UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Voxelbasierte globale Beleuchtung in dynamischen Szenen

Diplomarbeit

zur Erlangung des Grades einer Diplom-Informatikerin
im Studiengang Computervisualistik

vorgelegt von
Sinje Thiedemann

Erstgutachter: Prof. Dr. Stefan Müller
Institut für Computervisualistik, AG Computergraphik

Zweitgutachter: Dipl.-Inform. Niklas Henrich
Institut für Computervisualistik, AG Computergraphik

Koblenz, im August 2010

Erklärung

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

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

.....

(Ort, Datum) (Unterschrift)

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

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Diplomarbeit
Sinje Thiedemann (Matr.-Nr. 205 210 268)

Thema: Voxelbasierte globale Beleuchtung in dynamischen Szenen

Für die realistische Beleuchtung einer virtuellen Szene spielt neben der direkten Beleuchtung auch die Ausbreitung des indirekten Lichts eine wichtige Rolle. Die Berechnung der indirekten Beleuchtung benötigt grundsätzlich Informationen über die gesamte Szene, nicht nur über den für die Kamera sichtbaren Ausschnitt, der in bildraumbasierten Techniken zum Einsatz kommt. Mittels Voxelisierung kann die Szene in eine dreidimensionale, diskrete und GPU-freundliche Repräsentation überführt werden.

In dieser Diplomarbeit werden Voxelrepräsentationen hinsichtlich ihrer Eignung für den globalen Lichtaustausch in dynamischen und großen Szenen untersucht.

Zunächst soll geprüft werden, welche Verfahren sich am besten eignen, um eine Szene in eine Voxelrepräsentation zu überführen. Ferner wird ermittelt, ob es möglich ist, nur bestimmte Teile der Szene zu voxelisieren, um eine adaptive Struktur und Speichervorteile zu erreichen. Anschließend werden Algorithmen untersucht, mit denen sich Schnitttests innerhalb der Voxelstruktur performant durchführen lassen. Darauf aufbauend soll geprüft werden, inwieweit sich globale Monte-Carlo Beleuchtungsalgorithmen integrieren lassen. Zusätzlich liegt der Fokus auf einem geschickten Sampling der indirekten Beleuchtung um temporale Konsistenz (Vermeidung von springendem oder flackerndem Licht bei Objekt- und Kamerabewegungen) in dynamischen Szenen zu erhalten.

Schwerpunkte dieser Arbeit sind:

1. Literaturrecherche zum Thema voxelbasierte globale Beleuchtung
2. Entwicklung von Verfahren zur Voxelisierung großer dynamischer Szenen
3. Ausnutzung der voxelbasierten Szenenrepräsentation für globale Beleuchtungseffekte
4. Evaluierung der Ergebnisse anhand verschiedener dynamischer Szenen in Hinblick auf Qualität und Performance
5. Dokumentation der Ergebnisse

Koblenz, den 01. März 2010

- Prof. Dr. Stefan Müller-

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

In der realen Welt ist es tagtäglich zu sehen: Diverse Lichtquellen emittieren Licht, welches mit Oberflächen und Materialien der Objekte in der Welt interagiert. Es wird oftmals unzählige Male reflektiert oder gebrochen und kann sich in verschiedenen Medien ausbreiten, bis es schließlich in das menschliche Auge fällt. Was der Mensch mit seinem Sehsinn wahrnimmt, ist somit das Ergebnis eines Lichtausbreitungsprozesses.

Für die Computergrafik ist die Nachbildung dieses Prozesses immer noch ein herausforderndes Problem, wenn die Lichtberechnung in der virtuellen Szene nur wenige Sekunden oder gar Millisekunden dauern soll. Die zum Einsatz kommenden Verfahren sind globale Beleuchtungsalgorithmen, bei denen neben den lokalen Eigenschaften eines Punktes auch der Rest der Szene herangezogen wird, um die Beleuchtung zu berechnen. Entsprechend rechenaufwändig sind die Verfahren. Daher ist es oft nötig, sich auf einige der real existierenden Phänomene zu beschränken und Abstriche in der Genauigkeit der Berechnung zu machen.

Der Teilbereich der Computergrafik, der sich mit der Erzeugung realistischer Bilder beschäftigt, wird fotorealistische Computergrafik genannt. Allerdings stellten [Cohen et al. 1993, S. 2] schon vor über anderthalb Jahrzehnten die Frage, ob das Ziel der Bildsynthese nicht höher als „fotorealistisch“ gesteckt sein sollte. Denn auch in der Fotografie gibt es Faktoren, die den Realismus eines Bildes limitieren: Eigenschaften der Kamera wie Linsen oder das Aufnahmemedium. Ein Bild zu erzeugen, das der visuellen Wahrnehmung des Menschen entspricht, wäre wohl das Optimum. Darüber hinaus gibt es Bestrebungen, physikalisch korrekte und verlässliche Ergebnisse zu liefern.

Neben der Glaubwürdigkeit bzw. Plausibilität für die menschliche Wahrnehmung spielen auch die Faktoren Geschwindigkeit und Flexibilität der Simulation eine bedeutende Rolle. Jedes Anwendungsgebiet (zum Beispiel Architekturvisualisierung, Produktdesign, Unterhaltungsindustrie) hat verschiedene Anforderungen. Eine „schnelle“ Berechnung kann bedeuten: interaktive Frameraten (das heißt, mindestens 6 Bilder pro Sekunde können angezeigt werden) oder „Echtzeit“ für Anwendungen wie Computerspiele, bei denen die Rechenleistung nicht nur für die Grafik, sondern auch für die Spielmechanik benötigt wird.

Für komplett statische Szenen kann die aufwändige globale Beleuchtung vorberechnet und in Echtzeit angezeigt werden. Sobald jedoch dynamische

Elemente hinzukommen, ist eine Vorberechnung nicht ohne Weiteres möglich: Potentiell kann jeder Punkt der Szene die Beleuchtungssituation jedes anderen Punktes beeinflussen. Es sind also zu jedem Zeitpunkt Informationen über den Zustand der Szene als Grundlage für die Berechnung notwendig.

Verfahren, die den Bildraum als Grundlage nutzen, haben nur sehr eingeschränkte Informationen zur Verfügung, nämlich nur die Punkte, die die Kamera direkt sehen kann. Geometrie, die sich zum Beispiel hinter dem Betrachter, außerhalb seines Blickfeldes oder hinter anderen Objekten befindet, wird ignoriert und beeinflusst die Beleuchtung nicht. In der Realität hat aber auch Geometrie, die nicht unmittelbar sichtbar ist, einen großen Einfluss auf die Beleuchtung der sichtbaren Geometrie. Doch die starke Vereinfachung hat den Vorteil, dass die Verfahren wegen des reduzierten Rechenaufwandes in PC-Spielen einsetzbar sind.

Neben dem Bildraum gibt es auch andere Szenenrepräsentationen, die als Texturen gespeichert werden können und somit für die Verarbeitung mit Grafikkarten geeignet sind. Diese Diplomarbeit nutzt Voxeldatenstrukturen als diskrete Annäherung der Inhalte des dreidimensionalen Raums. Sie werden hinsichtlich ihrer Eignung für eine effiziente Lichtsimulation auf der Grafikkarte untersucht. Die Voxel dienen Lichtberechnungsshadern als Informationsquelle für Abfragen wie „Ist dort etwas in der Szene?“ oder „Was ist dort in der Szene?“. Neben binären Voxelinformationen, welche die erstgenannte Frage beantworten können, liefern nicht-binäre Voxelisierungen genauere Dateninhalte für den angefragten Ort. Solche räumlichen Datenstrukturen lassen sich sehr intuitiv bei der Implementierung nutzen, da ein direkter Zusammenhang zwischen dem Speicherort und dem realen Ort der Daten in der Szene besteht [Bresink 1999, S. 75]. Die Erwartung, dass die Verwendung der groben Repräsentation eine schnelle globale Beleuchtungsberechnung mit qualitativ ansprechenden Bildern ermöglicht, wurde in dieser Arbeit bestätigt (Beispiel-Ergebnisbild in Abbildung 1).

1.2 Aufbau der Diplomarbeit

Nach dieser Einführung erläutert Kapitel *2 Grundlagen und Begriffe* die im Titel der Diplomarbeit erwähnten elementaren Konzepte. Kapitel *3 Verwandte Arbeiten* gibt einen Literaturüberblick über existierende Voxelisierungsverfahren und Beleuchtungsverfahren, die Voxel als Grundlage nutzen. Die Beleuchtung mit virtuellen Punktlichtquellen wird anhand ausgewählter relevanter Arbeiten ebenfalls in diesem Kapitel eingeführt.

Zur Erzeugung der Voxelrepräsentation wurde im Rahmen dieser Diplomarbeit zunächst ein Voxelisierungsverfahren entwickelt, das sich für Szenen mit dynamischen Objekten eignet. Dieses Verfahren ist in Kapitel *4 Voxelisierung mit Texturatlanten* beschrieben. Das in Kapitel *5 Strahlschnitttest mit binärer*

Voxelhierarchie vorgestellte Verfahren bildet die Grundlage für die beiden innerhalb dieser Arbeit umgesetzten voxelbasierten Beleuchtungsalgorithmen:

a) Monte-Carlo-Integration der Hemisphäre eines Szenenpunktes (Kapitel *6 Indirektes Licht durch Abtastung der Hemisphäre*) und
b) Beleuchtung der Szenenpunkte mit virtuellen Punktlichtquellen (Kapitel *7 Indirektes Licht mit virtuellen Punktlichtquellen (VPLs)*).

Der Strahlschnitttest basiert auf einer hierarchischen Organisation der binär voxelisierten Szene. Das erstgenannte Beleuchtungsverfahren muss für einen gegebenen Strahl den vordersten Schnittpunkt mit der Szene finden. Das zweite Verfahren benötigt eine Sichtbarkeitsinformation zwischen zwei gegebenen Punkten. Neben einem nicht-binären Volumen, das Beleuchtungsdaten enthält, wird in Kapitel 6 zudem eine andere Speichermöglichkeit aufgezeigt („Spot Maps“ und „Cube Maps“).

Kapitel *8 Implementierungsdetails* nennt Informationen zur konkreten praktischen Umsetzung und Implementierung, die über die Erläuterungen in den jeweiligen vorherigen Kapiteln hinausgehen.

Anschließend betrachtet Kapitel *9 Ergebnisse und Bewertung* die Resultate der in dieser Arbeit umgesetzten Voxelisierung und Beleuchtungsverfahren. Es umfasst Zeitmessungen und eine Bewertung der Qualität anhand ausgewählter Ergebnisbilder, die mit den implementierten Verfahren erzeugt wurden. Einige der Bilder werden zusätzlich mit Referenzbildern eines auf Ray Tracing basierenden Frameworks verglichen. Die Arbeit endet mit einem Fazit und zeigt Weiterentwicklungsmöglichkeiten auf.

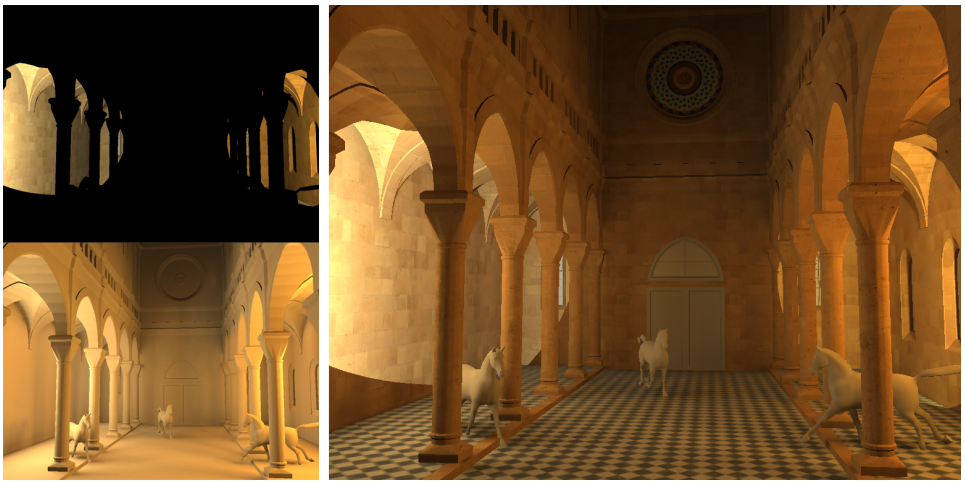

Abbildung 1: Mit dem in dieser Arbeit entstandenen Programm berechnete Beleuchtung mit zwei Interreflexionen in einer rein diffusen Szene. Links oben die direkte Beleuchtung, darunter die indirekte Beleuchtungsstärke, rechts das finale Bild.

2 Grundlagen und Begriffe

2.1 Direkte und indirekte Beleuchtung

Für den Bildentstehungsprozess wird häufig angenommen, dass sich das Licht geradlinig als Strahlen ausbreitet und sich gemäß der von der Strahlenoptik definierten Gesetze verhält. Bei dieser Betrachtungsweise wird der Wellencharakter des Lichts ignoriert. Sie unterstellt, dass die Objekte, mit denen das Licht interagiert, wesentlich größer als eine Wellenlänge sind.

Die *direkte Beleuchtung* begrenzt den Weg des Lichtstrahls auf die Stationen Lichtquelle, Oberfläche und Auge. Jeder Szenenpunkt, der von keiner Lichtquelle aus auf direktem Weg erreicht werden kann, ist komplett unbeleuchtet, also schwarz. Wird das Licht über mehrere Oberflächen hinweg reflektiert, handelt es sich um *indirekte* Beleuchtung: Beleuchtete Teile der Szene beleuchten wiederum andere Teile der Szene. Der dabei auftretende Effekt des „Abfärbens“ von Objekten auf andere nahe Objekte wird im Englischen auch *„Color Bleeding“* genannt. Theoretisch sind unendlich viele Interreflexionen möglich. In der Praxis wird der Lichtweg jedoch auf eine gewisse Länge begrenzt.

Generell wird unter einer globalen Beleuchtung die Abdeckung sämtlicher Phänomene verstanden, die durch die Lichtausbreitung auf allen möglichen Lichtwegen in der Szene entstehen können[1].

Die grundlegende Gleichung zur globalen Lichtsimulation ist die *Rendering Equation*, die von [Kajiya 1986] vorgestellt wurde. Sie berechnet, wie viel Licht ein Oberflächenelement in eine bestimmte Richtung abgibt. In der Computergrafik interessiert letztlich, wie viel Licht die betrachteten Oberflächen der Szene ins Auge bzw. in die Kamera abstrahlen. Gleichung (1) sagt aus, dass sich dieses Licht aus dem von dem Flächenelement in die Ausfallsrichtung *emittierten* und *reflektierten* Licht zusammensetzt. Der reflektierte Anteil des aus allen Einfallsrichtungen empfangenen Lichts hängt von den Materialeigenschaften ab. Es handelt sich um ein rekursives Problem, denn um für ein Oberflächenelement eine Aussage über seine Beleuchtungssituation machen zu können, muss die aller anderen bereits bekannt sein.

In der Gleichung von Kajiya läuft das Integral über alle Oberflächen der Szene. Ein umgeformter Ausdruck ist Gleichung (2), die über alle Einfallsrichtungen des vorderen Halbraums des betrachteten Flächenelements integriert.

[1] vgl. Lichtpfadnotation: L(D|S)*E = Lichtquelle(diffuse Oberfläche | spiegelnde Oberfläche)* Auge, Regeln gemäß regulärer Ausdrücke

Sie entspricht der Form, die zeitgleich zu Kajiyas Artikel von [Immel et al. 1986] veröffentlicht wurde.

$$L_o(\mathrm{d}A_e, \mathrm{d}\vec{\omega}_o) = L_e(\mathrm{d}A_e, \mathrm{d}\vec{\omega}_o) + L_r(\mathrm{d}A_e, \mathrm{d}\vec{\omega}_o) \tag{1}$$

$$L_o(\mathrm{d}A_e, \mathrm{d}\vec{\omega}_o) = L_e(\mathrm{d}A_e, \mathrm{d}\vec{\omega}_o) + \int_\Omega f_r(\mathrm{d}A_e, \mathrm{d}\vec{\omega}_i, \mathrm{d}\vec{\omega}_o) \cdot L_i(\mathrm{d}A_e, \mathrm{d}\vec{\omega}_i) \cdot \cos\theta_i \ \mathrm{d}\vec{\omega}_i \tag{2}$$

mit

$\mathrm{d}A_e$	infinitesimal kleines Flächenelement
$\mathrm{d}\vec{\omega}_o$	Ausfallsrichtung
$\mathrm{d}\vec{\omega}_i$	Einfallsrichtung
θ_i	Winkel zwischen Einfallsrichtung und Normale des Flächenelements
Ω	vorderer Halbraum des Flächenelements (Raumwinkel: 2π)
L_o	vom betrachteten Flächenelement in Ausfallsrichtung ausgehende Leuchtdichte
L_e	vom betrachteten Flächenelement in Ausfallsrichtung emittierte Leuchtdichte
L_r	vom betrachteten Flächenelement in Ausfallsrichtung reflektierte Leuchtdichte
L_i	aus Einfallsrichtung auf das betrachtete Flächenelement einfallende Leuchtdichte
f_r	Bidirektionale Reflexionsverteilungsfunktion[2]

Photometrische und thematisch verwandte Größen wie Leuchtdichte und Raumwinkel werden als bekannt vorausgesetzt. Der Wert des Integrals entspricht dem Licht, das das betrachtete Flächenelement empfängt und in die Ausfallsrichtung reflektiert. Die BRDF beschreibt das Reflexionsverhalten des Materials des Flächenelements. Sie kann neben Einfalls- und Ausfallsrichtung auch von anderen Größen wie der Wellenlänge des Lichts abhängig sein. Das empfangene Licht hängt von der geometrischen Konstellation des Flächenelements zu allen anderen Oberflächen und dem von diesen in Richtung Flächenelement abgestrahlten Licht ab.

Für die direkte Beleuchtung und eine Menge punktförmiger Lichtquellen wird das Integral zu einer Summe über alle Lichtquellen. In der Realität

[2] engl. BRDF = bidirectional reflectance distribution function

gibt es zwar keine punktförmigen, unendlich kleinen Lichtquellen, sondern nur Flächenlichtquellen. Da die Beleuchtungsbeiträge der punktförmigen Lichtquellen wesentlich schneller berechnet werden können, werden diese trotzdem häufig in der Computergrafik eingesetzt.

In dieser Arbeit werden Szenen ohne selbstleuchtende Flächen betrachtet. Es werden Punktlichtquellen und Spotlichtquellen verwendet, die harte Schatten in der Szene produzieren. Da das Hauptaugenmerk der Diplomarbeit auf der Berechnung der indirekten Beleuchtung liegt, wird die direkte Beleuchtung mit Standard-Techniken wie Shadow Mapping erzeugt. Es ist daher zweckmäßig, die Leuchtdichte jedes Oberflächenelements in einen direkten und einen indirekten Anteil aufzuteilen: $L_{direkt} + L_{indirekt}$. Beide Anteile werden getrennt berechnet. Die Kapitel *6 Indirektes Licht durch Abtastung der Hemisphäre* und *7 Indirektes Licht mit virtuellen Punktlichtquellen (VPLs)* beschreiben zwei Methoden zur Berechnung des indirekten Anteils.

Die direkte Beleuchtung wurde mit *Deferred Shading* umgesetzt. Aus Sicht der Szenenkamera werden die für die Beleuchtung nötigen Daten in mehrere Texturen gerendert, die in ihrer Gesamtheit *G-Buffer* genannt werden. Der G-Buffer in dieser Arbeit setzt sich aus Texturen mit Positionen in Weltkoordinaten, Normalen und Material zusammen. Anschließend wird mit diesen Texturen für jede Lichtquelle die Beleuchtung pro Pixel berechnet und aufaddiert. Der G-Buffer wird später ebenfalls für die indirekte Beleuchtung benötigt.

2.2 Dynamische Szene

Eine dynamische Szene erweitert eine statische Szene durch dynamische Elemente. Eine komplett statische Szene ist eine starre Konfiguration der Faktoren, die die Beleuchtung bestimmen: Geometrie, Lichtquellen, Material. Der Betrachter darf sich in der Regel frei bewegen. Ein klassisches Beispiel für eine solche Simulation ist die Architekturvisualisierung.

In dieser Arbeit darf eine Szene folgende *dynamische Bestandteile* haben:

- Starrkörper können frei verschoben und rotiert werden (z. B. durch Nutzerinteraktion oder Bewegung entlang von Animationspfaden).
- Animierte Objekte mit leichter Deformation sind möglich.
- Die Lichtquellen dürfen manipuliert werden (Position sowie beim Spotlicht der Lichtkegel).
- Das Material darf geändert werden.

Nicht unterstützt werden beliebig stark deformierbare Objekte, was durch den Einsatz des in Kapitel *4 Voxelisierung mit Texturatlanten* beschriebenen Verfahrens bedingt ist.

2.3 Voxelrepräsentation

Ein *Voxel* – analog zum Kunstwort *Pixel* (Picture Element) – ist ein diskretes Volumenelement (zum Beispiel ein Quader oder Würfel). Pixel bilden eine zweidimensionale diskrete Struktur, die beispielsweise durch die Abtastung eines kontinuierlichen Bildsignals erzeugt werden kann. Entsprechend bilden Voxel eine dreidimensionale (räumliche) diskrete Gitterstruktur, die ebenso wie Pixel verschiedenste Daten enthalten kann. Manche Autoren differenzieren genauer zwischen *Voxel* (diskreter Datenpunkt im Raum) und *Voxelzelle* (das den Punkt umgebende Volumen) [Bresink 1999, S. 86]. In dieser Arbeit werden die Zellen als Voxel bezeichnet.

Der Vorgang, die Voxel aus der gegebenen Geometrie der virtuellen Objekte zu erzeugen, heißt *Voxelisierung*. Auch hierbei handelt es sich um einen Abtastungsprozess, der für gewisse Punkte oder Bereiche im Raum bestimmt, welche Information das diesen Bereich repräsentierende Voxel tragen soll.

Abgesehen von der Medizin, in der Volumendatensätze von Patienten mittlerweile zum Standard gehören, nutzen auch andere Anwendungsgebiete Volumendaten. In der Computergrafik sind dies zum Beispiel die Kollisionserkennung in Fluidsimulationen, bestimmte Operationen in CAD-Systemen[3] oder Strahlverfolgung und Sichtbarkeitsbestimmung wie in dieser Arbeit.

Voxeldaten lassen sich nach verschiedenen Kriterien klassifizieren:

- *Binär* vs. *nicht-binär* („multi-valued“):
  Wie viele Daten enthält ein Voxel? Falls der Speicherplatz begrenzt ist, entsteht hierbei ein Trade-off zwischen der Gitterauflösung und Menge der Informationen pro Voxel.
- *Oberflächen* („boundary“) vs. *Körper* („solid“):
  Was wird voxelisiert? Bei der Oberflächenvoxelisierung sind im Idealfall alle Voxel, die von Geometrie geschnitten werden, nicht leer. Bei der Körpervoxelisierung (auch „volle Voxelisierung“) wird zusätzlich das Innere des Körpers erfasst, sodass klar definiert sein muss, was Innen und was Außen ist.
- Art des *Gitters*: Welche Gestalt haben die Zellen des Voxelgitters?
  [Bresink 1999, S. 76] nennt unter anderem folgende Kategorien: *kartesisch* (Würfel), *regelmäßig* (achsenparallele Quader), *strukturiert* (keine quaderförmigen Volumenstücke, z. B. Prismen), *unstrukturiert* (beliebige endliche Form des Volumenstücks).

In dieser Arbeit kommen sowohl binäre als auch nicht-binäre Voxel zum Einsatz, überwiegend Oberflächenvoxelisierungen und ausschließlich regelmäßige Gitter (samt dem Spezialfall der würfelförmigen Voxel).

[3] Computer-Aided Design, computergestütze Konstruktion von Gegenständen oder Bauwerken

[Cohen-Or und Kaufman 1997] befassen sich mit formalen Grundlagen diskreter dreidimensionaler Linien in regelmäßigen Voxelräumen. Ein Voxel hat (maximal) 26 Nachbarvoxel: 6 grenzen an eine Fläche, 8 an eine Ecke und 12 an eine Kante. Somit kann definiert werden, wie zwei Voxel benachbart sind: 6-benachbart bedeutet, dass sie eine gemeinsame Fläche haben, 18-benachbart, dass sie eine gemeinsame Fläche oder Kante haben, und 26-benachbart, dass sie eine gemeinsame Fläche, Kante oder Ecke haben. Dementsprechend können diskrete 3D-Strahlen (Voxelstrahlen) gemäß der Nachbarschaftseigenschaften ihrer Voxel klassifiziert werden, zum Beispiel als „26-Strahl“. Wichtig ist die Kenntnis, dass gewisse Konstellationen von Voxelstrahlen und voxelisierten Objekten bei der Strahlverfolgung zu verpassten Oberflächen führen können (siehe Abbildung 2). Cohen-Or und Kaufman nennen Kriterien für diskrete Strahlen und Oberflächen sowie Ansätze, um dieses Problem zu lösen. Für diese Diplomarbeit ist eine derartige Präzision nicht erforderlich.

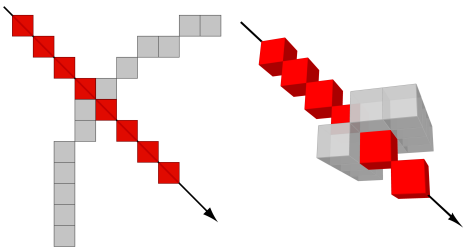

Abbildung 2: Visualisierung sogenannter „Tunnel“ (links in 2D als Pixel, rechts in 3D als Voxel) – Strahlen, die eine (Voxel-)Oberfläche verpassen. Abbildung nach [Cohen-Or und Kaufman 1997, S. 81].

3 Verwandte Arbeiten

3.1 Voxelisierungsalgorithmen

3.1.1 Slicing-Verfahren

Der Begriff „Slicing“[4] verrät bereits die Grundidee dieser Verfahrensart: Die zu voxelisierenden Objekte werden in Scheiben geschnitten.

Es handelt sich um eines der einfachsten und zugleich rechenaufwändigsten Verfahren: Für jede Schicht muss das Modell erneut gerendert werden, wobei alles außerhalb der Schicht weggeschnitten wird. Erwartungsgemäß nimmt die Effizienz bei höheren Volumenauflösungen in Voxelisierungsrichtung (Projektionsrichtung) stark ab.

Die „Urversion“ des Algorithmus beschreiben [Fang und Chen 2000]. Das orthografische Frustum der Voxelisierungskamera bestimmt die zu voxelisierende Region. In jedem Pass werden die Near- und Far-Plane auf die vordere und hintere Grenzfläche der aktuellen Schicht gesetzt. Es wird nur die Geometrie innerhalb dieser Schichtgrenzen eingetragen, da alles außerhalb liegende weggeclippt wird. Eine volle Voxelisierung wird mit einer XOR-Operation realisiert. Eine wichtige Bedingung ist, dass das Mesh des Objektes geschlossen ist, also dass Innen und Außen definiert sind. Die Grundannahme ist, dass Flächen immer paarweise als „vorne“ und „hinten“ auftreten. Die XOR-Operation beim Eintragen einer rasterisierten Fläche in den Framebuffer bewirkt, dass zwischen „1“ (Objektinneres betreten, Innen) und „0“ (Objektinneres verlassen, Außen) umgeschaltet wird. Das Ergebnis der vorherigen Schicht ist immer die Eingabe für die nächste Schicht. Solange also in einem Pixel eine „1“ steht, werden hier volle Voxel in die Schichten eingetragen – bis eine Fläche an diese Stelle rasterisiert und somit das Innere des Objektes verlassen wird.

In GPU Gems 3 stellen [Crane et al. 2007] eines der neusten Slicing-Verfahren vor (*Real-Time Simulation and Rendering of 3D Fluids*). Eines der Ziele ist eine nicht-binäre Oberflächenvoxelisierung (beispielsweise Geschwindigkeitsvektoren im Anwendungsgebiet der Fluidsimulation). Ein Geometry-Shader schneidet alle Dreiecke mit der aktuellen Ebene und gibt das Ergebnis in Form von schmalen Quads samt der einzutragenden Daten an den Fragment-Shader weiter. Bei der Rasterisierung geschieht die lineare Interpolation dieser Attributwerte. Der Schnitttest ist der geschwin-

[4] engl. *to slice* = (in Scheiben) schneiden

digkeitskritische Teil des Verfahrens, weshalb die Autoren darauf hinweisen, möglichst Modelle mit niedriger Polygonzahl zu verwenden.

3.1.2 Voxelisierung mittels rasterisierter Geometrie

Diese Klasse von Verfahren macht sich die Rasterisierungsfunktionalität der Grafikhardware zu nutze. Der Rasterisierer in der Grafikpipeline konvertiert die geometrischen Primitive (Dreiecke bzw. Polygone, Linien, Punkte) in Pixel[5], aus denen sich der Framebuffer zusammensetzt.

Die Übertragung auf den dreidimensionalen Fall ist wie folgt möglich: Die zweidimensionale Diskretisierung übernimmt der Rasterisierer, die Diskretisierung der Tiefe muss gesondert behandelt werden. Die Verwendung von Framebuffer-Objekten ermöglicht es, anstatt in den System-Framebuffer zu rendern, direkt in Texturen mit diversen Formaten zu schreiben. Binäre Voxel benötigen nur ein Bit zur Speicherung ihrer Information: „voll“ (1) oder „leer“ (0). Letztendlich bestehen Texturen auch nur aus Bits. Somit kann jedes Bit einer Textur als Voxel interpretiert werden, indem jedes gesetzte Voxel mit einer gewissen Tiefe auf eine Farbe abgebildet wird. Hierfür bieten sich insbesondere die Integer-Texturformate an, die seit OpenGL 2.0 verfügbar sind und mit Shader Model 4.0 auch für die Shader-Programmierung unterstützt werden. Die maximale Bit-Zahl pro Texel beträgt derzeit 128 (4 Kanäle à 32 Bit). Die Verwendung einer einzelnen zweidimensionalen Textur beschränkt die Auflösung des Voxelgitters in einer Richtung auf 128 Voxel. Mit mehreren Texturen (Multiple Render Targets) oder einer größeren Textur kann diese Auflösung jedoch erhöht werden.

Der in *„Fast Scene Voxelization and Applications“* [Eisemann und Décoret 2006] beschriebene Algorithmus liefert eine binäre Oberflächenvoxelisierung in einem Render-Pass mit Hilfe logischer Operationen und einer Bitmaske. Das Verfahren kann für alle rasterisierbaren Eingabedaten ausgeführt werden. Allerdings erfasst es nur eine Untermenge der Zellen, die von einer Oberfläche geschnitten werden. Besonders problematisch sind Primitive, die parallel zur Projektionsrichtung (Voxelisierungsrichtung) liegen, wie Abbildung 3 zeigt. [Dong et al. 2004] präsentieren im Artikel *„Real-Time Voxelization for Complex Models“* eine Lösung für dieses Problem: Es werden drei Voxeltexturen erstellt, für drei orthogonal aufeinander stehende Voxelisierungsrichtungen (vgl. Abbildung 3). Jedes Dreieck wird abhängig von seiner Normalen mit einer der drei Projektionsrichtungen rasterisiert: Entweder wird alles dreimal gerendert und in jedem Pass die Normale im Shader mit der Rasterisierungsrichtung abgeglichen, oder die Dreiecke werden in einem Vorverarbeitungsschritt gemäß ihrer Normalen (Flächennormalen) in drei

[5] in OpenGL-Terminologie: „Fragmente“

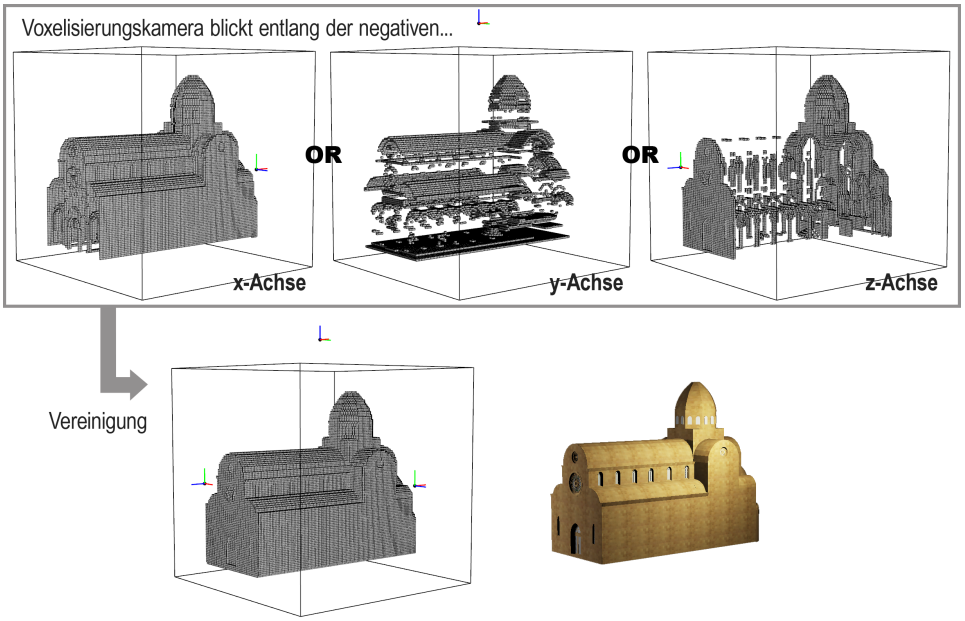

Abbildung 3: Die zu Beginn der Diplomarbeit umgesetzte Oberflächenvoxelisierung mit dem Verfahren von [Eisemann und Décoret 2006]. Um Löcher zu schließen, wird aus drei orthogonal aufeinander stehenden Richtungen voxelisiert. Die Einzelergebnisse werden mittels Veroderung in die finale Voxeltextur eingefügt.

Gruppen sortiert. Anschließend werden die drei Texturen in einer einzigen Textur zusammengefügt.

Eine weitere Verbesserung, die auf die Qualität der Voxelisierung abzielt, stellen [Zhang et al. 2007] vor: die konservative Voxelisierung (*Conservative Voxelization*). Sie lehnt sich an die konservative Rasterisierung [Hasselgren et al. 2005] an. Die Standard-Rasterisierung der Pipeline erzeugt nicht alle Pixel, die ein Polygon schneidet. Bei der konservativen Rasterisierung sollen alle Pixel, die das Polygon ganz oder teilweise überdeckt, ausgefüllt werden. Abbildung 4 zeigt den Unterschied zwischen Standard- und konservativer Rasterisierung. Analog definieren die Autoren die konservative Voxelisierung so, dass genau dann ein volles Voxel generiert wird, wenn es das Eingabemodell schneidet. In der Praxis konnte die Definition nicht ganz eingehalten werden, teilweise werden auch Voxel erzeugt, die das Modell nicht schneidet. Die konservative Voxelisierung muss neben einer konservativen Rasterisierung, die nur den zweidimensionalen Fall abdeckt, auch die Tiefe konservativ diskretisieren. Aufgrund der Tiefenwerte werden für ein Pixel entsprechend viele Voxel in dem Bereich ausgegeben. Für alle Pixel wird daher ein Tiefenintervall berechnet. Für Randpixel ist die Berechnung kompliziert und aufwändig, da das Dreieck mit dem Pixel geschnitten werden muss, um die Tiefenwerte der Schnittpunkte zu erhalten. Der Algorithmus liefert also ein

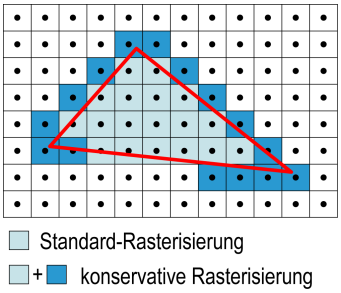

Abbildung 4: Im Unterschied zur Standard-Rasterisierung erfasst die konservative Rasterisierung alle Pixel, die von dem Dreieck überdeckt werden (im Bild dunkel- und hellblaue Pixel). Abbildung nach [Zhang et al. 2007] (Präsentation).

qualitativ hochwertiges Ergebnis. Er ist jedoch im Vergleich zu den anderen ungenaueren Verfahren langsamer, insbesondere bei Modellen mit vielen Dreiecken.

Der später veröffentlichte Artikel *„Single-Pass GPU Solid Voxelization for Real-Time Applications“* [Eisemann und Décoret 2008] überträgt den Ansatz des ersten Papers auf die Körpervoxelisierung und erläutert, wie mit der binären Voxelisierung als Ausgangspunkt die Voxel-Dichte effizient bestimmt und gespeichert werden kann. Mit Hilfe der Dichte können zudem Normalen pro Voxel bestimmt werden. Das Verfahren benötigt genau wie das erste nur wenige Millisekunden, was im Wesentlichen die reine Rendering-Zeit der Geometrie ist.

3.1.3 Depth-Peeling-Verfahren

Die dritte Klasse von Voxelisierungsverfahren führt als ersten Schritt ein Depth-Peeling der zu voxelisierenden Objekte aus. Depth-Peeling schält von vorne nach hinten entlang der Kamerablickrichtung die verschiedenen Schichten der Geometrie ab, bis sämtliche Geometrie erfasst wurde. Es ist somit von der Tiefenkomplexität des Objektes abhängig. Im Unterschied dazu sind die Slicing-Methoden von der Anzahl der Volumen-Schichten abhängig. Das Ergebnis des Depth-Peelings ist eine Reihe von Texturen mit abgeschälten rasterisierten Flächen. Die erste Textur entspricht der ersten Schicht, welche alle Pixel mit dem kleinsten Abstand zur Peeling-Kamera beinhaltet, die zweite Schicht jene Pixel mit dem zweitkleinsten Abstand, usw. Aus den Texturen wird die Voxelrepräsentation generiert.

Ebenso wie bei der Oberflächenvoxelisierung mittels Rasterisierung werden Dreiecke, die nahezu oder ganz parallel zur Blickrichtung der Depth-Peeling-Kamera liegen, nicht erfasst. [Passalis et al. 2007] stellen einen binären Körpervoxelisierungsalgorithmus mit Hilfe von Depth-Peeling vor. Sie lösen das Problem analog zu [Dong et al. 2004], indem sie aus verschiedenen Pro-

jektionsrichtungen peelen, typischerweise aus drei orthogonal aufeinander stehenden Richtungen. Ebenso verfahren [Li et al. 2005], die die Voxelisierung für eine Fluidsimulation benötigen (*Flow Simulation with Complex Boundaries*). Sie speichern das Volumen als 2D-Textur („Flat Volume“), was den Vorteil von weniger Texturwechseln hat. Interessant ist ihre Vorgehensweise zur Konvertierung der Peeling-Texturen in die Volumen-Textur, da diese Diplomarbeit in Kapitel *4 Voxelisierung mit Texturatlanten* ein Verfahren vorstellt, das ähnlich funktioniert. Sie konvertieren das Bild mit den Voxelattributen in ein Vertex-Array (ein Array von Voxelpositionen und weiteren Attributen pro Voxel). Für jeden Vertex dieses Arrays wird dann ein Punkt der Größe 1 in die finale Voxeltextur gezeichnet.

3.2 Voxelbasierte Beleuchtungsalgorithmen

3.2.1 „The Irradiance Volume“ und Nachfolger

Im Jahr 1996 stellte Gene Greger das *„Irradiance Volume“*[6] vor [Greger 1996]. Konzeptionell handelt es sich hierbei um eine dreidimensionale Lightmap. Herkömmliche Lightmaps sind zweidimensionale Texturen mit vorberechneter Beleuchtung, die zur Laufzeit auf statische Objekte gelegt werden. Vor allem in Spielen kommt diese Technik zum Einsatz. Die dreidimensionale Erweiterung auf volumetrische Texturen geschieht wie folgt: Die Volumentextur wird ebenfalls in einem Vorverarbeitungsschritt für gegebene statische Geometrie und Lichtquellen berechnet. Für viele Abtastpunkte im Raum wird zunächst eine Strahldichte-Verteilungsfunktion geschätzt. Aus dieser wird die

[6] engl. *irradiance* = Bestrahlungsstärke (radiometrische Größe, Einheit $[\frac{W}{m^2}]$, misst auf ein Flächenelement einfallenden Strahlungsfluss)

Abbildung 5: Beispiel für ein *Irradiance Volume*. Die Bestrahlungsstärke-Verteilungsfunktion in einer Gitterzelle ist als Kugel visualisiert. Abbildung entnommen aus [Greger 1996, S. 66].

Bestrahlungsstärke-Verteilungsfunktion berechnet, die zur Laufzeit für sämtliche Positionen im Raum bzw. im Volumen mittels Interpolation abgefragt werden kann (siehe Abbildung 5). Somit werden dynamische Objekte, die sich durch das Volumen bewegen, mit Hilfe der räumlichen radiometrischen Information beleuchtet. Umgekehrt haben die dynamischen Objekte aber keinen Einfluss auf ihre Umgebung, da sie nicht im Volumen erfasst sind.

Zehn Jahre später stellte [Oat 2006] mit *„Irradiance Volumes For Real Time Rendering“* eine moderne Modifikation vor. Jeder Abtastpunkt kann als Cube Map gespeichert werden, welche wiederum in Spherical-Harmonics-Koeffizienten überführbar ist. Im Falle diffuser Beleuchtung ist diese Repräsentation effizient zu speichern und abzufragen. Zudem schlägt Oat verschiedene Verfahren für den adaptiven Aufbau des Volumens als Octree vor, um Beleuchtungswechsel im Raum besser erfassen zu können. Zu beachten ist hierbei, dass das Volumen vorberechnet wird und der Fokus nicht unbedingt auf der Geschwindigkeit der Volumenerstellung liegt. Ein Beispiel für ein aktuelles PC-Spiel, in dem diese Technik angewendet wird, ist *Split/Second* von Disney Interactive Studios [Moore und Jefferies 2009].

Ebenfalls Gebrauch von Cube Maps machen [Nijasure et al. 2005]. Der Raum wird in ein regelmäßiges, sehr grobes Gitter mit einer Auflösung in der Größenordnung von $4 \times 4 \times 4$ unterteilt. In jedem Frame wird für jede Gitterzelle eine sehr niedrig aufgelöste Cube Map erzeugt und ebenfalls in eine Spherical-Harmonics-Repräsentation konvertiert. Da sie alle Schritte für jedes Frame neu durchführen, sind dynamische Objekte und Lichtquellen möglich; die Performance ist jedoch naturgemäß geringer als bei den Varianten, die das Volumen als reine Lightmap nutzen.

Die neueste Entwicklung mit Irradiance-Volume-Ursprung präsentiert Crytek mit den *„Light Propagation Volumes“* (LPV) [Kaplanyan 2009] bzw. *„Cascaded Light Propagation Volumes“* [Kaplanyan und Dachsbacher 2010] für die CryEngine 3. Die Anwendungsgebiete der LPVs sind Annäherungen der direkten Beleuchtung extrem vieler Punktlichtquellen sowie der indirekten Beleuchtung mit virtuellen Punktlichtquellen. Das Volumen hat eine grobe Auflösung von beispielsweise $32 \times 32 \times 32$ und speichert pro Voxel Spherical-Harmonics-Koeffizienten. Es wird mit der Kamera mitgeführt. Erzeugt wird es durch ein initiales Einfügen der Daten *(„injection“)* und anschließende iterative Lichtausbreitung *(„propagation“)* (vgl. Abbildung 6). Die Ausbreitungssimulation ist inspiriert von Methoden zur Fluidsimulation. Die erreichte Performance ist extrem hoch: 3-4 ms insgesamt für alle Schritte. Hohe Genauigkeit lässt sich nicht erzielen, was unter anderem durch die Größe eines Voxels im Verhältnis zur Szene bedingt ist. Die Erweiterung *„Cascaded LPVs“* nutzt drei verschachtelte Volumen mit unterschiedlichen Abmessungen (siehe Abbildung 7), um Nahbereiche besser erfassen zu können.

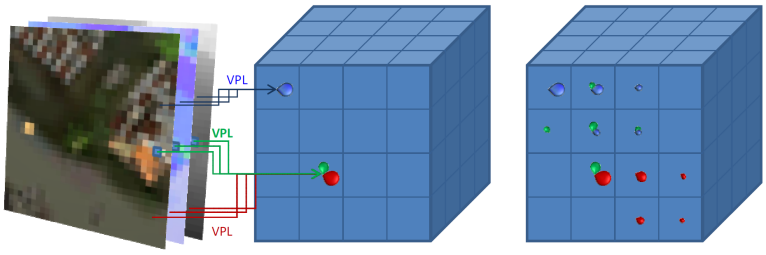

Abbildung 6: Die wichtigsten Schritte des LPV-Verfahrens von links nach rechts: Rendern der Szene aus Sicht der Lichtquellen, Einfügen („injection“) der virtuellen Punktlichtquellen (VPL, siehe auch Abschnitt *3.3 Beleuchtung mit virtuellen Punktlichtquellen*), Lichtausbreitung im Volumen („propagation“). Abbildung entnommen aus Präsentation von [Kaplanyan und Dachsbacher 2010].

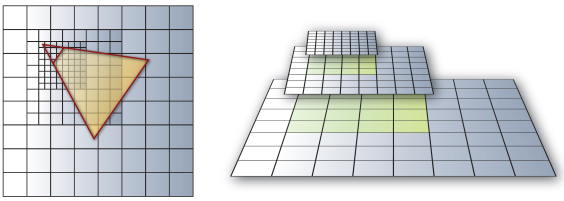

Abbildung 7: Drei unterschiedlich aufgelöste Volumen werden mit der Kamera mitgeführt. Abbildung aus [Kaplanyan und Dachsbacher 2010].

3.2.2 Radiosity im Voxelraum

Der Vollständigkeit halber sollen an dieser Stelle auch die Grundideen eines voxelbasierten Radiosity erwähnt werden, welches im Offline-Rendering anzusiedeln ist. Das originale Radiosity-Verfahren basiert auf einer möglichst geschickten Unterteilung der Szenenoberflächen, zwischen denen der Lichtaustausch stattfindet.

[Malgouyres 2002] entwickelte Radiosity im diskreten Voxelraum (*„A Discrete Radiosity Method“*). Alle Modelle werden regelmäßig voxelisiert und in einem Octree gespeichert. Zwischen den Voxeln findet der Lichtaustausch statt: Die Summe in der grundlegenden Radiosity-Gleichung wird so transformiert, dass sie nicht mehr über Patches, sondern über Voxel läuft. Genauer gesagt läuft sie über alle diskreten Richtungen, die durch die Verbindung des betrachteten Empfängervoxels mit den Voxeln einer um ihn zentrierten diskreten Kugel festgelegt sind. Für jede Richtung muss das erste volle Voxel gefunden werden. Die Autoren bezeichnen diesen Schritt als Analogon zur Formfaktor-Berechnung im klassischen Radiosity. Nachdem für das Voxel ein Radiosity-Wert bestimmt ist, wird das finale Bild mit Scankonvertierung oder Ray Tracing erzeugt. Dabei wird die Beleuchtungsinformation für jeden sichtbaren Punkt aus den Voxeln abgefragt.

3.2.3 Voxel für Ambient Occlusion und Flächenlichtquellen

Ambient Occlusion[7] simuliert den Verdeckungsgrad eines Punktes in der Szene: Wie viel Umgebungslicht erreicht diesen Punkt bzw. wie viel wird von naher Geometrie geblockt? Die in Abschnitt *3.1.2 Voxelisierung mittels rasterisierter Geometrie* vorgestellten Techniken schaffen die Voraussetzung für eine Alternative zu den bildraumbasierten Ambient-Occlusion-Verfahren: voxelbasiertes Ambient Occlusion in Szenen mit dynamischen Objekten. Binäre Voxel nähern die Existenz oder Abwesenheit von Geometrie an einer Stelle im Raum an. Wie bereits erwähnt, ist eine binäre Voxelisierung ebenso wie der Tiefenpuffer als zweidimensionale Textur speicherbar und kann in wenigen Millisekunden erzeugt werden.

„Hybrid Ambient Occlusion“ von [Reinbothe et al. 2009] nutzt zur Voxelisierung das Verfahren von Eisemann und Décoret. Mit Ray Marching verfolgen sie Strahlen in den vorderen Halbraum eines Punktes, um den Ambient-Occlusion-Wert zu schätzen. Sie rendern zwecks Performancesteigerung die Ambient-Occlusion-Textur nur auf halber Auflösung, welche mit einem geometrie-sensitiven Filter wieder hochskaliert wird (daher die Bezeichnung „hybrid“). Das Verfahren bot eine der Grundlagen für die dieser Diplomarbeit vorangegangene Studienarbeit [Thiedemann 2009]. Die Bestimmung des indirekten Lichts (Color Bleeding) beschränkte sich in der Studienarbeit auf den Bildraum und wies somit die typischen Artefakte auf.

Das in diesem Jahr erschienene Paper *„Real-Time Volume-Based Ambient Occlusion“* von [Papaioannou et al. 2010] schlägt verschiedene Voxelisierungsstrategien vor: Für Starrkörper kann eine Voxeltextur vorberechnet und zur Laufzeit mit entsprechend transformierten Zugriffen ausgelesen werden. Animationssequenzen können als eine Reihe von Volumentexturen gespeichert werden. Komplett deformierbare Körper müssen „on the fly“ voxelisiert werden. Allerdings merken die Autoren an, dass die Verwaltung der vorvoxelisierten Objekte zur Laufzeit zum Problem werden kann, insbesondere bei vielen Objekten. Insofern nutzen sie alternativ auch eine pro Frame und für die gesamte Szene komplett neu generierte Voxelisierung. Statt der kompakten 2D-Textur (1 Bit = 1 Voxel) tragen sie die binären Voxel in eine 3D-Textur ein. Somit erhalten sie beim Zugriff trilinear interpolierte Werte zwischen 0 und 1 und können über einen Schwellenwert den Ambient-Occlusion-Effekt steuern. Das Verfahren lässt sich übertragen auf halbtransparente Objekte, die als Voxel mit abgestuften Dichtewerten zwischen 0 und 1 vorliegen.

[Nichols et al. 2010] verwenden eine binäre Voxelisierung à la Eisemann und Décoret für Sichtbarkeitsanfragen zur Berechnung des Schattenwurfs einer großen Flächenlichtquelle mit dynamischem Inhalt (≈Videoleinwand). Die Sichtbarkeit wird mit uniform abgetasteten Strahlen (Ray Marching) ermit-

[7] dt. Umgebungsverdeckung

telt. Das zugrundeliegende Prinzip basiert auf einer Reihe vorangegangener Arbeiten derselben Autoren aus dem Bereich des Multiresolution Renderings, worauf in Abschnitt *7.4 Kombination mit Multiresolution-Verfahren* genauer eingegangen wird.

3.3 Beleuchtung mit virtuellen Punktlichtquellen

Dieser Abschnitt widmet sich ausgewählten Verfahren, die grundlegende Elemente und Ideen für die in Kapitel *7 Indirektes Licht mit virtuellen Punktlichtquellen (VPLs)* geschilderte Methode liefern.

Mit *„Instant Radiosity“* etablierte [Keller 1997] eine neue Klasse von Verfahren zur näherungsweisen Berechnung der indirekten Beleuchtung. Die Grundlage des Algorithmus bilden die sogenannten *virtuellen Punktlichtquellen*, die in der Szene verteilt werden. Lichtpartikel werden von den primären Lichtquellen aus verschossen und über die Oberflächen hinweg verfolgt, wobei jeweils ein gewisser Anteil absorbiert wird. An den Auftreffpunkten werden die VPLs platziert. Eine VPL auf einer diffusen Oberfläche ist eigentlich keine Punktlichtquelle, sondern ein infinitesimal kleiner diffuser Strahler – sie hat eine Ausrichtung (Normale) und strahlt nur in den vorderen Halbraum. Der Lichtstrom der VPL wird dabei mit der „Materialfarbe“ skaliert, d. h. für diffuses Material mit dem diffusen Reflexionsgrad. Für Interreflexionen nach dem ersten Bounce wird für das Material der durchschnittliche Reflexionsgrad der Szene angenommen. Für jede VPL wird ein Bild der Szene mit der Beleuchtung samt Schatten dieser VPL berechnet, sodass das Endergebnis der Durchschnitt aller Bilder ist.

Seit der Veröffentlichung wurden zahlreiche auf diesem Prinzip aufsetzende Verfahren vorgestellt.

Mit *„Reflective Shadow Maps“* (RSM) berechnen [Dachsbacher und Stamminger 2005] indirekte Beleuchtung mit einer Interreflexion. Die RSM erweitern die klassische Shadow Map um zusätzliche Informationen: Weltkoordinate, Normale und reflektierten Lichtstrom. Jedes Pixel der Textur wird als kleine Flächenlichtquelle angesehen. Die Beleuchtungsbeiträge dieser Pixellichtquellen nähern die indirekte Beleuchtung an. Aus Performancegründen ignorieren die Autoren die Verdeckung, das heißt, dass indirektes Licht z. B. durch Wände hindurchgeht. Theoretisch muss die Beleuchtung aller VPLs für alle Bildpunkte des Ergebnisbildes berechnet werden. Um den Aufwand zu verringern, berechnen sie diese nicht für jeden Bildpunkt, sondern nur dann, wenn die Beleuchtungsinformation nicht von umgebenden Pixeln interpoliert werden kann.

In dem Nachfolge-Artikel *„Splatting Indirect Illumination“* reduzieren [Dachsbacher und Stamminger 2006] die Berechnung auf noch weniger Pixel: Für jede virtuelle Punktlichtquelle wird ein „Splat“ in Form einer passend

skalierten und gestauchten Kugel ins Bild gerendert. Alle durch die Kugelgeometrie rasterisierten Pixel werden für diese VPL ausgewertet. Jede VPL hat somit nur einen beschränkten Einflussbereich. Das Rendering geschieht mittels Deferred Shading, um die Beleuchtung von der Szenenkomplexität zu entkoppeln. Die Sichtbarkeit wird weiterhin ignoriert.

Dem Thema Sichtbarkeit widmen sich unter anderem die Autoren von *„Imperfect Shadow Maps“* (ISM) [Ritschel et al. 2008]. Sie haben beobachtet, dass präzise Sichtbarkeitsergebnisse pro VPL nicht nötig sind, denn in der Summe aller Beiträge ist der Fehler kaum sichtbar. Die von ihnen verwendeten „nicht-perfekten“ Shadow Maps sind niedrig aufgelöste Shadow Maps, die aus einer Punkterepräsentation der Geometrie erzeugt werden. Dieses Vorgehen ermöglicht es, mehrere hundert Shadow Maps in wenigen Millisekunden zu rendern.

4 Voxelisierung mit Texturatlanten

4.1 Die Idee

In diesem Kapitel wird das in dieser Diplomarbeit entwickelte Verfahren zur Voxelisierung mit Texturatlanten beschrieben. Ihre Anfänge fanden Texturatlanten in Spielen und Anwendungen, die eine hohe Renderinggeschwindigkeit erfordern: Die zahlreichen Texturen der verschiedenen Objekte werden in größeren Texturen zusammengefasst, um zur Laufzeit die teuren Wechsel zwischen vielen kleinen Texturen zu vermeiden oder zumindest zu reduzieren.

Neben Spielen setzen auch andere Anwendungsbereiche Texturatlanten für andere Zwecke als zur State-Change-Minimierung ein. So benutzen beispielsweise [Ritschel et al. 2008] die Atlanten, um die Punkterepräsentation der Szene aufzubauen.

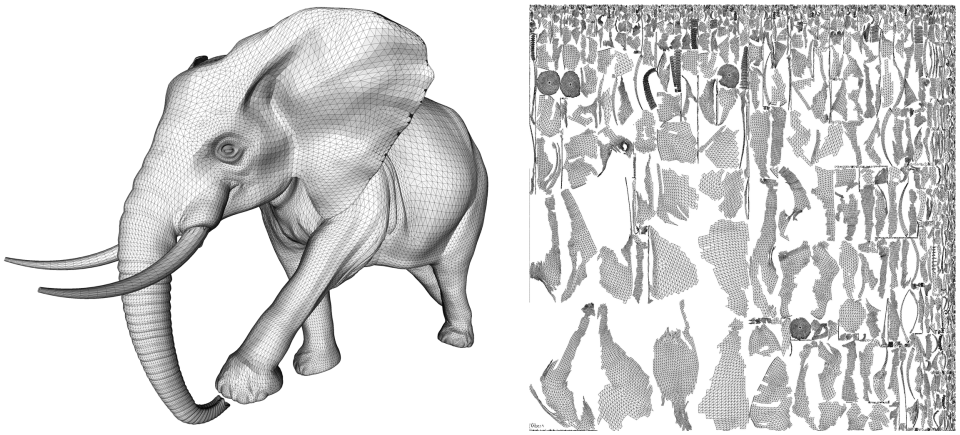

Abbildung 8: Texturatlas eines Elefanten-Modells (ca. 84.000 Dreiecke), automatisch mit Blender erstellt (Dauer: 40 Sekunden). In diesem Beispiel ist zu sehen, dass benachbarte Polygone des Modells nicht unbedingt auch im Atlas benachbart sind.

Die dahinter stehende Idee ist, dass ein Texturatlas letztlich eine andere Darstellungsform der Szenendaten ist. Die Szene liegt aufgefaltet als zweidimensionales Bild vor (Atlas eines Beispielobjekts, siehe Abbildung 8).

Die Vorstellung der aufgefalteten Szene liefert die Grundidee für das in dieser Diplomarbeit konzipierte Voxelisierungsverfahren: Die zu voxelisierende Geometrie ist in dem Texturatlas bereits erfasst. Die Daten müssen nur noch in die Voxelstruktur überführt werden. Das Ergebnis ist eine Oberflächenvoxelisierung.

4.2 Erstellung der Texturatlanten

4.2.1 Generierung der Atlaskoordinaten

Ein Texturatlas ist eine zweidimensionale Repräsentation von Modelloberflächen in Form einer Textur. Er benötigt eine Abbildung von dreidimensionalen Modellkoordinaten auf zweidimensionale Texturkoordinaten. Es gibt auch dreidimensionale Texturkoordinaten, diese spielen hier aber keine Rolle. Jedem Vertex wird eine zweidimensionale Texturkoordinate im Wertebereich $[0, 1]^2$ zugewiesen. Diese Texturkoordinaten sind kontinuierlich und werden später bei der Rasterisierung diskretisiert. Die in dieser Diplomarbeit für die Voxelisierung eingesetzten Texturatlanten erfordern eine eindeutige Abbildung. In herkömmlichen Texturatlanten darf ein Texturstück an mehreren Stellen des Modells verwendet werden. Hier jedoch muss die Bedingung gestellt werden, dass jedes Polygon einen eigenen Bereich im Texturatlas einnimmt, denn das Ziel ist die Erfassung des gesamten Modells.

Die Abbildung von Modellkoordinaten auf Texturkoordinaten wird auch als *UV-Mapping* bezeichnet. Der Prozess der Berechnung oder manuellen Erstellung heißt *UV-Unwrapping*. Für das UV-Unwrapping der in dieser Arbeit benutzten Modelle wurde das Open-Source-Modellierungsprogramm Blender[8] in der Version 2.49b benutzt. Blenders Methoden zur automatischen Berechnung eines UV-Mappings *„Lightmap UVPack“* und *„Unwrap (smart projections)“* berechnen Abbildungen, die eindeutig und daher als Eingabe für den folgenden Algorithmus geeignet sind (Beispiel für *„Unwrap (smart projections)“* in Abbildung 8). Abhängig von der Anzahl der Polygone und den gewählten Optionen zur Steuerung der Qualität dauert eine solche Berechnung zwischen wenigen Sekunden und Stunden.

In einem ersten Ansatz wurde jeweils die gesamte Szene (alle in ihr befindlichen Objekte) in einem einzigen Atlas abgebildet. Diese Vorgehensweise erwies sich jedoch sehr schnell als unflexibel und langwierig, da bei jeder Änderung der vorhandenen Objekte der Atlas neu berechnet werden musste. Somit wurde dieser Ansatz verworfen und stattdessen für jedes einzelne Objekt ein eigener Atlas erstellt. Dies spielt für den nachfolgend vorgestellten Algorithmus keine Rolle. Außerdem könnten bei Bedarf auch die Einzel-Atlanten wieder in einen größeren Gesamt-Atlas transformiert werden.

Animationssequenzen werden in dieser Arbeit durch das Anzeigen einer Folge von Wavefront-OBJ-Modelldateien[9] realisiert. Für den Fall, dass die Vertex-Reihenfolge in den Dateien identisch ist, reicht ein einziger Atlas aus, der exemplarisch für ein Modell der Sequenz erzeugt und für alle weiteren

[8] http://www.blender.org/

[9] Spezifikation des Formats unter `http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/` (letzter Zugriff am 22.07.2010)

Modelle genutzt wird. Wichtig für die Wiederverwendung ist allerdings, dass die Animation das Modell nicht zu stark deformiert.

4.2.2 Atlas-Rendering

Als Eingabedaten liegen die samt ihrer Texturatlas-Koordinaten geladenen Modelle und die zugehörigen leeren Atlas-Texturen vor. Die Auflösung der Texturen spielt eine essenzielle Rolle für die erreichte Qualität der Voxelisierung (vgl. Abbildung 9). Gleichzeitig sollte die Auflösung nicht unnötig hoch gewählt sein, um Geschwindigkeitseinbußen zu vermeiden. Es kann keine allgemeine Auflösungsempfehlung gegeben werden, da diese von diversen Faktoren abhängt: von der Voxelisierungsauflösung, der Größe des Objektes im Verhältnis zu den Voxeln und der Qualität der Abbildung (Stärke der Verzerrung, Beachtung der Größenverhältnisse der Polygone). Eine passende Auflösung muss daher manuell austariert werden.

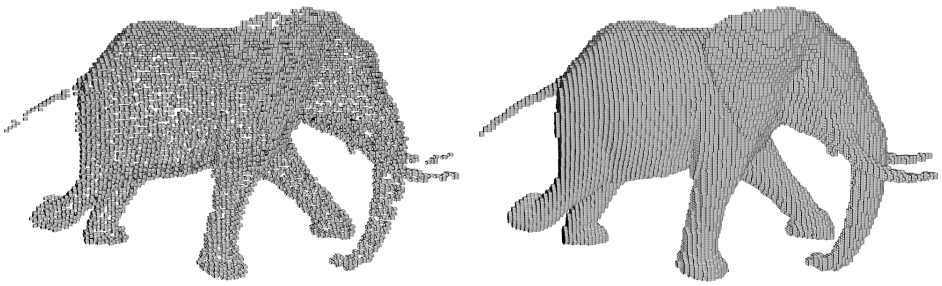

Abbildung 9: Auswirkung der Atlasauflösung auf das Voxelisierungsergebnis (Voxelauflösung hier 128^3). Links: Voxelisierung mit unzureichender Atlasauflösung von 128×128, es entstehen Löcher in der Voxeloberfläche. Rechts: Voxelisierung mit ausreichender Atlasauflösung: 384×384.

Für jedes Objekt wird eine Reihe an Atlanten erzeugt (Beispiele siehe Abbildung 10). Die Atlastexturen sind als Render-Targets gebunden. Der Vertex-Shader reicht nicht wie üblich die Modellkoordinaten als zu rasterisierende Position weiter, sondern die Atlastexturkoordinate. Für die binäre Voxelisierung genügt ein Positionsatlas mit 3D-Weltkoordinaten. Für die nicht-binäre Voxelisierung müssen zusätzlich alle anderen gewünschten Daten in weitere Texturen gerendert werden.

Für eine dynamische Voxelisierung müssen die Atlanten in jedem Frame neu gerendert werden. Daten, die für statische Objekte auf jeden Fall pro Frame gleich bleiben, brauchen nur einmal gerendert zu werden (z. B. Weltkoordinaten, Normalen, Material). Falls die direkte Beleuchtung ebenfalls in einem Atlas gespeichert werden soll, wird diese mit dem im implementierten System verfügbaren Deferred Shading berechnet, allerdings mit geringerer Schattenqualität. Darüber hinaus kann die Geschwindigkeit des

Abbildung 10: Atlastexturen des Elefanten-Modells. Von links nach rechts: Leuchtdichte (nur möglich, wenn ausschließlich diffuse Materialien), Normalen, Weltkoordinaten.

Atlas-Renderings – insbesondere für nicht-animierte Modelle mit sehr vielen Dreiecken – mit folgender Methode gesteigert werden: Bei Programmstart werden die Geometrie-Atlanten in Modellkoordinaten vorgerendert und dann die Texturinhalte in jedem Frame mit der Transformationsmatrix des Modells in Weltkoordinaten transformiert.

4.3 Binäre und nicht-binäre Voxelisierung

4.3.1 Ablauf der binären Voxelisierung

Die grundlegenden Schritte der binären Atlas-Voxelisierung sind:

1. Atlas-Textur mit 3D-Weltkoordinaten rendern.
2. Liste aller gültigen Atlas-Pixelkoordinaten erstellen (einmalig).
3. Für jeden gültigen Atlaspixel einen Vertex rendern, in Voxelkoordinaten transformieren und in die Voxeltextur eintragen.

Die Eingabe für den binären Voxelisierungsalgorithmus sind somit ein Positionsatlas mit 3D-Weltkoordinaten und eine Reihe zweidimensionaler Vertices, deren Koordinaten Pixelkoordinaten der Atlastextur sind. In OpenGL sind diese vom Typ `GL_POINTS`. Die Voxeltextur (das Render-Target) ist eine zweidimensionale vier-kanälige Unsigned-Integer-Textur mit 32 Bit pro Kanal. Abbildung 11 zeigt das binär und nicht-binär voxelisierte Elefanten-Modell.

Die Vertices können initial erstellt werden, nachdem ein erster Atlas gerendert wurde, in dem klar definiert ist, welche Pixel gültige Informationen enthalten. Nur für die Pixelkoordinaten, die auf Pixel mit gültigem Inhalt zeigen, werden entsprechende Vertices generiert. „Gültig“ bedeutet, dass diese Pixel beim Atlas-Rendering rasterisiert wurden. Alternativ wäre es möglich, einfach sämtliche Vertices (Anzahl = Pixelanzahl) zu erzeugen und später im Shader zu verwerfen, falls diese auf ungültige Daten in der Textur zeigen.

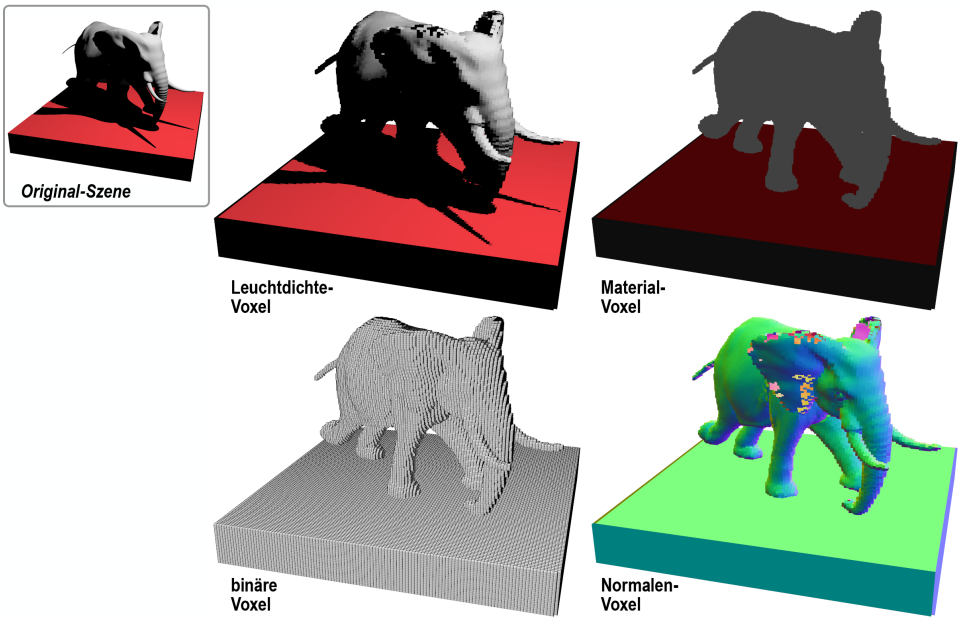

Abbildung 11: Ergebnis-Voxeltexturen der binären und nicht-binären Atlas-Voxelisierung. Die Voxelisierungsauflösung beträgt 128^3.

Da die Texturatlanten jedoch statisch sind und sich somit die Verteilung der gültigen Pixel nicht ändert, bietet sich das zuerst geschilderte Verfahren an.

Alle Vertices werden mit auf „1“ gesetzter Punktgröße in die Voxeltextur gerendert; jeder Vertex erzeugt ein volles Voxel. Der Vertex-Shader liest an der Vertexkoordinate die 3D-Weltkoordinate aus dem Positionsatlas und transformiert sie in Voxelkoordinanten. Die Transformation setzt sich aus der View-Transformation und der orthografischen Projektion der Voxelisierungskamera zusammen. Die so erhaltenen (x,y)-Koordinaten sind für die Rasterisierung relevant. Die z-Koordinate ist für das Setzen eines Bits in entsprechender Tiefe in der Voxeltextur wichtig. Der z-Wert wird in den Bereich zwischen 0 und 1 transformiert und an den Fragment-Shader weitergereicht. Mit Hilfe einer eindimensionalen Textur, die Bitmasken enthält (siehe Abbildung 12), wird die Bitfolge der Länge 128 mit dem zu setzenden Bit ausgelesen und ausgegeben. Eine logische OR-Operation ermöglicht das sukzessive Einfügen in die Voxeltextur.

4.3.2 Ablauf der nicht-binären Voxelisierung

Die nicht-binäre Voxelisierung verläuft analog zur binären mit folgenden Modifikationen:

Das Render-Target ist eine 3D-Textur, deren Format und Datentyp sich nach den pro Voxel zu speichernden Daten richtet. Ein Pixel der 3D-Textur

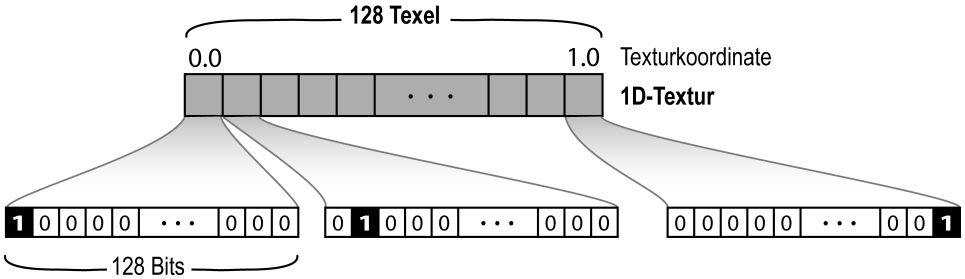

Abbildung 12: Bitmasken als Texel einer 1D-Textur: Pro Texel ist jeweils 1 Bit gesetzt. Die Stelle des gesetzten Bits hängt von der Texelposition in der Textur ab. Jeder transformierte z-Wert eines zu voxelisierenden Punktes entspricht einer Texturkoordinate der 1D-Textur, sodass die passende Bitmaske ausgelesen werden kann. Abbildung aus [Thiedemann 2009].

entspricht einem Voxel. Soll die binäre Voxelisierung mit der nicht-binären zur Deckung gebracht werden, müssen die Abmessungen entsprechend gewählt sein (Höhe und Breite analog zur Integer-Textur, 128 Ebenen).

Während die binäre Voxelisierung mit Vertex- und Fragment-Shader auskommt, benötigt die nicht-binäre zusätzlich einen Geometry-Shader. Der Geometry-Shader erlaubt es, pro Vertex die Ebene der 3D-Textur auszuwählen, in die gerendert werden soll. Die zur binären Voxelisierung deckungsgleiche Ebene wird mit einer eindimensionalen Lookup-Textur ermittelt. Die Lookup-Textur ist analog zur Bitmasken-Textur aufgebaut, beinhaltet aber Zahlen zwischen 0 und 127 als Ebenenindex für die 128 Ebenen. Zusätzlich zum Positionsatlas werden die Atlanten mit den Daten ausgelesen und an den Fragment-Shader weitergereicht.

Während der Voxelisierung ist keine logische Operation aktiv. Jeder rasterisierte Vertex überschreibt den Inhalt des Pixels, auf den er abgebildet wurde. Es kann durchaus passieren, dass mehrere Pixel der Atlastextur bzw. die in ihr eingetragenen 3D-Weltkoordinaten in ein Voxel fallen. In diesem Fall steht in dem Voxel am Ende der Wert des mit dem zuletzt gerenderten Vertex assoziierten Pixels. Sichtbar wird dies bei sehr dünnen Objekten wie dem Ohr des Elefanten (vgl. Abbildung 10 Normalenvoxel). Die Dicke des Ohrs unterschreitet an den Stellen, an denen die Normalen der hinteren Seite „durchscheinen“, eine Voxelbreite.

Sollen mehr Daten pro Voxel gespeichert werden, als es ein einzelnes Pixel der 3D-Textur zulässt, gibt es zwei Möglichkeiten: Eine Lösung ist es, mehrere 3D-Texturen als Render-Targets zu binden, in die gleichzeitig gerendert wird. Eine Alternative mit reduziertem Speicherplatz-Bedarf speichert statt der Daten nur die Atlastexturkoordinaten in einer einzigen 3D-Textur. Diese 3D-Textur fungiert als „Indirektionsvolumen“, dessen Voxel auf die entsprechenden Stellen in den Atlanten verweisen. Die Voraussetzung für diese

Variante ist, dass ein Gesamt-Atlas vorliegt. Falls mehrere Atlanten (zum Beispiel einer pro Objekt) benutzt werden sollen, müsste zusätzlich zur Texturkoordinate eine den Atlas identifizierende Nummer im Voxel gespeichert werden.

4.3.3 Vorvoxelisierung für statische Objekte

Oftmals bestehen große Teile der Szene aus statischer Geometrie. Um die Performance zu steigern, können alle statischen Teile vorvoxelisiert werden. Zur Laufzeit werden jeweils nur noch die dynamischen Objekte eingefügt. Dafür muss in jedem Frame zunächst eine Kopie der originalen Voxeltextur mit den statischen Voxeln angelegt werden. In die Kopie werden die dynamischen Voxel eingefügt. Die Original-Voxeltextur ist daher die ganze Zeit über im Speicher zu halten. Die binäre zweidimensionale Integer-Voxeltextur kann in einem Pass kopiert werden. Für die nicht-binären 3D-Voxeltexturen hängt die Anzahl der Kopierschritte von der Anzahl der Ebenen ab. Der Inhalt einer 3D-Textur kann nicht „auf einen Schlag“, sondern nur ebenenweise in eine zweite Textur kopiert werden. Durch die Verwendung mehrerer Render-Targets[10] kann die Anzahl der Durchläufe aber reduziert werden. Es ist daher jeweils abzuwägen, ob sich der Overhead des Kopierens im Vergleich zu einer kompletten Neuvoxelisierung lohnt. Zudem verdoppelt sich der Speicherplatzbedarf.

Erfüllt das dynamische Objekt die Bedingungen für eine Körpervoxelisierung, kann für das Einfügen statt der Atlas-Voxelisierung wahlweise auch das Verfahren von [Eisemann und Décoret 2008] zur binären Voxelisierung benutzt werden. Allerdings ist dann nicht mehr die Deckungsgleichheit mit einer parallel verwendeten nicht-binären Atlas-Voxelisierung gegeben.

4.4 Vor- und Nachteile gegenüber anderen Verfahren

Der Texturatlas als Repräsentation für die zu voxelisierende Geometrie beschränkt die Menge der verwendbaren Objekte auf Starrkörper und sich nicht allzu stark deformierende Modelle. Sinnvollerweise werden kleine Dreiecke des ursprünglichen nicht-deformierten Modells auf kleine Flächen in dem Atlas abgebildet. Würde die Fläche eines Dreiecks in Folge einer Deformation sehr groß werden, entstehen Löcher in der Voxelisierung, da die Abbildung statisch ist und sich nicht entsprechend verformt. Alle Transformationen, die den Flächeninhalt eines Polygons nicht oder nur wenig verändern, sind dagegen problemlos möglich.

Jedes Modell benötigt einen Texturatlas. Wie in Abschnitt *4.2.1 Generierung der Atlaskoordinaten* geschildert, kann die Erzeugung langwierig

[10] maximal 8 Render-Targets sind auf heutigen Grafikkarten möglich

sein. Aber immerhin ist dieser Schritt einmalig, die Atlaskoordinaten können danach für das Modell wiederverwendet werden. Für die Qualität der Voxelisierung ist die Qualität des Atlas ausschlaggebend. Der wichtigste Faktor für die Atlas-Qualität ist, wie sehr die Flächenverhältnisse eingehalten werden.

Ebenso wie beim Depth-Peeling liegen die Ausgangsdaten für die Voxelisierung als Texturen vor. Der Texturatlas ermöglicht es jedoch, das ganze Modell in eine einzige Textur in nur einem Render-Pass zu schreiben. Dagegen ist Depth-Peeling von der Tiefenkomplexität des Objektes abhängig, braucht somit mehr Render-Passes und benötigt ebenfalls mehr Texturen zur Speicherung. Darüber hinaus ist das Depth-Peeling zwecks Voxelisierung auch aus folgendem Grund aufwändig: Um die Flächen zu erfassen, die parallel zur Peeling-Kamera-Blickrichtung liegen, muss das Peeling aus drei Richtungen wiederholt werden. Das Atlas-Rendering ist folglich viel effizienter.

Im Vergleich zu den Voxelisierungsverfahren, die die Geometrie direkt in die Voxeltextur rasterisieren, ist die Atlas-Voxelisierung in verschiedener Hinsicht vorteilhaft. Sie ist unabhängig davon, ob Flächen parallel zur Voxelisierungsrichtung sind. Löcher können nicht durch die Ausrichtung der Flächen, sondern nur durch einen zu gering aufgelösten oder schlechten Atlas entstehen. Die Verfahren zur Oberflächenvoxelisierung mittels rasterisierter Geometrie schließen die Lücken, die aufgrund unzureichend rasterisierter Flächen entstanden, durch die Rasterisierung aus zwei weiteren orthogonalen Richtungen. Abgesehen davon, dass hierfür zwei weitere Render-Passes erforderlich sind, ist das wesentliche Problem die Vereinigung der drei entstandenen Voxeltexturen. Zum einen ist das Zusammenfügen performancekritisch, da für jedes Voxel erkannt werden muss, ob er in einer der zusätzlichen Repräsentationen an der entsprechend transformierten Position gesetzt ist. Zum anderen ist das Zusammenfügen nur dann umsetzbar, wenn die drei Voxelbereiche würfelförmig sind und die Auflösung in allen drei Richtungen gleich ist. Ansonsten könnten die Texturen nicht zur Deckung gebracht werden. Hieraus ergibt sich eine wesentliche Limitierung: Der zu voxelisierende Bereich ist auf einen Würfel beschränkt. Hat die Bounding-Box des Modells keine Würfelform, sondern die eines schmalen Quaders, wird unnötigerweise viel leerer Raum voxelisiert und gespeichert. Die Atlas-Voxelisierung hingegen erlaubt beliebige Voxelfrusta mit beliebigen Auflösungen der Voxeltextur in x- und y-Richtung. Für jede Bounding-Box kann ein eng anliegendes Voxelfrustum gewählt werden.

Die Eingabemodelle für Oberflächenvoxelisierungen unterliegen weniger Beschränkungen gegenüber Körpervoxelisierungen. Für eine Körpervoxelisierung muss das Modell „watertight“[11] sein, es darf keine Löcher haben. Für jeden Punkt im Raum muss genau definiert sein, ob sich dieser Punkt

[11] dt. wasserdicht

innerhalb oder außerhalb des Modells befindet. Andernfalls liefert die Körpervoxelisierung falsche Ergebnisse. Leider sind nur die wenigsten Modelle „watertight“. Für komplexe Szenen ist die Körpervoxelisierung daher nicht praxistauglich.

Um das Atlas-Rendering von der Dreieckszahl zu entkoppeln, bietet sich an, eine unterschiedliche Aktualisierungsmethode abhängig von der Art der Dynamik des Modells zu verwenden: Die Atlanten von Starrkörpern können texturbasiert aktualisiert werden (vgl. Abschnitt *4.2.2 Atlas-Rendering*, letzter Absatz); nur für animierte bzw. sich deformierende Objekte muss die Geometrie in jedem Frame erneut in den Atlas gerendert werden.

Die Atlas-Voxelisierung an sich hängt nicht von der Dreieckszahl des Modells ab, sondern von der Anzahl der in die Voxeltextur gerenderten Vertices. Die Zeitmessungen aus Abschnitt *9.2 Performance der Atlas-Voxelisierung* belegen dies. Je nach Komplexität des Modells (vgl. Erläuterungen in Abschnitt 9.2) beträgt die Gesamtdauer der Atlas-Voxelisierung wenige Millisekunden und ist daher für interaktive oder Echtzeit-Anwendungen geeignet.

4.5 Einsatz der Atlas-Voxelisierung in dieser Arbeit

Initial wird die Bounding-Box der gesamten Szene bestimmt – diese legt das orthografische Frustum der Voxelisierungskamera fest. Die Voxelisierungskamera blickt entlang der Achse der Bounding-Box, für die sie die geringste Ausdehnung hat, das heißt entlang der schmalsten Seite des Quaders. Damit werden die 128 zur Verfügung stehenden Tiefenebenen, bzw. die Voxelauflösung in z-Richtung, bestmöglich ausgenutzt.

Für die binäre Voxelisierung lohnt sich eine Vorvoxelisierung der statischen Geometrie mit anschließendem Einfügen der dynamischen Objekte insbesondere dann, wenn ein großer Teil der Szene statisch ist. Der Einsatz einer vorvoxelisierten statischen Umgebung, in die mehrere einzelne dynamische Objekte eingefügt werden, erwies sich daher in dieser Arbeit als die beste Variante. Bei der nicht-binären Voxelisierung ist der Performancegewinn geringer, der Ansatz wird dennoch verfolgt.

Daten, die in nicht-binären Volumen gespeichert werden, sind die Leuchtdichte der direkten Beleuchtung (RGB), Normalen und Material. Neben der Auflösung ist die Wahl des Texturformats für den Speicherplatzbedarf maßgebend. Bei der Entscheidung, wie viele Bits ein Voxel zur Verfügung hat, muss ein Kompromiss zwischen Performance (Zugriffszeit auf das Volumen), Speicherplatz und Präzision der Daten gefunden werden.

Es wurde zunächst versucht, mit 8 Bit pro Kanal auszukommen. Bei dem 8-Bit-Format wird der Wert intern auf den Bereich $[0, 1]$ geclampt[12]. Die Normalen und die Leuchtdichte müssen bei der Voxelisierung in diesen Bereich transformiert und beim Auslesen zurücktransformiert werden. Für die Normalen ist dies unproblematisch, da der Wertebereich der Komponenten bekannt ist: [-1, 1]. Schwierig ist jedoch die Behandlung der Leuchtdichte. Für die manuelle Transformation nach [0, 1] muss der maximale Wert bekannt sein. Theoretisch wäre dies möglich, indem auf den Atlanten zunächst eine Reduktionsoperation durchgeführt wird, um das Maximum zu ermitteln. Die geringe Präzision macht sich durch Color Banding bemerkbar. Abbildung 13 zeigt ein Volumen mit 8 Bit pro Kanal im Vergleich zu einem mit 16 Bit. Es stellte sich bei der Implementierung heraus, dass die Verwendung von 16 statt 8 Bits pro Kanal die Voxelisierungszeit nicht beeinträchtigt und auch bei dem Zugriff auf die Textur zum Auslesen der Werte die Zeiten kaum voneinander abweichen. Der verdoppelte Speicherplatz stellte auf dem verwendeten System kein Problem dar, daher wurde die 16-Bit-Variante bevorzugt eingesetzt.

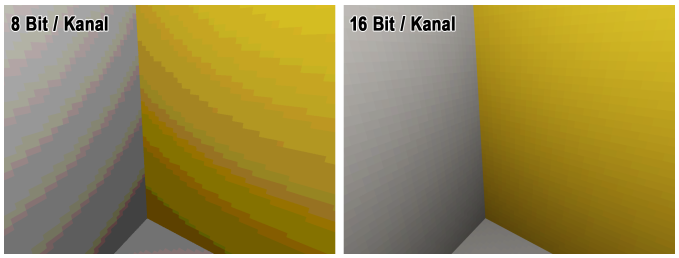

Abbildung 13: Darstellung der Leuchtdichte-Voxel (Ausschnitt): Das 8-Bit-Format diskretisiert den Leuchtdichte-Umfang auf 256 Stufen, was sich ab einem gewissen Dynamikumfang durch Color Banding bemerkbar macht.

[12] clamping = alle Werte unterhalb eines Minimums werden auf das Minimum abgebildet, alle Werte oberhalb eines Maximums auf das Maximum

5 Strahlschnitttest mit binärer Voxelhierarchie

5.1 Konzept und Zielsetzung

Das klassische Volumenrendering von medizinischen Datensätzen verwendet Techniken zum Überspringen von leerem Raum (*Empty-Space Skipping* oder *Leaping*). Oftmals wird hierfür ein *Octree* als hierarchische räumliche Datenstruktur eingesetzt. Ein Octree ist ein Baum, dessen Knoten jeweils acht oder keine Kinder haben. Da medizinische Daten im Allgemeinen statisch sind, wird der Octree in einem Vorverarbeitungsschritt angelegt [Engel et al. 2006, S. 196]. Empty-Space Skipping findet sich auch beim Traversieren von Octrees, die als Beschleunigungsdatenstruktur für Ray Tracing dienen. Eine an die Knotengröße angepasste Schrittweite reduziert die Anzahl der nötigen Schritte im Vergleich zu einem uniform aufgelösten Gitter (*Uniform Grid*).

Das gleiche Konzept kann für den Schnitt von Strahlen oder Geradensegmenten mit einer binären Voxelszene angewendet werden. Das Zusammenfassen von Voxeln erlaubt es, die Voxelstruktur effizienter zu durchqueren. Die Frage des *Sichtbarkeitstests* lautet für Geradensegmente „Befindet sich etwas zwischen dem Start- und Endpunkt?“, für Strahlen „Existiert für diese Richtung ein Schnitt mit der Szene?“. Eine andere Fragestellung ist: Wo liegt von der Startposition aus betrachtet der *erste Schnittpunkt* mit der Szene auf dem Strahl oder dem Segment? Für beide Fragestellungen gilt: Bereiche, die garantiert leer sind, können direkt übersprungen werden. Nur Bereiche, in denen sich etwas befindet, müssen genauer untersucht werden. Für diese Bereiche ist nur bekannt, *dass* sich etwas darin befindet, nicht *wo*.

Uniformes Ray Marching untersucht auch in großen leeren Bereichen jedes einzelne Voxel darauf, ob dieses voll oder leer ist. Dadurch wird Rechenzeit verschwendet. Das Überspringen leerer Regionen ist daher eine Erfolg versprechende Technik für Szenen, die ausreichend große zusammenhängende leere Bereiche haben. Stark gefüllte Szenen profitieren weniger.

Die folgenden Abschnitte stellen ein Verfahren für einen Schnitttest vor, das beide oben genannten Fragestellungen beantworten kann. Es läuft komplett auf der Grafikkarte und ist schneller als uniformes Ray Marching.

5.2 Erzeugung der hierarchischen Voxelstruktur

Die Grundlage bildet eine binäre Voxelisierung, die in Form einer zweidimensionalen Integer-Textur vorliegt. Das Verfahren zur Erzeugung der Voxel

spielt keine Rolle. In dieser Arbeit wird die in Kapitel *4 Voxelisierung mit Texturatlanten* vorgestellte Atlas-Voxelisierung verwendet, da sie schnell und für dynamische Szenen geeignet ist. Die Hierarchie muss ebenfalls in sehr kurzer Zeit erstellt werden können, da sie in jedem Frame neu aufgebaut wird.

Die Grundlage des Verfahrens wurde bereits in der vorangegangenen Studienarbeit geschaffen. Nahezu parallel veröffentlichten [Forest et al. 2009] einen identischen Ansatz. Für die zweidimensionale Voxeltextur werden Mipmaps[13] angelegt und diese in jedem Frame mit Inhalt gefüllt. Im Folgenden werden die Begriffe „fein“ und „grob“ zur Einordnung der Mipmap-Stufen verwendet. Das feinste Level ist Level *0* mit einer Auflösung von $2^n \times 2^n$, das gröbste Level hat die maximale Mipmap-Stufe n mit einer Auflösung von 1×1. Abbildung 14 stellt die verschiedenen Mipmap-Stufen anhand einer simplen Szene dar.

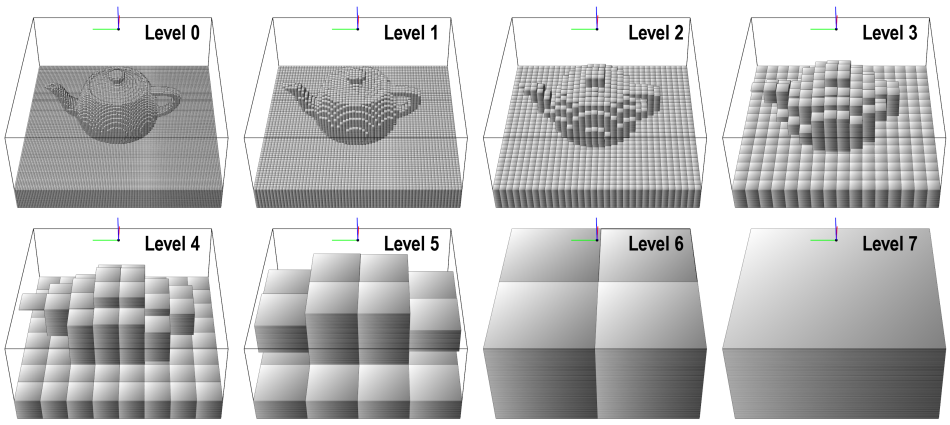

Abbildung 14: Mipmap-Stufen von Level 0 (128^3) bis Level 7 ($1 \times 1 \times 128$). Das Voxelisierungsfrustum ist die Bounding Box der Szene. Die Voxelisierungskamera blickt entlang der negativen y-Achse des Weltkoordinatensystems. Dies ist die z-Richtung der Voxelisierung. Die Reduzierung wird entlang x- und y-Achse des Voxelisierungskamera-Koordinatensystems ausgeführt.

Der Ausgangspunkt für das Mipmapping ist die feinste Stufe, die originale Voxeltextur. Für jedes Texel der nächstgröberen Stufe werden jeweils 2×2 Voxelspalten (Texel) der feineren Stufe verodert. Auf diese Weise bleibt entlang der Voxelisierungsrichtung die Auflösung von 128 Bits erhalten. Die Reduzierung entlang dieser Achse ist durch weitere Bitoperationen zwar möglich, es ist jedoch für die Strahltraversierung geschickter, die Information nicht zu verlieren (vgl. Abbildung 19 auf Seite 40). Zudem wird trotz einer Reduzierung in Voxelisierungsrichtung kein Speicher gespart, da das

[13] auch MIP-Map, MIP = lat. „multum in parvo“, dt. „Vieles im Kleinen“

Texturformat für die Mipmap-Stufen jenem der feinsten Stufe entsprechen muss.

Die Gesamtheit der Mipmap-Stufen bildet eine Menge von Quadtrees. Jede Ebene der Tiefenbits wird durch einen eigenen Quadtree repräsentiert. Ein Octree würde erst durch Reduzierung der Tiefenbits entstehen.

5.3 Traversieren der Voxelstruktur

5.3.1 Wahl von Offsets

Im betrachteten Szenario wird der Schnitttest für den Lichtaustausch genutzt. Daher starten die untersuchten Strahlen stets von Punkten aus, die auf Objektoberflächen liegen. Da alle Objekte voxelisiert wurden, muss verhindert werden, dass der Schnitttest die unmittelbar um die Startposition herum gelegenen Voxel testet und sofort abbricht. Dasselbe gilt für eine definierte Endposition, von der bekannt ist, dass sie auf einer Objektoberfläche liegt. Im Bild würde sich dieses Selbstschnitt-Verhalten durch ungewollte Selbstverschattung und zum Shadow-Mapping ähnliche Artefakte (oft „shadow acne“ genannt) zeigen (siehe Abbildung 15, rechts).

Offsets, die die ursprüngliche Start- bzw. Endposition geschickt verschieben, vermindern das Problem. Gegeben sind die Normale $\vec{N}_{start}$ am Startpunkt P_{start} und die Strahlrichtung $\vec{d}$ bzw. der Verbindungsvektor zwischen Start- und Endpunkt. Die Startpunktnormale gibt einen Hinweis darauf, wie die Nachbarvoxel angeordnet sein könnten. Ein Strahl, der in Bezug auf die Oberfläche, auf der der Startpunkt liegt, sehr flach verläuft, ist anfällig für Selbstschnitte. „Flach“ heißt hier, dass die Strahlrichtung eher orthogonal als parallel zur Normalen ist. Eine Lösung ist ein winkelabhängiger Offset entlang der Strahlrichtung. [Papaioannou et al. 2010] führen diesen Offset

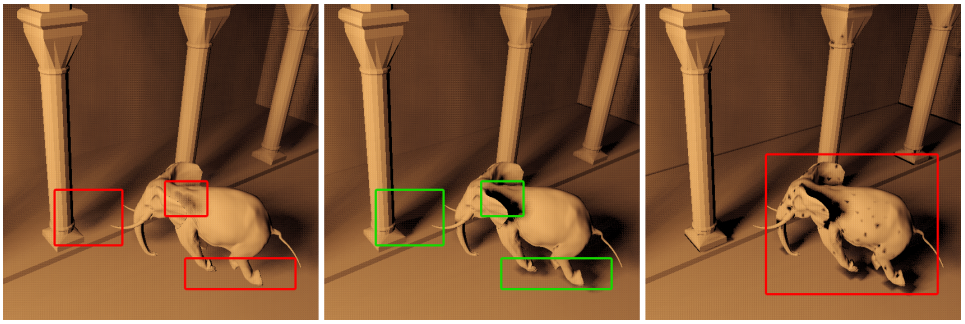

Abbildung 15: Indirekte Schatten. Links: Ein zu großer Offset führt zu fehlenden Kontaktschatten. Mitte: Passende Offset-Wahl für diese Szene. Rechts: Ein zu kleiner Offset führt zu Artefakten, da die objekteigenen Voxel Schatten auf das Objekt werfen.

in Verbindung mit der Voxeldiagonalen h ein: $d_0 = h/(\vec{d} \cdot \vec{N}_{start})$. Das Skalarprodukt im Nenner ist der Cosinus des Winkels zwischen Normale und Strahlrichtung. Dieser Offset wird bei flachen Strahlen sehr groß, sodass diese im Endeffekt verworfen werden. Hier zeigte sich ein Problem in Form von „schwebenden Objekten“: Kontaktschatten gehen verloren durch zu viele verworfene Strahlen (siehe Abbildung 15, links). Ein manuell gesetzter Skalierungsfaktor hilft kaum, da sich dann Flächen in teilweise großen Bereichen selbst verdecken.

Ein Kompromiss kann in Kombination mit einem zweiten Offset gefunden werden: Dieser zweite Offset verschiebt zunächst den Startpunkt entlang der Normalen um eine Voxeldiagonale. Auch dieser Offset kann mit einem manuell gesetzten Skalierungsfaktor für jede Szene angepasst werden. Die Verschiebung um eine ganze Voxeldiagonale behandelt den Worst Case, dass sich der voxelisierte Startpunkt an einer Ecke des zugehörigen Voxels befindet und die Strahlrichtung entlang der Voxeldiagonalen verläuft. Zusätzlich wird die Position um den skalierten winkelabhängigen Offset entlang der Strahlrichtung verschoben. Durch eine geschickte Kombination beider Offsets und Skalierungsfaktoren ist es möglich, schwebende Objekte und Selbstverdeckung zu vermeiden (Abbildung 15, Mitte).

Findet der Schnitttest zwischen zwei Punkten statt, wird für die Endposition analog verfahren, falls die Normale bekannt ist. Für einen Strahl ohne definierten Endpunkt wird nur die Startposition behandelt.

5.3.2 Schneiden von Voxelspalten mit Strahl-Bitmustern

Der Strahlschnitttest beruht auf zwei Beobachtungen:

- Ein *leeres Voxel* auf einer groben Mipmap-Stufe bedeutet, dass sich in dem durch die Voxelabmessungen definierten Bereich keine Geometrie befindet. In diesem leeren Bereich kann kein Schnittpunkt gefunden werden. Ein *volles Voxel* bedeutet, dass sich in dem Bereich Geometrie an unbekannter Stelle befindet. Das heißt, auf der feinsten Stufe muss in diesem Bereich mindestens ein volles Voxel gesetzt sein. Ein Schnittpunkt des Strahls mit den dort gesetzten Voxeln ist möglich. Auf der feinsten Stufe wird entschieden, ob tatsächlich ein Schnitt vorliegt.
- Ein Texel der Voxeltextur ist eine *Voxelspalte*: in der Voxelisierungsrichtung in einer Reihe hintereinander liegende Voxel. Der zugehörige Hüllkörper ist ein *Quader*, dessen Achsen gemäß denen des Koordinatensystems der Voxelisierungskamera ausgerichtet sind. Alle Quader haben die gleiche Länge, nämlich den Abstand zwischen der Near- und Far-Plane der Voxelisierungskamera. Ihre Höhe und Breite hängen von der betrachteten Mipmap-Stufe ab. Sie lassen sich aus der Höhe

und Breite des Voxelfrustums sowie der Auflösung der Mipmap-Stufe berechnen.

Für die nachfolgenden Betrachtungen wird angenommen, dass sich der Startpunkt des Strahls immer innerhalb des Voxelfrustums befindet. Da das Volumen endlich ist, können keine unendlich langen Strahlen verfolgt werden. Der maximal mögliche Endpunkt ist der Schnittpunkt des Strahls mit den Grenzflächen des Volumens.

Um teure Konvertierungsschritte zwischen verschiedenen Koordinatensystemen während des Tests zu vermeiden, finden alle Berechnungen im Einheitsvolumen $[0, 1]^3$ statt.

Nachdem die in Weltkoordinaten vorliegende Start- und Endposition jeweils um die in *5.3.1 Wahl von Offsets* erläuterten Offsets verschoben wurde, durchläuft sie folgende Transformationsschritte: Mit der View-Transformation der Voxelisierungskamera und anschließender orthografischer Projektion entstehen normierte Gerätekoordinaten (*Normalized Device Coordinates*) im Wertebereich $[-1, 1]^3$ (kanonisches Volumen). Eine perspektivische Division entfällt bei der orthografischen Projektion (bzw. Division durch 1). Die normierten Koordinaten werden durch Skalierung und Verschiebung in den Bereich $[0, 1]^3$ transformiert. Die gesamte Transformation kann in einer einzigen Matrix zusammengefasst werden:

$$M_{transform} = M_{translate} \cdot M_{scale} \cdot M_{proj} \cdot M_{view}$$

M_{view} ist die View-Transformation der Voxelisierungskamera, M_{proj} die orthografische Projektionsmatrix. $M_{translate}$ ist eine Translationsmatrix mit dem Translationsvektor $(0.5, 0.5, 0.5)$, M_{scale} eine Skalierungsmatrix mit den Skalierungsfaktoren $(0.5, 0.5, 0.5)$. Die resultierenden Koordinaten sind zugleich Texturkoordinaten für die Voxeltextur.

Beim Ray Marching wird mit einzelnen Abtastpunkten nach vollen Voxeln in der Voxeltextur gesucht. Für jeden Abtastpunkt wird geprüft, ob sich dieser innerhalb eines vollen Voxels befindet.

Der hier vorgestellte Schnitttest arbeitet stattdessen mit ganzen Strahlabschnitten innerhalb der Mipmap-Hierarchie. Ein Strahlabschnitt wird durch volle Voxel repräsentiert (*Strahl-Bitmuster*). Der Schnitttest kann auf jeder Mipmap-Stufe beginnen. Manche Start-Stufen führen jedoch schneller zum Ergebnis.

Abbildung 17 auf Seite 37 zeigt den im Folgenden geschilderten Ablauf als Aktivitätsdiagramm. Startposition und „aktuelle Position“ auf dem Strahl begrenzen den Strahlbereich, der bereits auf einen Schnitt hin untersucht wurde. Zu Beginn jeder Iteration wird der untersuchte Strahl mit dem

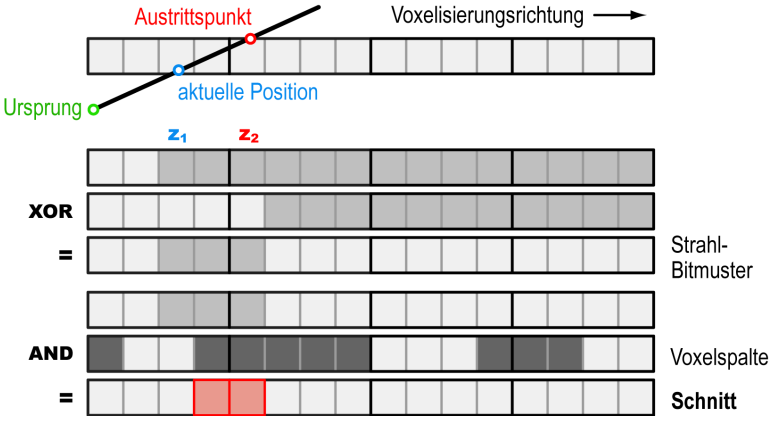

Abbildung 16: Konstruktion des Strahl-Bitmusters: Die Bitmasken für z_1 und z_2 werden aus einer 1D-Textur ausgelesen und mit einer XOR-Operation verknüpft. Das entstandene Strahl-Bitmuster wird mit der zugehörigen Voxelspalte geschnitten (AND-Operation).

Voxelspaltenquader geschnitten, innerhalb dessen sich die aktuelle Position[14] auf dem aktuellen Mipmap-Level befindet. Das gesuchte Ergebnis ist der skalare Strahlparameter, der in die parametrische Strahlgleichung eingesetzt den Austrittspunkt ergibt. Zwischen dem z-Wert des Austrittspunktes und der aktuellen Position wird ein Voxelstrahl entlang der Voxelisierungsrichtung aufgespannt. Diese Voxel werden nun mit der Voxelspalte geschnitten, die aus der Mipmap-Voxeltextur an der entsprechenden Position ausgelesen wurde (vgl. Abbildung 16). Der Schnitt ist eine bitweise AND-Operation, welche auf der Grafikkarte für alle 128 Bits gleichzeitig ausgeführt wird. Zwei Fälle können auftreten:

- Falls *alle resultierenden Bits 0* sind, heißt dies, dass der Strahlabschnitt hier einen leeren Bereich durchquert hat. In diesem Fall wird der nächste Strahlabschnitt berechnet und untersucht. Dafür wird die aktuelle Position auf dem Strahl auf die Position des Austrittspunktes gesetzt und um einen sehr kleinen Offset entlang des Strahls vorgerückt. Diese Verschiebung stellt sicher, dass im folgenden Traversierungsschritt der benachbarte Quader untersucht wird.
- *Mindestens ein gesetztes Bit* im Ergebnis-Bitmuster bedeutet, dass sich in der Region auf der feinsten Mipmap-Stufe volle Voxel befinden, die der Strahl potentiell schneiden kann. In diesem Fall muss die nächstfeinere Hierarchiestufe untersucht werden. Der Strahl wird mit dem nächstfeineren Voxelspaltenquader geschnitten. Die anschließenden Schritte sind analog zu den oben geschilderten. Wurde der aktuelle

[14] in der ersten Iteration des Schnitttests ist dies die Startposition

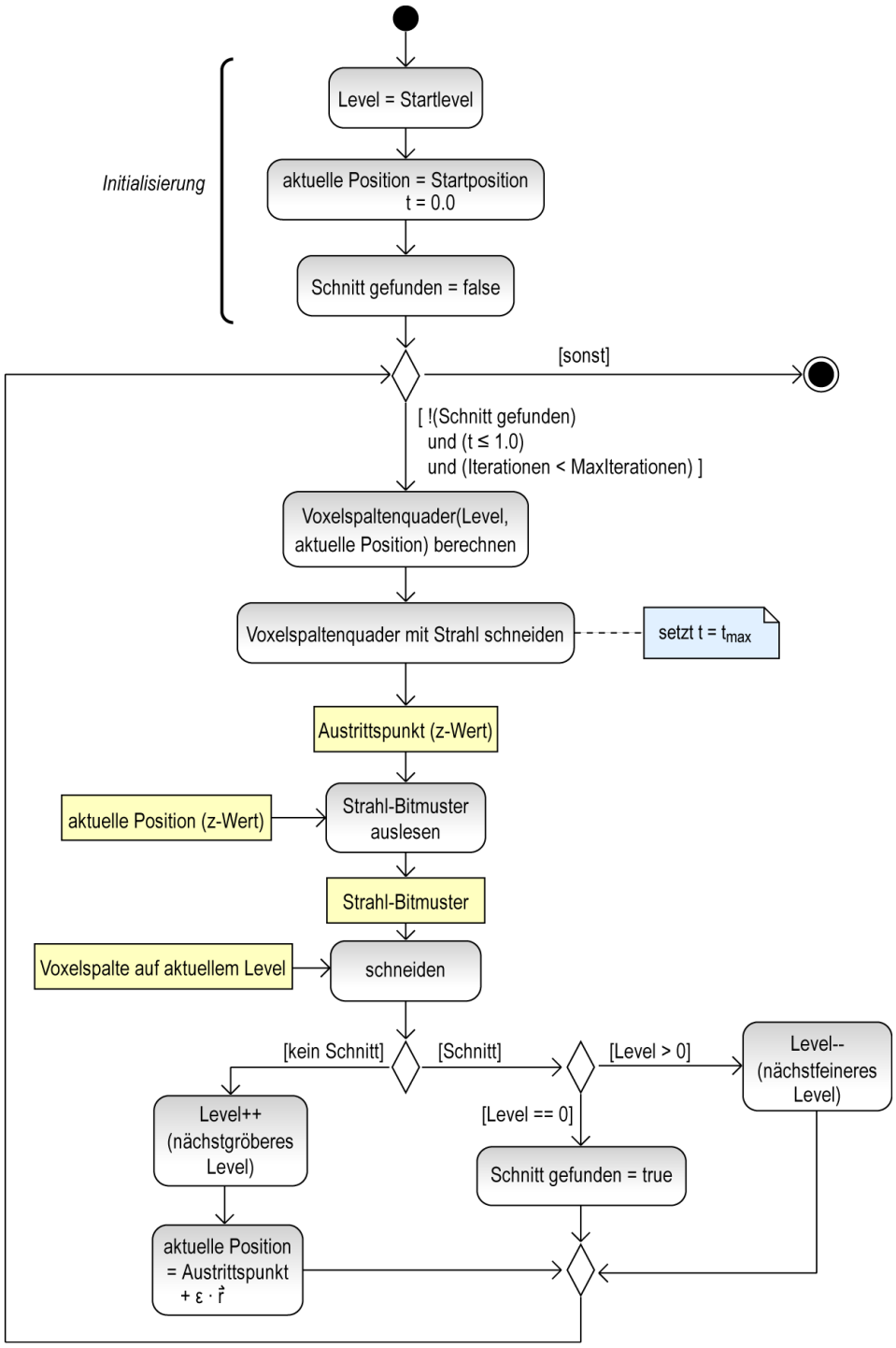

Abbildung 17: Ablauf des Strahlschnitttests. $\vec{r}$ ist der Vektor zwischen Startposition (O) und Endposition. t ist der Parameter der parametrischen Strahlgleichung: $O + t \cdot \vec{r}, t \in [0, 1]$.

Durchlauf bereits auf dem feinsten Level ausgeführt, so steht fest, dass ein Schnitt gefunden wurde.

Der Schnitttest ist beendet,

- sobald ein Schnitt gefunden wurde,
- wenn der gesamte Strahl untersucht wurde

  oder

- wenn die Traversierungsschritte eine manuell gesetzte maximale Anzahl übersteigen.

In den letzteren beiden Fällen lautet das Ergebnis: kein Schnitt gefunden.

[Forest et al. 2009] legen dar, dass es vorteilhaft für die Strahlverfolgung ist, die Tiefenbits nicht zu gröberen Blöcken zusammenzufassen. Abbildung 19 auf Seite 40 zeigt, dass mit der vollen Tiefeninformation für bestimmte Strahlabschnitte auf groben Levels besser erkannt werden kann, dass feinere Mipmap-Stufen nicht untersucht werden müssen.

5.3.3 Beispieldurchlauf

Abbildung 18 veranschaulicht den Schnitttest anhand eines Beispieldurchlaufs.

Schritt 1: Das Segment wird mit dem Quader geschnitten (es interessiert jeweils nur der Austrittspunkt). Das Strahl-Bitmuster wird für den z-Bereich des geschnittenen Segments konstruiert und mit der Voxelszene geschnitten.

Schritt 2: Da in Schritt 1 nichts geschnitten wurde, wird die nächstgröbere Mipmap-Stufe (Level 2) untersucht. Das verbliebene Segment wird mit dem entsprechenden Quader geschnitten, das Strahl-Bitmuster erstellt und mit der Szene geschnitten. Es wird ein möglicher Schnitt erkannt (rotes Voxel im Bild).

Schritt 3: Da ein möglicher Schnitt erkannt wurde, muss die nächstfeinere Stufe in dem Bereich untersucht werden. Dies ist Level 1. Das Ergebnis lautet wieder: Schnitt möglich.

Schritt 4: Es wird erneut in der Hierarchie abgestiegen und die feinste Stufe (Level 0) untersucht. Ein Schnitt auf dieser Stufe (volles Voxel erkannt) bedeutet, dass ein Schnitt gefunden wurde. Der Test ist beendet.

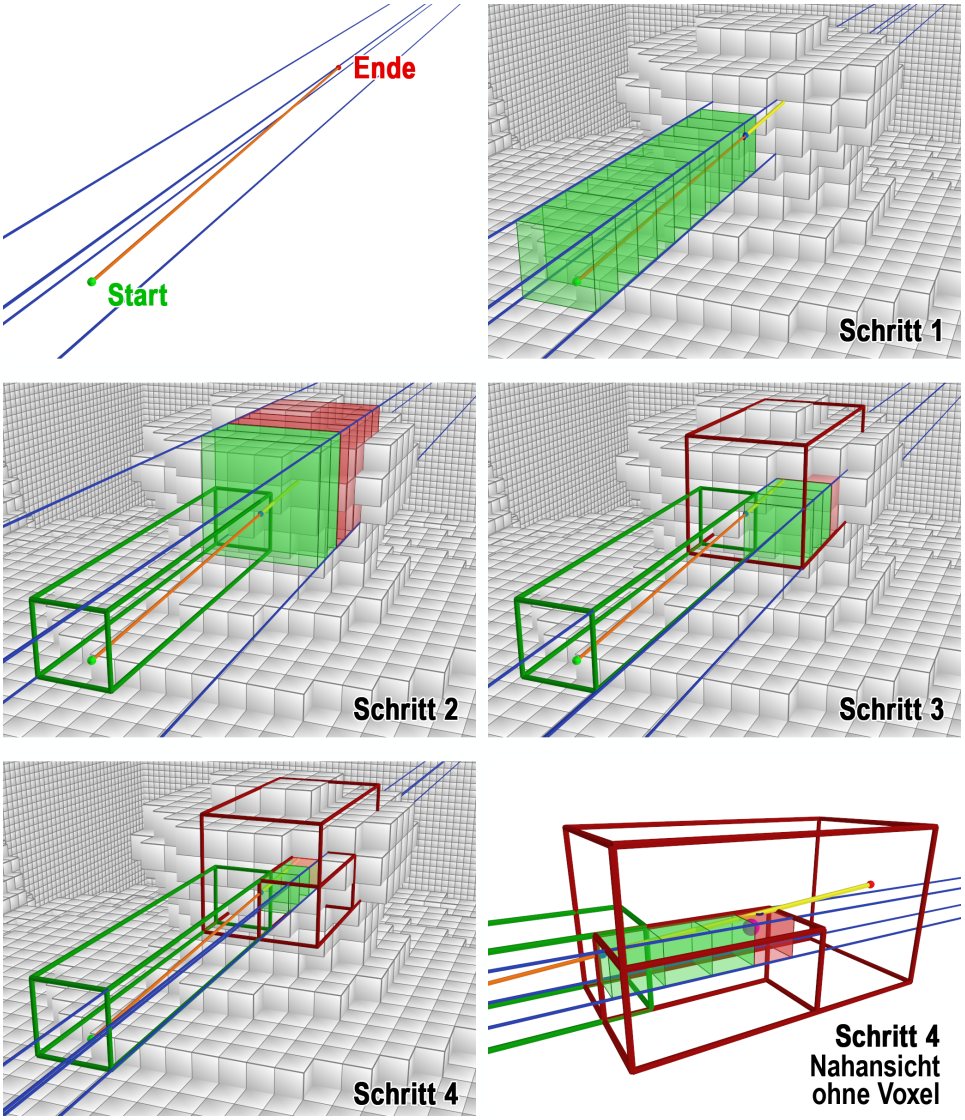

Abbildung 18: Visualisierung des Schnitttests für einen einfachen Fall. Untersucht wird das orangene Geradensegment (links oben). Der Schnitttest startet hier bei Mipmap-Stufe 1. Die blauen Linien stellen die Kanten des aktuellen Voxelspaltenquaders auf Stufe 1 dar. Das Bitmuster ist als grüne oder rote transparente Voxel dargestellt – grün = leeres Voxel in binärer Voxelszene, rot = volles Voxel. Die grünen und roten Quader stellen die „Historie“ des Schnitttests dar: Welche Bereiche wurden bereits mit welchem Ergebnis untersucht?

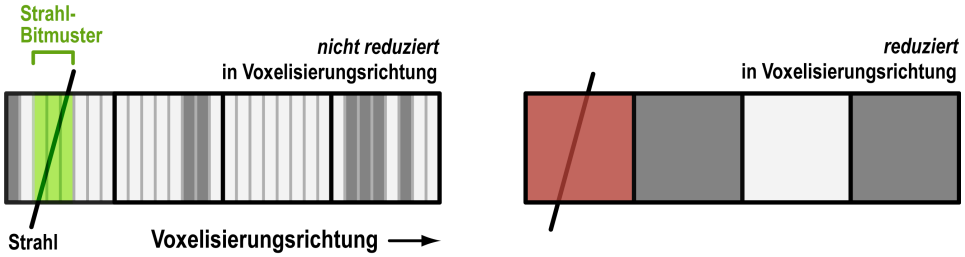

Abbildung 19: Vorteil der feinen Auflösung in Voxelisierungsrichtung:
Links: In diesem Beispiel kann mit dem Strahl-Muster direkt erkannt werden, dass der Strahl in dem Bereich keine vollen Voxel schneidet. Rechts: Dagegen liefern die in Voxelisierungsrichtung zusammengefassten Voxel das Ergebnis „möglicher Schnitt“, was eine genauere Untersuchung des Bereichs zur Folge hat. Abbildung in Anlehnung an [Forest et al. 2009].

5.3.4 Finden eines genauen Schnittpunktes

Wurde ein Schnitt gefunden, so ist noch nicht bekannt, welches Voxel genau der Voxelstrahl als erstes getroffen hat. Optional kann der Schnitttest daher um eine Bestimmung des „genauen“ (voxelgenauen) Schnittpunktes erweitert werden.

Innerhalb der Voxelspalte, in der der Schnitt stattfindet, muss das vorderste oder hinterste volle Voxel gefunden werden, um eine genauere z-Koordinate berechnen zu können. Anders gesagt wird die Stelle des vordersten oder hintersten gesetzten Bits innerhalb der 128 Bits gesucht, die aus der Verundung beim Schnitt hervorgingen. Ob das vorderste oder hinterste Bit gesucht ist, hängt von der Richtung des Strahls in Bezug auf die Voxelisierungsrichtung ab. Die 128 Bits sind in 4 Texturkanälen gespeichert. Der Rot-Kanal der Voxeltextur enthält die von der Voxelkamera aus gesehen am weitesten vorn liegenden Voxel. Verläuft die Strahlrichtung ähnlich zur Voxelisierungsrichtung[15], wird die Stelle des vordersten Bits gesucht, und umgekehrt bei entgegengesetzter Richtung die Stelle des hintersten Bits. Nachdem der Kanal mit dem gesuchten Bit ermittelt wurde, muss die Bitposition des vordersten oder hintersten Bits berechnet werden. Dies ist zum Beispiel mit einem Zweierlogarithmus möglich. Für das hinterste Bit muss dieses zuvor extrahiert werden, beispielsweise durch folgende Bitoperation: `x = x & ~(x-1)`. Anschließend wird die ermittelte Bitposition in das für den Schnitttest verwendete Koordinatensystem überführt, und die z-Koordinate der aktuellen Position auf dem Strahl auf diesen Wert gesetzt. Diese Koordinate ist die ungefähre Schnittposition.

[15] Kriterium: Skalarprodukt der normierten Vektoren > 0

6 Indirektes Licht durch Abtastung der Hemisphäre

6.1 Überblick

Das Ziel ist die näherungsweise Lösung der in Abschnitt *2.1 Direkte und indirekte Beleuchtung* eingeführten Rendering Equation. Das in diesem Kapitel erläuterte Verfahren wurde zunächst für diffuses indirektes Licht mit einer Interreflexion konzipiert und später auf mehrere Interreflexionen und glänzendes Phong-Material erweitert. Die wichtigsten Schritte des Verfahrens sind:

- Atlas-Rendering und binäre (je nach Umsetzung und Erweiterung zusätzlich auch nicht-binäre) Voxelisierung der Szene
- G-Buffer-Rendering (Inhalt: Position und Normale in Weltkoordinaten sowie Material)
- Abhängig vom konkreten Verfahren: Rendern der Szene aus Sicht der Lichtquellen für Spot Maps und Cube Maps
- Verschießen von Strahlen in ausgewählte Richtungen der Hemisphäre jedes G-Buffer-Pixels und Bestimmen des vordersten Schnittpunkts
- Auslesen bzw. Berechnen der Leuchtdichte, welche die getroffene Oberfläche zum Strahlursprung reflektiert
- Akkumulieren aller Strahl-Ergebnisse
- Filtern des finalen Bildes, das die indirekte Leuchtdichte enthält

6.2 Schätzung der indirekten Leuchtdichte

6.2.1 Monte-Carlo-Integration

Die Monte-Carlo-Integration wird in dieser Arbeit verwendet, um die indirekte Leuchtdichte für alle von der Kamera aus gesehenen Oberflächen zu berechnen. Bei dem gewählten Deferred-Shading-Ansatz ist dies letztlich ein Leuchtdichte-Wert pro Pixel des zu berechnenden Bildes.

Falls in dem Integral aus Gleichung (2) auf Seite 6 nur andere Objekte als mögliche Sender betrachtet werden – nicht aber die tatsächlichen Lichtquellen –, so berechnet das Integral die Leuchtdichte, die die Oberfläche aufgrund von indirekter Beleuchtung abgibt. Dieses Integral ist im Allgemeinen nicht analytisch lösbar. Stochastische Monte-Carlo-Methoden bieten eine Möglichkeit, den Wert des Integrals mit Hilfe von Zufallsexperimenten näherungsweise zu berechnen.

Das Integral einer eindimensionalen Funktion $f(x)$ lässt sich mit einem Monte-Carlo-Schätzer (rechte Seite der Gleichung) wie folgt annähern [Pharr und Humphreys 2004, S. 636]:

$$I = \int_a^b f(x)\,\mathrm{d}x \;\approx\; \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)} = F_N \tag{3}$$

mit	N	Anzahl der Stichproben *(Samples)*
	X_i	Zufallsvariable $\in [a, b]$
	$p(x)$	Dichtefunktion

Strebt N gegen unendlich, konvergiert der Näherungswert F_N gegen die tatsächliche Lösung I. Der Fehler $\epsilon = |F_N - I|$ halbiert sich aufgrund der Konvergenzrate $O(1/\sqrt{N})$ durch eine Vervierfachung der Stichproben [Pharr und Humphreys 2004, S. 632]. Im Bild ist der Fehler als Rauschen sichtbar (Rauschmuster, zu helle und zu dunkle Pixel nebeneinander, Abbildung 20).

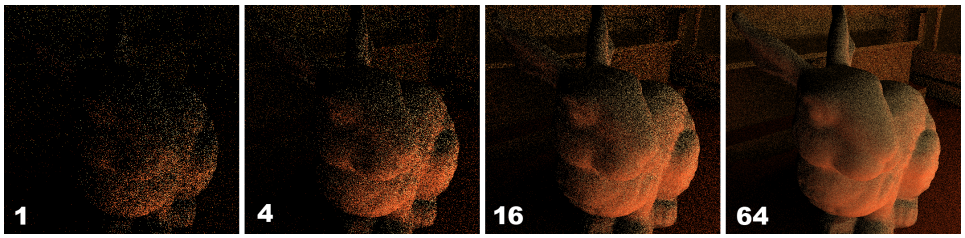

Abbildung 20: Rauschen im Bild bei der Monte-Carlo-Integration der Hemisphäre. Die Zahlen unten links in den Bildern benennen die Anzahl der pro Pixel verfolgten Strahlen.

Die Monte-Carlo-Integration kann auch für Integrale höherer Dimensionen angewendet werden. Die Konvergenzrate bleibt dabei gleich.

Die Stichproben für die Schätzung des Leuchtdichte-Integrals sind zufällige Einfallsrichtungen aus dem vorderen Halbraum des betrachteten Oberflächenelements. Der vordere Halbraum kann als Hemisphäre mit dem Raumwinkel $\Omega = 2\pi$ aufgefasst werden, die gemäß der Normalen des Flächenelements orientiert ist. Diese Betrachtungsweise eignet sich für die Darstellung der Richtungen in Kugelkoordinaten. Eine Richtung $\vec{\omega}$ wird durch das Winkelpaar (φ,θ)[16] beschrieben.

[16] Azimut- und Polarwinkel

Mit der Monte-Carlo-Integration wird das Integral über die Hemisphäre zu einer Summe über alle zufällig gewählten Einfallsrichtungen X_i:

$$\begin{aligned} &\int_\Omega f_r(\mathrm{d}A_e, \mathrm{d}\vec{\omega}_i, \mathrm{d}\vec{\omega}_o) \cdot L_i(\mathrm{d}A_e, \mathrm{d}\vec{\omega}_i) \cdot \cos\theta_i \ \mathrm{d}\vec{\omega}_i \\ &= \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\mathrm{d}A_e, X_i, \mathrm{d}\vec{\omega}_o) \cdot L_i(\mathrm{d}A_e, X_i) \cdot \cos\theta_i}{p(X_i)} \end{aligned} \tag{4}$$

Die Summe kann durch die Wahl einer geeigneten Dichte weiter vereinfacht werden. Die Dichte bestimmt, welche Richtungen oft und welche seltener erzeugt werden.

6.2.2 Importance Sampling

Beim *Importance Sampling* wird die Dichte ähnlich zum Integranden gewählt. Die Funktion soll an Stellen abgetastet werden, an denen sie hohe Werte annimmt. Dies führt zu einer schnelleren Verringerung des Fehlers bei der Monte-Carlo-Integration. Im Falle der Leuchtdichte-Schätzung sollen also insbesondere die Richtungen ausgewertet werden, für die ein hoher Leuchtdichtebeitrag für das Ergebnis zu erwarten ist. Der Cosinus-Term im Integral legt nahe, ein cosinusgewichtetes Sampling durchzuführen. Ebenfalls bietet sich eine Gewichtung gemäß der BRDF an. Somit wird oftmals das Produkt von BRDF und Cosinus-Term als Grundlage für die Stichprobengenerierung herangezogen.

Diese Arbeit verwendet

- rein diffuse Materialien (Lambert-Modell) mit der BRDF

  $$f_r = \frac{\rho_d}{\pi} \ ,$$

  wobei ρ_d der diffuse Reflexionsgrad ist,
- und das physikalisch plausible Phong-Modell mit der BRDF

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

  wobei ρ_s der spekulare Reflexionsgrad und ψ der Winkel zwischen reflektierter Einfallsrichtung und Ausfallsrichtung ist.

Für beide Modelle existieren passende Dichtefunktionen zum Sampling der Hemisphäre [Dutré 2003, S. 19 f.]. Das Sampling wird in Abbildung 21 skizziert.

Für rein diffuses Material ist dies die Dichte

$$p(\vec{\omega}) = \frac{\cos\theta}{\pi} \ .$$

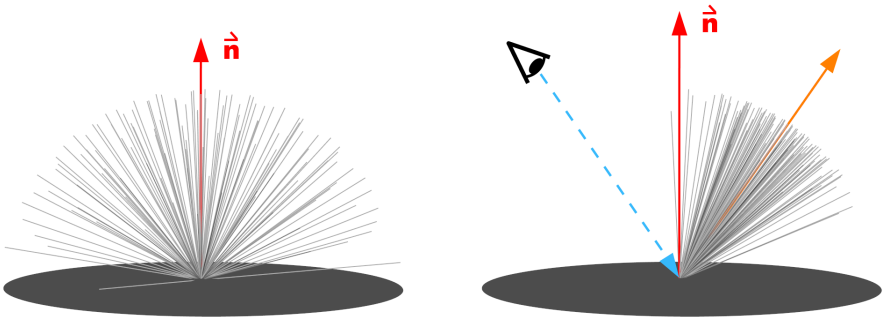

Abbildung 21: Importance Sampling der Hemisphäre: Links cosinusgewichtete Erzeugung der Strahlen für Lambert-Oberflächen. Rechts Erzeugung der Strahlen innerhalb der Phong-Keule für Phong-Oberflächen, hier für den ersten Bounce dargestellt. Es sollen die Richtungen abgetastet werden, aus denen viel Licht ins Auge fällt, also Richtungen um den reflektierten View-Vektor herum (View-Vektor: blau, reflektiert: orange).

Die Kugelkoordinaten der Richtung werden mit Hilfe zwei gleichverteilter Zufallszahlen (genannt ξ_1 und ξ_2) im Intervall $[0, 1]$ erzeugt:

$$(\varphi, \theta) = (\, 2\pi\xi_1,\, \arccos\sqrt{\xi_2}\,).$$

Für ein Phong-Material lautet die Dichte

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

und die Kugelkoordinaten der Richtung lauten

$$(\varphi, \theta) = (\, 2\pi\xi_1,\, \arccos\sqrt{\xi_2^{\frac{1}{n+1}}}\,).$$

Die Summe aus Gleichung (4) vereinfacht sich durch die Wahl der oben genannten Dichten für diffuses Material zu

$$\frac{\pi}{N}\sum_{i=1}^{N} L_i(\mathrm{d}A_e, X_i)$$

und für Phong-Material zu

$$\frac{1}{N}\left(\rho_s\,\frac{n+2}{n+1}\right)\sum_{i=1}^{N} L_i(\mathrm{d}A_e, X_i)\cdot\cos\theta_i\ .$$

Nach Umrechnung der Kugelkoordinaten in kartesische Koordinaten ist die Hemisphäre, innerhalb der die Richtungen erzeugt werden, je nach Lage

der Achsen im kartesischen Koordinatensystem zum Beispiel an dem Vektor (0,1,0) ausgerichtet. Die Hemisphäre bzw. die Richtungen müssen daher anschließend in die richtige Lage bezüglich des Oberflächenelements, von dem aus die Strahlen verfolgt werden, rotiert werden.

Bei diffusem Material wird die Hemisphäre an der Oberflächennormalen ausgerichtet. Beim Phong-Material ist die Phong-Keule zunächst so wie die Hemisphäre entlang des Vektors (0,1,0) orientiert. Sie muss so rotiert werden, dass ihre Hauptrichtung mit der Richtung übereinstimmt, die in erster Linie abgetastet werden soll. Diese bevorzugte Richtung ist jene, aus der das Licht vom Material in die Ausfallsrichtung reflektiert wird. Bei der ersten Interreflexion handelt es sich also um den reflektierten View-Vektor (von Auge zu Oberflächenelement), siehe auch Abbildung 21. Ein großer Teil des indirekten Lichts, das aus ähnlichen Richtungen auf das Oberflächenelement fällt, wird in das Auge abgestrahlt. Für weitere Bounces wäre die Hauptrichtung der an der Senderoberfläche reflektierte Vektor von Sender- zu Empfängeroberfläche.

6.2.3 Quasi-Monte-Carlo mit Low-Discrepancy-Sequenzen

Die Stichproben X_i werden aus (0,1)-verteilten Zufallszahlen generiert. Der Einsatz deterministischer *Low-Discrepancy-Sequenzen* anstelle von Zufallszahlen ermöglicht es, dass die Monte-Carlo-Integration schneller konvergiert. Im Bild ist dies durch eine Verminderung des Rauschens zu erkennen. Es handelt sich dem Namen nach um Sequenzen (eine Reihe von Zahlen oder Zahlentupeln), die eine niedrige Diskrepanz aufweisen. Die *Diskrepanz* ist ein mathematisch definiertes Maß dafür, wie gut die Gleichverteilung der Zahlenfolge ist. Eine niedrige Diskrepanz bedeutet, dass die Zahlen besser gleichverteilt sind.

Diese Arbeit setzt die *Hammersley-Sequenz* und die *Halton-Sequenz* ein. Die grundlegende Funktion zur Erzeugung dieser Sequenzen ist die sogenannte *radikal inverse Funktion* $\Phi_b(i)$ mit $i \in \mathbb{N}$ und Basis b. Sie konvertiert die Dezimalzahl i in das Zahlensystem mit der Basis b, spiegelt das Resultat am Komma und rechnet es wieder ins Dezimalsystem um. Das Ergebnis ist eine Zahl im Intervall $[0, 1]$.

Eine zweidimensionale Hammersley-Sequenz ist zum Beispiel als $X_i = (\frac{i}{N}, \Phi_2(i))$ definiert, sodass im Voraus die Länge N der Sequenz bekannt sein muss. Die Sequenz kann auf mehr Dimensionen erweitert werden, indem Elemente $\Phi_b(i)$ mit teilerfremden Basen dem Tupel hinzugefügt werden. Ein Beispiel für eine zweidimensionale Halton-Sequenz ist $X_i = (\Phi_2(i), \Phi_3(i))$. Als Basen können Primzahlen gewählt werden, da die Basen generell zueinander teilerfremd sein müssen.

6.3 Speichern und Auslesen der direkten Leuchtdichte

6.3.1 Direkte Beleuchtung als Grundlage für indirekte Beleuchtung

Das direkte Licht liefert die Ausgangsbasis für die Berechnung des indirekten Lichts. Im Folgenden wird der Einfachheit halber zunächst nur diffuses Material und eine Interreflexion (*Single Bounce*) betrachtet. Rein diffuses Material erleichtert den Umgang mit der Leuchtdichte, da es in alle Richtungen die gleiche Leuchtdichte reflektiert. Es ist möglich, die grundlegenden Konzepte mit einigen Änderungen und Ergänzungen auf Phong-Materialien und mehrere Interreflexionen zu übertragen.

Die unten stehende Gleichung (5) zeigt, welche Annahme der Berechnung einer einzigen Interreflexion zugrunde liegt. Zwecks Übersichtlichkeit wurde eine vereinfachte Notation ohne Argumente der Funktionen verwendet. L_o ist die Leuchtdichte, die im Auge des Betrachters ankommt. X_i ist eine während der Monte-Carlo-Integration erzeugte zufällige Richtung (ein Strahl) und L' die Leuchtdichte, die aus der Richtung auf das betrachtete Flächenelement fällt.

$$\begin{aligned} L_o &= L_{direct} + L_{indirect} \\ &= L_{direct} + f_r \cdot \frac{\pi}{N} \sum_{i=1}^{N} L'(X_i) \end{aligned}$$

wobei

$$\begin{aligned} L' &= L'_{direct} + L'_{indirect} \\ &= L'_{direct} + 0 \end{aligned} \tag{5}$$

Für jeden zufällig erzeugten Strahl X_i interessiert die Leuchtdichte L' aus dieser Richtung. Diese ist unbekannt und müsste wieder durch Monte-Carlo-Integration ermittelt werden. Für den ersten Bounce wird jedoch an dieser Stelle bereits abgebrochen. Die gesuchte Leuchtdichte wird als die direkte Leuchtdichte angenommen, die die von dem Strahl getroffene Oberfläche zum Strahlursprung (die Position des Flächenelements) abgibt. Das indirekte Licht am Strahlschnittpunkt wird also als null angenommen. Die wesentliche benötige Information ist daher jene über alle direkt beleuchteten Flächen. In dieser Arbeit wurden zwei Möglichkeiten untersucht, mit denen die direkte Beleuchtung der gesamten Szene repräsentiert werden kann. Diese werden im Folgenden (*6.3.2 Leuchtdichte-Voxel* und *6.3.3 Spot Maps und Cube Maps*) vorgestellt.

Abschnitt *6.4.1 Single Bounce* beschreibt die genaue Umsetzung für die Berechnung des indirekten Lichts mit einer Interreflexion. Die Erweiterung auf mehrere Interreflexionen wird in Abschnitt *6.4.2 Multiple Bounces (Voxel Path Tracer)* geschildert.

6.3.2 Leuchtdichte-Voxel

Die erste Idee war, die direkte Beleuchtung in nicht-binären Volumen zu speichern. Diffuses Material reflektiert in alle Richtungen der oberen Hemisphäre die gleiche Leuchtdichte. Daher muss für diffuses Material nur die Leuchtdichte pro Voxel gespeichert werden. Die 3D-Texturen werden mit der in Kapitel *4 Voxelisierung mit Texturatlanten* beschriebenen Methode erstellt. Wird bei der Strahlverfolgung in den binären Voxeln ein volles Voxel getroffen, wird die Leuchtdichte des entsprechenden Voxels aus der 3D-Textur ausgelesen.

Probleme bereiten sehr dünne Objekte. „Dünn“ heißt, dass Vorder- und Rückseite einen geringeren Abstand als eine Voxelbreite haben. Wie in *4.3.2 Ablauf der nicht-binären Voxelisierung* geschildert, hängt der Inhalt eines Voxels von der Reihenfolge ab, in der die Atlastextur-Pixelwerte in die Volumentextur eingefügt werden. Je nachdem, wie die Flächen im Atlas angeordnet sind, stammen die im Volumen eingetragenen Objektvoxel entweder alle von der Vorderseite, von der Rückseite oder von beiden. Das Ergebnis ist nicht vorhersehbar. Beim Sampling wird dann an einigen Stellen eine falsche Information ausgelesen. Das Volumen kann die Leuchtdichte-Information nicht genauer erfassen, als es die Voxelauflösung zulässt. In Abbildung 11 auf Seite 25 ist beispielsweise das dünne Ohr des Elefanten problematisch (dort sichtbar an der Normalenvoxelisierung).

Andere Objekte wiederum haben keine paarweise auftretenden Vorder- und Rückseiten. Hierbei kann das Problem des Überschreibens nicht auftreten. Gleichzeitig handelt es sich aber auch um ein unendlich dünnes Objekt. Im Atlas ist die beleuchtete Fläche eingetragen und somit in allen Voxeln die Leuchtdichte der beleuchteten Seite gespeichert. Es wird davon ausgegangen, dass die Rückseite schwarz ist. Dies erfordert die Erkennung der Rückseite beim Auslesen der Leuchtdichte aus den Voxeln. Nicht nur um diesen Sonderfall zu behandeln, sondern auch um andere falsche indirekte Beleuchtung zu vermeiden (*Light Bleeding*), wird geprüft, ob das Skalarprodukt von Strahlrichtung und Normale der getroffenen Fläche < 0 ist. Andernfalls wird die Leuchtdichte auf null gesetzt. Ein Volumen mit Normalen wird parallel zum Leuchtdichte-Volumen mittels Multiple Render Targets erzeugt, um auf diese Information beim Auslesen der Leuchtdichte zugreifen zu können.

6.3.3 Spot Maps und Cube Maps

Die direkt beleuchteten Flächen in der Szene sind diejenigen, die die punktförmigen Lichtquellen „sehen“. Aus dieser Tatsache entstand die zweite Idee zur Speicherung der direkten Leuchtdichte.

Bei *Spotlights* wird genauso wie beim Shadow Mapping eine Textur aus Sicht des Lichts erzeugt, allerdings wird statt der Tiefe die Leuchtdichte eingetragen. Texturen, die andere Informationen als die Tiefe enthalten, werden im Folgenden nicht Shadow Maps, sondern Spot Maps genannt. Die Leuchtdichte entsteht durch die alleinige Beleuchtung des aktuell betrachteten Spotlights. Für ein Spotlight ist wegen des begrenzten Lichtkegels, der das Viewing-Frustum definiert, eine einzelne 2D-Textur ausreichend.

Punktlichtquellen benötigen dagegen mehr Texturen bzw. eine andere Abbildung. Zwei Möglichkeiten, die für Punktlichtquellen in Frage kommen, sind paraboloide Abbildungen und Cube Maps. Da paraboloide Abbildungen eine ausreichend fein tesselierte Geometrie erfordern, verwendet diese Arbeit Cube Maps. Die sechs Würfelseiten sind durch sechs Frusta mit Öffnungswinkel 90° definiert. Der Ursprung ist die Lichtquellenposition, die sich in der Mitte des Würfels befindet. Cube Maps können durch sechs Render-Passes (einen für jede Würfelseite) oder in einem Pass mit Versechsfachung der Geometrie im Geometry-Shader erzeugt werden. Abbildung 22 zeigt Beispiele für Spot Maps und Cube Maps.

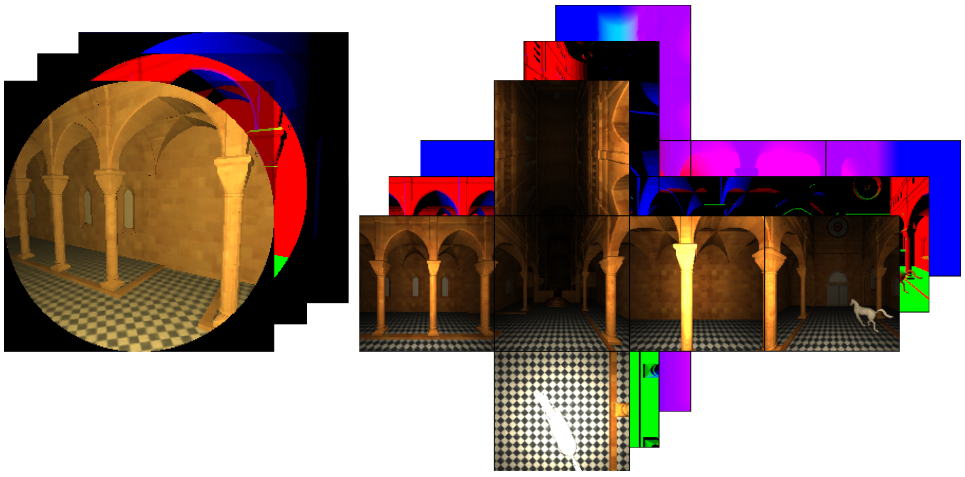

Abbildung 22: Links: Spot Maps (Auflösung 256^2, von vorne nach hinten: Leuchtdichte, Normalen, Weltkoordinate). Rechts: Cube Maps (Auflösung 192^2, Texturinhalte analog zu Spot Maps).

Die Strahlen werden wieder in der binären Voxelszene verfolgt. Ist ein volles Voxel getroffen, werden die Maps aller Lichtquellen an den entsprechenden Stellen ausgelesen. In jeder Map steht nur der Anteil, den das zugehöri-

ge Licht an der Beleuchtung der Szene hat. Indem alle Anteile aufaddiert werden, ergibt sich die Gesamtbeleuchtungssituation. Die ungefähre Schnittposition ist in Weltkoordinaten gegeben. Durch eine Transformation der Weltkoordinate ergibt sich die Texturkoordinate der Map. Für die Spot Maps handelt es sich um die Transformationsmatrix, die auch für Shadow Mapping eingesetzt wird: $M_{scaleBias} \cdot M_{lightProj} \cdot M_{lightView}$. Das Auslesen von Texeln aus einer Cube Map geschieht mittels dreidimensionaler Texturkoordinaten (Vektoren), die in der Mitte des imaginären Würfels starten. Daher muss hier nur der Verbindungsvektor von Lichtquellenposition zu Schnittpunktposition berechnet werden.

Eine Reihe von Bedingungen definiert, ob die ausgelesene Leuchtdichte gültig ist und verwendet werden darf. Einen optischen Hinweis darauf, welche Repräsentation des direkten Lichts durch Auslesen der Maps tatsächlich zugrunde liegt, liefert ein Ray Casting der binären Voxeltextur mit Auslesen der Spot und Cube Maps an den Trefferpositionen (Beispiel für Punktlichtquelle und Cube Map: siehe Abbildung 23). Anhand dieser Darstellung lässt sich die Auslese-Strategie überprüfen.

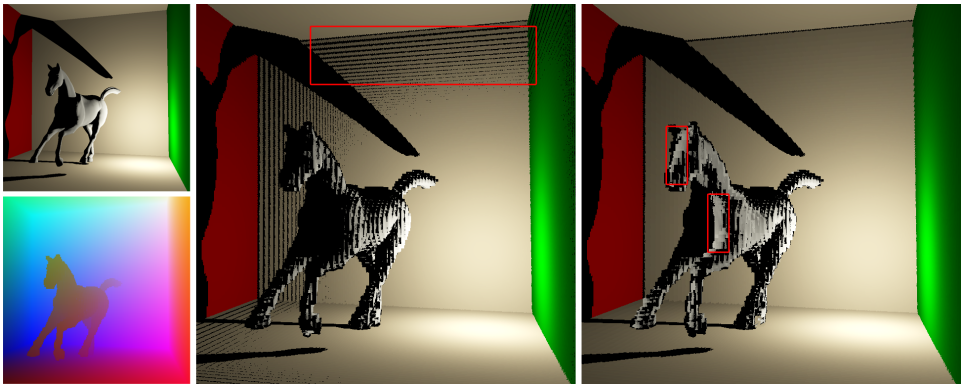

Abbildung 23: Links oben die normal gerenderte und direkt beleuchtete Szene. Links unten die Trefferpositionen des Ray Castings der binären Voxel. In der Mitte und rechts die in der Cube Map ausgelesenen Leuchtdichtewerte. Das Bild in der Mitte weist typische Shadow-Mappping-Artefakte auf, da hier die Neigung der ausgelesenen Pixel nicht beachtet wurde. Rechts wurde die Neigung in den Abstandstoleranzwert einbezogen – hier ist der Toleranzwert wiederum an einigen Stellen zu groß, was Light Bleeding verursacht (rot markiert).

Eine erste Variante nutzte eine zusätzliche Shadow Map, um verschattete Bereiche zu erkennen. Dies gelingt dadurch zwar, aber an vielen Stellen nehmen die Voxel mehr Raum ein als die wirkliche Geometrie. Auf diese Voxel werden falsche Informationen aus entfernten Bereichen projiziert. Deshalb wurde eine andere Lösung gewählt: Zu prüfen ist, ob sich der Abstand von Schnittpunktposition zur Position des ausgelesenen Texels in einem erlaubten Bereich befindet. Dafür wird eine Weltkoordinaten-Positionstextur

(Spot/Cube Map) erzeugt und ausgelesen. Der erlaubte Abstand steht im Zusammenhang zur Voxelgröße. Der intuitive Worst Case ist eine Voxeldiagonale: Hierbei hat der Strahl ein volles Voxel an einer Ecke getroffen, der tatsächlich voxelisierte Punkt liegt jedoch an der gegenüberliegenden Ecke und ist ggf. auch in der Spot/Cube Map eingetragen. Um solche Fälle abzudecken, muss die Voxeldiagonale als Toleranzwert zugelassen sein.

Zusätzlich muss beachtet werden, dass die perspektivische Projektion der Maps die Geometrie verzerrt darstellt, also nahe an der Lichtquelle gelegene Objekte mehr Platz in der Textur einnehmen. Somit besteht auch ein Zusammenhang des erlaubten Abstands zur Pixelgröße der Texturen. Wie beim Shadow Mapping können Banding-Artefakte auftreten (Abbildung 23 links), insbesondere dort, wo sich bei der Map-Generierung Flächen fast parallel zur Projektionsrichtung befinden. Um diesen Fall zu behandeln, muss bei der Berechnung der Pixelgröße die Neigung des Pixels (Normale an der rasterisierten Position) berücksichtigt werden. Als weitere Eingabe ist dann eine Map mit Normalen erforderlich. Sämtliche Spot und Cube Maps werden wie beim G-Buffer-Rendering in einem Pass erzeugt. Abbildung 24 skizziert die Zusammenhänge und nennt die für die Berechnung des Toleranzwertes zugrundegelegten Formeln. Dieser wird auf ein Maximum begrenzt, um Light Bleeding zu verringern (Light Bleeding: vgl. Abbildung 23 rechts).

Eine letzte Bedingung bezieht sich auf die Normalen, analog zur Problematik, die im vorherigen Abschnitt geschildert wurde. Voxel, auf die ein Wert abgebildet wurde, der eigentlich zu einer dem Strahl abgewandten Oberfläche gehört, dürfen keinen Beitrag zur indirekten Leuchtdichte liefern.

6.3.4 Vergleich der beiden Speichermöglichkeiten

Speicherbedarf. Ein wesentlicher Unterschied zwischen Maps und Volumen liegt im Speicherbedarf. Eine 3D-Textur der Auflösung 128^3 mit 24 Bit pro Voxel (8 Bit pro Kanal) benötigt 6 MB – bei 16 Bit pro Kanal das Doppelte. Das 3D-Volumen muss an die Auflösung der binären Voxel angepasst sein, die oftmals auch mehr als 128^3 beträgt. Mit der Auflösung $256^2 \times 128$ steigt der Speicherbedarf auf 24 MB (24 Bit pro Voxel). Dagegen hat eine Spot Map der Auflösung 256^2 mit 16 Bit pro Kanal nur 0,375 MB und eine Cube Map mit 192^2 aufgelösten Würfelseiten $\approx 1{,}27$ MB. Dies sind Beispiele für typische in der Anwendung gewählte Auflösungen. Der benötigte Speicherplatz ist für die Maps-Variante linear abhängig von der Anzahl der verwendeten Lichtquellen, bei der Volumen-Variante davon unabhängig.

Geschwindigkeit der Erzeugung. Die binäre Voxeltextur wird für beide Varianten gesondert erstellt. Die Erzeugung der nicht-binären Volumentexturen dauert abhängig von den eingesetzten Modellen und der Auflösung mehrere Millisekunden (vgl. Abschnitt *9.2 Performance der Atlas-Voxelisierung*).

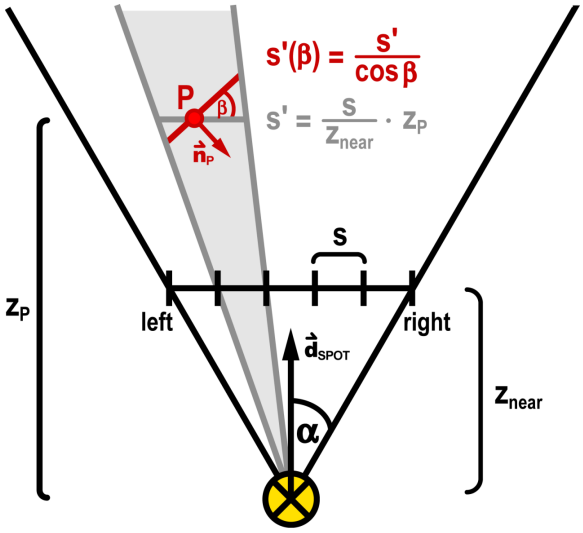

Abbildung 24: Berechnung des Abstandstoleranzwertes. Höhe und Breite des Lichtfrustums sind immer identisch und die Textur quadratisch, daher werden die Zusammenhänge in 2D gezeigt. Die Abbildung bezieht sich auf ein Spotlight. Alle Angaben sind direkt auf Punktlichtquellen mit 6 Frusta (Öffnungswinkel 90°, d. h. hier: $\alpha = 45°$) übertragbar. Die Länge einer Pixelseite beträgt $s = \textit{Frustumbreite}/\textit{Texturauflösung}$. Die Position P und die Normale $\vec{n}_P$ werden an der auf die Textur projizierten Koordinate ausgelesen. Die für den Abstandstoleranzwert benötigte Pixelgröße s' ist proportional zum Abstand z_P des Punktes P vom Spotlight. Diese Pixelgröße bezieht sich auf eine zur Near-Plane parallele Fläche. Die geneigte Fläche $s'(\beta)$ hängt von dem Winkel β zwischen Normale und Richtung des Lichtkegels ab. Als Abstandstoleranz wird die Diagonale benutzt, also $s'(\beta)\sqrt{2}$.

Das Atlas-Rendering mit Deferred Shading für die Berechnung der diffusen Leuchtdichten ist ebenso linear abhängig von der Anzahl der Lichtquellen wie die Generierung der Maps (eine Map pro Lichtquelle). Beispielhafte Zeiten für das Rendern der Spot Maps und Cube Maps sind in Abschnitt *9.3.1 Vergleich verschiedener Parametereinstellungen* genannt.

Lichtquellentypen. Die Map-Verfahren beruhen darauf, dass die von den Lichtquellen „gesehene“ Geometrie in Texturen abgebildet werden kann. Dies ist nur für punktförmige Lichtquellen möglich. Dagegen können im Volumen beliebige Lichtquellentypen zur Beleuchtung herangezogen werden, da die Beleuchtung auf den Atlastexturen stattfindet.

Geometrie. Die Leuchtdichte-Voxel können sehr dünne Objekte nur beschränkt erfassen. Dieses Problem besteht bei der Map-Variante höchstens bei den binären Voxeln, die für die Strahlverfolgung genutzt werden. Für die Beleuchtungsinformation gibt es keine solche Beschränkung. Einzelne

Flächen dürfen aus verschiedenen Richtungen beleuchtet werden und dünne Objekte können aus beiden Richtungen erfasst werden.

Werte auslesen. Das Auslesen der Leuchtdichte-Voxel ist unkompliziert, da keine Transformation der Schnittkoordinate in Texturkoordinaten und keine Abstandsberechnungen wie bei den Maps nötig sind. Gleichzeitig kann Letzteres nachteilig sein, da sich keine Abstandstoleranz einstellen lässt. Das Voxel, das ausgelesen wird, muss durch die Strahlverfolgung gefunden werden. Schnitttests, welche die Schnittposition nur annähern, sind weniger sinnvoll einsetzbar, da es passieren kann, dass aus leeren Voxeln gelesen wird.

Ein einziger Zugriff auf das Volumen reicht also aus, um die Leuchtdichte-Information zu erhalten. Für die Maps wurde ein Multi-Pass-Verfahren umgesetzt: ein Pass pro Map mit additivem Blending der Ergebnisse.

Material der Senderoberflächen. Das Verfahren mit den Leuchtdichte-Voxeln wurde nur für diffuse direkt beleuchtete Flächen umgesetzt. Das Volumen kann ausschließlich diffuse Sender indirekten Lichts erfassen, da nur ein einzelner Leuchtdichte-Wert pro Voxel gespeichert wird. Eine Erweiterung auf moderat glänzende Flächen mit Spherical Harmonics wäre hier denkbar. Es sind jedoch nicht-diffuse Empfänger des indirekten Lichts bei einem Bounce möglich, da das Reflexionsverhalten des Empfängers unabhängig von den Senderflächen ist.

Die Maps wurden erweitert, um auch sendende Oberflächen mit Phong-Material zu unterstützen (Abschnitt *6.4.3 Erweiterung für Phong-Material*).

Abschließende Bewertung. In dieser Arbeit wurde die Maps-Variante bevorzugt eingesetzt, da sie flexibler zu handhaben ist. Sie ist weniger speicherintensiv – insbesondere, wenn zusätzliche Informationen wie Normalen und Material herangezogen werden müssen – und lässt sich mit Verfahren zur näherungsweisen Schnittpunktermittlung kombinieren. Außerdem ist die Leuchtdichte-Voxelisierung bei großen bzw. komplexen Szenen zu rechenintensiv (vgl. Abschnitt *9.2 Performance der Atlas-Voxelisierung*, Diagramm in Abbildung 42, Seite 86).

6.4 Voxel-Strahlverfolgung

6.4.1 Single Bounce

Für jedes Pixel des Kamera-Viewports werden Strahlen, deren Ursprung die an dem Pixel eingetragene Weltkoordinate ist, mit dem zuvor beschriebenen Importance Sampling erzeugt. Die zugrundeliegenden (0,1)-verteilten Zahlen werden mit einer deterministischen 2D-Hammersley-Sequenz erzeugt. Zusätzlich ist es nötig, die Richtungsmenge pro Pixel zu variieren, um Aliasing-Artefakte zu vermeiden (siehe Abbildung 25). Die *Cranley-Patterson-Rotation*

[Pharr und Humphreys 2004, S. 384] ist eine Möglichkeit, die Richtungen zu modifizieren. Die ursprünglichen Zahlen X_i der Hammersley-Sequenz werden gemäß der Gleichung

$$X_i' = (X_i + \xi_i) \bmod 1$$

verändert. Anschließend wird die Richtung aus X_i' berechnet. Die Werte ξ_i liegen in einer kleinen gekachelten 2D-Zufallstextur mit zwei Kanälen vor, deren Werte im Intervall [0, 1] mit dem C++-Zufallszahlengenerator erzeugt werden. Je nach Größe der Zufallstextur und der eingetragenen Werte ist das Zufallsmuster auf dem Bild mehr oder weniger deutlich zu erkennen.

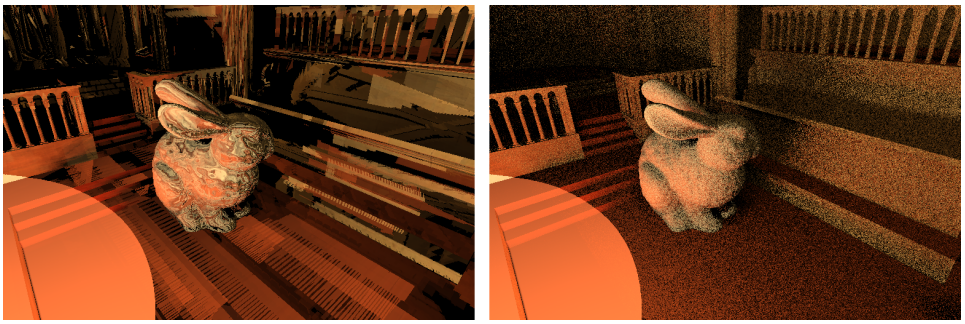

Abbildung 25: Indirektes Licht mit 1 Bounce: Variation der verfolgten Strahlen pro Pixel (rechts) vermeidet starke Aliasing-Artefakte (links). Die benutzte 2D-Zufallstextur im rechten Bild hat die Auflösung 64×64. Bildauflösung: 768×512.

Die Reichweite der Strahlen wird auf einen benutzerdefinierten Radius beschränkt. Sie ist zusätzlich durch die Abmessung des voxelisierten Bereichs begrenzt. Strahlen, die den Voxelraum verlassen, ehe sie etwas geschnitten haben, erhalten das Ergebnis „kein Treffer mit der Voxelszene“.

Das Hemisphären-Sampling wurde als Multi-Pass-Verfahren umgesetzt. Die Aufteilung in mehrere Passes – anstelle eines Passes mit einer Schleife im Shader über alle Strahlen – erwies sich als effizienter, insbesondere wenn viele Strahlen verfolgt werden.

Ein Pass ermittelt pro Pixel die aus einer Richtung einfallende Leuchtdichte und setzt sich aus folgenden Schritten zusammen: Der erste Shader verfolgt einen Strahl pro Pixel und schreibt im Fall eines Schnittes mit der Voxeltextur die Weltkoordinate des ersten Auftreffpunktes in eine Textur. Pixel, für die kein Schnitt gefunden werden konnte, werden für den weiteren Verlauf mit einem speziellen Wert gekennzeichnet. Diese Zwischenspeicherung der Schnittpunkte ist notwendig, da das Auslesen der Spot und Cube Maps ebenfalls als Multi-Pass-Verfahren konzipiert wurde. Die Ergebnis-Textur mit den Schnittpunkt-Weltkoordinaten dient als Eingabe für den zweiten Shader, der die Leuchtdichte aus Spot und Cube Maps oder aus dem Volumen ausliest.

Pixel, die den Wert für „kein Treffer“ enthalten, werden dabei auf Schwarz gesetzt. Additives Blending akkumuliert die Einzelergebnisse in einer Textur, die am Ende die indirekte Beleuchtung pro Pixel enthält.

Der Schnitttest aus Kapitel 5 oder alternativ uniformes Ray Marching findet den ersten Schnittpunkt. Das uniforme Ray Marching erzeugt äquidistante Abtastpunkte auf dem Strahl. Es wird jeweils das Voxel getestet, innerhalb dessen sich die Abtastposition auf dem Strahl befindet. Dafür wird ein entsprechendes Bitmuster aus einer kleinen 1D-Textur gelesen und mit der zugehörigen Voxelspalte aus der Voxeltextur verundet. [Nichols et al. 2010] benutzen uniformes Ray Marching für die Sichtbarkeitsermittlung von Flächenlichtquellen, die durch Punktlichtquellen angenähert sind. Sie schlagen vor, den Voxelstrahl „dicker“ zu machen, indem mehr Voxel gleichzeitig auf einen Schnitt untersucht werden. Das ausgelesene Bitmuster hat dann um die angefragte Position herum mehr als nur ein gesetztes Bit. Der Strahl wird damit allerdings nur in einer Dimension breiter, nämlich in Voxelisierungsrichtung. Dadurch werden potentiell mehr Schnitte näherungsweise erkannt, allerdings auch einige falsche, wenn z. B. ein Strahl nah zu einer Voxeloberfläche verläuft. Die Variante wurde im Laufe dieser Arbeit getestet. Abbildung 26 veranschaulicht die beiden Ray-Marching-Varianten. Abschnitt *9.3.1 Vergleich verschiedener Parametereinstellungen* zeigt Ergebnisbilder.

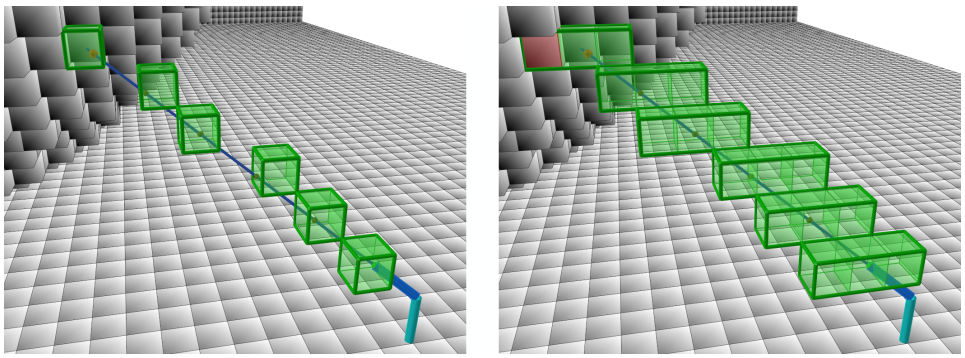

Abbildung 26: Ray-Marching-Varianten: Links: Der Strahl wird mit einzelnen Voxeln abgetastet. Rechts: Um die Abtastposition herum werden 3 Voxel geprüft (in Voxelisierungsrichtung, d. h. innerhalb der Voxelspalte, in der sich die Abtastposition befindet). Die Abtastrate ist in beiden Bildern identisch. In dem gewählten Beispiel erkennt die 3-Bits-Variante einen Schnitt (näherungsweise, an der falschen Stelle, aber nicht fälschlicherweise), den die 1-Bit-Variante aufgrund einer zu großen Schrittweite verpasst.

6.4.2 Multiple Bounces (Voxel Path Tracer)

Um mehr als eine Interreflexion zu berechnen, werden Pfade durch die Szene verfolgt. Bei nur einem Bounce wird das indirekte Licht an den Trefferpositio-

nen ignoriert (siehe Gleichung (5), Seite 46). Damit aber auch das tatsächlich vorhandene indirekte Licht an der getroffenen Oberfläche berücksichtigt wird, muss es abgeschätzt werden.

Dazu wurde während der Diplomarbeit ein Voxel Path Tracer, der ausschließlich das indirekte Licht bestimmt, umgesetzt. Dieser verfolgt einen Pfad von Strahlen in der Szene. Er wurde zunächst für rein diffuse Szenen umgesetzt und anschließend auf Phong-Materialien erweitert. Der nachfolgende Abschnitt *6.4.3 Erweiterung für Phong-Material* erläutert die dafür nötigen Änderungen.

Im Gegensatz zum klassischen Path Tracing starten die Pfade an den von der Kamera aus gesehenen Oberflächenpunkten, nicht an der Kameraposition. Nach dem ersten Auftreffpunkt des ersten Strahls wird ein weiterer von den Eigenschaften des Auftreffpunkts abhängiger Strahl verschossen, um die indirekte Leuchtdichte am Auftreffpunkt zu schätzen. Anschließend wird von dem nächsten Auftreffpunkt aus wieder ein Strahl verfolgt, und so weiter.

Das Verfahren ist ebenso wie jenes für einen einzigen Bounce als Multi-Pass-Technik ausgelegt. Der Path Tracer bricht ab, wenn eine definierte maximale Pfadlänge erreicht ist. Diese entspricht der Anzahl der Interreflexionen. Bei mehreren Bounces müssen an jeder Trefferposition entlang des verfolgten Pfades das Material und die Normale an der Stelle bekannt sein. Die Normale bestimmt die Hemisphäre, in der der nächste Strahl verfolgt wird. Das Material wird zur Berechnung der von der aktuellen Trefferposition zur vorherigen Trefferposition ausgehenden Leuchtdichte benötigt. Somit müssen an den Schnittpunkt-Weltkoordinaten die Normale und das Material abrufbar sein. Da die Schnittpunkte überall in der Szene liegen können, müssen diese Informationen aus 3D-Texturen gelesen werden, deren Voxel sich mit den binären Voxeln decken.

Der Algorithmus profitiert insbesondere von dem in Kapitel *5 Strahlschnitttest mit binärer Voxelhierarchie* geschilderten Schnitttest. Es kann passieren, dass Voxeloberflächen bei der äquidistanten Strahlverfolgung übersprungen werden, sodass der Strahl fälschlicherweise dahinter gelegene Flächen trifft oder das Voxelisierungsfrustum verlässt. Durch das Verfehlen von Oberflächen geht viel Licht verloren, da der Pfad dann nicht weiter verfolgt werden kann. Uniformes Ray Marching würde daher eine sehr feine Abtastung erfordern und entsprechend rechenintensiv sein. Der Schnitttest liefert im Vergleich zu Ray Marching mit sehr feiner Abtastung ein schnelleres und ebenfalls genaues Ergebnis.

Die entlang eines Pfades an den Auftreffpositionen bestimmten Informationen (direkte Leuchtdichte in Ausfallsrichtung, Material) werden in Texturen gespeichert und rückwärts vom letzten Schnittpunkt zum ersten hin miteinander kombiniert.

Für die Strahlerzeugung aus (0,1)-verteilten Zahlen wird auf die Halton-Sequenz und auf Zufallszahlen des C++-Pseudozufallgenerators zurückgegriffen. Die Strahlen des ersten Bounces werden mit der 2D-Halton-Sequenz $(\Phi(i)_2, \Phi(i)_3)$ konstruiert, die des zweiten Bounces mit $(\Phi(i)_5, \Phi(i)_7)$ und die der nächsten Bounces mit Zufallszahlen. Theoretisch wäre für einen Pfad der Länge n eine $2n$-dimensionale Sequenz erforderlich. Da in hohen Dimensionen die Verteilung der Elementpaare (z. B. bei Dimension 11 und 13) ungünstig ist, müsste die Halton-Sequenz modifiziert werden. Die Algorithmen zur Modifikation heißen z. B. Scrambling oder Randomization, wurden jedoch im Rahmen der Diplomarbeit nicht umgesetzt. Der Kompromiss aus Halton-Sequenz für die ersten beiden Bounces und Zufallszahlen für die folgenden Bounces erzeugte zumindest im Vergleich zu einem komplett auf Zufallszahlen basierenden Sampling Bilder mit geringerem Rauschen.

Damit der Nutzer trotz des hohen Rechenaufwandes mit der Szene interagieren kann, oder um den Verlauf der Berechnung zu beobachten, wurde ein progressives Verfahren implementiert. Die Monte-Carlo-Integration arbeitet bereits progressiv, da der Fehler sinkt, wenn die Anzahl der Stichproben erhöht wird. Es müssen daher pro Frame andere Samples (Pfade pro Pixel) erzeugt und die Ergebnisbilder akkumuliert und gemittelt werden. Die Akkumulation startet, sobald die Interaktion beendet ist.

6.4.3 Erweiterung für Phong-Material

Um die Verfahren von diffusen auf Phong-Oberflächen zu erweitern, sind zwei Anpassungen nötig:

- Für Licht empfangende Phong-Oberflächen: Die Strahlen zur Abtastung der Hemisphäre müssen gemäß der Phong-Dichte erzeugt werden.
- Für Licht sendende Phong-Oberflächen: Das Auslesen der direkten Leuchtdichte geschieht abhängig von der Strahlrichtung und der Phong-Keule der getroffenen Oberfläche.

Der erste Punkt wird in Abschnitt *6.2.2 Importance Sampling* behandelt. Die zweite Anpassung erfordert Wissen über den an der Oberfläche reflektierten Vektor von der primären Lichtquelle zum Oberflächenpunkt. Bei mehreren Lichtquellen in der Szene gibt es folglich mehrere Reflexionsrichtungen. Die Spot und Cube Maps sind geeignet, diesen lichtquellenspezifischen Vektor in einer zusätzlichen Textur abzuspeichern, da sie die Informationen für jede Lichtquelle getrennt speichern. Anhand des Reflexionsvektors und der Materialeigenschaften kann berechnet werden, welchen direkten Leuchtdichte-Anteil die Oberfläche aufgrund der aktuell betrachteten Lichtquelle in die Ausfallsrichtung abgibt.

7 Indirektes Licht mit virtuellen Punktlichtquellen (VPLs)

7.1 Überblick

Das Hemisphären-Sampling muss erst die Richtungen ermitteln, aus denen Licht auf die betrachtete Oberfläche fällt. Dagegen wird bei Ansätzen mit virtuellen Punktlichtquellen eine Menge ausgewählter beleuchteter Punkte herangezogen. Die Integration über die Hemisphäre entfällt, stattdessen ist die indirekte Leuchtdichte eine Summe über alle Beleuchtungsbeiträge der VPLs. Ein solches Verfahren mit virtuellen Punktlichtquellen, das eine binäre Voxelszene als Grundlage hat, wird in diesem Kapitel geschildert.

Abschnitt *3.3 Beleuchtung mit virtuellen Punktlichtquellen* in Kapitel *3 Verwandte Arbeiten* bietet eine knappe Übersicht über existierende Algorithmen, die mit virtuellen Punktlichtquellen arbeiten.

[Ritschel et al. 2008] weisen auf den Vorteil dieser Vorgehensweise hin: Die Sichtbarkeit muss nur noch zwischen wenigen VPLs und den Szenenpunkten ausgewertet werden anstatt zwischen sehr vielen beliebigen Punktepaaren. Die VPL-Verfahren skalieren linear mit der Anzahl der verwendeten VPLs. Ein schneller Sichtbarkeitstest ist daher ein essenzieller Bestandteil. Die Sichtbarkeit zwischen VPLs und den zu beleuchtenden Punkten der Szene wird mit dem in Kapitel 5 vorgestellten Voxel-Schnitttest ermittelt.

Die wesentlichen Schritte des in diesem Kapitel erläuterten Verfahrens sind:

- Atlas-Rendering und binäre Voxelisierung der Szene,
- Erzeugen der VPLs für den ersten und ggf. zweiten Bounce:
  - Spot Map und Cube Map Rendering,
  - Auslesen der VPL-Daten an Sample-Positionen,
- G-Buffer Rendering und ggf. Umordnung der Texturen für Interleaved Sampling,
- jedes Pixel des G-Buffers mit einer bestimmten Menge an VPLs beleuchten (inklusive Sichtbarkeitstest auf binäre Voxelszene),
- Filtern des Ergebnisbildes.

7.2 Erzeugung der VPLs

7.2.1 Erster Bounce

Diese Arbeit folgt dem Ansatz von *„Reflective Shadow Maps“*, um die virtuellen Punktlichtquellen für den ersten Bounce zu ermitteln. Spot Maps für Spotlights und Cube Maps für Punktlichtquellen stellen die Ausgangsbasis für eine ausgewählte Menge an VPLs dar. Mit einer zweidimensionalen Halton-Sequenz werden Texturkoordinaten für das Auslesen der VPL-Texel generiert. Die Spot Maps benötigen Samples innerhalb einer Kreisfläche; die Cube Maps brauchen Richtungsvektoren, da auf sie über dreidimensionale Texturkoordinaten zugegriffen wird. Die Abbildungen 27 und 28 zeigen Beispiele für Samplepositionen und -vektoren in den Texturen und die entsprechende Platzierung der VPLs in der Szene.

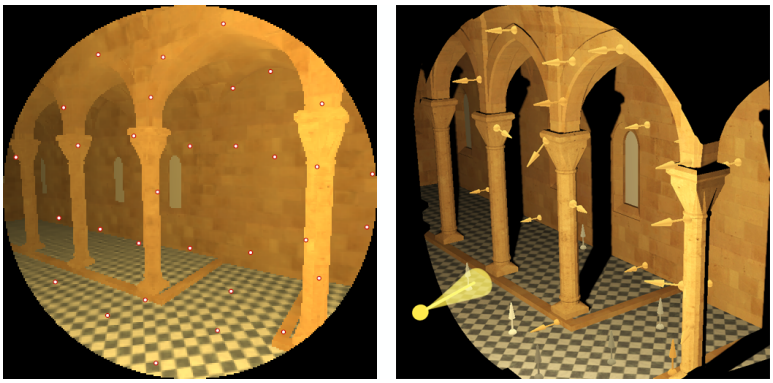

Abbildung 27: Auf die Spot Map wird mit 2D-Texturkoordinaten innerhalb des Lichtkegelkreises zugegriffen (links). Die VPLs in der Szene sind mit ihren Normalen rechts dargestellt.

Die Daten der durch die Samples ausgewählten VPLs werden aus den Maps extrahiert und in eine kleinere Textur eingefügt. Die Verwendung dieser kompakteren Textur anstatt der Original-Maps kann im weiteren Renderingverlauf vorteilhaft für die Performance sein, falls die Daten im Fragment-Shader ausgelesen werden. In dieser Arbeit wurde jedoch ein Ansatz gewählt, bei dem die Daten bereits im Vertex-Shader ausgelesen und an den Fragment-Shader weitergereicht werden. Hier zeigen sich keine Geschwindigkeitsunterschiede, allerdings ist durch die kompakten Texturen eine einheitliche Implementierung des Beleuchtungsshaders möglich. Zudem baut die im nächsten Abschnitt geschilderte Methode zur Ermittlung der VPLs für einen zweiten Bounce auf der kompakten Textur auf.

Wird an Stellen aus den Maps gelesen, die keine gültigen Daten enthalten, weil dorthin keine Geometrie rasterisiert wurde, werden die VPLs dennoch eingetragen. Dies ist dadurch bedingt, dass die kompakte Textur bereits im

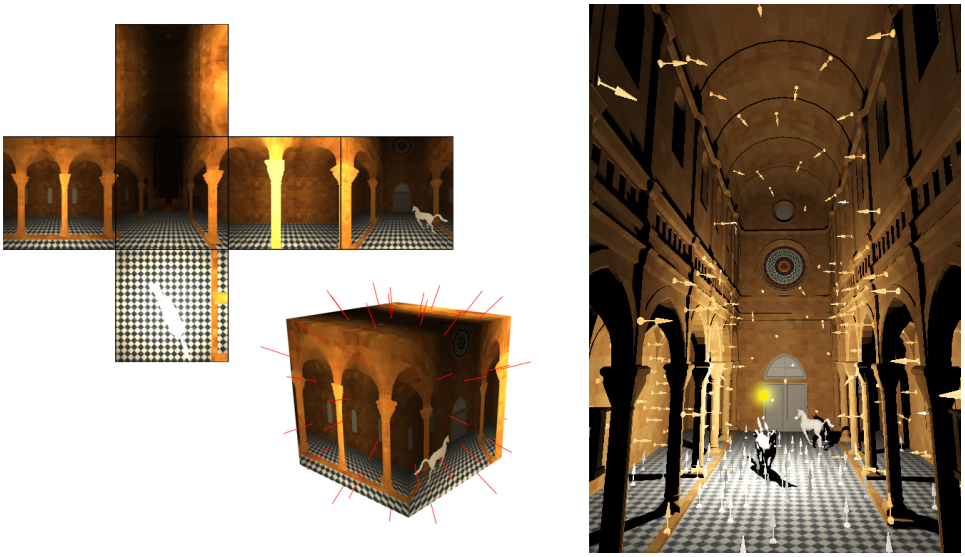

Abbildung 28: Auf die Cube Map (links aufgefaltet) wird mit 3D-Texturkoordinaten zugegriffen (Mitte: rote Vektoren in der als Würfel dargestellten Cube Map), um die VPL-Daten zu erhalten. Mit der in dieser Arbeit gewählten Sampling-Strategie werden mehr VPLs nahe der Lichtquelle platziert, da diese Regionen mehr Platz in der Cube Map einnehmen (rechts).

Voraus angelegt und jedem Pixel eine Sample-Texturkoordinate zugewiesen wurde. Eine dynamische Änderung der Texturgröße und das Verwerfen ungültiger Samples wäre an dieser Stelle weniger performant, als die ungültigen VPLs später gesondert zu behandeln.

7.2.2 Zweiter Bounce

Um das indirekte Licht von zwei Interreflexionen simulieren zu können, müssen weitere VPLs in der Szene verteilt werden. In dieser Arbeit wurde dazu ein Ansatz gewählt, der von den VPLs für den ersten Bounce ausgeht und konzeptionell dem Verfahren für Multiple Bounces im vorherigen Kapitel entspricht (*6.4.2 Multiple Bounces (Voxel Path Tracer)*). Daher kann das Verfahren auf mehr als zwei Bounces erweitert werden. Von jedem Pixel der kompakten VPL-Textur aus wird jeweils ein Voxelstrahl verfolgt. Die Daten am Auftreffpunkt werden für die Second-Bounce-VPLs eingetragen. Der Lichtstrom wird mit dem getroffenen Material skaliert. Die Anzahl der mit dieser Methode gefundenen VPLs für den zweiten Bounce entspricht maximal der Anzahl der VPLs für den ersten Bounce.

7.3 Berechnung der indirekten Beleuchtung mit VPLs

7.3.1 Eingabedaten

Die Angaben in den folgenden Abschnitten beziehen sich zunächst wieder nur diffuse Materialien, die Erweiterung auf Phong folgt in *7.3.4 Phong-VPLs und Phong-Empfängerflächen*.

Die wichtigsten Eingabedaten für den Algorithmus mit einem Bounce sind:

- eine binäre Voxeltextur mit Mipmaps für den Schnitttest,
- die G-Buffer-Texturen,
- Spot Maps und Cube Maps mit Weltkoordinaten, Normalen und Material-Information,
- kompakte VPL-Texturen (bekannt ist, welche VPLs von welchen primären Lichtquellen stammen),
- für die Berechnung des Lichtstroms bzw. Lichtstärke einer VPL: Lichtstärke und Raumwinkel der primären Lichtquellen.

Φ_{light}	$=$	$I_{light} \cdot \Omega_{light}$	Lichtstrom der primären Lichtquelle
Φ_{VPL}	$=$	$\frac{\Phi_{light} \cdot \rho_{VPL}}{N_{VPL}}$	Lichtstrom einer VPL
I_{VPL_max}	$=$	$\frac{\Phi_{VPL}}{\pi}$	maximale Lichtstärke einer VPL (Lambert-Strahler)
$I_{VPL}(\vec{\omega})$	$=$	$I_{VPL\_max} \cdot \cos\theta_{VPL}$	Lichtstärke einer VPL in Richtung $\vec{\omega}$

Der Raumwinkel Ω_{light} beträgt für Punktlichtquellen 4π, für Spotlichtquellen $2\pi(1.0 - \cos\alpha)$, wobei α der halbe Öffnungswinkel des Lichtkegels ist (vgl. Abbildung 24). ρ_{VPL} ist der Reflexionsgrad an der Position der VPL.

Für zwei Bounces ist zusätzlich eine nicht-binäre Voxelisierung mit Normalen und Material erforderlich, um die VPLs für den zweiten Bounce zu ermitteln.

7.3.2 Interleaved Sampling

Im Kontext der Beleuchtung mit virtuellen Punktlichtquellen bezeichnet Interleaved Sampling das Prinzip, für verschiedene Pixel-Gruppen jeweils verschiedene Mengen von VPLs für die Beleuchtung zu benutzen. Benachbarte G-Buffer-Pixel werden dann mit jeweils anderen VPLs beleuchtet. Diese Vorgehensweise vermindert Aliasing-Artefakte, führt stattdessen aber zu sich

wiederholenden Rausch-Mustern (siehe Abbildung 29). Letztere können durch einen geometrie-sensitiven Filter weitestgehend eliminiert werden. Generell

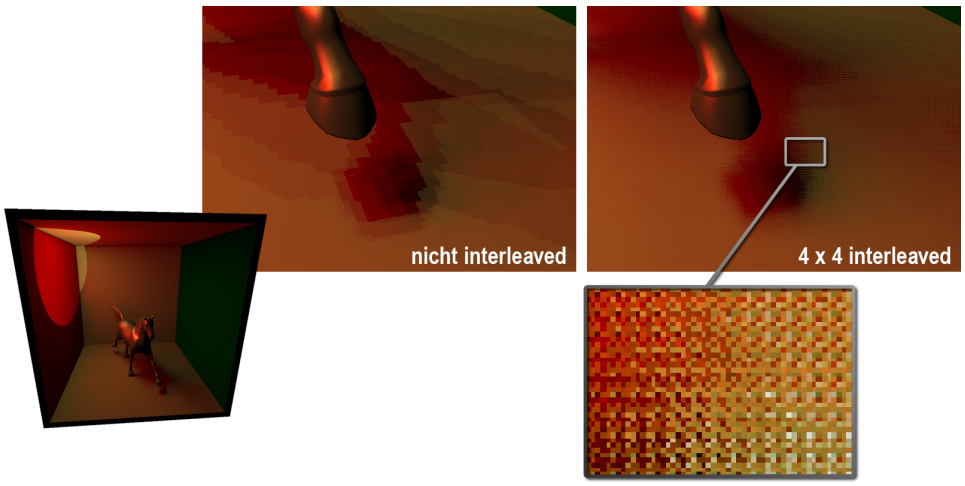

Abbildung 29: Vergleich der Bildqualität mit Interleaved Sampling und ohne Interleaved Sampling. Beide Bilder wurden mit jeweils 24 VPLs pro Pixel erzeugt. Im Bild links wurden die gleichen VPLs für jedes Pixel verwendet. Dagegen wurden im Bild rechts mit einem 4×4 Interleaved Sampling insgesamt 16 VPL-Mengen mit jeweils 24 VPLs verwendet. Der vergrößerte Ausschnitt zeigt das typische Rauschmuster.

steigt die Bildqualität durch Interleaved Sampling bei leicht sinkender Performance (abhängig vom gewählten Sampling-Muster, sehr große Muster beeinträchtigen die Geschwindigkeit stark). Ein Zahlenbeispiel zeigt: Werden für jedes Pixel 24 VPLs ausgewertet, wobei 16 verschiedene VPL-Mengen bei einem Sampling-Muster von 4×4 herangezogen werden, so werden effektiv $24 \cdot 16 = 384$ verschiedene VPLs ausgewertet.

[Segovia et al. 2006] beschreiben eine Methode, Interleaved Sampling für Deferred Shading effizient auf der Grafikkarte zu implementieren und dabei Cache-Kohärenzen auszunutzen. Hierfür werden die G-Buffer-Texturen entsprechend des angestrebten Sampling-Musters umgeordnet. Abbildung 30 veranschaulicht den Prozess, der *G-Buffer-Splitting* genannt wird. Zu beachten ist, dass die Auflösung der G-Buffer-Texturen ein Vielfaches des Sampling-Musters sein muss. Das Ergebnis sind „Split“-G-Buffer-Texturen mit Pixel-Blöcken, deren Anzahl der gewünschten Anzahl der VPL-Mengen entspricht. Jeder Pixel-Block wird dann mit der ihm zugewiesenen VPL-Menge beleuchtet. Für alle Pixel in einem Block werden also dieselben VPLs benutzt. Dieses Vorgehen ist performanter, als auf den originalen G-Buffer-Texturen für jedes Pixel die jeweilige VPL auszuwählen.

Nach der Beleuchtung wird eine inverse Split-Operation ausgeführt, um die Pixel wieder an ihren ursprünglichen Platz zu verschieben.

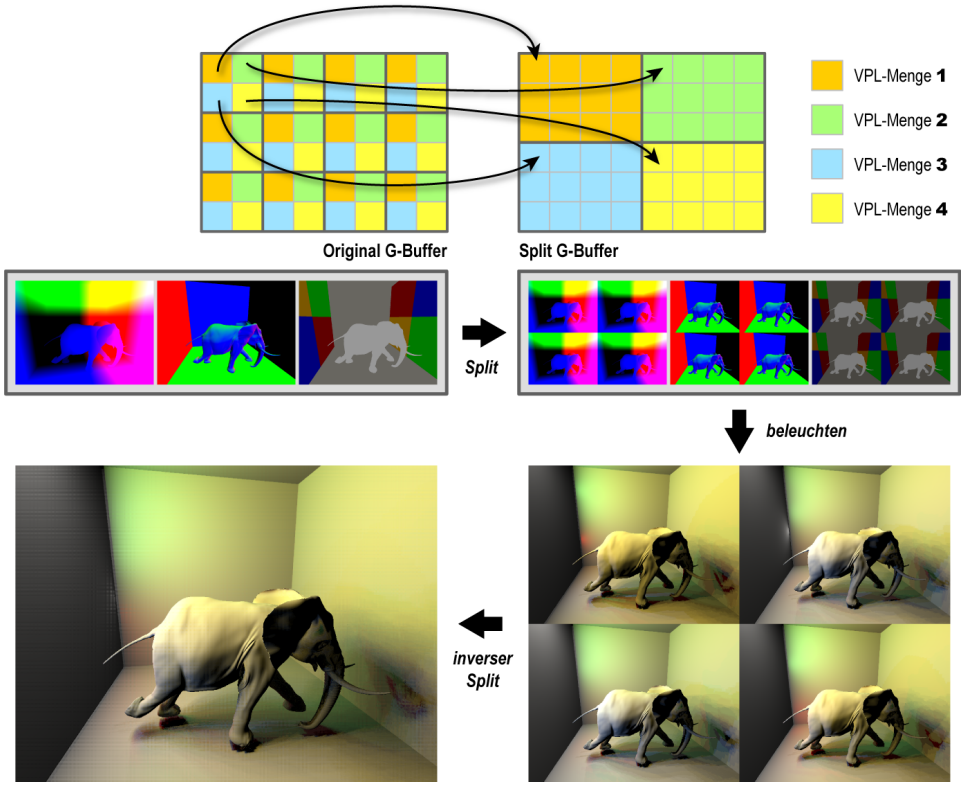

Abbildung 30: G-Buffer-Splitting für Interleaved Sampling. Dargestellt ist die Split-Operation für ein 2×2-Muster, also 4 VPL-Mengen. Die G-Buffer-Pixel jeder VPL-Menge werden so angeordnet, dass sie nicht mehr über die gesamte Textur verteilt sind, sondern sich kompakt in einem Rechteck befinden. Anschließend wird die Beleuchtung auf den einzelnen VPL-Mengen-Bereichen ausgeführt. So entstand das Bild unten rechts (indirekte Beleuchtungsstärke). Jenes muss anschließend mit der inversen Split-Operation wieder in die ursprüngliche Pixel-Anordnung überführt werden (unten links).

7.3.3 Ablauf

Wie das in Kapitel 6 vorgestellte Verfahren ist auch dieses als Multi-Pass-Verfahren konzipiert. Die Anzahl der Passes entspricht der Anzahl der VPLs. Um die Beleuchtung einer VPL für jedes (Split-)G-Buffer-Pixel auszuwerten, wird ein bildschirmfüllendes Rechteck gerendert. In der Literatur wird diese Vorgehensweise oft „Splatting“ genannt. Als „Gathering“ wird hingehen ein One-Pass-Verfahren bezeichnet, in dem der Fragment-Shader eine Schleife über alle VPLs enthält. Da aber der Shader zur Beleuchtung der VPLs auch den Schnitttest zwecks Sichtbarkeitsbestimmung enthält, erwies sich die Aufspaltung in mehrere Passes als wesentlich effizienter.

Für Interleaved Sampling werden pro Pass mehrere Rechtecke gerendert – eines für jeden Pixel-Block des Split-G-Buffers –, wobei der Viewport jeweils auf den aktuellen Pixel-Block gesetzt wird. Der Vertex-Shader erhält den Index der aktuellen VPL als Eingabe und liest die entsprechenden VPL-Daten aus den Eingabetexturen. Dabei kann sich herausstellen, dass es sich um eine ungültige VPL handelt. Der Vertex-Shader verwirft alle ungültigen VPLs, indem er die Vertices des gerenderten Rechtecks auf eine Position setzt, die beim Clipping in der Rendering-Pipeline verworfen wird. Gültige VPL-Daten werden an den Fragment-Shader weitergereicht. In diesem geschieht die Beleuchtung und der Sichtbarkeitstest mit dem Schnitttest aus Kapitel 5.

Der Beitrag einer VPL zur indirekten Beleuchtungsstärke eines Pixel ist:

$$E = I_{VPL\_max} \cdot \frac{\cos\theta_{VPL} \cdot \cos\theta_P}{d^2} \tag{6}$$

Abbildung 31 skizziert die Formelzeichen.

Das Teilen durch den quadratischen Abstand von VPL zu Pixelposition bewirkt einen für VPL-Methoden typischen Effekt: helle Bereiche im Bild um die VPL-Position herum, die je nach Platzierung der VPL mehr oder weniger stark zu erkennen sind. Die klassische Lösung[17], um das Problem zu verringern, ist das Clampen von d^2 auf einen minimalen Wert: $\max(d_{min}, d^2)$. Alternativ kann auch ein minimaler Wert addiert werden: $d_{min} + d^2$. In dieser Arbeit wurde die zweite Variante gewählt.

Alle Beiträge werden durch additives Blending akkumuliert. Zusätzlich erhöht sich eine Zählvariable für jede gültige VPL, sodass am Ende bekannt ist, wie viele gültige VPLs tatsächlich für die Beleuchtung benutzt wurden. Diese Anzahl kann von der Anzahl der initial aus den Spot Maps und den Cube Maps extrahierten VPLs abweichen. Eine abschließende Prüfung und ggf. Korrektur stellt für jedes Pixel sicher, dass die akkumulierte Beleuchtung richtig skaliert ist.

[17] eher Notbehelf

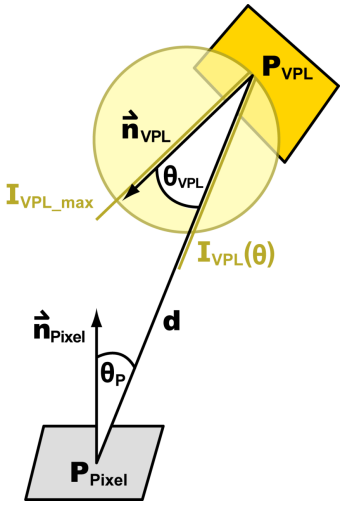

Abbildung 31: Konstellation von VPL und Pixel (beides infinitesimal kleine Flächen), siehe Gleichung (6).

7.3.4 Phong-VPLs und Phong-Empfängerflächen

Sollen Oberflächen mit Phong-Material als Sender zugelassen werden, ändert sich die Berechnung der VPL-Lichtstärke wie folgt [Dachsbacher und Stamminger 2006]:

$$I_{VPL}(\vec{\omega}) = \Phi_{VPL} \cdot \cos^{n_s} \psi_s$$

n_s ist der Phong-Exponent der Phong-VPL. ψ_s ist der Winkel zwischen Richtung $\vec{\omega}$ (Verbindungsvektor von VPL zu Pixelposition) und der Richtung der Phong-Keule.

Für eine Phong-Empfängerfläche mit Phong-Exponent n_e muss der Anteil des Lichts berechnet werden, der von der VPL in das Auge reflektiert wird:

$$L_{indirect} = \rho_s \frac{n+2}{2\pi} \cos^{n_e} \psi_e \cdot E_{indirect}$$

ψ_e ist analog zu ψ_s der Winkel zwischen Verbindungsvektor und der Hauptrichtung der Phong-Keule auf der Empfängerseite.

Grundsätzlich ist zu beachten, dass bei hohen Phong-Exponenten auf der Senderseite, also stark glänzenden, Licht reflektierenden Materialien, nur sehr viele virtuelle Punktlichtquellen gute Ergebnisse liefern können. Andernfalls sind die einzelnen VPLs deutlich als vereinzelte helle Bereiche im Bild zu erkennen. Je mehr VPLs verwendet werden, desto größer ist die Rechendauer, sodass eher moderat glänzende Flächen verwendet werden sollten. Auf der Empfängerseite ist das Problem weniger ausgeprägt.

7.4 Kombination mit Multiresolution-Verfahren

7.4.1 Prinzip

Die meiste Renderingzeit benötigen die Fragment-Shader, die das indirekte Licht berechnen. Bei VPL-Ansätzen muss theoretisch für jedes Pixel jede VPL ausgewertet werden. Allerdings ähnelt sich die Beleuchtungssituation vieler benachbarter Pixel, insbesondere, wenn es sich um diffuse Flächen handelt. Die diffuse indirekte Beleuchtung ändert sich im Allgemeinen nur langsam von Oberflächenpunkt zu Oberflächenpunkt bzw. von Pixel zu Pixel.

Eine Reihe von Veröffentlichungen von Greg Nichols und Chris Wyman konzentriert sich auf diesen Aspekt: Bildregionen werden auf einer möglichst niedrigeren Auflösung berechnet, falls sie als „homogen“ genug eingestuft werden. So werden Pixelblöcke, die auf der höchsten Auflösung bis zu mehreren tausend Pixel umfassen können, auf einen Schlag ausgewertet.

Ihre Multiresolution-Technik stellen [Nichols und Wyman 2009] zunächst in *„Multiresolution Splatting for Indirect Illumination“* vor. Diese verbessern sie in *„Hierarchical Image-Space Radiosity for Interactive Global Illumination“* [Nichols et al. 2009], indem sie die Bestimmung der auszuwertenden Rechteckregionen vereinfachen und stärker parallelisieren. Beide Paper ignorieren allerdings indirekte Schatten. Das Kriterium, mit dem die Homogenität einer Region bewertet wird, hängt daher nur von Normalen- und Tiefendiskontinuitäten ab. Dem Thema Sichtbarkeit in Verbindung mit einem Multiresolution-Verfahren nähern sich [Nichols et al. 2010] in der aktuellsten Publikation mit *„Incremental Voxel Visibility“*, allerdings im Kontext von dynamischen Flächenlichtquellen wie Videoleinwänden.

7.4.2 Umsetzung

Das Prinzip, größere homogene Bereiche in einem Schritt auszuwerten statt jedes einzelne Pixel darin, und nur Stellen mit starker Beleuchtungsänderung feiner abzutasten, verspricht eine Performancesteigerung. Deshalb wurde mit diesem Ansatz gegen Ende dieser Arbeit experimentiert.

Normalen- und Tiefendiskontinuitäten. Zunächst wurden die Regionen nur anhand von Normalen- und Tiefendiskontinuitäten eingestuft. Diese werden in Diskontinuitäten-Mipmaps gespeichert (siehe Abbildung 32).

Die Textur, in welche die indirekte Beleuchtung eingetragen wird, wird ebenfalls durch Mipmaps bis zu einer gewissen Stufe erweitert. Eine typische Auflösung des gröbsten Mipmap-Levels liegt zwischen 32 und 64 Pixeln in einer Dimension. Im Original-Verfahren benutzen die Autoren eine große Textur, in der alle Texturen der verschiedenen Auflösungen gespeichert sind. Für die Erweiterung auf Visibilitätsdiskontinuitäten ist jedoch eine Trennung

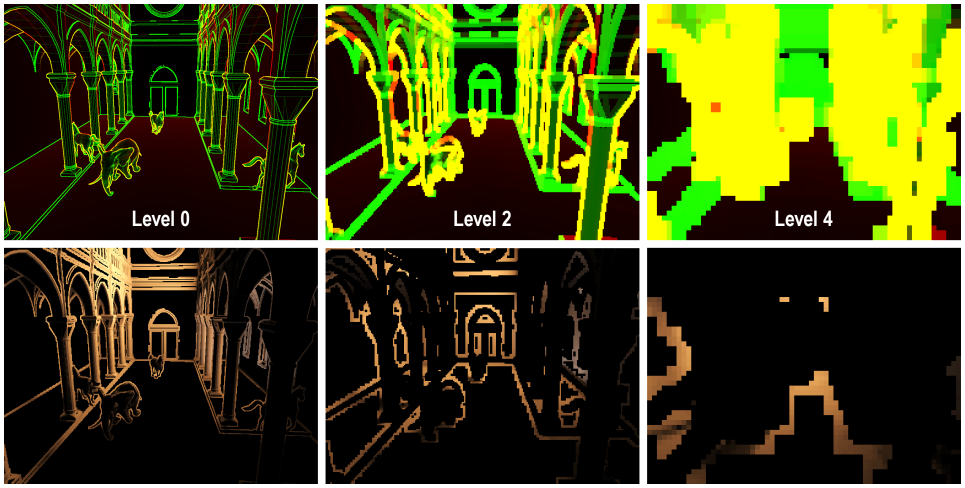

Abbildung 32: Diskontinuitäten-Texturen (oben) und Beleuchtungs-Texturen (unten). Die Levels 1, 3, 5 sind nicht dargestellt.

der Texturen für die Implementierung von Vorteil, sodass die Wahl auf Mipmaps fiel.

Die Diskontinuitäten werden initial auf der höchsten Auflösung in einer 3×3-Nachbarschaft um jedes Pixel herum erkannt. Die Normalendiskontinuitäten werden mit einer Formel zur Oberflächenkrümmung[18] lokalisiert. Die Tiefendiskontinuitäten basieren auf dem Differenzwert von minimaler und maximaler Tiefe in der Nachbarschaft. Ab welchen Werten eine Diskontinuität vorliegt, bestimmen manuell gesetzte Schwellenwerte. Die hoch aufgelöste Diskontinuitätentextur wird anschließend sukzessiv auf die niedrigeren Mipmap-Auflösungen heruntergerechnet.

Stencil-Rendering. Das Verfahren funktioniert durch das Setzen von Stencil-Bits in einem ebenfalls gemipmappten Stencil-Buffer. Pixel mit gesetztem Stencil-Bit werden von den VPLs beleuchtet. Insgesamt sind die durch die Stencil-Bits identifizierten Regionen disjunkt im höchstaufgelösten Bild. Die Mipmap-Texturen werden nach der Beleuchtung auf die höchste Auflösung skaliert und interpoliert. Welchen Bereich das Pixel einer Mipmap im finalen Bild einnimmt, hängt also von seiner Mipmap-Stufe ab. Abbildung 33 zeigt die disjunkt gesetzten Stencil-Bereiche für alle Mipmap-Stufen.

- Stencil-Bit auf einer Mipmap-Stufe *setzen* bedeutet: Beleuchtung soll auf dieser Stufe berechnet werden.
- Stencil-Bit auf einer Mipmap-Stufe *nicht setzen* bedeutet: Hier soll keine Beleuchtung berechnet werden, da sie entweder auf einer gröberen

[18] $2 \cdot \sin(0.5 \cdot \arccos(\vec{n}_0 \cdot \vec{n}_1))$ [Nichols 2010, S. 36]

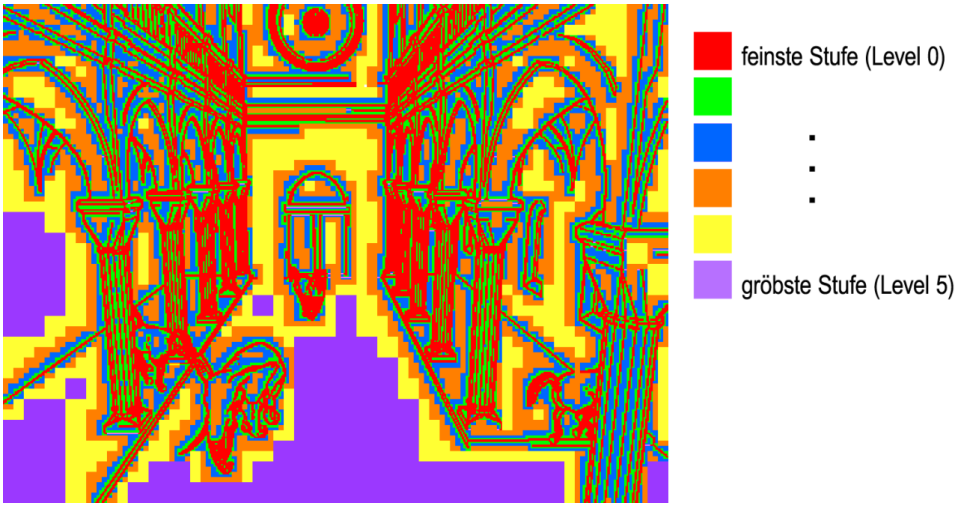

Abbildung 33: Pixel mit gesetzten Stencil-Bits auf allen Auflösungen. Auf der feinsten Stufe werden die hier rot dargestellten Pixel ausgewertet. Die niedrigeren Auflösungen sind auf die höchste Auflösung hochskaliert.

Stufe bereits berechnet wurde oder weil sie auf einer feineren Stufe berechnet werden muss.

Damit alle Pixel, für die kein Stencil-Bit gesetzt ist, bereits vor dem Fragment-Shader verworfen werden, muss die Grafikkarte Early-Stencil-Culling unterstützen. Bei NVIDIA-Grafikkarten ist dies seit der GeForce-6-Serie der Fall. Das frühe Verwerfen von Pixeln, für die der Stencil-Test fehlschlägt, erfordert, dass die Stencil-Werte nur gelesen, aber nicht geändert werden [Kilgariff und Fernando 2005, S. 482].

Interpolation. Der letzte Schritt des Verfahrens fügt die Ergebnisse der einzelnen Mipmap-Stufen zu dem finalen Bild zusammen. Die Mipmap-Texturen werden sukzessiv hochskaliert, kombiniert und manuell interpoliert (siehe Abbildung 34). Die Interpolationsfähigkeit der Grafikkarte kann nicht genutzt werden, da sichergestellt werden muss, dass nur gültige, beleuchtete Pixel in das Ergebnis einfließen.

Visibilitätsdiskontinuitäten. Bereiche, die Diskontinuitäten in der Sichtbarkeit der VPLs haben, sollen auf feineren Stufen gerendert werden. Dafür müssen nun pro Stufe zusätzlich zu Normale und Tiefe auch Visibilitätsdiskontinuitäten bekannt sein und beachtet werden. Der Ansatz hierfür stammt wie erwähnt von [Nichols et al. 2010] und nennt sich *„Incremental Voxel Visibility“*, da die Autoren ebenso wie diese Diplomarbeit eine binäre Voxelisierung[19] zur Ermittlung der Sichtbarkeit nutzen.

[19] Methode von [Eisemann und Décoret 2008]

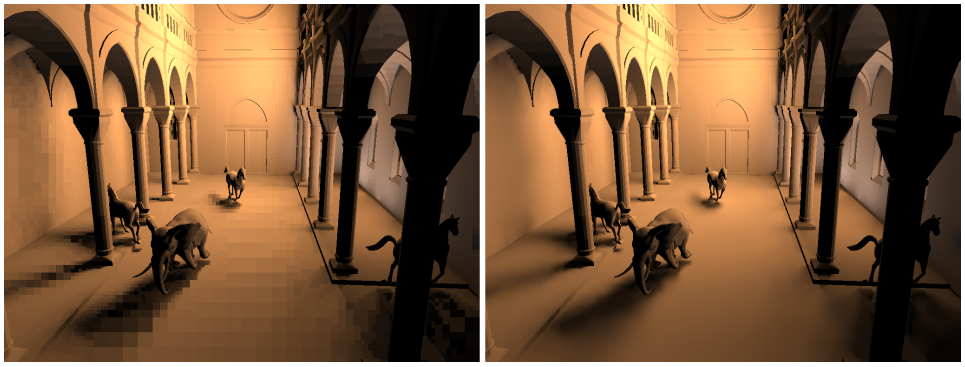

Abbildung 34: Indirekte Beleuchtungsstärke der Szene, für die bereits Diskontinuitätenbuffer und Stencil-Bits gezeigt wurden. Links nicht interpoliert, rechts interpoliert.

Der Ansatz basiert darauf, dass der Schattenwurf von Punktlichtquellen binär ist: „Schatten“ oder „kein Schatten“. In großen Teilen des Bildes ändert sich die Schatteninformation daher nicht, nur an den Übergangsstellen von „Schatten“ zu „kein Schatten“ haben benachbarte Pixel andere Schattenwerte. Bei den VPL-Methoden gibt es eine Vielzahl solcher Schattenwürfe. Die Autoren haben ihr Verfahren für durch VPLs angenäherte Flächenlichtquellen eingesetzt, bei denen die Kohärenz der Sichtbarkeitstests[20] generell höher ist als bei indirekter Beleuchtung. Für Spotlights jedoch, die eine Fläche in der Nähe anstrahlen, wird das angestrahlte Objekt (z. B. eine Wand) gewissermaßen zur Flächenlichtquelle. In diesen Fällen ist also eine ähnliche Ausgangssituation gegeben.

Für jedes Pixel muss nun geprüft werden, ob sich seine VPL-Sichtbarkeitsergebnisse von denen aus seiner Pixelnachbarschaft stark unterscheiden. In diesem Fall muss die Sichtbarkeit auf feineren Stufen nochmals berechnet werden. Hier unterscheidet sich das Verfahren zu jenem, das rein auf Normalen- und Tiefendiskontinuitäten basiert: Die zu beleuchtenden Regionen sind nicht mehr disjunkt, einige Bereiche müssen auf gröberen Stufen erst berechnet werden, um sie für die Diskontinuitätenbestimmung benutzen zu können. Stellt sich heraus, dass die grobe Stufe nicht ausreicht, müssen die Pixel erneut auf der feineren Stufe ausgewertet werden. Somit werden einige Pixel im Endeffekt doppelt ausgewertet. Die Annahme ist jedoch, dass trotzdem insgesamt weniger Pixel ausgewertet werden, als es Pixel auf der feinsten Stufe gibt.

Um die Sichtbarkeitsdiskontinuitäten zu ermitteln, schreiben [Nichols et al. 2010] die Sichtbarkeitsergebnisse als eine Reihe von Bits (z.B. „1“ = „nicht

[20] benachbarte Pixel haben mit einer hohen Wahrscheinlichkeit ähnliche Sichtbarkeitsergebnisse

sichtbar“ und „0“ = „sichtbar“) in Texturen. Es sind in einer Textur also maximal 128 Sichtbarkeitsergebnisse pro Pixel speicherbar.

In der Implementierung in dieser Diplomarbeit werden abwechselnd von der gröbsten bis zur feinsten Stufe Visibilitätsbits berechnet und Stencil-Bits gesetzt, da Letzteres von Ersterem abhängt. Parallel zu den Visibilitätsbits wird auch das Beleuchtungsergebnis in eine gemipmappte Textur geschrieben. Eine Visibilitätsdiskontinuität liegt vor, falls die Anzahl unterschiedlicher Visibilitätsergebnisse in der Nachbarschaft des aktuellen Pixels einen gewissen Schwellenwert überschreitet. Die Anzahl lässt sich durch eine XOR-Operation auf den ausgelesenen Texeln mit den Visibilitätsbits und anschließendes Zählen der gesetzten Bits ermitteln. [Nichols et al. 2010] nennen die Hälfte aller VPLs als geeigneten Schwellenwert, der hoch angesetzt ist und daher im Normalfall selten überschritten wird.

Betrachtung der Performance. Der theoretisch maximale Geschwindigkeitsgewinn durch den Einsatz der Multiresolution-Technik wurde durch Zählen aller tatsächlich berechneten Pixel ermittelt. Diese Anzahl wurde ins Verhältnis zur Anzahl aller Pixel der höchsten Auflösung gesetzt. Da das Verfahren sensitiv für die visuelle Komplexität ist, ändert sich die Anzahl der tatsächlich gerenderten Pixel abhängig davon, was die Kamera im Blickfeld hat. Der Anteil der während des Multiresolution-Verfahrens gerenderten Pixel zu allen Pixeln der höchsten Auflösung schwankte in Messungen zwischen circa 10 % und 40 %. Entsprechend wäre eine 2,5− bis 10-fache Geschwindigkeitssteigerung möglich.

Es ist nicht einfach, die um die Visibilität erweiterte Variante effizient zu implementieren, da die Abhängigkeiten der Schritte viele State-Changes wie Shader-Wechsel erfordern. Außerdem müssen an vielen Stellen Texturinhalte kopiert werden. Insbesondere die Kombination des Multi-Pass-Verfahrens mit dem Rendern der Visibilitätsbits stellte sich als schwierig heraus.

In der konkreten Umsetzung wurde die theoretisch erwartete Performancesteigerung nicht erreicht. Die zusätzliche Renderingzeit, die für die Schritte gebraucht wird, die das Multiresolution-Verfahren erst ermöglichen, überstieg im Allgemeinen die gewonnene Zeit. Der Mehraufwand müsste stark reduziert werden. Aus Zeitgründen konnten derartige Optimierungen jedoch im Rahmen dieser Arbeit nicht mehr umgesetzt werden. Dennoch erscheint der Ansatz im Hinblick auf den theoretisch möglichen Gewinn vielversprechend.

8 Implementierungsdetails

8.1 Allgemeine Informationen

Dieses Kapitel nennt Details der praktischen Umsetzung der Diplomarbeit, die noch nicht in den vorherigen Kapiteln erwähnt wurden. In diesen wurde bereits an vielen Stellen auf die Implementierung eingegangen, da die Konzepte und Entwicklung der Verfahren naturgemäß eng damit verbunden sind.

Das in Abbildung 35 gezeigte Programm wurde mit C++ unter Verwendung von OpenGL (Open Graphics Library) und GLSL (OpenGL Shading Language) implementiert. Für die Benutzeroberfläche wurde die Bibliothek Qt[21] in Kombination mit SFML (Simple and Fast Multimedia Library)[22] benutzt, um den OpenGL-Rendering-Kontext zu erstellen. Die Entscheidung für SFML fiel zu Beginn der Implementierungsphase, um die Möglichkeit offen zu halten, einen OpenGL 3.x Kontext erzeugen zu können. Die verwendete Qt-Version unterstützt dies nicht. Letztlich wurde jedoch die „alte“ OpenGL-Version 2.1 verwendet, um die Standard-Pipeline, die es ab Version 3 nicht mehr gibt, für einfache Debugging-Ausgaben nutzen zu können.

[21] Version 4.6.2, `http://qt.nokia.com` (12.08.2010)
[22] Version 2 aus dem SFML SVN-Repository, `http://www.sfml-dev.org` (12.08.2010)

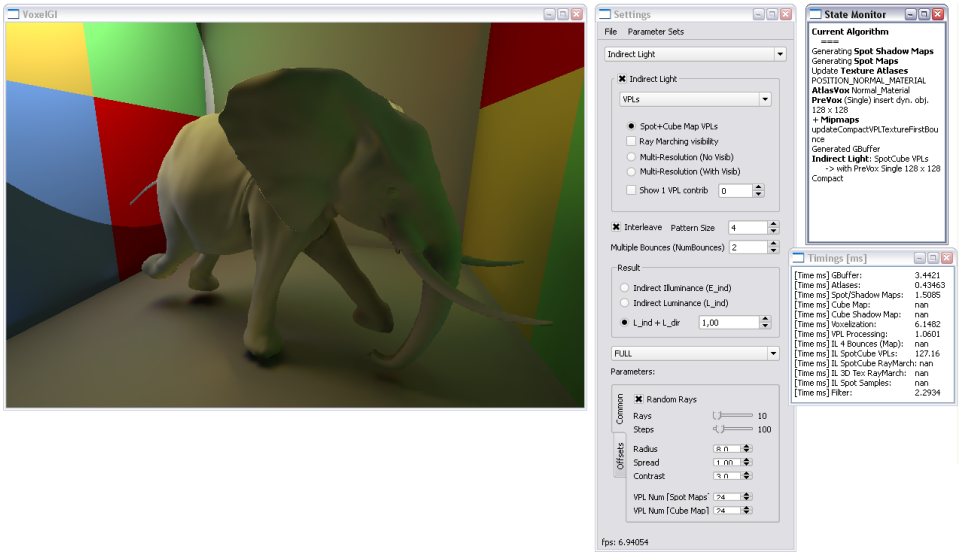

Abbildung 35: Benutzeroberfläche mit OpenGL-Fenster

Die Bibliothek OpenGL Mathematics Library (GLM)[23] erwies sich als besonders nützlich, da sie die GLSL-Datentypen (Vektoren, Matrizen), Funktionen und Operatoren als C++-Bibliothek zur Verfügung stellt. Somit lässt sich z. B. Shadercode einfach mit C++ nachprogrammieren und die Eingaben für Shader können vorbereitet werden.

Die in dieser Diplomarbeit entwickelte Anwendung umfasst alle Verfahren (Hemisphären-Sampling mit einem oder mehreren Bounces, Beleuchtung mit virtuellen Punktlichtquellen), damit zwischen diesen zur Laufzeit umgeschaltet werden kann. Zudem wurden Debugging-Visualisierungen implementiert, wie etwa die Anzeige der Voxel als Würfel, simples Ray Casting der Volumentexturen, die Verteilung der VPLs in der Szene und die Voxelstrahlverfolgung.

Die Benutzeroberfläche dient dazu,

- die verschiedenen Verfahren und Debugging-Ansichten anzuwählen,
- zahlreiche Parameter wie Strahlanzahl, VPL-Anzahl, Interleaved-Sampling-Muster, Tone Mapping und Filtereinstellungen zu justieren,
- die Szene zu gestalten:
  - *Kamera* bewegen, Kamerapose speichern und abrufen,
  - vom Benutzer manipulierbare *Modelle* verschieben und rotieren, Kopien eines Modells hinzufügen, Animation eines animierten Modells starten und beenden, Animationspfade modifizieren (Kontrollpunkte und Tangenten),
  - *Lichtquellen* positionieren, hinzufügen, Lichtstärke einstellen,
- die Szene zu speichern.

8.2 Laden und Speichern einer Szene

Die wesentlichen Bestandteile einer Szene sind statische und dynamische Modelle, Lichtquellen und Kameraposen. Zur Verwaltung mehrerer Szenen wurde eine Szenenbeschreibung als selbst definiertes XML-Format entworfen, das die Konfiguration der Szene beschreibt:

- gespeicherte Kameraposen
- Lichtquellen (Spot- und Punktlichtquellen) mit Position und Lichtstärke, bei Spotlights auch Öffnungswinkel und Ausrichtung des Lichtkegels
- statische Objekte der Szene, definiert durch
  - initiale Pose: Translation, Rotation, Skalierung
  - Dateipfad zur Modelldatei (Wavefront OBJ) und zur OBJ-Datei des Texturatlas
  - Höhe und Breite des Texturatlas

[23] Version 0.8.4.4, `http://glm.g-truc.net` (12.08.2010)

- dynamische Objekte, die wie statische Objekte definiert und um folgende Informationen zu ihren Dynamik-Eigenschaften erweitert sind:
  - vom Benutzer manipulierbar?
  - Animationssequenz? Geschwindigkeit, Angaben zum Speicherort der OBJ-Dateien
  - Bewegung entlang eines Hermite-Splines? Geschwindigkeit, aktuelle Position, Referenz auf den Spline, der in einer separaten XML-Datei gespeichert ist

In der konkreten Umsetzung darf die Szene nur maximal eine Punktlichtquelle, aber beliebig viele Spotlights enthalten. Dies ist allerdings keine konzeptionell bedingte Einschränkung. Die Erweiterung auf mehrere Punktlichtquellen ist möglich, wurde aber aus Zeitgründen nicht mehr implementiert.

Die Szenen werden als XML-Datei geladen und können zur Laufzeit verändert werden. Die aktuelle Konfiguration ist wieder als XML-Datei speicherbar. Eine geladene Szene kann zusätzlich aus dem Programm heraus als XML-Datei für das Framework PB [Bärz et al. 2010] gespeichert werden, das für Referenzrenderings genutzt wird.

Nach dem Start des Programms werden folgende Schritte durchlaufen: Nach dem Einlesen der XML-Datei werden OBJ-Modelle geladen (eigene Reimplementierung und Erweiterung des OBJ-Model-Loaders von Nate Robins[24]) und positioniert, Splines konstruiert, Lichtquellen erstellt und die Kamera gesetzt. Anschließend wird die Bounding Box der geladenen Szene ermittelt, welche die Basis für das Frustum der Voxelisierungskamera bildet. Die Berechnung der indirekten Beleuchtung ist auf den durch das Frustum definierten Bereich beschränkt, da sie von der Voxelszene abhängt. Andere Strategien zur Platzierung des Frustums sind denkbar, etwa der Ansatz von [Kaplanyan 2009] (Light Propagation Volumes), bei dem das Volumen mit der Szenenkamera mitgeführt wird. Im Rahmen dieser Arbeit wurden jedoch nur statische Voxelisierungskameras implementiert, keine dynamisch mitwandernden. Nachdem Modelle und ihre Texturatlaskoordinaten geladen sind, wird ein initiales Atlas-Rendering durchgeführt, um die Eingabe-Vertices für die Atlas-Voxelisierung zu generieren. Falls gewünscht, wird an dieser Stelle auch die Vorvoxelisierung der statischen Szenenelemente durchgeführt. Dieser Schritt kann als „Vorverarbeitung“ angesehen werden, allerdings ist er mit wenigen Millisekunden im Vergleich zum Rest des Ladeprozesses (Modelle und Texturen einlesen, Shader kompilieren, uvm.) vernachlässigbar.

[24] enthalten im *Tutors source code package*: `http://www.xmission.com/~nate/tutors.html` (17.08.2010)

8.3 Rendering-Ablauf

Dieser Abschnitt gibt einen Überblick über den gesamten Rendering-Ablauf zur Laufzeit. Alle wichtigen Berechnungen finden auf der GPU statt. Der Ablauf der Berechnung der indirekten Beleuchtung wird für beide implementierten Verfahren getrennt in Abbildung 36 betrachtet, vgl. Kapitel *6 Indirektes Licht durch Abtastung der Hemisphäre* und *7 Indirektes Licht mit virtuellen Punktlichtquellen (VPLs)*.

Die wesentlichen pro Frame durchgeführten Schritte sind:

1. Nutzerinteraktion behandeln
2. aktuelle Modell-Transformationen berechnen (Änderung durch Nutzerinteraktion und Spline-Animation möglich), bei Animationssequenz aktuelles Modell bestimmen
3. Maps für Spot- und Punktlichtquellen rendern
   (a) falls Spotlights in Szene: Spot Maps und Shadow Maps rendern
   (b) falls Punktlicht in Szene: Cube Maps und Cube Shadow Map rendern (6-Pass-Verfahren)
4. Atlastexturen rendern (Weltkoordinaten für binäre Voxelisierung, ggf. weitere Atlanten für nicht-binäre Voxelisierung)
5. Atlas-Voxelisierung: komplett neu oder dynamische Objekte in Vorvoxelisierung einfügen
6. G-Buffer Rendering
7. falls Interleaved Sampling: G-Buffer-Splitting
8. Deferred Shading
9. indirektes Licht berechnen und in Ergebnistextur speichern – vgl. Abbildung 36
10. falls Interleaved Sampling: inverse Split-Operation auf Ergebnistextur
11. Postprocessing (vgl. Abschnitt 8.6)

Deferred Shading Deferred Shading wird zur Berechnung der direkten Beleuchtung (direkte Leuchtdichte) eingesetzt. Abbildung 37 stellt den im Folgenden geschilderten Prozess dar. Die Grundlage des Deferred Shadings sind die G-Buffer-Texturen, die pro Pixel Position und Normale in Weltkoordinaten sowie die Materialinformation enthalten.

Die direkte Beleuchtung wird in mehreren Passes berechnet und in einer Textur akkumuliert. Es wird ein bildschirmfüllendes Rechteck pro Lichtquelle gerendert, welches die Ausführung des jeweils aktiven Beleuchtungsshaders für das Spotlight oder für die Punktlichtquelle anstößt. Der Shader gibt jeweils die direkte Leuchtdichte aus, die diese Lichtquelle zur gesamten

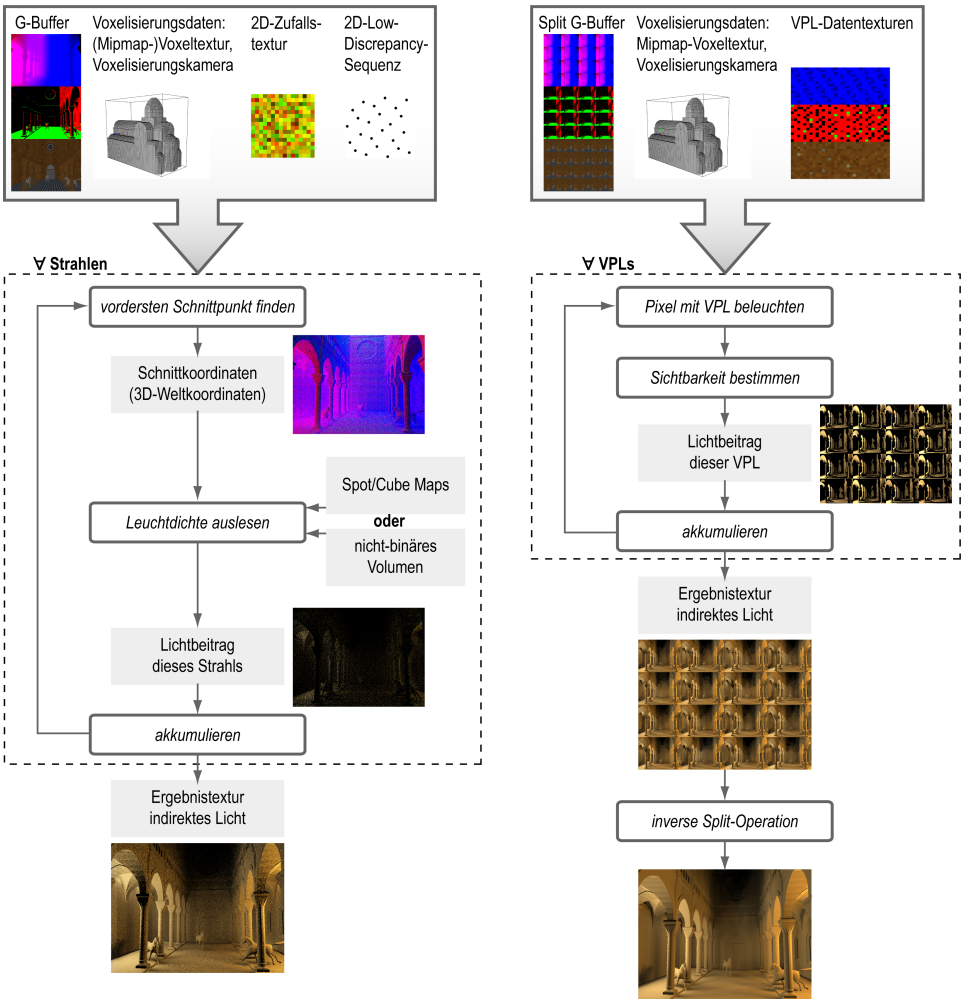

Abbildung 36: Ablauf des Hemisphären-Samplings (links) im Vergleich zur VPL-Beleuchtung mit Interleaved Sampling (rechts).
Hinweis: Jede Zeile der VPL-Datentexturen enthält jeweils eine andere VPL-Menge für das Interleaved Sampling.

Leuchtdichte jedes Pixels beisteuert. Das unverschattete Beleuchtungsergebnis wird mit dem Schattenwert der aktuellen Lichtquelle multipliziert. Hierfür wurde ein einfaches Shadow Mapping mit Percentage Closer Filtering implementiert, d. h. es werden nicht nur ein Tiefenwert, sondern auch Tiefenwerte aus benachbarten Pixeln gelesen und mit der Tiefe des aktuellen Pixels verglichen. Der Mittelwert der Vergleichsergebnisse ist der Schattenwert. Für Spotlights werden 2D Shadow Maps, für Punktlichtquellen Cube Shadow Maps eingesetzt.

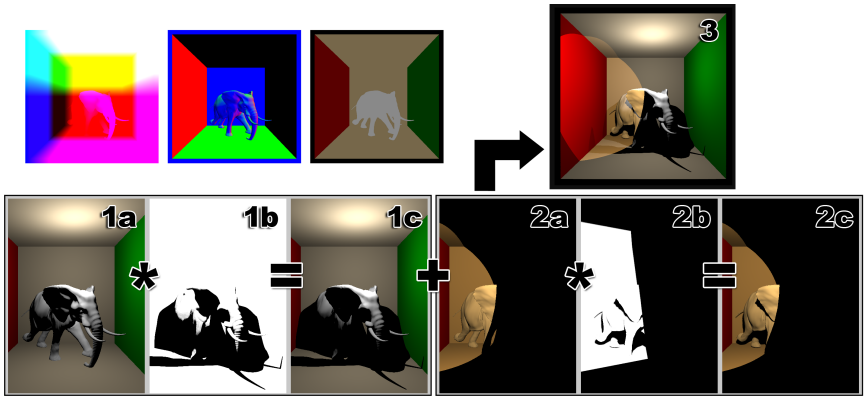

Abbildung 37: Deferred Shading: Ablauf für eine Beispielszene mit einem Punktlicht und einem Spotlight. Oben links: G-Buffer-Texturen. 1a) unverschattete direkte Beleuchtung der Punktlichtquelle, 1b) Schatten der Punktlichtquelle, 1c) verschatteter Beleuchtungsbeitrag der Punktlichtquelle; analog 2a)-c) für das Spotlight. 3) ist die finale Textur mit der direkten Leuchtdichte.

8.4 Technische Details

Während der Implementierung wurde auf einige OpenGL-Extensions zurückgegriffen. Im Folgenden werden die wichtigsten Erweiterungen mit ihrem Verwendungszweck genannt. Die Grundlage für fast alle Renderingschritte stellen *Framebuffer-Objekte* (FBOs) dar, da sie es erlauben, direkt in Texturen diverser Formate zu schreiben. Dabei kann sowohl in eine als auch in mehrere Texturen gleichzeitig geschrieben werden. Dass auch die Mipmap-Stufe der an ein Framebuffer-Objekt gehängten Textur spezifiziert wird, ermöglicht die manuelle Erstellung von Mipmaps. Auf diese Weise werden die in Abschnitt *5.2 Erzeugung der hierarchischen Voxelstruktur* geschilderten Voxeltextur-Mipmaps erzeugt. Es ist möglich, auf die Voxeltextur gleichzeitig lesend und schreibend zuzugreifen, wenn Quelle und Ziel jeweils unterschiedliche Mipmap-Stufen sind. Es wird also jeweils aus der feineren Stufe gelesen und in die nächstgröbere Stufe geschrieben.

An vielen Stellen der Shaderprogrammierung kommt *Shader Model 4* zum Einsatz. Es ist die Voraussetzung für einige andere Extensions, etwa für *Integer-Texturen*, Integer-Arithmetik und Bitoperationen mit GLSL. Integer-Texturen werden für die binäre Voxelisierung und Bitmasken genutzt. Für den Schnitttest aus Kapitel *5 Strahlschnitttest mit binärer Voxelhierarchie* werden initial alle möglichen Voxelstrahl-Bitmuster in einer zweidimensionalen Textur der Größe 128^2 abgelegt. Jedes Texel entspricht einem Bitmuster, das – wie in Abschnitt *5.3.2 Schneiden von Voxelspalten mit Strahl-Bitmustern* geschildert und in Abbildung 16 auf Seite 36 gezeigt – mittels einer auf zwei Bitmasken ausgeführten XOR-Operation erstellt wird. Das Bitmuster kann dann durch einen einzigen Texturzugriff mit den beiden den Strahlabschnitt begrenzenden z-Werten als Texturkoordinaten ausgelesen werden. Dies erwies sich als wesentlich schneller, als das Strahl-Bitmuster für jeden benötigten Strahl „on the fly“ im Shader mit dem Auslesen der zwei Bitmasken und der XOR-Operation zu konstruieren.

Weiterhin ermöglicht Shader Model 4 die Verwendung von *Geometry-Shadern*, welche für die nicht-binäre Atlas-Voxelisierung gebraucht werden. Geometry-Shader stellen in Verbindung mit Framebuffer-Objekten die *Layered-Rendering-Funktionalität* zur Verfügung. Beim Layered Rendering wird eine Textur, die mehrere Ebenen hat (zum Beispiel eine 3D-Textur oder ein Texturarray), an ein Framebuffer-Objekt gebunden und im Geometry-Shader mit der Variable `gl_Layer` entschieden, in welche der Ebenen das aktuelle Geometrieprimitiv rasterisiert werden soll. Diese Technik lässt sich zudem mit Multiple Render Targets kombinieren; das heißt, es lassen sich mehrere 3D-Texturen an ein Framebuffer-Objekt binden und gleichzeitig verschiedene Daten in deren Ebenen schreiben. Von dieser Möglichkeit wurde bei der nicht-binären Atlas-Voxelisierung Gebrauch gemacht, wenn mehr als in einem Texel speicherbare Daten pro Voxel eingetragen werden sollen.

Die Spot Maps wurden zunächst in zweidimensionalen *Texturarrays* mit jeweils einer Ebene pro Spotlight organisiert. Das Ziel war hier, die Anzahl der Texturwechsel beim Auslesen der für das indirekte Licht benötigten Werte zu verringern. Mit einzelnen Texturen steigt die Anzahl der Texturwechsel linear mit jener der Spotlights. Allerdings stellte sich heraus, dass diese Variante im Vergleich zu Einzeltexturen keine Performancevorteile brachte. Eine mögliche Erklärung hierfür ist, dass der Texturzugriff auf ein Texturarray langsamer ist als auf eine herkömmliche 2D-Textur.

CPU und GPU arbeiten asynchron zueinander. Daher sind CPU-Timer nicht geeignet, um die Zeit zu messen, die benötigt wird, um bestimmte OpenGL-Befehle innerhalb eines Frames abzuarbeiten. Aus diesem Grunde wurden OpenGL *Timer Queries* zur genauen Zeitmessung einzelner Renderingschritte benutzt. Diese liefern die zwischen einem gesetzten Start- und Endpunkt vergangene Zeit in Nanosekunden.

8.5 Rückgriff auf existierende Implementierungen

Radikal inverse Funktion. Die Implementierung der radikal inversen Funktion für die Low-Discrepancy-Sequenzen wurde [Pharr und Humphreys 2004, S. 319] entnommen.

Quader-Strahl-Schnitttest. Der Schnitttest aus Kapitel *5 Strahlschnitttest mit binärer Voxelhierarchie* benötigt einen Quader-Strahl-Schnitttest, um die das Voxelstrahl-Bitmuster begrenzenden z-Werte zu ermitteln. Der untersuchte Strahl wird mit verschiedenen Voxelspaltenquadern geschnitten. Gesucht ist jeweils nur der Strahlparameter für den Austrittspunkt, sofern einer existiert. Die Seiten der Voxelspaltenquader sind parallel zu dem orthografischen Voxelisierungsfrustum, sodass ein Axis-Aligned-Bounding-Box-Test genügt. Hierfür wurde auf eine sehr kompakte GLSL-Implementierung von H. M. Piloni[25] zurückgegriffen. Implementiert ist der AABB-Spezialfall des Strahl-Slabs-Schnitttests von [Kay und Kajiya 1986].

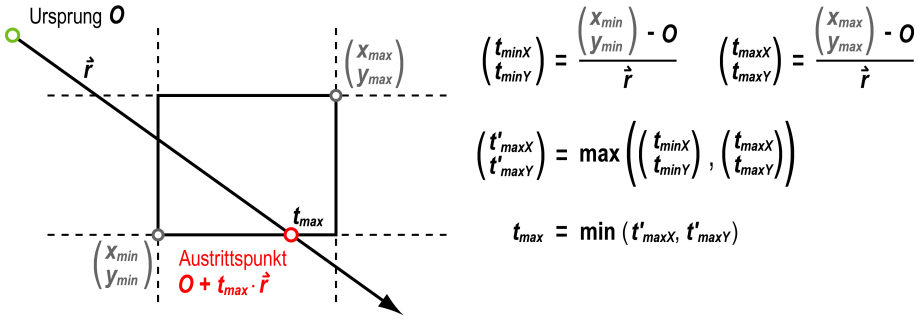

Abbildung 38: Quader-Strahl-Schnitttest nach [Kay und Kajiya 1986], dargestellt in 2D. Das Minimum der Maxima der Strahlparameter, die durch Schnitt des Strahls mit den Ebenen der Box berechnet werden, ist der gesuchte Parameter für den Austrittspunkt.

Ein „Slab“ ist der Raum zwischen zwei parallelen Ebenen. Diese parallelen Ebenen sind bei einer AABB durch die sich jeweils gegenüberliegenden Seiten definiert. Da bei der Berechnung durch die Komponenten der Strahlrichtung geteilt wird (vgl. Abbildung 38), kann eine Division durch null auftreten. Die Autoren des Original-Papers schlagen als Lösung vor, in diesem Fall das Ergebnis auf eine sehr große Zahl zu setzen. Der Voxelstrahlschnitttest setzt dies wie folgt um: Die Strahlrichtungskomponenten des untersuchten Strahls werden initial auf einen sehr kleinen Wert gesetzt, falls der Betrag der Komponenten einen minimalen Wert unterschreitet.

[25] GLSL Ray/AABB Intersection implementation: `http://blog.piloni.net/?p=114` (12.08.2010)

8.6 Postprocessing

8.6.1 Geometrie-sensitiver Filter

Nachdem das berechnete indirekte Licht in einer 2D-Textur vorliegt, muss es mit der direkten Beleuchtung kombiniert und die Kombination dargestellt werden.

Zuvor kann optional ein geometrie-sensitiver Filter auf die Textur mit der indirekten Beleuchtung angewendet werden, um Rauschmuster zu verringern oder – je nach Stärke des Rauschens – ganz zu unterdrücken. Es wurden verschiedene Filter-Algorithmen ausprobiert.

Das von [Herzog et al. 2010] beschriebene *Spatial Upsampling* lieferte gute Ergebnisse und wurde daher in dieser Arbeit für das Filtern der Bilder eingesetzt. Obwohl es sich um ein „Upsampling“ handelt, kann der Algorithmus auch direkt auf voll aufgelöste Texturen angewendet werden. In der Theorie ist der Filter nicht separierbar, in der Praxis wird aus Performancegründen dennoch so verfahren. Statt eines einzelnen Durchlaufs, in dem eine $m \times m$-Filtermaske auf die zu filternde Textur angewendet wird, werden zwei Passes hintereinander durchgeführt. Der erste Pass filtert die Eingabe-Textur in horizontaler Richtung mit einer $m \times 1$-Maske, der zweite Pass filtert die Ergebnis-Textur des ersten Passes in vertikaler Richtung mit einer $1 \times m$-Maske.

Die Stärke der Filterung hängt von der Größe der Maske und der gewählten Strategie zur Gewichtung der ausgelesenen Pixel ab. Grundsätzlich steuert der Inhalt des G-Buffers (Normalen und Positionen) die Filter-Gewichte. Bildpunkte, die in der Szene nah zur Position des aktuell betrachteten Pixels sind und ähnliche Oberflächenorientierungen haben, fließen stärker in das Ergebnis des betrachteten Pixels ein (Pixel in der Mitte der Maske). Um den Filtereffekt zu verstärken, kann statt oder zusätzlich zu der Vergrößerung der Maske der Filter auch mehrere Male hintereinander ausgeführt werden.

Da durch einen starken Filter Details der Oberflächenform verloren gehen, schlägt [Nichols 2010, S. 50] einen „Schummelfaktor“ vor, der die weichgezeichnete Textur durch folgende Modifikation pro Pixel wieder mit Oberflächendetails anreichert (siehe Abbildung 39). $\vec{v}$ ist der View-Vektor zum Pixel, $\vec{n}$ die Pixelnormale, α der den Effekt kontrollierenden Wert (größter Effekt mit $\alpha = 1.0$).

$$color_{out} = color \cdot (\alpha \cdot (\vec{v} \cdot \vec{n}) + (1 - \alpha)) \tag{7}$$

Da diese Modifikation das Ergebnis verfälscht und generell abdunkelt, wird der Effekt allerdings nur bei sehr starken Filtern hinzugeschaltet.

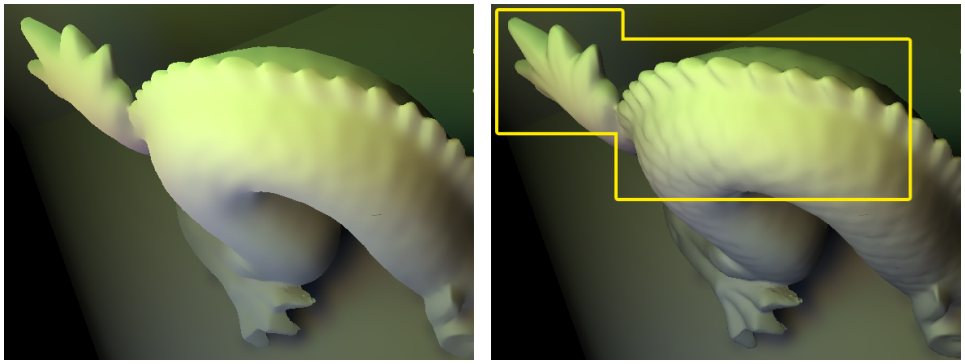

Abbildung 39: Oberflächendetails. Links: zu Demonstrationszwecken stark gefilterte Textur (Filtermaske 15 Pixel breit, 2 Durchläufe), rechts: mit Gleichung (7) modifizierte Farbe, $\alpha = 0.75$.

8.6.2 Unterschiedliche Behandlung von Lambert- und Phong-Oberflächen

Nachdem Phong-Materialien in die Verfahren integriert wurden, stellte sich im Zusammenhang mit dem Filter ein Problem heraus. Zuvor wurde bei rein diffusen Szenen nur die indirekte Beleuchtungsstärke pro Pixel gespeichert, diese Textur gefiltert und anschließend die indirekte Leuchtdichte durch Multiplikation mit der konstanten BRDF berechnet. Somit blieben trotz Filter Materialdetails, insbesondere solche von Modelltexturen, erhalten.

Mit der Erweiterung auf Phong-Materialien war dieses Vorgehen zunächst nicht mehr möglich. Die von Phong-Oberflächen zur Kamera reflektierte indirekte Leuchtdichte hängt von der Position der Kamera und der Richtung der einfallenden Leuchtdichte ab. Diese Faktoren müssen bereits während der Akkumulation des indirekten Lichts berücksichtigt werden. In der ersten einheitlichen Implementierung wurden Lambert- und Phong-Oberflächen gleich behandelt, es wurde also jeweils die indirekte Leuchtdichte akkumuliert. Allerdings sind darin bereits die Materialdetails enthalten, welche folglich durch den Filterprozess verloren gehen bzw. unscharf werden.

Als Kompromiss wurden daher Lambert- und Phong-Oberflächen bei der Akkumulation unterschiedlich behandelt: Für Lambert-Oberflächen wird die indirekte Beleuchtungsstärke gespeichert, für Phong-Oberflächen die indirekte Leuchtdichte. Beim anschließenden Filtern verschwimmen nur die Materialdetails der Phong-Objekte. Es wurden jedoch in den in dieser Arbeit verwendeten Szenen keine texturierten, sondern nur einfarbige Phong-Modelle eingesetzt. Die indirekte Leuchtdichte der diffusen Objekte wird wie zuvor berechnet, indem die indirekte Beleuchtungsstärke der betreffenden Pixel mit der konstanten diffusen BRDF multipliziert wird:

$$L_{indirect}(Pixel) = \begin{cases} \frac{\rho_{diffuse}}{\pi} \cdot T(Pixel) & \text{falls Material}(Pixel) = \text{diffus} \\ T(Pixel) & \text{falls Material}(Pixel) = \text{Phong} \end{cases}$$

$T(Pixel)$ ist der in der Ergebnistextur gespeicherte indirekte Beleuchtungswert für einen Pixel. Für den diffusen Fall ist also $T(Pixel) = E_{indirect}$ und für Phong $= L_{indirect}$.

8.6.3 Kombination und Tonemapping

Der Kombinationsshader am Ende des Renderingprozesses berechnet zunächst die indirekte Leuchtdichte für jedes Pixel. Der finale Leuchtdichtewert pro Pixel ist die Summe aus direkter und indirekter Leuchtdichte.

Falls die Leuchtdichtewerte den vom Monitor darstellbaren Bereich von 0 bis 1 überschreiten, ist es nötig, sie vor der Anzeige in diesen Bereich zu transformieren (*Tone Mapping*). Implementiert wurden globale Tone-Mapping-Operatoren, welche auf jedes Pixel die gleiche Abbildung anwenden: eine einfache lineare und logarithmische Abbildung sowie der Tone-Mapping-Operator von [Drago et al. 2003]. Da das Ziel die Kompression der Leuchtdichtewerte ist, muss für die Abbildung der maximale Leuchtdichtewert bekannt sein. Dieser kann z. B. mit einer Reduktionsoperation auf der Textur mit der finalen Leuchtdichte ermittelt werden. Den automatisch für jedes Frame ermittelten maximalen Leuchtdichtewert als Eingabe für das Tone-Mapping zu benutzen, erwies sich jedoch als keine gute Lösung. Insbesondere bei Kameraschwenks wurde das Bild durch die sich schnell ändernden Leuchtdichtewerte heller und dunkler. Daher wird der ermittelte maximale Wert dem Nutzer mitgeteilt, damit dieser einen geeigneten festen Wert manuell setzen kann.

Der letzte Schritt vor der Anzeige des finalen Bildes ist die *Gamma-Korrektur*

$$(R', G', B') = (R, G, B)^{\frac{1}{\gamma}}.$$

Der Gamma-Wert wird im Allgemeinen auf 2.2 gesetzt.

9 Ergebnisse und Bewertung

9.1 Testsystem

Die Zeitmessungen wurden auf folgendem System mit den in Abschnitt *8.4 Technische Details* erwähnten Timer Queries durchgeführt:

Grafikkarte	NVIDIA GeForce GTX 260 (896 MB, 192 Streamprozessoren), Treiberversion 258.96
CPU	Intel Core 2 Duo 6420 ($2 \times 2{,}13$ GHz)
Arbeitsspeicher	3 GB RAM
Betriebssystem	Windows XP

Falls nicht anders angegeben, handelt es sich bei den gemessenen Zeiten um den Durchschnitt der Ergebnisse aller Frames, die innerhalb von fünf Sekunden gerendert wurden.

9.2 Performance der Atlas-Voxelisierung

Im Folgenden wird die Performance der in Kapitel 4 vorgestellten Atlas-Voxelisierung betrachtet. Sie hängt unmittelbar von der Anzahl der in die Voxeltextur gerenderten Vertices ab. Mit jeder Erhöhung der Voxeltexturauflösung muss auch die Anzahl der gerenderten Vertices erhöht werden, was gleichbedeutend mit einer Vergrößerung der Atlastextur ist.

Im Idealfall würde jeder Vertex genau ein volles Voxel erzeugen.[26] Hieraus ergibt sich das Kriterium, welches die Anzahl der benötigten Vertices und unmittelbar auch die Geschwindigkeit bestimmt: die *Datendichte* des voxelisierten Modells. Als Datendichte wird hier der prozentuale Anteil aller vollen Voxel an der Gesamtvoxelzahl des Volumens verstanden.

Das Modell der Sibenik Kathedrale (ca. 80.000 Dreiecke) hat eine recht hohe Dichte von 4-7 % (abhängig von der Voxelauflösung). Zudem enthält das Modell Dreiecke sehr unterschiedlicher Größen, was die Atlaserstellung verkompliziert. Bei Zeitmessungen wurde zum einen die Vertexanzahl manuell pro Voxelauflösung so gesetzt, dass die Voxelisierung möglichst wenige Löcher enthält. Die Ergebnisse zeigt das Diagramm auf der linken Seite von Abbildung 40. Zum anderen wurde die Vertexzahl für jede Voxelauflösung gleich gehalten, um zu belegen, dass die Voxelisierung nicht durch die Füllrate der Grafikkarte, sondern die hohe Anzahl zu rendernder Vertices

[26] In der Praxis wird dieser Idealfall nie erreicht. Einige Vertices fallen in dieselben Voxel.

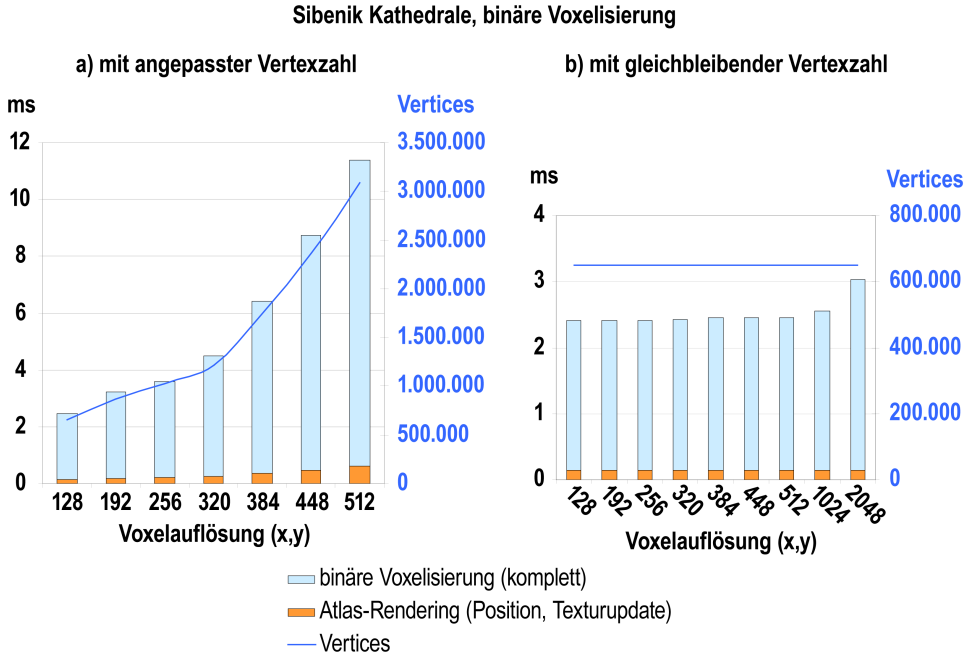

Abbildung 40: Zusammenhang von Vertexanzahl, Voxeltexturauflösung (die Dimensionen x und y werden variiert, z bleibt jeweils = 128) und Voxelisierungdauer. Dargestellt sind die Zeiten für eine komplette binäre Atlas-Voxelisierung der Sibenik Kathedrale.

limitiert ist (siehe Diagramm auf der rechten Seite von Abbildung 40). Die Ergebnisvoxelisierungen für die Messungen auf der rechten Seite sind folglich bei hohen Voxelauflösungen löchrig (ähnlich wie in Abbildung 9 auf Seite 23).

Das bereits in Kapitel *4 Voxelisierung mit Texturatlanten* gezeigte (voxelisierte) Elefanten-Modell hat etwa gleichviele Dreiecke wie die Sibenik Kathedrale, allerdings eine viel geringere Datendichte von 1-2 % und überwiegend fast gleich große Dreiecke. Die Voxelisierungszeit ist entsprechend geringer. Sie liegt unter 1 Millisekunde für eine Voxelauflösung von $256 \times 256 \times 128$ (ca. 200.000 Vertices) und knapp über 4 Millisekunden für $512 \times 512 \times 128$ (ca. 1,1 Mio. Vertices).

Der Ansatz dieser Diplomarbeit für die binäre Voxelisierung ist daher, eine statische Umgebung vorzuvoxelisieren, in die zur Laufzeit die dynamischen Objekte eingefügt werden. Das Diagramm in Abbildung 41 nennt die Zeiten für das Einfügen mehrerer dynamischer Objekte, die im Verhältnis zur gesamten Szene klein sind.

Die Vorvoxelisierung mit Einfügen dynamischer Objekte kann ebenfalls für nicht-binäre Voxeldaten genutzt werden, die für statische Teile zur Laufzeit gleich bleiben, z. B. die Normalen und das Material (vgl. Diagramm links

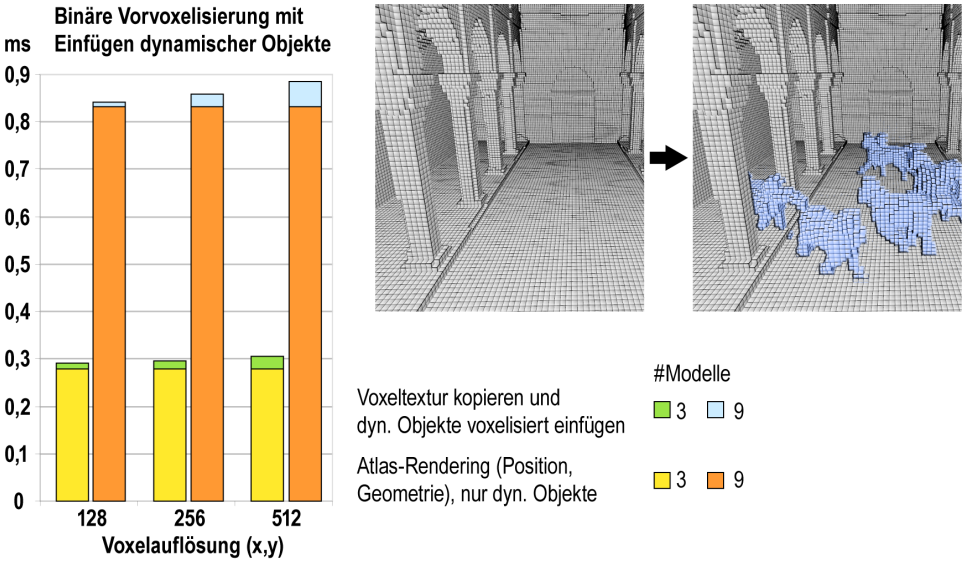

Abbildung 41: Die statische binär vorvoxelisierte Szene ist die Sibenik Kathedrale. In diese werden mehrere (hier im Vergleich drei und neun) animierte Modelle eines Pferdes eingefügt. Da es sich um animierte Objekte handelt, wird der Atlas pro Objekt in jedem Frame neu erzeugt, indem die Geometrie des aktuellen Modells aus der Animationssequenz in die Atlastextur rasterisiert wird. Da die Modelle im Verhältnis zur Kathedrale klein sind, sind sehr geringe Atlasauflösungen und somit wenige Vertices ausreichend.

in Abbildung 42). Für die Leuchtdichte ist dies nicht möglich, daher wurde für eine komplette nicht-binäre Voxelisierung eine gesonderte Messung durchgeführt (Diagramm rechts in Abbildung 42).

Zusammenfassend gesehen eignet sich die Atlas-Voxelisierung grundsätzlich auch für große Szenen. Bei Modellen, deren Voxelrepräsentation eine hohe Datendichte aufweist, steigt der Aufwand jedoch stark für höhere Auflösungen, da entsprechend mehr Vertices gerendert werden müssen. Die Variante, einzelne dynamische Objekte in eine größere statische Umgebung einzufügen, stellt sich als am schnellsten heraus.

Die Qualität der Voxelisierungen wird hier nicht gesondert beurteilt. Sie muss für den Einsatz in der näherungsweisen Berechnung globaler Beleuchtung ausreichen, nicht aber Kriterien wie jene der konservativen Voxelisierung erfüllen.[27]

[27] Diese wird in Kapitel *3 Verwandte Arbeiten*, Abschnitt *3.1.2 Voxelisierung mittels rasterisierter Geometrie* erläutert.

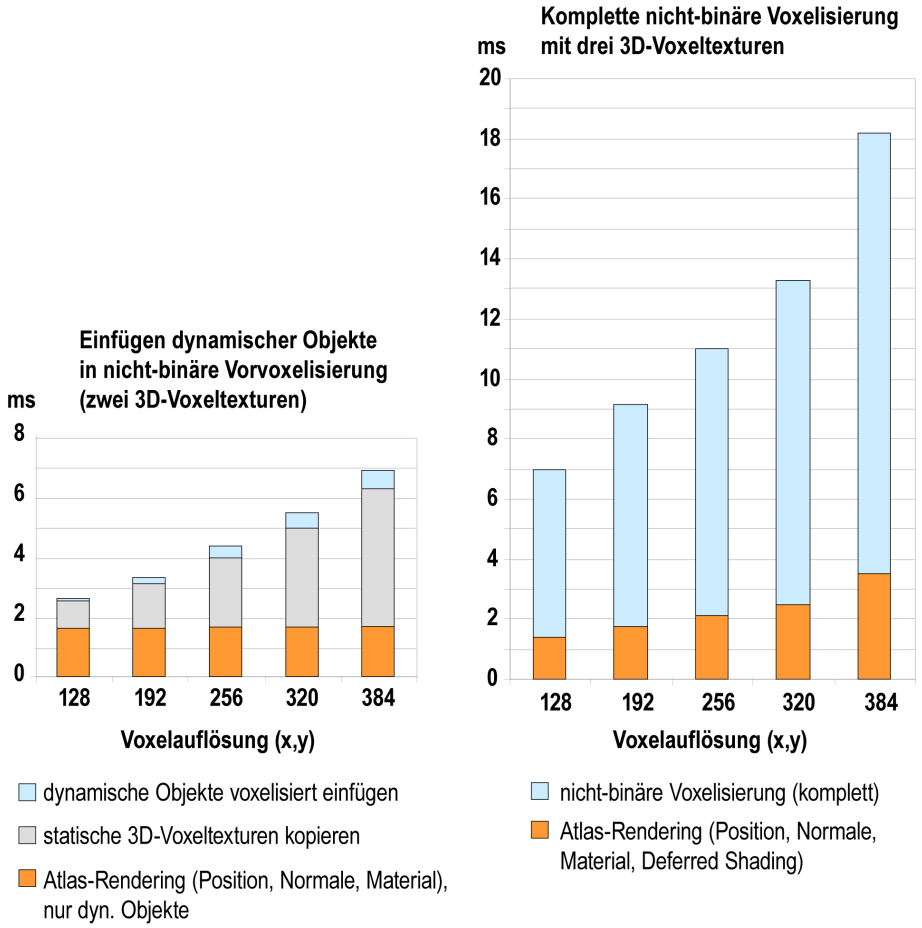

Abbildung 42: Diagramm links: Analog zur Zeitmessung in Abbildung 41 wurde die Sibenik Kathedrale nicht-binär vorvoxelisiert. Die Daten (Normalen und Material) sind in zwei 3D-Texturen gespeichert. Diese müssen zur Laufzeit für jede Aktualisierung des Volumens (in der implementierten Version: in jedem Frame) zunächst kopiert werden, damit in die Kopie die dynamischen Objekte (neun animierte Pferdemodelle) eingefügt werden können. Das Anlegen der Kopie stellt neben dem Atlas-Rendering den größten Zeitfaktor dar.
Diagramm rechts: Da sich die direkte Beleuchtung auch auf statischen Objekten zur Laufzeit ändern kann, ist es nicht möglich, eine Vorvoxelisierung der Leuchtdichte einzusetzen. Daher müssen sämtliche Teile der Szene (hier: Sibenik Kathedrale) voxelisiert werden. In dieser Messung werden drei Volumentexturen gleichzeitig mittels Multiple Render Targets und Layered Rendering erzeugt. Für den Texturatlas muss ein Deferred Shading ausgeführt werden, um die Leuchtdichte zu ermitteln (hier: zwei Spotlights in der Szene).

9.3 Beleuchtungssimulation

9.3.1 Vergleich verschiedener Parametereinstellungen

Dieser Abschnitt zeigt anhand zweier Testszenen („Sibenik Kathedrale“ und Cornell Box ähnliche Umgebung), wie verschiedene Parametereinstellungen das Ergebnis der Beleuchtungssimulation hinsichtlich Qualität und Geschwindigkeit der Berechnung beeinflussen. Unter guter Bildqualität wird hier ein optisch ansprechendes Ergebnis mit möglichst wenigen den Betrachter störenden Artefakten verstanden. Die Korrektheit der Lichtsimulation steht weniger im Vordergrund, wird aber im nachfolgenden Abschnitt *9.3.2 Vergleich mit Referenzbildern* betrachtet. Für alle hier diskutierten Parameter ist ein Kompromiss zwischen Renderingzeit und Bildqualität zu finden. Die Auflösung der in diesem Abschnitt gezeigten Ergebnisbilder beträgt 768×512. Bei Bildern, die mit Hemisphären-Sampling erzeugt wurden, handelt es sich um die Spot-/Cube-Maps-Variante.

Anzahl verfolgter Strahlen (Hemisphären-Sampling). Wie in Abschnitt *6.2.1 Monte-Carlo-Integration* erläutert, sinkt der Fehler im Bild (= weniger Rauschen), wenn die Anzahl der Strahlen erhöht wird, mit denen die Hemisphäre abgetastet wird. In Abbildung 20 auf Seite 42 war bereits ein Beispiel hierfür zu sehen.

Anzahl der virtuellen Punktlichtquellen. Werden nur wenige virtuelle Punktlichtquellen zur Beleuchtung benutzt, können einzelne Schatten der VPLs sichtbar werden – insbesondere ohne Interleaved Sampling (siehe hierzu auch Abbildung 29 auf Seite 61). Es wurde daher grundsätzlich Interleaved Sampling eingesetzt. Als geeignetes Interleaved-Sampling-Muster stellte sich 4×4 in Kombination mit 16-32 VPLs pro Pixel heraus. In Abbildung 43 sind Ergebnisbilder mit steigender VPL-Anzahl zu sehen. Die dort genannten VPL-Zahlen sind dadurch bedingt, dass das Box-Modell eine offene Wand hat und somit ungültige VPL-Samples in der Cube Map der Punktlichtquelle verworfen werden mussten. Die gewählten Sample-Zahlen waren 1, 4, 16, 32.

Wenn sich direkt beleuchtete Objekte oder die Lichtquelle bewegen, ändern einige virtuelle Punktlichtquellen sprunghaft ihre Position. Der Grund liegt darin, dass bei der implementierten Sampling-Strategie die Sampling-Punkte, die die VPLs bestimmen, feste Positionen auf der aus Sicht der Lichtquelle gerenderten Textur haben. Bei der indirekten Beleuchtung kann dieses Springen als Flackern sichtbar werden. Je mehr virtuelle Punktlichtquellen verwendet werden, desto weniger trägt jede einzelne VPL zum Gesamtergebnis bei und das Flackern wird abgeschwächt.

Begrenzung der Länge der Strahlen und des Beleuchtungsradius der VPLs. Die Länge der für den Lichtaustausch verfolgten Strahlen ist

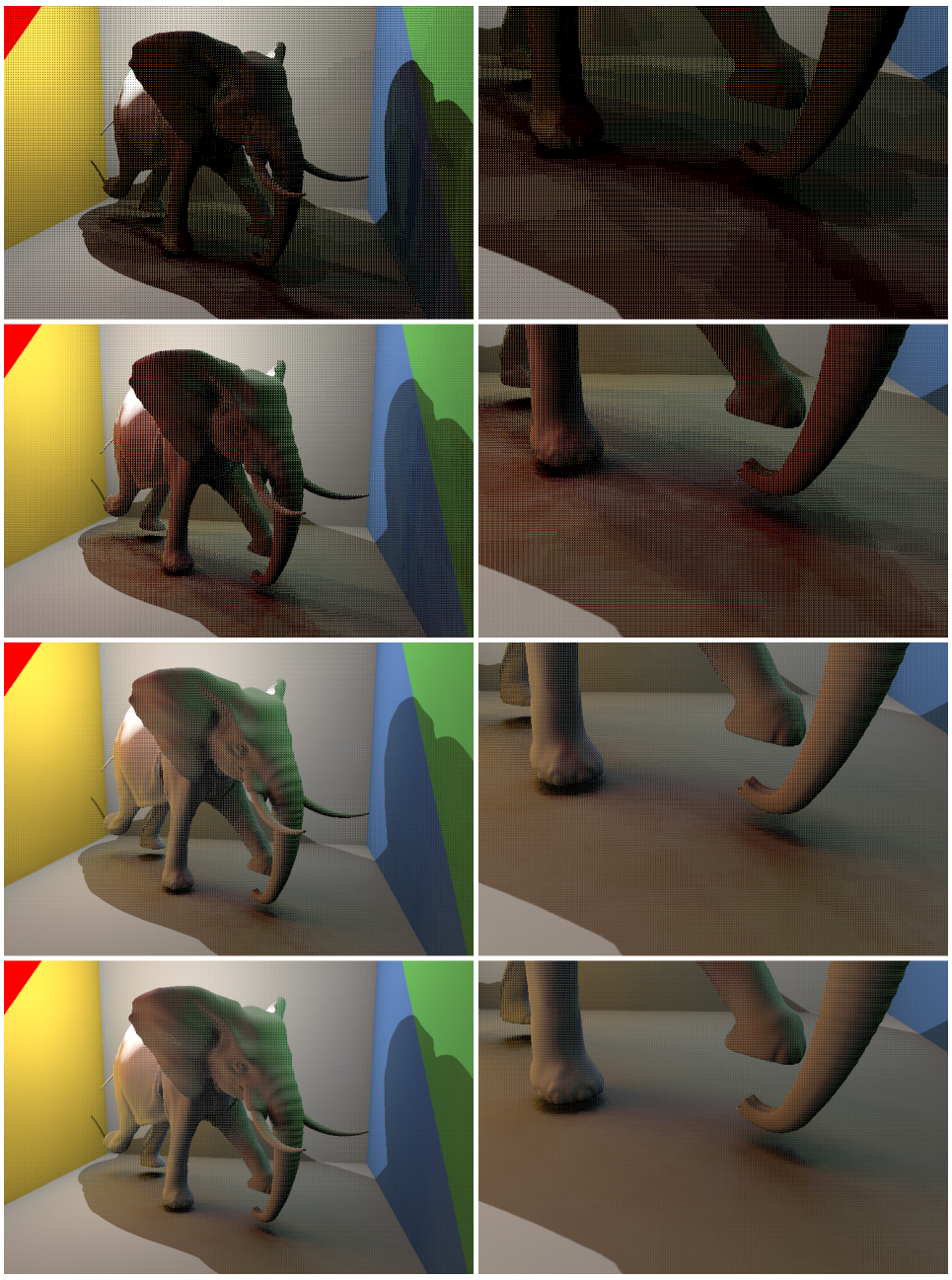

Abbildung 43: Variation der VPL-Anzahl, ungefilterte Ergebnisbilder mit erhöhtem Kontrast. Primäre Lichtquelle: Punktlichtquelle. VPL-Anzahl in den Bilderreihen von oben nach unten: 1, 4, 14, 28. Die zugehörigen Renderingzeiten (komplettes Bild): 16 ms, 24 ms, 59 ms, 102 ms. Rechts Nahansicht des indirekten Schattens auf dem Boden der Box. Interleaved-Sampling-Muster: 4×4. Voxeltexturauflösung: 128^3.

grundsätzlich durch den Voxelisierungsbereich limitiert. Innerhalb dieses Bereichs können die Strahlen zusätzlich künstlich verkürzt werden. Beim Hemisphären-Sampling entspricht dies dem Radius der Hemisphäre. Nur Objekte, die innerhalb dieses Radius liegen, können also zur indirekten Beleuchtung des betrachteten Punktes beitragen (Abbildung 44, obere Reihe). Bei der Beleuchtung mit VPLs entspricht die Begrenzung dem Beleuchtungsradius der VPLs. Eine VPL beleuchtet dann nur Objekte innerhalb des gesetzten Radius (Abbildung 44, untere Reihe). Die Performance steigt durch eine solche Begrenzung, da weniger Abtastpunkte auf dem kurzen Strahl untersucht werden müssen oder der Schnitttest früher abgebrochen werden kann. Der Nachteil ist, dass der Lichttransport so auf Objekte aus der nahen Umgebung begrenzt wird.

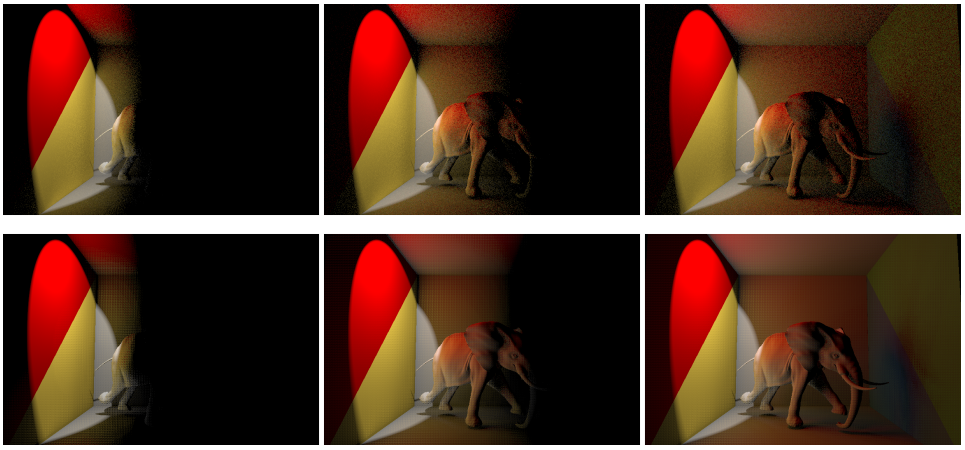

Abbildung 44: Variation des Radius, ungefilterte Ergebnisbilder mit erhöhtem Kontrast. Obere Reihe: Hemisphären-Sampling mit jeweils 16 Strahlen, untere Reihe: VPL-Beleuchtung mit 24 VPLs pro Pixel und 4×4 Interleaved Sampling. Von links nach rechts: Radius 2, 4, 8. Die Box hat die ungefähren Abmessungen $6 \times 6 \times 6$. Zeiten für indirekte Beleuchtung von links nach rechts: Strahlschnittpunkte mit Mipmap-Schnitttest: 55 ms, 85 ms, 110 ms, Strahlschnittpunkte mit uniformen Ray Marching (für vergleichbare Qualität angepasste Schrittzahl): 100 ms, 180 ms, 215 ms, VPL-Beleuchtung: 15 ms, 30 ms, 70 ms.

3D-Volumen vs. Spot und Cube Maps (Hemisphären-Sampling). Optisch unterscheiden sich die beiden Varianten nur geringfügig, bei der Geschwindigkeit jedoch merklich. Das Bild ganz rechts in Abbildung 44 mit Radius 8 benötigt 110 ms für die Berechnung des indirekten Lichts und 120 ms insgesamt für ein Bild, wenn die Maps-Variante verwendet wird. Mit dem Leuchtdichte-Volumen steigt die Zeit für ansonsten identische Einstellungen auf 126 ms für das indirekte Licht und 140 ms insgesamt (davon fast 6 ms für die Voxelisierung, vgl. auch Diagramm in Abbildung 42).

Genauigkeit des Schnittpunktes bei der Voxelstrahlverfolgung. Wie gut der Schnittpunkt eines Strahls mit der Voxelszene den tatsächlichen Schnitt (Strahl/Dreieck) annähert, hängt vor allem von der Voxelisierungsauflösung ab. Diese wird gesondert im nachfolgenden Absatz behandelt. In Abbildung 45 werden die beiden in Abschnitt *6.4.1 Single Bounce* erwähnten Varianten zum Ray Marching (1 Bit vs. 3 Bits) und Mipmap-Schnitttest mit Schnittpunktfindung gegenübergestellt. Uniformes Ray Marching mit zu geringer Abtastrate des Strahls erzeugt Banding, der Mipmap-Schnitttest zeigt ebenfalls Artefakte bei zu wenigen maximal zugelassenen Iterationen (Abbildung 46).

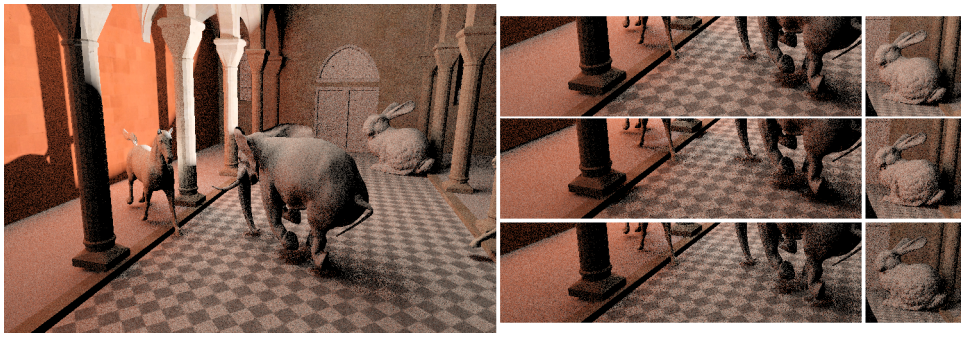

Abbildung 45: Unterschiedliche Strahlverfolgung beim Hemisphären-Sampling. Anzahl der Strahlen: 32. Großes Bild links: Mit Mipmap-Schnitttest gefundene Schnittpunkte. Die Ausschnitte in der Mitte und rechts zeigen von oben nach unten das Ergebnis mit Mipmap-Schnitttest, mit 3-Bits-Ray-Marching und mit 1-Bit-Ray-Marching. Anzahl der Abtastpunkte beim Ray Marching: jeweils 90 Schritte pro Strahl. Voxelauflösung: $256^2 \times 128$. Die Unterschiede zeigen sich bei den indirekten Schatten (Säule, Beine) und dem Ausmaß des Rauschens. Das Mipmap-Schnitttest-Bild ist am rauschärmsten, jenes mit 1-Bit-Ray-Marching am stärksten verrauscht.

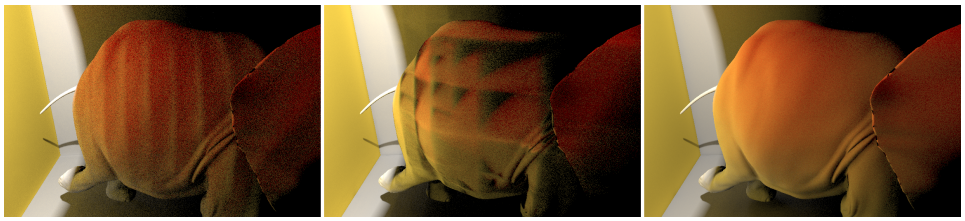

Abbildung 46: Banding bei Ray Marching mit zu geringer Schrittzahl (links) und Artefakte bei Mipmap-Schnitttest mit zu frühem Abbruch des Tests (Mitte). Rechts: ausreichend viele Iterationen. Ausschnitt aus der in Abbildung 44 gezeigten Szene.

Voxeltexturauflösung. Die Auflösung der Voxeltextur bestimmt, wie genau die Voxelstrahlverfolgung in der Szene sein kann. Eine grobe Voxelrepräsentation kann zu blockigen indirekten Schatten führen, zu Selbstverschattung oder verringertem Lichtaustausch aufgrund eines großen Offsets, der

Selbstverschattung vermeiden soll. Eine feine Auflösung dagegen bedeutet nicht nur mehr Speicherplatzverbrauch, sondern vor allem auch einen Anstieg der Rechenzeit bei der Voxelstrahlverfolgung bzw. dem Schnitttest. Je feiner die Auflösung, desto mehr Voxel müssen auf derselben Strecke untersucht werden. Dies gilt insbesondere für Ray-Marching-Verfahren. Während der Diplomarbeit wurden im Allgemeinen für die eingesetzten Testszenen die Auflösungen 128^3 und $256^2 \times 128$ verwendet. Abbildung 47 zeigt den Schatten einer einzigen VPL bei verschiedenen Voxeltexturauflösungen. In Abbildung 48 werden die indirekten Schatten derselben Szene für Hemisphären-Sampling und VPL-Beleuchtung gegenübergestellt.

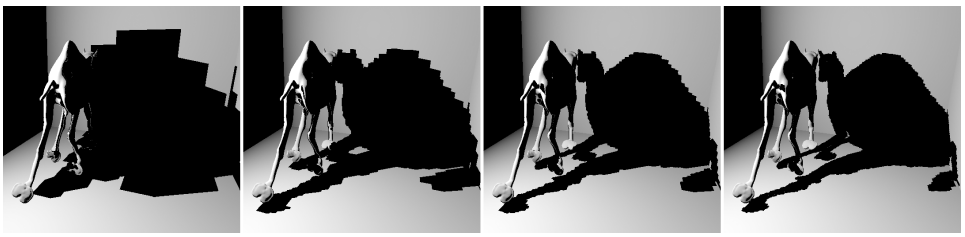

Abbildung 47: Variation der Voxeltexturauflösung, Beleuchtungsbeitrag (hier in Graustufen) und Schatten einer einzigen VPL. Voxeltexturauflösungen (x,y) von links nach rechts (z-Richtung jeweils 128): 16^2, 64^2, 128^2, 256^2.

Auflösung der Reflective Shadow Maps, Spot Maps und Cube Maps. Die Szene wird aus Sicht der Lichtquellen gerendert, um die Ausgangstexturen für die VPL-Generierung (Reflective Shadow Maps) oder die Texturen zur Speicherung der direkten Beleuchtung (Spot und Cube Maps) zu erzeugen. Die Rechenzeit dieses Schritts ist weniger durch die Füllrate, d. h. die Auflösung der Texturen, als durch das Zeichnen der Szenengeometrie limitiert. Die Spot Maps eines Spotlights in der Sibenik Kathedrale benötigen für eine Auflösung von 128^2 ca. 1,5 ms, für 1024^2 ca. 3,0 ms. Die Generierungszeiten der Cube Maps einer Punktlichtquelle in der Sibenik Kathedrale sind ca. 5,0 ms für 128^2-aufgelöste Würfelseiten und 6,2 ms für 1024^2.

Für die Zwecke in dieser Diplomarbeit müssen die Texturauflösungen nicht sonderlich hoch sein. Es zeigte sich, dass eine Auflösung von 256^2 für Spotlight-Texturen ausreichend ist, ebenso wie eine Auflösung zwischen 128^2 und 256^2 für die Seiten der Cube Map einer Punktlichtquelle.

Die Cube-Map-Generierung wird bei der gewählten Implementierung mit sechs Render-Passes schnell aufwändig. Ein einziger Pass mit Geometry-Shader stellte sich jedoch als noch langsamer heraus. Eine Erklärung dafür ist, dass der Geometry-Shader viele Werte für die in den Cube Maps zu speichernden Informationen an den Fragment-Shader weiterreichen muss. Die Performance des Geometry-Shaders sinkt ab einer gewissen Anzahl von weitergereichten Float-Werten: Laut [NVIDIA 2008, S. 41] ist mit 1-20 aus-

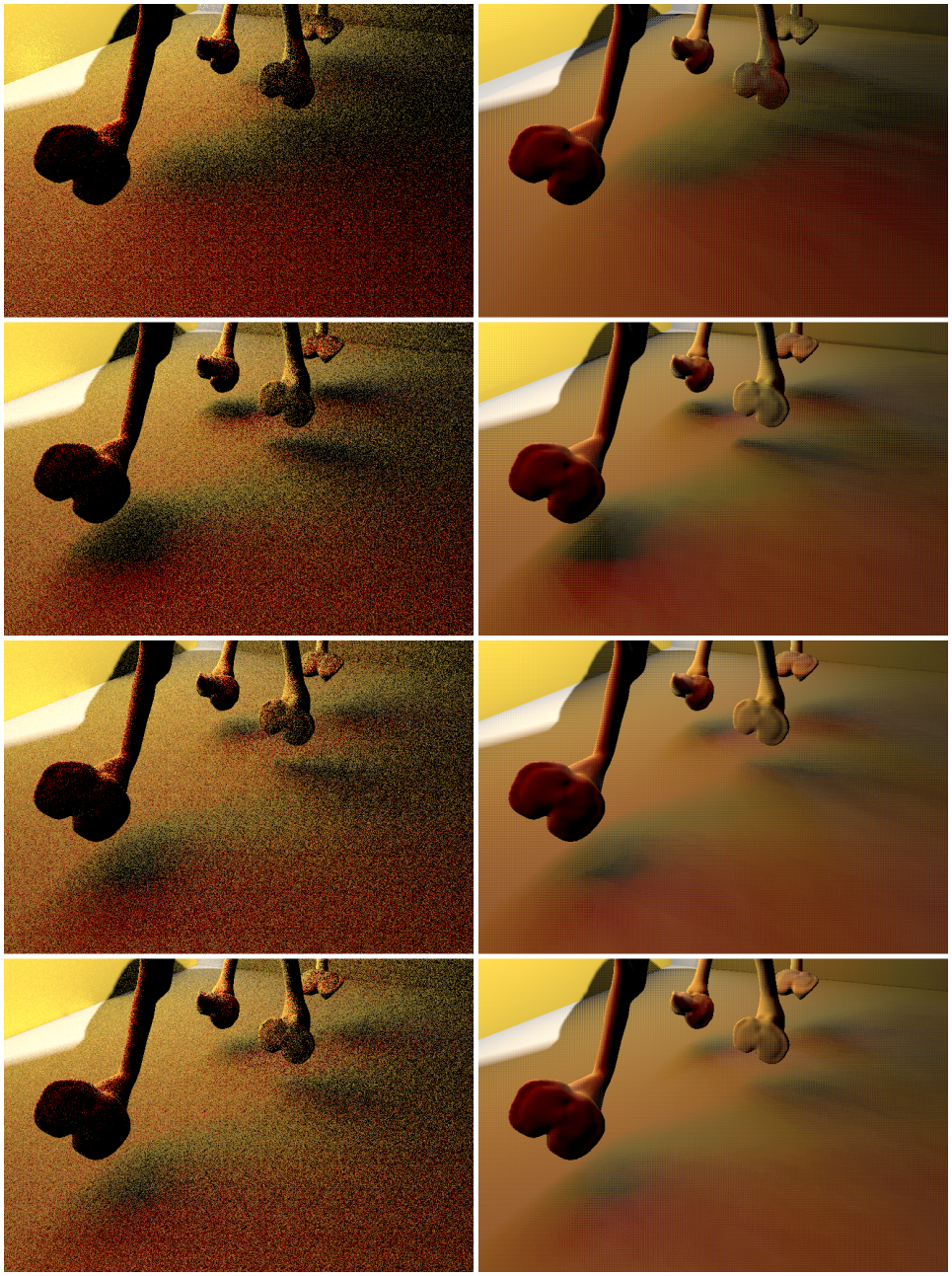

Abbildung 48: Variation der Voxeltexturauflösung, ungefilterte Ergebnisbilder mit erhöhtem Kontrast. Szene: Kamel in Cornell Box, die linke Wand wird von einem Spotlight angestrahlt. Linke Reihe: Hemisphären-Sampling, rechte Reihe: VPL-Beleuchtung. Voxeltexturauflösungen (x,y) von oben nach unten (z-Richtung jeweils 128): 16^2, 64^2, 128^2, 256^2. Gröbere Auflösungen bewirken stärkere Verdeckung und somit dunklere indirekte Schatten. Bei der VPL-Beleuchtung führen die Voxeloffsets zu fehlender Beleuchtung, insbesondere dort, wo Wand und Boden aneinander grenzen (besonders deutlich in der rechten Bilderreihe im obersten Bild zu sehen, gelbe Wand links grenzt an grauen Boden). Kleinere Offsets bewirken jedoch Selbstverschattung, vgl. Abbildung 15, Seite 33.

gegebenen Skalarwerten die höchste Geschwindigkeit des Geometry-Shaders möglich[28]. Bei 27-40 Skalarwerten halbiert sich die Geschwindigkeit bereits. Beim Cube-Map-Rendering führt alleine die Versechsfachung jedes Dreiecks zu $3 \cdot 6 = 18$ ausgegebenen Vertices, die zudem Attribute wie Normale und Weltkoordinate (je 3 Float-Werte) haben – insgesamt also über 100 Skalarwerte. Techniken wie View Frustum Culling würden das sechsfache Rendering wahrscheinlich beschleunigen.

Anzahl der Interreflexionen. Es hängt stark von der Szene ab, wie groß der sichtbare Effekt von mehr als einer Interreflexion auf die indirekte Beleuchtung ist. Das implementierte VPL-Beleuchtungsverfahren kann maximal zwei Interreflexionen darstellen (Abbildung 49). Der Rechenaufwand kann sich hierbei potentiell verdoppeln.

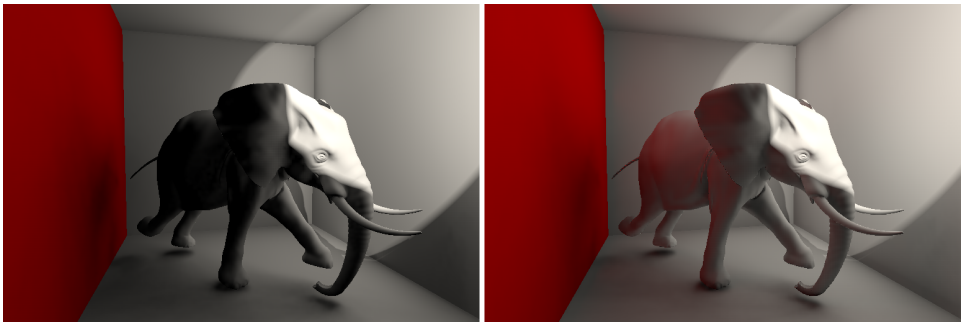

Abbildung 49: VPL-Beleuchtung (Voxelauflösung 128^3, 4×4 Interleaved Sampling, gefiltert): Links eine Interreflexion (24 VPLs, Gesamt-Renderingzeit $\approx$ 80 ms), rechts zwei Interreflexionen ($\approx$ 130 ms).

Abbildung 50: Voxel Path Tracer (100 Pfade pro Pixel, Voxelauflösung $256^2 \times 128$): Links eine Interreflexion (Gesamt-Renderingzeit $\approx$ 900 ms), rechts drei Interreflexionen ($\approx$ 3,3 sec). Zwei Interreflexionen (nicht dargestellt) $\approx$ 2 sec.

Mit dem implementierten Voxel Path Tracer (Abbildung 50) sind mehr Interreflexionen möglich. Da die Daten entlang eines Pfades in Texturen zwischengespeichert werden, ist die maximale Anzahl der Interreflexionen

[28] Angabe bezieht sich auf NVIDIA-Grafikkarten der GeForce-Serien 8, 9 und 200

durch den Speicherplatz der Grafikkarte beschränkt. Der Rechenaufwand steigt nahezu linear mit der Anzahl der Interreflexionen.

Bildauflösung. Die Rechenzeit hängt linear von der Anzahl der zu beleuchtenden Pixel ab. Eine Verringerung der Ausgabeauflösung beschleunigt somit die Berechnung – im Idealfall um den Faktor, mit dem die Auflösung reduziert wurde. Eine häufige Vorgehensweise ist es, die indirekte Beleuchtung auf halber Auflösung zu berechnen und anschließend mit einem Filter hochzuskalieren. Das linke Bild aus Abbildung 50 mit einer Interreflexion wurde in 900 ms für die Auflösung 768×512 berechnet. Für eine Auflösung mit halb so vielen Pixeln (512×384) sinkt die Zeit auf 410 ms, für eine Auflösung mit doppelt so vielen Pixeln (1024×768) steigt sie auf 1560 ms.

9.3.2 Vergleich mit Referenzbildern

Für das Rendering der Referenzbilder wird der rein CPU-basierte Path Tracer des Frameworks von [Bärz et al. 2010] benutzt. Die Renderingzeiten sind bei den Bildern, die mit den in dieser Diplomarbeit implementierten Verfahren erzeugt wurden, wesentlich kürzer als jene der Referenz. Genaue Zeiten nennen die Abbildungsunterschriften. Es gibt bei den verglichenen Bildern generell nur geringe Unterschiede. Abbildung 51 zeigt ein Beispiel für ein Objekt mit Phong-Oberfläche. Im Gesamteindruck ist das mit dem eigenen Voxel Path Tracer erzeugte Bild heller als die Referenz. In Abbildung 52 ist ein weiteres Beispielobjekt mit Phong-Material im Vergleich zu einem mit Lambert-Material zu sehen. Das mit der eigenen Implementierung gerenderte Bild unterscheidet sich beim Lambert-Material kaum. Das Phong-Objekt ist wie auch in Abbildung 51 etwas heller.

In Abbildung 53 ist erkennbar, dass sich die grobe Voxelstruktur auf die indirekten Schatten auswirkt. Zu sehen ist das „Yeah Right“-Modell[29] (60.000 Dreiecke) in der Sibenik Kathedrale (80.000 Dreiecke). Die direkte Beleuchtung stammt von einem Spotlight. Die in der Diplomarbeit implementierten Verfahren produzieren bei dem gezeigten Objekt, das einen Worst Case darstellt, übermäßig viele und dunkle indirekte Schatten. Der Grund liegt darin, dass die das Objekt repräsentierenden Voxel wesentlich mehr Raum einnehmen als das eigentliche Modell.

[29] Ursprung der Bezeichnung: „yeah, right, we can come up with a good texture atlas for this...“ (Keenan Crane, Urheber des Modells)

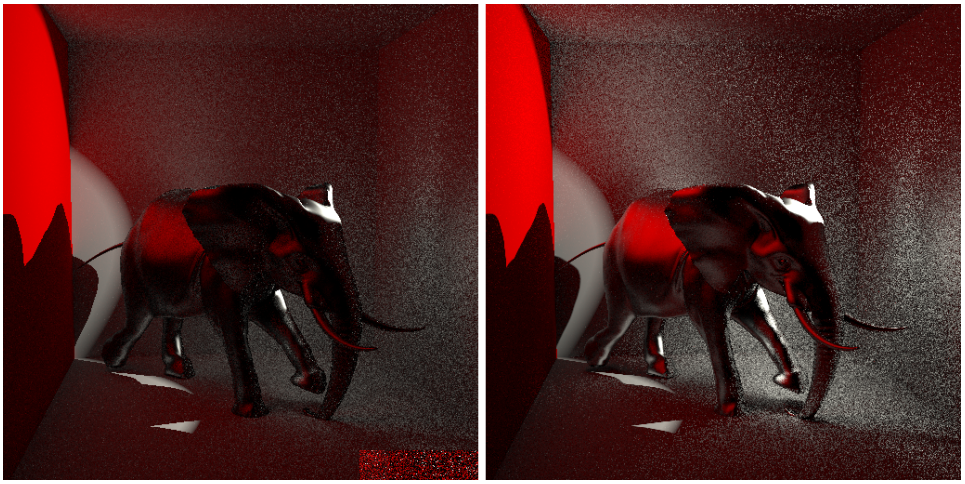

Abbildung 51: Elefant-Modell (ca. 84.000 Dreiecke) mit Phong-Oberfläche (Phong-Exponent = 16) in diffuser Box mit roter Wand links, die von einem Spotlight angestrahlt wird. Zwei Interreflexionen, 192 Strahlen pro Pixel. Bildauflösung: 512×512. Bild links: Referenzbild, Path Tracing (Dauer: 14,4 Minuten). Bild rechts: Eigener Voxel Path Tracer (4 Frames à 48 Strahlen akkumuliert, Dauer: 3 Sekunden), Voxelauflösung 128^3.

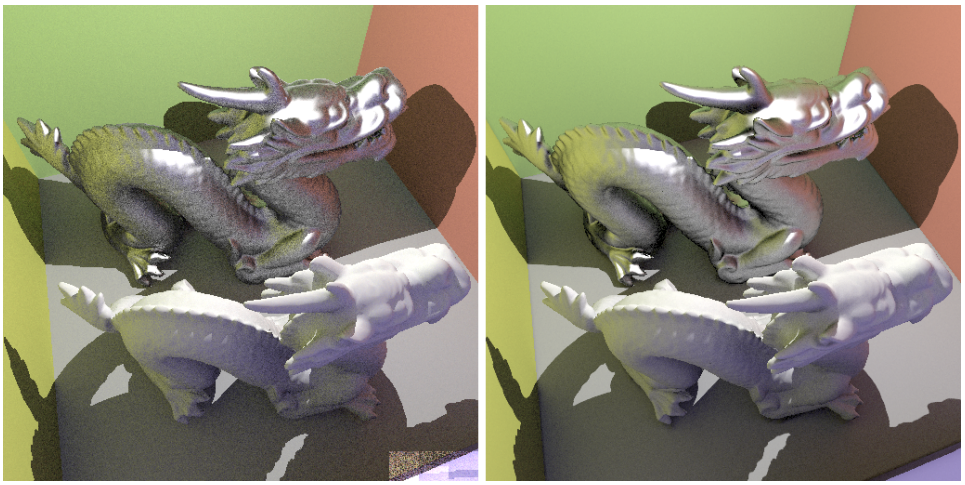

Abbildung 52: Stanford Dragons (je 100.000 Dreiecke) in geschlossenem Würfel (diffuse Wände) mit einer Punktlichtquelle, im Vordergrund mit Lambert-, im Hintergrund mit Phong-Material (Phong-Exponent = 4). Eine Interreflexion, 32 Strahlen pro Pixel. Bildauflösung: 512×512. Bild links: Referenzbild, Path Tracing (Dauer: 3,5 Minuten). Bild rechts: Voxel Path Tracer (Dauer: 310 ms, entspricht 3,2 Frames pro Sekunde). Voxelauflösung: 128^3.

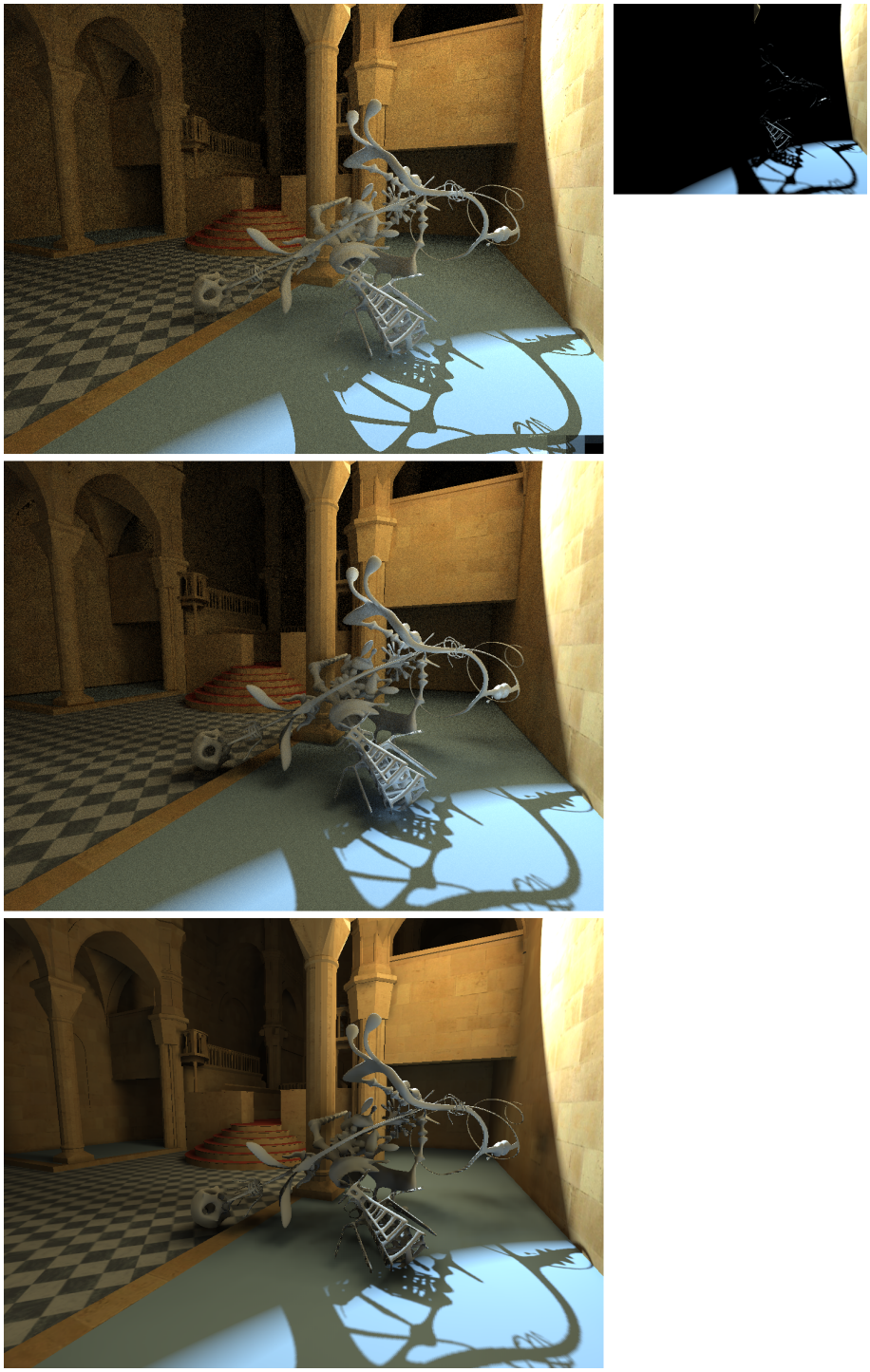

Abbildung 53: Unterschiede bei indirekten Schatten filigraner Modelle. Bildauflösung: 1024×768, zwei Interreflexionen. Rechts oben: Nur direkte Beleuchtung. Von oben nach unten: Referenzbild mit 64 Strahlen pro Pixel: 28 Minuten, Voxel Path Tracer mit 64 Strahlen pro Pixel: 2,2 Sekunden, VPL-Beleuchtung mit 32 VPLs pro Pixel, gefiltert (Filterradius 3, 2 Durchläufe): 204 ms (entspricht 4,9 Frames pro Sekunde). Voxelauflösung jeweils $256^2 \times 128$.

9.3.3 Weitere Beispielbilder

Dieser Abschnitt enthält weitere Ergebnisbilder der implementierten Verfahren. Die indirekte Beleuchtung wurde zu Präsentationszwecken mit Tone Mapping und Kontrasterhöhung verstärkt.

Die Abbildungen 54 und 55 demonstrieren das VPL-Verfahren für diffuse Materialien anhand kleiner Szenen. In Abbildung 55 ist außerdem ein Beispiel für glänzendes Material mit VPLs angeführt. Die Parameter sind jeweils: 24 VPLs, Interleaved Sampling (4×4), mit Filter, Voxelauflösung 128^3 und Bildauflösung 1024×768.

Die Abbildungen 56 und 57 zeigen eine große Szene, die oftmals zum Test globaler Beleuchtungsverfahren benutzt wird: das Sponza Atrium (170.000 Dreiecke). Die beiden Bilder gehen aus unterschiedlicher direkter Beleuchtung hervor. In Abbildung 56 ist ausschließlich indirekte Beleuchtung zu sehen. Daher wird die direkte Beleuchtung aus einer anderen Perspektive gezeigt. Die Voxelauflösung für beide Bilder betrug $256^2 \times 128$.

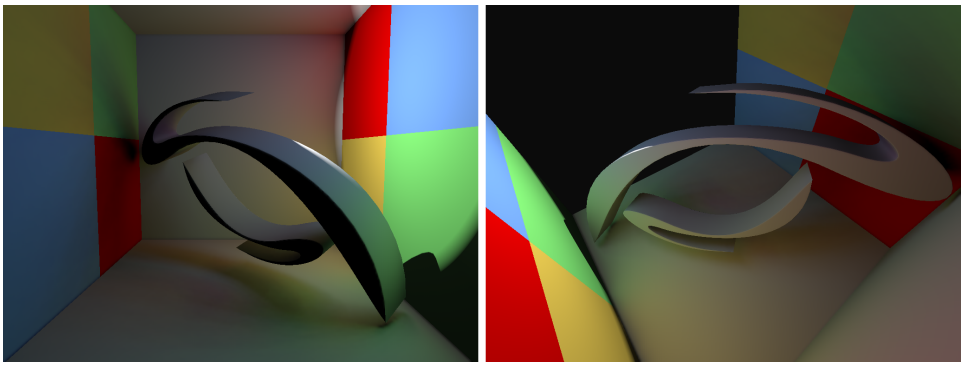

Abbildung 54: CV-Logo in Box (5 Wände), alles diffus, zwei Ansichten der gleichen Szene. Ansicht links: 7,5 Frames pro Sekunde (FPS), Ansicht rechts: 8,3 FPS.

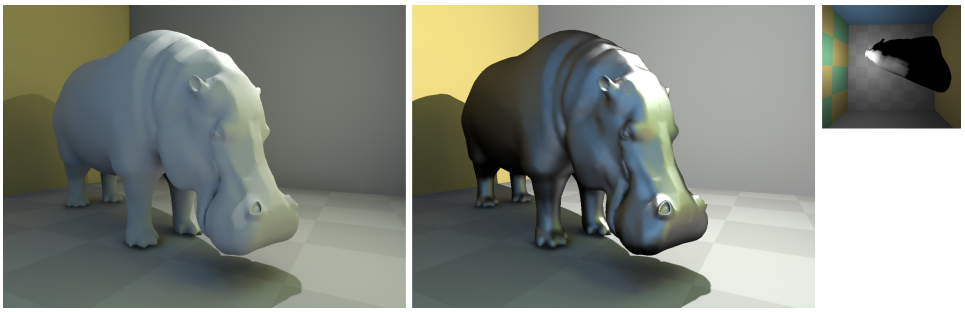

Abbildung 55: Nilpferd-Modell in geschlossenem Würfel (diffus) mit einer Punktlichtquelle. Links: Nilpferd mit Lambert-Oberfläche. Rechts: Phong-Exponent = 6. Beide 7,2 FPS. Oben rechts: direkte Beleuchtung aus anderer Perspektive.

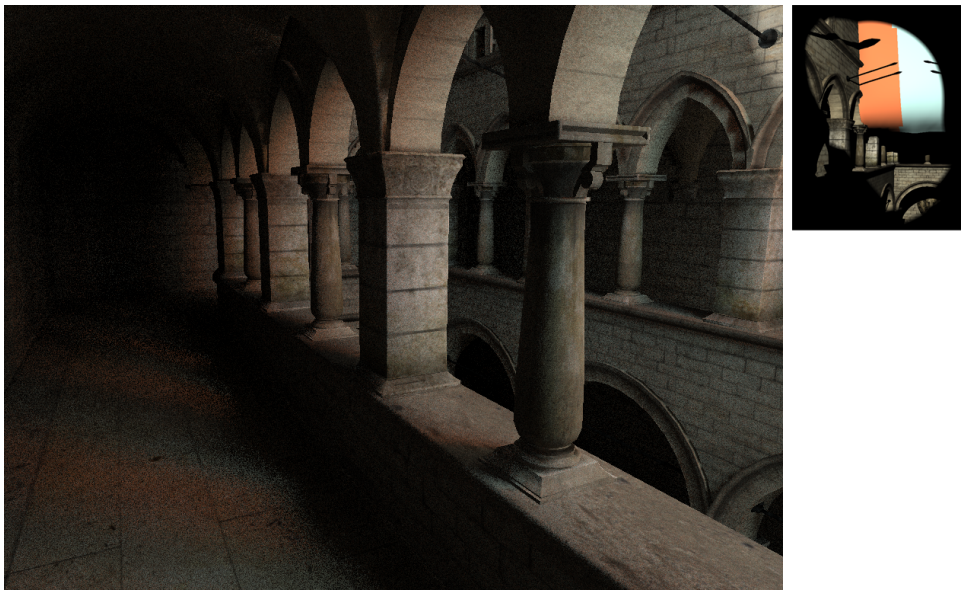

Abbildung 56: Sponza Atrium mit Voxel Path Tracing, zwei Interreflexionen mit 256 Strahlen, berechnet in 10 Sekunden.

Abbildung 57: Sponza Atrium mit VPL-Beleuchtung, 24 VPLs, eine Interreflexion, Interleaved Sampling (4×4), gefiltert, 6,3 Frames pro Sekunde.

10 Fazit und Ausblick

Diese Diplomarbeit hat gezeigt, dass Voxel eine interessante Form der Szenenrepräsentation für Beleuchtungssimulationen auf der GPU darstellen. Da die Daten in einer von der Szenenkamera unabhängigen Repräsentation vorliegen, treten keine Artefakte wie bei Screen-Space-Verfahren auf. Die Beleuchtung ändert sich nicht bei Kamerabewegungen. Die Voxelstrukturen lassen sich effizient nutzen und auch erstellen. Es wurde ein auf Texturatlanten basierendes *Voxelisierungsverfahren* entworfen, das binäre und nicht-binäre Volumen aus 3D-Modellen erzeugen kann. Die Atlas-Voxelisierung erwies sich insbesondere für einzelne dynamische Objekte, die in eine größere statische vorvoxelisierte Umgebung eingefügt werden, als geeignet. Welche Auflösung der Voxelstruktur ausreichend ist, hängt nicht von der Anzahl der Dreiecke in der Szene ab, sondern von deren Detailreichtum und den Anforderungen an die Genauigkeit.

Nach der Voxelisierung ist der Aufwand der Strahlverfolgung zwecks Beleuchtung nur noch von der Voxelauflösung und der Verteilung der Voxel abhängig. Eine Alternative zum uniformen Ray Marching stellt der in dieser Diplomarbeit vorgestellte *Schnitttest* dar, der auf einem Mipmapping der binären Voxeltextur basiert. Er wurde zunächst zur Lösung der Sichtbarkeit zwischen zwei Punkten konzipiert und anschließend auf das Finden des vordesten Schnittpunktes erweitert. Beide Varianten des Schnitttests erwiesen sich als geeigneter als uniformes Ray Marching, wenn große Radien, innerhalb deren sich das Licht ausbreiten kann, gewählt werden. Der Schnitttest wurde in zwei verschiedenen indirekten Beleuchtungsalgorithmen eingesetzt.

Zum einen löst er die Frage der *Sichtbarkeit* bei dem implementierten Verfahren mit *virtuellen Punktlichtquellen*. Ähnlich wie bei den Imperfect Shadow Maps (ISM) von [Ritschel et al. 2008] werfen die VPLs grobe Schatten, die sich durch eine ausreichend hohe Anzahl an VPLs optisch ausgleichen. Der Unterschied zu dem ISM-Verfahren ist, dass dieses für jede VPL eine eigene Shadow Map erzeugt, auf die anschließend für die Auswertung der Sichtbarkeit nur einmal zugegriffen werden muss. Mit der Voxelisierung verhält es sich umgekehrt: Es wird eine einzige Voxelrepräsentation erstellt, auf die anschließend aber für jede Sichtbarkeitsanfrage mehrmals zugegriffen werden muss, da die Voxelstruktur durchquert wird, um die Sichtbarkeit zu ermitteln. Die virtuellen Punktlichtquellen mit dem umgesetzten Sichtbarkeitstest eignen sich für globale Beleuchtung mit interaktiven Frameraten. Je nach Auflösung und Qualitätseinstellungen wurden im Allgemeinen 7-10 Frames pro Sekunde erreicht.

Hemisphären-Sampling mit begrenzter Strahllänge ermöglicht ebenfalls interaktive Frameraten. Mit dem *Voxel Path Tracer* lassen sich qualitativ hochwertige Bilder erzeugen, deren Berechnung um ein Vielfaches schneller ist als die der Referenzlösung (wenige Sekunden im Vergleich zu mehreren Minuten). Ein progressives Verfahren erlaubt es, trotz des hohen Rechenaufwands mit den Objekten in der Szene zu interagieren. Während der Nutzerinteraktion werden nur wenige Strahlen verschossen und nach Aktionsende für die aktuelle Ansicht Bilder mit jeweils verschiedenen Samples akkumuliert.

Weiterentwicklungsmöglichkeiten bestehen vor allem darin, noch vorhandene störende Artefakte zu verringern. Ein Beispiel ist das Flackern, das durch springende VPLs bei direkt beleuchteten bewegten Objekten oder bewegten Lichtquellen auftritt. Hier sind bessere Sampling-Strategien zur Platzierung der VPLs denkbar, zum Beispiel Importance Sampling, das mehr VPLs an hell beleuchtete Stellen setzt. Eine andere Möglichkeit ist eine hierarchische Organisation der VPLs, die [Nichols et al. 2009] vorschlagen. Für jedes Pixel sollen die wichtigsten VPLs bestimmt werden, die es beleuchten. Dadurch ist ein Vielfaches der VPL-Anzahl bei etwa gleichbleibender Performance möglich. Von den gleichen Autoren stammt die Idee, die Beleuchtung auf verschiedenen Auflösungsstufen zu berechnen, um so insgesamt die indirekte Beleuchtung für wesentlich weniger Pixel auswerten zu müssen. Hiermit wurde gegen Ende dieser Arbeit experimentiert. Eine effiziente Implementierung, die die Performance-Vorteile der Multiresolution-Verfahren zum Tragen bringt, konnte in der beschränkten Zeit nicht realisiert werden.

Beim Hemisphären-Sampling ist das Rauschmuster am störendsten. Dieses kann oftmals auch mit stark weichzeichnenden Filtern nicht eliminiert werden, sondern „scheint durch“. Die implementierten Filter sind allesamt räumliche Filter, d. h. benachbarte Pixel werden mit bestimmten Gewichtungsfunktionen gemittelt. Eine vielversprechende Ergänzung ist das Filtern über die Zeit hinweg. Diese sogenannten temporalen Filter (z. B. von [Herzog et al. 2010]) verbessern die Qualität der Ergebnisbilder, indem sie beispielsweise Rauschen oder Aliasing mindern. Zeitlich kohärente Ergebnisse ermöglichen es, Pixelwerte aus vergangenen Frames für den aktuellen Pixelwert wiederzuverwenden. Die Herausforderung liegt darin, zu erkennen, ob die alten Pixelwerte gültige Informationen enthalten und ob interpoliert werden darf. Gleichzeitig muss vermieden werden, dass bewegte Objekte „Schleier“ hinter sich herziehen (sog. Ghosting).

Bei der Literaturrecherche zeigte sich, dass aktuelle Veröffentlichungen im Bereich angenäherter Beleuchtung zunehmend Voxel als Hilfsmittel einsetzten. Voxelrepräsentationen werden daher voraussichtlich auch in der Zukunft ein für Lichtberechnungen relevantes Themengebiet darstellen.

Literaturverzeichnis

[**Bresink 1999**] Bresink, Marcel: *Diskrete Raumunterteilung zur globalen Beleuchtungsberechnung in der digitalen Bildsynthese*, Universität Koblenz-Landau, Dissertation, 1999

[**Bärz et al. 2010**] Bärz, Jakob ; Henrich, Niklas ; Müller, Stefan: Validating Photometric and Colorimetric Consistency of Physically-Based Image Synthesis. In: *5th European Conference on Colour in Graphics, Imaging and Vision (CGIV 2010)*, 2010

[**Cohen et al. 1993**] Cohen, Michael F. ; Wallace, John ; Hanrahan, Pat: *Radiosity and Realistic Image Synthesis*. 1993

[**Cohen-Or und Kaufman 1997**] Cohen-Or, Daniel ; Kaufman, Arie: 3D Line Voxelization and Connectivity Control. In: *IEEE Computer Graphics and Applications* 17 (1997), Nr. 9, S. 80–87

[**Crane et al. 2007**] Crane, Keenan ; Llamas, Ignacio ; Tariq, Sarah: *GPU Gems 3: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Kapitel 30 Real-Time Simulation and Rendering of 3D Fluids, S. 633–675, Addison-Wesley Professional, 2007. – Online unter http://developer.download.nvidia.com/books/gpu_gems_3/samples/gems3_ch30.pdf (letzter Zugriff am 04.04.2010)

[**Dachsbacher und Stamminger 2005**] Dachsbacher, Carsten ; Stamminger, Marc: Reflective Shadow Maps. In: *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM SIGGRAPH, 2005. – Online unter http://www.vis.uni-stuttgart.de/~dachsbcn/download/rsm.pdf (letzter Zugriff am 09.08.2010)

[**Dachsbacher und Stamminger 2006**] Dachsbacher, Carsten ; Stamminger, Marc: Splatting Indirect Illumination. In: *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM, 2006. – Online unter http://www.vis.uni-stuttgart.de/~dachsbcn/download/sii.pdf (letzter Zugriff am 09.08.2010)

[**Dong et al. 2004**] Dong, Zhao ; Chen, Wei ; Bao, Hujun ; Zhang, Hongxin ; Peng, Qunsheng: Real-time Voxelization for Complex Polygonal Models. In: *Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA : IEEE Computer Society, 2004, S. 43–50. – Online unter http://www.mpi-inf.mpg.de/~dong/download/PG04.pdf (letzter Zugriff am 04.04.2010)

[**Drago et al. 2003**] DRAGO, Frederic ; MYSZKOWSKI, Karol ; ANNEN, Thomas ; CHIBA, Norishige: Adaptive Logarithmic Mapping For Displaying High Contrast Scenes. In: BRUNET, Pere (Hrsg.) ; FELLNER, Dieter W. (Hrsg.): *The European Association for Computer Graphics 24th Annual Conference: EUROGRAPHICS 2003* Bd. 22(3). Granada, Spain : Blackwell, 2003, S. 419–426. – Online unter http://www.mpi-inf.mpg.de/resources/tmo/logmap/logmap.pdf (letzter Zugriff am 17.09.2009)

[**Dutré 2003**] DUTRÉ, Philip: *Global Illumination Compendium*. Computer Graphics, Department of Computer Science, Katholieke Universiteit Leuven. September 2003. – Online unter http://people.cs.kuleuven.be/~philip.dutre/GI/TotalCompendium.pdf (letzter Zugriff am 04.08.2010)

[**Eisemann und Décoret 2006**] EISEMANN, Elmar ; DÉCORET, Xavier: Fast Scene Voxelization and Applications. In: *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM SIGGRAPH, 2006, S. 71–78. – Online unter http://artis.imag.fr/Publications/2006/ED06 (letzter Zugriff am 19.07.2010)

[**Eisemann und Décoret 2008**] EISEMANN, Elmar ; DÉCORET, Xavier: Single-pass GPU Solid Voxelization and Applications. In: *GI '08: Proceedings of Graphics Interface 2008* Bd. 322, Canadian Information Processing Society, 2008, S. 73–80. – Online unter http://artis.imag.fr/Publications/2008/ED08a (letzter Zugriff am 19.07.2010)

[**Engel et al. 2006**] ENGEL, Klaus ; HADWIGER, Markus ; KNISS, Joe M. ; REZK-SALAMA, Christof ; WEISKOPF, Daniel: *Real-Time Volume Graphics*. A. K. Peters, Ltd., 2006

[**Fang und Chen 2000**] FANG, Shiaofen ; CHEN, Hongsheng: Hardware Accelerated Voxelization. In: *Computers and Graphics* 24 (2000). – Online unter http://www.cs.iupui.edu/~sfang/vg99.pdf (letzter Zugriff am 19.07.2010)

[**Forest et al. 2009**] FOREST, Vincent ; BARTHE, Loic ; PAULIN, Mathias: Real-Time Hierarchical Binary-Scene Voxelization. In: *Journal of Graphics, GPU, & Game Tools* 14 (2009), S. 21–34

[**Greger 1996**] GREGER, Gene: *The Irradiance Volume*, Cornell University, Diplomarbeit, August 1996. – Online unter http://www.gene.greger-weltin.org/professional/publications/thesis.pdf (letzter Zugriff am 04.04.2010)

[**Hasselgren et al. 2005**] HASSELGREN, Jon ; AKENINE-MÖLLER, Tomas ; OHLSSON, Lennart: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Kapi-

tel 42 Conservative Rasterization, Addison-Wesley Professional, 2005. – Online unter http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter42.html (letzter Zugriff am 04.06.2010)

[**Herzog et al. 2010**] Herzog, Robert ; Eisemann, Elmar ; Myszkowski, Karol ; Seidel, H.-P.: Spatio-Temporal Upsampling on the GPU. In: *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, 2010. – Online unter http://www.mpi-inf.mpg.de/~rherzog/Papers/spatioTemporalUpsampling_preprintI3D2010.pdf (letzter Zugriff am 17.08.2010)

[**Immel et al. 1986**] Immel, David S. ; Cohen, Michael F. ; Greenberg, Donald P.: A radiosity method for non-diffuse environments. In: *SIGGRAPH Computer Graphics* 20 (1986), Nr. 4, S. 133–142

[**Kajiya 1986**] Kajiya, James T.: The rendering equation. In: *SIGGRAPH Compuer Graphics* 20 (1986), Nr. 4, S. 143–150

[**Kaplanyan 2009**] Kaplanyan, Anton: *Light Propagation Volumes in CryEngine 3*. ACM SIGGRAPH 2009 Courses – Advances in Real-Time Rendering in 3D Graphics and Games Course. 2009. – Online unter http://www.crytek.com/fileadmin/user_upload/inside/presentations/2009/Light_Propagation_Volumes.pdf (letzter Zugriff am 04.04.2010)

[**Kaplanyan und Dachsbacher 2010**] Kaplanyan, Anton ; Dachsbacher, Carsten: Cascaded Light Propagation Volumes for Real-Time Indirect Illumination. In: *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, Online unter http://www.vis.uni-stuttgart.de/~dachsbcn/download/lpv.pdf (letzter Zugriff am 04.04.2010), 2010. – Präsentation online unter http://www.crytek.com/sites/default/files/2010-I3D_CLPV.ppt (letzter Zugriff am 15.08.2010)

[**Kay und Kajiya 1986**] Kay, Timothy L. ; Kajiya, James T.: Ray Tracing Complex Scenes. In: *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM, 1986

[**Keller 1997**] Keller, Alexander: Instant Radiosity. In: *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 1997

[**Kilgariff und Fernando 2005**] Kilgariff, Emmett ; Fernando, Randima: *GPU Gems 2: Programming Techniques for High-Performance*

Graphics and General-Purpose Computation. Kapitel 30 The GeForce 6 Series GPU Architecture, S. 471–491, Addison-Wesley Professional, 2005. – Online unter http://http.download.nvidia.com/developer/GPU_Gems_2/GPU_Gems2_ch30.pdf (letzter Zugriff am 11.08.2010)

[**Li et al. 2005**] LI, Wei ; FAN, Zhe ; WEI, Xiaoming ; KAUFMAN, Arie: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Kapitel 47 Flow Simulation with Complex Boundaries, Addison-Wesley Professional, 2005. – Online unter http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter47.html (letzter Zugriff am 04.06.2010)

[**Malgouyres 2002**] MALGOUYRES, Rémy: A Discrete Radiosity Method. In: *DGCI '02: Proceedings of the 10th International Conference on Discrete Geometry for Computer Imagery*. London, UK : Springer-Verlag, 2002, S. 428–438

[**Moore und Jefferies 2009**] MOORE, Jeremy ; JEFFERIES, David: *Rendering Techniques in Split/Second*. Advances in Real-Time Rendering in 3D Graphics and Games Course – SIGGRAPH. 2009. – Online unter http://www2.disney.co.uk/cms_res/blackrockstudio/pdf/Rendering_Techniques_in_SplitSecond.pdf (letzter Zugriff am 19.07.2010)

[**Nichols 2010**] NICHOLS, Greg: *Multiresolution Image-Space Rendering for Interactive Global Illumination*, University of Iowa, Dissertation, 2010. – Online unter http://www.gregnichols.org/pubs/thesis.pdf (letzter Zugriff am 11.08.2010)

[**Nichols et al. 2010**] NICHOLS, Greg ; PENMATSA, Rajeev ; WYMAN, Chris: Interactive, Multiresolution Image-Space Rendering for Dynamic Area Lighting. In: *Eurographics Symposium on Rendering* 29 (2010), Nr. 4. – Online unter http://www.cs.uiowa.edu/~cwyman/publications/files/techreports/UICS-TR-10-01.pdf (letzter Zugriff am 19.07.2010)

[**Nichols et al. 2009**] NICHOLS, Greg ; SHOPF, Jeremy ; WYMAN, Chris: Hierarchical Image-Space Radiosity for Interactive Global Illumination. In: *Computer Graphics Forum* 28 (2009), Nr. 4, S. 1141–1149. – Online unter http://www.cs.uiowa.edu/~cwyman/publications/files/imgSpRadiosity/egsr09_imgSpRadiosity.small.pdf (letzter Zugriff am 11.08.2010)

[**Nichols und Wyman 2009**] NICHOLS, Greg ; WYMAN, Chris: Multiresolution Splatting for Indirect Illumination. In: *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, 2009. – Online unter http://www.cs.uiowa.edu/~cwyman/publications/files/

multiResSplat4Indirect/multiResolutionSplatting.pdf (letzter Zugriff am 11.08.2010)

[**Nijasure et al. 2005**] Nijasure, Mangesh ; Pattanaik, Sumanta N. ; Goel, Vineet: Real-Time Global Illumination on GPUs. In: *Journal of Graphics, GPU, & Game Tools* 10 (2005), Nr. 2, S. 55–71. – Online unter http://www.cs.ucf.edu/~ceh/Publications/Papers/Rendering/JGT05NijasurePattanaikGoel.pdf (letzter Zugriff am 04.04.2010)

[**NVIDIA 2008**] NVIDIA: *GPU Programming Guide GeForce 8 and 9 Series*, Dezember 2008. – Online unter http://developer.download.nvidia.com/GPU_Programming_Guide/GPU_Programming_Guide_G80.pdf (letzter Zugriff am 07.08.2010)

[**Oat 2006**] Oat, Chris: Irradiance Volumes for Real-Time Rendering. In: Engel, Wolfgang (Hrsg.): *ShaderX5: Advanced Rendering Techniques*. Charles River Media, 2006. – Präsentation (Irradiance Volumes for Games) online unter http://developer.amd.com/media/gpu_assets/GDC2005_PracticalPRT.pdf (letzter Zugriff am 04.04.2010)

[**Papaioannou et al. 2010**] Papaioannou, Georgios ; Menexi, Maria L. ; Papadopoulos, Charilaos: Real-Time Volume-Based Ambient Occlusion. In: *IEEE Transactions on Visualization and Computer Graphics* 16 (2010), S. 752–762

[**Passalis et al. 2007**] Passalis, Georgios ; Theoharis, Theoharis ; Toderici, George ; Kakadiaris, Ioannis A.: General Voxelization Algorithm with Scalable GPU Implementation. In: *Journal of Graphics, GPU, & Game Tools* 12 (2007), S. 61–71

[**Pharr und Humphreys 2004**] Pharr, Matt ; Humphreys, Greg: *Physically Based Rendering: From Theory to Implementation*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2004

[**Reinbothe et al. 2009**] Reinbothe, Christoph ; Boubekeur, Tamy ; Alexa, Marc: Hybrid Ambient Occlusion. In: *EUROGRAPHICS 2009 Areas Papers* (2009). – Online unter http://perso.telecom-paristech.fr/~boubek/papers/HAO/HAO.pdf (letzter Zugriff am 19.07.2010)

[**Ritschel et al. 2008**] Ritschel, Tobias ; Grosch, Thorsten ; Kim, Min H. ; Seidel, Hans-Peter ; Dachsbacher, Carsten ; Kautz, Jan: Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. In: *ACM Trans. Graph. (Proc. of SIGGRAPH ASIA 2008)* 27 (2008). – Online unter http://www.mpi-inf.mpg.de/resources/ImperfectShadowMaps/ (letzter Zugriff am 17.06.2010)

[**Segovia et al. 2006**] SEGOVIA, B. ; IEHL, J. C. ; MITANCHEY, R. ; PÉROCHE, B.: Non-interleaved Deferred Shading of Interleaved Sample Patterns. In: *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*. New York, NY, USA : ACM, 2006, S. 53–60

[**Thiedemann 2009**] THIEDEMANN, Sinje: *Globale Beleuchtung im Bildraum unter besonderer Berücksichtigung der Sichtbarkeitsbestimmung*. Studienarbeit Universität Koblenz-Landau. 2009. – Online unter http://kola.opus.hbz-nrw.de/volltexte/2009/464/pdf/SA_thiedemann.pdf (letzter Zugriff am 18.07.2010)

[**Zhang et al. 2007**] ZHANG, Long ; CHEN, Wei ; EBERT, David S. ; PENG, Qunsheng: Conservative Voxelization. In: *The Visual Computer* 23 (2007), S. 783–792. – Präsentation online unter http://www.cad.zju.edu.cn/home/chenwei/research/CGI2007.ppt (letzter Zugriff am 13.08.2010)

Quellenverzeichnis der 3D-Modelle

Sibenik Kathedrale Marko Dabrovic: http://hdri.cgtechniques.com/~sibenik2/download/ (letzter Zugriff am 21.08.2010)

Animationssequenzen Pferd, Elefant, Kamel: Robert W. Sumner und Jovan Popovic, Mesh Data from Deformation Transfer for Triangle Meshes: http://people.csail.mit.edu/sumner/research/deftransfer/data.html (letzter Zugriff am 21.08.2010)

Bunny, Dragon Stanford 3D Scanning Repository: http://graphics.stanford.edu/data/3Dscanrep/ (letzter Zugriff am 21.08.2010)

„Yeah Right"-Modell Keenan Crane, 3D Model Repository: http://www.cs.caltech.edu/~keenan/models.html (letzter Zugriff am 25.08.2010)

Hippo Espona, Suggestive Contour Gallery: http://www.cs.princeton.edu/gfx/proj/sugcon/models/hippo.ply (letzter Zugriff am 27.08.2010)

Sponza Crytek: http://crytek.com/cryengine/cryengine3/downloads (letzter Zugriff am 26.08.2010)