Ray Tracing mit dynamischer Bounding Volume Hierarchie

Diplomarbeit

Vorgelegt von
Martin Eisemann

Institut für Computervisualistik
Arbeitsgruppe Computergraphik

Betreuer:	Dipl.-Inform. Thorsten Grosch
Prüfer:	Prof. Dr.-Ing. Stefan Müller

Januar 2006

Eidesstaatliche Erklärung

Hiermit erkläre ich an Eides statt, dass die vorliegende Arbeit selbstständig verfasst wurde und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet wurden. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ort / Datum	Unterschrift

Danksagung

Hiermit möchte ich mich bei allen bedanken, die mir bei der Anfertigung dieser Arbeit beigestanden und sie in dieser Form erst ermöglicht haben.

Insbesondere richtet sich mein Dank an Dipl. Inf. Thorsten Grosch, der mir als Betreuer mit einer unglaublichen Geduld in vielen anregenden Gesprächen zur Seite stand.

Des weiteren danke ich meiner Familie und meiner Freundin Angela, die mich immer unterstützten, motivierten und die zeitraubende Arbeit des Korrekturlesens auf sich genommen haben.

Inhaltsverzeichnis

Kapitel 1

Einführung

Computergenerierte Bilder haben schon seit langer Zeit Einzug in unseren Alltag gehalten und sind dort nicht mehr wegzudenken. Sei es in Form von Computerspielen, Virtual Reality Anwendungen, Visualisierung medizinischer Datensätze oder photorealistischer Animationen in Film und Fernsehen. Unzählige Methoden zur Verbesserung und Beschleunigung sind in den letzten Jahren entwickelt worden und haben dafür gesorgt, dass heutzutage jeder normale Heimrechner in der Lage ist halbwegs komplexe Szenen in Echtzeit [1] darzustellen.

Dabei lassen sich die verwendeten Verfahren zur digitalen Bildsynthese grob in zwei Klassen aufteilen. Zum einen die Rasterisierungsalgorithmen, welche vor allem in der Computerspielebranche beliebt sind, da leicht in Hardware zu implementieren, und so in heutigen Grafikkarten billig vertrieben werden. Und zum anderen Ray Tracing. Dieses wurde für die Computergrafik von Appel [App68] zur Berechnung von Schatten eingeführt und von Whitted zu einem kompletten Beleuchtungsmodell erweitert [Whi80]. Ray Tracing Verfahren sind vor allem dafür bekannt, dass sie in der Lage sind photorealistische Bilder zu erzeugen, da sie die Grundlage für viele Methoden zur globalen Beleuchtung darstellen. Optische Effekte, wie Schattenwurf, Spiegelung von Objekten in den Oberflächen anderer Objekte, sowie Transparenzen, lassen sich verhältnismässig einfach und exakt berechnen, während Rasterisierungsverfahren diese nur schwierig und mit hohem Aufwand approximieren. Auf der anderen Seite ist Ray Tracing aber auch dafür bekannt einen sehr hohen Rechenaufwand mit sich zu bringen, weswegen es lange Zeit lediglich für offline Rendering verwendet wurde.

[1] Echtzeit bedeutet in diesem Zusammenhang ≥ 30 Bilder pro Sekunde. Andere Definitionen gehen eventuell von leicht anderen Zahlen aus.

In den letzten Jahren ist es jedoch verschiedenen Gruppen gelungen, Ray Tracing mit interaktiven Frameraten zu erzielen [PMS+99] [WBWS01] [Gei05]. Und es wird schon seit längerer Zeit gerätselt, wann der Punkt erreicht sein wird, an dem Ray Tracing die Rasterisierungsverfahren überholen wird. Denn anders als letztere, welche einen Aufwand von $O(n)$ bei n Objekten, mit sich bringen, kann durch den Einsatz von effizienten Beschleunigungsstrukturen die Ray Tracing Phase im Durchschnitt in $O(\log n)$ pro Pixel durchgeführt werden.

Leider benötigen diese Beschleunigungsstrukturen eine verhältnismäßig lange Vorverarbeitungszeit, welche mit hohem Aufwand verbunden ist, meist $O(n \log n)$ oder höher. Dies beschränkte Ray Tracing meist auf statische Szenen oder Szenen in denen sich lediglich die Kamera bewegte. Hinzu kommt, dass sich die Ray Tracing Phase meist leicht parallelisieren lässt, da jedes Pixel des darzustellenden Bildes im Prinzip unabhängig von den anderen berechnet werden kann. Nahezu linearer Zuwachs der Rechengeschwindigkeit konnte bei bis zu 128 Prozessoren gezeigt werden [PMS+99]. Hingegen scheint keinerlei Literatur zu existieren, welche sich mit der Parallelisierung der Rekonstruktionsphase für die Beschleunigungsstruktur beschäftigt. Die inhärenten Probleme dabei sind leicht ersichtlich. Teilt man die Objekte auf die verschiedenen Prozesse auf, müssen die einzelnen Beschleunigungsstrukturen später zu einer gemeinsamen zusammengeführt werden. Dies wirft vor allem Schwierigkeiten auf, wenn keine räumliche Lokalität in den jeweiligen Objekten gegeben ist. Eine andere Möglichkeit wäre es, den Raum, in welchem sich die Szene befindet aufzuteilen und die entsprechenden Bereiche separat voneinander aufbauen zu lassen. Problematisch hierbei sind dann allerdings vor allem die möglichen Ungleichheiten in der Aufgabenverteilung. Gemäß Amdahls Gesetz [Amd67] wird stets der nicht parallelisierte Bereich zum Flaschenhals werden. Im Falle des Ray Tracings kann dies wie folgt dargestellt werden: Die benötigte Rechenzeit zur Erstellung eines Bildes $T(c)$, bei der Verwendung von c Prozessoren in der Ray Tracing Phase t_{rt} kann beschrieben werden als

$$T(c) = t_{rc} + \frac{1}{c} t_{rt}. \tag{1.1}$$

Folglich wird über kurz oder lang die Rekonstruktionsphase t_{rc} der Beschleunigungsstruktur zum limitierenden Faktor werden. Aber auch ohne Parallelisierung kann ein ähnlicher Effekt über Frameless Rendering [BFMZ94] erreicht werden, wobei für jedes Bild nur jeder n-te per Zufall ausgewählte Pixel neu berechnet wird. Dies ist in Abbildung 1.1 angedeutet. Die x-Achse gibt dabei an, jeder wievielte Pixel berechnet wird, während die y-Achse für die benötigte Rechenzeit steht. Es ist deutlich erkennbar, dass diese für die Ray

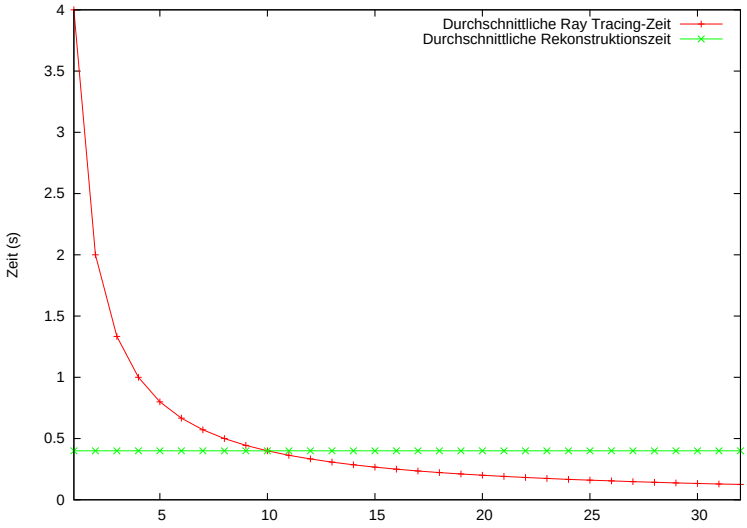


Abbildung 1.1: Vergleich der benötigten Zeit in der Ray Tracing Phase mit der Rekonstruktionszeit, unter Verwendung von Frameless Rendering. Die nahezu horizontale Kurve (grün) ist die Rekonstruktionszeit. Die zweite Kurve (rot) gibt die benötigte Renderingzeit an, während die Anzahl an zu berechnenden Pixeln vermindert wird. Bspw. $x = 4$ bedeutet, dass lediglich für $\frac{1}{4}$ der Pixel in jedem Bild neue Primärstrahlen erzeugt werden.

Tracing Phase monoton fallend ist, während die Rekonstruktionszeit nahezu gleich bleibt.

Aus dieser Problematik heraus entstand die Motivation, Beschleunigungsstrukturen auch hinsichtlich ihres Nutzens für dynamische Szenen zu untersuchen und ob ein kompletter Neuaufbau für jedes Bild nicht zu vermeiden ist. Der Inhalt dieser Arbeit konzentriert sich dabei insbesondere auf die Anwendbarkeit von Bounding Volume Hierarchien (BVHs). Ziel ist die Realisierung beliebiger dynamischer Szenen mit nicht-deterministischen Bewegungen, sprich ohne Vorwissen über die Bewegung der Objekte, bzw. welches Objekt bewegt werden kann. Dafür werden verschiedene Ansätze wie Lazy Evaluation, Ausnutzung lokaler Gegebenheiten, Qualitätsprüfungen der Hierarchie und die Möglichkeit, Objekte in $O(1)$ in eine BVH einfügen zu können, untersucht und in verschiedenen Verfahren vorgestellt.

Der Aufbau ist dabei wie folgt. In Kapitel 2 soll zunächst eine Einführung in das Verfahren des Ray Tracings, sowie seiner möglichen Beschleunigung gegeben werden, was dem weiteren Verständnis der Arbeit dienen soll. Anschließend wird in Kapitel 3 ein Überblick über die bisherigen Ansätze für das Ray Tracing dynamischer Szenen gegeben.

In Kapitel 4 wird das Prinzip der Bounding Volume Hierarchien detaillierter vorgestellt, insbesondere in Hinsicht auf Methoden zu ihrer Erstellung und ihre Traversierungsmöglichkeiten. Darauf folgend werden verschiedene Heuristiken zur Entwicklung eines Qualitätskriteriums für BVHs in dynamischen Szenen erläutert und diskutiert. In Abschnitt 4.3 wird dann auf die in dieser Arbeit entwickelten Rekonstruktionsmethoden eingegangen. Wie sich die Informationen eines Szenegraphens noch zusätzlich nutzen lassen, wird in Abschnitt 4.4 erläutert.

In Kapitel 5 wird ein neues Verfahren zur Beschleunigung des Schnitttestes zwischen Strahlen und achsenparallelen Boxen dargestellt. Anschließend werden in Kapitel 6 die gesammelten Testergebnisse präsentiert, sowie in Kapitel 7 ein Fazit gezogen und Ausblicke auf Ansatzpunkte für weiterführende Arbeiten gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel sollen kurz die Grundlagen des Raytracing-Verfahrens erläutert werden, welche zum weiteren Verständnis der Arbeit notwendig sind. Dazu wird zunächst der klassische von Whitted in [Whi80] eingeführte rekursive Ray Tracing Algorithmus vorgestellt, sowie darauf folgend einige der bekannteren Verfahren zur Beschleunigung desselben.

2.1 Das Ray Tracing Verfahren

Die Hauptaufgabe eines jeden Ray Tracing System besteht darin, auf die eine oder andere Art und Weise, aus einer strukturellen Szenenbeschreibung ein zweidimensionales Bild für das Ausgabegerät zu erzeugen, oder zwei, falls mittels Stereoprojektion der Eindruck einer dreidimensionalen Welt verstärkt werden soll. Ähnlich der Aufnahme eines klassischen Filmes bedient sich das Ray Tracing Verfahren dafür ebenfalls einer Kamera, welche das Licht der sie umgebenden Szene einfängt und so für jedes Pixel des zu berechnenden Bildes eine passende Farbe setzt.

Dafür soll im folgenden kurz das Prinzip der Lochkamera erläutert werden, welche wohl auch die am Häufigsten verwendete Variante in der Computergrafik darstellt. Normalerweise besteht eine solche Kamera aus einer lichtundurchlässigen Box, an deren Rückseite im Inneren ein lichtempfindlicher Film angebracht ist. An der Vorderseite wird nun ein kleines Loch erzeugt, welches Lichtstrahlen durch die Öffnung auf den Film fallen lässt. Ist das Loch klein genug und die Belichtungszeit korrekt gewählt, entsteht auf dem Film ein umgekehrtes Abbild der Wirklichkeit vor der Kamera. Grund dafür ist die relativ geradlinige Ausbreitung des Lichtes, sieht man von einigen physikalischen Effekten, wie bspw. Interferenzen ab.

Überträgt man dieses Konzept auf die Computergrafik, können sogar noch weitere Vereinfachungen vorgenommen werden. Dabei wird der Film, im folgenden Projektionsebene genannt, vor den Augpunkt verschoben, welcher der Linsenöffnung der Kamera entspricht. D.h ausgehend von einem Blickpunkt $\mathbf{e}$ (eye-point), soll die Szene entlang einer Blickrichtung $\vec{\mathbf{d}}$ (direction), bzw. dem betrachteten Punkt $\mathbf{l}$, dargestellt werden, die durch die Projektionsebene sichtbar wäre. Die Strahlen, welche vom Augpunkt $\mathbf{e}$ durch die Eckpunkte der Projektionsebene verlaufen, definieren dabei das so genannte *viewing frustum*, welches den sichtbaren Bereich vom Augpunkt aus festlegt.

Wenn nun ein Bild generiert wird, bspw. für die Ausgabe auf einem Monitor, haben wir jedoch keinen Film vorliegen, sondern eine feste Anzahl an Pixeln, welche die Projektionsebene bilden, für die jeweils die entsprechende Farbe ermittelt werden muss. Dafür würde man, wolle man physikalisch korrekt vorgehen, Informationen über jeden Lichtstrahl benötigen, der ausgehend von einer Lichtquelle in der Szene auf direktem oder indirektem Weg die Projektionsebene schneidet und im Auge des Betrachters landet. Dieses Verfahren hat jedoch gravierende Nachteile. Erstens, können nicht unendlich viele Lichtstrahlen erzeugt und durch die Szene verfolgt werden, weswegen hier Einschränkungen vorgenommen werden müssen. Und zweitens würde lediglich ein Bruchteil der so erzeugten Strahlen die Projektionsebene schneiden und somit zum Bild beitragen. Dieses Verfahren wird auch *Forward Ray Tracing* genannt.

Ein weniger aufwändiges Verfahren bildet das so genannte *Backward Ray Tracing*, welches von Appel in [App68] für die Berechnung von Schatten entworfen und von Whitted in [Whi80] für Reflexionen und Transparenzen erweitert wurde. Ausgehend von der Annahme, dass sich Photonen, die Lichtträger, auf geradlinigen Bahnen bewegen, wird die Frage gestellt, welche dieser Photonen wirklich einen Beitrag zu dem zu generierenden Bild liefern. Dafür werden ausgehend vom Blickpunkt $\mathbf{e}$ ein oder mehrere so genannte *Primär-* oder *Sehstrahlen* durch das gewünschte Pixel geschickt und das erste Objekt der Szene ermittelt, welches diese Strahlen schneidet. Um herauszufinden wieviel Licht dieses in Betrachterrichtung reflektiert, versucht man die in [Kaj86] eingeführte *Rendering Equation* zu lösen, welche die ausgehende Strahldichte für einen beliebigen Punkt berechnet. Dies gelingt meist nur durch starke Approximation. Im klassischen Ray Tracing-Verfahren geschieht dies folgendermaßen: Ausgehend vom Augpunkt wird durch jeden Pixel innerhalb der Projektionsebene ein Strahl verschickt. Verfehlt dieser die Szene, so wird für diesen Pixel die Hintergrundfarbe als Farbe gesetzt. Trifft er jedoch auf mindestens ein Objekt der Szene, so wird das dem Ursprung des Strahles am nächsten liegende, geschnittene Objekt bestimmt und getestet,

ob dieser Schnittpunkt direkt von den Lichtquellen der Szene erhellt wird. Indem so genannte *Schattenfühler* oder *Schattenstrahlen* vom Schnittpunkt aus Richtung Lichtquellen geschickt werden, wird ermittelt, ob sich zwischen Schnittpunkt und Lichtquelle ein Objekt befindet. Ist dies der Fall, so liegt der Schnittpunkt bezüglich dieser Lichtquelle im Schatten. Ist dies nicht der Fall, kann anhand der Materialeigenschaften und des eintreffenden Lichtes eine Farbe für diesen Punkt durch diese Lichtquelle errechnet werden.

Nun lassen sich auf einfache Art auch Effekte, wie Reflexionen oder Transmission, darstellen, falls das Material des getroffenen Objektes dies erfordert. Für die Reflexion, also die Spiegelung anderer Objekte in dem getroffenen Objekt, wird der eintreffende Sehstrahl an der Oberflächennormalen gespiegelt und wieder in die Szene verschickt. Für die Berechnung des Transmissionsstrahles müssen die Brechungsgesetze aus der Optik angewandt werden. Wie stark das Ergebnis dieser so genannten *Sekundärstrahlen* in das Endergebnis einfließt, hängt unter anderem von den getroffenen Materialien ab, welche jeweils einen entsprechenden Reflexions- und Transmissionskoeffizienten, k_r und k_t besitzen. Trifft einer dieser *Sekundärstrahlen* wieder auf ein Objekt, so müssen von dort aus gegebenenfalls erneut Reflexions-, Transmissions und Schattenstrahlen erzeugt werden. Das Ganze entwickelt sich somit zu einem stark rekursiven Prozess. Damit der Algorithmus in jedem Fall terminiert, sollte eine maximale Rekursionstiefe vorgegeben werden. Da vor allem indirekte, diffuse Beleuchtung durch andere Objekte bei diesem Verfahren außer Acht gelassen werden, wird je nach gewünschten Materialeigenschaften oft auch noch ein ambienter Term hinzugefügt, welcher sozusagen eine globale Beleuchtung aus allen Richtungen darstellt. Dies ist das gängigste, von Whitted [Whi80] eingeführte Beleuchtungsmodell. Natürlich existieren auch deutlich komplexere, oder einfachere, je nach gewünschter Bildqualität.

Abbildung 2.1 gibt einen möglichen Pseudocode für das klassische Ray Tracing Verfahren wieder, zudem ist in Abbildung 2.2 ein Beispiel für das Bisherige gegeben. Ausgehend vom Blickpunkt werden Strahlen durch die Projektionsebene in die Szene verschickt. Während Strahl $\mathbf{A}(t)$ alle Objekte verfehlt, und somit für diesen Pixel die zuvor definierte Hintergrundfarbe gesetzt wird, trifft Strahl $\mathbf{B}(t)$ auf einen Zylinder. Für diesen Schnittpunkt wird ein Schattenstrahl $\mathbf{C}(t)$ zur Lichtquelle erzeugt, um zu testen, ob sich der Schnittpunkt im Schatten befindet. Zudem werden ein Reflexionstrahl $\mathbf{D}(t)$ und ein Refraktionsstrahl $\mathbf{E}(t)$ erzeugt und weiterverfolgt.

Dies soll für das weitere Verständnis der Arbeit eine ausreichende Einführung sein. Für eine umfassendere Beschreibung sei auf vorhandene Literatur verwiesen [Gla89] [SM03].

```
void main()
{
 for every pixel x, y{
   Ray r = getPrimaryRay(x,y);
   color(x,y) = trace(r);
 }
}

color trace(Ray ray)
{
  if( closest intersection of ray with scene exists )
  return shade(intersectionpoint);

  else
    return backgroundcolour;
}

color shade(Intersection intersectionpoint)
{
  color = ambient illumination;

  for every lightsource l{
    trace shadow ray from intersectionpoint to l;
    if( intersectionpoint is not in shadow )
      color += direct illumination;
  }

  if( material at intersectionpoint is reflective )
    color += kr * trace(reflection ray);
  if( material at intersectionpoint is transmissive )
    color += kt * trace(transmission ray);
  return color;
}
```

Abbildung 2.1: Pseudocode des klassischen Ray Tracing-Verfahrens

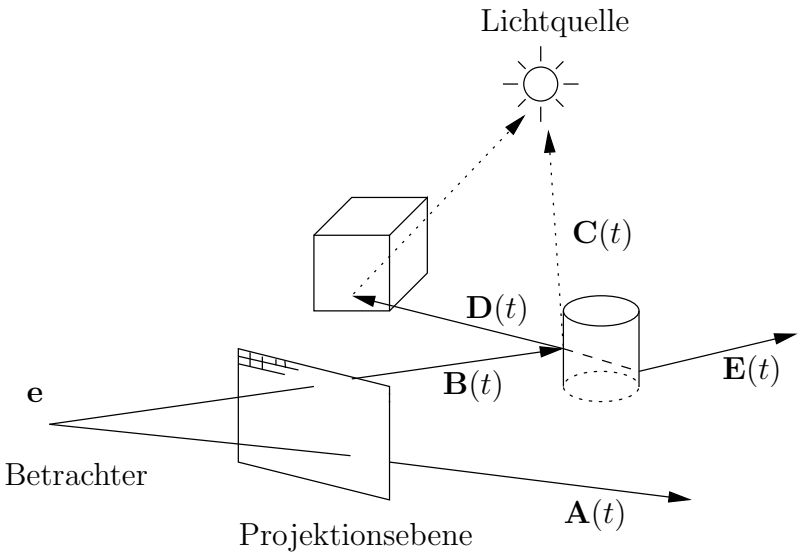


Abbildung 2.2: Klassisches Ray Tracing Modell

2.2 Beschleunigungsverfahren

Der bisher vorgestellte Algorithmus ist in seiner Komplexität mehr als aufwändig. Jeder erzeugte Strahl muss mit jedem Objekt der Szene auf Schnittpunkte getestet werden und von diesen Schnittpunkten muss dann der jeweils vorderste ausgewählt werden, damit mit der Berechnung fortgefahren werden kann. Es verwundert kaum, dass sich im Laufe der Jahre die verschiedensten Methoden entwickelt haben um diesen aufwändigen Prozess zu beschleunigen. Eine komplette Auflistung sämtlicher Methoden würde weit über den Rahmen dieser Arbeit hinausgehen. Daher sollen im folgenden nur ein Überblick über die gängigsten Beschleunigungsmethoden für das soeben vorgestellte Ray Tracing-Verfahren gegeben werden, vergleiche auch Abbildung 2.3. In die erste Kategorie *Beschleunigung der Schnittpunktberechnung* fallen dabei alle Arten von Algorithmen und Strukturen, die den Rechenaufwand zur Bildgenerierung verringern, sei es durch schnellere Algorithmen, oder Datenstrukturen, die es erlauben ganze Gruppen von Objekten von der weiteren Betrachtung auszuschließen. Zur zweiten Kategorie *Reduzierung der Strahlanzahl* gehören alle Techniken, die die Gesamtanzahl der zu untersuchenden Strahlen reduzieren. Einen ähnlichen Weg beschreiten die *generalisierten Strahlen*, wobei keine einzelnen Strahlen mehr verfolgt werden, sondern komplette Bündel von Strahlen. In den Bereich *Parallelisierung* fällt alles, was die benötigten Berechnungen simultan ausführt. Und zuletzt

existieren Ansätze, die versuchen, spezielle Hardware für den Ray Tracing-Prozess zu nutzen.

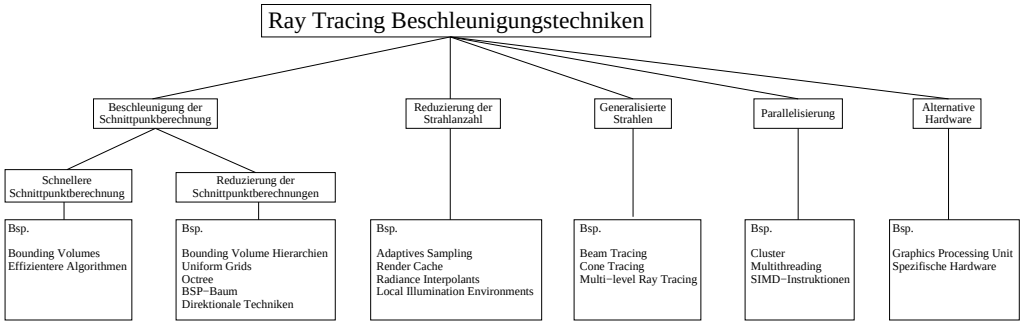


Abbildung 2.3: Eine generelle Klassifizierung von Ray Tracing Beschleunigungstechniken, in Anlehnung an [Gla89] und [Gei05]

2.2.1 Beschleunigung der Schnittpunktberechnungen

Die Kategorie *Beschleunigung der Schnittpunktberechnungen* spaltet sich auf in *schnellere Schnittpunktberechnung* und *Reduzierung der Schnittpunktberechnungen*. Unter den ersten Punkt fällt vor allem die Wahl eines geeigneten Hüllvolumens (engl. *Bounding Volume*) für Objekte und schnellere Algorithmen für die Schnittberechnung. Ein Hüllvolumen ist dabei ein geometrisches Primitiv, wie eine Kugel, eine Box oder auch komplexere Strukturen wie k-DOPs[1] [KK86], für die eine Schnittpunktberechnung einfacher auszuführen ist, als der Test mit dem enthaltenen Objekt selbst. Die Idee dabei ist, dass das enthaltene Objekt gar nicht erst weiter getestet zu werden braucht, wenn der Strahl bereits das Hüllvolumen verfehlt. Durch den vereinfachten Schnitttest kann dies zu einer deutlichen Rechenersparnis führen kann. Auch wenn die Anzahl an zu testenden Objekten konstant bleibt. In Abschnitt 4.1 wird auf die Wahl des Hüllvolumens für die in dieser Arbeit vorgestellten Verfahren noch einmal näher eingegangen.

Daneben werden immer neue Algorithmen entwickelt, welche die Schnittpunktberechnung selbst beschleunigen, oder vereinfachen. Eine neue Möglichkeit für den Schnitttest mit achsenparallelen Boxen, wird in Abschnitt 5 vorgestellt.

Auch wenn der Geschwindigkeitsgewinn durch das geschilderte Verfahren bereits beträchtlich ist, liegt der Aufwand für jeden Strahl immer noch bei $O(n)$

[1] DOP = Directed Oriented Polytope

bei n Objekten. Erst durch die Einführung von spatialen und Objektstrukturen, konnte ein durchschnittlicher logarithmischer Rechenaufwand $O(\log n)$ erreicht werden, welcher Ray Tracing wieder attraktiv werden ließ. Durch eine entsprechende Vorverarbeitung der Szene, sollen überflüssige Schnittpunktberechnungen vermieden und nicht relevante Bereiche der Szene ausgespart werden. Die wichtigsten Verfahren sollen im Folgenden näher beschrieben werden.

Bounding Volume Hierarchien

Aus der Erkenntnis heraus, dass ein Objekt, dessen Hüllvolumen vom Strahl verfehlt wird, nicht auf Schnittpunkte getestet werden muss, gelangt man schnell zu der Einsicht, dass dies auch fortgeführt werden kann, um jeweils Gruppen von Objekten zusammenzufassen und so eine hierarchische Datenstruktur entstehen zu lassen. Man spricht von einer *Bounding Volume Hierarchie* (BVH) [RW80]. So können komplette Teilbereiche der Szene ausser Acht gelassen werden, wenn der Strahl bereits hoch in der Hierarchie das entsprechende Hüllvolumen verfehlt.

Das Verfahren, welches zur Erstellung der Hierarchie verwendet wird, hat dabei entscheidenden Einfluss auf die Qualität und damit auf die benötigte Rechenzeit. Wird bei der Erstellung die Nähe der Objekte untereinander nicht beachtet, kann im schlimmsten Falle die BVH den Ray Tracing-Vorgang sogar verlangsamen. In Abschnitt 4.1.1 werden verschiedene dieser Methoden angeführt.

Die Reihenfolge, in welcher die Hierarchie traversiert wird, ist dabei implementierungsabhängig und nicht immer liefert eine Traversierung entlang des Strahles die besten Ergebnisse. In Abschnitt 4.1.2 werden die beiden gängigsten Methoden vorgestellt.

Einer der größten Vorteile von BVHs, insbesondere in Hinsicht auf dynamische Szenen, ist, dass jedes Objekt in nur genau einem Knoten vorliegt, was ein effizientes Entfernen und Einfügen ermöglicht. Ebenfalls vorteilhaft ist ihr impliziter Umgang mit leeren Räumen. Da die Hüllvolumen die Objekte bestmöglich umschließen, werden leere Räume bei einer günstigen BVH während der Traversierung relativ gut ausgespart, so dass diese verhältnismäßig wenig Einfluss auf die benötigte Rechenzeit haben. Dass dies nicht selbstverständlich ist, wird bei der Beschreibung der nächsten Verfahren deutlich. Andererseits ist die Traversierung der Hierarchie im Vergleich recht aufwändig. Dennoch konnte dieses Verfahren bereits sehr erfolgreich von Geimer in einem Ray Tracer mit interaktiven Frameraten eingesetzt werden [Gei05].

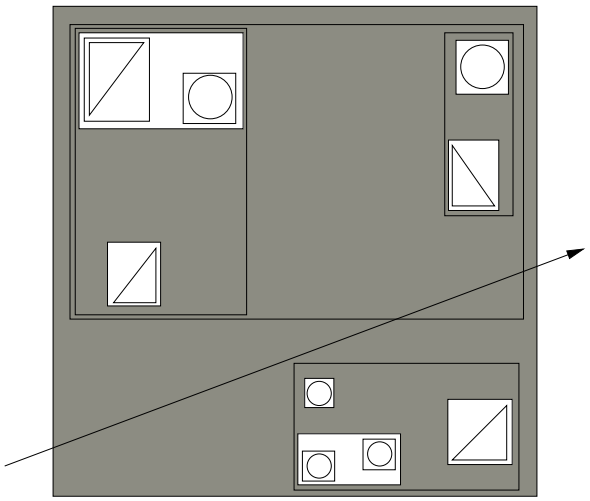


Abbildung 2.4: 2D-Darstellung einer Bounding Volume Hierarchie

Uniform Grid

Fujimoto *et al.* [FTI86] haben eine andere Möglichkeit entwickelt die Schnittpunktfindung zu beschleunigen. Dafür unterteilen sie die Szene in eine Menge von so genannten *Voxeln*[2], welche zu einem regulären 3D-Gitter (*Uniform Grid*) angeordnet sind. Jedes Objekt der Szene wird nun genau den Voxeln zugeordnet, mit welchem es in irgendeiner Art überlappt. Meist wird der Test mit einem einfachen Hüllvolumen der Objekte durchgeführt. Ein schöner Nebeneffekt dabei ist, dass die Einordnung eines Objektes im Durchschnitt in $O(1)$ möglich ist, und somit sehr schnell verläuft. Große Objekte verursachen dabei einen erhöhten Aufwand, da sie mit verhältnismäßig mehr Voxeln überlappen.

Für die Strahltraversierung wird das Gitter mittels eines *three-dimensional digital difference analyzer*, kurz 3DDDA, entlang des Strahles durchlaufen. Jedes getroffene Voxel liefert dann eine Kandidatenliste der in ihm enthaltenen Objekte, mit denen der Strahl getestet wird. Wird ein Schnittpunkt gefunden, welcher innerhalb des aktuell untersuchten Voxels liegt, kann die Traversierung abgebrochen werden, da kein näherer Schnittpunkt möglich ist. Da der 3DDDA Algorithmus inkrementell arbeitet, ähnlich einem Linien-Rasterisierungsalgorithmus, ist die Traversierung extrem schnell durchzuführen, birgt aber auch einige Nachteile.

Durch die reguläre Unterteilung des Gitters, wird keinerlei Rücksicht auf

[2]Voxel = Volume Pixel

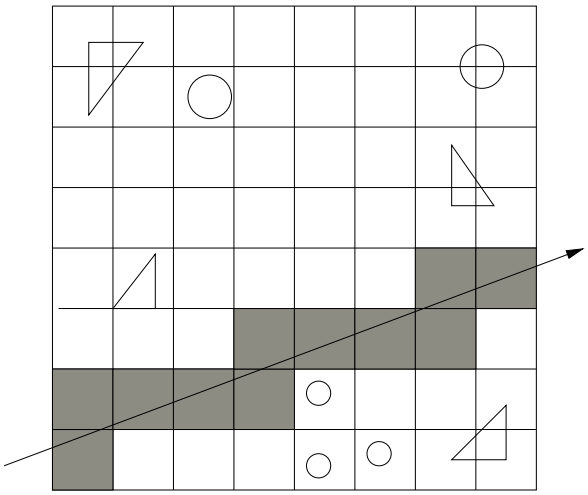


Abbildung 2.5: 2D-Darstellung eines Uniform Grids

etwaige nicht-uniforme Verteilung der Objekte in der Szene genommen. So ist es szenenabhängig möglich, dass einige Voxel sehr viele Objekte enthalten. Inzwischen gibt es allerdings auch Ansätze, die dieses Problem durch eine hierarchische Weiterunterteilung solcher Voxel zu lösen suchen. Dabei wird für ein solches Voxel, sollte es mehr Objekte als ein zuvor festgelegter Threshold enthalten, erneut ein Uniform Grid angelegt, welches dann traversiert wird, sollte der Strahl dieses schneiden. Man spricht dann von einem *Recursive Grid* [JW89]. In eine ähnliche Richtung geht auch der Ansatz der *Hierarchy of Uniform Grids* von Cazals *et al.* [CDP95]. Diese nehmen vor der Unterteilung noch eine Filterung und ein Clustering vor, um Objekte gleicher Größe zusammenfassen zu können, und sich so einige der Unterteilungen ersparen zu können.

Ein weiteres bestehendes Problem ist das der leeren Voxel. In Bereichen, die wenig oder keine Objekte enthalten, wird viel Arbeit auf die Traversierung des Grids verwendet. Devillers [Dev89] umgeht diesen Nachteil durch die Einführung von Macro-Regionen, d.h. leere Regionen, die direkt übersprungen werden können.

Octree

Der von Glassner [Gla84] erstmals im Ray Tracing Kontext eingeführte *Octree* stellt eine Datenstruktur dar, welche sich im Gegensatz zum Uniform Grid adaptiv an die Distribution der Szenengeometrie anpasst, d.h. Berei-

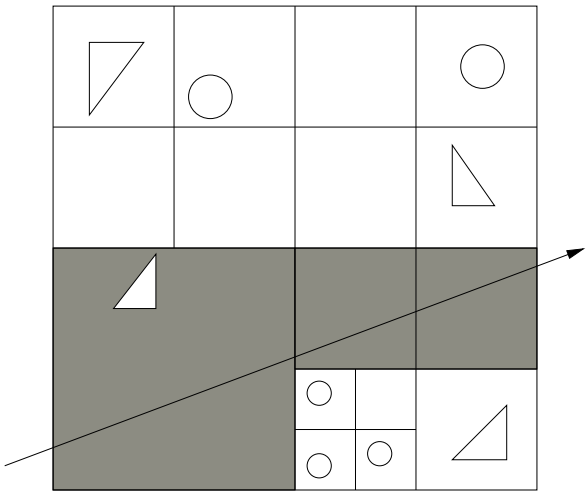


Abbildung 2.6: 2D-Darstellung eines Octrees

che, welche viele Objekte enthalten, werden feiner unterteilt als Regionen mit wenigen. Dabei wird die Szene zunächst von einem einzigen Voxel umschlossen. Dieses wird in acht weitere disjunkte Subvoxel gesplittet und die Objekte auf diese verteilt, je nachdem ob eine Überlappung mit diesem Subvoxel vorliegt oder nicht. Dabei kann es geschehen, dass Objekte in mehr als ein Voxel einsortiert werden müssen. Anhand eines Stoppkriteriums wird entschieden, ob der Vorgang rekursiv auf die Subvoxel angewandt wird oder der Vorgang abbricht. Dieses kann bspw. abhängig von der Objektanzahl oder Rekursionstiefe sein. Auf diese Weise enthalten nach Abschluss der Konstruktionsphase lediglich die Blätter des Octrees Kandidatenlisten, gegen welche ein Strahl getestet werden muss, sollte er den entsprechenden Voxel schneiden. Die restlichen dienen lediglich dem Auffinden des nächsten zu testenden Voxels.

Die Traversierung eines Strahles kann jeweils durch Nachbarschaftsfindung geschehen [Gla84], indem in jedem Schritt ein Punkt ausgewählt wird, der sich garantiert im nächsten zu untersuchenden Voxel befindet und für diesen der entsprechende Knoten im Octree gesucht wird, oder durch eine beim Wurzelknoten beginnende Top-Down Prozedur [GA93]. Bei dieser werden in jedem Schritt die weiter zu untersuchenden Octanten gemäß dem Strahl sortiert, indem dieser gegen die drei Schnittebenen, welche den Knoten unterteilen, getestet wird und anhand der errechneten Strahlparameter entschieden wird, welcher Voxel als nächstes zu testen ist.

Die Unterteilung verläuft klassischerweise durch den Mittelpunkt des jewei-

ligen Voxels. Es existieren jedoch auch fortschrittlichere Varianten, wie der *Octree-R* von Whang *et al.* [WSC+95]. Hier wird versucht die Anzahl an zu erwartenden Strahl-Objekt-Schnitttests zu minimieren, und zwar mittels einer Abwandlung der *Surface Area Heuristic* von MacDonald und Booth [MB90], welche in 4.1.1 noch einmal genauer betrachtet wird.

BSP-Bäume /k-d-Bäume

Die *Binary Space Partitioning Trees* (BSP-Trees) [Ben75][Kap85] unterteilen die Szene in jeweils zwei Teile entlang einer so genannten *Splitting Plane*. Die Objekte der Szene werden daraufhin auf die beiden so entstandenen Halbräume verteilt, notfalls in beide, oder sie werden vorher aufgesplittet. Diese Unterteilung wird fortgesetzt, bis ein vorher festgesetztes Abbruchkriterium erreicht wird. Die Splitting Plane wird dabei entweder entlang eines Polygons aus der Szene gelegt (*Polygon-aligned BSP Tree*), oder sie wird anhand einer Kostenfunktion senkrecht zu einer der drei Koordinatenachsen gelegt. Man spricht in diesem Fall von einem *k-d-Baum*. Der daraus resultierende, vereinfachte Schnitttest mit der Splitting Plane ist auch der Grund, weshalb k-d-Bäume im Ray Tracing Kontext deutlich häufiger eingesetzt werden als BSP-Bäume.

Jansen [Jan86] entwickelte einen rekursiven Traversierungsalgorithmus, der es erlaubt, den Baum entlang des Strahles zu durchsuchen. Dabei wird jeder Strahl zunächst an der Szene und später an der jeweiligen Splitting Plane geclippt und nur das jeweilige daraus entstehende Segment $[t_{in}, t_{out}]$ wird in den Kindknoten weiterverfolgt. Dabei gilt, verfehlt der Strahl die Szene, so kann die Traversierung direkt abgebrochen werden. Andernfalls wird der Strahlparameter t_{in} bei Eintritt in die Szene und t_{out} bei Austritt berechnet und die Traversierung am Wurzelknoten begonnen. Handelt es sich beim aktuell untersuchten Knoten um einen Blattknoten, so wird der Strahl gegen die darin enthaltenen Objekte getestet. Wurde so mindestens ein Schnittpunkt gefunden, wird der dem Strahlursprung nächstgelegene zurückgeliefert und die Traversierung beendet. Handelt es sich um einen inneren Knoten wird der Strahlparameter t_s des Schnittpunktes mit der Splitting Plane des aktuellen Knotens berechnet und mit dem Strahlintervall $[t_{in}, t_{out}]$ verglichen. Nun gibt es drei Fälle:

1. $t_{out} < t_s$: Es wird lediglich der Halbraum weiter untersucht, welcher näher zum Strahlursprung liegt. Das betrachtete Strahlintervall bleibt $[t_{in}, t_{out}]$

2. $t_s < t_{in}$: Lediglich der hintere Halbraum muss weiter untersucht werden. Das betrachtete Strahlintervall bleibt ebenfalls $[t_{in}, t_{out}]$

3. $t_{in} < t_s < t_{out}$: Der Strahl schneidet die Splitting Plane und damit beide Halbräume. Findet sich im vorderen Halbraum mit dem Strahlintervall $[t_{in}, t_s]$ kein Schittpunkt, muss der hintere mit dem Strahlintervall $[t_s, t_{out}]$ weiter untersucht werden.

Die einfachste Form eine Splitting-Plane auszuwählen, ist entlang der Mitte eines jeden Knotens mit jeweils alternierenden Achsen. Bessere Ergebnisse werden meist durch Anwendung der *Surface Area Heuristik*[3] erreicht. Welche jedoch verhältnismässig aufwendig zu bestimmen ist.

Auch dieses Verfahren wurde bereits erfolgreich von Wald *et al.* in einem auf PC-Clustern basierenden Ray Tracer mit interaktiven Frameraten eingesetzt [Wal04]

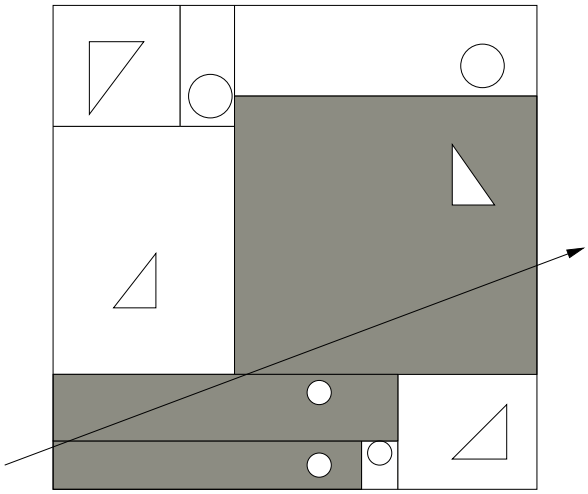


Abbildung 2.7: 2D-Darstellung eines k-d-Baumes

Direktionale Techniken

Anders als die bisherigen Verfahren verwenden Direktionale Techniken zudem die Richtung des Strahles für ihre Szenenunterteilung. Die Varianten dabei sind sehr groß und vielfältig und sollen nur kurz erläutert werden. Eine genauere Beschreibung findet sich etwa in [AK89].

[3] vgl. Abschnitt 4.1.1

Da Schattenfühler die wohl am häufigsten erzeugten Strahlen während der Bildsynthese mittels Ray Tracing sind, entwickelten Haines und Greenberg den so genannten *Light Buffer*[HG86]. Dabei wird um jede Lichtquelle ein uniform unterteilter *Direction Cube* gelegt. Für jede Blickrichtung von der Lichtquelle durch eine der Facetten des Direction Cubes kann so eine Kandidatenliste erstellt werden, welche als Blocker für diese Lichtquelle dienen könnten und nur diese werden entsprechend auf Schnittpunkte getestet.

Ohta und Maekawa [OM87] entwickelten den so genannten *Ray Coherence Algorithm*. Dabei werden Richtungsgrenzen berechnet, welche ein Strahl haben darf, der an einer Oberfläche eines Objektes beginnt und ein anderes Objekt treffen soll. Erfüllt der Strahl die Bedingungen nicht, verfehlt er das andere Objekt. Das ganze wird diskretisiert, indem um jedes Objekt ein Direction Cube gelegt wird.

Eine etwas andere Variante bietet das *Ray Classification* Verfahren von Arvo und Kirk [AK87]. Dieses basiert auf der Tatsache, dass ein Strahl in einem fünfdimensionalen Raum als Punkt repräsentiert werden kann, nämlich durch seine drei Ursprungskoordinaten sowie zwei Richtungskomponenten. Die Szene selbst wird in so genannte *5D-Hypercubes* unterteilt, was im Wesentlichen einem 3D-Voxel entspricht, der in eine Blickrichtung geöffnet ist, ähnlich einer Pyramide ohne Boden. Der Aufbau erfolgt dabei vergleichbar zu einem Octree, nur dass jeder Knoten 32 Kinder besitzt (2^5). Anhand der Strahlrepräsentation lässt sich so sehr einfach der entsprechende Hypercube finden, welcher den Strahl umschließt. Jedem Hypercube wird eine Liste der potenziellen Objekte zugewiesen, die dieser enthält und nur gegen diese muss der Strahl getestet werden.

Problematisch an den Direktionalen Techniken ist vor allem ihr hoher Speicherverbrauch, weswegen auch häufig nur Teile der Hierarchie aufgebaut und wieder verworfen werden und sie zudem inzwischen nur noch eine meist untergeordnete Rolle in den verwendeten Beschleunigungstechniken spielen.

2.2.2 Reduzierung der Strahlanzahl

Auch wenn mit dem Ray Tracing Modell nach Whitted [Whi80] bereits eine starke Reduzierung der Strahlen vorgenommen wurde, da pro Pixel nur noch ein Primärstrahl versendet wird, sind in den letzten Jahren Techniken entwickelt worden, die dies noch weiter treiben, immer mit der Prämisse, dass etwaige Qualitätsverluste bei der Bilddarstellung in Kauf genommen werden. Einige davon sollen im Folgenden vorgestellt werden.

Adaptives Sampling

Anstatt durch jedes Pixel einen Sehstrahl zu schicken, besteht natürlich auch die Möglichkeit nur jedes zweite, dritte, oder fünfte Pixel zu berechnen und den Rest durch Interpolation der Farbwerte zu füllen, oder den alten Farbwert beizubehalten, dies wird z.B. beim *Frameless Rendering* von Bishop *et al.* durchgeführt [BFMZ94]. Das Bild erscheint bei starker Bewegung zunächst etwas verwaschener, konvergiert allerdings recht bald, sobald die Bewegung nachlässt. Zudem kann unangenehm auffallendes Ruckeln, bedingt durch eine sonst geringe Framezahl, vermieden werden.

Die Qualität der Bilder bei Bewegung wird jedoch sehr stark beeinträchtigt, weswegen eigentlich trotz der dramatischen Reduzierung der Renderingzeit in der Regel eher davon abgeraten werden sollte. Allerdings existieren Ansätze, welche die Primärstrahlen geschickter auswählen und somit eine höhere Qualität liefern, wodurch adaptives Sampling wiederum zu einer sehr mächtigen Methode werden kann, siehe etwa [DWWL05].

Render Cache

Ebenfalls weniger Primärstrahlen werden beim *Render Cache* Verfahren erzeugt [WDP99]. Bei dieser Methode werden die gewonnenen Samples aus dem vorherigen Frame wieder auf die Projektionsebene rückprojiziert. Die restlichen, noch nicht gefüllten Pixel werden mit aufwändigen Heuristiken berechnet, oder neue Primärstrahlen erzeugt. Dies führt zwar zu einem enormen Geschwindigkeitsgewinn. Allerdings leidet die Qualität recht deutlich. Sollte keine Änderung an der Szene oder der Blickposition stattfinden, so konvergiert das Bild jedoch recht bald gegen die exakte Lösung.

Radiance Interpolants

Bala *et al.* [BDT99] haben ein Ray Tracing System entworfen, welches für sichtbare Oberflächen des Bildes so genannte *Interpolanten* speichert, welche genutzt werden um die Radiance im Bild zu approximieren. Shading und Sichtbarkeitsprüfungen werden dabei getrennt voneinander durchgeführt. Der Vorgang des Shadings wird durch quadrilineare Interpolation von bei Bedarf erworbenen Radiance Samples realisiert. Der dabei entstehende Fehler wird verkleinert, indem an Stellen mit Diskontinuitäten und Nicht-Linearitäten in der Radiancefunktion adaptiv neue Samples erworben werden. Die Sichtbarkeit wird mittels Rückprojektion der Interpolanten ermittelt, welche für jedes

Objekt in so genannten Linetrees gespeichert werden, einer Art 4D-äquivalent eines Octrees.

Local Illumination Environments

Steigt die Anzahl an Lichtquellen, so können Schattenstrahlen schnell einen Großteil der versendeten Strahlen ausmachen. Für viele Regionen sind jedoch stets nur die gleichen Lichtquellen sichtbar. Diese Feststellung machen sich Fernandez *et al.* mit ihren *Local Illumination Environments* zunutze [FBG02]. Sie unterteilen die Szene mittels eines Octrees und speichern für jede Zelle, ob eine Lichtquelle völlig verdeckt, teilweise verdeckt wird oder sichtbar ist, inklusive möglicher Objekte, die eine Verdeckung hervorrufen. Dadurch können die meisten Schattenstrahlen eingespart werden. Zudem muss bei partieller Verdeckung nur noch die Liste möglicher Okkluder abgearbeitet werden. Die Listen werden erstellt, indem einige ausgewählte Samples pro Zelle verschickt werden. Dadurch ist das Ergebnis nur approximiert, der Fehler jedoch meist sehr gering.

Der Vorverarbeitungsaufwand ist jedoch sehr groß, weswegen häufig auf eine *Lazy Evaluation* Technik gesetzt wird, bei der die Berechnungen für die benötigten Zellen zur Laufzeit durchgeführt werden.

2.2.3 Generalisierte Strahlen

Interessanterweise wurde lange Zeit das Konzept des Ray Tracings mit einzelnen Strahlen nicht in Frage gestellt. Erst mit Heckberts und Hanrahans *Beam Tracing* [HH84] und Amanatides *Cone Tracing* [Ama84], wurden erstmals Ansätze präsentiert, die sich von der Einzelstrahlverfolgung lösten und stattdessen mehrere Strahlen in einem Frustum kombinierten. Allerdings mussten dafür Abstriche bei der Genauigkeit der Schnittpunktberechnung gemacht werden, sowie bei deren Simplizität. Um die Genauigkeit wieder herzustellen wurde von Reshetov *et al.* [RSH05] ein Verfahren verwendet, welches zwar im Prinzip auch Bündel von Strahlen verfolgt, diese aber später in einzelne Strahlen aufbricht, um so die Exaktheit der berechnete Lösung zu erhalten.

2.2.4 Parallelisierung

Der Ray Tracing Algorithmus an sich ist nahezu ideal geeignet für Parallelisierung verschiedenster Art, da jeder Strahl unabhängig von allen anderen

berechnet werden kann. Dennoch müssen die Systeme, welche auf Parallelisierung aufsetzen, mit Bedacht konzipiert sein, um typische Engpässe zu umgehen, wie etwa eine möglichst gute Auslastung aller beteiligten Komponenten, Latenzzeiten bei der Datenübertragung in Netzwerken, etc..

Besonders beliebte Varianten sind dabei insbesondere die Ausnutzung von SIMD[4]-Befehlssätzen, sowie Rechnercluster, also der Zusammenschluss mehrerer Rechner in einem Netzwerk und in Zukunft wohl die Ausnutzung von Mehrprozessorsystemen, wie den neuen Dual-Cores.

2.2.5 Hardware Unterstützung

Vielversprechende Ansätze zeigen sich auch bei der Ausnutzung spezifischer Hardware. So entwickelten Schmittler *et al.* [SWS02] mit ihrem *SaarCOR* eine Hardware Architektur, welche speziell auf die Anforderungen des Ray Tracings zugeschnitten wurde. Damit konnten beeindruckende Ergebnisse erzielt werden, die in ihrer Anzahl an Frames üblicher Rasterisierungshardware in nichts nachstand und zusätzlich alle Vorteile des Ray Tracings bot. Einen Schritt weiter ging die Entwicklung in [SWW+04], bei der ein FPGA Chip verwendet wurde und sogar dynamische Szenen dargestellt werden konnten.

Einen ebenfalls interessanten Ansatz bieten GPU[5]-basierte Ray Tracer oder auch hybride Systeme, die sowohl GPU als auch CPU verwenden. Die Geschwindigkeit heutiger Graphikkarten verdoppelt sich nahezu alle sechs bis zwölf Monate und schlägt damit momentan die Entwicklung der CPUs[6]. Zudem scheinen GPUs mit ihren stark auf parallele Berechnungen ausgelegte Hardwarestruktur geradezu prädestiniert zu sein. Purcell *et al.* [PBMH02] waren wohl die ersten, denen es gelang, den gesamten Ray Tracing Prozess in einem Fragment-Programm zu implementieren und damit nahezu komplett auf der GPU auszuführen. Leider bieten aber heutige GPUs einige Nachteile, wie etwa stark beschränkten Speicher oder Maximallängen von Fragment-Programmen.

Ob und inwiefern GPUs sinnvoll und gezielt für den Ray Tracing Vorgang einzusetzen sind, ist noch ein relativ offenes Feld, die Ansätze sind jedoch vielversprechend.

[4]SIMD = Single Instruction Multiple Data
[5]GPU = Graphics Processing Unit
[6]CPU = Central Processing Unit

Kapitel 3

Stand der Technik

Obwohl es bereits eine geraume Menge an Methoden und Algorithmen gibt, welche den Ray Tracing Vorgang beschleunigen, so sind diese allesamt auf die Berechnung einfacher *walkthroughs* oder sogar einzelner Bilder ausgelegt, wobei wenig Aufmerksamkeit auf die Zeit gelegt wurde, welche zur Erstellung der Beschleunigungsstruktur notwendig war. Somit stellt Ray Tracing von dynamischen Szenen ein noch relativ unerforschtes Feld dar. Allgemein scheint es schwierig, Datenstrukturen zu finden, die geeignet sind komplexe, dynamische Inhalte zu verwalten. Es beschäftigt sich zumindest verhältnismässig wenig Literatur mit diesem Thema.

Im Bereich der auf Rasterisierungstechniken basierenden Renderern sieht es nicht viel anders aus. Meist wird vorab eine Trennung zwischen statischen und dynamischen Bereichen der Szene vorgenommen. Erstere werden in einem Vorverarbeitungsschritt zu einer möglichst guten Hierarchie zusammengesetzt, welche für ein beschleunigtes View Frustum Culling verwendet werden kann, sei es auf Basis eines Octrees, Quadtrees, BSP- oder k-d-Bäumen, oder Bounding Volume Hierarchien. Die dynamischen Bereiche der Szene werden dabei jedoch meist separat behandelt, indem jedes dynamische Objekt von einem einzigen Hüllvolumen umgeben wird, welche dann der Reihe nach gegen das View Frustum getestet werden [Ada02]. Dieses mag für Rasterisierungsverfahren ausreichend sein, führt aber beim Ray Tracing schnell zu einem Flaschenhals, da jeder Strahl gegen jedes dynamische Objekt getestet werden müsste.

Interessanterweise ist aber gerade dies der Ansatz, der von Parker *et al.* für ihren interaktiven Ray Tracer gewählt wurde, welcher auf einem Shared Memory Supercomputer lief [PMS+99]. Man muss ihnen allerdings zugute halten, dass es auch weniger in ihrem Interesse lag, komplexe dynamische Szenen

darzustellen, als überhaupt interaktives Ray Tracing mit interaktiver Bildwiederholfrequenz darstellen zu können. Somit beschränkte sich die Anzahl der möglichen bewegten Objekte auch auf einige wenige (≤ 10).

Reinhard *et al.* [RSH00] verwenden hierarchische Grids zum Ray Tracen dynamischer Szenen. Im Prinzip handelt es sich dabei um einen ausbalancierten Octree, der Objekte abhängig von ihrer Größe in verschiedenen Leveln der Hierarchie hält. Dadurch wird gewährleistet, dass jedes von ihnen eine ungefähr konstante Anzahl an Voxeln belegt, was ein Einfügen und Entfernen von Objekten in nahezu konstanter Zeit erlaubt. Um auch Ausdehnungen der Szene zu erlauben, unterscheiden Reinhard *et al.* zwischen der logischen und physischen Ausdehnung der Szene. Verlässt ein Objekt das physische Hüllvolumen der Szene, so findet ein *wrap around* statt, bei dem das Objekt auf der anderen Seite der Datenstruktur wieder eingefügt wird. Verlässt ein Strahl während des Ray Traversals das physische Hüllvolumen der Szene, so wird dieses ebenso auf der gegenüberliegenden Seite der Beschleunigungsstruktur wieder fortgesetzt, sofern das logische Grid noch nicht verlassen ist. Trifft der Strahl auf ein mit einem Objekt gefülltes Voxel während des Traversals, so wird getestet, ob die logischen Koordinaten von Strahl und Objekt übereinstimmen, sonst darf das Objekt selbst nicht getestet werden. Wenn sich das logische Hüllvolumen zu sehr ausdehnt, muss von Zeit zu Zeit ein kompletter Neuaufbau der Beschleunigungsstruktur stattfinden.

Lext und Akenine Möller [LAM01b] versuchen die Informationen des Szenegraphens zu nutzen, indem sie die statischen und dynamischen Bereiche auftrennen und jedes so entstandene Objekt mit einem beliebig orientierten Hüllvolumen umgeben, welche jeweils ein Recursive Grid enthalten. Innerhalb dieser Objekte muss die Datenstruktur nur einmalig aufgebaut werden, da sie alle derselben Transformation unterliegen und sich somit nicht relativ zueinander bewegen. Strahlen, welche das äußere Hüllvolumen schneiden, werden in das lokale Koordinatensystem des Objektes transformiert. Aus diesen Objekten wird entsprechend dem Szenegraphen eine Hierarchie erstellt, wobei jeder Transformationsknoten wiederum ein Recursive Grid darstellt, welches die Kinder enthält. In der Aktualisierungsphase zwischen den einzelnen Bildberechnungen, werden die Hüllvolumen der Recursive Grids angepasst und diese gegebenenfalls neu aufgebaut. Damit konnte die Rekonstruktionsphase, verglichen mit einem einfachen Recursive Grid, etwa um den Faktor 10 beschleunigt werden. Da jedoch die direkte Umsetzung des Szenegraphen mitunter zu schlechten Ergebnissen führt, schlagen sie zusätzlich vor, auf Lazy Evaluation Techniken zu setzen, welche lediglich die Bereiche der Szene aktualisieren, welche während der Traversierung durchlaufen werden. Diese Idee wurde jedoch letzten Endes nicht von ihnen umgesetzt.

Eine erste Umsetzung einer Lazy Evaluation Technik findet sich bei McNeill *et al.* [MSMP+92]. Diese verwenden einen Octree, bei dem zunächst nur die obersten Ebenen aufgebaut werden. Die unteren werden nur bei Bedarf erzeugt. Sie erwähnen zwar, dass ihr Ansatz auch für dynamische Szenen zu gebrauchen wäre, testen ihn aber nur für statische Szenen.

Larsson und Akenine-Möller [LAM03] hingegen machen starken Gebrauch von dieser Technik, um deformierbare Objekte zu Ray Tracen. Dabei verwenden sie eine vorab angelegte BVH mit einem hybriden Bottom-Up/Top-Down Update-Mechanismus. Für jeden Frame werden zunächst nur die Hüllvolumen der Ebene $d = \lfloor \frac{h}{2} \rfloor$ an die in ihrem Teilbaum enthaltenen Primitive angepasst, wobei h die Höhe der Hierarchie darstellt. Anschließend werden die Hüllvolumen oberhalb dieses Levels angepasst, bis hin zum Wurzelknoten. Während des Ray Traversals wird der Rest der Hierarchie nur bei Bedarf in Top/Down Manier aufgebaut. Jeder Knoten umschließt stets die gleichen Primitive, wodurch sich die Struktur der Hierarchie nicht ändert, was eine effizientere Traversierung mittels eines 1D-Arrays erlaubt, gleichzeitig aber auch die möglichen Bewegungen stark einschränkt, ohne dass es zu starken Qualitätsverlusten der Hierarchie kommt. Da sie jedoch lediglich deformierbare Objekte darstellen wollen, ist dies ohne größeren Belang.

Wald *et al.* [WBS03] präsentierten eine Methode dynamische Szenen zu bewältigen, indem sie zunächst eine Klassifizierung der in der Szene enthaltenen Objekte vornehmen in statische Bereiche, Objekte, die affinen Transformationen unterliegen und unstrukturiert animierte Objekte. Für jedes Objekt beliebiger Komplexität, sowie für die statische Szene, wird in einem Vorverarbeitungsschritt jeweils eine separate Beschleunigungsstruktur erstellt. Ray Traversal zwischen diesen Objekten, wird mittels eines Top-Level BSP-Baumes durchgeführt, welcher allerdings für jeden Frame neu aufgebaut wird, sollte eine Bewegung stattfinden. Das gleiche wird für Objekte mit unstrukturierter Bewegung durchgeführt. Trifft dabei ein Strahl auf das Hüllvolumen eines Objektes, wird dieser in das entsprechende lokale Koordinatensystem transformiert und dort weiter verfolgt.

Schmittler *et al.* nutzten dieses Vorgehen auch für eine Hardwareimplementation eines Ray Tracers mittels eines FPGA Chips [SWW+04]. Dieser erreicht, obwohl lediglich mit 90 MHz getaktet, Frameraten zwischen 15 und 90 Frames pro Sekunde, bei einer Auflösung von immerhin 512×384 Pixeln und lediglich Primärstrahlen, auch bei ein paar tausend bewegten Objekten.

Bereits relativ früh hat Glassner mit seinem *Spacetime Ray Tracing* [Gla88] eine Methode entworfen, welche dafür gedacht ist Motion Blur und relativistische Effekte darzustellen, gleichzeitig aber auch das Rendern von Animatio-

nen beschleunigt. Dafür dienen zwei zentrale Konzepte: Spacetime Ray Tracing und eine hybride Raumunterteilungs/Bounding Volume Technik, welche eine Hierarchie aus nicht überlappenden Hüllvolumen erstellt. Für letztere wird die Szene zunächst in Form eines normalen Octrees unterteilt. Die einzelnen Voxel werden dann an die in ihnen enthaltenen Primitive angepasst und gegebenenfalls an den alten Voxelgrenzen geclippt, so dass es zu keiner Überlappung kommen kann. Beim Spacetime Ray Tracing wird dieses Verfahren nun auf eine vierdimensionale Szene angewandt, wobei die vierte Dimension die Zeit darstellt. Es findet quasi ein Ray Tracing von statischen 4D-Objekten statt, anstelle von dynamischen 3D-Objekten. Der Vorteil liegt darin, dass die Beschleunigungsstruktur nur ein einziges Mal erstellt werden muss, statt für jeden Frame aufs neue. Der Nachteil andererseits liegt darin, dass die Bewegung der Objekte vorab bekannt sein muss, was dieses Verfahren für nicht-deterministische Bewegungen unbrauchbar macht.

Etwas weniger einschränkend ist der Ansatz der *Temporary Bounding Volumes* (TBVs) von Sudarsky und Gotsman [SG96]. Diese TBVs sind im Prinzip einfache Hüllvolumen, welche ein Objekt nur lose umgeben und damit garantieren, dass sich das Objekt für eine bestimmte Anzahl an Frames innerhalb dieses TBVs aufhalten wird. Die TBVs selbst werden dann in einen Octree einsortiert und verwendet, solange eine festgelegte Zeitspanne nicht überschritten wird, oder das TBV sichtbar wird. Insgesamt sparen TBVs nicht nur Aktualisierungen der Datenstruktur, sondern ersparen auch die Berechnung der Animation selbst. Problematisch für eine Verwendung im Ray Tracing Kontext wäre dabei allerdings, dass die Vergrößerung des Hüllvolumens auch in gleichem Maße die Trefferwahrscheinlichkeit und damit den Aufwand erhöht. Hinzukommt, dass man für eine effiziente Nutzung Vorwissen über die Geschwindigkeit benötigt.

Erwähnenswert ist ebenfalls der Ansatz, der von Ulrich [Ulr00] vorgestellten *Loose Octrees*, auch wenn dieser noch nicht für Ray Tracing getestet wurde, sondern nur für View Frustum Culling und Kollisionserkennung. Die Loose Octrees sind dabei eine Variante der normalen Octrees, welche ein Einfügen in $O(1)$ erlauben und das Problem der „klebrigen“ Ebenen vermeidet. Dieses entsteht, wenn ein Objekt genau auf der Trennlinie zwischen zwei oder mehr Voxeln liegt. Um zu verhindern, dass es in mehr als einen Voxel eingefügt wird, würde das Objekt in der Regel in dem größten es umschließenden Voxel gespeichert, was jedoch dazu führt, dass viele kleine Objekte in sehr großen Knoten gespeichert werden könnten. Die Loosen Octrees vermeiden dieses Problem, indem sie die Hüllvolumen der einzelnen Voxel vergrößern, so dass sie sich überlappen. Dadurch können die Objekte tiefer in der Hierarchie eingefügt werden, was zu präziseren räumlichen Anfragen führt. Diese

starke Überlappung sorgt allerdings auch dafür, dass der Vorteil gegenüber konventionellen Verfahren des View-Frustum Cullings nicht übermässig groß, aber doch merklich ausfällt. Bei Objekt-zu-Objekt-Abfragen, wie z.B. Kollisionserkennung, hat er sich jedoch als deutlich effektiver gegenüber einem normalen Octree erwiesen.

Kapitel 4

Dynamische Bounding Volume Hierarchien

Das nun folgende Kapitel beschäftigt sich mit der Fragestellung, inwiefern BVHs für dynamische Szenen nutzbar gemacht werden können. Auch wenn Larsson und Akenine-Möller bereits einen ersten Ansatz zur Verwendung von Bounding Volume Hierarchien für dynamische Szenen gegeben haben [LAM03], so ist dieser bei weitem nicht ausreichend gewesen, um auch beliebige, nicht-deterministische Bewegungen darstellen zu können. Es stellt sich also die Frage, ob nicht andere Wege existieren, Bounding Volume Hierarchien hierfür zu adaptieren. Die Voraussetzungen dafür sind recht vielversprechend.

- BVH sind mächtig genug, um, richtig eingesetzt, auch interaktive Frameraten auf einem einzelnen Rechner zu erlauben [Gei05].
- Die Ausmaße der BVs werden in der Regel durch die darin enthaltenen Objekte bestimmt, so dass dadurch implizit ein Mittel gegeben ist, um auch mit sich ausdehnenden Szenen umgehen zu können, ohne die gesamte Hierarchie zwangsweise neu erstellen zu müssen, wie es in vielen anderen Ansätzen der Fall ist.
- Anders als die rein spatialen Datenstrukturen, hat eine BVH, sofern sie gut erstellt wurde, keine Probleme mit leeren Räumen, die traversiert werden müssen. Dass dies nicht zu unterschätzen ist, zeigen Arbeiten wie von Subramanian und Fussel [SF92], welche BVs als Ergänzung zu ihren k-d-Bäumen verwendeten, sowie in neuerer Zeit von Reshetov *et al.* [RSH05].

- Der Fakt, dass Objekte in BVHs nur genau einem Knoten zugeordnet sind, erleichtert zudem die für dynamische Szenen benötigten Lösch- und Einfügeprozeduren. Denn ein einfacher Zeiger pro Objekt genügt um die Position der Objekte in der Hierarchie aufzufinden.

Nach einer detaillierteren Beschreibung von BVHs, die auch die Wahl des Hüllvolumens umschließt, bekannte Verfahren zur Erstellung erläutert, sowie die gebräuchlichsten Traversierungsmethoden beschreibt, sollen in Abschnitt 4.2 verschiedene Heuristiken vorgestellt werden, welche als Qualitätskriterium für einen Knoten in der BVH dienen sollen und zu erkennen geben, wann ein Teilbaum neu aufgebaut werden sollte, oder zumindest Änderungen an diesem vorgenommen werden sollten. Direkt daran anschließend werden schließlich die in dieser Arbeit entwickelten Verfahren zur Rekonstruktion der Hierarchie erläutert, sowie eine Erweiterung zur Behandlung von hierarchischen Animationen vorgestellt.

4.1 Bounding Volume Hierarchien

Da im Ray Tracing ein Brute-Force Ansatz, bei dem jeder Strahl mit jedem Objekt der Szene geschnitten wird, schnell zu untragbaren Renderingzeiten führt, wurde bald erkannt, dass man bereits eine deutliche Steigerung der Rechengeschwindigkeit erzielen kann, indem man die Objekte mit einfacheren *Hüllvolumen* (*engl. Bounding Volume* (BV)) umgibt [Whi80]. Diese sollten das Objekt möglich eng umschließen, um unnötige Strahltests mit dem inneliegenden Objekt zu vermeiden, und zum anderen einen deutlich einfacheren Schnitttest erlauben, um dem Mehraufwand gerecht zu werden. Weghorst *et al.* haben dafür eine Faustformel entwickelt, welche die Kosten für den Schnitttest mit einem Objekt und seinem Hüllvolumen angibt [WHG84]. Diese lautet

$$T = bB + iI \quad , \tag{4.1}$$

wobei T die Gesamtkosten angibt, b die Anzahl, wie oft das Hüllvolumen auf Schnitte getestet wird, B die Kosten für einen Schnitttest mit dem Hüllvolumen, i die Anzahl, wie oft das Objekt auf Schnitte getestet wird, sowie I die Kosten das Objekt selbst zu testen. Da dies jedoch vorab nicht bekannt ist, kann man nur auf empirische Daten zurückgreifen. In der Literatur haben sich im Ray Tracing Kontext dabei die achsenparallelen Boxen (*engl. axis-aligned bounding box* (AABB)) oft als primäre Wahl erwiesen [Smi98] [Gei05]. Andere durchaus beliebte Möglichkeiten stellen aber auch Kugeln, orientierte Boxen, oder *k-DOPs*[KK86] dar.

In dieser Arbeit habe ich mich für die achsenparallelen Boxen entschieden. Diese haben verschiedene Vorteile gegenüber anderen Hüllvolumen. Sie können sehr effizient gespeichert werden, umschließen die meisten gängigen Objekte relativ eng, und der Schnittest mit einem Strahl ist sehr effizient durchführbar. Zudem erlaubt die folgende Eigenschaft ein sehr simples und effizientes Anpassen einer AABB an seine Kinder in einer BVH. Sei P eine Menge von Primitiven oder Objekten und P^+, sowie P^- jeweils Teilmengen von P mit

$$P^+ \cup P^- = P \quad ,$$

sowie B^+ die kleinstmögliche AABB welche P^+ umschließt, B^- in äquivalenter Weise P^-. Sei B die kleinstmögliche AABB, für welche gilt

$$B^+ \cup B^- = B \quad ,$$

dann ist B auch die kleinstmögliche AABB, welche P umfasst. Diese Eigenschaft teilen sich die AABBs, mit den sonst gebräuchlichen Hüllvolumen nur noch mit den k-DOPs. Auch das in Kapitel 5 vorgestellte Verfahren nutzt AABBs.

Nachdem erstmal der Nutzen von Hüllvolumen erkannt worden war, war es zu einer Bounding Volume Hierarchie nur noch ein kleiner Schritt. Die Idee einer BVH ist sehr simpel zu verstehen, aber sehr effizient, wenn sie korrekt eingesetzt wird. Wie bereits in Abschnitt 2.2.1 beschrieben, handelt es sich bei einer BVH genau genommen um einen gerichteten, azyklischen Graph, oder Baum, mit beliebigem Verzweigungsfaktor, der an seinen Blättern die Szenenobjekte trägt, in der Regel pro Blatt genau eines und diese bestmöglich umfasst. Jeder darüberliegende Vaterknoten im Graph wird nun in seinen Ausmaßen so gewählt, dass er seine Kinder möglichst eng umschließt. Die zu Grunde liegende Idee ist, dass man einen Strahl nicht auf jedes Szenenobjekt testet, sondern stattdessen damit beginnt ihn gegen den Wurzelknoten des Graphen zu testen. Schneidet der Strahl diesen, fährt man mit den Kindern fort. Schneidet er jedoch in der weiteren Strahlverfolgung einen Knoten nicht, so kann der gesamte darunterliegende Teilbaum außer Acht gelassen werden. Genau dieser Effekt ist es, der dafür sorgt, dass sich die Berechnungszeit drastisch verringern lässt, trotz des scheinbaren Mehraufwandes, der dadurch entsteht, dass man zusätzlich zu den Objekten noch die verschiedenen Hüllvolumen schneiden muss. In der Regel ist es jedoch so, dass ein Strahl, in einer auch nur etwas komplexeren Szene, nur einen Bruchteil der Objekte schneidet.

Zwar ist es manchmal von Vorteil die Graphstruktur der BVH in ein 1D Array umzukopieren, wie später noch erläutert wird. Doch der Anschaulich-

keit halber soll auch weiterhin jedes Element als Knoten bezeichnet und von seinem dazugehörigen Teilbaum gesprochen werden.

4.1.1 Aufbau einer Bounding Volume Hierarchie

Das Verfahren, welches die BVH erstellt, ist von entscheidender Bedeutung für die Qualität der Hierarchie. Ein zufälliges Zusammenfassen von Objekten könnte im schlimmsten Falle sogar den Ray Tracing Vorgang noch erheblich verlangsamen, im Vergleich zu einem einfachen Strahltest gegen sämtliche Objekte. Auf der anderen Seite kann eine intelligent erstellte Hierarchie die Renderingzeiten deutlich verringern, insbesondere wenn die Anzahl an Objekten in der Szene drastisch zunimmt. Dabei kann man sich häufig an die Faustregel halten: Objekte die eine räumliche Nähe zueinander aufweisen, sollten auch in den gleichen Teilbaum eingefügt werden.

In den ersten Versuchen mit BVHs wurden diese zumeist noch per Hand oder direkt aus dem Szenengraph erstellt [RW80]. Die Ergebnisse waren für die damalige Zeit zufriedenstellend, allerdings war man auch weit davon entfernt, einen Anspruch auf interaktive Bildwiederholraten zu stellen. Bessere Ergebnisse liefern die automatischen Verfahren, welche auf verschiedenen Heuristiken beruhen, um die Anzahl der Schnitttests mit einem Strahl zu minimieren. Die drei wichtigsten sollen dabei im folgenden vorgestellt werden.

Median-Cut

Bei diesem von Kay und Kajiya [KK86] vorgestellten Verfahren wird in einem Top-Down-Vorgang ein binärer, ausbalancierter Baum aufgebaut. Für diesen soll gelten, dass die Anzahl an Objekten, die im jeweils linken und rechten Teilbaum eines Knotens enthalten sind, sich um maximal eins unterscheiden dürfen. Zudem soll gelten, dass die Sortierschlüssel der Objekte, bezüglich eines vorab gewählten Sortierkriteriums, im linken Teilbaum allesamt kleiner als die im rechten Teilbaum sind. Meist wird dabei der Mittelpunkt der Objekte oder einer der Eckpunkte des sie umschließenden Hüllvolumens gewählt, welche entlang einer gewählten Achse sortiert wurden. Etwas mathematischer ist dies in Formel (4.2) ausgedrückt, welche für jeden Knoten der BVH erfüllt sein muss.

$$\forall x \in T_l : \forall y \in T_r : (x.key \leq y.key) \quad , \tag{4.2}$$

wobei T_l die Menge aller Elemente im linken Teilbaum des Knotens ist, respektive im rechten Teilbaum für T_r, x bezeichnet ein Element aus T_l, y eines

aus T_r. *key* ist dabei der Sortierschlüssel nach dem sortiert wird. Dieser ist in unseren Verfahren der jeweilige Minimalwert, bzw. Maximalwert des Hüllvolumens für die momentan betrachtete Achse des Weltkoordinatensystems. Häufig wird auch empfohlen, den Mittelpunkt des BV zu wählen, jedoch mathematisch gesehen läßt sich kein Grund dafür finden, und geometrisch lassen sich für jeden gewählten Referenzpunkt Beispiele finden, in denen ein anderer besser gewesen wäre. Zudem sind so die für die Sortierung benötigten Werte bereits vorhanden und müssen nicht separat gespeichert oder berechnet werden, was wiederum auch einen Geschwindigkeitsvorteil beim Aufbau mit sich bringt.

Das Anlegen der Beschleunigungsstruktur ist dabei eine einfache rekursive Prozedur. Gegeben eine Liste an Objekten, bzw. deren Hüllvolumen: Berechne das kleinste BV, welches alle Objekte umschließt und weise es dem aktuellen Knoten zu. Teile die Liste in zwei Hälften und verfahre genauso mit ihnen bei den Kindern des Knotens. Der Vorgang wird rekursiv fortgesetzt, bis ein Knoten nur noch ein Objekt enthält. Um die Überschneidung zwischen zwei Geschwisterknoten möglichst gering zu halten, werden die Objekte in jedem Rekursionsschritt entlang einer beliebigen Achse sortiert, bevor sie unterteilt werden. Eine gängige Methode ist es, eine der drei Achsen des Weltkoordinatensystems zu wählen und in jedem Rekursionsschritt zwischen den Achsen zu alternieren, oder immer entlang der längsten Achse zu unterteilen. Bei halbwegs uniformer Verteilung und Größe der Objekte, sind mit diesem Verfahren beeindruckende Geschwindigkeitssteigerungen zu erreichen und auch der Aufbau kann trotz seiner relativ hohen Komplexität von $O(n^2)$, bedingt durch die Sortierverfahren (optimiert $O(n \log n)$ im Mittel), verhältnismäßig schnell durchgeführt werden. Abbildung 4.1 gibt das Verfahren in seiner einfachsten Variante als Pseudocode wieder.

Schwierigkeiten zeigen sich bei diesem Verfahren allerdings in zwei Fällen: Erstens, wenn Objekte sehr groß sind, und zweitens, wenn sich große Lücken zwischen den Objekten befinden. Da der Ray Tracing Vorgang schneller ist, je mehr Geometrie in jedem Schritt verworfen werden kann, ist es verständlich, dass dies umso besser geschieht, je kleiner und weniger überlappend die Kinder eines untersuchten Knotens sind. Eine Herleitung dieser Tatsache findet sich im Unterabschnitt 4.1.1 Goldsmith und Salmon. Große Objekte verhindern dies jedoch und verstopfen somit die Hierarchie. Der zweite Schwachpunkt ist eine inhomogene Verteilung. Das Verfahren nimmt keinerlei Rücksicht auf die genaue räumliche Verteilung der Objekte. Man stelle sich etwa eine Szene vor, die aus zwei weit voneinander entfernten Ansammlungen von Objekten besteht. Die natürliche Art diese aufzuteilen wäre, alle Objekte der einen Ansammlung zusammenzufassen, sowie alle der anderen.

Da der Median-Cut Algorithmus aber streng an die Anzahl der Objekte gebunden ist, kann es sein, dass die kleinere Ansammlung noch mit Teilen der größeren zusammengefasst wird und Strahlen, die zwischen beiden verlaufen, gegen unnötig viele nahezu leere Hüllvolumen getestet werden müssen.

Eine Variante, welche ebenfalls manchmal eingesetzt wird, teilt nicht die Objekte sondern den Raum entlang einer Achse [SM03]. Dadurch entsteht zwar meist ein besserer Umgang mit leeren Räumen, die BVH ist jedoch deutlich unausgeglichener. Der Vor- oder Nachteil ist somit immer szenenabhängig.

```
BoundingVolume buildHierarchy(BVNodeArray bvList, char axis,
                              int startnr, int endnr)
{
  if(endnr == startnr) //just one BV left
    return bvList[start];
  BoundingVolume parent;
  for each BoundingVolume bv in bvList
    expand parent to enclose bv;
  sort bvList along axis;
  axis = next axis;

  // recurse on the next level
  parent.addChild(buildHierarchy(bvList,
                  startnr, (startnr+endnr)/2, axis));
  parent.addChild(buildHierarchy(bvList,
                  1+(startnr+endnr)/2, endnr, axis);
  return parent;
}
```

Abbildung 4.1: Pseudocode zur Erstellung einer Bounding Volume Hierarchie, basierend auf dem Median-Cut Algorithmus von Kay/Kajiya [KK86]

Surface Area Heuristik

Ein anderes Verfahren, welches ebenfalls eine binäre BVH erzeugt, ist die so genannte Surface Area Heuristik, welche von MacDonald und Booth eingeführt wurde [MB90]. Auch hier wird die BVH Top-Down aufgebaut, wobei in jedem Schritt versucht wird folgende Kostenfunktion zu minimieren:

$$C_d(T) = \frac{S(T_l)}{S(T_{\text{root}})} \cdot \sum_{o_i \in T_l} C_{obj}(o_i) + \frac{S(T_r)}{S(T_{\text{root}})} \cdot \sum_{o_j \in T_r} C_{obj}(o_j) \quad , \tag{4.3}$$

wobei $C_d(T)$ die Kostenfunktion ist, welche für jeden Knoten minimiert werden soll, die Objekte sind dabei entlang der Achse d sortiert, mit $d \in x, y, z$. $S(X)$ ist die Oberfläche des Hüllvolumens X, T der Knoten, dessen Kosten optimiert werden sollen, T_l das linke Kind von Knoten T und T_r das rechte Kind von Knoten T. T_{root} stellt den Wurzelknoten der Hierarchie dar. $C_{obj}(o)$ ist eine Funktion, welche die Kosten berechnet einen beliebigen Strahl mit dem Primitiv o zu schneiden. Wird nur eine Art Primitiv unterstützt, können diese Kosten auf 1 gesetzt werden und entsprechen damit genau der Anzahl an Objekten in diesem Knoten, was häufiger auch zur Vereinfachung angewandt wird. Bei diesem Verfahren werden somit kleine Hüllvolumen mit möglichst vielen Objekten bevorzugt. Auch dieses Verfahren ist von quadratischer Komplexität, kann jedoch verbessert werden zu $O(n \log n)$ im Mittel, indem die Objektliste nur einmal entlang aller drei Achsen sortiert wird.

Allerdings wird diese Heuristik meist eher für k-d-Bäume eingesetzt und seltener für BVH [WBWS01], [RSH05], [MF99]. Hinzu kommt, dass diese Kostenfunktion leider nicht monoton verläuft und mehrere lokale Maxima und Minima aufweisen kann. Dies führt dazu, dass bei stärkeren Bewegungen der Objekte, die Hierarchie eigentlich komplett neu wieder aufgebaut werden müsste, wollte man die optimale Hierarchie erhalten. Dazu folgendes Beispiel: Ein sich sehr schnell bewegendes, sehr kleines Objekt befindet sich in der Mitte einer Szene aus relativ uniform verteilten, ähnlich großen Objekten. Die Struktur der Hierarchie wäre dann vergleichbar zu einer Median-Cut Hierarchie. Im nächsten Frame könnte sich das Objekt bereits so weit aus der Szene entfernt haben, dass nach Gleichung (4.3) die Kinder des Wurzelknotens aus eben diesem Objekt und dem gesamten Rest der Szene bestehen könnten. Dies würde man jedoch nur herausfinden können, indem die Szene komplett neu aufgebaut wird, was aber möglichst verhindert werden soll. Da es folglich mit großem Aufwand verbunden ist, die Kostenfunktion zu minimieren, nachdem sich Objekte in der Szene bewegt haben, wurde diese Methode in den weiteren Untersuchungen nicht näher betrachtet.

Goldsmith und Salmon

Als die bisher wohl erfolgreichste Heuristik im Zusammenhang mit BVHs bewährte sich die von Goldsmith und Salmon eingeführte Variante [GS87]. Bei diesem Verfahren wird ebenfalls versucht, die vermeintlich beste Stelle in der Hierarchie für jedes der sequentiell einzufügenden Objekte anhand

eines Kostenkriteriums zu finden. Die Idee dabei ist, dass die durchschnittliche Anzahl an Schnittpunkttests eines beliebigen Strahles mit der Hierarchie minimiert werden soll. Der durchschnittliche Aufwand von $O(n \log n)$ bei n Objekten ist zudem verhältnismäßig moderat. Da sowohl einige unserer in Abschnitt 4.2 angesprochenen Qualitätskriterien, sowie in Abschnitt 4.3 entwickelten Methoden, dieses Verfahren verwenden, oder darauf referenzieren, soll es hier etwas genauer vorgestellt werden.

Bei ihrem Kostenkriterium gehen Goldsmith und Salmon davon aus, dass die bedingte Wahrscheinlichkeit eines Strahles ein konvexes Hüllvolumen B zu schneiden, unter der Annahme, dass der Strahl bereits ein anderes konvexes Hüllvolumen A geschnitten hat, gerade $S(B)/S(A)$ ist, also das Verhältnis der Oberflächen zueinander. Dies ist wahr, solange folgendes gilt:

- $B \subseteq A$
- A und B sind konvex
- Die Menge aller Strahlen $\Re$ sind uniform verteilt.[1]

Sei gegeben, dass A und B konvexe Hüllvolumen sind mit $B \subseteq A$ und $\mathbf{R}(t)$ ist ein beliebiger Strahl aus $\Re$, der A schneidet, was mit $\mathbf{R}(t) \cap A$ beschrieben werden soll. Gesucht ist nun die Wahrscheinlichkeit, dass B von $\mathbf{R}(t)$ geschnitten wird, oder anders ausgedrückt $P(\mathbf{R}(t) \cap B \mid \mathbf{R}(t) \cap A)$. Nimmt man zunächst als Vereinfachung an, dass alle Strahlen parallel zu einem Vektor $\vec{\nu}$ verlaufen, so ist die Wahrscheinlichkeit, dass $\mathbf{R}(t)$ B schneidet genau das Verhältnis der in Richtung $\vec{\nu}$ projizierten Fläche $P_{\vec{\nu}}(B)$ von B zur projizierten Fläche $P_{\vec{\nu}}(A)$ von A. Die Fläche, auf welche projiziert wird, ist dabei als orthogonal zu $\vec{\nu}$ anzusehen. Mathematisch ausgedrückt

$$P(\mathbf{R}(t) \cap B \mid (\mathbf{R}(t) \cap A) \wedge (\mathbf{R}(t) || \vec{\nu})) = \frac{P_{\vec{\nu}}(B)}{P_{\vec{\nu}}(A)} \quad , \tag{4.4}$$

wobei $\mathbf{R}(t)||\vec{\nu}$ bedeutet, dass $\mathbf{R}(t)$ parallel zu $\vec{\nu}$ verläuft und $P_{\vec{\nu}}(X)$ die in Richtung $\vec{\nu}$ projizierte Fläche von X ist. Da allerdings die Möglichkeit bestehen soll, von beliebigen Strahlrichtungen ausgehen zu können, wird über alle möglichen Richtungen Ω integriert und man gelangt so zu

[1] Als eine praktische Umsetzung dessen,stelle man sich vor, dass die Ursprünge der Strahlen innerhalb einer großen, die Szene umschließenden Kugel liegen und die Richtungsvektoren uniform und unabhängig voneinander über die Einheitskugel verteilt sind [Arv90]

$$P(\mathbf{R}(t) \cap B \mid \mathbf{R}(t) \cap A) = \int_{\Omega} P(\mathbf{R}(t) \cap B \mid (\mathbf{R}(t) \cap A) \wedge (\mathbf{R}(t) || \vec{\nu})) f(\vec{\nu}) d\vec{\nu} \quad , \tag{4.5}$$

wobei $f(\vec{\nu})$ eine wahrscheinlichkeitsbasierte Dichtefunktion ist, die die Dichte der Strahlen in Richtung $\vec{\nu}$ angibt. Diese Dichte wiederum ist abhängig von der in Richtung $\vec{\nu}$ projizierten Fläche von A. Daraus ergibt sich für $f(\vec{\nu})$

$$f(\vec{\nu}) = \frac{P_{\vec{\nu}}(A)}{\int_{\Omega} P_{\vec{\nu}}(A) d\vec{\nu}} \tag{4.6}$$

Die Substitution von Gleichung (4.4) und (4.6) in Gleichung (4.5) führt zu

$$P(\mathbf{R}(t) \cap B \mid \mathbf{R}(t) \cap A) = \frac{\int_{\Omega} P_{\vec{\nu}}(B) d\vec{\nu}}{\int_{\Omega} P_{\nu}(A) d\vec{\nu}} \tag{4.7}$$

Da das Theorem gilt, dass die durchschnittliche projizierte Fläche eines konvexen Körpers ein Viertel seiner Oberfläche darstellt (siehe etwa [Arv90]) kann Gleichung (4.7) umgeformt werden zu

$$P(\mathbf{R}(t) \cap B \mid \mathbf{R}(t) \cap A) = \frac{S(B)}{S(A)} \quad , \tag{4.8}$$

wobei $S(X)$ die Oberfläche des konvexen Körpers X darstellt.

Gegeben sei nun ein Strahl $\mathbf{R}(t) \in \Re$, der den Wurzelknoten A einer BVH schneidet, und ein BV B, welches Teil der Hierarchie ist. Dann ist, wie oben angeführt, die bedingte Wahrscheinlichkeit, dass $\mathbf{R}(t)$ B schneidet, das Verhältnis der Oberfläche von B zur Oberfläche von A. Die Kosten eines Knotens der BVH sind dann seine bedingte Wahrscheinlichkeit multipliziert mit der Anzahl seiner Kinder, da jedes von ihnen ebenfalls auf Schnitte mit $\mathbf{R}(t)$ getestet werden muss, sobald $\mathbf{R}(t)$ B schneidet. Summiert man die Kosten für jeden Knoten auf, so erhält man die Gesamtkosten der Hierarchie, welche eben genau die durchschnittliche Anzahl an Schnittberechnungen für einen beliebigen Strahl $\mathbf{R}(t) \in \Re$ mit der Hierarchie ist. Das Ganze ist natürlich insofern eine vereinfachte Annahme, da alle Strahlen außerhalb des Wurzelknotens beginnen müssten, was vor allem bei Sekundärstrahlen nie der Fall ist. Es werden zudem nur Strahlen betrachtet, welche wirklich den Wurzelknoten schneiden, was aber keine großen Konsequenzen nach sich zieht, da alle anderen nur genau einen Schnitttest benötigen. Auch werden Faktoren

wie Okklusion, die eine Rolle spielen können, in keinster Weise mit einberechnet. Dennoch handelt es sich hierbei um eine äußerst wirksame Heuristik.

Bis jetzt wurde nur das Kostenkriterium erläutert, aber der eigentliche Aufbauvorgang ausgespart. Das soll an dieser Stelle nachgeholt werden. Gegeben eine Reihe von Objekten, so sollen diese sequentiell in eine BVH eingefügt werden, jeweils beginnend beim Wurzelknoten, von wo aus sie herabgereicht werden an ihre vermeintlich bestmögliche Position. Aus obiger Argumentation wissen wir, dass die Wahrscheinlichkeit eines Knotens von einem Strahl getroffen zu werden, proportional zu seiner Oberfläche ist und seine Kosten gerade die Oberfläche multipliziert mit der Anzahl seiner Kinder ist. Da alle Kosten relativ zueinander betrachtet werden, ist eine Division durch die Oberfläche des Wurzelknotens für das Verfahren unnötig. Darauf aufbauend versucht der Algorithmus von Goldsmith und Salmon, die Einfügeposition für ein Objekt zu finden, welche den geringsten Kostenzuwachs nach sich zieht. Beginnend beim Wurzelknoten wird entschieden, wie ein Objekt A eingefügt werden soll, oder ob A an die Kinder des aktuellen Knotens weitergereicht wird, wo die Prozedur rekursiv fortgesetzt wird. Insgesamt gibt es dabei die drei folgenden Möglichkeiten:

1. Das Objekt A wird mit dem aktuellen Knoten B zu einem neuen Bounding Volume B^* zusammengefasst (vgl. Abb. 4.2(a)). Die zusätzlichen Kosten C_{inc} sind dabei

$$C_{\text{inc}} = 2S(B^*) \quad , \tag{4.9}$$

   wobei $S(B^*)$ die Oberfläche des neu entstandenen Knotens ist, multipliziert mit 2, da der Knoten genau zwei Kinder hat.

2. Das Objekt A wird als neues Kind zum aktuellen Knoten B hinzugefügt (vgl. Abb. 4.2(b)). Die zusätzlichen Kosten errechnen sich aus der Differenz der neuen Kosten des Knotens

$$C_{\text{new}} = S(B')(k+1) \tag{4.10}$$

   und den vorherigen Kosten

$$C_{\text{old}} = S(B)k \quad , \tag{4.11}$$

   wobei $S(B')$ die, durch das Hinzufügen eventuell vergößerte, Oberfläche des Knotens B darstellt und k die Anzahl der Kinder von B,

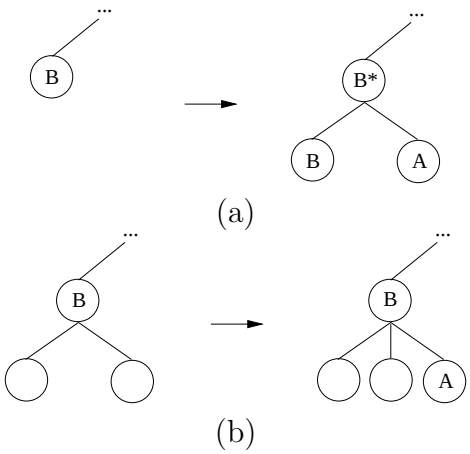


Abbildung 4.2: In (a) werden Knoten A und B zu einem neuen Knoten zusammengefasst. In (b) wird A als weiteres Kind zu B hinzugefügt

vor Einfügen von A darstellt. Daraus ergeben sich die inkrementellen Kosten

$$C_{\text{inc}} = C_{\text{new}} - C_{\text{old}} = (S(B') - S(B))k + S(B') \tag{4.12}$$

3. Das Objekt A wird nicht als Bruder oder neues Kind eines Knotens B eingefügt, sondern weiter herabgereicht an eines seiner Kinder, wo die Prozedur von neuem beginnt, allerdings mit zusätzlichen Kosten, genannt *Inheritance Cost* C_{ic}, durch die mögliche Vergrößerung der Oberfläche des Knotens B. Diese betragen abermals die Differenz zwischen den neuen Kosten des Knotens

$$C_{\text{new}} = S(B')k \tag{4.13}$$

und den alten Kosten

$$C_{\text{old}} = S(B)k \quad . \tag{4.14}$$

Die Inheritance Cost beträgt dann genau

$$C_{\text{ic}} = C_{\text{new}} - C_{\text{old}} = (S(B') - S(B))k \quad , \tag{4.15}$$

da sich die Anzahl der Kinder diesmal nicht verändert. Die Auswahl an welches Kind Objekt A weitergereicht wird, wird dabei anhand der

Kosten für Methode 1 und 2 plus C_{ic} getroffen. Das Kind, welches die niedrigsten Kosten verursacht wird ausgewählt.

Stößt das Objekt im Laufe der Einfügeprozedur an ein Blatt, wird es mit diesem zu einem neuen Knoten gemäß Punkt 1 zusammengefasst. Andernfalls wird die Methode ausgewählt, welche die geringsten Kosten verursacht und gegebenenfalls rekursiv bei einem der Kindknoten fortgefahren. Eine sehr detaillierte Beschreibung des kompletten Einfügeprozesses findet sich auch in [Hai89].

Vor allem im späteren Verlauf der Gesamtprozedur kann es geschehen, dass die Inheritance Cost bei mehreren Kindern einen gleichen Wert aufweist und so eigentlich nicht direkt entschieden werden kann, an welches der Kinder das Objekt weitergereicht werden soll. Goldsmith und Salmon schlagen vor, entweder alle möglichen Pfade weiter zu untersuchen, was jedoch zu einem deutlichen Mehraufwand führen würde, oder die Entscheidung abhängig zu machen von der Position der Mittelpunkte, oder einfach per Zufall eines der Kinder auszuwählen. Anders als Goldsmith und Salmon, schlägt Haines vor, dass die Entscheidung, welcher Kindknoten weiter betrachtet wird, nicht nur vom Anwachsen der Oberfläche dieses Knotens abhängig gemacht werden sollte, sondern ebenfalls anhand der Kosten durch Einfügen mittels Methode 1 und 2 zu bestimmen ist [Hai89]. Haines belegt diese Wahl allerdings lediglich anhand eines Beispieles, welches hier kurz aufgeführt werden soll. Gegeben sind zwei Kindknoten eines gemeinsamen Vaterknotens, von denen der erste eine 50 prozentige Wahrscheinlichkeit aufweist von einem Strahl getroffen zu werden, und der andere eine einprozentige Wahrscheinlichkeit. Das einzufügende Objekt würde komplett in den größeren Knoten passen, ohne diesen zu vergrößern, während der kleinere erweitert werden müsste, so dass er eine Wahrscheinlichkeit von 3% erhalten würde, von einem Strahl getroffen zu werden. Haines testet nun die Kosten für Methode 1 und 2 jeweils auf beide Kinder und kommt zu dem Schluss, dass es sinnvoller wäre, das Objekt in den kleineren Knoten einzufügen, da es dann nur in 3% der Fälle auf Schnitte mit einem Strahl getestet werden müsste, während es bei dem anderen 50% wären, Goldsmith und Salmon würden nach ihrem ursprünglichen Verfahren den größeren Knoten wählen, da hier kein Oberflächenzuwachs stattfindet. Allerdings läßt sich auch genauso einfach ein Gegenbeispiel finden, bei dem wiederum Haines Methode die schlechtere Wahl wäre. Denn nehmen wir einmal an, dass sich in dem größeren BV noch ein weiteres BV befindet, welches eine Trefferwahrscheinlichkeit von nur 1% aufweist, aber dennoch das einzufügende Objekt komplett umschließen würde, dann wäre die beste Einfügeposition dort, da das Objekt dann ebenfalls nur noch in 1% der Fälle getestet werden müsste.

Abschließend ist festzustellen, dass lokale Entscheidungen der Einfügeposition immer zu suboptimalen Ergebnissen führen können, welche sich jedoch in der Praxis als nicht sonderlich gravierend erwiesen haben, so dass in der Regel beide Methoden, die nach Goldsmith und Salmon, sowie die abgeänderte Variante nach Haines, zu sehr guten Ergebnissen führen.

Was diese Methode ganz besonders für dynamische Szenen interessant macht, ist die Möglichkeit des sequentiellen Einfügens. Goldsmith und Salmon selbst erwähnen bereits die Anwendbarkeit ihres Verfahrens für dynamische Szenen, indem die bewegten Objekte herausgelöscht und erneut wieder eingefügt werden. Wie wir später sehen werden, ist dies nicht immer ohne weiteres möglich. Es funktioniert nur für Szenen, die zu einem Großteil aus statischer Geometrie bestehen, so dass die dynamischen Objekte nur einen geringen Einfluss auf die Qualität der Hierarchie haben.

Nachteilig hingegen ist, dass, abhängig von der Einfügereihenfolge, sich unterschiedliche Hierarchien ergeben, die in ihrer Qualität recht stark variieren können. Die Standardabweichung kann dabei nach Haber *et al.* [HSS00] zwischen 10% und 30% betragen. Häufig wird aus diesem Grund die BVH mehrmals mit unterschiedlichen Objektreihenfolgen aufgebaut und die gemäß ihrem Kostenkriterium erfolgreichste gewählt.

4.1.2 Traversierung

Es existieren zwei grundlegende Arten eine BVH zu traversieren. Zum einen eine simple Tiefensuche, die jedoch keine Rücksicht auf die Anordnung von Strahl und Hüllvolumen nimmt, sowie eine etwas aufwändigere Variante nach Kay und Kajiya [KK86], welche eine Traversierung entlang des Strahles durchführt.

Traversierung durch Tiefensuche

Das Verfahren der Tiefensuche ist sehr simpel zu erläutern. Beginnend beim Wurzelknoten der BVH wird der aktuelle Strahl auf Schnittpunkte mit diesem getestet. Fällt dieser positiv aus wird rekursiv bei allen Kindern dieses Knotens fortgefahren. Wird ein Knoten nicht getroffen, so kann der gesamte darunter liegende Teilbaum missachtet werden, da keine der in ihm enthaltenen Geometrien geschnitten werden kann. Die Idee dabei ist, möglichst bald einen Blattknoten und damit einen möglichen Schnittpunkt zu erreichen, um alle hinter diesem liegenden Teile der Hierarchie ausser acht lassen zu können.

Um den recht hohen Rekursionsoverhead zu vermeiden wird zumeist noch eine Datenvorverarbeitung vorgenommen, die die Hierarchie in eine 1D-Array Struktur umkopiert und einen iterativen Traversierungsalgorithmus verwendet (vgl. [Smi98]). Der rekursive Algorithmus, verwaltet sehr viel unnötige Information. Bspw. ist es nicht notwendig eine Liste mit allen Kindern in jedem Knoten abzuspeichern. Es ist vollkommen ausreichend einen Zeiger auf das linke Kind, den nächsten Geschwisterknoten, sowie den Vaterknoten zu speichern (*Left Child - Right Sibling - Parent* - Struktur). In der Traversierung würde dann solange mit dem linken Kind fortgefahren, bis man an einem Blatt angekommen wäre oder der Strahl den Knoten verfehlt. In diesem Falle folgt man dem Zeiger auf den nächsten Geschwisterknoten und fährt dort fort. Ist dieser nicht vorhanden, wird dem Zeiger auf den Vaterknoten gefolgt, bis wieder ein Geschwisterknoten existiert, oder der Wurzelknoten erneut erreicht wird und die Traversierung abgebrochen werden kann.

Da sich in statischen Szenen die Struktur der Hierarchie nicht verändert, kann für den Fall, dass kein Geschwisterknoten existiert, der nächste zu untersuchende Knoten vorberechnet werden. Dies erspart den Zeiger auf den Bruder- und Vaterknoten und ersetzt ihn durch einen so genannten *Skip-Pointer*. Wird die Hierarchie in derselben Reihenfolge im Array abgelegt, wie sie bei der Tiefensuche traversiert würde, angenommen kein Knoten würde verfehlt, so kann sogar der Zeiger auf den linken Kindknoten gespart werden, denn das erste Kind eines Knotens mit Index i im Array besitzt dann stets den Index $i + 1$.

Diese Umstrukturierung hat verschiedene Vorteile. Erstens wird jegliche Rekursion vermieden, zweitens sinkt der Speicherbedarf für die Knoten, da sie nur noch ihr Hüllvolumen, bestehend aus sechs Float-Werten, sowie einem Zeiger auf ein möglicherweise enthaltenes Primitiv und den Skip-Pointer enthalten. Hinzu kommt, dass durch die Umsortierung im besten Falle die nächsten zu untersuchenden Knoten bereits im Speicher vorliegen.

Diese Art der Traversierung ist vor allem in Verbindung mit der Verwendung von SIMD-Ray Tracern empfehlenswert, siehe [Gei05]. In Abbildung 4.3 ist der Pseudocode für die Traversierung mittels Tiefensuche im 1D-Array gegeben.

Traversierung nach Kay/Kajiya

Eine andere Variante der Traversierung bietet das Verfahren nach Kay/Kayija [KK86]. Insbesondere in komplexen Szenen und bei Einzelstrahlverfolgung liefert dieses sehr gute Ergebnisse, da die Traversierung entlang des Strahles

```
bool trace(BVNodeArray bvNode, Ray ray)
{
  BVNode* stopNode = bvNode.end();
  BVNode* actNode = bvNode.begin();
  bool result = false;

  while(actNode != stopNode)
  {
    if(intersect(ray, actNode))
      if(actNode->hasPrimitive())
        if(actNode->getPrimitive()->intersect(ray))
          result = true;
      // Nächster Knoten
      actNode++;

    else
    // fahre mit Skip-Pointer fort
      actNode = actNode->skipPointer;
  }
  return result;
}
```

Abbildung 4.3: Pseudocode der Traversierung mittels Tiefensuche

```
bool trace(BVNode root, Ray ray)
{
  BVNode* actNode = root;
  bool result = false;
  float tempDistance = FLT_MAX;
  float bestDistance = FLT_MAX;

  if(intersect(ray, actNode, tempDistance))
    heap.insert(actNode, tempDistance);

  while((!heap.empty()) && (heap.topDistance < bestDistance))
  {
    actNode = heap.removeTop();

    if(actNode->hasPrimitive())
      if(actNode->getPrimitive()->intersect(ray, bestDistance))
        result = true;

    else
      for all children c_i of actNode
        if(intersect(ray, c_i, tempDistance))
          heap.insert(c_i, tempDistance);
  }
  return result;
}
```

Abbildung 4.4: Pseudocode der Traversierung nach Kay/Kajiya.

erfolgt. Beginnend beim Wurzelknoten wird getestet, ob der Strahl den aktuellen Knoten schneidet oder nicht. Ist dies nicht der Fall, kann er und der darunterliegende Teilbaum in der weiteren Betrachtung ausser Acht gelassen werden. Andernfalls werden die Kinder getestet und für jedes, sofern es ebenfalls getroffen wird, eine so genannte *estimated distance* (ED) berechnet. Diese entspricht dem Strahlparameter an der Stelle des Schnittpunktes von Strahl und BV des Knotens. Anhand dieser ED wird eine *priority queue* (PQ) angelegt, welche stets den Knoten zurückliefert, dessen ED am niedrigsten ist. Die Traversierung kann abgebrochen werden, sobald ein Schnittpunkt mit einem Primitiv gefunden wurde, der noch vor der nächsten ED in der PQ liegt. Mittels eines *Heaps* lässt sich die PQ sehr effizient umsetzen. Der Pseudocode findet sich in Abbildung 4.4

Die Traversierung entlang des Strahles bietet noch einen anderen großen Vorteil. Auf diese Weise werden während der Traversierung ebenfalls nur geringstmögliche Teile der Hierarchie überhaupt untersucht. Dies ist vor allem bei der Verwendung von Lazy Evaluation Techniken von Vorteil, wie sie in den Verfahren unter Abschnitt 4.3.4 und 4.3.5 zum Einsatz kommen. Um einen fairen Vergleich zu erlauben, und weil sie bei Einzelstrahlverfolgung meist bessere Ergebnisse liefert, wurde für alle unsere getesteten Verfahren diese Art der Traversierung gewählt.

4.2 Entwicklung eines Qualitätskriteriums für BVHs in dynamischen Umgebungen

Nachdem das Konzept, sowie Aufbau und Traversierung, von BVHs nun ausreichend vorgestellt wurden, soll in diesem Abschnitt erstmals auf dynamische Aspekte eingegangen werden. Es soll die Frage erörtert werden, welche Art von Qualitätskriterien (QK) die Möglichkeit bieten Teile einer Hierarchie zu erfassen, welche sich im Laufe der Zeit derartig verändert haben, dass ein Neuaufbau, oder zumindest eine Abänderung, welcher Art auch immer, zu empfehlen ist. Dies ist vor allem sinnvoll, wenn man nicht für jedes Bild die gesamte Beschleunigungsstruktur neu aufbauen möchte. Beginnend mit einer Analyse, welche Schwierigkeiten sich zeigen könnten, sollen danach mögliche Lösungsansätze präsentiert und diskutiert werden. Dabei wird zunächst einmal davon ausgegangen, dass Bewegungen in der Szene lediglich ein Anpassen der Hüllvolumen innerhalb der Hierarchie nach sich ziehen, um somit wieder zu einem konsisten Zustand zu führen, in dem alle Objekte von ihren Hüllvolumen und alle Kinder eines Knotens der Hierarchie von von ihrem Vater bestmöglich umschlossen sind. In Abschnitt 4.3.1 werden dazu noch verschiedene Möglichkeiten vorgestellt.

4.2.1 Voraussetzungen und Schwierigkeiten

Qualitätskriterien aufzustellen für den Umgang mit dynamischen Umgebungen mittels BVHs ist kein triviales Unterfangen, was sich bspw. bereits darin zeigt, dass die Struktur einer BVH, welche in einer Szene sehr gut geeignet ist, für eine andere minderwertig sein kann. Dies sei an einem Beispiel erläutert: Man stelle sich ein Schachbrett vor, welches dargestellt werden soll. Sowohl die schwarzen als auch die weißen Felder seien dabei zunächst räumlich weit voneinander entfernt, so dass der Wurzelknoten der BVH zwei

Kinder besitzt. Eines, welches sämtliche schwarzen Felder enthält und eines für die weißen. Im Laufe der Zeit sollen diese nun aufeinander zuwandern und so das endgültige Schachbrett bilden. Dies führt jedoch zu einer starken Überlappung direkt am Wurzelknoten, sowie zu nahezu einer Verdoppelung des gesamten Traversierungsaufwandes, weswegen starke Veränderungen in den Überlappungen der Kinder eines Knotens erkannt werden sollten.

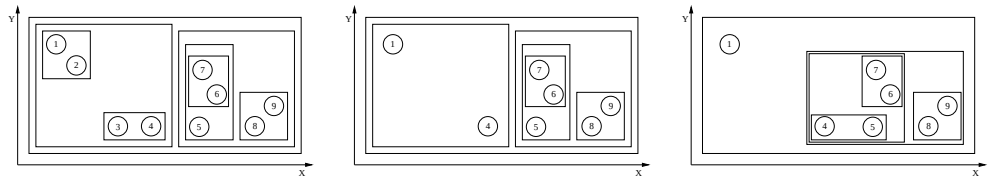
Ein QK muss des weiteren stets relativ zu einem Ausgangswert betrachtet werden und nicht absolut gewählt werden, um für beliebige Szenen einsatzfähig zu sein. Einzige Möglichkeit bietet hierbei zunächst die initiale Hierarchie, welche zu Beginn des ersten Frames erstellt wird und damit so optimal, wie mittels des Erstellungsverfahrens möglich, vorliegt. Allerdings zeigt sich hier bereits eine Schwierigkeit, da das Erstellungsverfahren selbst großen Einfluss auf das QK und seine Eignung hat.

Einer der wohl wichtigsten Punkte ist das Erkennen von Bewegung, der in einem Knoten enthaltenen Geometrie. Idealerweise würde ein QK dabei sogar feststellen, ob es sich um eine *gutartige* oder *bösartige* Bewegung handelt. Erstere würde die Qualität der Hierarchie verbessern, während letztere das Gegenteil zur Folge hat.

Periodische, also sich wiederholende, oder lokal begrenzte Bewegungen bilden dabei noch eine Untergruppe, die eventuell eine gesonderte Behandlung verdienen würde. Dies ist jedoch in dem in dieser Arbeit behandelten Falle nicht direkt möglich, da keinerlei Vorwissen über die Objektbewegungen vorliegt. Eine automatische Erkennung hingegen wäre in einer Szene mit tausenden von bewegten Objekten wohl zu aufwändig.

Des weiteren sollten große Objekte nicht zwangsläufig einen schlechten Wert hervorrufen, solange keine Möglichkeit zur Verbesserung besteht. Dies kann leicht passieren, wenn man Knoten in ein Verhältnis zu ihren Elternknoten setzt.

Ein ebenfalls großes Problem bereitet ein Effekt, der *Ausdünnung* genannt werden soll. In ihrer einfachsten Form entsteht sie durch das Herauslöschen von Objekten aus der BVH. Dies kann zu unausgeglichenen Hierarchien führen, im schlimmsten Falle zu einer linearen Liste. Ein Beispiel für eine mögliche Form der Ausdünnung ist in Abbildung 4.5 gegeben. In Abbildung 4.5(a) ist die Ausgangssituation gegeben. Objekte 1 bis 4, sowie 5 bis 9 liegen dabei jeweils in einem separaten Teilbaum der Hierarchie vor. Werden Objekte 2 und 3 gelöscht (Abb. 4.5(b)), so wäre es unter dem Gesichtspunkt der Minimierung der benötigten Strahltests jedoch deutlich sinnvoller Objekt 1 in einem separaten Knoten zu speichern und die Objekte 4 bis 9 in einem Teilbaum zusammenzufassen (Abb. 4.5(c)). Ausdünnung ist ein bei weitem



(a) (b) (c)

Abbildung 4.5: Beispiel für eine mögliche Ausdünnung einer BVH

nicht zu unterschätzendes Problem, da es viele der intuitiven Ansätze für QK zunichte macht, wie sich im weiteren Verlauf auch nochmal zeigen wird.

Im Gegenzug sollte ein Einfügen von Objekten nicht zwangsläufig eine Verschlechterung hervorrufen, da z.B. bei sequentiellen Einfügeverfahren, wie etwa dem vom Goldsmith und Salmon [GS87], automatisch die bestmögliche Einfügeposition gesucht wird.

Ein Qualitätskriterium muss für einen Knoten der BVH gegen einen festen Wert konvergieren, sollte die darunter liegende Geometrie statisch sein, bzw. falls sich keine Änderung im Vergleich zu den letzen Aktualisierungen ergeben haben sollte. Wie in Abschnitt 4.3.2 noch gezeigt wird, ist dies keinesfalls selbstverständlich und kann im schlimmsten Falle sogar dafür sorgen, dass Endlosschleifen in der Rekonstruktion der Hierarchie entstehen könnten.

Nicht zuletzt bleibt einer der wichtigsten Punkte, dass jede mögliche Heuristik für ein QK simpel, bzw. effizient zu berechnen sein muss. Da es, je nach Szene, für einige tausend Knoten der Hierarchie ermittelt werden muss. Dazu zählt auch, ob man es in beliebiger Reihenfolge für die verschiedenen Knoten berechnen kann. Dies alles könnte allerdings zur Folge haben, dass kein QK gefunden werden kann, welches allen Ansprüchen genügt.

Zusammenfassend lassen sich folgende Ansprüche an QK stellen, welche von ihnen erfüllt werden sollten:

- *Relativität*, das QK muss im Verhältnis zu einem Vergleichswert stehen.
- *Bewegung*, muss erkannt werden. Im besten Falle gäbe es noch eine Unterscheidung zwischen gutartiger, bösartiger, periodischer und lokal begrenzter Bewegung.
- *Große Objekte*, große Objekte verdienen eine Sonderbehandlung, da sie leicht die ermittelten Werte stark beeinflussen können.
- *Überlappung*, starke Überlappungen der Hüllvolumen führen zu verlangsamter Traversierung.

- *Ausdünnung*, das Herauslöschen von Objekten aus einem Teilbaum muss erkannt werden.
- *Statische Bereiche*, statische Bereiche sollten gegen einen festen Wert konvergieren.
- *Simplizität*, jedes QK muss simpel und effizient zu berechnen sein.

4.2.2 Mögliche Heuristiken

In der Literatur finden sich nur äußerst wenige Ansätze, die sich überhaupt mit dem Problem der Qualitätssicherung für Bounding Volume Hierarchien beschäftigen. Natürlich existieren verschiedenste Verfahren um qualitativ hochwertige Hierarchien zu erzeugen [KK86] [GS87] [MF99], aber diese beziehen sich stets nur auf den statischen Fall.

Ein erster Ansatz in Richtung Qualitätsberechnung für Knoten einer BVH, findet sich bei Haber *et al.* [HSS00]. Auch wenn sie den Test für einen globalen Optimisierungsschritt in einer statischen Hierarchie verwenden, ließen sich die eingeführten Kriterien vielleicht auch für dynamische Szenen nutzen und sie sollen deshalb der Vollständigkeit halber hier auch in den Punkten 1-3 angeführt werden. Wurde bei ihnen das QK nicht erfüllt, so wurde der Knoten gelöscht und die Kinder in die bereits bestehende Hierarchie neu einsortiert, oder der Knoten wird entlang seiner längsten Kante in zwei Gruppen aufgeteilt. Dabei dient der Mittelpunkt der Kindknoten als Entscheidungskriterium, welcher der beiden Gruppen sie angehören sollen. Zudem führten die Verfahren nicht immer zu einer Verbesserung, weswegen sie sich die Möglichkeit offen hielten, gegebenenfalls die Änderung wieder rückgängig zu machen. Dies wäre jedoch für dynamische Szenen aus Zeitgründen nicht praktikabel.

Im folgenden sollen die von Haber *et al.* entworfenen, sowie einige selbstentwickelte QK vorgestellt und analysiert werden.

1. *Achsenverhältnis*: Die erste Variante berechnet das Verhältnis zwischen den Ausdehnungen der Kindsknoten entlang der Achsen des Weltkoordinatensystems zur entsprechenden Ausdehnung seines Elternknotens. Der höchste Wert dient als *Badness* des Knotens. Ein benutzerdefinierter Threshold bestimmt, ob der Knoten gelöscht werden muss oder nicht.

   Auf den ersten Blick scheint es einzuleuchten, dass es voraussichtlich zu einer minderen Qualität führt, wenn die Ausdehnung entlang einer

Achse bei einem Kindknoten annähernd gleich ist zur Ausdehnung seines Elternknotens, doch ist diese Aussage nicht zutreffend. Denn es besteht die Möglichkeit, dass gar keine Besserung möglich ist. Ein Beispiel sind uniform verteilte Objekte. Die optimale Unterteilung würde voraussichtlich entlang der längsten Achse und zwar in der Mitte erfolgen. Das QK wird jedoch eine mindere Qualität für diesen Knoten ermitteln, da die Ausdehnung entlang zwei der drei Koordinatenachsen identisch mit dem Vaterknoten ist.

Relativität ist gegeben, Bewegung wird in der Regel erkannt, aber keine Klassifizierung vorgenommen. Große Objekte verursachen entsprechend Probleme. Das Problem der Ausdünnung wird missachtet, statische Bereiche bleiben konstant im Wert, aber die Berechnung ist immerhin verhältnismässig simpel.

2. *Volumenverhältnis*: Alternativ vergleichen Haber *et al.* das Volumen der Kindknoten mit ihrem Elternknoten. Wobei das Volumen der Kindknoten gebildet wird, indem die beiden am weitesten voneinander entfernten Kinder, bezüglich eines zuvor festgelegten Distanzmaßes, ausgewählt werden und das Volumen der sie umgebenden AABB berechnet wird.

   Das Ganze ist sehr ähnlich zum ersten Verfahren, mit nahezu den gleichen Stärken und Schwächen. Allerdings dürften Überlappungen besser erkannt werden, zumindest bei einer binären BVH.

3. *Erweiterung des Achsen- und Volumenverhältnisses durch Packdichte*: Der dritte Ansatz ist lediglich eine Variante der beiden zuerst genannten, bei der das Ergebnis noch durch die Anzahl der Objekte im Knoten geteilt wird. Dies soll dafür sorgen, dass Knoten mit wenigen Objekten schlechtere Werte erhalten als Knoten mit vielen Objekten, was quasi eine Art Packdichte darstellt. Dies bietet zudem eine Möglichkeit ausgedünnte Knoten zu erkennen und wäre damit voraussichtlich eine Verbesserung.

4. *Veränderung der Oberflächen*: Aus 4.1.1 wissen wir, dass sich die Oberfläche eines BV proportional zu seiner Trefferwahrscheinlichkeit verhält. Es liegt also nahe, diesen Wert direkt als QK zu verwenden, indem man den stets aktuellen Wert mit dem Initialwert bei der Erstellung des BV vergleicht.

   Die Oberfläche alleine gibt jedoch keinerlei Aufschluss darauf, ob eventuell eine Ausdünnung vorliegen könnte, da innere Objekte einfach gelöscht werden könnten, ohne Änderungen an der Oberfläche nach sich

zu ziehen. Zudem könnten sich innere Knoten völlig überlappen, weswegen es evtl. besser wäre, statt durch die Oberfläche des Wurzelknotens zu teilen, die Oberfläche des Vaterknotens zu wählen. Dies würde, zumindest bei lediglich zwei Kindern, eine zu starke Überlappung verhindern. Große Objekte würden weniger Schwierigkeiten verursachen und die Berechnung fällt sehr simpel aus.

5. *Durchschnittliche Schnittpunkttests*: Wie in Unterabschnitt 4.1.1 Goldsmith und Salmon erläutert, lässt sich eine ungefähre Anzahl an benötigten Schnittpunkttests eines Strahles mit einer Hierarchie berechnen. Genau diese Kosten können ebenso als Gütekriterium für einen Knoten verwendet werden, setzt man die stets aktuellen Kosten des Teilbaumes mit denen in Beziehung, die bei der Erstellung des Knotens berechnet wurden.

   Auch wenn zunächst intuitiv einleuchtend, bereitet dieser Ansatz mehr Probleme, als vielleicht zunächst anzunehmen. Wichtigster Aspekt hierbei ist zum einen, dass auch ein Absinken der Kosten in dynamischen Szenen eine Verschlechterung der Hierarchie nach sich ziehen kann. Bspw. eine Ausdünnung hat zwangsläufig ein Sinken der durchschnittlich benötigten Schnittpunkttests zur Folge, weil sich weniger Objekte und damit weniger Knoten mit geringerer Oberfläche in der Hierarchie befinden. Dennoch könnte eine deutlich bessere BVH existieren. Abfangen könnte man dies, indem ein zweiter Threshold eingefügt wird, der sowohl Veränderungen der Kosten nach oben, als auch nach unten erkennt. Bewegung wird prinzipiell erkannt, es sei denn, sie geht mit einer entsprechenden Skalierung einher. Aber die Berechnung kann nicht sequentiell für jedes bewegte Objekt durchgeführt werden, d.h. sämtliche Objekte müssen erst ihre nächste Position eingenommen haben, bevor das Kostenkriterium berechnet werden kann. Zudem ist die Berechnung sehr aufwändig, da jede Bewegung Änderungen entlang des kompletten Pfades bis zum Wurzelknoten nach sich zieht. Große Objekte können Schwierigkeiten verursachen, da sie in Erstellungsverfahren, die diese nicht gesondert behandeln, auf den berechneten Wert zuviel Einfluss nehmen können. Immerhin sind die Punkte statische Bereiche und Relativität gut erfüllt.

6. *Lokale Kosten*: Statt die kompletten Kosten eines Teilbaumes zu berechnen, könnte man erwägen lediglich die lokalen Kosten eines Teilbaumes T mittels

$$cost(T) = S(T) \cdot \#Kinder \tag{4.16}$$

zu ermitteln, wobei $S(T)$ die Oberfläche des Knotens T darstellt und $\#Kinder$ die Anzahl an Kindern von T. Nimmt man als Referenzwert die initialen Kosten, ist die Relativität erfüllt. Zwar wird Bewegung erkannt, große Objekte bereiten keine Probleme und da nur lokale Daten benötigt werden, ist die Berechnung auch sehr simpel gehalten. Aber die Ausdünnung könnte ein größeres Problem bereiten, da sie in höheren Knoten erst sehr spät oder gar nicht erkannt wird. Bei binären BVH, wie bei Median-Cut oder SAH, reduziert sich dieses Kriterium auf Punkt 4.

7. *Objektanzahl*: Gegen Ausdünnung könnte auch einfach die Anzahl der Objekte in jedem Knoten mitgeführt werden. Fällt sie unter einen gewissen relativen Threshold, könnte davon ausgegangen werden, dass der Knoten ausgedünnt ist.

   Leider besitzt die Anzahl an Objekten aber keinerlei Aussagekraft über die Qualität des Knotens, hinzu kommt, dass entfernte Objekte vielleicht einfach nur zusammen mit dem Knoten einen neuen Knoten gebildet haben, und dieser somit immer noch optimal aufgebaut ist. Bewegung wird zudem überhaupt nicht beachtet. Somit würde sich dieses Kriterium höchstens als Zusatz eignen.

8. *Relative Kosten pro Objekt*: Wie in Punkt 5 basieren die Kosten auf den Kosten des Teilbaumes, also den durchschnittlich zu erwartenden Schnittpunkttests. Um allerdings zu kompensieren, dass die Kosten bei Ausdünnung sinken, werden diese zusätzlich durch die Anzahl der in diesem Teilbaum enthaltenen Objekte dividiert. Das Ergebnis ließe sich ungefähr als die durchschnittlichen Kosten pro Objekt in diesem Teilbaum interpretieren. Die Folge ist, das ein Ansteigen der Kosten voraussichtlich eine Ausdehnung oder, wenn sich die Oberfläche des Knotens nicht verändert hat, eine Ausdünnung als Ursache hat. Auf jeden Fall lässt sich sagen, dass sich die Hierarchie verschlechtert hat. Relativität ist gegeben, wird der aktuell berechnete Wert stets mit seinem Initialwert verglichen. Große Objekte verursachen keine direkten Schwierigkeiten, statische Bereiche sind konstant. Problematisch bei diesem Ansatz ist jedoch, dass die Kosten bei jeder Bewegung bis zum Wurzelknoten aktualisiert werden müssen. Dies ist mit einem sehr hohen Aufwand verbunden.

9. *Geschätzte relative Kosten pro Objekt*: Um den Aufwand aus Punkt 8 zu verringern, wird versucht, anhand lokal gegebener Informationen die Kosten pro Objekt abzuschätzen. Genauer gesagt, wird die Oberfläche

durch die Anzahl der in diesem Knoten enthaltenen Objekte dividiert. Vorteilhaft daran wäre, dass lediglich die Kosten der betroffenen Knoten aktualisiert werden müssten, und nicht zwangsweise alle Knoten bis hinauf zur Wurzel. Die Berechnung ist sehr simpel und kann schnell durchgeführt werden. Ein Einfügen von Objekten verschlechtert die Kosten nur, wenn das Objekt die Oberfläche unverhältnismäßig vergrößert, was jedoch nicht geschehen sollte, wenn eine gewisse Lokalität gegeben ist.

Wichtig wäre jedoch bei dieser Methode (wie auch bei der vorherigen) einen doppelten Threshold anzusetzen. Denn auch ein Sinken der Kosten kann eine Verschlechterung der Hierarchie darstellen, z.B. durch Überlagerung der Kindknoten.

10. *Wechsel der Einfügeposition*: Ein Knoten gilt als verbesserungswürdig, sollte ein bewegtes Objekt theoretisch nicht mehr an derselben Einfügeposition landen, von der aus es seine Bewegung begonnen hat, gemäß der verwendeten Methode. Dieses QK ist natürlich sehr intuitiv, da es im Prinzip nichts anderes besagt, als dass Änderungen vorgenommen werden sollten, wenn die Einfügeposition nicht mehr optimal ist. Das Problem dabei ist, dass dieselben Berechnungen durchgeführt werden müssten, um herauszufinden, ob es eine bessere Einfügeposition gäbe, als wenn das Objekt tatsächlich herausgelöscht und neu eingefügt würde.

11. *Löschen eines Vaterknotens*: Immer wenn ein Objekt sich bewegt, könnte es aus der Hierarchie herausgelöscht und neu eingefügt werden, falls die Methode nach der die BVH aufgebaut ist, dieses erlaubt. Besitzt der Vaterknoten nur noch ein weiteres Kind und wird daraufhin entfernt, wird sein zweites Kind nicht einfach an die Stelle des Vaters gesetzt, sondern dieses wird ebenfalls neu in die Hierarchie eingefügt. Pro bewegtem Objekt entstehen so durchschnittlich $\log n$ weitere Kinder die neu eingefügt werden müssten, dass heißt ein durchschnittlicher Aufwand von $O(n \log^2 n)$ wäre gegeben, bei n Objekten. Allerdings läge dieser fast immer vor, wenn sich verhältnismäßig viele Objekte bewegen. Dieser Effekt tritt erst recht bei binären Bäumen auf. Der Rest der Bedingungen wäre prinzipiell erfüllt.

12. *Bewegung der Kindknoten*: Ein Knoten zählt als erneuerungswürdig, wenn sich seine Kinder zu sehr bewegt haben. Dies bedeutet allerdings, dass jeder Knoten die ursprüngliche Position seiner Kinder speichern muss. In jedem Schritt müsste er dann aus der ursprünglichen Positi-

on der Kinder und der aktuellen Position berechnen, ob eine Rekonstruktion notwendig ist. Dies müsste für jeden Knoten der Hierarchie geschehen, welcher sich verändert hat, was einen ebenfalls recht hohen Aufwand nach sich zieht. Ausdünnung könnte zudem nur schwerlich erkannt werden. Skalierungen sind schwierig umzusetzen, da abhängig vom Bezugspunkt unterschiedliche Ergebnisse ermittelt würden.

13. *Wahrscheinlichkeit, dass alle Kinder eines Knotens von einem Strahl getroffen werden*: Um dem logarithmischen Aufwand des Ray Tracing Verfahrens zu genügen, müssten theoretisch bei jedem Strahltest gegen ein Hüllvolumen während der Traversierung ungefähr die Hälfte aller Objekte von der weiteren Betrachtung ausgeschlossen werden. Dies geschieht natürlich nur, wenn der Strahl auch stets ungefähr die Hälfte aller Kinder eines Knotens T verfehlt, sollte er T schneiden. Sollte also die Wahrscheinlichkeit, dass er alle Kinder des Knotens schneidet, steigen, so könnte man davon ausgehen, dass eventuell bessere Aufteilungen möglich wären. Die dafür notwendigen Berechnung sind jedoch alles andere als trivial. Deshalb wird sich im Weiteren auf den Fall von lediglich zwei Kindern beschränkt. Dadurch wird dieses Kriterium zudem zuverlässiger, da es andernfalls geschehen könnte, dass sich einzelne Kindsknoten komplett überlappen, dies aber nicht zwingendermaßen erkannt würde. Damit ließe es sich allerdings nicht mehr für Verfahren, wie die von Goldsmith und Salmon, wohl aber für Median-Cut Algorithmen oder die Surface Area Heuristik verwenden.

    Seien A und B die beiden Kinder eines Knotens C, der von einem Strahl $\mathbf{R}(t)$ geschnitten wird. Verläuft $\mathbf{R}(t)$ in Richtung $\vec{\nu}$, so trifft er beide Kinder mit der Wahrscheinlichkeit P. P ist dabei das Verhältnis der Schnittmenge, der in Richtung $\vec{\nu}$ projizierten Flächen der beiden Kindknoten, zur projizierten Fläche von C.

$$P((\mathbf{R}(t) \cap A) \wedge (\mathbf{R}(t) \cap B) | (\mathbf{R}(t) \cap C) \wedge (\mathbf{R}(t) \| \vec{\nu})) = \frac{P_{\vec{\nu}}(A) \cap P_{\vec{\nu}}(B)}{P_{\vec{\nu}}(C)} \tag{4.17}$$

    $\mathbf{R}(t) \cap X$ bedeutet, dass $\mathbf{R}(t)$ Knoten X schneidet. $\mathbf{R}(t) \| \vec{\nu}$ gibt an, dass $\mathbf{R}(t)$ parallel zur Richtung $\vec{\nu}$ verläuft. $P_{\vec{\nu}}(X)$ ist die in Richtung $\vec{\nu}$ projizierte Fläche von X.

    Integriert über alle möglichen Richtungen Ω ergibt sich für den rechten Teil der Gleichung (4.17), analog zu Gleichung (4.4) bis (4.7),

$$\frac{\int_\Omega P_{\vec{\nu}}(A) \cap P_{\vec{\nu}}(B) d\vec{\nu}}{\int_\Omega P_{\vec{\nu}}(C) d\vec{\nu}} \tag{4.18}$$

Da für konvexe Objekte gilt, dass die durchschnittliche projizierte Fläche genau $\frac{1}{4}$ der Oberfläche entspricht (vgl. [Arv90]), kann Formel 4.18 umgeformt werden zu

$$\frac{\int_\Omega P_{\vec{\nu}}(A) \cap P_{\vec{\nu}}(B) d\vec{\nu}}{\frac{1}{4} S(C)} \tag{4.19}$$

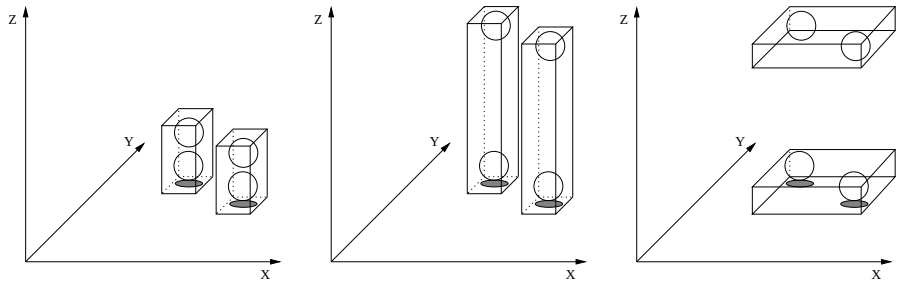
Leider lässt sich die Oberfläche des durch die Projektion der Schnittmengen definierten Körpers nicht so einfach bestimmen, weswegen eine Approximation vorgenommen wird, indem das Integral auf die drei Achsen des Weltkoordinatensystems diskretisiert wird.

$$\frac{1}{4S(C)} \frac{P_{\vec{\mathbf{x}}}(A) \cap P_{\vec{\mathbf{x}}}(B) + P_{\vec{\mathbf{y}}}(A) \cap P_{\vec{\mathbf{y}}}(B) + P_{\vec{\mathbf{z}}}(A) \cap P_{\vec{\mathbf{z}}}(B)}{3} \quad , \tag{4.20}$$

mit $\vec{\mathbf{x}} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $\vec{\mathbf{y}} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ und $\vec{\mathbf{z}} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$. Dies entspricht bis auf die Skalierungsfaktoren genau der relativen Oberfläche des Überlappungsbereiches zwischen den Kindern im Verhältnis zum Vaterknoten, sofern eine Überlappung vorliegt.

Problematisch hierbei ist, dass eine Ausdünnung eher für weniger Überlappung sorgt als für mehr, sowie dass große Objekte meist für mehr Überlappung sorgen. Zudem sorgt die Approximation des Integrales auf die drei Weltkoordinatenachsen dafür, dass die Wahrscheinlichkeit bei Null liegen kann, obwohl sich der Knoten immer weiter verschlechtert. Ein Beispiel sei dazu in Abbildung 4.6 gegeben. Die Projektionen entlang der drei Weltkoordinatenachsen der beiden in Abb. 4.6 (a) dargestellten Hüllvolumen überschneiden sich nicht. Die in Abb. 4.6(b) dargestellte Bewegung der beiden Objekte entlang der z-Achse ändert nichts an dieser Tatsache, dennoch wäre es, ab einer gewissen Entfernung, sinnvoller eine Unterteilung wie in Abb. 4.6(c) vorzunehmen.

Abhilfe könnte eine Verbindung mit Punkt 4 bringen, da so zumindest das Anwachsen der Hüllvolumen erkannt wird und eine geeignetere Einteilung vorgenommen werden kann.



(a) (b) (c)

Abbildung 4.6: Schwäche des Überlappungskriteriums

	1.	2.	3.	4.	5.	6.	7
Relativität	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
Bewegung	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\ominus\ominus$
Große Objekte	$\ominus$	$\ominus$	$\ominus$	$\oplus\ominus$	$\oplus$	$\oplus\ominus$	$\oplus$
Überlappung	$\ominus$	$\oplus\ominus$	$\oplus\ominus$	$\oplus\ominus$	$\oplus\ominus$	$\oplus\ominus$	$\ominus\ominus$
Ausdünnung	$\ominus$	$\ominus$	$\oplus$	$\ominus$	$\oplus$	$\ominus$	$\oplus$
statische Bereiche	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$
Simplizität	$\oplus$	$\oplus$	$\oplus\ominus$	$\oplus$	$\ominus$	$\oplus\ominus$	$\oplus\oplus$
Eignung für G&S	$\oplus\ominus$	$\oplus\ominus$	$\oplus\ominus$	$\oplus\ominus$	$\oplus$	$\oplus$	$\oplus$
Eignung für M-C	$\oplus\ominus$	$\oplus$	$\ominus$	$\oplus$	$\oplus\ominus$	$\ominus\ominus$	$\ominus\ominus$

	8.	9.	10.	11.	12.	13.
Relativität	$\oplus$	$\oplus$	$\oplus\oplus$	$\oplus$	$\ominus$	$\oplus$
Bewegung	$\oplus$	$\oplus$	$\oplus\oplus$	$\oplus$	$\oplus$	$\oplus\ominus$
Große Objekte	$\oplus\ominus$	$\oplus\ominus$	$\oplus\oplus$	$\oplus$	$\oplus$	$\ominus$
Überlappung	$\oplus\ominus$	$\oplus\ominus$	$\oplus$	$\oplus$	$\oplus\ominus$	$\oplus\oplus$
Ausdünnung	$\oplus$	$\oplus$	$\oplus\oplus$	$\ominus$	$\ominus$	$\ominus$
statische Bereiche	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$
Simplizität	$\ominus\ominus$	$\oplus\ominus$	$\ominus\ominus$	$\ominus\ominus$	$\ominus\ominus$	$\oplus\ominus$
Eignung für G&S	$\oplus$	$\oplus$	$\oplus\oplus$	$\oplus$	$\ominus$	$\ominus$
Eignung für M-C	$\ominus\ominus$	$\ominus\ominus$	$\oplus\ominus$	$\ominus$	$\oplus$	$\oplus$

$\oplus\oplus$ = sehr gut geeignet, $\oplus$ = relativ gut geeignet
$\oplus\ominus$ = mittelmässig geeignet, $\ominus$ = kaum geeignet
$\ominus\ominus$ = ungeeignet, $\oslash$ = keine direkte Aussage möglich

Tabelle 4.1: Eigenschaften der Qualitätskriterien. Die Nummerierung entspricht den auf Seite 46 bis Seite 52 vorgestellten QK.

Tabelle 4.1 zeigt noch einmal alle Punkte im Überblick. Die Eignung für die verschiedenen Verfahren ist dabei keine Angabe über die Güte der Heuristiken, sondern lediglich, ob sie sich überhaupt sinnvoll einsetzen ließen. Ob die Anforderung an statische Bereiche bei den einzelnen QK erfüllt ist, lässt sich leider nicht direkt beantworten, da dies immer abhängig von der Anwendung ist, wie in Abschnitt 4.3.2 gezeigt wird.

Leider konnte kein Kriterium gefunden werden, welches für sich alleine allen Anforderungen genügt. Dennoch können sie, wie in den nächsten Abschnitten gezeigt wird, durchaus ihren Zweck erfüllen. Für die richtige Auswahl ist es sehr wichtig darauf zu achten, ob ein QK alleine den Ausschlag für eine Strukturveränderung in der BVH geben soll, oder nur zusätzlich verwendet wird und welche sonstigen Gegebenheiten vorliegen.

4.3 Rekonstruktionsmethoden

Die bisher vorgestellten Verfahren zur Erstellung einer BVH waren lediglich auf statische Szenen ausgelegt. Die folgenden Überlegungen bauen hierauf auf, und machen einige dieser Verfahren auch für dynamische Szenen anwendbar. Neben der *Vorverarbeitungs-Phase*, in welcher die Beschleunigungsstrukturen zum ersten Mal aufgebaut werden, und der *Ray Tracing-Phase*, in welcher die Bildberechnung stattfindet, wird noch eine dritte eingeführt, die *Rekonstruktions-Phase*. In dieser werden die Objekte animiert und gegebenenfalls Änderungen an der Beschleunigungsstruktur vorgenommen.

4.3.1 Referenzmethoden

Um einen akzeptablen Vergleich zu haben, sowie eine Möglichkeit die Qualität unserer dynamischen Hierarchien zu ermitteln, wurden vier Referenzmethoden implementiert, welche ihren Schwerpunkt auf verschiedene Aspekte legen. So bedeutet der Appendix *Rebuild*, dass die Hierarchie nach jedem Frame komplett verworfen wird und ein Neuaufbau stattfindet, um eine optimale Ray Tracing-Phase zu ermöglichen. Während die Methoden mit dem Zusatz *Refit*, ihre Hierarchie bestmöglich für den ersten Frame aufbauen, danach jedoch die Struktur unverändert lassen und lediglich die Hüllvolumen anpassen, was wiederum die Rekonstruktions-Phase minimieren soll.

Rebuild

Es mag zunächst nach keiner guten Idee klingen, nach jedem berechneten Frame, die komplette Beschleunigungsstruktur zu verwerfen und diese neu aufzubauen. Bedingt durch optimierte Algorithmen, sowie verhältnismäßig schnelle Verfahren zur Erstellung der Hierarchie mit einer durchschnittlichen Komplexität von $O(n \log n)$ kann dies jedoch auch auf heute üblichen Arbeitsplatzrechnern durchaus bis zu einer relativ stattlichen Anzahl an Objekten in Echtzeit durchgeführt werden. In unseren Tests lag dies, je nach Szene, bei bis zu ca. 2000 Objekten, sowohl bei Verwendung einer Median-Cut Hierarchie nach Kay/Kajiya [KK86], als auch dem Verfahren nach Goldsmith und Salmon [GS87] (siehe auch die Ergebnisse aus Abschnitt 6). Dies ist nicht besonders viel, zählt man die statischen Objekte einer Szene hinzu. Es bliebe jedoch abzuwägen, ob eine Trennung von statischen und dynamischen Szeneinhalten Vorteile bringen würde. Positiv wäre natürlich die Möglichkeit auch komplexere Szenen stets in nahezu optimaler Form vorliegen zu haben, nachteilig wäre allerdings, dass ein Großteil des Nutzens, den Ray Tracing durch seine logarithmische Komplexität mit sich bringt, zunichte gemacht würde. Denn eine Trennung zwischen statischer und dynamischer Szenengeometrie verlangt nicht nur ein Vorwissen über die Geometrie, sondern kann den Ray Tracing Aufwand nahezu verdoppeln, da für jeden Strahl sowohl in der statischen, als auch in der dynamischen Szenengeometrie nach dem vordersten Schnittpunkt gesucht werden muss. Da in dieser Arbeit jedoch von dem Fall ausgegangen wird, dass sich theoretisch jedes Objekt der Szene bewegen kann, kann eine solche Trennung nicht vorgenommen werden.

Dennoch, und vor allen Dingen auch um einen Vergleich zu unseren dynamischen Methoden zu haben, wurden zwei Verfahren implementiert, welche nach jedem Frame ihre Beschleunigungsstruktur komplett verwerfen und neu aufbauen. In der ersten, genannt *Median-Cut Rebuild* wird die Hierarchie nach der in Unterabschnitt 4.1.1 Median-Cut vorgestellten Methode erstellt, wobei immer entlang der längsten Achse geteilt wird. Die Methode *Goldsmith and Salmon Rebuild* basiert auf der in Unterabschnitt 4.1.1 Goldsmith und Salmon erläuterten Variante. Ziel ist es, möglichst optimale Renderingzeiten zu erreichen. Methoden, die Objekte sequentiell in die Hierarchie einfügen, wie es bei Goldsmith und Salmon der Fall ist, und damit in ihrer Qualität abhängig von der Objektreihenfolge sind, haben dabei gegebenenfalls einen kleinen Nachteil, da aus Zeitgründen die Hierarchie zwischen den Frames nicht mehrmals erstellt werden kann, um so eine optimale Hierarchie zu finden. Abbildung 4.7 gibt den Rekonstruktionsvorgang als Pseudocode wieder.

```
void update()
{
  // Aktualisierung der Objektposition
  updateObjects();

  // Aktualisierung der Hierarchie
  clearHierarchy(); // Löschen der alten Hierarchie

  buildHierarchy(); // Neuaufbau der Hierarchie
}
```

Abbildung 4.7: Pseudocode für ein komplettes Rebuild

Refit

Anders als beim Rebuild wird in diesen Varianten versucht, die Rekonstruktions-Phase ohne Rücksicht auf eine mögliche Verlangsamung der eigentlichen Ray Tracing-Phase möglichst kurz zu halten. Dafür wird die Beschleunigungsstruktur vor dem ersten Frame ebenfalls gemäß der beiden in Abschnitt 4.1.1 vorgestellten Methoden nach Goldsmith und Salmon oder Kay und Kajiya erstellt. In den darauffolgenden Frames werden jedoch lediglich die Hüllvolumen angepasst, wogegen die Struktur der Hierarchie bestehen bleibt. Dies hat voraussichtlich zur Folge, dass sich die Ray Tracing-Phase über die Zeit erheblich verlängern wird, da keinerlei Rücksicht mehr auf die Position der Objekte genommen wird.

Das eigentliche Refitting, also das Erstellen einer wiederum konsistenten Hierarchie, nach Bewegung der Objekte, kann in drei verschieden Varianten geschehen, welche allesamt verschiedene Vor- und Nachteile mit sich bringen.

1. *Sequentielles Refitting*: Für sehr wenige bewegte Objekte eignet sich ein sequentielles Refitting. Dabei wird jeweils ein Objekt bewegt und die Hierarchie von seinem Blattknoten aus bis zum Wurzelknoten hin aktualisiert, indem die entsprechenden Hüllvolumen angepasst werden. Um die Anpassung der Hüllvolumen effizient durchzuführen, müssen allerdings zusätzliche Informationen für jedes Objekt und jeden Knoten der BVH abgespeichert werden. So benötigt jedes Objekt gegebenenfalls einen *Hierarchie-Zeiger*, der auf seine Position innerhalb der Beschleunigungsstruktur verweist und somit einen direkten Zugriff erlaubt. Jeder Knoten der BVH benötigt einen zusätzlichen Zeiger auf seinen Vater. Abbildung 4.8 gibt den Pseudocode wieder. Dieser ließe

sich evtl. noch verbessern, indem man bei der Aktualisierung jeweils testet, ob noch eine Änderung an den Hüllvolumen der Knoten stattfindet, oder ob bereits abgebrochen werden kann. Denn meist wird ein aktuelles Objekt bereits nach wenigen Schritten Richtung Wurzelknoten wieder komplett umschlossen, was ein weiteres Anpassen unnötig macht.

```
void seqUpdate(){
for every object o
    seqRefit(o);
}

void seqRefit(Object o){
  // Aktualisierung der Objektposition
  updateObject(o);

  // Objektposition in der BVH
  node = o.hierarchyPointer;

  // Anpassung des Hüllvolumens an das enthaltene Primitiv
  node.setBBox(node.getPrimitive().inquireBounds());

  while(node != NULL)
  {
    // Anpassung des Hüllvolumens an seine Kinder
    node.encloseChildren();
    node = node.getParent();
  }
}
```

Abbildung 4.8: Pseudocode für das sequentielle Refitting

Sobald sich jedoch mehrere Objekte bewegen, die sich Teile des Pfades zum Wurzelknoten teilen, wird ein deutlicher Overhead erzeugt. Ausgehend von dem Fakt, dass jeder innere Knoten einer BVH mindestens zwei Kinder besitzt, ist die maximal mögliche Anzahl an inneren Knoten genau $n - 1$ bei n Objekten, sowie die Länge eines Pfades von einem Blattknoten zum Wurzelknoten durchschnittlich $\log n$. Bewegt sich jedes Objekt, würde diese Methode $n \log n$ Anpassungen der Hüllvolumen benötigen, was von der Komplexität ähnlich einem kompletten Neuaufbau wäre.

2. *Complete Refit*: Sobald $(2n-1) < m \log n$ gilt, wobei n die Anzahl der Objekte in der Szene, sowie m die Anzahl der in diesem Frame bewegten Objekte ist, würde sich in der Regel eher lohnen die gesamte Hierarchie einem Update zu unterziehen, so dass jeder Knoten lediglich einmal an die veränderten Gegebenheiten angepasst werden muss. Dies ist in zwei Varianten möglich. In der rekursiven Form startet man beim Wurzelknoten und ruft die Refit-Funktion für jeden Kindknoten erneut auf. Danach wird das Hüllvolumen an diese, nun aktualisierten Knoten, angepasst. Dabei lässt sich nicht immer vermeiden, dass ein relativ großer Rekursionsoverhead erzeugt wird. Eine Ausnahme bildet das von van den Bergen verwendete Refitting [Ber97]. Ist es möglich die Hierarchie vorab in einem eindimensionalen Array abzulegen, so dass für jeden Knoten gilt, dass sein eigener Index größer als der des Vaterknotens ist, ließe sich ein komplettes Refit mittels einmaliger Iterierung über das Array von hinten nach vorne durchführen. Van den Bergen erzielte damit Geschwindigkeitssteigerungen um den Faktor 10 gegenüber einem kompletten Neuaufbau. In Abbildung 4.9 ist der Pseudocode für beide Varianten angegeben.

3. *Priority Queue*: Einen Mittelweg zwischen den beiden Extremen bildet das Refit mittels *Priority Queue* (PQ) nach Brown [BSB+01]. Dabei wird ein Bottom-to-Top Refit vorgenommen, indem alle Knoten, welche ein bewegtes Objekt haben, in diese PQ eingefügt werden. Ähnlich einem Heap werden sie dort nach ihrer Tiefe sortiert, sowie Duplikate entfernt. In jedem Schritt wird das vorderste Element entfernt, sein Hüllvolumen angepasst und sein Vaterknoten der PQ wieder hinzugefügt. Danach wiederholt sich der Vorgang, bis diese leer ist. Abbildung 4.10 gibt dies als Pseudocode wieder. Diese Methode ist vor allen Dingen empfehlenswert, wenn lediglich lokale Änderungen in der Hierarchie vonnöten sind, da im besten Falle bei n Objekten und k bewegten Objekten der Aufwand bei $O(k + \log n)$ liegt. Im schlimmsten Falle liegt er bei $O(n)$. Bei beiden ist die Verwaltung der PQ noch nicht mit einberechnet.

4.3.2 Dynamic Goldsmith and Salmon

Das im folgenden vorgestellte Verfahren ist nun das erste in dieser Arbeit, welches direkt für dynamische Szenen entworfen wurde. Es basiert dabei im Wesentlichen auf der von Goldsmith und Salmon eingeführten Heuristik [GS87] und versucht zum einen lokale Änderungen auch lokal zu behandeln, sowie

```
// Rekursive Variante
void recursiveUpdate(){
  updateObjects();
  recursiveRefit(root);
}

void recursiveRefit(BVNode node){
  for every child b of node
    recursiveRefit(b);

  if(node.hasPrimitive())
    // Anpassung des Hüllvolumens an das enthaltene Primitiv
    node.setBBox(node.getPrimitive().inquireBounds());
  else
   // Anpassung des Hüllvolumens an seine Kinder
    node.encloseChildren();
}

// Iterative Variante
void iterativeUpdate(){
  updateObjects();
  iterativeRefit(bvNodes);
}

void iterativeRefit(BVNodeArray bvNodes){
  for every element n of bvNodes from back to front
    if(n.hasPrimitive())
      // Anpassung des Hüllvolumens an das enthaltene Primitiv
      n.setBBox(node.getPrimitive().inquireBounds());
    else
      // Anpassung des Hüllvolumens an seine Kinder
      n.encloseChildren();
}
```

Abbildung 4.9: Pseudocode für einen Complete Refit, in der rekursiven und iterativen Variante

```
priorityqueueUpdate(){
  for every object o
    updateObject(o);
    PQ.insert(o.hierarchyPointer);

  priorityQueueRefit(PQ);
}
priorityQueueRefit(PriorityQueue PQ){
  while(!PQ.empty)
    node = PQ.extractTop(); // vorderstes Element der PQ

    if(node.hasPrimitive())
      // Anpassung des Hüllvolumens an das enthaltene Primitiv
      node.setBBox(node.getPrimitive().inquireBounds());
    else
      // Anpassung des Hüllvolumens an seine Kinder
      node.encloseChildren();

    PQ.insert(node.getParent());
}
```

Abbildung 4.10: Pseudocode für ein Refitting mittels Priority Queue PQ

Ausdünnung durch Verwendung eines der in Abschnitt 4.2 vorgestellten Qualitätskriterien zu verhindern.

Initialaufbau

Initial wird die Hierarchie nach dem Verfahren von Goldsmith und Salmon [GS87] bestmöglich aufgebaut. Die benötigte Zeit für die erstmalige Erstellung der BVH gilt dabei als vernachlässigbar. Von daher kann die BVH mehrmals erstellt werden, mit jeweils unterschiedlicher Objektreihenfolge und die voraussichtlich beste wird gewählt. Als Vergleichswert dient dabei die theoretisch durchschnittlich benötigte Schnitttestanzahl pro Strahl, wie in [GS87] vorgestellt.

Rekonstruktions-Phase

Auch Goldsmith und Salmon schlugen bereits vor, dynamische Objekte so zu behandeln, dass sie nach jedem Frame aus der Hierarchie gelöscht werden, die Hüllvolumen neu angepasst und die Objekte komplett neu eingefügt werden. Der Aufwand von $O(m \log n)$ bei n Objekten, und m bewegten Objekten ist bei wenig Bewegungen noch relativ moderat. Dieser Ansatz birgt jedoch einige Probleme. Denn erstens müsste die Hierarchie komplett neu aufgebaut werden, sollten sich alle Objekte der Szene bewegen und zweitens benötigt dieses Verfahren auch wegen der Gefahr der Ausdünnung einen Großteil an statischer Szenengeometrie, so dass ein Einfügen und Herauslöschen von dynamischen Objekten kaum Einfluss auf die Qualität der Hierarchie hätte.

Aus diesem Grund wird in dieser Methode ein anderer Weg vorgestellt, um diese Schwächen abzumildern. Ähnlich wie beim Erstellen der Beschleunigungsstruktur werden auch in der Rekonstruktions-Phase die Objekte sequentiell bearbeitet. Das Vorgehen ist dabei wie folgt. Zunächst werden sämtliche Objekte aktualisiert. Jedes von ihnen besitzt einen Hierarchie-Zeiger, der ein effizientes Auffinden dieser Objekte in der BVH ermöglicht. Um zu erkennen, ob sich das Objekt überhaupt bewegt hat, wird das alte Hüllvolumen zwischengespeichert, anhand der Objektdaten ein neues berechnet und diese verglichen. Liegt keine Bewegung vor, so wird ohne weitere Änderungen an der BVH direkt zum nächsten Objekt übergegangen. Hat es sich jedoch bewegt, so wird das neue Hüllvolumen berechnet und der Knoten zunächst aus der Hierarchie entfernt.

Das Objekt wieder komplett am Wurzelknoten einzufügen hätte jedoch einige Nachteile. Denn erstens ergeben sich Schwierigkeiten beim Einfügen in der

Nähe des Wurzelknotens, bedingt dadurch, dass bereits eine komplette Hierarchie vorliegt und die voraussichtlich relativ große Überlappung der BVs in diesem Bereich dafür sorgt, dass nicht ohne weiteres der perfekte Einfügeknoten gefunden werden kann. Und zweitens ist dies auch ein unnötiger Mehraufwand, da das Objekt entfernt, die Hüllvolumen bis zum Wurzelknoten angepasst werden müssten, nur um das Objekt dann erneut einzufügen, obwohl es in den meisten Fällen dieselbe Position in der Hierarchie annehmen wird. Dies dürfte vor allem in realistischen Szenen der Fall sein, in denen Bewegungen von einem zum nächsten Bild meist nur gering ausfallen. Aus diesem Grund wird das Objekt in dieser Methode nicht komplett entfernt, sondern der Vaterknoten wird zunächst an seine Kinder, ohne das Objekt, angepasst. Umschließt dieser Knoten es dennoch bereits wieder, wird er als neuer Einfügeknoten ausgewählt. Andernfalls wird in gleicher Weise mit dessen Vaterknoten fortgefahren, bis ein neuer Einfügeknoten gefunden wurde.

Die Wahl dieses Einfügeknotens ist nicht durch reine Willkür bestimmt, sondern würde mit größter Wahrscheinlichkeit auch bei einer kompletten Einfügeprozedur nach Goldsmith und Salmon vom Wurzelknoten aus durchlaufen. Dies liegt daran, dass die Inheritance Cost, also die Kosten, welche beim Weiterreichen eines Objektes in der Hierarchie durch Vergrößerung der Hüllvolumen entstehen, bei diesem Knoten genau null sind. Startet die Einfügeprozedur also von diesem Knoten aus, hat dies verschiedene Vorteile. Es werden nur lokale Teile der Hierarchie betrachtet, was zur Folge hat, dass nicht die komplette Hierarchie bis zum Wurzelknoten angepasst werden muss, nachdem das Objekt entfernt wurde, sondern nur der absolut notwendige Teil, zudem verkürzt sich entsprechend die Einfügeprozedur. Alleine durch dieses Hochreichen kann sich die Rekonstruktionsphase bedeutend beschleunigen, im schlimmsten Falle entspricht sie jedoch einem einfachen Rauslöschen und erneuten Einfügen vom Wurzelknoten aus, und somit einem Aufwand von $O(m \log n)$, bei m bewegten Objekten und n Objekten in der Szene.

Dabei kann allerdings das Problem der Ausdünnung bestehen bleiben. Angenommen der Wurzelknoten hätte genau zwei Kinder. Bewegt sich ein Objekt im linken Teilbaum, genannt T_l, so kann es geschehen, dass seine neue Einfügeposition im rechten Teilbaum, T_r, liegen würde. Selbstverständlich kann dies einige Male geschehen, was letzten Endes dazu führt, dass immer mehr und mehr Objekte aus T_l entfernt werden, wodurch seine relativen Kosten letzten Endes immer größer werden, da nur noch wenige Objekte aus der weiteren Betrachtung entfernt werden können, sollte der Strahl ihn verfehlen. Im schlimmsten Falle könnte die Hierarchie zu einer Liste verkommen.

Genauso kann es allerdings auch den anderen Fall geben, dass Objekte aus einem Teilbaum entfernt werden und es gerade deswegen empfehlenswert wäre

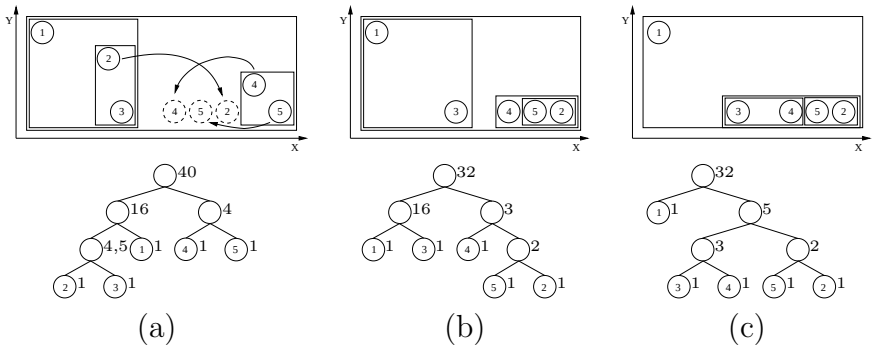


Abbildung 4.11: Entfernung eines Bad Nodes.

noch mehr von ihnen zu entfernen, um die Hierarchie zu verbessern. Ein Beispiel zum besseren Verständnis ist in Abbildung 4.11 gegeben. Objekt 2, 4 und 5 bewegen sich, wie in Abbildung 4.11(a) angegeben. Die gestrichelten Kreise stellen dabei die Zielposition dar. Ab einem gewissen Zeitpunkt wird das Einfügekriterium nach Goldsmith und Salmon verlangen, dass Objekt 2 in den rechten Teilbaum wechselt. Obwohl sich Objekt 3 nicht bewegt hat, wäre es trotzdem für die Hierarchie von Nutzen, wenn dieses ebenfalls den Teilbaum wechseln würde. Dies lässt sich über die durchschnittlich erwarteten Schnitttests begründen. Die Nummern an den einzelnen Knoten der BVHs stellen dabei die Oberfläche des jeweiligen Hüllvolumens dar. Die erwartete Anzahl an Schnitttests für die BVH in Abb. 4.11(b) ist $4,3125$, während es für die BVH in Abb. 4.11(b) lediglich $3,625$ sind. Dies ergibt sich folgendermaßen: In Abb. 4.11(b) ist ein Schnitttest für den Schnitt mit dem Wurzelknoten notwendig, für welchen angenommen wird, dass es ein Treffer ist. Dadurch werden zwei weitere Schnitttests in Ebene zwei benötigt. Der linke der beiden Knoten weist eine Trefferwahrscheinlichkeit von $0,5(16/32)$ auf und zwei Kinder, was Kosten von einem weiteren Schnittpunkt verursacht. Die Trefferwahrscheinlichkeit für den rechten Knoten beträgt $0,9375(3/32)$. Bei zwei Kindern führt dies zu zusätzlichen Kosten von $0,1875$. In Ebene drei befindet sich nur noch ein innerer Knoten mit einer Oberfläche von 2, was bei zwei Kindern zu $2/32 \cdot 2 = 1/8$ Schnitttests führt. Insgesamt also $1+2+0,5 \cdot 2+\frac{3}{32} \cdot 2+\frac{1}{16} \cdot 2 = 4,3125$ voraussichtliche Schnitttests. Verfährt man analog mit der BVH aus Abb. 4.11(c) gelangt man auf den deutlich geringeren Wert von $1+2+\frac{5}{32} \cdot 2+\frac{3}{32} \cdot 2+\frac{1}{16} \cdot 2 = 3,625$.

Es böte sich also an, eines der in Abschnitt 4.2 vorgestellten Qualitätskriterien zu verwenden, um solch ausgedünnte Knoten zu erkennen, aus der Hierarchie zu entfernen und seine Kinder neu einzufügen, in der Hoffnung,

dadurch eine bessere Hierarchie zu erhalten. In den meisten realistischen Szenen wird dieses Problem nur geringen Einfluss haben. Dennoch sollte es nicht missachtet werden. In dieser Arbeit wurde sich für die geschätzten relativen Kosten pro Objekt entschieden (Punkt 9 Abschnitt 4.2.2). Dieses Kriterium behebt die meisten, der durch Ausdünnung verursachten Probleme, wie etwa das aus Abbildung 4.11, und lässt sich leicht berechnen. Dabei wird ein doppelter Threshold verwendet, im weiteren auch *Qualitätsgrenzen* genannt. D.h. , sinkt der Qualitätswert unter den ersten Threshold T_1 oder steigt er über den zweiten T_2 wird der Knoten gelöscht. Für die Wahl der Thresholds wird nach dem Initialaufbau der Hierarchie einmalig für jeden Knoten sein momentaner Qualitätswert berechnet. T_1 und T_2 werden dann relativ dazu gewählt. In unserer Anwendung gilt als Standardwert $T_1 = 0.5$ und $T_2 = 2$, welche aus empirischen Testdaten gewonnen wurden. D.h. sinken die relativen Kosten um mehr als 50% oder steigen sie um mehr als 100%, wird der Knoten gelöscht und die Kinder neu eingefügt. Die Schwäche dieses Kriteriums, dass eine Überlappung der Kindknoten nicht immer festgestellt werden kann, wird bereits dadurch behoben, dass jedes Objekt, welches sich bewegt, zunächst aus der Hierarchie entfernt und danach wieder eingefügt wird, wie bereits beschrieben. Durch ersteres kann die Überlappung nur sinken, für letzteres gilt, dass keine bessere Einfügeposition gefunden werden kann, da gemäß der Standardeinfügeprozedur von Goldsmith und Salmon verfahren wird.

Jedes Qualitätskriterium birgt jedoch bei Verfahren, die versuchen Teile der Hierarchie zu erhalten, einige versteckte Gefahren. Dies hat folgenden Grund: Wird ein Knoten in der Hierarchie gelöscht, weil er nur noch ein Kind hat, wird dieses Kind an die Stelle seines Vaters gesetzt. Aber wessen Qualitätsgrenzen erhält es? Wenn man ihm die seines Vaters gibt, kann es zu einer Kettenreaktion kommen, die im schlimmsten Falle dafür sorgt, dass jedes zweite Objekt in jedem Frame aktualisiert werden muss. Angenommen ein Knoten A besitzt zwei Kinder B und C. Nun wird B, weil es sich bewegt hat, aus dem Knoten entfernt (vgl. Abb. 4.12(a)). Weil A jetzt nur noch ein Kind aufweist, wird der A gelöscht. Dies muss jedoch geschehen, um die Hierarchie anzupassen, so dass B wieder eingefügt werden kann. Dies müsste dazu führen, dass die Qualitätsgrenzen von A daraufhin auf C übergehen. Je nachdem, wie der Knoten C aufgebaut ist, kann es sein, dass das Qualitätskriterium besagt, dass dieser Knoten ebenfalls neu eingefügt werden müsste (vgl. Abb. 4.12(b)). Nun wird aber zunächst B wieder eingefügt und landet an derselben Stelle wie vorher, d.h. er wird mit C zusammengefasst (vgl. Abb. 4.12(c)). Wird C nun entfernt in einem Folgeschritt (Abb. 4.12(d)), so kann es geschehen, dass die Qualitätsgrenzen von B wiederum unterschrit-

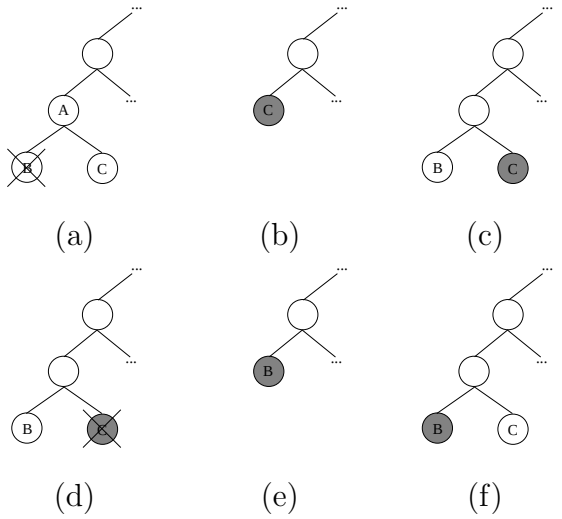


Abbildung 4.12: Ein Zyklus der möglichen Kettenreaktion bei Verwendung eines Qualitätskriteriums. Die grau markierten Knoten stellen die so genannten Bad Nodes dar.

ten werden (Abb. 4.12(e)) und er im nächsten Frame neu eingefügt werden muss (Abb. 4.12(f)). Nehmen wir an, B und C würden sich nicht mehr bewegen, so würde dennoch eine Kettenreaktion weiterlaufen, die ein Neueinfügen von B oder C in jedem Frame forciert.

Eine alternative Lösung dazu bestünde darin, dass beim Löschen eines Knotens nicht seine Qualitätsgrenzen an sein Kind weitergegeben werden. Die oben dargestellte Kettenreaktion würde verhindert. Nun kann es allerdings geschehen, dass ein Teilbaum bis auf ein Objekt ausdünnt. Da für dieses Objekt, solange es sich nicht bewegt, sein Qualitätskriterium nicht unter- oder überschritten werden kann, wird es solange dort festhängen, bis der Vaterknoten aktualisiert wird, unabhängig davon ob es schon längst eine bessere Einfügeposition geben könnte. D.h. mit der Zeit könnte die BVH unbemerkt in der Qualität sinken. Eine mögliche Lösung fände sich jedoch, wenn vorab die Szene in dynamische und statische Objekte unterteilt werden könnte. In diesem Falle könnte die Szene bestmöglich aus den statischen Objekten aufgebaut und erst danach die dynamischen Objekte hinzugefügt werden. So würde der Teilbaum schlimmstenfalls auf seine statische Anordnung zurückfallen, welche aber ja für diesen Bereich optimal erstellt wurde. Dies ist allerdings für den hier vorgestellten Anwendungsfall nicht möglich, da vorab keine derartige Einteilung vorgenommen werden kann.

Das eben vorgestellte Problem ist allerdings eines der allgemeinen Verwendung von Qualitätskriterien und nicht eines der hier vorgestellten Methode

an sich. Auch bei zunächst kompletter Entfernung aller dynamischen Objekte aus der Hierarchie und darauffolgendem Neueinfügen ergeben sich dieselben Schwierigkeiten. Die Kettenreaktion ließe sich jedoch verhindern, indem das Qualitätskriterium stets nur für die Knoten überprüft wird, die sich seit dem letzten Bild verändert haben. Dies wird auch von der hier vorgestellten Methode genutzt, da es zudem die Anzahl an Knoten der BVH, für die ein QK berechnet werden muss, drastisch verringern kann.

Interessant ist der positive Effekt auf die Renderingzeiten. So ließen sich durchaus Verbesserungen in der Renderingzeit um bis zu 34% erzielen, bei gerademal durchschnittlich 16% längerer Rekonstruktionszeit. In Abbildung 4.13 ist ein Vergleich der benötigten Ray Tracing Zeit sowie der Rekonstruktionszeit dargestellt für eine Beispielszene. Diese Werte sind allerdings, wie so oft natürlich, szenenabhängig. Theoretisch wäre sogar eine Verschlechterung möglich [HSS00]. Dies ist jedoch deutlich seltener der Fall, kann aber leider nicht ohne größeren Aufwand verhindert werden.

Insgesamt ergibt sich dadurch für die hier vorgestellte Methode folgendes Vorgehen: Für jedes Objekt, teste ob eine Bewegung vorliegt. Falls nein, fahre mit dem nächsten Objekt fort. Falls ja, entferne dieses aus der BVH, suche durch Hochreichen in der Hierarchie den nächsten, es umschließenden Knoten und füge es dort gemäß dem Verfahren nach Goldsmith und Salmon ein. Fahre dann mit dem nächsten Objekt fort. Markiere dabei stets die Knoten, welche sich verändert haben und setze die Markierungen bis zum Wurzelknoten fort. Wurden alle Objekte aktualisiert, suche entlang dieser Markierungen nach so genannten *Bad Nodes*, also Knoten, die ihr Qualitätskriterium nicht mehr erfüllen und speichere sie zwischen. Lösche der Reihe nach jeden *Bad Node* und füge seine Kinder in der gleichen Art und Weise neu ein, wie auch schon bei einzelnen Objekten verfahren wurde.

Dabei hat die Reihenfolge in welcher die Objekte aktualisiert werden, auch weiterhin Einfluss auf die Hierarchie. Dieser ist jedoch nicht so stark anzusetzen wie bei einem kompletten Neuaufbau. Abbildung 4.14 gibt den gesamten Rekonstruktions-Vorgang noch einmal als Pseudocode wieder.

Traversierung

Da die hier vorgestellte Methode auf Lazy Evaluation Techniken, d.h. Teilrekonstruktionen der Hierarchie verzichtet, kann die Traversierung in der Ray Tracing-Phase wie gewohnt durchgeführt werden. Nachteilig ist lediglich, dass die Hierarchie in Baumstruktur vorliegt, also nicht mit optimaler Cache-Ausnutzung traversiert werden kann. Natürlich könnte man die BVH

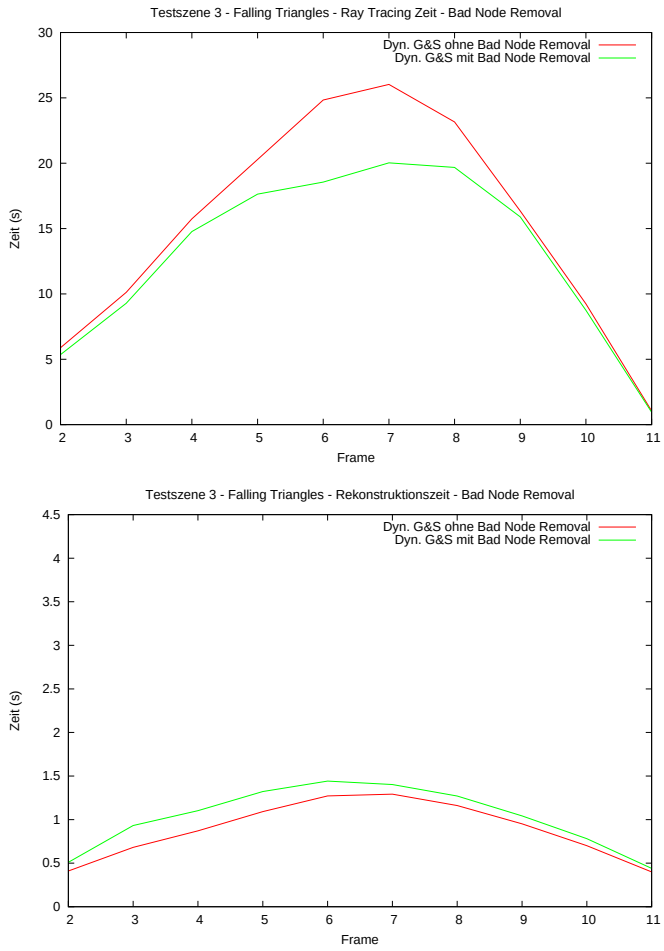


Abbildung 4.13: Unterschied in der benötigten Ray Tracing Zeit (oben) und Rekonstruktionszeit (unten) für die Testszene 3 - Falling Triangles aus Abschnitt 6.

```
void updateDynGandS(){
  updateObjects(); // Aktualisierung der Objektpositionen

  for all objects o{
    if(!o.moved) // Objekt hat sich nicht bewegt
      continue;

    // Aktualisierung des Hüllvolumens
    node = o.hierarchyPointer;
    node.setBBox(o.inquireBounds());

    // Entferne das Objekt aus der Hierarchie
    oldposition = removeFromHierarchy(node);

    // Ausgehend von der alten Position,
    // finde neuen Einfügeknoten
    insertionNode = refitHierarchy(oldposition, node);

    // Füge das Objekt neu ein
    insertIntoHierarchy(insertionNode, node);

    // Markiere den Pfad von der alten und
    // der neuen Position zum Wurzelknoten
    setMarkings(node, oldposition);
  }
  // Finde die ausgedünnten Knoten
  // und füge deren Kinder erneut ein
  removeBadNodes();
}
```

Abbildung 4.14: Pseudocode der Rekonstruktions-Phase des Dynamic Goldsmith and Salmon Verfahrens

nachträglich jeweils in ein 1D-Array umkopieren, was in $O(n)$ möglich wäre. Davon ist jedoch in der Regel abzuraten, da der zeitliche Verlust meist größer wäre als der Gewinn. Die Art der Traversierung, ob mittels Tiefensuche oder nach Kay/Kajiya steht jedoch frei.

Zusammenfassung

In diesem Abschnitt wurde ein Verfahren präsentiert, welche in der Lage ist, dynamische Szenen zu handhaben, indem es sich Lokalitäten in der Bewegung effektiv zu Nutzen macht und somit nur die nötigsten Bereiche der Beschleunigungsstruktur aktualisiert. Das Problem der Ausdünnung wird durch den Einsatz eines effizienten Qualitätskriteriums weitestgehend gelöst. Dabei konnte bisher noch keine nicht konstruierte Szene gefunden werden, in der Ausdünnung dieser Methode Probleme bereitet hätte. Der maximal mögliche Aufwand liegt zwar immer noch bei $O(n \log n)$, dürfte aber in seinem Durchschnitt deutlich geringer sein, als bei einem Neuaufbau.

4.3.3 Dynamic Median-Cut

In ähnlicher Art und Weise wie in Abschnitt 4.3.2, könnte man natürlich auch mit einer auf dem Median-Cut Algorithmus basierenden BVH verfahren, um so eine Neusortierung der Objekte umgehen zu können. Dies soll durch einen effizienten Lösch- und Einfügevorgang, sowie bei Bedarf durchgeführter Rebalancierungsschritte geschehen, welche die Ausgewogenheit der Objektanzahl in den verschiedenen Teilbäumen garantieren, und das QK aus 4.3.2 ersetzen.

Initialaufbau

Da es deutliche Vorteile beim Einfügen von Objekten und späterem Rebalancieren der Hierarchie mit sich bringt, wird zunächst eine kleine Änderung an dem in Formel (4.2) eingeführten Sortierkriterium vorgenommen. Die neue Variante lässt sich folgendermaßen beschreiben:

$$\forall x \in T_l : \forall y \in T_r : (x.key_{\min} \leq y.key_{\min}) \vee (x.key_{\max} \leq y.key_{\max}) \quad , \quad (4.21)$$

wobei $key \in x, y, z$ die Achse darstellt, nach der sortiert wird. T_l die Menge aller Objekte des linken Teilbaumes bezeichnet, T_r die des rechten. x bezeichnet ein Element aus T_l und y eines aus T_r. Der linke Teil der Formel ist dabei

nahezu identisch zu Formel (4.2). Der rechte bietet eine alternative Sortierung. D.h. jedes Objekt besitzt zwei Sortierschlüssel für jede Achse, seinen kleinsten, wie größten Wert entlang dieser.

Dieses Kriterium liefert bei uniform großen Objekten identische Ergebnisse zu Formel (4.2) und führt lediglich bei vollständiger Überlappung zwischen Objekten zu Unterschieden, welche aber in Hinblick auf den Ray Tracing Vorgang mit gleicher Wahrscheinlichkeit schlechter, aber auch besser sein können. Dennoch ermöglicht diese Formel deutliche Vereinfachungen in der Rekonstruktions-Phase, wie im weiteren Verlauf gezeigt werden soll.

Der initiale Aufbau kann zunächst identisch zu Unterabschnitt 4.1.1 Median-Cut durchgeführt werden, mit in jedem Schritt alternierenden Achsen. Um allerdings eine größtmögliche Flexibilität zu erhalten, wird die Beschleunigungsstruktur in Parent-Left/Right Child-Struktur angelegt, d.h. jeder Knoten des Baumes hat über einen Zeiger direkten Zugriff auf seinen Vaterknoten und seine beiden Kinder. Dies erlaubt auch ein Herauslöschen oder Einfügen von Objekten, ohne die gesamte Hierarchie zwangsweise neu aufbauen zu müssen. Die Objekte selbst besitzen erneut einen zusätzlichen Hierarchie-Zeiger, der einen direkten Zugriff auf die Objektknoten in der Hierarchie ermöglicht.

Rekonstruktions-Phase

Nach Berechnung des ersten Frames wird die Rekonstruktions-Phase gestartet, in der sequentiell die Objektpositionen und Hierarchie aktualisiert werden. Nachdem die neue Position für das aktuell untersuchte Objekt berechnet wurde, wird das neue Hüllvolumen für dieses angefordert und mit der des vorherigen Frames verglichen. Sollte keine Änderung stattgefunden haben, kann direkt mit dem nächsten Objekt fortgefahren werden. Sollte dies nicht der Fall sein, wird zunächst einmal entschieden, ob es sich um eine vorteilhafte oder nachteilige Bewegung handelt. Vorteilhaft bedeutet in diesem Zusammenhang, dass sich das Eltern-BV des Objektes verkleinert, sobald es an seine Kinder angepasst wird, d.h. es umschließt seine Kinder auch weiterhin. In diesem Falle werden strukturelle Änderungen in der BVH eingespart, da sich diese lediglich in ihrer Qualität verbessert hat, weil die Gesamtoberfläche der Teilhierarchie, welches das Objekt enthält, gesunken ist, und damit auch die Wahrscheinlichkeit von einem Strahl getroffen zu werden. Dies gilt natürlich nur, falls die Szenenausdehnung konstant geblieben ist. Das Median-Cut Sortierkriterium aus Formel (4.21) wird dabei nicht zwangsläufig weiterhin erfüllt. Man kann es jedoch in Kauf nehmen, da die

Unterteilung nach Objekten ein ungenaueres Gütekriterium bietet, als die Wahrscheinlichkeit von einem beliebigen Strahl getroffen zu werden.

Umschließt der Elternknoten jedoch nicht mehr das bewegte Objekt, so sind gegebenenfalls strukturelle Änderungen an der Hierarchie vorzunehmen. Mittels des Hierarchiezeigers kann der Ankerpunkt des Objektes in der Hierarchie ausfindig gemacht und es effizient herausgelöscht werden. Da der Vaterknoten nun nur noch ein Kind besitzt, wird dieser ebenfalls entfernt und der Bruderknoten tritt an dessen Stelle. Durch die Entfernung des Objektes liegt eventuell ein Ungleichgewicht in einem der darüberliegenden Knoten vor, d.h. die Teilbäume der Kinder enthalten nicht mehr die gleiche Anzahl an Objekten. Ein zusätzlicher Rebalancierungsschritt könnte dies beheben. Da sich Objekte, zumindest solche, die einer einigermaßen stetigen Bewegung unterliegen, meist nicht sehr stark in ihrer Position verändern, ist es sinnvoller, das Objekt erst einmal neu in die Hierarchie einzufügen. Erst dann sollte sich eine Rebalancierung anschließen, da es durchaus geschehen kann, dass das Objekt wieder an seine alte Position zurückfällt.

Wurde das Objekt nun aus der Hierarchie entfernt, so wird versucht den bestmöglichen Platz für es wiederzufinden. Ein simples Vorgehen wäre es dabei, das Objekt Top-Down vom Wurzelknoten aus einzufügen, mittels einer vorher gewählten Heuristik. Gemäß der Annahme, dass sich das Objekt nur geringfügig in Relation zur Gesamtgröße der Szene bewegt hat, ist es, wie auch schon in Abschnitt 4.3.2 bei größeren Szenen effektiver, das Objekt stattdessen von seiner alten Position aus hochzureichen, bis es wieder komplett vom Hüllvolumen eines seiner Vorgängerknoten umschlossen wird, oder den Wurzelknoten erreicht, und von da aus die Suche nach seiner neuen Position in der BVH zu starten. Der so gewählte Knoten wird im weiteren Verlauf Einfügeknoten genannt, da von ihm aus die nachfolgende Einfügeprozedur beginnt.

Die Wahl dieses Vorgehens hat verschieden Vorteile. Geht man von einer moderaten Überlappung der Hüllvolumen zwischen Bruderknoten aus, bedingt durch die Sortierung entlang einer Achse, so besteht eine relativ große Wahrscheinlichkeit, dass dieser Teilbaum die optimale Einfügeposition für unser Objekt enthält. Dies führt dazu, dass in den meisten Fällen statt der gesamten Hierarchie voraussichtlich nur ein kleiner Teilbaum betrachtet werden muss.

Liegt eine Sortierung nach den Minimalwerten der Objekthüllvolumen vor, so genügt es in der Regel beim Einfügen zu testen, ob der *min*-Wert des einzufügenden Objektes kleiner als der des rechten Kindes des aktuellen Einfügeknotens ist. O.B.d.A. gelte, dass das linke Kind die Objekte mit kleinerem

```
geg: o Hüllvolumen des einzufügenden Objektes,
     n aktueller Einfügeknoten
     n.leftChild linkes Kind von n
     n.rightChild rechtes Kind von n
     objCount() liefert Anzahl der Objekte in einem Knoten
     a Achse nach der sortiert wurde

pushDown(BVNode o, BVNode n){
  if(n.hasPrimitive()){
    //Blattknoten erreicht
    combineBVs(o,n);
    return;
  }
  if(o.a_min <= n.rightChild.a_min)
    // Fahre bei linkem Kind fort
    pushDown(o,n.leftChild);
  else
    // Fahre bei rechtem Kind fort
    pushDown(o,n.rightChild);
  if(|n.leftChild.objCount() - n.rightChild.objCount()| > 1)
    rebalance(); // Führe Rebalancierung durch
}
```

Abbildung 4.15: Pseudocode für das Herabreichen eines Objektes

Sortierschlüssel enthält, als das rechte. Ist dies der Fall, müsste es in das linke Kind eingefügt werden, gemäß den Vorgaben aus (4.2), sonst in das rechte. Der Pseudocode aus Abbildung 4.15 veranschaulicht dies.

Auf diese Weise würde zwar gemäß Formel (4.2) die Sortierung aufrecht erhalten, jedoch viele unnötige Rebalancierungsschritte forciert, um eine korrekte Objektverteilung im Baum zu erhalten. Eine Verbesserung läßt sich erzielen, indem man die Anzahl der Objekte in jedem Teilbaum mit in Betracht zieht, sowie Formel (4.21) als Sortierkriterium wählt. Der Pseudocode ist in Abbildung 4.16 gegeben.

Sollte es sich bei dem aktuellen Knoten nicht um einen Blattknoten handeln, fangen die nächsten beiden *if*-Abfragen die Fälle ab, in denen es gar keine andere Möglichkeit gibt, als das Objekt in den entsprechenden Teilbaum weiterzureichen. In den anderen Fällen, steht die Wahl offen, da beide Varianten das Sortierkriterium erfüllen würden. Aus diesem Grund wird der kleinere der

```
pushDown(BVNode o,BVNode n){
  if(n.hasPrimitive()){
    //Blattknoten erreicht
    combineBVs(o,n);
    return;
  }
  if(o.a_min >= n.rightChild.a_min)
    // Fahre bei rechtem Kind fort
    pushDown(o,n.rightChild);
  else if(o.a_max <= n.leftChild.a_max)
    // Fahre bei linkem Kind fort
    pushDown(o,n.leftChild);
  else if(n.leftChild.objCount() > n.rightChild.objCount())
    // Fahre bei rechtem Kind fort
    pushDown(o,n.rightChild);
  else if(n.leftChild.objCount() < n.rightChild.objCount())
    // Fahre bei linkem Kind fort
    pushDown(o,n.leftChild);
  else
    //Anzahl der Objekte in beiden Kindknoten gleich
    füge o gemäß Surface Area Heuristik ein;

  if(|n.leftchild.objCount() - n.rightChild.objCount()| > 1)
   rebalance(); // Führe Rebalancierung durch
}
```

Abbildung 4.16: Pseudocode für das verbesserte Herabreichen eine Objektes

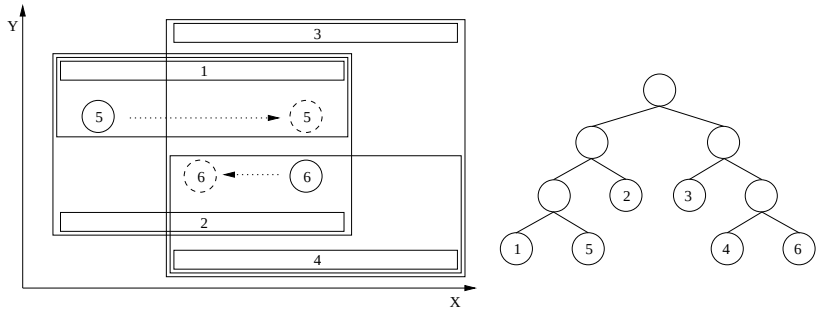


Abbildung 4.17: Durch Überlappung entstehender Fehler in der Sortierung

beiden Teilbäume ausgewählt, um spätere Rebalancierungsschritte zu sparen. Die im letzten Fall angesprochene Surface Area Heuristik ist hier eine lokale Funktion, welche die Oberflächen für den linken und rechten Teilbaum mit und ohne das Objekt berechnet und es so einfügt, dass der Oberflächenzuwachs minimiert wird. Obige stark rekursive Funktion, lässt sich ohne weiteres in eine iterative Variante umschreiben, da in jedem Schritt lediglich der Zeiger auf den aktuellen Knoten aktualisiert werden muss.

Trifft man in der Prozedur auf ein Blatt der Hierarchie, ist der Zielknoten erreicht und das Objekt wird mit diesem zu einem neuen Knoten zusammengefasst. In einem nächsten Schritt werden die Hüllvolumen angepasst, so dass die BVH wieder in einem konsistenten Zustand vorliegt. Dies geschieht, angefangen beim neu erzeugten Knoten, über die Väterknoten, bis notfalls zum Wurzelknoten, oder bis keine neue Anpassung mehr notwendig ist, je nachdem, was zuerst eintritt.

Dabei können jedoch Skalierungen sowie besonders große Objekte die Sortierung aushebeln. Dies ist in Abbildung 4.17 dargestellt. Der dargestellte Knoten sortiert nach der x-Achse. Die Objekte 1,2,3 und 4 verursachen entlang dieser Achse eine starke Überlappung. Bewegen sich die Objekte 5 und 6 nun, wie durch die Pfeile dargestellt, aufeinander zu, so wird irgendwann der Punkt eintreten, an dem sie ihre Reihenfolge entlang der Sortierachse tauschen und dementsprechend den Teilbaum wechseln müssten. Da sie jedoch nie bis zu dem entsprechenden Knoten hochgereicht werden, kann dies nicht geschehen und die Sortierung schlägt fehl. Auch wenn dies einen Bruch des Kriteriums (4.21) darstellt, muss es allerdings nicht bedeuten, dass sich die Hierarchie dadurch verschlechtert, weshalb dies als Trade-Off für eine kürzere Rekonstruktions-Zeit in Kauf genommen wird.

Auf gleiche Art und Weise wird nun mit den restlichen bewegten Objekten verfahren. Dadurch können voraussichtlich Ungleichgewichte in der Objekt-

anzahl in einigen Knoten entstehen, die es nun auszugleichen gilt. Um nicht die komplette Hierarchie durchsuchen zu müssen, wurden die Knoten, an denen in den vorherigen Schritten Änderungen vorgenommen wurden, mit einer Markierung versehen, welche auch sämtliche Väterknoten bis hinauf zum Wurzelknoten erhalten. In einem Top-Down-Verfahren werden nun alle markierten Knoten untersucht und gegebenenfalls ausbalanciert. Diese Prozedur lässt sich mit Hilfe eines Stacks sehr direkt implementieren. Beginnend mit dem ersten markierten Knoten, wird dieser auf den Stack gelegt und solange dieser noch nicht leer ist, wird das oberste Element betrachtet und getestet, ob die Objektverteilung im linken und rechten Kind ausgeglichen ist. Ist dies der Fall, wird das Element aus dem Stack entfernt, untersucht welche Kinder ebenfalls markiert sind und diese auf den Stack gelegt. Sollten die Kinder sich jedoch um mehr als eins in der Anzahl ihrer enthaltenen Objekte unterscheiden, muss im größeren Teilbaum das Objekt gesucht werden, welches gemäß des Sortierkriteriums den Teilbaum wechseln müsste. Normalerweise wäre es sehr aufwändig, dieses Objekt zu finden, da lokal keine Anhaltspunkte vorliegen, in welchem Teilbaum es sich befinden könnte. Durch das erweiterte Sortierkriterium aus Formel (4.21), ist dies jedoch mittels einfacher if-Abfragen in maximal $\log n$ Schritten möglich.

Dies soll anhand eines Beispieles näher erläutert werden: Ein Knoten sortiert entlang der x-Achse und hat o.B.d.A. im linken Teilbaum T_l genau zwei Objekte mehr als im rechten T_r. Bei einem größeren Unterschied würde das Verfahren genauso funktionieren, müsste nur entsprechend oft wiederholt werden. Die Suche beginnt folglich bei T_l, indem getestet wird, ob der $x_{\max}$-Wert des Hüllvolumens seines linken Kindes größer ist, als der seines rechten. Falls ja, muss sich das gesuchte Objekt im linken Teilbaum befinden. Falls nein, dann im rechten. Dies folgt aus dem Umstand, dass jedes Hüllvolumen das seiner Kinder exakt umschließt. Somit muss sich folgerichtig das gesuchte Objekt immer in dem Teilbaum befinden, welcher den größeren $x_{\max}$-Wert aufweist. Nun wird rekursiv fortgefahren, bis ein Blatt erreicht wird, dieses entspricht dem gesuchten Objekt. Dadurch wird nach maximal $\log n$ Schritten das auszutauschende Objekt gefunden. Es wird aus der Hierarchie entfernt und in T_r gemäß der bereits beschriebenen Einfügeprozedur einsortiert. Alle in diesem Schritt besuchten Knoten müssen nun ebenfalls als Kandidaten für eine weitere Rebalancierung markiert werden, da dort gleichermassen Ungleichgewichte entstehen konnten. In Abbildung 4.18 ist der Rebalancierungsvorgang noch einmal als Pseudocode gegeben. `tolerance` ist dabei der kleinste nicht mehr erlaubte Unterschied an Objekten im linken und rechten Kind eines Knotens, also bei einer gleichmäßigen Aufteilung 2.

Mit einer einfachen Sortierung nach *min*-, *max*-Werten oder dem Mittel-

```
void rebalance()
{
  stack s;
  s.push(root);

  while(!s.empty){
    BVNode node  = s.extractTop();
    BVNode left  = node.leftChild;
    BVNode right = node.rightChild;
    int numObjLeft  = left.objCount();
    int numObjRight = right.objCount();
    if(|numObjLeft - numObjRight| < tolerance)
      // Keine Rebalancierung notwendig
      continue;

    if(numObjLeft < numObjRight){
      while(numObjRight - numObjLeft >= tolerance){
        // Rebalancierungsschritt
        pushDown(getMin(right, node.getAxis()), left);
        // Kinder haben sich möglicherweise verändert
        left  = node.leftChild;
        right = node.rightChild;
      }
    }
    else{
      while(numObjLeft - numObjRight >= tolerance){
        // Rebalancierungsschritt
        pushDown(getMax(left, node.getAxis()), right);
        // Kinder haben sich möglicherweise verändert
        left  = node.leftChild;
        right = node.rightChild;
      }
    }
    // Knoten ist rebalanciert, fahre mit Kindern fort
    s.push(left);
    s.push(right);
  }
}
```

Abbildung 4.18: Pseudocode der Rebalancierung

```
void updateDynMedCut(){
  updateObjects(); // Aktualisierung der Objektpositionen

  for all objects o{
    if(!o.moved) // Objekt hat sich nicht bewegt
      continue;

    // Aktualisierung des Hüllvolumens
    node = o.hierarchyPointer;
    node.setBBox(o.inquireBounds());

    // Reiche das Objekt bis zum neuen Einfügeknoten hoch
    topNode = liftUp(node);

    // Einfügen des Knotens
    pushDown(node, topNode);
  }
  // Führe das Rebalancing durch
  rebalance();
}
```

Abbildung 4.19: Pseudocode der Rekonstruktions-Phase des Dynamic Median-Cut Verfahrens

punkt, wie es sonst üblich ist bei Median-Cut, wäre das Auffinden des richtigen Objektes deutlich aufwändiger, da nicht lokal bestimmt werden kann, in welchem Kindknoten sich das gesuchte Objekt befindet. Aufwändigere Suchverfahren wären notwendig, welche entweder Brute-force den gesamten Teilbaum durchsuchen, oder Graphsuchalgorithmen verwenden, da andernfalls das Sortierkriterium verletzt werden könnte.

Zur besseren Übersicht ist in Abbildung 4.19 nocheinmal die gesamte Rekonstruktions-Phase als Pseudocode dargestellt. Man könnte natürlich das Kriterium zur Ausbalancierung eines Knotens lockern, indem man einen größeren Unterschied in der Objektanzahl erlaubt als eins. In unseren empirischen Tests hat sich jedoch ergeben, dass durch die Einfügeprozedur bereits eine relativ gute Aufteilung der Objekte stattgefunden hat, so dass es in der Regel keinen übermässig großen Unterschied zwischen einem strengen und gelockerten Kriterium gibt. Von daher sollte es empfehlenswerter sein, einen niedrigen Wert zu wählen, um eine höhere Qualität der Hierarchie garantieren zu können.

Traversierung

Da die Hierarchie nach Abschluss der Rekonstruktions-Phase komplett in einem konsistenten Zustand vorliegt, kann die Traversierung wie gewöhnlich durchgeführt werden. Auch hier ist natürlich zu einem gewissen Grade nachteilig, dass die BVH in Baumstruktur vorliegt, ein Umkopieren nach jeder Rekonstruktions-Phase wäre jedoch voraussichtlich zu aufwändig.

Zusammenfassung

In diesem Abschnitt wurde ein weiteres Verfahren für Ray Tracing von dynamischen Szenen präsentiert. Ausgehend von einer nach der Median-Cut Methode erstellten Hierarchie, vermeidet sie komplette Neuerstellungen der BVH durch Bearbeitung lokaler Teilbäume. Eine Einfügeprozedur wurde vorgestellt, die einerseits versucht, das Sortierkriterium des Median-Cut Algorithmus zu erhalten, andererseits unnötige Rebalancierungsschritte verhindert. Außerdem ein verändertes Sortierkriterium, welches es möglich macht beim Ausgleichen eines Knotens die benötigten Objekte in jeweils maximal $O(\log n)$ Schritten zu finden.

4.3.4 Local Sort

In den letzten beiden vorgestellten Verfahren wurden bewegte Objekte grundsätzlich aus der BVH entfernt und neu eingefügt. Theoretisch besteht die Möglichkeit auch diesen Schritt abhängig von einem Qualitätscheck zu machen und nur diejenigen Teile einer Hierarchie neu aufzubauen, welche diesen nicht bestanden haben. Doch auch dies bedeutet noch viel unnötige Arbeit. Denn in Bereichen der Szene, die vom Betrachter aus gar nicht zu sehen sind, ist es auch unnötig, irgendetwas an der Beschleunigungsstruktur zu verändern. Des weiteren stellt sich die Frage, ob es nicht bereits genügen würde, mittels vorhandener Teilbäume diese Bereiche neu aufzubauen. Das Verfahren nach Goldsmith und Salmon lässt sich, wie bereits in Abschnitt 4.3.2 gezeigt, schließlich auch für interne Knoten verwenden und nicht nur für Objekte.

Ausgehend von diesen Beobachtungen soll in diesem Abschnitt eine Methode vorgestellt werden, welche folgende Punkte austestet:

- Minimierung der Rekonstruktionsphase durch ein einfaches Refit

- Anwendung eines Qualitätskriteriums, um während der Traversierung erneuerungswürdige Knoten ausfindig zu machen
- Anwendung einer Lazy Evaluation Strategie, um lediglich die für die Bildberechnung relevanten Bereiche der BVH bei Bedarf neu aufzubauen.
- Beschleunigter Neuaufbau von Teilbereichen der Hierarchie durch Verwendung kompletter Teilbäume, welche das Qualitätskriterium noch erfüllen, statt auf Objektebene.

Initialaufbau

Der Aufbau der BVH verläuft in gleicher Weise, wie bereits in Abschnitt 4.3.2. D.h. die Hierarchie wird erneut gemäß dem Verfahren von Goldsmith und Salmon [GS87] erstellt. Auch hier besteht natürlich die Möglichkeit, die Beschleunigungsstruktur vor Berechnung des ersten Bildes mehrfach aufzubauen, um möglicherweise durch Vertauschung der Objektreihenfolge eine bessere BVH zu erhalten. Zusätzlich dazu wird für jeden Knoten sein initiales Qualitätsmaß bestimmt. Da zunächst immer nur ein Refit auf der BVH in der Rekonstruktionsphase durchgeführt werden soll, kann das Qualitätskriterium aus Abschnitt 4.3.2 vereinfacht werden. Denn solange keine Strukturveränderung an der BVH vorgenommen wird, verbleibt die Objektanzahl konstant. Dies reduziert das Kriterium auf die relative Veränderung der Oberfläche. Einzige Ausnahme wäre hierbei das komplette Rauslöschen eines Objektes aus der Hierarchie. In diesem Falle würde es Sinn machen, das Qualitätskriterium unverändert zu übernehmen.

Rekonstruktions-Phase

Die Rekonstruktions-Phase fällt bei diesem Verfahren sehr simpel aus, da der Großteil der Arbeit in die Ray Tracing-Phase verlagert wird. Dadurch reduziert sich die Aufgabe der Rekonstruktions-Phase auf ein Refit sowie eine Neuberechnung des Qualitätskriteriums der veränderten Knoten. Beides ist in $O(n)$ möglich. Es bestünde natürlich die Möglichkeit das Refit, wie in Abschnitt 4.3.1 beschrieben, sequentiell für jedes Objekt durchzuführen, doch dies würde unnötig viel Arbeit bedeuten. Ein Refit mittels Priority Queue (PQ) ist nicht ohne weiteres durchzuführen, da in einem späteren Schritt Teilbäume der Hierarchie neu angeordnet werden sollen. Dadurch würde sich häufig die Tiefe der einzelnen Knoten verändern, was die Anwendung einer

PQ sehr schwierig werden lässt. Eine bessere Wahl bildet daher ein abgewandelter Complete Refit. Um den nicht unerheblichen Aufwand zu vermeiden oder zumindest abzuschwächen, werden vor dem Refit sämtliche Knoten markiert, welche bewegte Objekte in dem ihnen unterliegenden Teilbaum enthalten, damit nur diese später aktualisiert werden müssen. Dies geschieht folgendermaßen: Sequentiell werden die Objektpositionen aktualisiert. Bei jedem Objekt, wird zunächst getestet, ob es sich bewegt hat. Falls nicht, kann mit dem nächsten fortgefahren werden. Falls ja, so werden entlang des Pfades vom Objekt bis zum Wurzelknoten Markierungen gesetzt. Während des Refits können nun beginnend beim Wurzelknoten entlang der Markierungen genau die Knoten ausfindig gemacht werden, welche sich verändert haben und die Hüllvolumen entsprechend angepasst werden. Bereiche, in denen sich nichts verändert hat, werden so automatisch außer Acht gelassen.

Parallel dazu können auch gleich die neuen Qualitätswerte für die markierten Knoten berechnet werden und gegebenenfalls eine weitere Markierung, genannt Bad Node-Markierung, gesetzt werden. Diese gibt an, dass die dem Knoten unterliegende Hierarchie nicht mehr den vom Benutzer festgelegten Ansprüchen genügt. Letztere muss jedoch nicht nur bei dem entsprechenden Knoten gesetzt werden, sondern auch noch bei den m Level darüber liegenden Knoten, wobei m ein benutzerdefinierter Parameter ist. Dies hat folgenden Grund: Zwar mögen die Hüllvolumen oberhalb des Bad Nodes noch den Qualitätsansprüchen genügen, doch würde in vielen Fällen ein Neuaufbau des unterhalb des Bad Nodes liegenden Teilbaumes, mit diesem als Wurzelknoten, zu keiner Verbesserung führen. Ein Beispiel dafür ist in Abbildung 4.20 gegeben. Objekt 2 bewegt sich im Laufe der Zeit immer weiter entlang der positiven x-Achse (Abb. 4.20(b)). Der Knoten, welcher Objekt 1 und 2 umschließt, sinkt daraufhin in seiner Qualität und wird irgendwann markiert. Da er jedoch lediglich diese beiden Objekte enthält, würde ein Neuaufbau dieses Knotens lediglich zum gleichen Ergebnis führen. Wird die Markierung jedoch höher gesetzt, so werden die Objekte 3 und 4 ebenfalls in den Neuaufbau miteinbezogen, wodurch eine bessere BVH erstellt wird (siehe Abb. 4.20(c)).

Doch auch damit wird verständlicherweise oft nur eine suboptimale Hierarchie erzeugt, da ein Objekt unter Umständen erst deutlich höher in der Hierarchie den Teilbaum wechseln würde. Sollte sich jedoch die Bewegung fortsetzen, werden irgendwann auch die darüberliegenden Knoten markiert werden, so dass ein Objekt nach einer gewissen Zeit meist seine optimale Position finden wird. Bis zu diesem Zeitpunkt muss die suboptimale Hierarchie in Kauf genommen werden. Verschiedenen durchgeführten Tests zufolge scheint $m = 3$ ein guter Standardwert zu sein.

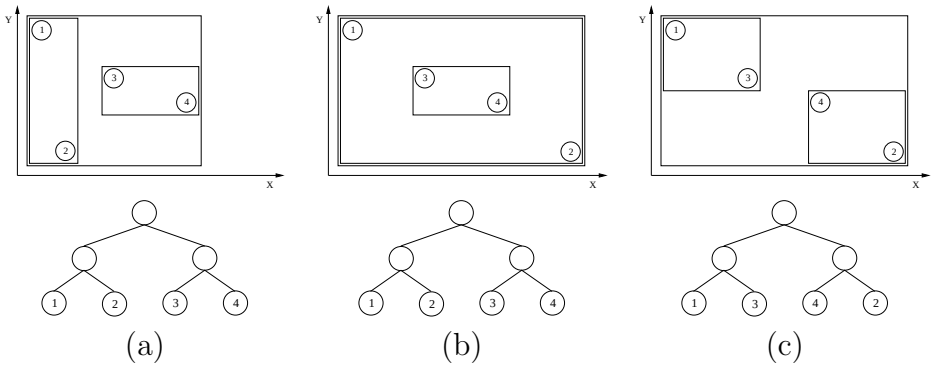


Abbildung 4.20: Lokale Neusortierung

Damit wäre die Rekonstruktions-Phase bereits abgeschlossen. Der entsprechende Pseudocode ist in Abbildung 4.21 gegeben.

Traversierung

Bisher unterscheidet sich die Local Sort-Variante nicht besonders von einem normalen Refit. D.h. die Qualität der Hierarchie wird mit der Zeit mehr und mehr leiden. Dies soll nun während der Traversierung abgefangen werden.

Solange man während des Ray Tracing-Vorgangs nicht auf einen Bad Node trifft, wird wie gewohnt verfahren. Andernfalls muss die Traversierung unterbrochen und die Rekonstruktion für diesen Knoten durchgeführt werden. Beginnend beim markierten Knoten werden alle n Level tiefer liegenden Knoten aufgesammelt und in einer Liste zwischengespeichert. Eine Ausnahme findet statt, wenn es sich bei einem der einzusammelnden Knoten ebenfalls um einen Bad Node handelt. In diesem Falle werden rekursiv alle Nachfahren dieses Knotens eingesammelt, welche ihr Qualitätskriterium noch erfüllen, unabhängig davon, wie viele Ebenen tiefer sie liegen sollten. Dies ist leider unumgänglich, da Bereiche einer Szene, welche lange Zeit nicht beachtet wurden, aber einer starken Bewegung unterlagen, gegebenenfalls in ihrer Qualität so stark gesunken sind, dass ein Anordnen der lediglich n Ebenen tiefer liegenden Knoten nicht mehr genügen würde. Mit dieser Erweiterung wiederum reicht es völlig aus, $n = 1$ zu wählen. Die entsprechenden Bad Nodes können gelöscht werden. Aus dieser Liste wird nun, ausgehend vom letzten traversierten Knoten, eine neue Hierarchie aufgebaut. Für die neu erzeugten Knoten werden auch neue Qualitätsgrenzen berechnet. Dabei können die eingesammelten Teilbäume genau wie Objekte in die Hierarchie eingefügt werden. Sobald sie eingefügt wurden, werden sie jedoch wiederum wie normale Teile der BVH behandelt, d.h. weitere Elemente aus der Liste könnten theoretisch

```
void updateLocalSort(){
  updateObjects(); // Aktualisierung der Objektpositionen

  for all objects o{
    if(!o.moved) // no movement of this object
      continue;

    // Aktualisierung des Hüllvolumens
    node = o.hierarchyPointer;
    node.setBBox(o.inquireBounds());

    // Setzen der Markierungen für das Refit
    setMarkings(node);
  }

  // Anpassen der Hüllvolumen und
  // Neuberechnung des Qualitätskriteriums
  refitHierarchy();
}
```

Abbildung 4.21: Pseudocode der Rekonstruktions-Phase des Local Sort Verfahrens

auch weiter in diese herabgereicht werden. Ist dieser Vorgang abgeschlossen, wird mit der Traversierung fortgefahren.

Eine Traversierung nach Kay/Kajiya [KK86] ist dabei von Vorteil. Denn es wird das absolute Minimum an Knoten vom Strahl besucht, da die Traversierung entlang des Strahles verläuft. Dies bedeutet weiterhin, dass ebenfalls lediglich das absolute Minimum an Knoten der BVH neu aufgebaut werden muss. Da die Tiefensuche die Strahlrichtung nicht beachtet, ist es durchaus möglich, dass Bereiche der Szene traversiert werden, die keinerlei Beitrag zum Bild liefern.

Da die gefundenen Schnittpunkte eines Strahles mit der Szene in bereits aktualisierten Bereichen liegen, ist es meist von Vorteil Schattenstrahlen vom Schnittpunkt Richtung Lichtquelle zu versenden und nicht umgekehrt. Vor allem in komplexeren Szenen, mit mehreren Lichtquellen und voneinander streng abgetrennten Bereichen, wäre dies wichtig.

Zusammenfassung

In diesem Abschnitt wurde ein Verfahren vorgestellt, welches extensiven Gebrauch von einer Lazy Evaluation Strategie macht, um lediglich die absolut notwendigen Bereiche einer Szene zu aktualisieren, während der Rest einem simplen Refit unterzogen wird. Dies verkürzt die Rekonstruktions-Phase enorm, da der Aufwand auch im Worst Case bei einer Komplexität von $O(n)$ liegt. Der Neuaufbau findet zudem nicht auf Objektebene statt, sondern komplette Teilbäume werden verwendet, was diesen ebenso beschleunigt.

Die Anwendung einer Lazy Evaluation Technik hat zudem noch andere Vorteile. In realen Szenen gibt es häufig periodische, d.h. wiederkehrende Bewegungen, z.B. Avatare, die immer dieselbe Strecke ablaufen, Blätter eines Baumes, die sich im Wind wiegen, etc., auch wenn das Wissen darüber von vornherein nicht gegeben sein mag. Hier erweist sich Lazy Evaluation als sehr starkes Konzept, da es zudem einer Teilszene sozusagen die Chance gibt, sich auch selbst gegebenenfalls wieder zu reparieren.

4.3.5 Dynamic Median-Cut 2

Auch wenn sich das Verfahren nach Goldsmith und Salmon [GS87] meist als überlegen gegenüber dem Median-Cut Schema von Kay/Kajiya [KK86] gezeigt hat, so gibt es doch Gründe sich auch näher mit diesem Verfahren und seiner möglichen Anwendung für dynamische Szenen zu beschäftigen. Denn bedingt durch hochoptimierte Sortieralgorithmen, bzw. Teilsortie-

rungen, kann eine BVH nach dem Median-Cut Verfahren von Kay/Kajiya durchaus für ca. 2000 Objekte bereits in Echtzeit erstellt werden und liefert dennoch gute Ergebnisse auch für die Ray Tracing-Phase. Genau genommen eignet sich der Median-Cut für uniform verteilte Objekte sogar besser, da bei diesem keine Abhängigkeit der Einfügereihenfolge besteht, welche sich gerade bei solchen Szenen bemerkbar macht.

Die Anwendung einer der Heuristiken aus Abschnitt 4.2.2 kann darüber Auskunft geben, welche Teile der BVH lediglich neu aufgebaut werden müssten. Geschieht dies zudem erst während der Ray Tracing Phase und lediglich in den Bereichen, welche wirklich einen Beitrag zum Bild leisten, wird der Aufwand nochmals minimiert. Zudem kann der Neuaufbau beschleunigt und die Traversierung verbessert werden, indem davon Gebrauch gemacht wird, dass sich die Struktur einer Median-Cut Hierarchie niemals ändert, solange keine Objekte hinzugefügt oder gelöscht werden. Lediglich das Objekt, auf welches ein Blattknoten verweist, kann wechseln.

Insgesamt ergeben sich damit die folgenden Punkte, welche in dieser Methode ausgetestet werden sollen:

- Minimierung der Rekonstruktionsphase durch ein einfaches Refit
- Anwendung eines Qualitätskriteriums, um während der Traversierung erneuerungswürdige Knoten ausfindig zu machen
- Anwendung einer Lazy Evaluation Strategie, um lediglich die für die Bildgenerierung relevanten Bereiche der BVH bei Bedarf neu aufzubauen.
- Ausnutzung der gleichbleibenden Struktur einer Median-Cut Hierarchie, sowohl für die Traversierung, als auch für die Rekonstruktion.

Initialaufbau

Wie bereits erwähnt wird das Median-Cut Verfahren nach Kay und Kajiya [KK86] zur Erstellung der Hierarchie verwendet. Allerdings werden dabei nicht einfach die Achsen alternierend verwendet, nach denen sortiert wird, sondern die Unterteilung findet stets entlang der längsten Achse statt. Solange die Ausdehnungen entlang aller Achsen relativ gleichmäßig ist, ist dies meist nicht von bedeutendem Vorteil [SH93]. Dies ändert sich jedoch, sollten die Verhältnisse zueinander stark unterschiedlich sein.

Ausgehend von der Feststellung, dass sich die Struktur einer Median-Cut BVH in der Regel nicht verändert, kann die Hierarchie vorab, passend zur

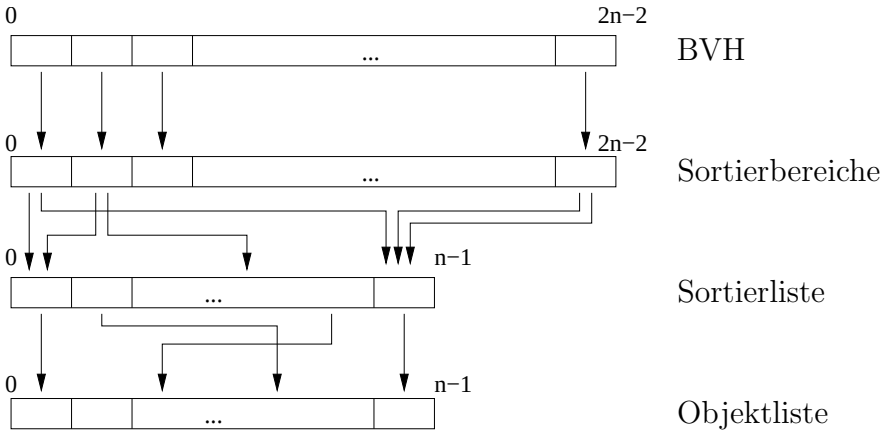


Abbildung 4.22: Struktureller Aufbau für die Dynamic Median-Cut 2 Methode

gewählten Traversierungsmethode, optimal in einem 1D-Array angelegt werden. Sämtliche benötigte Zusatzinformation kann in separaten Strukturen gespeichert werden, so dass zudem eine sehr gute Cacheausnutzung während des Ray Tracings gewährleistet werden kann. Die Objekte sowie ihre direkten Hüllvolumen werden in separaten Listen gespeichert, auf die die Blattknoten der BVH verweisen. Jeder innere Knoten kennt zudem den Start- und Endindex für die in seinem Teilbaum enthaltenen Objekte auf dieser Sortierliste, und kann somit bei Bedarf eine effiziente Neusortierung vornehmen. Die beschriebene Anordnung ist in Abbildung 4.22 dargestellt. Der Übersichtlichkeit halber wurde eine leichte Vereinfachung dabei vorgenommen und die Zeiger der Blattknoten auf die Sortierliste ausgespart.

Für jeden Knoten werden nun die initialen Qualitätsgrenzen berechnet und ebenfalls in einer separaten Liste gespeichert. Diese dienen später der Erkennung von Bad Nodes, also Knoten, die das festgelegte Qualitätskriterium nicht mehr erfüllen. Da jeder innere Knoten genau zwei Kindknoten besitzt, was einige Berechnungen vereinfacht, wird als QK eine Kombination der in Abschnitt 4.2.2 vorgestellten Heuristiken gesetzt. Erstens die Wahrscheinlichkeit, dass beide Kinder von einem Strahl getroffen werden, sowie zweitens die Oberflächenveränderung, um die approximationsbedingten Schwächen des ersten Kriteriums auszugleichen. Wie strikt die Qualitätskriterien sind, hängt dabei von den Benutzereinstellungen ab. Die Wahl derselben sollte immer von der Frage abhängen, wie wichtig eine gute Hierarchie gegenüber einer schnellen Aktualisierung ist. Man sollte jedoch stets beachten, dass vor allem in niedrigeren Leveln der Hierarchie, in der Nähe der Blätter, schnell sehr hohe Werte für das erste Kriterium entstehen können, z.B. durch zwei

nahe beieinanderliegende, parallele Dreiecke. Und selbst eine optimale Unterteilung kann auch in höheren Leveln leicht einen Wert von $\frac{1}{3}$ erreichen, wobei 1 eine komplette Überlappung darstellt und 0 der beste Wert wäre. Aus diesem Grund ist es schwer einen Wert zu bestimmen, der in jeder Szene gute Ergebnisse liefert. In dem hier vorgestellten System wurde sich für einen Standardwert von 0.7 entschieden, welcher empirisch anhand verschiedener Testszenen ermittelt wurde. Wird dieser Überschritten, wird der Knoten als Bad Node markiert. Um jedoch zu verhindern, dass auch statische Bereiche die festgesetzten Qualitätsgrenzen überschreiten, müssen diese eventuell individuell angepasst werden. Sollte der Initialwert eines Knotens höher liegen, als der benutzerdefinierte Threshold, so wird der höhere von beiden als Grenze für diesen Knoten gesetzt. Dies ist leider unumgänglich um beliebige Szenen und Objektgrößen zu erlauben. Für die Oberflächenveränderung werden standardmäßig die Werte 0.5 und 2 gesetzt.

Rekonstruktions-Phase

Die Rekonstruktions-Phase besteht bei dieser Variante lediglich aus einem Refit, um konsistente Hüllvolumen zu erhalten, sowie eines Qualitätstestes für die inneren Knoten. Da die Hierarchie in einem 1D-Array vorab angelegt wurde, kann ein Complete Refit sehr effizient durchgeführt werden, indem einmal von hinten nach vorne über dieses iteriert wird und die Hüllvolumen angepasst werden [Ber97]. Gleichzeitig werden die veränderten Qualitätswerte berechnet und notfalls Markierungen für die Bad Nodes gesetzt. Zwar würde sich für wenige bewegte Objekte evtl. ein Refit mittels Priority Queue besser eignen, doch wirft dies einige Probleme in Zusammenarbeit mit der Lazy Evaluation Technik auf. Durch die mögliche Neuanordnung der Sortierliste während der Traversierung und dadurch, dass die BVH-Blattknoten nur auf den Index dieser Liste verweisen, geschieht es, dass Objekte zwischen zwei Rekonstruktions-Phasen ihre Position im Array verändern, ohne dass die entsprechenden darüberliegenden Hüllvolumen angepasst werden. Dies wiederum hat zur Folge, dass auch Knoten der BVH in die PQ eingefügt werden müssten, bei denen sich das Objekt eigentlich nicht bewegt hat. Dadurch kann dieses Verfahren sehr leicht aufwändiger werden als ein Complete Refit. Abbildung 4.23 gibt den Pseudocode der Rekonstruktions-Phase wieder.

Traversierung

Um die Anzahl an nötigen Neusortierungen der Objekte abermals so gering wie möglich zu halten, wird auch in dieser Methode eine Traversierung nach

```
void updateDynMedCut2(){
  updateObjects(); // Aktualisierung der Objektpositionen

  // Aktualisierung der Hüllvolumen in der Sortierliste
  updateObjBVList();

  // Complete Refit
  refitHierarchy();

  // Markierung der Bad Nodes
  recalcQuality();
}
```

Abbildung 4.23: Pseudocode der Rekonstruktions-Phase des Dynamic Median-Cut 2 Verfahrens

Kay/Kajiya durchgeführt. Wird dabei ein innerer Knoten aktiv, welcher eine Markierung trägt, so wird die Traversierung unterbrochen und das Update des Knotens gestartet. Anstatt den kompletten darunterliegenden Teilbaum aufzubauen, werden jedoch nur die Knoten des nächsten Levels erzeugt und diese ebenfalls markiert, damit der Aufbau dort fortgesetzt wird, sollte ein Strahl sie schneiden. Da durch das Refit gewährleistet ist, dass das Hüllvolumen des Knotens genau die in ihm enthaltenen Objekte bestmöglich umschließt, wird lediglich eine erneute Anpassung für die Kindknoten benötigt. Es wird zunächst die längste Achse des Knotens ermittelt, welche anhand der Ausdehnung des Hüllvolumens sehr simpel gewonnen werden kann. Danach wird eine Sortierung der Objekte entlang dieser Achse vorgenommen und die Hüllvolumen der Kindsknoten berechnet, bevor mit der Traversierung fortgefahren wird.

Eine komplette Sortierung der Objekte entlang der festgesetzten Achse ist jedoch überflüssig, da es ausreichend ist, die betroffenen Objekte in zwei Partitionen aufzuteilen, so dass das Sortierkriterium gilt:

$$\forall x \in P_l : \forall y \in P_r : (x.key \leq y.key) \tag{4.22}$$

P_l und P_r sind dabei die zu erstellenden Partitionen. Die Anordnung der Elemente innerhalb ist nicht von Belang. Es genügt also das Objekt M zu finden, welches den Median darstellt und die Partitionen so zu erstellen, dass gilt:

$$\forall x \in P_l : \forall y \in P_r : (x.key \leq M.key \leq y.key) \tag{4.23}$$

Dies kann in einem durchschnittlichen Aufwand von $O(n)$ mittels des Algorithmus von Hoare zum Auffinden des k-ten Elementes durchgeführt werden [Ben88]. Dabei handelt es sich um eine Variante des Quicksort-Algorithmus, wobei in jedem Rekursionsschritt lediglich die größere Partition weiter untersucht wird. Dieses Verfahren funktioniert umso besser, je näher man mit dem gewählten Testmedian an dem wirklichen Median liegt. Da im Array bereits eine gewisse Sortierung vorliegt, stellt die genaue Mitte zwischen Start- und Endindex eine gute Wahl für den Startwert dar.

Zusammenfassung

Die Vor- und Nachteile des hier vorgestellten Verfahrens sind dabei sehr ähnlich zu der in Abschnitt 4.3.4 vorgestellten Local Sort-Methode. Beide Methoden nehmen, bedingt durch die Qualitätskriterien, eine suboptimale Hierarchie in Kauf. Sollte jedoch ein Neuaufbau eines Teilbaumes stattfinden, so wird er bei dem in diesem Abschnitt vorgestellten Verfahren bestmöglich erstellt, da dies durch optimierte Verfahren verhältnismässig schnell möglich ist. Dabei wird zusätzlich auf eine Lazy Evaluation Strategie gesetzt, welche lediglich die absolut notwendigen Bereiche der Szene neu erstellt. Hinzu kommt, dass bei den meisten realistischeren Szenen hoch gelegene Knoten eher seltener neu erzeugt werden müssen.

In Szenen ohne jegliche Verdeckung ist es zudem möglich, dass ein kompletter Neuaufbau erzwungen wird. D.h. es können keine Rekonstruktions-Zeiten garantiert werden. Außerdem würde ein Löschen oder Einfügen von Objekten dafür sorgen, dass die Struktur der Hierarchie verändert werden müsste. Dadurch wäre abermals ein kompletter Neuaufbau notwendig. In Szenen mit viel verdeckter Geometrie sind jedoch mit diesem Verfahren gute Ergebnisse zu erwarten.

4.3.6 Loose Bounding Volume Hierarchy

Einer der größten Schwachpunkte der bisher vorgestellten Verfahren basiert darauf, dass es nicht möglich war, den Rekonstruktions-Vorgang in einer besseren Komplexität als $O(n \log n)$ bei n Objekten durchzuführen. Auch Lazy Evaluation Techniken verschieben den Aufwand lediglich in die Ray Tracing-Phase. Dies mag bei einer moderaten Anzahl von bewegten Objekten zwar noch durch den logarithmischen Aufwand während der Ray Tracing-Phase wett zu machen sein. Letztendlich sorgt es aber dafür, dass die Ray Tracing

Methode irgendwann ihren größten Vorteil gegenüber den Rasterisierungsverfahren verliert, welche zumeist in $O(n)$ laufen.

Ein möglicher Grund für die erhöhte Komplexität in der Rekonstruktions-Phase liegt darin, dass in einer BVH in der Regel keine direkte Einfügeposition berechnet werden kann, sondern die Hierarchie Ebene für Ebene durchlaufen werden muss. Im folgenden Abschnitt soll ein Verfahren vorgestellt werden, welches es erlaubt die Position der Objekte in einer vorab angelegten Hierarchie direkt zu berechnen und auch die Anpassung der Hüllvolumen in maximal $O(m)$ vornimmt, wobei m die Anzahl der Knoten in der Hierarchie sind. Dabei wird ein hybrides System verwendet, welches die Vorzüge von BVHs, wie engumschließende Hüllvolumen, und den Umstand dass ein Objekt immer nur in einem Knoten der Hierarchie vorliegt, kombiniert mit einer spatialen Anordnung der Hüllvolumen, welche ein direktes Bestimmen der Einfügeposition von Objekten in $O(1)$ ermöglichen. Damit wird eine Gesamtkomplexität von $O(n + m)$ erreicht, bei n Objekten und m Knoten in der Hierarchie, bzw. sogar $O(n)$, da m vorab fest gewählt wird. Dies stellt eine deutliche Verbesserung zu der sonst üblichen Komplexität von $O(n \log n)$ dar.

Die größten Ähnlichkeiten finden sich hierbei zu den Hierarchical Grids von Reinhard *et al.* [RSH00] und den Loose Octrees von Ulrich [Ulr00], daher auch der Name *Loose Bounding Volume Hierarchy*. Das Verfahren dürfte sich dabei besonders für Szenen mit mehreren tausend bewegten Objekten oder Primitiven eignen, wie es etwa bei deformierbaren Objekten der Fall sein kann. Aber anders als in bisherigen Ansätzen auf diesem Gebiet, in denen eigentlich lediglich die Hüllvolumen angepasst wurden [LAM03], können in dieser Methode die Objekte ihre Position in der Hierarchie verändern und erlauben damit vollkommen willkürliche, nicht-deterministische Bewegungen.

Im Folgenden soll die Struktur der Loose Bounding Volume Hierarchy beschrieben werden, inklusive direkter Positionsbestimmung in der Hierarchie für animierte Objekte, sowie die Rekonstruktions-Phase und Traversierung. Zum Abschluss werden noch einige Vor- und Nachteile dieser Methode aufgeführt.

Initialaufbau

Bei der Loose Bounding Volume Hierarchy handelt es sich um einen ausbalancierten, binären Baum mit fester, benutzerdefinierter Tiefe. Diese sollte in Abhängigkeit von der Objektanzahl gewählt werden, wobei ein Überschätzen nur einen geringen Unterschied macht, wie noch gezeigt wird. In diesem Sys-

tem wurde sich für eine Standardtiefe von 18 entschieden.

Üblicherweise werden Knoten einer BVH nur bei Bedarf angelegt, da leere Knoten von keinem Nutzen für die Traversierung sind. Paradoxerweise wird die verringerte Komplexität gerade durch dieses Hinzufügen von Knoten erreicht. Um ein direktes Einfügen und Löschen von Objekten zu erlauben, muss die Beschleunigungsstruktur vorab mit einer festgelegten Tiefe angelegt und nicht genutzte Knoten mit einem entsprechenden Empty-Flag versehen werden, zunächst alle. Zwar erfordert das Empty-Flag bei der Traversierung einen zusätzlichen Aufwand, da stets vorab abgefragt werden muss, ob ein Knoten aktiv ist oder nicht, doch ist dies ein kleines Opfer für den beschleunigten Einfügevorgang.

Die Hierarchie kann somit optimal im Speicher als 1D-Array abgelegt werden. In dieser Methode wird eine Breadth-First Order verwendet um eine optimale Traversierung nach Kay/Kajiya [KK86] zu gewährleisten und um die Möglichkeit zu haben, anhand des Indizes eines Knotens seine Kinder zu bestimmen. Wird ein ungenutzter Eintrag an den Anfang des Arrays gesetzt, so dass der Wurzelknoten den Index 1 besitzt, so sind die Indices der Kinder eines Knotens mit Index n jeweils $2n$ und $2n + 1$. Dadurch benötigt ein Knoten für seine Speicherung lediglich sechs Float-Werte (meist 4 Byte), zur Speicherung der Boxenausdehnung, einen Zeiger (4 Byte) auf die enthaltenen Objekte, sowie theoretisch ein Bit für das Empty-Flag, welches jedoch als Byte abgelegt ist. Somit können die Hüllvolumen mit lediglich 29 Byte beschrieben werden, was sogar eine noch kompaktere Darstellung ist als die Repräsentation von Smits [Smi98], allerdings nur von geringerem Vorteil sein dürfte gegenüber der 32 Byte Darstellung, weil die Cachelines heutiger Prozessoren meist 64 oder 128 Byte umfassen.

Der Einfügevorgang läuft dann wie folgt ab: Zunächst wird die Szene von einem einzigen Voxel umschlossen. Dieser wird bis zu der vorab festgelegten Tiefe stets entlang der Mitte einer der Achsen des Koordinatensystems in zwei Subvoxel unterteilt, wobei die Reihenfolge der Achsen alternierend verwendet wird. Setzt man für die Anzahl an Unterteilungen fest, dass jede Achse N-mal während dieses Vorgangs geteilt wird, so bildet die unterste Ebene nun ein uniformes Gitter aus $2^N \times 2^N \times 2^N$ Voxeln. Um nun Objekte dieser Hierarchie hinzuzufügen, wird anhand des Mittelpunktes des sie umschließenden Hüllvolumens entschieden, in welches Voxel dieses Gitters sie fallen würden. Um die Beschreibung verständlicher zu machen, werden die Begriffe Objekt und ihr Hüllvolumen im Folgenden äquivalent verwendet. Die Objekte dürfen das Voxel dabei auch überlappen. Formel (4.26) berechnet den Index in x-Richtung, der Index in y- und z-Richtung wird entsprechend

bestimmt:

$$index_x = \left\lfloor 2^N \left(\frac{O_{x_{\text{mid}}} - S_{x_{\text{min}}}}{S_{x_{\text{max}}} - S_{x_{\text{min}}}}\right) \right\rfloor \quad , \tag{4.24}$$

wobei $O_{x_{\text{mid}}}$ der Mittelpunkt des Objektes O auf der x-Achse ist, und $S_{x_{\text{min}}}$ und $S_{x_{\text{max}}}$ jeweils den kleinsten, bzw. größten Wert des Hüllvolumens der Szene entlang der Achse x darstellt. Dies stellt im Prinzip nichts anderes dar, als wenn ein Punkt in ein Uniform Grid einsortiert wird.

Auf diese Art und Weise würden jedoch alle Objekte theoretisch auf der untersten Ebene der BVH landen. Große Objekte würden dadurch allerdings Probleme bereiten, da sie die Hierarchie verstopfen, wie es etwa bei einer Median-Cut BVH geschehen kann. Aus diesem Grund soll es erlaubt sein, dass Objekte auch in innere Knoten der Hierarchie eingefügt werden. Dafür muss zunächst anhand der relativen Ausdehnung des Objektes sein Einfügelevel L bestimmt werden. L gibt hier allerdings den Einfügelevel an, welcher vorläge bei einem balancierten Octree. Für eine binäre BVH wäre er genau $3L$. Aber für die weitere Berechnung wird lediglich L benötigt. Dies geschieht mittels der Formel

$$L = \left\lfloor \log_2 \left(\min\left(\frac{S_x}{O_x}, \frac{S_y}{O_y}, \frac{S_z}{O_z}\right)\right) \right\rfloor \quad , \tag{4.25}$$

wobei O_a die Kantenlänge des Objektes O entlang der Achse a beschreibt, und S_a die Kantenlänge der Szene entlang der Achse a. Sollte $3L$ allerdings größer sein, als der maximale Level der Hierarchie, welcher vorab festgesetzt wurde, so muss der Wert entsprechend auf das mögliche Maximum abgeändert werden. Dies ist voraussichtlich einer der Flaschenhälse dieses Verfahrens in komplexeren Szenen, da dadurch viele Objekte im gleichen Voxel landen könnten.

In Abhängigkeit von L werden nun wieder die gedanklichen Indizes berechnet, diesmal allerdings für ein uniformes Gitter mit den Ausmaßen $2^L \times 2^L \times 2^L$. Somit wird Formel 4.24 zu

$$index_x = \left\lfloor 2^L \left(\frac{O_{x_{\text{mid}}} - S_{x_{\text{min}}}}{S_{x_{\text{max}}} - S_{x_{\text{min}}}}\right) \right\rfloor \quad , \tag{4.26}$$

die Berechnung des Index in y- und z-Richtung verläuft wieder analog.

Da jedoch ein binärer Baum verwendet werden soll, muss aus diesen Indizes noch der Index in der BVH berechnet werden. Das Vorgehen ist dabei in Abbildung 4.24 als Pseudocode wiedergegeben. Aus den berechneten Indizes in x-, y-, z-Richtung wird durch Interleaving der einzelnen Bits der Pfad berechnet, welcher vom Wurzelknoten zum Einfügeknoten führen würde, wobei

eine 0 auf dem Pfad bedeutet, dass zum linken Kind hinabgestiegen werden muss und eine 1, zum rechten. Da der BVH ein ungenutzter Knoten an den Anfang gestellt wurde, so dass der Wurzelknoten den Index 1 besitzt, gilt für jeden Knoten mit Index n, dass sein linkes Kind den Index $2n$ besitzt und sein rechtes Kind den Index $2n + 1$. Mit diesem Wissen kann über den Pfad nun sehr effizient der entsprechende Index in der BVH berechnet werden, indem beginnend beim Wurzelknoten, die Bits des Pfades untersucht werden. Verweist der Pfad auf ein linkes Kind, so wird der aktuelle Index verdoppelt, verweist er auf ein rechtes Kind wird zusätzlich zur Verdoppelung noch der Wert 1 hinzugerechnet. Auf diese Art erhält man sehr schnell den gesuchten Einfügeknoten in der Hierarchie und kann das Objekt dort an die Objektliste anhängen, in unserem Falle ein Vektor der Standard Template Library. Da diese Berechnung unabhängig von den Objekten ist, kann die Abbildung auch in einer Look-Up-Table gespeichert werden, so dass die Berechnungen lediglich einmalig durchgeführt werden müssten. Wurde das Objekt am berechneten Index eingefügt, kann das Hüllvolumen dieses Knotens entsprechend angepasst, sowie das Empty-Flag auf False gesetzt werden.

Bis jetzt wurden zwar die Objekte eingefügt, aber die Hierarchie ist noch inkonsistent. Während der Einfügeprozedur konnten die Knoten, welche Objekte enthalten, bereits in ihrem Hüllvolumen an diese angepasst werden. Die restlichen Hüllvolumen in der BVH sind jedoch noch undefiniert. Da die Hierarchie als 1D-Array abgelegt ist, kann ein effizientes Complete Refit durch Iterierung von hinten nach vorne über dieses vorgenommen werden, wie auch in Abschnitt 4.3.5. Aus der Einfügeprozedur wissen wir jedoch auch, dass für den größten Index einer noch inkonsistenten Box, genannt $index_{\text{refit}}$, gilt

$$index_{\text{refit}} = \lfloor index_{\max}/2 \rfloor \quad ,$$

wobei $index_{\max}$ der größte aus den Objektpositionen berechnete Index in der BVH ist. Da die Hierarchie in Breadth-First-Order abgelegt wurde, kann das Complete Refit von diesem Index starten und die Hüllvolumen, durch merging der Kindboxen und der aktuellen Box, berechnet werden. Sollten beide Kinder, sowie der Knoten selbst, leer sein, wird das Empty-Flag auf True gesetzt, sonst False. Da das Anpassen eines Knotens nahezu konstante Zeit in Anspruch nimmt, ist das Verfahren ungefähr linear zur Anzahl der untersuchten Knoten [Ber97]. Hier zeigt sich, warum eine Überschätzung der Objektzahl und damit der Hierarchietiefe meist keine bedeutende Rolle spielt. Sind weniger Objekte in einer Szene vorhanden, so sind diese auch meist, in Relation zur Szenenausdehnung, größer. Dadurch werden sie höher in der Hierarchie eingefügt und lediglich die oberen Level der BVH müssen

```
geg: index[X] Index in x-Richtung
     index[Y] Index in y-Richtung
     index[Z] Index in z-Richtung
     level    Einfügelevel

unsigned int calcbinaryIndex(unsigned int index[3], int level){
  // Berechne Pfad vom Wurzelknoten zum Einfügeknoten
  unsigned int path = 0;
  for(int t = 0; t < level; t++)
  {
    path <<= 1;
    path |= ((index[Z] >> t) & 1);
    path <<= 1;
    path |= ((index[Y] >> t) & 1);
    path <<= 1;
    path |= ((index[X] >> t) & 1);
  }

  // Berechne den Index in der BVH aus dem Pfad
  unsigned int result = 1;
  for(t = 0; t < 3*level; t++)
  {
    if(path & (1<<t))
    {
      result <<= 1;
      result++;
    }
    else
    {
      result <<= 1;
    }
  }
  return result;
}
```

Abbildung 4.24: Pseudocode für die Berechnung des Index in der BVH

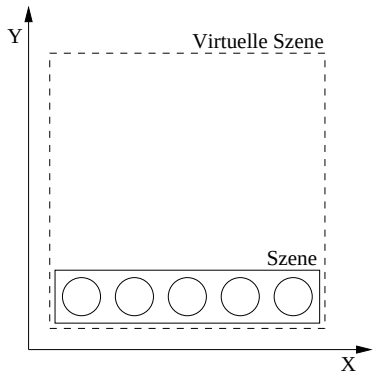


Abbildung 4.25: Beispiel der virtuellen Szene in einer Loose Bounding Volume Hierarchy

aktualisiert werden.

Bei stark nicht uniformer Verteilung der Objekte eignet sich eventuell ein Verfahren nach Brown *et al.* [BSB+01] besser, bei der die Reihenfolge der Anpassungen streng nach einer Priority-Queue abläuft, welches dann jedoch nicht mehr so straight-forward verläuft. In den durchgeführten Tests konnten bis zu 50% Geschwindigkeitseinbußen durch den Einsatz einer PQ verzeichnet werden, weswegen darauf verzichtet wurde.

Durch das Einfügen der Objekte gemäß ihrer Größe wird zudem auch eine Maximalausdehnung für jeden Knoten der Hierarchie garantiert, was im Umkehrschluß zu einer paarweisen Maximalüberlappung von 50% zwischen benachbarten Knoten eines Levels und damit zu einer guten räumlichen Aufteilung der Hierarchie führt.

Bei der Umsetzung dieses Verfahrens, dürften jedoch bei sehr flachen Szenen viele Objekte in sehr niedrigen Leveln landen, weil als Einfügelevel die relativ längste Achse des Objektes als Grundlage der Berechnung dient. Um die Methode jedoch auch für solche Szenarien praktikabel zu machen, wird die Szene künstlich vergrößert. D.h. es wird eine virtuelle Szene in Kubusform berechnet, welche die Originalszene umschließt, deren Seitenlängen jedoch ihrer längsten Ausdehnung entsprechen, wie in Abbildung 4.25 für den zweidimensionalen Fall dargestellt. Dadurch werden die Objekte tiefer in der Hierarchie eingefügt, was wiederum eine bessere Unterteilung bedeutet, auch wenn dadurch mehr Knoten traversiert werden müssen.

Dem aufmerksamen Leser wird nicht entgangen sein, dass die Hierarchie Knoten erlaubt, die lediglich ein Kind besitzen. Dies kann geschehen, wenn sich ein kleines Objekt in einem größeren, leeren Raum befindet. Für die Tra-

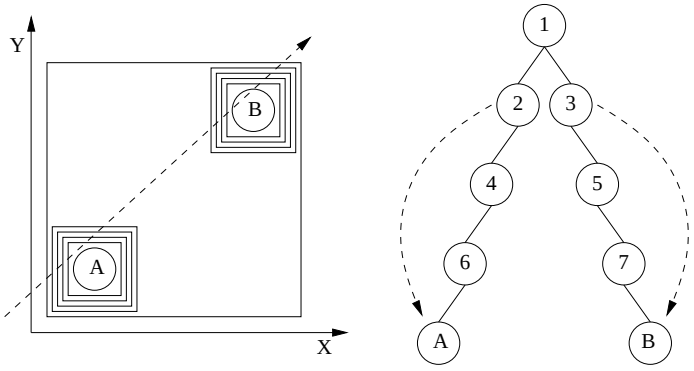


Abbildung 4.26: Anwendung des SkipIndex

versierung ist dies jedoch nachteilig, da so eventuell mehrmals Hüllvolumen mit identischen Ausmaßen auf Schnitte mit einem Strahl getestet werden müssen. Leider ist es nicht ohne größeren Aufwand möglich die Objekte soweit in der Hierarchie heraufzureichen, dass diese Fälle ausgemerzt werden, da der Aufwand dafür deutlich größer würde als der Overhead während der Traversierung. Stattdessen kann man jedem Knoten einen *SkipIndex* mitgeben. Dieser verweist normalerweise auf den eigenen Index. Sollte jedoch nur ein Kind vorhanden sein, verweist er auf dessen SkipIndex. Dadurch können auch mehrere Ebenen in einem Schritt übersprungen werden. Die Aktualisierung der SkipIndizes kann sehr simpel während des Refittings durchgeführt werden, indem bei einem untersuchten Knoten mit nur einem nichtleeren Kindknoten als SkipIndex der SkipIndex des Kindes gesetzt wird, ansonsten der eigene Index. Abbildung 4.26 verdeutlicht dies noch einmal für den zweidimensionalen Fall. Die dargestellte Szene enthält zwei relativ kleine Objekte. Dadurch ergäbe sich der Baum aus der Abbildung zur Traversierung, leere Knoten wurden nicht dargestellt. Um mittels eines Strahles eines der Objekte zu treffen, müsste eigentlich der gesamte Pfad bis zum Blattknoten traversiert und die entsprechenden Knoten getestet werden. Da jedoch, oberhalb der Objektknoten, die Hüllvolumen zunächst alle die gleichen Ausmaße haben, wäre dies absolut unnötig, da bereits bekannt ist, dass der Strahl, schneidet er einmal den Knoten 2 oder 3, alle darunter liegenden ebenfalls schneiden wird. Der als Pfeil eingezeichnete SkipIndex sorgt jedoch dafür, dass diese Knoten effizient übersprungen werden und die Traversierung am entsprechenden Objektknoten fortgesetzt wird.

In Abbildung 4.27 ist noch einmal ein Beispiel für jeden der Schritte zur Erstellung der Loose Bounding Volume Hierarchy gegeben. In Abb. 4.27(a) werden die Objekte einfach anhand ihres Mittelpunktes in den entsprechenden Voxel auf der untersten Ebene der BVH eingefügt. Die entstehende Hierar-

chie ist als Baum rechts daneben zu sehen, leere Knoten wurden nicht dargestellt. Eine erste Verbesserung findet statt, indem die Objekte entsprechend ihrer Größe eingeordnet werden. Dadurch wird Objekt A in die Objektliste des Wurzelknotens eingefügt und kann früher getestet werden, dadurch werden untere Ebenen nicht mehr verstopft. Eine letzte Verbesserung wird in Abb. 4.27(c) erreicht, indem ein SkipIndex verwendet wird, um Hüllvolumen gleichen Ausmaßes nicht öfters auf Schnitte mit demselben Strahl testen zu müssen.

Rekonstruktions-Phase

Bedingt durch die schnelle Art des Refittings und die direkte Berechnung der Indizes für die Objekte lässt sich auch die Rekonstruktion sehr direkt umsetzen. So werden zunächst, unabhängig davon, ob sie sich bewegt haben oder nicht, sämtliche Objekte aus der Hierarchie entfernt. Dazu wurde beim Einfügen der Objekte ihr Index in einem Array gespeichert, welches nun abgelaufen wird und die entsprechenden Knoten leert, sowie das Empty-Flag setzt. Danach wird die Hierarchie wieder, wie im vorherigen Abschnitt erläutert mit einer kleinen Änderung aufgebaut. Durch das Leeren der Knoten, wurden diese zwar als leer gekennzeichnet, die darüber liegenden Knoten jedoch nicht, bei ungünstiger Bewegung der Objekte, könnte es so geschehen, dass fälschlicherweise Teile der Hierarchie noch als nicht leer bezeichnet sind und somit während der Ray Tracing-Phase traversiert werden. Dies kann natürlich zu einem nicht zu unterschätzenden Overhead führen. Eine kleine Abänderung des Refittings sorgt allerdings auch hier für Abhilfe. So beginnt dieses nicht bei dem maximalen berechneten Objektindex beim Einfügevorgang, sondern dieser Wert wird noch verglichen mit dem maximalen Objektindex vom letzten Frame. Der größere von beiden, dient als Startindex für das Refitting, bzw. die Hälfte davon. Da die Blattknoten bereits angepasst sind, genügt es beim Vaterknoten zu beginnen. So werden leere Teilbäume erfolgreich entfernt.

In Abbildung 4.28 sind die wichtigsten Schritte des Rekonstruktions-Vorgangs noch einmal als Pseudocode wiedergegeben.

Traversierung

Die Traversierung verläuft nahezu nach dem üblichen Vorgehen nach Kay/Kajiya [KK86]. Der Unterschied besteht jedoch darin, dass Knoten mehrere Objekte enthalten können, welche allesamt getestet werden müssen, sollte ein

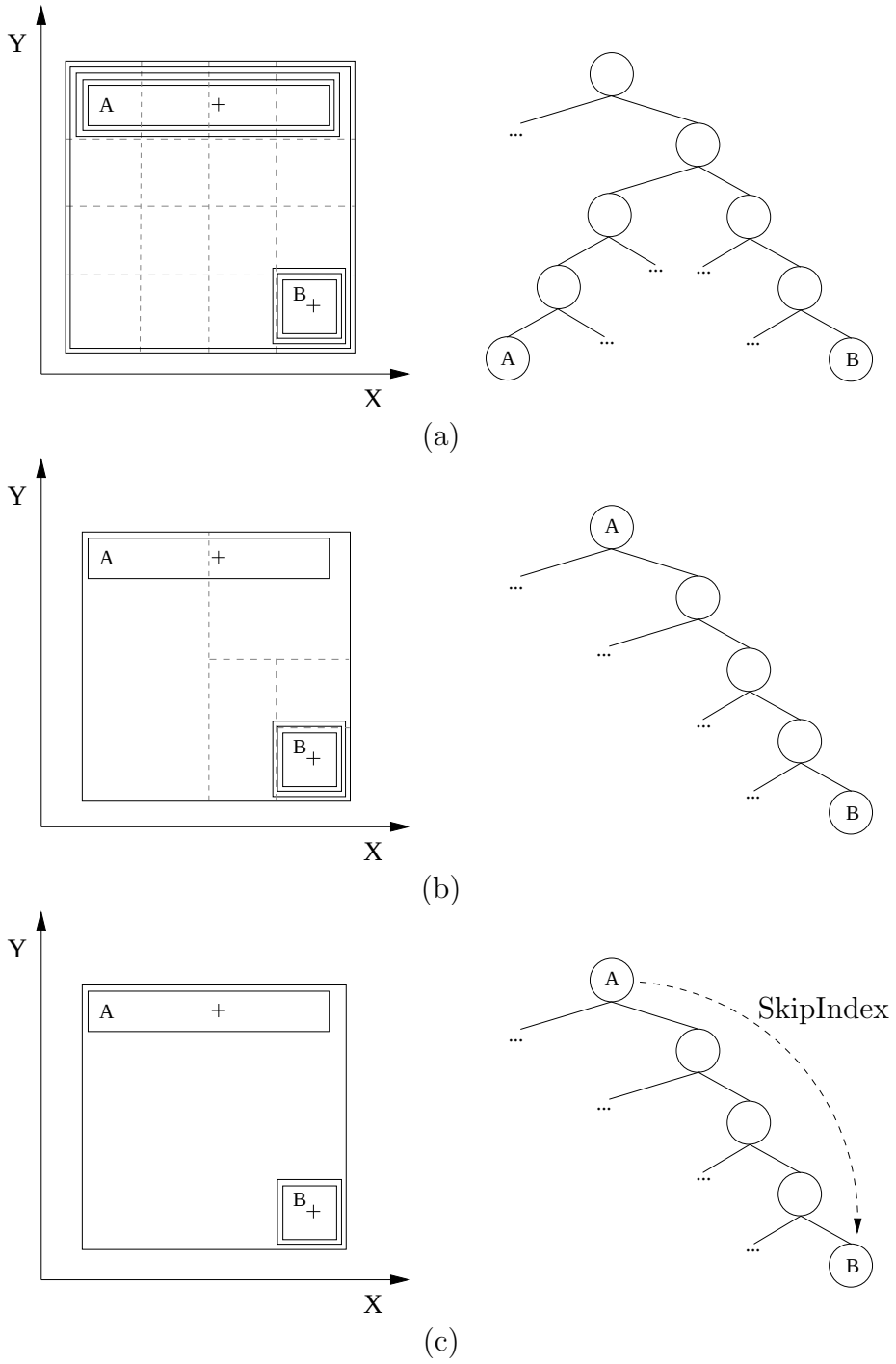


Abbildung 4.27: Verschiedene Konstruktionsmöglichkeiten einer Loose BVH. (a) Objekte werden in die unterste Ebene eingefügt. (b) Einfügen gemäß der Größe. (c) Zusätzlicher Einsatz eines SkipIndex

```
void updateLooseBVH(){
  // Aktualisierung der Objektpositionen
  updateObjects();

  // Leeren der gesamten Hierarchy
  emptyHierarchy();

  // Anpassen der Objekthüllvolumen
  for all objects o
  {
    Berechne das Hüllvolumen von o;
  }

  // Neuaufbau der Hierarchie
  // Neubestimmung der Szenenausdehnung
  expandRootToEncloseScene();
  calcSceneExtends(); // Virtuelle Szene

  for all objects o
  {
    // Berechnung des Einfügeindex
    index = calcIndex();
    // Einfügen in Objektliste des Knotens
    BVH[index].objectlist.push_back(o);
  }

  refitHierarchy();

}
```

Abbildung 4.28: Pseudocode der Rekonstruktions-Phase des Loose Bounding Volume Hierarchy Verfahrens

entsprechender Knoten von einem Strahl getroffen werden und zwar zunächst auf ihre direkten Hüllvolumen und letzlich die Objekte selbst.

Die Kinder eines Knotens werden zudem mittels ihres Index berechnet. Dies erfordert lediglich eine Binäroperation, $index << 1$, für das linke Kind, sowie ein zusätzliches Inkrement für das rechte. Bevor der Strahltest jedoch mit dem Knoten durchgeführt wird, findet ein Test statt, ob der Knoten Teil der aktuellen Hierarchie oder leer ist. In letzterem Falle darf sofort angenommen werden, dass der Strahl den Knoten verfehlt. Wird ein Kind getroffen, wird jedoch nicht unbedingt dieses selbst, sondern sein SkipIndex in die Priority-Queue eingefügt, um unnötige Strahltests zu verhindern.

Eine Tiefensuche wäre theoretisch ebenso möglich, würde allerdings etwas Mehraufwand verlangen. Werden die SkipPointer eines jeden Knotens vorab berechnet, so könnte es passieren, dass der Strahl durch eine Reihe von leeren Knoten traversiert werden muss, bevor er über die SkipPointer den nächsten aktiven Teil der BVH erreicht. Mittels einer veränderten Look-Up-Table für die Indexberechnungen könnte man die Hierarchie auch in Depth-First Order ablegen, um eine verbesserte Cache-Ausnutzung zu erhalten.

Zusammenfassung

In diesem Abschnitt wurde eine hybride BVH vorgestellt, welche die Vorteile einer BVH, mit denen einer rein spatialen Datenstruktur verbindet. Es wurde gezeigt, wie man mittels einer vorab angelegten Hierarchie und leerer Knoten eine Komplexität von lediglich $O(n + m)$ erreicht. Der Nachteil einer Datenstruktur, wie den Loose Octrees von Ulrich [Ulr00], bei denen die Hüllvolumen stark über die Objektgrenzen hinausragen können, wurde durch ein effizientes Refitting behoben. Unnötige Traversierung leerer Bereiche in der Szene, wie es bei Reinhard *et al.* der Fall ist [RSH00], da sie eine Traversierung ähnlich einem Uniform Grid durchführen, wird durch die Markierung leerer Knoten vermieden. Große Objekte, welche dadurch auch eine höhere Trefferwahrscheinlichkeit aufweisen, werden hoch in der Hierarchie gehalten und verstopfen somit nicht die unteren Ebenen. Das frühzeitige Testen großer Objekte kann vor allem aber auch von Vorteil sein, sollte man sich für eine SIMD-optimierte Tiefensuche entscheiden, um frühzeitig einen Objektschnittpunkt zu finden. Weit voneinander entfernte Objekte werden wie bei Goldsmith und Salmon [GS87] in einem eigenen Knoten gehalten und können somit frühzeitig geculled werden. Nahe beieinanderliegende Objekte, bei denen, bedingt durch voraussichtlich starke Überlappung, eine Hierarchie keine wirklichen Vorteile mehr bringen würde, werden im selben Knoten gehalten. Da die Kindknoten in unserem Falle hintereinander abgelegt sind, ist

eine relativ gute Cache-Ausnutzung für eine Traversierung nach Kay/Kajiya gegeben.

Natürlich bringt dieses Verfahren auch einige Nachteile mit sich. Der Speicherverbrauch etwa ist für nicht-uniforme Objektverteilungen, verglichen mit anderen Methoden, immens hoch. Die Traversierung kann nicht ganz optimal durchgeführt werden, da in jedem Schritt getestet werden muss, ob der Knoten leer ist oder nicht. Um auch eine Szenenausdehnung zu erlauben, muss die Hierarchie in jedem Frame komplett neu aufgebaut werden, da sich dadurch auch der Index statischer Objekte verändern kann. Da die Hierarchie vorab mit einer maximalen Tiefe angelegt wurde, leidet das Verfahren bei sehr komplexen Szenen unter dem „Teapot in the stadium“ Problem, wenn zuviele Objekte in denselben Knoten fallen. Abhilfe könnte eine Trennung von statischen und dynamischen Szeneobjekten bringen, falls das entsprechende Vorwissen vorhanden ist, sowie die Ausnutzung lokaler Koordinatensysteme, wie sie in Abschnitt 4.4 noch beschrieben werden.

4.4 Lokale Koordinatensysteme und hierarchische Animation

Bisher wurden verschiedene Ansätze präsentiert, die einen kompletten Neuaufbau der Beschleunigungsstruktur weitestgehend zu verhindern suchen, oder diesen möglichst schnell durchführen. Da jedoch die meisten Szenenbeschreibungen in Form eines Szenegraphens vorliegen, welcher die Positionierung, Ausrichtung und Abhängigkeiten der Objekte untereinander vorgibt, stellt sich die Frage, ob man sich diesen zunutze machen kann, um den Rekonstruktions-Vorgang zu beschleunigen.

Ein für statische Szenen ausgelegtes Ray Tracing System würde in der Regel jedes Primitiv aus diesem Szenegraphen in Weltkoordinaten umrechnen und darauf aufbauend eine komplett neue Beschleunigungsstruktur erstellen. Dieses ist jedoch absolut unpraktikabel für ein schnelles Ray Tracing dynamischer Szenen, wie an einem einfachen Beispiel ersichtlich werden dürfte. Man stelle sich ein komplexes Objekt aus Hundertausenden oder sogar Millionen von Dreiecken vor, welches sich durch den Raum bewegt. Jedes dieser Dreiecke für jedes Bild in Weltkoordinaten umzurechnen und darauf die Beschleunigungsstruktur aufzubauen, würde jedes interaktive Ray Tracing System in die Knie zwingen. Auch die in dieser Arbeit vorgestellten Methoden bieten selbstverständlich ab einer gewissen Anzahl an bewegten Objekten keine zufriedenstellende Performanz mehr. Aus diesem Grunde ist es wichtig

soviel von der Beschleunigungsstruktur wie möglich zwischen den einzelnen Frames zu erhalten.

Dies ist jedoch nicht so ohne weiteres möglich. Es ist ersichtlich, dass sich viele Bereiche einer Szene gleichförmig bewegen, nämlich genau die, welche demselben Transformationsknoten im Szenegraph unterliegen. In der Regel weisen diese Bereiche zudem eine gute Lokalität auf, d.h. sie überschneiden sich selten mit anderen Bereichen der Szene, weswegen die Qualität einer BVH nicht bedeutend sinken sollte, fasst man die Primitive unterhalb solcher Transformationsknoten in einem eigenen Teilbaum der BVH zusammen. Dadurch wäre allerdings noch nicht viel gewonnen, da auch weiterhin jede Transformation eine Anpassung aller Knoten des entsprechenden Teilbaumes verlangen würde. Anders ist dies, arbeitet man mit lokalen Koordinatensystemen. Für das Ergebnis ist es irrelevant, ob ein Objekt O transformiert wird mit einer 4×4 affinen Transformationsmatrix $\mathbf{M}$ und auf einen Strahl $\mathbf{R}(t)$ getestet wird, oder ob ein Strahl $\mathbf{R}(t)$ mittels der inversen Transformationsmatrix $\mathbf{M}^{-1}$ in das lokale Koordinatensystem von O überführt wird und dort auf Schnittpunkte getestet wird. Dies ist in Abbildung 4.29 dargestellt. Der parametrisch durch seinen Ursprung $\mathbf{o}$ und seine Richtung $\vec{\mathbf{d}}$ definierte Strahl

$$\mathbf{R}(t) = \mathbf{o} + t\vec{\mathbf{d}}$$

soll gegen das Objekt O getestet werden, was mit $\mathbf{R}(t) \cap O$ ausgedrückt werden soll. Wird dieses Objekt nun mittels einer Transformationsmatrix $\mathbf{M_O}$ transformiert, so kann entweder

$$\mathbf{R}(t) \cap \mathbf{M_O} O \quad \text{oder} \quad \mathbf{M_O}^{-1} \mathbf{R}(t) \cap O$$

getestet werden. Der Strahlparameter t wird in beiden Fällen das gleiche Ergebnis liefern.

Bevor die Transformation für einen Strahl mathematisch beschrieben wird, muss sie zunächst einmal für Punkte und Richtungsvektoren definiert sein. Der Unterschied zwischen beiden besteht darin, dass Letzterer translationsinvariant ist, d.h. eine Translation verändert seinen Wert nicht. Dies lässt sich sehr einfach mittels vorherigem Festsetzen der so genannten homogenen Koordinate realisieren. Die Transformation eines Punktes $\mathbf{p}$ wird dann wie in Gleichung (4.27) durchgeführt, mit der homogenen Koordinate gleich 1, während bei einer Vektortransformation (siehe Gleichung (4.28)) die homogene Koordinate des Vektors $\vec{\mathbf{v}}$ auf 0 gesetzt wird.

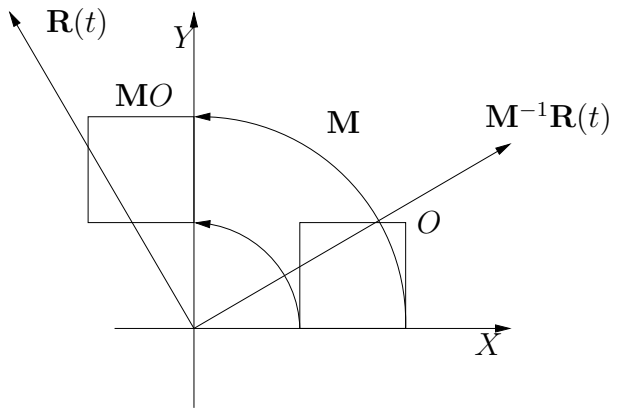


Abbildung 4.29: Transformation eines Strahles in ein lokales Koordinatensystem

$$\mathbf{Mp} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} m_{00}p_x + m_{01}p_y + m_{02}p_z + m_{03} \\ m_{10}p_x + m_{11}p_y + m_{12}p_z + m_{13} \\ m_{20}p_x + m_{21}p_y + m_{22}p_z + m_{23} \\ m_{30}p_x + m_{31}p_y + m_{32}p_z + m_{33} \end{pmatrix} , \tag{4.27}$$

$$\mathbf{M}\vec{\mathbf{v}} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \begin{pmatrix} m_{00}v_x + m_{01}v_y + m_{02}v_z \\ m_{10}v_x + m_{11}v_y + m_{12}v_z \\ m_{20}v_x + m_{21}v_y + m_{22}v_z \\ m_{30}v_x + m_{31}v_y + m_{32}v_z \end{pmatrix} , \tag{4.28}$$

wobei die unterste Zeile der Matrizen in der Praxis meist die Form $(0, 0, 0, 1)$ haben wird.

Eine Ausnahme bildet die Transformation einer Oberflächennormalen $\vec{\mathbf{n}}$, welche zur Beleuchtung an einem Schnittpunkt benötigt wird. Diese werden ähnlich einem Vektor, transformiert, allerdings nicht mittels $\mathbf{M}\vec{\mathbf{n}}$ sondern mittels der transponierten Inversen $(\mathbf{M}^{-1})^\top\vec{\mathbf{n}}$. Gehen wir von rein affinen Transformationen aus, so ist die Berechnung dieser relativ simpel, da für eine Translation $\mathbf{T}$ um den Vektor $\vec{\mathbf{t}} = (t_x, t_y, t_z)^\top$, für eine Rotation $\mathbf{R}$ um den Winkel α und für eine Skalierung $\mathbf{S}$ um die Werte s_x, s_y, s_z entlang der entsprechenden Achsen gilt:

$$\mathbf{T}^{-1}(t_x, t_y, t_z) = \mathbf{T}(-t_x, -t_y, -t_z) \quad (4.29)$$

$$\mathbf{R}^{-1}(\alpha) = \mathbf{R}(-\alpha) \quad (4.30)$$

$$\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(\frac{1}{s_x}, \frac{1}{s_y}, \frac{1}{s_z}) \quad (4.31)$$

In den meisten Fällen kann die inverse Matrix einfach durch Multiplikation der drei inversen Transformationsmatrizen berechnet werden, wenn vom Programm eine entsprechende Reihenfolge vorgesehen ist. Sind diese Matrizen nicht gegeben, erweist sich die Berechnung als komplizierter:

$$\mathbf{M}^{-1} = \frac{1}{\det(\mathbf{M})}\mathbf{M}^* \quad , \quad (4.32)$$

det($\mathbf{M}$) bezeichnet jeweils die Determinante der Matrix $\mathbf{M}$, $\mathbf{M}^*$ ist die adjungierte Matrix zu $\mathbf{M}$, mit

$$\mathbf{M}^* = (m^*_{ij})$$
$$m^*_{ij} = (-1)^{i+j} \det(\mathbf{M}_{ij}). \quad (4.33)$$

$\mathbf{M}_{ij}$ entspricht der Matrix $\mathbf{M}$ ohne die i-te Zeile und j-te Spalte.

Ein Strahl $\mathbf{R}(t) = \mathbf{o} + t\vec{\mathbf{d}}$ lässt sich dann wie folgt in ein lokales Koordinatensystem transformieren:

$$\mathbf{M}^{-1}\mathbf{R}(t) = \mathbf{M}^{-1}\mathbf{o} + t\mathbf{M}^{-1}\vec{\mathbf{d}} \quad (4.34)$$

Das Ganze lässt sich erweitern auf mehrere hintereinander geschaltete Transformationen. Gleichung (4.35) gibt dabei die Transformation eines Objekte O mittels der Transformationsmatrizen $\mathbf{M_0}$ bis $\mathbf{M_n}$ wieder und Gleichung (4.36) die entsprechende Transformation des Strahles $\mathbf{R}(t)$.

$$(\mathbf{M_0} \cdot \ldots (\mathbf{M_n} O)) = (\mathbf{M_0} \cdot \ldots \cdot \mathbf{M_n})O = \mathbf{M}O \quad (4.35)$$

$$\mathbf{M}^{-1}\mathbf{R}(t) = (\mathbf{M_0} \cdot \ldots \cdot \mathbf{M_n})^{-1}\mathbf{R}(t) = \mathbf{M_n}^{-1} \cdot \ldots \cdot \mathbf{M_0}^{-1}\mathbf{R}(t) \quad (4.36)$$

Diese Idee kann nun erweitert werden, indem nicht nur ein einzelnes Primitiv transformiert wird, sondern komplette Teilbereiche der Hierarchie. D.h. es muss eine neue Klasse, genannt Instanz, eingeführt werden. Diese bietet ein identisches Interface, wie ein normales Primitiv, kann also vom Ray Tracer gleich behandelt werden. Sie speichert jedoch lediglich eine Transformationsmatrix, ein *äußeres* Hüllvolumen, welches das transformierte Objekt

in Weltkoordinaten, bzw. in den Koordinaten des entsprechenden Raumes in denen sich das Objekt befindet, darstellt, sowie einen Zeiger auf eine BVH, in welcher das Objekt in seinem eigenen, lokalen Koordinatensystem gespeichert ist. Das äußere Hüllvolumen wird benötigt, um einen Strahl gegen das Objekt testen zu können, bevor er transformiert wird. Das Hüllvolumen wird dabei wie folgt berechnet. Die Transformationsmatrix des Objektes wird auf alle acht Punkte angewendet, welche den Wurzelknoten des Objektes aufspannen. Aus diesen acht Punkten lässt sich dann die entsprechende AABB berechnen, welche diese umschließt. Je nachdem wie das Objekt selbst modelliert und welche Transformationen angewandt wurden, kann diese deutlich größer sein, als wenn das Objekt in Weltkoordinaten umgerechnet worden wäre. Schneidet ein Strahl diese äußere Hülle, wird er in das lokale Koordinatensystem transformiert und eine Traversierung der lokalen BVH begonnen.

In Ray Tracern für statische Szenen wird dieses Konzept häufig verwendet, um zum einen eine einfache Berechnung komplexerer Primitive zu ermöglichen, z.B. von Ellipsoiden, oder um Instanzen von ganzen Objekten zu erzeugen, die mehrfach in der Szene auftauchen, um Speicherplatz zu sparen [SM03]. In Ray Tracern für dynamische Szenen, wird dieses Verfahren zur Verringerung der Rekonstruktions-Zeit geschätzt [WBS03] [LAM01b].

Als Methode zur Erstellung der lokalen Beschleunigungsstrukturen wurde das Verfahren nach Goldsmith und Salmon [GS87] ausgewählt, da es für die meisten Objekte bessere Ergebnisse liefert als andere Methoden. Die Wahl mag zunächst verwundern, da hierarchische Animationen ebenfalls einen Neuaufbau dieser Strukturen erfordern könnten, doch mit den im folgenden vorgestellten Veränderungen ist dies nicht mehr notwendig.

Die einfachste Variante einen Szenegraphen zu übernehmen wäre also, für jeden Transformationsknoten, eine Instanz zu erzeugen und den Strahl jeweils entsprechend zu transformieren. Ein Beispiel für einen Szenegraphen und die entsprechende BVH ist in Abbildung 4.32 (a) und (b) auf Seite 108 gegeben.

Diese Darstellung würde jedoch zu keinem guten Ergebnis führen. Da die Transformationsmatrizen direkt gespeichert werden, muss jedes mal, wenn ein Strahl das äußere Hüllvolumen schneidet, die inverse Transformationsmatrix berechnet werden. Das bedeutet einen erheblichen Mehraufwand. Deswegen führt ein erster Schritt zur Verbesserung dahin, dass in jeder Rekonstruktions-Phase die aktuelle Transformationsmatrix verwendet wird, um das äußere Hüllvolumen zu berechnen, danach jedoch einmalig invertiert wird und in dieser Form in den Objekten gespeichert wird (Abb. 4.32 (c)).

Zudem birgt diese Darstellung auch noch andere Schwierigkeiten. Einige Objekte unterliegen nicht nur einer einzelnen sondern mehreren Transformati-

on. Dies kann insbesondere bei hierarchisch animierten Objekten der Fall sein. Ein klassisches Beispiel wäre der Roboterarm aus Abbildung 4.31(a), die Hand bewegt sich dabei relativ zum Unterarm, welcher sich relativ zum Oberarm bewegt. Für die Traversierung dieser Struktur, müsste nicht nur der Strahl mehrfach transformiert werden, sondern es ergeben sich auch Schwierigkeiten für die Normalenberechnung. Da für diese die komplette Transformation benötigt wird, der der Strahl bis zum Schnittpunkt unterworfen wurde, müssten die entsprechenden Transformationen zwischengespeichert, und, sobald ein Schnittpunkt gefunden wurde, für die spätere Normalentransformation aufmultipliziert werden. Und dies für jeden Strahl.

Eine weitere Verbesserung läge also darin, anstelle der inversen Matrizen die aufmultiplizierten inversen Matrizen, welche einen Strahl aus Weltkoordinaten in das entsprechende lokale Koordinatensystem transformieren, in den Instanzen abzuspeichern. Dies verlangt eine zusätzliche Matrixmultiplikation pro Objekt, spart aber dafür sämtliche Matrixmultiplikationen, die eventuell zur Berechnung der Matrix für die Normalentransformation nötig wären (siehe auch Abb. 4.32 (d)).

Matrixmultiplikationen sind teuer und sollten deswegen so weit es geht verhindert werden. Deswegen wird in einem letzten Schritt die n-Level BVH, welche momentan noch vorliegt, auf eine 2-Level BVH reduziert wird. D.h. jede Instanz, welche erzeugt wird, darf keine weiteren Instanzen enthalten. Um dennoch hierarchische Animationen zu ermöglichen, wird der Szenegraph und die darin enthaltenen Transformationsmatrizen von seinen Objekten getrennt. Jedes Objekt, bzw. Instanz, verwaltet dann einen zusätzlichen Zeiger, welcher auf den entsprechenden Eintrag im Szenegraphen verweist, um an seine aktuelle Transformationsmatrix zu gelangen. Die Transformationsmatrizen des Szenegraphens selbst werden in ein 1D Array umkopiert, wobei gelten muss, dass jede Transformationsmatrix zum einen, falls vorhanden, den Index der Transformation kennt, von welcher sie selbst abhängig ist (im folgenden *Vatertransformation* genannt). Zum anderen muss ihr Index im Array größer sein als der der Vatertransformation (siehe Abb. 4.30). Diese Darstellung des Szenegraphens ist für unser System so gewählt worden, weil sie lediglich die zur Animation noch benötigten Teile des Szenegraphens enthält, sämtliche Geometrie befindet sich in der BVH, Materialien werden separat gespeichert.

Die Rekonstruktions-Phase wird nun zweigeteilt. In einem ersten Schritt, werden die Transformationsmatrizen Top-Down aktualisiert. Dies kann effizient in der 1D Array Repräsentation geschehen, indem einmalig von Anfang bis Ende über das Array iteriert und für jeden Eintrag die aktuelle, lokale Transformationsmatrix bestimmt wird, welche das Objekt selbst animiert. Diese

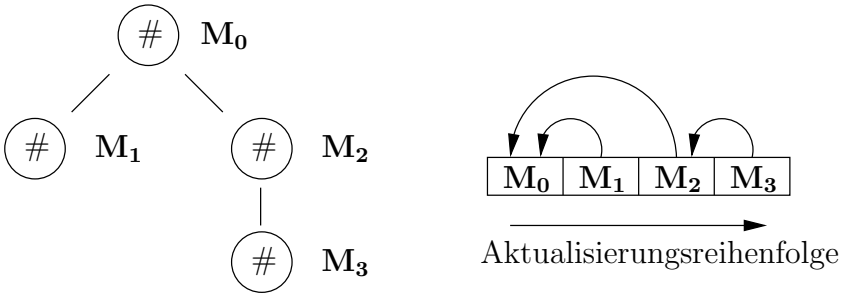


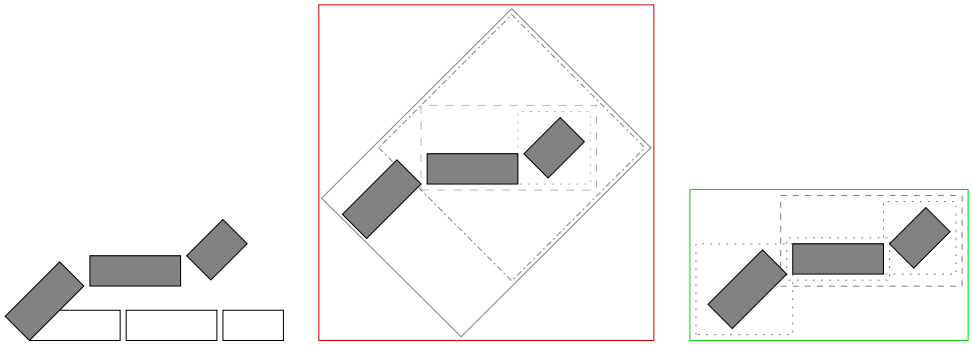
Abbildung 4.30: Baumstruktur eines Szenegraphens und Darstellung in einem 1D Array. Es werden lediglich Transformationsknoten betrachtet. Die Pfeile symbolisieren den Verweis auf die Vatertransformation.

wird mit der Vatertransformationsmatrix multipliziert, um Abhängigkeiten in hierarchischen Animationen gerecht zu werden. In einem zweiten Schritt, werden die Instanzen aktualisiert, indem sie zunächst ihre aktuelle Transformationsmatrix **M** abfragen. Multiplikation von **M** mit den Eckpunkten des Wurzelknotens der lokalen BVH liefert die Ausdehnung in Weltkoordinaten. Um diese transformierten Punkte wird eine AABB gelegt, welche das äußere Hüllvolumen bildet. **M** wird invertiert und für die Strahltraversierung in der Instanz abgespeichert.

Dies ist sehr effizient, da für jedes animierte Objekt innerhalb einer hierarchischen Animation, lediglich eine zusätzliche Matrixmultiplikation notwendig ist, nämlich die lokale Transformationsmatrix mit der Vatermatrix. Andererseits bedeutet es einen erheblichen Vorteil bei der Traversierung. Erstens muss, wie bereits im letzten Schritt, kein Matrizenstack für die spätere Normalenberechnung mitgeführt werden und zweitens wird ein Strahl nur noch transformiert, wenn es absolut notwendig ist, also nicht mehr für jeden Level der hierarchischen Animation, sondern lediglich für den letzten.

Einer der größten Vorteile liegt jedoch darin, dass ein „Anwachsen“ der AABB eines mehrfach transformierten Objektes verhindert wird. Alleine durch eine einfache Rotation kann die Oberfläche der AABB eines Objektes um den Faktor $3\sqrt{3}$ oder sogar mehr vergrößert werden, verglichen zur ursprünglichen Oberfläche[2]. Bei mehreren hintereinandergeschalteten Transformationen kann dieser Effekt das Hüllvolumen exponentiell anwachsen lassen. Dies ist in Abbildung 4.31 dargestellt. Der ursprünglich gerade Arm wird an jedem Gelenk um jeweils 45, bzw. -45, Grad rotiert (Abb. 4.31(a)).

[2]Der Wert von $3\sqrt{3}$ ergibt sich durch Rotation eines Einheitswürfels, dessen Mittelpunkt sich im Ursprung befindet, um jeweils 45 Grad um die x-Achse und y-Achse. Bei anderen Quadern kann der Faktor noch deutlich höher liegen.



(a) (b) (c)

Abbildung 4.31: (a)Beispiel für eine hierarchische Animation und die entsprechenden transformierten Primitive. (b) Darstellung bei Verwendung einer n-Level BVH. (c) Darstellung bei Verwendung einer 2-Level BVH.

Die entsprechenden transformierten Hüllvolumen einer n-Level BVH sind in Abbildung 4.31(b) dargestellt. Die aus den einzelnen Transformationen entstandenen Hüllvolumen sind grau dargestellt. Der 2-Level BVH Ansatz ist in Abbildung 4.31(c) zu sehen. Die Verbesserung in (c) gegenüber (b) dürfte deutlich sichtbar sein. Abbildung 4.32 (e) zeigt den 2-Level BVH Ansatz für den Beispielszenegraphen.

Ein weiterer Vorteil ist zudem die verbesserte Traversierungsmöglichkeit zwischen den Instanzen. Ein Schnittpunkttest mit einem Intanzenobjekt ist wie eine Black-Box. D.h. der Ray Tracer fragt nach einem Schnittpunkt und bekommt das Ergebnis zurück geliefert. Bis dahin muss die restliche Traversierung der Szene aussetzen. In einzelnen Fällen kann es aber sein, dass sich auch dynamische Objekte überlappen. Insbesondere bei einer Traversierung nach Kay/Kajiya [KK86] kann es dann von Vorteil sein, lediglich eine 2-Level BVH vorliegen zu haben.

In diesem Abschnitt wurde ein wichtiger Schritt für die Beschleunigung der Rekonstruktions-Phase eines Ray Tracers für dynamische Szenen vorgestellt. Statt jedes Primitiv einzeln zu aktualisieren, können auch komplette Teilbäume, welche einzelne Objekte repräsentieren, aktualisiert werden, indem BVHs in ihrem lokalen Koordinatensystem verwendet werden, in welche der Strahl transformiert wird. Dies reduziert den Aufwand für diese Objekte auf eine simple Aktualisierung ihres äußeren Hüllvolumens. Hierarchische Strukturen werden soweit wie möglich heruntergebrochen, um ein übersteigertes „Anwachsen“ der Hüllvolumen zu verhindern.

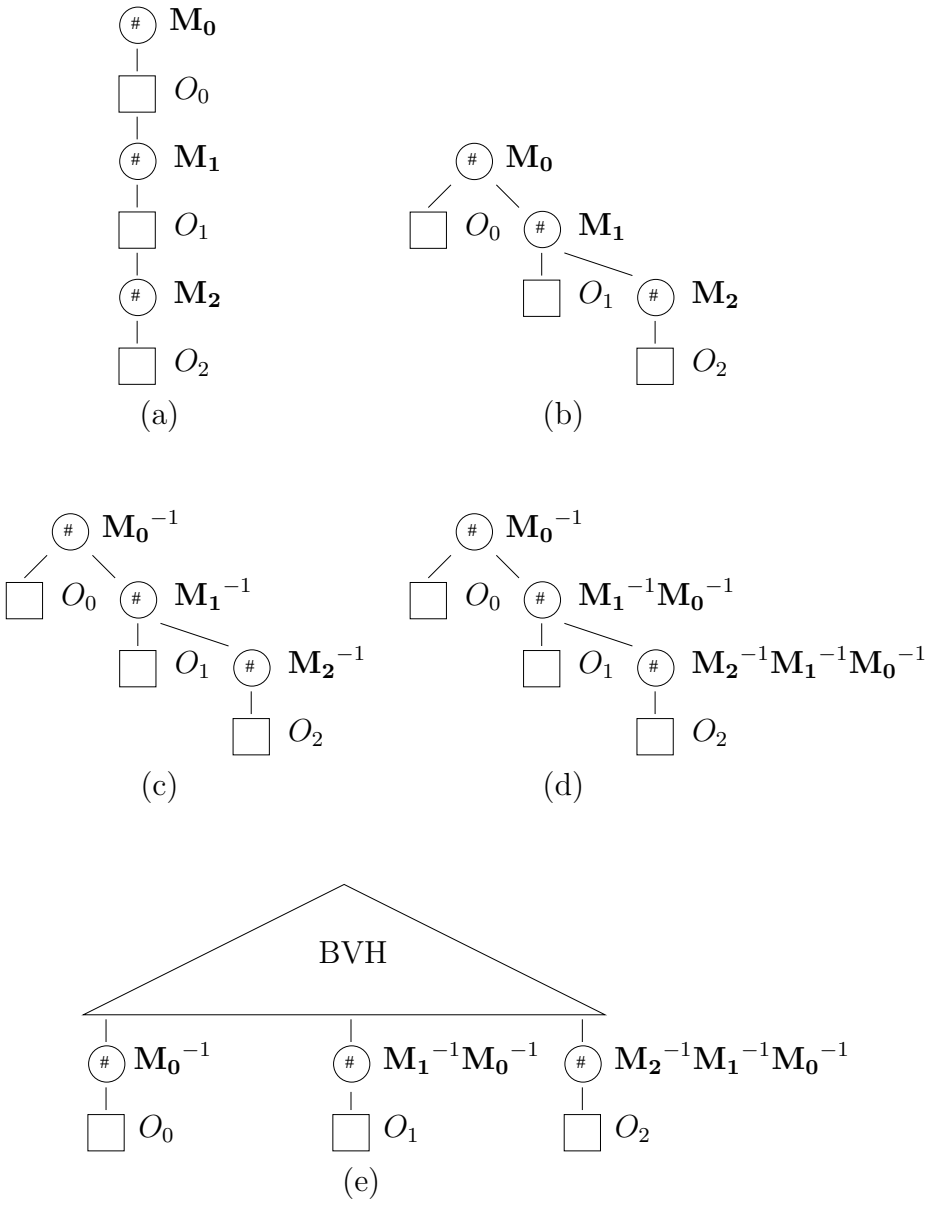


Abbildung 4.32: (a) Beispiel eines einfachen Szenegraphens. (b) Mögliche Darstellung des Szenegraphens in einer BVH. (c) Mögliche Darstellung des Szenegraphens in einer BVH mittels Speicherung der inversen Matrizen an den Knoten. (d) Verbesserung durch Aufmultiplizieren der Matrizen (e) 2-Level BVH. $\mathbf{M_0}$ bis $\mathbf{M_2}$ sind Transformationsknoten, O_0 bis O_2 Geometrieknoten, welche in den jeweiligen BVHs zu einem Instanzenobjekt mit eigener Beschleunigungsstruktur zusammengefasst werden.

Kapitel 5

Ray-Slope Schnitttest

Im folgenden Kapitel wird eine neue Schnittmethode zwischen einem Strahl und einer achsenparallelen Box (AABB) vorgestellt, welche sich die Steigung des Strahles zunutze macht, um Großteile der benötigten Informationen vorab zu berechnen, was vor allem bei Tests gegen mehrere AABBs von Vorteil ist, wie es etwa in einer BVH der Fall ist.

Die Schnittpunktberechnung zwischen einem Strahl und einer AABB ist eine sehr häufig verwendete und benötigte Technik im Feld der Computergraphik, bspw. bei vielen Visibilitätsanfragen oder insbesondere in Ray Tracing Systemen, welche auf Bounding Volume Hierarchien basieren [GS87][KK86], sowie auch das in dieser Arbeit vorgestellte System. Im nächsten Abschnitt werden die gängigsten Varianten für Schnittpunkttests zwischen einem Strahl und einer AABB vorgestellt, und darauf folgend in Abschnitt 5.2, die neu entwickelte *Ray-Slope* Variante.

5.1 Schnitttest Strahl-AABB

5.1.1 Slab-Test nach Kay/Kajiya

Der bekannteste Weg einen Strahl gegen eine AABB zu testen, ist der so genannte *Slab-Test* nach Kay/Kajiya [KK86]. Ein *Slab* bezeichnet dabei den Raum zwischen zwei parallelen Ebenen. Mindestens drei solcher Slabs mit linear unabhängigen Normalen definieren durch ihre Schnittmenge einen konvexen Körper. Eine AABB B kann somit durch je drei Slabs repräsentiert werden, deren Normalen auf die Achsen des Koordinatensystems beschränkt sind und die jeweils durch einen der Extrempunkte $\mathbf{b_0} = (x_0, y_0, z_0)$ oder

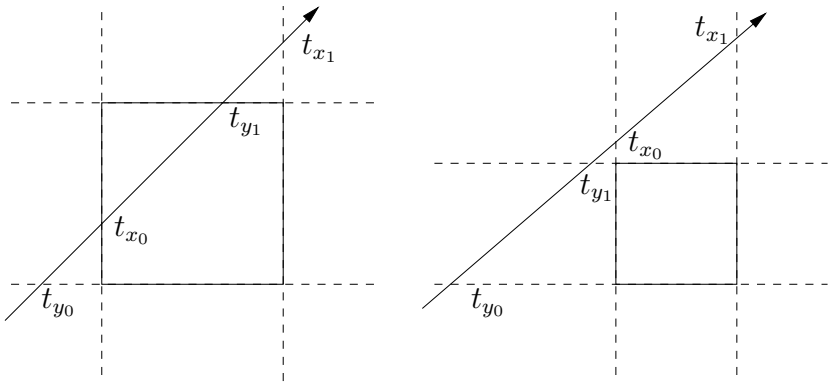


Abbildung 5.1: Beispiel für einen Slab-Test nach Kay/Kajiya. Im linken Bild trifft der Strahl die Box, da gilt $[t_{x_0}, t_{x_1}] \cap [t_{y_0}, t_{y_1}] = [t_{x_0}, t_{y_1}]$. Im rechten Bild verfehlt der Strahl die Box, da gilt $[t_{x_0}, t_{x_1}] \cap [t_{y_0}, t_{y_1}] = \{\}$

$\mathbf{b_1} = (x_1, y_1, z_1)$, welche die AABB B definieren, verlaufen. Dabei gilt für alle möglichen Punkte $\mathbf{p} = (x_i, y_i, z_i)$ welche innerhalb von B liegen

$$\forall \mathbf{p} \in B : (x_0 \leq x_i \leq x_1) \wedge (y_0 \leq y_i \leq y_1) \wedge (z_0 \leq z_i \leq z_1). \tag{5.1}$$

Der Strahlparameter t_{x_0} für Schnitt eines Strahles der Form

$$\mathbf{R}(t) = \mathbf{o} + t\vec{\mathbf{v}} \quad ,$$

mit $\mathbf{o} = (o_x, o_y, o_z)^\top$ und $\vec{\mathbf{v}} = (v_x, v_y, v_z)^\top$ mit der senkrecht zur x-Achse verlaufenden Ebene durch x_0 kann dann wie folgt berechnet werden:

$$t_{x_0} = \frac{x_0 - o_x}{d_x} \tag{5.2}$$

Die Berechnung für die übrigen fünf Ebenen verlaufen analog. Ein Strahl trifft nun diese Box genau dann, wenn gilt, dass der Überlappungsbereich der Intervalle $[t_{x_0}, t_{x_1}]$, $[t_{y_0}, t_{y_1}]$ und $[t_{z_0}, t_{z_1}]$ nicht leer ist (vgl. Abbildung 5.1). Sollte eine der Strahlrichtungen negativ sein, muss das Intervall dementsprechend umgekehrt werden.

Problematisch wird diese Variante, sollte eine der Richtungskomponenten des Strahles null werden. Normalerweise verursacht dies Probleme, da eine Division durch Null nicht definiert ist. Smits [Smi98] stellte jedoch fest, dass der IEEE-Standard für Fließkomma-Arithmetik eine Division durch den Wert Null als entweder $+\infty$ oder $-\infty$ definiert, abhängig von den Vorzeichen. Dies löst das Problem, da die Division $[-\infty, -\infty]$ oder $[+\infty, +\infty]$ liefert, wenn der Strahl die Box verfehlt, oder $[-\infty, +\infty]$, wenn der Strahl die Box trifft. Also

genau wie erwünscht. Dadurch wird ein explizites Testen der Strahlrichtung auf Null überflüssig. Ein Fehler tritt jedoch noch auf, wenn eine Division durch -0 durchgeführt wird. Dies kann jedoch behoben werden, indem die Division durch die Strahlrichtung $\vec{\mathbf{d}} = (d_x, d_y, d_z)^\top$ in der Berechnung durch eine Multiplikation mit den entsprechenden Kehrwerten $\vec{\mathbf{d}^*} = (\frac{1}{d_x}, \frac{1}{d_y}, \frac{1}{d_z})^\top$ ersetzt wird [Smi02]. Eine weitere Optimierung für BVHs lässt sich erzielen, berechnet man $\vec{\mathbf{d}^*}$ vorab bei der Erzeugung des Strahles, anstatt bei jedem Schnitttest erneut zu berechnen [WBMS05]. Ein letzter Schritt, der durchgeführt werden könnte, wäre anhand der Strahlrichtung vorab eine Klassifizierung des Strahles vorzunehmen, wodurch der Test, welche Ebene eines Slabs zuerst geschnitten wird unnötig wird.

5.1.2 Schnitttest nach Woo

Woo [Woo90] schlug 1990 einen Test vor, der lediglich zu drei der sechs so genannten *Kandidatenebenen* der AABB einen Schnitt berechnen musste. Dies geschieht, durch Betrachtung der relativen Position des Strahlursprungs zur AABB. Gilt $o_x < x_0$, so muss der Strahl bspw. gegen die durch x_0 verlaufende Ebene getestet werden, gilt $o_x > x_1$, dann diejenige durch x_1. Falls der Strahl zwischen beiden Ebenen liegt, muss keine getestet werden. Mit den übrigen Ebenen wird analog verfahren.

Liegt der Strahl nun im Inneren der Box, ist ein Treffer garantiert. Andernfalls ist das Maximum der berechneten Strahlparameter der Schnittpunkte mit den zuvor berechneten Kandidatenebenen zu bestimmen. Anhand dessen werden die Koordinaten des Schnittpunktes berechnet, mit der getroffenen Ebene verglichen und auf dieser Grundlage entschieden, ob der Strahl wirklich die Box trifft.

Der benötigte Code für diesen Test ist im Verhältnis zum Slab-Test recht kompliziert und für heutige Prozessorarchitekturen eher ungeeignet. Aus diesem Grund wurde in dieser Arbeit von einer weiteren Untersuchung dieses Testes abgesehen.

5.1.3 Schnitttest mit Plücker Koordinaten

Vor Kurzem präsentierten Mahovsky und Wyvill [MW04] einen gänzlich neuen Ansatz, welcher sich *Plücker Koordinaten* zunutze machte, um den Strahl gegen die Silhouette der AABB zu testen, ohne diese auf eine zweidimensionale Ebene projizieren zu müssen. Die Methode ist gänzlich divisionsfrei,

verwendet lediglich Skalarprodukte und erzielte beeindruckende Ergebnisse, verglichen mit den Implementationen von Kay/Kajiyas und Smits Algorithmus.

Eine Gerade im dreidimensionalen Raum, welche durch zwei Punkte $\mathbf{a} = (a_x, a_y, a_z)^\top$ und $\mathbf{b} = (b_x, b_y, b_z)^\top$ verläuft, lässt sich mittels sechs Plücker Koordinaten $G_0 \dots G_5$ beschreiben:

$$\begin{aligned} G_0 &= a_x b_y - b_x a_y \\ G_1 &= a_x b_z - b_x a_z \\ G_2 &= a_x - b_x \\ G_3 &= a_y b_z - b_y a_z \\ G_4 &= a_z b_z \\ G_5 &= b_y a_y \end{aligned} \tag{5.3}$$

Ein Strahl $\mathbf{R}(t) = \mathbf{o} + t\vec{\mathbf{d}}$, mit $\mathbf{o} = (o_x, o_y, o_z)^\top$ und $\vec{\mathbf{d}} = (d_x, d_y, d_z)^\top$, kann nun ebenfalls durch Plücker Koordinaten ausgedrückt werden, indem man in Gleichung (5.3) für $\mathbf{a}$ $\mathbf{o}$ einsetzt und für $\mathbf{b}$ $(\mathbf{o} + \vec{\mathbf{d}})$. Die entsprechenden Plücker Koordinaten $R_0 \dots R_5$ sind dann

$$\begin{aligned} R_0 &= o_x d_y - d_x o_y \\ R_1 &= o_x d_z - d_x o_z \\ R_2 &= -d_x \\ R_3 &= o_y d_z - d_y o_z \\ R_4 &= -d_z \\ R_5 &= d_y \end{aligned} \tag{5.4}$$

Die Relation $side(G, R)$ beschreibt nun die relative Lage zweier Geraden zueinander mittels

$$side(G, R) = R_0 G_4 + R1_G 5 + R_3 G_2 + R_2 G_3 + R_5 G_1 + R_4 G_0 \tag{5.5}$$

Dadurch lässt sich ermitteln, ob eine Gerade links oder rechts aus Sicht einer anderen Geraden verläuft, je nachdem ob $side(G, R) > 0$ oder $side(G, R) < 0$ gilt.

Auch die Silhouette einer AABB lässt sich nun mit Plücker Koordinaten beschreiben. Da lediglich AABBs betrachtet werden, vereinfacht sich der Test

enorm, da bspw. die Kante **FE** aus Abbildung 5.2 auf Seite 115 sich beschreiben lässt mittels

$$\begin{aligned} FE_0 &= -x_0 \\ FE_1 &= 0 \\ FE_2 &= 0 \\ FE_3 &= z_1 \\ FE_4 &= 0 \\ FE_5 &= 0 \end{aligned} \tag{5.6}$$

Wodurch die *side*-Relation vereinfacht wird zu

$$side(R, FE) = -R_1 - d_x z_1 + d_z x_0 \tag{5.7}$$

Da durch die Vorzeichen des Richtungsvektors eines Strahles bereits vorab feststeht, welche Kanten getestet werden müssen, muss obige Relation lediglich gegen diese getestet werden, um zu wissen ob der Strahl die Box trifft oder an ihr vorüberläuft. Für einen Strahl mit sämtlichen Richtungskomponenten negativ ergäbe sich folgender Pseudocode:

```
if((side(R,DH) > 0) ||
   (side(R,BF) < 0) ||
   (side(R,EF) > 0) ||
   (side(R,DC) < 0) ||
   (side(R,BC) > 0) ||
   (side(R,EH) < 0))
     return false;
else
  return true;
```

Um zu verhindern, dass negative Strahlparameter gefunden werden, muss vorab noch ein Test stattfinden, der anhand der Strahlrichtungskomponenten bestimmt, ob eine AABB im positiven Halbraum eines Strahles liegt.

5.2 Der Ray-Slope Schnitttest

In der nun vorgestellten Methode wird, anstatt den Strahl in 3D gegen die AABB zu testen, dass Problem aufgespalten. Stattdessen werden drei Tests

in 2D mittels der Projektionen auf die Ebenen senkrecht zu den Weltkoordinatenachsen durchgeführt. Durch Ausnutzung des Wissens, welches durch die Steigung des Strahles gegeben ist, kann ein Großteil der benötigten Informationen für den Schnitttest vorab berechnet werden. Das Verfahren ist divisionsfrei und seine Berechnungen können unabhängig voneinander durchgeführt werden. Die Testergebnisse zeigen, dass diese Methode bis zu 15% schneller ist als alle anderen mir bekannten Methoden, auch wenn keine Schnittdistanz berechnet wird. Dies ist jedoch kein allzu großer Nachteil, wie Smits bereits in [Smi98] dargelegt hat, da viele Ray Tracer, welche BVH nutzen, eine vorab festgelegte Reihenfolge besitzen, z.B. bei der in Abschnitt 4.1.2 vorgestellten iterativen Variante der Tiefensuche, in welcher die BVH traversiert wird, und deswegen eine Schnittdistanz nicht zwingend benötigt wird. Desweiteren kann, falls notwendig, diese zusätzlich und ohne größeren Aufwand berechnet werden.

Üblicherweise werden alle sechs Seiten einer AABB auf Schnitte mit einem Strahl $\mathbf{R}(t)$ getestet. Diese Anzahl kann auf drei reduziert werden, falls die Vorzeichen der Richtungskomponenten von $\mathbf{R}(t)$ bekannt sind. Wenn z.B. ein Strahl mit allen Richtungskomponenten negativ gegeben ist, dann muss dieser durch die Seiten FEHG, CGHD oder BFGC verlaufen, wie in Abbildung 5.2 dargestellt. Der Strahlparameter könnte dabei jedoch auch negativ sein. Wenn im Moment von einem Strahl gesprochen wird, ist somit eine Gerade im Raum gemeint. Ein solcher Strahl wird als MMM für „minus minus minus“ klassifiziert. Die anderen sieben Klassifizierungen, sind MMP, MPM, MPP, PMM, PMP, PPM und PPP, wobei M und P jeweils korrespondierend zum Vorzeichen der entsprechenden Richtungskomponente d_x, d_y and d_z verwendet wird, (vgl. [MW04]). Auch wenn eine solche Klassifizierung für die anderen Methoden ausreichend ist, benötigt die hier vorgestellte *Ray Slope*-Methode noch eine weitere Klasse, genannt O. Diese wird für jede Richtungskomponente verwendet, welche genau Null ist, so würde bspw. ein Richtungsvektor $\vec{\mathbf{d}} = (1, 0, -1)^\top$ als POM klassifiziert. Dies führt zu einer Gesamtanzahl von 26 verschiedenen Klassen (theoretisch 27, aber der Fall OOO kann normalerweise in einem Ray Tracer nicht auftreten, ein Hinzufügen wäre allerdings trivial). Auch wenn dies zunächst sehr viel scheint, treten die meisten der Fälle sehr selten auf und brauchen dementsprechend auch nicht oft getestet zu werden. Und selbst wenn, so vereinfacht sich die Berechnung enorm, wodurch die benötigte Ausführungszeit erneut gewinnt.

Um das Problem zu lösen, einen Strahl $\mathbf{R}(t)$ mit einer AABB B im dreidimensionalen Raum zu schneiden, wird das Problem, wie bereits angemerkt, darauf reduziert, den projizierten Strahl und die Box im zweidimensionalen Raum zu schneiden. Um jedoch gleiche Resultate zu erhalten, muss dies

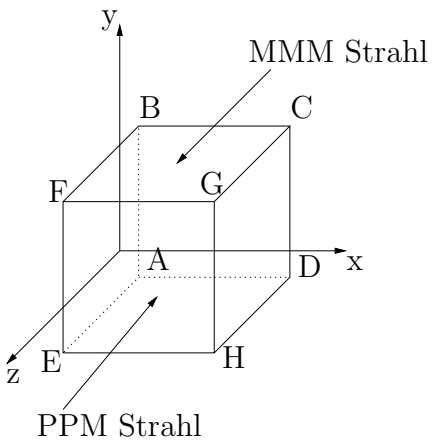


Abbildung 5.2: Achsenparallele Box, welche von einem MMM Strahl und einem PPM Strahl geschnitten wird.

für alle drei Ebenen, welche senkrecht zu den Achsen des Weltkoordinatensystems stehen, durchgeführt werden, welche im folgenden xy-, xz- und yz-Ebene genannt werden. Dies ist möglich da AABBs benutzt werden. Diese Projektionen benötigen keinerlei zusätzliche Berechnungen, da jeweils lediglich eine der Koordinaten ausgelassen werden muss. Dabei gilt, dass der Strahl die Box verfehlt wenn nur einer der drei Schnitttests fehlschlägt. Falls alle drei gelingen, wurde ein Schnitt mit der Box gefunden.

Im folgenden wird der Algorithmus für eine der drei Projektionsebenen, die xy-Ebene, und einen PPP-Strahl beschrieben, da alle anderen Fälle äquivalent durchgeführt werden können. Ausnahmen bilden diejenigen, bei denen mindestens eine der Richtungskomponenten des Strahles gleich null wird, welche danach beschrieben werden. Für die restlichen Fälle sei auf den Sourcecode verwiesen.

Sei die Steigung (engl. *slope*) eines parametrisch definierten Strahles $\mathbf{R}(t) = \mathbf{o} + t\vec{\mathbf{d}}$ in der xy-Ebene wie folgt definiert:

$$S_{\mathrm{xy}}(\mathbf{R}(t)) = d_y/d_x \tag{5.8}$$

Damit ein PPP Strahl eine AABB schneiden kann, muss er zwingend durch das Liniensegment verlaufen, welches durch die Punkte (x_0, y_1) und (x_1, y_0) der AABB definiert ist. Zunächst werden nun zwei Hilfsstrahlen $\mathbf{A}(t)$ und $\mathbf{B}(t)$ gebildet. Diese beginnen im Ursprung des Strahles $\mathbf{R}(t)$ und verlaufen

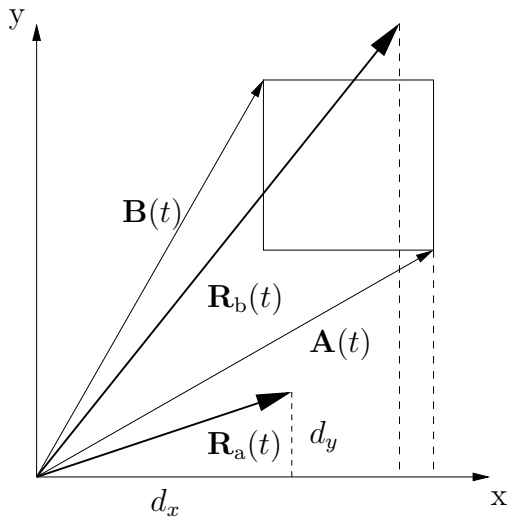


Abbildung 5.3: Ray Slope Test in der xy-Ebene

durch diese beiden Punkte, mit

$$\begin{aligned} \mathbf{A}(t) &= \mathbf{o} + t \begin{pmatrix} x_1 - o_x \\ y_0 - o_y \end{pmatrix} \quad , \\ \mathbf{B}(t) &= \mathbf{o} + t \begin{pmatrix} x_0 - o_x \\ y_1 - o_y \end{pmatrix} \end{aligned}$$

Im Falle eines PPP Strahles $\mathbf{R}(t)$, muss $S_\mathrm{xy}(\mathbf{R}(t))$ folgende Bedingungen erfüllen, damit der Strahl die Box sicher verfehlt:

$$\begin{aligned} S_\mathrm{xy}(\mathbf{R}(t)) &< S_\mathrm{xy}(\mathbf{A}(t)) \\ \Leftrightarrow S_\mathrm{xy}(\mathbf{R}(t)) &< (y_0 - o_y)/(x_1 - o_x) \quad , \end{aligned} \tag{5.9}$$

oder

$$\begin{aligned} S_\mathrm{xy}(\mathbf{R}(t)) &> S_\mathrm{xy}(\mathbf{B}(t)) \\ \Leftrightarrow S_\mathrm{xy}(\mathbf{R}(t)) &> (y_1 - o_y)/(x_0 - o_x) \end{aligned} \tag{5.10}$$

Dies ist in Abbildung 5.3 dargestellt. Der Strahl $\mathbf{R}_\mathrm{a}(t)$ verfehlt die Box, bedingt durch seine niedrigere Steigung als $\mathbf{A}(t)$. Der Strahl $\mathbf{R}_\mathrm{b}(t)$ dagegen trifft die Box in der xy-Ebene.

Gleichung (5.9) gilt für jeden als PPP klassifizierten Strahl mit $o_x \leq x_1$ und $o_y \leq y_1$, Gleichung (5.10) lediglich für solche Strahlen, für die gilt $o_x < x_0$ und

$o_y \leq y_1$, da sich sonst das Vorzeichen des rechten Teiles der Gleichung (5.10) verändern könnte. Um diesen Nachteil zu überbrücken, wird die Richtung in welcher die Steigung berechnet wird abgeändert. Statt den Vergleich mittels $S_{\text{xy}}(\mathbf{R})(t)$ durchzuführen, wird

$$S_{\text{yx}}(\mathbf{R}(t)) = d_x/d_y \tag{5.11}$$

für den zweiten Fall gewählt, was die Bedingung aus Gleichung (5.10) zu Gleichung (5.12) abändert.

$$S_{\text{yx}}(\mathbf{R}(t)) < (x_0 - o_x)/(y_1 - o_y) \tag{5.12}$$

Falls vorab getestet wird, dass die AABB im positiven Raum des Strahles liegt, mittels $o_x < x_1$, $o_y < y_1$ und $o_z < z_1$, dann sind die Kriterien aus Gleichung (5.9) und (5.12) hinreichend, um einen PPP Strahl mit Box B in seiner positiven Richtung in der xy-Ebene auf Schnitte zu testen.

Bis jetzt scheint die Berechnung noch sehr kompliziert zu sein. Um die erwünschte Vereinfachung herbeizuführen, wird die Anordnung abgeändert. Dies wird für Gleichung (5.9) gezeigt, Gleichung (5.12) kann äquivalent umgeformt werden.

Dadurch erhält man:

$$\begin{aligned} S_{\text{xy}}(\mathbf{R}(t))x_1 - y_0 + (o_y - S_{\text{xy}}(\mathbf{R}(t))o_x) &< 0 \\ \Rightarrow S_{\text{xy}}(\mathbf{R}(t))x_1 - y_0 - C_{\text{xy}}(\mathbf{R}(t) &< 0 \end{aligned} \tag{5.13}$$

Auch wenn dadurch zunächst nicht viel gewonnen zu sein scheint, so kann der Großteil der benötigten Berechnungen einmalig bei Erzeugung des Strahles durchgeführt werden, statt für jeden Boxtest aufs neue. Dies gilt für $S_{\text{xy}}(\mathbf{R}(t))$ und selbst $C_{\text{xy}}(\mathbf{R}(t)) = o_y - S_{\text{xy}}(\mathbf{R}(t))o_x$. Dies führt dazu, dass im besten Falle bereits nach einer Multiplikation, zwei Subtraktionen und einem Vergleich feststehen kann, dass der Strahl die AABB verfehlt. Eine weitere Subtraktion könnte noch zusätzlich entfernt werden, indem y_0 auf die andere Seite der Gleichung gebracht wird. Aber die meisten C++ Compiler scheinen bessere Ergebnisse mit dieser Variante zu liefern.

Folglich lässt sich für einen PPP Strahl und eine AABB der komplette Algorithmus wie in Abbildung 5.4 darstellen. `r` bezeichnet dabei den Strahl und `b` die AABB.

Falls eine der Richtungskomponenten des Strahles, z.B. d_x gleich null wird, dann vereinfacht sich der Code wie in Abbildung 5.5 angegeben.

```
if ((r->x > b->x1) || (r->y > b->y1) || (r->z > b->z1)
   || (r->s_xy * b->x1 - b->y0 + r->c_xy < 0)
   || (r->s_yx * b->y1 - b->x0 + r->c_yx < 0)
   || (r->s_xz * b->x1 - b->z0 + r->c_xz < 0)
   || (r->s_zx * b->z1 - b->x0 + r->c_zx < 0)
   || (r->s_yz * b->y1 - b->z0 + r->c_yz < 0)
   || (r->s_zy * b->z1 - b->y0 + r->c_zy < 0))
   return false;

return true;
```

Abbildung 5.4: Pseudocode des Ray Slope Schnitttests für einen PPP Strahl und eine AABB

```
if((r->x < b->x0) || (r->x > b->x1)
  || (r->y > b->y1) || (r->z > b->z1)
  || (r->s_yz * b->y1 - b->z0 + r->c_yz < 0)
  || (r->s_zy * b->z1 - b->y0 + r->c_zy < 0))
  return false;

return true;
```

Abbildung 5.5: Pseudocode des Ray Slope Schnitttests für einen OPP Strahl und eine AABB

D.h. Zeile 2,3,4 und 5 des ursprünglichen Codes können ersetzt werden, durch den Vergleich, ob o_x zwischen den Ausdehnungen der Box entlang der x-Achse liegt. Falls zwei der Richtungkomponenten null werden, kann der Schnitt durch einen einfachen Vergleich zwischen dem Strahlursprung und -richtung und den Ausdehnungen der Box ersetzt werden.

Bisher wurde durch den in Abbildung 5.4 dargestellten Code, keine Schnittdistanz berechnet. Falls diese jedoch zwingend benötigt würde, so kann sie hinzugefügt werden, indem die Distanz zu den drei möglichen Schnittebenen berechnet und die größte Distanz ausgewählt wird, falls vorab ein Schnitt bestätigt wurde. Die Berechnung ist dabei äquivalent zu Gleichung (5.2), nur, dass mit dem Kehrwert der Strahlrichtung multipliziert wird, um die Division zu ersetzen. Diese können ebenfalls bei Erzeugung des Strahles berechnet werden. Welche drei Ebenen getestet werden müssen, steht bereits durch die Klassifizierung der Strahlrichtung fest.

Die Steigung eines Strahles als Test auf Schnitte mit einer AABB zu verwenden hat verschiedene Vorteile. Da die meisten der benötigten Informationen vorab berechnet werden, konnte eine Methode präsentiert werden, welche divisionsfrei ist, falls keine Schnittdistanz benötigt wird. Und selbst wenn eine benötigt wird, können sämtliche Divisionen durch Multiplikation mit dem Kehrwert der Richtungskomponenten des Strahles ersetzt werden. Die verschiedenen Berechnungen sind unabhängig voneinander und könnten folglich parallel zueinander berechnet werden. Die Testergebnisse sind in Abschnitt 6.4 aufgeführt.

Kapitel 6

Testergebnisse

Um die vorgestellten Verfahren testen und vergleichen zu können, wurde eine Reihe von Testszenen ausgewählt, welche ein umfassendes Bild möglicher Schwierigkeiten, aber auch Vorteile der einzelnen Methoden in dynamischen Szenen hervorheben sollen und im folgenden Abschnitt vorgestellt werden. Anschließend werden die Testergebnisse aufgeführt und diskutiert. Danach werden die Ergebnisse für den in Abschnitt 5.2 vorgestellten Schnitttest präsentiert und verglichen.

6.1 Testumgebung

Im folgenden werden die Besonderheiten der verschiedenen Testszenen erläutert. Einige Bilder jeder Szene sind in Anhang A aufgeführt.

- *Testszene 1 - BART Kitchen Scene:* In dieser, leicht abgewandelten, Szene aus dem BART[1]-Paket von Lext *et al.* [LAM01a] fährt ein kleines Spielzeugauto durch eine Küche. Dieses wurde hierarchisch animiert, mittels Translation, Rotation und Skalierung. Die Szene besteht aus 110.540 Dreiecken und sechs Punktlichtquellen. Die Auflösung der 800 berechneten Bilder wurde auf 300×225 Pixel gesetzt, sämtliche Oberflächen sind reflektierend. Bis auf das Modellauto, welches aus fünf Objekten mit jeweils lokaler Beschleunigungsstruktur besteht, ist die Szene statisch. Dadurch finden sich in dieser Szene vor allem drei Schwierigkeiten, die es zu bewältigen gilt. Erstens hierarchische Animationen, hier kann allerdings kein Vergleich angestellt werden, da alle

[1]BART = Benchmark for animated Ray Tracing

Methoden auf gleiche Art den Szenegraphen umformen. Zweitens das so genannte *Teapot in a stadium*-Problem, da die Küche sowohl aus sehr großen Primitiven, wie dem Boden, sowie sehr kleinen, aber komplexen Objekten besteht, wie etwa dem Türknauf. Und als dritte Schwierigkeit enthält die Szene sehr viele statische Bereiche. Gerade bei Methoden für dynamische Szenen ist es wichtig, wie diese mit statischen oder momentan nicht bewegten Objekten umgehen.

- *Testszene 2 - BART Museum Scene:* Diese Szene entstammt ebenfalls dem BART-Paket und stellt einen Museumsraum dar, in dessen Mitte sich ein abstraktes Kunstwerk befindet. Da unser Ray Tracer nicht den kompletten Funktionsumfang von AFF[2]-Dateien unterstützt, fehlen die Wände und die verwendeten Materialien wurden abgeändert. Oberhalb dieses abstrakten Kunstwerkes, befindet sich eine Ansammlung von 65.536 animierten Dreieckspatches. Ein Dreieckspatch ist ein Dreieck mit Normalen an jedem Eckpunkt. Insgesamt enthält die Szene 75.546 Primitive und zwei Punktlichtquellen. Die Animation besteht aus 300 Bildern mit einer Auflösung von 800×600 Pixeln. Diese Dreieckspatches werden im Laufe der Zeit von einer Konstellation in fünf andere interpoliert. Diese Szene bietet vor allem zwei große Probleme. Erstens eine sehr große Anzahl an dynamischen Objekten, bei einer vergleichsweise sehr geringen Anzahl an statischen Objekten. Und zweitens eine starke Verschiebung der Objektdistributionen. Zu Beginn der Animation liegen die Dreieckspatches sehr nah beieinander, bevor sie in einem rotierenden Zylinder aufsteigen und sich schließlich zu einem Tetraeder, einer Kugel, sowie zuletzt einem abstrakten Kunstgebilde verformen.

- *Testszene 3 - Falling Triangles:* In dieser aus 149.058 animierten Objekten und zwei Punktlichtquellen bestehenden Szene fallen zufällig in einer Ebene angeordnete Dreiecke in ebenfalls zufälliger Reihenfolge und Geschwindigkeit von der Decke herab und fügen sich am Boden zu einem Bild zusammen. Die Kamera verharrt dabei auf einer Position, die stets die gesamte Szene im Blick hat. Anders als in der vorhergehenden Testszene, besteht diese lediglich aus 11 Bildern, wodurch sich die Objekte sehr stark in Relation zueinander bewegen. Dies macht es schwieriger die Lokalitäten zwischen den Objekten auszunutzen. Hinzu kommt, dass die Objektdistribution im Verlaufe der Animation sehr stark schwankt und sowohl am Anfang als auch am Ende in eine Form übergeht, die einigen Verfahren Schwierigkeiten bereiten

[2]AFF = Animated File Format

könnte, nämlich eine einzige flache Ebene, deren Ausdehnung entlang der y-Achse null ist. Die Auflösung beträgt jeweils 512×512 Bildpunkte.

- *Testszene 4 - Okklusion:* Bei dieser Testszene handelt es sich um eine Abwandlung der vorigen. Die Kamera wurde dabei so in die Szene verschoben, dass die Primärstrahlen in der ersten Hälfte der Szene auf kein Objekt treffen, jedoch in der zweiten Hälfte Teilbereiche der Szene sichtbar machen. Dies soll vor allem den Einfluss der Lazy Evaluation Strategie deutlich machen.

- *Testszene 5 - Robotdance:* Insgesamt 1600 animierte Objekte mit insgesamt 624.100 Primitiven und einer Punktlichtquelle werden in dieser Szene dargestellt. Die Auflösung der berechneten 800 Bilder beträgt jeweils 800×600 Pixel. Die in dieser Szene dargestellten Roboter sind zu einem zweidimensionalen Gatter aus insgesamt 100 Robotern angeordnet, mit jeweils 16 animierten Gliedern, und je 6.241 Dreiecken. Diese laufen auf der Stelle und drehen sich dabei, während die Kamera, welche zunächst nur wenige der Roboter im Blickfeld hat, langsam zurückfährt, um letzten Endes die gesamte Szene zu überblicken. Diese Szene wirft vor allem zwei Probleme auf. Erstens die relativ zweidimensionale Ausrichtung der Szene, alle Roboter stehen auf einer Ebene, und zweitens die lediglich lokal stattfindende Bewegung. Die Gliedmaßen der Roboter verändern zwar ihre Position, die Relation der Roboter untereinander jedoch bleibt gleich. Dies soll quasi Szenen symbolisieren, die in verschiedene voneinander getrennte Bereich aufgeteilt sind, wie es häufig in Computerspielen praktiziert wird. Durch die Kamerabewegung wird zudem erreicht, dass zu Beginn nur ein relativ kleiner Ausschnitt der Szene zu sehen ist, welcher mit der Zeit größer wird, bis quasi die gesamte Szene überblickt wird.

6.2 Testergebnisse

Die verschiedenen Tests wurden auf einem 2,0 GHz Pentium Mobile mit 512MB RAM durchgeführt. Als Programmiersprache diente C++. Die Bilder wurden mit einer maximalen Rekursionstiefe von eins gerendert, d.h. eine Reflexion pro Pixel war erlaubt. Auf Frameless Rendering wurde verzichtet, was bedeutet, dass für jeden Pixel des Bildes ein Primärstrahl erzeugt wurde.

Dabei stellen die angegebenen Rekonstruktionszeiten, die reinen Zeiten für die Aktualisierung der Hierarchie dar. Sie beinhalten nicht mehr die Berech-

nung der Bewegung der einzelnen Objekte, wohl aber die Berechnung der benötigten inversen Matrizen oder das Anpassen der Hüllvolumen der Objekte. Dies wurde so gewählt, weil dieser Teil sehr implementationsabhängig ist und die Ergebnisse somit verfälschen würde. Ob schließlich ein Objekt mittels einfacher Translation animiert wird, oder Spline Interpolation zur Berechnung der entsprechenden Transformationsmatrix verwendet wurde, ist für die Verfahren selbst uninteressant.

Für jede Testszene werden die Rekonstruktions- (*up*) und Ray Tracing-Zeiten (*rt*) anhand eines Graphen über die Länge der Animation verglichen, sowie in der jeweils darunterstehenden Tabelle allgemeinere Informationen zum Vergleich gegeben, wie durchschnittliche (*avg*), beste (*best*) und schlechteste (*worst*) Zeiten, sowie ein relativer Vergleich der benötigten Zeit im Vergleich zu einem kompletten Neuaufbau (*speedup*). Dabei ist zu beachten, dass die Methoden Dynamic Median Cut 1 und 2 jeweils mit der Median-Cut Rebuild Variante verglichen werden, und Dynamic Goldsmith and Salmon und Local Sort, mit Goldsmith and Salmon Rebuild. Ein Vergleich für die Loose Bounding Volume Hierarchy ist schwierig, da das Verfahren gänzlich unterschiedlich zu den anderen ist. Deswegen wurde ein Vergleich zu beiden Referenzmethoden erstellt. Der erste Wert gibt jeweils das Verhältnis zur Median-Cut Rebuild Methode, der zweite zur Goldsmith und Salmon Rebuild Variante.

Bei der Angabe der mittleren Rekonstruktionszeiten, wurde die Zeit für den Initialaufbau nicht mitbeachtet. Die benötigte Zeit für diesen ist zu vernachlässigen und zählt als Vorverarbeitungsschritt. Vor allem bei den kürzeren Testszenen könnte sonst ein falsches Bild entstehen. Die angegebenen Zeiten sind jeweils in Sekunden angegeben. Die Zeiten für die Median-Cut Refit und Goldsmith und Salmon Refit-Methode wurden der Übersichtlichkeit halber nicht mehr verglichen, da zum einen die Ray Tracing Phase sehr schnell Größen erreichte, die ein Rendern unmöglich machten, und die Rekonstruktionszeiten vergleichbar sind mit denen der Local Sort und Dynamic Median-Cut 2 Methoden.

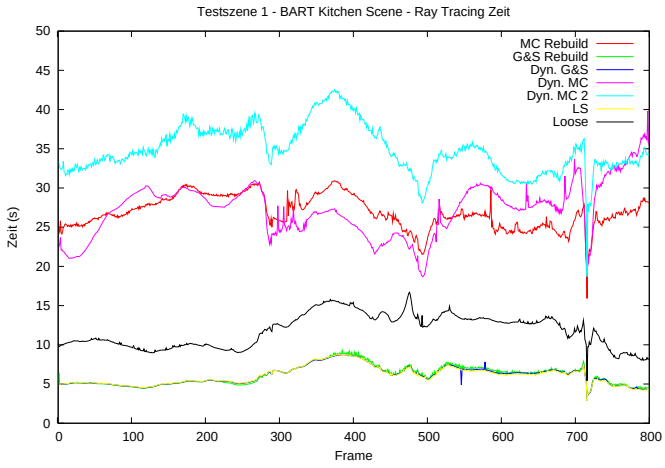
Für die TestSzene 5 - Robotdance sind die Rekonstruktionszeiten der Übersichtlichkeit halber lediglich für einen Bruchteil der Frames angegeben, da diese während der Animation durchgängig ähnlich verlaufen.

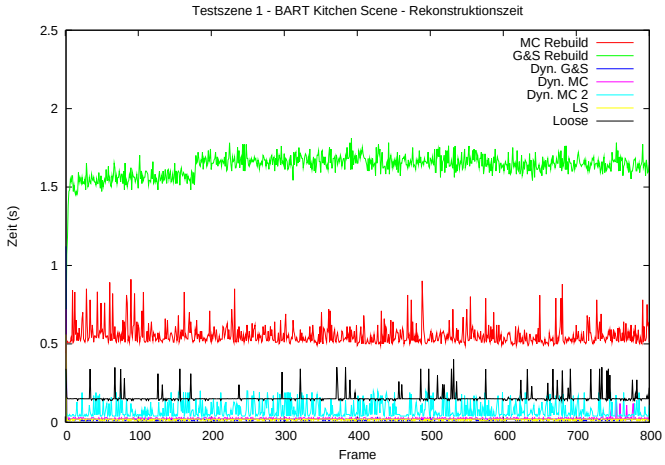
Die Abkürzungen sind dabei wie folgt zu deuten:

- *MC Rebuild* - Median Cut Rebuild, kompletter Neuaufbau der mittels Median-Cut Methode erstellten Hierarchie in jedem Frame (siehe auch Abschnitt 4.3.1).
- *G&S Rebuild* - Goldsmith & Salmon Rebuild, kompletter Neuaufbau

der nach dem Verfahren von Goldsmith & Salmon erstellten Hierarchie in jedem Frame (siehe auch Abschnitt 4.3.1).

- *Dyn. G&S* - Dynamic Goldsmith and Salmon. Methode aus Abschnitt 4.3.2
- *Dyn. MC* - Dynamic Median-Cut Methode aus Abschnitt 4.3.3
- *Dyn. MC2* - Dynamic Median-Cut 2 Methode aus Abschnitt 4.3.5
- *LS* - Local Sort Methode aus Abschnitt 4.3.4
- *Loose* - Loose Bounding Volume Hierarchy Methode aus Abschnitt 4.3.6

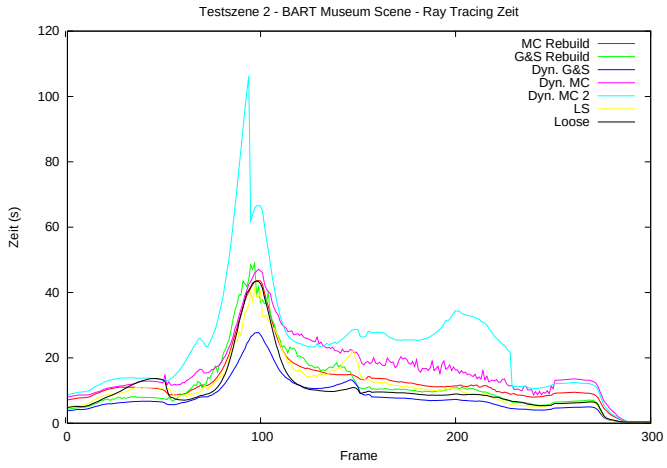


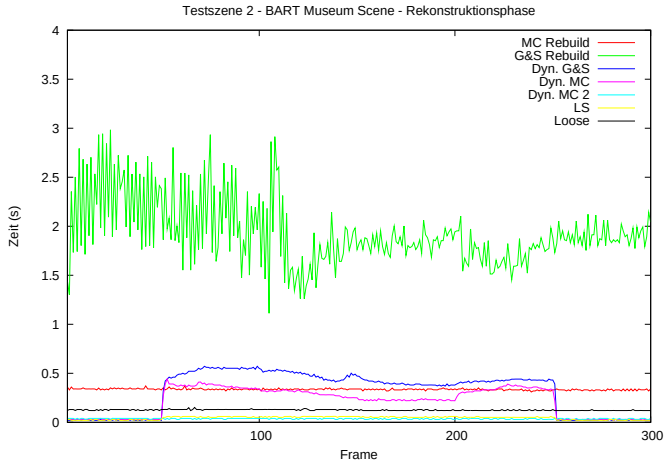


Methode	avg. up	avg. rt	best up	best rt	worst up	worst rt
MC Rebuild	0.557	26.880	0.490	15.903	0.912	30.914
G&S Rebuild	1.759	6.142	1.102	2.904	2.444	9.273
Dyn. G&S	0.017	6.082	0.010	2.934	0.030	8.762
Dyn. MC	0.025	27.267	0.020	18.687	0.150	39.807
Dyn. MC2	0.075	34.851	0.040	18.697	0.201	42.541
LS	0.017	6.056	0.010	2.884	0.0210	8.822
Loose	0.157	11.771	0.140	5.418	0.401	16.704

Methode	avg. speedup up	avg. speedup rt	avg. speedup total
Dyn. G&S	103.471	1.010	1.283
Dyn. MC	22.28	0.986	1.005
Dyn. MC2	7.427	0.771	0.786
LS	103.471	1.014	1.301
Loose	3.548 / 11.204	2.284 / 0.522	2.300 / 0.662

Abbildung 6.1: Ergebnisse TestSzene 1 - BART Kitchen Scene

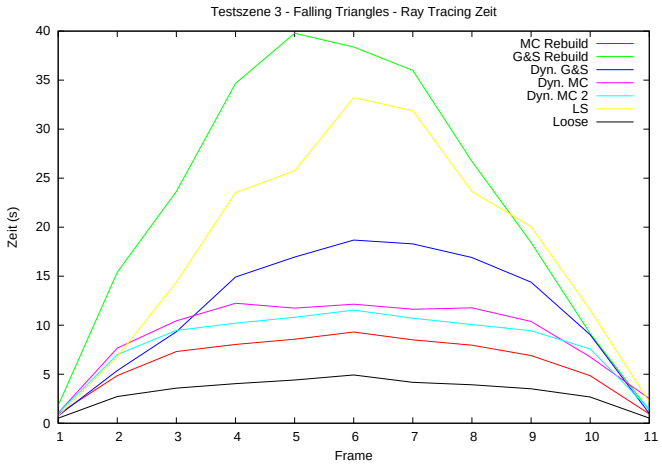


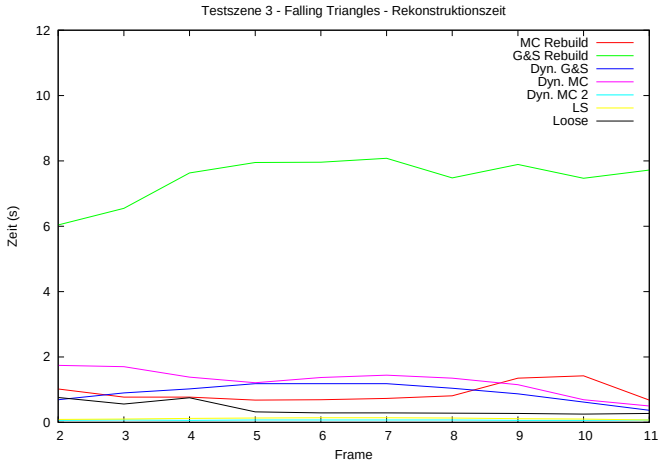


Methode	avg. up	avg. rt	best up	best rt	worst up	worst rt
MC Rebuild	0.336	12.708	0.310	0.360	0.371	43.653
G&S Rebuild	1.933	11.839	1.112	0.380	2.984	48.921
Dyn. G&S	0.315	7.950	0.020	0.360	0.571	27.759
Dyn. MC	0.217	16.392	0.020	0.360	0.441	47.128
Dyn. MC2	0.036	23.045	0.030	0.360	0.041	106.100
LS	0.044	11.298	0.020	0.360	0.061	41.270
Loose	0.125	10.323	0.120	0.360	0.151	43.503

Methode	avg. speedup up	avg. speedup rt	avg. speedup total
Dyn. G&S	6.137	1.489	1.666
Dyn. MC	1.548	0.775	0.785
Dyn. MC2	9.333	0.551	0.565
LS	43.931	1.048	1.214
Loose	2.688 / 15.464	1.231 / 1.147	1.248 / 1.318

Abbildung 6.2: Ergebnisse Testszene 2 - BART Museum Scene

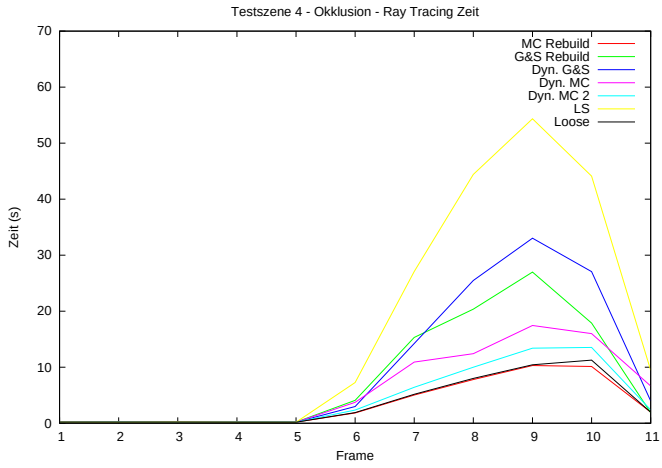


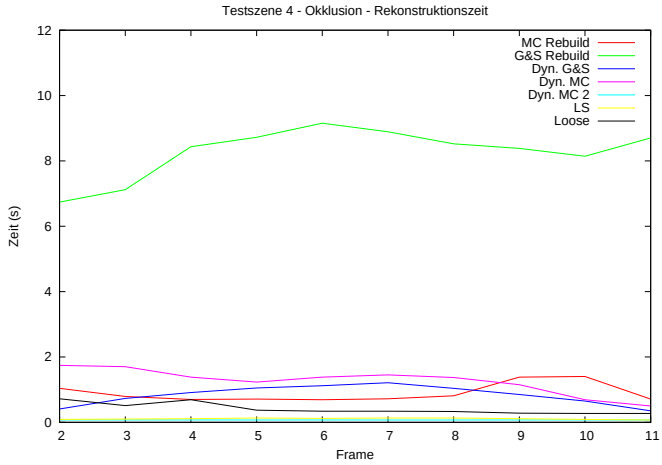


Methode	avg. up	avg. rt	best up	best rt	worst up	worst rt
MC Rebuild	0.893	6.199	0.680	0.941	1.422	9.303
G&S Rebuild	7.478	22.298	6.039	1.312	8.081	39.777
Dyn. G&S	0.907	11.420	0.370	0.751	1.182	18.686
Dyn. MC	1.255	8.941	0.501	1.051	1.743	12.238
Dyn. MC2	0.066	8.118	0.060	1.021	0.071	11.546
LS	0.113	17.613	0.070	0.751	0.141	33.218
Loose	0.404	3.182	0.251	0.511	0.761	4.927

Methode	avg. speedup up	avg. speedup rt	avg. speedup total
Dyn. G&S	8.245	1.953	2.416
Dyn. MC	0.712	0.693	0.696
Dyn. MC2	13.530	0.764	0.867
LS	66.177	1.266	1.680
Loose	2.210 / 18.510	1.948 / 7.008	1.978 / 8.303

Abbildung 6.3: Ergebnisse Testszene 3 - Falling Triangles

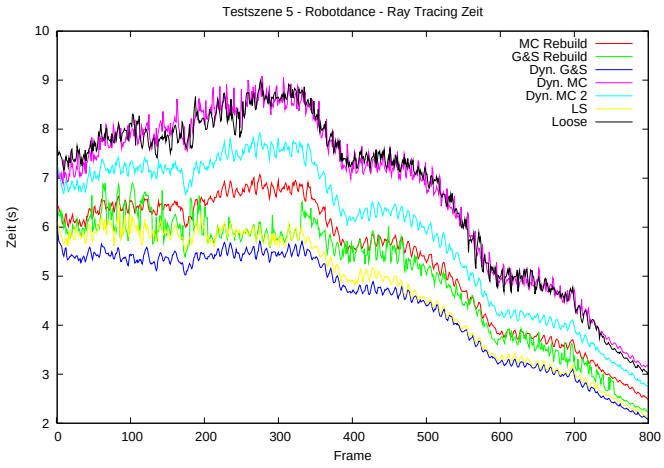


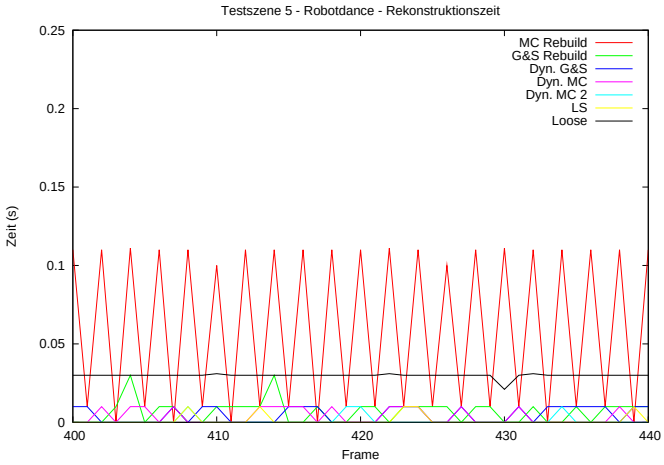


Methode	avg. up	avg. rt	best up	best rt	worst up	worst rt
MC Rebuild	0.896	3.461	0.691	0.190	1.402	10.305
G&S Rebuild	8.281	7.959	6.740	0.200	9.153	26.979
Dyn. G&S	0.833	9.794	0.351	0.190	1.212	33.028
Dyn. MC	1.261	6.197	0.501	0.190	1.743	17.455
Dyn. MC2	0.070	4.460	0.070	0.190	0.070	13.540
LS	0.108	17.060	0.071	0.190	0.130	54.358
Loose	0.413	3.611	0.270	0.190	0.721	11.266

Methode	avg. speedup up	avg. speedup rt	avg. speedup total
Dyn. G&S	9.941	0.813	1.528
Dyn. MC	0.711	0.558	0.410
Dyn. MC2	12.800	0.776	0.962
LS	76.676	0.467	0.946
Loose	2.169 / 20.051	0.958 / 2.204	1.083 / 4.036

Abbildung 6.4: Ergebnisse Testszene 4 - Okklusion





Methode	avg. up	avg. rt	best up	best rt	worst up	worst rt
MC Rebuild	0.058	5.351	0.000	2.484	0.141	7.07
G&S Rebuild	0.010	5.029	0.000	2.243	0.130	6.91
Dyn. G&S	0.005	4.451	0.000	2.083	0.031	6.018
Dyn. MC	0.003	6.735	0.000	3.144	0.011	9.083
Dyn. MC2	0.001	5.956	0.000	2.744	0.010	7.921
LS	0.001	4.710	0.000	2.123	0.011	6.45
Loose	0.030	6.747	0.020	3.014	0.090	8.993

Methode	avg. speedup up	avg. speedup rt	avg. speedup total
Dyn. G&S	19.200	1.130	1.150
Dyn. MC	19.333	0.795	0.803
Dyn. MC2	58.000	0.898	0.908
LS	96.000	1.068	1.088
Loose	1.933 / 0.333	0.793 / 0.745	0.798 / 0.744

Abbildung 6.5: Ergebnisse Testszene 5 - Robotdance

6.3 Diskussion und Fazit

Im folgenden werden die Ergebnisse der Testszenen einzeln diskutiert und abschließend ein Gesamtfazit gezogen.

6.3.1 Diskussion

- *Testszene 1 - BART Kitchen Scene*: In dieser Szene fällt vor allem der deutliche Unterschied zwischen den auf Goldsmith und Salmons Verfahren basierenden Methoden sowie den Median-Cut BVHs auf. Dies ist vor allem dadurch bedingt, dass die Szene, die denkbar schlechteste Anordnung für letztere aufweist. Vor allem die großen, die Szene umschließenden Primitive sind hier der größte Flaschenhals, da sie die Hierarchie quasi verstopfen, sowie die völlig irreguläre Verteilung der Objekte, was im Vergleich deutlich mehr Schnitttests erfordert. Vergleicht man jedoch nur MC Rebuild, Dyn. MC und Dyn. MC2 miteinander, sowie G&S Rebuild, Dyn. G&S und LS, so zeigen sich sehr große Ähnlichkeiten. Das spricht für die dynamischen Hierarchien, da zu erwarten gewesen wäre, dass diese insgesamt ein schlechteres Ergebnis liefern. Die schwächeren Zeiten des Dyn. MC2 sind nicht durch die Anwendung der Lazy Evaluation zu begründen, sondern eher durch die damit bedingte schwierigere Traversierung, was bei einer schwächeren Hierarchie natürlich dementsprechend höher ins Gewicht fällt. Der Vorteil durch die bessere Cacheausnutzung scheint damit leider in keinem Vergleich zu stehen. Die größeren Objekte der Szene sind es auch, die die starken Schwankungen bei Dyn. MC hervorrufen, da diese am ehesten beim Rebalancing ausgewählt werden und somit starke Änderungen in der Ausgestaltung der Hierarchie hervorrufen.

  In der Rekonstruktionsphase zeigen die dynamischen Bounding Volume Hierarchien deutliche Vorteile. Geschwindigkeitssteigerungen zwischen dem durchschnittlich dreieinhalbfachen und 22fachen konnten erreicht werden. Es zeigt sich auch ein deutlicher Unterschied zwischen den verschiedenen Refitting-Methoden. Da sich nur sehr wenige Objekte in dieser Szene bewegen, ist ein sequentielles Refitting wie das der Dyn. MC Methode natürlich gegenüber einem Complete Refit bei Dyn. MC2 bevorteilt.

  Betrachtet man die auf Goldsmith und Salmon basierenden Varianten, so schneiden diese in der Ray Tracing Phase nahezu identisch ab. Dies liegt zum einen daran, dass sich die beste Hierarchie für diese Szene

ergab, wenn die Objektreihenfolge aus der Szenenbeschreibung übernommen wurde, so dass alle drei Methoden die gleiche Ausgangsbasis aufweisen, und zum anderen ist der Großteil der Szene statisch, wodurch sich die Qualität der Szene im Laufe der Animation nicht allzu stark verändert. Betrachtet man jedoch die Rekonstruktionsphase, so zeigen sich immense Unterschiede. Durch Verwendung der Dyn. G&S oder LS Strategie, konnte diese um den Faktor 100 beschleunigt werden.

Die Loose Bounding Volume Hierarchy schneidet bei diesem Test in einem sehr guten Mittelfeld ab. Dies hat vor allem zwei Gründe. Die Verbesserung gegenüber den Median-Cut Varianten basiert auf dem besseren Umgang mit großen Objekten, was die benötigte Schnitttestanzahl deutlich reduziert, auf der anderen Seite ist die Szene bereits zu komplex für dieses Verfahren, so dass es in vielen Bereichen zum „Teapot in the stadium“ Problem kommt. Eine statistische Analyse zeigt, dass im Durchschnitt jeder Blattknoten 100 Objekte enthält, mit einem Maximalwert von 1339 Objekten. Basierend auf diesen Zahlen wären theoretisch, noch deutlich schlechtere Werte zu erwarten gewesen, aber da ein Großteil der versendeten Strahlen auf große Primitive, wie Boden, Tischplatte oder Schrankwände trifft, konnte so ein schlechteres Ergebnis verhindert werden. In der Rekonstruktionsphase zeigt sich die Loose BVH zwar als langsamste der dynamischen Methoden, aber noch immer deutlich besser als ein kompletter Neuaufbau, mit Geschwindigkeitssteigerungen zwischen dem dreieinhalbfachen, bzw. 22fachen. Dies hat vor allem den Grund, dass bei der Loose BVH kein Unterschied in den benötigten Berechnungen zwischen dynamischen und statischen Objekten besteht, sondern alle gleich behandelt werden. So gesehen, musste der Rest nur mit fünf dynamischen Objekten zurechtkommen, während es bei der Loose BVH ca. 110.000 waren. Vor diesem Hintergrund ist dies ein sehr zufrieden stellender Wert.

- *Testszene 2 - BART Museum Scene*: Für die Ray Tracing Phase fällt auf, dass die Methoden relativ ähnlich abschneiden. Dies liegt vor allem daran, dass abgesehen vom Boden und den Bildern an den Wänden, die Objektgröße relativ konstant ist. Dyn. G&S zeigt sich hier sogar als schnellste Methode, was zunächst durch den verbesserten Initialaufbau bedingt ist, aber auch die Qualität der Aktualisierung deutlich zeigt.

  Dyn. MC weist ein paar Schwächen auf, bedingt durch die starr alternierenden Achsen, sowie ein Problem, welches entsteht, wenn sich die Dreieckspatches zusammenziehen. Da sich die Oberflächen der Hüllvolumen dabei verkleinern, werden keine strukturellen Änderungen vor-

genommen, was aber zu größeren Überlappungen führen kann.

Dyn. MC2 zeigt einen deutlichen Anstieg der benötigten Renderingzeit, sobald sich der Zylinder aus rotierenden Dreieckspatches ausdehnt, da die Ausdehnung lediglich entlang der y-Achse verläuft. Dies hat zur Folge, dass das zweite Qualitätskriterium nicht anschlägt, und das oberflächenbasierte QK eine relativ starke Ausdehnung der Hüllvolumen erlaubt, welche proportional zur Renderingzeit verläuft. In der Nähe von Frame 100 findet dann ein nahezu kompletter Neuaufbau statt, wodurch sich die Renderingzeit knapp halbiert. Ein ähnliches Phänomen findet sich später in der Animation noch einmal.

Beim Local Sort Verfahren zeigt sich dieser Effekt bei weitem nicht so dramatisch, was vor allem daran liegen dürfte, dass die Markierungen für einen Neuaufbau, nicht bei den Bad Nodes selbst, sondern ihren Vätern gesetzt wird. Dadurch findet eine Rekonstruktion statt, bevor der Strahl überhaupt in die kritischen Bereiche gelangt.

Die Loose BVH zeigt ebenfalls exzellente Ergebnisse und ist sogar etwas schneller als die Referenzmethoden. Im Bereich um Frame 50 wird noch einmal das bereits angesprochene „Teapot in the stadium“ Problem deutlich. Sobald jedoch die Verteilung zunimmt, verbessert sich die Situation dramatisch.

Für die Rekonstruktionszeiten zeigen sich erneut sehr gute Leistungen für die dynamischen BVH. Allesamt bleiben sie im Durchschnitt unter einer Sekunde, was interaktive Frameraten erlauben würde, z.B. durch Parallelisierung.

- *Testszene 3 - Falling Triangles*: Die geringe Überlappung zwischen den Objekten und deren uniforme Größe führt dazu, dass die Median-Cut Verfahren deutlich besser abschneiden, da für sie die Szene relativ ideal ist. Allerdings gilt das auch für den Aufbau, weswegen MC Rebuild oft bessere Ergebnisse als die dynamischen Varianten liefert. Dyn. G&S und LS profitieren vor allem von ihrer Möglichkeit zu Beginn eine bessere Hierarchie aufbauen zu können. Bei letzterem verlangsamt sich natürlich erneut die für das Ray Tracing benötigte Zeit durch das QK.

  Ebenfalls ideal ist diese Szene für die Loose BVH. Ähnlich den Median-Cut Verfahren kann eine nahezu optimale Unterteilung vorgenommen werden. Hinzu kommt, dass, bedingt durch den Blickwinkel, deutlich mehr Primitive bereits in frühen Schritten von der weiteren Betrachtung ausgeschlossen werden können, da die Dreiecke, welche sich noch an der Decke oder bereits am Boden befinden, einen entsprechend geringen Raumwinkel vom Betrachter aus aufweisen, wodurch die spatiale

Aufteilung der Loose BVH hier noch einmal ein paar Vorteile daraus ziehen kann. Des weiteren kommt es vor, dass mehr als ein Dreieck in einem Blattknoten landet. In großer Menge ist dies selbstverständlich ein großer Nachteil, wie in Testszene 1. Bis zu einer gewissen Anzahl an Primitiven kann es jedoch sogar die Traversierung beschleunigen.

In der Rekonstruktionsphase weisen die meisten Methoden relativ ähnliche Werte auf. Lediglich G&S Rebuild liegt weit über dem Durchschnitt, weil die Hierarchie sehr unausgeglichen ist. Dies führt nicht nur zu schlechten Ray Tracing Zeiten, sondern ebenso dazu, dass die Einfügeprozedur bei vielen Objekten deutlich über den üblichen $\log n$ Schritten liegt.

Für die Dyn. G&S Methode wäre eigentlich ein schlechterer Wert zu erwarten gewesen, da sich die Objekte in der Szene sehr schnell bewegen, was zur Folge hat, dass sie sehr hoch in der Hierarchie gereicht werden müssen, bevor sie wieder eingefügt werden können. Und auch das Qualitätskriterium hätte dementsprechend für einen nicht zu verachtenden Mehraufwand sorgen müssen. Von einem gewissen Vorteil ist allerdings, dass sich selten alle Primitive gleichzeitig bewegen, wodurch eine gewisse Kompensation stattfindet.

Die Rekonstruktionszeit der Dyn. MC Methode leidet vor allem stark an der sich ausdehnenden Szene, wodurch sich der Aufwand, die Objekte neu einzufügen, deutlich erhöht. Bedingt durch die starre Alternierung der Sortierachsen, kommt es öfters zu stärkeren Überlappungen zwischen Bruderknoten, was im Einfügeprozess zu einem sehr unausgeglichenen Baum führt. Dies hat zur Folge, dass viele teure Rebalancierungsschritte durchgeführt werden müssen, wodurch sich die Rekonstruktionszeit drastisch verlängert.

- *Testszene 4 - Okklusion* Die Rekonstruktionsphase ist bei dieser Szene nahezu identisch mit der aus Testszene 3. In der Ray Tracing Phase zeigen sich jedoch deutliche Unterschiede, vor allem bei den G&S Varianten. Erstens sieht man, dass der betrachtete Bereich in der G&S Rebuild Methode etwas besser aufgebaut zu sein scheint, als bei Dyn. G&S. Interessanter ist jedoch, dass sich die Local Sort Variante derart abspaltet. Dies lässt sich damit begründen, dass die BVH in der ersten Hälfte der Animation nicht in ihrer Struktur verändert wurde, wodurch sie sich stark verschlechtert hat. Findet nun ein Neuaufbau statt, so werden zwar alle Bad Nodes aus dem entsprechenden Teilbereich entfernt, aber die Knoten, aus denen die Hierarchie neu aufgebaut wird, sind voraussichtlich auch nah an der Grenze zum Neuaufbau, so dass sich eine

sehr suboptimale Beschleunigungsstruktur ergibt. Mit dem veränderten Blickwinkel sinkt auch der Vorteil der Loose BVH gegenüber den Median-Cut Varianten, so dass sich jetzt nahezu gleiche Ray Tracing Zeiten ergeben.

Der Vorteil der Lazy Evaluation zeigt sich natürlich in der ersten Hälfte der Animation. Auch wenn diese Szene ein relativ konstruiertes Beispiel ist, so zeichnet sich doch ab, dass sich in Szenen mit viel Okklusion ihre Anwendung lohnen kann.

- *Testszene 5 - Robotdance* Betrachtet man die reinen Rekonstruktionszeiten, so könnte man meinen, dass die dynamischen Methoden unglaublich gut abschneiden. Allerdings sollte man sich dabei vor Augen halten, dass sich die Ergebnisse im Millisekunden-Bereich bewegen, so dass diese Werte leider nicht allzu aussagekräftig sind. Die Schwankungen in der MC Rebuild Variante sind voraussichtlich auf Speicherallokierungsprobleme zurückzuführen, welche mit etwas vorsichtigerem Umgang voraussichtlich noch zu verhindern wären, indem bspw. nicht die komplette BVH gelöscht und neu aufgebaut würde, sondern der bereits belegte Speicher genutzt würde, um die Hüllvolumen neu anzupassen und man die Objektzeiger in den Blattknoten entsprechend aktualisieren würde. Was allerdings schwieriger würde, wenn die Objektanzahl nicht konstant bleibt. Die Ray Tracing-Zeiten weisen einen sehr ähnlichen Verlauf auf. Da sich mit 1.600 Objekten nur recht wenig bewegte Objekte in der Szene befinden, werden sehr ähnliche Hierarchien erstellt, von den einzelnen Verfahren. Das schlechtere Abschneiden der Dyn. MC und Loose BVH Methode lässt sich abermals durch die starre Achsenaufteilung begründen, wodurch ca. ein Drittel mehr Schnitttests benötigt werden, als bei den anderen Varianten.

  Diese Szene zeigt vor allem den Vorteil lokaler Beschleunigungsstrukturen. Denn obwohl alle 624.100 Primitive nahezu durchgängig animiert sind, erlaubt die Anwendung der lokalen Koordinatensysteme die Hierarchie in Echtzeit zu aktualisieren.

6.3.2 Fazit

Die Testergebnisse haben gezeigt, dass es mittels der in dieser Arbeit präsentierten Methoden möglich ist die Rekonstruktionsphase durch adaptive Anpassung der Beschleunigungsstrukturen im Vergleich zu einem kompletten Neuaufbau drastisch zu senken. Dabei wurde in vielen Fällen deutlich, dass die dynamischen BVHs oft immer noch gleiche Qualität aufwiesen, häufig

sogar einen Neuaufbau übertreffen konnten. Dies trifft insbesondere auf Dynamic Goldsmith and Salmon, Local Sort und Loose Bounding Volume Hierarchy zu. Bei der Dynamic Median-Cut und Dynamic Median-Cut 2 Variante war dies leider nicht immer der Fall, insbesondere in Szenen mit sehr viel Dynamik, wobei die hier getesteten Szenen Extrembeispiele darstellen.

Lazy Evaluation kann zwar ein sehr starkes Konzept sein, ist jedoch mit großer Vorsicht einzusetzen. Ebenso wie Qualitätskriterien, welche sehr leicht die Ray Tracing Phase zu Gunsten einer beschleunigten Rekonstruktionsphase drastisch verlängern können. In stärker optimierten Ray Tracing Systemen könnte dies jedoch von größerem Vorteil sein, da sich dort das Verhältnis zwischen Rekonstruktions- und Ray Tracing Phase mehr Richtung ersterem verschieben wird, was eine schnelle Aktualisierung umso wichtiger macht.

In Szenen mit viel statischer Geometrie und weniger dynamischen Objekten hat sich ein sehr deutlicher Vorteil dynamischer BVHs gezeigt. Viele der bisher existierenden Ansätze lösten dieses Problem, indem sie die Szene vorher in statische und dynamische Bereiche unterteilten und unabhängig voneinander bearbeiteten. Dies hat jedoch entscheidende Nachteile gegenüber den hier vorgestellten Varianten, denn zum einen kann sich die benötigte Ray Tracing Zeit dadurch nahezu verdoppeln, und zum anderen ist ein entsprechendes Vorwissen über die Szene vonnöten, was jedoch nicht immer gegeben ist.

Sehr deutlich hat sich auch die Szenenabhängigkeit der verwendeten Verfahren gezeigt. Für nahezu uniform verteilte Objekte gleicher Größe, eignet sich ein Median-Cut Ansatz sehr gut, während das Problem nicht-uniformer Objektdistribution und starker Größenunterschiede besser von auf Goldsmith und Salmons Verfahren basierenden Methoden gelöst wird. Möchte man szenenunabhängig bleiben, so bieten sich eher Ansätze an, die auf spatiale Aufteilungen setzen, wie etwa die Loose Bounding Volume Hierarchy, welches durchgängig sehr gute Ergebnisse geliefert hat, und lediglich am „Teapot in the stadium“ Problem leidet. Dafür konnten, dank der geringeren Komplexität selbst 150.000 Objekte in weniger als einer Sekunde verarbeitet werden.

6.4 Testergebnisse Ray Slope-Schnitttest

Die Ray-Slope Methode wurde mit drei anderen Methoden verglichen. Erstens der Methode nach Kay/Kajiya [KK86] (`standard` im folgenden genannt), wobei jede Seite der Box explizit getestet wurde, der verbesserten Variante nach Smits [Smi98] (`smits`) bzw. Williams [WBMS05] (`smits_mul`) und der Plücker-Methode von Mahovsky und Wyvill [MW04] (`pluecker`),

welche vorab den Strahl entsprechend seiner Richtungskomponenten klassifiziert und explizit die vorberechneten Plücker-Koordinaten speichert, welches die schnellste ihrer vorgestellten Methoden darstellt, was unsere Untersuchungen ebenfalls bestätigten.

Um den Algorithmus zu testen, wurden per Zufall 500.000 Strahl/Box-Pärchen erstellt, welche in einem Array gespeichert wurden, so dass jeder Methode die gleiche Datenbasis zur Verfügung stand. Das Verhältnis zwischen Treffer und Nicht-Treffer variiert dabei zwischen 0%, 50% und 100%, um abzusichern, dass vergleichbare Zeiten für eine niedrige Trefferfrequenz, eine im mittleren Bereich, sowie eine hohe Trefferfrequenz vorliegen. Dies ist wichtig, da einige der Algorithmen, verschiedene Abbruchkriterien enthalten, welche sie Teile der Berechnungen überspringen lassen, falls der Strahl die Box verfehlt. Jedes Strahl/Box Päarchen wurde jeweils 100 mal hintereinander getestet, um Cache-Vor- oder -Nachteile zu verhindern, welche anderweitig auftreten könnten. Dies bedeutet, dass jede Methode insgesamt 50.000.000 Strahl/Box-Schnitttests pro Durchlauf durchgeführt hat. Die Tests wurden sowohl mit floating-point als auch mit doppelter floating-point Präzision durchgeführt. Jedes Ergebnis eines Strahl/Box Schnitttest wurde vor der Zeitmessung auf Korrektheit überprüft, indem die Ergebnisse mit den Ergebnissen der `standard` Variante verglichen wurden. Die Tests selbst wurden auf einem Pentium4 2.4GHz Prozessor mit 1GB RAM durchgeführt.

Tabelle 6.1 zeigt einige der Testergebnisse. Dabei sollten nur Methoden verglichen werden, welche auch das Gleiche berechnen. So sollte eine Methode, welche eine Schnittpunktdistanz berechnet, nicht unbedingt mit einer anderen Methode verglichen werden, welche keine berechnet, oder Techniken, welche Divisionen für die Berechnung der Schnittdistanz verwenden, sollten nicht mit Techniken verglichen werden, welche dafür Multiplikationen verwenden und so fort.

Die verschiedenen Typen sind:

- `slope`: Die neue in dieser Arbeit präsentierte Methode. Sie liefert `true`, falls der Strahl die AABB schneidet, sonst `false`. Sie berechnet keine Schnittdistanz.

- `pluecker`: Die Plücker-Variante von Mahovsky und Wyvill bestimmt, ob ein Strahl, die Box schneidet oder nicht, berechnet jedoch keine Schnittdistanz. Es werden sowohl die benötigten Plücker-Koordinaten, sowie die Klassifizierung der Strahlrichtung vorberechnet.

- `standard`: Alle sechs Seiten der AABB werden auf Schnitte getestet

und eine Schnittdistanz ermittelt. Es findet ein expliziter Test statt, ob eine der Richtungskomponenten null ist.

- `smits`: Im Prinzip dieselbe Methode wie `standard`, aber es wird expliziter Nutzen aus der IEEE Arithmetik gezogen, um auf den Test für Richtungskomponenten, welche null sind, verzichten zu können.
- `int`: Falls dass Suffix `int` hinzugefügt wurde, wird ein separater Test für die Schnittdistanz durchgeführt.
- `cls`: Das Suffix `cls` gibt an, dass der Strahl bei Erzeugung vorklassifiziert wurde (MMM, etc.). Diese Klassifizierung wird einmalig berechnet, wenn der Strahl erzeugt wird und gespeichert, so dass nur drei der Ebenen einer AABB auf Schnitte getestet werden müssen.
- `div`: Zur Berechnung des Strahlparameters am Schnittpunkt werden Divisionsberechnungen durchgeführt, falls das Suffix `div` vorliegt.
- `mul`: Die Divisionen bei der Berechnung des Strahlparameters wird durch eine einfachere Multiplikationen ersetzt, indem die Kehrwerte der Strahlrichtungen bei Erzeugung des Strahles gespeichert werden.

Die Ergebnisse aus Tabelle 6.1 belegen, dass die Ray Slope-Methode (`slope`) schneller als die schnellste der Vergleichsmethoden (`pluecker`) in 6 von 6 Tests war. Die Performanz war dabei bis zu 15% höher (4.81 Sekunden gegenüber 5.69 Sekunden), im Vergleich zur zweitschnellsten Methode. Was ein sehr zufriedenstellendes Ergebnis darstellt, insbesondere in Hinsicht darauf, dass beim Ray Tracing ca. 90% der Zeit auf die Traversierung der Hierarchie verfallen.

Um auch in einem realen Ray Tracing System bestehen zu können, wurde die Methode auch in den in dieser Arbeit verwendeten Ray Tracer implementiert und zum Test drei Bilder der Kitchen-Szene aus dem BART-Paket [LAM01a] ausgewählt und gerendert in einer Auflösung von 800×600, diese sind in Abbildung 6.6 dargestellt. Die Ergebnisse finden sich in Tabelle 6.2. Auch hier zeigt sich die Ray Slope-Methode als beste, in allen drei Tests, mit einer durchschnittlichen Beschleunigung von ca. 9%.

	double precision			single precision		
Hit-ratio	0.0	0.5	1.0	0.0	0.5	1.0
slope	8.58	8.92	9.22	4.58	4.69	4.81
slopeint_div	8.67	9.42	10.17	4.49	5.44	6.38
slopeint_mul	8.69	9.27	9.88	4.48	4.81	5.13
pluecker	8.99	9.28	9.63	4.61	5.14	5.69
plueckerint_div	9.06	9.92	10.80	4.52	5.56	6.63
plueckerint_mul	9.05	9.53	10.00	4.50	4.81	5.11
standardint_div	14.03	16.78	19.55	9.67	11.72	13.80
standardint_mul	11.58	12.98	14.42	6.03	7.17	8.30
smitsint_div	13.97	16.64	19.28	9.25	11.28	13.31
smitsint_div_cls	12.70	13.77	14.81	8.81	10.45	12.08
smitsint_mul	11.38	12.61	13.86	5.84	6.86	7.88
smitsint_mul_cls	10.14	10.53	10.91	6.09	6.88	7.66

Tabelle 6.1: Test Ergebnisse auf einem Pentium4 2.4 GHz. Die Zeiten sind in Sekunden angegeben.

BART: Kitchen Scene Frame	1	289	679
slope	27.44	32.357	35.001
slopeint_div	30.123	36.082	39.116
slopeint_mul	28.15	33.338	36.283
pluecker	29.873	35.331	38.446
plueckerint_div	35.711	42.642	46.467
plueckerint_mul	33.338	39.647	43.222
standardint_div	57.032	67.817	72.855
standardint_mul	42.401	49.802	53.598
smitsint_div	54.228	64.533	69.239
smitsint_div_cls	47.648	57.022	61.158
smitsint_mul	38.776	45.746	49.101
smitsint_mul_cls	35.26	42.07	45.325

Tabelle 6.2: Schnitttestergebnisse für die BART: Kitchen Testszene. Die Zeiten sind in Sekunden angegeben

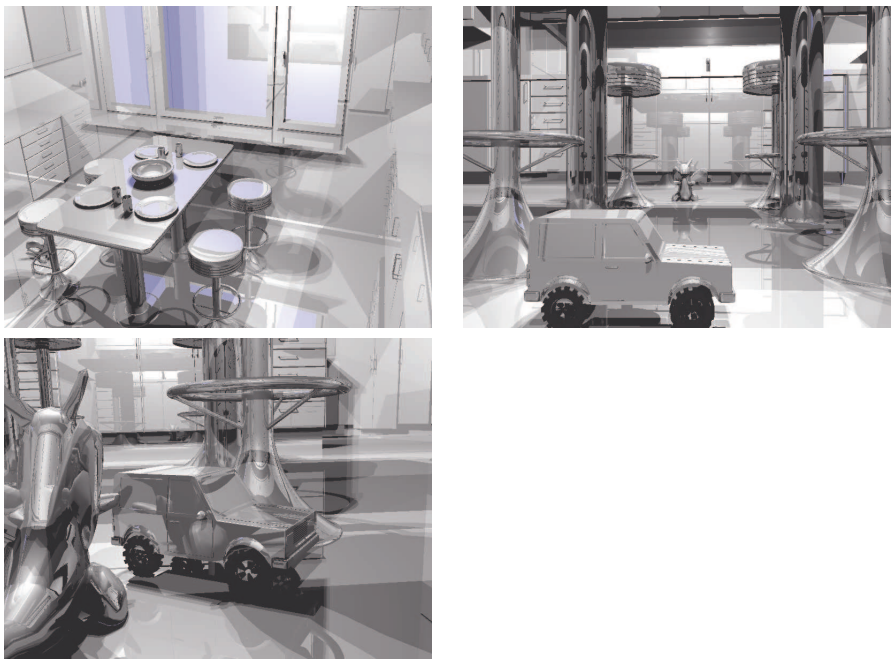


Abbildung 6.6: BART Kitchen Szene Frame 1, 289 und 679, von links oben nach rechts unten

Kapitel 7

Zusammenfassung und Ausblick

In der vorliegenden Diplomarbeit wurden Entwürfe und Umsetzungen verschiedener Verfahren zur Beschleunigung der Rekonstruktionsphase für Bounding Volume Hierarchien in Zusammenhang mit dynamischen Szenen im Ray Tracing Kontext betrachtet. Dazu wurden zunächst die Grundlagen des Ray Tracings vorgestellt sowie ein Abriss über die wichtigsten Beschleunigungstechniken gegeben.

Nach einem Überblick über die momentan existierenden Methoden auch dynamische Szenen mittels Ray Tracing zu handhaben, wurden in Hinblick darauf Bounding Volume Hierarchien detaillierter untersucht. Neben einer Zusammenfassung der gebräuchlichsten Methoden zu ihrer Erstellung und deren Traversierung, wurden Anforderungen für Qualitätskriterien präsentiert, die Aufschluss über die Güte bzw. die Veränderung einer BVH geben sollen, sowie daran anschließend unterschiedliche Heuristiken vorgestellt und diskutiert.

Darauf aufbauend wurden in dieser Arbeit verschiedenste Verfahren entwickelt, welche die Rekonstruktionszeit von BVHs verringern sollen. Dabei hat sich gezeigt, dass eine Fülle an Möglichkeiten existiert, die, je nach Anwendungsbereich, verglichen mit einem kompletten Neuaufbau, sehr gute Ergebnisse liefern. Dies zeigt sich insbesondere in komplexeren Szenen.

Des weiteren wurde ein neuer Schnitttest für Strahlen gegen achsenparallele Boxen präsentiert, der sich die Steigung eines Strahles zu nutzen macht, um viele der benötigten Informationen vorab zu berechnen und somit besonders für die Traversierung von BVHs geeignet ist. In den durchgeführten Testreihen erwies sich dieser, mit einer Geschwindigkeitssteigerung von bis zu 15%, als schnellste aller getesteten Varianten.

Dies ist sicher nicht das Ende der Entwicklung. Insbesondere die Umsetzung

der Verfahren für einen SIMD-optimierten Ray Tracer, welcher vielleicht bereits interaktive Frameraten in statischen Szenen erreicht, wäre interessant, genauso wie die Anwendbarkeit von Rechnerclustern oder der Einsatz der GPU zur parallelen Berechnung der Rekonstruktion und des Renderings.

Auch die Loose Bounding Volume Hierarchy liefert noch viele Erweiterungsmöglichkeiten. So könnte ein besseres Vorwissen über die Szene ein adaptives Anpassen der Beschleunigungsstruktur erlauben, was erstens die Refitting-Phase verkürzen könnte, aber vor allem den sehr hohen Speicherbedarf reduzieren würde. Es könnten noch beträchtliche Geschwindigkeitssteigerungen in der Rekonstruktionsphase erzielt werden, wenn man gewisse Einschränkungen vornehmen würde. Wenn sich Szenen nicht ausdehnen dürfen, erspart dies die Anpassung des Wurzelknotens in jeder Rekonstruktionsphase und statische Objekte würden stets im gleichen Knoten gehalten. Wenn Objekte nicht skaliert werden, würde dies die Berechnung des Einfügelevels ersparen. Interessant wäre auch die Anwendbarkeit für View-Frustum Culling, sowie für statische Szenen, da das Verfahren einige der Schwachpunkte einer Median-Cut Hierarchie oder einer Hierarchie nach dem Verfahren von Goldsmith und Salmon umgeht. So verstopfen große Objekte die Hierarchie nicht mehr, indem sie höher in der Hierarchie gehalten werden, und können zudem frühzeitig getestet werden. Dies wäre vor allem bei einer iterativen Traversierung von Vorteil, da voraussichtlich schneller ein Schnittpunkt gefunden würde, der behilflich sein kann, weitere Teile der Hierarchie zu cullen. Leere Räume werden sehr gut ausgespart, für statische Szenen kann die starre Alternierung der Achsen durch eine intelligentere Wahl ersetzt werden. Leere Knoten würden in einem Vorverarbeitungsschritt entfernt, so dass eine optimale Traversierung stattfinden würde. Zudem sorgt die spatiale Aufteilung dafür, dass das Verfahren nicht abhängig von der Objektreihenfolge ist.

Für den Ray-Slope Schnitttest wäre es ebenfalls interessant, ob sich eine Variante entwickeln liesse, welche sich SIMD-Instruktionen zu nutzen macht und wie groß hier der Geschwindigkeitsgewinn wäre.

Es ist deutlich, dass ein Ende der Entwicklung von dynamischen Beschleunigungsstrukturen noch lange nicht in Sicht ist. Die in dieser Arbeit gefundenen Ansätze können wohl auch in anderen Gebieten weiterführend verwendet werden. Ob sie dabei einen generellen Ansatz für neue Wege des Ray Tracings bieten, wird sich zeigen.

Anhang A

Testszenen

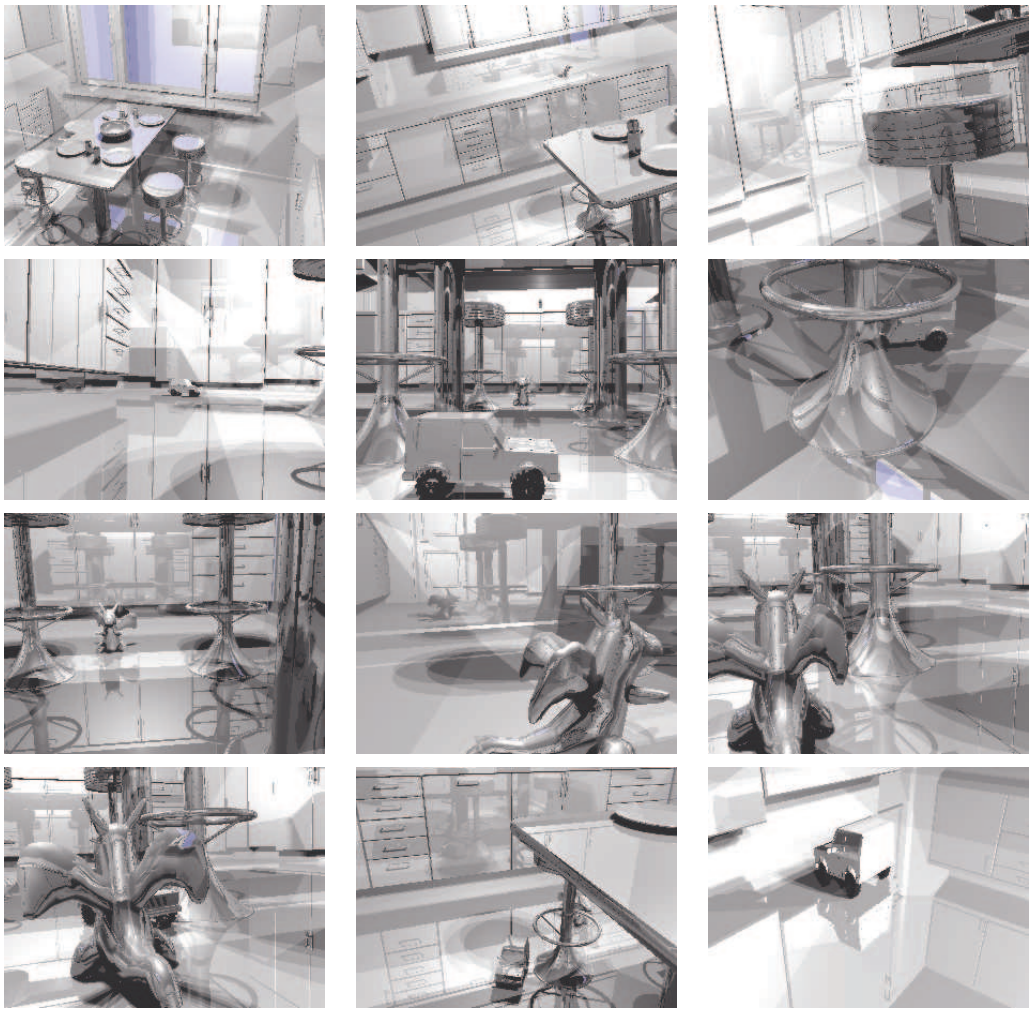


Abbildung A.1: Testszene 1 BART Kitchen Scene. Die Animation beginnt oben links und sollte von links nach rechts und von oben nach unten gelesen werden.

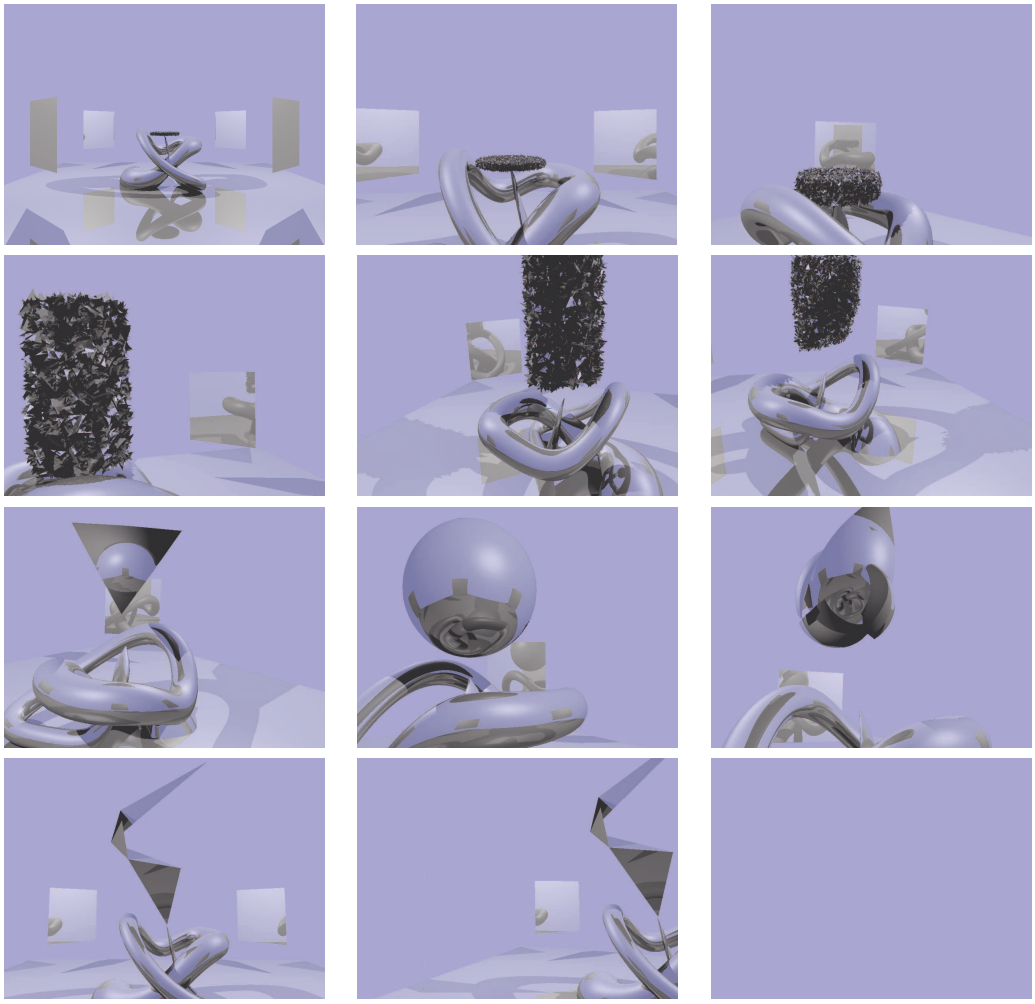


Abbildung A.2: Testszene 2 BART Museum Scene. Die Animation beginnt oben links und sollte von links nach rechts und von oben nach unten gelesen werden.

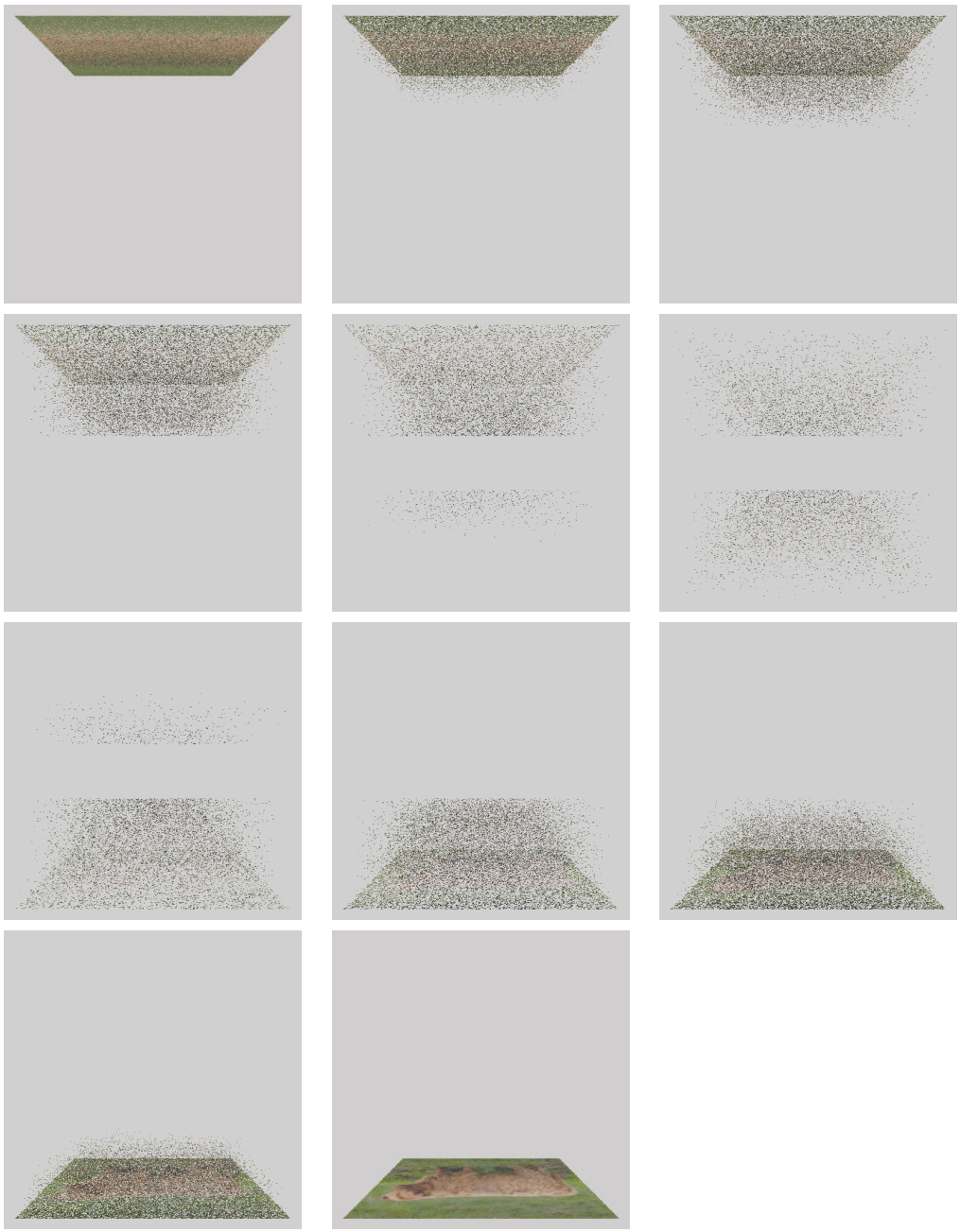


Abbildung A.3: Testszene 3 Falling Triangles. Die Animation beginnt oben links und sollte von links nach rechts und von oben nach unten gelesen werden.

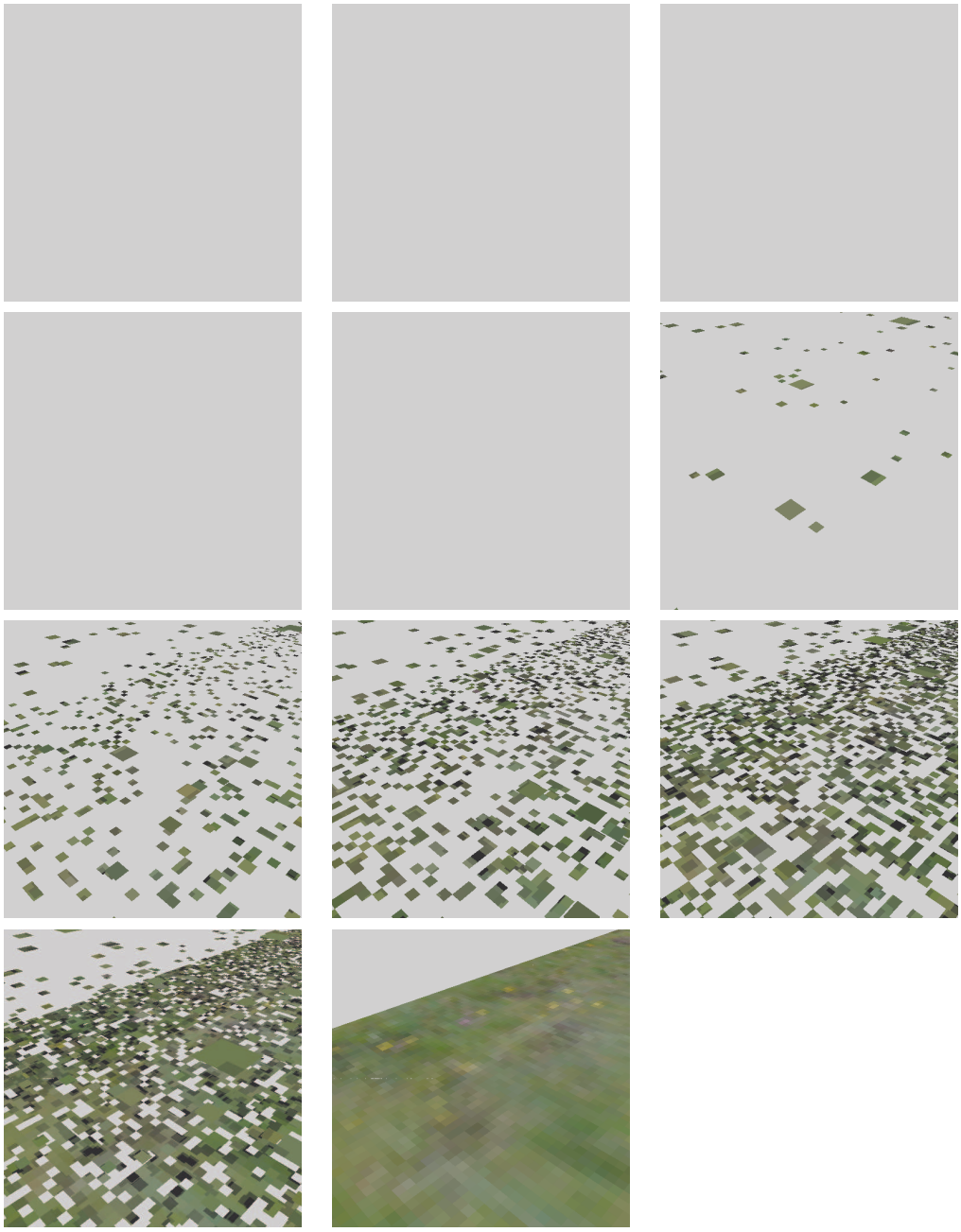


Abbildung A.4: Testszene 4 Okklusion. Die Animation beginnt oben links und sollte von links nach rechts und von oben nach unten gelesen werden.

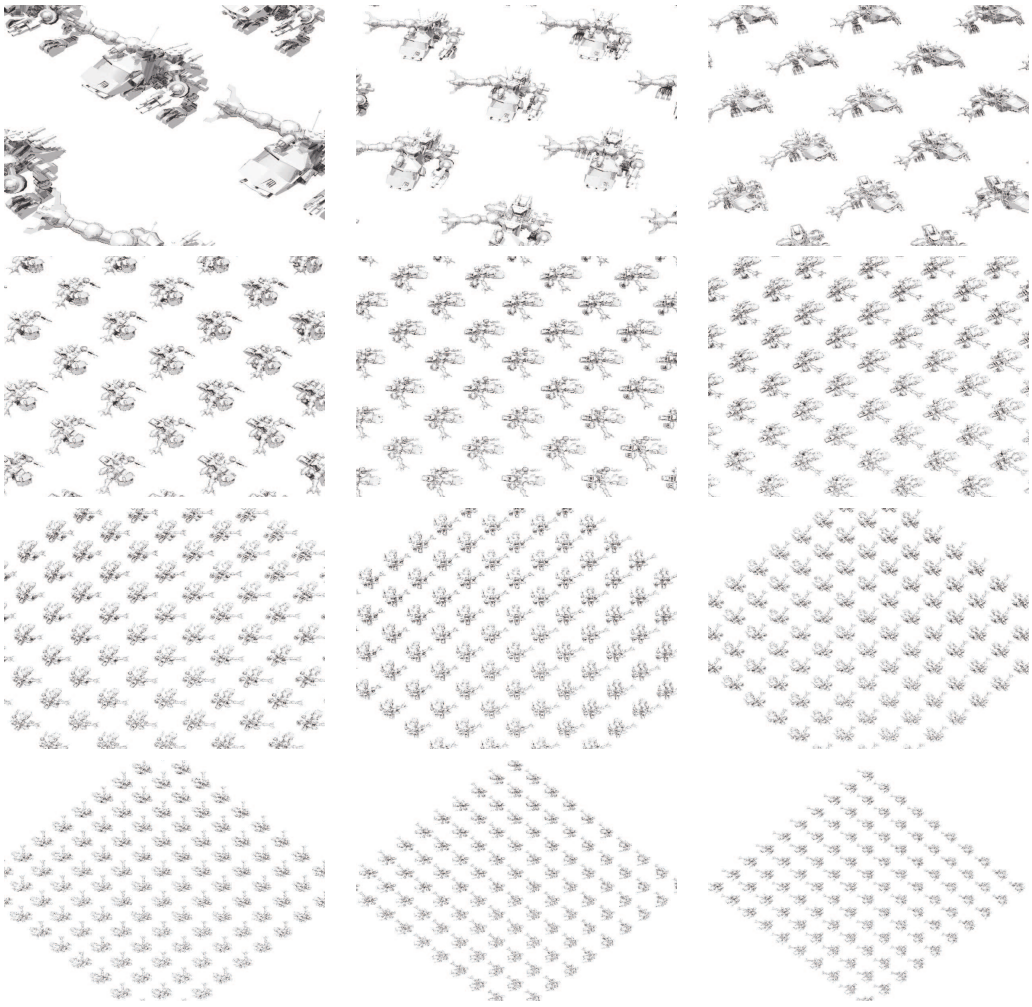


Abbildung A.5: Testszene 5 Robotdance. Die Animation beginnt oben links und sollte von links nach rechts und von oben nach unten gelesen werden.

Literaturverzeichnis

[Ada02] Adams, Jim: *Programming Role Playing Games with DirectX 8.0*. First. André LaMothe, 2002

[AK87] Arvo, James ; Kirk, David: Fast ray tracing by ray classification. In: *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1987. – ISBN 0–89791–227–6, S. 55–64

[AK89] Arvo, James ; Kirk, David: A Survey of Ray Tracing Acceleration Techniques. In: Glassner, Andrew S. (Hrsg.): *An Introduction to Ray Tracing*, Academic Press, 1989, S. 227–238

[Ama84] Amanatides, J.: Ray Tracing with cones. In: *Computer Graphics* 18 (1984), July, Nr. 3, S. 129–135

[Amd67] Amdahl, Gene: Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In: *AFIPS Conference Proceedings* Bd. 30, 1967, S. 483–485

[App68] Appel, Arthur: Some techniques for shading machine renderings of solids. In: *AFIPS 1968 Spring Joint Computer Conference* Bd. 32, 1968, S. 37–45

[Arv90] Arvo, James: Ray Tracing with Meta-Hierarchies. In: *SIGGRAPH '90: Advanced Topics in Ray Tracing course notes* Bd. 24, 1990

[BDT99] Bala, Kavita ; Dorsey, Julie ; Teller, Seth: Radiance interpolants for accelerated bounded-error ray tracing. In: *ACM Transactions on Graphics* 18 (1999), Nr. 3, S. 213–256. – ISSN 0730–0301

[Ben75] Bentley, Jon L.: Multidimensional binary search trees used for associative searching. In: *Communications of the ACM* 18 (1975), September, Nr. 9

[Ben88] Bentley, Jon L.: *More Programming Pearls*. First. Addison-Wesley, Januar 1988

[Ber97] van den Bergen, Gino: Efficient Collision Detection of Complex Deformable Models using AABB Trees. In: *Journal of Graphic Tools* 2 (1997), Nr. 4, S. 1–13. – ISSN 1086–7651

[BFMZ94] Bishop, Gary ; Fuchs, Henry ; McMillan, Leonard ; Zagier, Ellen J. S.: Frameless rendering: double buffering considered harmful. In: *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1994. – ISBN 0–89791–667–0, S. 175–176

[BSB+01] Brown, Joel ; Sorkin, Stephen ; Bruyns, Cynthia ; Latombe, Jean-Claude ; Montgomery, Kevin ; Stephanides, Michael: Real-Time Simulation of Deformable Objects: Tools and Application. In: *Computer Animation 2001*, 2001, S. 228–236

[CDP95] Cazals, Frédéric ; Drettakis, George ; Puech, Claude: Filtering, Clustering and Hierarchy Construction: a New Solution for Ray-Tracing Complex Scenes. In: *Computer Graphics Forum* 14 (1995), Nr. 3, S. 371–382

[Dev89] Devillers, O.: The macro-regions: an efficient space subdivision structure for ray. In: *Proceedings of Eurographics '89*, Elsevier Science Publishers, September 1989, S. 27–38

[DWWL05] Dayal, Abhinav ; Woolley, Cliff ; Watson, Benjamin ; Luebke, David P.: Adaptive Frameless Rendering. In: Deussen, Oliver (Hrsg.) ; Keller, Alexander (Hrsg.) ; Bala, Kavita (Hrsg.) ; Dutré, Philip (Hrsg.) ; Fellner, Dieter W. (Hrsg.) ; Spencer, Stephen N. (Hrsg.): *Rendering Techniques*, Eurographics Association, 2005. – ISBN 3–905673–23–1, S. 265–275

[FBG02] Fernandez, Sebastian ; Bala, Kavita ; Greenberg, Donald P.: Local illumination environments for direct lighting acceleration. In: *Proceedings of the 13th Eurographics workshop*

on Rendering. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2002. – ISBN 1–58113–534–3, S. 7–14

[FTI86] Fujimoto, Akira ; Tanaka, Takayuki ; Iwata, Kansei: ARTS: Accelerated ray-tracing system. In: *IEEE Compututer Graphics and Applications* 6 (1986), April, Nr. 4, S. 16–26. – ISSN 0272–1716

[GA93] Gargantini, Irene ; Atkinson, H. H.: Ray Tracing an Octree: Numerical Evaluation of the First Interaction. In: *Compututer Graphics Forum* 12 (1993), Oktober, Nr. 4, S. 199–210

[Gei05] Geimer, Markus: *Interaktives Ray Tracing*, Universität Koblenz-Landau, Diss., 2005

[Gla84] Glassner, Andrew: Space Subdivision for Fast Ray Tracing. In: *IEEE Computer Graphics and Applications* 4 (1984), October, Nr. 10, S. 15–22

[Gla88] Glassner, Andrew S.: Spacetime Ray Tracing for Animation. In: *IEEE Computer Graphics and Applications* 8 (1988), Nr. 2, S. 60–70. – ISSN 0272–1716

[Gla89] Glassner, Andrew S. (Hrsg.): *An Introduction to Ray Tracing*. First. Academic Press, 1989

[GS87] Goldsmith, Jeffrey ; Salmon, John: Automatic Creation of Object Hierarchies for Ray Tracing. In: *IEEE Computer Graphics and Applications* 7 (1987), May, Nr. 5, S. 14–20

[Hai89] Haines, Eric: Automatic Creation of Object Hierarchies for Ray Tracing. In: *Ray Tracing News* 1 (1989), October, Nr. 6. – http://www.acm.org/tog/resources/RTNews/html/rtnews3a.html#art6

[HG86] Haines, Eric ; Greenberg, D. P.: The Light Buffer: a Shadow Testing Accelerator. In: *IEEE Computer Graphics and Applications* 6 (1986), September, Nr. 9, S. 6–16

[HH84] Heckbert, Paul ; Hanrahan, P.: Beam tracing polygonal objects. In: *Computer Graphics* 18 (1984), July, Nr. 3, S. 119–127

[HSS00] Haber, Jörg ; Stamminger, Marc ; Seidel, Hans-Peter: Enhanced Automatic Creation of Multi-Purpose Object Hierarchies. In: *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA : IEEE Computer Society, 2000. – ISBN 0–7695–0868–5, S. 52

[Jan86] Jansen, Frederik W.: Data structures for ray tracing. In: *Proceedings of a workshop (Eurographics Seminars on Data structures for raster graphics*. New York, NY, USA : Springer-Verlag New York, Inc., 1986. – ISBN 0–387–16310–7, S. 57–73

[JW89] Jevans, D. ; Wyvill, B.: Adaptive voxel subdivision for ray tracing. In: *Proceedings of Graphics Interface*, 1989, S. 164–172

[Kaj86] Kajiya, James T.: The rendering equation. In: *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1986. – ISBN 0–89791–196–2, S. 143–150

[Kap85] Kaplan, Michael R.: Space Tracing a constant time ray tracer. In: *State of the Art in Image Synthesis (Course Notes on ACM SIGGRAPH '85* Bd. 11, 1985, S. 149–158

[KK86] Kay, Timothy L. ; Kajiya, James T.: Ray tracing complex scenes. In: *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1986. – ISBN 0–89791–196–2, S. 269–278

[LAM01a] Lext, J. ; Assarsson, U. ; Moeller, T.: A benchmark for animated ray tracing. In: *IEEE Computer Graphics and Applications* 21 (2001), March, S. 22–31

[LAM01b] Lext, Jonas ; Akenine-Möller, Tomas: Towards Rapid Reconstruction for Animated Ray Tracing. In: *Eurographics 2001 - Short Presentations*, 2001, S. 311–318

[LAM03] Larsson, Thomas ; Akenine-Möller, Tomas: Strategies for Bounding Volume Hierarchy Updates for Ray Tracing of Deformable Models / MRTC Mälardalen Real-Time Research Centre, Mälardalen University. 2003. – Forschungsbericht. http://www.mrtc.mdh.se/index.phtml?choice=publications&id=0501

[MB90] MacDonald, David J. ; Booth, Kellogg S.: Heuristics for ray tracing using space subdivision. In: *Visual Computer* 6 (1990), Nr. 3, S. 153–166. – ISSN 0178–2789

[MF99] Müller, Gordon ; Fellner, Dieter W.: Hybrid Scene Structuring with Application to Ray Tracing / Institute of ComputerGraphics, TU Braunschweig. 1999. – Forschungsbericht. citeseer.ist.psu.edu/508223.html

[MSMP+92] McNeill, M.D.J. ; Shah, B.C. ; M-P.Hébert ; Lister, P.F. ; R.L.Grimsdale: Performance of Space Subdivision Techniques in Ray Tracing. In: *Computer Graphics Forum* 11 (1992), Nr. 4, S. 213–220

[MW04] Mahovsky, Jeffrey ; Wyvill, Brian: Fast Ray-Axis Aligned Bounding Box Overlap Tests With Plücker Coordinates. In: *Journal of Graphics Tools* 9 (2004), Nr. 1, S. 35–46

[OM87] Ohta, M. ; Maekawa, M.: Ray coherence theorem and constant time ray tracing algorithm. In: *CG International '87 on Computer graphics 1987*. New York, NY, USA : Springer-Verlag New York, Inc., 1987. – ISBN 4–431–70022–6, S. 303–314

[PBMH02] Purcell, Timothy J. ; Buck, Ian ; Mark, William R. ; Hanrahan, Pat: Ray tracing on programmable graphics hardware. In: *ACM Transactions on Graphics* 21 (2002), July, Nr. 3, S. 703–712

[PMS+99] Parker, Steven ; Martin, William ; Sloan, Peter-Pike J. ; Shirley, Peter ; Smits, Brian ; Hansen, Charles: Interactive ray tracing. In: *Symposium on Interactive 3D Graphics*, 1999, S. 119–126

[RSH00] Reinhard, Erik ; Smits, Brian ; Hansen, Chuck: Dynamic Acceleration Structures for Interactive Ray Tracing. In: *Proceedings of the 11th Eurographics Workshop on Rendering*, 2000, S. 299–306

[RSH05] Reshetov, Alexander ; Soupikov, Alexei ; Hurley, Jim: Multi-level ray tracing algorithm. In: *ACM Transactions on Graphics* 24 (2005), Nr. 3, S. 1176–1185. – ISSN 0730–0301

[RW80] Rubin, Steven M. ; Whitted, Turner: A 3-dimensional representation for fast rendering of complex scenes. In: *SIGGRAPH*

'80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques. New York, NY, USA : ACM Press, 1980. – ISBN 0–89791–021–4, S. 110–116

[SF92] Subramanian, K. ; Fussel, D.: A search structure based on k-d trees for efficient ray tracing / The University of Texas at Austin. 1992. – Forschungsbericht. citeseer.ist.psu.edu/subramanian92search.html

[SG96] Sudarsky, Oded ; Gotsman, Craig: Output-Sensitive visibility algorithms for dynamic scenes with application to virtual reality. In: *Computer Graphics Forum* 15 (1996), Nr. 3, S. 249–258

[SH93] Smits, Brian ; Haines, Eric: Faster Bounding Volume Hierarchies. In: *Ray Tracing News* 6 (1993), September, Nr. 3. – http://www.acm.org/tog/resources/RTNews/html/rtnv6n3.html#art9

[SM03] Shirley, Peter ; Morley, R. K.: *Realistic Ray Tracing*. Second. Natick, MA, USA : A. K. Peters, Ltd., 2003. – ISBN 1568811985

[Smi98] Smits, Brian: Efficiency issues for ray tracing. In: *Journal of Graphic Tools* 3 (1998), Nr. 2, S. 1–14. – ISSN 1086–7651

[Smi02] Smits, Brian: Efficient Bounding Box Intersection. In: *Ray Tracing News* 15 (2002), Nr. 1. – http://www.acm.org/tog/resources/RTNews/html/rtnv15n1.html#art4

[SWS02] Schmittler, Jörg ; Wald, Ingo ; Slusallek, Philipp: SaarCOR – A Hardware Architecture For Ray Tracing. In: *Proceedings of the conference on Graphics Hardware 2002* Saarland University, Eurographics Association, 2002. – available at http://www.openrt.de. – ISBN 1–58113–580–7, S. 27–36

[SWW+04] Schmittler, Jörg ; Woop, Sven ; Wagner, Daniel ; Paul, Wolfgang J. ; Slusallek, Philipp: Realtime ray tracing of dynamic scenes on an FPGA chip. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware*, 2004, S. 95–106

[Ulr00] Ulrich, Thatcher: Loose Octrees. In: *Game Programming Gems* Bd. 1. Mark DeLoura, 2000. – ISBN 3–8266–0923–9, S. 434–442

[Wal04] Wald, Ingo: *Realtime Ray Tracing and Interactive Global Illumination*, Universität des Saarlandes, Diss., January 2004

[WBMS05] Williams, Amy ; Barrus, Steve ; Morley, R. K. ; Shirley, Peter: An Efficient and Robust Ray-Box Intersection Algorithm. In: *Journal of Graphics Tools* 10 (2005), Nr. 1, S. 55–60

[WBS03] Wald, Ingo ; Benthin, Carsten ; Slusallek, Philipp: Distributed Interactive Ray Tracing of Dynamic Scenes. In: *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*, 2003, S. 77–86

[WBWS01] Wald, Ingo ; Benthin, Carsten ; Wagner, Markus ; Slusallek, Philipp: Interactive Rendering with Coherent Ray Tracing. In: *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001* 20 (2001), Nr. 3. – available at http://graphics.cs.uni-sb.de/ wald/Publications

[WDP99] Walter, Bruce ; Drettakis, George ; Parker, Steven: Interactive Rendering using the render cache. In: *Proceedings of the 10th EUROGRAPHICS Workshop on Rendering Techniques*, 1999, S. 235–246

[WHG84] Weghorst, Hank ; Hooper, Gary ; Greenberg, Donald P.: Improved Computational Methods for Ray Tracing. In: *ACM Transactions on Graphics* 3 (1984), Nr. 1, S. 52–69. – ISSN 0730–0301

[Whi80] Whitted, Turner: An improved illumination model for shaded display. In: *Commun. ACM* 23 (1980), Nr. 6, S. 343–349. – ISSN 0001–0782

[Woo90] Woo, Andrew: Fast ray-box intersection. In: Glassner, Andrew (Hrsg.): *Graphic Gems* Bd. 1. Academic Press, 1990, S. 395–396

[WSC+95] Whang, Kyu-Young ; Song, Ju-Won ; Chang, Ji-Woong ; Kim, Ji-Yun ; Cho, Wan-Sup ; Park, Chong-Mok ; Song, Il-Yeol: Octree-R: An Adaptive Octree for Efficient Ray Tracing. In: *IEEE Transactions on Visualization and Computer Graphics* 1 (1995), Nr. 4, S. 343–349. – ISSN 1077–2626