

---

# **Wavelet Radiosity und ihre Einbettung in das 3D-Graphiksystem MRT**

---

Diplomarbeit

vorgelegt von  
Stefan Struck

Rheinische Friedrich-Wilhelms Universität Bonn  
Institut für Informatik III

8. Oktober 1998

## **Zusammenfassung**

Diese Arbeit befaßt sich mit Wavelet Radiosity, einem Radiosity-Ansatz, der den Vorzug bietet, sowohl hierarchische Radiosity, wie auch Galerkin Radiosity gemeinsam in einem mathematischen Grundgerüst zu vereinen. Dies geschieht durch eine eingehende Betrachtung des Radiosity-Verfahrens aus Sicht der Funktionenprojektionen und Anwendung der gewonnenen Erkenntnisse auf Familien von hierarchischen Basisfunktionen, den Wavelets.

Besonderer Schwerpunkt dieser Arbeit liegt auf den Aspekten einer Implementierung. Dabei wird die Einbettung in das 3D-Graphiksystem *Minimal Rendering Toolkit* (MRT) im Detail beschrieben.

# Versicherung

Hiermit versichere ich an Eides Statt, daß ich die vorliegende Arbeit „Wavelet Radiosity und ihre Einbettung in das 3D-Graphiksystem MRT“ selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Leverkusen, den 8. Oktober 1998

Stefan Struck

# Danksagung

An dieser Stelle möchte ich mich herzlich bei den folgenden Personen bedanken:

Herrn Prof. Fellner für seine Unterstützung bei dieser Arbeit und seinen unermüdlichen Einsatz für uns Studenten.

Stephan Schäfer für die tapfere Beantwortung der vielen, vielen Fragen und dafür, daß er nie eine Miene verzogen hat, egal, wie „dumm“ die Frage auch war. Jens Leitert für die tatkräftige Unterstützung bei WxWin. Susanne Krause für psychologische Unterstützung und dafür, daß sie mich auch mit schlechter Laune ertragen kann. Stephan Proksch für die Korrekturlesung und konstruktive Kritik. Dominik Gassen für die Möglichkeit, seine Drucker zu benutzen sowie die Gastfreundschaft für die Zeit des Ausdrucks. Allen Freunden und Bekannten, die den Mut hatten nach dem Thema der Arbeit zu fragen und sich den anschließenden Vortrag anhören durften.

Ein besonders großes Dankeschön geht an Andrew Willmott, für die vielen Fragen, die er via email beantwortet hat. Thanks, Andrew, you have a special place in my personal hall of fame.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>6</b>
<b>2</b>	<b>Radiosity-Grundlagen</b>	<b>10</b>
2.1	Klassische Radiosity ( <i>Classical Radiosity, CR</i> )	10
2.2	Hierarchische Radiosity ( <i>Hierarchical Radiosity, HR</i> )	11
2.2.1	Das N-Körper Problem	12
2.2.2	Beschreibung des Algorithmus	12
2.3	Die Radiosity-Integralgleichung	13
<b>3</b>	<b>Projektionen</b>	<b>15</b>
3.1	Projektion in einen endlichdimensionalen Funktionenraum	15
3.2	Rendering-Gleichung im neuen Gewand	17
<b>4</b>	<b>Wavelets</b>	<b>19</b>
4.1	Aufbau einer Wavelet-Basis	19
4.2	Eigenschaften einer Wavelet-Basis	22
4.2.1	Vanishing Moments	22
4.2.2	Dünnbesetzte Repräsentierung	24
4.2.3	Beziehung zwischen verschiedenen Stufen der Hierarchie( <i>two-scale Relationship</i> )	25
4.3	Wavelets in höheren Dimensionen	26
<b>5</b>	<b>Wavelet Radiosity</b>	<b>29</b>
5.1	Grundsätzliches Vorgehen	29
5.1.1	Die Reihenfolge der einzelnen Schritte	31
5.2	Laufzeitbetrachtung	31
5.3	Die Kernel Projektion	31
5.3.1	Bottom-Up Berechnung	31
5.3.2	Top-Down Berechnung	32
5.4	Die Implementierung des Algorithmus	32
5.4.1	Implementierte Wavelet-Basen	32
5.4.1.1	Haar-Basis	32
5.4.1.2	Flatlets	33
5.4.1.3	Multiwavelets	34
5.4.1.4	Vor- und Nachteile von Tree Wavelets	35
5.4.2	Das Orakel	35
5.4.3	Quadratur	36
5.4.4	Gathering: Das Sammeln von Radiosity	38
5.4.4.1	Die Berechnung des Interaktionsfaktors für Mult2-Wavelets	38
5.4.5	Abbruchkriterium	40
5.5	Dreiecke und Wavelet-Radiosity	40

5.5.1	Änderungen gegenüber Vierecken	41
5.5.1.1	Haar- und Flatlet-Basen	41
5.5.1.2	Multiwavelets	41
5.5.1.3	Allgemeine Änderungen für alle Basen	42
<b>6</b>	<b>Einbettung in den MRT</b>	<b>43</b>
6.1	Einführung in den MRT	43
6.2	Besonderheiten des MRT	43
6.2.1	Objekte der Szene	44
6.3	Benötigte Datenstrukturen	45
6.3.1	Speicherverbrauch	45
6.4	Ausgabe des Ergebnisses	45
6.4.1	Anzeige über BReps	45
6.4.1.1	Darstellung bei Verwendung der Haar-Basis	46
6.4.1.2	Darstellung bei Verwendung der Multiwavelet2-Basis	46
6.4.1.3	Darstellung bei Verwendung anderer Basen	47
6.4.2	Erzeugung einer Textur zur Darstellung des Ergebnisses	47
6.4.2.1	Erzeugen und Aufbringen der Textur	48
6.4.2.2	Grenzen des texturbasierten Ansatzes	49
6.4.2.3	Flatlets und Gouraud-Schattierung	49
6.4.3	Direkte Anzeige unter Umgehung des BRep-Standards	50
6.5	Gemeinsamkeiten und Unterschiede der implementierten Versionen	50
6.5.1	Gemeinsamkeiten	50
6.5.2	Unterschiede	51
<b>7</b>	<b>Anbindung an den MRT</b>	<b>52</b>
7.1	Ansatzpunkte	52
7.1.1	Allgemeiner Aufbau	52
7.1.2	Geänderte Klassen	52
7.1.2.1	Der Link <code>t_HLink</code>	53
7.1.2.2	Die Szene <code>t_IllumSceneHR</code>	53
7.1.2.3	Das Patch <code>t_HradPatch</code>	53
7.1.3	Neue Klassen	54
7.1.3.1	Optionenklasse <code>t_WaveletOptions</code>	55
7.1.3.2	Statistikklasse <code>t_Statistics</code>	57
7.1.3.3	Der Interaktionsfaktor	58
7.2	Implementierung verschiedener Wavelet-Basen	58
7.2.1	Haar-Basis	58
7.2.2	Flatlets	59
7.2.3	Multiwavelets	59
7.2.3.1	Negative Interaktionsfaktorelemente	59
7.2.3.2	Negative Farbwerte	60
<b>8</b>	<b>Ergebnisse</b>	<b>61</b>
<b>9</b>	<b>Diskussion und Ausblick</b>	<b>66</b>
9.1	Wavelet Radiosity: Theoretisches Konstrukt oder praktische Anwendung?	66
9.2	Welche Basen sind für einen praktischen Einsatz geeignet?	67
9.3	Gibt es effizientere Speicherungsmöglichkeiten?	67
9.4	Ausblick	68

9.4.1	Adaptive Auswahl einer Wavelet-Basis . . . . .	68
9.4.2	Realisierung importance-basierter Radiosity . . . . .	68
9.4.3	Umsetzung anderer Wavelet-Basen . . . . .	68
<b>10</b>	<b>Anhang A Beispiel einer Basis Implementierung: Mult2</b>	<b>69</b>
10.1	Die Szene . . . . .	69
10.1.1	Eigene Methoden . . . . .	69
10.1.2	Überschriebene Methoden der Basisklasse . . . . .	70
10.2	Das Patch . . . . .	70
10.2.1	Eigene Methoden . . . . .	70
10.2.2	Überschriebene Methoden der Basisklasse . . . . .	71
10.2.3	Datenfelder der Klasse . . . . .	73

# Kapitel 1

## Einleitung

Um photorealistische Bilder von vollständig diffusen Szenen zu erzeugen, haben sich Radiosity-Methoden als effektive Vorgehensweise erwiesen, indem sie das globale Beleuchtungsproblem lösen. Die bekannteste und älteste dieser Methoden ist die klassische Radiosity (*Classical Radiosity* CR), die mit Hilfe eines Energieerhaltungsarguments ein System von linearen Gleichungen aufstellt, um den Austausch von Energie innerhalb der Szene zu modellieren. Eine Lösung dieses Gleichungssystems liefert schließlich die Information, wieviel Energie jedes Oberflächenelement der Szene abgibt. Zeigt man nun die vom Betrachtungspunkt aus sichtbaren Elemente an, erhält man das gewünschte Bild der Szene. Die eigentliche Radiosity-Berechnung ist durch die Beschränkung auf perfekte diffuse Reflexion blickwinkelunabhängig; nur der letzte Schritt muß die Kamera miteinbeziehen. Das hat zur Konsequenz, daß nur die genannte Sichtbarkeitsbetrachtung wiederholt werden muß, wenn sich der Betrachtungspunkt oder der Blickwinkel verändert. Voraussetzung für dieses Lösungsverfahren sind dabei die folgenden Annahmen:

- Alle Oberflächen sind opak und perfekte diffuse Reflektoren.
- Die Energie, die von den Oberflächenelementen, den sogenannten Patches, ausgestrahlt wird, ist konstant für jedes dieser Elemente.
- Die eintreffende Energie ist ebenfalls für jedes Patch konstant.

Betrachtet man diese Einschränkungen und Annahmen, so bieten sich zwei Möglichkeiten an, das Verfahren zu erweitern:

Erstens besteht die Möglichkeit, die Einschränkungen des zweiten und dritten Punktes wegfallen zu lassen, d.h. weder die eintreffende, noch die ausgestrahlte Energie muß konstant sein. Aus Sicht der Funktionenprojektion bietet das klassische Verfahren die Möglichkeit des Einsatzes konstanter Basisfunktionen, während der Wegfall genannter Beschränkungen bedeutet, daß man auch Basisfunktionen höherer Ordnung zuläßt, um mit ihrer Hilfe die Radiosity-Funktion zu approximieren (*higher order Radiosity*). Diese Erweiterung hat zur Folge, daß man für jedes Patch nicht nur einen konstanten Farbwert angeben kann, sondern das Farbverläufe direkt darstellbar sind, ohne eine feinere Unterteilung des Patches vorzunehmen. Bemühungen in diese Richtung der Erweiterung gibt es bei Heckbert [14], der Radiosity-Funktionen betrachtete, die stückweise linear sind, sowie bei Zatz [28], der mit Hilfe von Legendre-Polynomen Funktionen berechnete, die sich stückweise wie Polynome höherer Ordnung verhalten. Zusammengefaßt wurde die Verwendung von Basen höherer Ordnung unter dem Begriff *Galerkin Radiosity* GR. Es wurde gezeigt, daß mit Hilfe dieser Verfahren eine geringere Anzahl von Basisfunktionen benötigt wird, allerdings entstehen höhere Kosten für jede dieser Basisfunktionen.

Eine zweite Möglichkeit der Erweiterung gelang Hanrahan et al. [12] durch Einsetzung eines hierarchischen Verfahrens (*Hierarchical Radiosity* HR). Es wurde erfolgreich der Versuch unternommen,

die Komplexität der Lösung zu senken, indem die Anzahl der zu berechnenden Formfaktoren verringert wurde. Diese Formfaktoren drücken für ein Paar von Patches ihre gegenseitige geometrische Lage aus. Untersucht man das CR-Verfahren, so scheint eine Komplexität von  $O(n^2)$  unausweichlich, wobei  $n$  die Anzahl der Oberflächenelemente in der Szene bezeichnet. Dies ergibt sich aus der Notwendigkeit, daß alle Elemente paarweise betrachtet werden müssen, um einen möglichen Energieaustausch zwischen ihnen zu modellieren. Mit anderen Worten: für  $O(n^2)$  Paare von Patches ist jeweils ein Formfaktor zu berechnen. Auch HR kann dieses Hindernis nicht überwinden, denn HR läßt, wie CR, keine dieser Interaktionen unbeachtet. Man beginnt die Berechnung jedoch mit einer wesentlich kleineren Anzahl von Patches als bei CR. Dies ist möglich, da die Patches während des laufenden Verfahrens, wenn nötig, verfeinert werden. Man bemüht sich also, die Anzahl der Interaktionen zwischen Elementen und somit die Berechnung von Formfaktoren gering zu halten. Das folgende Beispiel zeigt, daß es diese Möglichkeit gibt:

Angenommen man hat zwei Gruppen von Elementen in der Szene, die aber sehr weit voneinander entfernt angeordnet sind. Es ist leicht einzusehen, daß man die vollständige Interaktion zwischen diesen beiden Gruppen durch einen einzigen Formfaktor darstellen kann, ohne dabei einen großen Fehler zu machen. Dieser Formfaktor wird nur einmal für die gesamte Gruppe berechnet und nicht für jedes einzelne Element, das in ihr enthalten ist.

Hanrahan et al. entwickelten eine Methode, wie die Patches einer Szene hierarchisch unterteilt werden können, zusammen mit einer Vorgehensweise, wie Energie zwischen den einzelnen Elementen solcher Hierarchien ausgetauscht werden kann. Diese Hierarchie bietet nun die Möglichkeit, Interaktionen zwischen Gruppen von Patches zu modellieren. Das Verfahren erreicht es, viel weniger als  $O(n^2)$  Interaktionen zu berechnen; in der Tat wurde gezeigt, daß man innerhalb einer gegebenen Fehlertoleranz mit der Berechnung von  $O(n)$  Formfaktoren auskommt.

Das oben beschriebene hierarchische Radiosity-Verfahren HR kann alternativ auch mit Hilfe von hierarchischen Basisfunktionen beschrieben werden. Dieser Ansatz macht es möglich, andere geeignete hierarchische Basisfunktionen in die Betrachtung miteinzubeziehen. Diese können über den stückweise konstanten Ansatz hinausgehen. Als Alternative bietet sich hier eine Familie von Basen, die Wavelets, an.

Dem *Wavelet Radiosity* Verfahren (WR) kommt der Verdienst zu, die beide oben genannten Ansätze unter dem Dach einer gemeinsamen Theorie zu vereinigen. Wavelets bilden hierbei das mathematische Werkzeug, mit dessen Hilfe ein allgemeiner Rahmen entworfen wird, in dem nun beide Methoden gemeinsam betrachtet und erweitert werden können. Das Verfahren und die Herleitung von Wavelet Radiosity wurde 1993 von Gortler et al. vorgestellt [10].

Abbildung 1.1 setzt die genannten verschiedenen Verfahren in Beziehung zueinander. Die horizontale Achse bezeichnet dabei den Grad der genutzten Basisfunktionen, die vertikale Achse drückt die Art der Hierarchiebildung aus. CR nutzt konstante Funktionen, also Polynome nullter Ordnung. GR setzt Polynome höherer Ordnung ein (ausgedrückt durch den Pfeil). HR macht sich konstante Glätte des Radiosity-Kernels zu Nutze und kann als Wavelet erster Ordnung angesehen werden. Dieser Sichtweise würde Wavelet Radiosity mit Haar-Basis entsprechen. Weiterhin wurden zwei Familien von Wavelets höherer Ordnung untersucht: Flatlets und Multiwavelets [1]. Flatlets wurden in [10] eingeführt und bieten den Vorteil, daß keine Quadraturmethoden höherer Ordnung benutzt werden müssen, sie aber trotzdem die meisten Vorteile der anderen Wavelets bieten.

Im Rahmen dieser Arbeit soll jedoch weniger eine Vorstellung bekannter Theorien präsentiert werden, da dies an anderer Stelle bereits ausführlich geschehen ist. Die sich hier präsentierende Aufgabe besteht darin, Probleme, die bei der Implementierung von WR auftreten, zu behandeln, sowie Fragen zu beantworten, die an anderer Stelle ausgelassen wurden.

Bei der Implementierung wurden fünf verschiedene Wavelet-Basen berücksichtigt (Haar, Flatlets mit zwei und drei vanishing Moments, sowie Multiwavelets mit zwei und drei vanishing Moments). Diese

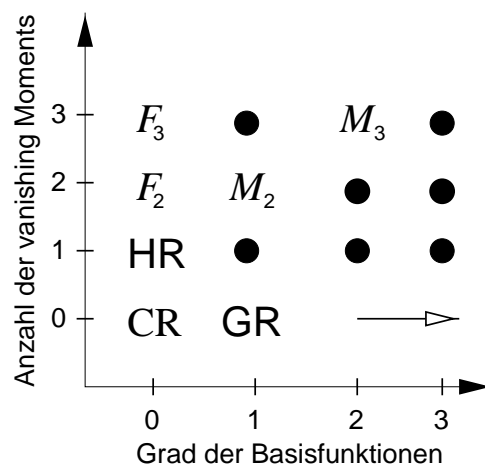


Abbildung 1.1: Übersicht über Radiosity-Methoden: klassische Radiosity (CR), hierarchische Radiosity (HR) und Galerkin Radiosity (GR). Wavelet Radiosity gliedert sich in die verschiedenen Wavelet-Basen der Flatlets ( $F_2$  und  $F_3$ ) und Multiwavelets ( $M_2$  und  $M_3$ ) (nach [10]).

können als Beispiel für die Implementierung eigener Basen herangezogen werden. Die neuen Klassen wurden so konzipiert, daß der Aufwand hierfür möglichst gering ausfällt.

Eine Implementierung setzt immer eine Umgebung voraus, in die das neue Verfahren eingebettet wird. Der zweite und nicht minder wichtige Teil der Arbeit besteht deshalb darin, zu erläutern, wie WR in das 3D-Graphiksystem MRT eingearbeitet wurde: welche Schwierigkeiten sind aufgetreten, wie wurden sie gelöst und welche Besonderheiten des MRTs führen zu Änderungen im allgemeinen Verfahren?

Der MRT (*Minimal Rendering Toolkit*) ist ein 3D-Graphiksystem, das in der Abteilung für Computergraphik der Universität Bonn entwickelt wurde. Viele verschiedene Rendering-Verfahren, d.h. computergraphische Verfahren, um Szenen darzustellen, sind in dieses System eingebracht worden, darunter auch Ray-Tracing und ein hierarchischer Radiosity-Ansatz. Mit Hilfe verschiedener Eingabeformate können virtuelle Szenen übergeben werden, aus deren Beschreibung der MRT dann Bilder von photorealistischer Qualität erzeugt. Bei dieser Berechnung gehen eine große Anzahl von Gestaltungsmöglichkeiten ein, wie z.B. die Beschaffenheit einzelner Oberflächen, der Lichtquellen und des Mediums, durch das sich das Licht zwischen den Lichtquellen und den Objekten bewegt. Eine Vielzahl unterschiedlicher 3D-Objekte steht für eine Darstellung zur Verfügung.

Mit Hilfe des MRTs ist es neben der Gestaltung eigener Szenen aber auch ohne größere Probleme möglich, eigene Algorithmen der Computergraphik zu realisieren, seien es Optimierungen oder komplett neue Verfahren.

Ein wesentlicher Punkt beim Umgang des Programmierers mit dem MRT ist die objekt-orientierte Beschaffenheit des MRTs sowie eine topologische Datenstruktur, die allen zu behandelnden 3D-Objekten zugrunde liegt: Die Boundary Representation oder kurz BRep. Durch diese Struktur ist es möglich, bei jedem Schritt nicht nur die Approximation eines Objektes zu betrachten, sondern direkt auf die exakte Definition zurückzugreifen. Gerade bei Verfahren, die darauf beruhen, die grobe Darstellung eines 3D-Objektes so weit zu verfeinern, bis eine angemessene Darstellung entsteht, ist dieser Punkt ausgesprochen wichtig und tragend. Hierarchische Radiosity-Algorithmen und deren Erweiterungen, wie WR, stellen solche Verfahren dar.

MRT steckt somit die Grenzen ab, in der sich das neu eingebrachte Verfahren WR entfalten darf und muß. Die sich hieraus ergebenden Vorteile und Einschränkungen werden im zweiten Teil der Arbeit behandelt.

Im folgenden Text werden englische Ausdrücke und Fachtermini benutzen, weil sie durch eine Übersetzung ins Deutsche zumindest einen Teil ihrer Aussagekraft verlieren würden. Weiterhin haben sich in der Radiosity-Literatur viele englische Ausdrücke eingebürgert (siehe schon Bezeichnung „Radiosity“) und durch deren Verwendung im Rahmen dieser Arbeit wird gewährleistet, daß man Begriffe, die man sich bereits angeeignet hat, hier wiedererkennt.

# Kapitel 2

## Radiosity-Grundlagen

An dieser Stelle soll ein kurzer Überblick über die Grundlagen der Radiosity gegeben werden. Eine vollständige Einführung würde nicht nur den Rahmen dieser Arbeit, sondern auch so manchen Buches sprengen. Weiterführende ausführliche Einleitungen und Erläuterung finden sich bei Glassner [8], Cohen & Wallace [6] und Sillion & Puech [24].

### 2.1 Klassische Radiosity (*Classical Radiosity*, CR)

Das Verfahren der klassischen Radiosity wurde 1984 von Goral et al. [9] und Nishita und Nakamae [16] entwickelt. Es leitet sich von Methoden der Wärmetechnik ab und modelliert die Ausbreitung des Lichtes innerhalb einer geschlossenen Szene mit Hilfe eines Energiegleichgewichtarguments.

Die grundlegenden Annahmen im Ansatz der CR sind:

- Die Lichtstrahlen innerhalb der zu betrachtenden Szene bewegen sich durch ein vollkommenes Vakuum. Es gibt kein Medium in der Szene, das Einfluß auf die Lichtstrahlen ausübt (kein *Participating Media*).
- Alle Oberflächen in der Szene sind vollständig diffus (lambertisch diffus) und opak. Es gibt somit keine transparenten Flächen.
- Jede Oberfläche hat eine konstante Farbe, d.h. Farbänderungen über einer Oberfläche sind nicht möglich. Um einen solchen Verlauf darzustellen, muß diese in mehrere Teilflächen unterteilt werden.

Der erste Schritt des Verfahrens besteht deshalb darin, die Oberflächen der einzelnen Objekte in kleinere Stücke, die sogenannten *Patches*, zu unterteilen. Dieser Vorgang wird als *meshing* bezeichnet (*mesh*, engl. Netz).

Durch die oben gemachten Annahmen ergibt sich, daß das reflektierte, wie auch das einfallende und ausgestrahlte Licht eines Patches sowohl für alle Punkte auf dem Patch, als auch für alle Richtungen konstant ist. Hat man nun eine bestimmte Wellenfrequenz des Lichtes gegeben, so kann man die Energie, die das Patch verläßt, durch einen einzigen skalaren Term beschreiben. Dieser Term setzt sich aus der Emission (die eigene Strahlung des Patches) und der *radiant exitance* (die von anderen Patches empfangene und reflektierte Strahlung) des Patches zusammen. Die *radiant exitance* ist dabei besser bekannt als *Radiosity* und ist definiert als ausgestrahlte Energie pro Flächeneinheit.

Wie bereits gesagt ist die Radiosity eines Patches unabhängig von einer bestimmten Richtung. Dies führt dazu, daß Szenen nach der Berechnung der einzelnen Radiosity-Werte aus jedem beliebigen Blickwinkel betrachtet werden können, ohne daß eine neue Berechnung nötig wird. Es ergibt sich ein blickwinkelunabhängiges Ergebnis.

Der Radiosity-Algorithmus bildet nun eine Menge von linearen Gleichungen, mit deren Hilfe die Patches, die in der Szene enthalten sind, in Beziehung zueinander gestellt werden. Diese Beziehung sieht so aus, daß beruhend auf geometrischen Informationen festgestellt wird, wieviel von der Energie, die ein Patch verläßt, bei einem anderen Patch ankommt. Ist in der Szene Energie vorhanden, so tauschen zwei Patches, die sehr nah zusammenliegen und sich gegenseitig zugewandt sind, viel Energie aus. Zwischen zwei kleinen Patches, die weit von einander entfernt sind, wird nur wenig oder gar keine Energie transferiert.

Das resultierende Gleichungssystem wird dann üblicherweise durch iterative Methoden, wie Gauss-Seidel-, Jacobi- oder Southwell-Iteration, gelöst. Das Ergebnis besagt dann, wieviel Licht von jedem Patch in die Szene abgegeben wird. Bestimmt man nun, welche Patches vom Betrachtungspunkt aus sichtbar sind, so kann ein Bild der Szene dargestellt werden. Sollte sich der Standort der virtuellen Kamera verändern, so muß nur die Sichtbarkeitsberechnung wiederholt werden, nicht aber die eigentliche Radiosity-Berechnung, da die Energie, die von jedem Patch abgegeben wird, von der Position des Auges unabhängig ist.

Jeder Punkt auf dem Patch gibt die gleiche Energie ab, d.h. jedes Patch in der Szene hat eine konstante Färbung. Durch diese Einschränkung müssen Bereiche, die Farbverläufe aufweisen, stark unterteilt werden, um diese Besonderheiten darstellen zu können. Ein Beispiel für eine extreme Veränderung wäre eine Schattenkante, die quer über das Patch verläuft. Um sie darstellen zu können, müßte das Patch entlang dieser Kante entsprechend fein unterteilt werden. Es bleibt aber auch weiterhin das Problem bestehen, daß man keinen fließenden Farbübergang zwischen benachbarten Patches erreicht. In der Praxis ist es deshalb üblich, nach der eigentlichen Berechnung noch einen Glättungsschritt durchzuführen. Dabei werden die Farbwerte für die Ecken eines Patches durch Interpolation aller Farbwerte der angrenzenden Patches berechnet. Die Anzeige erfolgt dann mit Hilfe einer linearen Interpolation dieser neu gewonnenen Farbwerte entlang der Polygongrenzen (Gouraud-Schattierung). Diese Vorgehensweise ist nicht unproblematisch (siehe Erläuterungen zur Problematik der Gouraud-Schattierung in [6]).

Es gibt verschiedene Optimierungen und Abwandlungen der klassischen Vorgehensweise, auf die an dieser Stelle nicht eingegangen wird. Dies gilt ebenso für einen weiteren wichtigen Punkt bei Radiosity-Verfahren: Die Berechnung von Formfaktoren (siehe [6], [8] oder [24] für Vorgehensweise und Einzelheiten).

Wichtig hier ist nur, daß es sehr aufwendig ist, Formfaktoren zu berechnen. Ein möglicher Weg, um diese teuren Berechnungen zu vermeiden, beschreibt das HR-Verfahren.

## 2.2 Hierarchische Radiosity (*Hierarchical Radiosity*, HR)

Die meiste Zeit wird bei einem Radiosity-Programm für die Durchführung zweier Schritte benötigt:

1. Die Berechnung der Formfaktoren
2. Das Lösen des linearen Gleichungssystems

Würde man es schaffen, die Anzahl der beteiligten Formfaktoren zu verringern, so ergäbe sich für beide Punkte ein zeitlicher Gewinn: Es müßten nicht nur weniger Formfaktoren berechnet werden, sondern die Größe des Gleichungssystems würde sich ebenfalls verringern. Doch wie soll es möglich sein, die Berechnung von Formfaktoren wegzulassen, da doch sie es sind, die die geometrischen Informationen über die Lage der einzelnen Patches zu einander beinhalten, eine Information, welche für die Berechnung der Lichtinteraktion unerläßlich ist? Eine entscheidende Erkenntnis war hier die Feststellung, daß das Formfaktorproblem viel mit einem anderen Problem aus der klassischen Physik gemeinsam hat: dem N-Körper Problem.

### 2.2.1 Das N-Körper Problem

Man betrachte ein System von  $N$  unabhängigen, massiven Körpern im Raum. Jeder dieser Körper wirkt mit einer gravitativen Kraft auf alle anderen Körper. Damit ergeben sich bei diesem Problem  $N \cdot (N - 1)/2$  mögliche Interaktionen. Um nun herauszufinden, in welche Richtung sich jeder Körper bewegt, muß man *alle* diese Interaktionen beachten.

Dies gilt für die theoretische Lösung des Problems. In der Praxis muß dies aber nicht mit absoluter Exaktheit geschehen, denn wenn Computer und Maschinen zur Berechnung eines mathematischen Problems eingesetzt werden, so ist die Exaktheit der Lösung schon durch die Genauigkeit der eingesetzten Hard- und Software begrenzt. Durch einen tolerierten Fehler ergibt sich die Möglichkeit, einzelne Objekte zu Gruppen oder *clustern* zusammenzufassen und sie nach außen hin als eine Einheit zu repräsentieren, die ein Verhalten zeigt, das durch alle Mitglieder der Gruppe bestimmt wird.

Hanrahan und Salzmann [11] stellten fest, daß sich dies auch auf das Formfaktorproblem übertragen läßt, da beide Probleme Ähnlichkeiten aufweisen: Bei beiden geht es um die Interaktion zwischen Paaren von Objekten. Weiterhin verhalten sich Gravitation und Formfaktoren proportional zur Größe der Objekte und umgekehrt proportional zur quadratischen Entfernung zwischen den Objekten. Die beiden Probleme sind jedoch nicht identisch, denn zwischen Gravitation und Licht bestehen Unterschiede; es gibt aber genügend ausnutzbare Gemeinsamkeiten.

### 2.2.2 Beschreibung des Algorithmus

Der Grundgedanke dieses Verfahrens ist, daß kleine Details einer Szene unwichtig werden, wenn man nur weit genug von ihnen entfernt ist. Detailinformationen müssen vorhanden sein, wenn sie benötigt werden sollten, ansonsten ist es aber nicht nötig, sie zu berechnen. Interaktionen sollten möglichst auf einer angemessenen Detailstufe (*level-of-detail*) betrachtet werden.

Das Herzstück des Verfahrens ist eine Hierarchie von unterteilten Patches. Um diese Hierarchie zu erhalten, beginnt man mit einer Menge von  $n$  großen Patches und berechnet die zugehörigen Formfaktoren. Dies bedeutet, daß auch hier  $n \cdot (n - 1)/2$  Interaktionen betrachtet werden müssen. Der große Unterschied zum klassischen Verfahren besteht jedoch darin, daß diese Ausgangs-Patches sehr groß sein können, viel größer als die Patches sein dürften, um durch CR ein akzeptables Ergebnis zu erhalten. Dies wiederum bedeutet, daß die Anzahl der Patches  $n$  weit unter der Zahl der Patches beim CR liegt. Das heißt: Der erste Schritt des Algorithmus hat wie CR eine Komplexität von  $O(n^2)$ , aber  $n$  ist hier weitaus kleiner.

Betrachtet man zum Beispiel eine Szene, die eine Wand enthält, die von einer Lichtquelle beleuchtet wird: Bei CR ist man gezwungen die Wand in viele kleine Patches zu unterteilen, um den Farbverlauf von gut beleuchteten Partien hin zu den Bereichen, auf die weniger Licht fällt, darstellen zu können. Bei einer gleichmäßigen Unterteilung führt dies in Bereichen, die keinen Farbverlauf aufweisen, zu einer unnötigen, starken Verfeinerung. Beim HR-Ansatz genügt als Ausgangspunkt ein einziges, großes Patch, das die gesamte Wand darstellt.

Nachdem nun alle Interaktionen zwischen den Ausgangs-Patches berechnet wurden, betrachtet man Paare von Patches dahingehend, wie groß der Fehler wäre, wenn man die betreffenden Patches nicht weiter unterteilen würde. Auf die Frage, wie dieser Fehler bestimmt wird, soll an dieser Stelle nicht weiter eingegangen werden. Eine Möglichkeit ist hierbei, den berechneten Formfaktor als Maß zu nehmen: Wenn der Formfaktor sehr groß ist, wird viel Energie von einem Patch zum anderen übertragen und es kann deshalb sinnvoll sein, die beteiligten Patches noch weiter zu unterteilen. Sollte der Fehler jedoch sehr gering sein, so kann man sich mit der gröberen Unterteilung zufrieden geben. Für die eventuell neu geschaffenen Sub-Patches werden neue Formfaktoren berechnet und das Verfahren wird rekursiv fortgesetzt. Auf diese Weise erhält man für jedes Ausgangs-Patch eine Patch-Hierarchie. Jeder innere Knoten in dieser Hierarchie repräsentiert dabei eine Gruppe von Patches durch ein gemeinsames Verhalten. Interaktion mit diesem Knoten repräsentiert die Interaktion mit der entsprechenden Gruppe.

Nun hat es den Anschein, daß man durch diese Vorgehensweise nicht etwa Aufwand einspart, sondern sich im Gegenteil noch mehr Arbeit aufbürdet. Dies ist aber nicht der Fall, da man nicht Interaktionen von jedem Element der Hierarchie zu jedem Element einer anderen Hierarchie betrachten muß, sondern nur ausgewählte Formfaktoren berechnet werden. Wenn man sich die Matrix des klassischen Verfahrens vorstellt, so stellt ein Formfaktor zwischen einem inneren Knoten der Hierarchie zu einem Blatt einen konstanten Block innerhalb der Matrix dar: Alle Formfaktoren der Kinder des Knotens werden mit dem Wert des Knotens angegeben. Deshalb wird bei diesem Verfahren die Matrix der Formfaktoren auch nicht explizit angelegt, weil dies eine ineffiziente Art der Speicherung darstellt. Vielmehr wird die Matrix implizit über eine Liste von *Links* verwaltet, die zwischen den einzelnen Patches aufgespannt werden.

Hat man nun alle Patches so weit unterteilt, daß der jeweilige Fehler unter einer gewünschten Grenze liegt, wird das Licht entlang der berechneten Links in der Szene verteilt (*Gathering*). Nach diesem Schritt ist es nun wichtig, daß die Energie, die auf verschiedenen Stufen der Hierarchie gesammelt wurde, auf den gesamten Baum konsistent verteilt wird, damit dieser für den nächsten Verteilungsschritt bereit ist. Hierfür wird ein sogenanntes *Push/Pull* durchgeführt, wobei ein Push den Vorgang bezeichnet, in dem ein Vater seine gesammelte Energie an die Kinder weitergibt, ein Pull beschreibt die entgegengesetzte Richtung. Einzelheiten hierfür sind bei [6], [8], [12], [24] nachzulesen. Die beiden Schritte *Gathering* und *Push/Pull* werden nun solange wiederholt, bis ein *Equilibrium*, ein Gleichgewicht der verteilten Energie in der Szene, erreicht wird.

Um nun zu verstehen, worin die Erweiterung der Wavelet Radiosity liegt und warum man den oben beschriebenen Ansatz der hierarchischen Radiosity als einen Spezialfall der Wavelet Radiosity ansehen kann, muß man die beschriebenen Verfahren aus einem weniger intuitiven Blickwinkel betrachten.

## 2.3 Die Radiosity-Integralgleichung

Kajiya schlug 1986 eine Integralgleichung vor, um das globale Beleuchtungsproblem zu lösen [15]. Diese Integralgleichung wurde *Rendering-Gleichung* (*rendering equation*) genannt und er zeigte, daß man CR als eine spezielle Approximation dieser allgemeinen Gleichung ansehen kann. Durch die Umformulierung des Problems in den Kontext der Integralgleichungen wurde es möglich, Techniken aus diesem Bereich nun auch auf die Lösung der Radiosity-Gleichung anzuwenden.

Wenn alle Oberflächen und Emitter einer Szene lambertisch diffus sind, kann man die *Rendering-Gleichung* in folgender Form darstellen:

$$B(s_1, s_2) = E(s_1, s_2) + \rho(s_1, s_2) \int \int \frac{\cos \theta_s \cos \theta_t}{\pi r_{st}^2} V_{st} B(t_1, t_2) dt_1 dt_2 \quad , \quad (2.1)$$

wobei  $B(s_1, s_2)$  die Radiosity angibt, die einen bestimmten Punkt verläßt. Dieser Punkt wird durch die Oberflächenparameter  $s_1$  und  $s_2$  spezifiziert.  $E$  bezeichnet die Emission und  $\rho$  die Reflektivität. Wir betrachten  $\rho$  hierbei als monochrom, um die Formeln übersichtlich zu halten. Eine Erweiterung auf eine Abhängigkeit von der Wellenlänge des Lichtes bringt hier keinen weiteren Erkenntnisgewinn. Ohne den Anspruch auf Allgemeingültigkeit zu verlieren, beschränken sich die Ausführungen im weiteren Verlauf der Arbeit auf eine rein monochrome Welt.

Betrachtet man nur die Teile der Gleichung, die die geometrischen Informationen und die Sichtbarkeit zwischen den Patches enthalten, so erhält man den Kern (*Kernel*) des Intergrals

$$k(s_1, s_2, t_1, t_2) = \rho(s_1, s_2) \frac{\cos \theta_s \cos \theta_t}{\pi r_{st}^2} V_{st}.$$

$\theta_s$  und  $\theta_t$  bezeichnen die Winkel zwischen der Oberflächennormalen und der Verbindungslinie, die durch  $s$  und  $t$  verläuft;  $r_{st}$  bezeichnet den Abstand zwischen den beiden Punkten und  $V_{st}$  ist der Sichtbar-

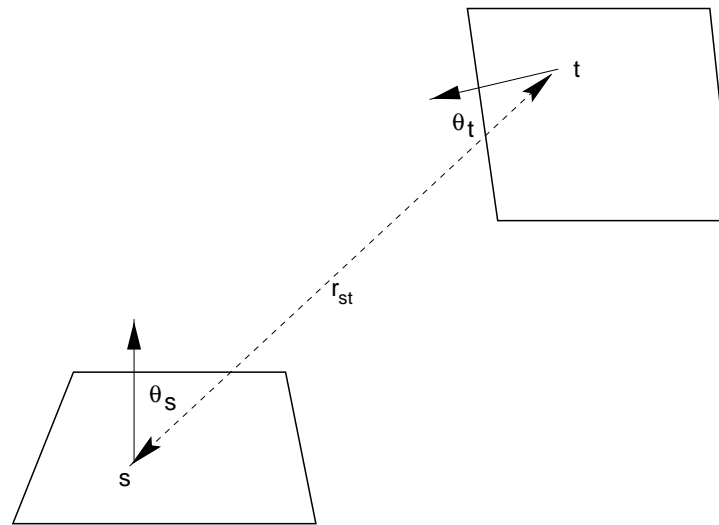


Abbildung 2.1: Anschauliche Darstellung der in die Bestimmung des Kerns einfließenden Parameter (Kernel-Geometrie).

keitsterm dieser Verbindung. Dieser hat den Wert Eins wenn Punkt  $s$  von Punkt  $t$  aus sichtbar ist, sonst hat er den Wert Null (siehe Abbildung 2.1).

Ist  $r$  sehr groß in Relation zur Größe der Patches, ist es möglich den Kernel durch ein Polynom niedrigen Grades zu approximieren. Man kann sich vorstellen, daß der Austausch zwischen zwei Patches, die weit auseinander liegen, nicht stark über diese variiert, d.h. wenn man den Austauschpunkt auf einem der Patches leicht verschiebt, wird sich der resultierende Wert, wenn überhaupt, nur sehr gering verändern.

Es gibt natürlich auch hier Ausnahmen: An den Schattenrändern, also an den Übergängen zwischen einem Teil, der beleuchtet wird und einem Teil, der sich im Schatten befindet, entstehen Diskontinuitäten innerhalb des Kerns. Zur Veranschaulichung seien wieder zwei Punkte auf verschiedenen Patches gegeben, die sich gegenseitig sehen können. Verschiebt man einen der Punkte um ein kleines Stück, so kann es vorkommen, daß der leicht verschobene Punkt nun bezüglich des anderen Punktes durch ein Objekt verdeckt wird. Der Sichtbarkeitsterm wechselt also von Eins auf Null und mit ihm ebenfalls der Wert des Kerns. Dieser Wechsel ist sehr abrupt, weil mit dem Sichtbarkeitsterm auch der Kernel den Wert Null annimmt.

Eine weitere Ausnahme liegt vor, wenn zwei Patches sehr nahe zusammenliegen oder sich berühren: Der Abstand zwischen ihnen geht gegen Null. Für den Wert des Kerns bedeutet dies, daß er gegen unendlich strebt, d.h. der Kernel ist singular, eine Repräsentation durch ein Polynom geringen Grades ist nicht mehr ohne einen großen Fehler möglich.

Es gilt also, solche Diskontinuitäten gesondert zu betrachten und eine entsprechende Handhabung dieser Fälle vorzusehen.

Wie löst man nun aber eine Gleichung der oben beschriebenen Form? Eine Möglichkeit stellt die Projektion in einen endlichdimensionalen Funktionenraum dar.

# Kapitel 3

## Projektionen

Es folgt eine kurze Übersicht über die Projektion von Funktionen und deren Anwendung, um eine approximierte Lösung der im vorherigem Kapitel beschriebenen Integralgleichung zu finden. Das allgemeine Konzept zur Anwendung von Projektionsmethoden auf die Radiosity-Gleichung wurde von Heckbert [13] eingeführt. Eine detaillierte Betrachtung von Projektionen findet man weiterhin in [20] und [28].

### 3.1 Projektion in einen endlichdimensionalen Funktionenraum

Ausgangspunkt an dieser Stelle sei eine Funktion  $B(s)$ . Man Betrachte nun einen endlichdimensionalen Funktionenraum, in dem alle in ihm enthaltenen Funktionen  $\hat{B}(s)$  als eine Linearkombination von  $n$  Basisfunktionen ausgedrückt werden können. Das Ziel ist es, eine Funktion aus diesem Funktionenraum zu finden, die eine gute Approximation von  $B(s)$  darstellt. Es handelt sich hierbei um eine Approximation, weil der genannte Funktionenraum endlichdimensional sein soll, für  $B(s)$  wurden jedoch keine Einschränkungen gemacht. Diese Approximation kann auf folgende Weise ausgedrückt werden:

$$B(s) \approx \hat{B}(s) = \sum_{i=1}^n B_i N_i(s),$$

wobei es sich bei den  $B_i$  um die entsprechenden Koeffizienten im Bezug auf die gewählten Basisfunktionen  $N_i$  handelt.

Wie kann man sich nun diese Approximation vorstellen? Der Raum der stückweise konstanten Funktionen wird beispielsweise durch eine Basis aufgespannt, die aus einer „Box“-Funktion und deren Translationen besteht. Versieht man diese Basisfunktionen nun mit entsprechenden Koeffizienten, ergibt sich eine stückweise konstante Approximation der Ausgangsfunktion  $B(s)$ . Die Qualität dieser Annäherung entscheidet sich durch die Eigenschaften der Ausgangsfunktion. So wird eine lineare Funktion besser durch eine Projektion in einen Funktionenraum approximiert, der nicht durch „Box“-, sondern durch „Hat“-Funktionen aufgespannt wird. Durch eine solche Basiswahl läßt sich ein stückweise linearer Verlauf erreichen. Wichtig ist die Beobachtung, daß die Wahl der Basis des Raumes, in den projiziert wird, ebenfalls einen großen Einfluß auf die Güte der Approximation hat.

Gegeben sei nun die Ausgangsfunktion und der Raum, in den projiziert wird. Die Bestimmung der oben erwähnten Koeffizienten  $B_i$  ist durch die Berechnung des inneren Produktes der Ausgangsfunktion mit den Basisfunktionen möglich. Hierzu zunächst die benötigten Definitionen:

- **Definition:** inneres Produkt

Das innere Produkt zweier Funktionen  $f(s)$  und  $g(s)$  ist definiert als

$$\langle f, g \rangle = \int f(s)g(s) ds \quad .$$

- **Definition:** orthogonale Funktion

Zwei Funktionen  $f$  und  $g$  sind orthogonal, wenn gilt

$$\langle f, g \rangle = 0 \quad .$$

- **Definition:** orthogonale Familie von Funktionen

Eine Familie  $f_i$  von Funktionen heißt orthogonal, wenn gilt

$$f_k \text{ ist orthogonal zu } f_l \quad \forall f_k, f_l \in f_i, k \neq l \quad .$$

- **Definition:** orthogonale Projektion

$\hat{B}(s)$  ist die orthogonale Projektion von  $B(s)$  in den endlichdimensionalen Funktionenraum  $R$ , wenn gilt

$$\langle B - \hat{B}, N_i \rangle = 0 \text{ für alle Basisfunktionen } N_i \text{ von } R.$$

- **Definition** orthonormal

Eine Familie  $f_i$  von Funktionen heißt orthonormal, wenn gilt

$$f_i \text{ ist orthogonal und } \forall f \in f_i \text{ gilt: } \langle f, f \rangle = 1 \quad .$$

- **Definition** duale Funktion

Sei  $N_i$  eine Familie von Funktionen. Dann ist die zu  $N_i$  duale Familie von Funktionen  $\hat{N}_j$  definiert durch die Eigenschaft

$$\langle N_i, \hat{N}_j \rangle = \delta_{ij} \quad ,$$

wobei es sich bei  $\delta_{ij}$  um das Kronecker Delta handelt.

Mit Hilfe dieser Werkzeuge können nun die gesuchten Koeffizienten berechnet werden. Man bestimmt die Koeffizienten einer Funktion  $B(s)$  bezüglich dieser Basis durch Berechnung des inneren Produktes  $\langle B, \hat{N}_i \rangle$ . Es gilt somit

$$\hat{B}(s) = \sum_{i=1}^n B_i N_i(s) = \sum_{i=1}^n \langle B, \hat{N}_i \rangle N_i(s) \quad .$$

Sind die Basisfunktionen orthonormal, gilt für die Koeffizienten  $B_i = \langle B, N_i \rangle$ , d.h. man bildet das innere Produkt der Funktion mit der ursprünglichen, und nicht mit der dualen Basisfunktion. Für die Funktion  $B(s)$  gilt somit im orthonormalen Fall

$$\hat{B}(s) = \sum_{i=1}^n B_i N_i(s) = \sum_{i=1}^n \langle B, N_i \rangle N_i(s) \quad .$$

Zu denen, noch an späterer Stelle ausführlich vorgestellten Wavelet-Basen, die im Zuge dieser Arbeit implementiert wurden, sei hier angemerkt, daß es sich bei der Haar-Basis, wie auch bei Multiwavelets um orthonormale Basen handelt, während Flatlet-Basen nicht orthonormal sind. Weitere Ausführungen über Projektion für Wavelet Radiosity finden sich in [20].

## 3.2 Rendering-Gleichung im neuen Gewand

Betrachten wir noch einmal die spezielle Rendering-Gleichung (2.1):

$$B(s_1, s_2) = E(s_1, s_2) + \rho(s_1, s_2) \iint \frac{\cos \theta_s \cos \theta_t}{\pi r_{st}^2} V_{st} B(t_1, t_2) dt_1 dt_2 \quad .$$

Anstatt nun diese Integralgleichung zu lösen, löst man entsprechende projizierte Variante <sup>1</sup>

$$\hat{B}(s) = \hat{E}(s) + \sum_{i=1}^n \langle \int k(s, t) \hat{B}(t) dt, N_s(s) \rangle N_i(s) \quad .$$

Hier wird eine Integration des projizierten Kernel durchgeführt. Die resultierende Funktion liegt im allgemeinen nicht mehr im endlichdimensionalen Funktionenraum. Deshalb wird anschließend das Ergebnis wieder auf  $N_i(s)$  projiziert, um diese Eigenschaft zu gewährleisten.

$\hat{B}$  erhält man nun durch die Lösung des folgenden linearen Gleichungssystems:

$$B_i = E_i + \sum_{j=1}^n B_j K_{ij} \quad ,$$

wobei  $K_{ij} = \int \int k(s, t) N_j(t) N_i(s) dt ds \quad .$

An dieser Stelle kann bereits eine Ähnlichkeit zwischen dieser Formel und der Hauptformel aus dem Bereich der klassischen Radiosity auffallen. Verwendet man stückweise konstante Basisfunktionen, so ergeben sich aus den zu berechnenden Integralen die bekannten Formfaktoren. Benutzt man anders geartete Basen, so setzt man eine Form von numerischer Quadratur, oder für Spezialfälle auch eine geschlossene Lösung ein, um den Wert der Integrale zu ermitteln. Ein Beispiel für einen solchen Spezialfall ist der von Schröder und Hanrahan vorgestellte Ansatz, um den Formfaktor zwischen zwei Polygonen zu bestimmen [23].

Benutzt man den Mechanismus der Projektion, um die Ausgangsgleichung zu lösen, so sind zwei Fragen von besonderer Wichtigkeit:

1. In welchen Funktionenraum wird projiziert?
2. Wie sieht eine „geeignete“ Basis für diesen Funktionenraum aus?

Die Fragen sind deshalb von entscheidender Bedeutung, weil es sich bei der Lösung „nur“ um eine Approximation des Ergebnisses handelt: Es wird die projizierte Version der Gleichung gelöst, nicht die gegebene Ausgangsgleichung. Projektionen in unterschiedliche Funktionenräume werden auch unterschiedliche Fehler sowohl in ihrer Art, als auch im Ausmaß, zur Folge haben.

Im allgemeinen liegt der Projektionsfehler im Bereich  $O(h^{p+1})$ , wobei  $h$  die Auflösung des Raumes bezeichnet und  $p$  den Grad der Basisfunktionen. Somit scheinen auf den ersten Blick Basisfunktionen höherer Ordnung besser geeignet zu sein als Funktionen niedriger Ordnung. Durch Funktionen höherer Ordnung ist es auch möglich, glattere Radiosity-Lösungen zu berechnen und somit Artefakte in resultierenden Bildern zu vermeiden. Für diesen Vorteil zahlt man jedoch den Preis eines höheren Arbeitsaufkommens bei der Auswertung der inneren Produkte. Weiterhin stellen Schattenkanten, die über ein

<sup>1</sup>An dieser Stelle beschränken wir uns, der Einfachheit wegen, auf nur eine Variable. Die Erweiterung auf den dreidimensionalen Fall bedeutet, daß wir Basisfunktion in Abhängigkeit von zwei Variablen bekommen, durch die ein Punkt auf der Oberfläche eines Patches bestimmt wird. Da wir es immer mit Paaren von Patches zu tun haben, resultiert dies in einem Kernel, der in Abhängigkeit von vier Variablen steht. Auf die entsprechende Erweiterung wird an späterer Stelle eingegangen.

Patch verlaufen, einen Bruch dieses glatten Verlaufes dar, der ebenfalls angemessen umgesetzt werden muß.

Der entstehende Fehler ist also sowohl vom gewählten Funktionenraum als auch von dessen Basis abhängig. Es gilt an dieser Stelle eine zweckmässige Wahl zu treffen.

Bei der Wahl der Menge der Funktionen, die die Basis des Funktionenraumes ausmachen sollen, gibt es eine Vielzahl von Möglichkeiten. Eine davon besteht in einer Familie von Funktionen, die als Wavelets bezeichnet wird.

# Kapitel 4

## Wavelets

Eine allgemeine Einführung zum Thema Wavelets findet sich bei Glassner [8]. Auch bei Cohen & Wallace [6] findet sich eine kurze Beschreibung über die für die Radiosity relevanten Teile. Diese geht nicht sehr in die Tiefe, sie eignet sich aber gut, um ein erstes Verständnis des Mechanismus und der Vorgehensweise aufzubauen. Ein Ziel, das auch hier verfolgt wird. Eine weitere gute Einführung gibt Schröder in [22]. Er stellt die einzelnen Probleme und Vorgehensweisen in übersichtlicher Form dar.

### 4.1 Aufbau einer Wavelet-Basis

Ein großer Vorteil von Wavelets ist, daß sie eine *hierarchische* Basis für bekannte endlichdimensionale Funktionenräume bilden. An dieser Stelle sollen an Hand der einfachsten Wavelet-Basis, der Haar-Basis, die wesentlichen Eigenschaften und Mechanismen demonstriert werden.

Für den Aufbau einer Wavelet-Basis benötigt man zwei Funktionen:

1.  $\Phi(x)$ , die Glättfunktion (*smooth function* oder *scaling function*),
2.  $\Psi(x)$ , die Detailfunktion (*detail function*).

Abbildung 4.1 zeigt, wie man eine stückweise konstante Funktion mit Hilfe der Haar-Basis ausdrücken kann. Betrachtet man die beiden Box-Funktionen  $N_1$  und  $N_2$  und die Glätte- und Detailfunktionen  $\Phi$  und  $\Psi$ , so kann man sehen, daß beide den gleichen Funktionsraum aufspannen, d.h. eine Linearkombination der beiden Box-Funktionen oder der beiden Haar-Basen kann jede in den Intervallen  $x_i$  bis  $x_{i+2}$  stückweise konstante Funktion  $F(x)$  repräsentieren.

Ein entsprechendes Beispiel für den Aufbau der Funktion mit Hilfe der Haar-Basis wird ebenfalls in Abbildung 4.1 gezeigt. Hierbei ist der Koeffizient der  $\Phi$ -Funktion gleich dem Mittelwert über dem Intervall und der Koeffizient der  $\Psi$ -Funktion gleich der Abweichung von diesem Mittelwert.

Betrachten wir nun größere Intervalle:  $n$  sei eine Zweierpotenz und gebe die Breite des Intervalls an. Gegeben sei eine auf diesem Intervall stückweise konstante Funktion. Abbildung 4.2 zeigt am Beispiel  $n = 8$  wie nun aus Box-Funktionen die Wavelet-Basis entsteht. Dafür wird eine Hierarchie von  $\phi$ - und  $\psi$ -Funktionen aufgebaut.

Im folgenden bezeichnen  $\Phi$  und  $\Psi$  die ursprünglichen Waveletfunktionen, also Operatoren, die auf Funktionen angewendet werden.  $\phi$  und  $\psi$  hingegen beschreiben spezielle Funktionen, die in einer Wavelet-Basis enthalten sind. Diese werden mit Indizes versehen, um ihre Stufe in der Hierarchie zu bezeichnen bzw. eine Unterscheidung mehrerer Funktionen innerhalb einer dieser Stufen zu gewährleisten.

Beginnend mit acht Box-Basisfunktionen werden nun jeweils paarweise die beiden Operatoren angewandt. Es wird pro Paar einmal der Durchschnitt und die Differenz vom Durchschnitt gebildet. Die resultierenden acht Funktionen werden nach dem angewandtem Operator gruppiert (Abbildung 4.2 oben).

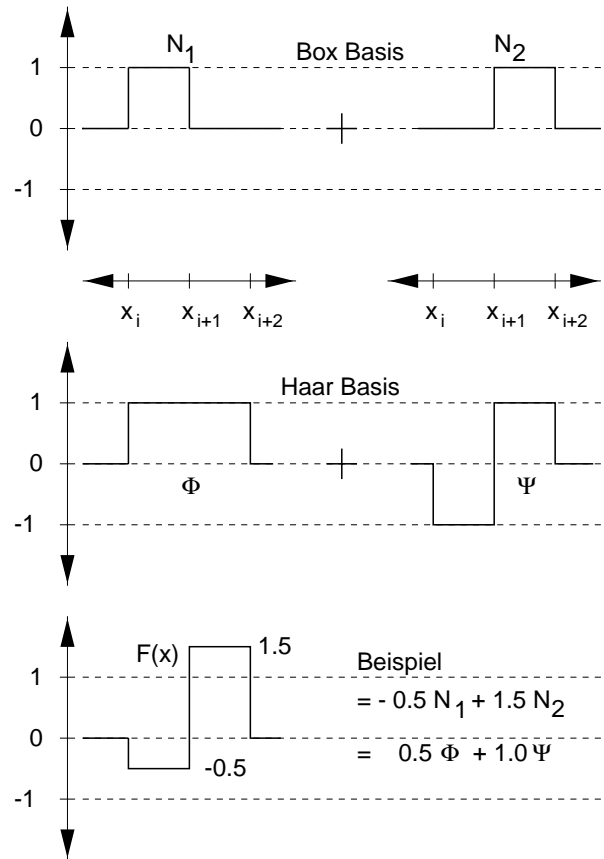


Abbildung 4.1: Konstruktion einer stückweise konstanten Funktion (unten) aus der Box- und Haar-Basis (oben bzw. in der Mitte). Die Box-Basis besteht aus den Funktionen  $N_1$  und  $N_2$ , die Koeffizienten geben den lokalen Wert der Funktion an. Die Haar-Basis besteht aus der Glättefunktion  $\Phi$  und der Detailfunktion  $\Psi$ . Die Werte der Koeffizienten ergeben sich aus dem Durchschnitt bzw. aus der Abweichung (nach [6]).

In der zweiten Zeile ist zu sehen, daß die vier  $\phi$ -Funktionen fast genau wie die Ausgangsfunktionen aussehen. Der Unterschied besteht darin, daß sie doppelt so breit wie ihre Vorgänger sind. Man kann also auf diese Funktionen wieder paarweise den Glätte- und Differenzoperator anwenden und das Ergebnis ordnen (Abbildung 4.2 Mitte). Wieder ergeben sich Funktionen, die den Ausgangsfunktionen auf die oben beschriebene Art ähneln. Diese sind nun viermal so breit wie die Funktionen, mit denen begonnen wurde. In der letzten Zeile des Beispiels werden die beiden neuen Funktionen als eine  $\phi$ - und eine  $\psi$ -Funktionen ausgedrückt. Das Ergebnis ist schließlich die hierarchische Haar-Wavelet-Basis (siehe Abbildung 4.3), die aus einer  $\phi_0$ -Funktion auf der obersten Stufe, sowie einer Pyramide von  $\phi_{i,j}$ -Detailfunktionen besteht.  $L$  bezeichnet die Tiefe der Hierarchie und entspricht  $\log_2(n)$ . Der zweite Index numeriert auf jeder Stufe die auf ihr vorhandenen Funktionen. Eine genaue Einführung dieser Schreibweise findet sich in Abschnitt 4.2.3.

Jede über Einheitsintervallen stückweise konstante Funktion  $F(x)$  kann als lineare Summe der Box-Funktionen dargestellt werden:

$$F(x) = \sum_{i=1}^n n_i N_i(x)$$

Nun kann  $F(x)$  genauso als eine Linearkombination der Haar-Wavelet-Basisfunktionen dargestellt

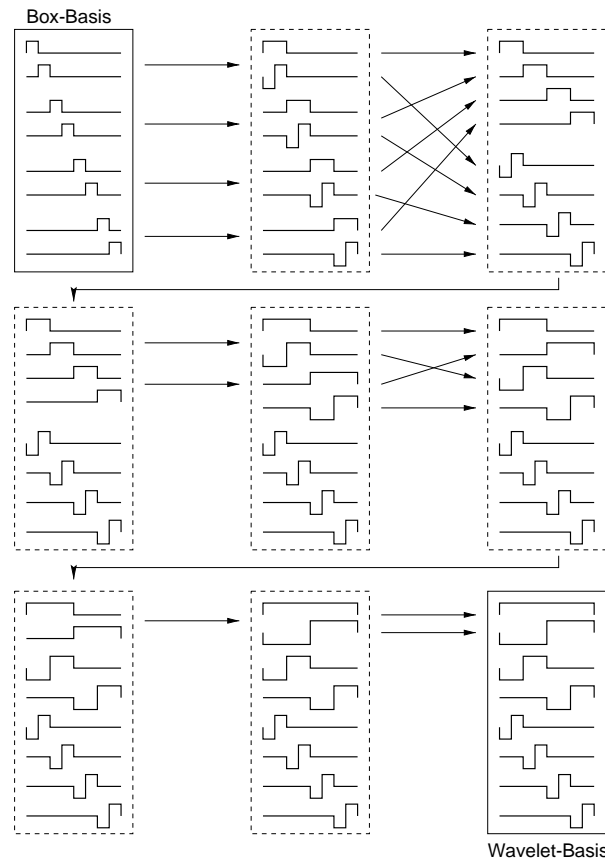


Abbildung 4.2: Konstruktion der hierarchischen Haar-Basis. Ausgehend von einer Box-Basis werden paarweise Durchschnitts- und Differenzbildungsschritte ausgeführt. Die Zwischenergebnisse werden nach dem angewendetem Operator sortiert (Nach [10]).

werden:

$$F(x) = b_{\phi_0} \phi_0(x) + \sum_{i=1}^L \sum_{j=1}^{n/2^{i-1}} b_{\psi_{i,j}} \psi_{i,j}(x) \quad (4.1)$$

Bei der Schreibweise mit Hilfe der Box-Basisfunktionen repräsentieren die Koeffizienten  $n_i$  den lokalen Werte von  $F(x)$  an entsprechender Stelle. Bei der Wavelet-Basis hingegen bezeichnet der Koeffizient  $b_{\phi_0}$  den Durchschnitt der gesamten Funktion über dem Intervall und die  $b_{\psi_{i,j}}$  repräsentieren die lokale Abweichung der Funktion  $F(x)$  von diesem Durchschnitt auf jeder Stufe  $i$ . Hieraus ergibt sich eine der Schlüsseleigenschaften der Wavelet-Basis: Ist die Funktion  $F(x)$  lokal gesehen glatt über einem Teil der Domäne (in diesem Fall hieße das konstant), wäre der Koeffizient, der die Abweichung dokumentiert, gleich Null.

Haben die Koeffizienten der Box-Darstellung noch den Wert an entsprechender Stelle angegeben, so werden sie nun bei der Haar-Darstellung durch die lokale Abweichung der Funktion vom Durchschnitt bestimmt. Sollte die Funktion also Bereiche haben, in denen sie konstant ist, so wären die zugehörigen Koeffizienten an dieser Stelle Null.

Betrachten wir nun das in Abbildung 4.4 dargestellte Beispiel einer stückweise konstanten Funktion: In der Mitte der Abbildung ist die Funktion selber zu sehen. Im unteren Teil ist die hierarchische Box-Basis und im oberen Teil die Haar-Basis dargestellt. Das Beispiel hat eine Auflösung von acht. Beginnend bei der Box-Basis auf der untersten Stufe der Hierarchie ergibt sich der Wert des Koeffizienten

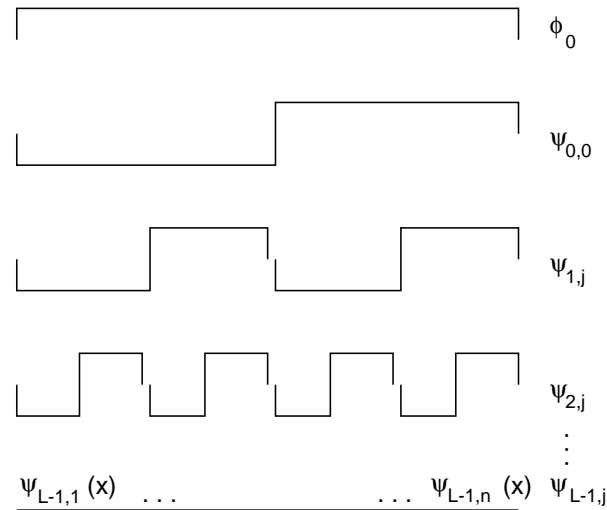


Abbildung 4.3: Die Haar-Wavelet-Basis. Sie besteht aus einer  $\phi_0$ -Funktion auf der obersten Stufe, sowie einer Pyramide von  $\phi_{i,j}$ -Detailfunktionen.  $L$  bezeichnet die Tiefe der Hierarchie (nach [6]).

auf der nächsthöheren Stufe jeweils aus dem Durchschnitt eines Paares von Koeffizienten. Bei der Haar-Basis beginnt man ebenfalls auf der unteren Stufe, bildet nun aber nicht nur den Durchschnitt, sondern ebenfalls die Differenz der Koeffizienten von diesem neu berechneten Wert. So liefert die Durchschnittsbildung auf der untersten Stufe der Hierarchie die Koeffizienten 11.5, 10.5, 8.0 und 10.0. Die Differenz der einzelnen Koeffizienten beträgt somit 0.5, -0.5, 0.0 und -1.0 (die Zeile mit der Bezeichnung  $\psi_{2,j}$ ). Der Koeffizient  $\psi_{2,0} = 0.5$  ergibt sich zum Beispiel als die Differenz zwischen dem lokalem Durchschnitt 11.5 und den beiden Segmenten der Originalfunktion 11.0 und 12.0. Besondere Aufmerksamkeit verdient hier das dritte Segment: der zugehörige Koeffizient hat den Wert Null, weil der Verlauf der Funktion flach ist, d.h. er enthält keine weiteren Details.

Die Anwendung der beiden Operatoren wird nun auf die vier Koeffizienten fortgesetzt, die man mit Hilfe der Durchschnittsbildung erhalten hat. Dadurch gewinnt man wiederum zwei Durchschnitts- und zwei Differenzwerte. Setzt man das Verfahren um noch einen Schritt fort, so erreicht man das Ergebnis: Die Haar-Basis besteht somit aus sieben  $\psi$ - und einer  $\phi$ -Funktion. Den Wert der Funktion  $F(x)$ , wie er durch die Basisfunktionen repräsentiert wird, kann man durch Auswertung der Gleichung 4.1 erhalten, da nun alle geforderten Koeffizienten bekannt sind und eingesetzt werden können.

## 4.2 Eigenschaften einer Wavelet-Basis

Es folgt eine kurze Erläuterung der verschiedenen, wichtigen Eigenschaften einer Wavelet-Basis.

### 4.2.1 Vanishing Moments

Das angestrebte Ziel besteht darin, mit Hilfe einer alternativen Basis für einen Funktionenraum eine möglichst dünnbesetzte Repräsentierung der gegebenen Funktion zu finden, d.h. die Anzahl der Koeffizienten mit Wert Null soll groß sein. Im speziellen geht es hierbei darum, eine Repräsentation des Radiosity-Kernels zu finden, die diese Eigenschaft besitzt. Hat man eine solche Repräsentierung gefunden und verfügt man weiterhin über die Möglichkeit vorherzusagen, welche Koeffizienten ungleich Null sein werden, dann ergibt sich eine Vorgehensweise, die es ermöglicht, die Anzahl der teuer zu berechnenden Formfaktoren extrem zu senken.

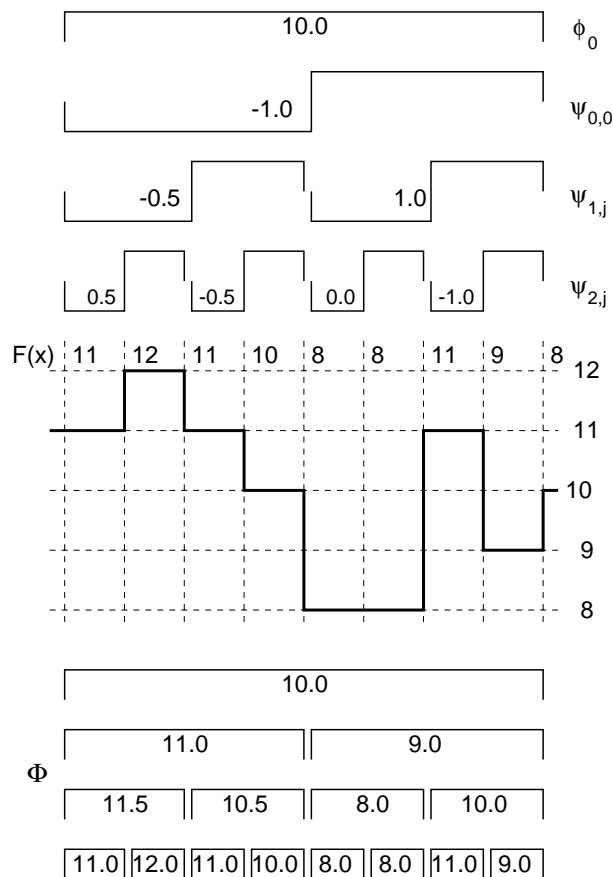


Abbildung 4.4: Eine stückweise konstante Funktion, repräsentiert durch eine hierarchische Haar-Basis (oben) und verschiedene Stufen einer hierarchische Box-Basis (unten). Die Beschriftung der Funktionen ist gleich dem Wert des zugehörigem Koeffizienten (nach [6]).

Eine wichtige Eigenschaft der Wavelets, die eine Aussage über die Güte der Approximation erlaubt, ist das Konzept der *vanishing Moments*.

**Definition:** Vanishing Moments

Eine Funktion  $F(x)$  hat  $M$  vanishing Moments, wenn gilt

$$\int_{-\infty}^{+\infty} F(x)x^i dx = 0 \text{ für alle } i = 0, \dots, M - 1 \quad .$$

Mit anderen Worten: Eine Funktion besitzt einen vanishing Moment, wenn das Integral ihres Produktes mit einer konstanten Funktion verschwindet, d.h. den Wert Null ergibt. Eine Funktion hat zwei vanishing Moments, wenn das gesagte ebenfalls auf das Integral des Produktes mit einer linearen Funktion zutrifft. Dies setzt sich entsprechend für quadratische und kubische Funktionen sowie Polynome noch höherer Ordnung fort.

Nach dieser Definition verfügt die Detailfunktion  $\Psi$  des Haar-Wavelets über einen vanishing Moment. Zieht man wieder das Beispiel in Abbildung 4.4 heran, so sieht man hier, daß der Koeffizient  $b_{\psi_{2,2}}$  den Wert Null hat, weil der entsprechende Teil der Funktion über zwei Intervalle konstant bleibt.

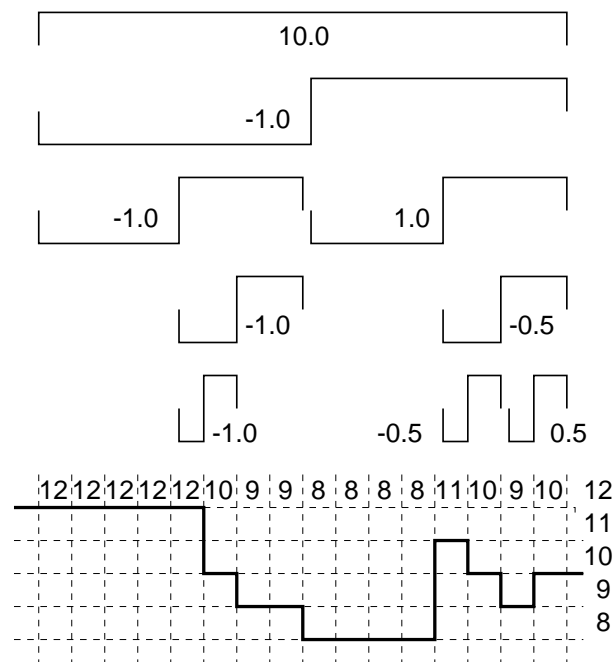


Abbildung 4.5: Stückweise konstante Funktion, repräsentiert durch 9 hierarchische Haar-Basisfunktionen (oben) sowie 16 Box-Funktionen (unten). Die Beschriftung der Funktionen gibt den Wert des jeweiligen Koeffizienten an (nach [6]).

## 4.2.2 Dünnbesetzte Repräsentierung

Wie nun die vanishing Moments der Basisfunktionen zu einer dünnbesetzten Repräsentierung der Funktion und des hier interessanten Integrals führen, läßt sich am besten an einem Beispiel verdeutlichen:

Hierzu betrachte man die in Abbildung 4.5 dargestellte Funktion: Man kann diese als gewichtete Summe von 16 Box-Basen darstellen. Die Werte der Gewichte ergeben sich aus den zugehörigen Werten der Funktion an entsprechender Stelle. Alternativ kann diese Funktion aber auch als gewichtete Summe von nur acht Detailfunktionen sowie einem Durchschnittswert dargestellt werden. Der Durchschnitt wird über das gesamte Intervall gebildet und beträgt in diesem Beispiel 10. Man kann gut erkennen, daß für die flachen Bereiche der Funktion weniger Basisfunktionen benötigt werden als für die Bereiche, welche Details enthalten. Die zugehörige Projektion auf die niedrigere Detailstufe in der Hierarchie verschwindet, weil die Funktion an dieser Stelle konstant ist.

Projiziert man auf Basen mit zwei vanishing Moments, so ergibt sich für Koeffizienten der Wert Null, wenn der Verlauf der Funktion über den von der Basis unterstützten Bereich linear ist, für drei vanishing Moments entsprechend quadratisch usw. Dabei sind Verläufe geringeren Grades mit eingeschlossen.

Im allgemeinen wird es nicht viele Fälle geben, in denen eine Funktion über den unterstützten Bereich einer Basisfunktionen exakt konstant, linear, oder quadratisch verläuft. Es gibt jedoch Bereiche, in denen Funktionen dies „fast“ tun, d.h. die errechneten Gewichte werden aus sehr kleinen, nahe bei Null liegenden Werten bestehen. Setzt man Werte, die eine vorgegebenen Grenze unterschreiten, auf Null, so erzeugt man einen nur geringen Fehler.

### 4.2.3 Beziehung zwischen verschiedenen Stufen der Hierarchie (two-scale Relationship)

Die gezeigte Haar-Basis ist nur das einfachste Beispiel einer unendlich großen Familie von ähnlichen Konstruktionen. Das aufgezeigte Grundprinzip bleibt jedoch bei anderen Wavelets bestehen:

Der Ausgangspunkt jeder Konstruktion sind auch weiterhin die folgenden beiden Funktionen:

1. die Glättfunktion (*smooth/scaling function*)  $\Phi(s)$  und
2. die Detailfunktion (*detail function*)  $\Psi(s)$ .

Im vorhergehenden Beispiel entsprachen diese Funktionen dem Bilden des Durchschnitts und der Differenz. Beide sind dabei auf dem Einheitsintervall definiert. Um mit ihrer Hilfe eine Basis bilden zu können, müssen diese Ausgangsfunktionen nun skaliert (vergrößert und verkleinert) und/oder verschoben werden. Dies ergibt eine Menge neuer Funktionen  $\phi$  und  $\psi$ . Dabei wird die Skalierung und Verschiebung durch Indizes am Fuße der Funktion ausgedrückt, wobei der erste Index die Skalierung und der zweite die Verschiebung angibt. Die Skalierung kann man sich hierbei auch als Stufe in der Hierarchie vorstellen, denn es wurde bereits darauf hingewiesen, daß Basisfunktionen einer bestimmten Stufe (bezeichnet durch den ersten Index) immer einen kleineren Teil des Intervalls abdecken als die der nächsthöheren Stufe.

Skalierungen  $i$  und Verschiebungen  $j$  von  $\phi(s)$  und  $\psi(s)$  werden auf folgende Weise definiert:

$$\begin{aligned}\phi_{i,j}(s) &= 2^{i/2} \phi(2^i s - j) \\ \psi_{i,j}(s) &= 2^{i/2} \psi(2^i s - j),\end{aligned}$$

wobei  $j = 0, \dots, 2^i - 1$ .

Die Funktion  $\phi_{i,j}$  entspricht also  $\phi_{i-1,j}$ , mit dem Unterschied, daß  $\phi_{i-1,j}$  doppelt so breit und  $1/\sqrt{2}$ -mal so hoch ist. Die breiteren Funktionen sind hierbei kleiner, damit unabhängig von  $i$  der Wert  $\langle \phi_{i,j}, \phi_{i,k} \rangle$  konstant bleibt. Funktion  $\phi_{i,j}$  ist genau wie  $\phi_{i,j+1}$ , nur daß diese verschoben ist.

Um nun die Basis eines Funktionenraumes mit Dimension  $n = 2^L$  aufzubauen, konstruiert man eine Funktionenhierarchie mit  $L$  Stufen. Jede dieser Funktionen ist eine skalierte und verschobene Version von  $\Phi$  und  $\Psi$ . Dies entspricht dem Beispiel in Abbildung 4.2 für  $L = 3$ . Die eigentliche Wavelet-Basis bilden nun die Detailfunktionen auf allen Stufen und die Glättfunktion der obersten Stufe, also:  $\psi_{i,j}$ ,  $i = 0, \dots, L-1$  und  $\phi_0$ .

Zwei Stufen einer Hierarchie sind somit nicht unabhängig voneinander; es besteht eine Beziehung zwischen ihnen, die sogenannte *two-scale relationship*:

$$\begin{aligned}\phi_{i-1,j} &= \sum_k h_{k-2j} \phi_{i,k} \\ \psi_{i-1,j} &= \sum_k g_{k-2j} \phi_{i,k} \quad .\end{aligned}$$

Hierbei bezeichnet  $h$  den paarweisen Durchschnitt und  $g$  die paarweise Differenz. Mit anderen Worten: Mit Hilfe einer Linearkombination der Funktion  $\phi$  auf einer gegebenen Stufe können die Funktion  $\phi$  und  $\psi$  auf der nächsthöheren Stufe berechnet werden.

Dies ist eine wichtige Eigenschaft der Wavelet-Basen, die es erlaubt eine Push/Pull Funktionalität zu realisieren, die notwendig ist, um Energie konsistent innerhalb einer Hierarchie von Patches zu verteilen (siehe Abschnitt 5.1).

Wie auch im Fall der Haar-Basis (siehe hierzu Abschnitt 4.1) kann nun eine beliebige Funktion  $B(s)$  formal in Wavelet-Basis-Schreibweise ausgedrückt werden als <sup>1</sup>

$$\hat{B}(s) = \langle B, \phi_0 \rangle \phi_0(s) + \sum_{ij} \langle B, \psi_{i,j} \rangle \psi_{i,j}(s).$$

Die Koeffizienten werden hierbei als innere Produkte ausgeschrieben. Durch die Ausnutzung der two-scale relationship ist es nun möglich, die Berechnung vieler innerer Produkte zu vermeiden. Sind die Koeffizienten auf der untersten Stufe der Hierarchie gegeben, so kann man alle Koeffizienten der höheren Stufen mit Hilfe eines Pyramidenalgorithmus berechnen. Dabei wird in jedem Schritt für alle Koeffizienten der Durchschnitt und die Differenz gebildet und das Ergebnis eine Stufe weiter nach oben gereicht. Für eine genauere Ausführung über diesen Pyramidenalgorithmus, zusammen mit Pseudocode, siehe [10].

All diese beschriebenen Eigenschaften werden später bei der Beschreibung des eigentlichen Algorithmus noch einmal aufgegriffen werden (siehe folgendes Kapitel 5).

### 4.3 Wavelets in höheren Dimensionen

Wie bereits an früherer Stelle erwähnt, kann man sich für eine Radiosity-Berechnung leider nicht auf Wavelets mit nur einer Variablen beschränken. Für ein 3D-Radiosity-Verfahren muß es möglich sein, einzelne Punkte auf einem Patch zu bezeichnen. Ein solcher Punkt kann durch zwei Parameter beschrieben werden, woraus folgt, daß es sich beim Radiosity-Kernel für den dreidimensionalen Fall um eine von vier Parametern abhängige Funktion handelt, da Paare von Patches betrachtet werden. Das Ziel ist es, diesen Kernel, eine vierdimensionale Funktion, auf eine Basismenge zu projizieren, um eine möglichst dünnbesetzte Repräsentation zu erhalten.

Um dieses Ziel zu erreichen beschäftigt man sich zunächst mit der Erweiterung des Kernels auf zwei Variablen. Dies entspricht 2D-Radiosity (siehe hierzu [14]). Es gibt verschiedene Möglichkeiten eine 2D-Wavelet-Basis ausgehend von einem eindimensionalen Wavelet zu konstruieren. An dieser Stelle sei nur der sogenannte non-standard Ansatz kurz vorgestellt (*non-standard pyramid algorithm* [10], [6] und [20]).

$s$  und  $t$  seinen jeweils Elemente eines endlichen Intervalls. Eine beliebige Funktion  $k(s, t)$  dieser Variablen kann durch eine Funktion  $\hat{k}(s, t)$  approximiert werden, die in einem endlichen, zweidimensionalen Funktionenraum liegt. Ist ein eindimensionales Wavelet gegeben, so enthält die 2D-Wavelet-Basis die Funktionen:

$$\begin{aligned} & \phi_0(s)\phi_0(t), \\ & \psi_{i,j}(s)\psi_{i,k}(t), \\ & \psi_{i,j}(s)\phi_{i,k}(t), \\ & \phi_{i,j}(s)\psi_{i,k}(t), \end{aligned}$$

mit  $i = 0, \dots, L-1$  und  $j, k = 0, \dots, 2^i - 1$ . Die Stufe der Funktionen in der Hierarchie (der erste Index der Funktion) ist hierbei jeweils  $i$ , d.h. es werden nur Funktionen zusammen betrachtet, die sich auf der gleichen Stufe innerhalb der Hierarchie befinden. Dieses Vorgehen unterscheidet den non-standard vom standard Ansatz, welcher ebenfalls in den genannten Quellen erläutert wird.

<sup>1</sup>Es wurde angenommen, daß es sich bei der Wavelet-Basis um eine orthonormale Basis handelt. Dies wurde getan, um die Darstellung zu vereinfachen. Der nicht orthonormale Fall wird in [20] behandelt und diskutiert.

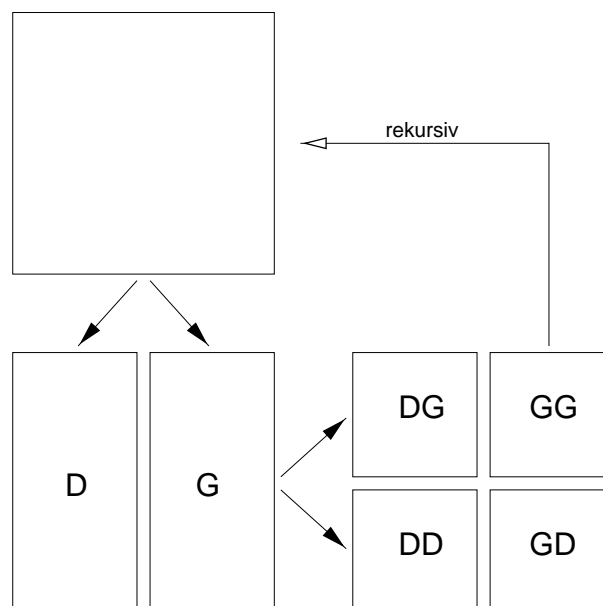


Abbildung 4.6: Abstrakte Darstellung des 2D-Pyramidenalgorithmus. Der Berechnungsschritt wird abwechselnd auf die Spalten bzw. Zeilen der Formfaktormatrix angewendet („D“ bezeichnet die Anwendung der Detailfunktion, „G“ die der Glättfunktion). Anschließend wird das Verfahren rekursiv auf dem GG-Quadranten fortgesetzt (nach [10]).

Die Koeffizienten des 2D-Wavelets können nun durch einen 2D-Pyramidenalgorithmus bestimmt werden. Abbildung 4.6 zeigt dessen Vorgehen:

- Ausgangspunkt ist eine vollständige  $n \times n$  Matrix (oben links).
- Betrachte die *Reihen* der Matrix: Wende sowohl einen Glätte- wie auch einen Detailschritt auf jeder der  $n$  Reihen an. Es ergeben sich  $n/2$  Glätte- und  $n/2$  Detailspalten, die entsprechend angeordnet werden (unten links).
- Betrachte nun die *Spalten* der Matrix: Wende wiederum einen Glätte- wie auch einen Detailschritt auf jeder der  $n$  Spalten an. Es ergeben sich  $n/4$  Detail-Detail Koeffizienten,  $n/4$  Detail-Glätte Koeffizienten,  $n/4$  Glätte-Detail Koeffizienten und  $n/4$  Glätte-Glätte Koeffizienten (unten rechts).
- Setze das Verfahren rekursiv auf dem Glätte-Glätte Viertel der Matrix fort (angedeutet durch den Pfeil).

Dies beschreibt einen 2D-PyramidUp Algorithmus. Ein 2D-PyramidDown Algorithmus (also eine Bewegung von der Wurzel der Hierarchie zu ihren Blättern) kann man analog aus einem 1D-Pyramid-Down Algorithmus herleiten (siehe [10] und [20]).

Diese Konstruktion kann nun auch auf Funktionen  $k(s_1, t_1, s_2, t_2)$  von vier Variablen angewendet werden. Dies entspricht dem Kernel für 3D-Radiosity.

Eine Erweiterung auf diesem Fall ergibt 16 Kombinationen der  $\phi$  und  $\psi$  Funktionen von 4 Variablen. Die Basis besteht dann aus den 15 Kombinationen auf gleicher Stufe  $i$ , die eine  $\psi$ -Funktion enthalten. Die zugehörige Pyramidentransformation wird analog dem zweidimensionalen Fall gebildet, mit dem Unterschied, daß der oben beschriebene Durchschnitts- und Differenzbildungsschritt der Reihe nach auf alle vier beteiligten Dimensionen angewendet wird.

Für diese Art von multidimensionaler Wavelet-Basis haben Beylkin et al. [5] gezeigt, daß bei einer gegebenen Fehlertoleranz nur  $O(n)$  Koeffizienten nötig sind, um ein Ergebnis zu erhalten, das einen Fehler unterhalb dieser Toleranz erreicht.

# Kapitel 5

## Wavelet Radiosity

Es folgt eine Beschreibung des originalen Wavelet-Algorithmus von Gortler et al. [10].

Ein guter Einstieg für das Verständnis dieses Algorithmus stellen Vorkenntnisse über den HR-Algorithmus dar. Hat man die grundlegende Vorgehensweise dieses Ansatz verstanden, so wendet man sich wieder dem WR-Algorithmus zu und versucht dann dort, die im Vergleich zum HR-Algorithmus vorgenommene Abstraktion nachzuvollziehen. Wenn man sich alle Vorgänge anhand der Haar-Basis veranschaulicht, ist ein relativ einfacher Einstieg möglich. Als nächstschwierigere Stufe sind dann Flatlets zu behandeln, gefolgt von Multiwavelets.

Eine leicht verständliche Erklärung des HR-Algorithmus, unter Benutzung von Pseudocode und Beschreibung der umgesetzten Mechanismen, ist bei Glassner [8] zu finden.

### 5.1 Grundsätzliches Vorgehen

Der Pseudocode für den Wavelet Radiosity-Algorithmus sieht wie folgt aus:

```
baue Hierarchien auf: Erzeuge einen Quad-Tree pro Ausgangspolygon
```

```
Für alle Paare von Patches(i, j)
```

```
  Wenn i von j aus sichtbar ist
```

```
    verbinde i und j durch einen Link
```

```
// Iteriere, abwechselndes Lösen und Verfeinern  
solange (wahr)
```

```
  // Ein Lösungsschritt
```

```
  Wiederhole bis Equilibrium erreicht ist
```

```
    Sammlle Radiosity für alle Patches über die Links
```

```
    Führe für jede Hierarchie ein Push/Pull durch und  
      ermittle den Fehler
```

```
  // Verfeinerungsschritt
```

```
  Untersuche alle Links und verfeinere sie, wenn nötig
```

```
  // Abbruchkriterium
```

```
  Wenn keine Verfeinerung stattgefunden hat und  
  der Fehler unter der gewünschten Grenze liegt
```

```
    Beende das Verfahren
```

Die Ähnlichkeit zum hierarchischen Radiosity-Verfahren ist nicht von der Hand zu weisen; eine so allgemeine Beschreibung des HR-Algorithmus würde identisch aussehen. Die Unterschiede sind hier im Detail verborgen.

Erzeugt man einen neuen Link zwischen zwei Patches, so wird eine Bestimmung des Interaktionsfaktors notwendig. Dies geschieht, indem ein projizierter Radiosity-Kernel für das jeweilige Paar berechnet wird. Dieser Schritt fällt während des Aufbaus der Hierarchie wie auch bei Verfeinerung der Patches an.

Der Begriff Interaktionsfaktor bezeichnet hierbei eine allgemeinere Form des Formfaktors. An dieser Stelle wurde ein neuer Begriff gewählt, weil die Eigenschaften und Merkmale eines Formfaktors nicht ohne weiteres auf einen Interaktionsfaktor zu übertragen sind. Es ist daher anzunehmen, daß mit dieser neuen Begriffsbildung zukünftigen Mißverständnissen vorgebeugt wird. Die Unterschiede zwischen diesen beiden Faktoren werden an späterer Stelle noch behandelt.

Der innerhalb der Hauptschleife ausgeführte Lösungsschritt gliedert sich in die drei Teile *Gather*, *Pull* und *Push*. Diese einzelnen Schritte werden innerhalb einer weiteren äußeren Schleife ausgeführt bis ein Equilibrium erreicht ist. Die Aufgaben, die jedem Schritt zukommen, stellen sich wie folgt dar:

### 1. **Gather**

Der Gather-Schritt ist dafür zuständig, für jedes Patch die Energie, die von anderen Patches in der Szene ausgeht, einzusammeln. Jedes Patch besitzt eine bestimmte Anzahl von Koeffizienten, die die Koeffizienten der projizierten Radiosity-Funktion für dieses Patch darstellen. Der projizierte Kernel läßt sich als Matrix darstellen, so daß der Vorgang des Sammelns (*gathering*) durch eine Matrizenmultiplikation abgebildet werden kann: Die Koeffizienten des Patches, welches die Energie aussendet (*shooting patch*) wird mit der berechneten Kernel-Matrix multipliziert. Das Ergebnis liefert die (projizierten) Koeffizienten des bestrahlten Patches (*gathering patch*).

### 2. **Push**

Diesem Schritt kommt die Aufgabe zu, die Abwärtsbewegung in der Hierarchie durchzuführen, so daß nach der folgenden Aufwärtsbewegung für den nächsten Schritt der Iteration wieder konsistente Werte auf allen Stufen der Hierarchie zur Verfügung stehen. Man beginnt an der Wurzel der Hierarchie und übergibt jeweils einen anteiligen Radiosity-Wert an seine Kinder. Diese addieren die erhaltene Energie zu ihrer eigenen hinzu. Somit ergeben sich die Energiewerte, die durch die Kinder an die Szene abgegeben werden. Dieses Verfahren wird rekursiv fortgesetzt, bis die unterste Stufe erreicht ist.

### 3. **Pull**

Pull beginnt mit den Koeffizienten der Basisfunktionen auf der untersten Ebene der Hierarchie und berechnet von dort ausgehend mit Hilfe der two-scale Relationship alle Koeffizienten der oberen Stufen bis die Wurzel erreicht ist.

Dem Lösungsschritt folgt der Verfeinerungsschritt: Hierbei werden alle Links durchlaufen und geprüft, ob der Fehler, den der aktuell betrachtete Link erzeugt, klein genug ist. Sollte dies nicht der Fall sein, wird eines der beteiligten Patches unterteilt. Der Link, der einen zu großen Fehler aufwies, wird gelöscht und es entstehen neue Links zu den erzeugten Sub-Patches. Eine Berechnung der Interaktionsfaktoren für diese neuen Links wird initiiert.

Wenn keine Unterteilung vorgenommen wurde und der berechnete Fehler unterhalb der gewünschten Fehlergrenze liegt, ist der Algorithmus beendet. Ansonsten werden die oben genannten Schritte wiederholt bis beide Abbruchkriterien erfüllt sind.

### 5.1.1 Die Reihenfolge der einzelnen Schritte

Die originale Wavelet Radiosity Veröffentlichung [10] gibt in der Beschreibung des Algorithmus eine andere Reihenfolge der Gather-, Push- und Pull-Schritte an. Dort werden sie in der Reihenfolge Pull, Gather, Push durchgeführt. Dieses Vorgehen entspricht einer Gauss-Seidel Iteration. Die hier beschriebene Reihenfolge der Schritte entspricht jedoch einer Jacobi-Iteration. Der Nachteil dieser gegenüber einer Gauss-Seidel Iteration ist bekannt. Trotzdem wurde diese Alternative bewußt gewählt, da in [27] bemerkt wurde, daß die ursprüngliche Reihenfolge zu Stabilitätsproblemen in den ersten Iterationsschritten des Verfahrens führen kann, wenn die Anzahl der beteiligten Elemente noch gering ist.

## 5.2 Laufzeitbetrachtung

Sei  $n$  die Anzahl der Elemente in der Szene. Der Aufbau der Hierarchie benötigt auch hier  $O(n^2)$  Zeit. Es gilt aber weiterhin, daß  $n$  im Anfangsstadium sehr klein ist, da keine a priori Unterteilung notwendig ist.

Die Push und Pull Schritte können in Zeit  $O(n)$  durchgeführt werden.

Der Gather-Schritt benötigt  $O(m)$  Zeit. Hierbei bezeichnet  $m$  die Anzahl der Elemente des Kernels, die zu berücksichtigen sind, d.h. deren Werte über der gegebenen Fehlertoleranz liegen. Es wäre wünschenswert, wenn dieses  $m$  so klein wie möglich wäre.

Wavelet-Basen führen zu einem  $m = O(n)$ , wobei der konstante Faktor in  $O(n)$  kleiner wird, je mehr vanishing Moments die gewählte Basis vorweisen kann. Beim HR-Algorithmus wurde bereits dieses Laufzeitverhalten festgestellt; es wurde jedoch eine geometrisch anschauliche Begründung für diese These gegeben. Durch den nun zur Verfügung stehenden mathematischen Rahmen ist es an dieser Stelle möglich das geometrische Argument durch ein mathematisch-formales zu ersetzen.

Weitere Ausführungen und Herleitungen zum Laufzeitverhalten finden sich in [10].

## 5.3 Die Kernel Projektion

Ein wesentlicher Schritt wurde in der oben gemachten Beschreibung der allgemeinen Vorgehensweise nicht ausreichend genau besprochen, obwohl ihm eine große Bedeutung zukommt: Die Projektion des Kernels.

Für diese gibt es zwei unterschiedliche Ansätze:

### 5.3.1 Bottom-Up Berechnung

Um den auf die Wavelet-Basis projizierten Kernel zu bestimmen, kann man die folgende Vorgehensweise wählen:

1. Berechne alle Elemente des Kernels mit Hilfe einer Quadraturmethode.
2. Bestimme die Werte, die jede Stufe an die nächsthöhere Stufe weitergibt durch Ausführung eines Schrittes des Pyramidenalgorithmus (`PyramidUp`).
3. Setze alle Werte, die kleiner als die Fehlertoleranz sind, auf Null.

### 5.3.2 Top-Down Berechnung

Die Bottom-Up Berechnung des Kernels benötigt quadratische Zeit und Speicherplatz und ist somit zu teuer. Wenn man bereits vor der Berechnung der einzelnen Kernel-Elemente feststellen könnte, ob das Ergebnis kleiner oder größer als die Fehlertoleranz sein wird, so wäre es möglich, an dieser Stelle signifikante Einsparungen zu machen. Um dies zu erreichen wird mit Hilfe eines Orakels eine Top-Down Berechnung durchgeführt.

Sei ein Orakel gegeben, das nach der Glätte des Kernels einer bestimmten Region befragt werden kann. Für Basen mit einem vanishing Moment würde dies bedeuten, daß das Orakel vorhersagen kann, ob der Verlauf der Radiosity-Funktion in der gegebenen Region konstant sein wird. Für Basen mit zwei vanishing Moments beurteilt das Orakel, ob ein linearer Verlauf der Funktion vorliegt, usw.

Man kann mit Hilfe dieses Orakels den projizierten Kernel für zwei gegebene Patches  $p$  und  $q$  auf folgende Art und Weise berechnen:

1. Befrage das Orakel, ob der Kernel, der für  $p$  und  $q$  berechnet würde, bereits glatt genug ist. Wenn die Antwort Ja ist, gibt es nichts mehr zu tun.
2. Sonst: Berechne mit Hilfe einer Quadraturmethode den Wert des Kernels. Rufe dann, wenn noch nicht die unterste Stufe der Hierarchie erreicht ist, rekursiv diese Funktion für die nächstniedrigere Stufe auf. Dies beinhaltet bei 2D-Radiosity vier neue Aufrufe, für alle Kombinationen von rechten und linken Kindern. Für Radiosity in 3D bedeutet dies 16 neue Aufrufe, weil jedes Patch vier Kinder hat und für jede Kombination dieser Kinder mit den Kindern des anderen Patches ein Eintrag berechnet werden muß.

Wenn das Orakel verkündet, daß die betrachtete Region bereits glatt genug ist, müssen nicht noch weitere rekursive Aufrufe durchgeführt werden. Denn laut Annahme würde der so berechnete Kernelwert unterhalb der gegebenen Fehlertoleranz liegen. In Abschnitt 5.4.2 wird erklärt, wie das Orakel diese Entscheidung trifft.

## 5.4 Die Implementierung des Algorithmus

Die Implementierung folgt im wesentlichen der von [27]. Es folgen einige Erläuterungen und Hinweise auf Abweichungen.

### 5.4.1 Implementierte Wavelet-Basen

Es wurden die folgenden Basen implementiert: Haar, F2 und F3 Flatlets (Flat2, Flat3), sowie M2 und M3 Multiwavelets (Mult2, Mult3).

#### 5.4.1.1 Haar-Basis

Die Haar-Basis besteht nur aus  $\{\Phi_0\}$  (siehe Abbildung 5.1) und entspricht dem HR-Ansatz [12]. Jedes Patch hat nur einen Koeffizienten, jeder Link enthält einen Interaktionsfaktor, der aus einer einzigen Gleitkommazahl besteht. Wie bereits an früherer Stelle erwähnt, bezeichnen wir mit Interaktionsfaktoren eine allgemeinere Form von Formfaktoren. Dabei kann es sich, wie hier, um eine einzige Zahl, aber auch um eine Matrix handeln. Dies ist abhängig von der Anzahl der vanishing Moments, die eine Basis aufzuweisen hat. Bei der Haar-Basis entspricht ein Interaktionsfaktor genau einem Formfaktor.

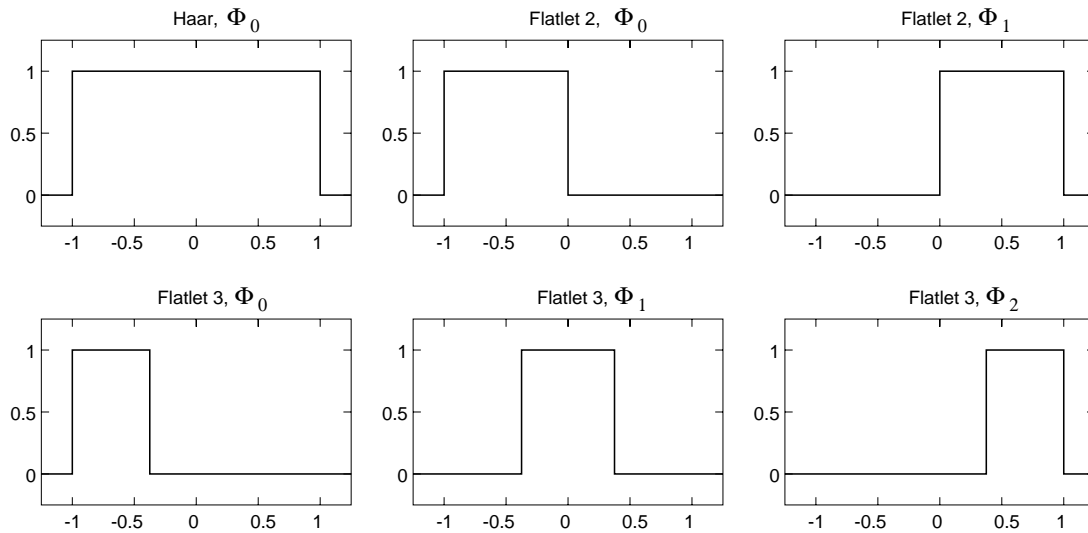


Abbildung 5.1: Haar- und Flatlet-Basisfunktionen in 2D. Die Haar-Basis besteht allein aus  $\{\Phi_0\}$ , die Flatlet2-Basis besteht aus  $\{\Phi_0\}$  und  $\{\Phi_1\}$ . Für drei vanishing Moments enthält die Flatlet-Basis die Funktionen  $\{\Phi_0\}$ ,  $\{\Phi_1\}$  und  $\{\Phi_2\}$ . Es läßt sich gut erkennen, daß die Funktionen einer Flatlet-Basis disjunkt sind (nach [27]).

#### 5.4.1.2 Flatlets

Die Flat2-Basis besteht aus  $\{\Phi_0, \Phi_1\}$ , eine Flat3-Basis aus  $\{\Phi_0, \Phi_1, \Phi_2\}$  (siehe Abbildung 5.1).

Ausgehend vom HR-Ansatz kann man sich Flatlets als ein Patch mit einer Anzahl von Sub-Patches vorstellen. Ein Flat2-Patch läßt sich als eine Oberfläche mit einer internen  $2 \times 2$  Unterteilung darstellen, ein Flat3-Patch hätte eine  $3 \times 3$  Unterteilung. Ein Flat2 Patch enthält somit zwei Koeffizienten in 2D und vier in 3D. Eine Interaktion in 3D zwischen zwei Patches muß also  $4 \cdot 4 = 16$  Interaktionen zwischen den Koeffizienten beachten und berechnen. Der zugehörige Interaktionsfaktor besteht somit aus einer  $4 \times 4$  Matrix. Für die Flat3-Basis ergeben sich 9 Koeffizienten und ein Interaktionsfaktor, der aus einer  $9 \times 9$  Matrix besteht.

Diese Unterteilung des Patches ergibt sich daraus, daß die Basisfunktionen, wie bei der Haar-Basis, konstant, aber nicht über das gesamte Patch von Null verschieden sind. Dies trifft nur auf einen Teilbereich zu. Weiterhin sind sie disjunkt. Die einzelnen Elemente der Matrix stellen somit normale Formfaktoren dar; bereits bestehende Formfaktorroutrinen können für ihre Berechnung eingesetzt werden. Diese müssen so modifiziert werden, daß sie nicht mehr den Formfaktor zwischen gesamten Patches, sondern zwischen Sub-Patches berechnen. Es verändert sich also nur die Größe und Lage der beteiligten Patches.

Beim Link zeigt es sich, daß es sinnvoll ist, Formfaktoren begrifflich von Interaktionsfaktoren zu unterscheiden. Der Link hat einen Interaktionsfaktor, er besteht aus einer Matrix von Formfaktoren.

Für die Darstellung der Patches ist es nötig festzustellen, welche Farbe ein beliebiger Punkt auf einem Patch besitzt. Es gilt herauszufinden, auf welchem (impliziten) Sub-Patch der Punkt liegt. Dies ist aus der relativen Lage des Punktes zu den Eckpunkten bekannt. Mit diesem Wissen kann nun der zugehörige Koeffizient des Patches ermittelt werden, welcher dann bereits den gesuchten Farbwert darstellt. Dieses Vorgehen ist möglich, weil die Bereiche, die von den einzelnen Basisfunktionen unterstützt werden, disjunkt sind.

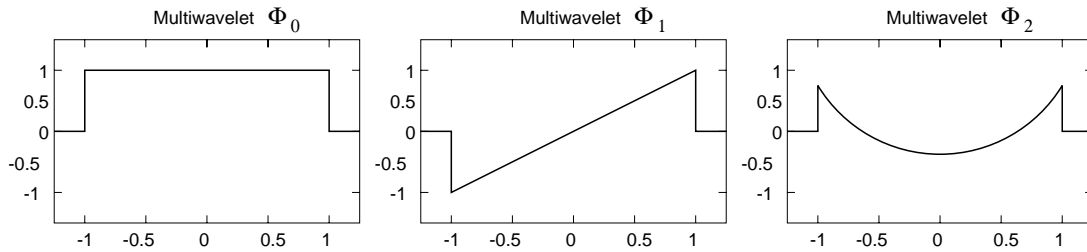


Abbildung 5.2: Multiwavelet-Basisfunktionen in 2D. Die Multiwavelet-Basis mit zwei vanishing Moments besteht aus den Funktionen  $\{\Phi_0\}$  und  $\{\Phi_1\}$ . Für drei vanishing Moments besteht die Basis aus  $\{\Phi_0\}$ ,  $\{\Phi_1\}$  und  $\{\Phi_2\}$  (nach [27]).

### 5.4.1.3 Multiwavelets

Die Basisfunktionen einer Multiwavelet-Basis sind, anders als bei den Flatlets, nicht disjunkt, sondern kumulativ (siehe Abbildung 5.2): Die Mult2-Basis besteht, analog zu den Flatlets, aus  $\{\Phi_0, \Phi_1\}$ , Mult3 aus  $\{\Phi_0, \Phi_1, \Phi_2\}$ , jedoch kann man sehen, daß sich die einzelnen  $\Phi$ -Funktionen über einen gemeinsamen Bereich erstrecken. Die Anzahl der Koeffizienten und die Art der Interaktionsfaktoren stimmt wieder mit den Flatlets überein, da diese Eigenschaften allein von der Anzahl der vanishing Moments abhängen und nicht von der vollständigen Basis. Ein Mult2-Patch besitzt also die gleiche Anzahl von Koeffizienten und die gleiche Art von Interaktionsfaktor wie ein Flat2-Patch. Analoges gilt für Mult3- und Flat3-Patches.

Die gleiche Art bedeutet aber nicht, daß die Koeffizienten und Matrixelemente mit den gleichen oder nur ähnlichen Werten wie bei den Flatlets gefüllt werden. Bei den Multiwavelets bestehen die Elemente der Interaktionsfaktormatrix nicht mehr aus einfachen Formfaktoren, wie auch die Koeffizienten nicht mehr einen konstanten Radiosity-Wert auf einem Sub-Patch angeben. Die Basisfunktionen sind nun, wie bereits gesagt, kumulativ, d.h. sie müssen miteinander additiv verknüpft werden, um den Radiosity-Wert an einer bestimmten Stelle auf dem Patch zu liefern.

Betrachten wir auch hier die Aufgabe, zu einem gegebenen Punkt des Patches den zugehörigen Farbwert  $c$  zu ermitteln: Aus den Multiwavelet-Basisfunktionen in Abbildung 5.2 läßt sich erkennen, daß es eine Funktion gibt, die für den konstanten Anteil zuständig ist ( $\Phi_0$ ). Genauso gibt es eine zuständige Funktion für den linearen ( $\Phi_1$ ) wie auch für den quadratischen Anteil ( $\Phi_2$ ).

An dieser Stelle sei noch einmal darauf hingewiesen, daß es sich bei den in der Abbildung gezeigten Funktionen um Basisfunktionen für 2D-Radiosity handelt. Um nun die 3D-Funktionen zu erhalten, werden zwei eindimensionale Funktionen, die den Verlauf in jeweils eine Richtung des Patches angeben, mit Hilfe eines Tensorproduktes verknüpft. Diese eindimensionalen Funktionen haben die Form  $F(s) = 1 + s$ . Die Funktion über den gesamten Patch sieht dann so aus:

$$F(x) \cdot F(y) = (1 + x)(1 + y) = 1 + x + y + xy$$

Am Beispiel der Mult2-Wavelets soll nun gezeigt werden, wie der Farbwert eines Punktes  $p$  zu berechnen ist.  $p$  sei bezüglich eines festen Punktes auf dem Patch durch die beiden Koordinaten  $x$  und  $y$  parametrisiert. Der gesuchte Wert  $c$  ergibt sich dann aus

$$c(x, y) = b_0 + x b_1 + y b_2 + xy b_3$$

Zur Erläuterung: Wir beziehen uns auf Multiwavelets mit zwei vanishing Moments, also haben wir es mit drei unterschiedlichen Anteilen zu tun: konstanter Anteil, linear in  $x$ -Richtung, linear in  $y$ -Richtung und bilinear in  $xy$ -Richtung.

Angenommen, der Ursprung des Patches liegt genau in der Mitte und es gilt  $x, y \in [-1, 1]$ . Für die Farben der Eckpunkte eines Mult2-Patches ergibt sich:

$$\begin{aligned}
 c_0 &= b_0 + -1 \cdot b_1 + -1 \cdot b_2 + -1 \cdot -1 \cdot b_3 \\
 &= b_0 - b_1 - b_2 + b_3 \\
 c_1 &= b_0 + -1 \cdot b_1 + 1 \cdot b_2 + -1 \cdot 1 \cdot b_3 \\
 &= b_0 - b_1 + b_2 - b_3 \\
 c_2 &= b_0 + 1 \cdot b_1 + 1 \cdot b_2 + 1 \cdot 1 \cdot b_3 \\
 &= b_0 + b_1 + b_2 + b_3 \\
 c_3 &= b_0 + 1 \cdot b_1 + -1 \cdot b_2 + 1 \cdot -1 \cdot b_3 \\
 &= b_0 + b_1 - b_2 - b_3
 \end{aligned}$$

Für Multiwavelets mit drei vanishing Moments geht man analog vor. Hier ergeben sich eine entsprechend größere Anzahl von Kombination, was zu einem höheren Berechnungsaufwand führt.

#### 5.4.1.4 Vor- und Nachteile von Tree Wavelets

Bei allen oben genannten Wavelet-Basen handelt es sich um sogenannte *Tree Wavelets*. Ein Tree Wavelet hat die Eigenschaft, daß sich die Bereiche, die von Basisfunktionen auf benachbarten Patches unterstützt werden, nicht überlappen. Sie sind ganz allein auf ein einziges Patch beschränkt und gehen nicht über dessen Grenzen hinaus. Diese Eigenschaft erlaubt es, alle Berechnungen mit Hilfe eines Baumes durchzuführen, der keine uniforme Tiefe aufweisen muß.

Die Tatsache, daß es sich hier um Tree Wavelets handelt, hat aber auch noch einen weiteren Vorteil: Nimmt man sich einen Knoten der Hierarchie heraus, so läßt sich feststellen, daß die Koeffizienten, die die Veränderung zur nächsttieferen Stufe angeben, auch mit Hilfe der two-scale Relationship und der Glättkoeffizienten der unmittelbaren Kinder berechnet werden können. Somit ist es nicht nötig, die Detailkoeffizienten  $\psi_{i,j}$  explizit zu speichern. Es genügt, daß sie somit implizit gegeben sind. Dies ist nur bei Tree Wavelets möglich, weil sonst nicht sichergestellt werden kann, daß nur die unmittelbaren Kinder an dieser Relation beteiligt sind. Für Wavelets, die diese Eigenschaft nicht besitzen, muß man ebenfalls die Knoten der Hierarchie beachten, deren Wirkungsbereiche auf das aktuelle Patch fallen. Da es sich bei diesen aber nicht nur um die direkten Kinder handeln muß ist man gezwungen, die Hierarchie aufwendig zu traversieren.

Tree Wavelets bieten jedoch nicht nur Vorteile: Durch die Beschränkung der unterstützten Bereiche auf ein einziges Patch ergibt sich zwangsläufig, daß keine der in den Wavelet-Basen enthaltenen Funktionen kontinuierlich über die Grenze des Patches hinaus verläuft. Die resultierenden Radiosity-Funktionen werden im allgemeinen also nicht an den Grenzen eines Patches stetig sein. Dies kann, vor allem bei großen Patches in der Szene, zu Artefakten wie Mach-Band Effekten führen, da das menschliche Auge sehr sensibel auf Unstetigkeiten in Farbverläufen reagiert. Abhilfe könnte hier ein nachträglicher Bearbeitungsschritt bieten, der eine Glättung zwischen den einzelnen Patches durchführt (z.B. Gouraud-Schattierung. Siehe hierzu auch Abschnitt 6.4.2.3).

#### 5.4.2 Das Orakel

Das Orakel fällt die Entscheidung, ob der Kernel, der für zwei verschiedene Patches in der Szene berechnet wurde, ausreichend glatt ist. Es entscheidet also, ob sich der Kernel einem Polynom vom Grad  $M - 1$  weit genug annähert. Wenn dies der Fall sein sollte, so verschwinden alle  $\psi$ -Terme, weil ihre Koeffizienten nahe bei Null sind. Die Frage, was nun „ausreichend nah“ bedeutet und welche Werte nahe bei Null liegen, wird mit Hilfe eines Grenzwertes entschieden. Durch das vom Orakel gefällte Urteil

kann vermieden werden, daß Interaktionen ausgewertet werden, die in der Hierarchie auf tieferen Stufen stattfinden, aber keinen signifikanten Einfluß auf das Ergebnis haben.

Nun stellt sich die Frage, wie das Orakel diese schwerwiegende Entscheidung fällt: Die genaueste Methode, um die Glätte des Kernels zu bestimmen, würde darin bestehen, das Integral aller  $\psi$ -Termen zu berechnen und festzustellen, ob alle diese Werte unterhalb der geforderten Fehlergrenze liegen. Diese genaue Berechnung ist allerdings zu teuer.

Leichter ist es, eine Approximation zu berechnen, von deren Ergebnis man abhängig macht, wie weiter verfahren wird. Hierzu wird der Kernel des Integrals an den Stützstellen einer Gauss-Legendre-Quadratur ausgewertet (*Kernelsampling*). Dann konstruiert man ein Polynom vom Grad  $M - 1$ , um die Werte dieses Polynoms an den gleichen Stützstellen zu bestimmen. Der Fehler wird dann als der  $L_1$ -Fehler  $\int |k_p - k|$  angegeben, wobei es sich bei  $k$  um den Kernelsample und bei  $k_p$  um den zugehörigen Sample des Polynoms handelt.

In [10] wird vorgeschlagen, das genannte Polynom mit Hilfe des Neville-Algorithmus [18] zu konstruieren. Die im Rahmen der Arbeit umgesetzte Implementierung folgt jedoch dem Ansatz von [27], der stattdessen die Samples des Polygons durch eine Matrizenmultiplikation einer vorberechneten  $M \times M$  Matrix mit den Quadratur-Samples des Integrals bestimmt. Dann wird, wie oben, der zugehörige  $L_1$ -Fehler berechnet.

Beiden Ansätzen ist gemein, daß sie das Resultat mit der vorgegeben Fehlerschranke vergleichen und dem Ergebnis entsprechend den Wert `True` (Kernel ist glatt genug) oder `False` (die Patches sind noch zu unterteilen) zurückgeben.

Nun muß nicht für alle Wavelet-Basen ein so großer Aufwand getrieben werden. Von der Implementierung des HR-Algorithmus [12] ist bekannt, daß dort der Wert des Formfaktors zwischen den Patches als ein Maß genommen wird, wie groß der Fehler würde, wenn die beiden Patches durch einen Link verbunden wären. Da die Haar-Basis dem HR-Verfahren entspricht, ist dies hier natürlich ohne weiteres übertragbar. Ähnliches gilt aber auch für die Flatlet-Basen, denn diese können als geordnete Zusammenfassung von Haar-Patches angesehen werden, da ja die Basisfunktionen ebenfalls konstant sind (siehe 5.4.1.2). In diesem Fall kann der Fehler als das Maximum der Fehler aller Sub-Patches bestimmt werden. Auf diese Art und Weise wurde auch bei der vorliegenden Implementierung verfahren.

### 5.4.3 Quadratur

Wavelet-Basen höherer Ordnung benötigen eine Verallgemeinerung des bekannten Formfaktors. Dies wird deshalb nötig, weil die gewählten Basisfunktionen im allgemeinen nicht mehr konstant sind. Für jedes Paar von Koeffizienten auf den beiden betrachteten Patches muß der Kernel integriert werden. Dies geschieht mit Hilfe verschiedener Quadraturmethoden, die eine gewichtete Summe von Kernelsamples bilden, um den Wert des Integrals anzunähern.

Unsere Implementierung folgt im wesentlichen der von [27] einschließlich des Einsatzes von Schröders C-C Transfer Regel [21]. Abweichungen gibt es dabei in der Handhabung der Sichtbarkeit: Bei der Berechnung der Formfaktoren werden um die jeweils zu betrachtenden Punkte mit Hilfe einer Poisson-Verteilung Sample-Punkte ausgelegt. Formfaktor *und* Sichtbarkeit zwischen diesen Punkten werden dann paarweise ermittelt. Die Sichtbarkeit geht somit direkt an der Stelle ein, an der auch der Formfaktor berechnet wird. Es wird nicht wie bei [27] ein Sampling der Sichtbarkeit über den gesamten Patch durchgeführt, um so einen Faktor zu bekommen, mit dem dann der ermittelte Formfaktor gewichtet wird.

Dies gilt nur für die Basen, die mit Standard Formfaktoren auskommen. Für die Multiwavelets ist die Situation etwas anders. Auch hier wird die Sichtbarkeit direkt in das Sampeln des Kernels mitbezogen. Allerdings werden keine zufälligen Samples generiert; ihre Positionen werden durch die Quadraturmethode vorgegeben. Einerseits hat das den Vorteil, daß die Geschwindigkeit zunimmt, weil

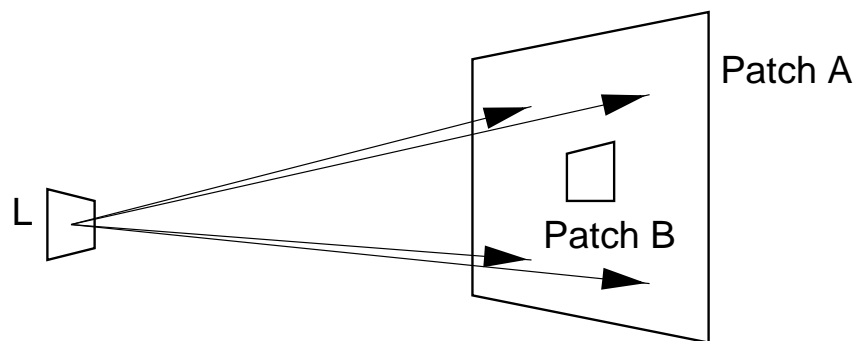


Abbildung 5.3: Beispielszene zur Demonstration der Sichtbarkeitsproblematik. Ein großer Patch (A) wird von einer Lichtquelle (L) bestrahlt. Direkt vor dem großen Patch befindet sich ein kleines, blockierendes Patch (B). Dieses wird von den Sichtbarkeitsstrahlen verfehlt.

weniger Strahlen geschossen werden müssen. Der Nachteil besteht aber darin, daß das Ergebnis nicht so genau ist wie in dem zuvor beschriebenen Ansatz.

Das hat drastische Auswirkungen, wenn man die folgende Optimierung einsetzt: Wenn die Sichtbarkeit entlang eines Links als voll sichtbar ermittelt wurde, so werden bei späteren Unterteilungen eines an diesem Link beteiligten Patches keine weiteren Sichtbarkeitsbetrachtungen angestellt. Das Argument bei diesem Vorgehen ist, daß, wenn ein Patch zu einem anderen Patch voll sichtbar ist, auch die Sub-Patches, die durch eine Unterteilung entstehen, zu diesem Patch voll sichtbar sind. Diese Optimierung wurde bei [12] vorgestellt.

Man betrachte Szene, die einen sehr großen Patch (A), eine Lichtquelle (L) und einen sehr kleinen Patch (B) enthält (siehe Abbildung 5.3). Die Sichtbarkeit zwischen der Lichtquelle und Patch A wird berechnet. Bei einer einfachen Formfaktorberechnung würden hierbei 32 Strahlen zwischen poissonverteilten Punkten auf dem Patch verschossen. Die Wahrscheinlichkeit ist ziemlich hoch, daß zumindest einer dieser Strahlen das blockierende, kleine Patch B trifft. Damit würde sich kein Sichtbarkeitswert von „voll sichtbar“ ergeben und bei einer Unterteilung wird die Berechnung für das Sub-Patch wiederholt.

Betrachtet man aber die gleiche Szene mit zugrundeliegender Mult2-Basis, so wird der Kernel nur an vier Punkten gesampelt, die nach der Gauss-Legendre-Quadraturregel verteilt sind. Alle vier Strahlen treffen das große Patch, es ergibt sich somit ein Sichtbarkeitswert von „voll sichtbar“, für Sub-Patches wird die Berechnung nicht wiederholt. Dies bedeutet, daß sich unter Patch B kein Schatten bildet, da das Verfahren angibt, daß Patch A voll sichtbar zu L ist.

Aus diesem Grund kann es zu großen Fehlern kommen, wenn man diese Optimierung bei Multiwavelets zusammen mit der einfachen Sichtbarkeitsermittlung einsetzt. Abhilfe würde hier eine aufwendigere Ermittlung der Sichtbarkeit für jedes Patch liefern. Dies bedeutet aber, daß für jedes Patch eine konstante Anzahl an Strahlen mehr geschossen werden muß und somit die Geschwindigkeit des Verfahrens abnimmt.

Um dieser Problematik Rechnung zu tragen, wurde eine Option aufgenommen, die es ermöglicht, diese Sichtbarkeitsoptimierung ein- bzw. auszuschalten (siehe `useVisOpt` in Abschnitt 7.1.3.1). Dies hat sich als sinnvoll herausgestellt, da es auch bei anderen Basen sein kann, daß kleine Details selbst bei einer großen Anzahl von geschossenen Strahlen verfehlt werden und es somit zu deformierten oder fehlenden Schatten kommt. Dieser Effekt hängt jedoch von der jeweiligen Szene ab, so daß ein großes Maß an Flexibilität gefordert und mit Hilfe der genannten Option umgesetzt wurde.

### 5.4.4 Gathering: Das Sammeln von Radiosity

Wie bereits an früherer Stelle gesagt, wurde in der Implementierung der Gather-Schritt vom Push/Pull-Schritt getrennt, um Instabilitäten des Verfahrens in der Anfangsphase der Berechnung zu vermeiden (siehe Abschnitt 5.1.1).

Während des Gather-Schrittes werden für jedes Patch alle zugehörigen Links durchlaufen (*gathering Links*). Die Aufgabe besteht darin, mit Hilfe des Quell-Patches die Radiosity des Ziel-Patches zu berechnen. Konzeptuell sind dabei die folgenden Schritte durchzuführen:

1. Berechnung der Radiosity-Funktion auf dem Quell-Patch (mit Hilfe der Koeffizienten).
2. Auswertung der Funktion über dem Interaktionskernel. Man erhält die eintreffende Energie auf dem Ziel-Patch.
3. Projektion dieser Funktion auf die Basis des Ziel-Patches liefert die Koeffizienten der Radiosity-Funktion.

Für alle Wavelet-Basen, deren Basisfunktionen allein aus konstanten und disjunkten Funktionen bestehen (Haar und Flatlets), stellt sich der erste und dritte Schritt als einfach heraus: Die Koeffizienten geben direkt den Wert der Funktion an entsprechender Stelle an. Also ist die Rekonstruktion der Radiosity-Funktion bzw. deren Projektion trivial.

Es bleibt die Auswertung des Kernels. Dabei wäre es wünschenswert, wenn der eigentliche Gather-Vorgang über einen Link möglichst einfach gehalten wäre. Von der Haar-Basis ist die Durchführung bekannt: Der berechnete Interaktions-Kernel besteht aus einem einzigen Formfaktor. Der konstante (und einzige) Koeffizient des Quell-Patches wird mit diesem Formfaktor multipliziert. Das Ergebnis ist die Energie, die vom Quell-Patch ausgehend den Ziel-Patch erreicht. Diese eintreffenden Werte werden für alle gathering Links des Patches akkumuliert und ergeben schließlich die gesamte gesammelte Energie. Eine Konvertierung in Radiosity-Werte, d.h. ein Einbringen des Spiegelungsverhaltens des Ziel-Patches erfolgt erst während der Push/Pull-Phase. Letztgenanntes gilt auch für alle anderen Basen.

Für die anderen Basen ergeben sich unterschiedliche Änderungen:

Bei Flatlet-Basen fallen diese nur gering aus; die Basisfunktionen sind immer noch konstant. Demnach besteht der Interaktionsfaktor ebenfalls aus Formfaktoren, nur handelt es sich jetzt um eine  $k \times k$ -Matrix von Formfaktoren, wobei  $k$  die Anzahl der Koeffizienten eines Patches ist. Der Mechanismus des Gatherings über einen Link bleibt abstrakt gesehen unangetastet: Die Koeffizienten des Quell-Patches (jetzt ein  $k$ -stelliger Vektor) wird mit dem Interaktionsfaktor des Links (einer  $k \times k$ -Matrix) multipliziert und ergibt somit die Koeffizienten des Ziel-Patches (wieder ein  $k$ -stelliger Vektor). Die Ergebnisse aller beteiligten Links werden addiert. Dieses Vorgehen ist nur möglich, weil auch hier Schritt eins und drei der oben beschriebenen Vorgehensweise wegfallen bzw. keine Auswirkung auf das Ergebnis haben.

Für Multiwavelets müssen aber genau diese Schritte durchgeführt werden. Um den Mechanismus der einfachen Multiplikation weiter verwenden zu können, berechnet man alle drei Schritte im voraus, wenn der Link zwischen den Patches gebildet wird. Das Ergebnis ist wieder ein Interaktionsfaktor, der aus einer Matrix besteht. Nur diesmal bestehen die einzelnen Elemente der Matrix nicht mehr aus einfachen Formfaktoren.

#### 5.4.4.1 Die Berechnung des Interaktionsfaktors für Mult2-Wavelets

Eine genaue Beschreibung der allgemeine Vorgehensweise findet sich bei Willmott [26]. Dort wird die Matrix Transport Gleichung vorgestellt (*matrix transport equation*):

$$\tilde{d} = \underbrace{D^{-1}}_{(3.)} \underbrace{M^t K}_{(2.)} \underbrace{M}_{(1.)} \tilde{s}$$

Die angegebene Numerierung der einzelnen Terme entspricht der unter 5.4.4 angegebenen und beschreibt die Aufgabe, die dem jeweiligen Teil der Formel zukommt.  $\tilde{s}$  bezeichnet die Koeffizienten des Quell-Patches (*source*),  $M$  übernimmt die Rekonstruktion der Radiosity-Funktion,  $K$  enthält die Kernel-samples,  $D^{-1}M^t$  schließlich übernimmt die Projektion auf die Koeffizienten des Ziel-Patches. Diese sind mit  $\tilde{d}$  bezeichnet (*destination*). Eine allgemeine Beschreibung der einzelnen Terme und deren Zusammensetzung ist der o.g. Veröffentlichung zu entnehmen.

In einer Implementierung können nun die Matrizen  $M$  und  $D^{-1}M^t$  vorberechnet werden. An dieser Stelle soll am Beispiel der Multiwavelet2-Basis erläutert werden, wie diese zu verwenden sind und wie sie berechnet werden.

Zur Anwendung kommen dabei die Matrizen  $M2\_R$  (entspricht  $D^{-1}M^t$ , das „R“ steht für receive) und  $M2\_S$  (entspricht  $M$ , das „S“ steht für source):

$$M2\_R = \begin{pmatrix} 0.5 & 0.5 \\ -0.866025 & 0.866025 \end{pmatrix} \quad M2\_S = \begin{pmatrix} 0.5 & -0.288675 \\ 0.5 & 0.288675 \end{pmatrix}.$$

Die Werte der Komponenten der Matrizen  $M2\_R$  und  $M2\_S$  ergeben sich dabei aus folgender Rechnung:

**Gegeben** Funktionenbasis  $b$ , bestehend aus der konstanten Funktion  $F(x) = 1.0$  und der linearen Funktion  $F(x) = x$ . Weiterhin gegeben ist die Quadraturregel  $GL_2 = \begin{pmatrix} 0.5 & -0.57735027 \\ 0.5 & 0.57735027 \end{pmatrix}$ . Die zweite Spalte dieser Matrix besteht aus den Stützstellen der Quadratur, die erste Spalte gibt die modifizierten Gewichte an (s.u.).

**Gesucht** Matrizen  $R$  und  $S$ , s.d.  $T = R \cdot K \cdot S$ .

Dazu werden die folgenden Schritte durchgeführt:

1. Berechne mit Hilfe der Funktionenbasis und der Quadraturregel die Basismatrix

$$B = \begin{pmatrix} 1 & -0.57735027 \\ 1 & 0.57735027 \end{pmatrix}.$$

Die zweite Spalte dieser Matrix entspricht weiterhin den benötigten Stützstellen für eine Gauß-Legendre Quadratur, die erste Spalte gibt die benötigten Gewichte an (siehe [4] für weitere Details zur Gauss-Legendre-Quadratur).

2. Berechne die Gewichtematrix  $W$ , eine Diagonalmatrix, die als Elemente die erste Spalte der Quadraturregel  $GL_2$  enthält.

$$W = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}$$

3. Berechne  $M2\_S$ .

$$S = W \cdot B = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix} \cdot \begin{pmatrix} 1 & -0.57735027 \\ 1 & 0.57735027 \end{pmatrix} = \begin{pmatrix} 0.5 & -0.288675 \\ 0.5 & 0.288675 \end{pmatrix} = M2\_S$$

4. Berechne  $M2\_R$ .

$$R = (B^t \cdot W \cdot B)^{-1} \cdot B^t \cdot W = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ -0.57735027 & 0.55535027 \end{pmatrix}$$

$$= \begin{pmatrix} 0.5 & 0.5 \\ -0.866025 & 0.866025 \end{pmatrix} = M2\_R$$

An dieser Stelle soll noch einmal betont werden, daß die Matrizen  $M2\_R$  und  $M2\_S$  im *Voraus* berechnet werden. Nur die Bestimmung der Kernelsamples  $K$  findet während des eigentlichen Verfahrens statt. Mit Hilfe dieser Matrizen wird für jeden Link die  $4 \times 4$ -Matrix des Interaktionsfaktors berechnet:

```

K = berechne den Kernel mit Hilfe von Kernelsampling
    bestimme dabei die Sichtbarkeit zwischen den Patches
T = M2_R · K · M2_S // Tensormultiplikation
intFaktor = T · ermittelte Sichtbarkeit

```

### 5.4.5 Abbruchkriterium

Idealerweise sollte das Verfahren beendet werden, wenn keiner der Links mehr unterteilt werden muß (d.h. der durch den Link provozierte Fehler liegt unterhalb der vorgegebenen Grenze) und wenn ein Equilibrium erreicht wurde (d.h. die gesamte Energie wurde in der Szene verteilt).

In der Praxis hat sich jedoch herausgestellt, daß nicht immer beide Fälle eintreten: Während des Sammelns von Energie kann es vorkommen, daß das System nicht zur Ruhe kommt. Bei stark spiegelnden Oberflächen kann es passieren, daß zyklisch immer wieder die gleichen Links ausgewertet werden und die gesammelte Energie kaum abnimmt. Es kommt zu einer großen Anzahl von Gathering-Schritten, die aber kaum Einfluß auf das Ergebnis haben. In der Implementierung ist es daher sinnvoll, eine Obergrenze für die Anzahl der Gathering-Schritte angeben zu können.

Weiterhin hat sich das Steuern des Ergebnisses über die Fehlerschranke der Links als relativ mühsam herausgestellt. Oft schießt man nach einem zu groben Ergebnis und einer geringfügigen Änderung der Fehlerschranke weit über das gewünschte Ziel hinaus. Aus diesem Grund wurde eine obere Schranke für die Anzahl der Verfeinerungsschritte eingerichtet. Mit ihrer Hilfe ist eine direkte Kontrolle über die Güte des Ergebnisses und die Höhe des betriebenen Aufwandes möglich.

## 5.5 Dreiecke und Wavelet-Radiosity

In den bisherigen Ausführungen blieben Dreiecke noch unbeachtet. Wenn man allerdings mit Szenen arbeitet, die eine Vielzahl von Objekten, z.B. Tetraeder oder gekrümmte Objekte, enthalten, so kann man eine Bearbeitung von Dreiecke nicht auslassen.

Im folgenden werden die Unterschiede zwischen Dreiecken und Vierecken thematisiert. Dies betrifft vor allem die Anzahl der benutzen Koeffizienten pro Patch, die zugrundeliegenden Basisfunktionen wie auch das Verwenden von Tensormultiplikation. An dieser Stelle sollen nur auf Besonderheiten hingewiesen werden. Nicht genannte Sachverhalte sind implizit als mit den Vierecken übereinstimmend anzunehmen.

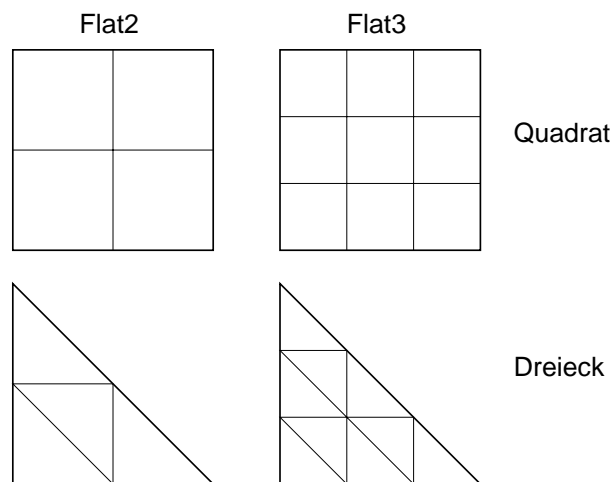


Abbildung 5.4: Unterstütze Bereiche der Flat2- und Flat3-Basisfunktionen bei Dreiecken und Vierecken. Die Anzahl der impliziten Sub-Patches bleibt erhalten.

## 5.5.1 Änderungen gegenüber Vierecken

### 5.5.1.1 Haar- und Flatlet-Basen

Bei den Wavelet-Basen mit ausschließlich konstanten Basisfunktionen ergeben sich keine weitgreifenden Änderungen: Die Anzahl der Koeffizienten für dreieckige Patches bleibt gleich der Anzahl für viereckige Patches. Bei Flatlets ergibt sich eine neue Lage der Bereiche, die vom jeweiligen Koeffizienten unterstützt werden. Konnte man sich diese Bereiche beim Viereck noch als eine regelmässige  $2 \times 2$  bzw.  $3 \times 3$  Unterteilung vorstellen, so sieht diese Aufteilung nun anders aus (siehe Abbildung 5.4). Es bleibt eine regelmäßige Unterteilung in vier bzw. neun Sub-Patches.

Für die Haar-Basis ergeben sich keine Änderungen.

### 5.5.1.2 Multiwavelets

Ein Punkt auf einem Dreieck wird, wie beim Viereck, durch zwei Koordinaten  $s$  und  $t$  spezifiziert. Der Unterschied besteht hierbei in Wertebereich und Ursprung:

Bei Vierecken galt  $(s, t) \in [-1, 1] \times [-1, 1]$  mit dem Mittelpunkt des Patches als Ursprung. Bei Dreiecken gilt nun  $(s, t) \in [0, 1] \times [0, 1]$ . Als Ursprung wird hierbei einer der Eckpunktes des Patches festgelegt (dies entspricht ebenfalls einem baryzentrischem Koordinatensystem, siehe [4]).

Eine grundlegende Änderung erfahren die Basisfunktionen: Für zwei vanishing Moments bestehen die Funktionen, die in der Basis enthalten sind, nun aus  $x$ ,  $y$  und  $1 - x - y$ . Es gibt also keine Basisfunktion mehr, die für den konstanten Anteil zuständig ist. Ein Patch besitzt nur noch drei Koeffizienten. Diese geben den Wert der Funktion an den drei Eckpunkten des Patches an. Will man eine Approximation des Ergebnisses anzeigen (siehe Abschnitt 6.4), so kann man nicht nur den konstanten Koeffizienten zu diesem Zweck heranziehen, denn dieser ist nicht mehr vorhanden. Stattdessen bildet man den Durchschnitt aller drei Koeffizienten um den konstanten Energietransfer anzunähern.

Ein analoges Vorgehen ergibt sich für Multiwavelets mit drei vanishing Moments; hier besitzt jedes Patch sechs Koeffizienten (gegenüber neun bei Vierecken). Auch hier besteht der konstante Anteil im Durchschnitt aller Koeffizienten. Seien  $a = x$ ,  $b = y$  und  $c = 1 - x - y$  die Farbwerte in den Eckpunkten des Dreiecks, so bestehen die Basisfunktionen aus  $a^2$ ,  $b^2$ ,  $c^2$ ,  $2 \cdot ab$ ,  $2 \cdot bc$  und  $2 \cdot ca$ . Man sieht, daß mit Hilfe dieser Basisfunktionen ein linearer oder quadratischer Funktionsverlauf darstellbar ist.

### 5.5.1.3 Allgemeine Änderungen für alle Basen

Bei Vierecken können die benötigten 2D-Funktion mit Hilfe von Tensorprodukten aus zwei 1D-Funktionen kombiniert werden. Bei Dreiecken ist dies nicht der Fall: Anstatt der Tensormultiplikation muß nun Matrizenmultiplikation eingesetzt werden. Dies führt zu einem höheren Berechnungsaufwand, da Tensorprodukte gegenüber Matrizenmultiplikationen weniger arithmetische Operationen benötigen.

Zieht man diese Änderungen wie auch den Wechsel der Basisfunktionen in Betracht, so bleiben alle anderen, bereits bei Vierecken genannten Vorgehensweisen unangetastet. Es gilt bei jedem Berechnungsschritt zu unterscheiden, welche Form die beteiligten Patches haben und entsprechend die Art der Basisfunktionen sowie die Möglichkeit einer Tensormultiplikation einzubeziehen.

So entstehen in einer Mult2-Szene, die sowohl Dreiecke wie auch Vierecke enthält, Interaktionsfaktoren, die aus einer  $4 \times 3$  oder  $3 \times 4$ -Matrix bestehen. Betrachte man einen Link, der ausgehend von einem Viereck zu einem Dreieck führt: Der Kernel zwischen den beiden Patches wird berechnet und es ergibt sich eine  $3 \times 4$  Matrix (3 ist die Anzahl der Koeffizienten auf dem Ziel-Patch, 4 der entsprechende Wert für das Quell-Patch). Da es sich bei der Quelle um ein Viereck handelt wird diese Kernelmatrix nun von rechts mit einer vorberechneten Matrix tensormultipliziert (siehe 5.4.4.1 für weitere Einzelheiten über die Aufgabe und Berechnung dieser Matrizen). Anschließend wird eine volle Matrizenmultiplikation von links durchgeführt. So geht die Art der Patches sowohl in die Berechnung des Kernels, wie auch in die Rekonstruktion bzw. Projektion der Koeffizienten ein.

# Kapitel 6

## Einbettung in den MRT

Um nun das Verfahren zu implementieren braucht man ein Programmiersystem, das die Rahmenbedingungen setzt und Werkzeuge zur Verwirklichung bereithält. Im Falle dieser Implementierung handelt es sich um das 3D-Graphiksystem MRT (*Minimal Rendering Toolkit*). Dieses System soll im folgenden kurz erläutert werden. Diese Arbeit beschränkt sich dabei auf eine allgemeine Einführung und eine Beleuchtung der für den Algorithmus relevanten Teile.

Dabei wird sowohl den benötigten Datenstrukturen als auch der Ausgabe des Ergebnisses besondere Aufmerksamkeit geschenkt.

### 6.1 Einführung in den MRT

Beim Minimal Rendering Toolkit MRT handelt es sich um ein Visualisierungswerkzeug, das in der Forschungsgruppe für Computergraphik der Universität Bonn als Lehr- und Forschungsplattform zum Einsatz kommt. MRT eignet sich besonders gut für diesen Einsatz, weil es für viele unterschiedliche Plattformen zur Verfügung steht. So sind zu nennen: Windows95, WindowsNT, Solaris von SUN, LINUX und IRIX von SGI.

Es ist daher möglich, daß viele Personen mit ihren unterschiedlichen Bedürfnissen an diesem System arbeiten und entwickeln können. Dies hat zur Folge, daß ein breites Spektrum von Einsatzmöglichkeiten abgedeckt werden kann, denn obwohl man vom Namen her auf eine kleine Bibliothek schließen könnte, handelt es sich doch um ein System, das eine Vielzahl von graphischen Algorithmen unter einem Dach vereinigt. Dabei kann man sich mit einem Teilaspekt beschäftigen, ohne ein Verständnis des gesamten Systems voraussetzen zu müssen. Ein einfacher Einstieg ist genauso durchführbar wie die Umsetzung komplexer Methoden.

Der MRT beinhaltet eine große Anzahl der gängigsten graphischen Algorithmen und Darstellungsverfahren. Darunter finden sich u.a. Ray-Tracing, Volume-Rendering und Radiosity-Verfahren.

### 6.2 Besonderheiten des MRT

Der Unterschied zwischen MRT und anderen Graphikbibliotheken besteht im objektbasierten Ansatz und in der konsequenten Anwendung des objektorientierten Paradigmas. Das System ist in C++ implementiert. Alle einzelnen Komponenten des Systems bestehen aus Objekten, d.h. sie können leicht verwendet werden, da durch die genaue Beschreibung der Schnittstellen klar wird, welche Funktionalität zur Verfügung steht. Sollte man feststellen, daß diese Funktionalität für die eigenen Zwecke nicht ausreicht, so kann durch Ableitung leicht eine eigene Klasse geschaffen werden, die dann die gewünschten Möglichkeiten bietet.

Das Design des MRTs folgt der in [7] beschriebenen Grundidee, daß den immer komplexer werden den Herausforderungen der graphischen Datenverarbeitung nur begegnet werden kann, wenn Szenen-

hierarchien optimal ausgenutzt werden und Darstellungsobjekte so lange wie möglich in ihrer ursprünglichen Repräsentation bereitgehalten werden. Dies bedeutet, daß 3D-Objekte erst zum spätest möglichen Zeitpunkt in polygonale Oberflächenapproximationen umgewandelt werden, was den entscheidenden Vorteil hat, daß zu jedem Zeitpunkt der Berechnung auf die Definition des Objektes zurückgegriffen werden kann. Man hat somit jederzeit Informationen zur Verfügung, die bei einer frühen Approximation verlorengehen.

Dazu ein kurzes Beispiel: Gegeben sei eine Radiosity-Szene, die eine Kugel enthält, die wiederum grob durch Dreiecke approximiert wurde. Startet man nun eine HR-Berechnung, so werden diese groben Patches in feinere unterteilt. Dank der vorhandenen Objektinformationen, nämlich dem Wissen, daß es sich um eine Kugel mit bestimmten Radius und Mittelpunkt handelt, ist es nun möglich, die Eckpunkte der neu entstandenen Patches genau auf die Oberfläche der Kugel zu setzen. Mit der weitergehenden Verfeinerung wird die Approximation sich immer mehr der exakten Kugel angleichen. Ohne die Objekt-Information ist aber ein solches Vorgehen nicht realisierbar. Man müßte von vornherein die Kugel stärker unterteilen, was zu einer höheren Komplexität des HR-Verfahrens führt (siehe [19]).

Für die Umsetzung von Radiosity-Verfahren ist es weiterhin wichtig, daß die Möglichkeit besteht, Schnittpunkte zwischen Strahlen und Objekten exakt zu bestimmen. Diese Fähigkeit findet Anwendung in der Bestimmung der Sichtbarkeit zwischen Paaren von Patches. Man erhält somit auch für gekrümmte Objekte einen realistischen Schattenwurf.

Für die Verfahren des Polygon-Renderns gilt, daß die oben beschriebene Plattformunabhängigkeit dadurch erreicht wird, daß alle darstellungsrelevanten Funktionsaufrufe in der Klasse `t_CGI3D` gebündelt werden. Alle anderen Klassen rufen Methoden innerhalb von `t_CGI3D` auf. Von dort wird dann zentral auf die 3D-Graphikbibliothek zurückgegriffen, über welche die jeweils zur Verfügung stehende 3D-Graphikhardware angesprochen wird. So können bereits OpenGL oder XGL unterstützt werden, aber auch hier kann leicht eine Erweiterung auf andere Graphikbibliotheken, wie z.B. Direct3D von Microsoft, realisiert werden.

Die Werkzeuge, die MRT zur Verfügung stellt, sind so vielfältig, daß eine Erklärung den Rahmen dieser Arbeit sprengen würde. Deshalb an dieser Stelle nur eine kurze Nennung der wichtigsten Klassen:

- `t_Camera` modelliert das Konzept einer virtuellen Kamera,
- `t_Light` setzt das Konzept einer Lichtquelle um,
- `t_Scene` repräsentiert eine Szenenhierarchie,
- `t_SurfaceShader` modelliert die Beschaffenheit einer Oberfläche.

Der Ansatzpunkt für die Implementierung von Radiosity-Verfahren stellt in diesem Rahmen die Illuminated Scene (Klasse `t_IllumScene`) dar. Diese enthält die Lichtquellen wie auch die Geometrie der Szene. `t_IllumScene` stellt die Funktionen `RadiosityStep` und `RadiosityInit` zur Verfügung, die durch Ableitung überschrieben werden können, um somit die Funktionalität eines Schrittes des Radiosity-Verfahrens bzw. die Initialisierung umzusetzen.

### 6.2.1 Objekte der Szene

Eine der zentralen Klassen stellt das 3D-Objekt `t_Object` dar. Bei dieser Klasse handelt es sich um die Basisklasse aller im MRT zu berücksichtigenden Objekte. Sie stellt eine Reihe grundlegender Methoden zur Verfügung, durch deren Überschreibung leicht eigene Objekte eingebracht werden können. Will man ein neues Verfahren implementieren, so muß man beachten, daß dieses mit einer Vielzahl unterschiedlichster Objekte zurecht kommen muß.

Für Radiosity-Verfahren gilt diese Aussage jedoch nur bedingt: Der erste Schritt, der vorgenommen wird, besteht im *Meshing*, der Umwandlung der zu betrachtenden Objekte in eine Anzahl von planaren

Patches. Mit anderen Worten: Es wird eine Umwandlung in eine polygonale Oberflächenapproximation des jeweiligen Objekts durchgeführt. Dies geschieht durch die `approxShape`-Methode der Klasse `t_Object`, die als Resultat eine Instanz der Klasse `t_Brep` liefert, eine Boundary Representation [2]. Dies ist die Datenstruktur, auf der der Wavelet Radiosity-Algorithmus arbeitet und mit dessen Hilfe die Anzeige des Ergebnisses erfolgt.

Nun ist bekannt, daß wir es nicht mehr mit einer Vielzahl von unterschiedlichen Objekten zu tun haben, sondern mit einer polygonalen Oberflächenapproximation dieser Objekte. Dies wiederum bedeutet aber, daß wir nicht damit rechnen dürfen, nur Vierecke präsentiert zu bekommen, denn viele der Approximation werden vor allem Dreiecke enthalten. Man stelle sich zum Beispiel eine Kugelapproximation vor, die allein aus Dreiecken besteht. Das in [10] beschriebene Verfahren geht auf Dreiecke nicht weiter ein, es sind aber einige Umstellung notwendig (für Informationen, wie das Verfahren für Dreiecke realisiert wird, siehe Abschnitt 5.5).

## 6.3 Benötigte Datenstrukturen

Für eine Beschreibung des Wavelet Radiosity-Algorithmus sei auf Kapitel 5 verwiesen. Hier werden die für den WR-Algorithmus relevanten Datenstrukturen erläutert. Im wesentlichen konzentrieren sich die Ausführungen auf die Unterschiede zum HR-Ansatz, der bereits als Implementierung vorlag (siehe [3]).

### 6.3.1 Speicherverbrauch

Eine Wavelet-Basis mit  $M$  vanishing Moments benötigt pro Patch eine Anzahl von  $M^2$  Koeffizienten, d.h. eine Gleitkommazahl für die Haar-Basis, vier für Flatlets und Multiwavelets mit zwei vanishing Moments und neun für die genannten Basen mit drei vanishing Moments.

Für einen Link werden noch mehr Datenfelder benötigt: Da ein Link die Interaktion zwischen den Koeffizienten eines Paares von Patches modelliert, werden diese einzelnen Terme in einer Matrix der Größe  $M^2 \times M^2$  festgehalten. Es handelt sich also um ein Aufkommen von  $M^4$  Gleitkommazahlen pro Link. Für Basen mit drei vanishing Moments bedeutet dies, daß 81 Gleitkommazahlen pro Link gespeichert werden müssen. Hinzu kommt noch ein erhöhter Bedarf an Speicher, weil für die Implementierung einer an dieser Stelle eingesetzten allgemeinen Matrizenklasse zusätzliche Datenfelder gehalten werden müssen (z.B. die Anzahl der Zeilen und Spalten). Dies alles stellt einen erheblichen Speicherverbrauch dar.

Ob und wie dieser hohe Verbrauch gesenkt werden könnte, wird in Abschnitt 9.3 diskutiert.

## 6.4 Ausgabe des Ergebnisses

Die Ausgabe des Ergebnisses stellt ebenfalls ein Problem dar, welches es zu lösen gilt: Wie jeder Polygon-Renderer unterliegt der Einsatz der BReps gewissen Einschränkungen, die sichtbar werden, wenn Radiosity-Funktionen dargestellt werden sollen, die nicht mehr einen konstanten Wert für jedes Patch vorweisen. Im folgenden wird deshalb beschrieben, wie eine Ausgabe erfolgen kann. Da sich unterschiedliche Anforderungen ergeben, muß hierbei die gewählte Wavelet-Basis einbezogen werden.

Wenn in diesem Abschnitt von einer exakten Lösung die Rede ist, so ist dies unter den gegebenen Rahmenbedingungen zu verstehen. So kann eine Lösung nicht im mathematischen Sinne exakt angezeigt werden, weil die Auflösung des Schirms eine „natürliche“ Grenze bildet.

### 6.4.1 Anzeige über BReps

Eine BRep-Struktur setzt sich aus den Ecken (*vertices*) und der Fläche zusammen, die von diesen Eckpunkten begrenzt wird (*face*). Weiterhin gehören noch die Halbkanten, welche die Eckpunkte mitein-

ander verbinden, dazu (*edges*). Details und Hintergründe hierzu finden sich in [2]. Die hier gemachten Angaben beschränken sich auf die bereits vorgestellten Wavelet-Basen.

Es besteht die Möglichkeit, Farbwerte in der Fläche sowie in den Halbkanten zu speichern, wobei der Farbwert der Fläche standardmäßig für eine Flat-Darstellung verwendet wird. Die Farbwerte in den Halbkanten finden beim Gouraud-Shading Anwendung.

Der Grund, weshalb man Farbwerte in den Halbkanten, aber nicht in den Eckpunkten speichern kann, besteht darin, daß man sonst z.B. einen Würfel nicht angemessen darstellen könnte. Denn betrachtet man einen beliebigen Würfeckpunkt, dessen angrenzende Seiten unterschiedliche Farben haben, so kann man nicht eindeutig die Farbe dieses Eckpunktes angeben.

#### 6.4.1.1 Darstellung bei Verwendung der Haar-Basis

Für ein Radiosity-Verfahren mit einer konstanten Basisfunktion pro Patch kann der jeweilige Wert der Radiosity-Funktion in der Fläche gespeichert werden. Um eine glattere Darstellung zu erhalten werden üblicherweise die Farbwerte der Eckpunkte durch eine Durchschnittsbildung der Farbwerte der umliegenden Flächen gebildet. Diese werden in den zugehörigen Halbkanten gespeichert. Der Farbwert der Halbkante wird also mit dem Farbwert des Eckpunktes assoziiert. In diesem Fall ist eine eindeutige Zuordnung möglich. Bei dieser Durchschnittsbildung werden die einzelnen Beiträge zur Gesamtsumme durch die gegenseitige Lage der Patches zum betrachteten Punkt gewichtet. Wird das Patches schließlich angezeigt, ist es möglich, mit Hilfe einer Gouraud-Schattierung einen glatten Verlauf über die Grenzen des Patches hinweg zu erzeugen.

Diese Darstellung stellt dann aber nicht mehr das exakte Ergebnis der Radiosity-Berechnung dar, sondern eine nachbehandelte, geglättete Version.

#### 6.4.1.2 Darstellung bei Verwendung der Multiwavelet2-Basis

Durch einen ähnlichen Mechanismus ist es möglich, Mult2-Patches auszugeben. Bei dem oben geschilderten Fall der Haar-Basis gab der Farbwert der Fläche das exakte Ergebnis an. Die Farbwerte der Eckpunkte wurden aus diesem Wert berechnet. Jetzt ist genau das Gegenteil der Fall: Die Werte der Radiosity-Funktion an den Eckpunkten können mit Hilfe der Koeffizienten berechnet werden. Diese Ergebnisse werden in den Halbkanten gespeichert. Der Farbwert der Fläche ergibt sich aus dem Koeffizienten des Patches, der für den konstanten Anteil der Funktion zuständig ist.

Der Farbwert der Fläche stellt also nur eine Approximation dar, während die Eckpunkte die exakte Lösung enthalten.

Das Ergebnis ist deshalb exakt, weil der Farbverlauf über ein Mult2-Patch höchstens linear sein kann und dies mit Hilfe einer Überblendung der Farbwerte entlang der Kanten erreicht wird.

Eine Einschränkung ist hierbei in der Anfangsphase der Berechnung zu machen: Die durch große Patches erzeugten Fehler können dazu führen, daß der Farbwert eines Eckpunktes, der an verschiedene Patches grenzt, nicht exakt bestimmt werden kann, da dieser nicht mehr, wie bei der Haar-Basis, durch Zusammenwirken aller benachbarten Faces ermittelt wird. Hier wird er allein durch die Koeffizienten *eines* Patches ermittelt. Solange der Fehler noch relativ groß ist, können sich also unterschiedliche Farbwerte ergeben, je nachdem, welche Face befragt wird. Diese Einschränkung wird jedoch aufgehoben, wenn die Patches im Verlaufe des Verfahrens weiter unterteilt werden, denn mit einem kleineren Fehler weichen die beschriebenen Werte auch immer weniger voneinander ab.

Eine Möglichkeit, diese Einschränkung zu umgehen, stellt folgendes Vorgehen dar: Alle benachbarten Faces eines Eckpunktes werden nach dem Farbwert befragt, dem sie den betrachteten Eckpunkt zuteilen würden. Das Ergebnis wird dann als der Durchschnitt aller dieser Farbwerte angegeben. Dies stellt nun nicht mehr das exakte Ergebnis der Berechnung dar, führt aber zu einem glatteren Verlauf über die Grenzen des Patches hinweg. Ob dieser Nachbearbeitungsschritt angewendet werden soll, kann über

die Option `avgVertex` festgelegt werden. Da alle Faces durchlaufen werden müssen, um die Durchschnittswerte zu ermitteln, ergibt sich bei eingeschalteter Option ein leicht erhöhter Zeitaufwand (siehe 7.1.3.1 für mehr Informationen über Optionen).

### 6.4.1.3 Darstellung bei Verwendung anderer Basen

Für alle anderen Wavelet-Basen ist die direkte Darstellung über einen BRep nur bedingt geeignet. Es ist nur möglich, eine Approximation des Ergebnisses anzuzeigen. Dies hat verschiedene Gründe:

- Flatlets mit zwei vanishing Moments

Es müssen im allgemeinen vier Sub-Patches mit unterschiedlichen Farben durch Flat-Shading ausgegeben werden. Die Face eines BReps kann jedoch nur einen Farbwert aufnehmen.

Alternativ könnte man die geforderten vier Farbwerte in den Kanten ablegen. Dieses Vorgehen kann aber nur als Umgehung des eigentlichen Problems angesehen werden, denn von einer sauberen Modellierung kann man hier nicht mehr sprechen. Außerdem würde dies nur eine Lösung für diese spezielle Basis darstellen und hätte keine Allgemeingültigkeit für Erweiterungen (siehe nächster Punkt).

Mit Hilfe der BReps wird für diese Basis nur eine Approximation ausgegeben: Der ermittelte Farbwert besteht dabei aus dem Durchschnittswert aller Sub-Patches auf dem Patch. Gerade bei Schattengrenzen, die über ein Patch verlaufen, kann dies zu extremen Ergebnissen führen, denn sollte z.B. die Schattengrenze genau in der Mitte des Patches verlaufen, so wären bei der exakten Lösung die beiden Sub-Patches der Schattenseite sehr dunkel und die beiden anderen wesentlich heller. Durch eine Durchschnittsbildung wird dieses Ergebnis stark „verwischt“; der Verlauf des Schattens ist somit nicht mehr klar zu erkennen.

- Flatlets mit drei vanishing Moments

Hier sind die Anforderungen und Vorgehensweise analog zu Flatlets mit zwei vanishing Moments: Es müssen neun verschiedene Farbwerte pro Patch ausgegeben werden. Die Durchschnittsbildung für die Approximation wird entsprechend erweitert.

Die gestiegene Anzahl von Koeffizienten bedeutet auch, daß hier die Möglichkeit, die exakten Farbwerte in den Halbkanten zu speichern, nicht mehr gegeben ist.

- Multiwavelets mit drei vanishing Moments

Diese Wavelet-Basis präsentiert ein anders gelagertes Problem: Die Farbwerte an den Eckpunkten eines Patches können, wie bei der gleichen Basis mit nur zwei vanishing Moments, exakt berechnet werden. Die Anzeige eines linearen Verlauf zwischen diesen Punkten genügt jetzt aber nicht mehr, da ein quadratischer Verlauf vorliegen kann. Eine lineare Darstellung würde dem Ergebnis nicht gerecht und stellt somit wieder nur eine Approximation dar.

## 6.4.2 Erzeugung einer Textur zur Darstellung des Ergebnisses

Durch die weite Verbreitung von Hardware-gestützter Anzeige von Texturen liefern diese ein alternatives Vorgehen, um das exakte Ergebnis der Wavelet Radiosity-Berechnung, bei Verwendung von quadratischen Patches, anzuzeigen. Warum dies nicht für den allgemeinen Fall möglich ist, wird anschließend erläutert.

### 6.4.2.1 Erzeugen und Aufbringen der Textur

Nachdem die Radiosity-Berechnung abgeschlossen ist, folgen zwei weitere Bearbeitungsschritte:

1. Berechnung einer Textur, die die Radiosity-Funktion widerspiegelt,
2. Aufbringen dieser Textur auf der Wurzel der Patch-Hierarchie.

Der zweite Schritt ist einfach umzusetzen, denn der MRT stellt die benötigten Methoden zur Verfügung. Beim ersten Schritt sieht das jedoch etwas anders aus. Eine Berechnung ist hierbei auch von der gewählten Wavelet-Basis abhängig, die, wie bereits gezeigt, unterschiedliche Anforderungen an die Anzeige stellt. Man geht wie folgt vor:

**Gegeben** Eine Menge von Patch-Hierarchien. Jede davon enthält auf der untersten Stufe Koeffizienten, mit deren Hilfe die Radiosity-Funktion in feinstmöglicher Auflösung berechnet werden kann.

**Gesucht** Eine Textur die, auf das Wurzel-Patch geklebt, den Verlauf der Radiosity-Funktion über das Patch wiedergibt.

Um das Ziel zu erreichen, werden mit jeder Hierarchie von Patches diese Schritte durchgeführt:

1. Bestimme die Tiefe  $d$  der Hierarchie.
2. Lege eine Textur von ausreichender Größe an, um den Farbverlauf über dem Wurzel-Patch darstellen zu können. Größe bezeichnet hierbei die Anzahl von Pixeln auf der Textur in *einer* Dimension des Patches. Eine  $4 \times 4$ -Textur hat somit die Größe 4.
3. Durchlaufe alle Patches auf der untersten Ebene und bestimme Farbwerte mit Hilfe der dort vorhandenen Koeffizienten. Speichere diese Werte an entsprechender Stelle der Textur.
4. Fülle die eventuell freigebliebenen Positionen der Textur mit interpolierten Werten, die aus den bereits gegebenen Werten von Schritt 3 errechnet werden.

Zu 1.: Der erste Schritt ist für alle Wavelet-Basen gleich; die Aufgabe, die Tiefe  $d$  der Hierarchie zu bestimmen, ist unabhängig von der gewählten Basis. Eine Hierarchie, die nur aus der Wurzel besteht, hat hierbei die Tiefe  $d = 0$ .

Zu 2.: Die Größenbestimmung der gesuchten Textur ist für alle Basen von der Tiefe der Hierarchie abhängig. Dies bedeutet aber nicht, daß das Ergebnis für alle diese Basen gleich ausfällt. Der Unterschied besteht darin, wieviele Texturelemente für ein Patch auf der untersten Ebene benötigt werden:

- Da die **Haar-Basis** dort nur einen konstanten Wert aufzuweisen hat, genügt für sie eine Größe von  $2^d$ .<sup>1</sup>
- Für **Flatlets** enthält die unterste Hierarchie vier bzw. neun konstante Farbwerte bereit. Dies sind zwei bzw. drei Werte in jeder Dimension. Für sie ist also die Größe der Textur mit  $2^d \cdot 2 = 2^{d+1}$  bzw.  $2^d \cdot 3$  anzugeben.
- Für **Multiwavelets** ist die Frage, wieviele Darstellungselemente genügen, nicht so ohne weiteres zu beantworten. Die Texturpixel können nur eine konstante Farbe annehmen, die dargestellte Funktion ist aber linear oder quadratisch. Es gilt diesen Verlauf zu diskretisieren: Das geschieht

<sup>1</sup>Bei der Implementierung hat sich herausgestellt, daß OpenGL Probleme mit Texturen hat, deren Größe unter vier liegt. Daher weisen alle realisierten Texturen eine Mindestgröße auf, die von der Art der gewählten Basis abhängt.

durch einfache Festlegung der Texturgröße. Mit anderen Worten: Man legt fest, wieviele Elemente der Textur auf ein Patch der untersten Hierarchiestufe fallen sollen. Dieser vorgegebene Wert kann über den entsprechenden Eintrag in den Optionen verändert werden (siehe 7.1.3.1). Dabei ist es sinnvoll, eine Zweierpotenz anzugeben, da die Textur am Ende der Berechnung auf die Größe einer Zweierpotenz skaliert wird. Man kann somit sicherstellen, daß die angeforderte Auflösung auch ausgegeben wird.

Weiterhin sollte man darauf achten, daß die Größe der Textur nicht zu klein gewählt wird, denn sie gibt direkt an, wieviele Farbwerte für das Patch ermittelt werden. Eine Verifizierung, ob ein geeigneter Wert gewählt wurde, sollte durch eine optische Kontrolle leicht durchzuführen sein.

Zu 3.: Das Durchlaufen der Hierarchie stellt sich wieder für alle Basen gleich dar. Die Berechnung der einzelnen Farbwerte jedoch weicht stark von Basis zu Basis ab. So erfolgt die Berechnung der Farbeinträge nach den Regeln der jeweilig vorgegebenen Basis: für Haar- und Flatlet-Basen direkt über die Werte der einzelnen Koeffizienten und für Multiwavelets als Linearkombination.

Zu 4.: Im letzten Schritt werden dann die Positionen der Textur gefüllt, die noch keinen Farbwert erhalten haben. Dies wird dann notwendig, wenn nicht alle Sub-Patches eines Ausgangs-Patches die gleiche Hierarchiestufe besitzen. Die Vorgehensweise ist auch hier von der Wahl der Basis abhängig. Bei Haar- und Flatlet-Basen werden alle freien Stellen mit den entsprechenden bereits ermittelten Farbwerten gefüllt. Bei Multiwavelets werden die exakten Farbwerte berechnet und eingetragen.

Als abschließender Schritt wird dann die Textur auf die nächstgrößere Zweierpotenz skaliert, da OpenGL nur Texturen von dieser Größe darstellen kann. Dies bedeutet, daß für Flatlets mit drei vanishing Moments strenggenommen *nicht* das exakte Ergebnis dargestellt wird. So würde ein einzelnes, nicht unterteiltes Patch der genannten Basis nach der Skalierung eine  $4 \times 4$ -Textur anzeigen, obwohl eine  $3 \times 3$ -Einteilung exakt wäre.

#### 6.4.2.2 Grenzen des texturbasierten Ansatzes

Für gekrümmte Objekte lassen sich keine Texturen erzeugen, da das Aufbringen einer quadratischen Textur auf einem solchen Objekt nicht allgemeingültig lösbar ist. Die Zuteilung, welcher Punkt der Textur einem Punkt auf dem Objekt entspricht (sog. *mapping*), ist im allgemeinen Fall nicht möglich. Deshalb kann der Farbwert eines bestimmten Texturpixels nicht bestimmt werden.

Eine Texturerzeugung für Dreiecke wurde ebenfalls nicht implementiert, da hiervon nur die Klasse der reinen Dreiecke sowie das Tetraederobjekt profitieren würden. Es müßte jedoch für jedes Dreieck eine viereckige Textur angelegt werden, die dann nur zum Teil gefüllt wäre. Dies stellt einen zu großen Aufwand für einen selten auftretenden Sonderfall dar.

Für Szenen, die die beiden genannten Arten von Objekten enthalten, kann an dieser Stelle nur auf die direkte Ausgabe verwiesen werden (siehe 6.4.3).

#### 6.4.2.3 Flatlets und Gouraud-Schattierung

Eine Gouraud-Schattierung des Ergebnisses stellt sich bei Flatlet-Basen als relativ aufwendig dar, weil die topologischen Information, die der BRep liefert, nicht mehr ohne weiteres ausgenutzt werden können.

Zur Erinnerung: Die Aufgabenstellung sieht vor, für einen Eckpunkt den geglätteten Farbwert zu berechnen. Dies wird durch gewichtete Durchschnittsbildung aller Farbwerte der umgebenen Flächen erreicht. Im Haar-Fall konnte über einen Iterator, der alle benachbarten Faces des Eckpunktes besucht, leicht der Werte des Koeffizienten bestimmt werden, denn dieser entsprach der Farbe der Face. Nun aber ist nicht ohne weiteres klar, welcher der gesuchte Koeffizient des Nachbar-Patches ist. Nehmen wir einen Eckpunkt heraus: Der durch den BRep bereitgestellte Iterator liefert der Reihe nach die benachbarten Patches. Doch um nun den Koeffizienten zu finden, der zu dem Sub-Patch gehört, das in

unmittelbarer Nähe des Eckpunktes liegt, müssen im Extremfall alle Eckpunkte der beiden beteiligten Patches miteinander verglichen werden, um die gegenseitige Lage der Patches festzustellen. Erst mit dieser Information kann der zugehörige Koeffizient gefunden werden.

Eine Durchschnittsbildung der Farbwerte von Sub-Patches, die auf einem gemeinsamen Patch liegen, ist wiederum einfach. Da die gegenseitige Lage bekannt ist und alle Sub-Patches den gleichen Normalenvektor besitzen, können die relevanten Koeffizienten und damit der Durchschnitt direkt bestimmt werden.

Alternativ könnte eine Glättung mit Hilfe der berechneten Textur umgesetzt werden, da hier Nachbarschaftsinformationen direkt vorliegen. Die konstante Färbung der einzelnen Patches führt jedoch bei einer ungleichmäßigen Unterteilung dazu, daß mehrere Texturpixel gleicher Farbe auf das selbe Sub-Patch fallen. Eine einfache Durchschnittsbildung aller benachbarten Farbwerte führt somit nur für Randpunkte zu einem Farbverlauf. Die inneren Punkte der Sub-Patches bleiben unberührt. Um eine vollständige Glättung über das Sub-Patch durchführen zu können benötigt man Informationen, die nicht ohne weiteres zur Verfügung stehen. Diese gehen beim Übergang vom BRep auf die Textur verloren und müssen erst durch einen Vergleich der beiden Datenstrukturen neu gewonnen werden. Eine kostengünstige Vorgehensweise im Umgang mit diesem Problem kann an dieser Stelle nicht gegeben werden (siehe auch Diskussion über den praktischen Einsatz von Flatlet-Basen in Abschnitt 9.2).

Wegen des großen Berechnungsaufwandes wurde aus den genannten Gründen bei der Implementierung auf eine Gouraud-Schattierung der Flatlet-Basen verzichtet.

### 6.4.3 Direkte Anzeige unter Umgehung des BRep-Standards

Eine weitere Möglichkeit ist der direkte Eingriff in die Ausgaberroutinen für die jeweilig zugrundeliegende Graphikbibliothek. So wurde eine spezielle Version der Datei `gh_ogl.cpp` geschrieben, um die Ausgabe des Ergebnisses unter OpenGL zu realisieren. Dabei wurden Informationen über das jeweilige Patch abgefragt und die entsprechenden Befehle zur Anzeige unter Umgehung der Klasse `t_cg13d` gegeben. Man verliert durch dieses Vorgehen die in der Einleitung genannte Plattform-Unabhängigkeit und den Vorteil einer sauberen Modellierung eines geschlossenen Systems.

An dieser Stelle soll darauf hingewiesen werden, daß es sich hierbei um eine rein provisorische Lösung des Ausgabeproblems handelt. Dieses kann unnötig werden, wenn es eine entsprechende Erweiterung des MRTs zuläßt; z.Zt. stellt diese Lösung aber die einzige Möglichkeit dar, dreieckige Patches anzuzeigen (siehe auch 6.4.2.2).

## 6.5 Gemeinsamkeiten und Unterschiede der implementierten Versionen

Im Verlauf dieser Arbeit wurden zwei Versionen des Programms `mrtwave` umgesetzt. Diese seien hier mit `mrtwave direct` und `mrtwave texture` bezeichnet. Welche Version vorliegt kann über den Menüpunkt `Help - About` abgefragt werden. Im folgenden soll erläutert werden, in welchen Teilen Gemeinsamkeiten und Abweichungen bestehen.

### 6.5.1 Gemeinsamkeiten

Die beiden Programme sind identisch in allen Bereichen, die nicht die Ausgabe betreffen. Dies bedeutet, daß die reine Radiosity-Berechnung keine Unterschiede aufweist.

Gemeinsamkeiten in der Ausgabe bestehen in der Wireframe-Darstellung. Die Farbwerte für die Eckpunkte aller Patches werden auf identische Weise ermittelt und angezeigt.

## 6.5.2 Unterschiede

Die Unterschiede sind allein in der Ausgabe des Ergebnisses zu finden: `mrtwave direct` benutzt die im Abschnitt 6.4.3 beschriebene direkte Ausgabe und bietet nicht die Möglichkeit, das Ergebnis über Texturen anzuzeigen. Dies hat im einzelnen die folgenden Auswirkungen:

- Im View-Menü gibt es *keine* Möglichkeit, die Radiosity-Textur einzuschalten. Man findet dort jedoch weiterhin einen Eintrag, um die Anzeige allgemeiner Texturen ein- bzw. auszuschalten.
- Unter Options-Radiosity kann man für Multiwavelets mit drei vanishing Moments die Anzahl der Farbwerte angeben, die für jedes Patch bestimmt werden sollen. Diese Angabe bezeichnet, wie auch bei den Texturen, die Anzahl der genommenen Samples in *einer* Richtung auf dem Patch, d.h. ein Eintrag von 4 bedeutet, daß ein Gitter von  $4 \times 4$  Sample-Punkten über das Patch gelegt wird.

Für Multiwavelets mit zwei vanishing Moments ist diese Angabe *nicht* nötig und wird somit auch nicht ausgewertet, denn wie bereits in Abschnitt 6.4.1.2 beschrieben, genügt für diese Art von Basis ein einfaches Auswerten der Farben in den Eckpunkten.

`mrtwave texture` nutzt die unter 6.4.2 beschriebene Ausgabe über die Berechnung von Texturen. Hieraus ergeben sich die folgenden Besonderheiten:

- Im View-Menü findet sich der Eintrag Radiosity-Textures, über den die genannte Textur ein- und ausgeschaltet werden kann.
- Unter Options-Radiosity kann nun für Multiwavelets nicht mehr die Anzahl der Sample-Punkte angegeben werden. Die Ausgabe für alle Multiwavelets wird über die Anzahl der Texturpixels für jedes Patch bestimmt (siehe Erläuterungen über Multiwavelets in Abschnitt 6.4.2).
- Wird die Textur bei gewählter Wireframe-Darstellung angezeigt, so ist nur das Patch sichtbar, welches der Wurzel der Hierarchie entspricht. Man kann also nicht sehen, wo Unterteilungen dieses Patches vorgenommen wurden. Um diese Information zu bekommen, muß die Anzeige der Radiosity-Textur erst wieder ausgeschaltet werden.
- Da für gekrümmte Objekte und Dreiecke keine Texturen berechnet werden können, werden diese durch ihre Approximationen in der Szene dargestellt.

# Kapitel 7

## Anbindung an den MRT

Wie wurde nun konkret die Wavelet Radiosity in den MRT eingebaut? Wie wurden bestehende Klassen modifiziert und wo wurden neue Klassen eingehängt? Diese Fragen sollen im folgenden beantwortet werden.

### 7.1 Ansatzpunkte

Da Wavelet Radiosity mit Haar-Basis konzeptuell mit hierarchischer Radiosity übereinstimmt und bereits eine Implementierung für HR vorhanden war, lag es nahe, hier den Ansatzpunkt zu wählen. Für eine genaue Beschreibung der HR-Implementierung siehe [3].

#### 7.1.1 Allgemeiner Aufbau

Aufbauend auf den Klassen `t_HLink`, `t_IllumSceneHR` und `t_HradPatch` sieht der allgemeine Aufbau wie folgt aus (die Pfeile geben dabei die Ableitungsrichtung an):



#### 7.1.2 Geänderte Klassen

Die gegebenen Klassen wurden so umgestaltet, daß sie als abstrakte Basisklassen dienen können. `t_HradPatch` beispielsweise stellt nun die Funktionalität eines abstrakten Wavelet-Patches zur Verfügung. Die Klasse enthält rein virtuelle Funktionen, die von den abgeleiteten Klassen implementiert werden müssen. Diese abgeleiteten Klassen (z.B. `t_Mult2`) ergänzen dann die Basisklasse zu einem funktionstüchtigen und erzeugbaren Wavelet-Patch.

### 7.1.2.1 Der Link `t_HLink`

Die Link-Klasse kann weiterhin von allen Wavelet-Basen benutzt werden, da der im Link enthaltene Formfaktor durch einen allgemeineren Interaktionsfaktor ersetzt wurde (siehe [7.1.3.3 Der Interaktionsfaktor](#)).

Es ist also *nicht* nötig für jede neue Wavelet-Basis eine Link-Klasse von `t_HLink` abzuleiten. Bis zu drei vanishing Moments werden bereits abgedeckt. Rein theoretisch gilt dies auch für Basen mit mehr als drei vanishing Moments. Da diese hier aber nicht verwendet werden, liegen keine praktischen Ergebnisse vor.

### 7.1.2.2 Die Szene `t_IllumSceneHR`

Die Szene enthält nun drei neue virtuelle Funktionen, die u.a. für die Erzeugung verschiedener Instanzen zuständig sind. Durch Überschreiben dieser Funktionen wird sichergestellt, daß immer der richtige Typ der jeweiligen Instanz erzeugt wird. Im einzelnen sind dies:

- `t_HradPatch* createPatch(t_RadFace* f, const t_ObjectPtr obj)`  
erzeugt einen Radiosity-Patch. Parameter sind die zugehörige BRep-Face und das Objekt. Rückgabewert ist ein Zeiger auf das erzeugte Patch.
- `t_RadiosityTexture* createRadiosityTexture()`  
liefert den Zeiger auf eine neu erzeugte, der Basis angemessene Textur (siehe [6.4.2](#)).
- `void getRadRange(t_Real& cmin, t_Real& cmax,  
t_Real& avgLuminance, bool includeLights)`  
berechnet die minimale sowie die maximale Farbe, die in der Szene enthalten ist. Diese Werte sind notwendig, um alle vorhandenen Farben vor der Anzeige in einen darstellbaren Bereich zu skalieren. Die Farbwerte sind dabei von der zugrundeliegenden Basis abhängig, da die Koeffizienten auf unterschiedliche Weise kombiniert werden müssen, um den gesuchten Farbwert zu erhalten (siehe hierzu auch die Abschnitte [5.4.1.2 Flatlets](#) und [5.4.1.3 Multiwavelets](#)).  
Eine Speicherung der ermittelten Farbwerte erfolgt mit Hilfe der Klasse `t_Statistics` (siehe [7.1.3.2](#)).

### 7.1.2.3 Das Patch `t_HradPatch`

Auch diese Klasse enthält zusätzlich zu den grundlegenden Funktionen, die bereits in der HR-Implementierung vorhanden waren, eine Reihe von virtuellen Funktionen. Diese werden hier abstrakt beschrieben. Die wirkliche Implementierung übernimmt die abgeleitete Patch-Klasse.

- `void gatherRad()`  
sammelt die Energie über alle Links, die auf dieses Patch zeigen (siehe hierzu die Ausführungen unter [5.4.4](#)).
- `t_Real Error(t_HLink* l)`  
`t_Real BError(t_HLink* l)`  
berechnen für den übergebenen Link `l` den durch ihn erzeugten Fehler. `Error` wird dabei für ein F-Refine der Links benutzt, `BError` für ein BF-Refine, d.h. `Error` beachtet nicht die Energie, die wirklich zwischen den Patches ausgetauscht wird, sondern berechnet den Fehler allein aufgrund der geometrischen Lage der beteiligten Patches.

- `void preparePatch()`  
führt eine Vorbereitung und Initialisierung der Patches auf der höchsten Hierarchiestufe durch: die Koeffizienten des Patches werden mit Ausgangswerten für die Berechnung gefüllt.
- `void updateApproxB()`  
übernimmt die Berechnung eines approximativen Farbwertes. Dieser wird in der BRep-Face gespeichert. Der Farbwert wird gepflegt, um ihn eventuell während der laufenden Berechnung als Zwischenergebnis anzuzeigen. Um der Frage nachzugehen, warum es sich hierbei nur um eine Approximation und nicht um das exakte Ergebnis handelt, siehe Abschnitt 6.4.
- `void add()`  
konvertiert gesammelte Energie in Radiosity-Werte. Um den Verlauf des Sammelns von Energie nachhalten zu können, verfügt jedes Patch nicht nur über einen Vektor von Koeffizienten, sondern darüber hinaus noch über einen Hilfsvektor. Der erste,  $B_s$ , beinhaltet den Radiosity-Wert, den das Patch nach außen repräsentiert, also die eigentlichen Koeffizienten, von denen in vorherigen Abschnitten die Rede war (das  $s$  steht hierbei für „shooting“). Der zweite,  $B_g$ , repräsentiert die in einem Schritt gesammelte Energie (das  $g$  steht für „gathering“). Diese gesammelte Energie stellt allein keinen Radiosity-Wert dar, eine Umrechnung wird durch diese Funktion `add` realisiert: Die gesammelte Energie  $B_g$  wird mit der Oberflächenreflektivität  $\rho$  multipliziert. Das Ergebnis wird zu  $B_s$  addiert.  
Gleichzeitig wird an dieser Stelle festgestellt, wie groß die Veränderung des Radiosity-Betrages war. Diese Veränderung ist ein Maß dafür, ob ein Equilibrium, eine ausgeglichene Verteilung der Energie in der Szene, erreicht wurde.
- `void push()`  
bestimmt die Koeffizienten  $B_s$ , die dieses Patches an vorhandene Kinder weitergibt.
- `void pull()`  
beschreibt die umgekehrte Richtung von `push`.
- `void calcInteractionFactors(t_IllumSceneHRPtr iSceneHr)`  
berechnet für alle gathering Links dieses Patches die noch nicht vorhandenen Interaktionsfaktoren. Die Szene wird übergeben, weil die Berechnung auch das Schießen von Strahlen beinhaltet. Dem Patch muß also bekannt sein, wo sich andere Objekte der Szene befinden.
- `bool radiator()`  
gibt an, ob es sich bei vorliegendem Patch um einen Radiator handelt, also ein Patch, das von sich aus Energie abstrahlt. Diese Information ist notwendig, weil Links zwischen Patches, die sehr weit auseinander liegen, nur dann gebildet werden, wenn eines dieser Patches Energie abstrahlt. Ansonsten wird die Bildung der Verbindung aus Kostengründen unterdrückt, da man annehmen kann, daß die ausgetauschte Energie durch die große Entfernung so gering ist, daß sie im Bild keine signifikante Änderung herbeiführt. Über den Einsatz dieser Funktionalität werden in Abschnitt 7.1.3.1 unter `maxDistSqr` weitere Ausführungen gemacht.

### 7.1.3 Neue Klassen

Die neu implementierten Klassen bestehen hauptsächlich aus den Ableitungen der oben erläuterten Basis-Klassen. Ihnen kommt die Aufgabe zu, die oben beschriebenen, abstrakten Aufgaben mit konkreten Anweisungen zu füllen.

Eine implementierte Basis besteht aus zwei verschiedenen Klassen: der Szene und dem Patch. Für ein exemplarische implementierte Wavelet-Basis siehe Anhang A.

### 7.1.3.1 Optionenklasse `t_WaveletOptions`

Diese Klasse beinhaltet sämtliche Optionen, die für eine Wavelet Radiosity-Berechnung gesetzt werden können. Dabei wurde Wert darauf gelegt, möglichst viele Werte beeinflussen zu können. Um jedoch nicht zu stark zu verwirren wurden sinnvolle Standardbelegungen vorgegeben, die für die meisten Fälle angemessen sein sollten. Es folgt eine Aufstellung der Datenfelder. Für jedes dieser Felder stehen Funktionen zum Setzen und Abfragen der Werte zur Verfügung. Die Angaben in eckigen Klammern bezeichnen jeweils den Standardwert.

- `t_Real gamma()`  
Gammakorrektur. [1.6]
- `t_PrevType preview()`  
gibt an, wie das Zwischenergebnis angezeigt wird (nur `mrthr`). Mögliche Werte sind `NoPreview`, `Preview` oder `PreviewWithPause`. [Preview]
- `bool simpleReconstruction()`  
gibt an, ob bei der Berechnung der Farbwerte für die Ecken einer Face der einfache Durchschnitt (`true`), oder die gewichtete Variante (`false`) benutzt werden soll. Bei einer Gewichtung wird die Stellung der Normalenvektoren der beteiligten Faces berücksichtigt. [`false`]
- `bool roughestApproximation()`  
bestimmt, ob die Berechnung mit der größtmöglichen Einstellung der Approximationsfeinheit gestartet werden soll (`true`). Als Alternative kann der vom Benutzer gewählte Qualitätswert übernommen werden (`false`). Empfohlen wird hier die Einstellung `true`, da das Radiosity-Verfahren am besten entscheiden kann, an welchen Stellen Unterteilungen vorgenommen werden müssen. Eine feinere Einstellung der Approximation bewirkt jedoch, daß alle Patches gekrümmter Objekte unterteilt werden. Die Anzahl der Ausgangs-Patches wird durch die empfohlene Einstellung auf ein Minimum reduziert, was einen wesentlichen Geschwindigkeitsgewinn bringt (siehe Abschnitt 5.2 und [3] für Hintergründe). [`true`]
- `int nSamplePointsM3()`  
zeigt nur Auswirkung, wenn sie im Zusammenhang mit `Mult3`-Basen bei direkter Ausgabe des Ergebnisses verwendet wird. Der hier eingetragene Wert bezeichnet die Anzahl der Punkte, die für die Anzeige eines `Mult3`-Patches gesampelt werden. Ein Wert von  $x$  entspricht dabei einem  $x \times x$ -Gitter, das über das Patch gelegt wird. Zwischen diesen gesampelten Werten wird linear interpoliert (siehe auch 6.4.3).  
  
Diese Vorgehensweise ist nötig, weil ein quadratischer Verlauf der Radiosity-Funktion, wie er bei der `Mult3`-Basis auftreten kann, nicht direkt darstellbar ist. [4]
- `int textureSize()`  
wird nur bei `Multiwavelets` verwendet und gibt die Größe der zu benutzenden Textur an (siehe Abschnitt 6.4.2). [16]
- `int useSimpleFF()`  
wird für `Flatlet`-Basen eingesetzt. Hier kann angegeben werden, ab welcher Hierarchiestufe ein vereinfachter Formfaktoralgorithmus verwendet werden soll: Für alle Stufen, deren Tiefe größer als der hier gesetzte Wert ist, werden nur ein Viertel der ursprünglichen Testpunkte zur Formfaktorberechnung erzeugt. Es kann somit vermieden werden, daß für sehr kleine Patches eine zu große Anzahl von Strahlen geschossen wird (siehe 7.2.2). [5]

- `bool useVisOpt()`  
gibt an, ob die bei [12] beschriebene Optimierung zur Berechnung der Sichtbarkeit eingesetzt werden soll (`true`): Wurde für einen Link der Sichtbarkeitswert von eins (d.h. die Patches sind voll sichtbar) ermittelt, so wird bei einer Unterteilung eines der beteiligten Patches für die neu entstehenden Links keine neue Sichtbarkeitsberechnung durchgeführt. Es wird davon ausgegangen, daß diese neuen Sub-Patches ebenfalls voll sichtbar sind (siehe Sichtbarkeitsproblematik in Abschnitt 5.4.3). Ein Einsatz dieser Optimierung kann dazu führen, daß kleine Details beim Schattenwurf verlorengehen. [`false`]
- `bool adjustVertices()`  
bewegt bei gekrümmten Objekten die neu erzeugten Knoten auf die Oberfläche des Objektes (`true`). Dieses Vorgehen wird erst möglich durch die in Abschnitt 6.2 beschriebenen Eigenschaften der 3D-Objekte. [`true`]
- `t_Real error()`  
legt die Fehlerschranke für das Einsammeln von Energie fest. Wurde bei einem Gathering-Schritt kein Energiebetrag verteilt, der oberhalb dieser Schranke liegt, so wird das Verfahren beendet. [`0.001`]
- `t_Real epsFF()`  
gibt die Grenze an, ab der ein bestehender Link bei Einsatz von F-Refinement nicht weiter unterteilt wird. Die Angabe des Wertes kann über die Szenenbeschreibung mit Hilfe der Variable FEPS erfolgen. [`0.001`]
- `t_Real epsBF()`  
hat die gleiche Funktion wie `t_Real epsFF()`, kommt aber bei BF-Refinement zum Einsatz. Die Angabe des Wertes kann über die Szenenbeschreibung mit Hilfe der Variable BFEPS erfolgen. [`0.001`]
- `t_Real epsArea()`  
gibt den Grenzwert für die Fläche eines Patches an. Die Unterteilung eines Patches wird auf jeden Fall verhindert, wenn die Fläche unterhalb dieser Grenze liegt. Eine Anpassung an die jeweilige Szene sollte auf jeden Fall vorgenommen werden. Dies kann mit Hilfe der Variable AEPS in der Szenenbeschreibung erfolgen. [`0.1`]
- `t_BasisType basis()`  
wählt die Wavelet-Basis aus, die der gesamten Berechnung zugrunde liegt. Die Auswahlmöglichkeiten hierbei sind `Haar`, `Flatlet2`, `Flatlet3`, `Multiwavelet2` oder `Multiwavelet3`. [`Haar`]
- `t_Real maxDistSqr()`  
Patches, die weit auseinander liegen, werden nicht durch einen Link verbunden. `maxDistSqr` gibt diese Entfernung zum Quadrat an (Ausnahme hierbei sind Lichtquellen, denn diese können trotz großer Entfernung wichtig für den Verlauf der Radiosity-Funktion sein).  
  
Diese Option ist dann sinnvoll anzuwenden, wenn man nur einen Teilbereich einer größeren Szene rendern möchte und somit daran interessiert ist, eine große Anzahl von Objekten a priori aus der Betrachtung auszuschliessen. Aus diesem Grund wurde als Standard ein sehr großer Wert gewählt. [`1e8`]

- `int maxGatherSteps()`  
gibt die maximale Anzahl der Gathering-Schritte an. Die Notwendigkeit für diese absolute Schranke wurde unter [5.4.5](#) erklärt. [4]
- `int maxIterations()`  
legt die maximale Anzahl der Verfeinerungsschritte fest. Die Erklärung für diese Option findet sich unter [5.4.5](#). [5]  
Anmerkung: Im Optionendialog der Implementierung wird man dieses Feld nicht finden. Stattdessen kann man dort die Anzahl der gewünschten Refine Schritte angeben. Es gilt  
$$\text{Anzahl der max. Refine Schritte} = \text{max. Iterationen} - 1,$$
da der erste Iterationsschritt aus einem Gathering besteht.
- `bool avgVertex()`  
schaltet die Durchschnittsbildung für Farbwerte der Eckpunkte ein (`true`) bzw. aus (`false`). Siehe Abschnitt [6.4.1.2](#) für eine vollständige Beschreibung. Diese Option ist nur bei Multiwavelets wirksam. [`false`]

### 7.1.3.2 Statistikklassse `t_Statistics`

Die Klasse `t_Statistics` enthält eine Reihe von Datenfeldern, die über die ausgeführte Berechnung Auskunft geben:

- `t_Timer* meshingTime()`  
enthält die für das Meshing benötigte Zeit.
- `t_Timer* intFactorTime()`  
enthält die für die Berechnung der Interaktionsfaktoren benötigte Zeit.
- `t_Timer* totalTime()`  
enthält die gesamte vom Radiosity-Verfahren benötigte Zeit.
- `unsigned nPolygons()`  
gibt die Anzahl der Polygone in der Szene an.
- `unsigned nPatches()`  
gibt die Anzahl der Patches in der Szene an.
- `t_Real areaMin()`  
`t_Real areaMax()`  
gibt die minimale bzw. maximale Fläche aller Patches in der Szene an.
- `t_Real maxColor()`  
`t_Real minColor()`  
enthält die minimale bzw. maximale Farbe aller Patches in der Szene.
- `int nLinks()`  
Gibt die Anzahl aller Links in der Szene an.

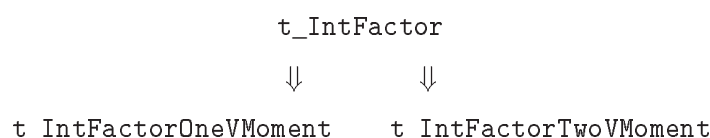
- `int newLinks()`  
`void incNewLinks()`  
`void decNewLinks()`

enthält die Anzahl aller im aktuellen Iterationsschritt neu erzeugten Links. Für diese wurde noch kein Interaktionsfaktor berechnet. Die letzten beiden Funktionen stehen zum Inkrementieren bzw. Dekrementieren des Wertes zur Verfügung.

- `memoryUsed()`  
Größe *eines* Patches in Bytes.

### 7.1.3.3 Der Interaktionsfaktor

Der Interaktionsfaktor ist Teil eines Radiosity-Links. Je nach gewählter Basis kann es sich dabei um eine Gleitkommazahl oder eine Matrix handeln. Diese beiden Fällen ließen sich in einer allgemeinen Matrizenklasse zusammenfassen. Um aber nicht für den ersten Fall einen zu großen Aufwand zu treiben, wurde folgende Klassenhierarchie eingerichtet:



Bei `t_IntFactor` handelt es sich um eine abstrakte Klasse. `t_IntFactorOneVMoment` ist für alle Basen mit einem vanishing Moment gedacht und enthält als einziges Datenelement einen Gleitkommawert mit den zugehörigen Zugriffsfunktionen. `t_IntFactorTwoVMoment` wird bei allen Basen eingesetzt, die zwei *oder mehr* vanishing Moments aufweisen. Das einzige Datenelement besteht hier aus einer Matrix allgemeiner Größe.

Durch diese Vorgehensweise wird erreicht, daß alle Wavelet-Basen mit der gleichen Link Klasse arbeiten können. Es ist somit nicht nötig, für jede Basis eine spezielle Klasse abzuleiten, da das einzige variable Elemente bei verschiedenen Basen der Interaktionsfaktor ist.

## 7.2 Implementierung verschiedener Wavelet-Basen

Wie bereits weiter oben erwähnt, besteht die Implementierung einer Wavelet-Basis darin, die virtuellen Methoden der Basisklassen `t_IllumSceneHR` und `t_HradPatch` zu überschreiben. Weiterhin muß jedes abgeleitete Patch seine eigenen Felder und Funktionen für die Speicherung und den Zugriff auf die Koeffizienten bereitstellen. Die Links werden durch die zur Verfügung gestellte Klasse `t_HLink` vollständig abgedeckt.

Im Rahmen dieser Arbeit wurden die Haar-Basis, Flatlets und Multiwavelets mit jeweils zwei bzw. drei vanishing Moments implementiert. An dieser Stelle soll nun auf besondere Problemstellen hingewiesen werden. Teilweise haben diese Stellen schon Erwähnung in früheren Abschnitten gefunden. Sollte dies der Fall sein, so wird ein entsprechender Verweis angegeben.

Ein vollständiges Beispiel einer Implementierung findet sich im Anhang A.

### 7.2.1 Haar-Basis

Der Interaktionsfaktor eines Haar-Links ist identisch mit einem Formfaktor. An dieser Stelle werden also weiterhin die von der Klasse `t_FormFactor` zur Verfügung gestellten Methoden aufgerufen.

Weitere Besonderheiten oder Probleme sind nicht zu nennen.

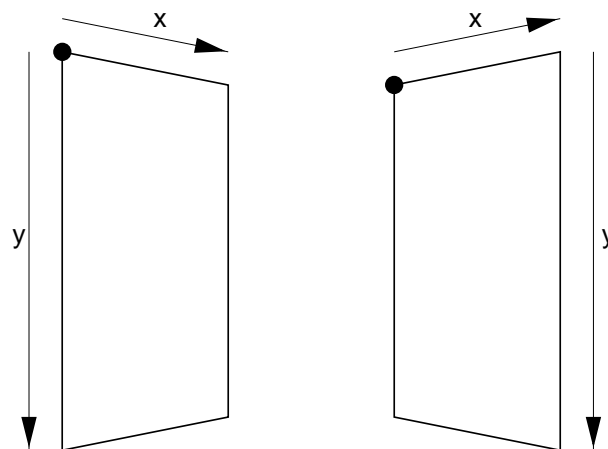


Abbildung 7.1: Beispiel für eine gegenseitige Lage zweier Patches, die zu negativen Elementen in der Interaktionsfaktormatrix führt. Die Patches sind sich zugewandt. Der Ursprung des lokalen Koordinatensystems ist durch einen Kreis markiert.

## 7.2.2 Flatlets

Der Interaktionsfaktor eines Flatlet-Links besteht aus einer Matrix von Formfaktoren. Um diese berechnen zu können, wurde die Klasse `t_FormFactor` um die Methode `calcSubToSubFormFactor` erweitert, die es erlaubt Sub-Formfaktoren, d.h. Formfaktoren zwischen Sub-Patches, zu berechnen. Diese Sub-Formfaktoren werden analog zu Standard-Formfaktoren bestimmt:

Um den Ziel- und Quellpunkt werden kreisförmig, mit Hilfe einer Poisson-Verteilung, Testpunkte verteilt. Zwischen Paaren dieser Testpunkte wird jeweils ein point-to-disc Formfaktor berechnet, die Sichtbarkeit geht hierbei direkt in die Berechnung ein. Aus den so gewonnenen Werten wird der Formfaktor berechnet.

Für sehr kleine Patches bedeutet dieses Vorgehen, daß eventuell zu viel Arbeit aufgewendet wird. Ein gesamtes Patch wird, wie bereits beschrieben, noch einmal intern in vier bzw. neun Patches geteilt. Deshalb ist es möglich, über die Optionen eine Stufe der Hierarchie anzugeben, ab der ein gröberes Verfahren eingesetzt wird (siehe `useSimpleFF` in Abschnitt 7.1.3.1). Dabei werden weniger Testpunkte gewählt und bearbeitet. Die Anzahl wird auf ein Viertel der ursprünglichen Anzahl gesenkt. Dieses vereinfachte Verfahren wird ebenfalls eingesetzt, wenn sich die Fläche des Patches der vorgegebenen unteren Schranke `epsArea` nähert: Ist die Fläche kleiner als das fünffache dieser Grenze werden ebenfalls nur ein Viertel der Testpunkte generiert und ausgewertet.

## 7.2.3 Multiwavelets

### 7.2.3.1 Negative Interaktionsfaktorelemente

Für die Multiwavelet-Basen werden die Interaktionsfaktoren wie unter 5.4.4 beschrieben berechnet. Dabei ist zu beachten, daß die Einträge der Matrix auch kleiner als Null werden können. Abbildung 7.1 zeigt ein Beispiel mit zwei sich zugewandten Patches, für die negative Werte auftreten. Vom Ursprung des Patches (Kreismarkierung) aus gesehen verlaufen die  $x$ -Funktionen auf beiden Patches in entgegengesetzte Richtungen. Das Interaktionsfaktorelement, welches die beiden zugehörigen Koeffizienten in Beziehung zueinander setzt, fällt negativ aus. Die  $y$ -Funktionen auf beiden Patches verlaufen in die gleiche Richtung: Der Eintrag in der Matrix ist nicht negativ.

### 7.2.3.2 Negative Farbwerte

Eine extreme gegenseitige Lage von Patches kann im Anfangsstadium der Berechnung dazu führen, daß negative *Farbwerte* ermittelt werden. Dies geschieht zum Beispiel, wenn sich ein Patch in unmittelbarer Nähe eines sehr großen Patches befindet. Diese negativen Werte werden durch die Skalierung der Farbwerte auf Null gesetzt. Dadurch können Artefakte im Bild auftreten. Diese Ergebnisse treten jedoch, wie gesagt, nur im Anfangsstadium der Berechnung auf, weil es hier sein kann, daß das Größenverhältnis zwischen den einzelnen Patches noch sehr extrem ausfällt. Im Lauf der Berechnung werden die großen Patches unterteilt und die berechneten Farbwerte befinden sich wieder im positiven Bereich. Es handelt sich also um einen Sonderfall, der bei angemessener Unterteilung im Endergebnis nicht auftreten wird.

# Kapitel 8

## Ergebnisse

An dieser Stelle werden einige Bilder gezeigt, die mit Hilfe der vorgestellten Implementierung der Wavelet Radiosity berechnet wurden<sup>1</sup>. Um einen guten Vergleich der einzelnen Basen mit der Haar-Basis zu ermöglichen, wurden alle Haar-Bilder sowohl in der Flat- als auch in der Gouraud-Darstellung abgebildet. Bilder, die mit Flatlet-Basen berechnet wurden, lassen sich besser mit der Flat-Darstellung der Haar-Basis vergleichen. Multiwavelet Bilder sollten mit der Gouraud-Darstellung verglichen werden.

Weiterhin finden sich hier alle Bilder in Wireframe-Darstellung. So ist es möglich, genau zu sehen, wie oft die beteiligten Patches unterteilt wurden. Alle Bilder sind mit gesetzter `roughest-Approximation` Option berechnet worden, d.h. alle gezeigten Quadrate und Dreiecke waren bei Beginn des Verfahrens nicht unterteilt.

Das erste Bild (Abbildung 8.1) zeigt die realisierte graphische Benutzeroberfläche, die mit Hilfe der Bibliothek WxWin realisiert wurde. WxWin bietet den Vorteil, auf unterschiedlichen Plattformen lauffähig zu sein. So wurde neben der Windows- auch eine Unix-Version ohne großen Mehraufwand umgesetzt. Die Radiosity-Berechnung, Optionen und Anzeige lassen sich interaktiv über Menüs, Tastatur- und Mauskommandos steuern. Die Abbildung zeigt ebenfalls den geöffneten Optionen-Dialog, mit dessen Hilfe die Parameter der Berechnung angegeben werden können. Die im Hauptfenster dargestellte Szene wurde mit Hilfe der Haar-Basis berechnet.

Die zweite Serie von Bildern (Abbildungen 8.2 und 8.3) zeigt eine einfache Szene, die aus einem Quadrat besteht, das im rechten Winkel von einer Lichtquelle bestrahlt wird. Diese Szene wurde unter Einsatz verschiedener Basen (Haar, Flat2, Flat3, Mult2 und Mult3) berechnet. Dabei wurden für die Basis mit einem vanishing Moment vier Unterteilungsschritte, für Basen mit zwei vanishing Moments drei und für Basen mit drei vanishing Moments nur zwei Unterteilungsschritte durchgeführt. Abbildung 8.2 zeigt das Ergebnis dieser Unterteilung in Wireframe-Darstellung. In Abbildung 8.3 werden zu vergleichende Bilder in jeweils einer Spalte präsentiert. Dabei nimmt die Anzahl der vanishing Moments von oben nach unten zu: So findet sich in der ersten Spalte das Haar-Ergebnis in Flat-Darstellung sowie das Flat2- und Flat3-Ergebnis (von oben nach unten). Die zweite Spalte zeigt das Haar-Ergebnis in Gouraud-Darstellung sowie das Mult2- und Mult3-Ergebnis (von oben nach unten). Zur Darstellung der Radiosity-Funktion kam bei allen Patches eine Textur zum Einsatz. Für Multiwavelet-Basen wurde hierfür eine Texturgröße von 16 Pixeln pro Patch gewählt.

Vergleicht man die Bilder jeder Spalte mit dem Haar-Ergebnis, so läßt sich feststellen, daß diese von vergleichbarer oder besserer Qualität sind. Dies trifft zu, obwohl das Ausgangs-Patch bei zunehmender Anzahl von vanishing Moments immer weniger stark verfeinert werden mußte.

Besondere Beachtung verdient hier der glatte Farbverlauf, der mit Hilfe der Mult3-Basis erzielt wurde. Vergleicht man das Bild mit seiner Wireframe-Darstellung, so kann man gut erkennen, daß hier nur

---

<sup>1</sup>Da die Qualität der Bilder sehr unter der Druckausgabe gelitten hat, verweise ich an dieser Stelle auf die online-Version der Arbeit. Diese steht unter <http://titan.informatik.uni-bonn.de/~struck/diplom> zur Verfügung.

sehr grob unterteilt wurde. Der darstellbare quadratische Verlauf der Radiosity-Funktion kommt hierbei zum Tragen.

Die dritte Serie von Bildern (Abbildung 8.4) zeigt die Vorteile der Basen mit mehr als einem vanishing Moment bei der Darstellung von Schatten. Man kann gut erkennen, daß die Schattengrenze der Haar-Basis bei der Darstellung mit Hilfe einer Gouraud-Schattierung nicht klar zu erkennen ist (obere Zeile). Durch die Durchschnittsbildung mit den Farbwerten der Nachbar-Patches „verläuft“ der Schatten sehr stark. Nicht so bei den Multiwaveletbildern (Mult2 in der Mitte und Mult3 in der unteren Zeile): Durch die exakte Darstellung des Ergebnisses und die Nichtbetrachtung der Nachbarn ist die Schattengrenze gut zu erkennen. Das schattenwerfende Dreieck wurde mit Absicht leicht gegenüber den Patch-Grenzen des Vierecks gedreht, um die gute Darstellung des Beleuchtungswechsels auch in dieser ungünstigen Situation herauszustellen. Dies zeigt sich in der Anzahl der Verfeinerungsschritte, die nötig waren, um das gezeigte Ergebnis zu erzielen: Die Patches der Haar-Szene wurden 15mal unterteilt, die der Multiwaveletszenen nur neun- bzw. achtmal.

Weiterhin interessant an dieser Serie von Bildern ist das Vorhandensein eines Dreiecks, welches zeigt, daß das Verfahren auch mit dieser Form des Patches funktioniert (für weitere Informationen über Dreiecke in Wavelet Radiosity siehe Abschnitt 5.5). Um das Ergebnis auch für das Dreieck exakt darstellen zu können, wurde hier eine direkte Ausgabe ohne Anwendung von Texturen verwendet (siehe Abschnitt 6.4.3).

Eine Zusammenstellung der Ergebnisse findet sich in folgenden Tabellen <sup>2</sup>:

Beispiel 1 (Einfache Bestrahlung)

Basis	Verfeinerungen	Zeit (min:sek:hsek)	Patch-Größe (bytes)	Anzahl der Patches
Haar	4	0:01.314	216	116
Mult2	3	0:00.664	360	59
Mult3	2	0:00.600	600	17
Flat2	3	0:03.209	360	65
Flat3	2	0:03.854	600	17

Beispiel 2 (Blockierendes Dreieck)

Basis	Verfeinerungen	Zeit (min:sek:hsek)	Patch-Größe (bytes)	Anzahl der Patches
Haar	15	5:42.913	216	13.593
Mult2	9	1:29.352	360	3.216
Mult3	8	1:02:021	600	1.284

Der Geschwindigkeitsvorteil beim Einsatz von Multiwavelets gegenüber den Flatlets und der Haar-Basis begründet sich durch den Einsatz eines einfachen Kernelsamplings, welches, im Gegensatz zur vollen Formfaktorberechnung, mit weitaus weniger Strahlenschüssen verbunden ist (siehe 5.4.3 für weitere Erläuterungen).

Im zweiten Beispiel ist der Vorteil zusätzlich mit der weitaus geringeren Anzahl von vorgenommenen Verfeinerungsschritten zu erklären. Durch die hohe Anzahl der Patches in der Haar-Szene liegt der Speicherverbrauch sogar noch über dem der Multiwavelets. Diese Beobachtung ist aber hier auf die einfache Szene zurückzuführen, die für die Multiwavelets dazu führt, daß die Ausgangs-Patches nicht stark unterteilt werden müssen, um ein ansprechendes Ergebnis zu erhalten. Bei einer anspruchsvolleren Geometrie müssen auch für diese Basen weitere Unterteilungen vorgenommen werden. Diese führen dann zu einem wesentlich höheren Speicherbedarf.

Der Speicherverbrauch der Links darf ebenfalls nicht vernachlässigt werden. Der Übersichtlichkeit wegen wurde er jedoch in den oben gezeigten Tabellen nicht aufgeführt.

<sup>2</sup>Zur Berechnung wurde ein AMD K6, 200 MHz mit 48 MB Hauptspeicher unter Windows95 eingesetzt.

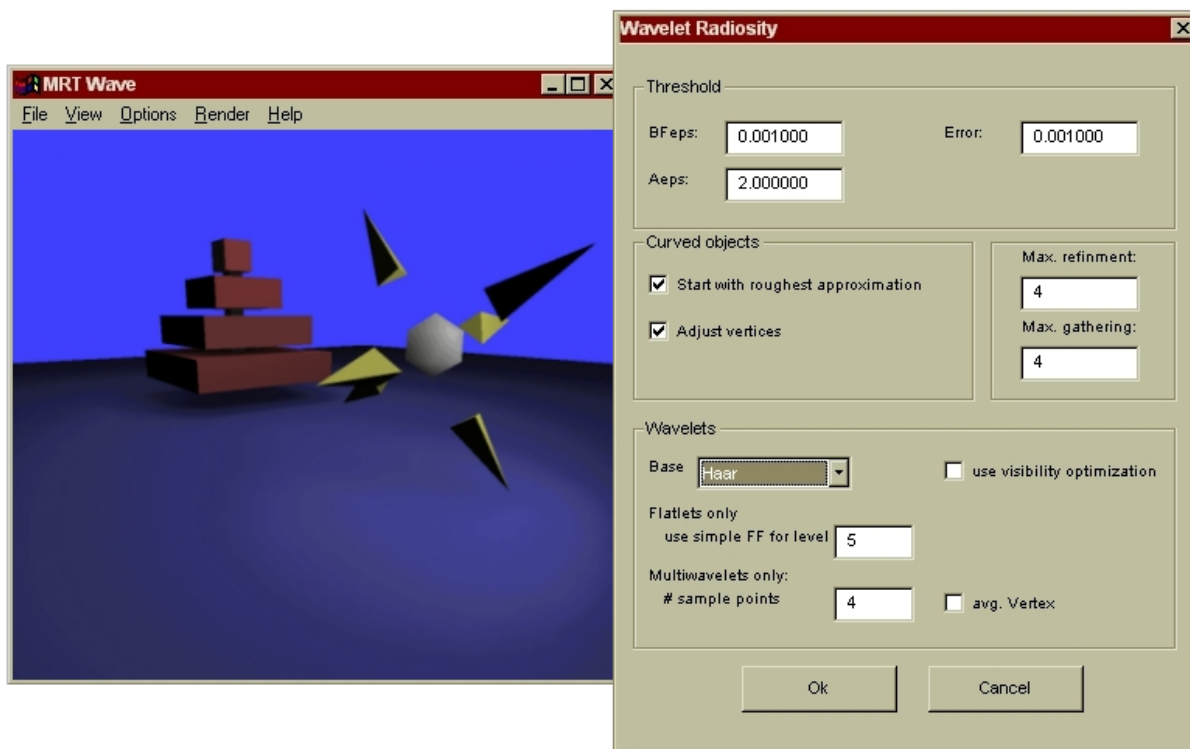


Abbildung 8.1: Die Benutzeroberfläche des MRT Wave: Alle Funktionen sind über Menüs und Tastaturkommandos zugänglich. Zu jedem Zeitpunkt der Berechnung ist die Szene im Hauptfenster sichtbar. Die Abbildung zeigt das Programm mit geöffnetem Optionendialog, über den der Zugriff auf diverse Parameter möglich ist. Die hier gezeigte Szene abstrakter geometrischer Objekte wurde mit Hilfe der Haar-Basis berechnet.

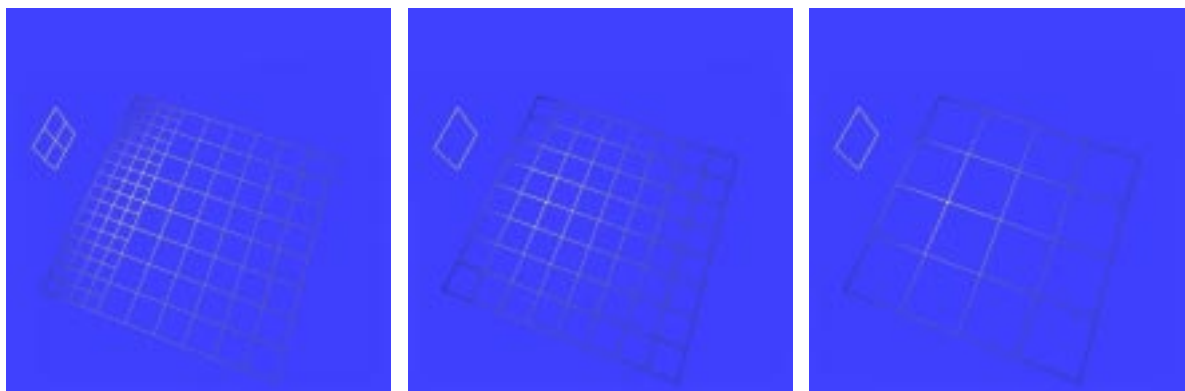


Abbildung 8.2: Wireframe-Darstellung der einfachen Beispielszene. Man erkennt die unterschiedliche Unterteilung des Ausgangs-Patches für Basen mit einem, zwei und drei vanishing Moments (von links nach rechts).

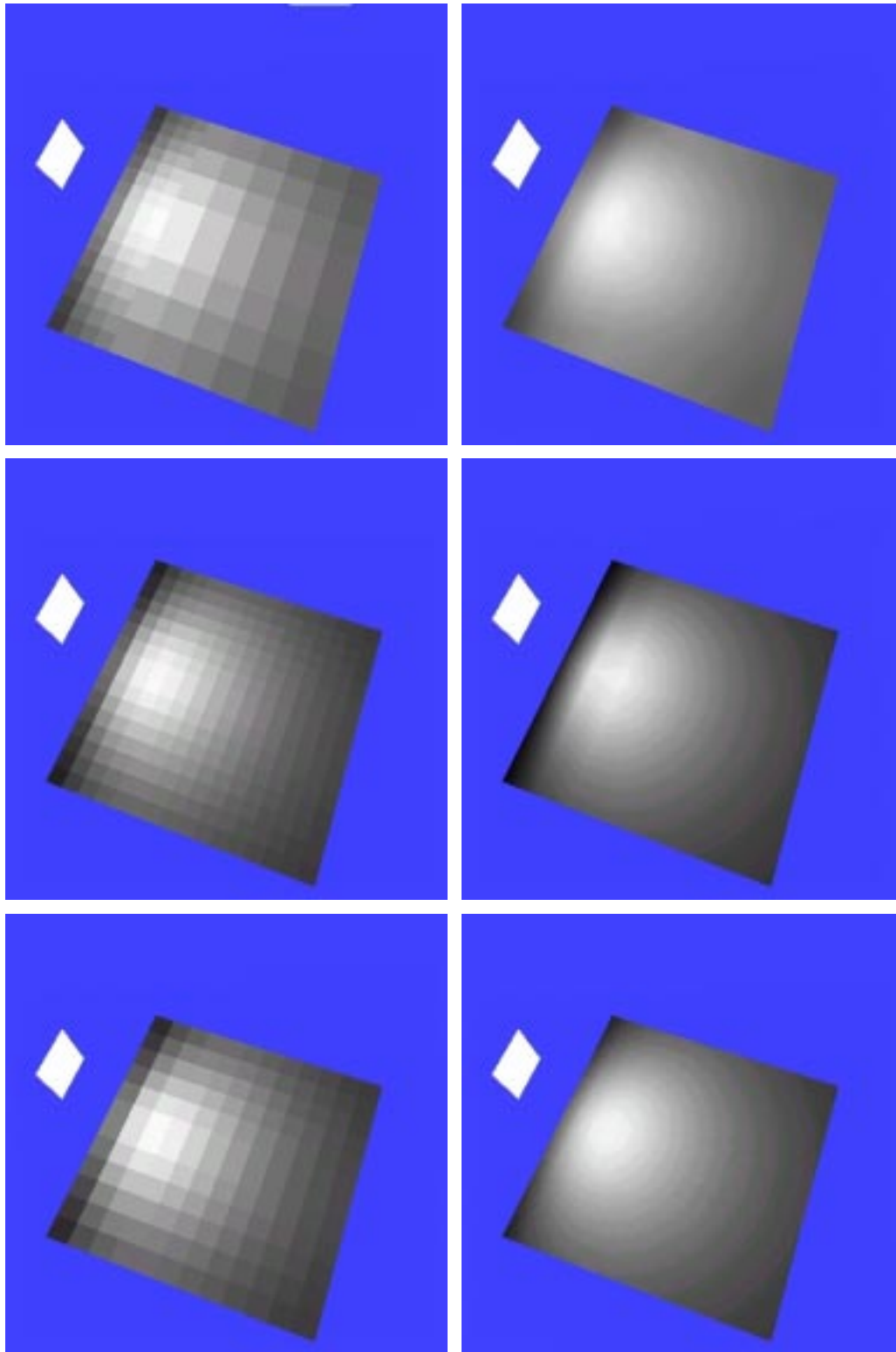


Abbildung 8.3: Vergleichbare Ergebnisse der einfachen Szene, in Spalten angeordnet: Flat-Darstellung der Haar-Basis, Flat2-Basis und Flat3-Basis (erste Spalte, von oben nach unten). Gouraud-Darstellung der Haar-Basis, Mult2-Basis und Mult3-Basis (zweite Spalte, von oben nach unten). Die Anzahl der vanishing Moments nimmt in jeder Zeile von oben nach unten zu, die Anzahl der Unterteilungen dagegen ab (siehe auch Wireframe-Darstellung in Abb. 8.2). Bis auf die Gouraud-Darstellung der Haar-Basis sind alle Darstellungen exakt, d.h. es wurde kein Glättungsschritt durchgeführt.

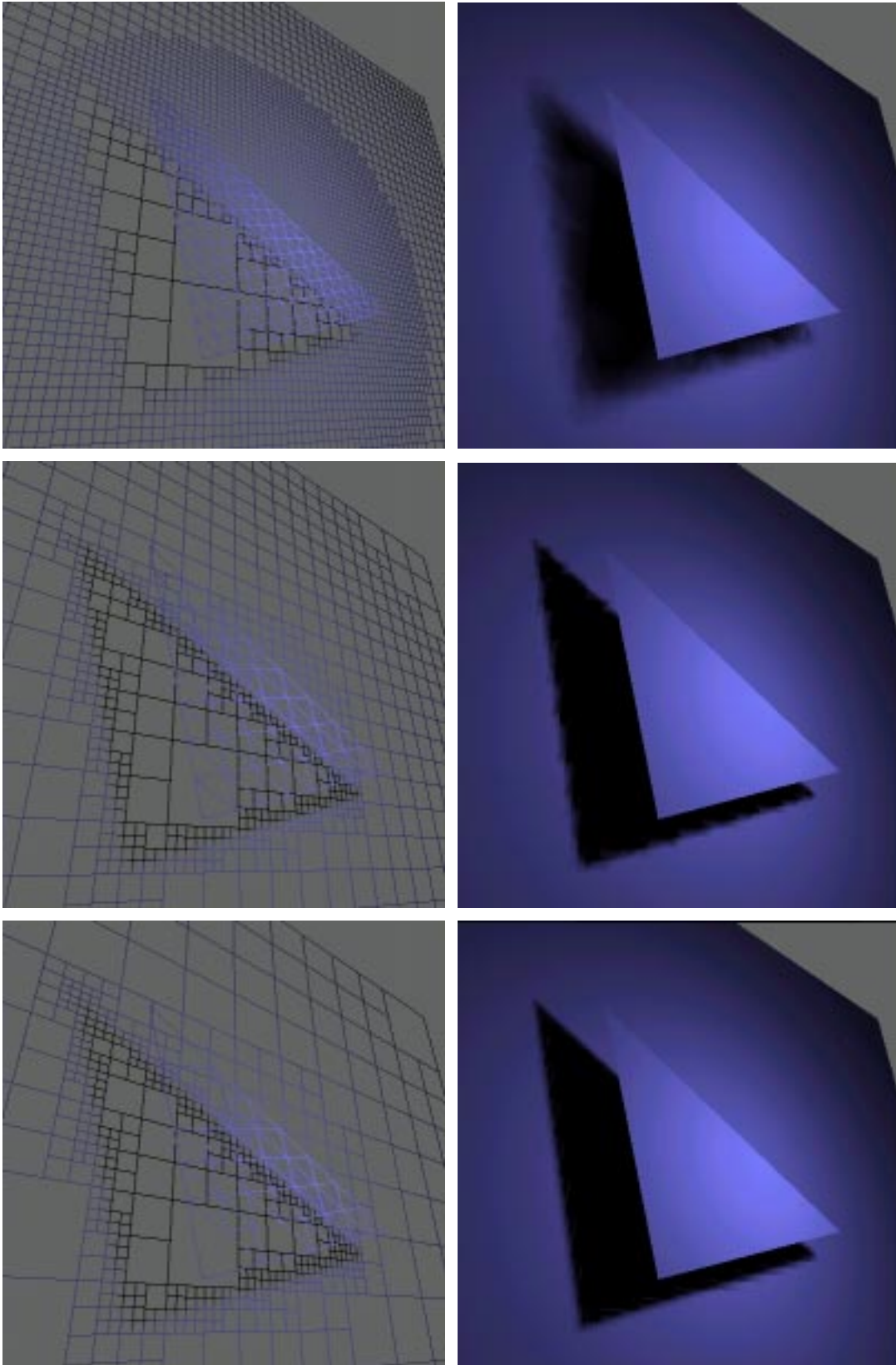


Abbildung 8.4: Schattenwurf beim Einsatz von Haar-, Mult2- und Mult3-Wavelets, dargestellt in Wireframe- und Gouraud- bzw. exakter Darstellung (von oben nach unten und links nach rechts). Man beachte die verbesserte Schattengrenze bei Zunahme der vanishing Moments.

# Kapitel 9

## Diskussion und Ausblick

### 9.1 Wavelet Radiosity: Theoretisches Konstrukt oder praktische Anwendung?

Betrachtet man den hohen Speicherbedarf des hier vorgestellten Radiosity-Verfahrens wie auch den erhöhten zeitlichen Aufwand, der für die Berechnung des Kernels anfällt, so stellt sich die Frage, ob es sich mehr um ein Gedankenexperiment oder um eine wirklich nutzbare Erweiterung des klassischen Verfahrens bzw. des hierarchischen Ansatzes handelt.

Meiner Meinung nach besteht der größte Verdienst der Wavelet Radiosity in der Vereinigung zweier anscheinend verschiedener Ansätze: HR und Galerkin Radiosity. Diese Vereinigung bringt jedoch auch einen entscheidenden Nachteil mit sich, denn man muß zu jeder Zeit darauf achten, allgemeingültig, d.h. offen für alle Basen zu bleiben. Diese Voraussetzung stellt auf theoretischer Ebene keine Einschränkung dar, auf der praktischen Ebene trifft dies aber nicht zu. So können Optimierungen, die auf niedriger Stufe angesiedelt sind, weit weg von der abstrakten Theorie, nicht ohne größere Eingriffe realisiert werden. Man bewegt sich immer im abstrakten Rahmen, der für alle Basen Gültigkeit behalten muß. Ein Verfahren, das alles kann bzw. darauf vorbereitet wird, mit Hilfe von Erweiterungen alles zu können, wird sich im allgemeinen nicht mit einem spezialisierten System messen können.

So wird die hier vorgelegte Implementierung bei Einsatz der Haar-Basis langsamer sein als die äquivalente Implementierung des HR-Ansatzes.

Dies spricht für eine auf eine einzige Basis beschränkte, spezialisierte Implementierung, wie es in der Vergangenheit mit dem HR-Ansatz bereits geschehen ist. Eine Erweiterung auf eine Basis mit einer höheren Anzahl von vanishing Moments könnte auch auf diesem Weg Umsetzung finden. Was bleibt ist der nicht unwesentlich erhöhte Speicher- und Zeitaufwand. Es ist nicht ohne weiteres einzusehen, warum man sich von vornherein eine Mehrarbeit in jedem Schritt aufbürden sollte, wenn mit der Haar-Basis eine Alternative zur Verfügung steht, die es zuläß, mit geringeren Kosten aber stärkerer Unterteilung der Patches zu einem vergleichbarem Ergebnis zu kommen.

Aus diesen Gründen ist die Frage nach der Zugehörigkeit des WR-Verfahrens zum theoretischem oder praktischen Bereich nicht eindeutig zu beantworten. Das allgemeine WR-Verfahren ist eher dem theoretischen Bereich zuzuordnen, die Frage nach dem praktischen Nutzen muß allerdings von einer speziellen Basis abhängig gemacht werden (siehe Abschnitt 9.2).

Dies soll allerdings nicht bedeuten, daß man keinen großen Nutzen aus den gemachten Überlegungen ziehen kann. Viele entscheidende Verbesserungen werden immer dann gefunden, wenn man die klassischen Pfade des Denkens verläßt und einen vollständig anderen, vielleicht auf den ersten Blick abwegigen Ansatz wählt. Von der Wärmetechnik über das N-Körper Problem bis hin zu Wavelets ist es ein langer Weg, aber er konnte fortschreitend begangen werden und so wurden immer wieder Verbesserungen erreicht. Man darf also auch in Zukunft von der Wavelet Radiosity neue Impulse für Algorithmen erwarten.

Zusammenfassend läßt sich sagen, daß der Vorteil einer allgemeinen Wavelet Radiosity nicht in einer direkten allgemeinen Umsetzung liegt, sondern eher in den Einflüssen auf Teilgebiete, die in ihr enthalten sind.

## 9.2 Welche Basen sind für einen praktischen Einsatz geeignet?

Die Eignung der Haar-Basis muß an dieser Stelle nicht diskutiert werden. Sie hat sich bereits seit langem in vielen Anwendungen bewährt. Wie sieht es aber mit Flatlets aus? Gerade im Bereich der gekrümmten Objekte ist diese Familie von Basisfunktionen kaum geeignet, ein schnelles Ergebnis zu liefern, denn der Vorteil der Haar-Patches liegt hier auf der Hand: Bei einer Unterteilung können die neu entstehenden Eckpunkte der Sub-Patches auf die Oberfläche des gekrümmten Objektes bewegt werden. Man erhält eine bessere Approximation des Ausgangsobjektes. Bei Flatlets ist dies leider nicht möglich, die Koeffizienten unterstützen nur einzelne Bereich eines *planaren* Patches. So kann z.B. der Mittelpunkt eines Flat2-Patches nicht bewegt werden.

Multiwavelets stellen eine andere Form der Alternative dar: Durch die Möglichkeit, die Farbwerte direkt in den Eckpunkten des Patches zu ermitteln, kann man vor allem in Übergangsregionen zwischen Schatten- und beleuchteten Partien Gewinne erzielen, in denen die Haar-Basis wegen der Durchschnittsbildung mit benachbarten Patches das Ergebnis stark verwischt (siehe hierzu auch Abbildung 8.4). Der entscheidende Nachteil hier ist der weitaus höhere Speicherbedarf. Zusammengenommen mit der Schwierigkeit der Anzeige für Multiwavelets mit drei vanishing Moments kann man von dieser Basis für den praktischen Gebrauch eher abraten. Anders sieht es jedoch bei Multiwavelets mit zwei vanishing Moments aus: Der erhöhte Verbrauch eines Faktors vier bei den Patches und 16 bei den Links gegenüber der Haar-Basis kann je nach Szene angemessen sein. Die Ausgabe des Ergebnisses kann dabei ebenfalls ohne Probleme erfolgen, so daß nach Meinung des Autors diese Basis eine praktische Alternative zur Haar-Basis darstellt.

## 9.3 Gibt es effizientere Speicherungsmöglichkeiten?

Wenn man einen Weg finden würde, diesen hohen Speicherverbrauch zu reduzieren, so hätte man einen großen Schritt nach vorne gemacht. Leider ist in der Literatur über solch eine Möglichkeit nichts bekannt.

Es wäre denkbar, die einzelnen Koeffizienten eines Flatlet-Patches, unter der Voraussetzung, daß sie nicht stark voneinander abweichen, zu einem einzigen Koeffizienten zusammenzufügen. Dies würde eine Mischung von Haar- und Flatlet-Patches in der Szene bedeuten. Zu diesem Zweck müßte man ein effektives Entscheidungskriterium entwickeln, das vorschreibt, ob ein „echtes“ Flatlet-Patch oder ein Haar-Patch benutzt werden soll.

Ob sich dieser zusätzliche Aufwand auszahlt, ist fraglich, denn wann würde dieser Fall eigentlich eintreten? Angenommen wir haben es mit einem Flat2-Patch zu tun: Wenn nun die vier Koeffizienten, die den direkten Wert der Funktion in ihrem Bereich angeben, vom Wert her nahe zusammenliegen, so hätte bereits vorher das Orakel entscheiden müssen, daß dieses Patch erst gar nicht entstehen darf. Der genannte Verlauf der Radiosity-Funktion kann auch mit einem größeren Patch modelliert werden. Wenn diese Situation aber trotzdem auftritt, so bedeutet dies zwangsläufig, daß noch keine Unterteilung (und somit eine Befragung des Orakels) stattgefunden hat. Daraus folgt, daß es sich bei dem betrachteten Patch um die Wurzel der zugehörigen Hierarchie handelt. Um nun für einen Spezialfall eine kleinere Speicherersparnis zu erzielen, sollte der Mehraufwand nicht betrieben werden, zumal es eine Fülle von neuen Spezialfällen zu beachten gäbe, die sich aus dem Vorhandensein von gemischten Basen in der Szene ergeben.

Das oben beschriebene Verfahren ließe sich auf das Mischen verschiedener Basisfunktionen innerhalb einer Szene verallgemeinern. Man könnte sich bemühen, für jedes Patch die Basis mit der kleinsten

Anzahl von vanishing Moments zu finden, die die Radiosity-Funktion gut modelliert (siehe Abschnitt 9.4.1).

Dieser hohe Speicherverbrauch führt dazu, daß in der Praxis meist nur die Haar-Basis eingesetzt wird, da es sinnvoller erscheint, an entscheidenden Stellen zusätzliche Unterteilungen vorzunehmen und somit den höheren Speicher- und Rechenaufwand, vor allem im Bereich der Links und der Quadratur, zu vermeiden.

## 9.4 Ausblick

### 9.4.1 Adaptive Auswahl einer Wavelet-Basis

Vom Standpunkt der hier vorgestellten Theorie ist es nicht nötig, nur eine einzige Basis in der gesamten Szene einzusetzen. Änderungen würden sich in der Berechnung des Interaktionsfaktors niederschlagen. Da dieser hier als eine Matrix beliebiger Größe implementiert wurde, sollte eine Umsetzung ohne große Änderungen durchführbar sein. Doch auch hier trifft wieder das oben Gesagte zu, daß ein System, welches auf viele variable Einflüsse reagieren muß, nicht die gleiche Effektivität an den Tag legen kann wie ein spezialisiertes System.

So muß bei jedem neu erzeugten Patch entschieden werden, welche Basis am besten geeignet wäre. Diese Entscheidung hängt jedoch von vielen Faktoren ab, die zum Zeitpunkt, an dem die Entscheidung fallen *muß*, nicht unbedingt bekannt sind. So kann der Farbverlauf über einem Patch eigentlich erst dann vollständig beurteilt werden, wenn das Ergebnis der Berechnung vorliegt. Es wären deshalb Vorkehrungen zu treffen, um getroffene Entscheidungen zurücknehmen zu können, wenn sich zu einem späteren Zeitpunkt herausstellen sollte, daß sie aus Mangel an Information nicht angemessen waren. Ein großer Verwaltungsaufwand bliebe nicht aus. Zudem stellt sich bei der Wahl der Basis zusätzlich noch die Frage nach sinnvollen Entscheidungskriterien.

Bei der Erzeugung eines Patches muß bekannt sein, welche Basen als Alternative zur Verfügung stehen. Diese Möglichkeiten werden durch ein zu bestimmendes Auswahlverfahren getestet. Zusätzlich käme ein zeitlicher Mehraufwand hinzu, weil bei jeder Interaktion bestimmt werden müßte, welche Wavelet-Basen beteiligt sind. Es ist daher anzunehmen, daß der gewonnene Speicherplatz weit von der verlorenen Zeit überboten wird. Ob es bereits Bemühungen in diese Richtung gegeben hat, ist dem Autor nicht bekannt.

### 9.4.2 Realisierung importance-basierter Radiosity

Der von Smits et al. [25] vorgestellte Ansatz zur Einbringung von importance in eine Radiosity-Szene, der als eine Erweiterung des HR-Ansatzes vorgeschlagen wurde, ließe sich auch hier einbringen. Dabei ist davon auszugehen, daß sich die Umsetzung eines konstanten importance Wertes pro Patch für den Großteil der Anwendung als ausreichend erweist. Der Frage, ob auch hier ein linearer bzw. quadratischer Verlauf über ein Patch sinnvoll ist, könnte nachgegangen werden.

### 9.4.3 Umsetzung anderer Wavelet-Basen

Wenn man sich die Literatur und die Erwartung nach dem Erscheinen der WR-Veröffentlichung ansieht, so fällt auf, daß oft erwähnt wurde, neue Basen zu untersuchen und zu implementieren. So wurden in [10] noch Coiflets oder Spline Wavelets genannt, die einen möglichen Einsatz erfahren sollten. Bis auf [17] sind dem Autor jedoch keine Veröffentlichungen zu diesen Themen bekannt. Daraus zu schließen, daß die genannten oder auch ungenannte Basen nicht geeignet sind, wäre voreilig. Es bleibt also die Frage, welche Basen noch implementiert werden können. Ich hoffe, den MRT zu einem geeignetem Werkzeug erweitert zu haben, damit Forschungen auf diesem Gebiet durchgeführt werden können.

# Kapitel 10

## Anhang A Beispiel einer Basis Implementierung: Mult2

An dieser Stelle soll die Implementierung einer Wavelet-Basis an Hand der Klassen `t_Mult2Scene` und `t_Mult2` demonstriert werden. Diese realisieren Multiwavelets mit zwei vanishing Moments.

Es sind für jede Basis zwei Klassen zu implementieren: die Szene und das Patch. Es ist davon auszugehen, daß alle neu implementierten Basen mehr als einen vanishing Moment aufweisen werden, da die Umsetzung sonst der Haar-Basis entsprechen würde. Diese ist bereits realisiert.

### 10.1 Die Szene

Name der Klasse `t_Mult2Scene`  
Basisklasse `t_IllumSceneHR`

#### 10.1.1 Eigene Methoden

- `t_Mult2Scene()`  
`t_Mult2Scene(const t_IllumScene& iScene)`  
Konstruktoren
- `virtual ~t_Mult2Scene()`  
Destruktor
- `t_HradPatch* createPatch(t_RadFace* f, const t_ObjectPtr obj)`  
Erzeuge ein Patch vom Typ `t_Mult2` und gebe einen Zeiger auf dieses Patch zurück. Diese Funktion wird aufgerufen, wenn ein neues Patch erzeugt werden soll.
- `t_RadiosityTexture* createRadiosityTexture`  
`(const t_IllumSceneHR* iSceneHr)`  
Erzeuge eine Radiosity-Textur vom Typ `t_RadiosityTextureMult2` und gebe einen Zeiger auf diese zurück.



- `t_Vector4dColor Bs() const`  
`void Bs(t_Vector4dColor& Bs)`

Setzen und Abfragen der Koeffizienten `Bs`. Diese stellen die Radiosity dar, die das Patch nach außen hin repräsentiert. Die Speicherung erfolgt in einem Vektor, der vier Elemente des Typs `t_Color` enthält (siehe `nCoeffs` weiter unten).

- `t_Vector4dColor Bg() const`  
`void Bg(t_Vector4dColor& Bg)`

Setzen und Abfragen der Koeffizienten `Bg`, die gesammelte Radiosity des Patches. Die Speicherung erfolgt ebenfalls in einem Vektor mit vier `t_Color` Elementen.

- `int nCoeffs() const`

Abfragen der Anzahl der benutzen Koeffizienten. Bei Quadraten ist der Wert der Funktion vier, bei Dreiecken drei. Der Speicherplatz wird nicht dynamisch verwaltet, d.h. bei Dreiecken sind vier Koeffizienten vorhanden. Der Wert des vierten ist jedoch auf Null gesetzt (siehe Zugriffsfunktionen der Koeffizienten weiter oben).

- `t_Color samplePatch(t_Real x, t_Real y)`

Ermittelt den Farbwert an der durch  $x$  und  $y$  spezifizierten Stelle. Für Vierecke gilt  $x, y \in [-1, 1] \times [-1, 1]$ , für Dreiecke  $x, y \in [0, 1] \times [0, 1]$ . Ursprung ist der Mittelpunkt bzw. die erste Ecke des Patches. Der Farbwert wird mit Hilfe der in 5.4.1.3 beschriebenen Vorgehensweise bestimmt.

Diese Funktion ist nicht in der Basisklasse enthalten, da das Patch-Sampling nur für Multiwavelets benötigt wird. Bei Flatlets kann man den Farbwert direkt über die Koeffizienten bestimmen.

## 10.2.2 Überscriebene Methoden der Basisklasse

- `t_Real Error(t_HLink* l)`

Berechnet den Fehler des übergebenen Links aus dem geschätzten Fehler des Links multipliziert mit der Sichtbarkeit.

- `t_Real BError(t_HLink* l)`

Diese Funktion wird beim BF-Refine eingesetzt. Der Fehler wird mit dem Produkt aus der Radiosity des Quell-Patches, dem geschätzten Fehler des Links und der Fläche des Empfängers angegeben.

Die Sichtbarkeit ist hierbei bereits in dem geschätzten Fehler des Links enthalten (siehe `calcKernel` weiter unten). Bei der Berechnung der Radiosity des Quell-Patches wird zwischen Vierecken (erster Koeffizient) und Dreiecken (Summe der Koeffizienten) unterschieden.

- `void preparePatch()`

Für Vierecke wird der erste Koeffizient (konstanter Anteil) auf die Emission des Patches gesetzt, während alle anderen Koeffizienten den Wert Null erhalten. Für Dreiecke werden die ersten drei Koeffizienten (Farbwerte der Eckpunkte) auf den Wert der Emission gesetzt, der vierte erhält den Wert Null (er wird nicht genutzt).

- `void add()`

Die gesammelte Energie `Bg` wird mit der Reflektivität der Oberfläche  $\rho$  multipliziert und somit zu einem Radiosity-Wert konvertiert. Das Ergebnis wird zu `Bs` hinzuaddiert. Hier läßt sich also feststellen, wie stark sich die Radiosity des Patches verändert hat. Dies ist ein Maß dafür, ob das System einen Ruhezustand erreicht hat. Der maximale Wert der Änderung wird deshalb in der

Variable `t_RadPatch::deltaBs` festgehalten, um einen Vergleich mit der Fehlerschranke zu ermöglichen.

- `void push()`

Mit Hilfe von Tensormultiplikation (Quadrate) bzw. Matrizenmultiplikation (Dreiecke) werden vier Radiosity-Werte ermittelt, die anschließend an die Kinder in der Hierarchie weitergegeben werden.

- `void pull()`

Analog zum `push`.

- `void updateApproxB()`

Diese Funktion ist dafür zuständig, während der Berechnung *einen* Farbwert für das Patch zu berechnen und ihn in die BRep-Face zu schreiben, damit dieser als Approximation während der Berechnung angezeigt werden kann.

Für Vierecke besteht diese Approximation aus dem ersten Koeffizienten, für Dreiecke wird der Durchschnitt der genutzten drei Koeffizienten gebildet.

- `void calcInteractionFactors(t_IllumSceneHRPtr iSceneHr)`

Der unter [5.4.4.1](#) beschriebene Vorgang wird ausgeführt. Der Kernel der Interaktion wird mit Hilfe der Funktion `calcKernel` (s.u.) berechnet.

- `t_Real calcEstError( t_HradPatch* q, t_HLink* l,  
t_IllumSceneHRPtr iSceneHr)`

Berechne eine einfache Schätzung (ohne Sichtbarkeit) des Formfaktors. Wird nur für F-Refine verwendet.

- `void gatherRad()`

Durchlaufe alle Links, die auf dieses Patch zeigen und sammle entsprechende Energie ein. Dies geschieht über eine Matrizenmultiplikation des im Link vorhandenen Interaktionsfaktors mit dem vierstelligen Vektor des Quell-Patches, der die Bs-Koeffizienten enthält (siehe [5.4.4](#)).

- `bool radiator() const`

Ob es sich bei diesem Patch um einen Radiator handelt, wird analog zur Bestimmung des approximativen Farbwertes der Face bestimmt (siehe `updateApproxB` oben).

- `t_Real calcKernel( t_HradPatch* src, t_Matrix& K, t_Real* vis,  
t_Scene* scene, bool& ccRuleUsed)`

Bestimme, ob die beteiligten Patches weit genug von einander entfernt sind, ansonsten benutze die CC-Rule von Schröder [21]. Führe ein Kernelsampling durch und befrage das Orakel, ob der Kernel glatt genug ist (siehe folgenden Punkt `smoothnessOracle`).

- `t_Real smoothnessOracle( t_Matrix& K, t_HradPatch* src,  
t_Scene* scene)`

Bestimme die Polynommatrix mit Hilfe einer Matrizenmultiplikation aus der Kernel-Matrix. Berechne die Abweichung zwischen dem Polynom und dem Kernel. Gebe diesen Wert zurück (siehe [5.4.2](#)).

- `t_Real smoothnessOracleCC( t_Matrix& K, t_HradPatch* src,  
t_Scene* scene)`

Analoges Vorgehen wie im vorigem Punkt, allerdings für den Fall das die CC-Regel eingesetzt wird.

### 10.2.3 Datenfelder der Klasse

- `t_Vector4dColor v_Bs`
- `t_Vector4dColor v_Bg`

Diese wurden bereits oben erläutert (siehe `Bs()` und `Bg()`).

# Literaturverzeichnis

- [1] Alpert, B.: *A Class of Bases in  $L^2$  for Sparse Representation of Integral Operators*  
SIAM Journal on Mathematical Analysis 24, 1 (Jan 1993).
- [2] Bendels, H.: *Eine topologische Datenstruktur und ihre Anwendung im 3D-Graphiksystem MRT*  
Diplomarbeit, Rheinische Friedrich-Wilhelms Universität Bonn (Mar 1997).
- [3] Bendels, H., Fellner, D.W., Schäfer, S. : *Hierarchical radiosity on topological data structures*  
3D Image Analysis and Synthesis 1996 (Nov 1996), 111-118.
- [4] Bronstein, I.N. und Semendjajew, K.A. : *Taschenbuch der Mathematik*  
B.G. Teubner Verlagsgesellschaft, Leipzig (1979).
- [5] Beylkin, G., Coifman, R., and Rokhlin, V.: *Fast Wavelet Transforms and Numerical Algorithms I*  
Communications on Pure and Applied Mathematics 44 (1991), 141-183.
- [6] Cohen, M., and Wallace, R.: *Radiosity and Realistic Image Synthesis*  
Academic Press, Cambridge, MA (1993).
- [7] Fellner, D.W.: *Extensible Image Synthesis*  
Peter Wisskirchen, editor, Object-Oriented and Mixed Programming Paradigms,  
Focus on Computer Graphics. Springer (Feb 1996), 7-21.
- [8] Glassner, A.S. : *Principles of Digital Image Synthesis*  
Morgan Kaufmann, San Francisco (1995)
- [9] Goral, C.M., Torrance, K.E., Greenberg, D.P, Battaile, B. :  
*Modelling the interaction of light between diffuse surfaces*  
Computer Graphics 18, 3 (July 1994), 213-222.
- [10] Gortler, S., Schröder, P., Cohen, M., and Hanrahan, P. : *Wavelet Radiosity*  
Computer Graphics, Annual Conference Series, 1003 (August 1993).
- [11] Hanrahan, P., Salzman, D. : *A Rapid Hierarchical Radiosity Algorithm for unoccluded environments*  
In K. Bouatouch and C. bouville, editors, Eurographics Workshop on Photosimulation, Realism  
and Physics in Computer Graphics (1990), 151-171.
- [12] Hanrahan, P., Salzman, D., and Aupperle, L. : *A Rapid Hierarchical Radiosity Algorithm*  
Computer Graphics 25, 4 (July 1991), 197-206.
- [13] Heckbert, P.S.: *Simulation Global Illumination Using Adaptive Meshing*  
PhD thesis, University of California at Berkeley (Jan 1991).
- [14] Heckbert, P.S.: *Radiosity in flatland*  
Computer Graphics 11, 3 (Sept. 1992), 181-192.

- [15] Kajiya, J.T.: *The Rendering Equation*  
Computer Graphics 20, 4 (1986), 143-150.
- [16] Nishita, T., Nakamae, E.: *Continuous tone representation of three-dimensional objects taking account of shadows and interreflection*  
Computer Graphics 19, 4 (July 1995), 23-30.
- [17] Pattanaik, S.N., Bouatouch, K.: *Fast Wavelet Radiosity Method*  
Computer Graphics 13, 3 (1994).
- [18] Stoer, J., Bulirsch, R.: *Introduction to Numerical Analysis*  
Springer, New York (1980).
- [19] Schäfer, S. : *Hierarchical Radiosity on Curved Surfaces*  
8th Eurographics Workshop on Rendering, Springer (1997), 187-192.
- [20] Schröder, P., Gortler, S.J., Cohen, M.F., and Hanrahan, P.: *Wavelet Projections For Radiosity*  
4th Eurographics Workshop on Rendering (June 1993).
- [21] Schröder, P.: *Numerical integration for radiosity in the presence of singularities*  
4th Eurographics Workshop on Rendering (June 1993).
- [22] Schröder, P.: *Wavelet Radiosity: Wavelet Methods for Integral Equations*  
ACM SIGGRAPH Course Notes Wavelets in Computer Graphics (1996), 143-165.
- [23] Schröder, P., and Hanrahan, P.: *On the Form Factor Between Two Polygons*  
Computer Graphics, Annual Conference Series, 1003 (Aug 1993).
- [24] Sillion, F.X., Puech, C.: *Radiosity & Global Illumination*  
Morgan Kaufmann, San Francisco (1994)
- [25] Smits, B.E., Arvo, J.R., Salesin, D.H.: *An importance-driven radiosity algorithm*  
Computer Graphics 23, 3 (July 1989), 335-344.
- [26] Willmott, A.J.: *High-Order Basis Functions in Radiosity*  
Not published. <http://www.cs.cmu.edu/~radiosity/notes>
- [27] Willmott, A.J., Heckbert, P.S.: *An Empirical Comparison of Radiosity Algorithms*  
CMU Tech report (April 1997)
- [28] Zatz, H.R.: *Galerkin Radiosity: A Higher-order Solution Method for Global Illumination*  
Computer Graphics, Annual Conference Series, 1003 (Aug 1993).