UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Ambient Occlusion zwischen sich frei bewegenden Starrkörpern mittels Coherent Shadow Maps

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von
Stefan Müller

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dipl.-Inform. Tobias Ritschel (Max-Planck-Institut für Informatik, Abteilung für Computergraphik)

Koblenz, im April 2008

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)

Danksagung

Ich möchte mich an dieser Stelle bei allen Personen bedanken, deren lenkende Hinweise, Hilfestellungen und motivierendes Feedback wesentlich zu der vorliegenden Arbeit beigetragen haben.

Mein Dank richtet sich insbesondere an Prof. Dr.-Ing. Stefan Müller, der mich auf CSMs und ihr Potential aufmerksam gemacht hat und sich in den letzten sechs Monaten stets die Zeit genommen hat meine Fortschritte nachzuvollziehen und meine Fragen zu beantworten. Des Weiteren danke ich Dipl.-Inform. Tobias Ritschel, der mir zahlreiche Hinweise zur Umsetzung von CSMs gegeben hat und meine Aufmerksamkeit auf Quasi-Monte-Carlo-Techniken gelenkt hat, die sich für diese Arbeit als besonders nützlich erwiesen haben.

Außerdem danke ich meiner Familie und meiner Freundin Nina für die Unterstützung und das entgegengebrachte Verständnis während der letzten fünf Jahre.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

BRDF Bidirektionale Reflexionsverteilungsfunktion

bzw. beziehungsweise

CDF *Cumulative Distribution Function*

Cg *C for graphics*

CPU *Central Processing Unit*

CSM *Coherent Shadow Map*

CSSM *Coherent Surface Shadow Map*

DCC *Digital Content Creation*

d.h. das heißt

DTD Dokumenttyp-Definition

DXUT *DirectX Utility Toolkit*

engl. englisch

et al. *et alii*

FPS *Frames* pro Sekunde

GPU *Graphics Processing Unit*

HDR *High-Dynamic Range*

HLSL *High-Level Shader Language*

Hrsg. Herausgeber

LDS *Low-Discrepancy Sequences*

MB Megabyte

MEL *Maya Embedded Language*

OpenGL *Open Graphics Library*

Pixel *Picture Element*

PRT *Precomputed Radiance Transfer*

RGB Rot-Grün-Blau

RGBA Rot-Grün-Blau-Alpha

S. Seite

SDK *Software Development Kit*

SFR *Split Frame Rendering*

SH *Spherical Harmonics*

SLI *Scalable Link Interface*

SSAO *Screen-Space Ambient Occlusion*

STL *Standard Template Library*

Texel *Texture Element*

u.a. unter anderem

vgl. vergleiche

Voxel *Volumetric Pixel*

W3C *World Wide Web Consortium*

XML *Extensible Markup Language*

1 Einleitung

Die Vorstellung eines Tages die Welt, wie sie von dem menschlichen Sehapparat wahrgenommen wird, durch einen Computer simulieren zu können, hat die Forschung innerhalb der Computergraphik in den letzten Jahrzehnten enorm vorangetrieben. Die Generierung photorealistischer Szenen in einem Programm, welches eine Interaktion mit dem Dargestellten ermöglicht, erfordert ein Verständnis von Licht das sich mathematisch ausdrücken lässt. Aus diesem Grund fußt die photorealistische Computergraphik zu großen Teilen auf Gebieten der Physik, wie etwa die Photometrie oder die Thermodynamik. Diese bieten eine Erklärung für die Verbreitung von Licht, welches in der Regel von mehreren Flächen in unterschiedlichem Ausmaß reflektiert wird und damit auch Stellen erreicht, die von einer Lichtquelle nicht direkt beleuchtet werden. Eben diese globale Natur von Licht macht eine Simulation enorm aufwändig, sodass die Erzeugung eines einzigen Bildes selbst mit modernsten Computern mehrere Stunden dauern kann.

Einige grundlegende Konzepte zur Erzeugung realistischer Bilder sind bereits seit Jahrhunderten bekannt und auf die Beobachtungen bedeutender Renaissance-Künstler zurückzuführen. Diese haben sich auch erstmals intensiv mit den Eigenschaften von Schatten beschäftigt und so beispielsweise festgestellt, dass Schatten zum Rand hin in der Regel einen weichen Übergang besitzen und, dass das Ausmaß dieses Halbschattens mit der Größe der Lichtquelle zunimmt. Auch wenn dieses Phänomen schon lange bekannt ist und jedem selbstverständlich erscheint, so stellt seine Berechnung in einer computergestützten Simulation der Realität eine so große Herausforderung dar, dass nach wie vor nach neuen Verfahren gesucht wird, die in kürzester Rechenzeit ein plausibles Ergebnis liefern.

Diese Arbeit beschäftigt sich mit der Berechnung des Anteils an Umgebungslicht, der einen beliebigen Punkt innerhalb einer Szene beleuchtet. Dieses Licht hat seinen Ursprung in allen denkbaren Richtungen und um zu ermitteln, wie viel davon bei dem zu untersuchenden Punkt ankommt, muss die Verdeckung durch alle übrigen Elemente der Szene berücksichtigt werden. Ist der Anteil des verdeckten direkten Umgebungslichts für einen Punkt bekannt, so kann darauf geschlossen werden, dass indirektes, d.h. von mehreren Flächen reflektiertes Licht in demselben Maße daran gehindert wird diesen Punkt zu beleuchten. Diese Beobachtung hat man sich erstmals in Filmproduktionen zunutze gemacht, um durch die Gewichtung einer konstanten ambienten Lichtfarbe mit dem individuellen, von der umliegenden Geometrie abhängigen Verdeckungsgrad auf wesentlich aufwändigere, globale Beleuchtungssimulationen verzichten zu können und damit Zeit und Kosten einzusparen. Obwohl es sich nur um eine Approximation handelt, ist der optische Vorteil gegenüber einer herkömmlichen direkten Beleuchtung durch einzelne Punktlichtquellen und den damit verbundenen harten Schattenkanten eindeutig.

Die steigende Leistungsfähigkeit moderner Graphikhardware und die zunehmende Flexibilität bei der Programmierung derselben haben dazu geführt, dass in den letzten vier Jahren vermehrt Ansätze zur performanten Berechnung des so genannten *Ambient Occlusion* in interaktiven Darstellungsraten erschienen sind. In diese Liste reiht sich konzeptionell auch das in dieser Arbeit beschriebene Verfahren ein, das auf einem kürzlich veröffentlichten, verlustlosen Kompressionsschema für *Depth Maps*, welches den Namen *Coherent Shadow Maps* trägt, basiert. Diese Arbeit untersucht, wie sich Coherent Shadow Maps gezielt zur Berechnung von Ambient Occlusion in Szenen mit frei transformierbaren Starrkörpern nutzen lassen.

Der Aufbau der vorliegenden Arbeit gestaltet sich folgendermaßen: Dieser Einleitung folgt eine Erörterung der Grundlagen. Diese umfassen eine Vorstellung der gebräuchlichsten Vorgehensweisen zur Umsetzung globaler, bzw. ausschließlich lokaler Beleuchtungssimulationen, sowie eine Erläuterung des *Shadow Mappings*, welches häufig zur Berechnung von Schatten eingesetzt wird und das Fundament des in dieser Arbeit beschriebenen Verfahrens zur Darstellung von Ambient Occlusion bildet. Mit Ambient Occlusion befasst sich Kapitel 3. Die Motive des Ansatzes werden dargestellt und neben der zugrunde liegenden Mathematik wird das Vorgehen in den zu diesem Thema bislang veröffentlichten Arbeiten nachgezeichnet. In Kapitel 4 wird die Komprimierung durch Coherent Shadow Maps vorgestellt und erläutert, wie sich anhand der Datenstruktur die Visibilität feststellen lässt. Kapitel 5 bildet den Hauptteil dieser Arbeit und befasst sich mit der Berechnung von Ambient Occlusion mittels Coherent Shadow Maps. Zum Ende des Kapitels wird das im Rahmen dieser Arbeit implementierte System vorgestellt. In Kapitel 6 werden die mit dem im vorausgehenden Kapitel erläuterten Vorgehen erzielten Ergebnisse präsentiert und bewertet. Kapitel 7 zieht ein Fazit und schließt die Arbeit mit einem Ausblick auf zukünftige Entwicklungsmöglichkeiten ab.

2 Grundlagen

Die Erzeugung photorealistischer Bilder am Computer geht eng mit der Simulation von Licht einher. In der Regel wird alles was der Mensch im Alltag betrachtet nicht direkt von einer einzigen Lichtquelle beleuchtet, sondern auch von dem Licht, das benachbarte Objekte zu einem Teil reflektieren. Die Simulation solcher globalen Beleuchtungseffekte ist ein aufwändiger Prozess. Oft ist es jedoch nicht akzeptabel mehrere Stunden auf die Berechnung eines einzigen Bildes zu warten und so werden häufig Vereinfachungen auf Kosten physikalischer Plausibilität in Kauf genommen.

Auch Schatten ist ein wichtiges Phänomen, das wesentlich dazu beiträgt eine virtuelle Szene als realistisch einzustufen und welches darüber hinaus hilft die Anordnung von Objekten innerhalb der Szene zu bestimmen. Allerdings ist auch die Berechnung von Schatten kein einfach zu lösendes Problem und es existiert keine Herangehensweise, die ohne Einschränkungen zu empfehlen wäre.

2.1 Modelle zur Beleuchtung von Geometrie

Die Form und das Material eines dreidimensionalen Objekts erschließen sich für einen Betrachter durch die Art, wie Licht auf das Objekt trifft. Beleuchtungsmodelle erlauben unterschiedliche Materialeigenschaften und Beleuchtungsverhältnisse anhand einstellbarer Parameter zu simulieren. Die Gratwanderung zwischen Realismus und Echtzeitfähigkeit führte dazu, dass in den gängigsten Beleuchtungsmodellen bestimmte Charakteristika des Lichts nur durch konstante Werte approximiert werden. Die Leistungsfähigkeit moderner Graphikhardware[1] ermöglicht es heute einige der Annahmen, die vor über 30 Jahren gemacht wurden um interaktive Computergraphik zu ermöglichen, durch genauere Berechnungen zu ersetzen und dadurch deutlich realistischere Ergebnisse zu erzielen.

2.1.1 Rendering Equation

Die von Kajiya [Kaj86] vorgestellte *Rendering Equation* beschreibt die Verteilung von Licht in einer dreidimensionalen Szene und ist von großer Bedeutung, da sie die mathematische Basis für sämtliche Beleuchtungsmodelle darstellt. Ein Vergleich mit der Rendering Equation ermöglicht es, eine Aussage über die Plausibilität von Render-Techniken zu machen.

[1] Ein grundlegendes Wissen zur Programmierung moderner Graphikhardware, das Begriffe wie *Fragment Program* oder *Vertex Program* einschließt, wird für ein Verständnis dieser Arbeit vorausgesetzt. Einen empfehlenswerten Einstieg in das Thema bietet das Buch „The Cg Tutorial“ [FK03].

Gleichung 2.1 zeigt die Rendering Equation in der Repräsentation eines Integrals über den gesamten Halbraum eines infinitesimalen Flächenelements dA_e:

$$L_\mathrm{o}(dA_\mathrm{e}, d\omega_\mathrm{o}) = L_\mathrm{e}(dA_\mathrm{e}, d\omega_\mathrm{o}) + \int_{\mathbf{\Omega}} f_\mathrm{r}(dA_\mathrm{e}, d\omega_\mathrm{i}, d\omega_\mathrm{o}) \cdot L_\mathrm{i}(dA_\mathrm{e}, d\omega_\mathrm{i}) \cdot \cos\theta_\mathrm{i} \cdot d\omega_\mathrm{i} \; . \tag{2.1}$$

Die Leuchtdichte L_o, die von einem Flächenelement dA_e in Richtung $d\omega_\mathrm{o}$ ausgestrahlt wird, entspricht demnach der emittierten Leuchtdichte L_e des Flächenelements in eben diese Richtung, plus allen aus der Hemisphäre $\mathbf{\Omega}$ auf das Flächenelement auftreffenden Leuchtdichten L_i, die in die Richtung $d\omega_\mathrm{o}$ reflektiert werden. Letztere werden mit dem Winkel θ_i zwischen der Einfallsrichtung $d\omega_\mathrm{i}$ und der Normalen des Flächenelements gewichtet. Die Reflexionseigenschaften eines Materials unter allen möglichen Lichteinfallswinkeln werden mit $f_\mathrm{r}(dA_\mathrm{e}, d\omega_\mathrm{i}, d\omega_\mathrm{o})$ durch die Bidirektionale Reflexionsverteilungsfunktion (BRDF) beschrieben, die sich aus dem Verhältnis von ausgehender Leuchtdichte und eingehender Beleuchtungsstärke berechnet.

Gleichung 2.1 zeigt, dass die Beleuchtung von Objekten nicht ausschließlich durch explizite Lichtquellen stattfindet. Es ist vielmehr so, dass die meisten Materialien einen Teil des Lichts, das auf sie trifft, reflektieren und damit wiederum andere Flächen indirekt beleuchten. Diese rekursive Natur der Rendering Equation macht die Berechnung einer realistischen Beleuchtung unter Berücksichtigung globaler Beleuchtungseffekte enorm aufwändig. Drei besonders bekannte Algorithmen, die sich als Ansätze zur Erzeugung photorealistischer Bilder mit der Lösung der Rendering Equation befassen, sind *Ray Tracing*, *Radiosity* und *Precomputed Radiance Transfer*.

2.1.2 Simulation globaler Beleuchtung

Ray Tracing wurde in der am häufigsten erwähnten Form 1980 von Whitted [Whi80] vorgestellt. Die Grundidee des Verfahrens besteht darin, durch jedes Bildschirmpixel einen oder mehrere[2] Strahlen zu verschicken und den ersten Schnittpunkt mit einem Objekt der Szene zu ermitteln. Je nach Materialeigenschaften dieses Objektes, werden von dem Schnittpunkt aus Sekundärstrahlen weiter verschickt. Dieses Vorgehen ermöglicht intuitiv die Berücksichtigung von Reflexionen und Refraktionen, sowie die Berechnung von Schatten durch die Erzeugung spezieller Strahlen („Schattenfühler“), die in Richtung der Lichtquellen versendet werden. Weiterentwicklungen dieser Grundidee, wie die gezielte Verteilung von Strahlen [CPC84], haben neue Effekte – beispielsweise die Erzeugung weicher Schatten, Tiefenschärfe und *Motion Blur* – ermöglicht, und den Weg für die Berechnung des Transports von Licht über mehrere Oberflächen geebnet um das Problem der globalen Beleuchtungssimulation auf diese Weise zu lösen. Algorithmen, die unter dem Begriff *Light Tracing* zusammengefasst werden, verschicken Strahlen von den Lichtquellen aus, was intuitiv der

[2] Werden keine zusätzlichen Sekundärstrahlen pro Schnittpunkt, dafür jedoch mehrere Strahlen pro Bildschirmpixel erzeugt, wird in der Regel von *Path Tracing* gesprochen.

Verbreitung von Licht entspricht und spezielle Effekte, wie beispielsweise Kaustiken, ermöglicht. Dieses Vorgehen kommt auch bei dem besonders populären *Photon Mapping* [JC95] zum Einsatz. Hier werden zunächst Photonen von der Lichtquelle aus in die Szene verschossen, weiter verfolgt und in einer Datenstruktur festgehalten. Im zweiten Schritt wird diese Datenstruktur zur Bilderzeugung herangezogen.

Die Berechnung photorealistischer Bilder mittels Ray Tracing ist enorm zeitaufwändig. Der Aufwand ist insbesondere von der Auflösung des darzustellenden Bildes abhängig. Trotz des technischen Fortschritts von Computer- und Graphikprozessoren, die es mittlerweile erlauben bestimmte Operationen parallel auszuführen, und zahlreicher Beschleunigungsstrategien, die in den letzten Jahren für Ray Tracing vorgeschlagen wurden, hat sich Ray Tracing bislang für Echtzeitanwendungen[3], wie *Virtual Reality*, nicht gegen die *scanline*-orientierte Rasterisierung von Polygonen durchsetzen können.

Beim *Radiosity*-Verfahren [GTGB84] wird eine Szene möglichst adaptiv derart in finite Elemente unterteilt, dass für jedes Element über seine gesamte Fläche und in alle Richtungen eine konstante Leuchtdichte angenommen werden kann. Um den Strahlungsaustausch zwischen zwei solchen Flächenelementen zu berechnen, wird mit dem Formfaktor ermittelt, wie groß der Anteil versendeter Strahlung ist, der beim Empfänger ankommt. Neben der Berechnung des Formfaktors kostet auch das Auswerten der Sichtbarkeit zwischen zwei sich zugewandten Flächenelementen besonders viel Rechenzeit. Um das Verfahren zu beschleunigen kann mit Hilfe einer hierarchischen Organisation der Elemente individuell entschieden werden, auf welcher Ebene es ausreicht den Strahlungsaustausch zu ermitteln [HSA91].

Ein Vorteil des Radiosity-Verfahrens im Vergleich zum Ray Tracing ist die unmittelbare Unterstützung indirekter Beleuchtungseffekte, wie das so genannte *Color Bleeding*. Ist der Strahlungsaustausch für eine statische Szene einmal ermittelt, kann die Kamera im Gegensatz zum Ray Tracing beliebig bewegt werden. Allerdings basiert das Verfahren auf der Annahme, dass alle Flächenelemente ideal diffuse Reflektoren sind und die Unterstützung dynamischer Objekte erfordert auch bei geschickter Organisation der Flächenelemente eine hohe Rechenzeit.

Precomputed Radiance Transfer (PRT) steht für eine Reihe moderner Techniken, die eine Darstellung komplexer Beleuchtungssituationen in Echtzeit durch eine Vorberechnung der Verbreitung von Licht ermöglichen. Ursprünglich basiert PRT auf der Annahme, dass eine statische Szene aus diffusen oder glänzenden Materialien besteht und durch unendlich weit entfernte Lichtquellen, beispielsweise in Form einer *High-Dynamic Range Environment Map*, beleuchtet wird [SKS02]. Dies ermöglicht eine Vereinfachung der Rendering Equation, die sich anschließend in einen vorberechneten Teil, der auch die Selbstverschattung berücksichtigt, und einen dynamisch zu berechnenden Teil, der sich Beleuchtungsänderungen anpasst, zerlegen lässt. Zur Repräsentation natürlicher Beleuchtungssituationen haben sich sphärisch-harmonische

[3] Von Echtzeit wird in der Regel gesprochen, wenn mindestens 30 Bilder (engl. *Frames*) pro Sekunde erzeugt werden. Daraus ergibt sich für einen menschlichen Betrachter der Effekt einer unterbrechungsfreien Darstellung.

Funktionen (engl. *Spherical Harmonics*, SH) als besonders hilfreiches Werkzeug erwiesen. So genügen gerade einmal neun SH-Koeffizienten zur Rekonstruktion von niedrigfrequentem Umgebungslicht mit einem kaum wahrnehmbaren Fehler [RH01]. Spätere Veröffentlichungen – Dutré et al. [DBB06] geben einen Überblick – haben PRT um die Unterstützung willkürlicher BRDFs und hochfrequenter Beleuchtung erweitert, wobei die Vorberechnungszeit der Verfahren in der Regel jedoch sehr hoch ist und dynamische Szenen nach wie vor eine Herausforderung darstellen.

2.1.3 Simulation lokaler Beleuchtung

Um eine Beleuchtung in Echtzeit zu ermöglichen, sodass die Position von Objekten und Lichtquellen interaktiv verändert werden darf, mussten Vereinfachungen gegenüber einer globalen Simulation von Licht vorgenommen werden. Der Aufbau von Objekten aus Dreiecken ermöglicht es, nur die Eckpunkte (engl. *Vertices*) zu beleuchten und dazwischen bi-linear zu interpolieren. Eine Möglichkeit genauere Ergebnisse zu erzielen besteht darin, für jeden darzustellenden Punkt eines Dreiecks das Beleuchtungsmodell auf eine Normale anzuwenden, die aus den Normalen der Eckpunkte interpoliert wird. In beiden Fällen wird jeder Punkt eines Polygons unabhängig von allen anderen Elementen der Szene beleuchtet, sodass ohne ein zusätzliches Verfahren keine Schatten und kein reflektiertes Licht berücksichtigt werden.

Beleuchtungsmodelle nehmen BRDFs ihre Komplexität, indem sie die Simulation unterschiedlicher Materialien mit möglichst intuitiven Parametern erlauben. Häufig sind diese Vereinfachungen jedoch nicht physikalisch plausibel und auf wenige Materialien zugeschnitten. Zwei der frühesten Beleuchtungsmodelle, die auch heute noch häufig eingesetzt werden, wurden von Gouraud [Gou71] und Phong [Pho75] entwickelt. Das erste der beiden Modelle eignet sich lediglich für diffuse Materialien, die dem Lambertschen Gesetz gehorchen, nach dem die Farbe einer Fläche sich proportional zum Winkel zwischen Flächennormale und Lichteinfallsrichtung berechnet. Das Modell von Phong berücksichtigt dagegen zusätzlich die Position des Betrachters und erlaubt Materialien, in denen sich die Lichtquelle spiegelt. Moderne Beleuchtungsmodelle sind häufig komplexer. Sie ermöglichen dafür aber eine realistischere Nachbildung vieler Materialien. Ein Beispiel ist das physikalisch basierte Modell von Cook-Torrance [CT82], das auf einer Verteilung von Mikrofacetten und den Fresnel-Gleichungen aufbaut.

Die beiden Graphikbibliotheken *OpenGL* und *Direct3D* bieten durch eine Kombination der Modelle von Gouraud und Phong, in einer Art wie sie die folgende Gleichung zeigt, Parameter für eine diffuse und eine spiegelnde Reflexion an:

$$c = c_{\mathrm{r}}(c_{\mathrm{a}} + c_{\mathrm{l}} \max(0, \mathbf{n} \cdot \mathbf{l})) + c_{\mathrm{l}} c_{\mathrm{p}} \max(0, \mathbf{e} \cdot \mathbf{r})^{p} \; . \tag{2.2}$$

Die Notation der Gleichung richtet sich nach Shirley [Shi02]. Die Farbe c eines darzustellenden Punktes berechnet sich aus der Normalen $\mathbf{n}$, dem Lichtvektor $\mathbf{l}$, der Richtung des reflektierten Lichtes $\mathbf{r}$, sowie der Richtung zum Augpunkt des Betrachters $\mathbf{e}$.

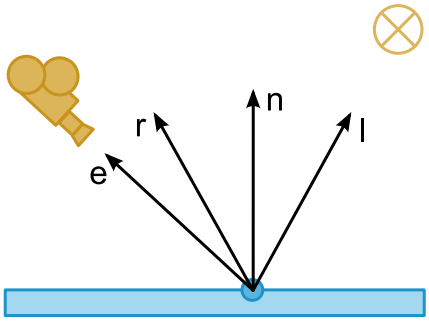
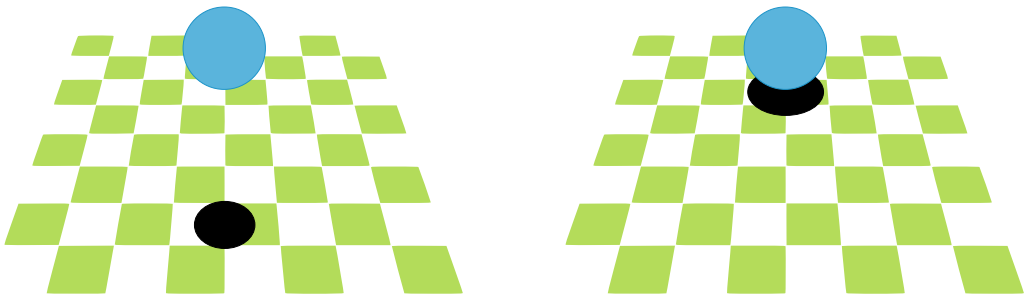


Abbildung 2.1: Vektoren der klassischen lokalen Beleuchtungsmodelle.

Je kleiner der Winkel zwischen den letzten beiden Vektoren ist, umso mehr spiegelt sich die Lichtquelle im darzustellenden Punkt. Der Exponent p regelt die Ausmaße dieser Spiegelung. Abbildung 2.1 stellt die Vektoren aus Gleichung 2.2 dar. Bei allen weiteren Variablen handelt es sich um RGB-Tripel. Das Material wird durch die diffuse Reflexionsfarbe c_r und die Glanzfarbe c_p definiert. Die Farbe der Lichtquelle geht mit c_l in die Gleichung ein. Um Geometrie die nur indirekt beleuchtet wird nicht ausschließlich schwarz darzustellen, wurde der ambiente Term c_a in das Beleuchtungsmodell integriert. Er dient der Approximation von indirektem Licht, das über umliegende Flächen hinweg reflektiert wird. Durch die geschickte Wahl einer ambienten Reflexionsfarbe ist es in Einzelfällen möglich ein überzeugendes Ergebnisbild zu erzielen. Da die ambiente Farbe einer Fläche jedoch prinzipiell konstant und völlig unabhängig von der umgebenden Geometrie ist, lässt sich in Bereichen die nicht direkt beleuchtet werden keine Struktur ausmachen.

Neben der unabhängigen Beleuchtung einzelner Punkte wird die Performanz dadurch gesteigert, dass von den gängigen Graphikbibliotheken nur bestimmte Arten von Lichtquellen unterstützt werden. Punktlichtquellen besitzen keine Ausdehnung und strahlen in alle Richtungen gleich viel Licht aus. Spotlichtquellen geben ihr Licht dagegen nur in eine bestimmte Richtung ab. Der Winkel zwischen dieser Richtung und dem Lichtvektor erlaubt dabei die Nachbildung des Halbschattens, der typisch für Flächenlichtquellen ist. Der Ursprung des Lichts wird jedoch auch hier als unendlich klein angenommen. Gerichtete Lichtquellen werden dagegen durch keine Position, sondern lediglich durch eine Richtung definiert. Sie approximieren damit annähernd unendlich weit entfernte Lichtquellen. Der Vorteil bei der Beschränkung auf die drei genannten Typen liegt darin, dass pro Lichtquelle und darzustellendem Punkt nur eine Richtung des Lichteinfalls berücksichtigt werden muss. Umgebungslicht, das einen Punkt aus allen Richtungen beleuchtet (wie etwa der Himmel), kann, ebenso wie indirekt einfallendes Licht, in den klassischen Beleuchtungsmodellen nur mit dem ambienten Term modelliert werden. Das Ergebnis leidet allerdings auch in diesem Fall daran, dass eine Verdeckung durch umgebende Geometrie komplett unberücksichtigt bleibt.



(a) (b)

Abbildung 2.2: Einsatz von Schatten zur eindeutigen Identifizierung der Position und Form von Objekten im Raum (nach [Wym04]).

2.2 Shadow Maps zur Erzeugung von Schatten

Rein lokale Beleuchtungsmodelle berücksichtigen keinen Schattenwurf. Dabei handelt es sich jedoch um ein wichtiges Phänomen, das die menschliche Wahrnehmung bei der Erkennung von Größe und Position von Objekten erheblich unterstützt. Abbildung 2.2 gibt hierfür ein Beispiel. Auf der Bildebene besitzt die blaue Kugel in beiden Illustrationen dieselbe Position und Größe. Lediglich der Schatten gibt Aufschluss darüber, dass sie sich im ersten Fall näher an der Kamera und in größerem Abstand über der karierten Ebene befindet. Des Weiteren lässt sich folgern, dass die Kugel im rechten Bild größer ist, da sie weiter von der Kamera entfernt ist und ihre Größe auf der Bildebene nach der Tiefenprojektion geringer wäre, wenn sie dieselben Ausmaße hätte, wie die Kugel im linken Bild. Zuletzt erlaubt erst der Schatten eine Aussage über die Form zu treffen – man könnte hier ebenso gut auf die Unterseite eines Kegels blicken. All diese Vergleiche finden in der menschlichen Wahrnehmung völlig unbewusst und in Sekundenbruchteilen statt. Die Voraussetzung ist jedoch, dass wichtige Anhaltspunkte, wie Schatten oder Beleuchtung, gegeben sind.

Zwei der bekanntesten Vorgehensweisen zur Erzeugung von Schatten sind *Shadow Volumes* [Cro77] und *Shadow Mapping* [Wil78]. Bei dem erstgenannten Verfahren wird um den Bereich, in dem ein Objekt Schatten wirft, anhand der Objekt-Silhouette aus Richtung des Lichtes, ein Volumen aufgespannt. Dies erfordert die Organisation der Objekte in einer Datenstruktur, die es erlaubt Nachbarschaftsinformationen zwischen Polygonen auszuwerten. Häufig wird mit Hilfe des *Stencil Buffers* ermittelt, ob ein darzustellender Punkt innerhalb des Schattenvolumens liegt. Dabei müssen einige Spezialfälle berücksichtigt werden, beispielsweise wenn die Kamera selbst im Volumen positioniert ist. Die Rechenzeit des Verfahrens hängt von der Komplexität der Schatten werfenden Objekte ab. Insbesondere die Füllrate der Graphikhardware erweist sich als Engpass, da das Schattenvolumen in Form großflächiger Polygone gerendert wird. Shadow Mapping wird dagegen vollständig und effektiv von der modernen Graphikhardware unterstützt. Da es auf Texturen basiert,

hat das Shadow Mapping jedoch mit, von der Auflösung abhängigen Artefakten zu kämpfen. Weil das Verfahren in dieser Arbeit von großer Bedeutung ist, wird seine Funktionsweise im Folgenden genau beschrieben.

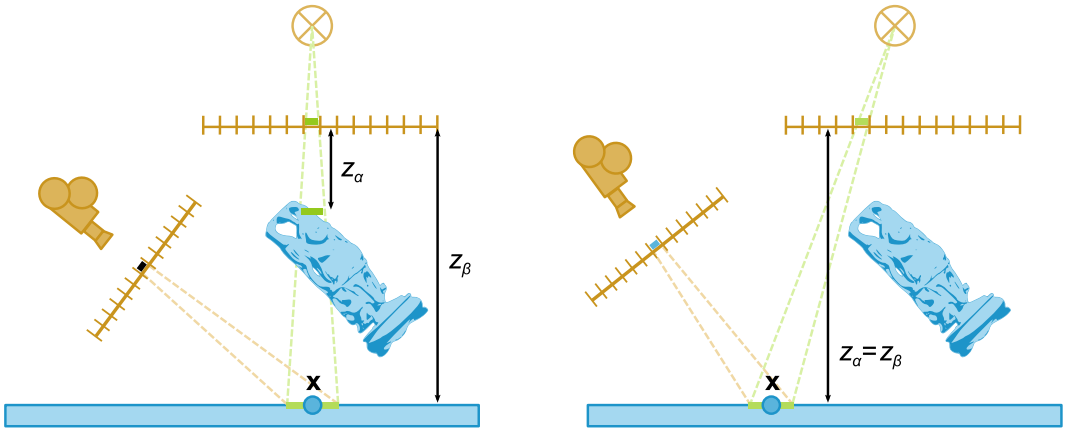
2.2.1 Beschreibung des Verfahrens

Die grundlegende Idee des Shadow Mapping ist, dass Punkte, die von einer Lichtquelle direkt beleuchtet werden, auch von einer Kamera gesehen werden können, die genau so ausrichtet und positioniert wird wie die Lichtquelle. Punkte, die dagegen im Schatten liegen werden von anderen Teilen verdeckt und sind daher nicht sichtbar. Die Analogie von Kamera und Lichtquelle beschränkt das Shadow Mapping in seiner einfachsten Form auf Spotlichtquellen, da diese durch eine Blickrichtung und einen Öffnungswinkel definiert werden.

Das Shadow Mapping basiert auf zwei Schritten. Zunächst wird die Szene aus Sicht der Lichtquelle gerendert. Dabei sind jedoch nur die Tiefenwerte jedes Pixels von Interesse. Diese werden in einer Textur gespeichert, die häufig *Shadow Map* oder *Depth Map* genannt wird. Im nächsten Schritt wird die Szene aus Sicht der Kamera gerendert. Die Entfernung eines von der Kamera sichtbaren Punktes zur Lichtquelle gibt, verglichen mit den Tiefenwerten der Shadow Map, Auskunft darüber, ob der Punkt im Schatten liegt. Um die Tiefe eines in Objektkoordinaten vorliegenden Eckpunktes aus Sicht der Lichtquelle zu ermitteln, sind eine Reihe von Transformationen, die sich durch Matrixmultiplikationen ausdrücken lassen, nötig:

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \text{Licht} \\ \text{Projekt.} \\ \text{Matrix} \\ (projection) \end{bmatrix} \begin{bmatrix} \text{Licht} \\ View \\ \text{Matrix} \\ (look\,at) \end{bmatrix} \begin{bmatrix} \text{Objekt} \\ \text{Transform.} \\ \text{Matrix} \\ (modeling) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{2.3}$$

Da die Koordinaten des Punktes nach der perspektivischen Transformation im *Clip Space* in einem Bereich zwischen -1 und 1 vorliegen, auf die Depth Map allerdings nur mit Werten zwischen 0 und 1 zugegriffen werden kann, teilt die letzte Matrix die Komponenten x, y und z durch 2 und addiert $1/2$. Für einen beliebigen Punkt wird das für die umliegenden Eckpunkte gültige Ergebnis von Gleichung 2.3 entsprechend allen übrigen Eckpunkt-Attributen interpoliert. Ein Tiefenvergleich wird anschließend zwischen dem an der Stelle $(s/q, t/q)$ aus der Depth Map ausgelesenen Wert z_α und der tatsächlichen Tiefe des Punktes $z_\beta = r/q$ vorgenommen. Abbildung 2.3 verdeutlicht die beiden Fälle zu denen es kommen kann. Entweder der Tiefenwert der Depth Map z_α ist kleiner als die tatsächliche Entfernung z_β des Punktes $\mathbf{x}$ zur Lichtquelle – in diesem Fall liegt der Punkt im Schatten –, oder die Tiefenwerte z_α und z_β sind gleich – dann wird Punkt $\mathbf{x}$ von der Lichtquelle beleuchtet. Aufgrund rechnerischer Ungenauigkeiten sollte jedoch nicht die exakte Gleichheit getestet, sondern diese innerhalb eines kleinen Bereichs als gültig anerkannt werden [Shi02]. Die Wahl eines Schwellwertes, der einen solchen Bereich festlegt, ist nicht unproblematisch. Es muss genau darauf geachtet werden, dass sich Polygone durch die Verschiebung



(a) x liegt im Schatten (b) x wird beleuchtet

Abbildung 2.3: Funktionsweise des Shadow Mappings (nach [FK03]).

des Tiefenwertes nicht selbst verschatten. Auf der anderen Seite kann es dazu kommen, dass gewünschte Schatten in der unmittelbaren Nähe des Schatten werfenden Polygons nicht als solche erkannt werden.

Eine andere Sichtweise auf das Shadow Mapping ist es, den Vergleich der Tiefenwerte durchzuführen, nachdem die Shadow Map mittels projektiver Texturierung von der Lichtquelle auf die Szene projiziert wurde [FK03]. Dies bildet eine anschauliche Erklärung für die Entstehung der typischen Artefakte des Shadow Mapping, auf die nun eingegangen wird.

2.2.2 Darstellungsprobleme durch Aliasing-Effekte

Wie es für bildbasierte Verfahren typisch ist, hängt die Ausgabequalität des Shadow Mappings von der Auflösung der Depth Map ab. Ist diese zu klein, wird mehreren darzustellenden Punkten der selbe Tiefenwert zugeordnet. Bei der Darstellung des Schattens führt dies, wie in Abbildung 2.4(a) dargestellt, zu gezackten Kanten – ein Effekt der häufig als *Jagged Shadow Boundaries* oder *Aliasing* bezeichnet wird. Anhand der in Abbildung 2.4(b) gezeigten Depth Map aus Sicht der Lichtquelle erklärt sich die Entstehung. Je näher sich die Kamera auf den Schattenbereich zu bewegt, um so mehr Pixel müssen hinsichtlich ihrer Tiefenwerte verglichen werden. Gleichzeitig müsste aber auch die Auflösung der Depth Map steigen um für alle Pixel aus Sicht der Kamera korrekte Vergleichswerte zu liefern. Diese ist jedoch endlich und so werden die Tiefenwerte vieler von der Kamera aus sichtbarer Pixel mit wenigen Repräsentanten aus Sicht der Lichtquelle verglichen.

Dem beschriebenen Problem kann in erster Linie durch die Wahl einer höheren Textur-Auflösung entgegen gewirkt werden. Dies erhöht jedoch die Speicheranforderung und verringert die Performanz der Anwendung. Eine Alternative stellt der Einsatz von Filtern dar. So werden bei dem weit verbreiteten *Percentage Closer Fil-*

(a) Aliasing-Effekt (b) Depth Map

Abbildung 2.4: Gezackte Schattenkanten durch niedrig aufgelöste Depth Map.

tering [RSC87] gleich mehrere Tiefenvergleiche pro Bildschirm-Pixel durchgeführt. Für wenige Samples wird dieser Prozess schon zu geringen Kosten von der modernen Graphikhardware realisiert. Moderne Varianten des Shadow Mapping, wie *Perspective Shadow Maps* [SD02] oder *Parallel-Split Shadow Maps* [ZHXL06], bemühen sich dagegen gezielt das Problem der unterschiedlichen Diskretisierung des Raumes auf der Bildebene von Kamera und Lichtquelle zu beheben.

2.2.3 Einschränkungen durch Punktlichtquellen

Für eine echtzeitfähige Beleuchtung in der Computergraphik werden neben gerichteten Lichtquellen häufig nur Punktlichtquellen zugelassen. Da diese Lichtquellen keine Ausdehnung haben, entstehen harte Schattenkanten und jeder darzustellende Punkt eines Objekts liegt entweder komplett im Schatten oder er wird normal beleuchtet. Ohne die Unterstützung von indirektionalem Licht oder die geschickte Platzierung mehrerer Punktlichtquellen im Raum lassen sich innerhalb des Schattenwurfs keine Formen erkennen. Häufig wird der Schattenbereich daher normal beleuchtet und anschließend nur leicht abgedunkelt.

In der Realität existieren keine unendlich kleinen Lichtquellen und die menschliche Wahrnehmung ist daran gewöhnt, dass mit zunehmender Entfernung zwischen einem lichtverdeckenden Objekt und der Fläche, auf der sich der Schatten abbildet, der Umriss des Schattens immer weicher wird. Die robuste Generierung weicher Schattenkanten zu annehmbaren Darstellungsraten ist immer noch ein aktives Forschungsfeld – Hasenfratz et al. [HLHS03] geben einen umfassenden, wenn auch nicht mehr ganz aktuellen Überblick über die zahlreichen Veröffentlichungen. Um interaktive Darstellungsraten zu gewährleisten verzichten diese Verfahren jedoch auf physikalisch korrekte Schatten.

(a) Ohne ambiente Farbe (b) Mit ambienter Farbe und Schatten

Abbildung 2.5: Lokale Beleuchtung durch eine Punktlichtquelle.

2.3 Zusammenfassung

Ein konstanter ambienter Term wird in den klassischen lokalen Beleuchtungsmodellen zur Simulation zweier Effekte eingesetzt. Zum einen bietet er die einzige Möglichkeit Bereiche, die ansonsten völlig unbeleuchtet wären, mit einer Farbe auf dem Bildschirm auszugeben. Dies entspricht einer äußerst groben Nachbildung indirekter Beleuchtung. Zum anderen ermöglicht nur der ambiente Term die Beleuchtung durch Umgebungslicht, da klassische Beleuchtungsmodelle explizit nur drei Typen von Lichtquellen unterstützen.

Abbildung 2.5 zeigt die Auswirkungen der oben genannten Probleme lokaler Beleuchtungsmodelle. Ohne den Einbezug eines ambienten Terms in die Beleuchtung werden manche Stellen der Geometrie komplett schwarz dargestellt. Die Verrechnung einer ambienten Farbe resultiert zwar in einem helleren Ergebnisbild, doch in den unbeleuchteten Bereichen sind dennoch keine geometrischen Details zu erkennen. Die Berücksichtigung von Schatten sorgt dafür, dass die Entfernung und relative Position des Modells zur Bodenebene deutlich wird, doch für jeden Punkt gilt, dass er entweder vollständig im Schatten liegt oder gar nicht.

3 Ambient Occlusion

Der Begriff *Ambient Occlusion*[1] steht für eine Reihe von Techniken, die durch eine genauere Berechnung des ambienten Terms deutlich realistischere Bilder erzeugen, als dies mit solchen Beleuchtungsmodellen, die üblicherweise für ein Rendering in interaktiven Frameraten eingesetzt werden, möglich ist. Konkret wird dies dadurch erreicht, dass unabhängig von der eigentlichen Beleuchtung ermittelt wird, wie stark sich die Elemente einer Szene gegenseitig verdecken und Umgebungslicht, das gleichmäßig aus allen Richtungen kommt, abschirmen.

Populär geworden ist Ambient Occlusion durch die Filmindustrie, wo es in manchen Situationen als lohnende Alternative zur aufwändigen Berechnung globaler Beleuchtung angesehen wird [Lan02, Chr03]. Für ein möglichst genaues Ergebnis bei der Verdeckungsberechnung wird in diesen Produktionen meist Ray Tracing eingesetzt [CFLB06].

Mit der steigenden Leistungsfähigkeit moderner Graphikhardware ist Ambient Occlusion auch für die Echtzeitgraphik zunehmend interessanter geworden. Um akzeptable Darstellungsraten zu erreichen, hat man sich dort in der Regel von einer Realisierung durch Ray Tracing abgekehrt und Ideen entwickelt, wie sich speziell die Fähigkeiten der Graphikprozessoren zur Berechnung von Ambient Occlusion ausnutzen lassen. Die Herausforderung bei der Entwicklung solcher Verfahren besteht darin, die Gratwanderung zwischen geringer Rechenzeit, moderatem Speicherbedarf und guter Darstellungsqualität zu meistern.

3.1 Motivation

Die Darstellung von Schatten bei einer Beleuchtung durch Punktlichtquellen erleichtert die Wahrnehmung der Anordnung von Objekten im Raum enorm. Harte Schattenkanten wirken jedoch unnatürlich auf den Betrachter, insbesondere wenn die virtuelle Szene unter freiem Himmel aufgebaut ist. Sowohl durch die Evolution als auch durch die Erfahrung jedes Einzelnen bedingt, hat sich die menschliche Wahrnehmung an die Beleuchtung durch Sonne und Himmel gewöhnt und lässt sich daher in einer computergenerierten Simulation der Realität nicht problemlos täuschen. Insbesondere an einem vollständig bewölkten Tag findet die Beleuchtung aus allen

[1] Ambient Occlusion wird manchmal mit „Umgebungsverdeckung“ oder „Selbstverschattung“ ins Deutsche übersetzt. Da die Verschattung eines Punktes prinzipiell sowohl von anderen Objekten als auch vom Objekt selbst verursacht werden kann, wird in dieser Arbeit der allgemeiner gehaltene, englische Begriff benutzt. In den Fällen, in denen eine Unterscheidung nötig ist, wird, analog zu manchen Veröffentlichungen [KL05, MMAH05], im ersten Fall der Begriff *inter-object Occlusion* und im zweiten Fall der Begriff *intra-object Occlusion* verwendet.

Richtungen der Hemisphäre annähernd gleichmäßig statt. Die Simulation dieser Beleuchtungssituation unter der ausschließlichen Berücksichtigung vollständig diffuser Materialien ist das Ziel von Ambient Occlusion. Da die Berechnung von Schatten durch Licht, das aus allen Richtungen kommt, den Schwerpunkt bildet, lassen sich die dem Ambient Occlusion zugeordneten Verfahren auch gleichzeitig als Vorgehensweisen zur Erzeugung weicher Schattenkanten kategorisieren. Dabei wird jedoch nicht die Fläche einer Lichtquelle mit endlichen Ausmaßen diskretisiert, sondern der obere Halbraum eines Punktes durch Auswahl repräsentativer Lichteinfallsrichtungen.

3.1.1 Unzulänglichkeit lokaler Beleuchtungsmodelle

In Abschnitt 2.1.3 wurde auf das Problem des konstanten ambienten Terms lokaler Beleuchtungsmodelle, der eine indirekte Beleuchtung approximieren soll, eingegangen. Eine Fläche, die nicht direkt beleuchtet wird, wird zwar nicht zwangsläufig schwarz dargestellt, aber in einer anderen konstanten Farbe – unabhängig vom Ausmaß ihrer Verdeckung, ihrer Position, oder ihrer Orientierung.

Ein erster Ansatz mit der Absicht den ambienten Term adaptiv zu gestalten geht auf Castro et al. [CNS00] zurück. Sie schlagen vor, jedes Polygon anhand seiner Normalen einer Klasse zuzuordnen, welche die ambiente Farbe festlegt. Die Verschattung bleibt bei diesem Vorgehen zu Gunsten einer hohen Performanz jedoch unberücksichtigt.

Ambient Occlusion behebt das Problem des konstanten ambienten Terms dagegen durch eine Verrechnung mit einem Skalierungsfaktor, der ein Maß dafür liefert, wie viel Umgebungslicht einen darzustellenden Punkt tatsächlich erreicht. Konkave Stellen einer Geometrie werden durch die Berücksichtigung von Selbstverschattung dunkler dargestellt als konvexe Bereiche. Zwei räumlich benachbarte Objekte schirmen gegenseitig das einfallende Umgebungslicht ab, sodass die sich gegenüberliegenden Flächen dunkler ausgegeben werden als die nach außen zeigenden Flächen. Aus Effizienzgründen berücksichtigt Ambient Occlusion in der Regel nur diffuse Materialien, sodass der Blickpunkt des Betrachters nicht in die Beleuchtung eingerechnet werden muss. Die Beleuchtung durch eine diffuse Lichtquelle, die gleichmäßig über den Raum aufgespannt ist und in alle Richtungen eine uniforme Lichtausstrahlung besitzt, sorgt für realistisch wirkende, weiche Schattenkanten.

Abbildung 3.1 zeigt das Ergebnis einer Beleuchtung durch Ambient Occlusion anhand von zwei Bildern, die mit dem in dieser Arbeit vorgestellten Ansatz erzeugt wurden. Ein Großteil des dargestellten Modells liegt dabei in einem Halbschatten, der sich auch über die Bodenebene erstreckt. Dadurch erscheint das Modell insgesamt gleichmäßiger ausgeleuchtet als in Abbildung 2.5. Lediglich die Stellen, an denen besonders wenig Licht einfällt (beispielsweise die Falten) werden beinahe schwarz dargestellt.

(a) (b)

Abbildung 3.1: Beleuchtung (a) alleine durch Ambient Occlusion und (b) in Kombination mit Texturen.

3.1.2 Psychophysikalische Befunde

In einer Reihe von Experimenten zeigen Langer und Bülthoff [LB99, LB00], dass ein Beleuchtungsmodell auf Grundlage von Ambient Occlusion, welches der diffusen Beleuchtung an einem bewölkten Tag entspricht, dem klassischen Lambert Shading durch eine Punktlichtquelle überlegen ist. Ihre Ergebnisse weisen darauf hin, dass dem visuellen System des Menschen bei der Erkennung von Formen durch Beleuchtung („*shape-from-shading*“) mehrere Modelle zugrunde liegen. Unter anderem werden dunkle Stellen als tiefer liegend wahrgenommen („*dark means deep*“). Diese unterbewusste Folgerung mag sich an einem durch Ambient Occlusion erzeugten Bild nachvollziehen lassen (vgl. Abbildung 3.1), bei der Beleuchtung durch eine Punktlichtquelle werden jedoch lediglich die dem Licht abgewandten Seiten dunkel dargestellt (vgl. Abbildung 2.5). Da diese Stellen allerdings nicht unmittelbar als weiter entfernt interpretiert werden dürfen, widerspricht ein solches Beleuchtungsmodell der menschlichen Intuition.

Kozlowski und Kautz [KK07] untersuchen in einer Studie den Einsatz verschiedener Verfahren zur Verschattungsberechnung in Kombination mit spiegelnden und glänzenden Materialien. Bei diesen Materialien sind Schatten, analog zum Glanzlicht, am ehesten in Richtung der Spiegelung sichtbar. Aufgrund der rein geometrischen Natur von Ambient Occlusion werden die Ergebnisbilder in diesem Fall in der Regel als weniger realistisch wahrgenommen, als dies mit Verfahren der Fall ist, die zusätzlich den Blickpunkt des Betrachters berücksichtigen. Die Ergebnisse von Kozlowski und Kautz [KK07] untermauern, dass Ambient Occlusion nur in Kombination mit diffusen Materialien eingesetzt werden sollte.

3.2 Zusammenhang mit anderen Verfahren zur Beleuchtung

Bei Ambient Occlusion wird jeder darzustellende Punkt einer Szene prinzipiell aus allen Richtungen des vorderen Halbraums beleuchtet. Dabei wird für jede Richtung

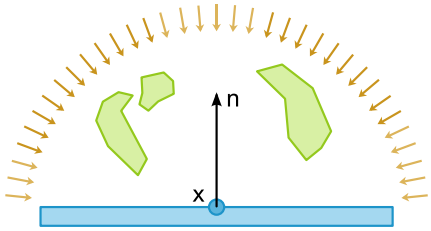


Abbildung 3.2: Messung des unverdeckten Anteils der Hemisphäre zur Berechnung von Ambient Occlusion.

ermittelt, ob es Geometrie gibt, die das Licht der Hemisphäre daran hindert den Punkt zu erreichen (einen so genannten *Occluder*). Abbildung 3.2 illustriert dies. Die mathematische Repräsentation des Problems, deren Auswertung ein Maß für die Beleuchtung eines Punktes unter Berücksichtigung der Verdeckung durch umliegende Geometrie der Szene liefert, lautet:

$$AO(\mathbf{x}, \mathbf{n}) = \frac{1}{\pi} \int_{\mathbf{\Omega}} V(\mathbf{x}, \omega_\mathrm{i}) \max(0, \mathbf{n} \cdot \omega_\mathrm{i}) d\omega_\mathrm{i} \ . \tag{3.1}$$

In der obigen Gleichung steht $\mathbf{x}$ für die Position und $\mathbf{n}$ für die Normale des Flächenelements, für welches die Verdeckung berechnet werden soll. Es wird über den gesamten vorderen Halbraum integriert und für jede Richtung ω_i die Sichtbarkeitsfunktion $V(\mathbf{x}, \omega_\mathrm{i})$ ausgewertet. Diese liefert den Wert 1, wenn keine Geometrie das aus dieser Richtung einfallende Licht daran hindert den Punkt zu beleuchten und nimmt ansonsten den Wert 0 an.[2] Durch eine Gewichtung mit dem Kosinus zwischen Normale $\mathbf{n}$ und Richtung ω_i tragen Occluder, die sich näher in Richtung der Normalen befinden stärker zum Ergebnis bei als solche, die näher am Horizont des Halbraums liegen. Dies entspricht der Reflexion diffuser Materialien nach dem Lambertschen Gesetz. Der Normalisierungsfaktor $1/\pi$ sorgt dafür, dass das Ergebnis der Berechnung in einem Bereich zwischen 0 und 1 liegt.

Bei der oben aufgeführten Gleichung handelt es sich um eine Abwandlung des so genannten *Obscurance Model* von Zhukov et al. [ZIK98]. Statt der binären Visibilitätsfunktion enthält dieses Modell einen Skalierungsfaktor, der sich umgekehrt proportional zur Distanz zwischen verdeckter Fläche und Occluder verhält. Das Obscurance Model wurde für einen Einsatz in *Game Engines* weiterentwickelt [IKSZ03] und in diesem Zusammenhang auch um *Color Bleeding* erweitert [MSC03]. Daneben wurde es unter der Bezeichnung *Vicinity Shading* für ein Rendering volumetrischer Daten angepasst [Ste03]. Mittlerweile scheint dieses Modell allerdings wesentlich seltener Anwendung (und Erwähnung) zu finden als Ambient Occlusion.

[2] Dass sich $V(\mathbf{x}, \omega_\mathrm{i})$ aus der Sichtbarkeit zwischen $\mathbf{x}$ und der Lichtquelle (und nicht zwischen $\mathbf{x}$ und einem Occluder) errechnet, führt dazu, dass $AO(\mathbf{x}, \mathbf{n})$ ein Maß für den Anteil der Hemisphäre liefert, der $\mathbf{x}$ beleuchtet. Wie groß die Verdeckung von $\mathbf{x}$ ist berechnet sich daher aus $1 - AO(\mathbf{x}, \mathbf{n})$. Dies mag widersprüchlich zur Bezeichnung *Ambient Occlusion* erscheinen, da nicht die *Verdeckung* sondern die *Beleuchtung* unter Berücksichtigung der Verdeckung ermittelt wird, richtet sich jedoch nach der konventionellen Definition von $V(\mathbf{x}, \omega_\mathrm{i})$.

3.2.1 Integration in lokale Beleuchtungsmodelle

Der durch Gleichung 3.1 errechnete Wert für Ambient Occlusion kann auf unterschiedliche Weise in das zum Einsatz kommende Beleuchtungsmodell integriert werden. Die ursprüngliche und namensgebende Idee besteht darin, lediglich die konstante ambiente Farbe in Form einer komponentenweisen Multiplikation mit dem skalaren Visibilitätsterm zu gewichten [DBB06]. Dies lässt sich einfach an dem in Abschnitt 2.1.3 vorgestellten Beleuchtungsmodell zeigen:

$$c = c_\mathrm{r}(AO(\mathbf{x},\mathbf{n})c_\mathrm{a} + c_\mathrm{l}\max(0,\mathbf{n}\cdot\mathbf{l})) + c_\mathrm{l}c_\mathrm{p}\max(0,\mathbf{e}\cdot\mathbf{r})^p\ . \tag{3.2}$$

Allerdings ist die Kombination von Ambient Occlusion und spiegelnden Materialen nicht sehr plausibel [KK07]. Ein weiteres Problem bei dieser Herangehensweise besteht darin, dass die tatsächliche Beleuchtung nach wie vor durch Punktlichtquellen stattfindet, die separat gesetzt werden. Der einzige Unterschied zu Gleichung 2.2 auf Seite 6 besteht darin, dass das ambiente Licht, das per Definition von überall kommt, nicht mehr überall in gleichem Maße ankommt.

Auch wenn ein solches Vorgehen von der namensgebenden Idee abweicht, so bietet es sich prinzipiell an auf zusätzliche Lichtquellen zu verzichten und den berechneten Visibilitätsterm direkt mit der diffusen Materialfarbe zu verrechnen. Auf diese Weise wurde Abbildung 3.1(b) erzeugt.

Statt der manuellen Positionierung der in Gleichung 3.2 berücksichtigten Lichtquellen ist es ebenso gut möglich die Beleuchtung mit Hilfe von Environment Maps bildbasiert zu steuern. Um dies auf effiziente Weise mit nur einem Texturzugriff durchführen zu können, wird bei der Berechnung von Ambient Occlusion für einen Punkt häufig eine so genannte *Bent Normal* [Lan02] ermittelt. Dabei handelt es sich um einen Vektor, der in die Richtung des meisten einfallenden Lichtes zeigen soll und der durch Mittelung aller untersuchten, unverdeckten Richtungen berechnet wird. Der Fehler, der entstehen kann, wenn die Bent Normal in Richtung eines Occluders zeigt, wird in der Regel toleriert [PG04]. Da Ambient Occlusion auf diffuse Materialien zugeschnitten ist, bietet es sich an unendlich weit entferntes Umgebungslicht durch Spherical Harmonics zu repräsentieren. Ein auf solche Weise kombiniertes Vorgehen findet sich etwa bei Biedermann [Bie04].

3.2.2 Abgrenzung gegenüber der Simulation globaler Beleuchtung

Ambient Occlusion erweitert klassische Beleuchtungsmodelle durch die Simulation einer homogenen Beleuchtung aus unendlicher Entfernung. Für die Erzeugung photorealistischer Bilder ist zusätzlich die Berücksichtigung indirekter Beleuchtung und unterschiedlicher Materialien unumgänglich. Die Integration über die Hemisphäre in Gleichung 3.1 auf der vorherigen Seite erinnert nicht ohne Grund an die Rendering Equation, die in Abschnitt 2.1.1 vorgestellt wurde. Um die Annahmen, die im Zuge von Ambient Occlusion gemacht werden, nachvollziehen zu können, soll die zugrunde liegende Gleichung im Folgenden aus der Rendering Equation hergeleitet werden. Der Anschaulichkeit halber wird zunächst noch einmal die Rendering Equation abgebildet:

$$L_{\mathrm{o}}(dA_e, d\omega_{\mathrm{o}}) = L_{\mathrm{e}}(dA_{\mathrm{e}}, d\omega_{\mathrm{o}}) + \int_{\mathbf{\Omega}} f_{\mathrm{r}}(dA_{\mathrm{e}}, d\omega_{\mathrm{i}}, d\omega_{\mathrm{o}}) \cdot L_{\mathrm{i}}(dA_{\mathrm{e}}, d\omega_{\mathrm{i}}) \cdot \cos\theta_{\mathrm{i}} \cdot d\omega_{\mathrm{i}} \ .$$

Ambient Occlusion beschränkt sich auf direkte Beleuchtung, welche aus allen Richtungen des oberen Halbraums kommt, in denen sich kein Occluder befindet. Um dies auszudrücken, wird die einfallende Leuchtdichte $L_{\mathrm{i}}(dA_{\mathrm{e}}, d\omega_{\mathrm{i}})$ durch $L_{\mathrm{i}_{\mathrm{direkt}}}(dA_{\mathrm{e}}, d\omega_{\mathrm{i}})$ ersetzt. Die binäre Visibilitätsfunktion $V(dA_{\mathrm{e}}, \omega_{\mathrm{i}})$ liefert 0, wenn sich in Richtung ω_{i} ein Occluder befindet und ansonsten 1. Da keine indirekte Beleuchtung berücksichtigt wird, wird im Folgenden auch die emittierte Leuchtdichte vernachlässigt:

$$L_{\mathrm{o}}(dA_{\mathrm{e}}, d\omega_{\mathrm{o}}) \approx \int_{\mathbf{\Omega}} f_{\mathrm{r}}(dA_{\mathrm{e}}, d\omega_{\mathrm{i}}, d\omega_{\mathrm{o}}) \cdot V(dA_{\mathrm{e}}, d\omega_{\mathrm{i}}) \cdot L_{\mathrm{i}_{\mathrm{direkt}}}(dA_{\mathrm{e}}, d\omega_{\mathrm{i}}) \cdot \cos\theta_{\mathrm{i}} \cdot d\omega_{\mathrm{i}} \ .$$

Außerdem beschränkt sich Ambient Occlusion auf diffuse Lambert-Reflexionen. In diesem Fall gilt, dass die BRDF nicht winkelabhängig ist und damit vor das Integral gezogen werden kann. Die BRDF diffuser Lambert-Reflexionen lässt sich durch den Reflexionsgrad $\rho = {}^{L_{\mathrm{o}}\pi}/_{E}$ ausdrücken, wobei E die Beleuchtungsstärke bezeichnet. Mit $f_{\mathrm{r}_{\mathrm{diffus}}} = {}^{L_{\mathrm{o}}}/_{E}$ ergibt sich $\rho = f_{\mathrm{r}_{\mathrm{diffus}}}\pi$ und $f_{\mathrm{r}_{\mathrm{diffus}}} = {}^{\rho}/_{\pi}$; damit gilt:

$$L_{\mathrm{o}}(dA_{\mathrm{e}}) \approx \rho(dA_{\mathrm{e}}) \frac{1}{\pi} \int_{\mathbf{\Omega}} V(dA_{\mathrm{e}}, d\omega_{\mathrm{i}}) \cdot L_{\mathrm{i}_{\mathrm{direkt}}}(dA_{\mathrm{e}}, d\omega_{\mathrm{i}}) \cdot \cos\theta_{\mathrm{i}} \cdot d\omega_{\mathrm{i}} \ .$$

Die einfallende Leuchtdichte wird zur Berechnung von Ambient Occlusion als konstant über die gesamte Hemisphäre angenommen. Dies motiviert die Analogie zur Beleuchtung an einem vollständig bewölkten Tag. Damit kann auch die Leuchtdichte vor das Integral geschoben werden:

$$L_{\mathrm{o}}(dA_{\mathrm{e}}) \approx L_{\mathrm{i}_{\mathrm{direkt}}}(dA_{\mathrm{e}}) \cdot \rho(dA_{\mathrm{e}}) \cdot \frac{1}{\pi} \int_{\mathbf{\Omega}} V(dA_{\mathrm{e}}, d\omega_{\mathrm{i}}) \cdot \cos\theta_{\mathrm{i}} \cdot d\omega_{\mathrm{i}} \ .$$

In der obigen Gleichung ist die reflektierte Leuchtdichte proportional zur einfallenden Leuchtdichte. Die Auswertung des Integrals liefert nun einen skalaren Wert mit dem das Produkt von einfallender Leuchtdichte und Reflexionsgrad gewichtet wird. Bei diesem Term handelt es sich um Ambient Occlusion. Alle weiteren Unterschiede zu Gleichung 3.1 sind rein kosmetischer Natur. Das infinitesimale Flächenelement dA_{e} wird durch eine Position $\mathbf{x}$ und eine Normale $\mathbf{n}$ repräsentiert und der Kosinus des Winkels zwischen Lichteinfallsrichtung ω_{i} und der Normalen wird durch das Skalarprodukt der beiden Vektoren berechnet:

$$AO(\mathbf{x}, \mathbf{n}) = \frac{1}{\pi} \int_{\mathbf{\Omega}} V(\mathbf{x}, \omega_{\mathrm{i}}) \max(0, \mathbf{n} \cdot \omega_{\mathrm{i}}) d\omega_{\mathrm{i}} \ .$$

Der zentrale Unterschied zwischen der Rendering Equation und Ambient Occlusion besteht neben der Beschränkung auf diffuse Reflexionen darin, dass bei Ambient Occlusion aus allen Richtungen, in denen sich *keine* Geometrie befindet, Licht mit konstanter Leuchtdichte eintrifft. Die Rendering Equation berücksichtigt dagegen beliebige Materialien, jede Art von Lichtquellen und insbesondere indirektes Licht, das über mehrere Flächen hinweg reflektiert wird. Die Möglichkeit, dass die Geometrie einer Szene Licht erzeugt oder weiterleitet, wird alleine durch Ambient Occlusion nicht berücksichtigt. Da eine Verdeckung von Occludern jedoch wesentlich schneller zu berechnen ist als die Simulation globaler Beleuchtung durch Radiosity, Photon Mapping oder Final Gathering und der visuelle Unterschied je nach Szene kaum auszumachen ist, wird Ambient Occlusion als lohnenswerte Alternative angesehen. Sowohl in Anwendungen mit harten Echtzeitanforderungen, etwa zur technischen, geographischen oder naturwissenschaftlichen Visualisierung [SBB*06, DBL07, TCM06] oder in aktuellen Computerspielen [Mit07], als auch in der Produktion von Filmen mit computergeneriertem Inhalt, wie „Cars“ (Disney/Pixar) [CFLB06] oder „Pearl Harbor“ (Lucas Digital Ltd.) [Lan02], wurde Ambient Occlusion bereits erfolgreich eingesetzt und wird von den beiden populären Renderern *Mental Ray* [Men05] und *Photorealistic RenderMan* [Chr03] unterstützt.

3.3 Vorausgegangene Arbeiten

In diesem Abschnitt werden einige Verfahren zur Berechnung von Ambient Occlusion vorgestellt und bewertet. Da sich die Vorgehensweisen zum Teil deutlich unterscheiden wird ein System zur Einordnung der Verfahren herangezogen. Obwohl nur ein Wert zwischen 0 und 1 für jeden darzustellenden Punkt gesucht wird, ist die Berechnung aufgrund der Tatsache, dass jedes Element einer Szene als möglicher Occluder angesehen werden muss, relativ aufwändig. Jede der Methoden, die im Folgenden vorgestellt werden, nutzt daher moderne Graphikhardware aufgrund ihrer Performanz. Darüber hinaus werden in der Regel verschiedene Vereinfachungen und Annahmen vorgenommen, sodass das Ergebnis im Vergleich zu einer, etwa durch Ray Tracing durchgeführten Auswertung von Gleichung 3.1 auf Seite 16 in unterschiedlich großem Maße von der Referenzlösung abweicht. Da es sich bei Ambient Occlusion selbst bereits um eine Vergröberung realistischer Beleuchtungssimulationen handelt, ist fragwürdig inwiefern solche Ansätze hinnehmbar sind. Neben einer Untersuchung von Performanz und Speicheranforderungen sollen die im Folgenden vorzustellenden Verfahren daher insbesondere hinsichtlich ihrer Darstellungsqualität beurteilt und verglichen werden.

Vor allem in zeitkritischen Filmproduktionen oder Anwendungen mit harten Echtzeitanforderungen wird Ambient Occlusion oft in einem Prozess namens *Baking* von 3D-Software vorberechnet oder von Künstlern gezeichnet und in Texturen oder als zusätzliches Vertex-Attribut gespeichert [Chr03, Men05, Sou07]. Dieses Vorgehen berücksichtigt jedoch nicht die korrekte Selbstverschattung verformbarer Geometrie oder die Verschattung zwischen sich frei im Raum bewegenden Objekten. Wie

solche vorberechneten Verdeckungswerte anhand einiger Referenzposen auf ein animiertes Modell abgebildet werden, sodass die Selbstverschattung möglichst genau angenähert wird, ist Gegenstand von Abschnitt 3.3.6. Darüber hinaus bilden im weiteren Verlauf dieser Arbeit Verfahren für eine dynamische Berechnung von Ambient Occlusion den Schwerpunkt.

Knecht [Kne07] ordnet einige der bislang vorgeschlagenen Ansätze zur Berechnung von Ambient Occlusion in vier Kategorien ein:[3]

Objektbasierte Methoden berechnen die Verdeckung mit Hilfe von Datenstrukturen, wie Texturen, welche die Sichtbarkeitsinformation für festgelegte Bereiche verwalten oder durch Texturen, die auf die Szene projiziert werden. Diese Verfahren stellen praktisch keine Anforderungen an die Geometrie der Modelle, aber die Vorberechnungszeit nimmt relativ viel Zeit in Anspruch. Die Geschwindigkeit zur Laufzeit hängt von der Anzahl sich gegenseitig verdeckender Objekte ab.

Vertexbasierte Methoden berechnen die Verdeckung für die einzelnen Eckpunkte eines Modells. Diese Verfahren benötigen relativ wenig Zeit zur Vorberechnung aber die Qualität des Ergebnisses hängt von der Triangulierung der eingesetzten Modelle ab. Die Performanz zur Laufzeit ist abhängig von der Komplexität der Modelle.

Bildbasierte Methoden berechnen die Verdeckung ausgehend von den Bildschirmpixeln. Dabei machen diese Verfahren von der Beobachtung Gebrauch, dass ein Occluder häufig auch auf der Bildebene in der Nähe des verdeckten Punktes anzutreffen ist. Es ist keine Vorberechnungszeit nötig und die Performanz ist unabhängig von der Komplexität der Geometrie. Die Geschwindigkeit zur Laufzeit hängt vielmehr von der Auflösung des Fensters ab, in welchem die Anwendung läuft.

Methoden für animierte Charaktere befassen sich mit einer Abbildung von Animationsparametern auf Verdeckungswerte. Dazu wird Ambient Occlusion bei einigen Referenzposen eines Modells für jeden Vertex beispielsweise mit Ray Tracing vorberechnet. Diese Methoden sind auf intra-object Occlusion beschränkt und werden daher hier nur kurz angerissen.

Eine andere, häufig vorzufindende Einteilung unterscheidet ein *inside-out* von einem *outside-in* Vorgehen. Im ersten Fall wird die Verdeckung von den darzustellenden Flächen aus untersucht. Wenn die Geschwindigkeit nicht ausschlaggebend ist, kann dazu beispielsweise Ray Tracing eingesetzt werden. Im zweiten Fall wird das Problem dagegen aus Sicht der Lichtquelle angegangen. Dabei wird typischerweise die Szene aus verschiedenen Richtungen gerendert und die Sichtbarkeit zusammengerechnet wird.

[3] Mit dem in Kürze erscheinenden, ersten Ansatz zur gezielten Berechnung von Ambient Occlusion im Kontext von Volumenrendering [RMD*08] ließe sich mittlerweile noch eine fünfte Kategorie hinzufügen. Aufgrund der geringen Relevanz für diese Arbeit wird hier jedoch darauf verzichtet.

Verfahren	Typ	Sicht	Qualität	Perform.	Speicher	Dyn.
[KL05]	Objekt	inside-out	–	+	+	–
[MMAH05]	Objekt	inside-out	–	++	–	–
[CL07]	Objekt	inside-out	+	–	– –	–
[PG04]	Objekt	outside-in	+	– –	++	–
[Bun05]/[HJ07]	Vertex	inside-out	–/+	+/–	+	+
[HPAD06]	Vertex	inside-out	– –	++	++	+
[SSZK04]	Vertex	outside-in	+	+	–	+
[GSF07]	Vertex	outside-in	–	+	– –	–
[SA07]	Bild	inside-out	–	–	+	+
[Mit07]	Bild	inside-out	– –	++	++	+

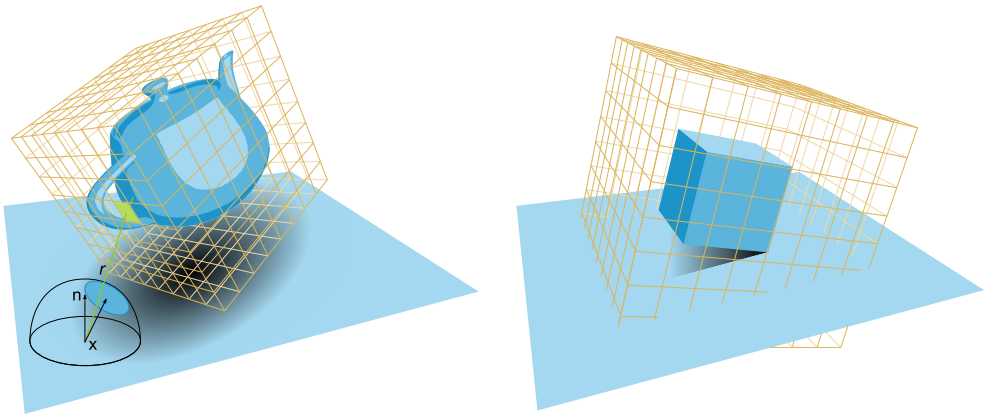
Tabelle 3.1: Verfahren zur Berechnung von Ambient Occlusion im Vergleich hinsichtlich Darstellungsqualität, Performanz, Speicherbedarf und Unterstützung animierter Objekte (Dyn.).

Aufgrund der Vielzahl von Veröffentlichungen, die in letzter Zeit zum Thema Ambient Occlusion erschienen sind, soll im Verlauf dieser Arbeit für eine möglichst genaue Einteilung eine Kombination beider Systeme zur Kategorisierung herangezogen werden.

Tabelle 3.1 gibt einen Überblick über die Verfahren zur Berechnung von Ambient Occlusion, die im Folgenden vorgestellt werden. Sämtliche Vorgehensweisen werden in der Tabelle grob hinsichtlich der Kriterien Darstellungsqualität, Performanz zur Laufzeit und Speicheranforderungen eingeteilt. Die Spalte „Dynamik“ erfasst, ob ein Verfahren auf Starrkörper (engl. *rigid bodies*) beschränkt ist (–) oder auch verformbare Geometrie unterstützt (+). Die Einteilung basiert auf expliziten Angaben, die in den entsprechenden Veröffentlichungen gemacht wurden, sowie auf eigenen Überlegungen zu Vor- und Nachteilen der Verfahren aufgrund des beschriebenen Vorgehens.

3.3.1 Objektbasierte inside-out Methoden

Kontkanen und Laine [KL05] berechnen Ambient Occlusion zwischen zwei Starrkörpern, indem sie einen Occluder durch eine kreisförmige Fläche auf der oberen Halbkugel der Empfängerfläche annähern (vgl. Abbildung 3.3(a)). Die verdeckende Fläche ist durch eine Richtung (die durchschnittliche Richtung der Verdeckung) und einen Raumwinkel definiert und ihre Größe entspricht dem Raumwinkel, der durch den Occluder aufgespannt wird. Richtung und Raumwinkel werden in einem Vorberechnungsschritt durch Ray Tracing oder Scanline-Rendering von Bildern in allen sechs Raumrichtungen ermittelt und anschließend in Form von radialen Funktionen in zwei *Cube Maps*, die das verdeckende Objekt umgeben, auf effiziente Weise gespeichert. Für jeden Occluder lässt sich damit für einen beliebigen Punkt im Raum ermitteln, wie stark er diesen Punkt verdeckt. Zur Laufzeit wird Ambient



(a) Cube Map (b) 3D-Textur

Abbildung 3.3: Ambient Occlusion unter Verwendung vorberechneter Texturen.

Occlusion unter Verwendung einer *Look-up Table* aus Raumwinkel und Richtung der Verdeckung berechnet. Liegt das Schatten empfangende Objekt innerhalb der konvexen Hülle des verdeckenden Objektes, kommt es zu Diskontinuitäten, die Kontkanen und Laine [KL05] durch Interpolation mit einem konstanten ambienten Term beheben. Mit dem Verfahren lässt sich darüber hinaus nur die Verdeckung durch einzelne Occluder berechnen. Um einen Wert für die gesamte Verschattung zu erhalten, werden die einzelnen Ergebnisse mit einer Wahrscheinlichkeit verrechnet, welche die Verteilung der Occluder im Raum approximiert. Das Ergebnis liegt dadurch zwischen den beiden unwahrscheinlichen Fällen, dass einerseits alle Occluder vom Empfängerobjekt aus gesehen in einer Reihe hintereinander liegen und, dass andererseits die Occluder so verteilt sind, dass sich ihr Schattenwurf auf der empfangenden Fläche nicht überschneidet.

Das Verfahren von Kontkanen und Laine [KL05] berechnet nur inter-object Occlusion und berücksichtigt keine Selbstverschattung. Durch die Annahme, dass die Silhouette der verdeckenden Objekte annähernd kreisrund ist und dadurch, dass die Verteilung der Occluder im Raum geschätzt wird, wirkt der Algorithmus nicht robust. Für weniger als 20 Objekte, die sich gegenseitig verdecken können, erreichen [KL05] eine interaktive Darstellungsgeschwindigkeit. Die Vorberechnungszeit liegt jedoch unter Verwendung einer Cube Map-Auflösung von 32×32 selbst bei einfacher Geometrie schon bei mehreren Minuten.

Malmer et al. [MMAH05] greifen die Idee vorberechnete Werte zur Ermittlung der Visibilität in einer Textur zu jedem potentiellen Occluder zu speichern auf. Sie verwenden eine dreidimensionale RGBA-Textur und verwalten in jedem Voxel einen vorberechneten Verdeckungswert sowie die durchschnittliche Richtung der Verdeckung. Zur Laufzeit wird vorab die gesamte Szene gerendert. In einem zweiten *Render Pass* werden zunächst Würfel mit deaktiviertem Tiefentest gerendert, die in Position, Orientierung und Größe mit den 3D-Texturen der Occluder korrespondie-

ren. Dies hat den Effekt, dass nur die Pixel, die daraufhin den Fragment Shader passieren, auf eine Verschattung hin untersucht werden müssen. Dazu wird die Position des Pixels in eine Voxelposition umgerechnet, sodass dort die vorberechnete Verdeckungsinformation ausgelesen und mit Nachbarwerten in geeigneter Weise interpoliert werden kann. Der Verschattungsgrad wird mit dem Kosinus des Winkels zwischen der Normalen des untersuchten Punktes und der durchschnittlichen Richtung der Verdeckung gewichtet und anschließend mit der Farbe des Pixels aus dem ersten Render Pass verrechnet. Die Verschattung von Geometrie kann nur berechnet werden, wenn diese innerhalb des Gitters liegt, das von der 3D-Textur des Occluders aufgespannt wird (vgl. Abbildung 3.3(b)). Daher ist es wichtig, die Ausmaße der Textur nicht zu klein zu wählen. Andererseits darf die Textur nicht zu groß sein, da ansonsten eine höhere Auflösung benötigt wird um Artefakte zu vermeiden, was wiederum mit einem kubischen Zuwachs an Speicherbedarf verbunden ist. Malmer et al. [MMAH05] stellen aus diesem Grund eine Formel vor, die zur Berechnung einer angemessenen Texturgröße anhand der *Bounding Box* des Occluder-Objekts herangezogen werden kann.

Das Verfahren von Malmer et al. [MMAH05] unterstützt Selbstverschattung und inter-object Occlusion. Die durchschnittliche Richtung der Verdeckung kann zwar mit Environment Maps kombiniert werden, sodass Licht aus dieser Richtung nicht zur Beleuchtung beiträgt, es handelt sich jedoch um eine grobe Schätzung, die das Verfahren mit einem Fehler behaftet. Ähnlich wie bei Kontkanen und Laine [KL05] kommt es bei der Verrechnung der Verdeckung durch mehrere Occluder zu Ungenauigkeiten, die sich bei Malmer et al. [MMAH05] in einem zu hellen Ergebnisbild im Vergleich zu einer durch Ray Tracing berechneten Lösung bemerkbar machen. Allerdings treten keine Fehler auf wenn sich ein Objekt innerhalb der konvexen Hülle des Occluders befindet und die Vorberechnungszeit sowie der rechnerische Aufwand zur Laufzeit sind geringer. Ein wesentlicher Nachteil ist jedoch der höhere Speicherbedarf.

Cadet und Lécussan [CL07] unterteilen die Szene adaptiv durch einen *Octree*, sodass jeder Knoten eine minimale Größe besitzt oder höchstens ein Polygon enthält. Die Bounding Box dieser minimalen Knoten bezeichnen sie als *Ambient Box* und in jedem der acht Eckpunkte dieser Box speichern sie in einem so genannten *Occlusion Buffer* die Verdeckung für verschiedene Raumrichtungen. Ein Occlusion Buffer ist ein Würfel, in dem für jede der sechs Seiten in einem 8×8 Gitter die Sichtbarkeitsinformationen verwaltet werden. Abbildung 3.4 verdeutlicht diese Struktur. Die Sichtbarkeit für eine Zelle des Gitters wird durch Ray Tracing vom Mittelpunkt des Occlusion Buffers durch einen zufälligen Punkt innerhalb der Zelle ermittelt. Für einen darzustellenden Punkt innerhalb der Ambient Box kann anhand seiner Normalen ermittelt werden, welche Occlusion Buffer ausgelesen und interpoliert werden müssen um Ambient Occlusion zu berechnen. Durch die adaptive Größe jeder Ambient Box treten Diskontinuitäten an benachbarten Punkten, die unterschiedlichen Boxen zugeteilt sind, auf. Aus diesem Grund werden auf der Bildebene benachbarte Ambient Boxes gesucht und die Visibilität eines Punktes durch Interpolation mit diesen ermittelt.

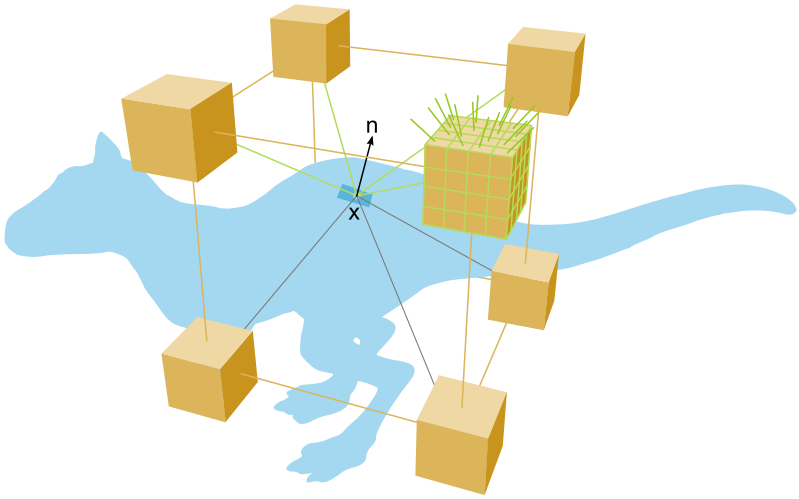


Abbildung 3.4: Verwaltung von Verdeckungsinformationen durch acht Occlusion Buffer.

Da das Vorgehen auf Ray Tracing basiert, ist die Performanz niedriger als bei den meisten GPU-basierten Ansätzen. Den Vorteil ihres Ansatzes sehen Cadet und Lécussan [CL07] allerdings darin, dass die Anzahl an Objekten innerhalb der Szene nicht die Geschwindigkeit beeinflusst und die Vorberechnungszeit in einem moderaten Rahmen liegt. Der deutlichste Nachteil dürfte darin bestehen, dass pro Ambient Box insgesamt 3072 Sichtbarkeitsinformationen verwaltet werden.

3.3.2 Objektbasierte outside-in Methoden

Einer der ersten Ansätze zur Berechnung von Ambient Occlusion durch Ausnutzung der Graphikhardware stammt von Pharr und Green [PG04]. Statt von einem darzustellenden Punkt aus Strahlen zu verschießen um die Verdeckung zu berechnen, betrachten sie das Problem von außen, indem sie eine große Anzahl von Lichtquellen auf einer Kugel, welche die Szene umgibt, verteilen. Eine Schleife durchläuft der Reihe nach alle Lichtquellen und startet jeweils zwei Render Passes. Im ersten Durchlauf wird eine Shadow Map aus Sicht der aktiven Lichtquelle erstellt und im zweiten Durchlauf wird die Szene normal gerendert, wobei nur die jeweilige Lichtquelle beleuchtet und einen Schatten wirft. Licht- und Materialparameter sind dabei so eingestellt, dass alle Bereiche, die direkt beleuchtet werden weiß dargestellt werden und alle übrigen schwarz (vgl. Abbildung 3.5 auf der nächsten Seite). Alle auf diese Art produzierten Bilder werden zu gleichen Teilen im *Accumulation Buffer* zusammengerechnet.

Akzeptable Ergebnisse lassen sich mit dem Verfahren von Pharr und Green [PG04] erst mit mehreren hundert Lichtquellen erzielen. Die Berechnung von Ambient Occlusion dauert dann mehrere Sekunden und muss vollständig neu vorgenommen werden sobald die Kamera oder ein Objekt der Szene transformiert wird.

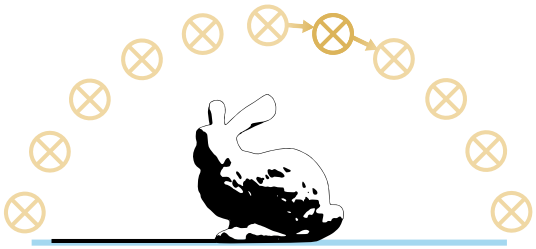


Abbildung 3.5: Ambient Occlusion durch Iteration über Lichtquellen.

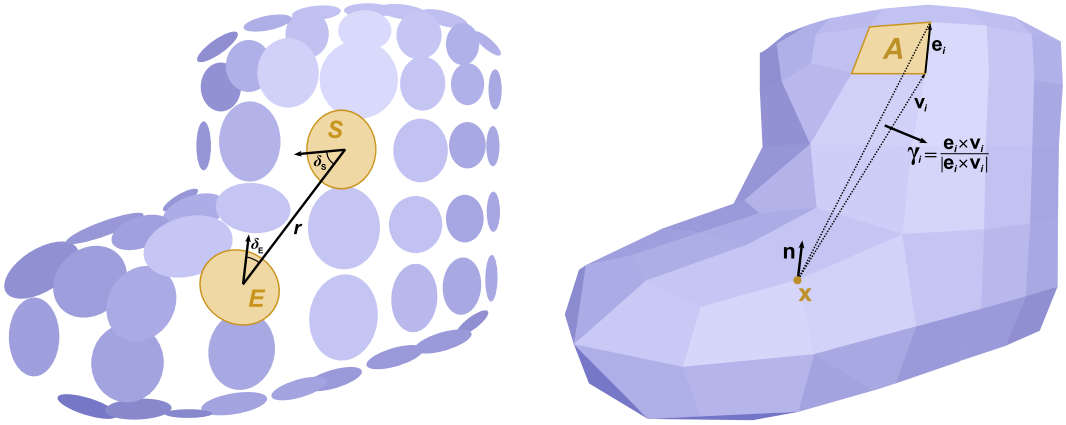
3.3.3 Vertexbasierte inside-out Methoden

Bunnell [Bun05] schlägt mit seinem Vorgehen vor, für jeden Eckpunkt eines Modells ein scheibenförmiges Element zu berechnen. Diese Elemente approximieren das Modell und vereinfachen die Verdeckungsberechnung. Jedes der Elemente besitzt dieselbe Position und Normale wie der entsprechende Eckpunkt und darüber hinaus eine Fläche, die sich aus einem Drittel der Summe der Flächen aller Dreiecke berechnet, zu denen der Eckpunkt gehört. Position, Normale und Fläche werden in Texturen gespeichert, damit im Fragment Program auf diese Werte zugegriffen werden kann, um die Verdeckung eines Empfänger-Elementes (E) durch ein Sender-Element (S) mit der folgenden Formel zu berechnen:

$$1 - \frac{r \cos \delta_{\mathrm{S}} \max(1, 4 \cos \delta_{\mathrm{E}})}{\sqrt{\frac{A_{\mathrm{S}}}{\pi} + r^2}} \,,$$

dabei steht A_{S} für die Fläche des Sender-Elements und δ_{S}, beziehungsweise δ_{E}, für den Winkel zwischen der jeweiligen Normalen eines Elements und der direkten Verbindung zwischen beiden Elementen (vgl. Abbildung 3.6(a)). Die gesamte Verdeckung eines Empfänger-Elements ergibt sich durch Summierung der Ergebnisse der oben abgebildeten Formel, in der alle übrigen Elemente einmal als Sender-Element eingesetzt werden. Dies resultiert in einer quadratischen Komplexität. Um den Aufwand auf $\mathcal{O}(n \log n)$ zu verringern, schlägt Bunnell [Bun05] eine hierarchische Organisation der Elemente vor, sodass die Berechnung der Visibilität bei großer Entfernung mit weniger Elementen, die dafür eine größere Fläche besitzen, durchgeführt werden kann. Da bei der Verdeckungsberechnung Elemente, die im Schatten liegen, wiederum Schatten werfen, sollte das Verfahren mindestens zweimal durchlaufen und die Ergebnisse gewichtet miteinander verrechnet werden. Die Approximation des Modells durch finite Elemente kann wie beim Radiosity-Verfahren genutzt werden, um eine indirekte Beleuchtung zu simulieren. Dazu ersetzt Bunnell die Berechnung der oben abgebildeten Formel im Fragment Program durch die zweite Formfaktorvereinfachung zwischen den Elementen E und S.

Hoberock und Jia [HJ07] decken zwei Probleme auf, die sich beim praktischen Einsatz von Bunnells [Bun05] Verfahren ergeben und liefern gleichzeitig Verbesserungsvorschläge, die zu einem robusteren Algorithmus führen sollen. Ein Schwach-



(a) Scheibenförmige Elemente (b) Punkt-zu-Polygon Formfaktor

Abbildung 3.6: Ambient Occlusion durch die Berechnung von Formfaktoren.

punkt der Originalimplementierung ist, dass die Verdeckung nur zwischen Eckpunkten berechnet wird. Insbesondere bei niedrig aufgelösten Modellen werden dadurch Schatten nicht erkannt oder es kommt zu Fehlern durch Interpolation. Bei einer dichteren Tesselierung des Modells werden die Schatten zwar genauer, aber es entstehen aufgrund der Diskretisierung durch die hierarchische Organisation der Elemente scheibenförmige Diskontinuitäten. Wird die Verdeckung dagegen zwischen Fragments statt Vertices berechnet, befreit dies das Verfahren zwar von Anforderungen an die Auflösung der Geometrie, aber eine zweite Art von Artefakten an Stellen mit sehr starker Krümmung, an denen sich ein darzustellender Punkt zu nah an einem der scheibenförmigen Elemente befindet, wird hervorgehoben. Zur Lösung des ersten Problems schlagen Hoberock und Jia [HJ07] eine Interpolation zwischen Vater- und Kindknoten in der Hierarchie der Elemente vor, sodass ein weicher Übergang zwischen der Verdeckung entsteht, die auf zwei unterschiedlichen Hierarchiestufen errechnet wird. Die zweite Klasse von Artefakten beheben sie, indem sie die Annäherung des Modells durch scheibenförmige Elemente an den Eckpunkten durch die Berechnung des Formfaktors zwischen einem Punkt und den tatsächlichen Polygonen mittels der Prismalösung ersetzen:

$$F_{xA} = \frac{1}{2\pi} \sum_i \mathbf{n} \cdot \gamma_i \ .$$

In dieser Gleichung ist $\mathbf{n}$ die Normale des Punktes $\mathbf{x}$ und γ_i der Einheitsvektor, welcher senkrecht auf der Ebene steht, die durch einen Vektor $\mathbf{v}_i$ und die Kante $\mathbf{e}_i$ des Polygons A aufgespannt wird (vgl. Abbildung 3.6(b)) – dabei wird i solange inkrementiert bis alle Kanten des Polygons in die Rechnung eingegangen sind. Die Berechnung des Formfaktors findet nur auf unterster Ebene der Hierarchie statt und erfordert zuvor das Clippen der Polygone an der Ebene, die durch den darzustellen-

den Punkt und orthogonal zu seiner Normalen verläuft. Da ihr Vorgehen generell zu dunkle Bilder liefert, schlagen Hoberock und Jia [HJ07] eine Abschwächung des Verdeckungsbeitrags weit entfernter Elemente vor. Das Auffinden eines passenden Wertes, ab wann zwei Elemente als zu weit voneinander entfernt angesehen werden können um sich gegenseitig zu beeinflussen, hängt jedoch von der Geometrie der Szene ab und muss manuell justiert werden.

Das Vorgehen von Bunnell [Bun05] ist so performant, dass auch für animierte Modelle Ambient Occlusion in Echtzeit berechnet werden kann. Wie Hoberock und Jia [HJ07] gezeigt haben, stellt das Verfahren jedoch Anforderungen an die zugrunde liegende Geometrie und führt auch auf Fragment-Basis zu Artefakten. Ihre Lösung ist zwar robuster, im Gegenzug aber auch aufwändiger zu berechnen. So müssen für jeden darzustellenden Punkt unter Umständen gleich mehrere Polygone an dessen Grundebene geclippt werden. Beide Verfahren benötigen mindestens zwei Render Passes und generell lässt sich festhalten, dass ihre Rechenzeit von der Auflösung der Geometrie abhängt.

Hegeman et al. [HPAD06] stellen ein Verfahren zur Berechnung von Ambient Occlusion vor, das speziell auf enorm komplexe Szenen, beispielsweise mehrere Bäume mit einer Vielzahl von Blättern, zugeschnitten ist. Um den Aufwand so gering wie möglich zu halten, verzichten sie auf eine globale Berechnung der Visibilität und wählen dagegen einen statistischen Ansatz, der lediglich auf der Position und der Normalen eines zu betrachtenden Punktes basiert. Ein einzelner Baum wird dazu durch eine oder mehrere kugelförmige Schalen angenähert. Ambient Occlusion für einen Punkt berechnet sich aus dem Abstand der Position zum Kugelzentrum, sowie aus dem Volumen des Kugelsegments – d.h. dem oberen Bereich der Kugel bezüglich der Normalen des darzustellenden Punktes, nach einem Schnitt mit der Ebene orthogonal zur Normalen. Daneben wird eine frei gesetzte Occluder-Dichte in die Berechnung einbezogen. Die Kugeln werden zusätzlich zur Berechnung der Verschattung zwischen Objekten eingesetzt. Damit ist vor allem der Schattenwurf eines Baumes auf den Boden gemeint. Abhängig von Distanz und Radius der Kugel wird dafür der aufgespannte Raumwinkel aus Sicht des verschatteten Punktes approximiert und Ambient Occlusion aus dem Verhältnis dieses Raumwinkels zum gesamten oberen Halbraum des Punktes berechnet. Abbildung 3.7 illustriert die Parameter, die Hegeman et al. [HPAD06] einsetzen um Ambient Occlusion für Blätter zu berechnen und um die Verschattung des Bodens durch eine Menge von Blättern anzunähern.

Der Ansatz von Hegeman et al. [HPAD06] kommt beinahe vollständig ohne eine Vorberechnung aus und benötigt nicht zuletzt deshalb relativ wenig Speicher. Wie sie selbst betonen, erzielt das Verfahren keine physikalische Korrektheit – es weicht sogar teilweise gravierend davon ab – sondern setzt auf ein visuell akzeptables Ergebnis, das sich im Gegensatz zu vielen anderen Veröffentlichungen auch bei hochkomplexen Szenen in interaktiven Frameraten erzielen lässt.

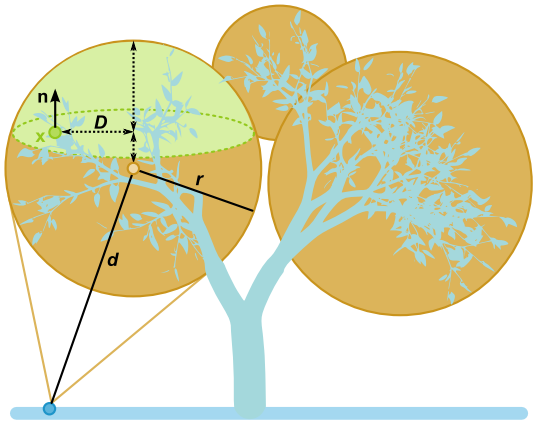


Abbildung 3.7: Approximation von Ambient Occlusion speziell für Bäume.

3.3.4 Vertexbasierte outside-in Methoden

Sattler et al. [SSZK04] präsentieren ein Verfahren, in dem sie, ähnlich wie Pharr und Green [PG04], das Problem der Beleuchtung aus allen Richtungen durch die gleichmäßige Platzierung mehrerer gerichteter Lichtquellen auf einer Kugel lösen, welche die darzustellende Szene umspannt.[4] Die Einfallsrichtung der ersten sechs Lichtquellen wird durch die Eckpunkte eines Oktaeders und den Szenenmittelpunkt bestimmt. Durch das n-fache Setzen eines Mittelpunktes auf der Verbindungslinie zwischen zwei Eckpunkten und seiner anschließenden Projektion auf die Kugel lassen sich gleichmäßig $2+4^{n+1}$ direktionale Lichtquellen verteilen. Aus der Sicht jeder Lichtquelle wird zunächst die gesamte Geometrie mit einer orthographischen Projektion in den Tiefenpuffer gerendert. Alle Eckpunkte der Geometrie werden anschließend zusätzlich als Punktemenge gerendert. Für jeden Eckpunkt wird dann mit der OpenGL-*Extension* `ARB_OCCLUSION_QUERY` ermittelt, ob dieser von der Lichtquelle aus sichtbar ist oder von Geometrie verdeckt wird. Die Ergebnisse der Sichtbarkeitsanfrage zwischen jeder Lichtquelle und den Eckpunkten der Geometrie wird in einer Sichtbarkeitsmatrix pro Eckpunkt festgehalten. Abbildung 3.8 zeigt das Vorgehen nach Sattler et al. [SSZK04] schematisch.

Der Rechenauwand zur Erstellung der Sichtbarkeitsmatrix hängt vor allem von der Anzahl an Lichtquellen ab, die nicht zu klein sein darf, um vernünftige Ergebnisse zu erzielen. Wird lediglich der Blickpunkt auf die Szene verändert, muss die Sichtbarkeitsmatrix nicht neu berechnet werden. Gleiches gilt für eine Veränderung der Beleuchtung, die beispielsweise durch Environment Maps bildbasiert gesteuert werden kann. Sobald ein Objekt jedoch verformt wird oder sich die relative Position zweier Objekte zueinander verändert, muss die Sichtbarkeitsmatrix neu berechnet werden. Als Optimierung schlagen Sattler et al. [SSZK04] vor, die zeitliche Kohärenz bei der Blickpunktänderung auszunutzen, sodass nur die Verdeckung für sichtbare

[4] Es werden keine echten gerichteten Lichtquellen erzeugt. Dieses Modell dient hier der besseren Anschauung für die Positionierung orthographischer Kameras.

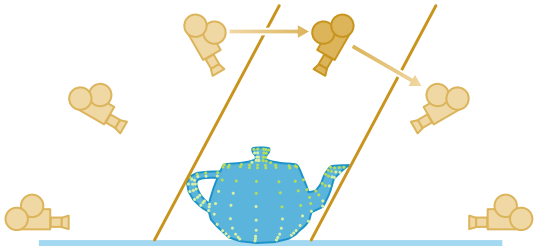


Abbildung 3.8: Rendering einer Punktewolke aus Sicht jeder orthographischen Kamera.

Punkte ermittelt wird, deren Menge sich innerhalb weniger Frames nur geringfügig ändert.

Einen innovativen Ansatz zur Darstellung von Ambient Occlusion basierend auf Konzepten der Informationstheorie stellen González et al. [GSF07] vor. Ähnlich wie in vorangehenden Verfahren [PG04, SSZK04] setzen sie verschiedene Blickpunkte rund um das zu darzustellende Objekt. Die Beziehung zwischen einer Menge von Blickpunkten und den Polygonen des Objekts wird durch einen Kanal ausgedrückt und das Konzept der Transinformation (engl. *mutual information*) eines solchen Kanals dient als Qualitätsmaß für einen Blickpunkt bezüglich eines Polygons. Konkret berechnen González et al. [GSF07] die Transinformation für einen Blickpunkt aus der projizierten Fläche der einzelnen Polygone relativ zur Fläche des gesamten Objekts von eben diesem Blickpunkt aus gesehen. Dazu wird jedes Polygon in einer eigenen Farbe gerendert und die Anzahl von Pixeln in dieser Farbe errechnet. Die Transinformation lässt sich damit als ein Grad der Korrelation zwischen Blickpunkt und Polygon interpretieren. Indem unter Zuhilfenahme des Bayes-Theorems der umgekehrte Kanal betrachtet wird, lässt sich die Transinformation für ein bestimmtes Polygon bezogen auf die Menge aller Blickpunkte bestimmen – d.h. ein Maß für die Sichtbarkeit des betrachteten Polygons und damit ein Wert, der Ambient Occlusion approximiert.[5]

Indem den unterschiedlichen Blickpunkten Farben und ein Wert der Wichtigkeit zugewiesen werden, ermöglicht das Vorgehen von González et al. [GSF07] eine Visualisierung durch *Non-Photorealistic Rendering*, die beispielsweise zur gezielten Hervorhebung wichtiger Objektteile eingesetzt werden kann. Bei einem Vergleich mit Ambient Occlusion durch Ray Tracing fällt auf, dass dem beschriebenen Vorgehen, neben den für einen vertexbasierten Ansatz typischen Interpolationsartefakten, einige besonders dunkle Stellen auftreten, die ihren Ursprung in einer zu geringen Auflösung bei der Projektion kleiner Polygone haben. Über Performanz und Speicherbedarf des Verfahrens machen González et al. [GSF07] keine Angaben, es ist jedoch davon auszugehen, dass letzterer aufgrund des umfangreichen Datengehalts der Transinformationsmatrix zu jedem Objekt der Szene nicht gering ist.

[5] Mit der Umkehrung des Kanals wandelt sich die outside-in Sichtweise in eine inside-out Sichtweise. Da die Daten jedoch aus der zuerst genannten Sicht herangezogen werden, wird der Ansatz hier den outside-in Methoden zugeordnet.

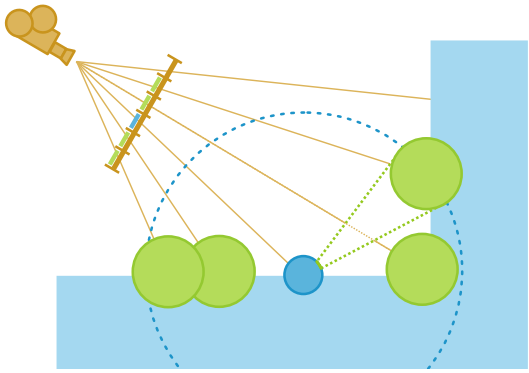


Abbildung 3.9: Bildraumbasierte Erkennung von Occludern.

3.3.5 Bildbasierte Methoden

Shanmugam und Arikan [SA07] unterteilen bei ihrem Vorgehen Ambient Occlusion zunächst in Bereiche hoher und niedriger Frequenz, die sie mit unterschiedlichen Ansätzen behandeln. Der erste Fall tritt durch nahe gelegene Occluder auf und führt zur unstetigen Variation von Ambient Occlusion zwischen benachbarten Punkten. Bei der Suche nach dieser Art von Occludern beschränken sich Shanmugam und Arikan [SA07] auf den Bildraum. Dazu werden zunächst nur die Normalen und Tiefenwerte aus Sicht der Kamera gerendert. Um Ambient Occlusion für ein Pixel des Bildraums zu ermitteln, werden benachbarte Pixel, die einen ähnlichen Tiefenwert besitzen, in das Weltkoordinatensystem zurück projiziert und dort als kugelförmige Occluder approximiert (vgl. Abbildung 3.9). Der Fall, dass einflussreiche Occluder verdeckt sein könnten, wird durch Berücksichtigung der ersten beiden Tiefenebenen mittels *Depth Peeling* [Eve01] behandelt. Ambient Occlusion für einen Punkt wird rechnerisch durch Aufsummierung aller angenäherten Kugelkalotten, die von sämtlichen kugelförmigen Occludern auf dem oberen Halbraum des Punktes aufgespannt werden, bestimmt. Um auch die niedrigfrequente Verschattung durch relativ weit entfernte Occluder zu berücksichtigen, wird die Geometrie sämtlicher Objekte innerhalb der Szene durch Kugeln approximiert. Der Aufwand wird dadurch verringert, dass diese Kugeln nur innerhalb eines frei wählbaren Radius als potentielle Occluder angesehen werden. Den Fragment Shader durchlaufen lediglich Pixel, die möglicherweise von den kugelförmigen Occludern verschattet werden. Dies geschieht indem zuvor im Vertex Shader für jede Kugel ein *Billboard* erzeugt wird, welches den Einflussbereich repräsentiert.

Das Verfahren von Shanmugam und Arikan [SA07] ist abhängig von der Bildschirmauflösung und der verfügbaren Speicherbandbreite. Erst die Graphikhardware der NVIDIA GeForce 8er-Serie ermöglicht eine Darstellung in Echtzeit. Je nach Orientierung eines Occluders relativ zur Kamera kann es bei dem bildbasierten Ansatz zu Artefakten kommen. Ein länglicher Occluder parallel zur Blickrichtung besitzt beispielsweise einen zu geringen Einfluss. Andererseits kann die Berechnung von

Ambient Occlusion mit niedriger Frequenz zu einer zu starken Verschattung eines Punktes führen, wenn sich mehrere Occluder vom jeweiligen Punkt aus gesehen in einer Richtung befinden, da sich ihr Einfluss durch Addition ermittelt, obwohl sich die Occluder tatsächlich gegenseitig verdecken. Auch wenn Shanmugam und Arikan [SA07] keinen Hinweis darauf liefern, so ist doch davon auszugehen, dass Pixel zum Rand des Bildschirms hin heller dargestellt werden, da sie weniger Nachbarn besitzen, die zur Approximation von hochfrequentem Ambient Occlusion herangezogen werden könnten. Dies könnte beispielsweise bei Kamerafahrten zu Unstetigkeiten führen. Die Vorteile des Verfahrens bestehen vor allem darin, dass keine Vorberechnung nötig ist und animierte Objekte keinen zusätzlichen Aufwand darstellen, da sämtliche Berechnungen für jedes Frame neu vorgenommen werden.

Ein ähnlicher Ansatz zu dem gerade vorgestellten Vorgehen wurde mit dem so genannten *Screen-Space Ambient Occlusion* (SSAO) [Mit07] in dem Computerspiel „Crysis“ (Crytek GmbH) umgesetzt. Die einzige Voraussetzung ist, dass die Tiefenwerte der Szene aus Sicht der Kamera in einer Textur vorliegen. Für jedes Pixel wird Ambient Occlusion lediglich anhand mehrerer Tiefenvergleiche mit benachbarten Pixeln durchgeführt. Nach der Einteilung durch Shanmugam und Arikan [SA07] entspricht dies einer Beschränkung auf Ambient Occlusion mit hoher Frequenz. Um die Anzahl an Samples möglichst gering zu halten, wird ihre Position auf der Bildebene innerhalb eines festen Radius zufällig gewählt.

Bei dem von Mittring [Mit07] beschriebenen Vorgehen handelt es sich lediglich um eine sehr grobe Approximation von Ambient Occlusion, die allerdings performant und mit sehr geringen Speicheranforderungen verbunden ist. Wie genau aus den Tiefenvergleichen ein Maß für die Verdeckung hergeleitet werden kann und welchen Einschränkungen das Verfahren unterliegt wird allerdings nicht erläutert.

3.3.6 Methoden für animierte Objekte

Kontkanen und Aila [KA06] stellen eine Möglichkeit vor, wie Animationsparameter eines Modells auf Werte für Ambient Occlusion abgebildet werden können. Sie gehen davon aus, dass Ambient Occlusion zuvor für einige Referenzposen des Modells auf Vertex-Basis ausgerechnet wurde. Diese Werte werden zeilenweise für alle Referenzposen in einer Matrix $\mathbf{A}$ gespeichert. In den Zeilen einer Matrix $\mathbf{J}$ werden entsprechend alle Animationsparameter der Referenzposen, also beispielsweise die Winkel der Gelenke (engl. *joints*), gespeichert. Gesucht ist nun eine Matrix $\mathbf{T}$ für die gilt $\mathbf{A} = \mathbf{JT}$. Diese bestimmen Kontkanen und Aila [KA06] durch Multiplikation der Pseudoinversen von $\mathbf{J}$ mit $\mathbf{A}$. Ein solches Vorgehen unterliegt den zwei vereinfachenden Annahmen, dass Ambient Occlusion linear von den Animationsparametern abhängt und, dass sich Ambient Occlusion unabhängig aus der Summe von Verdeckungen berechnen lässt. Um Artefakte auszuschließen wird daher bei der Vorberechnung eine große Anzahl von Referenzposen benötigt. Im Gegenzug ist die Performanz zur Laufzeit sehr günstig, da zur Berechnung von Ambient Occlusion für eine Pose lediglich so viele Multiplikationen und Additionen nötig sind, wie es Animationsparameter gibt.

Wie bei Kontkanen und Aila [KA06] basiert das Vorgehen von Kirk und Arikan [KA07] auf vorberechnetem Ambient Occlusion für jeden Vertex bei mehreren Referenzposen eines animierten Modells. Anstelle von Animationsparametern in Form von Gelenkwinkeln nutzen sie jedoch die Position von so genannten *Handles*, für jedes Gelenk und jeden Knochen (engl. *bone*) des Modells. Anhand der Freiheitsgrade der Referenzposen, werden diese in kleinere Gruppen (so genannte *pose clusters*) aufgeteilt, sodass angenommen werden kann, dass die Variation der Posen, und damit auch die Änderung von Ambient Occlusion, innerhalb einer solchen Gruppe nur gering ist. Eine lineare Abbildung der Animationsparameter auf Ambient Occlusion findet im Gegensatz zum Vorgehen von Kontkanen und Aila [KA06] nur innerhalb eines Clusters statt. Die ansonsten niedrige Performanz wird gesteigert, indem Mengen von benachbarten Eckpunkten zu Gruppen zusammengefasst werden, da diese in der Regel ähnliche Verdeckungswerte besitzen. Da eine lineare Abbildung auf vorberechnete Verdeckungswerte nur innerhalb bestimmter Gruppen von Posen zugelassen wird ist das Verfahren von Kirk und Arikan [KA07] robuster als der Ansatz von Kontkanen und Aila [KA06], andererseits ist es jedoch schwieriger zu implementieren und rechenintensiver.

3.4 Zusammenfassung

Unter Berücksichtigung der Verdeckung durch umgebende Geometrie wird bei Ambient Occlusion der Anteil des Umgebungslichts berechnet, der einen darzustellenden Punkt beleuchtet. Im Gegensatz zur herkömmlichen Beleuchtung durch einzelne Punktlichtquellen erscheinen auf diese Weise erzeugte Bilder, zumindest bei einer Beschränkung auf diffuse Materialien, deutlich realistischer und erleichtern darüber hinaus die Wahrnehmung von Formen.

Zur Berechnung von Ambient Occlusion unter Verwendung der Graphikhardware sind zahlreiche Veröffentlichungen erschienen, sodass eine Kategorisierung lohnenswert erscheint. Die Schwerpunkte der unterschiedlichen Vorgehensweisen verteilen sich auf gute Ausgabequalität, hohe Performanz und niedrige Speicheranforderungen. Ein Verfahren das gleichzeitig allen Anforderungen in vollem Maße genügt existiert bislang nicht, auch wenn das Kriterium des Speicherbedarfs in der Regel nicht denselben Stellenwert einnimmt wie Qualität und Performanz. Einige Methoden sind darüber hinaus prinzipiell auf Starrkörper beschränkt, sodass sie sich nur für bestimmte Anwendungsfälle eignen.

Neben den vorgestellten Verfahren, können manche Vorgehensweisen zur Darstellung weicher Schatten ebenso zur Berechnung von Ambient Occlusion eingesetzt werden. Besonders häufig wird hier das von Kautz et al. [KLA04] entwickelte Verfahren genannt, das zur Berechnung eines Visibilitätsterms pro Eckpunkt die Rasterisierung der Graphikhardware ausnutzt. Der Ansatz von Dong et al. [DKTS07] nutzt in ähnlicher Weise wie das Vorgehen von Bunnell [Bun05] eine Unterteilung der Szene in finite Elemente um Sichtbarkeit und indirekte Beleuchtung zu berechnen. Damit dabei interaktive Frameraten erzielt werden verwenden sie ebenfalls eine

hierarchische Organisation der Elemente. Zhou et al. [ZHL*05] stellen mit *Shadow Fields* eine Datenstruktur vor, die anhand vorberechneter Informationen effizient die Verschattung zwischen Starrkörpern berechnet. Ein ähnlicher Ansatz wird mit dem im nächsten Kapitel vorgestellten Verfahren verfolgt.

4 Coherent Shadow Maps

Flächige Lichtquellen können durch eine geschickte Diskretisierung unter Verwendung einer Vielzahl von Punktlichtquellen imitiert werden. Licht, das aus allen Richtungen der Hemisphäre eintrifft, kann auf die gleiche Weise nachgeahmt werden, indem der Bereich möglicher Einfallswinkel durch eine große Menge gerichteter Lichtquellen approximiert wird. Bei einer ausreichend großen Anzahl von Lichtquellen, die zur Diskretisierung eingesetzt werden, sind Fehler, die durch die Annäherung auftreten, kaum noch wahrnehmbar. Für eine realistische Darstellung muss für jede dieser Lichtquellen ein Schatten berechnet werden. Um dies möglichst effizient zu bewerkstelligen, können Informationen wie Depth Maps, die zur Bestimmung der Visibilität eingesetzt werden, bereits im Voraus berechnet werden. Ohne eine Komprimierung dieser Daten würde ein solches Vorgehen allerdings eine enorme Menge an Speicher erfordern. In diesem Kapitel wird eine Datenstruktur vorgestellt, die eine Verwaltung einer großen Menge vorberechneter Depth Maps durch eine verlustlose Komprimierung effizient gestaltet.

4.1 Motivation

Auf die gleiche Weise wie die Beleuchtung, lässt sich auch die Verschattung bei komplexen Beleuchtungssituationen, wie durch Flächenlichtquellen oder unendlich weit entferntes Umgebungslicht, annähern. Dazu wird für jede der zur Approximation der Beleuchtung eingesetzten Lichtquellen eine Shadow Map berechnet, sodass für einen darzustellenden Punkt ermittelt werden kann, ob er im Schatten liegt. Eine Menge von Punktlichtquellen kann eine lokale Lichtquelle beliebiger Form nachbilden. Für jede einzelne dieser Punktlichtquellen kann die Visibilität in allen sechs Raumrichtungen mit einer Depth Cube Map ermittelt werden. Eine Menge von gerichteten Lichtquellen kann Licht annähern, das aus allen Richtungen der Hemisphäre eintrifft. Der Schatten, der von einer gerichteten Lichtquelle geworfen wird, lässt sich anhand einer orthographischen Depth Map berechnen. Wird eine Vielzahl solcher orthographischen Depth Maps rund um ein Objekt errechnet, erlaubt dies die Approximation der Visibilität eines Punktes bei einer Beleuchtung aus allen Richtungen der Hemisphäre.

Es ist nahe liegend, dass die Berechnung von Schatten durch Shadow Mapping für jede einzelne von bis zu mehreren tausend Lichtquellen nicht in interaktiven Frameraten durchführbar ist. Auf der anderen Seite ist eine Vorberechnung der Verschattungsinformationen für eine komplexe Szene mit in der Regel unerfüllbaren Speicheranforderungen verbunden, wenn eine interaktive Transformation von Objekten erlaubt sein soll. Um Dynamik zwischen Starrkörpern zu unterstützen, kann mit

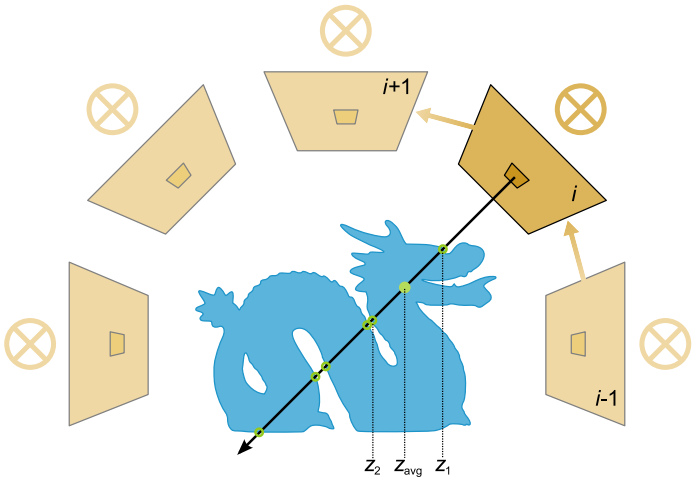


Abbildung 4.1: Für die Schattenberechnung kann ein Tiefenwert zwischen z_1 und z_2 gewählt werden.

jedem Objekt der Szene eine vorberechnete Depth Map für jede potentielle Richtung des Lichteinfalls verwaltet werden. Diese ermöglicht eine schnellere Berechnung von Selbstverschattung und unterstützt darüber hinaus die Verschattung zwischen Objekten, wobei zur Visibilitätsberechnung die Depth Map des Occluders herangezogen wird. Da für jeden Objekttyp jedoch eine Vielzahl von Depth Maps in ausreichend hoher Auflösung vorliegen müssen, liegt der Speicherbedarf einer Anwendung, die auf diese Weise verfährt schnell bei mehreren Gigabyte.

4.2 Beschreibung des Verfahrens

Ritschel et al. [RGKM07] lösen das Problem des hohen Speicherbedarfs bei der Verwaltung einer Vielzahl von vorberechneten Depth Maps pro Objekt durch eine verlustlose Komprimierung, welche die geringen Unterschiede zwischen einzelnen Depth Maps ausnutzt. Die komprimierte Information wird in einer Datenstruktur, die sie *Coherent Shadow Maps* (CSMs) nennen, gespeichert und kann für die Berechnung der Verschattung eines darzustellenden Punktes durch unendlich weit entfernte Beleuchtung und durch lokale Flächenlichtquellen eingesetzt werden. Die Effizienz der CSMs demonstrieren Ritschel et al. [RGKM07] durch Integration in einen interaktiven, GPU-gestützten Monte-Carlo-Renderer für direkte Beleuchtung.

4.2.1 Komprimierung von Shadow Maps

Die Komprimierung der Tiefeninformation basiert zum einen darauf, dass die Depth Maps in einer Reihenfolge abgefragt werden, von der erwartet werden kann, dass sich

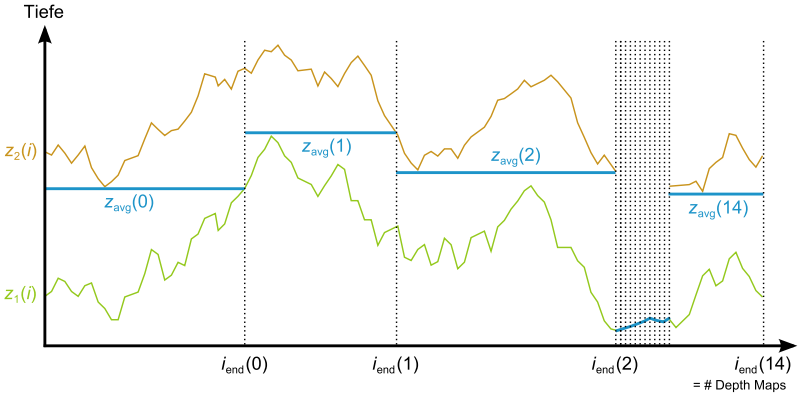


Abbildung 4.2: Zwischen einer Vielzahl von Tiefenwerten z_1 und z_2 genügt es einen stückweise konstanten, mittleren Tiefenwert z_{avg} zu speichern.

der Wert eines bestimmten Texels[1] zweier Depth Maps nur geringfügig unterscheidet. Dabei erweist sich eine Iteration, die einer raumfüllenden Hilbert-Kurve folgt als besonders effektiv. Zum wird bei der Komprimierung die Tatsache ausgenutzt, dass es für das Shadow Mapping genügt, im Texel einer Depth Map einen beliebigen Tiefenwert zwischen dem ersten und dem zweiten Schnittpunkt eines Strahls, der in dem Texel seinen Ursprung hat, abzuspeichern. Abbildung 4.1 zeigt, dass für alle eingezeichneten Punkte der Schatten durch die Lichtquelle zu Depth Map i mit einem Tiefenwert z_{avg}, zwischen z_1 und z_2 berechnet werden kann. Solange demnach für dasselbe Texel mehrerer aufeinander folgender Depth Maps der kleinste Tiefenwert des zweiten Schnittpunkts weiter entfernt ist als der größte Tiefenwert des ersten Schnittpunkts, genügt es, nur einen einzigen mittleren Tiefenwert in der CSM zu speichern.[2] Zu jedem mittleren Tiefenwert verwaltet die CSM einen Wert i_{end}, der angibt bis zur wievielten Depth Map der Iterationsfolge dieser gültig ist. Abbildung 4.2 zeigt wie mit diesem Vorgehen die ersten und zweiten Tiefenwerte vieler Depth Maps durch gerade einmal 14 mittlere Tiefenwerte komprimiert werden können. Die Abbildung illustriert gleichzeitig den Spezialfall, dass für einige Depth Maps keine zweite Tiefenebene existiert. In diesem Fall versagt die Komprimierung und alle z_1-Werte müssen einzeln gespeichert werden. Außerdem muss dann darauf geachtet werden, dass umgehend die Komprimierung der vorangegangen Texel abgeschlossen wird – in Abbildung 4.2 durch das Setzen von $i_{\text{end}}(2)$ gezeigt – da ansonsten die unendlich weit entfernte zweite Tiefenebene als Freiheitsgrad bei der

[1] Der Begriff *Texel* (für *Texture Element*) leitet sich vom Wort Pixel ab und wird hier benutzt, um zu verdeutlichen, dass es sich nicht um das kleinste durch einen Bildschirm darstellbare Element handelt, sondern um einen von einer Textur verwalteten Bildpunkt. Wie in Abschnitt 2.2.2 erläutert wurde, sind die typischen Aliasing-Effekte des Shadow Mapping gerade auf die Diskrepanz zwischen Pixel und Texel zurückzuführen.

[2] Für geschlossene Objekte, so betonen Ritschel et al. [RGKM07], kann der zu speichernde Tiefenwert sogar frei zwischen dem ersten und dem dritten Schnittpunkt gewählt werden, da die BRDF für die abgewandte Seite (in Gegenrichtung zur Normalen dieser Fläche) ohnehin Null ist und einen Visibilitätstest unnötig macht.

Komprimierung interpretiert würde. Existieren für ein Texel aufeinander folgender Depth Maps auch keine ersten Tiefenebenen, was häufig zu den Rändern der Depth Maps hin der Fall ist, kann die Komprimierung dagegen durch das Speichern des maximalen Tiefenwertes in z_{avg} voll ausgenutzt werden. Wurden für ein Texel alle Depth Maps durchlaufen und die Tiefenwerte komprimiert, wird mit dem nächsten Texel fortgefahren. Die CSM selbst ist eine zweidimensionale Textur, die Wertepaare der Form $(z_{\text{avg}}, i_{\text{end}})$ enthält.

4.2.2 Abfrage der Sichtbarkeitsinformationen

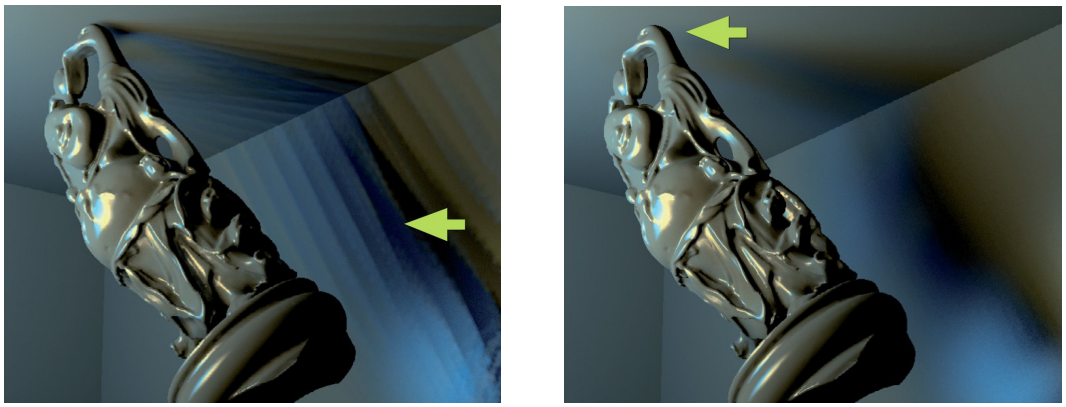
Um zu überprüfen, ob ein darzustellender Punkt im Schatten einer der vielen zur Diskretisierung eingesetzten Lichtquellen liegt, wird dieser in das Koordinatensystem der entsprechenden Depth Map projiziert und nach der perspektivischen Division mit dem passenden Eintrag innerhalb der CSM-Textur verglichen. Der jeweilige Tiefenwert z_{avg} einer Depth Map i lässt sich mit einer binären Suche über den monoton steigenden i_{end}-Werten realisieren. Obwohl die Position des auszuwertenden Texels in Depth Map i nach Projektion des Punktes in das Koordinatensystem der Depth Map bekannt ist, lässt sich jedoch nicht leicht feststellen, wo innerhalb der CSM-Textur die Suche gestartet werden muss – d.h., ab wann sich die $(z_{\text{avg}}, i_{\text{end}})$-Wertepaare auf das auszuwertende Texel beziehen. Neben der CSM-Textur selbst wird daher auf eine zusätzliche zweidimensionale Textur mit der selben Auflösung der komprimierten Depth Maps zugegriffen. Diese verwaltet für jedes Texel eine Position innerhalb der CSM-Textur, ab welcher sich die gespeicherten Tiefenwerte auf das entsprechende Texel beziehen.

Um eine Verschattung zwischen mehreren Objekten zu berücksichtigen, müssen zusätzlich die CSM-Texturen der potentiellen Occluder untersucht werden. Die Tiefe eines darzustellenden Punktes eines Objekts wird dazu aus Sicht der Depth Maps der objektfremden CSM-Texturen, die orthogonal zur untersuchten Lichteinfallsrichtung stehen, berechnet und mit den dort vorgespeicherten Tiefenwerten verglichen. Die Depth Maps frei transformierbarer Objekte müssen dabei stets mit der inversen Transformationsmatrix multipliziert werden um Gültigkeit zu bewahren.

4.2.3 Berücksichtigung lokaler Lichtquellen

Da lokale Lichtquellen in alle Richtungen Licht abgeben und frei in der Szene platziert werden können, wird die Tiefeninformation in Form von Cube Maps, die aus einer perspektivischen Depth Map pro Würfelseite bestehen, vorberechnet und verwaltet. Die Platzierung mehrerer Depth Cube Maps approximiert eine lokale Flächenlichtquelle beliebiger Form. Die Komprimierung nutzt dabei die Kohärenz gleicher Würfelseiten benachbarter Depth Cube Maps.

Da die vorberechnete Tiefeninformation in Form von Depth Cube Maps nur bei einer statischen Szene Gültigkeit behält, stellen Ritschel et al. [RGKM07] das Konzept der *semilokalen* Lichtquellen vor. Die CSMs werden dabei, wie im Fall unendlich weit entfernter Lichtquellen, durch orthographische Depth Maps generiert. Abhängig



(a) Banding-Artefakte (b) Fehlende Schatten

Abbildung 4.3: Diskretisierungsfehler beim Einsatz von CSMs (Abbildung überarbeitet aus [RGKM07]).

von Position und Form der semilokalen Lichtquelle werden zur Schattenberechnung nur bestimmte Depth Maps herangezogen. Auf diese Weise werden auch dynamische Szenen unterstützt, allerdings dürfen sich die Lichtquellen nicht innerhalb der konvexen Hülle eines Objekts befinden.

4.2.4 Integration in einen Monte-Carlo-Renderer

Ritschel et al. [RGKM07] nutzen die CSMs für eine Monte-Carlo-Integration der Rendering Equation zur Berechnung direkter Beleuchtung auf Fragment-Basis. Die Dichte wird dabei zur Reduzierung der Varianz mittels *Importance Sampling* gewählt und der Integrationsbereich wird zur besseren Verteilung der Samples in disjunkte Teilbereiche aufgeteilt (*Stratified Sampling*). Für jedes Fragment werden aufgrund der Sampling-Strategien mehrere Richtungen ermittelt aus denen die einfallende Beleuchtung zu untersuchen ist. Die CSM-Textur des dem Fragment entsprechenden Objekts kommt zum Einsatz, um festzustellen, ob es aus der gewählten Richtung zur Selbstverschattung kommt. Ist dies nicht der Fall, werden die CSM-Texturen der anderen Objekte, die Teil der Szene sind, herangezogen um zu überprüfen, ob es einen Occluder gibt.

Um interaktive Frameraten zu erreichen nutzen Ritschel et al. [RGKM07] auch das so genannte *Progressive Rendering* aus. Das Rauschen, zu dem es bei einer niedrigen Anzahl von Samples kommt, nimmt dabei innerhalb kurzer Zeit zunehmend ab, indem der Durchschnitt aufeinander folgender Frames, in denen die Kamera nicht bewegt wurde, ermittelt wird.

4.3 Typische Artefakte

Obwohl Coherent Shadow Maps eine enorme Komprimierungsrate besitzen und es für alle Instanzen eines 3D-Modells genügt eine einzige CSM-Textur zu verwalten, wachsen die Speicheranforderungen schnell an, wenn Auflösung und Anzahl der

Depth Maps genügend hoch sein sollen um Artefakte von vornherein zu vermeiden. Abbildung 4.3 auf der vorherigen Seite zeigt die beiden Arten von Darstellungsfehlern, die dem Einsatz von CSMs typischerweise anhängen. Zum einen führt eine zu geringe Anzahl von Depth Maps bei der Diskretisierung des Umgebungslichts, beziehungsweise der Flächenlichtquelle, zu Artefakten, die sich in Form von Bändern mit deutlichen Helligkeitsunterschieden bemerkbar machen. Zum anderen wird mit geringerer Auflösung der Depth Maps zunehmend weniger Schatten erkannt, insbesondere solcher, der von feinen geometrischen Details geworfen wird. Typische Aliasing-Artefakte in Form von gezackten Schattenkanten hängen zusätzlich direkt von der Auflösung der Depth Maps ab und werden von Ritschel et al. [RGKM07] durch eine geeignete Filterung behoben.

4.4 Zusammenfassung

Coherent Shadow Maps erlauben eine pixelgenaue Berechnung der Visibilität zwischen Starrkörpern. Im Fall distanter oder semilokaler Lichtquellen wird dazu der Raum um jeden potentiellen Occluder durch die Berechnung einer Vielzahl orthographischer Depth Maps diskretisiert. Die enorme Datenmenge zu der es dabei kommt, wird durch Ausnutzung der Kohärenz zwischen benachbarten Depth Maps und des Freiheitsgrades bei der Wahl eines repräsentativen Tiefenwertes verringert. Die Komprimierungsrate hängt von der Form und Komplexität des Objekts ab. Je mehr das Objekt einer Kugel ähnelt, umso geringer ist der Unterschied zweier aufeinander folgender Depth Maps und umso besser ist das Kompressionsverhältnis.

Zur Laufzeit werden keine Depth Maps generiert. Um die Visibilität entlang einer Richtung zu ermitteln, muss lediglich der vorberechnete Tiefenwert innerhalb der Datenstruktur gefunden und mit dem tatsächlichen Tiefenwert verglichen werden – dabei wird letzterer durch Projektion des darzustellenden Punktes in das Koordinatensystem der in der Sample-Richtung gelegenen, orthographischen Kamera berechnet. Obwohl sich der Aufwand der binären Suche logarithmisch zur Anzahl der komprimierten Depth Maps verhält, erweist sie sich als der rechenintensivste Teil des Vorgehens. Da die Visibilität eines Punktes erst bei mehreren Samples hinreichend genau ermittelt werden kann, wird auch der Suchalgorithmus für jedes Pixel mehrfach ausgeführt.

Mit CSMs lässt sich die Visibilität innerhalb von diskreten Raumabschnitten vorberechnen und auf effiziente Weise abspeichern. Je höher die Auflösung und die Anzahl der zugrunde liegenden Depth Maps ist, umso genauer lässt sich die Sichtbarkeit einzelner Punkte im Raum berechnen. Gleichzeitig steigt mit einer größeren Anzahl an Depth Maps die Komprimierungsrate. Ritschel et al. [RGKM07] haben gezeigt, dass, obwohl es sich um eine Approximation handelt, für den Fall direkter Beleuchtung mit CSMs eine mit Ray Tracing vergleichbare Darstellungsqualität erreicht werden kann. Der Vorteil von CSMs gegenüber Ray Tracing liegt insbesondere in der Darstellungsgeschwindigkeit. Da die Sichtbarkeitsinformationen vorberechnet und in Form von Texturen verwaltet werden, ermöglicht die Graphikhardware eine schnelle Auswertung.

CSMs unterliegen der Einschränkung, dass ein Emitter stets außerhalb der konvexen Hülle eines Objekts liegen muss, damit sich die Visibilität zwischen Sender und Empfänger feststellen lässt. Dass sich die Idee hinter CSMs dennoch nicht allein auf eine Simulation direkter Beleuchtung anwenden lässt, zeigen Ritschel et al. [RGKS08] anhand der *Coherent Surface Shadow Maps* (CSSMs). Hierbei wird eine Vielzahl von Depth Cube Maps über die gesamte Szene verteilt und komprimiert, sodass sich zwischen zwei beliebigen Punkten innerhalb der Szene die Visibilität feststellen lässt. Diese Information kann zu einer Beschleunigung der aufwändigen Visibilitätstests bei der Berechnung globaler Beleuchtung, etwa durch das Radiosity-Verfahren, eingesetzt werden. In Kombination mit CSMs erlaubt dieses Vorgehen frei transformierbare Starrkörper innerhalb der Szene.

5 Ambient Occlusion mittels Coherent Shadow Maps

Die Herausforderung bei der Berechnung von Ambient Occlusion liegt darin, für einen darzustellenden Punkt möglichst schnell und präzise zu ermitteln, wie groß der Anteil der Hemisphäre ist, der diesen beleuchtet. Durch die Komprimierung vieler Depth Maps, die von einer orthographischen Kamera generiert werden, deren Augpunkt an verschiedenen Positionen auf der Umkugel jedes potentiellen Occluders gesetzt wird, erlauben Coherent Shadow Maps in beliebige Richtungen die Visibilität zu überprüfen. Bei der Berechnung von CSMs sollte die Anzahl und Verteilung von Depth Maps so gewählt werden, dass die Komprimierungsrate möglichst hoch ist und Fehler durch die Diskretisierung der möglichen Lichteinfallsrichtungen möglichst gering sind. Um Ambient Occlusion anhand der vorberechneten und komprimierten Depth Maps möglichst schnell berechnen zu können, muss eine Menge repräsentativer Richtungen gefunden werden, für welche die Sichtbarkeit zu überprüfen ist. Da die Auswertung der Visibilität aufgrund der binären Suche in der CSM-Textur keinen geringen Aufwand darstellt, sollte die Menge an Sample-Richtungen möglichst gering sein. Andererseits führt eine zu geringe Anzahl an Samples zu einem stark verrauschten Ergebnisbild. Um den Ansprüchen an Geschwindigkeit und Qualität gerecht zu werden, sind daher ausgeklügelte Sampling-Strategien unabdingbar.

Zunächst wird der hier vorzustellende Ansatz zur Berechnung von Ambient Occlusion in einen Zusammenhang zu den bislang veröffentlichten Arbeiten gebracht. Danach wird auf die Realisierung von Ambient Occlusion durch CSMs eingegangen. Diese lässt sich in eine Vorverarbeitungsphase, die sich mit der Komprimierung der Depth Maps befasst, und eine Laufzeitphase, die anhand der erzeugten CSMs die Visibilität berechnet, aufteilen. Für das im Rahmen dieser Diplomarbeit umgesetzte System, auf das am Ende dieses Kapitels eingegangen wird, wurden zwei völlig unabhängige Programme zur Generierung und zur Nutzung von CSMs implementiert.

5.1 Motivation

Vorberechnete und in Form von Coherent Shadow Maps verwaltete Sichtbarkeitsinformation kann zur Berechnung von Ambient Occlusion herangezogen werden. Das Setzen einer großen Anzahl von Augpunkten bei der Generierung der CSMs entspricht dabei einem outside-in Vorgehen, das Ähnlichkeit zu den in Abschnitt 3.3.2 und Abschnitt 3.3.4 vorgestellten Verfahren aufweist. Die Diskretisierung des Raums um jeden potentiellen Occluder und die damit verbundene Organisation sichtbarkeitsrelevanter Informationen pro Objekt, bei dem es sich, um die Menge an Daten

gering zu halten, nur um einen Starrkörper handeln darf, findet sich, ebenfalls auf Texturen basierend, auch bei dem Vorgehen von Kontkanen und Laine [KL05] sowie Malmer et al. [MMAH05]. Während im ersten Fall jedoch keine Selbstverschattung berechnet wird und inter-object Ambient Occlusion nur gültig ist, falls sich das verdeckte Objekt nicht innerhalb der konvexen Hülle des Occluders befindet, liegt der Nachteil des zweiten Ansatzes im kubischen Speicherzuwachs mit feinerer Unterteilung des Raumes, da die Visibilität unkomprimiert gespeichert wird. Mit CSMs lässt sich sowohl intra-object Ambient Occlusion als auch die Verschattung zwischen Objekten berechnen, unabhängig von der Position der Objekte, solange diese sich nicht überschneiden und die Kamera nicht im Inneren eines Objekts positioniert ist. Die Komprimierungsrate der CSMs steigt mit der Anzahl gewählter Depth Maps, was eine hinreichende Diskretisierung des Raumes begünstigt. Sind die Depth Maps für ein Objekt berechnet und komprimiert, lassen sich die Daten in Form einer Datei abspeichern und für beliebig viele Instanzen des Objekts verwenden.

Zur Laufzeit lässt sich Ambient Occlusion auf Basis von CSMs als objektbasiertes inside-out Vorgehen klassifizieren. Dies ermöglicht es pixelgenau repräsentative Depth Maps auszuwählen um die Verdeckung zu berechnen. Eine genügend große Anzahl an Samples und komprimierten Depth Maps vorausgesetzt, lassen sich selbst bei Nahaufnahmen von Objektbereichen feine Unterschiede in der Schattierung erkennen. Dies ist ein deutlicher Vorteil gegenüber allen vertexbasierten Ansätzen, bei denen die Interpolation Fehler hervorruft. Zudem ist die Rechenzeit unabhängig von der Anzahl an Polygonen aus denen ein Objekt besteht, wenn sich die geometrische Komplexität nicht negativ auf die Komprimierungsrate auswirkt.

Für den Einsatz von CSMs bei der Berechnung von Ambient Occlusion spricht insbesondere, dass bis auf die Diskretisierung keine Annahmen gemacht werden, die dazu führen, dass das Ergebnis von einer Referenzlösung durch Ray Tracing abweicht. Auch wenn auf CSMs basierende Visibilitätstests von der Graphikhardware schneller durchführbar sind als das Verschießen von Schattenfühlern mittels Ray Tracing, so sind mit herkömmlicher Hardware dennoch momentan keine konstanten Darstellungsraten im Bereich von Echtzeit möglich. Aus diesem Grund ist der hier gewählte Ansatz sicherlich nicht mit allen in Abschnitt 3.3 vorgestellten Verfahren vergleichbar, deren Schwerpunkt, wie Tabelle 3.1 auf Seite 21 zeigt, zum Teil deutlich stärker auf Performanz als auf Darstellungsqualität liegt.

5.2 Vorverarbeitung

Bei der Vorverarbeitung werden mit dem in Abschnitt 4.2.1 beschriebenen Vorgehen für jeden Occluder eine Reihe von Depth Maps komprimiert, um während der Laufzeit zur Berechnung von Ambient Occlusion in beliebige Richtungen die Visibilität abfragen zu können. Faktoren, die die Komprimierungsrate beeinflussen, sind die Positionen der orthographischen Kameras, aus deren Sicht die Depth Maps erzeugt werden, die Anzahl der Depth Maps, die Reihenfolge in der die Depth Maps bei der Komprimierung abgefragt werden und die Ausrichtung des Occluder-Modells. Wel-

Modell		Depth Maps		Unkompr.	Kompr.	Rate
Bunny (mit Löchern)		32×32	1024	32 MB	3,7 MB	8,7 : 1
		32×64	2048	64 MB	5,1 MB	12,5 : 1
		45×45	2025	63,3 MB	6,0 MB	10,5 : 1
		45×90	4050	126,6 MB	8,5 MB	14,9 : 1
		64×64	4096	128,0 MB	10,1 MB	12,6 : 1
		64×128	8192	256,0 MB	14,6 MB	17,5 : 1
Buddha		32×32	1024	32 MB	2,4 MB	13,2 : 1
		32×64	2048	64 MB	3,4 MB	19,0 : 1
		45×45	2025	63,3 MB	3,9 MB	16,0 : 1
		45×90	4050	126,6 MB	5,3 MB	23,8 : 1
		64×64	4096	128,0 MB	6,5 MB	19,7 : 1
		64×128	8192	256,0 MB	8,5 MB	30,0 : 1

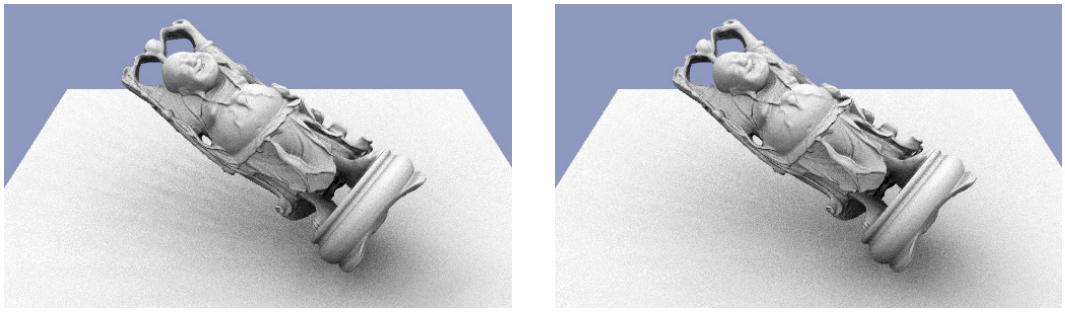
Tabelle 5.1: Komprimierungsraten für Depth Maps mit einer Auflösung von jeweils 128× 128 Texeln. Die Tiefe ist mit 16 Bit kodiert.

che Beobachtungen dabei im Einzelnen gemacht wurden, ist Thema des nächsten Unterabschnittes. Die Komprimierung basiert zudem auf einem Freiheitsgrad bei der Wahl eines repräsentativen Tiefenwertes. Abschnitt 5.2.2 beschäftigt sich mit der Umsetzung des Depth Peeling, das zur Ermittlung der ersten beiden Tiefenebenen eingesetzt wird, zwischen denen der Tiefenwert frei gewählt werden darf. Um die komprimierten Tiefeninformationen beliebig oft einsetzen zu können, ohne die aufwändige Komprimierung erneut durchführen zu müssen, wurde ein eigenes Dateiformat entwickelt, auf das in Abschnitt 5.2.3 eingegangen wird.

5.2.1 Diskretisierung der Bounding Sphere

Dieser Abschnitt befasst sich mit den untersuchten Möglichkeiten zur Positionierung der Augpunkte aller orthographischen Kameras, anhand derer die Depth Maps erzeugt werden. Die einfachste Möglichkeit besteht darin, eine Schrittgröße für die Polarkoordinaten θ und φ mit $0 \leq \theta < \theta_{\max} = \pi$, bzw. $0 \leq \varphi < \varphi_{\max} = 2\pi$, durch gleichmäßige Unterteilung zu berechnen.

Wird für $\varphi_{\max}$ eine doppelt so große Unterteilung wie für $\theta_{\max}$ gewählt, ist die Diskretisierung der Umkugel uniformer, da die Schrittgröße für θ und φ die gleiche ist, und es wird, wie Tabelle 5.1 deutlich zeigt, eine bessere Komprimierungsrate erzielt. Bei einer Unterteilung von $\theta_{\max}$ und $\varphi_{\max}$ in 32 bzw. 64 Bereiche (32×64) wird mit 2048 Depth Maps in etwa dieselbe Anzahl an Depth Maps komprimiert wie bei einer gleichmäßigen Unterteilung in jeweils 45 Bereiche ($45 \times 45 = 2025$). Dennoch werden nach der Komprimierung für das Buddha-Modell 0,5 Megabyte (MB) und für das Bunny-Modell sogar 0,9 MB weniger Daten benötigt. Tatsächlich ist es, wie Abbildung 5.1 demonstriert, sogar so, dass im Fall von 32×64 Depth Maps



(a) (b)

Abbildung 5.1: Banding-Artefakte nach (a) Komprimierung von 45×45 Depth Maps bzw. (b) Komprimierung von 32×64 Depth Maps. Im letzten Fall ist die CSM 0,5 MB kleiner.

die Informationsdichte deutlich höher ist als bei 45×45 Depth Maps. Dies macht sich in einer geringeren Anfälligkeit für Banding-Artefakte bemerkbar.

Wie Ritschel et al. [RGKM07] festgestellt haben, wächst die Komprimierungsrate annähernd mit der Quadratwurzel der Anzahl komprimierter Depth Maps. Dieses Verhalten gilt auch, wenn für $\varphi_{\max}$ doppelt so viele Unterteilungen vorgenommen werden wie für $\theta_{\max}$. Allerdings lässt sich dann, im Vergleich zu einer gleichmäßigen Unterteilung, bei der hier vorgestellten Größenordnung bereits mit der Hälfte an komprimierten Depth Maps annähernd dieselbe Komprimierungsrate erzielen. Um dies nachzuvollziehen, können in Tabelle 5.1 die Komprimierungsraten unter 32×64 und 64×64 verglichen werden.

Da die Komprimierungsrate zunimmt und Banding-Artefakte gleichzeitig abnehmen, wurde sich, im Gegensatz zum Vorgehen von Ritschel et al. [RGKM07], in dieser Arbeit für eine non-uniforme Unterteilung beider Dimensionen der Polarkoordinaten entschieden. In diesem Fall kann die Iterationsfolge der Depth Maps bei der Komprimierung nicht mehr durch eine einzige Hilbert-Kurve festgelegt werden. Auf Kosten eines größeren Sprunges zwischen zwei der zu komprimierenden Depth Maps ist es zwar prinzipiell möglich durch zwei raumfüllende Hilbert-Kurven eine uniforme Diskretisierung der Umkugel zu erzielen, aufgrund der schwierigen Umsetzung der Hilbert-Kurve wurde sich in dieser Arbeit jedoch gegen dieses Vorgehen entschieden. Stattdessen werden die Depth Maps bei der Komprimierung in einer Zickzack-Reihenfolge abgearbeitet. Dabei bleibt θ konstant wenn φ geändert wird und φ bleibt konstant wenn θ inkrementiert wird. Dies wird erreicht, indem für jede zweite Erhöhung von θ, φ schrittweise erniedrigt wird, bis $\varphi = 0$ gilt. Daraufhin wird θ inkrementiert, wobei $\varphi = 0$ zunächst gültig bleibt. Erst in den nächsten Iterationen wird φ erhöht, solange $\varphi < 2\pi$ gilt, bevor θ wieder inkrementiert wird. Auf diese Weise lässt sich eine geringfügig höhere Kohärenz zwischen den Depth Maps erreichen, als dies der Fall wäre, wenn nach jeder Erhöhung von θ, φ wieder von 0 aufwärts inkrementiert würde. Wie sich die Zickzack-Iteration implementieren lässt, zeigt Listing 5.1 auf der nächsten Seite.

Listing 5.1: C++-Code für eine Zickzack-Iteration über 32×64 Depth Maps.

```
typedef unsigned int uint;
const float PI = 3.14159f;
const uint thetaDivs = 32;
const uint phiDivs = 64;
bool incPhi = true;

for( uint iTheta = 0; iTheta < thetaDivs; iTheta++, incPhi = !incPhi )
    for( uint iPhi = 0 + !incPhi; iPhi < phiDivs + !incPhi; iPhi++ )
    {
        float theta = ( iTheta / static_cast<float>(thetaDivs) ) * PI;
        float phi = ( iPhi / static_cast<float>(phiDivs) ) * 2.0f * PI;
        if( !incPhi )
            phi = 2.0f * PI - phi;

        setOrthographicCameraEyePt( theta, phi );
        renderFirstDepthLayer();
        renderSecondDepthLayer();
    }
```

Wird bei jedem θ eine konstante Anzahl von Unterteilungen für $\varphi_{\max}$ gewählt, kommt es zu dem Effekt, dass die berechneten Punkte mit größerer Entfernung zur Äquatorialebene zunehmend dichter beieinander liegen (vgl. Abbildung 5.2 auf der nächsten Seite). Dem lässt sich entgegenwirken, indem die Anzahl an Unterteilungen für $\varphi_{\max}$, durch eine Gewichtung mit $\sin\theta$, abhängig von θ gewählt, oder die Umkugel mit demselben Vorgehen, welches Sattler et al. [SSZK04] beschreiben, unterteilt wird. Die Menge an Speicher, die sich auf diese Weise einsparen lässt ist allerdings gering, da Depth Maps, die von nahe beieinander liegenden Kameras generiert werden, auch stärker komprimiert werden können. Außerdem erschwert ein solches Vorgehen die Ermittlung der Depth Map zu einer gegebenen Sample-Richtung zur Laufzeit. Wie in Abschnitt 5.3 beschrieben, kann dies bei einer konstanten Unterteilung für $\varphi_{\max}$ direkt durch einen Zugriff auf eine zweidimensionale Textur, deren Auflösung der Anzahl an Unterteilungen für $\theta_{\max}$ bzw. $\varphi_{\max}$ entspricht, anhand der Sample-Richtung in Polarkoordinaten, d.h. mit $(\theta/\pi, \varphi/2\pi)$, geschehen.

In dieser Arbeit wurde, außer zu Testzwecken, eine konstante Unterteilung von $\theta_{\max}$ bzw. $\varphi_{\max}$ vorgenommen. In diesem Fall ist die Kohärenz zwischen Depth Maps, die aus der Sicht von Kameras generiert werden, deren Augpunkte in der Nähe der beiden Pole der Umkugel positioniert sind, in der Regel höher. Bei der Komprimierung der Depth Maps werden diese in einer Zickzack-Iteration so abgearbeitet, dass für ein festes θ, φ iterativ erhöht, bzw. erniedrigt wird. Aus diesen beiden Beobachtungen lässt sich ableiten, dass auch die Ausrichtung des Objekts innerhalb der Umkugel die Komprimierungsrate beeinflusst. Es kann sich daher lohnen das Objekt so zu rotieren, dass die Kohärenz zwischen den Depth Maps möglichst hoch ist. Für das in Abbildung 5.2 gezeigte Modell eines Fahrzeugs wurde für 64×128 Depth Maps im linken Fall eine Komprimierungsrate von 12,6 : 1 und im deutlich günstigeren,

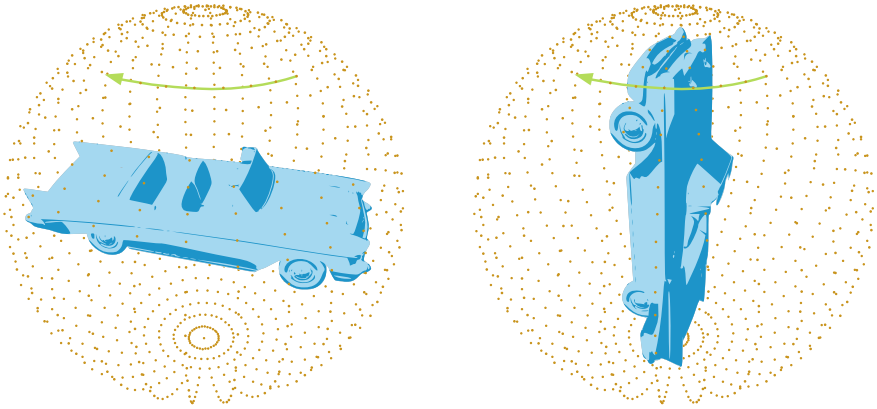


Abbildung 5.2: Die Ausrichtung des Objekts beeinflusst die Komprimierungsrate.

rechten Fall eine Komprimierungsrate von 16,6 : 1 erzielt. Da die Information, die in beiden Fällen komprimiert wird, jedoch nicht deckungsgleich ist, sollte nach einer Rotation des Modells bei der Komprimierung gesondert überprüft werden, ob zur Laufzeit zusätzliche Artefakte auftreten.

Eine andere Möglichkeit die Komprimierungsrate deutlich zu erhöhen besteht darin, Löcher in dem 3D-Modell nach Möglichkeit zu füllen.[1] Dies verhindert das in Abschnitt 4.2.1 erläuterte Problem, dass in manchen Fällen keine zweite Tiefenebene existiert, sodass die Komprimierung wirkungslos bleibt. Das in Tabelle 5.1 aufgeführte Bunny-Modell besitzt in Folge des 3D-Scanvorgangs mehrere Löcher auf der Unterseite. Werden diese Löcher gefüllt, steigt die Komprimierungsrate beispielsweise im Fall von 64×128 Depth Maps von 17,5 : 1 auf 30,5 : 1, wodurch der Speicherbedarf auf 8,4 MB sinkt.

Die Position des Kamera-Augpunkts wird in den beiden gängigen Graphikbibliotheken OpenGL und Direct3D üblicherweise in Form kartesischer Koordinaten gesetzt. Die Polarkoordinaten θ und φ lassen sich auf einfache Weise in kartesische Koordinaten umrechnen:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sin\theta \cdot \cos\varphi \\ \cos\theta \\ \sin\theta \cdot \sin\varphi \end{pmatrix} . \tag{5.1}$$

5.2.2 Realisierung des Depth Peelings

Die Komprimierung durch CSMs nutzt neben der Kohärenz zwischen zwei Depth Maps, die von ähnlich positionierten Kameras generiert wurden, auch die freie Wahl eines repräsentativen Tiefenwertes zwischen der ersten und der zweiten Tiefenebene. Um diese beiden Ebenen zu ermitteln, wird eine Technik namens *Depth Pee-*

[1] In dem 3D-Programm Maya werden Löcher in einem polygonalen Modell automatisch nach Aufruf des Befehls `polyCloseBorder`, der Teil der internen Skriptsprache MEL (*Maya Embedded Language*) ist, gefüllt.

ling [Eve01] eingesetzt, die ursprünglich entwickelt wurde um transparente Oberflächen korrekt darzustellen. Zur Ermittlung der ersten n Tiefenebenen aus Sicht der Kamera, sind beim Depth Peeling n Render Passes nötig. Auf Fragment-Ebene werden für jeden Render Pass die Tiefenebenen der vorherigen Passes ,,abgeschält".

Für eine Umsetzung des Depth Peelings kann Funktionalität der Graphikhardware genutzt werden, die auf die Unterstützung von Shadow Mapping zugeschnitten ist. Die Tiefenwerte eines Render Passes werden in eine Depth Map-Textur von der Größe des *Frame Buffers* kopiert, auf die im nächsten Render Pass zugegriffen werden kann, um zu überprüfen, ob ein Fragment näher oder genau so nah wie seine Vorgänger aus den vorangegangen Render Passes ist (in diesem Fall wird es verworfen), oder, ob es weiter entfernt ist (in diesem Fall gehört es zur gesuchten Tiefenebene und wird ausgegeben).

Mit der Methode `glCopyTexSubImage2D()` lässt sich bei einem Einsatz von OpenGL der *Depth Buffer* in eine Textur kopieren, wenn diese, unter Verwendung der Extension `GL_ARB_depth_texture`, als Depth Map initialisiert wurde. Der Vergleich der Tiefenwerte lässt sich mit der Extension `GL_EXT_shadow_funcs` realisieren. Im Gegensatz zur Extension `GL_ARB_shadow` erlaubt diese eine Überprüfung auf Tiefenwerte, die nicht nur größer oder gleich, sondern auch echt größer sind.

Mit Hilfe des Depth Peelings lassen sich nacheinander beliebig viele Tiefenebenen rendern. Damit der minimale und maximale Tiefenwert eines Fragments der CPU für die Komprimierung zur Verfügung steht, werden alle Pixel in jedem der beiden Render Passes mit der OpenGL-Methode `glReadPixels()` ausgelesen und in einem zweidimensionalen Array gespeichert.

5.2.3 Definition eines Dateiformats

Dateien ermöglichen es die vorberechneten Depth Maps beliebig oft wieder zu verwenden, ohne die aufwändige Komprimierung bei jedem Programmstart erneut durchführen zu müssen. Um ein Debugging zu vereinfachen, wurde sich dafür entschieden alle Daten in einer strukturierten und von Menschen lesbaren Form abzuspeichern. Dabei wurde das Regelwerk der Auszeichnungssprache XML[2] eingehalten, sodass es sich bei dem im Rahmen dieser Arbeit definierten CSM-Dateiformat um ein XML-Derivat handelt.

Jede CSM-Datei beginnt mit einer Dokumenttyp-Defintion (DTD), die es einem *Parser* erlaubt festzustellen, ob der Aufbau der Datei gültig ist. Listing 5.2 auf der nächsten Seite verdeutlicht anhand einer DTD die Struktur einer typischen CSM-Datei.[3] Die DTD definiert, wie Strukturelemente innerhalb der Datei ineinander verschachtelt werden dürfen und welche Attribute ein Strukturelement besitzen muss

[2] Die Spezifikationen zu XML (*Extensible Markup Language*) werden seit Ende der neunziger Jahre vom *World Wide Web Consortium* (W3C) herausgegeben.

[3] Um den Speicherbedarf der CSM-Dateien zu verringern, werden tatsächlich kürzere Elementnamen verwendet als die im Beispiel gezeigten. So wird `viewPoint` beispielsweise mit `vp` und `texel` mit `t` abgekürzt.

Listing 5.2: XML-Deklaration und Dokumenttyp-Definition einer CSM-Datei.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!-- Generated with CSM Creator 1.0 written by Stefan Müller -->
<!DOCTYPE csm [
    <!ELEMENT csm (view, texel+)>
    <!ATTLIST csm
      width    CDATA #REQUIRED      height   CDATA #REQUIRED
      scale    CDATA #IMPLIED       transX   CDATA #IMPLIED
      transY   CDATA #IMPLIED       transZ   CDATA #IMPLIED
      mesh     CDATA #REQUIRED      number   CDATA #REQUIRED
      frustum  CDATA #REQUIRED      >
        <!ELEMENT view (viewPoint)*>
            <!ELEMENT viewPoint (xPos, yPos, zPos)>
            <!ATTLIST viewPoint    i CDATA #REQUIRED>
                <!ELEMENT xPos (#CDATA)>
                <!ELEMENT yPos (#CDATA)>
                <!ELEMENT zPos (#CDATA)>
        <!ELEMENT texel (segment)+>
        <!ATTLIST texel    x CDATA #IMPLIED    y CDATA #IMPLIED>
            <!ELEMENT segment(zAvg, iEnd, lDist, rDist)>
                <!ELEMENT zAvg  (#CDATA)>
                <!ELEMENT iEnd  (#CDATA)>
                <!ELEMENT lDist (#CDATA)>
                <!ELEMENT rDist (#CDATA)>
]>
```

(#REQUIRED) oder kann (#IMPLIED). Ein gültiger Ausschnitt einer CSM-Datei sieht daher beispielsweise wie folgt aus:

```
<viewPoint i=``314``>
    <xPos>0.34</xPos>
    <yPos>0.64</yPos>
    <zPos>0.82</zPos>
</viewPoint>
```

Das Dokument-Element csm ist das äußerste Strukturelement einer CSM-Datei. Es besteht aus genau einem untergeordneten view-Element und mindestens einem untergeordneten texel-Element. Daneben besitzt es eine Reihe von Attributen mit Details zur Komprimierung, die auch bei einer Verwendung der Daten zur Laufzeit beachtet werden müssen. Diese umfassen die Auflösung der komprimierten Depth Maps (width, height), optionale Transformationen des Objekts vor der Komprimierung (scale, transX, transY, transZ), den Dateinamen des Objekts (mesh) sowie die Anzahl an komprimierten Depth Maps (number) und die Ausmaße des orthographischen Volumens (frustum), das bei der Erzeugung der Depth Maps für die Parallelprojektion des betrachteten Objekts herangezogen wird.

Jede auf die in Abschnitt 5.2.1 beschriebene Weise gesetzte Augpunkt-Position der orthographischen Kamera wird in Form kartesischer Koordinaten ebenfalls in der CSM-Datei gespeichert. Diese Positionen werden in einer Reihe von viewPoint-Elementen verwaltet, deren Anzahl dem Wert des number-Attributs, das Teil des csm-Elements ist, entspricht.

Die komprimierte Information der Depth Maps wird in den `texel`-Elementen festgehalten. Diese Strukturelemente entsprechen dem, was Ritschel et al. [RGKM07] als *Segment List* bezeichnen. Da alle Texel der Depth Maps sequentiell, in einer dem Parser bekannten Reihenfolge (von unten links nach oben rechts) aufgeführt sind und die Auflösung der Depth Maps aus dem `width`- und `height`-Attribut des `csm`-Elements ausgelesen wird, können die beiden Attribute zur Position des Texels (`x`, `y`) entfallen. Jede Segmentliste enthält eine Reihe von Segmenten, die wiederum jeweils einen mittleren Tiefenwert und die Nummer der Depth Map, bis zu welcher der Tiefenwert gültig ist, verwalten. Bei letzterem handelt es sich um das `iEnd`-Element, dessen Inhalt sich auf das Attribut `i` des `viewPoint`-Elements bezieht und maximal den Wert des Attributs `number` im `csm`-Element annimmt. Darüber hinaus besteht jedes `segment`-Element aus den beiden untergeordneten Strukturelementen `lDist` und `rDist`, welche die Anzahl benachbarter Segmente bis zum Start bzw. Ende der Segmentliste beinhalten – eine Information, die wie in Abschnitt 5.3.2 beschrieben, das Auffinden eines gesuchten Segments robuster gestaltet.

Die auf die soeben beschriebene Weise aufgebauten CSM-Dateien benötigen in der Regel eine große Menge Festplatten-Speicher. Beispielsweise handelt es sich für das Bunny-Modell bei 45×90 Depth Maps, einer Genauigkeit der Kamera-Augpunkt-Positionen auf zehn Nachkommastellen und mit 16 Bit kodierten mittleren Tiefenwerten um 165 MB. Davon beansprucht beispielsweise die komprimierte Tiefeninformation lediglich 8,5 MB (vgl. Tabelle 5.1 auf Seite 45). Für eine bessere Lesbarkeit sind die CSM-Dateien jedoch mit einer Vielzahl von Tabulator- und Zeilenumbruchzeichen formatiert, die zu dem hohen Speicherbedarf beitragen. Die Größe der Dateien lässt sich durch Komprimierung sehr stark herabsetzen. Im ZIP-Dateiformat sinkt der benötigte Speicher für CSM-Dateien um beinahe 90% – in diesem Fall benötigt die CSM-Datei zum Bunny-Modell lediglich noch 19,3 MB.

Die Einführung des CSM-Dateiformats bietet den Vorteil, dass die Komprimierung der Depth Maps und die Anwendung der komprimierten Daten, die nicht ausschließlich auf die Berechnung von Ambient Occlusion beschränkt ist, völlig unabhängig voneinander stattfinden kann – alle benötigten Informationen um die Verdeckung durch ein Objekt zu berechnen sind in der zugehörigen CSM-Datei vorhanden.

5.3 Laufzeit

Während der Programmlaufzeit wird zur Berechnung von Ambient Occlusion für jeden darzustellenden Punkt ermittelt, wie groß der Anteil des Umgebungslichts ist, der zu seiner Beleuchtung beiträgt. Um dies zu erreichen muss zuvor zu jedem potentiellen Occluder die entsprechende CSM-Datei ausgelesen werden. Für mehrere Instanzen eines Modells werden jedoch nur die Daten einer CSM-Datei benötigt. Aus den ausgelesenen Daten werden eine Reihe von Texturen generiert, die den gezielten Zugriff auf die vorgespeicherten Tiefenwerte ermöglichen und damit die Berechnung der Visibilität.

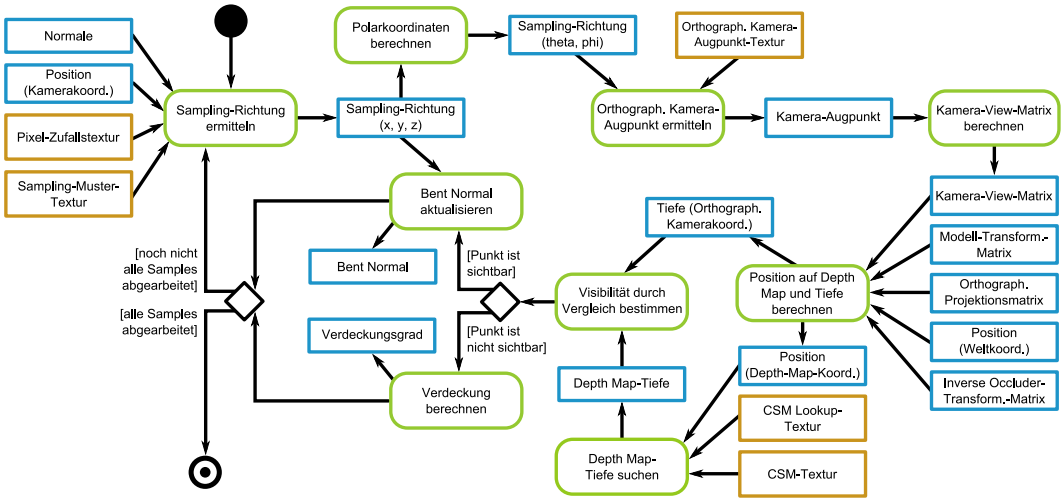
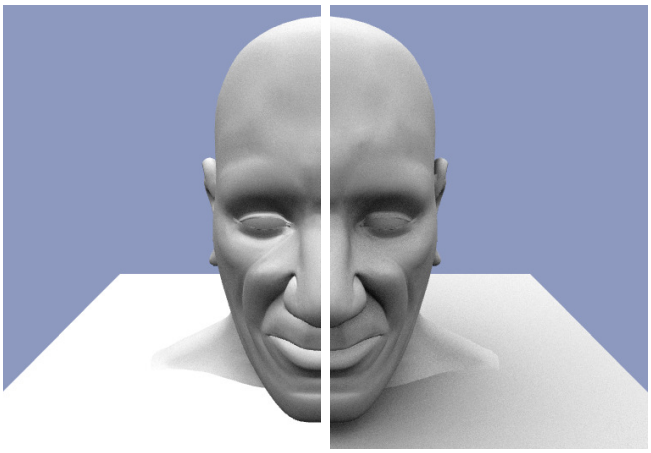


Abbildung 5.3: Aktivitätsdiagramm zur Berechnung von Ambient Occlusion mit CSMs.

Abbildung 5.3 zeigt anhand eines Aktivitätsdiagramms die Folge an Operationen, die für jeden Punkt von der Graphikhardware ausgeführt werden und die daraus resultierenden Daten. Letztere werden in Form von Rechtecken dargestellt, wobei Rot aus den Daten einer CSM-Datei vorberechnete Texturen und Blau skalare Werte, Vektoren und Matrizen symbolisiert.

Zunächst wird die Position eines Fragments aus Sicht der Kamera, die das Bild rendert, derart skaliert, dass mit den Koordinaten auf eine Textur zugegriffen werden kann – konkret heißt dies, dass wie beim Shadow Mapping der Wertebereich des Clip Space (von -1 bis 1 für die X- und die Y-Achse) auf Texturkoordinaten (von 0 bis 1) abgebildet wird. Die besagte Textur enthält $(0, 1)$-verteilte Zufallswerte und ihre Größe entspricht der des Frame Buffers. Auf diese Weise lässt sich für jedes Pixel eine zufällige Zahl ermitteln, die bestimmt in welcher Zeile auf die Sampling-Muster-Textur zugegriffen wird. Auf den genauen Aufbau dieser Textur wird in Abschnitt 5.3.1 eingegangen, im Moment genügt für das Verständnis die Feststellung, dass die Sampling-Muster-Textur eine Richtung liefert, für die ermittelt werden soll, ob es einen Occluder gibt.

Handelt es sich bei dem untersuchten Objekt selbst um einen Occluder, kann an dieser Stelle überprüft werden, ob die soeben ermittelte Sample-Richtung die Bodenebene schneidet. In diesem Fall darf die Visibilität ohne weitere Berechnungen für dieses Sample gleich Null gesetzt werden. Dies ist möglich, da beim Ambient Occlusion nur die Beleuchtung durch die obere Hemisphäre untersucht wird. Um einen frei beweglichen Occluder zu unterstützen, muss jedes Sample vor dieser Überprüfung mit der transponierten Inversen der Transformationsmatrix multipliziert werden. Obwohl die Bodenebene in diesem Fall das Licht abschirmt, ist es nicht nötig sie wie die übrigen Occluder zu behandeln, d.h. es wird keine CSM-Datei benötigt. Abbildung 5.4 verdeutlicht, dass sich auf diese Weise für viele Samples ein



(a) (b)

Abbildung 5.4: Vergleich zwischen (a) Verschattung durch Bodenebene und (b) zusätzlicher Verschattung durch Kopf-Modell. Pro Fragment wurden jeweils 81 Samples erzeugt.

genauer Visibilitätstest einsparen lässt. Manche Bereiche, wie die Stirn des gezeigten Modells, liefern auch nach zusätzlichen Sichtbarkeitstests keinen Unterschied. Für andere Bereiche, wie Augen, Ohren und Mund des dargestellten Modells ist eine Untersuchung der Visibilität entlang der Sample-Richtung unerlässlich, da es hier zur Selbstverschattung kommt.

Für alle Samples, die in den oberen Halbraum zeigen, muss daraufhin die Position der orthographischen Kamera und die zugehörige Depth Map gefunden werden, die am ehesten der Sample-Richtung entsprechen. Wie in Abschnitt 5.2.3 erwähnt, speichert die CSM-Datei sämtliche Positionen, die bei der Erzeugung der Depth Maps als Augpunkt für die orthographische Kamera gewählt wurden. Aus dieser Information ist es möglich, eine Textur zu erzeugen, welche im Rot-, Grün- und Blaukanal die Augpunktkoordinaten sowie eine Identifikationsnummer der entsprechenden Depth Maps im Alphakanal verwaltet. Um direkt mit der Sample-Richtung auf diese Textur zuzugreifen, müsste sie als Cube Map vorliegen. Aus den Augpunktkoordinaten eine Cube Map zu erstellen ist jedoch nicht trivial. Diese müsste genügend hoch aufgelöst sein, um die nahe beieinander liegenden Augpunkte an den beiden Polen der Umkugel zu berücksichtigen und würde dafür an anderen Stellen redundante Informationen enthalten. Die Alternative, nach der in dieser Arbeit vorgegangen wurde, besteht darin mit den Gleichungen 5.2 und 5.3 die kartesischen Koordinaten der Sample-Richtung in Polarkoordinaten umzurechnen und mit diesen auf eine zweidimensionale Textur zuzugreifen:

$$\theta \quad = \quad \arccos\left(\frac{y}{\sqrt{x^2+y^2+z^2}}\right) \; ; \tag{5.2}$$

$$\varphi = \begin{cases} \arccos\left(\dfrac{x}{\sqrt{x^2+y^2+z^2}\sin\theta}\right), & \text{für } z \geq 0 \\ -\arccos\left(\dfrac{x}{\sqrt{x^2+y^2+z^2}\sin\theta}\right), & \text{für } z < 0\ . \end{cases} \tag{5.3}$$

Nachdem θ und φ durch π bzw. 2π dividiert wurden, liegen sie in einem Bereich zwischen 0 und 1 und können für den Zugriff auf die Kamera-Augpunkt-Textur genutzt werden. Um diese Berechnung nicht für jedes Sample durchführen zu müssen, wurde bei der Implementierung der Versuch unternommen, eine im Voraus berechnete Cube Map einzusetzen, die kartesische Koordinaten auf Polarkoordinaten abbildet.[4] Dabei zeigte sich, dass eine Auflösung von 256×256 Texeln bei zwei Farbkanälen mit jeweils 32 Bit Genauigkeit, zusammen mit dem Einsatz bi-linearer Filterung beim Texturzugriff, im Ergebnis einen kaum wahrnehmbaren Fehler liefert. Eine Übersetzung des Shader-Codes in Assembler legt nahe, dass mit diesem Vorgehen gegenüber der expliziten Umwandlung in Polarkoordinaten 43 Instruktionen eingespart werden. Für die gesamte Berechnung von Ambient Occlusion durch CSMs entspricht dies einer Abnahme der Anweisungen von rund 20%, welche ansonsten für jedes Sample, das in den oberen Halbraum zeigt, ausgeführt werden würden. Es zeigte sich allerdings auch, dass aufgrund der hohen Auflösung der Cube Map mit diesem Vorgehen die Performanz nicht deutlich erhöht werden kann. Erst ab einer Auflösung von 128×128 Texeln und weniger zeichnete sich auf der Hardware, die für die Tests zum Einsatz kam, ein Performanzgewinn ab. Da mit einer niedrigeren Auflösung allerdings eine zunehmende Ungenauigkeit bei der Umrechnung in Polarkoordinaten einhergeht, wurde in dieser Arbeit letztlich auf die Cube Map verzichtet.

Die Graphikhardware bestimmt beim Zugriff auf die Kamera-Augpunkt-Textur welcher Eintrag am ehesten dem (θ, φ)-Paar der Sample-Richtung entspricht. Die Textur liefert daraufhin die Position des Kamera-Augpunktes in kartesischen Koordinaten und eine Nummer, welche die auszuwertende Depth Map identifiziert. Die normalisierten Koordinaten des Augpunktes entsprechen der Blickrichtung $\mathbf{z}$ der orthographischen Kamera. Aus dieser und dem Oben-Vektor (*up vector*) $\mathbf{o}$ des Weltkoordinatensystems lassen sich die übrigen beiden Basisvektoren des Kamerakoordinatensystems berechnen:

$$\begin{aligned} \mathbf{x} &= \frac{\mathbf{o} \times \mathbf{z}}{|\mathbf{o} \times \mathbf{z}|}\ ; \\ \mathbf{y} &= \mathbf{z} \times \mathbf{x}\ . \end{aligned} \tag{5.4}$$

Sobald mit Hilfe der Basisvektoren die View-Matrix aufgestellt ist, kann die Position des darzustellenden Punktes in das Koordinatensystem der Kamera transformiert werden, auf welche das gewählte Sample zeigt. Die dafür benötigten Matrixmultiplikationen werden in gleicher Form auch beim Shadow Mapping eingesetzt (vgl.

[4] Auf ähnliche Weise kann eine Cube Map auch zur Normalisierung von Vektoren eingesetzt werden. Dies lohnt sich, da ein einzelner Texturzugriff häufig effizienter ist als die Normalisierungsoperation der Shader-Sprachen [FK03].

Gleichung 2.3 auf Seite 9).[5] Die Projektionsmatrix der orthographischen Kameras ist die gleiche, die auch bei der Erzeugung der Depth Maps zur Vorverarbeitung eingesetzt wird. Die View-Matrix der Kamera errechnet sich aus den Basisvektoren **x**, **y** und **z**. Der Inhalt der Transformationsmatrix hängt davon ab, welchem Objekt das momentan untersuchte Fragment zugeordnet ist. Handelt es sich bei dem Objekt nicht um den auf Visibilität zu überprüfenden Occluder und soll dieser beliebig transformiert werden dürfen, muss der Punkt nach Multiplikation mit den drei genannten Matrizen zusätzlich mit der inversen Transformationsmatrix des Occluders multipliziert werden. Dies erklärt sich dadurch, dass jede Depth Map eine bestimmte Sicht auf den Occluder verwaltet und daher alle Depth Maps stets analog zum Occluder transformiert werden müssen.

Wenn der Punkt in das Koordinatensystem der orthographischen Kamera überführt worden ist, ergibt sich nach der perspektivischen Division die Position des Punktes auf der Depth Map und seine Tiefe aus Sicht der Kamera. Um zu überprüfen, ob das Fragment aus der Sample-Richtung beleuchtet wird, wird dieser Tiefenwert mit der vorberechneten und komprimierten Tiefe verglichen. Die Position auf der Depth Map ist zwar bekannt, doch der Tiefenwert, der für die gesuchte Depth Map gültig ist, muss innerhalb der CSM-Textur gesucht werden. Wie bei dieser Suche vorgegangen wird, beschreibt Abschnitt 5.3.2.

Ist die vorberechnete Tiefe kleiner als die tatsächliche Tiefe des untersuchten Punktes, so liegt dieser im Schatten und der Grad der Verdeckung kann um $1/N$ erhöht werden – wobei N die Anzahl der Samples bezeichnet. Da die Samples, wie im nächsten Abschnitt erläutert, mit einer kosinusgewichteten Dichte erzeugt werden, ist eine Verrechnung mit dem Skalarprodukt von Sample-Richtung und Normale nicht nötig. Liegt der untersuchte Bildpunkt dagegen nicht im Schatten, kann die Sample-Richtung zur Berechnung der Bent Normal herangezogen werden. Dazu werden alle unverdeckten Sample-Vektoren auf die Bent Normal aufaddiert und diese nach Abarbeitung aller Samples normalisiert. Die weitere Verwendung der Bent Normal wird in Abschnitt 5.3.4 erläutert.

Alle in diesem Abschnitt aufgeführten Schritte werden für jedes Fragment solange wiederholt bis die gewünschte Anzahl an Samples abgearbeitet wurde.

5.3.1 Strategien für ein Sampling

Dieser Abschnitt befasst sich mit der Erzeugung der Sample-Richtungen. Diese werden im Voraus berechnet und in einer Textur gespeichert. Die Generierung repräsentativer Samples kann stochastisch oder deterministisch geschehen. Wie dabei im einzelnen vorgegangen wird und wo die jeweiligen Vor- und Nachteile liegen, wird in den folgenden Teilabschnitten erläutert. Im Rahmen dieser Arbeit wurden beide Möglichkeiten zur Erzeugung von Samples umgesetzt. Der Benutzer kann die

[5] Da die Erstellung der View-Matrix auf einem Texturzugriff basiert und daher erst im Fragment Program stattfindet, werden, im Gegensatz zum üblichen Vorgehen beim Shadow Mapping, nicht die Eckpunkte, sondern die aus den Eckpunkten interpolierten Positionen der untersuchten Punkte in das Kamerakoordinatensystem transformiert.

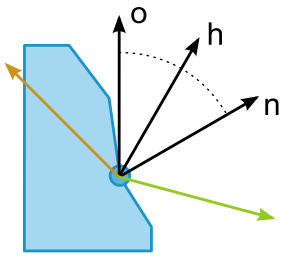


Abbildung 5.5: Die Sample-Richtung (grün) wird durch Spiegelung der vorgespeicherten Richtung (rot) an **h** ermittelt.

Sampling-Strategie zur Laufzeit stets über die Benutzerschnittstelle, auf die in Abschnitt 5.4.3 eingegangen wird, wechseln.

Die Sample-Richtungen werden von Polarkoordinaten ausgehend generiert. Diese kommen auch bei der Integration über die Hemisphäre zur Berechnung von Ambient Occlusion zum Einsatz (vgl. Gleichung 3.1 auf Seite 16). Die geschickte Verteilung von Samples dient, wie im nächsten Teilabschnitt beschrieben, der Approximation dieses Integrals über den oberen Halbraum. Zur Laufzeit müssen die Sample-Richtungen in kartesischen Koordinaten vorliegen. Aus diesem Grund wird jedes Sample gleich nach seiner Erzeugung mit Gleichung 5.1 auf Seite 48 umgerechnet. Das Ergebnis dieser Rechnung wird im Rot-, Grün-, und Blaukanal der zweidimensionalen Sampling-Muster-Textur gespeichert.

Da die ermittelte Sample-Richtung stets für die Hemisphäre in Weltkoordinaten gespeichert wird, wird sie so umgerechnet, dass sie für den oberen Halbraum des untersuchten Punktes gilt. Dazu wird hier wie in Abbildung 5.5 verdeutlicht vorgegangen. Es wird der Vektor **h** berechnet, der zwischen der Fragment-Normalen **n** und dem Oben-Vektor $\mathbf{o} = (0, 1, 0)^T$ liegt. An diesem Vektor lässt sich die vorberechnete Sample-Richtung spiegeln um die Richtung zu ermitteln, die für den Punkt repräsentativ ist. Die HLSL-Funktion[6] `reflect()` berechnet eine solche Spiegelung auf effiziente Weise.

Monte-Carlo-Integration

Die mathematische Grundlage für ein Sampling ist die Monte-Carlo-Integration. Das Integral einer Funktion g lässt sich als Erwartungswert E darstellen und damit durch die Generierung von N Zufallszahlen x_i im Integrationsbereich $[a, b]$ mit einer Dichte p schätzen. Für diese Zufallszahlen wird die Funktion ausgewertet und das Ergebnis aufsummiert. Diese Summe wird schließlich zur Schätzung des Integrals durch die Anzahl der Samples geteilt:

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

[6] HLSL (*High-Level Shader Language*) ist Teil der DirectX-Programmierschnittstelle. Die Sprache wurde von Microsoft in Zusammenarbeit mit NVIDIA entwickelt und weist aus diesem Grund große Ähnlichkeit zu Cg (*C for graphics*) auf [FK03].

In der Regel soll das Integral einer einzelnen Funktion approximiert werden. Durch Substitution $f = gp$, ergibt sich die gebräuchlichere Darstellung:

$$\int_a^b f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)} \; . \tag{5.5}$$

Die Monte-Carlo-Integration kann nach dem gleichen Prinzip auf mehrere Dimensionen angewandt werden. Im zweidimensionalen Fall sieht die Schätzung wie folgt aus:

$$\int_a^b \int_c^d f(x,y)dxdy \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i,y_i)}{p(x_i,y_i)} \; . \tag{5.6}$$

Die Schätzung durch Monte-Carlo-Integration ist erwartungstreu und konvergiert mit einer Rate von $1/\sqrt{N}$ zum tatsächlichen Ergebnis. Um die Varianz zu reduzieren und damit mit weniger Samples zu einem optisch ansprechenderen Ergebnis zu gelangen, existieren verschiedene Strategien. Drei dieser Strategien, die in dieser Arbeit zum Einsatz kommen, sind das *Importance Sampling*, das *Stratified Sampling* und *Quasi-Monte-Carlo-Techniken.*

Importance Sampling

Auch wenn die Dichte mit der bei der Monte-Carlo-Integration alle Zufallszahlen erzeugt werden prinzipiell beliebig gewählt werden kann, beeinflusst sie doch direkt die Varianz. Beim Importance Sampling wird die Dichte p möglichst ähnlich zu der zu integrierenden Funktion f gewählt, sodass in den ausschlaggebenden Bereichen mehr Samples erzeugt werden und die Varianz gegen 0 geht.

Mit Hilfe der Monte-Carlo-Integration kann die Lösung der in Abschnitt 3.2 vorgestellten Gleichung für Ambient Occlusion angenähert werden. Dazu werden gemäß dem Importance Sampling für jedes Fragment mehrere Samples mit einer kosinusgewichteten Dichte p erzeugt:

$$p(\theta,\varphi) = \frac{1}{\pi}\cos\theta \; . \tag{5.7}$$

Die Ergebnisse der Visibilitätstests aller mit dieser Dichte generierten Samples werden aufsummiert und durch die Anzahl an Samples geteilt, um ein Ergebnis für Ambient Occlusion zu erhalten.

Um Samples mit einer gegebenen Dichte aus zwei $(0,1)$-verteilten Zufallszahlen[7] ξ_1 und ξ_2 zu erzeugen, kann die Inverse CDF-Methode[8] eingesetzt werden. Dazu wird zunächst die Verteilungsfunktion F bestimmt, die sich durch Integration der Dichtefunktion p ergibt:

$$F(x,y) = \int_{y_{\min}}^{y}\int_{x_{\min}}^{x} p(x',y')dx'dy' \; .$$

[7] Viele Programmiersprachen bieten einfache Mechanismen zur Erzeugung uniform verteilter Zufallszahlen. In C++ ist diese Funktionalität mit der in `<stdlib.h>` definierten Methode `rand` umgesetzt. Eine Division durch `RAND_MAX` führt dazu, dass das Ergebnis zwischen 0 und 1 liegt.

[8] CDF steht für *cumulative distribution function*, die englische Bezeichnung einer Verteilungsfunktion.

Die Verteilungsfunktion F liefert für Zufallsvariablen mit der Dichte p zufällige Werte zwischen 0 und 1 mit uniformer Wahrscheinlichkeit. Im zweidimensionalen Fall gilt:

$$\begin{aligned} \xi_1 &= F(x, y_{\max}) \; ; \\ \xi_2 &= \frac{F(\xi_1, y)}{F(\xi_1, y_{\max})} \; . \end{aligned} \tag{5.8}$$

Durch Invertierung der Verteilungsfunktion lassen sich umgekehrt aus zwei $(0,1)$-verteilten Zufallszahlen ξ_1 und ξ_2, Zufallsvariablen mit einer Dichte p erzeugen.

Um anhand der Inversen CDF-Methode Zufallsvariablen mit der konkret durch Gleichung 5.7 gegebenen Dichte zu erzeugen, muss die zugehörige Verteilungsfunktion berechnet werden:

$$\begin{aligned} F &= \frac{1}{\pi} \int \cos\theta d\omega \; ; \\ F(\theta, \varphi) &= \frac{1}{\pi} \int_0^{\varphi} \int_0^{\theta} \cos\theta' \sin\theta' d\theta' d\varphi' \\ &= \frac{1}{\pi} \int_0^{\varphi} d\varphi' \int_0^{\theta} \cos\theta' \sin\theta' d\theta' \\ &= \frac{\varphi}{\pi} \left[-\frac{\cos^2\theta'}{2} \right]_0^{\theta} \\ &= \frac{\varphi}{2\pi} \left(1 - \cos^2\theta \right) \; . \end{aligned}$$

Diese Verteilungsfunktion liefert mit den in Gleichung 5.8 gezeigten Parametern zwei $(0,1)$-verteilte Zufallszahlen ξ_1 und ξ_2:

$$\begin{aligned} \xi_1 &= F(\theta, \varphi_{\max}) \\ &= 1 - \cos^2\theta \; ; \end{aligned}$$

$$\begin{aligned} \xi_2 &= \frac{F(\xi_1, \varphi)}{F(\xi_1, \varphi_{\max})} \\ &= \frac{\varphi \left(1 - \cos^2 \xi_1 \right)}{2\pi} \cdot \frac{2\pi}{2\pi \left(1 - \cos^2 \xi_1 \right)} \\ &= \frac{\varphi}{2\pi} \; . \end{aligned}$$

Durch Umkehrung der beiden obigen Gleichungen lassen sich aus ξ_1 und ξ_2 zufällige Werte für θ und φ mit der durch Gleichung 5.7 vorgegebenen Dichte generieren:

$$\begin{aligned} \theta &= \arccos(\sqrt{1 - \xi_1}) \; ; \\ \varphi &= 2\pi \cdot \xi_2 \; . \end{aligned} \tag{5.9}$$

Anstatt die mit der obigen Gleichung berechnete Sample-Richtung in Polarkoordinaten separat mit Gleichung 5.1 auf Seite 48 in kartesische Koordinaten umzurechnen,

können beide Gleichungen zur Reduzierung der benötigten trigonometrischen Funktionen auch auf die folgende Weise kombiniert werden [Shi02]:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sqrt{\xi_1}\cos(2\pi\xi_2) \\ \sqrt{1-\xi_1} \\ \sqrt{\xi_1}\sin(2\pi\xi_2) \end{pmatrix} .$$

Stochastisches Stratified Sampling

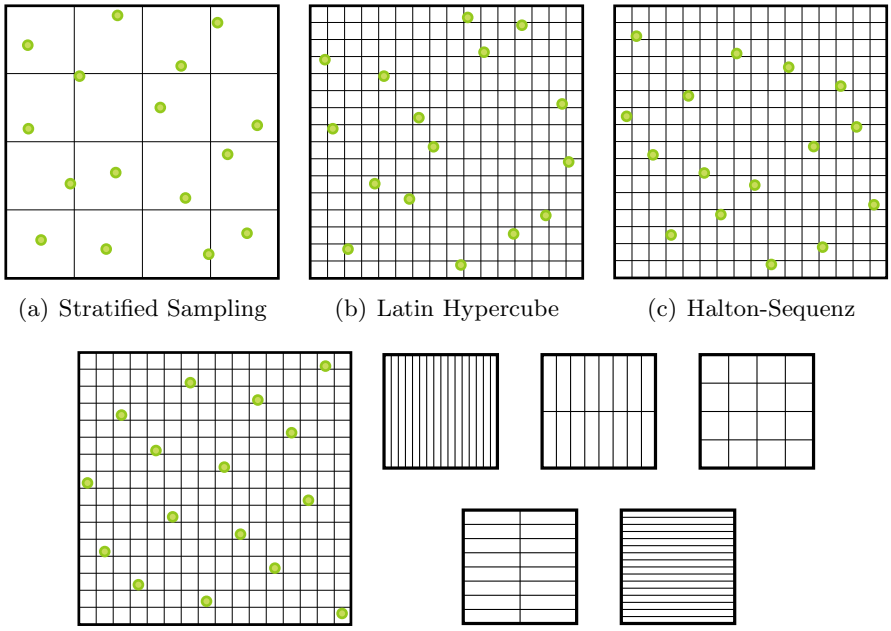
Die Strategie des Stratified Sampling zielt auf das Problem ab, dass sich Samples unabhängig von der Dichte ungünstig im Integrationsbereich verteilen können und somit das Integral nur schlecht approximieren. Um dies zu verhindern wird der Integrationsbereich in m disjunkte Strata aufgeteilt, deren Integrale separat geschätzt werden:

$$\int_a^b f(x)dx = \int_a^{\alpha_1} f(x)dx + \int_{\alpha_1}^{\alpha_2} f(x)dx + \ldots + \int_{\alpha_{m-2}}^{\alpha_{m-1}} f(x)dx + \int_{\alpha_{m-1}}^{b} f(x)dx .$$

Es kann gezeigt werden, dass die Varianz durch Stratified Sampling niemals höher ist als ohne die Anwendung dieser Strategie [Shi02] und, dass es zur Reduzierung der Varianz zudem stets vorteilhafter ist, den Integrationsbereich weiter gleichmäßig zu unterteilen, als mehr als ein Sample innerhalb eines Stratums zu generieren [DBB06].

Importance Sampling lässt sich auf einfache Weise mit Stratified Sampling kombinieren. Dafür können die beiden $(0, 1)$-verteilten Zufallszahlen ξ_1 und ξ_2 innerhalb von festgelegten Strata uniformer Größe erzeugt werden. Anschließend werden diese mit der inversen Verteilungsfunktion, gegeben durch Gleichung 5.9 auf der vorherigen Seite, in die gesuchten Zufallszahlen θ und φ transformiert. Um sicherzustellen, dass ξ_1 und ξ_2 nur innerhalb eines Stratums generiert werden, wurden in dieser Arbeit so lange Samples im Bereich $(0, 1)$ erzeugt und verworfen, bis das erste Sample im gewählten Stratum liegt. Da alle Sample-Richtungen im Voraus berechnet und in einer Textur gespeichert werden, wirkt sich dieses wenig performante Vorgehen nur auf die Vorberechnungszeit der Anwendung aus.

Da sowohl ξ_1 als auch ξ_2 innerhalb von Strata erzeugt werden, lässt sich der Integrationsbereich bei N Sample-Paaren (ξ_1, ξ_2) in gerade einmal $\sqrt{N}$ Strata für jede der beiden Dimensionen zerlegen, wenn sich in jedem Stratum genau ein Sample befinden soll (vgl. Abbildung 5.6(a) auf der nächsten Seite). Der *Latin Hypercube*-Algorithmus (auch *N-Rooks* genannt) geht dagegen so vor, dass jede Dimension in N Unterintervalle aufgeteilt wird und die Samples derart verteilt werden, dass sich in jedem Unterintervall genau ein Sample befindet. Für den in Abbildung 5.6(b) gezeigten, zweidimensionalen Fall bedeutet dies, dass in jeder Reihe und jeder Spalte nicht mehr und auch nicht weniger als ein Sample vorhanden sein darf. In der Praxis reduziert sich mit diesem Vorgehen oft die Varianz der Schätzung im Vergleich zum reinen Stratified Sampling [KK02b].



(a) Stratified Sampling (b) Latin Hypercube (c) Halton-Sequenz

(d) Hammersley-Sequenz (e) Intervalle der Hammersley-Sequenz

Abbildung 5.6: Sampling-Strategien im Vergleich mit jeweils 16 Samples.

Quasi-Monte-Carlo-Techniken

Auch Quasi-Monte-Carlo-Techniken haben das Ziel die Varianz der Schätzung bei der Monte-Carlo-Integration durch eine möglichst gute Verteilung der Samples zu verringern und stellen damit eine Alternative zum stochastischen Stratified Sampling dar. Der Unterschied besteht darin, dass keine Strata festgelegt werden, sondern die Samples direkt anhand deterministischer Sequenzen so gleichmäßig wie möglich verteilt werden. Das Qualitätsmaß für eine gute Verteilung ist eine niedrige Diskrepanz. Aus diesem Grund werden diese Sequenzen auch *Low-Discrepancy Sequences* (LDS) genannt. Es gibt eine Vielzahl solcher Sequenzen – beispielsweise Halton, Hammersley, Sobol, Larcher und Pillichshammer, sowie Niederreiter –, deren Umsetzung und Bewertung über den Rahmen dieser Arbeit hinaus gehen würde. Hier wurden daher mit Halton und Hammersley nur die beiden von Dutré et al. [DBB06] erläuterten Sequenzen in das implementierte System integriert.

Die Grundlage zur Bestimmung der Halton- und der Hammersley-Sequenz ist die Umrechnung einer Zahl k, die im dekadischen Zahlensystem gegeben ist, in die entsprechende Zahl im Zahlensystem zur Basis b. Für $b = 2$ ist dies das in der Informatik häufig zum Einsatz kommende duale Zahlensystem – hier gilt für k:

$$\begin{aligned} 1 &= [1]_2 \\ 2 &= [10]_2 \\ 3 &= [11]_2 \\ &\vdots \end{aligned}$$

Die radikal inverse Funktion Φ spiegelt k zur Basis b am Komma. Für das Beispiel $b = 2$ sieht diese Spiegelung so aus:

$$\begin{aligned} [,1]_2 &= 1/2 \\ [,01]_2 &= 1/4 \\ [,11]_2 &= 1/2 + 1/4 \\ &\vdots \end{aligned}$$

$\Phi_b(k)$ mit $b = 2$ liefert damit die Sequenz

$$(0{,}5,\ 0{,}25,\ 0{,}75,\ 0{,}125,\ 0{,}625,\ \ldots) .$$

An diesem Beispiel wird die gute Verteilung von Werten zwischen 0 und 1 deutlich, die sich die Halton- und die Hammersley-Sequenz zunutze machen. Für eine Verteilung in mehreren Dimensionen wird für jede Dimension eine andere Basis gewählt und, wie gerade an der Basis $b = 2$ demonstriert, die radikal inverse Funktion berechnet.

Der k-te Wert der d-dimensionalen Halton-Sequenz berechnet sich mit:

$$x_k = (\Phi_{b_1}(k),\ \Phi_{b_2}(k),\ \Phi_{b_3}(k),\ \ldots,\ \Phi_{b_d}(k)) , \tag{5.10}$$

dabei werden als Basen die ersten d Primzahlen gewählt (d.h. $b_1 = 2$, $b_2 = 3$, $b_3 = 5$, ...).

Um die Hammersley-Sequenz zu berechnen, muss die gesamte Anzahl an Samples (N) schon im Voraus bekannt sein. Der k-te Wert der d-dimensionalen Hammersley-Sequenz berechnet sich mit:

$$x_k = (\frac{k}{N},\ \Phi_{b_1}(k),\ \Phi_{b_2}(k),\ \ldots,\ \Phi_{b_{d-1}}(k)) , \tag{5.11}$$

dabei werden als Basen die ersten $d - 1$ Primzahlen gewählt.

Abbildung 5.6(c) und Abbildung 5.6(d) zeigen die Verteilung von 16 Samples durch die zweidimensionale Halton- bzw. Hammersley-Sequenz. Die Hammersley-Sequenz genügt sowohl den Einschränkungen des Stratified Sampling als auch denen des Latin Hypercube Sampling. Darüber hinaus lassen sich, wie Abbildung 5.6(e) zeigt, zwei weitere Arten von elementaren Intervallen auf die Verteilung auflegen, für welche gilt, dass sich genau ein Sample in jedem Unterintervall befindet.

Um Ambient Occlusion zu berechnen, könnten N Wertepaare anhand einer der beiden gerade vorgestellten zweidimensionalen Sequenzen ermittelt und diese jeweils mit der inversen Verteilungsfunktion (Gleichung 5.9) nach θ und φ transformiert werden. Der k-te Wert der Halton- bzw. Hammersley-Sequenz würde dann die beiden Zufallszahlen ξ_1 und ξ_2 ersetzen. Das Ergebnis wäre in diesem Fall völlig frei von dem für stochastische Vorgehensweisen typischen Rauschen, würde jedoch Aliasing-Artefakte aufweisen, auf die das visuelle System des Menschen wesentlich empfindli-

cher reagiert [Coo86]. Eine Alternative besteht darin, die deterministisch ermittelten Sequenzen zufällig derart zu vermischen, dass wieder ein vom menschlichen Sehapparat in größerem Maße toleriertes Rauschen an die Stelle der Aliasing-Artefakte tritt und, wie beim stochastischen Sampling, das zu berechnende Integral erwartungstreu geschätzt werden kann. Dieses Vorgehen ist als *Randomized Quasi-Monte Carlo Integration* [KK02a] bekannt und sorgt für eine schnellere Konvergenz zum erwarteten Ergebnis als etwa Stratified oder Latin Hypercube Sampling. Durch mehrere dieser zufälligen Vermischungen für jede der beiden eingesetzten Sequenzen entstehen, wie auch beim stochastischen Sampling, eine Reihe von Sampling-Mustern, die vorberechnet und in einer Textur gespeichert werden. Auf diese Weise ist die Wahrscheinlichkeit hoch, dass zwei benachbarte Pixel ein unterschiedliches Sampling-Muster einsetzen, und die durch das deterministische Sampling hervorgerufenen Aliasing-Artefakte werden vermieden.

Um M unterschiedliche Sampling-Muster mit jeweils N Samples zu generieren kann zur Vermischung einer zweidimensionalen Halton- oder Hammersley-Sequenz ein zufälliger Verschiebungsvektor $v_j = (\xi_{1_j}, \xi_{2_j})$ ermittelt und mit jedem Element x_k der Sequenz auf diese Weise verrechnet werden:

$$x_{k,j} = (x_k + v_j) \bmod 1 \ , \quad \text{mit } 1 \leq k \leq N \text{ und } 1 \leq j \leq M \ . \tag{5.12}$$

Dieses Vorgehen wird *Cranley-Patterson-Rotation* genannt. Die Rechnung bewirkt hier, dass alle Elemente einer Sequenz um einen zufällig generierten Wert verschoben werden. Dieser Zufallswert ist für eine Dimension derselben Sequenz jedoch konstant. Auch nach der Cranley-Patterson-Rotation liegen die Werte einer Sequenz in jeder Dimension noch zwischen 0 und 1, da nur der Rest nach einer Division durch 1 gewertet wird. Ein Nachteil der Cranley-Patterson-Rotation ist, dass die gleichförmige Verteilung der Samples durch die Halton-, bzw. Hammersley-Sequenz nach der Verschiebung möglicherweise nicht mehr in demselben Maße gegeben ist.

Eine alternative Strategie zur zufälligen Vermischung von Sequenzen, die auf der Basis $b = 2$ basieren, wie es bei der zweidimensionalen Hammersley-Sequenz der Fall ist (vgl. Gleichung 5.11 für $d = 2$), ist das von Kollig und Keller [KK02b] beschriebene *Random Digit Scrambling*. Dabei wird eine bitweise `XOR`-Operation zwischen jeder Koordinate eines Punktes der Sequenz und einem zufälligen Bit-Vektor berechnet. Im Gegensatz zur Cranley-Patterson-Rotation gewährleistet dieses Vorgehen die Gleichförmigkeit der Samples auch nach der zufälligen Permutation.

Abbildung 5.7 auf Seite 64 zeigt die vorgestellten Sampling-Ansätze bei der Berechnung von Ambient Occlusion durch das im Rahmen dieser Arbeit implementierte Programm im direkten Vergleich. Zur Erzeugung der Bilder wurden mit dem oben beschriebenen Vorgehen jeweils 256 Sampling-Muster pro Sampling-Strategie generiert.[9] Es ist deutlich zu erkennen, dass sich die gezielte Verteilung von Samples

[9] Diese Anzahl wurde auch von Ritschel et al. [RGKM07] eingesetzt. Die Varianz der Schätzung bei der Monte-Carlo-Integration kann zusätzlich leicht gesenkt werden wenn eine größere Anzahl an Sampling-Mustern verwendet wird, da auf diese Weise die Wahrscheinlichkeit steigt, dass für zwei benachbarte Pixel unterschiedliche Sample-Richtungen untersucht werden. Andererseits steigt dadurch aber die Größe der Textur in der alle generierten Samples vorgespeichert werden.

im Gegensatz zur rein zufälligen Erzeugung ausnahmslos lohnt. Dabei muss betont werden, dass sich der rechnerische Aufwand bei der Erzeugung der sechs gezeigten Bilder nicht unterscheidet,[10] da in allen Fällen 36 Samples untersucht wurden, deren Richtungen schon im Voraus berechnet und in einer Textur gespeichert wurden.

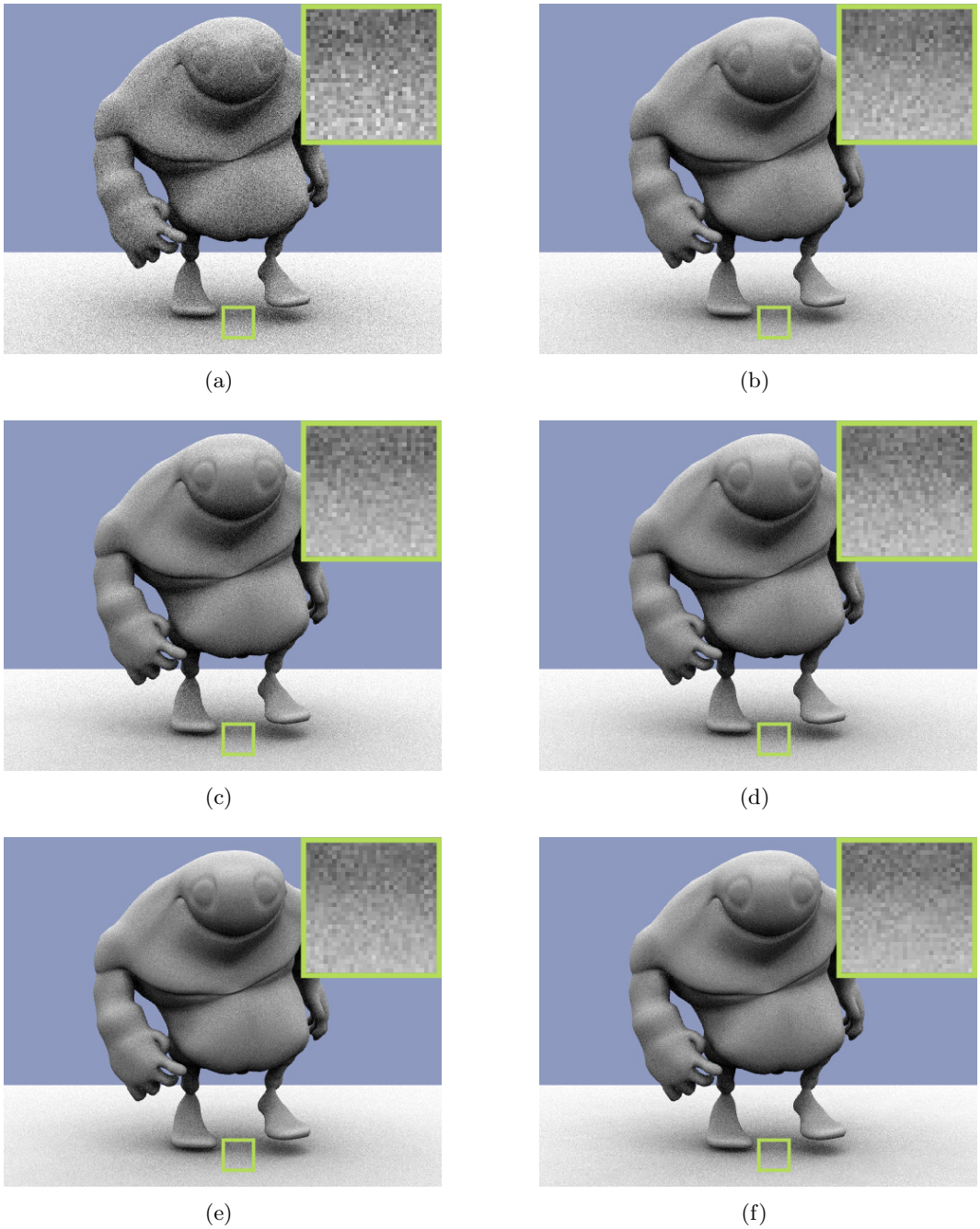
Wie erwartet senkt das Latin Hypercube Sampling die Varianz stärker als das Stratified Sampling. Eine zusätzliche Steigerung der Ausgabequalität bei gleicher Sample-Anzahl lässt sich durch die beiden Quasi-Monte-Carlo-Techniken erzielen. Dabei ist die Hammersley-Sequenz im Vergleich zur Halton-Sequenz deutlich im Vorteil. Obwohl die Permutationen der erstgenannten Sequenz durch Random Digit Scrambling theoretisch den durch die Cranley-Patterson-Rotation erzielten überlegen sind, zeigt sich optisch nur ein geringer Unterschied.

5.3.2 Realisierung der binären Suche

Die Komprimierung durch Coherent Shadow Maps basiert auf der Feststellung, dass es bei einer großen Menge von Depth Maps nicht für jedes Texel nötig ist, dieselbe Menge an Tiefenwerten zu speichern. Wie viele Tiefenwerte nach dem ausführlich in Abschnitt 4.2.1 vorgestellten Schema tatsächlich für ein Texel verwaltet werden müssen, variiert jedoch sehr stark. Während es für manche Texel in den Ecken der Depth Maps häufig genügt lediglich die unendliche Tiefe zu speichern, wobei die Komprimierung ihr volles Potential ausschöpfen kann, kommt es bei anderen Texeln zu so starken Variationen, dass annähernd so viele mittlere Tiefenwerte verwaltet werden müssen wie Depth Maps erzeugt wurden.

Das Sampling legt fest, welche Depth Maps zur Bestimmung der Visibilität eines Punktes herangezogen werden sollen. Um das auszuwertende Texel einer Depth Map zu ermitteln, wird der Punkt in das Koordinatensystem der orthographischen Kamera transformiert, die auf die gleiche Weise positioniert ist, wie zur Erzeugung der Depth Map im Rahmen der Vorverarbeitung. Dazu wird der Punkt mit der gleichen Folge an Matrixmultiplikationen verrechnet, die auch beim klassischen Shadow Mapping Anwendung findet. Auch wenn die Depth Map und die Position des Texels bekannt sind, so ist es dennoch nicht trivial den gültigen Tiefenwert innerhalb der CSM-Textur zu ermitteln. Ein direkter Verweis auf die entsprechende Stelle durch eine zusätzliche Textur würde so viel Speicher erfordern, dass der Gewinn durch die Komprimierung wieder aufgehoben wäre. Stattdessen verweist eine 2D-Lookup-Textur anhand der Texelposition auf eine Stelle in der CSM-Textur, an der mit der Suche nach dem mittleren Tiefenwert für jede beliebige Depth Map begonnen werden kann. Da die Depth Maps in streng monotoner Reihenfolge komprimiert werden, kann hierfür ein binärer Suchalgorithmus genutzt werden, der im schlimmsten Fall $\log_2(D)$ Iterationen benötigt, wenn D die Anzahl an komprimierten Depth Maps darstellt.

[10] Hier wird davon abgesehen, dass je nach Sampling-Strategie eine unterschiedlich große Anzahl an Samples nicht zur Prüfung der Visibilität anhand der CSM-Textur eingesetzt werden könnte, weil der in Abschnitt 5.3 beschriebene Fall eintritt, dass in Richtung des Samples die Bodenebene gekreuzt wird.



(a) (b) (c) (d) (e) (f)

Abbildung 5.7: Vergleich von (a) purem Monte-Carlo-Sampling, (b) Stratified Sampling, (c) Latin Hypercube Sampling, (d) 2D Halton-Sequenz mit Cranley-Patterson-Rotation, (e) 2D Hammersley-Sequenz mit Cranley-Patterson-Rotation und (f) 2D Hammersley-Sequenz mit Random Digit Scrambling. In allen Bildern wurden pro Fragment jeweils 36 Samples erzeugt. Der Ausschnitt zeigt einen Bereich von 30×30 Pixel mit 5facher Vergrößerung.

Die binäre Suche basiert auf dem Prinzip *Teile und Herrsche*. Ein Vergleich mit dem mittleren Element einer Liste gibt Aufschluss darüber, ob es sich bereits um das gesuchte Element handelt, oder ob im linken oder im rechten Teilbereich mit der Suche fortgefahren werden muss. Dort wird wiederum das mittlere Element betrachtet und das Suchintervall gegebenenfalls auf die gleiche Weise unterteilt. Spätestens wenn der unterteilte Bereich nur noch ein Element enthält, kann die Suche beendet werden.

Abbildung 5.8 verdeutlicht die Funktionsweise der binären Suche an einer fiktiven CSM-Textur. Der grüne Pfeil markiert den Einstiegspunkt für die Suche. Da die Tiefenwerte der 256 Depth Maps durch acht Segmente komprimiert wurden, lässt sich der durchschnittliche Tiefenwert zu jeder beliebigen Depth Map in höchstens drei Schritten auffinden.

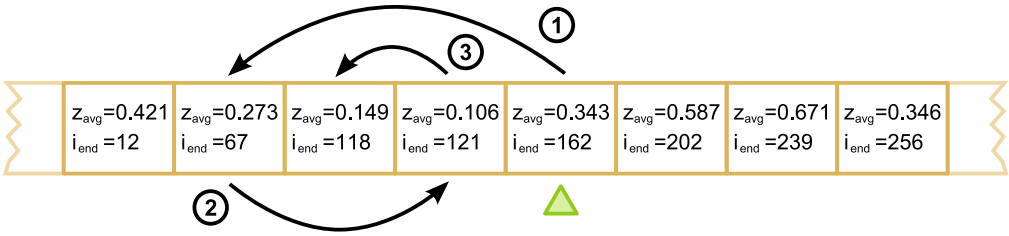


Abbildung 5.8: Erfolgreiche binäre Suche für Depth Maps > 67 und ≤ 118.

Eine Voraussetzung für die binäre Suche ist der wahlfreie Zugriff auf das mittlere Element eines Intervalls. Bei der Suche in einer CSM-Textur gestaltet sich dies besonders schwierig, da die Segmentlisten der Texel in der Regel unterschiedliche Längen besitzen. In der Lookup-Textur, die für jedes Texel auf den Startpunkt der Suche verweist, lässt sich zusätzlich die Länge der Segmentliste speichern. Wird diese Länge mit der Unterteilung der Suchintervalle durch zwei geteilt, lässt sich errechnen wie groß die beiden Teilbereiche sind. Besteht ein Bereich jedoch aus einer geraden Anzahl an Elementen, befindet sich der Mittelpunkt, je nachdem ob bei der Division auf- oder abgerundet wurde, an einer anderen Stelle und die beiden Teilbereiche besitzen eine unterschiedliche Länge. Zu welchem Problem es dabei ohne eine Behandlung dieses Spezialfalls kommen kann, zeigt Abbildung 5.9 auf der nächsten Seite. Gesucht wird hier der Tiefenwert zu einer Depth Map, die in der Iterationsfolge vor der 67. Depth Map steht. An derselben Stelle, an der im vorherigen Beispiel um zwei Segmente nach rechts gesprungen wurde, findet nun ein Sprung um zwei Segmente nach links statt. Dabei wurde nicht beachtet, dass der linke Teilbereich um ein Segment kürzer ist als der rechte. In der Folge wird in einer benachbarten Segmentliste mit der Suche fortgefahren und damit werden auch die Tiefenwerte eines anderen Texels zur Berechnung der Sichtbarkeit herangezogen.

Auch wenn es möglich ist die binäre Suche unter Berücksichtigung der genannten Spezialfälle allein mit der Länge der Segmentliste zu realisieren, wird in dieser Arbeit doch ein einfacherer Ansatz verfolgt. In jedem Eintrag der CSM-Textur wird

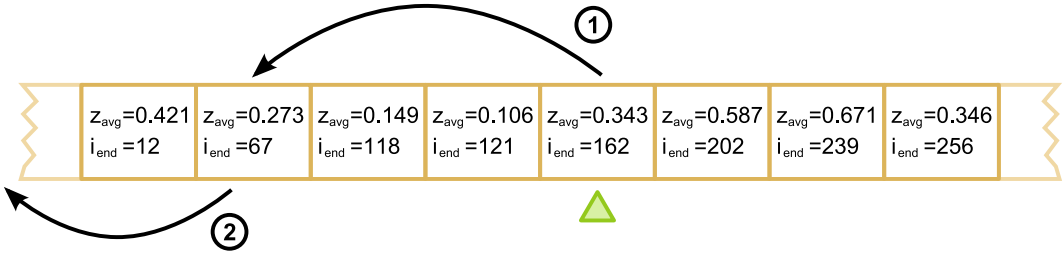


Abbildung 5.9: Fehlgeschlagene binäre Suche für Depth Maps < 67.

dazu neben den $(z_{\text{avg}}, i_{\text{end}})$-Paaren in den verbleibenden beiden Farbkanälen die Anzahl an benachbarten Segmenten zur Linken und zur Rechten gespeichert. Wie in Abbildung 5.10 verdeutlicht, kann sichergestellt werden, dass während der Suche nicht die Segmentliste verlassen wird, indem die Sprungweite maximal der Anzahl an benachbarten Segmenten in Sprungrichtung entspricht. Durch dieses Vorgehen steht auch fest, dass sich links vom ersten Segment kein weiteres Segment befindet und die Suche somit beendet werden kann.

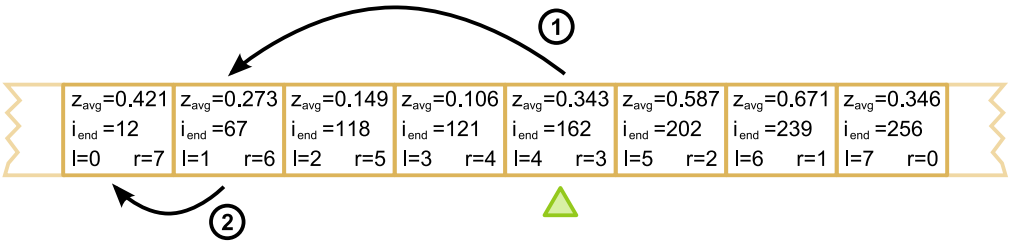


Abbildung 5.10: Erfolgreiche binäre Suche für Depth Maps < 67.

Listing 5.3 auf Seite 68 zeigt wie der vorgestellte Algorithmus in der Shader-Hochsprache HLSL umgesetzt werden kann. Die Anzahl an Iterationen hängt dabei von der Länge der Segmentliste ab. Dieser Wert wird außerhalb der Funktion aus der Lookup-Textur ausgelesen und als Parameter übergeben. Ein solches Vorgehen erfordert, dass die Hardware dynamische Schleifenobergrenzen unterstützt, was jedoch zu diesem Zeitpunkt erst bei sehr modernen Graphikkarten der Fall ist. Eine Alternative besteht darin, den Parameter `listLength` auf die maximal mögliche Länge der Segmentlisten zu setzen, d.h. die Anzahl komprimierter Depth Maps. Bei kürzeren Segmentlisten wird dann während den ersten Iterationen der Suche zwischen den beiden Grenzen der Segmentliste gesprungen. Dennoch kann jedes Segment in den letzten Iterationen gefunden werden. Eine feste Schleifenobergrenze ist zwar weniger performant, im Gegenzug wird jedoch die Portabilität zwischen unterschiedlichen Generationen an Graphikhardware gefördert. Ein zusätzlicher Vorteil betrifft den benötigten Speicher. Da die Lookup-Textur in diesem Fall nicht die Länge der Segmentliste enthält, benötigt sie lediglich zwei Kanäle. Dies ist nicht unerheblich, da die Lookup-Textur eine Präzision von 32 Bit pro Kanal erfordert,

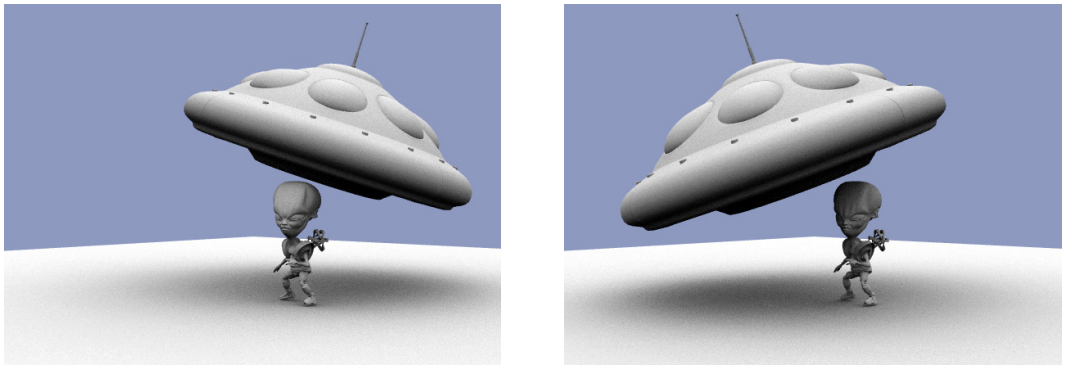


Abbildung 5.11: Ambient Occlusion zwischen verschiedenartigen, frei transformierbaren Occludern.

um bei einer großen CSM-Textur hinreichend genau auf den Startpunkt der Suche zu verweisen. Aus diesen Gründen wurde bei dem im Rahmen dieser Arbeit implementierten System eine Schleifenobergrenze gewählt, die für alle Fragments konstant ist.

5.3.3 Unterstützung mehrerer transformierbarer Occluder

Soll die darzustellende Szene aus mehr als einem Occluder bestehen, muss die Visibilität für jede Sample-Richtung der Reihe nach an allen entsprechenden CSM-Texturen getestet werden. Diese Tests finden so lange statt bis alle Occluder überprüft wurden, oder schon vorzeitig eine Verdeckung ermittelt wurde. Wird ein Occluder innerhalb der Szene transformiert, muss das Produkt der vier in Gleichung 2.3 auf Seite 9 gezeigten Matrizen vor dem Visibilitätstest zusätzlich mit der inversen Transformationsmatrix des Occluders multipliziert werden.

Es können drei Typen von Objekten unterschieden werden, deren Behandlung einen unterschiedlich großen Aufwand darstellt. Objekte, die selbst keinen Schatten werfen benötigen keine CSM-Textur und müssen lediglich an den vorhandenen Occludern auf Visibilität hin überprüft werden. Dieser Fall trifft beispielsweise auf den Boden zu. Ohne eine CSM-Textur ist es dennoch möglich, wie in Abschnitt 5.3 beschrieben, durch das frühzeitige Verwerfen von Samples zu verhindern, dass Licht durch den Boden dringt. Die zweite Klassen von Objekten umfasst Instanzen bereits vorhandener Occluder. Diese benötigen selbstverständlich keine eigene CSM-Textur. Damit sie jedoch frei bewegt werden können, wird eine individuelle Transformationsmatrix für jede dieser Occluder-Instanzen verwaltet. Der dritte und aufwändigste Typ ist der in Abbildung 5.11 demonstrierte Fall verschiedenartiger Occluder. Bis auf die beiden sampling-spezifischen Texturen, die pro Szene nur ein einziges Mal vorhanden sein müssen, benötigen diese Objekte eine eigene Ausführung aller in Abbildung 5.3 auf Seite 52 aufgeführten Texturen sowie eine individuelle orthographische Projektionsmatrix.

Listing 5.3: HLSL-Code für den binären Suchalgorithmus.

```
// Liefere die Position der Depth Map item innerhalb der Segmentliste mit
// der Länge listLength. Starte die Suche bei position.
float2 searchCSMEntry( int item, float2 position, float listLength )
{
    float2 resultPosition = {1.0, 1.0};

    // Ermittele die Länge des linken und rechten Suchintervalls
    int offset = listLength / 2.0 + 0.5;

    for( half i = 0.0; i < log2(listLength); i++ )
    {
        // Lese i_end-Wert an dieser Position
        int csmPosition = tex2D( CSMTextureSampler, position ).g;

        // Unterteile das Suchintervall
        offset = offset / 2.0 + 0.5;

        // Befindet sich die gesuchte Depth Map im rechten Intervall
        // springe zum errechneten Mittelpunkt. Prüfe mit dem Alpha-
        // kanal den Abstand zum Ende der Segmentliste.
        if( csmPosition < item )
        {
            float rightOffset = g_CSMTextureUDistance *
                ( min( tex2D( CSMTextureSampler, position ).a, offset ) );
            position.x += rightOffset;

            if( position.x > 1.0 )    // Behandlung für Texturrand
            {
                position.x -= 1.0;
                position.y += g_CSMTextureVDistance;
            }
        }
        // Befindet sich die gesuchte Depth Map im linken Intervall
        // springe zum errechneten Mittelpunkt. Prüfe mit dem Blau-
        // kanal den Abstand zum Anfang der Segmentliste.
        else if( csmPosition > item )
        {
            // Ist der Wert größer oder gleich der gesuchten Depth Map,
            // könnte dies die gesuchte Position sein.
            resultPosition = position;

            float leftOffset = g_CSMTextureUDistance *
                ( min( tex2D( CSMTextureSampler, position ).b, offset ) );
            position.x -= leftOffset;

            if( position.x < 0.0 )    // Behandlung für Texturrand
            {
                position.x += 1.0;
                position.y -= g_CSMTextureVDistance;
            }
        }
    }
    return resultPosition;
}
```

5.3.4 Beleuchtung auf Basis von Spherical Harmonics

Um Ambient Occlusion in Kombination mit einem Beleuchtungsmodell zu demonstrieren, wurde das im Rahmen dieser Arbeit implementierte System um die Möglichkeit erweitert HDR-Environment Maps zu laden. Diese speichern die Leuchtdichte einer Umgebung mit 16 Bit-Fließkommazahlen pro Kanal.

Die in Form von Environment Maps geladene Beleuchtungsinformation einer Umgebung lässt sich anhand einer Projektion auf die Basisfunktionen effizient durch Spherical Harmonics repräsentieren. Da sie hier nicht den Schwerpunkt bilden, wird der mathematische Hintergrund von Spherical Harmonics in dieser Arbeit nicht erläutert – einen empfehlenswerten Einstieg in dieses Thema bieten Green [Gre03] und Biedermann [Bie04]. Im Kern handelt es sich bei den Spherical Harmonics um ein mathematisches System, das sich als Erweiterung der Fouriertransformation auf die Oberfläche einer Kugel verstehen lässt. Wichtig ist im Zusammenhang mit Ambient Occlusion, dass im Gegensatz zum Precomputed Radiance Transfer nur die Lichtfunktion, und nicht die Transferfunktion unter Berücksichtigung der Visibilität, in die SH-Basis projiziert wird. Auf diese Weise entfällt eine aufwändige Vorberechnung und das Vorgehen beschränkt sich nicht ausschließlich auf statische Szenen.

Im Falle diffuser Materialien, auf die Ambient Occlusion zugeschnitten ist, sind, wie Ramamoorthi und Hanrahan [RH01] festgestellt haben, nur neun SH-Koeffizienten (L_{lm} für $l \leq 2$) nötig um eine natürliche Beleuchtungssituation in allen Richtungen mit einem hinreichend geringen Fehler festzuhalten. Diese Koeffizienten lassen sich durch Aufsummierung der in der Environment Map eingetragenen Leuchtdichte-Werte, gewichtet mit den SH-Basisfunktionen Y_{lm}, berechnen[11]:

$$L_{lm} = \int_{\theta=0}^{\pi} \int_{\varphi=0}^{2\pi} L(\theta,\varphi) Y_{lm}(\theta,\varphi) \sin\theta d\theta d\varphi \ .$$

Umgekehrt lässt sich anhand der ersten neun auf diese Weise ermittelten Koeffizienten die Beleuchtungsstärke E in Richtung der normalisierten Normalen $\mathbf{n} = (x,y,z)^T$ ausrechnen [RH01]:

$$\begin{aligned} E(\mathbf{n}) &= c_1 L_{22}(x^2-y^2) + c_3 L_{20} z^2 + c_4 L_{00} - c_5 L_{20} \\ &+ 2c_1(L_{2-2}xy + L_{21}xz + L_{2-1}yz) \\ &+ 2c_2(L_{11}x + L_{1-1}y + L_{10}z) \ , \\ &\quad \text{mit} \qquad c_1 = 0{,}429043 \quad c_2 = 0{,}511664 \\ &\quad c_3 = 0{,}743125 \quad c_4 = 0{,}886227 \quad c_5 = 0{,}247708 \ . \end{aligned} \tag{5.13}$$

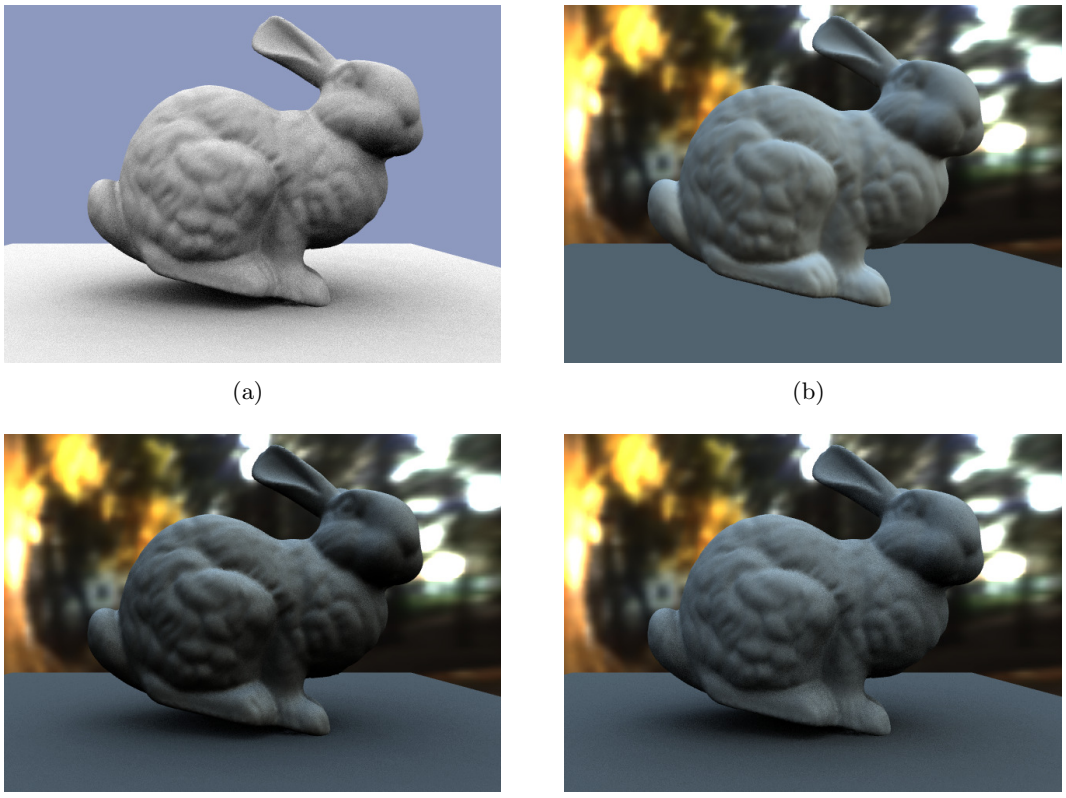
[11] Dieses Vorgehen, bei dem ermittelt wird wie stark das Gewicht der Basisfunktionen sein muss um die Beleuchtungssituation zu approximieren, wird *Projektion* genannt. Bei dem hier implementierten Programm wird diese Projektion durch die Direct3D-Methode `D3DXSHProjectCubeMap()` realisiert. Bei Verweis auf eine in Form einer Cube Map geladenen Textur und unter Angabe der gewünschten Ordnung o, liefert diese die entsprechenden o^2 SH-Koeffizienten, aufgeschlüsselt nach Rot-, Grün- und Blaukomponenten. Für Cube Maps mit einer Auflösung von bis zu 512×512 Texel pro Seite dauert die Berechnung der SH-Koeffizienten mit gängiger Hardware weniger als zwei Sekunden.

Da die Reflexion diffuser Materialien nicht sehr plötzlich variiert, genügt es oft die obige Gleichung im Vertex Shader auszuwerten. Alternativ kann aber auch die in Abschnitt 3.2.1 vorgestellte Bent Normal in die Gleichung eingesetzt werden. Diese wird bei der Berechnung von Ambient Occlusion auf Basis von CSMs durch Mittelung aller Sample-Richtungen in denen keine Verdeckung festgestellt wird bestimmt. Dies ist ein Vorteil gegenüber vielen der in Abschnitt 3.3 beschriebenen Ansätze, die diese Möglichkeit nicht bieten.

Der Einsatz von Bent Normals zur Steuerung der Beleuchtung durch eine Environment Map ist in Kombination mit Ambient Occlusion nicht völlig plausibel. Zum einen kann der Fall eintreten, dass eine Bent Normal in Richtung eines Occluders zeigt.[12] Zum anderen basiert Ambient Occlusion, wie in Abschnitt 3.2.2 nachgezeichnet, auf der Annahme, dass die Leuchtdichte über die gesamte Hemisphäre uniform ist. Genau genommen müsste daher in jede Sample-Richtung, in der kein Occluder festgestellt wird, die Environment Map ausgewertet werden. Tatsächlich wird jedoch aufgrund des damit verbundenen, höheren Rechenaufwands selbst bei Filmproduktionen, deren Fokus vor allem auf der Generierung eines überzeugenden Ergebnisbildes liegt, teilweise nicht auf diese Weise vorgegangen [Lan02]. Da bei dem hier beschriebenen Ansatz neben der Darstellungsqualität auch die Performanz eine Rolle spielt, wird auch bei dem im Rahmen der Arbeit implementierten Programm lediglich die Bent Normal zur Auswertung der Umgebungsbeleuchtung herangezogen.

Welchen Vorteil die Verwendung der Bent Normal bei der Nachbildung natürlichbasierter Beleuchtung in Kombination mit Ambient Occlusion liefert, demonstriert Abbildung 5.12 auf der nächsten Seite. In allen Bildern findet die Beleuchtung auf Fragment-Basis und ohne den Einsatz eines *Tone Mappings* statt. Die hier eingesetzte *Light Probe* „Eucalyptus Grove“ gibt die Beleuchtungssituation einer kleinen Lichtung wieder, in welcher ein Großteil des einfallenden Lichtes von oben kommt. Ohne eine Berücksichtigung der Visibilität wirkt das Ergebnis wenig realistisch. Durch eine Multiplikation mit dem Ambient Occlusion-Wert lässt sich diese jedoch integrieren. Eine solche Kombination von Ambient Occlusion und der SH-basierten Beleuchtung anhand der Normalen führt allerdings zu einem sehr dunklen Bild (vgl. Abbildung 5.12(c)). Wird dagegen die Bent Normal zur Beleuchtung herangezogen, erscheint das Modell wesentlich heller und optisch sogar eher wie eine Kombination aus Abbildung 5.12(a) und Abbildung 5.12(b). Der Unterschied ist darin begründet, dass zur Berechnung der Bent Normal nur Sample-Richtungen herangezogen werden, die keine Verdeckung liefern. Da alle Sample-Richtungen welche die Bodenebene schneiden als Verdeckung gewertet werden, zeigt die Bent Normal eher nach oben und damit im Falle dieser Environment Map in den Bereich, in dem die größten Helligkeitswerte kodiert sind. Umgekehrt macht demnach die Kombination von Ambient Occlusion und natürlichbasierter Beleuchtung anhand der Bent Normal nur

[12] Tatsächlich müsste, wie Biedermann [Bie04] anmerkt, die Richtung des meisten einfallenden Lichts durch den Median der sortierten, unverdeckten Sample-Richtungen und nicht durch den Mittelwert bestimmt werden. Dies ist allerdings nicht unproblematisch, da Vektoren keine Ordnungsrelation besitzen.



(a) (b)

(c) (d)

Abbildung 5.12: Nur (a) Ambient Occlusion, (b) SH-basierte Beleuchtung anhand der Fragment-Normalen, (c) Ambient Occlusion und SH-basierte Beleuchtung anhand der Fragment-Normalen und (d) Ambient Occlusion und SH-basierte Beleuchtung anhand der Bent Normal.

Sinn, wenn in der eingesetzten Environment Map der Himmel die stärkste Lichtquelle darstellt – dies ist wenig überraschend, da Ambient Occlusion ja bereits als eine Simulation der Beleuchtung an einem bewölkten Tag motiviert wurde.

5.4 Das implementierte System

Dieser Abschnitt befasst sich mit der Funktionalität des im Rahmen dieser Arbeit implementierten Systems. Darüber hinaus werden die zur Umsetzung herangezogenen Komponenten erläutert und es wird auf die Benutzerschnittstelle eingegangen. Die dabei getroffenen Entscheidungen sollen begründet und nachvollziehbar gemacht werden, sie stellen jedoch nicht den einzig möglichen Weg dar, um zu den erzielten Ergebnissen, die in Abschnitt 6 vorgestellt werden, zu gelangen.

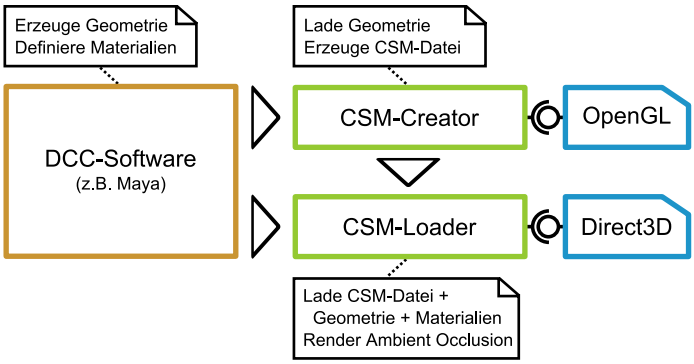


Abbildung 5.13: Das implementierte System im Kontext.

5.4.1 Die Funktionalität

Wie Abbildung 5.13 verdeutlicht, teilt sich das implementierte System auf in eine Anwendung zur Erzeugung von CSM-Dateien (*CSM-Creator*) und eine Anwendung, die vorhandene CSM-Dateien lädt und zur Berechnung und Darstellung von Ambient Occlusion nutzt (*CSM-Loader*). Beide Anwendungen sind in der Lage, 3D-Modelle im offenen `obj`-Format, das ursprünglich von der Firma Wavefront Technologies entwickelt wurde, zu laden. Ein solches Modell wird typischerweise mit einem Programm zur Erzeugung digitalen Inhalts (engl. *Digital Content Creation*, DCC) erstellt. Diese Programme bieten auch die Möglichkeit Materialparameter oder Texturen für ein 3D-Modell zu definieren. Für Modelle im `obj`-Format werden die Materialeigenschaften in einer separaten Datei mit der Endung `mtl` gespeichert. Da in CSM-Creator kein Shading zum Einsatz kommt, wird hier auf das Laden dieser Datei verzichtet.

Nach Eingabe der gewünschten Auflösung und Anzahl der Depth Maps beginnt CSM-Creator mit der Generierung der zu komprimierenden Depth Maps. Da zunächst alle Depth Maps in Form von Vektoren der *Standard Template Library* (STL) abgespeichert werden bevor die Komprimierung anhand der Vektor-Daten beginnt, ist die maximal mögliche Kombination von Auflösung und Anzahl der Depth Maps durch die verfügbare Menge an Arbeitsspeicher begrenzt. Dieses Vorgehen ermöglicht allerdings die sequentielle Erzeugung der CSM-Dateien, in denen für ein Texel (d.h. eine Segmentliste) die komprimierten Tiefenwerte aller Depth Maps (d.h. alle Segmente) aufgeführt werden, bevor mit dem nächsten Texel fortgefahren wird. Der Benutzer bekommt eine Rückmeldung darüber, wie weit die Akquirierung der Tiefendaten vorangeschritten ist, indem eine Statusleiste und die Depth Maps direkt bei ihrer Erzeugung ausgegeben werden. Nachdem alle Depth Maps generiert wurden, beginnt die Komprimierung. Auch hierbei wird der Benutzer anhand einer Statusleiste über den Fortschritt informiert. Sobald ein komplettes Segment gefunden wurde, werden die entsprechenden Daten direkt in die CSM-Datei geschrieben. Nach der Komprimierung aller Depth Maps gibt das Programm die erzielte Komprimierungsrate und die benötigte Dauer der Operation aus.

Für CSM-Loader wird zur Compile-Zeit festgelegt, welche Objekte und zugehörigen CSM-Dateien geladen werden sollen. Daneben muss die Auflösung des Fensters und die Anzahl an Samples zur Berechnung von Ambient Occlusion im Voraus definiert werden. Diese Werte sind über die gesamte Laufzeit konstant, da sie die Größe der in Abschnitt 5.3 beschriebenen Pixel-Zufalls- und Sampling-Muster-Textur determinieren. Alle weiteren benötigten Informationen werden direkt aus den CSM-Dateien ausgelesen. Je nach Größe der zum Einsatz kommenden Dateien kann dies bis zu mehreren Minuten dauern. Zur Laufzeit kann ein dargestelltes Modell über Tastatureingaben frei verschoben und rotiert werden. Die graphische Benutzerschnittstelle, die in Abschnitt 5.4.3 vorgestellt wird, bietet weitere Möglichkeiten zur Interaktion mit dem Dargestellten, wie etwa die Auswahl einer Light Probe oder einer Sampling-Strategie.

5.4.2 Bei der Umsetzung verwendete Komponenten

Das gesamte System wurde mit C++ unter Verwendung der Entwicklungsumgebung „Microsoft Visual Studio .NET 2003“ auf dem Betriebssystem „Microsoft Windows XP“ implementiert.

Zur Umsetzung der Teilanwendung CSM-Creator wurde die Graphikbibliothek OpenGL eingesetzt. Ausschlaggebend für diese Entscheidung war insbesondere die unkomplizierte Realisierung des Depth Peeling anhand der in Abschnitt 5.2.2 erläuterten Extensions. Da die Anwendung lediglich den Zweck erfüllt CSM-Dateien zu erzeugen, wurde auf eine ansprechende Benutzerschnittstelle verzichtet. Die Interaktion des Benutzers mit dem Programm findet ausschließlich durch Verwendung der Tastatur über Eingaben in die Kommandozeile statt.

CSM-Loader soll in erster Linie den Einsatz von CSMs zur Berechnung von Ambient Occlusion demonstrieren. Da dieses Programm auch ungeübten Anwendern zugänglich sein sollte, wurde besonders Wert auf eine intuitive Benutzerschnittstelle gelegt, die sowohl eine Interaktion per Tastatur als auch mit der Computermaus erlaubt. Aus diesem Grund wurde sich zur Umsetzung von CSM-Loader für eine Verwendung von Direct3D in Kombination mit dem *DirectX Utility Toolkit* (DXUT) entschieden. Dieses ermöglicht die einfache Realisierung einer graphischen Benutzerschnittstelle mit zahlreichen Elementen, wie Schaltflächen, Schiebereglern oder Eingabefeldern. Darüber hinaus wurde die Kameraklasse `CModelViewerCamera` des DXUT eingesetzt. Diese erlaubt die Rotation der Szene mit der linken, sowie eine Rotation der Kamera mit der rechten Maustaste. Mit dem Mausrad lässt sich die Entfernung der Kamera zum zentrierten Blickpunkt regulieren. Zur Programmierung der Graphikhardware kam in CSM-Loader die *High-Level Shader Language* (HLSL) zum Einsatz. Dieses Vorgehen bot sich an, da HLSL, ebenso wie Direct3D, als Teil des *DirectX Software Development Kit* (DirectX SDK) frei verfügbar und ausführlich dokumentiert ist. Außerdem erwies sich das bei dieser Kombination ermöglichte Debugging von Shadern durch das Setzen von Haltepunkten und das Anzeigen des Inhalts von Variablen, bei der Implementierung als besonders hilfreich.

Die Verwendung von OpenGL in CSM-Loader und Direct3D in CSM-Creator ist nicht unproblematisch, da OpenGL auf einem rechtshändigen und Direct3D auf ei-

Abbildung 5.14: Die graphische Benutzerschnittstelle von CSM-Loader.

nem linkshändigen Koordinatensystem basiert – d.h., dass die positive Z-Achse in beiden Graphikbibliotheken in die entgegengesetzte Richtung zeigt. Dies muss beachtet werden, wenn in CSM-Loader die in CSM-Creator vorberechnete Visibilität ausgewertet werden soll. Bei der Umsetzung von CSM-Loader wurde daher so vorgegangen, wie es in der Dokumentation des DirectX SDK [Mic08] zur Lösung dieses Problems empfohlen wird. Zum einen müssen dazu alle Polygone in umgekehrter Reihenfolge aufgebaut werden, nämlich indem die Eckpunkte von vorne gesehen im Uhrzeigersinn abgearbeitet werden. Zum anderen sollte jedes Element aus der dritten Zeile der View-Matrix mit -1 multipliziert werden. Die zuletzt genannte Aktion führt allerdings dazu, dass die Kamera-Steuerung bei Verwendung der Klasse `CModelViewerCamera` deutlich an Intuition einbüßt. Aus diesem Grund wurde auf eine Skalierung der View-Matrix verzichtet, mit dem für diese Anwendung hinnehmbaren Effekt, dass alle Objekte an der XY-Ebene gespiegelt dargestellt werden.

5.4.3 Die graphische Benutzerschnittstelle

Abbildung 5.14 zeigt die mit dem DXUT realisierte graphische Benutzerschnittstelle der Teilanwendung CSM-Loader, deren Elemente im Folgenden vorgestellt werden.

In der oberen linken Ecke liefert die Anwendung Informationen über die aktuelle Framerate (fps), die Deaktivierung der vertikalen Synchronisation (Vsync), die Auflösung des Fensters, das Format von Back Buffer, Depth Buffer und Stencil Buffer sowie das hier zum Einsatz kommende vierfache Multisampling zur Reduzierung von Aliasing. Die nächste Zeile bezieht sich auf die verwendete Hardware. Neben deren Bezeichnung wird auch Auskunft darüber gegeben, ob das Direct3D Device

und die Verarbeitung der Eckpunkte von der Hardware übernommen oder durch Software emuliert wird.

Auf Wunsch lässt sich ein Hilfetext mit Hinweisen zur Steuerung über die Tastatur, und weitere Statistiken, wie die Anzahl eingesetzter Samples und komprimierter Depth Maps ausgeben.

Mit den Schaltflächen im rechten oberen Bereich lässt sich in den Vollbildmodus umschalten, die Szene als Drahtgittermodell anzeigen und ein von Direct3D bereitgestellter Dialog zum Anpassen der eingesetzten Formate, des Multisampling und des Direct3D Device aufrufen.

Der Schieberegler darunter ermöglicht die Festlegung eines Toleranzbereichs beim Vergleich der Tiefenwerte zur Erkennung von Schatten. Dies ist in geringem Ausmaß nötig, da die zur Laufzeit mit Direct3D berechneten Tiefenwerte stets etwas größer als die mit OpenGL vorberechneten Tiefenwerte sind. Die Sampling-Strategie zur Berechnung von Ambient Occlusion kann frei über die Benutzerschnittstelle gewählt werden, da im Voraus drei verschiedene Sampling-Muster-Texturen generiert werden. Von den folgenden vier *Radio Buttons* ist zu jedem Zeitpunkt stets nur eine Einstellung aktiv:

No Ambient Occlusion In diesem Fall wird die Berechnung von Ambient Occlusion deaktiviert. Für jedes Fragment wird nur entlang seiner Normalen die Visibilität anhand der CSM-Textur ermittelt. Diese Berechnung eines Schatten-Samples findet auch unter Verwendung von Low-End-Graphikhardware in interaktiven Frameraten statt und ermöglicht die schnelle Erkennung fehlerhafter Schatten und damit eine präzise Einstellung des *Shadow offset*-Schiebereglers. Die Beleuchtung wird bei dieser Einstellung vertexbasiert durch eine Punktlichtquelle berechnet.

Monte Carlo In dieser Einstellung wird Ambient Occlusion anhand der zur Compile-Zeit festgelegten Anzahl von Samples berechnet, die per Monte Carlo in Kombination mit Latin Hypercube Sampling verteilt werden. Das Verhältnis von verschossenen Samples zu den Samples, für die ein Schatten erkannt wurde, bestimmt die Helligkeit eines Fragments.

Halton Alternativ zum Monte Carlo Sampling kann die Verteilung der Samples auch durch die 2D Halton-Sequenz mit Cranley-Patterson-Rotation festgelegt werden.

Hammersley Eine andere Quasi-Monte-Carlo-Technik, die durch zufällige Permutationen ein erwartungstreues Ergebnis liefert und im Vergleich zum Monte Carlo Sampling eine deutlich niedrigere Varianz aufweist, ist die 2D Hammersley-Sequenz mit Random Digit Scrambling.

Von den folgenden drei *Check Boxes* kann jede unabhängig voneinander aktiv sein und in beliebiger Weise mit den oben genannten Radio Buttons kombiniert werden:

Faked AO Die gewünschte Anzahl an Samples mit der gewählten Sampling-Strategie wird bei dieser Einstellung nur dazu genutzt, wie in Abschnitt 5.3 beschrieben, den Anteil an Samples zu ermitteln, welche die Bodenebene schneiden.

Hierbei wird kein Visibilitätstest anhand der CSM-Textur durchgeführt, aber Samples die nicht in den oberen Halbraum zeigen werden als Schatten gewertet. Auf diese Weise lässt sich auch mit älterer Graphikhardware eine Art Ambient Occlusion erzielen (vgl. Abbildung 5.4(a) auf Seite 53).

Render textures Mit dieser Einstellung lässt sich die Darstellung der in den Material-Dateien zu den geladenen Objekten angegebenen Texturen aktivieren.

Environm. lighting Bei Aktivierung dieser Einstellung wird wie in Abschnitt 5.3.4 beschrieben vorgegangen, um das Modell durch eine, auf die Basisfunktionen der Spherical Harmonics projizierte Light Probe zu beleuchten. Bei einer aktivierten Sampling-Strategie und deaktiviertem *Faked AO* wird die Beleuchtung in Richtung der Bent Normal untersucht, ansonsten in Richtung der Normalen. Welche Light Probe zum Einsatz kommt, kann über die darunter liegende Dropdown-Liste ausgewählt werden.

5.5 Zusammenfassung

In diesem Kapitel wurde gezeigt, wie sich die Berechnung von Ambient Occlusion mittels CSMs realisieren lässt und welche Funktionalität das im Rahmen dieser Arbeit implementierte System bietet. Die Unterschiede zu dem durch Ritschel et al. [RGKM07] vorgestellten Vorgehen für den Fall der Schattenberechnung bei einer Beleuchtung durch unendlich weit entferntes Umgebungslicht werden im Folgenden noch einmal zusammengefasst.

Es lässt sich eine wesentlich bessere Komprimierungsrate und gleichzeitig eine Abnahme von Banding-Artefakten erzielen, wenn die Augpunkte der zur Generierung der Depth Maps herangezogenen orthographischen Kameras gleichmäßiger auf der Umkugel eines Occluder-Objekts verteilt werden. Für den Fall von Polarkoordinaten heißt dies, dass für φ, welches für einen doppelt so großen Bereich wie θ definiert ist, auch eine doppelt so große Unterteilung vorgenommen werden sollte.

Die Speicherung der durch die Komprimierung erhaltenen Informationen in einem Dateiformat ermöglicht die Berechnung der Verschattung durch einen Occluder-Typ auf beliebig vielen Systemen und in verschiedenen Szenen, ohne die aufwändige Komprimierung ein zweites Mal durchführen zu müssen. Da sich die Daten gut gliedern und auch von Menschen interpretieren lassen, bietet sich der Einsatz eines XML-konformen Dateiformats an.

Indem zur Laufzeit frühzeitig Sample-Richtungen, welche die Bodenebene schneiden, verworfen und nicht für einen Visibilitätstest an CSM-Texturen herangezogen werden kann die Performanz gesteigert werden.

Sampling auf Basis von Low-Discrepancy Sequences unterliegt durch zufällige Permutationen keinen Aliasing-Artefakten und liefert darüber hinaus ein erwartungstreues und deutlich weniger verrauschtes Ergebnis als rein stochastische Vorgehensweisen. Auf diese Weise lässt sich, wie im nächsten Kapitel gezeigt wird, auch ohne Progressive Rendering eine akzeptable Darstellungsqualität in interaktiven Frameraten erreichen.

Die binäre Suche eines Segments innerhalb einer CSM-Textur kann durch das Einfügen der Anzahl benachbarter Segmente zu beiden Seiten einfacher und robuster gestaltet werden.

Die Berechnung von Ambient Occlusion durch CSMs ermöglicht es ohne großen Aufwand die Bent Normal zu bestimmen. Diese kann für eine Beleuchtung durch Environment Maps, die beispielsweise in die Basisfunktionen der Spherical Harmonics projiziert wurden, herangezogen werden um im Vergleich zur Fragment-Normalen in vielen Fällen ein optisch ansprechenderes Ergebnis zu erzielen.

6 Ergebnisse

6.1 Darstellungszeiten getesteter Szenen

Die Berechnung von Ambient Occlusion durch das in dieser Arbeit beschriebene Vorgehen wurde an vier Szenen hinsichtlich der Darstellungsgeschwindigkeit getestet. Diese Szenen bestehen jeweils aus einem Occluder-Modell und drei Flächen im Hintergrund zu denen keine CSM-Textur verwaltet wird. Der Einsatz eines einzigen Modells pro Testszene soll es erlauben einen Bezug zwischen Komprimierungsrate, welche direkt mit der Beschaffenheit des Objekts zusammenhängt, und Performanz herzustellen. Die drei Hintergrundflächen wurden so gesetzt, dass für jedes Pixel des Anwendungsfensters Ambient Occlusion berechnet wird, sodass auch zwischen Auflösung und Darstellungsgeschwindigkeit ein Zusammenhang besteht und sich umgekehrt die Anzahl an Samples pro Sekunde errechnen lässt. Für ein korrektes Ergebnis müssten auch die beiden Wandflächen jeder Szene als Occluder behandelt werden, worauf hier allerdings verzichtet wurde, um alle Visibilitätstests lediglich anhand der CSM-Textur des Testobjekts durchzuführen.

Wird an die zum Einsatz kommende Graphikhardware nicht die Anforderung gestellt, dass sie dynamische Schleifenobergrenzen unterstützen muss, so wird, wie in Abschnitt 5.3.2 erläutert, stets davon ausgegangen, dass die Länge einer Segmentliste der Anzahl an komprimierten Depth Maps entspricht. In diesem Fall lässt sich die Zeitkomplexität für die Berechnung von Ambient Occlusion durch CSMs aus den Eingabedaten abschätzen:

$$\mathcal{O}(X \times Y \times \log_2(D) \times N) \ ,$$

wobei X und Y die Auflösung und damit die Menge zu untersuchender Pixel kennzeichnen, D für die Anzahl an komprimierten Depth Maps und N für die Anzahl an Samples steht.

Eine Anzahl von 36 Samples hat sich in allen Versuchen als ein gutes Mittelmaß zwischen Performanz und Darstellungsqualität erwiesen. Insbesondere in Kombination mit den zweidimensionalen Sequenzen nach Halton und Hammersley fällt in diesem Fall das Rauschen in den Bildern kaum noch auf. Dies demonstriert beispielsweise Abbildung 6.1. Wie auch in den folgenden drei Abbildungen besitzt das gezeigte Bild eine Auflösung von 640×480 Pixel. Die zum Einsatz kommende bildbasierte Beleuchtung auf Basis von Spherical Harmonics (vgl. Abschnitt 5.3.4) beeinflusst die Performanz in keinem deutlich wahrnehmbaren Ausmaß. Die unmittelbare Nähe des Modells zur Wand – in der Tat schneidet das Modell die rechte Wand sogar – wird durch den Verlauf und die Stärke des Schattens deutlich. Bei einer Anzahl

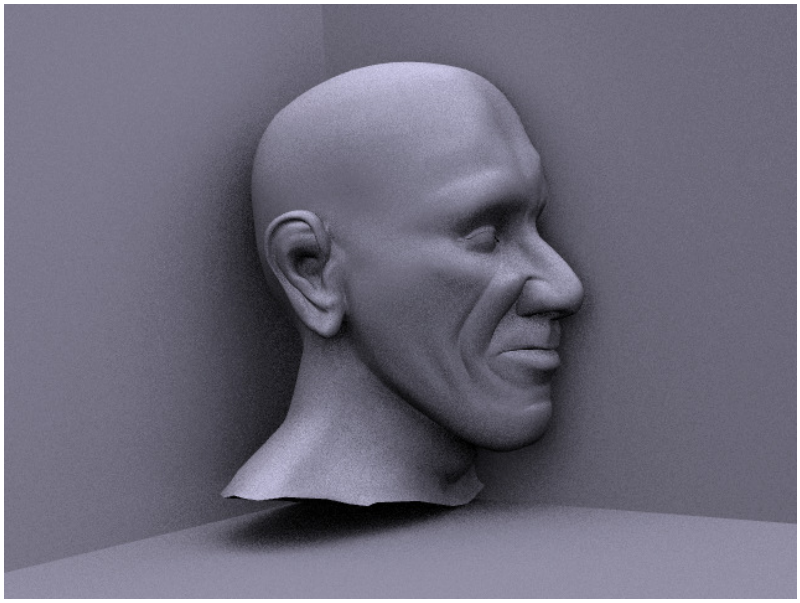


Abbildung 6.1: Testszene *Head* mit bildbasierter Beleuchtung durch die Light Probe *Uffizi Gallery* und Ambient Occlusion mittels 36 Samples pro Fragment verteilt durch die 2D Hammersley-Sequenz mit Random Digit Scrambling.

von 64×128 komprimierten Depth Maps mit einer Auflösung von jeweils 128×128 Texeln sind in dem gezeigten Bild kaum Banding-Artefakte sichtbar. Lediglich bei mehreren aufeinander folgenden Bildern, in denen der Occluder beispielsweise rotiert wird, werden diese in Form von leichten Unstetigkeiten des Schattens deutlich.

Abbildung 6.2 zeigt Ambient Occlusion anhand von zufällig gewählten Samples, deren günstige Verteilung mit Latin Hypercube Sampling sichergestellt wird. Wie auch bei einer Verwendung von Stratified Sampling wirkt das Ergebnis deutlich stärker verrauscht als bei einem Einsatz der 2D Halton- oder Hammersley-Sequenz. Um eine gute Komprimierungsrate zu erzielen wurden die in dem Bunny-Modell vorhandenen Löcher im Voraus ausgefüllt (vgl. Abschnitt 5.2.1).

Das sampling-bedingte Rauschen und Banding-Artefakte durch eine zu geringe Anzahl komprimierter Depth Maps werden deutlich weniger auffällig, wenn Ambient Occlusion, wie in Abbildung 6.3 und Abbildung 6.4 demonstriert, mit Texturen verrechnet wird. Wie auch bei der Beleuchtung durch Light Probes nimmt die Performanz dadurch nicht ab. Durch die Kombination mit Texturen wird der dargestellte Schatten zwar subtiler, er trägt aber dennoch wesentlich dazu bei, dass das Dargestellte als realistisch empfunden wird.

Tabelle 6.1 auf Seite 83 zeigt die mittleren Darstellungsgeschwindigkeiten sowie die Speicheranforderungen der vier Testszenen bei verschiedenen Auflösungen. Die Werte wurden auf einem System mit Intel Core 2 Duo 6400 Prozessor, einem Gigabyte RAM, und einer NVIDIA GeForce 8800 GTX Graphikkarte gemessen. Es ist zu erkennen, dass die Performanz wesentlich stärker von der Komprimierungsrate als

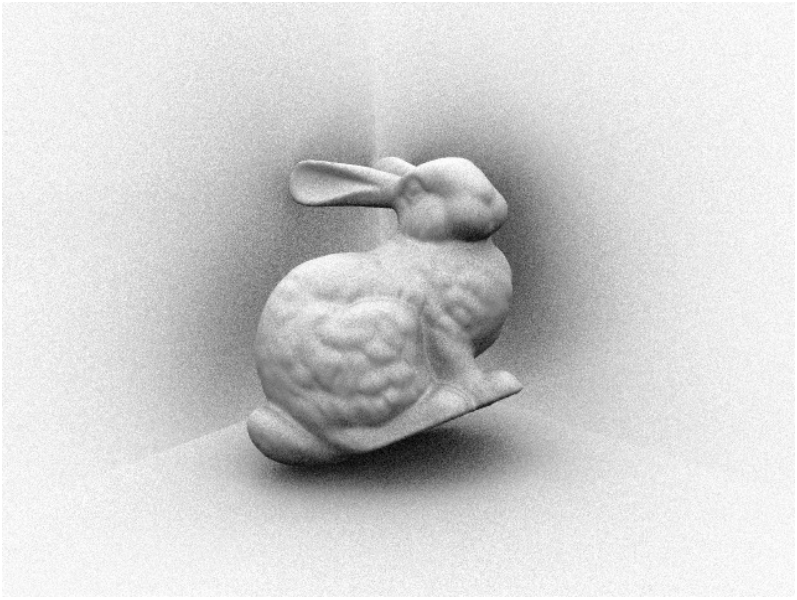


Abbildung 6.2: Testszene *Bunny* mit Ambient Occlusion mittels 36 Samples pro Fragment verteilt durch Monte Carlo Sampling mit Latin Hypercube.

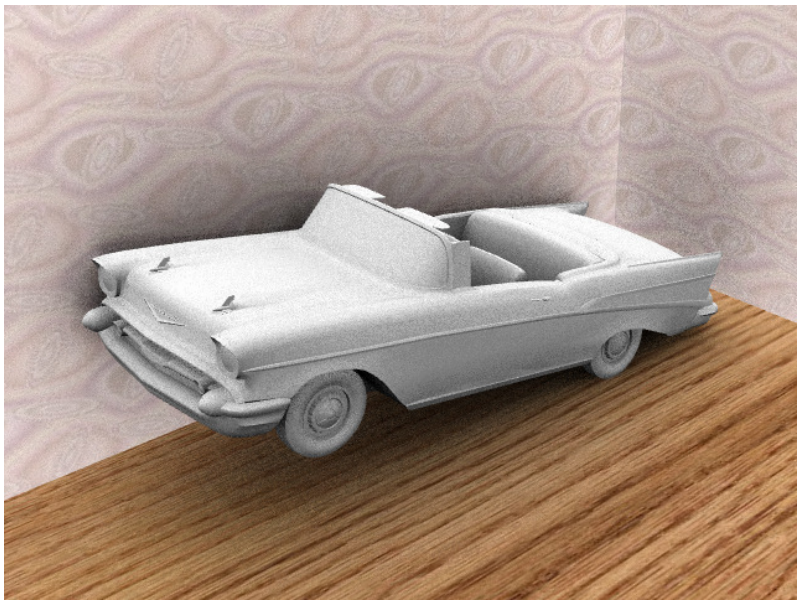


Abbildung 6.3: Testszene *Chevy* mit Texturen und Ambient Occlusion mittels 36 Samples pro Fragment verteilt durch die 2D Hammersley-Sequenz mit Random Digit Scrambling.

Abbildung 6.4: Testszene *Venus* mit Texturen, bildbasierter Beleuchtung durch die Light Probe *Eucalyptus Grove*, und Ambient Occlusion mittels 36 Samples pro Fragment verteilt durch die 2D Halton-Sequenz mit Cranley-Patterson-Rotation.

von der tatsächlichen Komplexität des Modells abhängt. Insbesondere für die Chevy-Szene nimmt die Darstellungsgeschwindigkeit mit zunehmender Auflösung deutlich drastischer ab, als beispielsweise für die Bunny-Szene. Die niedrige Komprimierungsrate des Chevy-Modells erklärt sich daraus, dass es aus zahlreichen Einzelteilen, wie Reifen, Türgriffen und Scheinwerfern, aufgebaut ist, wodurch die freie Wahl eines Tiefenwertes zwischen den ersten beiden Tiefenebenen sehr stark eingeschränkt wird. Während für die Chevy-Szene bei der höchsten getesteten Auflösung 51 Millionen Schatten-Samples pro Sekunde berechnet werden, sind es für die Venus-Szene mit 99 Millionen Samples pro Sekunde beinahe doppelt so viele. Die Ladezeit der Anwendung korreliert mit der Größe der CSM-Datei und hängt damit ebenfalls von der Komprimierungsrate ab. Diese Zeit umfasst die Erstellung aller CSM-spezifischen Texturen (darunter drei Texturen für unterschiedliche Sampling-Strategien), sowie das Laden von Geometrie und fünf, zur Laufzeit frei wechselbarer Light Probes. Daneben wird in dieser Zeit auch der zur Berechnung von Ambient Occlusion zum Einsatz kommende Shader kompiliert – dieser Prozess dauert aufgrund der nötigen Schleifen und Verzweigungen einige Sekunden.[1] Obwohl die Ladezeit insgesamt relativ hoch ist, ist sie für die getesteten Szenen um ein Zwei- bis Fünffaches niedriger als die benötigte Zeit zur Komprimierung der Depth Maps durch CSM-Creator –

[1] Durch den Einsatz von vorkompilierten Shadern kann die Ladezeit geringfügig herabgesetzt werden. Bei der Kompilierung von HLSL-Shadern kann dazu etwa das Tool `fxc`, das Teil des Microsoft DirectX SDK ist, mit der Option `/Fo` eingesetzt werden.

Szene	Polyg.	Kmpr.-Rate	CSM-Datei	Ladezeit	Auflösung	MB	fps
Head	20782	35,59 : 1	134 MB	65 s	320×240	56,5	9,7
					640×480	57,4	4,3
					1024×768	59,3	1,9
Bunny	69664	30,50 : 1	170 MB	89 s	320×240	55,6	8,4
					640×480	56,5	3,9
					1024×768	58,4	2,6
Chevy	29304	16,55 : 1	254 MB	123 s	320×240	64,1	14,5
					640×480	65,0	4,0
					1024×768	66,9	1,8
Venus	43357	67,56 : 1	71,5 MB	38 s	320×240	45,3	19,6
					640×480	46,2	6,7
					1024×768	48,1	3,5

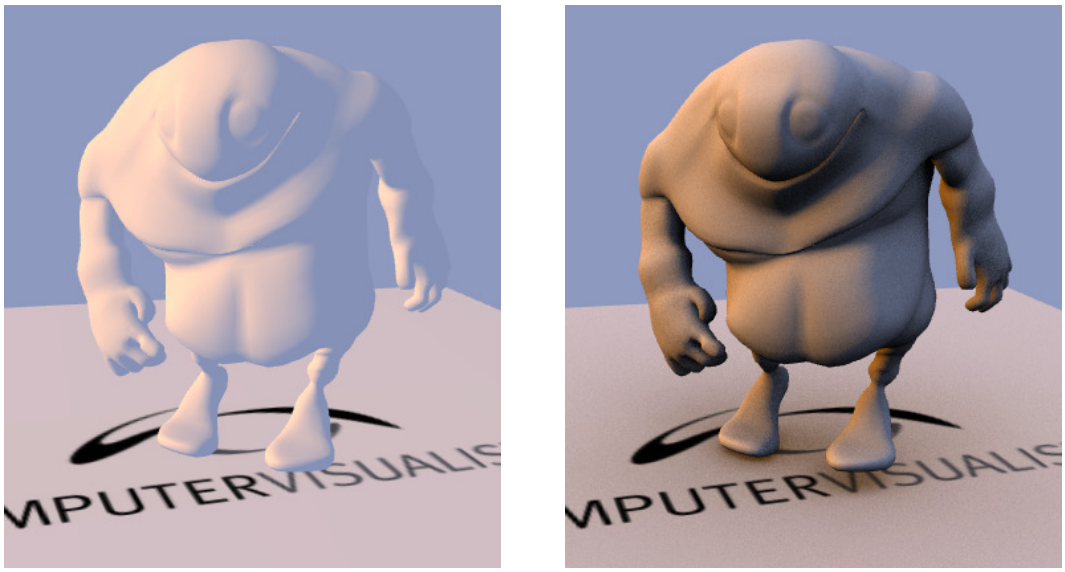
Tabelle 6.1: Daten, Speicherbedarf zur Laufzeit in Megabyte (MB) und mittlere Darstellungszeiten in Frames pro Sekunde (fps) der getesteten Modelle für Ambient Occlusion mit 36 Samples pro Fragment und 64×128 komprimierten Depth Maps mit einer Auflösung von 128×128 Texeln.

dies zeigt, dass sich die Erzeugung der CSM-Dateien auszahlt. Der Speicherbedarf der Anwendung wächst geringfügig mit der Auflösung aufgrund der in Abschnitt 5.3 beschriebenen, für das Sampling nötigen Textur, die jedem Bildschirmpixel einen zufälligen Wert zuweist.

6.2 Bewertung des Verfahrens

Ambient Occlusion zwischen transformierbaren Starrkörpern lässt sich unter Verwendung von Coherent Shadow Maps berechnen. Welchen Vorteil der Einsatz von Ambient Occlusion im Vergleich zu herkömmlichen lokalen Beleuchtungsmodellen bringt, zeigt Abbildung 6.5. Die dargestellte Szene wird von einer Punktlichtquelle mit roter Farbe von links beleuchtet. Eine blaue ambiente Farbe modelliert das Umgebungslicht. Abbildung 6.5(a) zeigt das übliche Vorgehen in klassischen Beleuchtungsmodellen, wobei alle Bereiche, die nicht direkt beleuchtet werden, mit einer konstanten Farbe (hier hellblau) dargestellt werden. An diesen Stellen kann der Betrachter daher keine geometrischen Strukturen erkennen. In Abbildung 6.5(b) wurde der ambiente Term, wie in Gleichung 3.2 auf Seite 17 demonstriert, mit dem Visibilitätsterm für Ambient Occlusion verrechnet. Das Ergebnis erscheint dadurch insgesamt dunkler, da die ambiente Farbe an den meisten Stellen weniger Einfluss besitzt. Der Vorteil zeigt sich auf der rechten Seite des dargestellten Modells, wo sich auch ohne direkte Beleuchtung Strukturen erkennen lassen.

Abbildung 6.6 auf Seite 85 demonstriert die Vorteile, die sich bei einer Kombination von Ambient Occlusion mit einer Beleuchtung auf Basis einer Environment Map ergeben. Die Darstellung des Schattens ermöglicht eine eindeutige Aussage über die



(a) (b)

Abbildung 6.5: Vergleich einer Szene mit lokaler Beleuchtung durch eine Punktlichtquelle (a) ohne und (b) mit Ambient Occlusion durch CSMs.

Anordnung der Objekte innerhalb der Szene und trägt wesentlich dazu bei, dass das Dargestellte als realistisch empfunden wird. In dem gezeigten Beispiel wird der Boden als Occluder angesehen und die Beleuchtung anhand der Bent Normal berechnet. Dadurch ist der Beitrag des größtenteils von unten eintreffenden roten Lichtes wesentlich geringer, sodass sich das Ergebnis auch in dieser Hinsicht deutlich von einer SH-basierten Beleuchtung ohne Berücksichtigung der Visibilität unterscheidet. Der Einsatz der Bent Normal macht sich auch bei der Farbe des Bodens bemerkbar. Während dieser in Abbildung 6.6(a) eine konstante Farbe besitzt, lassen sich in Abbildung 6.6(b) sowohl Bereiche mit roter als auch blauer Färbung ausmachen.

Die Performanz des Verfahrens ist weitgehend unabhängig von der Anzahl darzustellender Polygone und mehrere Instanzen eines Occluder-Typs erfordern nur eine einzige CSM-Textur. Eine genügend große Anzahl an komprimierten Depth Maps mit ausreichend hoher Auflösung vorausgesetzt, lässt sich das Ergebnis nicht von der Referenz durch Ray Tracing unterscheiden. Um Speicherbedarf und Darstellungszeiten gering zu halten existiert die Möglichkeit Diskretisierungsartefakte in Kauf zu nehmen. Besteht die Szene nicht aus einer Vielzahl individueller Occluder lassen sich dann, wie in Abschnitt 6.1 demonstriert, interaktive Frameraten erzielen.

Für samplingbasierte Ansätze ist Rauschen im erzeugten Bild inhärent. Der Einsatz zufällig vermischter niedrig-diskrepanter Sequenzen, wie Halton oder Hammersley, lohnt sich um dieses bereits mit wenigen Samples deutlich zu verringern. Damit ein Rauschen jedoch nicht mehr wahrgenommen werden kann, sind unweigerlich so viele Samples nötig, dass die Darstellungszeiten kaum noch im interaktiven Rahmen liegen.

(a)

(b)

Abbildung 6.6: Vergleich einer Szene mit bildbasierter Beleuchtung durch die Light Probe *Grace Cathedral* (a) ohne und (b) mit Ambient Occlusion durch CSMs.

Der in dieser Arbeit vorgestellte Ansatz zur Berechnung von Ambient Occlusion eignet sich zu diesem Zeitpunkt nicht für einen Einsatz in Graphikanwendungen, die echtzeitfähige Darstellungsraten auf handelsüblichen Computersystemen erzielen müssen. Mit zunehmender Anzahl und Leistungsfähigkeit der Fragment Shader zukünftiger Hardware könnte sich dies in den nächsten Jahren jedoch durchaus ändern. Speziell auf Leistung ausgelegte Systeme können, beispielsweise mit der SLI-Technologie[2] von NVIDIA [Nvi05], mit Sicherheit bereits heute Ambient Occlusion mittels CSMs in Echtzeit darstellen. Den deutlichsten Vorteil scheint der hier vorgestellte Ansatz allerdings in dem Bereich zu finden, der Ambient Occlusion geprägt hat – die Produktion von Filmen mit computergeneriertem Inhalt. Hier ist es auch in aktuellen Produktionen üblich, anstelle von reinem Ray Tracing, Shadow Maps zur Berechnung weicher Schatten einzusetzen, da die Performanz damit um ein Vielfaches höher ist [XTP07]. Der gleiche Grund spricht auch für den Einsatz von Ambient Occlusion (mittels Ray Tracing) anstelle einer globalen Beleuchtungssimulation. Im Zusammenhang mit der Produktion des Animationsfilms ,,Cars“ heißt es [CFLB06]:

> ,,It is worth emphasizing that for production scenes, the dominant cost of ray tracing is typically not the computation of ray intersections, but the evaluation of displacement, surface, and light source shaders at the ray hit points. (This is also why ambient occlusion has gained popularity in movie production so quickly as an alternative to more accurate global illumination solutions: even though it takes a lot of rays to compute ambient occlusion accurately, there are no shader evaluations at the ray hit points.)“

In dieser Arbeit wurde gezeigt wie sich Ambient Occlusion in mit Ray Tracing vergleichbarer Qualität durch den Einsatz von Shadow Maps realisieren lässt. Ließe sich mit diesem Ansatz bei einer Filmproduktion jedes Bild nur wenige Sekunden schneller erzeugen, könnten Produktionskosten und -zeit erheblich gesenkt werden. Die Verwendung des CSM-Dateiformats würde überdies ein Clustering unterstützen. Sind die CSM-Dateien für alle möglichen Occluder einmal erstellt, können beliebig viele Systeme darauf zugreifen um Ambient Occlusion oder andere Formen der Verschattung zu berechnen.

6.3 Einschränkungen und Lösungsansätze

Das im Rahmen dieser Arbeit implementierte System unterliegt zwei wesentlichen Einschränkungen. Wie in Abschnitt 5.4.1 erläutert wurde, werden in CSM-Creator alle Tiefenwerte vor der Komprimierung ermittelt und in Form von STL-Vektoren gespeichert. Dieses Vorgehen erleichtert zwar die Erstellung der CSM-Dateien, andererseits ist der zur Verfügung stehende Arbeitsspeicher auf diese Weise sehr schnell

[2] SLI steht für *Scalable Link Interface* und erlaubt die Koppelung mehrerer Graphikkarten. Insbesondere der Modus *Split Frame Rendering* (SFR), bei dem die Verarbeitung aller darzustellenden Pixel gleichmäßig auf die gekoppelten GPUs aufgeteilt wird, verspricht einen deutlichen Performanzgewinn für das hier beschriebene Vorgehen.

erschöpft, sodass der Kombination von Auflösung und Anzahl der zu komprimierenden Depth Maps eine maximal mögliche Obergrenze gesetzt ist. Deutlich eleganter wäre es etwa, die Depth Maps in Blöcke konstanter Größe zu zerlegen, die der Reihe nach generiert und komprimiert werden. Der benötigte Speicher wäre auf diese Weise konstant, sodass die Größe der Blöcke den Fähigkeiten der zum Einsatz kommenden Hardware entsprechend gewählt werden könnte. Eine andere Möglichkeit besteht darin die Depth Maps direkt bei ihrer Erzeugung *live* zu komprimieren. Dazu müsste für jedes Texel der maximale erste Tiefenwert und der minimale zweite Tiefenwert des aktuellen Segments festgehalten werden. Nach Erzeugung einer neuen Depth Map würden diese Werte wenn nötig aktualisiert werden. Anhand der beiden Tiefenwerte kann für jede neue Depth Map unmittelbar überprüft werden, ob es sich um eine Segment-Grenze handelt. Falls dies der Fall ist, könnte das ermittelte Segment dann sofort in der CSM-Datei abgespeichert werden.

CSM-Loader unterliegt der Einschränkung, dass die Anzahl der zur Laufzeit dargestellten Occluder an verschiedenen Stellen im Code festgelegt werden muss. Insbesondere im Shader-Code sind umfangreiche Änderungen notwendig, um innerhalb eines Render Passes die Verdeckung durch eine neue Anzahl an Occludern zu überprüfen. So ist der in Listing 5.3 auf Seite 68 gezeigte HLSL-Code für die binäre Suche speziell auf eine CSM-Textur zugeschnitten. Sollen dagegen mehrere verschiedenartige Occluder unterstützt werden, muss diese Methode in mehreren Ausführungen vorliegen oder um Abfragen ergänzt werden welche CSM-Textur momentan zu untersuchen ist. Eine denkbare Alternative besteht darin Art und Anzahl der Occluder im C++-Code oder durch Eingaben des Benutzers festzulegen und den entsprechenden Shader-Code dynamisch zu generieren.

6.4 Vergleich zu vorausgegangenen Arbeiten

Abschließend soll ein Vergleich zwischen der Berechnung von Ambient Occlusion mittels CSMs und den bislang veröffentlichten und in Abschnitt 3.3 vorgestellten Arbeiten zu Ambient Occlusion durchgeführt werden.

Die Darstellungsqualität des hier vorgestellten Ansatzes hängt direkt mit der Performanz und den Speicheranforderungen zusammen. Ein wesentlicher Vorteil gegenüber den meisten vorhandenen Verfahren ist, dass die Kriterien unterschiedlich stark gewichtet werden können. Ist eine hohe Qualität des Ergebnisbildes erwünscht, werden unweigerlich viele Samples generiert werden müssen um ein Rauschen zu reduzieren und eine Vielzahl von Depth Maps müssen komprimiert werden um Diskretisierungsartefakte zu vermeiden. Soll die Berechnung von Ambient Occlusion dagegen möglichst performant sein, ist ein Rauschen aufgrund weniger Samples unvermeidbar. Steht nur wenig Speicher zur Verfügung, müssen Artefakte in Kauf genommen werden, da nur wenige Depth Maps mit geringerer Auflösung komprimiert werden können. Aus diesem Grund fällt eine allgemeine Bewertung wie sie in Tabelle 3.1 auf Seite 21 vorgenommen wurde, schwer. Einen Versuch das Verfahren für die in dieser Arbeit üblicherweise zum Einsatz kommende Konfiguration von 36

Typ	Sicht	Qualität	Performanz	Speicher	Dynamik
Objekt	inside-out	+	–	–	–

Tabelle 6.2: Bewertung von Ambient Occlusion mittels CSMs (vgl. Tabelle 3.1).

Samples und 64×128 Depth Maps mit einer Auflösung von jeweils 128×128 Texel zu bewerten, zeigt Tabelle 6.2.

Die konzeptionell zu dem hier beschriebenen Vorgehen ähnlichen Verfahren von Kontkanen und Laine [KL05] und Malmer et al. [MMAH05] besitzen den Nachteil, dass sie Ambient Occlusion vor allem im Bereich harter Kontaktschatten relativ grob annähern, dafür erlauben sie jedoch eine deutlich bessere Performanz. Der Ray Tracing-basierte Ansatz von Cadet und Lécussan [CL07] besitzt eine ähnliche Performanz wie Ambient Occlusion mittels CSMs, allerdings wird eine große Menge an Informationen zur Visibilität verwaltet, wodurch der benötigte Speicher höher sein dürfte. Ambient Occlusion nach Pharr und Green [PG04] benötigt deutlich mehr Rechenzeit um ein vergleichbares Ergebnis zu liefern, allerdings wird keine Information vorberechnet, wodurch der Bedarf an Speicher minimal ist, und Szenen mit mehreren Occludern erhöhen den Aufwand nicht. Alle vertexbasierten Verfahren [SSZK04, Bun05, HPAD06, GSF07, HJ07] weisen eine bessere Performanz auf und sind, im Gegensatz zum CSM-basierten Ansatz, zumeist nicht auf Starrkörper beschränkt – allerdings liefern sie von vornherein nur eine grobe Annäherung an Ambient Occlusion oder benötigen sehr hoch aufgelöste Geometrie um eine mit Ray Tracing vergleichbare Ausgabequalität zu erzielen. Rein bildbasierte Verfahren, wie das von Mittring [Mit07] beschriebene, können kein korrektes Ergebnis für Ambient Occlusion liefern, da sie das globale Phänomen, dass sich alle Elemente einer Szene gegenseitig verdecken können, auf den durch die Kamera sichtbaren Bereich der Szene beschränken. Shanmugam und Arikan [SA07] berücksichtigen dagegen zwar auch nicht unmittelbar sichtbare Occluder, approximieren diese jedoch durch Kugeln. Der Vorteil bildbasierter Methoden liegt jedoch darin, dass animierte Objekte gegenüber Starrkörpern keine zusätzliche Herausforderung darstellen.

7 Zusammenfassung und Ausblick

In der vorliegenden Diplomarbeit wurde gezeigt, wie sich Ambient Occlusion in einer Szene aus frei transformierbaren Starrkörpern unter Verwendung von Coherent Shadow Maps in interaktiven Frameraten berechnen und darstellen lässt. Die Vorteile von Ambient Occlusion im Vergleich zu klassischen lokalen Beleuchtungsmodellen wurden aufgezeigt – den Ansatzpunkt bildet die Approximation von Umgebungslicht und indirekter Beleuchtung durch einen konstanten Farbwert – und die vereinfachenden Annahmen, die im Vergleich zu einer Simulation globaler Beleuchtung gemacht werden, wurden nachgezeichnet – Die Gültigkeit von Ambient Occlusion beschränkt sich demnach eigentlich auf eine direkte Beleuchtung diffuser Materialien mit konstanter Leuchtdichte. Um einen umfassenden Überblick zu gewährleisten, wurden zahlreiche existierende Ansätze zur Berechnung von Ambient Occlusion ausführlich vorgestellt. Anschließend wurde auf die Berechnung der Visibilität mittels CSMs eingegangen. Es wurde gezeigt wie sich die Komprimierungsrate von CSMs durch eine gleichmäßige Verteilung der zu komprimierenden Depth Maps deutlich erhöhen lässt und ein XML-konformes Dateiformat, das die beliebige Wiederverwendung einmal komprimierter Depth Maps erlaubt, wurde eingeführt. Das für einen sampling-basierten Ansatz typische Rauschen konnte durch den Einsatz von Quasi-Monte-Carlo-Techniken deutlich herabgesetzt werden kann. Die optischen Vorteile von Ambient Occlusion wurden durch Kombination mit einer bildbasierten Beleuchtung auf Basis von Spherical Harmonics präsentiert. Das im Rahmen dieser Arbeit implementierte System, das sich in ein Programm zur Komprimierung von Depth Maps und ein Programm, welches die komprimierten Daten zur Darstellung von Ambient Occlusion nutzt, aufteilt, wurde vorgestellt. Zuletzt wurde die Berechnung von Ambient Occlusion mittels CSMs vorangegangenen Ansätzen gegenübergestellt, woraufhin die mit Ray Tracing vergleichbare Darstellungsqualität als besondere Stärke des Verfahrens hervorgehoben werden konnte.

Ansatzpunkte zur Behebung von Einschränkungen, die speziell für das im Rahmen dieser Arbeit implementierte System gelten, wurden in Abschnitt 6.3 genannt. Darüber hinaus lohnt sich die Entwicklung von Vorgehensweisen zur gezielten Vermeidung der bei einem Einsatz von CSMs typischen Artefakte (vgl. Abschnitt 4.3).

So führt eine zu geringe Menge an komprimierten Depth Maps unweigerlich zu Banding-Artefakten. In dieser Arbeit wurde sich stets auf eine Anzahl an Depth Maps beschränkt, deren einzige Plausibilität ein optisch akzeptables Ergebnis ist. Es ist jedoch möglich anhand der Auflösung auszurechnen, wie groß der Winkel zwischen den einzelnen Depth Maps und damit ihre Anzahl sein muss, damit der Raum um einen Occluder derart diskretisiert wird, dass innerhalb eines festen Radius keine Banding-Artefakte mehr möglich sind. Dieses Vorgehen würde zwar bei ausreichen-

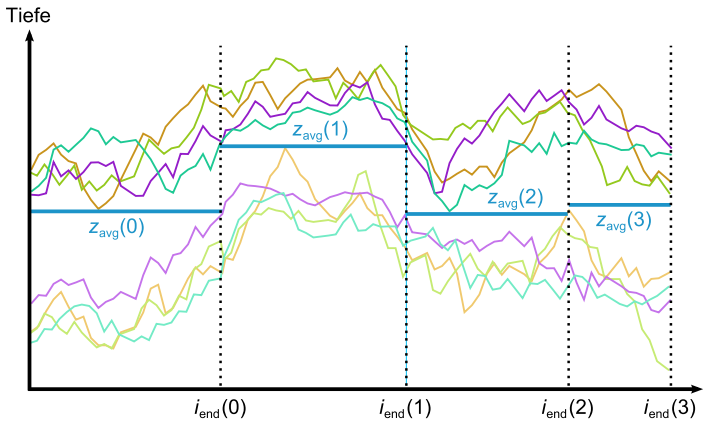


Abbildung 7.1: Für vier kohärente Texel gemeinsam gültige Segmente.

der Auflösung eine enorm hohe Menge an Depth Maps erfordern, gleichzeitig würde allerdings die Komprimierungsrate steigen. Die Verteilung der Depth Maps könnte darüber hinaus auch adaptiv geschehen, beispielsweise anhand der Objekt-Topologie oder auf Basis von Vorwissen über mögliche und unmögliche Ausrichtungen eines Occluders innerhalb der Szene. Auf diese Weise könnte gezielt die räumliche Auflösung der Diskretisierung in solchen Bereichen, die für Banding-Artefakte anfällig sind, erhöht werden.

Die zweite Sorte typischer Artefakte sind fehlende Schattenbereiche aufgrund zu niedrig aufgelöster Depth Maps. Die Tatsache, dass die Komprimierung durch CSMs lediglich die Kohärenz zwischen Depth Maps und nicht zwischen den Texeln der einzelnen Depth Maps ausnutzt, spricht bislang gegen den Einsatz einer hohen Auflösung. Wird diese nämlich erhöht, so führt dies unweigerlich zu einer proportionalen Steigerung des Speicherbedarfs, da die Anzahl der bei der Komprimierung erstellten Segmentlisten in jedem Fall der Anzahl an Texeln entspricht. Diesem ungünstigen Verhalten könnte etwa dadurch entgegengewirkt werden, dass für einen Abschnitt aufeinanderfolgender Depth Maps eine Menge kohärenter (also beispielsweise benachbarter) Texel gesucht wird, für die sich ein gemeinsam nutzbarer Segmentlisten-Abschnitt erstellen lässt. Wie Abbildung 7.1 demonstriert, könnten die Segment-Grenzen dafür so gewählt werden, dass der maximale erste Tiefenwert aller kohärenten Texel innerhalb des Segments kleiner ist als der minimale zweite Tiefenwert der entsprechenden Texel.

Es ist davon auszugehen, dass das Potential von CSMs auch mit der Umsetzung dieser Ideen noch nicht erschöpft ist. Denkbar ist beispielsweise die Aufhebung der Einschränkung auf Starrkörper, indem der Komprimierung eine zusätzliche Dimension hinzugefügt wird. Mehrere CSMs für Referenzposen eines animierten Modells weisen Ähnlichkeiten auf, die sich für eine weitergehende Komprimierung nutzen ließen. Die Visibilität eines Punktes zu einer beliebigen Pose könnte dann durch eine geeignete Interpolation der Referenzdaten gewonnen werden.

Die Einführung eines CSM-Dateiformats gibt Aufschluss darüber, wie die Berechnung von Schatten in Zukunft aussehen könnte. Wie auch Materialeigenschaften oder Texturen, lässt sich die Visibilität um ein Objekt in einer Datei abspeichern, auf die von allen Instanzen eines Occluder-Typs Bezug genommen wird. Aus dieser Sichtweise ist es denkbar, dass Graphikbibliotheken anhand derartiger Daten die Darstellung beliebiger Arten von Schatten mit einem einzigen Befehl ermöglichen könnten. Dass sich die Information in einer CSM-Datei nicht bloß zur Berechnung des Schattens durch unendlich weit entferntes Umgebungslicht, sondern ebenso durch lokale Lichtquellen beliebiger Form eignet, haben Ritschel et al. [RGKM07] bereits mit dem Konzept semilokaler Lichtquellen gezeigt.

Ambient Occlusion kommt in Bereichen zum Einsatz, in denen die Darstellungsqualität lokaler Beleuchtung durch einzelne Punktlichtquellen nicht ausreicht, aber die Simulation globaler Beleuchtung einen zu großen Aufwand darstellt. In einer der ersten Veröffentlichungen zu Ambient Occlusion schrieb Landis [Lan02]:

> ,,I am sure that at some point we will be ray tracing complex scenes on our palm pilots. Until then we'll continue to create efficient cheats and tricks to get the visual advantages of global illumination without the time and expense.“

Wie in dieser Arbeit gezeigt wurde, ermöglicht der technische Fortschritt in Kombination mit Vorgehensweisen, die speziell die Fähigkeiten der Graphikhardware ausnutzen, heute bereits die dynamische Berechnung von Ambient Occlusion in interaktiven Frameraten. In den nächsten Jahren könnte Ambient Occlusion schon in einer Vielzahl echtzeitfähiger Graphikanwendungen eingesetzt werden. Langfristig gesehen stellt es jedoch nur eine Zwischenstufe zur globalen Beleuchtungssimulation in Echtzeit dar. Bis dieses Ziel allerdings erreicht ist, lohnt sich nach wie vor die Schaffung neuer Ansätze und die Weiterentwicklung vorhandener Verfahren um die Lücke zwischen der photorealistischen und der echtzeitfähigen Computergraphik zunehmend zu schließen.

Modellverzeichnis

Die in Abbildung 2.3, 2.4, 3.5, 4.1, 4.3, 5.1, 5.12 und 6.2, sowie Tabelle 5.1 gezeigten oder angedeuteten Modelle wurden von der Stanford University erzeugt und freigegeben.
`http://graphics.stanford.edu/data/3Dscanrep/` (Abruf: 16.04.2008)

Die Modelle, die in Abbildung 2.5, 3.1, 5.4, 5.6, 5.11, 5.14, 6.1, 6.4, 6.5 und 6.6 zu sehen sind, sind Teil des NVIDIA SDK 9 (`http://developer.nvidia.com/object/sdk-9.html`, Abruf: 16.04.2008). Das Copyright unterliegt der NVIDIA Corporation.

Das in Abbildung 3.3(a) und 3.8 angedeutete Modell wurde von Martin Newell an der University of Utah entwickelt.

Das in Abbildung 3.4 angedeutete Modell wurde von Martin Rezard entworfen und von headus (metamorphosis) Pty Ltd. eingescannt.

Das in Abbildung 5.2 angedeutete und in Abbildung 6.3 gezeigte Modell wurde von Viewpoint DataLabs zur Verwendung freigegeben.

Das Copyright der in Abbildung 5.12, 5.14, 6.1, 6.4 und 6.6 gezeigten, oder zum Einsatz kommenden Light Probes unterliegt Paul Debevec.
`http://www.debevec.org/Probes/` (Abruf: 16.04.2008)

Literaturverzeichnis

[Bie04] Biedermann M.: *Hybrides Verfahren zur realistischen, echtzeitfähigen Beleuchtung virtueller Szenen unter Berücksichtigung dynamischer Objekte auf der Basis von Spherical Harmonics*. Diplomarbeit, Universität Koblenz-Landau, 2004.

[Bun05] Bunnell M.: Dynamic Ambient Occlusion and Indirect Lighting. In: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Pharr M., (Hrsg.), S. 223–233. Addison-Wesley, Boston, MA, USA, u.a., 2005.

[CL07] Cadet G., Lécussan B.: Fast Approximate Ambient Occlusion. Poster bei SIGGRAPH 2007 (2007).

[CNS00] Castro F., Neumann L., Sbert M.: Extended Ambient Term. In: *Journal of Graphics Tools 5*, 4 (November 2000). A K Peters, Ltd., Natick, MA, USA, S. 1–7.

[Chr03] Christensen P. H.: Global Illumination and All That. In: *ACM SIGGRAPH 2003 Course 9: Renderman, Theory and Practice* (Juli 2003), Batali D., (Hrsg.). ACM, New York, NY, USA, S. 31–72.

[CFLB06] Christensen P. H., Fong J., Laur D. M., Batali D.: Ray Tracing for the Movie 'Cars'. In: *Proceedings of the IEEE Symposium on Interactive Ray Tracing 2006* (September 2006). IEEE Computer Society, S. 1–6.

[Coo86] Cook R. L.: Stochastic Sampling in Computer Graphics. In: *ACM Transactions on Graphics 5*, 1 (Januar 1986). ACM, New York, NY, USA, S. 51–72.

[CT82] Cook R. L., Torrance K. E.: A Reflectance Model for Computer Graphics. In: *ACM Transactions on Graphics 1*, 1 (Januar 1982). ACM, New York, NY, USA, S. 7–24.

[CPC84] Cook R. L., Porter T., Carpenter L.: Distributed Ray Tracing. In: *ACM SIGGRAPH Computer Graphics 18*, 3 (Juli 1984). ACM, New York, NY, USA, S. 137–145.

[Cro77] Crow F. C.: Shadow Algorithms for Computer Graphics. In: *ACM SIGGRAPH Computer Graphics 11*, 2 (Juli 1977). ACM, New York, NY, USA, S. 242–248.

[DBL07] Döllner J., Buchholz H., Lorenz H.: Ambient Occlusion – ein

Schritt zur realistischen Beleuchtung von 3D-Stadtmodellen. In: *GIS-BUSINESS 11* (November 2006), S. 7–13.

[DKTS07] Dong Z., Kautz J., Theobalt C., Seidel H.-P.: Interactive Global Illumination Using Implicit Visibility. In: *PG '07: Proceedings of the 15th Pacific Graphics Conference on Computer Graphics and Applications* (Oktober 2007). IEEE Computer Society, S. 77–86.

[DBB06] Dutré P., Bala K., Bekaert P.: *Advanced Global Illumination*. A K Peters, Ltd., Wellesley, MA, USA, 2. Auflage, 2006.

[Eve01] Everitt C.: *Interactive Order-Independent Transparency*. NVIDIA, 2001.
`http://developer.nvidia.com/object/interactive_order_transparency.html` (Abruf: 16.04.2008)

[FK03] Fernando R., Kilgard M. J.: *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, Boston, MA, USA, 2003.

[GSF07] González F., Sbert M., Feixas M.: An Information-Theoretic Ambient Occlusion. In: *Computational Aesthetics in Graphics, Visualization, and Imaging* (2007), Cunningham D. W., Meyer G., Neumann L., (Hrsg.). The Eurographics Association, S. 29–36.

[GTGB84] Goral C. M., Torrance K. E., Greenberg D. P., Battaile B.: Modeling the Interaction of Light Between Diffuse Surfaces. In: *ACM SIGGRAPH Computer Graphics 18*, 3 (1984). ACM, New York, NY, USA, S. 213–222.

[Gou71] Gouraud H.: Continuous Shading of Curved Surfaces. In: *IEEE Transactions on Computers 20*, 6 (Juni 1971). IEEE Computer Society, S. 623–629.

[Gre03] Green R.: *Spherical Harmonics Lighting: The Gritty Details*. Sony Computer Entertainment America, 2003.
`http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.html` (Abruf: 16.04.2008)

[HSA91] Hanrahan P., Salzman D., Aupperle L.: A rapid hierarchical radiosity algorithm. In: *SIGGRAPH '91: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques* (1991). ACM, New York, NY, USA, S. 197–206.

[HLHS03] Hasenfratz J.-M., Lapierre M., Holzschuch N., Sillion F.: A Survey of Real-time Soft Shadows Algorithms. In: *Computer Graphics Forum 22*, 4 (Dezember 2003). Blackwell Publishers, Oxford, UK, u.a., S. 753–774.

[HPAD06] Hegeman K., Premo¸e S., Ashikhmin M., Drettakis G.: Approximate ambient occlusion for trees. In: *I3D '06: Proceedings of the 2006*

Symposium on Interactive 3D Graphics and Games (2006), Sequin C., Olano M., (Hrsg.). ACM, New York, NY, USA, S. 87–92.

[HJ07] Hoberock J., Jia Y.: High-Quality Ambient Occlusion. In: *GPU Gems 3*, Nguyen H., (Hrsg.), S. 257–274. Addison-Wesley, Upper Saddle River, NJ, USA, u.a., 2007.

[IKSZ03] Iones A., Krupkin A., Sbert M., Zhukov S.: Fast realistic lighting for video games. In: *IEEE Computer Graphics and Applications 23*, 3 (Mai 2003). IEEE Computer Society Press, Los Alamitos, CA, USA, S. 54–64.

[JC95] Jensen H. W., Christensen N. J.: Photon Maps in Bidirectional Monte Carlo Ray Tracing of Complex Objects. In: *Computers & Graphics 19*, 2 (März 1995), S. 215–224.

[Kaj86] Kajiya J. T.: The rendering equation. In: *ACM SIGGRAPH Computer Graphics 20*, 4 (August 1986). ACM, New York, NY, USA, S. 143–150.

[KLA04] Kautz J., Lehtinen J., Aila T.: Hemispherical Rasterization for Self-Shadowing of Dynamic Objects. In: *15th Eurographics Symposium on Rendering* (Juni 2004), Jensen H.W., Keller A., (Hrsg.). The Eurographics Association, S. 179–184.

[KA07] Kirk A. G., Arikan O.: Real-Time Ambient Occlusion for Dynamic Character Skins. In: *I3D '07: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (2007). ACM, New York, NY, USA, S. 47–52.

[Kne07] Knecht M.: *State of the Art Report on Ambient Occlusion.* Seminararbeit, Technische Universität Wien, 2007.
`http://www.cg.tuwien.ac.at/courses/Seminar/WS2007/AmbientOcclusion-Knecht.pdf` (Abruf: 16.04.2008)

[KK02a] Kollig T., Keller A.: Efficient Bidirectional Path Tracing by Randomized Quasi-Monte Carlo Integration. In: *Monte Carlo and Quasi-Monte Carlo Methods 2000* (2002), Niederreiter H., Fang K., Hickernell F., (Hrsg.). Springer, S. 290–305.

[KK02b] Kollig T., Keller A.: Efficient Multidimensional Sampling. In: *Computer Graphics Forum 21*, 3 (September 2002). Blackwell Publishers, Oxford, UK, u.a., S. 557–563.

[KL05] Kontkanen J., Laine S.: Ambient Occlusion Fields. In: *I3D '05: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (2005). ACM, New York, NY, USA, S. 41–48.

[KA06] Kontkanen J., Aila T.: Ambient Occlusion for Animated Characters. In: *Rendering Techniques 2006 (Proceedings of the 17th Eurographics Workshop on Rendering)* (Juni 2006), Akenine-Möller T., Heidrich W., (Hrsg.). The Eurographics Association, S. 343–348.

[KK07] Kozlowski O., Kautz J.: Is Accurate Occlusion of Glossy Reflections Necessary? In: *APGV '07: Proceedings of the 4th Symposium on Applied Perception in Graphics and Visualization* (Juli 2007). ACM, New York, NY, USA, S. 91–98.

[Lan02] Landis H.: Production-Ready Global Illumination. In: *ACM SIGGRAPH 2002 Course 16: RenderMan in Production* (Juli 2002). ACM, New York, NY, USA, S. 87–102.

[LB99] Langer M. S., Bülthoff H. H.: *Perception of shape from shading on a cloudy day.* Technical Report No. 73, Max-Planck-Institut für biologische Kybernetik, 1999.

[LB00] Langer M. S., Bülthoff H. H.: Depth discrimination from shading under diffuse lighting. In: *Perception 29*, 6 (2000), S. 649–660.

[MMAH05] Malmer M., Malmer F., Assarsson U., Holzschuch N.: *Fast Precomputed Ambient Occlusion for Proximity Shadows.* Technical Report RR-5779, INRIA, 2005.

[MSC03] Méndez A., Sbert M., Català J.: Real-time Obscurances with Color Bleeding. In: *SCCG '03: Proceedings of the 19th Spring Conference on Computer Graphics* (April 2003), Szirmay-Kalos L., (Hrsg.). ACM, New York, NY, USA, S. 171–176.

[Men05] mental images GmbH: *Occlusion Tutorial*, 2005. `http://lamrug.org/resources/doc/occlusion_tutorial.pdf` (Abruf: 16.04.2008)

[Mic08] Microsoft Corporation: Coordinate Systems. In: *MSDN Library.* Microsoft Corporation, 2008. `http://msdn2.microsoft.com/en-us/library/bb204853.aspx` (Abruf: 16.04.2008)

[Mit07] Mittring M.: Finding Next Gen - CryEngine 2. In: *ACM SIGGRAPH 2007 Course 28: Advanced Real-Time Rendering in 3D Graphics and Games* (2007). ACM, New York, NY, USA, S. 97–121.

[Nvi05] NVIDIA Corporation: *NVIDIA GPU Programming Guide*, 2005. `http://developer.nvidia.com/object/gpu_programming_guide.html` (Abruf: 16.04.2008)

[PG04] Pharr M., Green S.: Ambient Occlusion. In: *GPU Gems*, Fernando R., (Hrsg.), S. 279–292. Addison-Wesley, Boston, MA, USA, u.a., 2004.

[Pho75] Phong B.-T.: Illumination for Computer Generated Pictures. In: *Communications of the ACM 18*, 6 (Juni 1975). ACM, New York, NY, USA, S. 311–317.

[RH01] Ramamoorthi R., Hanrahan P.: An Efficient Representation for Irradiance Environment Maps. In: *SIGGRAPH '01: Proceedings of the 28th*

Annual Conference on Computer Graphics and Interactive Techniques (August 2001). ACM, New York, NY, USA, S. 497–500.

[RSC87] Reeves W. T., Salesin D. H., Cook R. L.: Rendering Antialiased Shadows with Depth Maps. In: *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (1987). ACM, New York, NY, USA, S. 283–291.

[RGKM07] Ritschel T., Grosch T., Kautz J., Müller S.: Interactive Illumination with Coherent Shadow Maps. In: *EGSR 2007: Eurographics Symposium on Rendering* (2007), Kautz J., Pattanaik S., (Hrsg.). The Eurographics Association, S. 61–72.

[RGKS08] Ritschel T., Grosch T., Kautz J., Seidel H.-P.: Interactive Global Illumination Based on Coherent Surface Shadow Maps. Soll erscheinen in: *Proceedings of Graphics Interface* (Mai 2008).
`http://www.uni-koblenz.de/~ritschel/` (Abruf: 16.04.2008)

[RMD*08] Ropinski T., Meyer-Spradow J., Diepenbrock S., Mensmann J., Hinrichs K. H.: Interactive Volume Rendering with Dynamic Ambient Occlusion and Color Bleeding. Soll erscheinen in: *Computer Graphics Forum 27*, 2 (2008).
`http://viscg.uni-muenster.de/publications/2008/RMDMH08/` (Abruf: 16.04.2008)

[SSZK04] Sattler M., Sarlette R., Zachmann G., Klein R.: Hardware-accelerated ambient occlusion computation. In: *Vision, Modeling, and Visualization (VMV)* (2004), Girod B., Magnor M., Seidel H.-P., (Hrsg.). Akademische Verlagsgesellschaft Aka GmbH, Berlin, Deutschland, S. 331–338.

[SA07] Shanmugam P., Arikan O.: Hardware Accelerated Ambient Occlusion Techniques on GPUs. In: *I3D '07: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (2007). ACM, New York, NY, USA, S. 73–80.

[Shi02] Shirley P.: *Fundamentals of Computer Graphics*. A K Peters, Ltd., Natick, MA, USA, 2002.

[SKS02] Sloan P.-P., Kautz J., Snyder J.: Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. In: *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (Juli 2002). ACM, New York, NY, USA, S. 527–536.

[Sou07] Sousa T.: Vegetation Procedural Animation and Shading in Crysis. In: *GPU Gems 3*, Nguyen H., (Hrsg.), S. 373–385. Addison-Wesley, Upper Saddle River, NJ, USA, u.a., 2007.

[SD02] Stamminger M., Drettakis G.: Perspective Shadow Maps. In: *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer*

Graphics and Interactive Techniques (Juli 2002). ACM, New York, NY, USA, S. 557–562.

[SBB*06] Stephens A., Boulos S., Bigler J., Wald I., Parker S.: An Application of Scalable Massive Model Interaction using Shared-Memory Systems. In: *Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization* (2006), Heidrich A., Raffin B., dos Santos L. P., (Hrsg.). The Eurographics Association, S. 19–26.

[Ste03] Stewart A. J.: Vicinity Shading for Enhanced Perception of Volumetric Data. In: *VIS '03: Proceedings of the 14th IEEE Visualization 2003* (Oktober 2003). IEEE Computer Society, Washington DC, USA, S. 355–362.

[TCM06] Tarini M., Cignoni P., Montani C.: Ambient Occlusion and Edge Cueing to Enhance Real Time Molecular Visualization. In: *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (September/Oktober 2006). IEEE Educational Activities Department, Piscataway, NJ, USA, S. 1237–1244.

[Whi80] Whitted T.: An Improved Illumination Model for Shaded Display. In: *Communications of the ACM 23*, 6 (1980). ACM, New York, NY, USA, S. 343–349.

[Wil78] Williams L.: Casting Curved Shadows on Curved Surfaces. In: *ACM SIGGRAPH Computer Graphics 12*, 3 (August 1978). ACM, New York, NY, USA, S. 270–274.

[Wym04] Wyman C. R.: *Fast Local Approximation to Global Illumination*. Dissertation, The University of Utah, 2004.

[XTP07] Xie F., Tabellion E., Pearce A.: Soft Shadows by Ray Tracing Multilayer Transparent Shadow Maps. In: *EGSR 2007: Eurographics Symposium on Rendering* (2007), Kautz J., Pattanaik S., (Hrsg.). The Eurographics Association, S. 265–276.

[ZHXL06] Zhang F., Hanqiu S., Xu L., Lee K.-L.: Parallel-Split Shadow Maps for Large-Scale Virtual Environments. In: *VRCIA '06: Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications* (2006). ACM, New York, NY, USA, S. 311–318.

[ZHL*05] Zhou K., Hu Y., Lin S., Guo B., Shum H.-Y.: Precomputed Shadow Fields for Dynamic Scenes. In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, (2005). ACM, New York, NY, USA, S. 1196–1201.

[ZIK98] Zhukov S., Iones A., Kronin G.: An Ambient Light Illumination Model. In: *Rendering Techniques '98 (Proceedings of the 9th Eurographics Workshop on Rendering)* (1998), Drettakis G., Max N., (Hrsg.). Springer-Verlag, Wien, New York, S. 45–56.