Universität Koblenz-Landau
Fachbereich 4: Informatik

RADIOCITY

Studienarbeit: Konzeption, Entwicklung und Implementation eines hierarchischen Radiosity - Systems mit Clustering
SS 2003

Guido Stegmann
djgid@groovalistic.de
Römerstr. 74
56073 Koblenz
Deutschland

14.10.2003, Koblenz

Betreuer: Dipl.-Inform. Thorsten Grosch

Prüfer: Prof. Dr.-Ing. Stefan Müller

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Historie . . . 3
1.2 Die Radiosity Idee . . . 4
1.3 Die Radiositiy-Gleichung . . . 5

2 Das hierarchische Radiosity Verfahren (HR) **6**
2.1 Vorraussetzungen . . . 6
2.2 Linking . . . 7
2.3 Der Ablauf des Verfahrens . . . 8
2.4 Der Prisma-Formfaktor . . . 9
2.4.1 Minimaler und Maximaler Formfaktor . . . 10
2.5 Visibilität . . . 14
2.6 Support Plane Split . . . 16
2.7 Refinement . . . 18
2.7.1 Refinementkriterium . . . 18
2.7.2 Verfeinern der Links . . . 21
2.7.3 Die *refine()* Methode . . . 22
2.8 Radiosityausgleich . . . 23
2.8.1 Die *gather()* Methode . . . 23
2.8.2 Die *pushpull()* Methode . . . 23
2.9 Darstellung der Patches . . . 25

3 Hierarchisches Radiosity mit Clustering (HRC) **25**
3.1 Die Idee . . . 25
3.2 Der Ablauf des HRC - Verfahrens . . . 28
3.3 Beta-Link . . . 29
3.4 Alpha-Link . . . 30
3.5 Refinementkriterien . . . 31
3.5.1 Besonderheiten beim Linking . . . 31
3.5.2 Die *refine()*-Methode . . . 32
3.6 Radiosityausgleich . . . 33
3.6.1 Die *gather()*-Methode . . . 33
3.6.2 Die *pushpull()*-Methode . . . 34

4 Spezielle Lichtquellen **35**
4.1 Pointlight . . . 36
4.2 Spotlight . . . 37

5 Implementierung **41**
5.1 Aufbau der Klassen . . . 42
5.1.1 Aufbau von ***Scene*** . . . 42
5.1.2 Aufbau von ***Cluster*** . . . 42

5.1.3 Erweiterung der Klasse ***Patch*** 42
5.1.4 Aufbau von ***BaseLink*** 43
5.1.5 Aufbau von ***Tools*** 43
5.2 Parameter 43
5.3 Tastatur-Shortcuts 44

6 Testergebnisse und Bewertung 45

7 Ausblick 58

1 Einleitung

1.1 Historie

Die Computergraphik, eines der jüngeren Teilgebiete der Informatik, erlangte in den letzten Jahrzehnten immer mehr an Bedeutung, insbesondere die computergestütze realitätsnahe Darstellung von dreidimensionalen Szenen, das sogennante photorealistische Rendering[1]. Bedingt durch den technischen Fortschritt in der Computertechnologie, wurde eine grosse Zeiteinsparung bei der Erzeugung photorealistischer Darstellungen erreicht. Je nach Komplexität der Szene, brauchen allerdings auch heute noch Rechner der neusten Generation eher Stunden als Minuten, bei entsprechender Genauigkeit der modellierten Effekte sogar Tage, wobei die Detailgenauigkeit schon um einiges höher als damals ist.

Mathematiker, Künstler, Architekten und Informatiker sind ständig mit der Entwicklung von computergenerierten photorealistischen 3D Bildern beschäftigt, dadurch dass die Rechnerleistung immer weiter zunahm, konnten komplexere Algorithmen entwickelt werden, die einen höheren Grad an Genauigkeit zuliessen. In den 60er Jahren konnten erste Erfolge mit der Ablösung der reinen Meshdarstellungen[2], die keinerlei Informationen von Farbe, Lichteffekten usw. enthielten, erzielt werden. Die Liniendarstellung wurde durch einfaches Füllen der Flächen mit Objektfarben, die in Abhängigkeit der Lichteinstrahlung unterschiedliche Helligkeiten besaßen, abgelöst. Die Ergebnisse sahen allerdings sehr grob aus, man erkannte noch jedes Flächenelement, und die Darstellung war nicht sehr wirklichkeitsnah. 1971 entwickelte Gouraud[Gou71] und 1975 auch Phong [Pho75] Schattierungsalgorithmen, die Farbverläufe auf Flächen interpolierten. Diese Effekte werden noch heute in vielen kommerziellen Renderer genutzt.

Der erste Raytracer [3] wurde 1968 [App68] konzipiert, der es ermöglichte Schatten darzustellen. Raytracing gehört zu dem am weitesten verbreiteste Verfahren. Die Lichtausbreitung beim Raytracer wird durch die Verfolgung einzelner diskreter Lichtstrahlen, die unter der Berücksichtigung der Gesetze für Spiegelung und Brechung, vom Betrachterauge über jedes Pixel der Bildschirmebene verschossen werden.

Das Radiosity-Verfahren ist wesentlich jünger und wurde erstmals 1984 von Goral *et al.*[GTGB84] vorgestellt. Das aus der Wärmestrahlungs-Simulation stammende Radiosity Verfahren behandelt im Gegenzug zu Raytracing den Energieaustausch zwischen Flächenelemente. Das Radiosity Verfahren ist in neuste Versionen einiger kommerzieller 3D Renderer integriert worden.

Beide Verfahren haben Schwächen und Stärken, Raytracing ist für die Berechnung vom direkten Licht besser, liefert aber kein indirektes Licht. Das Radiosity Verfahren hingegen liefert sehr gute Ergebnisse für die diffuse Lichtverteilung. Eine

[1] Rendering = automatische Erzeugung photorealistischer Bilder mittels eines Computers
[2] Mesh = Drahtgitter
[3] Raytracer = Strahlverfolgung

Kombination aus beiden Verfahren ist unter dem Namen Final Gathering bekannt.

Über die Zeit wurden einige verschiede Varianten des Radiosity Verfahrens entwickelt: Die Full Matrix Methode, Progressive Refinement, Hierarchisches Radiosity, Hierarchisches Radiosity mit Clustering, Face Cluster Radiosity, Monte Carlo Radiosity, Wavelet Radiosity. Die heutigen Verfahren liefern schon nahezu perfekt gerenderte Computerbilder, die es dem Betrachter nicht leicht machen von der Realität zu unterscheiden. Echtzeit Rendering liefert heutzutage auch schon sehr beeindruckende Ergebnisse, wie man es bei Computersimulationen, Virtual-Reality-Systemen und Computerspielen beobachten kann. Jedoch stößt man hier sehr schnell an die Grenze der Rechnerleistung und man muss weiterhin Abstriche in der Detailauflösung machen. Zukünftige Rechnergenerationen und Algorithmen werden sicherlich in der Lage sein, Computerwelten, die einen absolut glaubhaften Photorealismus entsprechen, in Echtzeit zu berechnen und auch darstellen zu können.

Das 2003 an der Universität Koblenz entwickelte Programm RADIOCITY ist ein hierarisches Radiositysystem mit Clustering, das in Rahmen dieser Studienarbeit und den Studienarbeiten von Dominik Rau (Meshing) und Florian Koller (Framework und photometrische Konsistenz) konzipiert und implementiert wurde. Der Printversion der Studienarbeit liegt eine CD mit der Software bei.

1.2 Die Radiosity Idee

Da das populäre Raytraycing bei einigen physikalischen Phenomenen doch Schwächen aufzeigte, begannen Forscher nach Alternativen zu suchen. Die indirekte Beleuchtung ist mit Raytracing gar nicht möglich. Mit Radiosity wurde ein, in gewisser Weise komplementäres Beleuchtungsverfahren vorgestellt. Das Radiosity Verfahren beruht auf der Wärmestrahlungs-Simulation, somit hat dieses Verfahren auch Auswirkungen von Lichtemission und Lichtreflexion an diffusen Objekten. Frei übersetzt würde Radiosity Lichtenergiekalkül bedeuten.

Durch Interreflexionen, dem sogenannten Color Bleeding und die Darstellunsmöglichkeit von Halbschatten liefert Radiosity eindeutig realitätsnähere Ergebnisse. Das Radiosity Verfahren ist ein *global illumination model* und befaßt sich mit der Simulation von diffus reflektierten und interreflektierten Lichts, sogenannte Lambert-Reflektoren. Dieses indirekte Licht verändert das Beleuchtungsergebniss nicht unerheblich. Beim Raytracing wird dieser Anteil allerdings nur durch einen konstanten Wert, dem ambienten Term abgeschätzt, was einen doch deutlichen Unterschied zum Radiosity Verfahren ausmacht.

Für das Radiosity-Verfahren wird als Eingabe eine 3D Szene benötigt, die in Flächenelemente unterteilt ist, den sogenannten Patches, einige dieser Patches bekommen eine Eigenemission zugeteilt, diese repräsentieren die Lichtquellen, die sogenannten Lambertemitter (Flächenlicht). Abhängig von der Oberflächeneigenschaft der Patches wird ein Teil der Energie absorbiert und ein Teil in die Umgebung reflektiert, alle Flächen geben die Energie in alle Richtungen gleichmäßig ab.

Im nächsten Schritt wird die reflektierte Energie betrachtet. Jedes Flächenelement wird somit wieder als Lichtquelle angesehen. Schritt für Schritt wird die Energiegrundmenge nun abgebaut bis sie vollständig absorbiert ist. Die Anzahl der nötigen Iterationen ist von der Szene abhängig, es ist aber nicht nötig das Verfahren solange laufen zu lassen, bis die gesamte Energie abgebaut wurde, denn je mehr Schritte gegangen werden, desto kleiner sind auch die ausgetauschten Energien. Am Ende sind dann Unterschiede kaum noch zu erkennen.

Ein weiterer Vorteil des Radiosity Verfahrens ist, das die gerenderte Szene blickpunktunabhängig ist, da ausschließlich diffuses Licht behandelt wurde. Somit können mit Radiosity gerenderte 3D Szenen aus allen Blinkwinkeln betrachtet werden und finden Anklang bei VR-Systemen[4] und in der Spieletechnik.

1.3 Die Radiositiy-Gleichung

Als Ausgangsbasis benötigt das Radiosity Verfahren eine 3D Szenenbeschreibung, die aus Flächenelementen, den sogenannten Patches aufgebaut ist. Diese Patches sind rein diffus reflektierend, undurchsichtig und planar. Desweiteren wird davon ausgegangen, dass sie sich im Vakuum befinden. Der gegenseitige Energieaustausch der Patches kann nun ausgewertet werden. Der **Formfaktor** F_{se} beschreibt den Energieanteil beim ankommenden Empfängerpatch, der vom Senderpatch versendeten Energie und ist abhängig von ihrer Orientierung (Aus- und Eintrittswinkel), den Flächengrößen und den Abstand zueinander:

$$F_{se} = \frac{1}{A_s} \int_{A_s} \int_{A_e} \frac{\cos\theta_e \cdot \cos\theta_s}{\pi d^2} dA_e dA_s \tag{1}$$

s bezeichnet den Sender und *e* entspricht dem Empfänger. Die exakte Berechnung des Doppelintegrals stellt sich als ziemlich schwierig heraus, deshalb wurde nach alternativen Berechnungsmethoden gesucht, die dieses Integral ausreichend genau annähern, in unserem System RADIOCITY verwenden wir den Prisma Formfaktor (siehe Kapitel 2.4). Die beim Empfänger ankommende Radiosity B_e[5] beim Versenden von Energie von allen Senderpatches zum Empfängerpatch kann über die **Radiosity-Gleichung** berechnet werden:

$$B_e = E_e + \rho_e \sum_{s=1}^{n} B_s \cdot F_{se} \tag{2}$$

B_e ist die Radiosity, welche beim Empfängerpatch ankommt. B_e wird von diesem Patch bei der nächsten Iteration dann wieder versenden kann. Sie berechnet sich aus der Eigenemission E_e, dem Reflexionsgrad des Empfängers ρ_e und die mit den Formfaktoren F_{se} von allen Sendern zum Empfänger gewichtete Radiosity B_s des entsprechenden Senders.

[4] Virtual Reality Systeme

[5] Radiosity = spez. Lichtaustrahlung $[\frac{lm}{m^2}]$

Diese Radiosity-Gleichung lässt sich in eine Matrixform umformen, um nun das Gleichungs-System zu lösen, müssen im Vorfeld alle Formfaktoren bekannt sein. Bei kleinen Szenen mit vielleicht nur 10000 Patches müssen schon 100 Millionen Formfaktoren berechnet werden, womit mit heutigen Arbeitsspeicherkapazitäten schon schnell die Grenzen erreicht sind. Die sogenannte Full Matrix Methode hatte keine Zukunft, bessere Verfahren mussten her. 1988 entwickelten daher Cohen *et al.*[CCWG88] das Progressive Refinement Verfahren, hierbei wurde der Sender mit der grössten Energie ausgewählt, die an alle Empfänger verteilt wurde, bis die zu versendende Energie unterhalb einer Abbruchsschranke lag. Es wurden schon wesentlich weniger Formfaktoren berechnet, doch trat öfter der Fall ein, dass ein Patch mehrmals die maximal zu versendende Energie hatte, und somit der Formfaktor nochmals berechnet werden musste. Eine weitere Verbesserung entwickelten 1990/91 Hanrahan *et al.*[HSA91], **das hierarchische Radiosity Verfahren**, dies ist Grundbestandteil des Radiosity Kerns des entwickelten Programms RADIOCITY. Das hierarische Radiosity Verfahren wird im nächsten Kapitel genauer erläutert.

2 Das hierarchische Radiosity Verfahren (HR)

Die aufwendigsten Bestandteile einer Radiosity Berechnung sind die Bestimmungen der Formfaktoren und der Visibilität (siehe Kapitel 2.5). Nach einfachen Abschätzungen müssen $n^2/2$ Formfaktoren berechnet werden (Formfaktoren sind reziprok, d.h es gilt $A_i F_{ij} = A_j F_{ji}$). Desweiteren ist die Berechnung des Formfaktors überflüssig, wenn sich die beiden Flächen nicht sehen können, d.h. das Empfängerpatch liegt nicht im vorderen Halbraum vom Senderpatch bzw. das Senderpatch liegt nicht im vorderen Halbraum vom Empfängerpatch, der Formfaktor wäre in diesen Fällen gleich null. Wenn die beiden Patches sehr weit voneinander entfernt sind, wäre es sinnvoll auf die Formfaktorberechnung zu verzichten, da dieser nahezu null wird. Jedoch liegt i.A. die Anzahl der Formfaktorberechnungen bei einem Aufwand von $O(n^2)$. Das hierarchische Radiosity Verfahren, dass von Hanrahan *et al.*[HSA91] vorgestellt wurde, bietet einen grundlegenden Ansatz zur Verringerung der Formfaktorberechnungen, da die Anzahl der Interaktionen zwischen den Patches erheblich reduziert wird.

2.1 Vorraussetzungen

Hierarchische Radiosity Verfahren besitzen eine adaptive Unterteilungsstrategie, d.h. die zu berechnende Szene besteht Anfangs nur aus Rootpatches[6], die bei Bedarf in Subpatches unterteilt werden, somit erhält die Szene eine hierarchische Struktur (siehe Abbildung 1). Ein Refinementkriterium (siehe Kapitel 2.7), das sogenannte „Orakel“ entscheidet, ob unterteilt werden soll und ob Sender oder Empfänger zu unterteilen sind.

[6]Mit Rootpatch bezeichnet man die initialen Patches der Szene

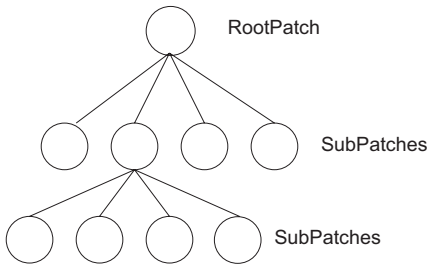


Abbildung 1: Hierarchische Baumstruktur eines Rootpatches

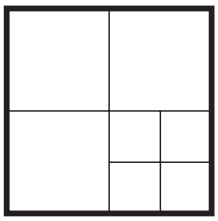


Abbildung 2: Hierarchische Patchstruktur des Rootpatches aus Abb.1

Durch die hierarchische Struktur kann nun Energie auf verschiedenen Level ausgetauscht werden, um festzuhalten, wer mit wem Energie auszutauschen hat, werden Links benötigt. Die Links verbinden Sender mit Empfängern verschiedener Level, siehe Abb.3.

2.2 Linking

Wie vorab erwähnt halten die Links fest, wer mit wem Energie auszutauschen hat. Um die initialen Links anzulegen wird allerdings ein Vorab-Prozeß benötigt, d.h. es müssen zwischen allen sich gegenseitigen sichtbaren Rootpatches Links generiert werden. Dies kann teuer werden, da hier der Aufwand $O(n^2)$ entspricht. Das initiale Mesh sollte damit möglichst wenige Rootpatches enthalten, um den Aufwand des Initial-Linking zu reduzieren, da feinere Patches durch die adaptive Unterteilung genau an den Stellen, wo es das Orakel für nötig empfindet, garantiert sind.

Dieser enorme Zeit und Rechenaufwand des Initial-Linkings, fällt beim hierarchischen Radiosity mit Clustering komplett weg, da hier kein initiales Linking verwendet wird, siehe dazu Kapitel 3.

In der nächsten Stufe werden alle Links abgearbeitet, wenn nun ein Empfängerpatch unterteilt werden soll, wird der zugehörige Link verfeinert, d.h. der Link von Sender zu Empfänger wird gelöscht und neue Links von Sender zu den Kinderpatches werden generiert, genauso wird verfahren, wenn der Sender unterteilt wird. Wie man in Abb.3 sieht, hat diese Miniszene nur 13 Links und besteht immerhin aus 12 Blättern. Würde man genau die gleiche Szene, bestehend aus den 12 Blättern

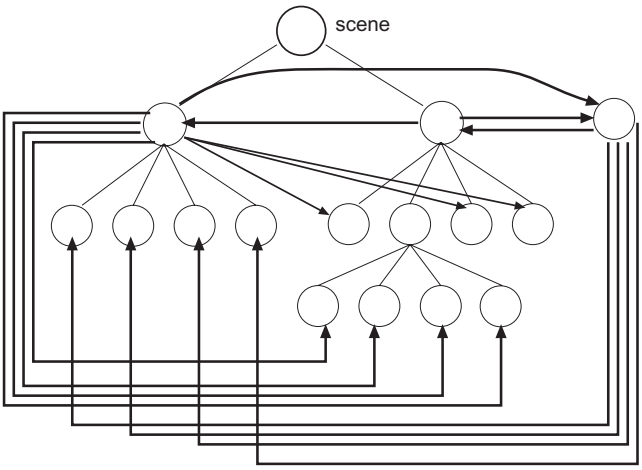


Abbildung 3: Eventuelle Linkstruktur einer Miniszene

ohne hierarchischer Struktur modellieren, würde man $n \cdot (n-1)$ Links benötigen, dies entspräche 132 Links, anhand dieser kleinen Beispielszene ist zu erkennen, dass sich der Aufwand des hierarchischen Linkings auf $O(n)$ veringern kann.

Links werden generell beim Empfänger gespeichert, jeder Empfänger hat somit eine Linkliste, die Links speichern wiederum Sender, Formfaktor und auch die Visibilität.

2.3 Der Ablauf des Verfahrens

Schematisch funktioniert das hierarchische Radiosity nach Hanrahan[HSA91] wie folgt:

1. Zu Anfang unterteilt man die Polygone, aus der die Szene besteht in möglichst wenige „wellshaped“[7], große Rootpatches man erhält damit das initiale Mesh. Dies ist die Aufgabe des Meshings und nachzulesen in der Studienarbeit „Meshing in RADIOCITY“ von Dominik Rau.

2. Jedes Paar von Rootpatches, die einander zugewandt sind, wird durch einen Link verbunden (Initial-Linking).

3. Alle Patches und deren Links werden nun durchlaufen, das Orakel testet, ob nun der Link ausreichend ist.

4. Falls der Link nicht ausreicht wird dieser gelöscht, das entsprechende Patch wird unterteilt und der Link wird wie vorab erwähnt verfeinert. Dieser Schritt wird nun auch rekursiv für alle neue erzeugten Subpatches und dessen Links durchgeführt (siehe Abb.4). Dies ist der sogenannte Refine-Schritt.

[7] wellshaped = wohlgeformt

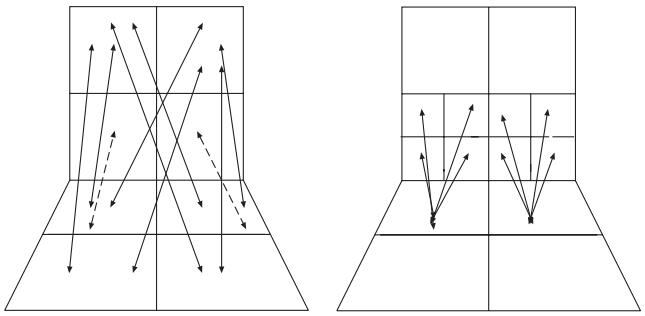


Abbildung 4: Das linke Bild zeigt Patches in einer einfache Szene, die miteinander verlinkt sind. Die gestrichelten Links sind laut Refinementkriterium nicht ausreichend und werden im nächsten Schritt verfeinert (siehe rechts)

5. Nach dem Refine, wird die Energie mit Hilfe der Radiostiy-Gleichung (siehe Gleichung 2 über alle Links bei jedem Patch eingesammelt. Dies ist der sogenannte Gather-Schritt.

6. Die berechneten Radiosity Werte auf unterschiedlichen Levels müssen in der gesamten Hierarchie konsistent sein. Durch den PushPull.Schritt wird dies erreicht.

7. Jetzt wird ab Punkt 3, die nächste Iteration gestartet und zwar solange bis die Patches ausreichend verfeinert sind, d.h. es wird kaum noch Energie übertragen.

2.4 Der Prisma-Formfaktor

Zu keiner Zeit des Verfahrens wird ein Fläche zu Fläche Formfaktor benötigt, da nur Radiositywerte an Empfängermittelpunkten berechnet werden, bzw. um minimale und maximale Formfaktoren abzuschätzen auch Formaktoren für bestimmte Punkte auf dem Patch berechnet werden. D.h. es werden nur Fläche zu Punkt Formfaktoren benötigt. Das Prisma Verfahren bietet eine relativ einfache und effiziente Methode den Formfaktor der ersten Formfaktorvereinfachung zu berechnen. Die Herleitung des Prisma-Verfahrens kann z.B. in [Gla95] nachgelesen werden. Mit Hilfe des Satzes von Stokes [AF02] kann man Flächenintergrale in Konturintegrale umformen und erhält letztendlich die recht einfache Formel:

$$F_{dA_e A_s} = -\frac{1}{2\pi}\vec{n_e} \cdot \sum \vec{\gamma_i} \tag{3}$$

Über den Sender wird ein Prisma aufgespannt (siehe Abb. 5), daher der Ausdruck Prisma Formfaktor. Der Vektor $\vec{\gamma}$, dessen Länge genau den Betrag des Winkels γ der aufgespannten Pyramidenseite entspricht, steht genau senkrecht auf dieser Seite. $\vec{n_e}$ ist die normierte Normale des Empfängerpatches.

Die Vektoren $\vec{\gamma_i}$ lassen wie folgt berechnen: Man berechnet einen auf der Pyramidenseite senkrecht stehenden, nach außen zeigenden Vektor, diesen normiert man und multipliziert ihn mit dem Betrag des aufgespannten Winkels γ_i. Den auf der Pyranidenseite senkrecht stehenden, normierten Vektor $\vec{g_i}$ erhält man über das Kreuzprodukt der Kantenvektoren r_{i+1} und r_i dividiert durch seine Länge:

$$\vec{g_i} = \frac{\vec{r}_{i+1} \times \vec{r}_i}{|\vec{r}_{i+1} \times \vec{r}_i|} \tag{4}$$

Den Winkel γ_i erhält man folgendermaßen:

$$\gamma_i = \arccos \frac{\vec{r}_{i+1} \circ \vec{r}_i}{|\vec{r}_{i+1}| \cdot |\vec{r}_i|} \tag{5}$$

Und aus dem Produkt des Skalars γ_i und dem Vektor $\vec{g_i}$ letzendlich den gesuchten Vektor:

$$\vec{\gamma_i} = \gamma_i \cdot \vec{g_i} \tag{6}$$

Der Prisma-Formfaktor (siehe Formel 3) kann somit ermittelt werden.

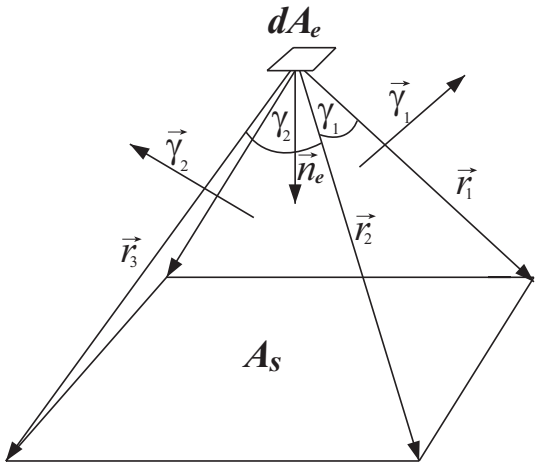


Abbildung 5: Skizze zur Berechnung des Prisma-Formfaktors

2.4.1 Minimaler und Maximaler Formfaktor

Das Refinement-Orakel(siehe Kap. 2.7) trifft Aussagen über Radiosityverteilungen die auf dem Empfängerpatch zu erwarten sind, wenn diese als nicht konstant angenommen werden kann (*constant radiosity asumption*), wird das Patch unterteilt und die Links werden entsprechend verfeinert, ansonsten ist der Link ausreichend. Eine gute Abschätzung, bzw. Berechnung des minimalen und maximalen Formfaktors vom Sender zum Empfängerpatch spielt dabei eine nicht unerhebliche Rolle.

Eine einfache Lösung wäre, den Prisma-Formfaktor von der Senderfläche zu dem Mittelpunkt bzw. Eckpunkten des Empfängerpatches zu bestimmen und daraus den maximalen und minimalen Formfaktor zu bestimmen, wie es auch Lischinski *et al.* [LSG94] handhaben. Partielle bzw. totale Verdeckungen (siehe Kapitel 2.5) sollten dabei auch schon in Betracht gezogen werden, bei partieller Verdeckung ist der minimale Formfaktor auf null zu setzen und bei totaler Verdeckung ist auch der maximale Formfaktor gleich null.

Diese Methode liefert schon sehr gute Ergebnisse, doch werden auf diese Weise keine tatsächlichen Schranken berechnet. Bei relativ großen Patches kann demnach der geschätze maximale Formfaktor doch um einiges vom realen Maximim abweichen, wenn z.B. das tatsächliche Maximum in der Mitte einer Kante liegen würde, Mittelpunkt und Eckpunkte liegen allerdings um einiges von diesem Punkt entfernt und deren Formfaktorbestimmungen können um einiges vom realen Wert abweichen.

Die Suche nach dem exakten Formfaktormaximum, erweist sich als etwas aufwendiger, wobei das Minimum auf einer der Eckpunkte des Empfängerpatches zu suchen ist und demnach über die Berechnung aller Eckpunktformfaktoren bestimmt wird[HS98].

Für die Bestimmung des Maximums wird die akzeptierte Annahme, dass nur ein Formfaktormaximum auf dem Empfängerpatch zu erwarten ist zu Hilfe gezogen. Holschuch *et al.* machen sich verschiedene Eigenschaften des bestimmten Radiositygradienten zu Nutze [HS98].

Im unserem entwickelten RADIOCITY-Systems, werden vorab minimaler und maximaler Formfaktor über die Berechnung von Senderfläche zu Empfängereckpunkt- und Empfängermittelpunkt-Formfaktoren bestimmt, wie es auch Lischinski *et al.* handhaben. Sollte der maximale Formfaktor allerdings für eine Unterteilung des Patches nicht ausreichen, bedarf es einer genaueren Näherung: Der maximale Formfaktor wird hierbei, wie auch bei Holzschuch, mit Hilfe der 1.Ableitung der 2.Formfaktorvereinfachung bestimmt, wobei Holzschuch‘s Methode die Ableitung der Prisma-Formel zu Nutze zieht. Es wird auch hier die allgemein als zutreffend akzeptierte Annahme, dass auf der gesamten Empfängerpatch-Ebene nur ein Maximum für den Formfaktor existiert und desweiteren, dass selbst auf einer Linie nur ein maximaler Formfaktor zu finden ist, zu Hilfe gezogen. Um diesen maximalen Formfaktor zu bestimmen, wird die 3D Szene zweidimesional interpretiert (siehe Abb. 6).

Der Mittelpunkt des Senderpatchs wird auf die Empfängerebene projeziert. An dieser Stelle entsteht der neue Ursprung für das zweidimensionale Koordiantensystem. Die x-Achse entpricht der Projektion von dem Abstandsvektor von Sendermittelpunkt zu Empfängermittelpunkt auf die Empfängerebene, diese wird normiert, auf dieser Achse befindet sich das zu suchende Formfaktormaximum. Die y-Achse entspricht dem normierten Vektor von projeziertem Mittelpunkt auf die Empfängerebene zu dem Mittelpunkt vom Senderpatch. Die zweite Formfaktorvereinfachung (siehe Gleichung 7) wird nun mit Hilfe des zweidimensionalen Koordina-

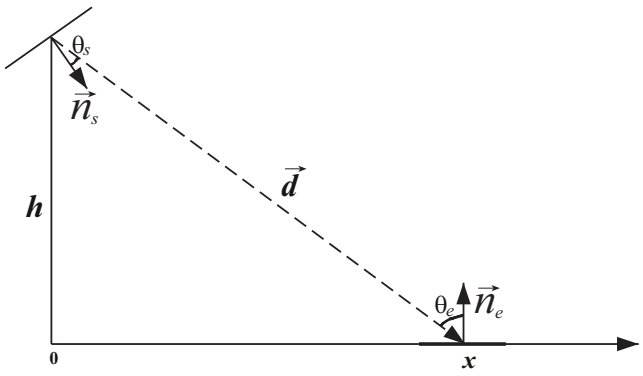


Abbildung 6: Bestimmung von ff(x)

tensystem beschrieben.

$$F_{se} = \frac{A_s \cos\theta_e \cdot \cos\theta_s}{\pi d^2} \tag{7}$$

Die in Abbildung 6 dargetellten Werte können nun folgendermaßen interpretiert werden:

$$\vec{n}_s = \begin{pmatrix} n_{sx} \\ n_{sy} \end{pmatrix} \text{ mit } n_{sx} = \vec{n}_s \circ \vec{x}_0 \text{ und } n_{sy} = \vec{n}_s \circ \vec{n}_e \tag{8}$$

$$\vec{x} = \vec{n}_e \times (\vec{n}_s \times \vec{n}_e) \tag{9}$$

$$\vec{x}_0 = \frac{\vec{x}}{|\vec{x}|} \tag{10}$$

$$\vec{n}_e = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{11}$$

$$\vec{d} = \begin{pmatrix} x \\ -h \end{pmatrix} \tag{12}$$

$$d^2 = x^2 + h^2 \tag{13}$$

Anhand der Definition des Skalarprodukts erhält man:

$$\vec{d} \circ \vec{n}_s = |\vec{d}| \cdot |\vec{n}_s| \cdot \cos\theta_s = |\vec{d}| \cdot \cos\theta_s, \text{da } |\vec{n}_s| = 1 \tag{14}$$

sowie:

$$-\vec{d} \circ \vec{n}_e = |-\vec{d}| \cdot |\vec{n}_e| \cdot \cos\theta_e = |\vec{d}| \cdot \cos\theta_e, \text{da } |\vec{n}_e| = 1 \text{ und } |-\vec{d}| = |\vec{d}| \tag{15}$$

Desweiteren gilt nach Definition des Skalarprodukts:

$$\vec{d} \circ \vec{n}_s = d_x n_{sx} + d_y n_{sy} \tag{16}$$

Nach Einsetzen der neu interpretierten Werte aus der Gleichung 12 $d_x = x$ und $dy = -h$ erhält man:

$$\vec{d} \circ \vec{n}_s = x n_{sx} - h n_{sy} = |\vec{d}| \cdot \cos\theta_s \tag{17}$$

Analog für den Empfänger:

$$-\vec{d} \circ \vec{n}_s = d_x n_{ex} + d_y n_{ey} \tag{18}$$

Nach Einsetzen von $d_x = x$, $dy = -h$, $n_{ex} = 0$ und $n_{ey} = 1$ erhält man:

$$-\vec{d} \circ \vec{n}_s = x \cdot 0 - h \cdot 1 = -h = |\vec{d}| \cdot \cos\theta_e \tag{19}$$

Folgende 2 Formeln haben wir aus den Herleitungen gewonnen:

$$|\vec{d}| \cdot \cos\theta_s = x n_{sx} - h n_{sy} \tag{20}$$

und

$$|\vec{d}| \cdot \cos\theta_e = -h \tag{21}$$

Die Gleichung der 2. Formfaktorvereinfachung wird nun noch mit d^2 erweitert so erhält man:

$$F_{se} = \frac{A_s \cdot d \cdot \cos\theta_e \cdot d \cdot \cos\theta_s}{\pi d^4} \tag{22}$$

Durch Einsetzen der Formeln 20 und 21 sowie $d^2 = x^2 + h^2$ aus Formel 13 erhält man die 2. Formafktorvereinfachung interpretiert durch das zweidimensionale Koordiantensystem in Abhängigkeit von x:

$$FF(x) = \frac{A_s \cdot (x \cdot n_{sx} - h \cdot n_{sy}) \cdot h}{\pi \cdot (x^2 + h^2)^2} \tag{23}$$

Diese Gleichung wird nun differenziert, der Funktionswert soll Maximal werden, d.h. die 1. Ableitung wird gleich null gesetzt und nach x umgeformt. Man erhält folgende Gleichung:

$$x_{1,2} = \frac{2}{3} \cdot \frac{h \cdot n_{sy}}{n_{sx}} \pm \sqrt{\left(\tfrac{2}{3} \cdot \tfrac{h \cdot n_{sy}}{n_{sx}} \right)^2 + \frac{1}{3} h^2} \tag{24}$$

Da sich im Bereich der negativen x-Achse kein Maximum befinden kann, berechnet man nur den positiven x-Wert. Der x-Wert muss nun allerdings noch zurück in den dreidimensionalen Raum transformiert werden, dies geschieht recht einfach indem man auf die 3D Koordinaten des Ursprungs $\vec{p}$ der 2D Ebene, x mal den $\vec{x}_0$ Vektor addiert.

$$\vec{x}_{3D} = \vec{p} + x \cdot \vec{x}_0 \tag{25}$$

Jetzt sind die 3D Koordinaten des Ortsvektors auf den Punkt berechnet für den der Formfaktor maximal wird, dieser muss sich allerdings nicht innerhalb des Empfängerpatches befinden. Falls dieser Punkt innerhalb des Patches liegt hat man den Punkt mit dem maximalen Formfaktor gefunden, von diesem berechnet man dann den Senderfläche zu Punkt Formfaktor mittels der Prisma Formel (siehe Formel 3). Falls der Punkt nicht innerhalb des Empfängerpatches liegt, wird der Ortsvektor auf den Punkt auf dem Patch bestimmt, der den kürzesten Abstand zu dem Punkt mit

maximalen Formfaktor hat, dies kann dann ein Eckpunkt oder ein Punkt auf der Kante sein. Für den ungünstigen Fall, dass der Punkt mit kürzestem Abstand zum Punkt mit maximalen Formfaktor im hinteren Halbraum des Senderpatches liegt, bedarf es an weiteren Untersuchungen, da hier der berechnete Prisma-Formfaktor null ergibt. Dieses Ergebnis liefert dann einen falschen Wert, wenn das Empfängerpatch nur zum Teil im hinteren Halbraum des Senderpatches liegt (Support Plane Split). In solch einem Fall erweist sich die Bestimmung des tatsächlichen Fornfaktormaximums, als äußerst aufwendig, doch reicht es aber in den meisten Fällen aus, sich auf die Näherung des Formfaktormaximums über die Patcheckpunkte zu beschränken.

2.5 Visibilität

Die Visibilitätsberechnungen im Rahmen eines Radiosity-Systems erweist sich als zeitaufwendigste Komponente. Exakte Visibilitätsbestimmungen, die zur Vermeidung von Artefakten (wie z.B. fehlende Schatten) nötig sind, bedeuten einen deutlichen Mehraufwand gegenüber weniger genauen Berechnungen. Um detailgenaue Ergebnisse zu erzielen ist dieser Mehraufwand zwingend erforderlich. Eines de Standardverfahren zur Visibilitätsbestimmung ist das sogenannte *jittered raycasting*[8], hierbei werden eine fixe Anzahl von Strahlen von Empfänger zu Sender geschossen, die innerhalb eines festen Rasters auf den Patches gejittered (leicht zufällig versetzt) werden (siehe Abb. 7). Die einzelnen Strahlen werden auf Sichtbarkeit

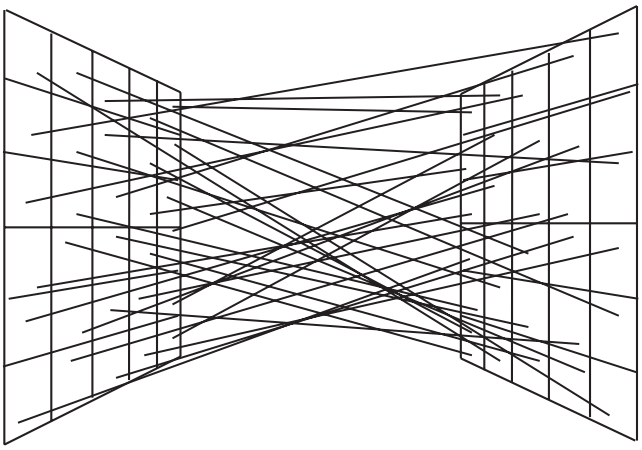


Abbildung 7: Gejittertes Raycasting mit 25 Strahlen

getestet, d.h. jeder Strahl wird mit allen anderen Rootpatches der Szene auf einen Schnittpunkt getestet. Falls ein anderes Patch getroffen wird, kann der Sichtbarkeitstest unterbrochen werden und der Strahl als nicht sichtbar klassifiziert werden. Falls keine anderen Patches getroffen werden ist der Strahl als sichtbar zu klassifizieren. Der Visibilitätswert berechnet sich demnach aus Anzahl der sichtbaren Strah-

[8] jittered raycasting könnte man grob als zitternder Strahlenwurf bezeichnen

len dividiert durch die Anzahl aller Strahlen. Die Visibilität ist demnach 1 bei voller Sichtbarkeit und 0 bei totaler Verdeckung. Holzschuch *et al.* und auch Hanrahan *et al.* benutzen bei Ihren Radiosity-Simulationen zur Sichtbarkeitsbestimmung nur ein Raycasting mit 16 Strahlen, so konnten sie natürlich keine hohe Qualität in Bezug auf Schattenkanten vorweisen. Um mit diesem Verfahren gute Ergebnisse zu erzielen ist es demnach umungänglich eine ausreichend grosse Menge an Strahlen zu verschiessen und dies kostet selbstverständlich sehr viel Zeit.

Um nun Zeit einzusparen ist es ratsam zuvor einen Halbraumtest durchzuführen, da laut Definition ein Patch die Energie nur in den vorderen Halbraum abstrahlen kann. D.h. man testet ob sich Sender und Empfänger überhaupt sehen können, mit anderen Worten, ob sie Energie austauschen können. Emfänger und Sender müssen sozusagen teilweise zugewandt sein. Dies geschieht indem man testet ob das Empfängerpatch im vorderen Halbraum des Senders liegt, falls dies nicht der Fall ist kann hier schon abgebrochen werden, desweiteren testet man auch die umgekehrte Richtung, ob der Sender im vorderen Halbraum des Empfängers liegt. Wenn dies auch nicht der Fall ist können sich die beiden Patches nicht sehen und der Visibilitätstest braucht nicht mehr durchgeführt werden, man kann abbrechen und die Visibilität auf Null setzen. Ansonsten ist als nächster Schritt das *gejitterte Raycasting* durchzuführen.

Eine weitere Verbesserung, zugunsten eines enormen Geschwindigkeitsgewinn, ist die Reduzierung der Teststrahlen. Dies geschieht zu einem indem man die Anzahl der Interaktionen, sprich die Anzahl der Links verringert, was z.B durch Clustering erzielt werden kann (siehe Kapitel 3), oder indem man natürlich direkt die Anzahl der Strahlen in Abhängigkeit von Patchgrösse pro Interaktion verringert, da man bei kleinen Sender- und Empfängerpatches weniger Strahlen benötigt. Besser wäre noch, die Strahlenanzahl in Bezug auf den aufgespannten Raumwinkel zwischen einem Punkt auf dem Senderpatch und dem Empfängeratch zu bringen. Ein weiterer Faktor, der die Anzahl der Strahlen mit bestimmt, ist die zu verschiessende Energie. Falls so gut wie keine Energie verschossen wird braucht man auch keine exakte Visibilitätsbestimmung, da der Betrachter dies auch nicht mehr erkennen kann. Dies sollte nochmals einen guten Zeitgewinn erzielen.

Eine grosse Zeitersparnis bei der Visibilitätsbestimmung bringt das sogenannte Shaft Culling[9], was 1991 von Haines und Wallace [HW91] vorgestellt wurde, mit sich. Bei diesem Verfahren wird vorab ein Schacht(*shaft*) zwischen Sender und Empfänger berechnet, sozusagen das Volumen das von Sender und Empfänger aufgespannt wird (siehe Abb. 8). Dazu benötigt man die achsenparallele Bounding Box von Sender- und Empfängerpatch, beide Bounding Boxes werden nun zu einem *Shaft* verknüpft, da dadurch die Berechnung der konvexen Hülle, die hier dem Shaft entspricht recht einfach ist. Die Bounding Boxes aller anderen Objekte in der Szene werden dann anhand dieses Shaftes getestet ob sie vollständig ausserhalb liegen, bzw. ob sie vollständig innerhalb des Shafts liegen. Die Objekte dich vollständig ausserhalb liegen brauchen nicht weiter beachtet werden, dies erbringt den enormen

[9] Shaft Culling könnte man grob mit Schacht-Auswahl übersetzen

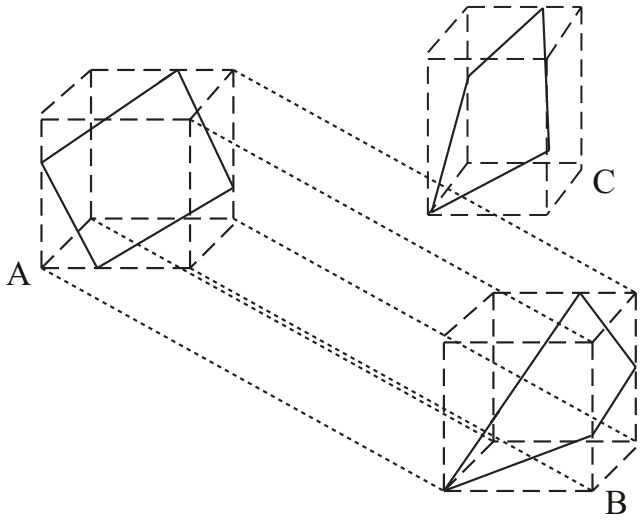


Abbildung 8: Shaft Culling, Shaft über die Bounding Boxes von A und B

Zeitgewinn, da doch einige Objekte nicht innerhalb des Shaftes liegen. Die Objekte die teilweise bzw. ganz im inneren des Shafts liegen, werden in einer sogenannten Kandidatenliste abgelegt, da dann nur noch mit diesen Objekten ein Raycasting nötig ist. Der Shaft Culling Test ist recht schnell, da nur ein Eckpunkt der Bounding Box mit allen Seitenflächen des Shafts getestet werden muss, welcher Eckpunkt getestet wird, wird allerdings vorab bestimmt [Hai00]. Wie ein solcher Shaft recht schnell und effizient berechnet werden kann ist in [Hai00] nachzulesen.

2.6 Support Plane Split

Wurde anhand des Visibilitätstest festgestellt, dass sich zwei Patches sehen können, d.h. der Empfänger ist nicht von einem dazwischen liegenden Objekt total verdeckt, ist der Halbraumtest genauer zu untersuchen. Man benötigt Informationen darüber, wie die beiden Flächenelemente im Raum zueinander plaziert sind, es muss getestet werden, ob die beiden Patches komplett zugewandt sind, wenn das nicht der Fall ist liegt ein sogenannter Support Plane Split vor. Ein Support Plane Split tritt dann auf, wenn sich die Senderebene im Empfängerpatch schneidet (*target split*) bzw. wenn sich die Empfängerebene im Senderpatch schneidet (*source split*, siehe Abb.9). Dieser Test lässt sich recht einfach im Halbraumtest integrieren, ein SPS liegt z.B. dann vor, wenn alle Eckpunkte des Empfängers im vorderen Halbraum des Senders liegen, und mindestens einer, aber nicht alle Eckpunkte des Senders im vorderen Halbraum des Empfängers liegen, dies wäre der *source split*, analog wird der (*target split* bestimmt. Falls ein Support Plane Split vorliegt, sollte die Schnittkante beim Sender (*source split*) bzw. Empfänger (beim *target split*) berechnet werden und nur für die sichtbaren Bereiche Formfaktoren bestimmt werden, da sonst die Formfaktorberechnung zu klein gegebenfalls sogar negativ werden kann. Bei einem *source split*

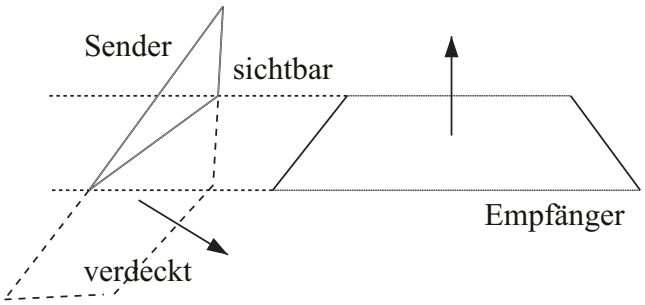


Abbildung 9: Support Plane Split beim Sender (*source split*)

wird die Schnittkante nur zur Berechnung des Formfaktors benötigt, es ist nicht nötig das Senderpatch entlang der Schnittkante zu unterteilen, da ein *source split* keine sichtbaren Artefakte erzeugt [Kre97]. Bei einem *target split* hingegen ist das Empfängerpatch definitiv zu unterteilen, da hier eine sichtbare *D1*-Unstetigkeit entlang der Schnittkante ensteht [Kre97]. Es ist von Vorteil, direkt entlang der Schnittkante das Patch zu unterteilen, falls das implementierte Radiosity System das *discontinuity meshing* unterstützt. Die standardmässige Unterteilung wird zu gezackten Schattenkannten führen, daher ist hier eine hohe Unterteilung entlang der Schnittkante nötig, um diese Artefakte möglichst klein zu halten. Ein Vorteil des *discontinuity meshing* ist, dass diese Diskontinuität entlang der Schnittkante, mit der minimal möglichen Unterteilung beseitigt werden kann, allerdings Bedarf es einigiger Berechnungen, da z.B. unser RADIOCITY-Systems nur auf dreieckige und viereckige Patches basiert, d.h. aus einem viereckigen Patch kann durch nur einen Schnitt ein dreieckiges Subpatch und fünfeckiges Subpatch entstehen (siehe Abb. 10). RADIOCITY arbeitet bislang auf einem adaptiven Mesh, das rekursiv unterteilt wird. RADIOCITY wird demnach bei Support Plane Splits das Empfängerpatch, entlang der Schnittkante angemessen häufig unterteilen (siehe Abb. 10). Die gestrichelten Linien, links in Abb.

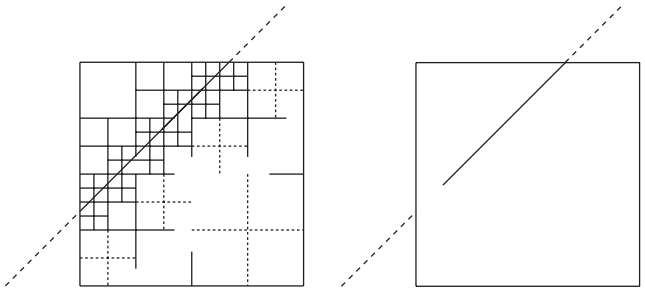


Abbildung 10: Vergleich Standard Unterteilung(links) und asymetrische Unterteilung(rechts) bei einem SPS

10 entsprechen den Subpatches die durch die Balancierung[10] entstehen. Alles über Meshing in RADIOCITY und was damit zusammenhängt, wie die Balancierung, die Subdivide-Methode, Nachbarschaftsbeziehungen usw. wird genauer in der Studienarbeit von Dominik Rau erläutert.

2.7 Refinement

2.7.1 Refinementkriterium

Das Refinementkriterium u.a auch Orakel genannt ist wohl der wesentliche Kernpunkt eines adaptiven Radiosity-Systems. Hier entscheidet sich ob ein Link verfeinert werden soll oder nicht, d.h. ob eine weitere Unterteilung des Patches nötig ist oder ob der Link für die Radiosity Berechnung ausreichend ist.

Ein zu grobes Mesh erzeugt zu viele Artefakte, es können Nähte, vor allem aber auch Machbänder entstehen. Desweitern können Schattenkanten sehr grob ausfallen, bzw. sehr zackige Schattenkanten entstehen, oder auch entsprechende Highlights nicht genügend genau dargestellt werden. Die Generierung eines zu feinen Meshes dahingegen bringt einen hohen Zeitaufwand und einen großen Speicherbedarf mit sich. Ein gutes Orakel muss demnach intelligent entscheiden, wann zu unterteilen ist und wann nicht, es werden vorab Informationen über zu erwartende Radiosity- und Visibilitätsverteilungen benötigt, um unnötige Unterteilungen zu vermeiden und dementsprechend fein an Stellen zu unterteilen bei denen es sinnvoll ist, wie z.B. dort wo Schattenkannten entstehen (siehe Abb.11).

Wann ist denn nun zu unterteilen?

Ein Link wird als ausreichend interpretiert, wenn die von einem Sender verursachten Radiosity-Werte für den Empfänger als konstant angenommen werden kann, d.h. unter Annahme einer konstanten Sender-Radiosity sind die Radiosity-Werte aller Punkte des Empfängers etwa gleich (*constant radiosity asumption*). Desweiteren wird untersucht wie sich die Radiosity-Verteilung auf dem Sender verhält.

Um nun Aussagen über die Radiostyverteilung treffen zu können, sind folgende vorab berechneten Werte nötig:

- F_{min} minimaler Formfaktor
- F_{max} maximaler Formfaktor
- B_{min} minimaler Radiositywert aller Sender-Subpatches
- B_{max} maximaler Radiositywert aller Sender-Subpatches

[10] Beim unterteilen eines Patches, darf der Rekursionsstufenunterschied der Nachbarpatches maximal eins betragen.

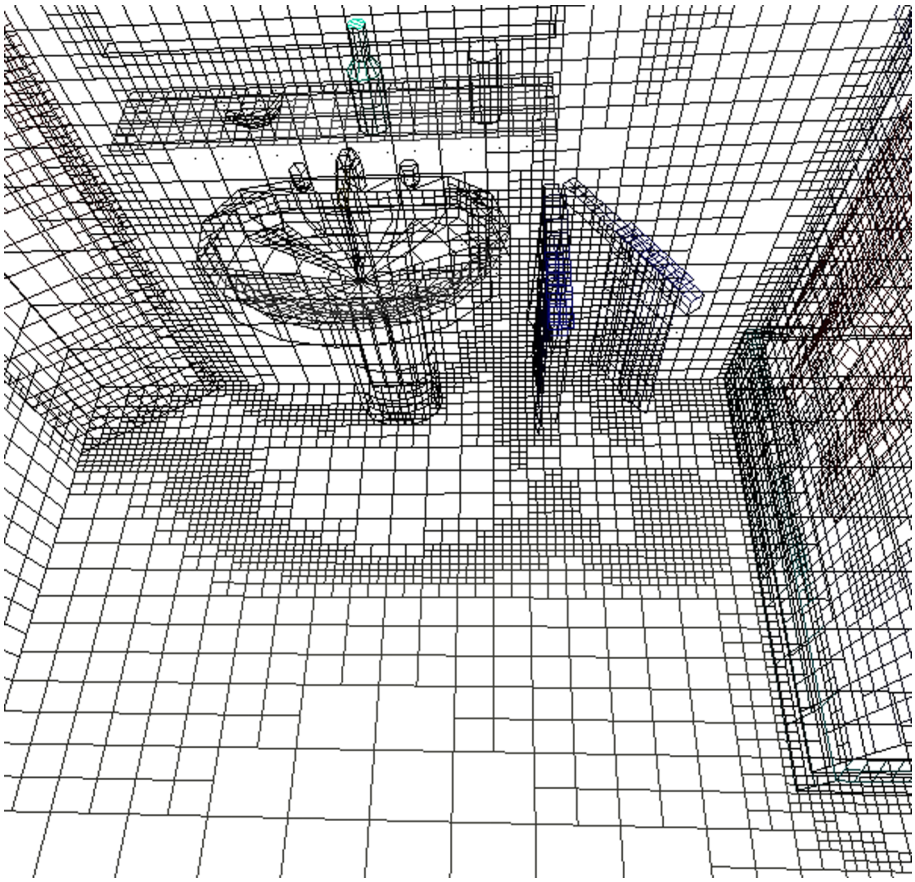


Abbildung 11: Adaptives Mesh mit genügend feiner Unterteilung im Halbschatten

Die Berechnung des minimalen und maximalen Formfaktors ist bereits in Kapitel 2.4.1 erläutert worden. Den minimalen und maximalen Radiosity Wert des Senders erhält man, indem man alle Subpatches des Senderpatches betrachtet und die entspechenden Radiosity Werte ausliest. Es ist ratsam minimale und maximale Radiosity Werte bei jedem Patch zu speichern.

Das nun angewandte Refinement-Kriterium nach Lischinski[LSG94] ist ein fehlerbasiertes Unterteilungskriterium, d.h. es werden bei einer Nichtunterteilung Fehler akzeptiert, die unterhalb der entsprechenden Epsilon-Schranke liegen. Dieser Epsilon-Wert muss natürlich sinnvoll gewählt werden, da sie für die Wirksamkeit des Kriteriums verantwortlich sind. Das Refinementkriterium sollte nun den Fehler minimieren und eine bestmögliche Annäherung einer korrekten Lösung liefern. Das sogenannte 1. Orakel entscheidet nun ob unterteilt wird oder nicht, das 2. Orakel entscheidet ob Sender oder Empfänger zu unterteilen sind. Das erste Orakel, das den Fehler der Energieübertragung beschreibt, ergibt sich somit zu:

$$\Delta E = A_e \cdot (F_{max}B_{max} - F_{min}B_{min}) \tag{26}$$

Dieses 1. Orakel beschreibt eine obere Schranke des Fehlers der Energieübertra-

gung, die durch Abschätzung der Variation der Sender-Radiosity und der Variation des Formfaktors auf dem Empfänger bestimmt ist. Liegt nun ΔE oberhalb einer vom Benutzer vorgegebenen Schranke ϵ_{refine} ist der Link zu verfeinern.

Ob nun Sender oder Empfänger zu unterteilen sind, hängt davon ab, wer den grösseren Fehler verursacht [LSG94]. Der Sender ist zu unterteilen falls:

$$F_{max} \cdot (B_{max} - B_{min}) > B_{max} \cdot (F_{max} - F_{min}) \tag{27}$$

gilt, ansonsten ist der Empfänger zu unterteilen.

Visibilität und Support Plane Split wurden bis jetzt außer Acht gelassen, doch lassen sich diese Faktoren recht einfach ins Refinement Kriterium integrieren. Des weiteren lässt sich das 1. Orakel in zwei getrennte Abfragen aufgliedern, zu einem wird getestet ob genügend Energie, d.h. ob es überhaupt Sinn macht zu unterteilen und zu anderem, ob die zu erwartende Radiosity unterhalb der vorgebenen Schranke liegt, wenn ja ist der Link ausreichend. Das komplette Refinement-Kriterium für das 1. Orakel hätte damit die Form wie in Algorithmus 2.1 dargestellt.

Algorithmus 2.1 Pseudo Code der Refinement Entscheidung (1.Orakel)

```
bool oracle1()
{

// Genügend Energie, wenn nicht -> Nicht unterteilen.
IF Bmax*Fmax*Ae < bfaEps THEN RETURN false;

// Totale Verdeckung, wenn ja nicht unterteilen.
IF visibility = 0 THEN RETURN false;

// Partielle Verdeckung, wenn ja minimaler Formfaktor auf null setzen.
IF (visibility > 0 AND visibility < 1) THEN Fmin = 0;

// Liegt ein target split vor, dann muss der Empfänger unterteilt werden.
IF (sps = targetsplit) THEN return SUBDIVIDE_RECEIVER;

// Fehlerabschätzung der Radiosity zu gross?
// Wenn nicht, nicht unterteilen.
IF (Fmax * Bmax - Fmin * Bmin) < refineEps THEN RETURN false;

// Falls keiner der Bedingungen eintritt ist zu unterteilen.
RETURN true;
}
```

Als erstes berechnet man die maximal mögliche Energie, die übertragen werden kann, d.h. maximale Sender-Radiosity B_{max} multiplizert mit maximalen Formfaktor des Empfängers F_{max} multipliziert mit der Fläche des Empfängers A_e. Falls dieser Wert kleiner der Schranke ϵ_{BFA} ist, wird nicht unterteilt und der Refinement-Test kann schon abgebrochen werden. Wird genügend Energie übertragen sind weitere Tests nötig. Bei einer totalen Verdeckung kann auch direkt abgebrochen werden, da hier nicht unterteilt werden muss. Bei der partiellen Verdeckung hingegen, muss der minimale Formfaktor korrigiert werden, da dessen Abschätzung wie in Kapitel

2.4.1 erläutert wurde, über die Eckpunkte bestimmt wurde. Liegt nun eine partielle Verdeckung vor, muss nicht unbedingt einer der Eckpunkte verdeckt sein, daher wird F_{min} auf null gesetzt, da F_{min} nur über die Eckpunkte bestimmt wurde. Wenn ein Support Plane Split vorliegt ist zwischen *source split* und *target split* zu unterscheiden (siehe Kapitel 2.6). Falls ein *target split* vorliegt ist definitiv der Empfänger zu unterteilen, der Aufruf des 2.Orakel wäre in diesem Fall nicht mehr nötig. Der Sender wird nur dann unterteilt, falls dies das 2.Orakel anhand der Gleichung 27 entscheidet. Die letzte Abfrage beschreibt nun, eine obere Schranke des Fehlers der zu erwartenden Radiosityübertragung. Liegt dieser Fehler unterhalb der gegebenen $epsilon_{refine}$ Wertes, kann die zu verteilende Radiosity als konstant angesehen werden und es wird nicht unterteilt, ansonsten wird unterteilt. Die Abfrage wer zu unterteilen ist, lässt sich recht einfach in den Algorithmus 2.1 integrieren, indem drei Rückgabewerte zugelassen werden, wie z.b ’0’ für nicht unterteilen, ’1’ für Empfänger unterteilen und ’2’, um den Sender zu unterteilen.

2.7.2 Verfeinern der Links

Nachdem die Empfänger bzw. Sender unterteilt worden ist, müssen nun allerdings noch die Links angepasst werden. Das soll heißen, falls der Empfänger unterteilt worden ist, wird der Link zwischen Sender und Empfänger gelöscht und neue Links von Sender zu den enstandenen Subpatches werden generiert und der Linkliste des entsprechenden Empfängers hinzugefügt (siehe Abb. 12). Falls der Sender unterteilt

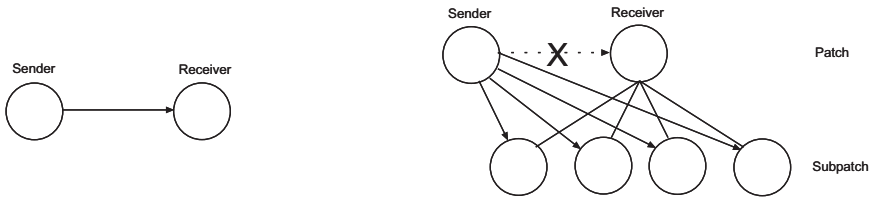


Abbildung 12: Receiver wird in vier Subpatches unterteilt, Sender zu Empfänger Link wird gelöscht, und vier neue Links zu den Subpatches generiert.

worden ist, wird auch der Link gelöscht und entsprechend viele neue Links von den Sender-Subpatches zu dem Empfänger generiert, diese neuen Links werden in der Linkliste des Empfängers abgelegt (siehe Abb. 13).

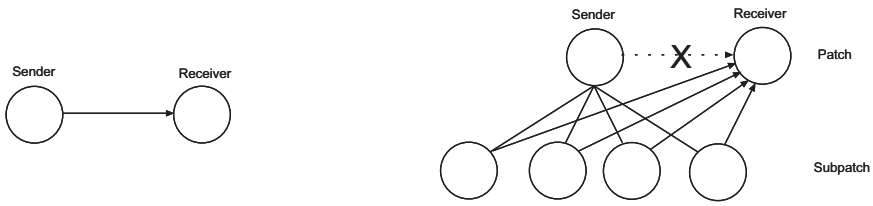


Abbildung 13: Sender wird in vier Subpatches unterteilt, Sender zu Empfänger Link wird gelöscht, und vier neue Links von den Subpatches zum Receiver generiert.

2.7.3 Die *refine()* Methode

Alle wichtigen Bestandteile der *refine()* Methode sind nun bekannt, der Pseudo-Code der refine Methode ist in Algorithmus 2.2 zu sehen. Alle Rootpatches[11] wer-

Algorithmus 2.2 Pseudo Code der refine Methode

```
refine(patch)
{
  FOR ALL links of patch
  {
    IF oracle(sender, receiver) = "SUBDIVIDE_SENDER"
    {
      IF sender->ISNOTSUBDIVIDED THEN SUBDIVIDE(sender};
      DELETE(link);
      FOR ALL childs of sender
      {
        CREATE_NEW_LINK(child, receiver);
        ADD_LINK to receiver;
      }
    }
    ELSE IF oracle(sender, receiver) = "SUBDIVIDE_RECEIVER"
    {
      IF receiver->ISNOTSUBDIVIDED THEN SUBDIVIDE(receiver);
      DELETE(link)
      FOR ALL childs of receiver
      {
       CREATE_NEW_LINK(sender, child);
       ADD_LINK to child;
      }
    }
  }
  IF patch->ISSUBDIVIDED
  {
    FOR ALL childs refine(child)
  }
}
```

den durchlaufen, für jedes Rootpatch wird die *refine()* Methode aufgerufen, diese untersucht die Links von den Sendern zum Patch. Falls das Orakel enscheidet den Sender zu unterteilen, wird der Sender, falls er noch nicht unterteilt ist, unterteilt[12], der Link wird gelöscht und die Links entsprechend verfeinert und dem Empfänger hinzugefügt, falls der Empfänger unterteilt werden soll, verfährt man entsprechend, wie vorab erläutert. Falls der Link ausreichend ist, wird nicht unterteilt und der nächste Link wird untersucht. Wenn nun das Patch unterteilt worden ist, werden auch dessen Kinder untersucht, d.h. die *refine()* Methode wird rekursiv für alle Kinder aufgerufen. Die *refine()* Methode läuft dann genaus solange bis alle Links als ausreichend klassifiziert sind und nicht weiter verfeinert werden muss.

[11] Rootpatches sind im initialen Zustand noch nicht unterteilt.

[12] Der Sender kann allerdings schon durch einen anderen Link bzw. durch die Balancierung unterteilt worden sein.

2.8 Radiosityausgleich

Die vorhergehenden Schritte dienten bisher nur dazu eine ausreichend feine Patch und Linkstruktur aufzubauen, der eigentliche Energieaustausch wird in den *gather()* und *pushpull()* Methoden ausgeführt.

2.8.1 Die *gather()* Methode

Die *gather()* Routine ist eine rekursive Schleife über alle Links der Szene, die für jeden Empfänger die Radiosity-Werte aufsummiert.

$$B_e = B_e + \rho_e \cdot B_s \cdot F_{es} \cdot visibility \tag{28}$$

Es werden wiederum alle Links der Patches durchlaufen, wenn das Patch unterteilt ist, werden rekursiv die Kinder und deren Links durchlaufen (siehe Algorithmus 2.3).

Algorithmus 2.3 Pseudo Code der gather Methode

```
gather(patch)
{
  gather = 0;
  FOR ALL links of patch
  {
    vis = VISIBILITY of link;
    sender = SENDER of link;
    receiver = RECEIVER of link;
    ff = FORMFACTOR of link;
    rho = REFLEXION of receiver;
    radiosity = RADIOSITY of sender;
    gather = gather + rho * radiosity * ff * vis;
  }
  IF patch ISSUBDIVIDED
  {
    FOR ALL childs DO gather(child);
  }
}
```

2.8.2 Die *pushpull()* Methode

Nachdem für alle Patches der Szene die Radiosity über die *gather()* Methode eingesammelt wurde, müssen nun noch alle Wert für die hierarchische Struktur konsistent gemacht werden, da die Gathering-Werte die bei den einzelnen Patches gespeichert sind, auf veschieden Leveln eingesammelt wurde. Zur weiteren Berechnung nicht zuletzt zur Darstellung muss jedoch ein Gleichgewicht hergestellt werden. Hat zum Beispiel ein Rootpatch über einen Link Energie erhalten, so muss diese Energie gleichmässig an alle Kinder weitergegeben werden (dies entspricht dem *push*-Schritt). Empfängt ein Patch auf einem niedrigeren Level Energie, dann muss die gemittelte Energie bis zum Rootpatch hinaufgereicht werden (dies bezeichnet man als *pull*-Schritt). Der *push*-Schritt reicht Energie von höheren Leveln bis hinunter zu

den Blättern (siehe Abb. 14), der *pull*-Schritt verteilt hingegen flächengewichtet die durch den *push*-Schritt aufsummierten Radiositywerte an den Blättern bis hinauf zu dem Rootpatch (siehe Abb. 15). Somit sind die jeweilgen Formeln bestimmt:

$$Push : B_i = B_i + B \tag{29}$$

$$Pull : B = \frac{1}{A} \sum B_i \cdot A_i \tag{30}$$

Wobei A_i die Fläche und B_i die Radiosity der entsprechenden Kinder beschreibt.

Abbildung 14: Push: Radiosity des Patches wird den Kindern aufaddiert

Abbildung 15: Pull: Radiosity des Patches wird auf die flächengewichtete Radiosity der Kinder gesetzt (unter der Annahme, dass alle Kinder gleich groß sind)

Die *pushpull* lässt sich auch wiederum als rekursive Methode beschreiben (siehe Algorithmus 2.4).

Algorithmus 2.4 Pseudo Code der pushpull Methode

```
pushpull(patch, down)
{
  IF patch IS_NOT_SUBDIVIDED up = patch.gather + down;
  ELSE
  {
    up = 0;
    FOR ALL childs OF patch
    {
      tmp = pushpull(child, patch.gather + down);
      up = up + tmp * child.area / patch.area
    }
  }
  patch.gather = 0;
  patch.shoot = up;
  RETURN up;
}
```

Falls das Patch nicht unterteilt ist, wird die heruntergereichte Radiosity (down) auf den Patch-Gatherwert aufaddiert, ansonsten werden die Kinder untersucht und

der flächengewichtete Radiositywert (up) wird rekursiv hinaufgereicht. Leztendlich, nachdem alle Rekursionen durchlaufen worden sind, wird der Shootwert des Patches und somit die eigentliche Radiosity auf den up-Wert gesetzt. Nun ist die Szene in einem Gleichgewicht und der Shootwert kann nun zur Dartsellung bzw. für die nächsten Iterationen verwendet werden, da nun wiederum die *refine()*-Routine aufgerufen werden kann und alle Patches mit einem Shootwert grösser null somit als neue Lichtquelle angesehen werden kann. Ab der 2. Iteration wird demnach das indirekte Licht berechnet, wobei die erste Iteration nur das direkte Licht berechnet.

2.9 Darstellung der Patches

Dargestellt werden letztendlich nur die Blätter der Baumhierarchie, das sogenannte *Flat-Shading* visualisiert die konstanten Radiositywerte der Blattpolygone, dadurch ist aber die eigentliche Patchstruktur immernoch extrem sichtbar. Das *Gouraud-Shading*[Gou71], interpoliert die Patchfarbdarstellung über die Eckpunktfarben des Polygons. D.h. für jeden Eckpunkt des Patches wird eine Eckpunktfarbe bestimmt, die sich aus einer Mittelung der Farbe aller Nachbarpatches berechnet. Die Ermittlung von Nachbarpatches, bzw. Punkt-Patch Beziehungen finden sich in der Studienarbeit von Dominik Rau wieder. Durch das *Gouraud-Shading* ist für den Betrachter die eigentliche Patchstruktur nicht mehr zu erkennen.

Es treten allerdings auch einige Radiositywerte auf, die außerhalb des darstellbaren Bereich liegen können, bzw. Werte die extrem klein sind. Die Werte, die grösser als eins sind, werden normalerweise einfach nur auf eins geclippt, dadurch sind aber an einigen Stellen extreme Überbelichtungen zu erkennen. Mittels *Tonemapping* ist es möglich dies Werte sinnvoll in den Wertebereich von [0..1] zu skalieren. Die Funktionsweise der *Tonemapping*-Verfahren die in RADIOCITY Anwendung finden sind in der Studienarbeit von Florian Koller näher erläutert.

3 Hierarchisches Radiosity mit Clustering (HRC)

3.1 Die Idee

Bei komplexeren Szenen ist das hierarchische Radiosity Verfahren noch sehr zeitaufwendig. Allein das Initial Linkg benötigt zum generieren aller Links extrem viel Zeit, da hier für alle möglichen Patch zu Patch Kombinationen, Links generiert, Formfaktor und Visibilität bestimmt werden müssen. Es ist ratsam die Anzahl der Links zu reduzieren. Das hierarische Radiosity mit Clustering, dass 1994 von B. Smits, J. Arvo und D. Greenberg vorgestellt wurde bietet eine Möglichkeit die Anzahl der erforderlichen Links enorm zu senken [SAG94]. Die Idee hierbei ist es, dass die Rootpatches, die geometrisch im Zusammenhang stehen zu Gruppen vereint werden, die von den sogenannten Cluster repräsentiert werden. Die so gewonnenen Cluster können untereinander Energie austauschen, falls eine bestimmte Fehlerschranke nicht überschritten wird und dadurch wird ein Großteil an Links nur durch

einen Link repräsentiert. Sinnvoll ist dies, falls nur sehr wenig Energie ausgetauscht wird bzw. Cluster sehr weit voneinander entfernt plaziert sind. Ein Cluster enthält weitere Elemente, dies können Cluster, wie auch Rootpatches sein, desweiteren wird der Cluster durch seine Bounding Box repräsentiert. Eine wichtige Vorraussetzung für das hierarchische Radiosity-Verfahren mit Clustering ist allerdings, dass die Szene in einer sinnvollen geometrischen hierachischen Struktur vorliegt, wie z.B. eine Szene, die aus einer Sitzgruppe und einer Lampe besteht, die Sitzgruppe wiederum aus einem Tisch und zwei Stühlen, die Stühle bestehen aus Lehne, Sitzfläche und Beine. Die Beine sind aus sechs Rootpatches aufgebaut (siehe Abb. 16). Es ist also schon beim Modellieren einer Szene darauf zu achten, sinnvolle hierachische Informationen zu integrieren bzw. sollten Algorithmen eingesetzt werden, die aus einer eingeladenen Szene automatisch Cluster (z.B. anhand der Plazierung der Rootpatches) generieren, damit das hierarische Radiosity mit Clustering gute Ergebnisse mit enormer Zeiteinsparung liefern kann.

Das standard hierarchische Radiosity-Verfahren wird somit mit Clustern erweitert, die eine Gruppe von Rootpatches bzw. weitere Cluster beinhalten. Der Tisch könnte z.B. mit einem Stuhl nach der 1. Iteration[13] Energie austauschen. Beim Visibilitätstest spart man auch an Aufwand da wesentlich weniger Tests durchgeführt werden müssen, wenn z.B. ein Cluster durch eine Wand verdeckt wird müssen nicht mehr Visibilitättests zu allen enthaltenden Rootpatches des Clusters durchgeführt werden, denn ein einziger Test genügt um festzustellen dass der gesamte Cluster nicht sichtbar ist. Zur Veranschaulichung, dass das Clustering die Anzahl der Links

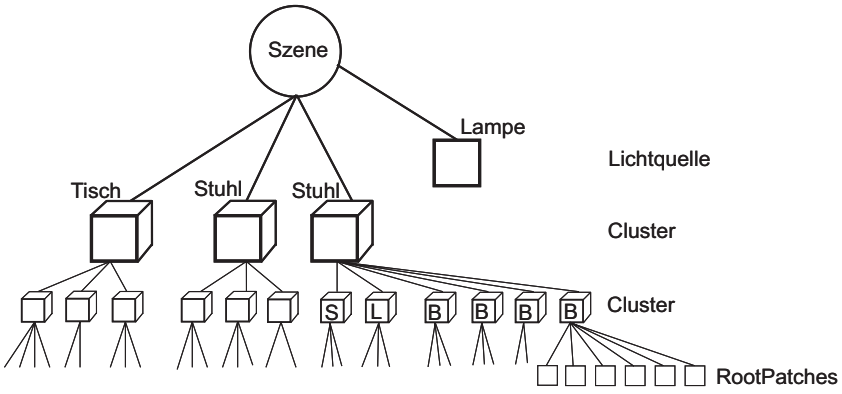


Abbildung 16: Ausschnitt einer hierarchischen Szenenanordnung mit Cluster

bedeutend reduzieren kann, geben Smits *et al.* folgendes einfache Beispiel: Eine einfaches Szenario besteht aus zwei Stühlen, jeder Stuhl setzt sich aus 100 Patches zusammen. Bei einer Standard-Verlinkung müssten 10.000 Interaktionen getestet werden (siehe Abb. 17, 18 und 19). Doch könnten die Interaktion schon mit einem Link (β-Link, die Energie wird konstant an alle Empfängerpatches weitergegeben)

[13] Nach der 1. Iteration ist der Radiositywert des Tisches gesetzt.

bzw. mit dem Aufwand von 200 Links (α-Link, die Energie wird erst über alle Patches des Sendercluster entsprechend der Ausrichtungen berechnet und dann auf alle Patches des Empfängers in Abhängigkeit derer Ausrichtung verteilt) repräsentiert werden.

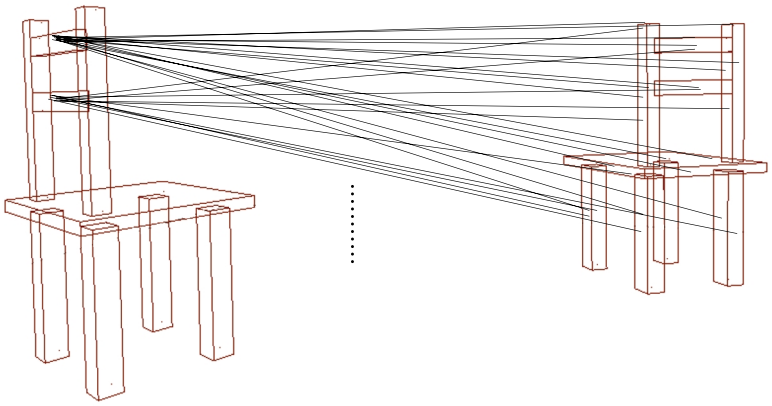


Abbildung 17: Standard HR-Links, quadratischer Aufwand

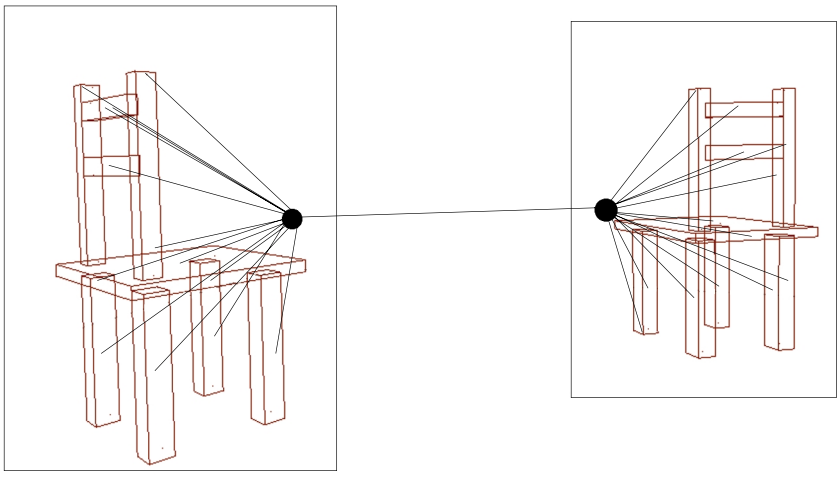


Abbildung 18: α-Link, linearer Aufwand

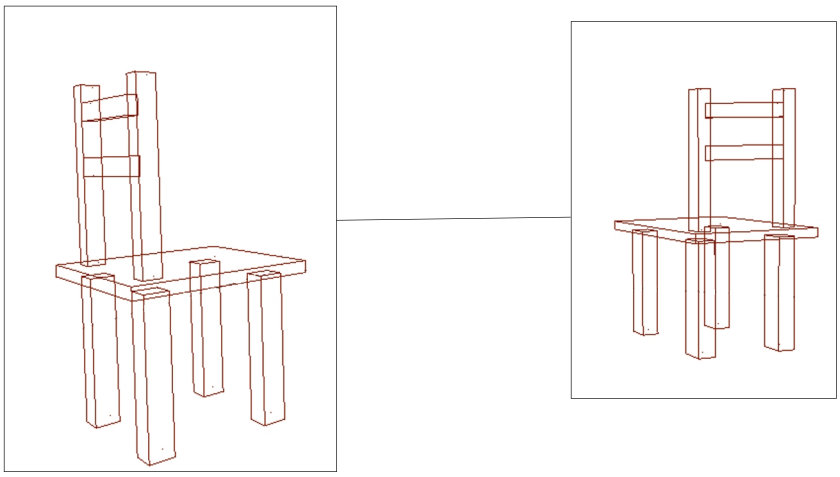


Abbildung 19: β-Link, konstanter Aufwand

3.2 Der Ablauf des HRC - Verfahrens

Ein großer Vorteil des hierarchischen Radiosity mit Clustering ist, dass das aufwendige initiale Linking nicht benötigt wird. Die Szene besteht aus einem Rootcluster, der alle weiteren Elemente enthält, die sowohl Rootpatches, wie auch weitere Cluster sein können. Die einzigen Links die initial erstellt werden, sind β-Links von den Lichtquellen zum Rootcluster, die beim Rootcluster gespeichert werden, desweiteren erhält der Rootcluster einen Link auf sich selbst, ein sogenannter Selflink (siehe Abb. 20). Der Selflink repräsentiert alle Ineraktionen untereinander aller Elemente, die dem entsprechenden Cluster zugeordnet sind, und wird nur dann verfeinert, wenn genügend Energie innerhalb des Clusters zu verteilen ist. Zu Anfang wird beim HRC wie auch beim HR die *refine()* Methode aufgerufen, alle Links werden wiederum getestet, wobei hier die Links nach β, α und Standard-Links unterschieden werden. Liegt ein β-Link vor wird getestet, ob dieser ausreicht, falls nicht wird getestet ob ein α-Link ausreicht, falls dieser auch nicht ausreicht, wird der Cluster verfeinert und entsprechend neue Links erstellt. Bei Patch zu Patch Links kommt das normale HR-Verfahren zum Zuge. Der zweite Schritt beim HRC ist genau wie bei HR die *gather()*-Methode, die entsprechend erweitert wurde, nach dem *gathering* muss auch hier noch *pushpull()* ausgeführt werden um die Radiositywerte wieder ins Gleichgewicht zu bringen. Die einzelnen Schritte mit den entsprechenden Erweiterungen werden in den nun folgenden Abschnitten erläutert.

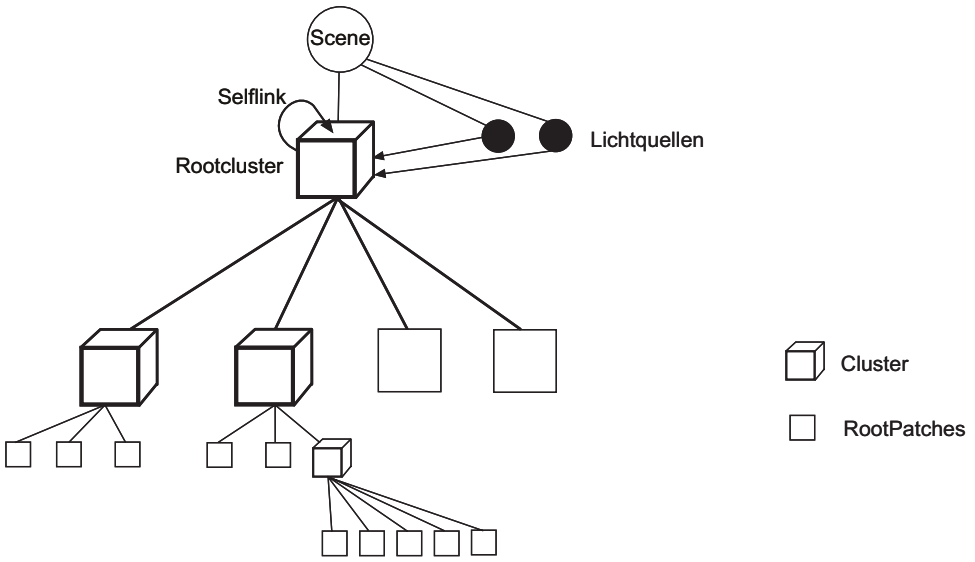


Abbildung 20: Initialer Zustand einer hierarchische Szene mit Cluster

3.3 Beta-Link

Smits *et al.* stellen zwei unterschiedliche Linkstrategien vor, für jeden Linktyp muss ein Fehler abgeschätzt werden. Wenn dieser berechnete Fehlerwert unterhalb des vom Benutzer vorgegeben $\alpha/\beta - boundary$ - Wert liegt, ist der entsprechende Link ausreichend und zwischen den beiden Clustern findet mittels der zugehörigen *Gather*-Routine eine Energieübertragung statt. Viele Interaktionen der Szene haben mit Sicherheit eine geringere Wirksamkeit auf die zu beleuchtende Szene. Diese Übertragungen können daher durch Links, die zwar eine grössere Fehlerabweichung mit sich bringen, aber mit recht geringem Aufwand zu berechnen sind, repräsentiert werden. Die Fehlerabschätzung des β-Links ist recht einfach und Bedarf nur einem konstanten Aufwand, der sogenannte $\beta - Error$ enthält auch keinerlei Informationen über die Ausrichtung der Patches der interagierenden Cluster. Demnach lässt sich der maximale Formfaktor zwischen zwei Cluster recht einfach abschätzen, wobei d_{min} der minimale Abstand zwischen zwei Clustern entspricht. Der minimale Abstand kann recht einfach über die Bounding Volumes der Cluster bestimmt werden, da diese achsenparallel sind. Der minimale Abstand ist gleich null, falls sich die Bounding Volumes berühren bzw. überschneiden. In dem Fall ist der grössere Cluster zu unterteilen.

$$FF_{max} \approx \frac{1}{d_{min}^2} \tag{31}$$

Der $\beta - Error$ setzt sich nun aus folgenden Komponenten zusammen:

$$\beta_{error} = FF_{max} \cdot B_s \cdot A_s \tag{32}$$

Falls β_{error} unterhalb der vom Benutzer vorgebenen Schranke liegt ist der β-Link ausreichend und eine weitere Unterteilung ist nicht mehr nötig. Ansonsten wird ge-

testet, ob die Fehlerabschätzung des α-Link unterhalb der $\alpha/\beta - boundary$ liegt.

3.4 Alpha-Link

Beim α-Link werden allerdings die Orientierungen der Patches im Cluster mitberücksichtigt. Wenn jedoch alle *n* Patches des Senderclusters mit allen *m* Patches des Empfängerclusters interagieren liegt ein quadratischer Aufwand von $O(n * m)$ vor. Smits *et al.* splitten die Energieübertragung vom Sender- zum Empfängercluster nach Sender- und Empfängeranteilen so, damit diese Anteile getrennt voneinander berechnet werden können. Man erhält für den Senderanteil folgende Formel, wobei d nun dem Abstand vom Mittelpunkt des Patches des Senderclusters zu dem Mittelpunkt des Empfängerclusters entspricht.

$$F_s = \frac{\cos\theta_s}{d^2} \tag{33}$$

und für den Empfänger folgende Gleichung:

$$F_e = \rho_e \cdot \cos\theta_e \tag{34}$$

Vorab werden nun alle n Senderpatches abgearbeitet, d.h. alle maximalen Senderanteile werden aufaddiert:

$$error_{source} = \sum_{n=0}^{n} F_{n,max} \cdot A_n \cdot B_n \tag{35}$$

Den maximalen Formfaktor erhält man durch Berechnung einiger Testpunkte (z.B. Mittelpunkt und Eckpunkte).

Im nächsten Schritt wird die maximal ankommende Energie aller m Patches des Empfängerclusters berechnet:

$$error_{max,e} = \max(F_{e,max} \cdot error_{source}) \tag{36}$$

Auch hier berechnet sich $F_{e,max}$ über Testpunkte auf dem Empfängerpatch. Das Maximum aller m $error_{max,e}$ - Werte liefert den Wert der Fehlerabschätzung, den α-error (siehe u.a. auch Algorithmus 3.1).

$$error_\alpha = \max(error_{max,1}, \cdots, error_{max,m}) \tag{37}$$

Falls der α-error unterhalb des gegebenen α/β-Schranke liegt ist der Link ausreichend, falls nicht wird wiederum der grössere Cluster unterteilt.

Der Aufwand des α-Links ist nun um einiges geringer, als der Aufwand von entsprechenden Standard Links zwischen allen Patches der interagierenden Cluster. Der Aufwand hat sich vom Quadratischen $O(n * m)$ auf einen nur noch linearen Aufwand $O(n + m)$ reduziert.

Algorithmus 3.1 Pseudo Code zur Abschätzung des Alpha-Errors

```
getAlphaError(Cluster Cs,Cluster Cr)
{
  src_error = 0;
  max_error = 0;
  FOR EACH patch i in Cs
    scr_error = src_error + Fmax(patch[i], Cr) * Area(patch[i]) * B(Patch[i]);
  FOR EACH patch j in Cr
    max_error = max(max_error, Fmax(Cs, patch[j]) * src_error);

  RETURN max_error;
}
```

3.5 Refinementkriterien

Es gibt verschiedene Refinementkriterien, die bei den Clustern von Bedeutung sind. Cluster selbst müssen nicht unterteilt werden, da sie ja selbst aus elementaren Objekten bestehen, nur Links werden hier entsprechend verfeinert, wenn ich demnach von dem Cluster verfeinern, schreibe ist letztendlich das Verfeinern der entsprechenden Links gemeint.

3.5.1 Besonderheiten beim Linking

Zu aller Anfang wird beim HRC die *refine()* - Methode auf den RootCluster angewandt. Alle Links des RootClusters werden durchlaufen. Initial liegen nur β-Links von den Lichtquellen zum RootCluster und ein Selflink vor. Zuerst wird getestet ob sich die Bounding Volumes des Sender und Empfängers überschneiden, wenn dies der Fall ist wird der grössere Cluster unterteilt, d.h. der Link wird gelöscht und entsprechen neue Links zu den Elementen des unterteilten Clusters werden generiert und den entsprechenden Empfänger der Linkliste hinzugefügt. Falls ein Patch-Cluster Link vorliegt, und Empfänger und Sender überschneiden sich, wird immer der Cluster unterteilt. Die Lichtquellen liegen immer innerhalb des RootClusters, da der RootCluster die gesamte Geometrie enthält, d.h. ein Lightsource-Cluster Link wird beim ersten Aufruf von *refine()* immer unterteilt. Es werden also neue Links von der Lichtquelle zu den Elementen des Clusters generiert. Die Elemente des Clusters werden durchlaufen und mit der Lichtquelle verlinkt, falls ein Cluster vorliegt wird ein neuer β-Link von Lichtquelle zu Cluster generiert, liegt ein RootPatch vor wird ein Standard-Link von Lichtquelle zu Rootpatch erzeugt, dieser wird nach den Refinementkriterien des HR-Verfahrens untersucht.

Liegt ein Selflink vor, wird getestet ob innerhalb des Clusters genügend Energie vorhanden ist, die es zu verteilen gilt, falls B_{max} sehr klein, oder gleich null ist, lohnt sich das verfeinern des Selflinks nicht. Diese Unterteilung wird mit einer vom Benutzer zu regulierende Schranke gesteuert und ist mit $\epsilon_{Selflink}$ gekennzeichnet. Wenn dieser Wert zu klein ist, könnten zu viele Selflinks von Clustern verfeinert werden, im schlechtesten Fall würde dies wieder dem *initial linking* entsprechen.

Der Selflink des RootClusters wird initial auch immer verfeinert, da B_{max} des Rootclusters dem maximalen Radiosity-Wert aller Lichtquellen entspricht. Die Verfeinerung des Selflinks bedeutet, dass alle Elemente innerhalb des Clusters untereinander verlinkt werden, zusätzlich werden, falls ein Element einem Cluster entspricht ein neuer Selflink angelegt (siehe Abb. 21).

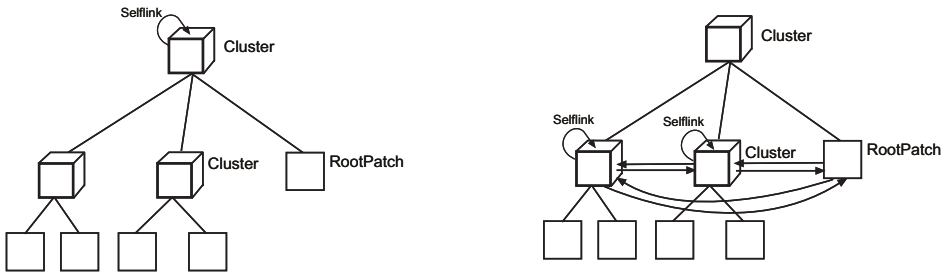


Abbildung 21: Links: Cluster besitzt einen Selflink, der rechts im Bild entsprechend verfeinert worden ist.

Als nächsten Schritt werden nun alle Elemente des RootClusters durchlaufen und deren Links untersucht. Folgende Linktypen können vorliegen:

- Selflink
- β-Link
- α-Link
- Standardlink

Ein Selflink, wie vorab gesehen wird genau dann generiert, wenn ein Selflink eines Clusters verfeinert wird und dieser wiederum Cluster enthält, die inneren Cluster erhalten neue Selflinks. Ein β-Link liegt genau dann vor, wenn mindestens ein Cluster bei der Interaktion beteiligt, d.h. β-Links können zwischen Patch und Cluster, wie auch Cluster und Cluster und auch Lichtquelle und Cluster erzeugt werden. Ein β-Link ist genau dann ausreichend für den Energietransfer, wenn der β-Error unter der Fehlerschranke liegt. Ein α-Link wird dann generiert, wenn ein β-Link nicht ausreicht, liegt der α-Error unterhalb der Fehlerschranke ist der Link ausreichend, wenn nicht wird der Link weiter verfeinert und es können neue Standard-, α- und β-Links entstehen. Ein Standard Link wird erzeugt wenn nur noch Patches interagieren, dieser Link wird mittels den Refinemnetkriterien des HR-Verfahrens weiter untersucht.

3.5.2 Die *refine()*-Methode

Wie bereits mehrfach erwähnt, entscheidet die *refine()*-Methode, ob ein Link verfeinert werden muss oder ob er ausreicht. Drei Linktypen werden in der *refine()*-

Routine beim HRC unterschieden, der Selflink, β-Link und α-Link. Wie mit dem Selflink verfahren wird wurde im vorangegangenen Kapitel schon erläutert. Beim β-Link, wird nun zwischen Sendern und Empfängern unterschieden. Ist der Sender ein Patch und Empfänger ein Cluster, wird der Cluster verfeinert, falls sich Sender und Empfänger überschneiden. Ist dies nicht der Fall wird untersucht ob B_{max} des Senders grösser null, bzw. die Visibilität grösser null ist, trifft dies nicht zu braucht der Link nicht weiter untersucht werden. Wird der Empfängercluster durch ein Objekt zum Teil verdeckt, wird der Cluster unterteilt. Wenn keiner dieser Fälle eingetreten ist, und ein β-Link vorliegt, wird der β-Error berechnet und getestet ob dieser ausreicht, wenn nicht wird der Link gelöscht und ein α-Link angelegt und entprechend getestet, sollte dieser auch nicht ausreichen wird er wieder gelöscht und dementsprechend verfeinert. Analog verfährt man mit den anderen Link-Kombinationen, wobei beim Cluster-Cluster Link, im Falle einer Unterteilung, immer der grössere Cluster zu unterteilen ist.

3.6 Radiosityausgleich

Der Energieaustausch beim HRC benötigt gegenüber dem normalen HR-Verfahren einige Ergänzungen. Das *Gathern* muss entsprechend erweitert werden und die *pushpull*-Methode hat bei Clustern eine leicht abweichende Form.

3.6.1 Die *gather()*-Methode

Die *gather()*-Routine unterscheidet beim HRC zwischen β- und α-Links. Liegt ein β- Link von Cluster zu Cluster vor gestaltet sich das einsammeln der Radiosity recht einfach:

$$gather_{rec} = gather_{rec} + FF_{avg} \cdot B_s \cdot vis \tag{38}$$

Der mittlere Formfaktor FF_{avg} muss allerdings noch abgeschätzt werden, dies kann über einige Testpunkte geschehen, was natürlich wieder einen relativen Aufwand mit sich bringt, oder man schätzt den mittleren Formfaktor sehr grob ab. Der β-Link nutzt keinerlei Informationen über die Orientierung der enthaltenen Patches im Cluster, von daher könnte man ganz grob $\cos\theta_s$ und $\cos\theta_e$ abschätzen. Wenn man von einer gleichverteilten Ausrichtung der Patches ausgeht, wäre $\cos\theta_s = \cos\theta_e = \frac{1}{4}$ eine recht gute Abschätzung. Daraus erhält man dann durch Einsetzen der Werte in die 2. Formfaktorvereinfachung folgenden gemittelten Formfaktor:

$$FF_{avg} = \frac{\frac{1}{4} \cdot \frac{1}{4} \cdot A_s}{\pi d^2} \tag{39}$$

Wobei A_s die Gesamtfläche, genauer gesagt die Summe aller Patchflächen des Senderclusters entspricht.

Liegt ein β- Link von Cluster zu Patch vor, berechnet man das *Gathering* wie folgt:

$$gather_{rec} = gather_{rec} + FF_{es} \cdot \rho_e \cdot B_s \cdot vis \tag{40}$$

Der Formfaktor wird auch in diesem Fall abgeschäzt, wobei diesmal $\cos\theta_e$ bekannt ist.

$$FF_{es} \approx \frac{\frac{1}{4} \cdot \cos\theta_e \cdot A_s}{\pi d^2} \tag{41}$$

α-Links werden nun allerdings etwas aufwendiger behandelt. Liegt ein α-Link zwischen zwei Clustern vor, so muss vorab der Senderanteil aufsummiert werden, dieser Senderanteil wird dann mit jedem mittleren Formfaktor der Empfängerelemente im Empfängercluster multipliziert und aufaddiert:

$$B_{s,avg} = \sum_{i=1}^{n} F_{i,avg} \cdot B_i \tag{42}$$

$$gather_r = gather_r + F_{e,avg} \cdot B_{s,avg} \cdot vis \tag{43}$$

Die gemittelten Formfaktoren werden schon beim Generieren des α-Links berechnet. (z.B. durch Testpunkte) und in einem Array abgelegt.

Liegt ein α-Link zwischen einem Cluster und einem Patch vor, so kann das *Gathern* etwas vereinfacht werden. Man berechnet eine gerichteten Senderradiosityanteil der sich wie folgt zusammensetzt:

$$B_{dir} = \sum B_{Si} \cdot A_{Si} \cdot \cos\theta_{Si} \tag{44}$$

Alle einzelnen Energieanteile der Patches im Sendercluster werden somit aufaddiert, Der Formfaktor lässt sich wie folgt abschätzen:

$$F_{es} = \frac{\cos\theta_e}{\pi d^2} \tag{45}$$

Das *Gathern* für das Empfängerpatch hätte nun folgende Gestalt:

$$gather_r = gather_r + \rho_e \cdot B_{dir} \cdot F_{es} \cdot vis \tag{46}$$

Wie man sieht bedarf es nun beim *Gathern* schon an einigen Fallunterscheidungen, diese sind dann entsprechend zu erweitern, wenn neue Elementtypen dem RADIOCITY-System hinzugefügt werden sollten, wie z.B die Face-Cluster. An dieser Stelle soll nun nur noch kurz auf einen Pseudo-Code der *Gather()*-Methode hingewiesen werden, der in Algorithmus 3.2 zu sehen ist.

3.6.2 Die *pushpull()*-Methode

Wie auch beim klassischen HR-Verfahren, müssen beim HRC die eingesammelten Radiosity-Werte wieder ausgeglichen werden. Dies unterscheidet sich etwas vom HR-Verfahren (siehe Algorithmus 3.3). Beim HRC-Verfahren werden die Beleuchtungsstärken E der Cluster beim *Push*-Schritt bis zum Rootpatch aufaddiert, die

Algorithmus 3.2 Pseudo Code zum Einsammeln der Radiosity bei einem Alpha-Link zwischen 2 Clustern

```
gather(Cluster Cs, Cluster Cr)
{
  gather = cr->getGather();
  src_rad = 0;
  FOR EACH patch i in Cs
    scr_rad = src_rad + Favg(patch[i],Cr) * Area(Patch[i]) * B(patch[i]);
  scr_rad = src_rad * vis;
  FOR EACH patch j in Cr
    gather = gather + Favg(Cs, patch[j]) * src_rad;
  cr->setGather(gather);
}
```

Summe wird mir E_{down} bezeichnet. Ab dem Rootpatch wird B_{down} bestimmt (siehe Gleichung 47), und das „normale“ *pushpull* des klassischen HR-Verfahrens wird mit B_{down} aufgerufen.

$$B_{down} = \rho_{RootPatch} \cdot E_{down} + B_{gather} \tag{47}$$

Der *Pull*-Schritt ist identisch wie beim HR-Verfahren, auch hier werden die Radiosity-Werte der Cluster flächengewichtet nach oben weitergegeben.

Algorithmus 3.3 Pseude-Code von PushPull beim HRC

```
pushpull(E_down)
{
  E_down = E_down + gather;
  IF Cluster is a leaf THEN
    FOR EACH patch j in Cluster
    {
      B_down = B_gather[j] + E_down * rho(patch[j]);
      B_up = B_up + patch[j]->pushpull(B_down) * Area(patch[j]/Area of Cluster;
    }
  ELSE
    FOR EACH cluster i in Cluster
    {
      B_up = B_up + cluster[i]->pushpull(E_down) * Area(cluster[i])/Area of Cluster;
    }

  B_Cluster = B_up;
  gather = 0;
  return B_up;
}
```

4 Spezielle Lichtquellen

Bisher wurden nur Flächenlichtquellen, die sogenannten Lambertemitter behandelt. Es können aber auch weitere Lichtquellen in das System integriert werden. Diese müssen zu Anfang in die Lightsourceliste hinzugefügt werden und ein entprechender Link von der Lichtquelle zum Rootcluster generiert werden. Desweiteren

müssen natürlich die entsprechneden Refinementkriterien und die *Gather* Methode angepasst werden. Im RADIOCITY-System wurden bisher Punktlichtquellen und Spotlights integriert, auf die Refinementkriterien und *Gather*-Methoden soll nun näher eingegangen werden.

4.1 Pointlight

Pointlights stellen eine sehr einfache Lichtquelle dar, sie besitzen keinerlei Geometrie und geben Licht in alle Richtungen des Raumes gleichmässig ab. In der Realität gibts solche Lichtquellen allerdings nicht. Bei diesen speziellen Lichtquellen findet die sogenannte Transferfunktion[14] Verwendung, dieser Faktor gibt an wie hoch der Anteil der Energie des Senders ist, die beim Empfänger ankommt. Patches haben eine Ausrichtung, Cluster allerdings nicht, daher ist hier eine Fallunterscheidung zwischen Empfängertypen zu treffen. Die Transferfunktion würde demnach für Empfängerpatches folgende Form haben:

$$T_{es} = \frac{\cos\theta_e}{\pi d^2} \tag{48}$$

und für Cluster:

$$T_{es} = \frac{1}{\pi d^2} \tag{49}$$

Wann ist denn nun ein Link von einer Punktlichtquelle zu einem Patch bzw. Cluster zu unterteilen. Es werden Informationen über die zu erwartende Energieverteilung benötigt.

Beim Patch wird ähnlich wie bei den Standardlinks verfahren, d.h. man benötigt T_{max} und T_{min} um Aussagen über die Energieverteilung auf dem Empfänger treffen zu können. Desweiteren benötigt man die Visibilitätswert um evtl. Schattenkannten genügend fein zu unterteilen.

T_{max} und T_{min} lassen sich recht einfach bestimmen. T_{min} ist, wie auch F_{min} über die Eckpunkte des Empfängerpatches zu bestimmen, man berechnet alle T_{es} Werte der Eckpunkte, der minimalste Wert entspricht dem T_{min}. T_{max} lässt sich auch noch relativ einfach bestimmen, da eine Punktlichtquelle nach allen Richtungen gleichmässig Energie abgibt. Die Position der Punktlichtquelle im rechten Winkel auf die Patchebene projeziert, liegt dieser Punkt innerhalb des Patches hat man bereits den Punkt gefunden der den grössten Wert für die Transferfunktion hergibt gefunden. Falls der projezierte Punkt außerhalb des Patches liegt, nimmt man den Punkt auf dem Patch der den kürzesten Abstand zum projezierten Punkt hat. Wenn nun

$$I^s \cdot (T_{max} - T_{min}) > \epsilon_{PointlightToPatch} \tag{50}$$

ist, ist zu unterteilen. Die Visibilität wird analog, wie beim Refinementkriterium der Patches mit berücksichtigt.

[14] Die Transferfunktion gleicht im wesentlichen dem Formfaktor

Bei einem Link von Pointlight zu Cluster werden vorab ein paar wenige Fallunterscheidungen berücksichtigt, ob unterteilt werden soll oder nicht. Zu einem wird auch wieder die Visibilität ähnlich wie bei den Patches mit einbezogen, bei totaler Verdeckung wird natürlich nicht unterteilt, bei partieller Verdeckung dagegen ist zu unterteilen. Desweiteren wird getestet ob das Pointlicght innerhalb des Clusters liegt, in diesem Fall ist zu unterteilen. Ansonsten wird ein Cluster bei einem Pointlight-Cluster Link genau dann unterteilt, wenn die maximal zu übertragende Energie zum Cluster über einer vorgegebenen Schranke liegt. Man benötigt den maximalen T_{es} Wert, der sich ganz einfach mittels des kürzten Abstands vom Pointlight zum Cluster bestimmen lässt:

$$T_{max} = \frac{1}{\pi d_{min}^2} \tag{51}$$

Ob der Cluster unterteilt wird oder nicht kann man dann mittels folgender Gleichung bestimmen:

$$\frac{I^s}{\pi d_{min}^2} > \epsilon_{PointlightToCluster} \tag{52}$$

Die *gather*-Methoden sind auch recht einfach und ähneln den uns schon bekannten *gather*-Routinen, mit dem einzigen Unterschied, dass die Punktlichtquelle einen Wert für für die Lichtstärke I^s besitzt. Mit der *send()*-Methode ist bei RADIOCITY die zu übertragende Energie der Punktlichtquelle abzurufen. Die beiden *Gather*-Varianten für Cluster und Patch als Empfänger haben folgende Form:

$$Patch : B_{gather} = B_{gather} + \rho_e \cdot I^s \cdot T_{es} \cdot vis \tag{53}$$

und

$$Cluster : B_{gather} = B_{gather} + I^s \cdot T_{es} \cdot vis \tag{54}$$

Beim Cluster ist allerdings zusätzlich darauf zu achten, dass die eingesammelte Radiosity B_{gather} bis zum Rootpatch gepusht wird und dort $cos\theta_e$ hineinmultipliziert wird, damit nicht alle Rootpatches des Clusters den gleichen Anteil an Energie erhalten.

4.2 Spotlight

Die Refinementkriterien beim Spotlight erweisen sich als wesentlich komplexer im Vergleich zum einfachen Pointlight. Ein Spotlight wird definiert durch Position, Richtung, den Exponent n, einem Cutoff, der Lichtstärke und der Farbe. Die Spotlichtquelle strahlt Licht nur um einen bestimmten Winkel θ_{max}, dem sogenannten Cutoff um eine angegebene Richtung ab, je weiter man sich von dieser Richtung entfernt umso schwächer wird das Licht, bedingt durch die cos^n-Verteilung. Die Transferfunktion der Spotlichtquelle ist folgendermaßen definiert:

$$T_{es} = \frac{\cos^n \theta_s \cos \theta_e}{\pi d^2}, \theta_s < \theta_{max} \tag{55}$$

Um nun wiederum Aussagen über die Energieverteilung auf dem Empfänger zu treffen, sollte T_{max} und T_{min} bekannt sein.

Ist der Empfänger ein Patch, so lässt sich T_{min} wie bei der Punktlichtquelle über die Eckpunkte bestimmen. Wie man leicht erkennen kann ist die Bestimmung T_{max} um einiges schwieriger, zu einem weil die Funktion durch die Cutoff-Schranke nicht stetig ist, und zu anderem dass es mehrere variable Parameter gibt. Die Idee hierbei ist es, die Funktion zu differenzieren um das Maximum zu finden. Dafür sollte die Funktion zuvor entsprechend umgeformt werden, damit dies möglich ist. Dieser Vorgang gleicht dem Verfahren zur Bestimmung von ff(x) in Kapitel 2.4.1. Die Transferfunktion wird auch durch eine zweidimensionale Interpretation beschrieben. Der Punkt mit dem gesuchten Maximum befindet sich auf der Geraden, die durch senkrechter Projektion der Spotlightposition auf die Patchebene und des projezierten Richtungsvektors $\vec{dir}$ der Spotlichtquelle definiert. Wie bei der Suche von ff(x) wird auch die Szene zweidimensional interpretiert. Die vorab definierte Gerade wird als x-Achse aufgefasst, der projezierte Positionspunkt der Lichtquelle ist der neue Ursprung, die y-Achse wird durch den normierten Richtungsvektor interpretiert (siehe Abb. 22). Die Funktion muss nun dementsprechend umformuliert werden, damit ein maximaler Wert für x bestimmt werden kann. Dadurch müssen vorab die Winkel, der Richtungvektor, die Empfängernormale im neuen Koordinatensystem beschrieben werden, siehe dazu Kapitel (2.4.1). Um nun die Transferfunktion

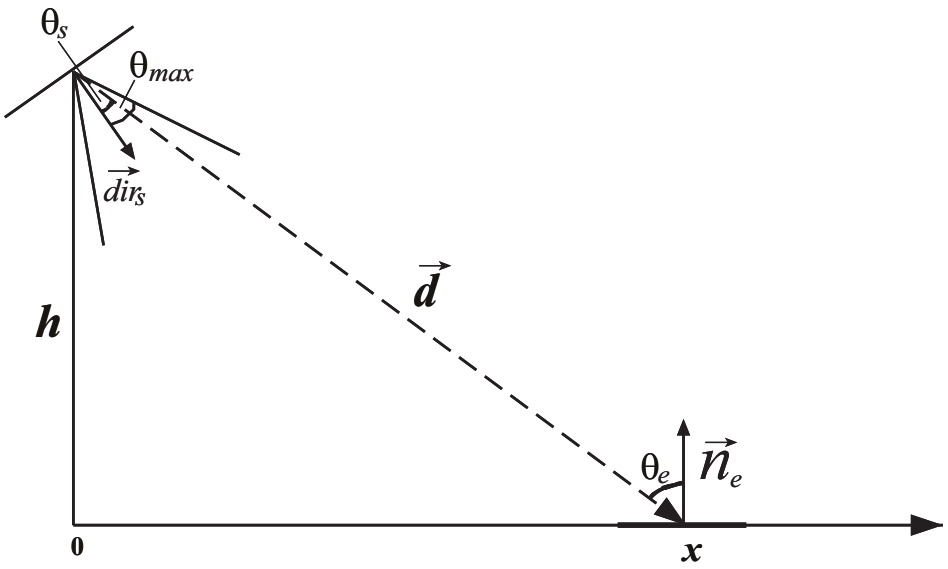


Abbildung 22: Zweidimensionale Interpretation der Szene um T_{max} zu bestimmen

durch die in Kapitel (2.4.1) erläuterten Werte neu zu formulieren, bedarf es vorb einer Erweiterung der Formel mit d^{n+1}, daraus ergibt sich:

$$T_{es} = \frac{d^n \cdot \cos^n \theta_s \cdot d \cdot \cos \theta_e}{\pi d^2 \cdot d^{n+1}}, \tag{56}$$

mit $d^2 = x^2 + h^2$ bzw. $d = \sqrt{x^2 + h^2}$ sowie $d \cdot \cos\theta_e = h$
und $d \cdot \cos\theta_s = x \cdot dir_x - h \cdot dir_y$ ergibt sich:

$$T(x) = \frac{(x \cdot dir_x - h \cdot dir_y)^n \cdot h}{\pi(x^2 + h^2)^{\frac{n+3}{2}}} \tag{57}$$

Wir haben das Maximum für einen x-Wert genau dann gefunden, wenn die erste Ableitung an dieser Stelle gleich null ist. Die 1.Ableitung von $T(x)$ ist wie folgt definiert:

$$T'(x) = \frac{h}{\pi} \cdot \frac{2 \cdot dir_y + h \cdot (3+n) \cdot x + dir_x \cdot (h^2 \cdot n - (6+n) \cdot x^2)}{(dir_x \cdot x - dir_y \cdot h)^{n-1} \cdot (x^2 + h^2)^{4+n}} \tag{58}$$

Die Nullstelle dieser Funktion zu bestimmen erweist sich als äußerst schwierig, falls es überhaupt möglich sein sollte diese direkt zu berechnen. Eine recht gute Annäherung, die auch in RADIOCITY Anwendung findet, ist, sich der Nullstelle mittels binärer Suche zu nähern. Zwei Fälle müssen dabei betrachtet werden:

Fall 1: Der Richtungsvektor der Spotlichtquelle zeigt in Richtung der Empfängerebene[15].

Fall 2: Der Richtungsvektor ist parallel[16] bzw. zeigt von der Empfängerebene weg[17]. Falls ein Schnittpunkt im positiven Bereich der x-Achse liegt, ist das Maximum genau in dem Intervall zwischen Ursprung und Schnittpunkt zu finden. Mittels der binären Suche, d.h. man berechnet den Wert für den Intervallanfang, ist dieser Wert grösser null, ist die Funktion an dieser Stelle steigend, man berechnet den Wert für das Intervallende, ist dieser größer gleich null hat man das Maximum bereits gefunden, ansonsten teilt man das Intervall und berechnet den Wert in der Mitte, ist dieser Wert kleiner null liegt das Maximum links, und man unterteilt wiederum die linke Hälfte. Ist der Wert grösser null unterteilt man die rechte Hälfte usw. Man fährt so lange mit der Intervallschachtelung fort, bis man eine bestimmte Rekursionstiefe erreicht hat. Der so gefundene x-Wert liefert eine recht gute Approximation für die Position auf der Empfängerebene, die den maximalen Transferfunktionswert liefert. Der x-Wert muss allerdings noch zurück in 3D-Koordinaten berechnet werden. Liegt der Punkt innerhalb des Patches und innerhalb des Cutoffs hat man das Maximum gefunden, liegt der Punkt außerhalb des Patches, liefert der Punkt auf dem Patch mit dem kürzesten Abstand zum berechnet Punkt recht gute Ergebnisse, allerdings ist auch hier zu beachten, dass er innerhalb des Cuttoffs liegt. Es wäre daher sinnvoller vorab den berechneten x-Wert darauf zu testen ob er innerhalb des Cuttoffs liegt, falls nicht wird der Punkt, an die Stelle entlang der x-Achse in Richtung des Patches verschoben bis man innerhalb des Cuttoffs liegt.

Fall 2 kann fast genauso behandelt werden, allerdings ist hier kein Intervall bekannt in dem das Maximum liegen könnte, eine Möglichkeit wäre, der Abstand der

[15] Es existiert damit ein Schnittpunkt im postiven Bereich der x-Achse
[16] Es existiert kein Schnittpunkt
[17] Der Schnittpunkt liegt im negativen Bereich der x-Achse

Lichtquelle zur Patchebene als Startintervall auf die x-Achse zu legen. Falls nun am Intervallende der Wert der Ableitung immer noch positiv ist, wird das Intervall verdoppelt usw., ist der Wert am Intervallende kleiner null liegt das Maximum links, ab jetzt werden die Intervalle wieder halbiert, bis man wie vorab die maximal erlaubte Rekursionstiefe erreicht hat, nun verfährt man weiter wie in Fall 1. T_{max} für eine Spotlight-Patch Energie Transfer wäre damit bestimmt.

Was nun noch fehlt, ist die T_{max} und T_{min} - Bestimmung bei Spotlight-Cluster Links. T_{min} ist hier auch wieder über die Berechnung der Transferfunktion (siehe Formel 59) der Eckpunte zu bestimmen, der minimalste Wert entspricht dem T_{min}. T_{es} ist bei einer Spotlight-Cluster Interaktion definiert als:

$$T_{es} = \frac{\cos^n \theta_s}{\pi d^2}, \theta_s < \theta_{max} \tag{59}$$

Dieser Wert wird maximal, wenn θ_s minimal und d minimal sind. d_{min} ist der minimalste Abstand von Lichtquellenposition zum Cluster und lässt sich recht einfach über Bounding Box Werte des Clusters bestimmen.

θ_{min} lässt sich über folgendes Verfahren recht einfach bestimmen, mein Dank geht an dieser Stelle an unseren Betreuer Dipl.-Inform. Thorsten Grosch, der mir des öfteren einige gute Ideen im Laufe dieser Studienarbeit unterbreitet hatte. Die Idee hierbei ist es, den Strahl mit minimalen Winkel θ_{min} zu finden, der den Cluster gerade berührt. Dazu wird eine Kugel um die Bounding Box des Clusters gelegt, der Mittelpunkt entspricht dem Mittelpunkt des Clusters und der Radius wird bestimmt als die Hälfte der grösseren Raumdiagonalen: $r = \frac{D}{2}$ (siehe Abb. 23). L ist der Abstand von der Lichtquellenposition zum Kugelmittelpunkt. Falls nun $L \leq r$ ist, liegt die Lichtquelle innerhalb des Clusters und der Cluster ist zu unterteilen. Der andere Fall $(L > r)$ muss nun weiter untersucht werden. $\vec{L_0}$, $\vec{d_0}$, r und L sind bekannt, darüber lassen sich θ und α bestimmen. Der Winkel α, der Winkel der den Kugelradius aufspannt ist definiert als $\alpha = \arcsin \frac{r}{L}$ und aus $\cos\theta = \vec{d_0} \circ \vec{L_0}$ folgt $\theta = \arccos \vec{d_0} \circ \vec{L_0}$.
Ist nun $\theta < \alpha$ trifft der Richtungsvektors $\vec{d_0}$, indem man ihn entsprechend verlängert, den Cluster und somit ist der gesuchte Winkel $\theta_{min} = 0$, ansonsten berecnet sich θ_{min} aus der Differenz von θ und α: $\theta_{min} = \theta - \alpha$. Liegt nun jedoch θ_{min} ausserhalb des Cutoffs θ_{max} der Spotlichtquelle, ist T_{max} auf null zu setzen, ansonsten berechnet sich der Wert wie folgt:

$$T_{max} = \frac{\cos^n \theta_{min}}{\pi d_{min}^2}, \theta_{min} < \theta_{max} \tag{60}$$

Ob nun unterteilt werden soll oder nicht wird anhand der gleichen Refinementkriterien, wie bei der Punktlichtquelle entschieden (siehe Kapitel 4.1).

Die *Gather*-Methoden haben genau die gleiche Gestalt wie bei der Punktlichtquelle, wobei hier natürlich die entsprechende Transferfunktionen einzusetzen sind.

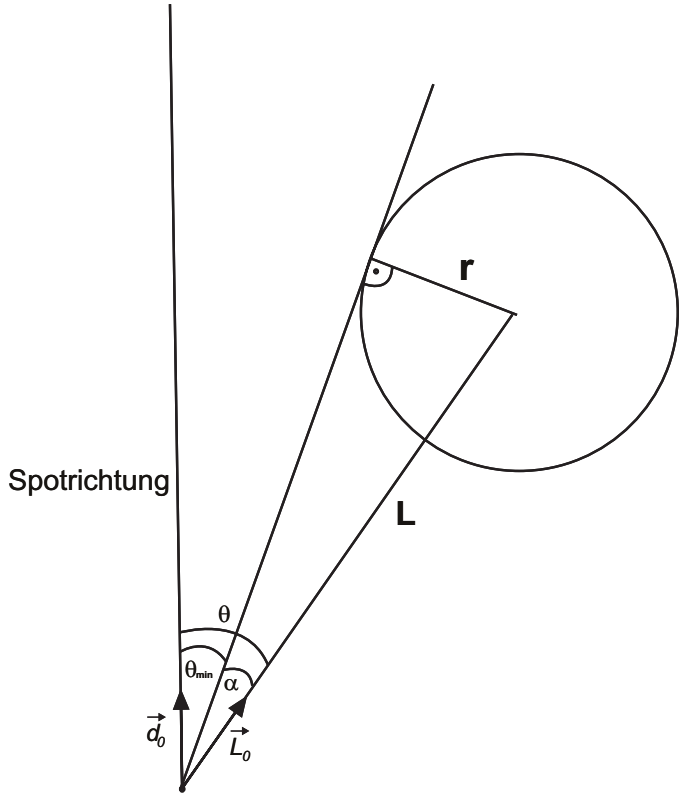


Abbildung 23: Skizze zur Bestimmung von T_{max} bei Spotlight-Cluster Links

5 Implementierung

Ziel dieser Arbeit war die Implementierung und Integration des Radiosity-Kerns, für das im Sommersemester 2003 von Florian Koller, Dominik Rau und von mir entwickelte Radiosity-Komplett-System RADIOCITY. Dieser Kern besteht aus dem hierarchischen Radiosity-Verfahren mit Clustering, wobei alle möglichen Refinement- und Gatherfallunterscheidungen berücksichtigt werden sollten. Desweiteren sollten zu den Lambertemittern auch Energieübertragungen mit Point- und Spotlights möglich sein. Die Energieverteilung sollte sowohl mit dreieckigen wie auch mit viereckigen Flächenelementen möglich sein. Die Berechnung der Visibilität und der Formfaktoren war auch Teil meiner Aufgabenstellung.

Die verwendete Programmiersprache ist C++, die Implementierung von RADIOCITY basiert auf dem Szenengraphen-API OpenSG (www.opensg.org), daraus verwendete Datentypen sind am Präfix OSG:: zu erkennen. Weiterhin werden übliche Container der Standard Template Library (STL), wie z.B. vector verwendet.

5.1 Aufbau der Klassen

5.1.1 Aufbau von *Scene*

Die erzeugte bzw. geladene Szene `global_scene` wird global angelegt, damit in anderen Klassen ohne Parameterübergabe direkt auf Inhalte der Szene zugegriffen werden kann. In der Klasse **Scene**, wird *hardcoded* eine Default-Szene erzeugt, die der Cornell-Box entspricht, wobei die Surfaces der beiden inneren Blöcke zu Clustern gruppiert sind. Die gesamte Szene wird durch den RootCluster beschrieben.

Beim Einladen von Szenen werden genau dann Surfaces Clustern zugeordnet wenn im SceneGraph Group- bzw. Material-Nodes vorliegen, dadurch wird die beim Modellieren entstandene Hierarchie der internen Szene mitübergeben, oft liefert diese Methode schon sehr sinnvolle hierarchische Strukturen die vom RADIOCITY-System verwendet werden kann. Besser wäre allerdings einen eigenständigen Algorithmus bzw. ein System zu entwickeln, um automatische generierte Clusterhierarchien aus einer Menge von Surfaces zu erhalten.

Desweiteren wird in der Klasse ***Scene*** je ein STLVector mit Zeigern auf alle Surfaces und Zeigern auf alle RootPatches abgelegt, sowie ein STLVector mit Zeigern auf alle Lichtquellen. Das Objekt der Klasse ***Scene*** entspricht somit der *root*, mit Zeigern auf alle Lichtquellen der Szene und sie enthält den RootCluster, der wiederum alle weiteren Elemente der Szene hierarchisch strukturiert enthält.

RADIOCITY bietet die Möglichkeit VRML[18] 2.0 (ohne Animation und Interaktion) und OSG Dateiformate zu laden und als interne Szenenstruktur abzulegen.

5.1.2 Aufbau von *Cluster*

Die Klasse ***Cluster*** wird von der Klasse ***Element***, die alle nötigen Membervariablen für den Radiosityausgleich und eine Liste mit Zeigern auf Links enthält, abgeleitet. Der Cluster enthält nach dem Laden der Szene eine Liste mit Zeigern auf die Surfaces, die dem Cluster zugeordnet sind, bzw. Zeiger auf untergeordnete Cluster. Da bislang noch kein Meshing-Algorithmus aufgerufen wurde, sind die Rootpatches noch nicht initialisiert. D.h. nach der Mesh-Initialisierung ist der Aufruf von `initCluster()` nötig, um an dieser Stelle bei jedem Cluster die Elementliste mit Zeigern auf die entsprechenden Rootpatches zu erweitern. Desweiteren enthält ein Cluster eine achsenparallele Bounding-Box, die durch minimalen und maximalen Eckpunkt bestimmt ist, einen Mittelpunkt und eine Gesamtfläche[19] Da in der Klasse ***Element*** die Radiosity Methoden *reset()*[20]*, refine(), gather()* und *pushpull()* als rein virtuell definiert sind, sind diese in der Klasse ***Cluster*** entsprechend implementiert.

5.1.3 Erweiterung der Klasse *Patch*

Da die Klasse Patch auch von der Klasse ***Element*** abgeleitet wird, müssen auch hier die entsprechenden Radiosity-Methoden implementiert werden. Die Klasse ***Patch*** wurde demnach mit den Radiosity-Methoden *reset(), refine(), gather() und pushpull()* entsprechend der Refinementkriterien und Gatherfallunterscheidungen für ein Patch als Empfänger erweitert.

[18] VRML = Virtual Reality Modelling Language

[19] Gesamtfläche = Summe aller Flächen

[20] reset() setzt alle Radiositywerte zurück

5.1.4 Aufbau von *BaseLink*

Die Klasse ***BaseLink*** liefert die Basisklasse für alle Linktypen. Die Klassen ***Link***, ***AlphaLink*** und ***BetaLink*** werden von ***BaseLink*** abgeleitet. Die Basisklasse speichert Zeiger auf Empfänger und Sender, sowie den Visibilitätswert und den Formfaktor. Die Klassen ***AlphaLink*** und ***BetaLink*** enthalten zusätzliche Methoden um Alpha- und Beta- Error zu berechnen, wobei die Klasse ***AlphaLink*** zusätzlich vier *arrays* benötigt um die jeweiligen mittleren und maximalen Formfaktoren für Sender- und Empfängeranteile zu speichern, da diese auch beim *gathern* benötigt werden. Die Klasse ***Link*** repräsentiert den Standardlink zwischen Patches. Ein Zeiger auf den Link wird beim Empfängerelement (Patch oder Cluster) der Linkliste hinzugefügt.

5.1.5 Aufbau von *Tools*

Tools ist eine statische Klasse, liefert demnach Methoden die nicht an ein Objekt gebunden sind. Hier finden sich die Methoden zur Berechnung der Formfaktoren, Visibilität und Support-Plane-Splits wieder.

Formfaktor Bei Patch-zu-Patch Interaktionen werden Punkt-zu-Fläche Formfaktoren nach der Prisma-Methode berechnet, die Fläche des Senders und Mittelpunkt des Empfängers wird dabei in Betracht gezogen. Bei anderen Interaktionsmöglichkeiten werden die Formfaktoren bzw. die Transferfunktionswerte entsprechend der Erläuterungen aus den vorangegangenen Kapiteln berechnet.

Visibilität Die Visibilität wird mittels *gejittertem Raycasting* berechnet, die Anzahl der Schattenfühler sind vom Benutzer einzugeben. Vorab wird allerdings ein Halbraumtest durchgeführt, ob eine Interaktion überhaupt möglich ist.

5.2 Parameter

Das Radiosityprinzip zur globalen Beleuchtungssimulation virtueller Szenen erfordert eine intensive Interaktion von Seiten des Benutzers, da von ihm eine Vielzahl von miteinander verknüpften Parameter, deren Justierung ein Verständnis der genauen Funktionsweise des Verfahrens vorraussetzt [Kre97], eingegeben werden müssen. Eine Automatisierung dieser Parameter, um brauchbare Voreinstellungen zu liefern, wäre von großem Vorteil, erweist sich aber als sehr komplex und ist bisher in RADIOCITY leider nicht integriert. Von daher hängt die Qualität und der Berechnungsaufwand sehr von den eingegebenen Parametern ab. Es erwieß sich als äußerst schwierig gute Parameter zu finden, die einen akzeptablen Kompromiß zwischen Berechnungszeit und Qualität des Ergebnisses liefern. Die einzelnen Parameter und deren Auswirkungen sollen an dieser Stelle kurz erläutert werden:

amount of visibility rays Anzahl der Schattenfühler, umso höher umso grösser der Zeitaufwand, bei zu geringer Anzahl können Schatten komplett verpasst werden, bzw. werden Schattenkanten extrem grob. Initial ist dieser Wert auf 100 eingestellt.

alpha-, beta- link boundary Schwellwert für die Alpha- und Betalinks, ein höherer Wert hat zur Folge, dass mehr Betalinks als ausreichend klassifiziert werden, etwas geringer sind einige Betalinks nicht mehr akzeptabel und Alphalinks werden generiert. Ist der Wert extrem klein gewählt worden werden fast nur noch Standardlinks generiert. Dieser Wert hat demnach auch wieder Auswirkungen auf Geschwindigkeit und Qualität. Initial (für die Default-Szene) ist dieser Wert auf 0.09 gesetzt. Setzt man den Wert bei der Default-Szene auf 10 würden von Lichtquelle zu Cluster Alphalinks generiert. Bei 100 wären die Betalinks ausreichend.

deltaTI Schwellwert für Spotlight bzw. Pointlight zu Patch Interaktionen, wenn die zu erwartende Energieverteilung auf dem Patch als nicht konstant angesehen werden kann wird unterteilt. D.h. kleinere Werte liefern ein feineres Mesh.

tmax to cluster Schwellwert für Spotlight bzw. Pointlight zu Cluster Interaktionen, bei kleineren Werten verringert sich die Anzahl der akzeptierten Lichtquelle-zu-Cluster Links und der Cluster wird entsprechen verfeinert.

BFF Schwellwert für Patch-zu-Patch Interaktionen, wenn die zu erwartende Energieverteilung auf dem Patch als nicht konstant angesehen werden kann wird unterteilt. D.h. kleinere Werte liefern ein feineres Mesh. Für die Default-Szene ist dieser Wert auf 0.15 gesetzt.

BFA Schwellwert für die Energieübertragung zwischen Patches, testet ob genügend Energie übertragen wird, wenn dieser Wert entsprechend höher gesetzt wird, werden weniger Interaktionen in Betracht gezogen. Bei diesem Wert ist darauf zu achten, dass die Fläche und Radiosity des Senders zu berücksichtigen sind. D.h. je nach Größe der Szene ist dieser Wert anzupassen. Initial ist dieser Wert auf 75 gesetzt, da die Default-Szene einer Größenordnung von etwa 500 x 500 x 500 Längeneinheiten entspricht, wäre die Szene auf 1 normiert, müsste der der Wert neu justiert werden.

selfLinkRefineEps Schwellwert zur Entscheidung ob ein Selflink verfeinert wird oder nicht. Testet, ob innerhalb des Clusters genügend Energie vorhanden ist, die zu verteilen ist. Wird der Wert entsprechend groß gewählt wird die Energie innerhalb des Cluster nicht weiter verteilt. Initial steht dieser Wert auf 5.

minArea Schwellwert der die minimale Grösse der Subpatches bestimmt, ist diese Grösse erreicht wird nicht weiter unterteilt, auch wenn das Orakel dies für nötig sieht. Bei der Default-Szene ist dieser Wert auf 20 gesetzt.

5.3 Tastatur-Shortcuts

Einige Tastaturshortcuts die bei RADIOCITY Anwendung finden, um diverse Methoden aufzurufen bzw. Ergebnisse zu visualisieren sind im folgenden erläutert:

„h“ Start einer Radiosity-Iteration
„w“ Darstellung der Szene als Wireframe

„e“	Normale Darstellung
„g“	Ein- und ausschalten des Gouraud-Shading
„b“	Darstellung der Bounding-Volumes und Ausgabe der Anzahl
„l“	Darstellung aller Links zu dem Patch unterhalb der Maus und deren Väter, um so dunkler, um so geringer die Visibilität
„L“	Darstellung aller Links, um so dunkler, um so geringer die Visibilität
„0“	Darstellung aller Alpha- und Betalinks, die Betalinks sind blau und die Alphalinks rot dargestellt
„v“	Darstellung der Visibilitätsstrahlen von den Sendern zu dem Patch unterhalb der Maus, rot nicht sichtbar, weiß die sichtbaren Strahlen
„d“	Ausgabe der Linkanzahl
„D“	Ausgabe der Linkanzahl sortiert nach Linktyp

6 Testergebnisse und Bewertung

Die nun folgenden Ergebnisbilder und die entsprechenden Zeitmessungen sind an einem AMD Athlon XP 2200+ 1,8GHZ Rechner mit 512MB DDRAM durchgeführt worden. Es werden Szenen gegenübergestellt die zu einem mit Cluster und ohne Cluster modelliert sind. Die einfache Defaultszene von RADIOCITY (siehe Abb. 30(a)), die Cornellbox soll nun mit und ohne Cluster betrachtet werden. Mit den vorab eingestellten Parametern: Visibilitätsstrahlen $= 100$, α/β boundary$= 0,09$, $BFF = 0,2$, $BFA = 100$, Selflink Refine $= 5$ und Minimal Area $= 20$ und einem Lambertemitter mit einer Luminanz von $40.000\frac{cd}{m^2}$ sind die in Abb. 31 und Abb 32 dargestellten Bilder berechnet worden. Die nun folgenden Tabellen (siehe Abb. 24 und 25) sollen die Bilder näher dokumentieren.

Cornell-Box mit Cluster (α, β-Boundary = 0.09)					
	zu Anfang	1. Iteration	2. Iteration	3. Iterartion	4. Iteration
Zeit	-	4,906 sec	6,940 sec	0,810 sec	0,313 sec
Patches	15	13.937	18.447	18.507	18.507
Standard Links	-	10.231	21.567	22.315	22.336
α-Links	-	0	8	9	9
β-Links	1	20	1	0	0
Selflinks	1	2	1	1	1

Abbildung 24: Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Cornell-Box mit Cluster

Wie man sieht ist das *Clustern* bei kleinen Szenen nicht von Vorteil, die Anzahl der Links und Patches sind in etwa gleich und die berechneten Bilder (siehe Abb. 31 und Abb. 32) sind für den Betrachter fast identisch. Wird jedoch die Schranke für den α und β-Error nach oben gesetzt, erreicht man selbst bei dieser kleinen Szene eine enorme Zeiteinsparung (siehe Abb. 26 und 27), allerdings müssen Abtriche in der Darstellungsgenauigkeit gemacht werden (siehe Abb. 33).

Cornell-Box ohne Cluster (α, β-Boundary = 0.09)					
	zu Anfang	1. Iteration	2. Iteration	3. Iterartion	4. Iteration
Zeit	-	5,470 sec	6,266 sec	1,875 sec	1,109 sec
Patches	15	13.817	17.633	17.837	17.893
Standard Links	-	10.327	21.514	23.110	23.230
α-Links	-	0	0	0	0
β-Links	1	0	0	0	0
Selflinks	1	0	0	0	0

Abbildung 25: Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Cornell-Box ohne Cluster

Cornell-Box mit Cluster (α, β-Boundary = 100)					
	zu Anfang	1. Iteration	2. Iteration	3. Iterartion	4. Iteration
Zeit	-	3,672 sec	4,797 sec	0,360 sec	0,187 sec
Patches	15	11.343	14.811	14.855	14.855
Standard Links	-	8.276	17.932	18.523	18.694
α-Links	-	0	0	0	1
β-Links	1	22	16	16	15
Selflinks	1	2	2	2	2

Abbildung 26: Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Cornell-Box ohne Cluster bei einem Alpha/Beta Error-Schranke von 100

Cornell-Box mit Cluster (α, β-Boundary = 5)					
	zu Anfang	1. Iteration	2. Iteration	3. Iterartion	4. Iteration
Zeit	-	3,938 sec	4,953 sec	0,453 sec	0,219 sec
Patches	15	11.455	14.923	14.999	14.999
Standard Links	-	8.390	18.175	18.934	19.180
α-Links	-	2	10	10	10
β-Links	1	20	6	6	6
Selflinks	1	2	2	2	2

Abbildung 27: Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Cornell-Box ohne Cluster bei einem Alpha/Beta Error-Schranke von 5

Bei Betrachtung von Abb. 33 erkennt man, dass bei einer recht großen Schranke die inneren Cluster mit der Lichtquelle durch β-Links verknüpft sind, dadurch erhält die Energieübertragung keinerlei Informationen über die Lage der Patches des Clusters und diese werden dadurch alle gleich stark beleuchtet. Beim α-Link wird die Ausrichtung allerdings mitberücksichtigt (siehe Abb. **??**), aber die Rootpatches des Clusters werden nicht weiter unterteilt, bei grösseren, komplexeren Szenen ist daher der Einsatz von Alpha- und Beta-Links sinnvoller, denn diese Beispielbilder liefern bei zu hoher Schranke falsche Ergebnisse. Alpha und Beta Links sollten dann zum Einsatz kommen, wenn Objekte sehr weit voneinander entfernt bzw. wenig Energie übertragen wird.

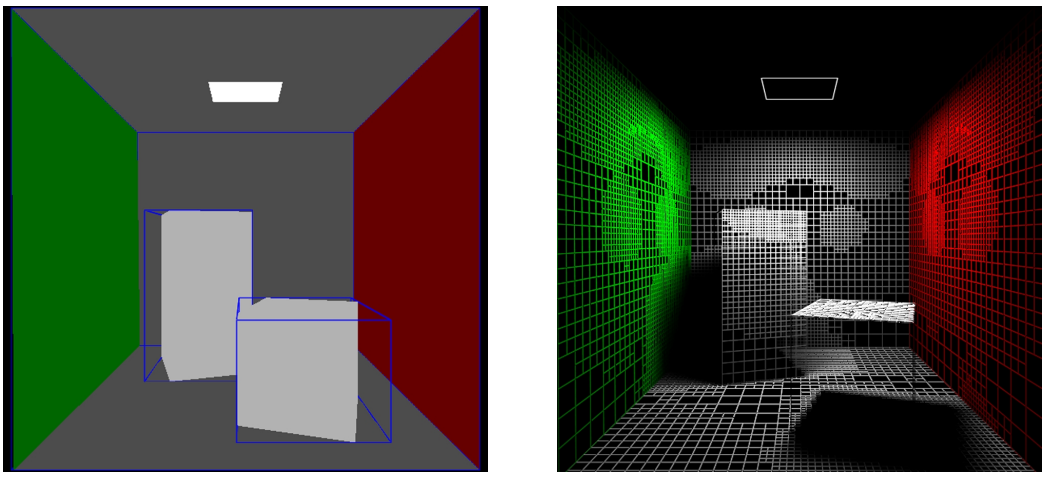
Nun sollen etwas komplexere Szenen weiter untersucht werden die u.a. auch eine größere Hierarchietiefe haben. Die Raumszenem bestehend aus Stühlen, Regal und drei Lambertemittern besteht aus 20 Clustern, wobei zwei der Stühle sogar noch, weiter strukturiert sind und je aus Cluster für Beine, Sitzfläche und Lehne bestehen (siehe Abb. 6). Die Berechnungsdaten sind in Abb. 28 und Ergebnissbilder in Abb 35 zu sehen. Man erkennt, dass man eine enorme Zeitersparnis der Szene mit Cluster gegenüber der Szene ohne Cluster erreicht. Die 1. Iteration benötigt bei der Raumszene ohne Cluster über 3 Minuten (siehe Abb. 29), dagegen ist die Szene mit Cluster schon nach 25 Sekunden berechnet, wie man weiterhin erkennt ist die Anzahl der Links bei der der Raumszene ohne Clustering um einiges höher als mit Clustering. Man sieht keine Unterschiede mehr bei den Ergebnissbildern (siehe38), obwohl mit Clustering wesentlich weniger Interaktionen stattfanden, daraus lässt sich selbst bei dieser kleinen Szene schon erkennen, dass das Clustern von großem Vorteil ist. Ab Abb. 40 folgen weitere Beispielbilder, die aber nicht weiter dokumentiert werden sollen.

Raumszene mit Stühlen und Regal mit Cluster				
	zu Anfang	1. Iteration	2. Iteration	3. Iterartion
Zeit	-	25,140s	3m 13,344s	20m 47,906s
Patches	230	19.824	35.264	35.848
Standard Links	-	18.786	62.792	68.021
α-Links	-	8	40	51
β-Links	3	131	111	100
Selflinks	1	7	13	13

Abbildung 28: Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Raumszene mit Cluster

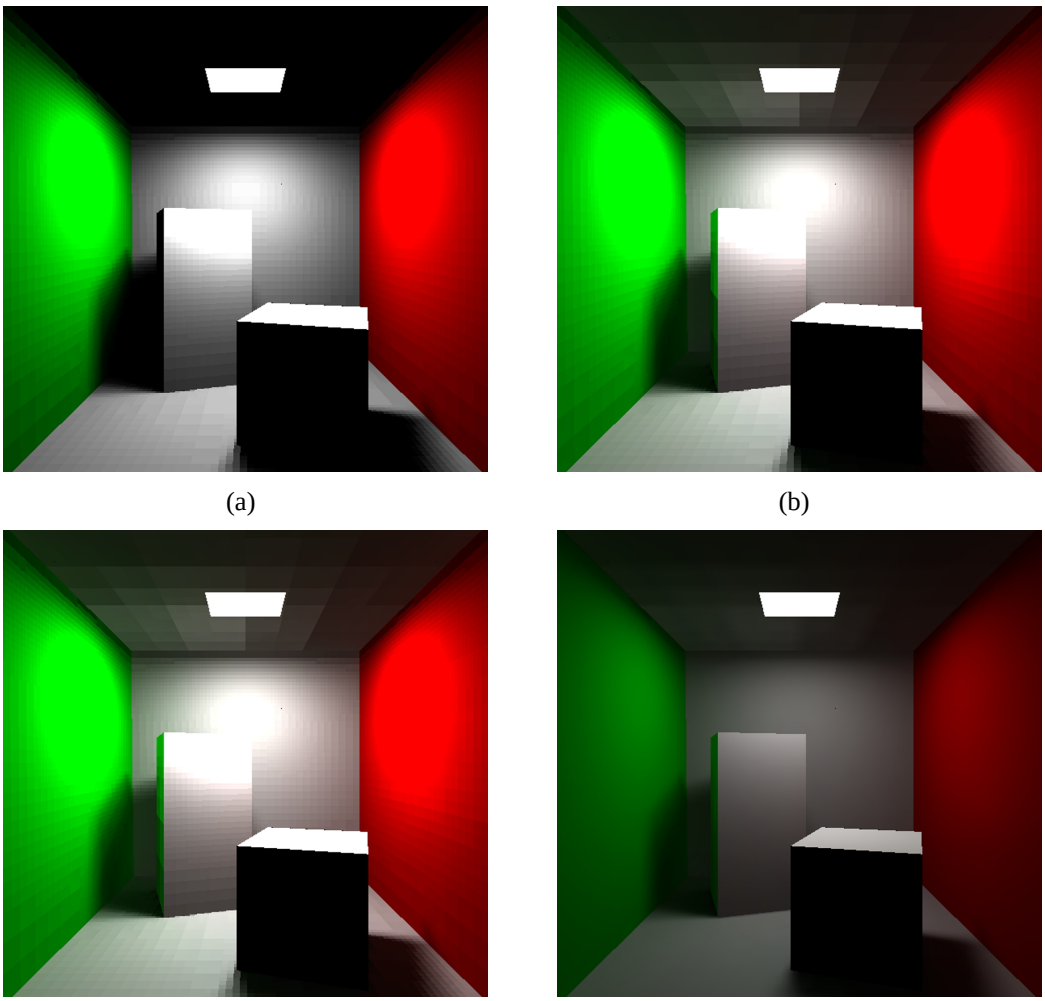
Raumszene mit Stühlen und Regal ohne Cluster)				
	zu Anfang	1. Iteration	2. Iteration	3. Iterartion
Zeit	-	3m 2,31s	6m 14,718s	33m 50,594s
Patches	230	20.286	37.956	38.436
Standard Links	-	72.521	111.020	116.046
α-Links	-	-	-	-
β-Links	3	-	-	-
Selflinks	1	-	-	-

Abbildung 29: Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Raumszene ohne Cluster



(a) (b)

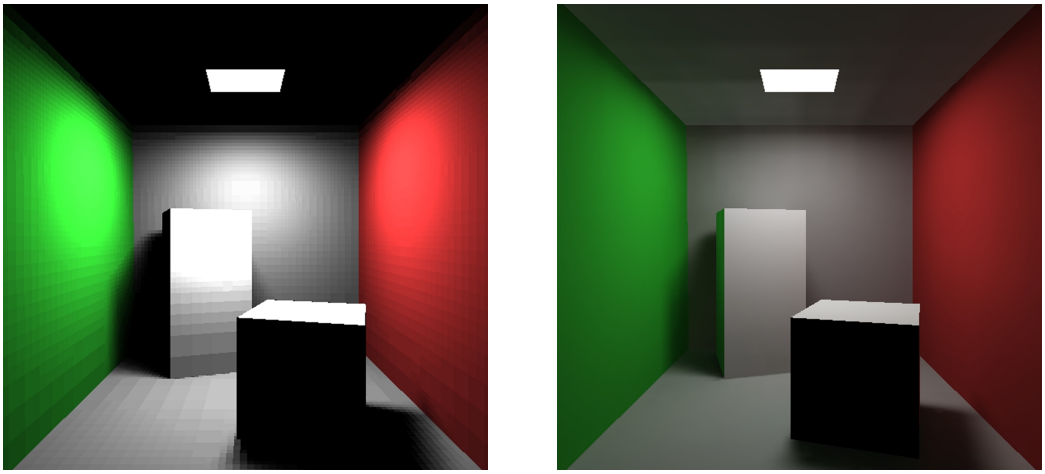
Abbildung 30: (a) Cornell-Box, bestehend aus drei Clustern (RootCluster und die beiden inneren Cluster), die Bounding Boxen der Cluster sind blau markiert. (b) Mesh der Cornell-Box mit Cluster nach 1. Iteration.



(a) (b)

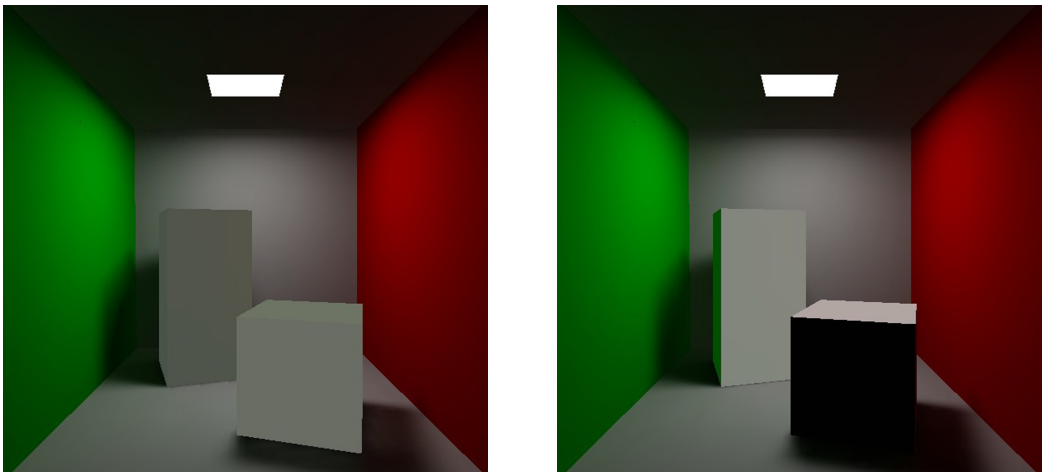
(c) (d)

Abbildung 31: Cornell Box mit Cluster (a) nach 1.Iteration (direkte Licht). (b) nach 2. Iteration mit indirektem Licht (Color Bleeding). (c) nach 3.Iteration (d) nach 4.Iteration mit Gouraud-Shading und Tonemapper



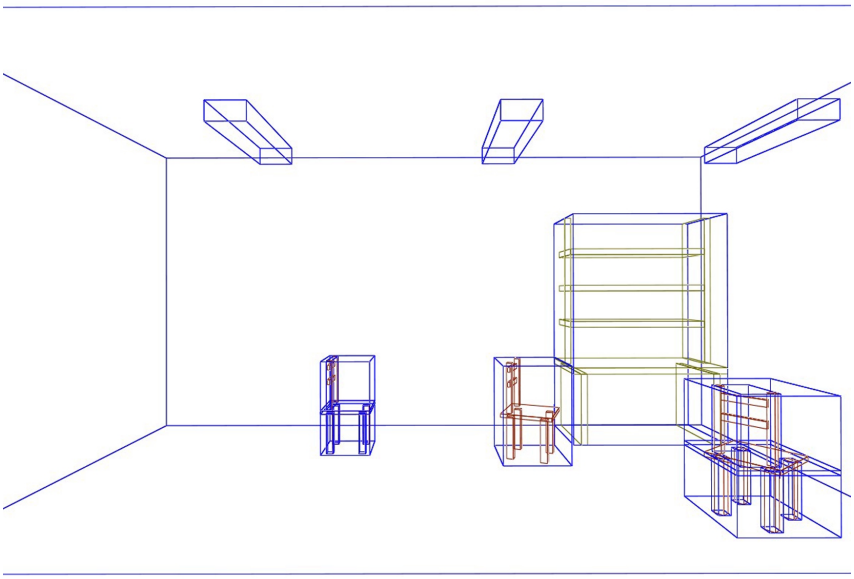
(a) (b)

Abbildung 32: Cornell-Box, ohne inneren Cluster (a) nach 1.Iteration (b) nach 4.Iteration mit Gouraud-Shading und Tonemapper

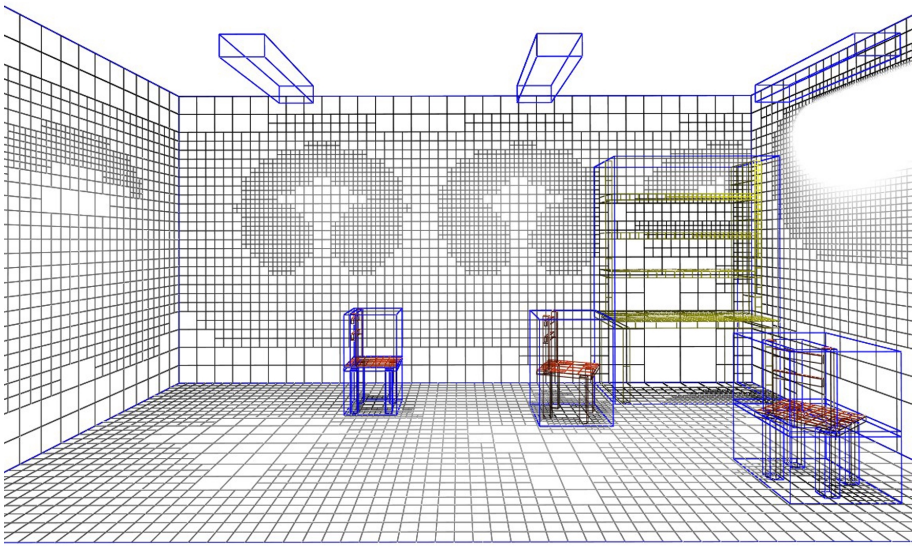


(a) (b)

Abbildung 33: Cornell-Box, mit Cluster (a) nach 4.Iteration mit Gouraud-Shading und Tonemapper und einem α/β-Boundary Wert von 100 (b) nach 4.Iteration mit Gouraud-Shading und Tonemapper und einem α/β-Boundary Wert von 5



(a)



(b)

Abbildung 34: Raumszene mit Cluster(a) Darstellung der Cluster (b) Mesh nach 1.Iteration

(a)

(b)

(c)

Abbildung 35: Raumszene mit Cluster(a) nach 1.Iteration (b) nach 2.Iteration (c) nach 3.Iteration (mit Tonemapper und Gouraud-Shading)

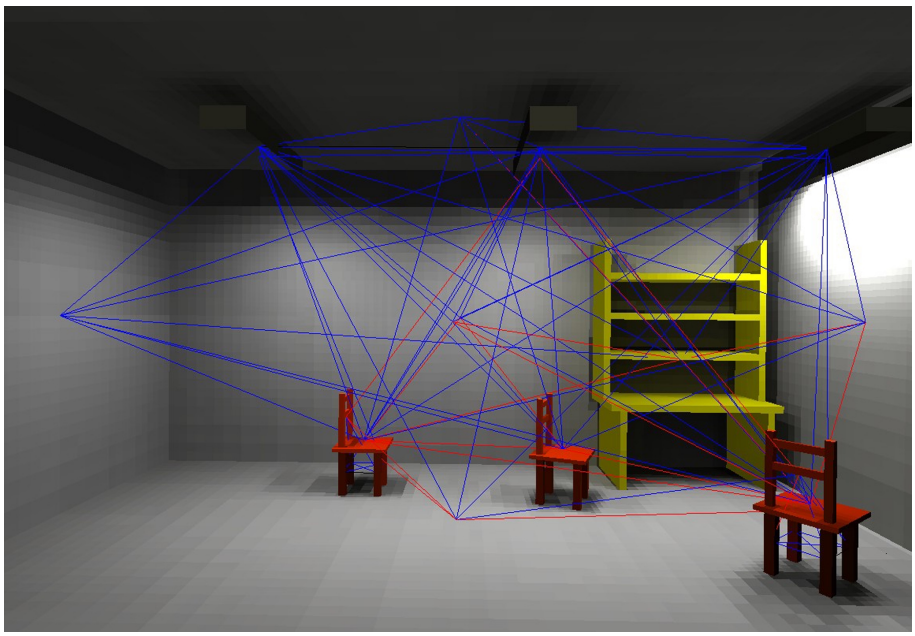


Abbildung 36: Raumszene mit Cluster nach 2. Iteration, Darstellung der Alpha-(rot) und Beta-Links(blau)

Abbildung 37: Detailansicht der Raumszene nach der 4.Iteration (mit Gouraud-Shading und Tonemapper)

Abbildung 38: Raumszene ohne Cluster nach der 4.Iteration (mit Gouraud-Shading und Tonemapper)

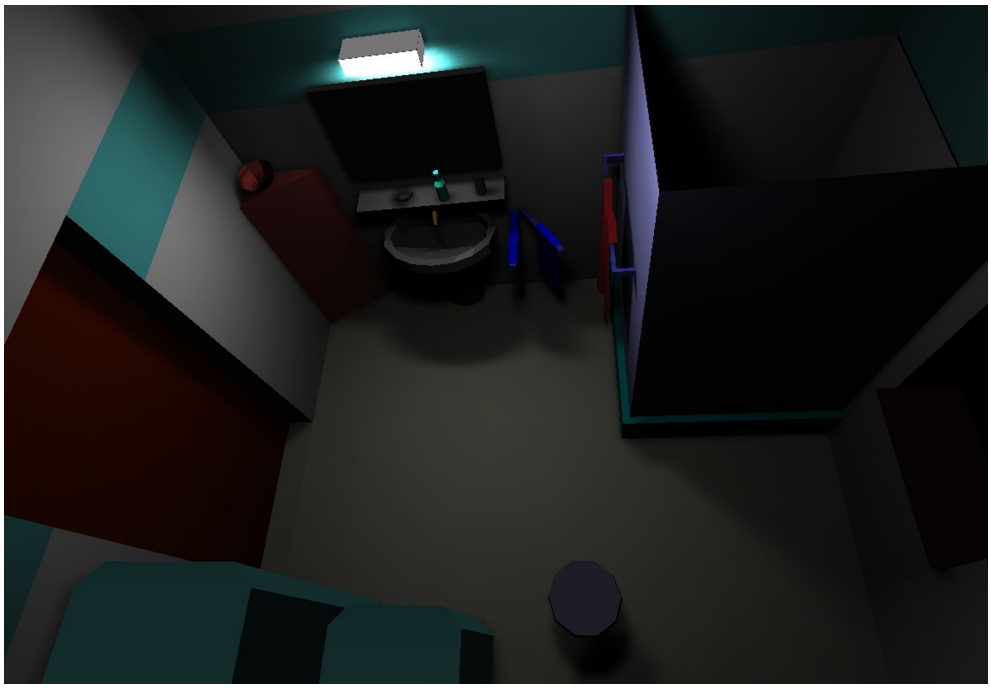


Abbildung 39: Badezimmerszene nach 2 Iterationen, 24 Cluster, 45.268 Patches, 44.127 Links

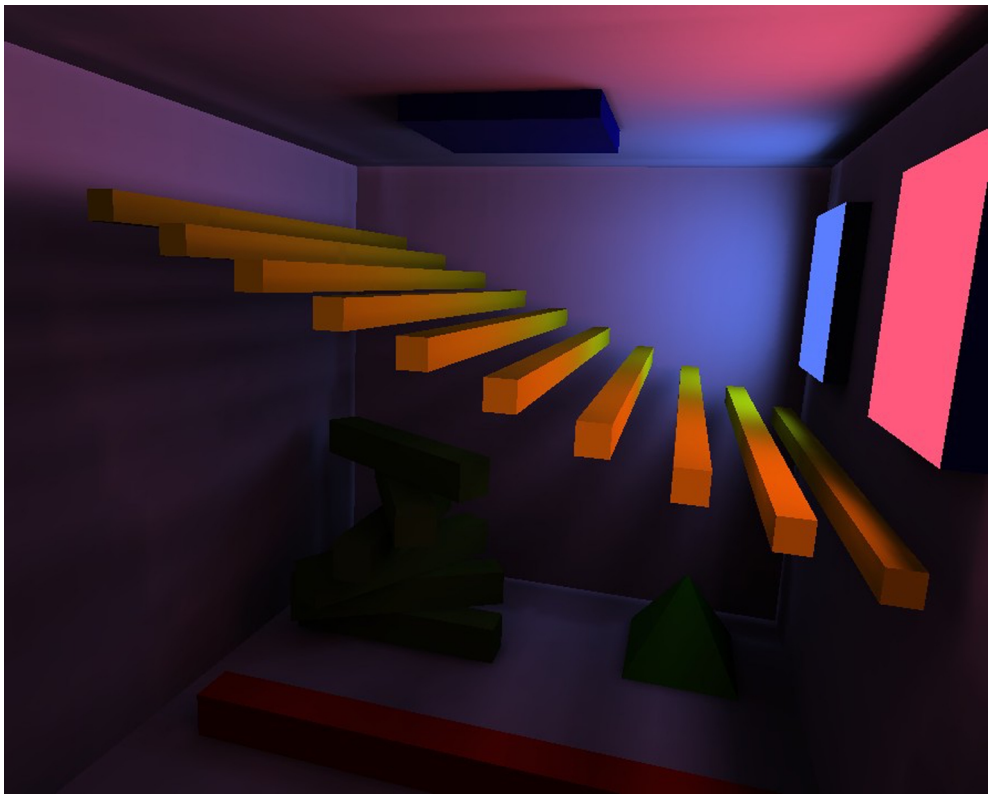
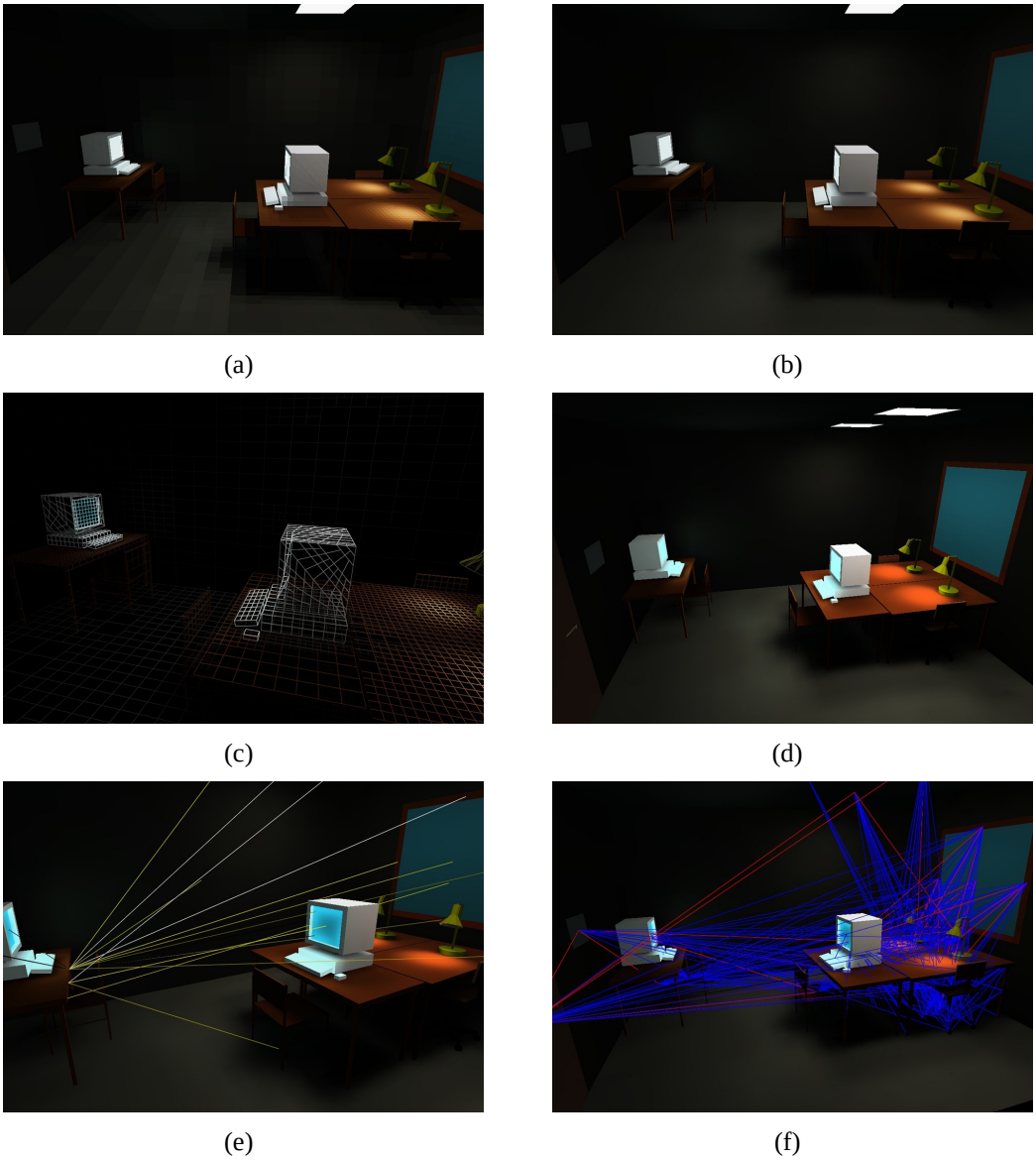
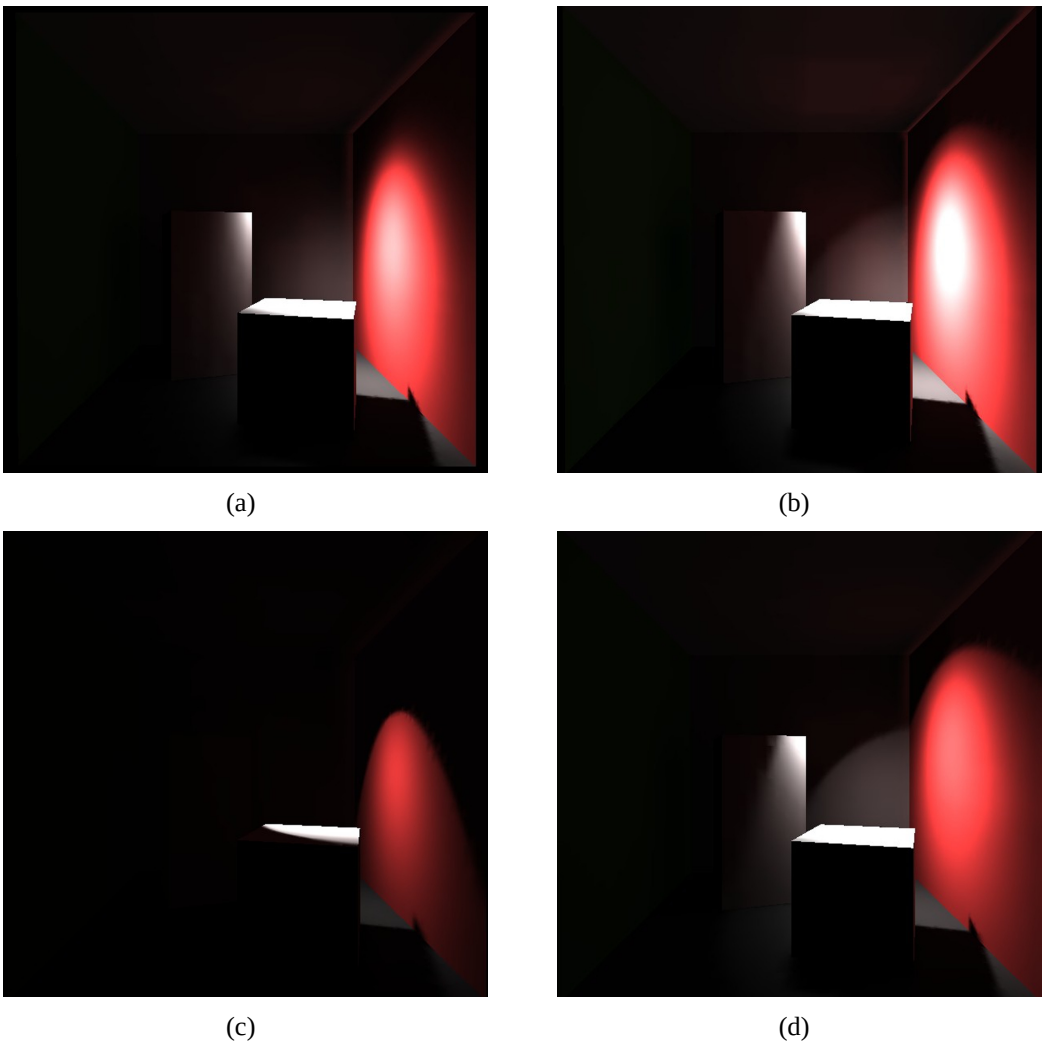


Abbildung 40: Spaceszene nach 2 Iterationen, 23 Cluster, 62.503 Patches, 109.103 Links



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

Abbildung 41: Büroszene mit 4 Lambertemittern und 2 Spotlights nach 2. Iteration (a) mit Flat-Shading (b) mit Gouraud-Shading und URQ-Tonemapper (c) Detailansicht Wireframe (d) mit Gouraud-Shading und NURQV-Tonemapper (e) Visualisierung einiger Links (f) Alpha und Beta Links



(a) (b)

(c) (d)

Abbildung 42: Cornell Box mit Spotlight, unter diversen Einstellungen (a) θ_max = 90°, n = 10 (b) θ_max = 50°, n = 10 (c) θ_max = 30°, n = 10 (d) θ_max = 50°, n = 6

7 Ausblick

Das Gebiet der Lichtsimulation ist sehr komplex und umfassend. Alle Ideen und Aspekte konnten sicherlich nicht im Rahmen dieser Studienarbeit behandelt werden. Es soll daher nur ein Ausblick auf weitere Möglichkeiten gegeben werden, die RADIOCITY verbessern, bzw. erweitern könnten.

RADIOCITY funktioniert fehlerfrei und liefert in angemessener Zeit sehr gute Ergebnisbilder, es bestehen allerdings noch genügend Möglichkeiten das System zu optimieren, gerade bei der Visibilitätsberechnung ist noch einiges an Zeit einzusparen. Dies würde die Integration von *Shaft Culling* und eine Automatisierung der Anzahl der Schattenfühler ermöglichen.

Das System lässt sich recht einfach mit weiteren virtuellen Lichtquellen erweitern, bisher sind Lambertemitter, Point- und Spotlights integriert so wäre es zum Bsp. möglich das System mit LVKs[21] zu erweitern.

Weitere Elementtypen sind auch denkbar, sehr effizient sind u.a. *Face Cluster*[22]

Um gezackte Schattenkanten zu vermeiden, wäre es möglich das System mit *discontinuity meshes* zu erweitern, d.h. man schneidet das Patch genau entlang der Schattenkanten, näheres dazu ist der Studienarbeit „Meshing Algorithmen in RADIOCITY “ von Dominik Rau zu finden.

Eine Parameterautomatisierung wäre auch von großem Vorteil, da es sich für den Benutzer nicht einfach erweist gute Parametereinstellungen zu finden, um bestmögliche Ergebnisbilder zu erzielen.

Cluster werden bisher anhand der Scenengraph-Struktur generiert, d.h. aus Gruppen und Material-Nodes werden beim Einlesen Cluster generiert, dieses liefert nicht immer sinnvolle Cluster. Es wäre von Vorteil RADIOCITY mit einem Verfahren zu erweitern, dass aus einer Surface-Menge Cluster generiert, anhand deren geometrischen Lage und Materialbeschaffenheit.

[21] Lichtquellen mit Lichtstärkeverteilungskurven z.B. nach Herstellerangaben

[22] Face Cluster räpresentieren eine Gruppierung von Rootpatches, die ähnlich wie bei Cluster, geometrisch beisammen liegen, allerdings ähnliche Normalen haben, ein Face Cluster hat somit im Gegensatz zum Cluster eine interpolierte Normale

Abbildungsverzeichnis

1 Hierarchische Baumstruktur eines Rootpatches 7
2 Hierarchische Patchstruktur des Rootpatches aus Abb.1 7
3 Eventuelle Linkstruktur einer Miniszene 8
4 Das linke Bild zeigt Patches in einer einfache Szene, die miteinander verlinkt sind. Die gestrichelten Links sind laut Refinementkriterium nicht ausreichend und werden im nächsten Schritt verfeinert (siehe rechts) 9
5 Skizze zur Berechnung des Prisma-Formfaktors 10
6 Bestimmung von ff(x) 12
7 Gejittertes Raycasting mit 25 Strahlen 14
8 Shaft Culling, Shaft über die Bounding Boxes von A und B 16
9 Support Plane Split beim Sender (*source split*) 17
10 Vergleich Standard Unterteilung(links) und asymetrische Unterteilung(rechts) bei einem SPS 17
11 Adaptives Mesh mit genügend feiner Unterteilung im Halbschatten 19
12 Receiver wird in vier Subpatches unterteilt, Sender zu Empfänger Link wird gelöscht, und vier neue Links zu den Subpatches generiert. 21
13 Sender wird in vier Subpatches unterteilt, Sender zu Empfänger Link wird gelöscht, und vier neue Links von den Subpatches zum Receiver generiert. . 21
14 Push: Radiosity des Patches wird den Kindern aufaddiert 24
15 Pull: Radiosity des Patches wird auf die flächengewichtete Radiosity der Kinder gesetzt (unter der Annahme, dass alle Kinder gleich groß sind) . . . 24
16 Ausschnitt einer hierarchischen Szenenanordnung mit Cluster 26
17 Standard HR-Links, quadratischer Aufwand 27
18 α-Link, linearer Aufwand 27
19 β-Link, konstanter Aufwand 28
20 Initialer Zustand einer hierarchische Szene mit Cluster 29
21 Links: Cluster besitzt einen Selflink, der rechts im Bild entsprechend verfeinert worden ist. 32
22 Zweidimensionale Interpretation der Szene um T_{max} zu bestimmen 38
23 Skizze zur Bestimmung von T_{max} bei Spotlight-Cluster Links 41
24 Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Cornell-Box mit Cluster 45
25 Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Cornell-Box ohne Cluster 46
26 Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Cornell-Box ohne Cluster bei einem Alpha/Beta Error-Schranke von 100 46
27 Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Cornell-Box ohne Cluster bei einem Alpha/Beta Error-Schranke von 5 46
28 Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Raumszene mit Cluster 47
29 Berechnungszeiten und Anzahl der Patches und Links der einzelnen Iterationen der Raumszene ohne Cluster 48

30 (a) Cornell-Box, bestehend aus drei Clustern (RootCluster und die beiden inneren Cluster), die Bounding Boxen der Cluster sind blau markiert. (b) Mesh der Cornell-Box mit Cluster nach 1. Iteration. 48
31 Cornell Box mit Cluster (a) nach 1.Iteration (direkte Licht). (b) nach 2. Iteration mit indirektem Licht (Color Bleeding). (c) nach 3.Iteration (d) nach 4.Iteration mit Gouraud-Shading und Tonemapper 49
32 Cornell-Box, ohne inneren Cluster (a) nach 1.Iteration (b) nach 4.Iteration mit Gouraud-Shading und Tonemapper 50
33 Cornell-Box, mit Cluster (a) nach 4.Iteration mit Gouraud-Shading und Tonemapper und einem α/β-Boundary Wert von 100 (b) nach 4.Iteration mit Gouraud-Shading und Tonemapper und einem α/β-Boundary Wert von 5 . 50
34 Raumszene mit Cluster(a) Darstellung der Cluster (b) Mesh nach 1.Iteration 51
35 Raumszene mit Cluster(a) nach 1.Iteration (b) nach 2.Iteration (c) nach 3.Iteration (mit Tonemapper und Gouraud-Shading) 52
36 Raumszene mit Cluster nach 2. Iteration, Darstellung der Alpha-(rot) und Beta-Links(blau) 53
37 Detailansicht der Raumszene nach der 4.Iteration (mit Gouraud-Shading und Tonemapper) 53
38 Raumszene ohne Cluster nach der 4.Iteration (mit Gouraud-Shading und Tonemapper) 54
39 Badezimmerszene nach 2 Iterationen, 24 Cluster, 45.268 Patches, 44.127 Links 54
40 Spaceszene nach 2 Iterationen, 23 Cluster, 62.503 Patches, 109.103 Links . 55
41 Büroszene mit 4 Lambertemittern und 2 Spotlights nach 2. Iteration (a) mit Flat-Shading (b) mit Gouraud-Shading und URQ-Tonemapper (c) Detailansicht Wireframe (d) mit Gouraud-Shading und NURQV-Tonemapper (e) Visualisierung einiger Links (f) Alpha und Beta Links 56
42 Cornell Box mit Spotlight, unter diversen Einstellungen (a) θ_max = 90°, n = 10 (b) θ_max = 50°, n = 10 (c) θ_max = 30°, n = 10 (d) θ_max = 50°, n = 6 . 57

Literatur

[AF02] Ilka Agricola and Thomas Friedrich. *Global Analysis - Differential Forms in Analysis, Geometry and Physics*. AMS, Graduate Studies in Mathematics 52, Rhode Island, 2002.

[App68] A. Appel. *Some Techniques for Shading Machine Renderings of Solids*. SJCC, 1968.

[CCWG88] M. F. Cohen, S. E. Chen, J. R . Wallace, and D. P. Greenberg. *A Progressive Refinement Approach to Fast Radiosity Image Generation*. Computer Graphics (ACM SIGGRAPH '88 Proceedings) Vol. 22, Nr. 4, August 1988.

[Gla95] A. S. Glassner. *Principles of digital Image Synthesis Vol.1 und Vol.2*. Morgan Kaufmann Publishers, San Francisco, 1995.

[Gou71] H. Gouraud. *Continous Shading of Curved Surfaces*. IEEE Transactions on Computers Vol. 20, Nr.6, Juni 1971.

[GTGB84] Cindy M. Goral, Kenneth E. Torrance, Donal P. Greenberg, and Bennett Battaille. *Modelling the Interaction of Light between Diffuse Surfaces*. Computer Graphics (ACM SIGGRAPH '84 Proceedings) Vol. 18, Juli 1984.

[Hai00] Eric. A. Haines. *Shaft Culling Tool*. Journal of Graphics Tools V 5, No. 1, 2000.

[HS98] Nicolas Holzschuch and Francois Sillion. *An Exhaustive Error-Bounding Algorithm for Hierarchical Radiosity*. Euro Graphics Association, 1998.

[HSA91] Pat Hanrahan, David Salzman, and Larry Aupperle. *A Rapid Hierachical Radiosity Algorithm*. Computer Graphics (ACM SIGGRAPH '91 Proceedings) Vol. 25, Nr. 4, Juli 1991.

[HW91] Eric. A. Haines and J. Wallace. *Shaft Culling for Efficient Ray-Traced Radiosity*. In Proceedings of the Second Eurographics Workshop on Rendering, 1991.

[Kre97] Wolfram Kresse. *Effizientes und anwendbares hierarchisches Radiosity unter Berücksichtigung der Visibilitätsbetrachtung*. Diplomarbeit, Technische Universität Darmstadt, 1997.

[LSG94] Daniel Lischinski, Brian Smits, and Donald P. Greenberg. *Bounds And Error Estimates for Radiosity*. ACM SIGGRAPH '94 Proceedings, 1994.

[Pho75] B. T. Phong. *Illumination for Computer Generated Pictures*. Communications of ACM Vol.18, Nr.6, Juni 1975.

[SAG94] Brian Smits, James Arvo, and Donald Greenberg. *A Clustering Algorithm for Radiosity in Complex Environments*. Computer Graphics Vol.28 Annual Conference Series, 1994.