Brandenburgische Technische Universität Cottbus
Institut für Informatik, Informations- und Medientechnik
Lehrstuhl Grafische Systeme

Studienarbeit

von Ralf Kopsch
Matrikel: 2015737

Erweiterung eines bestehenden Raytracers um Photon-Mapping sowie Radiosity

Betreuer:
Prof. Dr. Winfried Kurth

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Cottbus, 10. April 2008 _____

Unterschrift

Danksagung

Ich möchte mich bedanken

bei Herrn Prof. Dr. Winfried Kurth, Ole Kniemeyer und Reinhard Hemmerling für die Betreuung meiner Arbeit,

bei Hagen Steidelmüller für lange und anregenden Diskussionen über die verschiedenen Algorithmen,

bei André Preussner für das Korrekturlesen, die konstruktive Kritik und das rege Interesse,

und nicht zuletzt bei meinen Eltern für ihr Vertrauen und den bedingungslosen Rückhalt, den ich immer wieder bei ihnen finden konnte.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Am Lehrstuhl für Grafische Systeme der BTU Cottbus wird die 3D-Modellierungsplattform *GroIMP* [Gro07] entwickelt (siehe Kapitel 2). Diese besitzt bereits einen einfachen Raytracer, welcher um stochastische Verfahren sowie Photon-Mapping und Radiosity erweitert werden soll.
Die Erweiterung ist eine gemeinschaftliche Studienarbeit von Hagen Steidelmüller und mir. Dabei haben wir die Raytracing Verfahren gleichmäßig zwischen uns aufgeteilt. Hagen hat einen bidirektionalen Raytracer und den Metropolis-Algorithmus implementiert. Diese Arbeit beschreibt mein Vorgehen bei der Implementierung des Photon-Mapping- und Radiosity-Verfahrens.

1.1 Motivation

Durch meine Erweiterung soll erreicht werden, dass Szenen möglichst schnell aus verschiedenen Blickwinkeln gerendert werden können. Dafür sind Photon-Maping und Radiosity besonders gut geeignet, da bei beiden der erste und aufwendige Teil der Berechnung von der Position der Kamera unabhängig ist. Während einer Vorverarbeitungsphase werden diese Berechnungen durchgeführt. Danach kann man die Kamera beliebig in der Szene umher schwenken und sich den ausgewählten Bereich schnell darstellen lassen.
Somit lässt sich der Komfort für den GroIMP-Nutzer weiter erhöhen.

1.2 Zielsetzung

Ziel dieser Arbeit ist es, das Vorgehen bei der Implementierung des Photon-Mapping- und Radiosity-Verfahrens zu beschreiben. Weiterhin wird erläutert, wie die Verfahren zu benutzen sind und was bei deren Einsatz zu beachten ist.
Diese Arbeit soll außerdem für Personen, welche die implementierten Algorithmen erweitern möchten, einen Ausgangspunkt darstellen.

1.3 Aufbau der Arbeit

Kapitel 2 gibt einen kurzen Überblick über die 3D-Modellierungsplattform GroIMP.
In Kapitel 3 wird das Photonen-Map-Verfahren vorgestellt. Hierbei wird zunächst das allgemeine Verfahren und anschließend die Implementierung beschrieben. Dabei werden auch die Parameter erläutert, welche die Ausgabe beeinflussen. Am Ende des Kapitels werden einige Beispielbilder gezeigt und es werden mögliche Erweiterungen vorgestellt.
Kapitel 4 beschreibt das Radiosity-Verfahren und die zugehörigen Parameter.
Im letzten Kapitel werden Photonen-Map- und Radiosity-Verfahren gegenübergestellt und die Unterschiede diskutiert.

An diese Arbeit schließt sich direkt die Arbeit von Hagen Steidelmüller an. Darin liegt der Schwerpunkt bei den stochastischen Rendering-Verfahren. Es wird der bidirektionale Pathtracer und das Metropolis-Verfahren beschrieben, sowie Vor- und Nachteile erläutert. Am Ende von Hagens Arbeit werden die von ihm implementierten Verfahren gegenübergestellt, dabei werden die gleichen Szenen genutzt, wie im Abschlusskapitel dieser Arbeit. Dadurch sollte es möglich sein, sich eine Übersicht über alle von uns implementierten Rendering-Verfahren zu verschaffen.

2 GroIMP

Dieses Kapitel gibt eine kleine Übersicht über die Modellierungsplattform *GroIMP*. Diese soll nicht vollständig sein. Es werden lediglich die Einstellungen beschrieben, die nötig sind, um eine Szene mittels Raytracer zu rendern.

2.1 GroIMP Übersicht

GroIMP steht für „Growth Grammar-related Interactive Modelling Platform“ [Gro07] und wird am Lehrstuhl Grafische Systeme der BTU Cottbus entwickelt. Es handelt sich um eine in Java programmierte 3D-Modellierungsplattform, welche sich durch die Möglichkeit auszeichnet, Modelle mithilfe von Wachstumsgrammatiken zu erstellen.

2.2 GroIMP Raytracing-Optionen

Mit dem Konfigurationfenster bietet GroIMP dem Nutzer einen einfachen und flexiblen Weg, das Verhalten der verschiedenen Programmkomponenten zu steuern. In diesem ist es auch möglich, die in dieser Arbeit vorgestellten Rendering-Verfahren auszuwählen und zu konfigurieren.
Um das in Abbildung 2.1 dargestellte Konfigurationsfenster zu öffnen, wählen Sie in GroIMP den Menüpunkt «Panels → Preferences».
Im linken Teil des Fensters zeigt eine Baumstruktur die verschiedenen Programmkomponenten und deren Unterkomponenten. Wählt man hier «Renderer → Twilight», dann erscheinen die Optionen des Raytracers. Unter der Option «Ray processor» kann man den Raytracer-Algorithmus anwählen, der zum Rendern eines Bildes genutzt werden soll. Wird hier «Photon Mapping» ausgewählt, so wird der in Kapitel 3 beschriebene Photon-Mapping-Algorithmus angewendet. Durch die Auswahl von «Radiosity» wird das in Kapitel 4 beschriebene Verfahren genutzt.

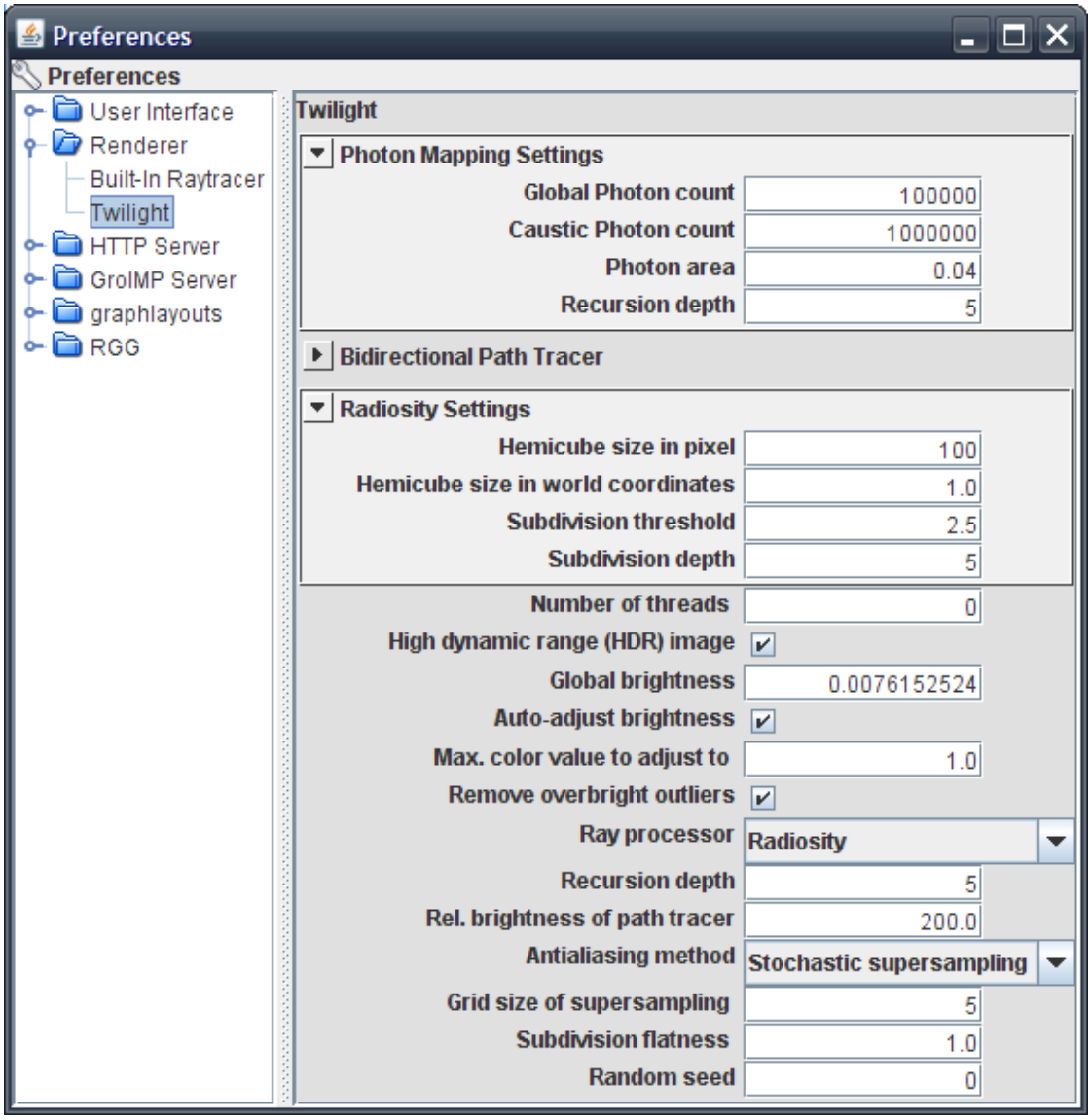


Abbildung 2.1: Die GroIMP Raytracing-Optionen

Im oberen Bereich des Fensters befindet sich zu jedem Algorithmus eine Gruppierungsbox mit Parametern, welche die Qualität eines mittels des jeweiligen Algorithmus gerenderten Bildes beeinflussen. Die Parameter für das Photonen-Mapping werden im Abschnitt 3.3.1 näher erläutert, die für das Radiosity-Verfahren im Abschnitt 4.2.8.

3 Photon Mapping

In diesem Kapitel wird das Photonen-Map-Verfahren vorgestellt. Das Verfahren wurde 1995 von Henrik Wann Jensen entwickelt [JC95]. Es handelt sich dabei um eine Erweiterung des Raytracing-basierten Verfahrens. Das Ziel von Photon Mapping ist es, die globale Beleuchtung einer Szene effizient zu ermitteln und somit realistische Bilder bei geringerem Zeitaufwand zu erzeugen.
Es lässt sich in zwei Phasen unterteilen. In der ersten Phase werden ausgehend von einer Lichtquelle Photonen in die Szene gesendet. Bei jeder Interaktion mit einer diffusen Oberfläche wird ein Eintrag in einer Photonen-Map abgelegt, was in Abschnitt 3.1 näher erläutert wird. In der im Abschnitt 3.2 beschriebenen zweiten Phase wird aus der so erzeugten Repräsentation die Beleuchtung der Szene berechnet [Ott99].

3.1 Erzeugung der Photonen-Map

Im ersten Teil des Algorithmus wird eine grobe Repräsentation der Lichtverteilung erstellt und in einer Photonen-Map gespeichert. Dafür speichert sie die 3D-Koordinaten, an welchen die Photonen mit Szenenobjekten interagieren, und die Energie des jeweiligen Photons zu diesem Zeitpunkt.
Diese Photon-Map ist unabhängig vom Standort des Betrachters und muss daher nur einmal berechnet werden. Danach kann die Szene für verschiedene Betrachtungswinkel visualisiert werden.

3.1.1 Aussenden der Photonen

Es werden kleine Energiepartikel, die Photonen, von den Lichtquellen in die Szene geschossen. Der folgende Pseudo-Code beschreibt die allgemeine Vorgehensweise.

```
while ( Photonenanzahl != 0 ) {
    wähle eine Lichtquelle l aus;
    wähle einen Punkt p auf l aus;
    wähle Austrittsrichtung w für p aus;
    schießePhoton(l, p, w);
    Photonenanzahl--;
}
```

Listing 3.1: Pseudo-Code für das Versenden von Photonen

Die Energie einer Lichtquelle wird gleichmäßig auf die von ihr ausgesandten Photonen aufgeteilt. Die von einem Photon transportierte Energie ergibt sich aus der folgenden Gleichung:

$$L_{Photon} = \frac{L_{Lichtquelle}}{Anzahl\ der\ Photonen} \tag{3.1}$$

Existieren in der Szene verschiedene Lichtquellen, so ist die Anzahl der auszusendenden Photonen entsprechend der Energie und Größe auf diese zu verteilen. Eine Lichtquelle mit doppelter Intensität sollte demnach auch doppelt so viele Photonen aussenden [Maa01].

3.1.2 Verfolgen der Photonen

Die ausgesandten Photonen werden durch die Szene verfolgt. Kommt es zu einer Interaktion mit einem Medium, so wird das Photon an dieser Position in der Photonen-Map abgelegt. Danach wird mittels der Reflexionseigenschaften des getroffenen Objektes eine neue Richtung bestimmt und das Photon in diese weiter verfolgt.
Um Kaustik-Effekte zu erreichen, werden Photonen auf zwei verschiedene Arten in die Szene gesendet. Die erste Art sind globale Photonen; diese werden an allen diffus reflektierenden Objekten gespeichert, auf die sie treffen, und erzeugen so die indirekte diffuse Beleuchtung der Szene. Die zweite Art sind die Kaustik-Photonen, die nur in die Photonen-Map eingetragen werden, wenn sie nach beliebig vielen Interaktionen mit spekularen oder transparenten Objekten das erste Mal auf ein diffus reflektierendes Objekt treffen. Die Bilder a) und b) der Abbildung 3.2 zeigen den Unterschied zwischen globalen und Kaustik-Photonen.

3.1.3 Speicherung der Photonen

Damit das Photon-Mapping-Verfahren möglichst schnell ist, ist eine effiziente Datenstruktur für das Speichern der Photonen unerlässlich. Die Hauptfunktion, welche die Photon-Map bereitstellen muss, ist die möglichst effiziente Bestimmung aller Punkte in einer bestimmten Umgebung. Dies ist für eine schnelle Visualisierung der Szene nötig (siehe Abschnitt 3.2). Eine Bereichsabfrage wird effizient von einem k-d-Baum bereitgestellt. Dabei handelt es sich um einen k-dimensionalen Suchbaum [Kur07]. Für unsere Anwendung ist nur der Fall $k = 3$ interessant. Bei dieser Datenstruktur wird der Raum sukzessive durch achsenparallele Hyperebenen unterteilt. Diese Splithyperebenen verlaufen senkrecht zu den Koordinatenachsen. Bei jedem Unterteilungsschritt wechselt die Koordinatenachse zyklisch: $(x, y, z, x, y, ...)$

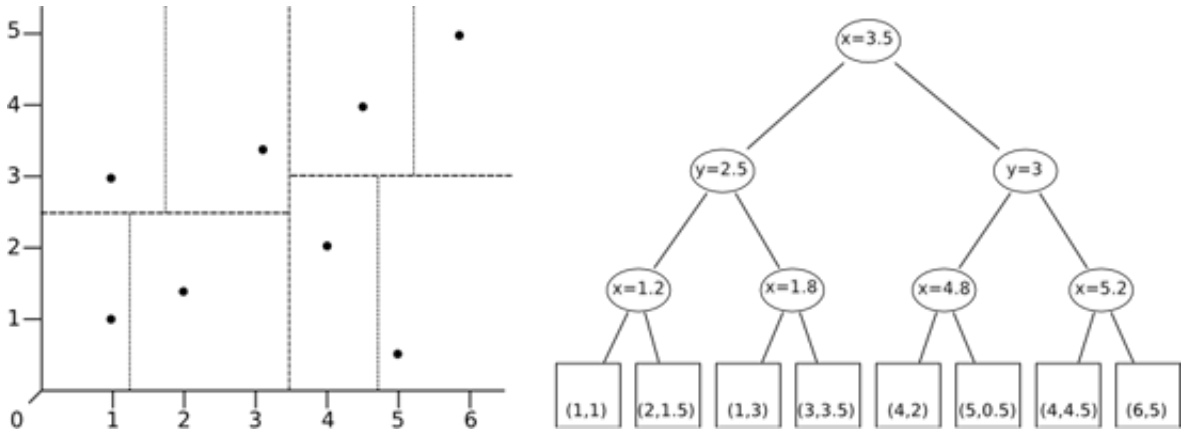


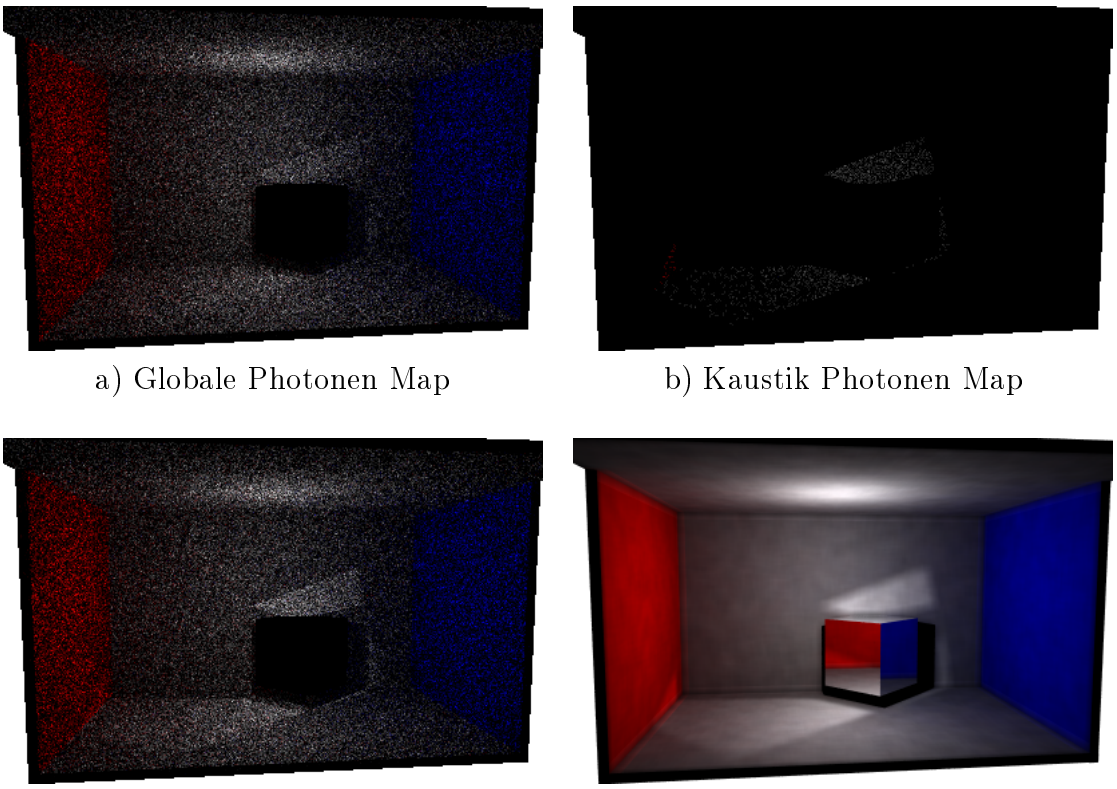
Abbildung 3.1: Beispiel für einen 2D-Baum

Abbildung 3.1 zeigt einen k-d-Baum, bei welchem zur besseren Lesbarkeit $k = 2$ gewählt wurde. Die inneren Knoten des Baumes entsprechen dabei den Splitgeraden. Die zu speichernden Punkte werden in den Blättern abgelegt.
Für die Implementierung wurde auf eine fertige Java-Bibliothek zurückgegriffen, welche eine k-d-Baum-Datenstruktur zur Verfügung stellt [Lev07].

3.2 Visualisierung

Der zweite Teil des Algorithmus besteht aus der Visualisierung der Szene mithilfe der zuvor erstellten Photonen-Map. Für jeden Pixel des Bildes wird ein Strahl durch die Kamera-Ebene geschickt und durch die Szene verfolgt. Trifft der Strahl ein Objekt, so wird in dessen näherer Umgebung nach Einträgen in der Photonen-Map gesucht. Diese Suche wird in einem kugelförmigen Bereich durchgeführt. Dabei kann es allerdings zu Fehlbeleuchtungen an Objektkanten kommen, da in der Kugel auch Photonen enthalten sind, die nicht vom Auftreffpunkt aus sichtbar sind. Daher wird für die Suche nach Photonen keine Kugel, sondern ein Kreis um diesen Punkt eingesetzt, dessen Radius durch den Parameter «Photon area» festgelegt ist (siehe Abschnitt 3.3.1).
Alle in diesem Bereich gefundenen Photonen werden aufsummiert und das Ergebnis durch den Flächeninhalt des Kreises dividiert. Zu diesem Wert wird noch die Eigenfarbe des getroffenen Objektes hinzugerechnet. Danach wird die Reflexionsrichtung des Strahls bestimmt und dieser weiterhin durch die Szene verfolgt [YLS05].

Wie in Abschnitt 3.1.2 erläutert gibt es zwei Arten von Photonen, die nacheinander in die Szene gesendet werden. Dadurch wird die Photonen-Map in zwei Schritten aufgebaut, was in Abbildung 3.2 für eine Beispielszene dargestellt ist. Auf den Bildern a), b) und c) wurde jeweils die berechnete Photonen-Map direkt visualisiert. Dabei stellt a) den Inhalt der Map nach dem Aussenden von 500.000 globalen Photonen dar und b) das Ergebnis nach 500.000 Kaustik-Photonen. In Abbildung c) ist die Kombination dieser beiden Photonen-Arten dargestellt. Das fertig gerenderte Bild ist in Abbildung d) zu sehen.



a) Globale Photonen Map

b) Kaustik Photonen Map

c) Kombinierte Photonen Map

d) das fertig gerenderte Bild

Abbildung 3.2: Aufbau eines Photon Map Bildes

3.3 Implementierung

Das Photonen-Mapping-Verfahren wird durch die folgenden drei Klassen implementiert: `PhotonMapRayProcessor`, `PhotonMap` und `OptionReader`. Diese liegen in den Paketen `de.grogra.ray2.tracing` bzw. `de.grogra.ray2.photonmap`.

Die Klasse `OptionReader` ist für das Einlesen der Parameter zuständig (siehe Abschnitt 3.3.1). Weiterhin überprüft sie, ob die Szene geändert wurde und dadurch ein erneutes Berechnen der Photonen-Map erforderlich ist. Sollte dies der Fall sein, so liefert die Funktion `isPhotonMapCalcNeeded()` den Wert *true* zurück.

Die Klasse `PhotonMap` stellt eine Photonen-Map-Datenstruktur bereit. Sie besitzt zwei wichtige Funktionen. Als erstes die Funktion `insertPhoton()`, welche ein neues Pho-

ton in die Datenstruktur einfügt, und die Funktion `sumPhotons()`, welche die Farbe für einen gegebenen Punkt berechnet.

Der größte Teil der Berechnungen findet in der Klasse `PhotonMapRayProcessor` statt. Die Funktion `createPhotonMap()` erzeugt für eine gegebene Szene eine neue Photonen-Map, indem zunächst mithilfe der Funktion `computeLightProbs()` die Anzahl der Photonen pro Lichtquelle berechnet wird. Danach werden die globalen und Kaustik-Photonen wie in Abschnitt 3.1.1 beschrieben ausgesendet. Dazu wird die Funktion `shootPhoton()` aufgerufen, welche die Photonen durch die Szene verfolgt und sie in die Photonen-Map einträgt.
Wenn die Photonen-Map erstellt ist, so muss noch die Szene gerendert werden. Dafür ist die Funktion `traceRay()` zuständig, welche die Farbe für einen gegebenen Pixel unter Berücksichtigung der Werte in der Photonen-Map bestimmt.

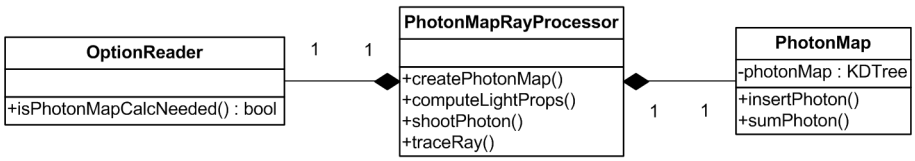


Abbildung 3.3: Klassendiagramm für das Photon-Mapping

3.3.1 Parameter

Es gibt vier wichtige Parameter, die die Qualität eines mittels Photon-Mapping gerenderten Bildes beeinflussen. Dies sind «Global Photon count», «Caustic Photon count», «Photon area» und «Recursion depth». Diese Parameter können in den GroIMP-Programmoptionen geändert werden (siehe Kapitel 2.2).

Global Photon count legt fest, wie viele Photonen in die Szene ausgesendet werden sollen. Je höher dieser Wert, desto exakter wird die Szene ausgeleuchtet. Bei Szenen, bei welchen nur die Kaustik-Effekte von Interesse sind, kann es von Vorteil sein, diesen Wert auf 0 zu setzten.

Caustic Photon count legt fest, wie viele Kaustik-Photonen in die Szene ausgesendet werden sollen. Kaustik-Photonen werden nur in der Photonen-Map gespeichert, wenn die erste Interaktion mit einem spiegelnden oder transparenten Objekt stattfindet. Wie der Name bereits vermuten lässt, sind diese Photonen für Kaustik-Effekte im gerenderten Bild verantwortlich. Eine Erhöhung dieses Wertes sorgt für eine stärkere und detailliertere Sichtbarkeit von Kaustik-Effekten. Bei einer Szene mit ausschließlich diffusen Oberflächen kann dieser Wert auf 0 gesetzt werden, um Rechenzeit zu sparen.

Photon area bestimmt die Größe der Fläche, welche beim Rendern des Bildes nach Photonen abgetastet wird. Je höher dieser Wert, desto heller wird die Szene ausgeleuchtet. Ein geringerer Wert sorgt für exaktere Beleuchtung. Negative Werte sind nicht möglich.

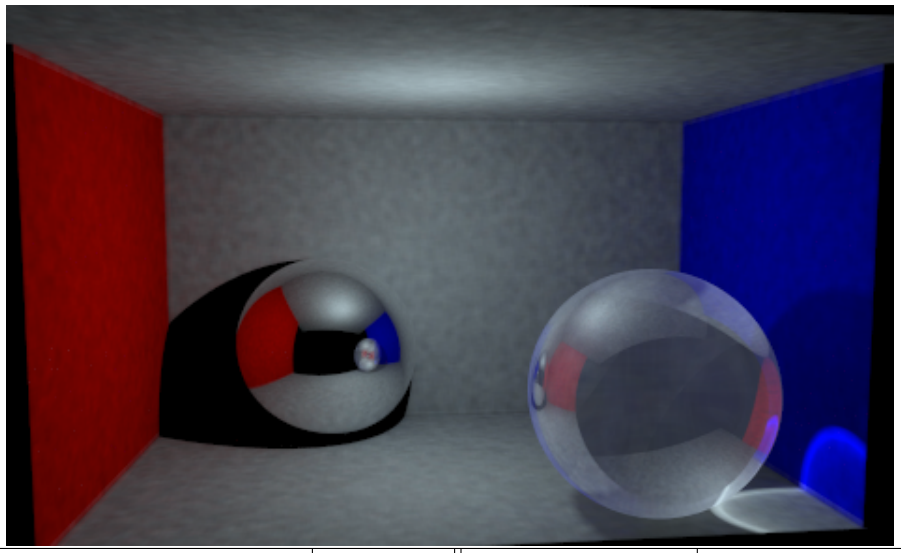
Recursion depth entscheidet, bei welcher Rekursionstiefe die Berechnung der Photonen-Map abgebrochen werden soll.

3.4 Beispielbilder

Die folgenden Beispielbilder wurden mit GroIMP erstellt. Für die Berechnung wurde ein Athlon X2 3800+ mit 2000 MHz und Dual Core eingesetzt. Alle Bilder wurden mit einer Rekursionstiefe von 5 berechnet. Für die Visualisierung wurde 5x5 stochastisches Supersampling eingesetzt. Unter jedem Bild befindet sich eine Tabelle, in der die verwendeten Photonen-Map-Parameter angegeben sind.

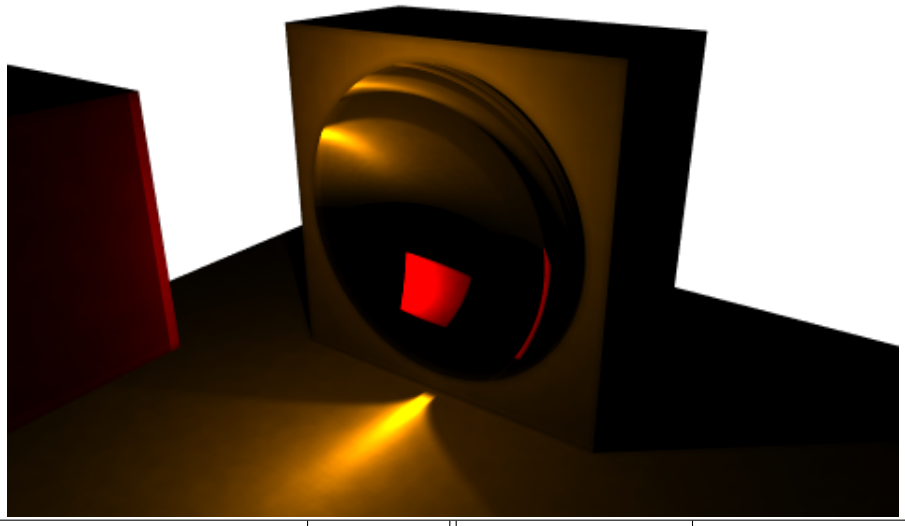
Global Photon Count	0	Photon Area	0.04
Caustic Photon Count	5000000	Rendering-Zeit	2 min 38,09 sek

Abbildung 3.4: Ein Ring, sie zu knechten



Global Photon Count	800000	Photon Area	0.04
Caustic Photon Count	1000000	Rendering-Zeit	9 min 4,89 sek

Abbildung 3.5: Cornell Box mit Chrom- und Glaskugel



Global Photon Count	4000000	Photon Area	0.04
Caustic Photon Count	1000000	Rendering-Zeit	26 min 5,56 sek

Abbildung 3.6: Würfel vor einem Hohlspiegel

3.5 Vor- und Nachteile

Die Photonen-Map eignet sich hervorragend, um Kaustik zu berechnen. Dabei ist sie dem Path-Tracing-Verfahren überlegen, welches dies ungenügend unterstützt. Die Berechnung der Photonen-Map ist nicht vom Blickwinkel abhängig. Dadurch muss diese nur einmal berechnet werden, danach ist die Szene von allen Richtungen aus betrachtbar. Die indirekte diffuse Beleuchtung einer Szene kann ebenfalls sehr gut berechnet werden. In meiner Implementierung ist es nötig, mehrere Millionen Photonen in eine Szene zu „schießen“, um eine gleichmäßige Beleuchtung zu erreichen. Dies könnte man durch einige Optimierungen verbessern, welche ich im nachfolgenden Kapitel näher erläutern möchte.

3.6 Ausblick

Verwendung von Projektionskarten

Bei Szenen mit sehr wenigen Objekten treffen viele ausgesendete Photonen kein Objekt der Szene. Dies führt zu unnötigem Rechenaufwand und einer schlechten Beleuchtung der Szene. Eine mögliche Erweiterung des vorhandenen Algorithmus ist die Einführung von Projektionskarten. Diese beschreiben die Sichtbarkeit der Objekte von der Lichtquelle aus betrachtet. Dabei wird in einem ersten Schritt für jede Lichtquelle ein binäres Rasterbild erstellt, indem die Szenenobjekte auf dieses projiziert und die überdeckten Pixel eingefärbt werden. Dabei können die Objekte auch durch einfachere, umhüllende Körper repräsentiert werden.
Das Aussenden erfolgt dann nur noch entlang der markierten Zellen. Da so allerdings nur ein Teil der Energie einer Lichquelle betrachtet wird, muss auch die transportierte Energie des Photons folgendermaßen angepasst werden [Jen04]:

$$L_{Photon} = \frac{L_{Lichtquelle}}{Anzahl\ der\ Photonen} * \frac{Anzahl\ der\ besetzten\ Zellen}{Gesamtanzahl\ der\ Zellen} \tag{3.2}$$

Eine weitere Optimierung, die der vorhandenen Implementierung noch fehlt, ist die Unterstützung von Multiprozessorsystemen. Dafür müsste man für jeden vorhandenen Prozessor einen eigenen Thread erzeugen und die auszusendenden Photonen gleichmäßig auf diese verteilen.

3.7 Zusammenfassung

Mit dem in diesem Kapittel beschriebenen Photon-Map-Verfahren lassen sich Szenen mit Kaustik und indirekter Beleuchtung sehr gut berechnen. Ein großer Vorteil gegenüber anderen strahlenbasierten Verfahren ist die Vorausberechnug der Lichtverhältnisse in der Szene. Dadurch kann sie schnell aus unterschiedlichen Richtungen dargestellt werden.
Durch dieses Verfahren ergeben sich neue Möglichkeiten in GroIMP; so wäre es jetzt zum Beispiel möglich, Animationen in Echtzeit zu berechnen und einen virtuellen Flug über einen Wald darzustellen.

4 Radiosity

Der Radiosity-Algorithmus wurde 1984 von Goral, Torrance, Greenberg und Battaile entwickelt [GTGB84]. Es handelt sich um ein globales Beleuchtungsverfahren, welches ideal für diffus reflektierende Oberflächen ist. Dies entspricht auch gut der Realität, da diffuse Reflexionen in der Praxis häufig anzutreffen sind, während spiegelnde Flächen eher die Ausnahme bilden.
Die Basis für diesen Algorithmus bildet der Energieerhaltungssatz. Es wird ein Gleichgewicht zwischen der von Lichtquellen zugeführten und der von allen Oberflächen absorbierten Strahlungsenergie angestrebt. Dies wird erreicht, indem für jede Fläche die spezifische Beleuchtungsstärke berechnet wird.
In folgenden Abschnitt 4.1 werden zunächst die theoretischen Grundlagen dieses Verfahrens beschrieben. Danach gehe ich im Abschnitt 4.2 auf meine Implementierung ein, und abschließend werden noch einige mögliche Erweiterungen für dieses Verfahren vorgestellt.

4.1 Grundlagen

Das Radiosity-Verfahren beruht auf der Theorie der Wärmeausbreitung, dabei wird von der Erhaltung der Lichtenergie in einer geschlossenen Szene ausgegangen. Für die Beleuchtung der Szene werden dafür nicht nur die Szenen-Lichtquellen, sondern auch das Licht, welches von Gegenständen der Szene reflektiert wird, betrachtet. Dadurch sind auch Szenen möglich, die nur durch die diffuse Reflexion einer angestrahlten Wand beleuchtet werden.

Ähnlich wie das zuvor beschriebene Photon-Mapping-Verfahren wird auch beim Radiosity-Verfahren zuerst unabhängig vom Blickwinkel des Betrachters die Lichtverteilung der Szene berechnet [RS03].
Die nachfolgende Abbildung 4.1 zeigt den allgemeinen Ablauf bei der Berechnung einer

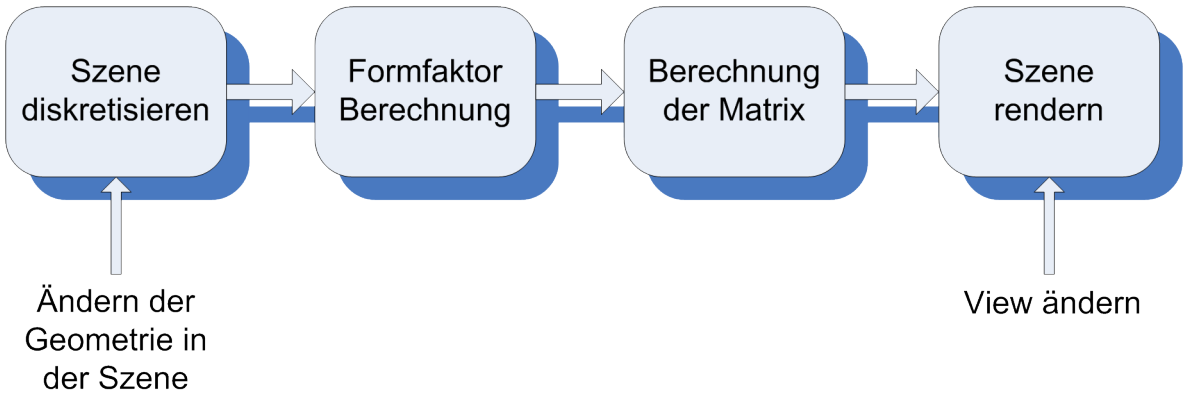


Abbildung 4.1: Ablauf des Radiosity Algorithmus

Szene. Dabei ist auch zu erkennen, an welchen Stellen die Berechnung beginnen muss, wenn die Szene bzw. nur der Blickwinkel verändert wird [Deu03].

Zunächst wird die Szene in endlich viele diskrete Flächenelemente, die sogenannten Patches, zerlegt. Dabei wird von den Annahmen ausgegangen, dass jedes Element diffuse lambertsche Eigenschaften besitzt, undurchsichtig ist und über die gesamte Fläche gleichmässig Licht abstrahlt und reflektiert [RS03].
Im nächsten Schritt werden die sogenannten Formfaktoren berechnet, welche für je zwei Flächen der Szene beschreiben, wie hoch der Anteil des Lichtaustausches zwischen ihnen ist. Dabei werden die geometrischen Beziehungen, sowie die Entfernungen und die relative Orientierungen zwischen ihnen berücksichtigt.
So würde beispielsweise bei zwei nah beieinander und parallel zueinander liegenden Flächen viel Energie ausgetauscht. Dabei muss allerdings auch die Verdeckung durch andere Flächen berücksichtigt werden. Der Abschnitt 4.1.1 beschreibt, wie diese Größe ermittelt wird.
Die Formfaktoren müssen nicht nur für zwei Flächen in der Szene, sondern für alle Paare von Flächen berechnet werden, wodurch sich eine Matrix ergibt. Dies wird im Abschnitt 4.1.3 beschrieben.
Zuletzt müssen die so berechneten Lichtverteilungen noch visualisiert werden.

4.1.1 Die Formfaktoren

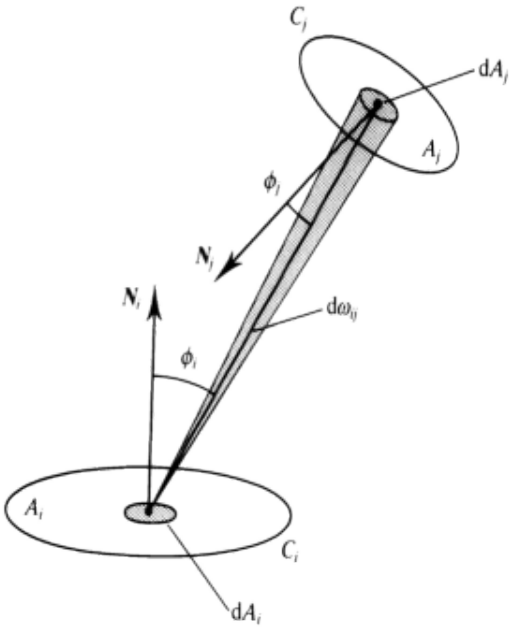


Abbildung 4.2: Formfaktor

Der komplizierteste und rechenintensivste Teil beim Radiosity-Verfahren ist die Berechnung der Formfaktoren. Sie beschreiben, wieviel Strahlungsenergie zwischen zwei Patches ausgetauscht wird. Dabei sind die Formfaktoren nur von der Geometrie der Szene abhängig, die reflektierenden und emittierenden Eigenschaften einer Oberfläche haben darauf keinen Einfluss. Bei der Berechnung wird die Orientierung, die Entfernung sowie die Sichtbarkeit zwischen diesen Elementen berücksichtigt.
Die Anzahl der Formfaktoren wächst quadratisch mit der Anzahl der Flächen in einer Szene.
Für zwei Patches A_i und A_j ist der Formfaktor definiert durch die abgestrahlte Energie, die ein unendlich kleines Oberflächenstück von A_i verlässt und auf ein unendlich kleines Stück von A_j auftritt, geteilt durch die abgestrahlte Energie, welche das Oberflächenstück in alle Richtungen verlässt (siehe Abbildung 4.2) [RS03].

Genauer wird der Formfaktor durch das folgende schwer zu lösende Doppelintegral beschrieben [RS03]:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\phi_j \ \cos\phi_i}{\pi r^2} dA_j \ dA_i \tag{4.1}$$

Selbst bei einfachen geometrischen Verhältnissen ist die Berechnung dieses Integrales sehr aufwendig. Außerdem setzt es voraus, dass sich beide Flächen komplett sehen. Wegen dieser Schwierigkeiten greift man auf eine Näherung zurück, welche das Integral auf geometrische Weise löst, dies wird im folgenden Abschnitt 4.1.2 näher erläutert.

4.1.2 Hemicube-Verfahren

Da die exakte Berechnung der Formfaktoren zu aufwendig ist, wurde schon 1928 von Wilhelm Nusselt ein Verfahren entwickelt, um diese durch eine geometrische Näherung zu bestimmen. Dabei wird um das betrachtete Flächenelement eine Einheitshalbkugel gelegt. Um das Integral zu lösen, wird nun eine Projektion des Flächenstückes dA_j auf die Einheitshalbkugel durchgeführt. Anschließend wird der entstandene Halbkugelausschnitt orthogonal auf die Grundfläche der Halbkugel projiziert (siehe Abbildung 4.3) [Kur04].

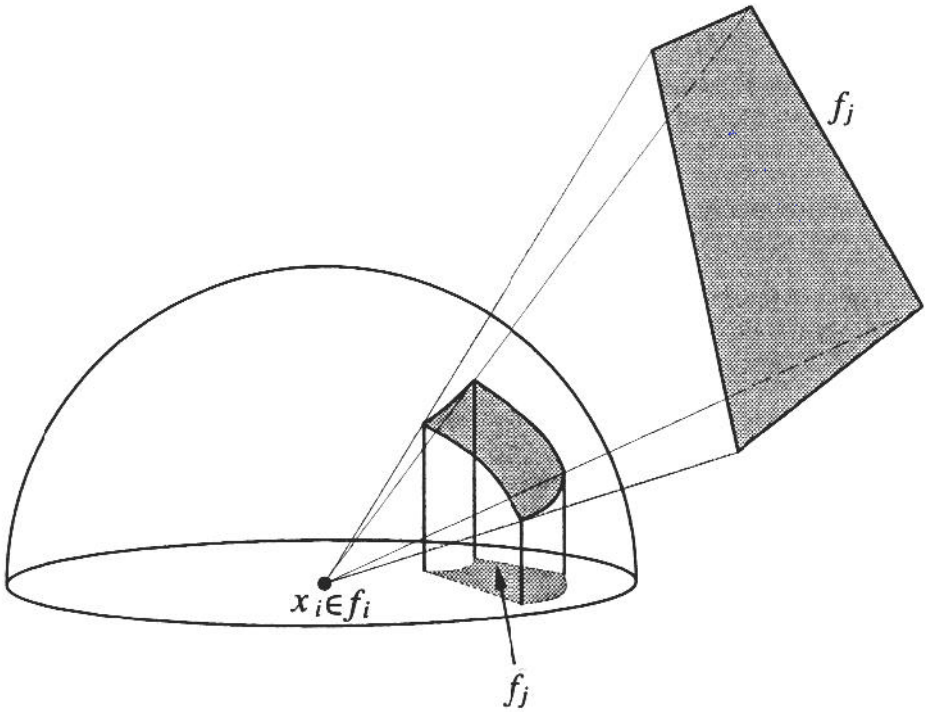


Abbildung 4.3: Projektion auf Einheitskugel

Der Anteil dieser Projektion an der Gesamtgrundfläche der Halbkugel entspricht dem Formfaktor. Für die allgemeine Berechnung von Formfaktoren wird nun die Oberfläche der Halbkugel in kleine, aber gleichgroße Segmente aufgeteilt. Für jedes dieser Segmente wird der Formfaktor nach obigem Projektionsverfahren berechnet. Diese werden auch Delta-Formfaktoren genannt. Die Effektivität dieser Methode besteht darin, dass man diese Delta-Formfaktoren im Voraus berechnen kann und in einer Look-Up Tabelle ablegt. Anschließend werden die Flächen der Szene auf diese Halbkugel projiziert, wobei festgestellt wird, welche Segmente auf der Halbkugel von den Projektionen der

Flächen überdeckt werden. Der gesamte Formfaktor der Fläche ist die Summe der Delta-Formfaktoren der von der Projektion bedeckten Segmente.
Dieses Verfahren ist allerdings nicht praxistauglich, da die Einteilung der Halbkugeln in gleichgroße Segmente und die Projektion der Flächen auf die Halbkugel für einen Computer schwer zu berechnen sind.

Das Verfahren wurde 1985 von Cohen und Greenberg zu einem wesentlich praxistauglicheren Verfahren weiterentwickelt, dem Hemi-Cube-Verfahren [CG85]. Da flache Projektionsebenen für den Computer einfacher zu berechnen sind, wird die Halbkugel durch einen Halbwürfel, dem sogenannten Hemi-Cube, angenähert. Wie in Abbildung 4.4 dargestellt, wird jede der fünf Seiten mit einem rechteckigen Raster überzogen, den Hemi-Cube-Pixeln. Die Projektion der Szene auf eine Seite des Hemi-Cube stellt prinzipiell das gleiche Problem wie die Rasterisierung einer 3D-Szene dar. Dafür existieren bereits verschiedene effiziente Algorithmen.

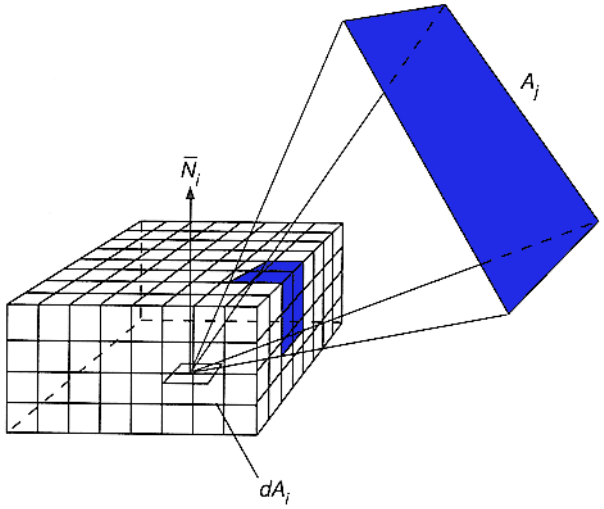


Abbildung 4.4: Hemi-Cube

Jedem der Hemi-Cube-Pixel wird ein Delta-Formfaktor zugeordnet, der den Beitrag zum gesamten Formfaktor beschreibt. Diese Delta-Formfaktoren können für einen Einheitswürfel in einem Vorverarbeitungsschritt berechnet werden. Den Formfaktor einer Fläche erhält man, indem man die Delta-Formfaktoren derjenigen Hemi-Cube-Pixel, die durch die Projektion der Fläche bedeckt werden, aufsummiert. Für die Fläche A_i in Abbildung 4.4 bedeutet das, der Formfaktor F_{ji} entspricht der Summe der Delta-Formfaktoren der blau markierten Hemi-Cube-Pixel.
Wird bei einer Projektion ein Pixel von einer näher liegenden Projektion überdeckt, so wird sein Delta-Formfaktor bei der Berechnung nicht berücksichtigt. Dies kann durch die Verwendung eines z-Buffer-Algorithmus berechnet werden [Kur04].

4.1.3 Die Formfaktor-Matrix

Der Ansatz des Radiosity-Verfahrens bezieht sich wie oben erwähnt auf ein Berechnungsverfahren für den Energieaustausch zwischen Oberflächen. Dabei wird angenommen, dass die Emission für jede Fläche homogen und der Reflexionsgrad konstant ist. Alle Flächen in der Szene werden durchnummeriert. Dadurch lässt sich mit der folgenden Formel der Energieaustausch zwischen zwei Flächen i und j der Szene beschreiben [CWH93]:

$$B_i A_i = E_i A_i + \rho_i \sum_{j=1}^{n} B_j A_j F_{ji} \tag{4.2}$$

Dabei bezeichnet:

$B_i A_i$ die gesamte Energie, die ein Objekt verlässt,

$E_i A_j$ die erzeugte Energie des Flächenelements i,

ρ_i das Reflexionsvermögen des Elements i und

F_{ji} den im Abschnitt 4.1.1 beschriebenen dimensionslosen Formfaktor. Er beschreibt den Anteil des Lichtes, das von A_i ausgehend A_j trifft. Dabei werden die Orientierung beider Flächenelemente sowie eventuell vorhandene blockierende Elemente berücksichtigt.

Diese Beziehung besteht zwischen je zwei Patches der Szenen, daher kann man den gesamten Energieaustausch einer Szene, bestehend aus n Flächen, durch das folgende Gleichungssystem beschreiben [CWH93]:

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \tag{4.3}$$

Für ein konvexes Objekt wird F_{ii} immer gleich null sein, da es keinen Beitrag zu seiner eigenen reflektierten Energie beitragen wird.
Da dieses Gleichungssystem nicht vom Blickwinkel des Betrachters abhängt, kann es komplett während einer Vorverarbeitungsphase des Radiosity-Verfahrens gelöst werden. Wie dies erfolgt, wird im nachfolgenden Abschnitt 4.1.4 erläutert.

4.1.4 Lösungsverfahren

Die einfachste Möglichkeit, das in Abschnitt 4.1.3 beschriebene Gleichungssystem zu lösen, wäre eine direkte Lösung mittels gaußschem Eliminationsverfahren. Dies ist aber nicht praktikabel, da es einen kubischen Rechenaufwand bedeutet.
Es existieren zwei Gruppen von Lösungsverfahren, diese werden als *Gathering*- und *Shooting*-Verfahren bezeichnet:

Gathering

Beim Gathering-Verfahren handelt es sich um Algorithmen, bei denen jeder Patch das Licht von allen anderen Flächen einsammelt und dieses zum, eigenen Leuchten addiert [Deu03].
Das so entstehende Gleichungssystem kann durch das «Gauß-Seidel-Verfahren» gelöst werden. Der folgende Pseudo-Code veranschaulicht dieses Vorgehen [Deu03]:

```
1 for (jedes i) {
2     B_i = Startwert ;
3 }
4 while (keine Konvergenz) {
5     for (jedes i) {
6         B_i = E_i + sum_{j=1,j≠i}^{n} p_i B_j K_ij
7     }
8 }
9 Stelle Ergebnis dar, nutze B_j als Farbe für Fläche j
```

Listing 4.1: Gauß-Seidel-Verfahren Pseudo-Code

Bei diesem Algorithmus wird für jede Fläche i die Radiosity B_i berechnet. Dafür wird das Gleichungssystem zeilenweise gelöst. K_{ij} bezeichnet dabei die in Gleichung 4.3 angegebene Matrix.
Der Vorteil dieses Algorithmus besteht darin, dass er sich intuitiv implementieren lässt.

Shooting

Bei dieser Gruppe von Verfahren wird die noch unverbrauchte Energie einer Fläche auf alle anderen Flächen, welche von dieser aus sichtbar sind, verteilt [Deu03].
Ein Vertreter dieser Gruppe ist «Progressive Refinement». Der folgende Pseudo-Code veranschaulicht dieses Verfahrens [Deu03]:

```
1  for (jedes i) {
2      B_i = E_i ;
3      ΔB_i = E_i ;
4  }
5  while (keine Konvergenz) {
6      i = Element mit maximalem ΔB_i * A_i ;
7      for (jedes j) {
8          Δrad = ΔB_i * p_j F_ji ;
9          ΔB_j = ΔB_j + Δrad ;
10         B_j = B_j + Δrad ;
11     }
12     ΔB_i = 0 ;
13     Stelle Zwischenergebnis dar, nutze B_j als Farbe für Fläche j
14 }
```

Listing 4.2: Progressive Refinement Pseudo-Code

Während des Durchlaufs der ersten Schleife wird jede Fläche i initialisiert. Dabei wird die Radiosity B_i und die „unverteilte Radiosity“ ΔB_i mit der Eigenemission E_i belegt. Danach wird die Fläche i mit der größten unverteilten Radiosity gesucht und diese auf alle anderen verteilt. Danach wird die unverteilte Radiosity der Fläche i auf 0 gesetzt, da sie ihre gesamte Energie abgegeben hat. Dies wird so lange wiederholt, bis das Ergebnis gegen eine stabile Lösung konvergiert.
Der Vorteil dieses Verfahrens ist, dass pro Iterationsschritt nur eine Zeile der Formfaktormatrix bekannt sein muss. Dadurch wird auch nur der Speicherplatz für diese Zeile benötigt. Dieses Verfahren liefert bereits nach wenigen Iterationsschritten gute Ergebnisse, dadurch liegt der Gesamt-Rechenaufwand unter dem von von anderen Lösungsansätzen für das Gleichungssystem [Kur04].

4.1.5 Adaptive Unterteilung

Bei der bisher betrachteten Lösung hängt die Qualität des berechneten Bildes sehr stark von der Größe der einzelnen Patches ab. Deshalb werden Flächen unterteilt, welche einen

zu hohen Gradienten aufweisen. Dies erhöht die Genauigkeit der Lösung. Der folgende Pseudo-Code beschreibt dieses Vorgehen [Deu03]:

```
Erzeuge initiales Netz;
Berechne Formfaktoren; Löse LGS
do {
    Berechne Genauigkeit durch Vergleich benachbarter Elementstrahlungen
    Unterteile Elemente mit Fehler über gegebener Schranke
    for (jedes neue Element) {
        Berechne FF vom neuen Element zu allen anderen
        Berechne Strahlung des neuen Elements
    }
} until alle Elemente unterhalb Fehlerschranke or Abbruchgrenze erreicht
```

Listing 4.3: Pseudo-Code für das Unterteilen von Flächen

Dabei werden die Flächen dynamisch während der Berechnung immer weiter unterteilt.

4.2 Implementierung

In diesem Kapitel möchte ich auf einige Implementierungsdetails eingehen. Dies soll einen Ausgangspunkt für Personen darstellen, die an meine Arbeit anknüpfen möchten.

Bei der Implementierung wurde der in Abschnitt 4.1.5 vorgestellte Pseudo-Algorithmus umgesetzt. Als Basis für meine Implementierung habe ich eine in C++ geschriebene Umsetzung des Radiosity-Verfahrens verwendet [Roh07].

4.2.1 Zerlegung der Objekte

Zuerst möchte ich erläutern, wie die vorhandenen Objekte der Szene in Flächen zerlegt werden. Ich habe mich dabei für Dreiecksflächen entschieden. Eine Unterteilung in Rechtecke wäre ebenfalls möglich, diese führt aber zu Problemem bei nichtuniformer Unterteilung [Deu03].
Die momentane Implementierung unterstützt nur Standardobjekte sowie einfache Punkt- und Flächenlichtquellen. Eine Unterstützung von gekrümmten Flächen wie Nurbs und Splines ist noch nicht enthalten. Wenn eine Szene ein Punktlicht enthält, wird dieses durch einen kleinen Würfel aus Flächenlichtquellen ersetzt, da der implementierte Algorithmus nur Flächen unterstützt. Näheres dazu und mögliche Erweiterungen sind im

Abschnitt 4.5 beschrieben.
Für jedes unterstützte Objekt existiert eine eigene Klasse, welche einen Algorithmus enthält, um dieses in eine Menge von Dreiecken zu zerteilen. Diese Klassen befinden sich im Paket `de.grogra.ray2.radiosity.triangulation`. Ein Dreieck wird dabei durch die Klasse `SubPatch` beschrieben.
Für die spätere dynamische Unterteilung der Flächen (siehe Abschnitt 4.2.5) ist es nötig, dass die Flächen in Gruppen verwaltet werden. Dabei enthält jede Gruppe vier benachbarte Flächen. Dafür ist die Klasse `PatchGroup` zuständig. Die Klasse `GroupListBuilder` erzeugt und verwaltet eine globale Liste aller dieser Gruppen. Alle diese Klassen befinden sich im Paket `de.grogra.ray2.radiosity`.

4.2.2 Berechnung der Delta-Formfaktoren

Die Klasse `Hemicube` ist für die Berechnung der Delta-Formfaktoren eines Hemi-Cubes zuständig (siehe Abschnitt 4.1.2).
Der im Abschnitt 4.2.8 beschriebene Parameter «Hemicube size in pixel» bestimmt, in wieviele Hemi-Cube-Pixel dieser zerlegt werden soll. Die Deckfläche eines Hemi-Cubes besteht dabei aus der eingestellten Pixelzahl zum Quadrat. Jede der vier Seitenflächen hat halb so viele Pixel wie die Deckfläche.
Die Klasse berechnet für jeden dieser Pixel den Delta-Formfaktor und speichert ihn in einer Matrix. Diese Berechnung ist nur einmal nötig, solange der erwähnte Parameter nicht verändert wird.

4.2.3 Projektion der Flächen

Für die vorhandene Implementierung wurde ein modifiziertes Gauß-Seidel-Verfahren eingesetzt (4.1.4). Dabei sammelt ein ausgewähltes Patch die Energie aller anderen Patches in der Szene ein.
Dazu wird für jedes Dreieck ein Hemi-Cube erzeugt, und alle anderen Dreiecke werden auf diesen projiziert. Die Klasse `ViewCube` beschreibt einen Hemi-Cube für ein ausgewähltes Patch und speichert für jeden Hemi-Cube-Pixel, welches andere Patch auf diesen projiziert wurde.
Um verdeckte Flächen zu erkennen, wird für jeden dieser Pixel ein Z-Buffer verwaltet, wofür die Klasse `ZBuffer` zuständig ist.

Nachdem alle Flächen auf den Hemi-Cube projiziert wurden, werden die Delta-Formfaktoren aufsummiert, um für alle Paare von Patches den Formfaktor zu bestimmen.

4.2.4 Energieaustausch

Nachdem alle Formfaktoren berechnet sind, findet der Energieaustausch zwischen den Patches statt. Dafür ist die Funktion `computeAllPatchColors` der Klasse `RadiosityAlgorithm` zuständig. Dabei wird in einer Schleife so oft iteriert, bis die gesamte zur Verfügung stehende Energie auf alle Patches verteilt ist.

4.2.5 Unterteilung der Flächen

Nach dem Energieaustausch werden die Energiewerte eines Dreiecks mit den Dreiecken innerhalb der gleichen `PatchGroup` verglichen. Sollte die Differenz für alle Flächen kleiner sein als «Subdivision threshold» (siehe Abschnitt 4.2.8), so ist die Radiosity-Berechnung abgeschlossen, und das Bild muss nur noch visualisiert werden. Sollte diese Schwelle allerdings bei einer `PatchGroup` überschritten werden, so werden alle Dreiecke dieser Gruppe weiter unterteilt. Danach wird der Algorithmus mit der so erzeugten Dreiecksliste von neuem gestartet.
Um diese Iteration bei einer bestimmten Anzahl abzubrechen, wurde der Parameter «Subdivision depth» eingeführt. Wenn der eingestellte Wert erreicht ist, so erfolgt keine weitere Unterteilung der Patches und das momentan berechnete Ergebnis wird visualisiert.

4.2.6 Visualisierung

Momentan wird ein sehr einfacher Algorithmus eingesetzt, um die berechnete Szene zu visualisieren. Dafür werden für jeden Pixel Strahlen durch die Kameraebene gesendet. Trifft dieser Strahl auf eine Dreiecksfläche, wird der Pixel mit der Farbe des Dreiecks eingefärbt.
Die Implementierung für dieses Verfahren befindet sich in der Klasse `de.grogra.ray2.tracing.Radiosity`.
Um dieses Verfahren zu beschleunigen, wird ein Octree benutzt. Dieses und weitere Optimierungsverfahren werden im folgenden Abschnitt 4.2.7 beschrieben.

4.2.7 Optimierungen

Ich habe verschiedene Optimierungsverfahren implementiert, um die Berechnung eines Bildes zu beschleunigen. In diesem Abschnitt möchte ich diese vorstellen.

Beschleunigung durch Octree

Alle während der Zerlegung (4.2.1) erstellten Flächen werden in einen Octree eingeordnet.
Ein Octree ist eine hierarchische Einteilung einer Szene durch künstlich eingeführte Bereiche. Jeder Ast der baumartigen Struktur eines Octrees ist hierbei ein nach den Weltachsen ausgerichteter Würfel. Dieser kann wiederum in acht gleich große Würfel unterteilt werden, welche dann Kinder darstellen. Die Wurzel des Baumes bildet ein Würfel, der die gesamte Szene umfasst [Kur04].
Während der Visualisierungsphase werden Strahlen vom Auge in die Szene gesendet, dabei wird zunächst überprüft, welcher Würfel des Octrees getroffen wird, und danach, welches der Patches innerhalb des Würfels.
Dadurch ist ein schnelles Visualisieren der Szene möglich.

Parallelisierung

Damit der Radiosity-Algorithmus von Multiprozessor-Systemen profitiert, werden die Berechnung der Formfaktoren und die Projektionen der Flächen auf den Hemi-Cube gleichmäßig auf je einen Thread pro vorhandenem Prozessor aufgeteilt.

4.2.8 Parameter

Es gibt vier wichtige Parameter, um die Qualität eines mittels Radiosity gerenderten Bildes zu beeinflussen. Dies sind «Hemicube size in pixel», «Hemicube size in world coordinates», «Subdivision threshold» und «Subdivision depth». Diese Parameter können in den Programmoptionen von GroIMP geändert werden (siehe Kapitel 2.2).

Hemicube size in pixel bestimmt die Anzahl der Hemi-Cube-Pixel (siehe Abschnitt 4.1.2) aus denen ein Hemi-Cube besteht. Je höher dieser Wert, um so exakter wird das Licht berechnet, das auf ein Dreieck fällt.

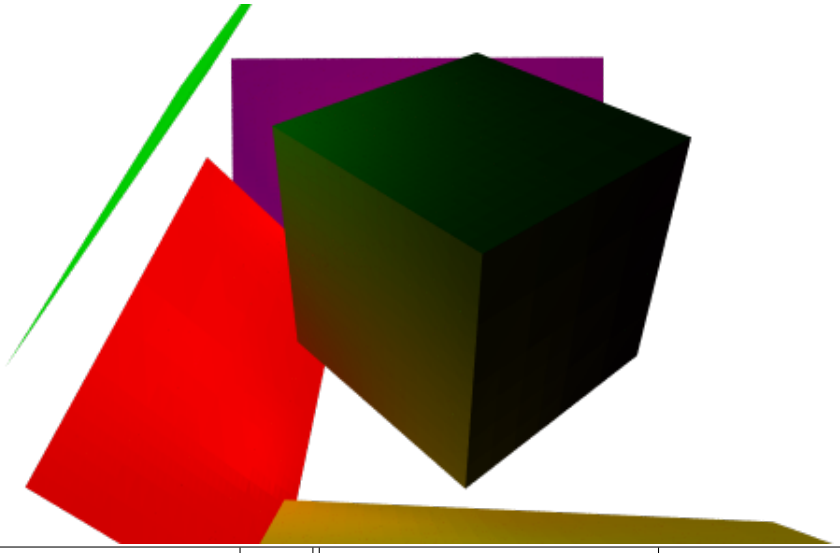
Hemicube size in world coordinates legt fest, wie groß ein Hemi-Cube im Koordinatensystem der Szene ist.

Subdivision threshold Nach jedem Berechnungsschritt werden die Farbwerte benachbarter Dreiecke verglichen. Wenn der Unterschied größer ist als dieser Schwellenwert, so werden die Dreiecke für den nachfolgenden Berechnungsschritt unterteilt. (siehe Abschnitt 4.2.5)
Ein geringerer Wert verbessert die Farbabstufungen zwischen benachbarten Dreiecken, aber erhöht die Rechenzeit und den Speicherverbrauch.

Subdivision depth legt fest, wie viele Berechnungsschritte durchgeführt werden sollen. (siehe Abschnitt 4.2.5)
Je höher der Wert, um so exakter wird das Bild berechnet. Jedoch erhöht sich auch die Berechnungszeit.

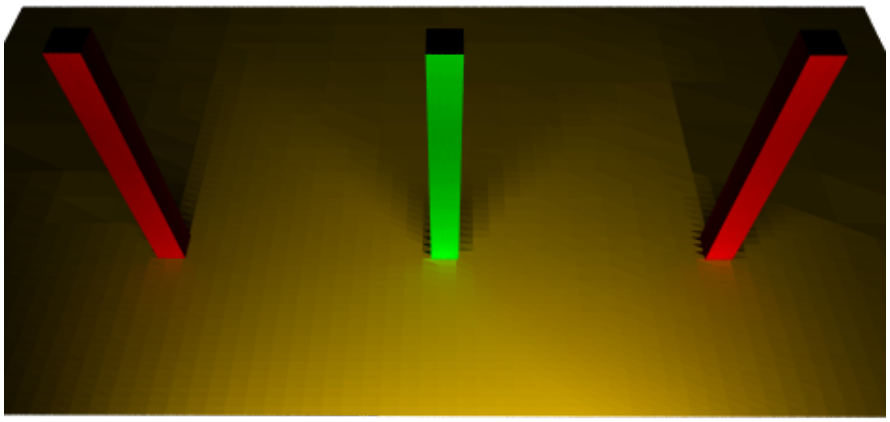
4.3 Beispielbilder

Die folgenden Beispielbilder wurden mit GroIMP erstellt. Für die Berechnung wurde ein Athlon X2 3800+ mit 2000 MHz und Dual Core eingesetzt. Alle Bilder wurden mit einer Rekursionstiefe von 5 berechnet. Für die Visualisierung wurde 5x5 stochastisches Supersampling eingesetzt. Unter jedem Bild befindet sich eine Tabelle, in der die verwendeten Radiosity-Parameter angegeben sind.



Subdivision threshold	2.5	Hemicube size in pixel	100
Subdivision depth	5	Rendering-Zeit	5 min 35,22 sek

Abbildung 4.5: Radiosity-Szene mit 4 farbigen Lichtquellen



Subdivision threshold	0.4	Hemicube size in pixel	100
Subdivision depth	6	Rendering-Zeit	42 min 13 sek

Abbildung 4.6: Radiosity-Szene mit 3 Säulen

4.4 Vor- und Nachteile

Das Radiosity-Verfahren ist sehr gut für geschlossene Räume geeignet. Es werden nur Oberflächen mit diffusen Eigenschaften unterstützt. Diese sind in der Natur auch wesentlich häufiger anzutreffen als spekulare. Radiosity ist imstande, diffuse Reflexionen zu berechnen, was von den meisten strahlenbasierten Verfahren nicht unterstützt wird. Auch globale Beleuchtung, bei der nicht nur die direkte Lichteinstrahlung berücksichtigt wird, sondern auch das Licht, das durch Reflexion an mehreren Objekten auf eine Fläche fällt, ist gut mittels Radiosity berechenbar.

Allerdings besitzt das Verfahren auch einige Probleme. So ist der Speicherverbrauch sehr hoch, und es ist eine sehr lange Rechenzeit nötig. Auch unterstützt das Standard-Radiosity-Verfahren keine Spiegelungen und Transparenz.
Meine Implementierung ist noch nicht optimal, durch einige gezielte Verbesserungen könnte man den Algorithmus anpassen, um so weitere Objekttypen, Spiegelungen und Transparenz zu unterstützen.
Im nachfolgenden Kapitel möchte ich einige dieser möglichen Verbesserungen vorstellen.

4.5 Ausblick

Für den momentan implementierten Radiosity-Algorithmus gibt es noch Verbesserungs- und Erweiterungspotential. Ich möchte in diesem Abschnitt einige Änderungen vorschlagen, die das Ergebnis eines mittels Radiosity gerenderten Bildes positiv beeinflussen würden.

Unterstützung weiterer Lichtquellen

Wie in Abschnitt 4.2.1 erläutert, unterstützt die momentane Implementierung lediglich Flächen- und einfache Punktlichtquellen. Dabei werden Punktlichtquellen durch einen Würfel aus Flächenlichtquellen simuliert. Eine Unterstützung für «SpotLight», «Directional Light» und «Sky» fehlt. Diese könnte man ebenfalls mittels Flächenlichtquellen modellieren oder aber den Radiosity-Algorithmus anpassen. Dazu ist eine Modifikation des Projektionsteils (4.2.3) nötig. Man müsste die Lichtquellen auf alle anderen Flächen projizieren und den Energieaustausch bestimmen.

Unterstützung weiterer Objekte

Ähnlich wie bei den Lichtquellen werden auch nur Standardobjekte, wie z.B. Würfel, Kugeln, Zylinder unterstützt. Szenen mit Splines oder Nurbs sind momentan nicht möglich. Dabei handelt es sich um mathematisch definierte Kurven oder Flächen. Um eine Unterstützung für diese Objekte zu integrieren, müsste der Algorithmus um Funktionen erweitert werden, welche diese effizient in Dreiecksnetze zerlegen. Dies ist z.B. mittels des Sweep-Line-Algorithmus von Garey, Johnson, Preparata und Tarjan möglich (siehe Kapitel 6 [Kur07]).

Kombination mit anderen Raytracing-Verfahren

Um ein Bild zu visualisieren, stellt der momentan implementierte Algorithmus nur die berechneten Farbwerte aller Flächen dar. Dadurch sind keine Kaustik-, Transparenz- oder Spiegelungseffekte möglich. Um diesen Mangel zu beseitigen, wurden von John Wallace und Sillion zwei Methoden entwickelt, die Radiosity und erweiterte Raytracing-Algorithmen kombinieren. Dafür wird in einem Vorverarbeitungsschritt eine erweiterte Radiosity-Lösung berechnet und in einem zweiten Schritt ein erweitertes Raytracing-Verfahren für die Visualisierung eingesetzt. Nähere Informationen dazu liefern die Arbeiten [RS03] und [Kur04].

Interpolation

Auf den gerenderten Bildern sind recht harte Kanten zwischen zwei benachbarten Dreiecken zu erkennen. Diese entstehen, da es keine Farbverläufe innerhalb eines Dreiecks gibt. Jedes Dreieck besitzt nur einen durch den Radiosity-Algorithmus berechneten Farbwert. Die Qualität des Bildes könnte durch Interpolation verbessert werden. Dazu müssten die Farben aller drei Eckpunkte eines Dreiecks bestimmt und ein Farbverlauf über die gesamte Dreiecksfläche berechnet werden. Die Farbe eines Eckpunktes erhält man, indem der arithmetische Mittelwert aller angrenzenden Dreiecke bestimmt wird. Da nicht alle Dreiecke einer Szene gleich groß sind, kann sich die Anzahl der Dreiecke, die einen gemeinsamen Punkt haben, unterscheiden.
Die momentan verwendete Datenstruktur ist leider nicht in der Lage, alle Dreiecke zu liefern, die einen gemeinsamen Eckpunkt haben. Sie müsste angepasst werden, um eine Interpolation zu ermöglichen.

4.6 Zusammenfassung

Mit dem in diesem Kapitel beschriebenen Radiosity-Verfahren lassen sich Szenen mit indirekter Beleuchtung und Flächenlichtquellen sehr gut berechnen. Besonders für Innenräume ist dieses Verfahren gut geeignet und erzeugt eine wesentlich realistischere Darstellung als strahlenbasierte Verfahren. Durch das beobachterunabhängige Berechnen kann eine Szene schnell aus unterschiedlichen Richtungen dargestellt werden. Das Rendern mittels Radiosity erfordert allerdings wesentlich mehr Speicheraufwand und Rechenzeit als strahlenbasierte Verfahren. Auch sind mittels Raytracing-Verfahren Spiegelungen und Transparenz wesentlich einfacher zu berechnen.
Um dieses Verfahren für den alltäglichen Einsatz in GroIMP fit zu machen, wäre eine Erweiterung um die noch fehlenden Objekte und Lichtquellen von Vorteil. Desweiteren würde eine Reduzierung des recht hohen Speicherbedarfs zu einer besseren Nutzbarkeit führen.

5 Vergleich der Verfahren

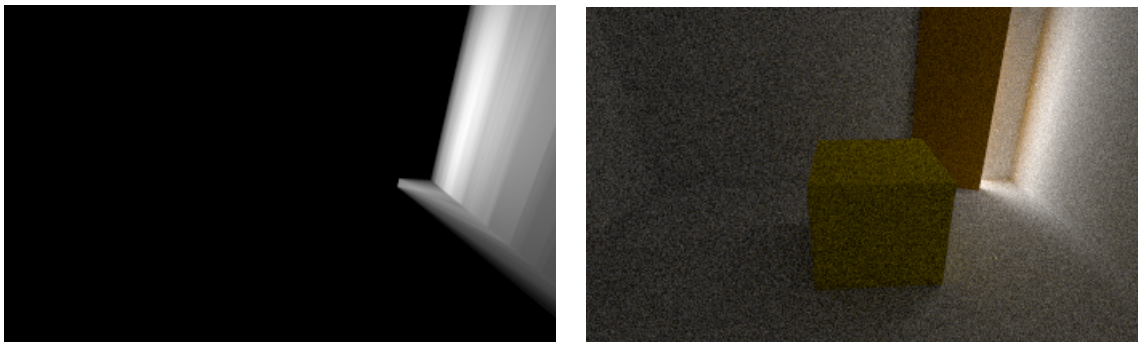
In diesem abschließenden Kapitel möchte ich das Photon-Mapping- und Radiosity-Verfahren gegenüberstellen und mit dem Standard-Raytracer und dem Path Tracer vergleichen. Dabei versuche ich die Vorteile des jeweiligen Verfahrens anhand von Beispielbildern zu belegen. Die Parameter, welche für die Berechnung der einzelnen Bilder eingestellt wurden, sind in einer Tabelle in Abschnitt 5.6 aufgelistet.
Einige der hier gezeigten Bilder werden in der Arbeit von Hagen Steidelmüller ebenfalls zu sehen sein. Dadurch soll es möglich sein, die von ihm implementierten Rendering-Verfahren mit denen aus dieser Arbeit zu vergleichen.

5.1 Rechenzeit und Speicherverbrauch

Das Vergleichen der Rechenzeit und des Speicherverbrauchs der verschiedenen Algorithmen ist nur schwer möglich. Diese ist sehr stark von den eingestellten Parametern abhängig.
Generell lässt sich sagen, dass die von mir implementierten Algorithmen einen recht hohen Speicherverbrauch haben. Dies ist durch die Art der Verfahren bedingt, da in der ersten Phase die Lichtverteilung der gesamten Szene berechnet und diese in einer Datenstruktur gespeichert wird. Weiterhin ist der Speicherverbrauch des Radiosity-Verfahrens größer als der des Photon-Mapping-Verfahrens.
Beide von mir implementierten Algorithmen benötigen während der Initialisierungsphase die meiste Rechenzeit. Nachdem diese abgeschlossen ist, kann die Szene mit beiden Verfahren recht zügig aus verschiedenen Blickwinkeln berechnet werden. Während diese Initialisierung beim Photon-Mapping-Verfahren in wenigen Minuten abgeschlossen ist, benötigt Radiosity für komplexe Szenen Stunden. Diese lange Berechnungsdauer zusammen mit dem hohen Speicherverbrauch sorgt dafür, dass das Radiosity-Verfahren für komplexe Szenen ohne weitere Optimierungen nicht geeignet ist.

5.2 Indirekte Beleuchtung

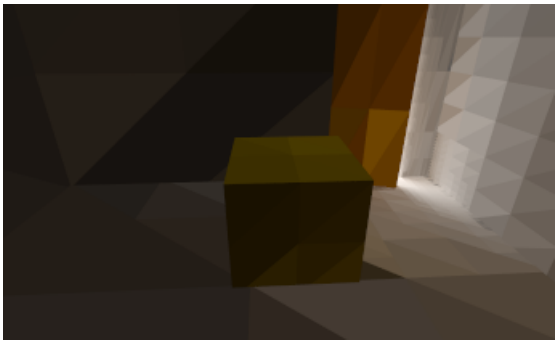
Beide in dieser Ausarbeitung vorgestellte Algorithmen sind gut für Szenen mit indirekter Beleuchtung geeignet. Um dies zu zeigen, habe ich eine Szene erstellt, die nur durch eine Lichtquelle beleuchtet wird, welche sich hinter einer halb geöffneten Tür befindet.



a) Standard-Raytracer

b) Path Tracer

c) Photonen Map



d) Radiosity

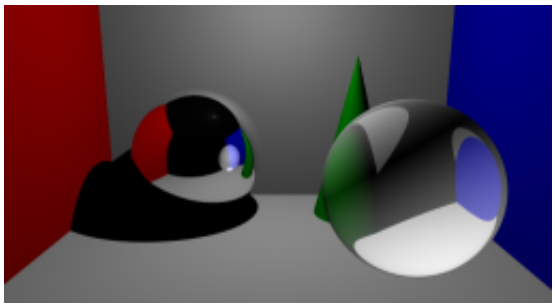
Abbildung 5.1: Vergleich bei indirekter Beleuchtung

Auf Bild 5.1 ist zu erkennen, dass der Standard-Raytracer für Szenen mit indirekter Beleuchtung nicht geeignet ist. Er erzeugt ein hauptsächlich schwarzes Bild. Der Path Tracer erzeugt ein sehr verrauschtes Bild, da er für indirekte Beleuchtung nicht optimal geeignet ist.
Bei den Photon-Mapping- und Radiosity-Bildern ist gut zu erkennen, dass die gesamte Szene ausgeleuchtet ist. Das erzeugte Bild des Photon-Mapping-Verfahrens ist kaum verrauscht, im Gegensatz zum Path Tracer-Bild.

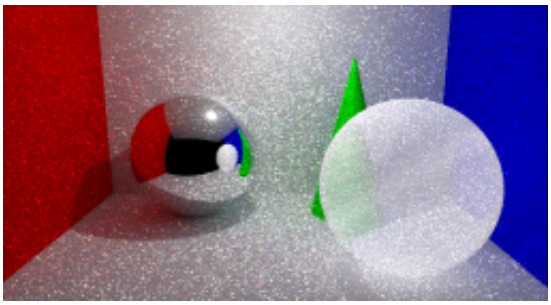
Das Radiosity-Bild ist sehr grob in Dreiecke unterteilt, was für sehr harte Kanten und eine ungenaue Berechnung sorgt. Um ein besseres Bild zu erhalten, müsste man den Parameter «Subdivision depth» weiter erhöhen, was aber zu einem sehr hohen Rechenaufwand und Speicherverbrauch führt.

5.3 Spekular reflektierende Objekte

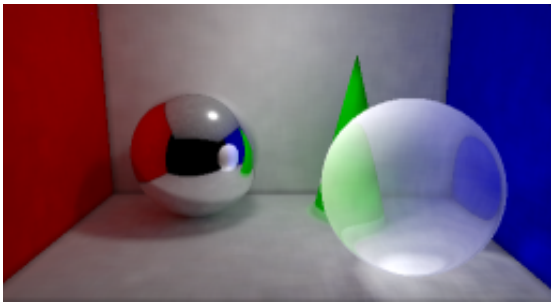
Spekular reflektierende Objekte werden vom Radiosity-Algorithmus nicht unterstützt. Deshalb werde ich in diesem Abschnitt nur den Standard Raytracer, den Path Tracer und das Photon-Mapping-Verfahren vergleichen.
Um zu verdeutlichen, welche der Verfahren gut für spekular reflektierende Objekte geeignet sind, habe ich eine Szene bestehend aus einer spiegelnden und einer transparenten Kugel ausgewählt:



a) Standard-Raytracer



b) Path Tracer



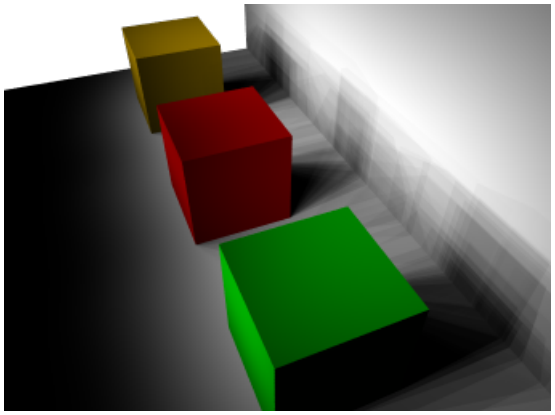
c) Photonen Map

Abbildung 5.2: Vergleich spekular reflektierender Objekte

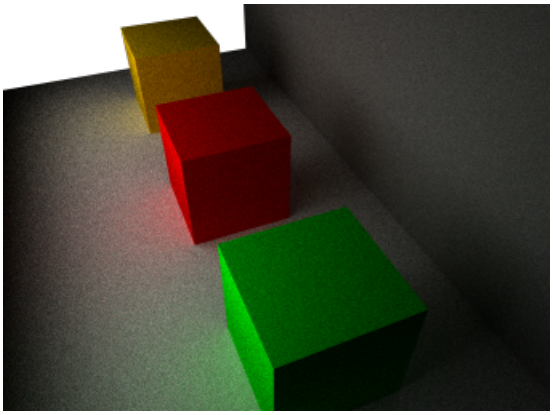
Die Qualität der Spiegelkugel ist bei allen Bildern sehr ähnlich. Auf Bild a) ist diese Szene mittels Standard-Raytracer gerendert. Dabei fällt auf, dass dieses Verfahren einen sehr harten Schatten bei der spiegelnden Kugel erzeugt. Bei der transparenten Kugel ist hingegen kein Schatten zu erkennen.
Bild b) zeigt den Path Tracer. Dieser erzeugt ein sehr verrauschtes Bild, obwohl für die Berechnung dieses Bildes 400 Strahlen pro Bildpixel in die Szene gesendet wurden. Der Schatten hinter der spiegelnden Kugel ist wesentlich realistischer als beim Standard-Raytracer. Ein Schatten oder Kaustik unter der transparenten Kugel ist aufgrund des Rauschens nur schwer zu erkennen.
Das Bild c) zeigt die Szene mittels Photon-Mapping-Verfahren gerendert. Dabei ist ein sehr weicher Schatten unter der spiegelnden Kugel zu erkennen. Auch die transparente Kugel erzeugt eine realistische Kaustik. Dies ist weder beim Standard-Raytracer noch beim Path Tracer zu erkennen. Weiterhin ist das Bild auch wesentlich rauschfreier als das des Path Tracers.

5.4 Diffus reflektierende Objekte

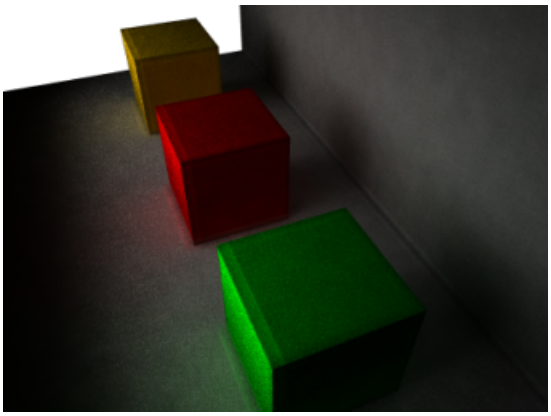
Die meisten Objekte in der realen Welt besitzen eine raue Oberfläche. Mit der auf Abbildung 5.3 dargestellten Szene soll getestet werden, ob auch diese diffus reflektierenden Objekte Spiegelungen erzeugen. Dazu besteht die Beispielszene aus einer Grund- und Wandfläche, sowie 3 verschiedenfarbigen Würfeln. Alle Objekte dieser Szene besitzen Oberflächen, welche diffus reflektieren.
Auf Bild a) ist zu erkennen, dass die Würfel beim Standard-Raytracer keinerlei Reflexionen auf der Grundfläche erzeugen. Auf den Bildern der anderen drei Rendering-Verfahren ist eine farbige Spiegelung auf der Grundfläche zu erkennen. Alle diese Verfahren unterstützen diffuse Reflexion.
Die Bilder b) und c) vom Path Tracer bzw. vom Photon-Mapping sind etwas verrauscht. Der Grund hierfür ist, dass die Szene von einer Flächenlichtquelle beleuchtet wird. Das Radiosity-Verfahren auf Bild d) hat dieses Problem nicht, allerdings sind einzelne Dreiecke zu erkennen, was auf die noch fehlende Interpolation zwischen benachbarten Dreiecken zurückzuführen ist (siehe Abschnitt 4.5).



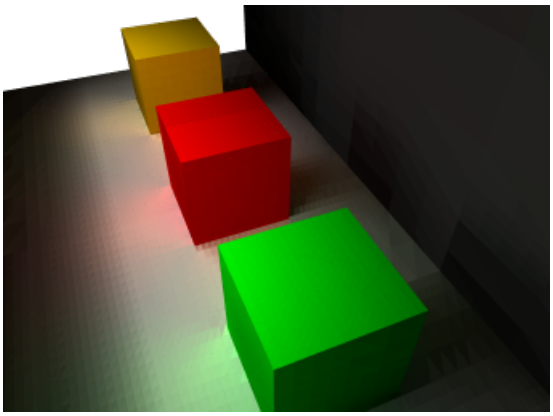
a) Standard-Raytracer



b) Path Tracer



c) Photonen Map



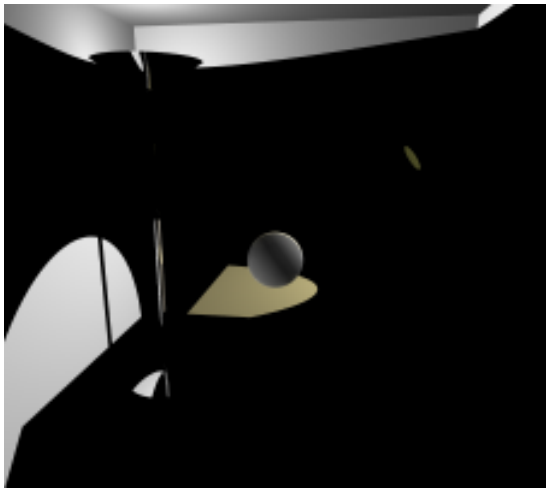
d) Radiosity

Abbildung 5.3: Vergleich diffus reflektierender Objekte

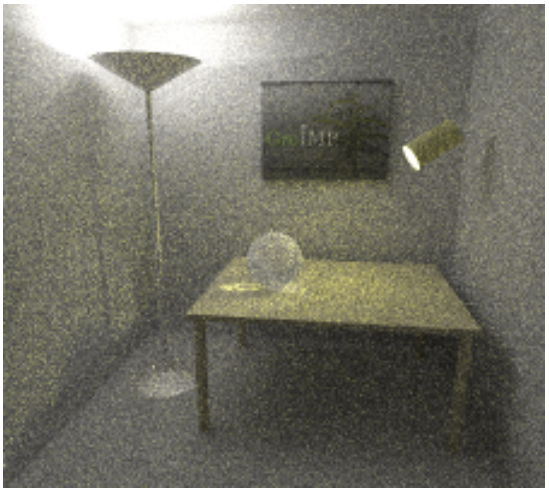
5.5 Komplexes Beispiel

Zum Abschluss möchte ich die verschiedenen Rendering-Verfahren anhand einer recht aufwendigen Szene vergleichen. Dabei treten alle zuvor erläuterten problematischen Situationen gemeinsam auf. Sie wird größtenteils durch indirekte Lichteinstahlung beleuchtet, enthält eine transparente Kugel und ist mit verschiedenen Texturen ausgestattet. Das Radiosity-Verfahren wurde dabei nicht mit betrachtet, da es aufgrund der erwähnten Einschränkungen der momentanen Implementierung (siehe 4.4) nicht für diese Szene

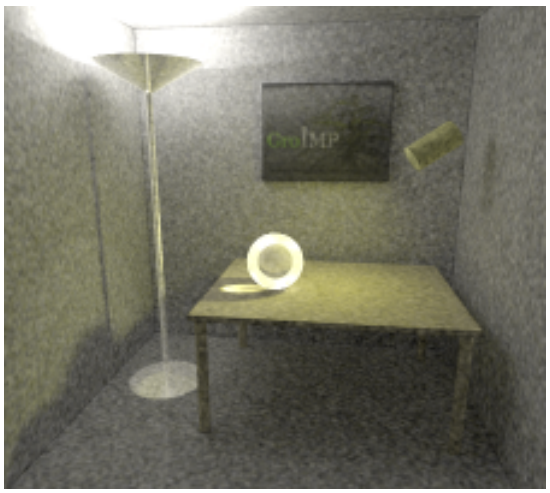
geeignet ist.
Diese Szene wurde der Abbildung 10.3 der Arbeit von Eric Veach [Vea97] nachempfunden, welche die Grundlage für die Arbeit von Hagen Steidelmüller bildet.



a) Standard-Raytracer



b) Path Tracer



c) Photonen Map

Abbildung 5.4: Vergleich anhand eines komplexen Beispiels

Auch bei dieser Szene erzeugt der Standard-Raytracer ein unbrauchbares Bild. Nur die Stellen, die direkt von einer Lichtquelle beleuchtet werden, sind erkennbar.

Der Path Tracer erzeugt eine bessere Beleuchtung der Szene, allerdings ist das Bild sehr stark verrauscht, da benachbarte Pixel teilweise eine sehr hohe Farbdifferenz haben. Das Photon-Mapping-Verfahren erzeugt eine klar zu erkennende Kaustik, diese ist auf dem Path Tracer-Bild nur angedeutet. Auch das Photon-Mapping-Bild ist leicht verrauscht. Dies entsteht durch die zu geringe Anzahl an Photonen in den dunklen Bereichen des Bildes. Um dies zu mindern, müsste man die Anzahl der Photonen weiter erhöhen, was schnell zu einem recht hohen Speicherbedarf führt. Die in Abschnitt 3.6 beschriebenen Projektionskarten könnten hier zu einem besseren Ergebnis führen.

5.6 Parameter der Vergleichsbilder

Die nachfolgende Tabelle zeigt die verwendeten Parameter für die in diesem Kapitel dargestellten Bilder:

	Bild 5.1	**Bild 5.2**	**Bild 5.3**	**Bild 5.4**
Auflösung	344x210	512x277	341x254	450x402
Recursion Depth	5	5	5	5
Standard-Raytracer				
supersampling Grid	5x5	5x5	5x5	5x5
Path Tracer				
supersampling Grid	20x20	20x20	20x20	20x20
Photon-Mapping				
supersampling Grid	5x5	5x5	5x5	5x5
Global Photons	500000	1500000	1500000	1500000
Caustic Photons	0	1000000	0	1000000
Photon area	0.08	0.04	0.08	0.08
Recursion Depth	4	4	5	5
Radiosity				
supersampling Grid	5x5	-	5x5	-
Hemicube Pixel size	100	-	100	-
Hemicube World size	1.0	-	0.1	-
Subdiv. threshold	2.0	-	2.0	-
Subdiv. depth	6	-	6.0	-

Tabelle 5.1: Parameter der Vergleichsbilder

5.7 Berechnungsdauer der Vergleichsbilder

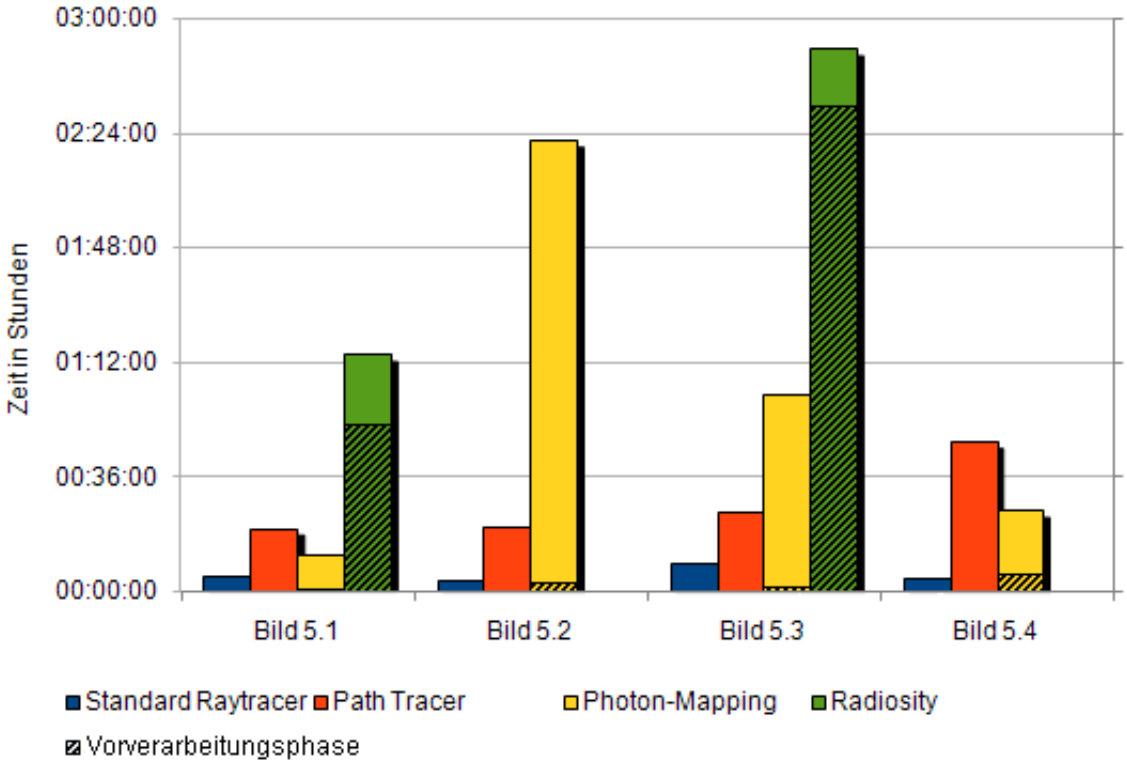


Abbildung 5.5: Berechnungsdauer der Vergleichsbilder

Die Abbildung 5.5 zeigt die Berechnungsdauer der Bilder in diesem Kapitel. Das Photon-Mapping- und Radiosity-Verfahren sind jeweils in eine Vorverarbeitungs- und eine Renderphasen unterteilt. Die Vorverarbeitungsphasen sind als schraffierte Balken im Diagramm dargestellt.
Während dieser ersten Phase erzeugt der Photon-Mapping-Algorithmus die Photonen-Map (siehe Abschnitt 3.1). Beim Radiosity-Algorithmus wird während dieser Phase die Unterteilung der Objekte in Dreiecke und die Lichtverteilung in der Szene berechnet (siehe Abschnitt 4.1). Bei beiden Verfahren muss diese Vorverarbeitungsphase nur einmal berechnet werden. Nach dem Ändern des Blickwinkels entspricht die Berechnungsdauer nur noch dem unschraffierten Bereich des entsprechenden Balkens.

A Literaturverzeichnis

[CG85] Cohen, Michael F. ; Greenberg, Donald P.: The hemi-cube: a radiosity solution for complex environments. In: *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1985. – ISBN 0–89791–166–0, S. 31–40

[CWH93] Cohen, Michael F. ; Wallace, John ; Hanrahan, Pat: *Radiosity and realistic image synthesis*. San Diego, CA, USA : Academic Press Professional, Inc., 1993. – ISBN 0–12–178270–0

[Deu03] Deussen, Oliver: *Script Computergrafik III*. TU-Dresden, 2003

[Gro07] GroIMP: *Internetseite von GroIMP*. URL: http://www.grogra.de/software/groimp, Stand: 03. August 2007

[GTGB84] Goral, Cindy M. ; Torrance, Kenneth E. ; Greenberg, Donald P. ; Battaile, Bennett: Modeling the interaction of light between diffuse surfaces. In: *SIGGRAPH Comput. Graph.* 18 (1984), Nr. 3, S. Seiten 213–222. – ISSN 0097–8930

[JC95] Jensen, Henrik W. ; Christensen, Niels J.: Photon maps in bidirectional Monte Carlo ray tracing of complex objects. In: *Computers & Graphics 19* (1995), S. 215–224

[Jen96] Jensen, Henrik W.: *The photon map in global illumination.*, Technical University of Denmark, Diss., 1996

[Jen04] Jensen, Henrik W.: A Practical Guide to Global Illumination using Ray Tracing and Photon Mapping. In: *Siggraph 2002 Course 19* (2004)

[Kur04] Kurth, Winfried: *Script Computergrafik*. BTU-Cottbus, 2004

[Kur07] Kurth, Winfried: *Script Algorithmische Geometrie*. BTU-Cottbus, 2007

[Lev07] Levy, Simon D.: *Java class library for KD-tree.* URL: http://www.cs.wlu.edu/~levy/software/kd, Stand: 03. August 2007

[Maa01] Maass, Stefan: *Effiziente Photonmap-Techniken*, Technische Universität Braunschweig, Diplomarbeit, 2001

[Ott99] Ott, Matthias: *Hybride Beleuchtungssimulation für Flächen und Volumen mittels Photon Maps und Radiosity*, Friedrich-Alexander-Universität Erlangen-Nürnberg, Diplomarbeit, 1999

[Roh07] Rohrer, Jason: *RadiosGL.* URL: http://hcsoftware.sourceforge.net/RadiosGL/RadiosGL.html, Stand: 03. August 2007

[RS03] Richter, Marcus ; Schäfe, Dörte: *Advanced Raytracing.* Seminar 3D-Grafik, Humboldt Universität Berlin, 2003

[Sch98] Schregle, Roland: *Rendering with Photon Maps*, University of Bonn, Diplomarbeit, 1998

[Vea97] Veach, Eric: *Robust Monte Carlo Methods for Light Transport Simulation*, Stanford University, Diss., 1997

[YLS05] Yu, Tin-Tin ; Lowther, John ; Shene, Ching-Kuang: Photon mapping made easy. In: *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education.* New York, NY, USA : ACM, 2005. – ISBN 1–58113–997–7, S. Seiten 201–205