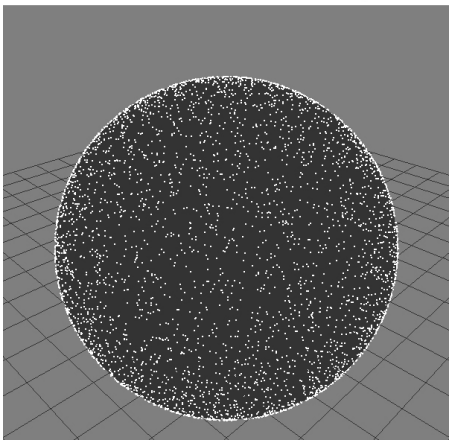
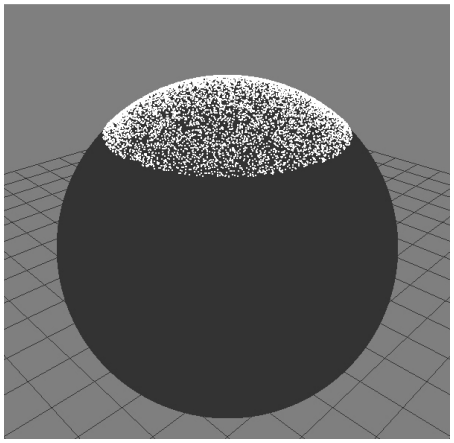
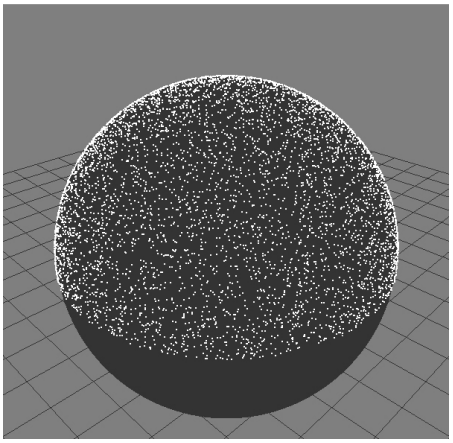
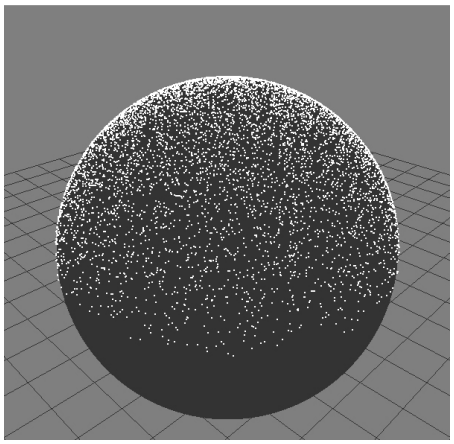
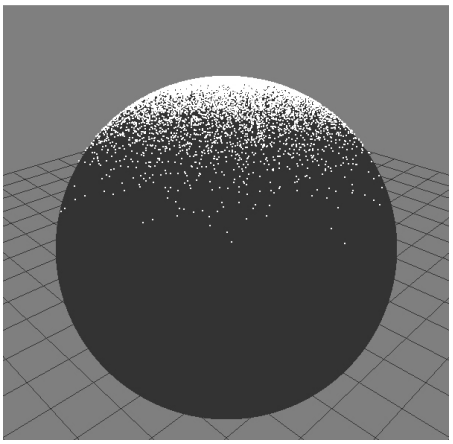
Kreisscheibe	Kugel
	
Raumwinkel	**Hemisphäre Uniform**
	
Hemisphäre Importance	**Hemisphäre Phong**
	

Tabelle 6: Sampling Beispiele.

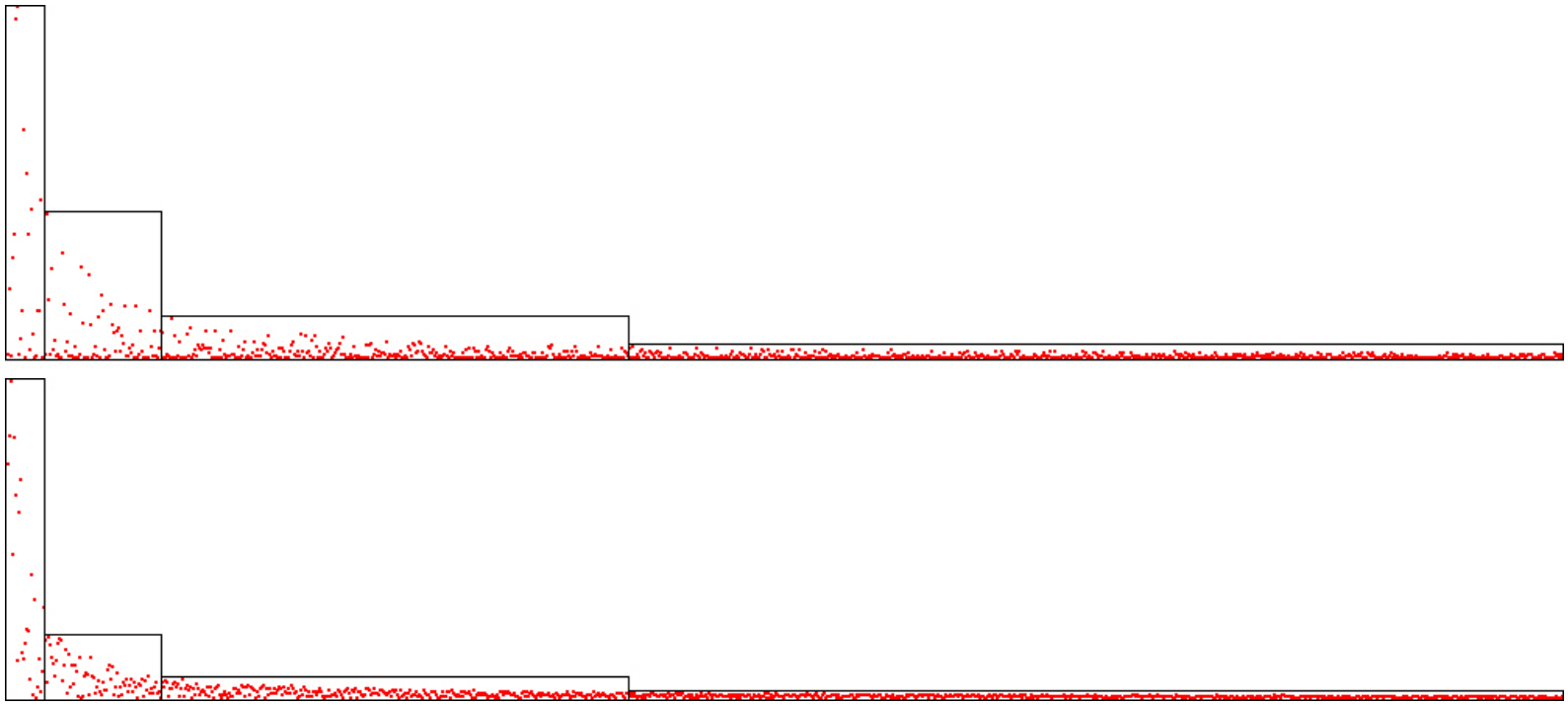


Abbildung 43: Maximale Abweichung der Schätzung des Integrals über $f(x) = x^3$ zum korrekten Ergebnis nach 25, 100, 400 und 1000 Samples. Oben: Dichte $p(x) = 1 = const$. Unten: $p(x) = 2x$. Mit beiden Dichtefunktionen konvergiert die Monte-Carlo-Schätzung zum Ergebnis, jedoch unterschiedlich schnell.

Literatur

[Abe09] O. Abert. *Augenblick: ein effizientes Framework für Echtzeit Ray Tracing*. Eul, 2009.

[AHJ+01] James Arvo, Pat Hanrahan, Henrik Wann Jensen, Henrik Wann Jensen, Don Mitchell, Matt Pharr, Peter Shirley, Jim Arvo, Marcos Fajardo, and Marcos Fajardo. State of the art in monte carlo ray tracing for realistic image synthesis, 2001.

[DBBS06] Philip Dutre, Kavita Bala, Philippe Bekaert, and Peter Shirley. *Advanced Global Illumination*. AK Peters Ltd, 2006.

[DH04] H. Dehling and B. Haupt. *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Statistik und ihre Anwendungen. Springer, 2004.

[DHM+01] Philip Dutre, Paul Heckbert, Vincent Ma, Fabio Pellacini, Robert Porschka, Mahesh Ramasubramanian, Cyril Soler, and Greg Ward. Global illumination compendium, 2001.

[Geo09] H.O. Georgii. *Stochastik: Einführung in die Wahrscheinlichkeitstheorie und Statistik*. De Gruyter Lehrbuch. De Gruyter, 2009.

[Gla89] A.S. Glassner. *An Introduction to Ray Tracing*. Academic Press. Academic Press, 1989.

[Jen01] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[Kaj86] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM.

[Kir92] David Kirk. *Graphics Gems III*. Graphics gems series. Academic Press, 1992.

[Laf96] Eric Lafortune. Mathematical models and monte carlo algorithms for physically based rendering. Technical report, 1996.

[MT97] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *J. Graph. Tools*, 2(1):21–28, October 1997.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[SW92] Peter Shirley and Changyaw Wang. Distribution ray tracing: Theory and practice. In *In Proceedings of the Third Eurographics Workshop on Rendering*, pages 33–43, 1992.

[SWZ96] Peter Shirley, Changyaw Wang, and Kurt Zimmermann. Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15:1–36, 1996.

[VG95] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 419–428, New York, NY, USA, 1995. ACM.