Universität Gesamthochschule Paderborn
Fachbereich 17 · Mathematik/Informatik

Approximative Darstellung dreidimensionaler Szenen mit randomisiertem z-Buffer

Michael Wand

Diplomarbeit im Fach Informatik

Paderborn, im März 2000

vorgelegt bei
Prof. Dr. Friedhelm Meyer auf der Heide

Inhaltsverzeichnis

Einleitung

In dieser Arbeit wird ein neues, approximatives Renderingverfahren für dreidimensionale Szenen vorgestellt, das *„randomisierte z-Buffer Verfahren“*. Das Verfahren basiert darauf, eine Menge zufälliger Punkte so aus der Szene auszuwählen, daß sie auf der ins Bild projizierten Fläche der Objekte gleichverteilt sind. Aus diesen Stichprobenpunkten wird dann ein Bild der Szene rekonstruiert. Unter schwachen Voraussetzungen kann dabei eine gute Bildqualität erreicht werden. Der Erwartungswert für den Darstellungsaufwand für eine Klasse von Szenen mit gleichmäßig auf der Einheitskugel verteilten Oberflächennormalen wächst dabei mit $O((a \cdot \log v + \log \tau) \cdot \log n)$, mit v als Auflösung des Bildes, a als gesamter projizierter Fläche (sichtbar und verdeckt), τ als Verhältnis von minimaler Sichtweite zum Szenendurchmesser und n als Anzahl von Elementarobjekten (Dreiecken). Der Aufwand ist also nur sehr schwach vom Detaillierungsgrad der Szene abhängig.

Problembeschreibung

Ein zentrales Problem in der Computergrafik ist das sog. *Walkthrough-Problem*. Dabei soll einem interaktiven Benutzer ermöglicht werden als „virtueller Besucher“ eine von einem Rechner in Echtzeit dargestellte dreidimensionale Szene zu begehen. Dazu ist eine Darstellung (*„Rendering“*) der selben Szene aus unterschiedlichen Perspektiven im Abstand weniger Sekundenbruchteile notwendig. In vielen Anwendungen ist es wünschenswert, Szenen hoher Komplexität darzustellen. Um Szenen mit wachsender Komplexität noch effizient behandeln zu können, werden *outputsensitive* Algorithmen gesucht, deren Laufzeit im wesentlichen von der Auflösung des erzeugten Bildes und nur schwach von der Komplexität der Szene abhängt. Um dies erreichen zu können, wird auch eine *approximative* Darstellung der Szene zugelassen, bei der gewisse Einschränkungen bei der erzielten Bildqualität in Kauf genommen werden.

Im folgenden sei eine dreidimensionale Szene gegeben, deren Oberfläche vollständig durch eine Menge von Dreiecken sowie Oberflächenattributen zur Berechnung der Oberflächenfärbung beschrieben wird.

Konventionelle und approximative Renderingverfahren

Das konventionelle z-Buffer Verfahren

Der für Echtzeitanwendungen am häufigsten verwendete Algorithmus ist der z-Buffer Algorithmus [Catmull 74]. Dieser zeichnet alle Dreiecke nacheinander im Bild ein und prüft dabei die Sichtbarkeit jedes Pixels dadurch, daß er in einem zusätzlichen Tiefenpuffer für jeden Pixel seine Tiefe ablegt und einen Pixel nur dann überschreibt, wenn dessen Tiefe geringer als die bisher dort vorhandene ist. Das Verfahren benötigt einen Aufwand, der linear mit der Anzahl der Dreiecke und linear mit der von den Dreiecken im Bild bedeckten Fläche steigt [Heckbert und Garland 94]. Bei Szenen mit vielen Details oder mit starker Verdeckung ist die Effizienz des Verfahrens daher nicht mehr ausreichend.

Bekannte Verbesserungen und Alternativen

Zur Steigerung der Effizienz sind zwei verschiedene Ansätze denkbar. Zum einen versuchen viele Verfahren, durch eine Vorverarbeitung der Szene die Anzahl oder die Fläche der Dreiecke auf ein handhabbares Maß zu reduzieren. Der andere Ansatz besteht darin, von vornherein ein

Renderingverfahren mit outputsensitivem Laufzeitverhalten zu benutzen. Das randomisierte z-Buffer Verfahren fällt dabei in diese zweite Kategorie.

Unter den Verfahren der ersten Kategorie finden sich solche, die versuchen, die Dreiecksnetze der Szene zu vereinfachen oder Teile durch Ersatzdarstellungen zu ersetzen oder versuchen, verdeckte Szenenteile vom Rendering auszuschließen.

- **Vereinfachung von Dreiecksnetzen („*mesh simplification*“):** Die Idee basiert darauf, für weiter entfernte Objekte der Szene gröbere Dreiecksnetze als für näher gelegene zu verwenden. Im einfachsten Fall werden mehrere, verschieden detaillierte Modelle per Hand erzeugt und automatisch je nach Entfernung zwischen ihnen umgeschaltet. Es gibt eine Reihe von Algorithmen (siehe z.B. die Übersichtsartikel [Puppo und Scopigno 97], [Heckbert und Garland 97]), die aus einem gegebenen Dreiecksnetz für ein Objekt vollautomatisch verschieden detaillierte Modelle, auch mit fließendem Übergang zwischen den Detaillierungsstufen („*multi resolution modelling*“), herstellen können. Dreiecksnetzvereinfachungsmethoden leiden unter dem Problem, daß komplizierte Oberflächen mit vielen kleinen, aber wesentlichen Objektdetails nicht ohne starken Qualitätsverlust vereinfacht werden können. Insbesondere Szenen aus sehr vielen unabhängigen Objekten wie z.B. Waldszenen, bereiten Probleme.
- **Ersetzungsmethoden:** Hier werden größere Szenenteile durch Ersatzdarstellungen, meistens Rasterbilder (Texturen, „*image-based rendering*“) ersetzt. Die Ersatzdarstellungen können Objektgruppen („*textured clusters*“, [Maciel und Shirley 95]) oder für eine Menge von Betrachterstandpunkten jeweils alle in der Szene weiter entfernten Objekte („*textured walls*“, [Ebbesmeyer 98]) ersetzen. Zudem ist es sowohl möglich die Ersatzdarstellungen statisch vorzuberechnen als auch diese dynamisch zu erzeugen [Shade et al. 96]. Die statischen Methoden leiden dabei oft unter dem großen Speicherbedarf der Ersatzdarstellungen. Alle Ersetzungsmethoden haben Probleme damit, daß eine Ersetzung im allgemeinen nur für einen Betrachterstandpunkt korrekt sein kann und danach mehr oder weniger starke Fehler im Bild in Kauf genommen werden müssen, deren Ausmaß stark von der Szene abhängen kann.
- **Unterdrückung verdeckter Flächen:** Das Ziel ist hier, möglichst wenige Objekte zu bearbeiten, die im späteren Bild nicht sichtbar sein werden. Dazu können statische Methoden verwendet werden, die den Raum in Zellen einteilen und vorberechnen, von welchen Zellen welche anderen sichtbar sein werden („*potentially visible sets*“, [Teller und Séquin 91]), und dynamische Methoden, die beim Bearbeiten der Objekte die Sichtbarkeit feststellen (z.B. „*hierarchical z-Buffer*“, [Green et al. 93]). Letztere sind flexibler und arbeiten auf allgemeineren Szenen, erfordern allerdings oft an sich einen größeren Grundaufwand.

Als von vornherein outputsensitives Renderingverfahren ist vor allem das Raytracing interessant [Appel 68], hier wird für jeden Bildpunkt ein Strahl in die Szene geschickt und der erste Schnittpunkt mit einem Objekt der Szene gesucht. Entscheidend dabei ist die Geschwindigkeit, mit der der erste Schnitt zwischen einem Objekt und einem Strahl gefunden werden kann. Zur Beschleunigung dieser Suche können verschiedene Datenstrukturen eingesetzt werden, eine effiziente Behandlung von komplexen Szenen bleibt allerdings problematisch.

Mit den genannten Verbesserungsmöglichkeiten bleiben noch viele Szenen übrig, die nur schwer effizient behandelt werden können. Eine solche Klasse sind Szenen, in denen eine große Menge von unabhängigen Objekten sichtbar sind, wie z.B. Waldszenen oder Ansichten von Stadtszenen aus einer erhöhten Perspektive. Diese können nicht mit „*mesh simplification*“ behandelt werden, da zu viele unabhängige Teilobjekte in der Szene vorhanden sind. Ersetzungsmethoden haben Probleme, die lokalen Verdeckungen der Objekte bei einem sich bewegenden Beobachter korrekt wiederzugeben (z.B. die komplexen Verdeckungen von Ästen und Blättern in Waldszenen) und für eine Anwendung von Verfahren zur Unterdrückung verdeckter Flächen werden zu wenige Flächen verdeckt. Das Raytracing scheidet aus, da die Schnittsuche in solchen Szenen zu lange dauert.

Diese Lücke soll in dieser Arbeit geschlossen werden: Das hier vorgestellte, neu entwickelte randomisierte z-Buffer Verfahren kann Szenen dieser Art auch bei sehr hoher Detaillierung und extrem vielen unabhängigen Objekten effizient darstellen. Das Verfahren ist nicht auf diese Art von Szenen begrenzt, allgemein ist mit Laufzeitengewinnen zu rechnen, wenn eine begrenzte Verdeckung und eine sehr hohe Dreiecksanzahl vorliegt. Für eine gute Bildqualität müssen dabei nur relativ schwache Voraussetzungen an die Szene gemacht werden.

Das randomisierte z-Buffer Verfahren

Das randomisierte z-Buffer Verfahren wählt zufällige Punkte so aus der Szene aus, daß diese nach der Projektion im Bild auf den projizierten Flächen ihrer Objekte (annähernd) gleichverteilt sind. Dabei wird nicht berücksichtigt, ob die Objekte im Bild verdeckt sind oder nicht. Aus der Stichprobe wird dann ein Bild der Szene rekonstruiert. Es stellen sich nun zwei Fragen: Zum einen ist fraglich, ob aus einer derartigen, unstrukturierten Punktmenge überhaupt ein sinnvolles Bild der Szene konstruiert werden kann. Im nächsten Abschnitt „Bildrekonstruktion“ werden die Ergebnisse zu dieser Frage zusammengefaßt. Zum anderen besteht das Problem, effizient Stichprobenpunkte mit einer Wahrscheinlichkeit proportional zur projizierten Fläche der Objekte auszuwählen. Die Ergebnisse zu diesem Teilproblem werden im darauf folgenden Abschnitt zusammengefaßt.

Bildrekonstruktion

Sei hier angenommen, daß eine Menge von Stichprobenpunkten gegeben ist, die auf der ins Bild projizierten Flächen der (sichtbaren *und* verdeckten) Dreiecke gleichverteilt ist.

Pixelweise Bildrekonstruktion

Im einfachsten Fall werden zur Rekonstruktion eines Bildes alle gewählten Stichprobenpunkte projiziert, die entsprechende Oberflächenfärbung berechnet und dann als einzelne Pixel ins Bild eingezeichnet. Um die Verdeckungen der Objekte untereinander richtig zu rekonstruieren, wird dabei ein Tiefenpuffer und ein Tiefenvergleich bei jedem Pixel wie beim konventionellen z-Buffer Verfahren verwendet. Damit Vordergrundbereiche sicher abgedeckt werden, d.h., daß alle von ihnen bedeckten Pixel mit Stichprobenpunkten aus dem Vordergrund bedeckt werden und kein Hintergrund durchscheint, ist ein Stichprobenumfang notwendig, der so groß ist, daß in alle vollständig vom Vordergrund bedeckten Pixel mindestens ein Stichprobenpunkt des Vordergrundes fällt.

Die erwartete Stichprobengröße, die notwendig ist, um dies zu erreichen, liegt bei $a \cdot \ln v$ Punkten, wenn v die sichtbare und a die gesamte (sichtbare + verdeckte) projizierte Fläche des Bildes

in Pixeln ist. Weiterhin ist eine stärkere Abweichung von diesem Wert sehr unwahrscheinlich, für große Werte von v fällt die Wahrscheinlichkeit, um mehr als $c \cdot v$ davon abzuweichen, exponentiell mit c. Somit ist eine korrekte Rekonstruktion quasi garantiert, wenn ein Mindeststichprobenumfang proportional zur projizierten Gesamtfläche, multipliziert mit einem Überhang in $\Theta(\log v)$, eingehalten wird. Bei typischen Auflösungen (v liegt maximal in der Größenordnung der Auflösung des Bildes in Pixeln) entsteht so allerdings ein beachtlicher Grundaufwand, jeder Pixel muß im Durchschnitt etwa 10 bis 15 mal gefüllt werden, um „Löcher“ im Vordergrundbereich zu vermeiden.

Konkret bedeutet dies für ein vollständig bedecktes Bild von 640×480 Pixeln einen erwarteten Stichprobenumfang von 3,9 Millionen Punkten die alle projiziert und eingezeichnet werden müssen (unabhängig von der Anzahl der Dreiecke der Szene), wobei dieser Wert mit zunehmender Verdeckung noch linear ansteigt. Entsprechend ist erst für Szenen mit mehr als einigen Millionen Dreiecken ein Laufzeitgewinn gegenüber einem konventionellen z-Buffer Algorithmus zu erwarten, da dieser zuvor weniger Projektionen von Punkten benötigt. Da aktuelle Hardware nur ein bis zwei Millionen Projektionen pro Sekunde berechnen kann, ist so auch noch keine Echtzeitdarstellung erreichbar.

Bildqualität

Eine genauere Untersuchung des pixelweisen Rekonstruktionsprozesses zeigt, daß auch bei einer vollständigen Abdeckung des Vordergrundes noch falsche Bilder entstehen können. Nur für diejenigen Pixel, in denen der Vordergrund und der Hintergrund in disjunkten Tiefenintervallen liegen, kann bewiesen werden, daß dort bei ordnungsgemäßer Abdeckung immer ein korrektes Bild entsteht. Von den anderen Fällen haben sich in der Praxis diejenigen als problematisch erwiesen, in denen ausgedehnte Oberflächen in sehr geringem Abstand zueinander liegen und unter einem nicht senkrechten Beobachtungswinkel betrachtet werden. Hier kann der Hintergrund nach der Rekonstruktion teilweise durchscheinen. Dies tritt z.B. bei Oberflächendetails auf, die durch aufgesetzte Polygone modelliert wurden. In vielen Fällen können diese Probleme allerdings relativ leicht bei der Modellierung vermieden werden.

In Bezug auf Aliasingartefakte zeigte sich, daß das randomisierte Verfahren in Bereichen, in denen nur einige wenige unterschiedlich gefärbte Oberflächenelemente auf jeden Pixel fallen, ähnliche Artefakte wie das konventionelle z-Buffer Verfahren liefert. In Bereichen mit hoher Detaillierung treten allerdings keine systematischen Aliasingartefakte mehr auf, sondern ein gleichmäßiges Rauschen im Bild. Durch Mittelwertbildung kann hier ein störungsarmes Bild erzeugt werden, für eine Reduktion des Rauschanteils um einen Faktor $1/q$ ist eine Mittelung über $\Theta(q^2)$ Bilder notwendig. Prinzipiell liefert das randomisierte Verfahren nie mehr systematische Aliasingartefakte als das konventionelle z-Buffer Verfahren.

Alternative Bildrekonstruktionsmethoden

Es gibt zwei Gründe dafür, andere Bildrekonstruktionsalgorithmen zu suchen: Zum einen ist eine Reduktion des hohen Stichprobenaufwandes beim pixelweisen Vorgehen wünschenswert, zum anderen soll eine höhere Bildqualität, insbesondere in Bezug auf Aliasingartefakte erreicht werden. Hier wurden daher noch folgende weitere Methoden untersucht:

Große Farbklötze: Quadraten mit d Pixeln Kantenlänge (etwa 2-5 Pixel), für die jeweils ein konstanter Farb- und Tiefenwert angenommen wird, werden anstelle der Pixel in den Tiefenpuffer eingezeichnet. Die Verwendung pixelgenau positionierbarer Farbquadrate sorgt dafür, daß Kanten von Objekten deutlich besser als bei einer schlichten Reduzierung der Auflösung wiedergegeben werden. Der Stichprobenumfang reduziert sich dadurch auf $a' \cdot \ln v'$, mit $a' = a/d$ und

$v' = v/d$, also grob um den Faktor d^2. Diese Methode liefert deutliche Laufzeitverbesserungen: Bei akzeptabler Bildqualität konnte in der Praxis so eine Beschleunigung bis zu einem Faktor 15 erreicht werden.

Rekonstruktion ohne Tiefenpuffer: Bei der Bildrekonstruktion müssen zwei Teilaufgaben gelöst werden: Die Rekonstruktion der Verdeckung und das Auffüllen des Bildes nach den verbleibenden Stichprobenpunkten. Die Verfahren, die Pixel oder Farbklötze mit einem z-Buffer Verfahren einzeichnen, lösen beide Aufgaben in einem Schritt. Durch eine Trennung der beiden Schritte kann die Bildrekonstruktion verbessert werden. Zur Verdeckungsrekonstruktion werden in einer festen, quadratischen Umgebung jedes Stichprobenpunktes alle Punkte mit größerer Tiefe gelöscht. Die dabei erreichbare „Verdeckungsauflösung" hängt vom Stichprobenumfang ab, man erhält den selben Zusammenhang wie bei der Farbklotzmethode, der Durchmesser der Umgebung entspricht dabei dem Durchmesser der Farbklötze.

Da sich die Stichprobenpunkte bei einer zufälligen Verteilung in einigen Bereichen stärker häufen können als in anderen, kann eine bessere Ausnutzung der Stichprobe dadurch erreicht werden, daß nicht alle Punkte mit größerer Tiefe gelöscht werden, sondern daß durch eine Heuristik Punkte, die sich in ihrer Tiefe oder ihrer Normalenrichtung nur schwach vom nächsten Punkt unterscheiden, erhalten werden. Für einige Szenen sind dann allerdings Fehlinterpretationen möglich. Nach der Verdeckungserkennung können zum Auffüllen des Bildes die folgenden beiden Methoden verwendet werden:

Diskrete Voronoiregionen: Hier wird die nächste Nachbarschaft jedes sichtbaren Stichprobenpunktes in dessen Farbe gefüllt: Es wird eine simultane Breitensuche von allen Stichprobenpunkten im Rasterbild durchgeführt um in linearer Zeit (linear in der Pixelanzahl) die Voronoiregionen bezüglich der Einsnorm zu berechnen. Die Methode liefert ähnliche Laufzeitverbesserungen wie die Farbklotzmethode, bei einer (leicht) verbesserten Bildqualität: Das Auslaufen von Kanten durch die breiten Farbklötze entfällt und das Bild zeigte bei Verwendung von Verdeckungsheuristiken weniger starke Aliasingartefakte.

Gaußsche Tiefpaßfilter: Eine Standardmethode zur Reduktion von Aliasingartefakten aus dem Bereich des stochastischen Raytracings [Cook 91] ist die Verwendung von Tiefpaßfiltern. Hier werden Farben für Punkte des Bildes als gewichteter Mittelwert der benachbarten Stichprobenpunkte gebildet. Diese Methode kann Aliasingartefakte stark reduzieren, benötigt aber lange Rekonstruktionszeiten (u.U. mehrere Minuten). Da die Tiefpaßfilterung niedrige Frequenzanteile im Bild betont, ist außerdem ein großer Stichprobenumfang notwendig, um Artefakte durch niederfrequentes Rauschen zu vermeiden. Bei geringem Stichprobenumfang ist das Kosten/Nutzen Verhältnis daher schlecht. Bei einem großen Stichprobenumfang können Bilder hoher Qualität mit beliebig geringen Aliasingartefakten erzeugt werden.

Wenn die Geschwindigkeit das Hauptkriterium ist, erscheint die Rekonstruktionsmethode mit Farbklötzen als die Methode der Wahl, alle hier benötigten Operationen werden ggf. auch von Grafikhardware unterstützt. Die Verbesserungen der Bildqualität durch eine aufwendigere Rekonstruktion lohnen nicht unbedingt nicht den zusätzlichen Aufwand. Für Szenen, die für Aliasing anfällig sind, konnte allerdings nur die Filtermethode überzeugende Ergebnisse liefern.

Effiziente Auswahl der Stichprobenpunkte

Entscheidend für Laufzeitgewinne beim randomisierten z-Buffer Verfahren gegenüber dem konventionellen Verfahren ist eine effiziente Auswahl der Stichprobenpunkte. Für eine exakte Auswahl ist eine genaue Klassifizierung aller Objekte nach ihrer projizierten Fläche, die sich bei

Bewegungen des Betrachters ständig ändern, notwendig. Während dies leicht durch Bearbeitung aller Objekte erreicht werden kann, erscheint die Aufgabe für eine Lösung in sublinearer Zeit schwierig. Hier wurde daher eine approximative Lösung eingesetzt: Die Wahrscheinlichkeitsdichte für die Stichprobenauswahl folgt nicht exakt der projizierten Fläche, sondern es werden Abweichungen vom idealen Wert zugelassen. In einem Bereich, von dem bekannt ist, daß die Stichprobendichte teilweise um einen Faktor zu niedrig liegen kann, wird dies kompensiert, indem eine entsprechend höhere Stichprobenanzahl verwendet wird, so daß die für eine korrekte Bildrekonstruktion notwendige Stichprobendichte nicht unterschritten wird; eine zu hohe Stichprobendichte hat keinen negativen Effekt auf die Bildrekonstruktion.

Bei der perspektivischen Projektion (Zentralprojektion) verringert sich die projizierte Fläche eines Oberflächenstückes im wesentlichen mit dem Quadrat der Tiefe und proportional zum Kosinus zwischen der Oberflächennormalen und dem Blickvektor. Die Idee der Approximation besteht darin, Gruppen von Objekten dynamisch zu bilden, in denen der *Projektionsfaktor*, also der Wert, um den Flächen bei der Projektion skaliert werden, wenig zwischen den Objekten abweicht und dann innerhalb dieser Gruppen zufällige Punkte gewichtet nach den nicht projizierten, absoluten Flächeninhalten auszuwählen. Letzteres geschieht dadurch, daß Listen mit aufsummierten Flächen („*Verteilungslisten*“) vorab für alle möglichen Gruppen gebildet werden, in denen dann dynamisch mit binärer Suche nach dem Dreieck gesucht wird, bei dem eine gleichverteilt gezogene Zufallszahl überschritten wird, um die Stichprobenpunkte auf die Dreiecke zu verteilen. Auf jedem Dreieck wird dann ein zufälliger Punkt bestimmt. Die Suche des Dreiecks benötigt nur logarithmische Zeit in der Anzahl der Dreiecke der Szene und die Gruppeneinteilung ist davon unabhängig, so daß man ein sehr schwaches Wachstum des Aufwandes mit dem Detaillierungsgrad der Szene erhält.

Um den Einfluß der Tiefe für die Gruppen zu beschränken, wird ein Octree [Samet 90] auf den Dreiecken der Szene in einem Vorberechnungsschritt aufgebaut aus dem dynamisch Boxen von Objekten entnommen werden, in denen die verschiedenen Tiefen nur zu einer beschränkten Abweichung des Projektionsfaktors führen. Größere Boxen außerhalb des Sichtpyramidenstumpfes werden dabei ausgelassen, so daß sich der Querschnitt der betrachteten Sichtpyramide nur um einen konstanten Faktor erhöht. Wenn τ die „*Tiefendynamik*“ der Szene, d.h. das Verhältnis des nächsten darstellbaren Punktes (Mindestabstand der Zentralprojektion) zum Durchmesser der Szene ist, so sind nur $O(\log \tau)$ Boxen dazu notwendig, die auch in $O(\log \tau)$ Zeit ausgewählt werden können. Zur späteren Verteilung der Stichprobenpunkte nach den Flächeninhalten werden für alle Knoten des Baumes Verteilungslisten aufgebaut, die so ineinander verschachtelt gespeichert werden können, daß insgesamt nur linearer Speicherplatz benötigt wird.

Diese Strategie alleine führt dazu, daß der erwartete Stichprobenumfang nur um einen Faktor 4ε (für ein beliebiges $\varepsilon > 1$) größer als der bei einer exakten Auswahl ist, wenn man voraussetzt, daß alle Oberflächen zufällig orientiert sind, so daß ihre Oberflächennormalen auf der Einheitskugel gleichverteilt sind. Im schlimmsten Fall kann die Abweichung natürlich noch beliebig groß sein.

Um die Abweichung weiter einzuschränken, werden die Szenenteile bei der Vorberechnung zusätzlich nach ähnlichen Normalen gruppiert und für jede Gruppe eine räumliche Aufteilung vorgenommen. Zur Einteilung wird ein Gitter von Winkelklassen auf den Normalen in Polarkoordinaten aufgebaut. Dabei ist zu beachten, daß der Blickwinkel nicht nur von den Normalen, sondern auch von der Position der Objekte im Raum abhängt, so daß zur Erhöhung der Winkelungenauigkeit auch eine räumliche Mindestunterteilung notwendig wird. Für jede Objektgruppe werden alle Informationen über Winkel und Tiefe in die Schätzung des Projektionsfaktors für die Gruppe einbezogen. Für eine Kantenlänge λ des Winkelgitters erhält man damit eine Reduk-

tion der Überschätzung des Stichprobenumfangs auf höchstens etwa $1+3/2\pi^2\lambda$ (für kleine λ) bei einem Auswahlaufwand von $\Theta(\lambda^{-5})$. Diese Methode läßt sich, wie die Experimente bestätigen, bei nicht zu kleinen Werten für λ, gut in der Praxis einsetzen. Die Gittermethode benötigt für n Dreiecke $O(n)$ Speicher und $O(n \cdot \log n)$ Aufbauzeit.

Um die Szene nach einer Änderung (einfügen oder löschen von Objekten) wieder darzustellen, müssen die vorberechneten Datenstrukturen angepaßt werden. Hierzu ist bei n Dreiecken neben der Suche der Box des Objektes im Octree noch ein Aufwand von $O(\log n)$ zur Änderung der Flächenlisten, die im dynamischen Fall als balancierter Baum dargestellt werden, pro Änderung notwendig. Die dynamische Version wurde im Rahmen dieser Arbeit nicht mehr implementiert.

Bei einer groben Gittereinteilung können immer noch worst-case Szenen konstruiert werden, bei denen der Stichprobenumfang beliebig stark überschätzt wird. Um einen maximal konstanten Approximationsfehler für die Stichprobenverteilung garantieren zu können, benötigt man auch nach Verbesserungen des Algorithmus (Verwendung eines Quadtrees statt eines regelmäßigen Gitters auf den Polarkoordinaten) immer noch einen Zeitaufwand von $\Omega(\omega^{-4})$ zur Boxenauswahl im worst-case, wobei ω die Differenz des größten Winkels in der Szene unter $\pi/2$ zu $\pi/2$ ist. Da ω je nach Perspektive beliebig klein sein kann, ist ein solches exaktes Approximationsverfahren nicht praxistauglich. Entscheidend für die Praktikabilität des randomisierten z-Buffer Verfahrens ist daher, daß eine nur grobe Berücksichtigung des Blickwinkels für die meisten Szenen genügt.

Ausnutzung von hierarchischen Instanzen in Szenen

Das randomisierte Verfahren lohnt sich gegenüber dem konventionellen z-Buffer Verfahren erst für Szenen, die sehr viele Dreiecke aufweisen (mehrere Millionen) und bietet das Potential, Szenen mit einer nochmals um mehrere Größenordnungen höherer Dreiecksanzahl effizient darzustellen. Die Speicherung entsprechend großer Szenen bereitet allerdings Speicherplatzprobleme; eine explizite Speicherung aller Dreiecke ist kaum noch möglich. Um diese zu umgehen, wurde ein Verfahren entwickelt, um Szenen, die aus Instanzen gleicher Objekte bestehen, effizienter zu speichern. Der Modellierer gibt dazu einen Szenengraphen (ähnlich wie in Open Inventor [Wernecke 94]) vor, der Instanzen von Objekten hierarchisch durch Referenzen in einem azyklisch gerichteten Graphen beschreibt. Die Instanzen können sich dabei untereinander durch Hintereinanderausführungen von Translationen, Drehungen, Skalierungen und Spiegelungen unterscheiden. Die Benutzung von Instanzen von Objekten, die hierarchisch ihrerseits wieder aus Instanzen bestehen, ermöglicht eine sehr kompakte Kodierung vieler Szenen.

Für einen gegebenen Szenengraphen werden die für die Stichprobenauswahl benötigten Raumunterteilungsbäume parallel in einer ähnlichen Struktur angelegt, die entsprechend wenig Speicher benötigt. Mit dieser Darstellung konnten Szenen mit bis zu 10^{13} Dreiecken modelliert und mit dem randomisierten z-Buffer Verfahren effizient dargestellt werden. Bei der Stichprobenauswahl muß für jeden Punkt und jede Instantiierungsebene eine zusätzliche Transformation ausgeführt werden, so daß vom Modellierer nicht mehr als drei oder vier Ebenen verwendet werden sollten.

Implementation und experimentelle Ergebnisse

Das randomisierte z-Buffer Verfahren wurde prototypisch als C++-Bibliothek unter Benutzung der Grafikschnittstelle OpenGL [Woo et al. 97] implementiert. Die Implementation gestattete, Szenengraphen aus einigen einfachen Grundprimitiven zu erstellen und mit dem konventionellen und dem randomisierten z-Buffer Verfahren darzustellen.

Die Implementation wurde an verschiedenen Testszenen erprobt. Die Berechnung der Verteilungsbäume benötigte dabei (unter Ausnutzung hierarchischer Instanzenbildung) nie mehr als einige Minuten.

Stadt- und Waldszenen: Szenen aus vielen unabhängigen Objekten mit bis zu $6 \cdot 10^{11}$ Dreiecken konnten bei guter Bildqualität in einigen Sekunden (etwa 4-60 Sekunden, je nach projizierter Fläche) dargestellt werden. Eine Szene, die eine Übersicht über eine komplexe Waldszene zeigt, konnte z.B. in 32 Sekunden dargestellt werden, während ein konventionelles z-Buffer Rendering auf der gleichen Plattform etwa 39 Tage gedauert hätte. Wesentliche Darstellungsfehler ergaben sich nur in einigen Nahansichten der Stadtszene: Da einige Oberflächendetails (wie Straßen und Fenster) als aufgesetzte Oberflächendetailpolygone modelliert waren, schien unter diesen teilweise der Hintergrund durch, da hier die Verdeckung bei der Bildrekonstruktion falsch rekonstruiert wurde. Analytisch kann gezeigt werden, daß eine Darstellung einer Szene von Objekten, die in einer zweidimensionalen Scheibe mit Radius r gleichmäßig verteilt sind, mit dem randomisierten z-Buffer Verfahren nur $\Theta(\log^2 r)$ Zeit gegenüber $\Theta(r^2)$ beim konventionellen Verfahren benötigt.

Ausgedehnte Dreiecksnetze: In einer Szene eines Dreieckmodells einfacher Topologie konnte eine Darstellung mit hoher Bildqualität (kaum Unterschiede zu einer konventionellen z-Buffer Darstellung) erreicht werden. Ersetzungsverfahren haben in solchen Szenen oft Probleme mit der Bildqualität. Da Szenen dieser Art oft viele redundante Dreiecke enthalten, liefern *„mesh simplification“*-Verfahren hier allerdings meistens bessere Laufzeiten.

Schachbrettszenen: Eine Ansicht über ein sich in die Tiefe erstreckendes Schachbrett diente als Test für die Anfälligkeit für Aliasing. Das randomisierte z-Buffer Verfahren zeigte dabei im mittleren Tiefenbereich ähnliche Artefakte wie das konventionelle z-Buffer Verfahren, im weiter entfernten Bereich gingen diese jedoch in ein unstrukturiertes Rauschen über. Mit Hilfe von Mittelwertbildung über mehrere Bilder konnte das Rauschen reduziert werden. Die Rekonstruktion mit Voronoiregionen lieferte weniger Aliasingartefakte und mit Hilfe von Tiefpaßfiltern konnte eine praktisch störungsfreie Ansicht berechnet werden, wobei allerdings ein großer Stichprobenumfang für eine gute Bildqualität notwendig war.

In Abhängigkeit der verschiedenen Szenen- und Verfahrensparameter zeigte sich folgendes Bild:

- Mit wachsender Detaillierung stieg der Rechenaufwand des Verfahrens nur logarithmisch mit der Dreiecksanzahl. Für jede zusätzliche Instantiierungsebene mußte eine leichte Erhöhung des Aufwandes zur Projektion der Stichprobenpunkte hingenommen werden (etwa 30% des Aufwandes ohne Instanzen). Die Zeit zur Darstellung einer Szene aus 10^{13} Dreiecken lag nur um einen Faktor drei höher als die zur Darstellung einer gleichartigen Szene aus nur 32 Dreiecken. Der Grundaufwand lag allerdings so hoch, daß erst in der Größenordnung von einer Millionen Dreiecken eine höhere Geschwindigkeit als mit dem konventionellen z-Buffer Verfahren erreicht werden konnte.
- Für eine Szene mit relativ zufälligen Oberflächenorientierungen (eine Waldszene) wurde die Anzahl von Winkelklassen und die Genauigkeit der räumlichen Unterteilung, d.h. der maximale Fehler ε im Projektionsfaktor bezüglich der Tiefe, variiert. Bei einer Ansicht mit großer Tiefendynamik stellte sich heraus, das der Verzicht auf Winkelklassen und die Verwendung einer sehr feinen räumlichen Unterteilung die besten Laufzeiten lieferte, während sich bei geringer Tiefendynamik durch die Verwendung von Winkelklassen die Laufzeit verbessern ließ.

- Die Überschätzung des Stichprobenumfangs durch die ungenaue Schätzung des Projektionsfaktors stieg für große ε mit $\sqrt{\varepsilon}$ und die Boxenanzahl für kleine ε mit $(\sqrt{\varepsilon}\text{-}1)^{-3}$, was entsprechende theoretische Analysen bestätigte. Es ergab sich ein ausgeprägtes Minimum der Laufzeit, welches je nach Szene, Tiefendynamik und Anzahl von Winkelklassen bei unterschiedlichen Werten für ε lag.
- Eine Kombination des randomisierten z-Buffer Verfahrens mit dem konventionellen z-Buffer Verfahren zur Darstellung des Nahbereiches lieferte für Stadtszenen aus vielen wenig detaillierten Objekten Geschwindigkeitsgewinne. Für eine Szene mit relativ geringer projizierter Fläche konnten auf aktueller PC-Hardware Renderingzeiten erreicht werden, die fast im Echtzeitbereich lagen (< 200 msec).

Insgesamt zeigte sich, daß hochdetaillierte Szenen, für die eine konventionelle z-Buffer Darstellung quasi unmöglich gewesen wäre, in mit dem randomisierten Verfahren in sehr kurzer Zeit von einigen Sekunden und in guter Qualität dargestellt werden konnten. Echtzeitnahe Laufzeiten waren (unabhängig von der Detaillierung) nur bei einer geringen projizierten Gesamtfläche bzw. geringer Bildauflösung zu erreichen. Szenen mit größerer Tiefendynamik ließen sich bei einem Verzicht auf Winkelklassen am effizientesten darstellen. Die maximale Abweichung des Projektionsfaktors bezüglich der Tiefe für die räumliche Unterteilung sollte im Zweifelsfall, aufgrund des asymmetrischen Wachstums des Aufwandes, eher etwas zu groß als zu klein gewählt werden.

Gesamtergebnis

Das randomisierte z-Buffer Verfahren kann eine approximative Darstellung von hochdetaillierten Szenen berechnen, wobei die Darstellungszeit nur sehr schwach von der Anzahl der Details abhängt. Es läßt sich auf eine große Klasse von Szenen anwenden, die Topologie der verwendeten Dreiecksnetze spielt dabei, im Gegensatz zu Dreiecksnetzvereinfachungsmethoden, keine Rolle. Die Bildqualität ist mit Ergebnissen des konventionellen z-Buffer Verfahrens vergleichbar und ebenfalls, bis auf den Sonderfall von dicht beieinander liegenden ausgedehnten Oberflächen, weitgehend unabhängig vom Szenenmaterial, im Gegensatz zu vielen Ersetzungsmethoden. Verdeckte Flächen werden nicht erkannt, der Aufwand wächst linear mit der projizierten Gesamtfläche.

Mit dem neuen Verfahren läßt sich das Problem der Überdetaillierung für eine große Klasse von Szenen mit einem universellen, algorithmisch einfachen Prinzip lösen, wobei allerdings ein Aufwand in Kauf genommen werden muß, der um einen logarithmischen Faktor superlinear mit der Bildauflösung wächst und so zu einem relativ hohen Grundaufwand des Verfahrens führt, der auf aktueller Hardware dazu führt, daß für übliche Bildauflösungen noch keine Echtzeitdarstellung erreicht werden kann. In der Praxis konnten mit dem Verfahren Szenen mit mehreren hundert Milliarden Dreiecken in wenigen Sekunden in einer Qualität dargestellt werden, die sich kaum von einer konventionellen z-Buffer Darstellung unterschied, welche aber eine Berechnungszeit von mehreren Wochen benötigt hätte.

1 Problemstellung

1.1 Das Walkthrough-Problem

Ein zentrales Problem in der Computergrafik ist das sogenannte *Walkthrough-Problem*. Dabei soll einem interaktiven Benutzer ermöglicht werden, als „virtueller Besucher“ eine von einem Rechner in Echtzeit dargestellte dreidimensionale Szene zu begehen. Der Besucher wandert dabei typischerweise auf einem stetigen Pfad mit stetig variierendem Blickwinkel durch die Szene, die in einer perspektivischen Projektion dargestellt wird. Das Walkthrough-Problem heißt dynamisch, wenn sich die Szene über die Zeit verändern kann, und sonst statisch.

Um das Problem zu lösen, muß das Darstellen der Szene, das sog. *Rendering*, online gegenüber der Bewegung des Benutzers mehrmals pro Sekunde (typischerweise 10-50 mal) durchgeführt werden. Diese Form der Echtzeitdarstellung ist oft wünschenswert, da sie einen besonders intuitiven Eindruck der Szene vermittelt:

Eine einzelne, zweidimensionale Ansicht einer dreidimensionalen Szene kann in der Regel nicht alle geometrischen Informationen über die Szene liefern. Der Mensch und speziell sein visuelles Wahrnehmungssystem hat sich daher im Laufe der Evolution darauf angepaßt, die Objekte seiner Umwelt aufzufassen, indem er sich bei ihrer Betrachtung stetig um sie herum bewegt. Eine flüssig animierte Bewegung durch eine Szene erleichtert daher stark das intuitive Verständnis.

Entsprechend gibt es viele Anwendungsgebiete, für die die Walkthroughanimation als Teilproblem grundlegend ist, z.B. bei der Visualisierung großer Datenmengen, beim Entwurf dreidimensionaler Objekte am Computer (CAD) und nicht zuletzt in der Unterhaltungsbranche (vor allem bei vielen modernen Computerspielen). Für die viele dieser Anwendungen ist gerade die interaktive Steuerung der Bewegung besonders wichtig, was eine Bildsynthese online und in Echtzeit verlangt.

Ein Verringern der Bildaufbaufrequenz auf unter 5 bis 10 Bilder pro Sekunde zerstört erfahrungsgemäß den Eindruck von fließender Bewegung und damit auch einen Teil des dreidimensionalen Eindrucks. Es wird schwierig, Geschwindigkeiten einzuschätzen, und die Navigation in der Szene wird stark erschwert.

1.2 Outputsensitives Rendering und Approximation

Viele Anwendungen verlangen aus verschiedenen Gründen die Darstellung sehr komplexer Szenen: Im CAD-Bereich möchte man auch komplexe Entwürfe interaktiv bearbeiten und begutachten können, bei wissenschaftlichen Visualisierungen sollen auch große Datenmengen darstellbar sein. Anwendungen aus der Unterhaltungsbranche legen auf ästhetische Qualitäten Wert, oft sollen dreidimensionale Szenen dem Eindruck einer photorealistischen, natürlichen Szenen nahekommen und auch dies erfordert sehr komplexe Modelle der Szene. Was genau als Komplexität einer Szene aufzufassen ist, ist nicht immer eindeutig definiert; je nach dem verwendeten Renderingverfahren können verschiedene Einflüsse von Szenenparametern auf die Laufzeit bestehen, dies wird im folgenden noch präzisiert. Ein wesentlicher Faktor für alle bekannten Verfahren ist die Anzahl von Elementarobjekten, aus denen eine dreidimensionale Szene aufgebaut ist, und die im wesentlichen festlegt, wie viele Details ein Szene aufweisen kann. Hinzu

können weitere Parameter, wie z.B. der Flächeninhalt der ins Bild projizierten Objektoberflächen, die sogenannte *projizierte Fläche* der Szene, oder auch die Komplexität der Interaktion des Lichtes mit den Szenenelementen kommen.

Die meisten klassischen Renderingverfahren benötigen einen Berechnungsaufwand, der mindestens linear mit der Anzahl von Elementarobjekten in der Szene ggf. zusätzlich noch mit weiteren Parametern wächst. Damit ist die maximale Komplexität einer in Echtzeit darstellbaren Szenen beschränkt, und auch die Verwendung leistungsfähiger Hardware kann die Grenze nicht allzu weit nach oben verschieben.

Da ein Bild der Szene nach dem Rendering nur noch eine begrenzte Anzahl von Informationen, beschränkt durch die Anzahl von Pixeln des Bildes[1], aufweist, erscheint es denkbar, mit einem geringeren Aufwand bei komplexen Szenen auskommen zu können. Ein Renderingalgorithmus wird als *outputsensitiv* bezeichnet, wenn der Aufwand zum Erzeugen eines Bildes im wesentlichen von der Auflösung und Qualität des erzeugten Bildes und nur schwach von der Komplexität der Szene abhängt. Das in dieser Arbeit entwickelte randomisierte z-Buffer Verfahren wird sich zumindest gegenüber einem Teil der Szenenparameter (genauer gesagt gegenüber der Anzahl von Elementarobjekten) outputsensitiv verhalten, allerdings nicht gegenüber allen (lineares Wachstum des Aufwandes mit der projizierten Fläche der Szene).

Vorausgesetzt, daß ein extrem detailreiches und damit komplexes Modell einer Szene zur Verfügung steht, haben outputsensitive Renderingmethoden prinzipiell einen wichtigen Vorteil gegenüber klassischen Methoden, die einfach mit einer weniger komplexen Szenenbeschreibung betrieben werden. Eine outputsensitive Methode kann die Detaillierung der Szene adaptiv für jeden Blickpunkt steuern, so daß immer alle bei der gegebenen Auflösung sichtbaren Details der Szene dargestellt werden. Eine klassische Methode angewandt auf ein wenig komplexes Modell der Szene würde nur in einer Übersichtsperspektive gute Ergebnisse liefern, wenn einzelne Objekte von nahem betrachtet werden, würden hier die fehlenden Details sichtbar.

Ein weiterer Begriff in diesem Zusammenhang ist das *approximative* Rendering. Hierbei wird bewußt eine Darstellung mit kleinen Fehlern in Kauf genommen, um dadurch eine schnellere Darstellung zu ermöglichen. Hängt der Aufwand für das Rendering dann nur noch von der Größenordnung der produzierten Fehler im Bild und nur noch schwach von der Szenenkomplexität ab, so erhält man dadurch eine in diesem Sinne outputsensitve Methode. In diesem Sinne sind diese beiden Begriffe eng verwandt.

In beiden Fällen wird die „Bildqualität“ oder der „Fehler“ der Ausgabe zu einem Parameter, der einen wesentlichen Einfluß auf die Laufzeit hat. Eine Quantifizierung eines solchen „Fehlers“ ist allerdings eine problematische Aufgabe, da er von der subjektiven menschlichen Wahrnehmung abhängt. Es ist auch zu beachten, daß das Ausmaß und die Art von Fehlern, die noch toleriert werden können, wesentlich vom Anwendungsgebiet abhängen. So könnten in medizinischen Anwendungen bestimmte Arten von falschen Darstellungen u.U. katastrophale Folgen haben, während in anderen Anwendungen, wie z.B. bei Flugsimulatoren oder interaktiven Spielen, eine flüssige Darstellung der Animation mit hohen Bildwiederholraten die höchste Priorität hat: Um die Navigationsfähigkeit in der Szene zu erhalten sind in solchen Anwendungen Kompromisse bei der Bildqualität eher tolerierbar als lange Renderingzeiten.

Bei approximativen Verfahren lassen sich generell zwei Klassen von Verfahren ausmachen: Der erste Ansatz besteht darin, die Szenendaten für jeden gegebenen Blickpunkt so vorzuverarbei-

[1] Hier wird stillschweigend davon ausgegangen, daß stets Rasterbilder als Ausgabe des Renderings erzeugt werden sollen, was beim heutigen Stand der Technik in fast allen Bereichen üblich ist.

ten, daß die Komplexität handhabbar wird, und sie dann mit einem klassischen, nicht outputsensitiven Verfahren darzustellen. Typischerweise werden für eine schnelle, blickpunktabhängige Vereinfachung der Szene vorberechnete, blickpunktunabhängige Datenstrukturen benötigt, aus denen Datensätze geeigneter Komplexität effizient dynamisch entnommen werden können.

Der zweite Ansatz besteht darin, von vornherein ein outputsensitives Renderingverfahren zu entwerfen. Auch hier werden typischerweise Datenstrukturen vorberechnet, die später beim Walkthrough für beliebige Blickpunkte ein beschleunigtes Rendering ermöglichen. Das randomisierte z-Buffer Verfahren gehört zu dieser zweiten Kategorie.

Für ein universelles outputsensitives Renderingverfahren für Walkthroughanwendungen würde man bei einer (abstrakten) Szenenkomplexität von n und einer Auflösung von k eine Laufzeit von $O\tilde{}(k)$ bei einer Vorberechnungszeit von $O\tilde{}(n)$ wünschen. Dynamische Änderungen der Szene sollten in $O\tilde{}(1)$ möglich sein. Derzeit ist allerdings kein Verfahren bekannt, daß bei strengen Anforderungen an die Bildqualität diese Kriterien erfüllen würde und auch das hier vorgestellte randomisierte z-Buffer Verfahren stellt keine solche universelle Lösung dar.

In den letzten Jahren sind allerdings große Anstrengungen zur Entwicklung von outputsensitiven Renderingverfahren unternommen worden, so daß heute eine Vielzahl von Methoden für verschiedene Anwendungsfälle zur Verfügung steht. In Kapitel 2 wird ein Überblick sowohl über die bekannten klassischen wie auch approximativen Renderingverfahren gegeben.

1.3 Schritte eines Renderingverfahrens

Zur Darstellung einer Szene sind mehrere Schritte notwendig, die je nach Renderingalgorithmus unterschiedlich verzahnt ausgeführt werden. Die verschiedenen Teilaufgaben lassen sich wie folgt klassifizieren (grob nach [Foley et al. 96]):

Projektion: Die Elementarobjekte der Szene werden auf das Bild projiziert. Dazu können verschiedene Projektionsfunktionen zur Anwendung kommen, in der Walkthroughanimation wird fast ausschließlich die Zentralprojektion angewandt, die die Arbeitsweise einer Filmkamera bzw. eines Photoapparates simuliert. Diese Art der Projektion wird daher im weiteren zu Grunde gelegt. Die Eigenschaft, daß Objekte bei dieser Projektion reziprok zur Entfernung skaliert werden, ist der wichtigste Ansatzpunkte für die meisten approximativen Renderingverfahren.

Unterdrückung verdeckter Flächen (*hidden surface removal*): Die näher zum Betrachter liegenden Objekte verdecken alle weiter hinten liegenden Objektteile, welche im erzeugten Bild nicht mehr sichtbar sein dürfen und daher in diesem Schritt aus dem Bild entfernt werden müssen. Idealerweise sollten möglichst viele dieser weiter hinten liegenden Objektteile in der Berechnung erst gar nicht bearbeitet werden.

Rasterung: Je nach Renderingverfahren ist es noch notwendig, die im Inneren der projizierten Objekte liegenden Punkte zu bestimmen und mit entsprechenden Farbwerten aufzufüllen.

Schattierung: Für die sichtbaren Oberflächen im Bild muß ein Farbwert zur Darstellung der Oberfläche im Bild berechnet werden. Neben einfachen Modellen mit fest vorgegebenen Farben für die Elementarobjekte werden auch verschieden realistische Simulationen der Interaktion der Oberfläche mit Lichtquellen und u.U. auch mit anderen Oberflächen durchgeführt. Je nach Realismus kann der Berechnungsaufwand hierfür sehr unterschiedlich ausfallen.

Anti-Aliasing: Unter der Fläche eines Pixels kann, wie z.B. an Kanten von Objekten, mehr als ein Objekt sichtbar sein (schematisch in Abbildung 3). Außerdem kann auch die Oberflächenschattierung eines Objektes innerhalb eines Pixels variieren.

In diesem Fall muß festgelegt werden, welcher Farbwert für den Pixel verwendet werden soll. Einfache (vor allem schnelle) Renderingalgorithmen wählen willkürlich die Farbe eines Objektpunktes aus den zur Verfügung stehenden aus, was die tatsächlichen Verhältnisse nicht korrekt wiedergibt. Wenn z.B. systematisch immer der in der Mitte gelegene Objektpunkt (Raytracing) oder der am weitesten vorne gelegene Objektpunkt (z-Buffer) verwendet wird, können dadurch niederfrequente Interferenzmuster, genannt Aliasing, zwischen dem Pixelraster und regelmäßigen Objektdetails entstehen, die den Bildeindruck stark stören können (Abbildung 1).

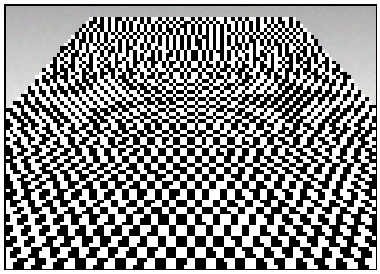


Abbildung 1: Aliasing Artefakte (mittlerer Objektpunkt pro Pixel)

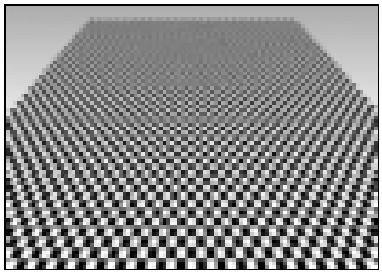


Abbildung 2: mit Anti-Aliasing (gewichtete Mittelwertbildung)

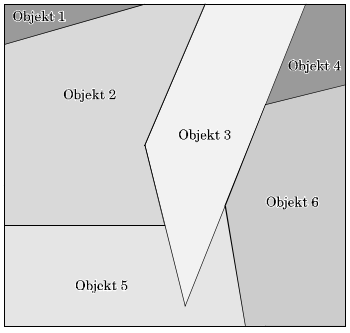


Abbildung 3: Objektteile innerhalb eines Pixels

Eine signaltheoretische Analyse ([Foley et al. 96], S. 617ff) erklärt das Phänomen durch eine Betrachtung des Diskretisierungsprozesses im Frequenzraum und zeigt Lösungsmöglichkeiten (*Anti-Aliasing*): Eine Mittelwertbildung über die in der Nähe des Pixels liegenden Objektpunkte kann die Störungen vermindern (Abbildung 2). Dabei sind verschieden gewichtete Mittelwerte über unterschiedlich große Bereiche (auch über die Grenzen des Pixels hinaus) möglich, wobei verschiedene Kompromisse gemacht werden müssen (siehe auch [Glassner 95a], S. 115ff). Im einfachsten Fall, der keine optimalen Ergebnisse liefert, wird das arithmetische Mittel über alle innerhalb des Pixels sichtbaren Objektteile gebildet („*unweighted area sampling*“). Eine andere beliebte Gewichtungsfunktion, die oft bessere Ergebnisse liefert, ist die Dichte der (zweidimensionalen) Normalverteilung.

1.4 Das formale Modell

In diesem Abschnitt sollen die bisher nur grob umrissenen Begriffe einer Szenenbeschreibung sowie auch der Begriff von Darstellungsfehlern so weit formalisiert werden, daß sie eine ausreichend präzise Formulierung und Analyse des randomisierten z-Buffer Verfahrens erlauben.

1.4.1 Das Szenenmodell

Grundlage für die weitere Verarbeitung ist das im Rechner gespeicherte Modell einer dreidimensionalen Szene. Im Laufe der Jahre wurden viele verschiedene Möglichkeiten zur Modellierung dreidimensionaler Szenen entwickelt. Ein Überblick findet sich z.B. in dem Standardwerk [Foley et al. 96]. Zur Echtzeitdarstellung großer Szenen werden in der Regel Oberflächenmodelle

eingesetzt: Nur die Oberflächen der Objekte im Raum sowie deren optische Eigenschaften werden gespeichert und zur Darstellung der Szene verwendet. Die Oberflächen werden dabei aus vielen Instanzen elementarer Oberflächentypen zusammengesetzt.

Im Folgenden wird zwischen zwei Modellierungsebenen unterschieden, dem *Low-Level-Modell* und dem *High-Level-Modell*. Das Low-Level-Modell wird direkt vom Renderingverfahren verarbeitet, dementsprechend ist hier vor allem Effizienz wichtig. Es unterstützt in der Regel nur wenige, einfache Oberflächentypen. Das Rendering dieser Oberflächentypen kann oft durch Spezialhardware unterstützt werden.

Das High-Level-Modell dient dem Modellierer der Szene als Abstraktionsschicht, es soll eine einfache und kompakte Formulierung der Szene erlauben. Hier werden mehr und komplexere Oberflächentypen unterstützt. Außerdem werden bestimmte Typen von Redundanz in der Szene ausgenutzt, um eine kompaktere Darstellung zu erhalten. Dieses Modell muß vor dem Rendering erst in die Low-Level-Darstellung umgesetzt werden. Diese Trennung ist in vielen Standardbibliotheken zu finden (z.B. in Open Inventor [Wernecke 94]).

1.4.1.1 Das Low-Level-Modell

Für das Echtzeitrendering werden meistens Dreiecke als einzige elementare Oberflächentypen eingesetzt, da sich diese einfach und effizient handhaben lassen und sich dazu eignen, komplexere Oberflächen damit anzunähern. Je nach Renderingverfahren werden diese u.U. in unterschiedlichen Datenstrukturen gespeichert. Jedem Dreieck wird eine Normale zugewiesen, die in Richtung der sichtbaren Außenseite der Oberfläche zeigt. Es wird angenommen, daß nur geschlossene Oberflächen in der Szene enthalten sind, daher werden die Innenseiten der Dreiecke beim Rendering nicht dargestellt („*culling*“, siehe [Foley et al. 96]), da diese stets von anderen Außenseiten verdeckt werden.

Definition 1: *Low-Level-Darstellung der Szenengeometrie*

Ein *Dreieck* t ist die konvexe Hülle von drei Punkten im Raum:
$t = \{ \lambda_1 \cdot p_1 + \lambda_2 \cdot p_2 + \lambda_3 \cdot p_3 \mid \lambda_i \geq 0, \lambda_1 + \lambda_2 + \lambda_3 = 1 \}$ ist das *von* $p_1, p_2, p_3 \in \mathbb{R}^3$ *gebildete Dreieck*.

Eine *Szene* $S \subseteq \mathbb{R}^3$ ist eine Menge von n Dreiecken im Raum: $S = t_1 \cup ... \cup t_n$ ist die *von den Dreiecken* $t_i \subseteq \mathbb{R}^3$ *gebildete Szene*. n ist die *geometrische Komplexität* der Szene.

Sei $n(t_i)$ die von den Punkten p_1, p_2, p_3 eines Dreieckes t_i gebildete Normale $(p_2 - p_1) \times (p_3 - p_1)$. Diese sei stets so skaliert, daß $\|n(t_i)\| = 1$ gilt.

Zusätzlich müssen die Oberflächen mit optischen Eigenschaften versehen werden, die die Interaktion der Oberflächen untereinander und mit Lichtquellen in der Szene beschreiben. Auch dabei gibt es sehr viele verschiedene Möglichkeiten zur Modellierung (siehe z.B. [Foley et al. 96], S. 721ff oder [Glassner 95b], S. 541ff). Zur Formulierung des randomisierten z-Buffer Verfahrens ist die folgende Abstraktion ausreichend:

Definition 2: *Schattierungsfunktion*

Die Menge RGB := $[0,1]^3$ sei die Menge aller (auf einem üblichen Computerbildschirm) darstellbaren Farben.

> Sei *farbe*: $S \times (S \backslash \mathbb{R}^3) \rightarrow \text{RGB}$, $(x,v) \mapsto (r,g,b)$ eine Funktion, die jedem Punkt x auf einer Oberfläche für jeden möglichen Betrachtungsstandpunkt v einen Farbwert zuordnet.
>
> Falls *farbe* nicht von v abhängt, heißt die Oberflächenfärbungsfunktion *blickpunktunabhängig*, sonst *blickpunktabhängig* (oft auch als *dynamische Beleuchtung* bezeichnet). Die Schattierungsfunktion heißt *effizient auswertbar* wenn sie sich in O(1), unabhängig von allen Komplexitätsparametern der Szene, berechnen läßt.

Diese Definition berücksichtigt der Einfachheit halber keine halbtransparenten Oberflächen. Für den Echtzeiteinsatz kommen nur effizient auswertbare Schattierungsfunktionen mit hinreichend kleiner Konstante in der Auswertungszeit in Betracht, da die Schattierungsfunktion für ein Bild in der Regel mehrere tausendmal berechnet werden muß. Alle im Echtzeitbereich gängigen Methoden wie das *Phong Beleuchtungsmodell*, *Texture Mapping*, *Environment Mapping* (alle z.B. in [Foley et al. 96] beschrieben) oder auch *Lightmaps* [Zerbe 99] erfüllen diese Bedingung, und können daher prinzipiell auch beim randomisierten z-Buffer verwendet werden. Aufwendige Algorithmen zur Berechnung der Oberflächenschattierung, wie rekursives Raytracing ([Kay 79], [Whitted 80]), sind nicht hinreichend effizient und sind daher in der Regel nicht unter Echtzeitbedingungen einsetzbar.

Im Spezialfall einer statischen, blickpunktunabhängigen Schattierungsfunktion ist es möglich, alle Oberflächenfärbungen im voraus zu berechnen, was spürbare Laufzeiteinsparungen bringen kann. In den meisten Beispielszenen, die später in Kapitel 5 untersucht werden, wird diese Optimierung angewandt.

1.4.1.2 Das High-Level-Modell

Für den menschlichen Modellierer wäre es mühsam und unübersichtlich, Szenen nur mit Dreiecken zu beschreiben. Daher wird eine weitere Abstraktionsschicht notwendig. Das hier verwendete High-Level-Modell ist eine Variante des Szenengraphmodells, das z.B. in [Foley et al. 96] (S. 285ff) beschrieben wird und auch in vielen Softwaresystemen verwendet wird (z.B. in Open Inventor [Wernecke 94]). Da das High-Level-Modell für das randomisierte z-Buffer Rendering nur am Rande von Bedeutung ist (es wird bei der Instantiierung von Verteilungsstrukturen verwendet, siehe Kapitel 3.3.6.2), wird es hier nur kurz beschrieben.

Die Szene wird als azyklisch gerichteter Graph beschrieben. Es gibt 3 Typen von Knoten: *Listen*, *Referenzen* und *Grundobjekte*. Grundobjekte können neben Dreiecken auch komplexere Objekte wie Kugeln, Rotationskörper oder Splineoberflächen sein. Die anderen Knoten sind entweder Listen, die Verweise auf andere Knoten enthalten, oder Referenzen, die genau einen Verweis auf einen weiteren Knoten enthalten. Kein Knoten darf in mehr als einer Liste enthalten sein, eine Referenz zeigt immer auf einen Knoten, der bereits in einer Liste enthalten ist. Der Graph hat eine Liste als Wurzelknoten.

Jeder Knoten enthält eine homogene 4×4-Matrix, die die *relative Transformation* des Knotens beschreibt: Für jeden Knoten bildet die Hintereinanderausführung (Matrizenmultiplikation) aller darüberliegenden Transformationen, angefangen beim Knoten, aufsteigend zur Wurzel, die *absolute Transformation* des Objektes.

Die Listen im Szenengraphen dienen zur logischen Gruppierung, die das Modellieren erleichtern soll. Die Referenzen beschreiben neue *Instanzen* von bereits definierten Objekten: Für eine weitere Instanz eines Objektes, das schon einmal durch einen Teilgraphen beschrieben wurde und

sich nur durch eine Transformation vom Original unterscheidet, muß nur noch ein Referenzknoten mit der entsprechenden Matrix zusätzlich abgespeichert werden.

Auf diese Art und Weise läßt sich viel Speicherplatz einsparen, da Szenen oft viel Redundanz dieser Art aufweisen. Man beachte, daß referenzierte Objekte auch ihrerseits wieder Referenzen enthalten können. Die Instantiierung auf mehreren Ebenen erlaubt mit wenigen Informationen sehr große Szenen zu beschreiben. Der Modellierer muß dazu allerdings beim Entwurf der Szene von entsprechenden Referenzen Gebrauch machen.

Es ist wünschenswert, daß auch die Low-Level-Darstellung für ein Renderingverfahren unter Verwendung dieser Instantiierungen gespeichert werden kann, so daß der Speicherbedarf durch die Umwandlung nicht zu stark wächst. Abbildung 4 zeigt eine Beispielszene aus vier Autos mit jeweils vier Rädern, die durch eine Szenengraphdarstellung speichereffizient repräsentiert wurde.

Im Falle dynamischer Szenen kann die durch das High-Level-Modell gegebene Strukturierung zusätzliche Optimierungen ermöglichen. Wenn nur die Transformationsmatrix eines Knotens geändert wird (z.B. wenn ein Auto, modelliert in einem Teilgraphen, durch eine Szene fährt), kann dies u.U. besonders schnell in den vorberechneten Datenstrukturen berücksichtigt werden. Das in Kapitel 3.3.6.2 beschriebene Instantiierungsschema erlaubt in einem solchen Fall eine Anpassung der Datenstrukturen mit konstantem Zeitaufwand.

1.4.2 Das Kameramodell

Im Rahmen dieser Arbeit wird stets eine Zentralprojektion zur Abbildung der Objekte vorausgesetzt. Eine detaillierte Beschreibung der verschiedenen Projektionen und ihrer Berechnung findet sich in [Foley et al. 96], S. 229ff. Die Zentralprojektion wird definiert durch ein Kamerakoordinatensystem aus drei orthonormalen Koordinatenvektoren und einem Ursprungspunkt, welche die Kameraposition und Blickrichtung angeben und einem Brennweitenfaktor d, welcher die Vergrößerung des Bildausschnittes bestimmt. Der Ursprungspunkt bestimmt den Brennpunkt der Kamera, die beiden ersten Koordinatenvektoren spannen im Abstand d zum Brennpunkt die Bildebene auf. Der dritte Koordinatenvektor steht senkrecht darauf und gibt die

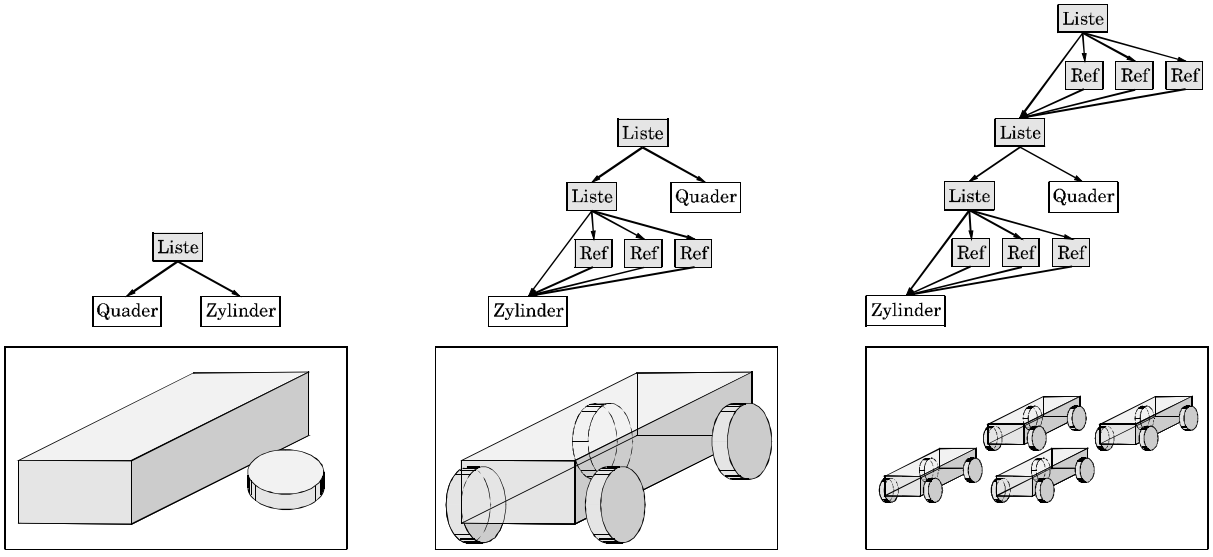


Abbildung 4: Instantiierung im Szenengraphen

Blickrichtung an (siehe Abbildung 5). Das Skalarprodukt dieses Vektors mit dem Verbindungsvektor zwischen Ursprung und Objektpunkt gibt den Abstand des Objektes in Blickrichtung, im folgenden *Tiefenwert* genannt, an und dient in einigen Algorithmen zur Erkennung verdeckter Flächen.

Im Fall, daß die Kamera im Ursprung liegt und in Richtung der positiven z-Achse blickt, erhält man die Projektion mittels Strahlensätzen als:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} \mapsto \begin{pmatrix} d \cdot \frac{x}{z} \\ d \cdot \frac{y}{z} \end{pmatrix}$$

Mit x' und y' erhält man die Bildschirmkoordinaten. Der Punkt (0,0) liegt in der Mitte des Bildschirms; die Transformation auf Pixel wird so vorgenommen, daß das Rechteck $[-w/2, w/2] \times [-h/2, h/2]$ den Bildschirm bildet, wobei w die Breite und h die Höhe des Bildes in Pixeln angibt; ein Pixel hat also genau eine Seitenlänge von 1 in beiden Richtungen. Bei einer anderen Skalierung würde sich die Bedeutung des Brennweitenparameters d proportional dazu ändern.

Im allgemeinen Fall, bei beliebigen Kameraabstand- und Blickpunkt, muß zuvor die Szene entsprechend um den negativen Kamerastandpunkt verschoben und mit dem Inversen des Kamerakoordinatensystems transformiert werden. Die gesamte Projektion kann durch eine homogene 4×4-Matrix dargestellt werden.

Die unendliche Pyramide mit Spitze im Projektionszentrum, deren vier Kanten durch die Ecken des Bildschirmrechtecks in der Bildebene laufen, wird als *Sichtpyramide* bezeichnet. Oft wird die Sichtpyramide durch zusätzliche Beschränkung des Tiefenwertes mit einer Unter- und einer Obergrenze zu einem endlichen Pyramidenstumpf („*viewing frustum*“) reduziert.

Die genaue Definition der Transformation wird bei verschiedenen Autoren leicht unterschiedlich dargestellt und auch in verschiedenen Grafikbibliotheken unterschiedlich definiert. Die hier verwendete Formulierung vereinfacht die Berechnungen in Kapitel 3.

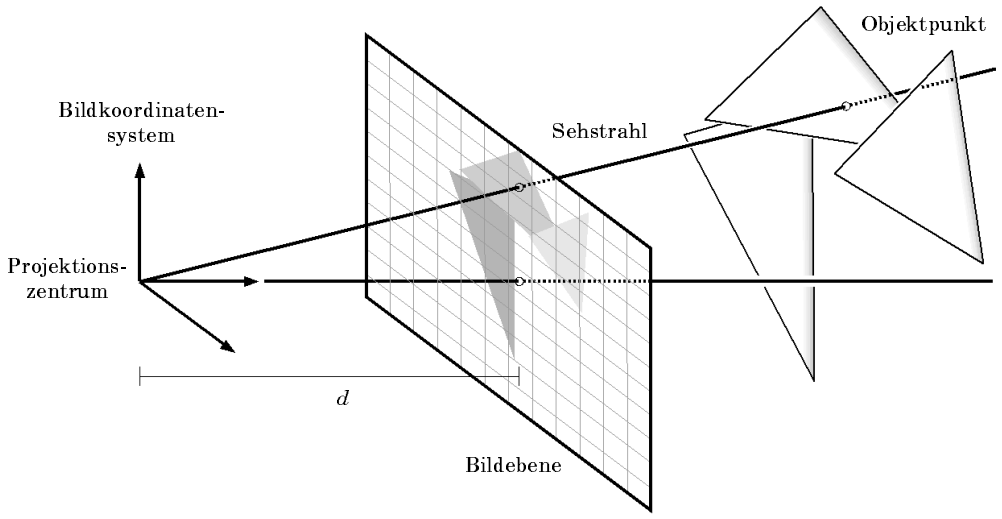


Abbildung 5: Zentralprojektion

1.4.3 Ein „korrektes" Bild eines allgemeinen Rendering Verfahrens

Zur analytischen Beurteilung der Bildqualität eines approximativen Renderingverfahrens muß zunächst erst definiert werden, was ein „korrektes" Bild ist. Leider können sich die Resultate verschiedener „exakter" Renderingverfahren bei pixelweisem Vergleich durchaus unterscheiden, je nachdem, welche sub-Pixel Anteile (siehe Abbildung 3) ausgewählt werden. Daher folgende Definition:

Definition 3: *ein korrektes Bild (ohne Anti-Aliasing)*

Sei *bild*: $[0, w] \times [0, h] \rightarrow \text{RGB}, (x,y) \mapsto (r,g,b)$ die Funktion, die jedem (nicht notwendig ganzzahligen) Punkt auf der Bildfläche der Breite w und der Höhe h den Farbwert des darunter sichtbaren Objektpunktes zuordnet. Falls kein Objekt an Position (x,y) vorhanden ist, liefere $bild(x,y)$ eine feste Hintergrundfarbe.

Sei *rendered*: $\{0,1, \ldots, w-1\} \times \{0,1, \ldots, h-1\} \rightarrow \text{RGB}$ die Funktion, die jedem (ganzzahligen) Pixel das Ergebnis des Renderingverfahrens zuordnet.

Ein Renderingergebnis *rendered* heißt *korrekt* (*ohne Anti-Aliasing*) wenn für alle Pixel (x,y) gilt:

$$rendered(x,y) \in \left\{ bild(x',y') \,\middle|\, x' \in [x, x+1], y' \in [y, y+1] \right\}$$

Ein Bild gilt also als korrekt, wenn die auf dem Pixel wiedergegebene Farbe sich tatsächlich auch unter den unter dem Pixel befindlichen Farben befindet (in Abbildung 3 könnte also für den gezeigten Pixel irgendeine der gezeigten Farben ausgewählt werden). Die meisten Verfahren treffen eine solche Auswahl systematisch, dies kann allerdings von Nachteil sein, da es zu Interferenzmustern mit systematischen Szenendetails, also zu Aliasing, führen kann.

Definition 4: *das Mittelwertbild*

Ein Renderingverfahren liefert das korrekte *Mittelwertbild*, falls gilt:

$$rendered(x,y) = \int_{y}^{y+1} \int_{x}^{x+1} bild(u,v)\, du\, dv$$

(Es kann für dieses Anwendungsgebiet ohne Probleme angenommen werden, daß die Funktion *bild* integrierbar ist.)

Man beachte, daß das Mittelwertbild im allgemeinen keine optimale Bildqualität liefert; es sind bessere Anti-Aliasing Methoden bekannt. Ein Beispiel für die möglichen Artefakte ist in Abbildung 24 (d) (S. 72) zu sehen. In einem allgemeineren Fall wird man ein gewichtetes Mittel verwenden:

Definition 5: *das gewichtete Mittelwertbild zur Filterfunktion f*

Ein Renderingverfahren liefert *das gewichtete Mittelwertbild zur Filterfunktion f* für eine Funktion f: $\mathbb{R}^2 \to \mathbb{R}^{\geq 0}$ falls gilt:

$$rendered(x,y) = \int_0^h \int_0^w filter(u-x, v-y) \cdot bild(u,v)\, du\, dv$$

(Auch hier sei die Existenz der Integrale vorausgesetzt.)

Mit signaltheoretischen Überlegungen können verschiedene Filterfunktionen bestimmt werden, die verschiedene Kompromisse zwischen Aufwand, Aliasing und Schärfe bieten ([Foley et al. 96], [Glassner 95a]). Häufig benutzt wird die Dichtefunktion einer zweidimensionalen Gaußschen Normalverteilung mit Erwartungswert (x, y) und vorgegebener Varianz (z.B. in [Cook 91]). Abbildung 24 (f) auf Seite 72 wird zeigen, daß damit eine deutlich bessere Bildqualität als bei einer arithmetischen Mittelwertbildung erzielt werden kann.

Probleme bei der Beurteilung der Bildqualität

Idealerweise würde man auch gerne ein Maß zur Verfügung haben, welches die Qualität eines u.U. nicht korrekten Bildes im Sinne einer quantifizierbaren Abweichung beschreibt. Dies stellt sich allerdings als problematisch heraus ([Stollnitz et al. 96], S. 46). Naheliegende Maße, wie z.B. eine Maximumsnorm oder eine l_2-Norm der pixelweisen Differenz von zwei Bildern lassen sich schon durch einfache Beispiele „austricksen“: So würde z.B. einer Näherung eines vollständig grauen Bildes durch ein schwarzweiß Raster (wie in einer Zeitung) eine sehr starke Abweichung bescheinigt, im offenen Widerspruch zum optischen Eindruck. [Heckbert und Garland 94] schlagen eine Anreicherung der l_2-Norm um perzeptuelle Faktoren vor und verweisen darauf, ggf. doch den menschlichen Betrachter zur Beurteilung der Bildqualität von approximativen Renderingverfahren heranzuziehen.

[Stollnitz et al. 96] (Kapitel 5) erläutern eine Methode auf zum Vergleich von Bildern auf Basis quantisierter Haarbasiswavelettkoeffizienten, die sehr gute Ergebnisse bei der Suche von Bildern in Datenbanken liefert. Die vorgeschlagene Metrik erscheint aufgrund ihres grobgranularen und diskreten Charakters allerdings wenig geeignet, um feine Abweichungen in Bildern zu quantifizieren.

Aufgrund der vielfältigen Unzulänglichkeiten quantitativer Qualitätsmaße wird im weiteren daher die Bildqualität nicht quantitativ angegeben. Im Rahmen der analytischen Betrachtung wird versucht, die Form der vom randomisierten z-Buffer produzierten Artefakte zu beschreiben. Kriterium sind dabei Unterschiede zu einem nach Definition 3 korrekten Bild. Außerdem werden die Renderingergebnisse bei verschiedenen Testszenen beschrieben. In Anhang C finden sich einige als Farbbilder abgedruckt, so daß man sich einen augenscheinlichen Eindruck von der Qualität verschaffen kann.

2 Bekannte Verfahren

In diesem Kapitel wird der Stand der Technik bei konventionellen und approximativen Renderingalgorithmen zur Echtzeitdarstellung beschrieben. In einem speziellen Abschnitt wird auf randomisierte Techniken eingegangen.

Zuerst wird in Abschnitt 2.1 der z-Buffer Algorithmus beschrieben, der heute Grundlage der meisten Walkthroughsysteme ist. Danach werden in den Abschnitten 2.2 - 2.5 verschiedene Methoden zur Verbesserung der Skalierbarkeit des Algorithmus bei komplexen Szenen beschrieben. Im Abschnitt 2.6 wird das Raytracing als algorithmische Alternative zum z-Buffer Verfahren vorgestellt. In Abschnitt 2.7 werden dann die ungelösten Probleme beim heutigen Stand der Technik zusammengefaßt. Abschnitt 2.8 beschreibt schließlich einige randomisierte Methoden, die nicht aus dem Echtzeitbereich stammen, die aber den Entwurf des randomisierten z-Buffer Verfahrens beeinflußt haben.

2.1 Der z-Buffer Algorithmus

2.1.1 Das Verfahren

Das bekannteste und wohl auch im Echtzeitbereich am häufigsten verwendete Renderingverfahren ist der z-Buffer Algorithmus [Catmull 74]. Er arbeitet wie folgt:

- Als Datenstruktur steht der Bildpuffer (ein zweidimensionales Feld aus Farbwerten) sowie ein Tiefenpuffer („z-Buffer“, ein weiteres zweidimensionales Array der gleichen Größe mit skalaren Einträgen) zur Verfügung. Jedem Pixel im Bildpuffer ist unter den gleichen Indexkoordinaten ein Tiefenwert im Tiefenpuffer zugeordnet (dies ist schematisch in Abbildung 6 dargestellt).
- Der Bildpuffer wird auf die Hintergrundfarbe und der Tiefenpuffer auf einen maximalen Wert initialisiert.
- Alle Dreiecke der Szene werden in willkürlicher Reihenfolge aufgegriffen und ihre Eckpunkte auf die Bildebene projiziert.

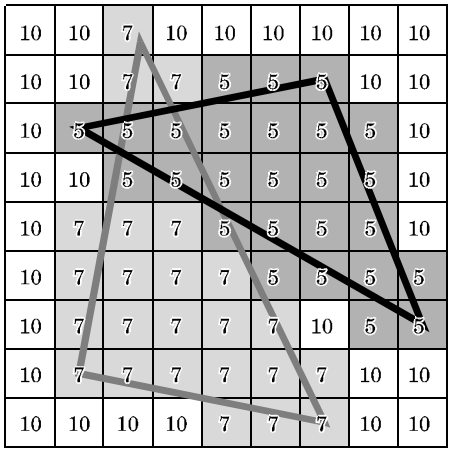


Abbildung 6: der Tiefenpuffer

- Die Pixel innerhalb der Dreiecke werden berechnet (Rasterung) und für jeden Pixel wird die Entfernung des Objektpunktes auf dem Dreieck zum Beobachter (bzw. der Tiefenwert, siehe Abschnitt 1.4.2) bestimmt.
- Ist dieser Wert kleiner als der im Tiefenpuffer gespeicherte, so wird dort der neue Wert eingetragen, die Farbe des Objektpunktes wird berechnet und im Bildpuffer abgelegt. Andernfalls wird der Punkt ignoriert.

Der Algorithmus bestimmt das Bild und insbesondere die richtigen Verdeckungen der Punkte untereinander also dadurch, daß in jedem Pixel ein minimaler Tiefenwert mit dem dazugehörigen Farbwert durch Suche über alle Dreiecke bestimmt wird.

Aufwand: Sei n die Anzahl von Dreiecken und a die von ihnen insgesamt bedeckte Pixelfläche (vor der Berücksichtigung von Verdeckungen). Dann hat der z-Buffer Algorithmus einen Aufwand von $O(n + a)$ [Heckbert und Garland 94].

Im Gegensatz zu einigen konkurrierenden Verfahren kommt der z-Buffer Algorithmus ohne Sortierung der Objekte aus[2]. Der Algorithmus kann leicht in Hardware implementiert werden, und er läßt eine Vielzahl von Schattierungsfunktionen[3] zu, die ihrerseits auch von der Hardware unterstützt werden können (insbesondere Texturen). Diese Vorteile haben den Algorithmus quasi zum Standardverfahren im Echtzeitbereich gemacht.

2.1.2 Der Stand der Technik bei z-Buffer Rendering

Die Leistung von z-Buffer Implementationen läßt sich naheliegenderweise durch die *Fülleistung* und die *Transformationsleistung* charakterisieren: Die Fülleistung gibt an, wie viele Pixel (Aufwandsparameter a) gefüllt werden können, wenn jeglicher Überhang für die Verarbeitung der Dreiecke vernachlässigt wird. Die Transformationsleistung gibt an, wie viele Dreiecke (Aufwandsparameter n) verarbeitetet werden können, wenn die Zeit zum Beschreiben der Pixel vernachlässigt wird. Zur Messung dieses Parameters wird in der Regel ein günstiger Fall (reguläres Gitter, d.h. 2 Dreiecke pro Eckpunkt im Schnitt) für die Anordnung der Dreiecke angenommen.

Zu dem Zeitpunkt, an dem diese Arbeit geschrieben wird, ist z.B. „Consumer"-Hardware für handelsübliche PCs zur Unterstützung von texturiertem z-Buffer Rendering erhältlich, die vom Hersteller mit einer Transformationsleistung von mehreren Millionen Dreiecken und einer Fülleistung von mehreren 100 Millionen Pixeln pro Sekunde spezifiziert wird ([Bertuch 99], Seite 204). In der Praxis sollen Modelle in der Größenordnung von 100.000 Dreiecken damit in Echtzeit darstellbar sein. In naher Zukunft sind noch weitere Steigerungen bei z-Buffer Hardware zu erwarten. Da die Transformation der Eckpunkte aufwendige Fließkommaarithmetik erfordert, ist allerdings zu erwarten, daß auch in Zukunft die Transformationsleistung weit unterhalb der Fülleistung bleibt.

Man sieht an diesen Werten auch, daß algorithmische Verbesserungen erst für relativ große Szenen notwendig werden. Dies führt dazu, daß auch eine effiziente Speicherung der Szene wichtig wird: Eine Million Dreiecke verbrauchen bei grob geschätzten 50 Bytes pro Dreieck (mit Texturkoordinaten, Normalen, Farb- und Transparenzwerten) etwa 50 MB Speicher. Eine Szene, die ein Vielfaches dieser Komplexität aufweist, läßt sich daher kaum im Hauptspeicher un-

[2] Sortieren wird erst wieder notwendig, wenn auch halbtransparente Oberflächen dargestellt werden sollen.
[3] quasi alle, die unter das Modell von Definition 2 fallen und sich ausreichend schnell berechnen lassen.

terbringen. Eine mögliche Lösung für die Speicherplatzprobleme ist die Ausnutzung von Redundanz bei der Beschreibung der Szene, z.B. durch die Vermeidung der Speicherung mehrerer ähnlicher Instanzen gleicher Objektgruppen oder durch die Beschreibung der Szene mit komplexeren Grundprimitiven (wie Splineoberflächen o.ä.).

2.1.3 Grenzen des z-Buffer Verfahrens bei Walkthroughanimationen

Bei der Walkthroughanimation können verschiedene Szenarien auftreten, die dazu führen, daß der Rechenzeitbedarf des z-Buffer Algorithmus die Echtzeitschranken verläßt:

- **Zu hohe Verdeckung:** Eine Szene mit nicht notwendig vielen geometrischen Details aber großer projizierter Fläche a. Das mehrfache Auffüllen der Pixel überfordert die Hardware. Je nach Auflösung und Hardware liegt die Grenze schon bei 5-10 bildschirmfüllenden Dreiecken.
  Typische Szenen: Architektonische Modelle großer Gebäude mit vielen Wänden.
- **Zu viele Details:** Schon der sichtbare Teil der Szene enthält zu viele, nicht notwendig große Dreiecke. Die vielen Details haben nur noch in ihrer Summe, aber kaum noch einzeln Einfluß auf das Renderingergebnis.
  Dabei sind beispielsweise folgende zwei Varianten denkbar:
  1. Die Szene besteht nur aus wenigen sehr komplexen Objekten (CAD-Werkstücke, Modelle aus 3D-Scannern).
  2. Die Szene besteht aus vielen einfachen Objekten, die aber in ihrer Gesamtheit sehr viele geometrische Details aufweisen.
  Typische Szenen hier: Stadt- und Waldszenen. Zum Horizont hin werden bei solchen Szenen bei entsprechenden Blickpunkten immer mehr Objekte in immer kleinerem Maßstab sichtbar.

Oft treten beide Probleme (Verdeckung und Überdetaillierung) zusammen auf. Für beide Phänomene sind Beschleunigungstechniken entwickelt worden. Das randomisierte z-Buffer Verfahren ist nur dazu in der Lage, Szenen mit sehr vielen Details schnell darzustellen. Für Szenen mit hoher Verdeckung wird asymptotisch kein Gewinn erzielt. Da wesentlich schlechtere Konstanten auftreten, ist in der Praxis sogar mit einem erhöhten Aufwand zu rechnen. Das Verfahren läßt sich allerdings relativ leicht mit anderen Methoden zur Unterdrückung verdeckter Flächen, z.B. PVS-Methoden (siehe Abschnitt 2.5.1.1), kombinieren.

2.1.4 Klassifizierung der bekannten Verbesserungsmöglichkeiten

Wie bereits eingangs erläutert, gibt es prinzipiell zwei Möglichkeiten, eine höhere Renderingleistung zu erhalten:

1. Vereinfachung der Szene vor dem Rendering: Das Szenenmodell wird in einem Approximations- oder Verdeckungserkennungsschritt so vorverarbeitet, so daß eine für ein konventionelles Verfahren handhabbare Komplexität entsteht. Aufgrund der Verfügbarkeit leistungsfähiger Hardwares wird diese Vorverarbeitung heute fast immer mit z-Buffer Rendering kombiniert, obwohl auch eine Kombination mit anderen Verfahren (z.B. Verfahren, die auf Sortieren der Dreiecke basieren) möglich ist. [Shade et al. 96] teilen die Beschleunigungsverfahren in folgende Klassen ein:

- „*level-of-detail modelling*“: Darstellung der Objekte der Szene durch verschieden detaillierte Dreiecksnetze, abhängig von ihrer Größe auf dem Bildschirm.
- „*image-based rendering*“: Ersetzen von Objekten durch Rasterbilder, die bereits gerenderte Szenenteile enthalten. Neben der Verwendung von Rasterbildern sind hier auch allgemeinere Ersetzungsstrategien denkbar.
- „*visibility culling*“: Verhindern, daß nicht sichtbare Szenenteile bearbeitet werden.

2. Verwendung eines Renderingverfahrens mit outputsensitiven Eigenschaften: In diesem Bereich sind unter den bekannten Ansätzen vor allem die verschiedenen Varianten des Raytracingverfahrens interessant. Auch das randomisierte z-Buffer Verfahren fällt in diese zweite Kategorie, allerdings ähnelt das Vorgehen in einigen Aspekten dem klassischen z-Buffer, daher die Namensgebung. Dadurch, daß das Verfahren zum konventionellen z-Buffer Algorithmus kompatible Tiefenpufferinformationen erzeugt, kann es auch verwendet werden, um nur einige Teile einer Szene, die eine zu hohe Komplexität aufweisen, zu approximieren, während grobe detaillierte Teile der Szene weiterhin mit dem gewöhnlichen z-Buffer Algorithmus behandelt werden können. Auch eine Kombination mit anderen z-Buffer basierten Vereinfachungs- und Verdeckungserkennungsmethoden ist denkbar.

Im folgenden werden einige wichtige Methoden aus der Literatur in der oben aufgeführten Reihenfolge beschrieben.

2.2 „Level-Of-Detail Modelling“-Verfahren

Die Objekte der Szene werden für das Rendering in der Regel zunächst durch Dreiecksnetze angenähert; diese lassen sich besonders effizient (auch in Hardwareimplementationen) verarbeiten. Die Geometrie der meisten Arten von Objekten wie z.B. mathematische Kugeln oder natürliche Bäume lassen sich nicht exakt durch Dreiecke darstellen. Je mehr Dreiecke verwendet werden können, um so genauer läßt sich allerdings die Geometrie approximieren. Eine offensichtliche Beobachtung ist, daß Approximationsfehler bei Objekten, die relativ klein im Bild erscheinen, weniger stark zu Tage treten als bei solchen, die sehr groß im Bild zu sehen sind. Da die Größe allerdings vom Betrachtungsstandpunkt, der beim Walkthrough frei gewählt werden kann, abhängt, ist eine feste Approximation durch Dreiecke ungünstig: Bei großer Entfernung verursacht sie zu hohen Aufwand beim z-Buffer Rendering, bei geringer Entfernung wird das Objekt grob und ungenau dargestellt.

„*level-of-detail*“-Methoden versuchen, die Granularität der Dreiecksnetze der Objekte dynamisch an die sichtbare Objektgröße anzupassen. Nach der Anpassung können die entsprechenden Dreiecksnetze direkt durch einen klassischen z-Buffer Algorithmus verarbeitet werden. Dabei muß neben der Anpassung der Geometrie auch auf die Anpassung von Oberflächenattributen (Schattierung) geachtet werden, da diese in der Regel fest mit der Geometrie verknüpft sind.

Nicht zuletzt wegen der hohen Qualität und Leistung bestehender z-Buffer Systeme sind auf diesem Gebiet in den letzten Jahren große Anstrengungen unternommen und beachtliche Erfolge erzielt worden. Eine systembedingte Schwäche aller dieser Verfahren ist allerdings, daß sie sich nicht für beliebige Objekte eignen: Objekte, die aus vielen unabhängigen Teilen mit bereits sehr einfacher Geometrie bestehen, wie z.B. ein Lattenzaun, eine Wiese mit Millionen von Grashalmen oder ein Baum mit vielen Blättern, können nicht wesentlich vereinfacht werden, ohne die Geometrie in ihrer Bedeutung stark zu verändern; dem Baum müßten z.B. die vielen einzel-

nen Blätter entfernt werden, um ihn mit sehr wenigen Dreiecken darstellen zu können. Bei einfachen, zusammenhängenden Dreiecksgittern, wie z.B. denen von Gesichtern, Tieren oder Höhenkarten, sind dagegen sehr starke Vereinfachungen bei geringem Qualitätsverlust möglich.

2.2.1 Manuelles „Level-of-Detail"

Die älteste und wahrscheinlich auch am meisten gebrauchte Methode ist die manuelle: Von allen wichtigen Objekten einer Szene werden mehrere (etwa 3-5) Modelle unterschiedlicher Komplexität von Hand entworfen. Jedem Modell wird ein Abstandsbereich zugeordnet, in dem dieses verwendet werden soll.

Die Methode hat den Vorteil, daß sie sehr einfach implementiert werden kann und daher in den meisten Grafikbibliotheken schon direkt unterstützt wird (z.B. in Open Inventor [Wernecke 94]). Der Nachteil ist, daß alle Objekte von Hand erstellt werden müssen, was viel Arbeit machen kann. Außerdem ist die Anpassung des Detaillierungsgrades durch die wenigen zur Verfügung stehenden Modelle nur sehr grob möglich und beim Wechsel der Darstellung ist das Umschalten daher meist deutlich sichtbar, was den Betrachter irritieren kann.

2.2.2 „Mesh Simplification"-Verfahren

Ausgehend von den Problemen beim manuellen *„level-of-detail modeling"* wurden Verfahren entwickelt, die das Erzeugen von verschieden detaillierten Modellen automatisieren (*„mesh simplification"*). Eingabe für automatische Verfahren ist entweder eine High-Level-Szenenbeschreibung, die erst noch in Dreiecke zerlegt werden muß, oder ein fertiges, hochdetailliertes Dreiecksnetz, daß auf einen vorgegebenen Detaillierungsgrad angepaßt werden soll.

Eine Übersicht über die bekannten Verfahren findet sich z.B. in [Puppo und Scopigno 97], [Erikson 96] oder [Heckbert und Garland 97]. Die folgende Darstellung orientiert sich an diesen Arbeiten. Da in den letzten Jahren eine große Fülle an Algorithmen auf dem Gebiet publiziert wurde, können hier nur einige Verfahren exemplarisch vorgestellt werden. Die genannten Übersichtsartikel bieten einen umfassenderen Überblick.

Einige Verfahren ermöglichen auch einen fließenden Übergang zwischen verschieden detaillierten Modellen (*„multi resolution modeling"*). Datenstrukturen, die viele verschieden detaillierte Versionen eines Dreiecksnetzes gleichzeitig bereithalten, werden im folgenden als *Multi Resolution Modelle* bezeichnet. [Puppo und Scopigno 97] beschreiben insbesondere ein allgemeines Modell für Multi Resolution Modelle, das quasi alle bekannten Verfahren beschreiben kann und eine Einschätzung der Performance für verschiedene Aufgaben ermöglicht.

Optimale Verfahren und Heuristiken

Nach [Heckbert und Garland 97] ist das Bestimmen einer bezüglich der Maximumsnorm optimalen polygonalen Näherungsoberfläche bereits für einfachere Spezialfälle NP-hart und damit auch das allgemeinere Problem für parametrische oder beliebige Dreiecksnetze. Für andere Normen ist die Frage noch nicht geklärt. Bekannte Polynomialzeitalgorithmen zur Approximation mit beweisbaren Schranken für die Abweichung vom Optimum sind ebenfalls für praktische Problemgrößen zu langsam. Daher werden bei den in der Praxis verwandten Methoden, von denen im folgenden einige dargestellt werden, entweder heuristische Approximationsstrategien

angewandt oder der Suchraum für die Approximation so stark eingeengt, daß eine in diesem Sinne optimale Lösung effizient berechenbar ist (z.B. bei den Waveletmethoden).

Klassifizierung

Nach [Erikson 96] lassen sich die Verfahren grob danach klassifizieren, ob sie die Topologie der Eingabe erhalten oder ob sie diese auch vereinfachen können. [Heckbert und Garland 97] unterscheiden bei den verarbeiteten Oberflächentypen zwischen parametrischen Oberflächen, triangulierten Mannigfaltigkeiten und allgemeinen Oberflächen. Algorithmen für die beiden ersten Typen von Eingaben verhalten sich in der Regel dementsprechend auch topologieerhaltend, während letztere auch in der Lage sind, die Topologie der Oberflächen zu verändern. Ob dies wünschenswert ist, hängt natürlich von der Anwendung ab. Im Walkthroughbereich sind für weit entfernte Objekte oft sehr starke Vereinfachungen notwendig, und solche lassen sich oft nur mit topologievereinfachenden Verfahren erzielen.

Des weiteren unterscheiden sich die Verfahren darin, wie die Approximationsfehler bewertet werden: Üblich sind z.B. die Maximumsnorm oder die 2-Norm bei parametrischen Verfahren und die Hausdorfdistanz bei allgemeineren Eingaben. [Puppo und Scopigno 97] führen neben diesen Maßen noch eine ganze Anzahl weiterer auf. Je nach Aufwand an Rechenzeit, den man betreiben möchte, können die Maße exakt berechnet oder mehr oder weniger genau geschätzt werden. Dementsprechend gibt es Verfahren, die den Approximationsfehler garantieren, während andere nur die ungefähre Größenordnung einhalten können. Es ist auch möglich, den aktuellen Blickpunkt in die Fehlerberechnung einfließen zu lassen, was für ausgedehnte Objekte wie Höhenkarten von Geländedaten oft wichtig ist. Des weiteren ist für viele Anwendungen wünschenswert, daß auch die Fehler von Oberflächenattributen, wie Normalen, Farben oder Texturkoordinaten berücksichtigt werden, da diese im späteren Bild neben den geometrischen Fehlern ebenfalls erheblichen Einfluß auf die Bildqualität haben können.

2.2.2.2 Parametrische Oberflächen

Der einfachste Fall ist der, daß für die Oberfläche eine Parametrisierung in Form einer Funktion s: $\mathbb{R}^2 \supseteq P \rightarrow \mathbb{R}^3$ bekannt ist. In dem Fall läßt sich das Problem reduzieren auf eine zweidimensionale Triangulierung des Parameterbereiches P und die Bestimmung geeigneter Koordinaten für die dabei entstehenden Eckpunkte mit Hilfe der Funktion s. Diese Form von Algorithmen wird insbesondere dann verwendet, wenn eine High-Level-Beschreibung einer Szene trianguliert werden soll, da viele High-Level-Modellierungstechniken mit parametrischen Oberflächen arbeiten (z.B. Spline Oberflächen, Rotationskörper u.a. [Foley et al. 96]).

In diesem Abschnitt werden Methoden erläutert, die sich speziell für parametrische Oberflächen eignen:

Gittermethoden

Die einfachste Methode ist die, den Parameterbereich mit Hilfe eines Gitters mit uniformen Abständen zu diskretisieren (aus jedem quadratischen Bereich entstehen dabei 2 Dreiecke) und für die entstehenden Eckpunkte die Funktion s zu berechnen. Die Methode ist sehr schnell und einfach zu implementieren, liefert allerdings wegen der fehlenden Adaptivität schlechte Resultate bei nicht gleichmäßig gekrümmten Oberflächen. Außerdem ist eine Berücksichtigung des Blickpunktes innerhalb eines Dreiecksgitters kaum möglich, so daß sie sich nicht für Objekte mit großem Tiefenbereich (wie z.B. Höhenkarten von Geländedaten) eignet.

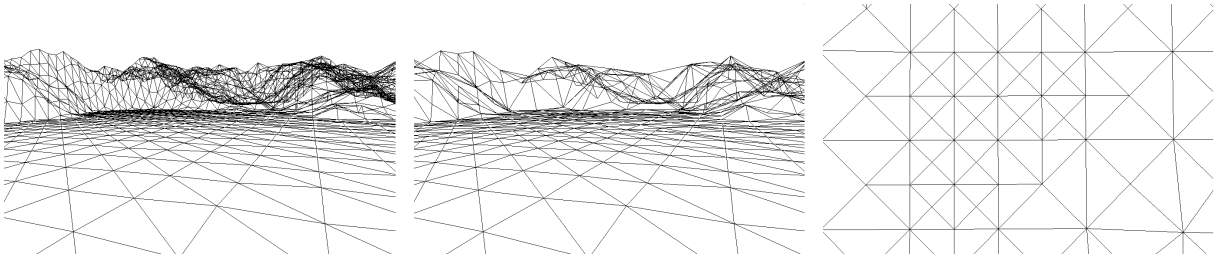
Hierarchische Unterteilung

Bei diesen Methoden wird der Parameterbereich, ausgehend von einer einfachen Form (Quadrat, Dreieck), rekursiv unterteilt. Kompliziert geformte Parameterbereiche müssen daher zuerst in einfache Teilbereiche zerlegt werden.

Balancierte Quadtree Meshes: [de Berg et al. 97] beschreiben eine Methode basierend auf Ergebnissen von [Bern et al. 90] zur Erzeugung eines nicht-uniformen Dreiecksnetzes mit Hilfe eines balancierten Quadtrees. Die Methode läßt sich leicht auf das Vereinfachungsproblem anwenden: Der quadratische Parameterbereich wird mit Hilfe eines Quadtrees rekursiv in kleinere Quadrate unterteilt. Die Unterteilung stoppt, sobald die Funktion s über dem Bereich hinreichend genau angenähert wird, wobei verschiedene Fehlermaße benutzt werden können.

Danach wird der Quadtree balanciert, was dazu führt, daß jede Kante nur durch maximal einen Eckpunkt unterbrochen wird. Danach kann durch eine einfache Fallunterscheidung in 16 Fälle für jede Konstellation in einer Box eine lokale Triangulierung bestimmt werden, die sich nahtlos zusammensetzen läßt. Falls mehrere Parameterbereiche stetig aneinanderstoßen, muß die Balancierungsbedingung über die entsprechenden Kanten hinaus aufrechterhalten werden.

Die Methode läßt sich relativ einfach implementieren und ist bei einem geeigneten Fehlermaß sehr effizient. Die Erweiterung zu einem Multi Resolution Modell ist ebenfalls einfach. Die Tiefe des Quadtrees kann dazu dynamisch angepaßt und die Balancierung dabei aufrechterhalten werden. Der Nachteil ist die reguläre Unterteilung: Bei anisotropischer Verzerrung (z.B. die perspektivische Verzerrung bei der Projektion einer Ebene, siehe Abbildung 7b) werden in der Regel zu viele Dreiecke erzeugt, da diese prinzipiell symmetrisch sind (siehe Abbildung 7c).



a) Gelände (Höhenkarte) als reguläres Gitter

b) vereinfacht mit einer blickpunktabhängigen Quadtreemethode

c) Quadtree Mesh: Ansicht von oben (Ausschnitt)

Abbildung 7: Vereinfachung mit einem balanciertem Quadtree

„Multiresolution Analysis“ mit Wavelets [Lounsbery et al. 94]: Bei dieser Methode kann der Parameterbereich aus beliebig vielen, über die Kanten verbundenen Dreiecken bestehen (nicht notwendigerweise planar), so daß sich Oberflächen mit beliebiger Topologie bearbeiten lassen. Für die gegebene Parametrisierung, auch „*initial mesh*“ genannt, wird eine Darstellung der darüber aufgespannten Oberfläche durch eine Linearkombination von Waveletbasisfunktionen bestimmt. Die Basisfunktionen werden so gewählt, daß sie annähernd orthogonal zueinander sind und nur konstanten Träger haben. Dadurch läßt sich die Darstellung effizient (in linearer Zeit) berechnen und eine Auswahl von Koeffizienten finden, die gleich null gesetzt werden können, um eine fast[4] optimale Näherung im Sinne kleinster Quadrate zu finden. Als Dreiecksnetz entsteht dabei eine rekursive Unterteilung analog zu den balancierten Quadtreemeshes

[4] Optimalität erfordert exakte Orthogonalität, was allerdings nicht ohne weiteres bei beschränktem Träger möglich ist.

(die Waveletbasisfunktionen werden bereits durch rekursive Unterteilung gebildet). Durch eine dynamische Auswahl verschieden vieler Waveletkoeffizienten und stetige Überblendung auf null an den Grenzen können absolut „flüssige“ Multi Resolution Modelle gebildet werden (d.h. die Oberfläche verändert sich stetig mit der Entfernung).

Nicht-hierarchische Methoden

Anders als bei den hierarchischen Methoden werden hier beliebige Triangulierungen des Parameterbereiches verwendet, die schrittweise durch das Hinzufügen von Punkten und lokale Retriangulierung verfeinert werden. Dies erlaubt oft eine geringere Dreiecksanzahl bei gleicher Qualität als die hierarchischen Methoden.

Inkrementelle Delaunaytriangulierung [Klein und Straßer 97]: Die Methode benutzt eine zweidimensionale Delaunaytriangulierung im Parameterbereich. Für eine feste Auflösung werden markante Punkte ausgewählt und eine Delaunaytriangulierung berechnet, die eindeutig ist, falls keine 4 Punkte auf einem Kreis liegen. Zur Erzeugung feingranularer Detaillierungen genügt es daher, eine Folge von Punkten zu speichern, die mit entsprechenden Fehlerwerten der inzidenten Dreiecke gekennzeichnet und danach sortiert sind. Die weiteren Informationen über die Dreiecksnetze sind implizit durch die (inkrementelle) Konstruktion der Delaunaytriangulierung gegeben. Dem dadurch gesparten Speicherplatz steht ein höherer Aufwand bei der Rekonstruktion einer Auflösung entgegen.

Weitere Methoden

[Heckbert und Garland 97] führen eine große Anzahl von weiteren parametrischen Methoden, insbesondere auch für den wichtigen Spezialfall von Höhenfeldern, auf. Darunter finden sich auch Dezimierungs- und Verfeinerungsalgorithmen ähnlich denen, die im nächsten Abschnitt beschrieben sind.

2.2.2.3 Triangulierte Mannigfaltigkeiten

Besonders in den letzten Jahren (etwa seit Anfang der 90er Jahre) wurden Verfahren entwikkelt, die auch ohne Kenntnis einer Parametrisierung direkt auf Dreiecksnetzen arbeiten können. Diese haben den Vorteil, daß sie auch für Modelle anwendbar sind, deren High-Level Modellierung unbekannt, unzugänglich oder nicht parametrisch ist. Der Nachteil ist, daß sie immer ein maximal detailliertes Modell aus Dreiecken als Eingabe benötigen, was die erreichbare maximale Detaillierung bei gegebenem Speicherplatz beschränkt. (Man beachte, daß eine Kugel sich durch drei Zahlen genau beschreiben läßt, während ein schönes Dreiecksnetz vielleicht 1000 Dreiecke benötigt.)

An die Dreiecksnetze können dabei verschiedene Anforderungen gestellt werden. Die Algorithmen in diesem Abschnitt verlangen Oberflächen als Eingabe, die Triangulierungen zweidimensionaler Mannigfaltigkeiten sind und produzieren eben solche als Ausgabe, ohne die Topologie der Oberflächen zu verändern; Algorithmen, die diese Anforderung nicht stellen, finden sich im nächsten Abschnitt.

Dreiecksnetze, die als gültige Triangulierungen zweidimensionaler Mannigfaltigkeiten betrachtet werden, erfüllen die Bedingung, daß zu jeder Kante auf Rändern genau ein und im Inneren genau zwei Dreiecke inzident sind und daß um jeden Punkt eine zyklische Schlaufe (bzw. auf den Rändern ein an genau einer Stelle unterbrochener Fächer) von benachbarten Dreiecken zu

finden ist ([Garland 99], S. 8f). Die Dreiecke dürfen sich nur an gemeinsamen Kanten schneiden, dabei dürfen keine Eckpunkte auf durchgehenden Kanten liegen.

Die meisten Verfahren benutzen ein solches Kriterium, um festzustellen, ob eine lokale Veränderung die Topologie erhält. Man beachte, daß der Begriff einer „gültigen triangulierten Mannigfaltigkeit“ schärfer als der mathematische Begriff einer zweidimensionalen Mannigfaltigkeit[5], angewandt auf eine Menge von Dreiecken ist, da hier zusätzliche Bedingungen an die Beziehungen der Dreiecke untereinander gestellt werden.

Die Verfahren lassen sich nach [Heckbert und Garland 97] in Dezimierungs- („*decimation*“) und Verfeinerungsmethoden („*refinement*“) einteilen.

Dezimierungsmethoden

Dezimierungsmethoden gehen vom maximal detaillierten Dreiecksnetz aus und löschen dann schrittweise Teile daraus, die unter ein gewisses Fehlertoleranzkriterium fallen und bilden lokal eine neue Triangulierung.

[Heckbert und Garland 97] klassifizieren die Methoden als:

- „*vertex decimation*“: Eckpunkte werden successive gelöscht und die entstandenen Löcher neu trianguliert.
- „*edge decimation*“: Kanten werden zusammengezogen und damit zwei Dreiecke gelöscht.
- „*triangle decimation*“: Dreiecke werden gelöscht und die Nachbarschaft neu trianguliert.
- „*patch decimation*“: mehrere benachbarte Dreiecke werden gleichzeitig gelöscht und das entstandene Loch in der Oberfläche wird neu trianguliert.

Decimation of Triangle Meshes [Schröder et al. 92]: Die Autoren beschreiben eine „*vertex decimation*“ Methode. Als Fehlermaß wird der Abstand des Eckpunktes von einer Näherungsebene durch die benachbarten Eckpunkte benutzt. Bei Eckpunkten auf Rändern oder auf scharfen Kanten im Inneren wird zusätzlich der Abstand zur neuen Rand- oder Kantenlinie berücksichtigt. Es werden nacheinander jeweils Eckpunkte mit minimalem Fehler aufgegriffen und gelöscht. Dadurch können sich Fehler bei der Behandlung direkt benachbarter Punkte aufsummieren; es kann also keine absolute Fehlerschranke garantiert werden. Vorteilhaft ist, daß der Algorithmus wegen des einfachen Fehlermaßes und der „*greedy*“-Strategie relativ schnell arbeitet.

Mesh Optimization [Hoppe et al. 93]: Die Methode versucht eine *Energiefunktion* für das Dreiecksnetz durch eine heuristische Suche zu minimieren. Die Energie steigt mit dem Quadrat des Abstandes des vereinfachten Netzes vom Ausgangsnetz und linear mit der Anzahl von Eckpunkten. Zusätzlich steigt die Energie mit der Summe der quadratischen Länge aller Kanten, um „Ausreißer“ zu verhindern. Dieser Anteil wird im Laufe der Suche reduziert.

---

[5] Eine zweidimensionale Mannigfaltigkeit (mit Rand) im mathematischen Sinne ist eine Menge von Punkten, für die in jedem Punkt eine infinitesimale Umgebung des Punktes topologisch äquivalent zu einer Kreisscheibe (oder zu einer Halbkreisscheibe) ist.

Das Dreiecksnetz wird sowohl durch Kontraktion von Kanten („*edge decimation*“) wie auch durch Aufteilen von Eckpunkten in zwei neue Eckpunkte mit einer neuen Kante und zwei neuen Dreiecken (also der inversen Operation zum „*edge decimation*“) verändert. Zusätzlich können „*edge swaps*“ vorgenommen werden, d.h. eine Kante, die zwei Dreiecke verbindet wird herausgenommen und zwischen die beiden anderen freien Eckpunkte gelegt. Jede dieser Operationen wird an zufälligen Stellen vorgenommen, und zwar nur, falls sie die Energie nicht erhöht und die Topologie nicht verändert.

Nach diesen Änderungen werden in zwei Schritten, einem Projektionsschritt und einer kleinste-Quadrate Näherung, die Positionen der Eckpunkte neu bestimmt, so daß die Summe der quadratischen Abstände zur Originaloberfläche minimiert wird. Die Optimierung des Netzes und die der Punktpositionen wird solange iteriert, bis ein Energieminimum gefunden wurde.

Nach [Heckbert und Garland 97] liefert diese Methode sehr gute Ergebnisse, allerdings benötigt sie dazu auch sehr lange Rechenzeiten.

Speicherung als Multi Resolution Modelle

Um aus einer Dezimierungsmethode Multi Resolution Modelle zu erhalten, kann die „Geschichte“ dieser Operationen aufgezeichnet und in beide Richtungen (löschen / hinzufügen) durchgeführt werden, um verschieden detaillierte Modelle zu erhalten. [Puppo und Scopigno 97] bezeichnen die entsprechenden Multi Resolution Modelle daher als „*historical models*“.

Progressive Meshes [Hoppe 96], [Hoppe 97]: Es wird zunächst eine Vereinfachung des Ausgangsmodells bis zu einer sehr groben Version vorgenommen, wobei ein ähnlicher Algorithmus wie in [Hoppe et al. 93] eingesetzt wird: Der neue Algorithmus verwendet allerdings Kantenkontraktionen als einzige Operation. Diese werden mit Hilfe einer Prioritätenwarteschlange in Reihenfolge fallender Energiegewinne durchgeführt. Die dabei insgesamt vorgenommenen Kontraktionen werden dadurch in nach „Wichtigkeit“ sortierter Reihenfolge gespeichert. Die Folge wird zur Rekonstruktion rückwärts gelesen: Ausgehend vom gröbsten Dreiecksnetz werden successive Eckpunkte in zwei durch eine neue Kante verbundene Teile aufgespalten. Sobald die gewünschte Genauigkeit erreicht ist, kann der Prozeß vorzeitig abgebrochen werden. Neben der reinen geometrischen Vereinfachung wird auch die Berücksichtigung scharfer Kanten sowie verschiedener Oberflächenattribute, getrennt nach diskreten (z.B. Materialindizes) und kontinuierlichen Attributen (z.B. Farben, Normalen, Texturkoordinaten), unterstützt.

In einer Erweiterung [Hoppe 97] unterstützen die „*progressive meshes*“ auch die blickpunktabhängige Extraktion von vereinfachten Modellen. Hierbei wird die Verfeinerung abhängig vom Fehler im projizierten Bild gesteuert. Zusätzlich werden vom Betrachter abgewandte und außerhalb der Sichtpyramide liegende Teile des Objektes so grob wie möglich dargestellt. Dazu wird ein Algorithmus vorgestellt, der die Abhängigkeiten der „*vertex splits*“ untereinander feststellt und die für die Verfeinerung notwendigen „*vertex splits*“ vornimmt. In Experimenten erlaubte dieser Algorithmus eine Extraktion der vereinfachten Dreiecksnetze in Echtzeit, die Geschwindigkeit ließ sich weiter steigern durch eine inkrementelle Verfeinerung bzw. Vergröberung bei sich stetig bewegendem Blickpunkt.

In einer anderen Erweiterung („*progressive simplicial complexes*“, [Popovic und Hoppe 97]) werden die „*progressive meshes*“ auch auf Topologievereinfachung von Dreiecksnetzen verallgemeinert.

Verfeinerungsmethoden

„Multiresolution Analysis of Arbitrary Meshes“ [Eck et al. 95]: Die im vorigen Abschnitt beschriebene Waveletmethode von [Lounsbery et al. 94] kann nur angewandt werden, wenn eine Parametrisierung in Form eines *„initial mesh“* bekannt ist. In [Eck et al. 95] wird eine Methode gezeigt, um für beliebige triangulierte Mannigfaltigkeiten eine solche Parametrisierung zu bestimmen. Dazu wird ein neues Dreiecksnetz erzeugt, welches aus rekursiv unterteilten Dreiekken besteht und das ursprüngliche bis auf einen vorgegebenen Fehler annähert. Auf dieses kann dann die Methode aus [Lounsbery et al. 94] angewandt werden. Ein Nachteil bei diesem Vorgehen ist, daß auch die feinste Detaillierung nur eine Näherung ist, die zudem noch für typische Beispiele etwa doppelt so viele Dreiecke wie das Ausgangsnetz enthält. Zudem ist die Berechnung der Parametrisierung relativ aufwendig.

Andere Methoden

Es finden sich auch einige anders arbeitende Verfahren, z.B.:

[Turk 92]: Es werden zufällige Punkte auf der Oberfläche verteilt, die dann durch „Abstoßungskräfte“ möglichst gleichmäßig angeordnet werden. Danach wird mit den alten Punkten zusammen eine Triangulierung bestimmt (um eine korrekte Topologie zu gewährleisten). Aus dieser werden anschließend die alten Punkte entfernt. Die Methode eignet sich nicht besonders gut für Objekte mit scharfen Kanten.

[Varshney 94]: Aus der gegebenen Oberfläche werden zwei neue gebildet, die nach innen und außen um ein vorgegebenes Fehlermaß verschoben sind. Dann wird versucht, eine Oberfläche aus möglichst wenigen Dreiecken zu konstruieren, die den durch die verschobenen Flächen definierten Bereich nicht verläßt. Hierfür wird mit einer heuristischen Strategie ein Optimum gesucht. Für eine exakte Optimierung wurde gezeigt, daß dieses Problem NP-hart ist. Man erhält eine Approximation mit garantiertem Fehler, die allerdings auch bei der heuristischen Suche noch mit einem relativ hohen Aufwand verbunden ist.

2.2.2.4 Vereinfachung ohne Erhaltung der Topologie

Im allgemeineren Fall sind beliebige Dreiecksnetze als Eingabe und damit auch als Ausgabe erlaubt.

[Rossignac und Borrel 93]: Bei dieser Methode werden alle Eckpunkte durch Quantisierung der Koordinaten in ein reguläres, dreidimensionales Gitter eingeteilt. Alle Eckpunkte, die in die selbe „Box“ in diesem Gitter fallen, werden zu einem zusammengefaßt. Dreiecke, die dadurch zu Punkten oder Linien degenerieren, werden in einer Tabelle abgespeichert und alle darin doppelten werden gelöscht, die übrigen werden als Linien bzw. als Punkte im vereinfachten Modell erhalten.

Die Methode ist einfach zu implementieren und sehr effizient. Die Approximationsergebnisse sind allerdings auch nicht besonders gut.

Voxel Based Object Simplification [He et al. 95]: Diese Methode verwendet ebenfalls ein reguläres Gitter von dreidimensionalen Boxen. Die Geometrie wird mittels eines dreidimensionalen Rasterungsalgorithmus in die Boxen „eingezeichnet“, so daß ein Voxel-Volumenmodell der Geometrie entsteht. Um Aliasing durch die reguläre Abtastung der Geometrie zu verhindern, wird beim Rastern ein auf die Rasterfrequenz abgestimmter Low-Pass-Filter verwendet. Aus dem Volumenmodell wird mit Hilfe eines Isoflächenrekonstruktionsalgorithmus (Marching Cubes, [Lorensen und Cline 87]) ein geometrisches Modell rekonstruiert. Dieses Dreiecksnetz hat in

der Regel wesentlich mehr Dreiecke als nötig. Daher wird ein weiteres, topologieerhaltendes Verfahren (z.B. [Schröder et al. 92]) nachgeschaltet, um redundante und fast redundante Dreiecke zu entfernen. Die Größe der Boxen gibt damit implizit den Grad an topologischer Vereinfachung an.

Quadric-Based Polygonal Surface Simplication [Garland 99]: Hierbei handelt es sich um eine Dezimierungsmethode, die auf einer Verallgemeinerung der Kantenkontraktion („*edge decimation*“) beruht: Es wird zugelassen, auch Eckpunkte, die nicht durch eine Kante im Ausgangsnetz verbunden sind, zusammenzuziehen, falls diese einen vorgegebenen Maximalabstand nicht überschreiten. Eine Besonderheit dieses Verfahrens ist das verwendete Fehlermaß („*quadric error metric*“). Es ermöglicht eine gleichzeitig sehr effiziente und genaue Bestimmung des durch eine Kantenkontraktion verursachten Fehlers. Dadurch liefert das Verfahren (auch im Vergleich zu auf triangulierte Mannigfaltigkeiten beschränkten Verfahren) sehr gute Approximationsergebnisse in vergleichsweise kurzer Zeit, so daß auch große Modelle bearbeitet werden können. Das Fehlermaß kann auf die Unterstützung von Oberflächenattributen wie Texturkoordinaten, Normalen oder Farben erweitert werden.

Progressive Simplicial Complexes [Popovic und Hoppe 97]: Diese Methode ähnelt der zuvor genannten. Neben der abweichenden Fehlermetrik (siehe Beschreibung der „*progressive meshes*“) unterscheidet sich die Methode noch darin, daß zusätzliche Kanten zwischen unzusammenhängenden Objekten nicht durch ein Distanzmaß in die Berechnung eingefügt werden, sondern durch eine dreidimensionale Delaunaytriangulierung.

2.2.3 Möglichkeiten und Grenzen

Der Blick in die Literatur über „*mesh simplification*“ zeigt, daß die automatische Vereinfachung von Dreiecksnetzen weit fortgeschritten ist. Verfahren wie z.B. „*progressive meshes*“ lassen kaum noch Wünsche offen und lassen durch den hohen Standard der z-Buffer Hardware auch eine gute Performance bei der Darstellung erwarten. Die dazu notwendigen Vorberechnungszeiten können durch geschickte Fehlermetriken auch für Objekte aus einigen hunderttausend Dreiecken in den Minutenbereich gesenkt werden ([Garland 99], S. 98).

Problematisch bleiben Szenen, die aus vielen unabhängigen Objekten bestehen: Für solche Fälle erscheinen Verfahren, die topologische Vereinfachungen durchführen können, interessant. Ein topologievereinfachender Algorithmus könnte eine ganze Gruppe eng benachbarter Objekte zu einem Objekt verschmelzen. Bei derart starker Vereinfachung ist allerdings unklar, wie die Oberflächenattribute eines solchen Ersatzobjektes gewählt werden können, um die komplexe Struktur der ursprünglichen Objekte, insbesondere die Orientierung der Oberflächen und die lokalen Verdeckungen, gut wiederzugeben.

Der randomisierte z-Buffer Algorithmus wurde entwickelt, um gerade in Fällen, in denen durch eine komplizierte Topologie der Eingabe praktisch keine Anwendung von „*mesh simplification*“ mehr möglich ist, trotzdem gute Approximationen liefern zu können.

2.3 Ersetzungsverfahren und „Image-Based Rendering“

Ein Ersetzungsverfahren ersetzt Gruppen von Szenenteilen durch eine einfachere Darstellung. Die häufigste Form der Ersatzdarstellung sind Pixelbilder („*image-based rendering*“), die eine bereits gerenderte Ansicht eines Szenenteils darstellen. Mit einem vorberechneten Bild ist die

exakte Approximation eines Szenenteiles in optimaler Zeit möglich, diese gilt aber bei allgemeinen Szenen allerdings nur für genau einen Blickpunkt. Sobald sich der Abstand oder der Winkel zum Objekt ändert, treten Fehler auf. Unter der Annahme, daß die zu vereinfachenden Objekte relativ weit entfernt sind, treten hauptsächlich Parallaxefehler durch abweichende Betrachtungswinkel auf. Ein Beispiel aus [Chamberlain et al. 95] zeigt, daß die lokale Verdeckung in einer Objektgruppe sich u.U. sehr drastisch mit dem Blickwinkel ändern kann, was zu einer völligen Veränderung des Bildes führen kann (siehe Abbildung 8): Eine große Menge paralleler, dicht übereinander angeordneter Flächen befindet sich vor einem weiteren Objekt. Dieses wird nur unter ganz bestimmten Blickwinkeln sichtbar. Der Winkelbereich, für den dies zutrifft, kann dabei beliebig klein werden.

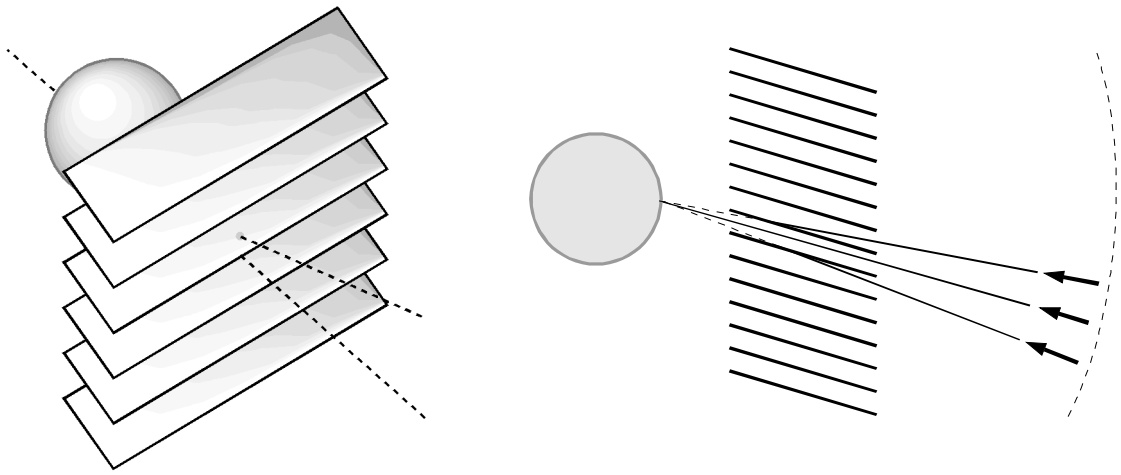


Abbildung 8: Das „Jalousie“-Beispiel (nach [Chamberlain et al. 95])

Diese Szene stellt nicht nur die bildbasierten Verfahren vor Probleme: Auch ein „*mesh simplification*“ Algorithmus kann die geometrische Komplexität einer solchen Szene nicht reduzieren. Die Szene kann als eine Art „*benchmark*“ für die Topologieunabhänigkeit eines approximativen Renderingverfahrens angesehen werden. Das randomisierte z-Buffer Verfahren kann solche Szenen korrekt darstellen, da dabei die projizierte Gesamtfläche der Ansicht linear in die Laufzeit eingeht, hängt die dabei erzielte Laufzeit allerdings vom Blickwinkel und vom Flächeninhalt der Lamellen ab.

Ein weiteres Problem für bildbasierte Verfahren sind Szenen mit einer blickpunktabhängigen Schattierungsfunktion. Dadurch hängt die Farbe der Oberflächen u.U. stark von Position („Nebel“) und Blickwinkel („Glanzlichter“) des Betrachters ab und kann durch ein festes Bild nicht über einen größeren Bereich angenähert werden.

2.3.1 Statische Methoden

Bei statischen Methoden werden alle verwendeten Ersatzdarstellungen im voraus berechnet. Zur Laufzeit werden dynamisch einige davon ausgewählt, um komplexe Geometrien dadurch zu ersetzen. Die Reaktion auf dynamische Veränderungen der Szene erfordert daher u.U. einen hohen Aufwand.

2.3.1.1 Textured Walls

[Ebbesmeyer 98] beschreibt einen Ansatz namens „*Textured Virtual Walls*“, der speziell für Innenraumszenen mit geringer Verdeckung gedacht ist. Dabei werden weiter entfernte Objekte nicht gerendert und statt dessen auf texturierte Wände projiziert, die den Betrachter umgeben. Um Parallaxefehler zu kontrollieren, wird die Szene in viele rechteckige, in einem regelmäßigen Gitter angeordnete Zellen unterteilt, für die jeweils andere texturierte Wände gelten. Beim Übergang von einer zur anderen Zelle werden die beteiligten Texturen ineinander übergeblendet.

Unter der Annahme, daß der Nahbereich der Szene nicht komplex ist erreicht die Methode dank hardwarebeschleunigter Texturierung gute Laufzeiten. Problematisch ist, daß der Speicherplatzbedarf mit der Ausdehnung der Szene stark wächst: Eine quadratische, in der Ebene angelegte Szene mit Durchmesser r benötigt $\Theta(r^2)$ Zellen. Jede Zelle benötigt mit Texturen mittlerer Auflösung, wie etwa $4 \times 256 \times 256$ 24-Bit-Pixel, schon 768 KB Speicher. Ohne zusätzliche Maßnahmen wie Kompressionstechniken oder wie der Verwendung von Hintergrundspeicher sind daher größere Szenen nicht möglich.

[Sillion et al. 97]: Hier wird ein ähnlicher Ansatz benutzt, der speziell auf die Darstellung städtischer Szenen optimiert ist. Die Einteilung in Zellen erfolgt dabei nach dem Straßennetz. Statt einfacher „Wände“ benutzen die Autoren komplexere Geometrien („*impostors*“), auf die die Texturen projiziert werden. Die Geometrie wird aus Kanten einer projizierten Ansicht sowie aus den zugehörigen Tiefeninformationen des z-Buffers bestimmt. Dadurch entsteht eine höhenfeldartige Geometrie, die in einem wesentlich größeren Winkelbereich mit geringen Parallaxefehlern betrachtet werden kann. Die „*impostors*“ können wahlweise statisch oder dynamisch beim Betreten einer neuen Region berechnet werden.

2.3.1.2 Volumenbasierte Ersetzung

Diese Methoden stellen sozusagen das Gegenstück zu den Texturwänden dar: Hier werden Ersatzdarstellungen für Objektgruppen statt für alle Objekte außerhalb eines räumlichen Gebietes gebildet. Man kann die Methoden auch als volumenbasierte Verfahren auffassen: Für hierarchische Gruppen von Teilvolumina der Szene werden Repräsentationen vorberechnet, aus denen sich für bestimmte (üblicherweise weiter entfernte) Perspektiven schnell näherungsweise Ansichten bestimmen lassen.

[Maciel und Shirley 95] gruppieren die Szene mit Hilfe eines Octrees hierarchisch in „*cluster*“ von benachbarten Objekten. Für jeden dieser Cluster werden verschiedene Ersatzdarstellungen in Betracht gezogen, darunter „*billboards*“[6], texturierte Boxen, farbige Boxen und normale „*level-of-detail*“-Modelle (nicht bei großen Gruppen von Objekten). Aus diesen wird nach einer Kosten/Nutzen-Heuristik ausgewählt. Zur Darstellung der Szene wird die Hierarchie mit einem

[6] ein Bild, daß mit der Entfernung zum Betrachter skaliert aber nicht mit dem Betrachtungswinkel gedreht wird; für symmetrische Objekte geeignet.

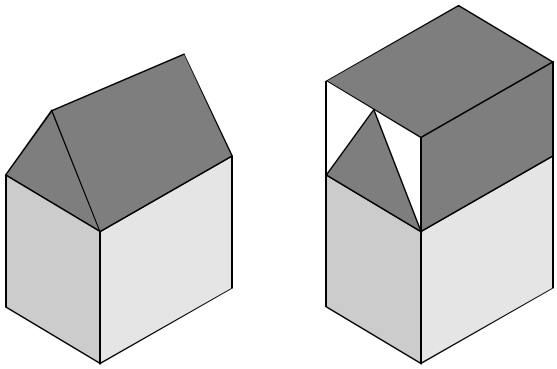


Abbildung 9: Ersetzung durch texturierte Boxen

Greedyalgorithmus traversiert, und geeignete Knoten werden wiederum nach einer Kosten/Nutzen-Heuristik zur Anzeige ausgewählt. Dabei wird ein solcher Schnitt durch den Baum vorgenommen, daß jedes Blatt durch die Wurzel eines Teilbaumes repräsentiert wird.

Speziell die allgemein einsetzbare Vereinfachung durch texturierte Boundingboxen führt in vielen Fällen zu ungünstigen Ergebnissen (siehe Abbildung 9). Durch die Verwendung komplexerer Basisgeometrien[7] anstelle einfacher, achsenparalleler Boxen könnten diese Probleme gemindert, aber sicher nicht völlig beseitigt werden.

[Chamberlain et al. 95] beschreiben einen vollständig volumenbasierten Ansatz. Für die gegebene Szene wird ein Octree aufgebaut und für die Seiten der so entstandenen Zellen wird jeweils eine Farbe und ein Transparentwert berechnet, indem die darin enthaltenen Objekte gerendert und anschließend ein entsprechender Farbmittelwert bestimmt wird. Zur Darstellung der Szene wird der so entstandene Baum soweit in die Tiefe durchlaufen, bis die projizierte Fläche der Boxen eine vorgegebene Größe unterschreitet oder Blätter sind. Im ersten Fall wird die Box mit zuvor berechneten Farben auf den Seitenflächen eingezeichnet, im zweiten Fall die in der Box enthaltene Geometrie. Die Autoren begründen, daß bei entsprechenden Annahmen über eine gleichmäßige Verteilung der Objekte in der Szene, eine Approximation der Szene in logarithmischer Zeit (genauer: in logarithmischer geometrischer Komplexität gemessen an der ursprünglichen geometrischen Komplexität) erreicht werden kann. Da nur Farbwerte für die Boxen gespeichert werden, ist der Speicherplatzbedarf bei dieser Methode vergleichsweise moderat.

Versuche mit Außenszenen aus vielen unabhängigen, sich wenig verdeckenden Objekten (insbesondere einer Waldszene) lieferten einen guten optischen Eindruck; mit *„mesh simplification“* oder *„visibility culling“* wären in diesen Szenen keine großen Gewinne zu erwarten gewesen. Das Verfahren bewirkte sogar eine gewisse Form von Anti-Aliasing.

Bei großflächigen Dreiecksnetzen wurden Artefakte sichtbar: Durch die Einschränkung der Richtungen bei der Berechnung der Farbwerte traten vor allem Probleme mit falscher Transparenz auf geschlossenen Oberflächen auf: Die farbigen Boxen sind nicht in der Lage, die darin enthaltene Geometrie aus beliebigen Perspektiven richtig wiederzugeben. Die Autoren weisen darauf hin, daß dies ein Kernproblem bei einem derartigen Ansatz ist. Die Bestimmung effizient auswertbarer und genauer Ersatzrepräsentation für allgemeine Geometrien innerhalb eines zu ersetzenden Volumens ist oft nur schwer möglich. Als Beispiel für eine schwer zu ersetzende Geometrie führen sie das in Abbildung 8 zitierte „Jalousiebeispiel“ an. Dieses Problem tritt sicherlich bei vielen volumenbasierten Ersetzungsmethoden auf.

---

[7] Ähnlich wie in [Sillion et al. 97]; es wäre auch denkbar, die Basisgeometrien durch topologievereinfachendes *„mesh simplification“* zu bestimmen.

Die Strategie, mit der [Chamberlain et al. 95] die Boxen aus einem Octree auswählen, ähnelt der, mit der beim randomisierten z-Buffer Boxen mit bezüglich der Tiefe schwach variierendem Projektionsfaktor bestimmt werden (siehe Abschnitt 3.3). Hier werden allerdings keine festen Farbwerte zur Approximation der Boxen verwendet, sondern zufällig daraus ausgewählte Punkte. Diese liefern in allgemeineren Fällen eine gute Ersatzrepräsentation für den Inhalt der Box, gleichzeitig steigt aber auch der Aufwand, um diese zu berechnen, der Aufwand pro Box ist nicht mehr konstant.

2.3.2 Dynamische Methoden

Statische bildbasierte Vereinfachungsmethoden leiden oft unter dem sehr großen Speicherplatzbedarf der vielen vorberechneten Texturen. Damit einher gehen lange Vorberechnungszeiten und Probleme bei dynamischen Szenen. Daher liegt es nahe zu versuchen, die benötigten Ersatzdarstellungen dynamisch „bei Bedarf“ zu erzeugen.

Das Potential dieses Ansatzes läßt sich an einem einfachen Modell untersuchen: Angenommen, die Szene besteht aus gleichmäßig in einem ebenen Kreis mit Radius r verteilten Objekten (Typ „Stadtszene“). Der Beobachter bewegt sich auf einer Geraden durch die Szene. Ein Bild eines Szenenteils ist jeweils innerhalb eines konstanten Winkelbereichs genau genug. Die Winkelabweichung verkleinert sich bei einer Bewegung des Betrachters antiproportional mit dem Abstand zum Objekt (zumindest für einen konstanten Anteil der Objekte in kegelförmigen Bereichen senkrecht zur Bewegungsrichtung, siehe Abbildung 10). Daher müssen spätestens bei einem konstanten Bruchteil des Abstandes d alle Objekte in diesem Abstand mindestens einmal neu gezeichnet werden. Bei gleichmäßig verteilten Objekten finden sich stets $\Theta(d)$ Objekte im Abstand d. Diese verursachen bei der Bewegung um $\Theta(r)$ Einheiten also einen Aufwand von $\Theta(r/d)$. Man erhält für den Gesamtaufwand bei der Bewegung um $\Theta(r)$ Einheiten also einen Aufwand von

$$\Theta\left(\int_{\Theta(1)}^{r} x\frac{r}{x}dx\right) = \Theta\left(r\int_{\Theta(1)}^{r} 1dx\right) = \Theta\left(r^2\right)$$

für die lineare Bewegung um eine Distanz von $\Theta(r)$ Einheiten. Dies bedeutet, daß der durchschnittliche Aufwand für Neuberechnungen bei einer Distanz von $\Theta(1)$ Einheiten mindestens mit r wächst, oder anders ausgedrückt: Bei beschränkter Rechenleistung fällt die zulässige Maximalgeschwindigkeit des Beobachters bei gleichmäßiger, linearer Bewegung mindestens mit $O(1/r)$.

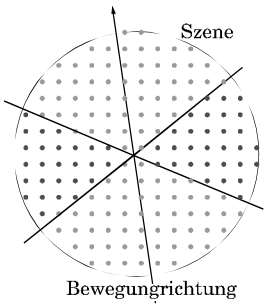


Abbildung 10: Bereiche, in denen sich der Parallaxefehler proportional zum Abstand verhält

Das Walkthroughproblem für große Szenen läßt sich also nicht mit dynamischer bildbasierter Vereinfachung allein lösen. Andererseits würde ein vollständiges Neuberechnen $\Theta(r^2)$ Zeit pro Schritt benötigen gegenüber $\Theta(r)$ bei der amortisierten Variante. In einem solchen Modellfall würde sich der Aufwand also von $\Theta(n)$ auf $\Theta(\sqrt{n})$ reduzieren, wobei der Aufwand für die Komposition der gespeicherten Bildteile allerdings noch nicht berücksichtigt ist.

Die folgenden Methoden versuchen nun, diese Form der Amortisierung möglichst gut zu nutzen.

2.3.3 Eine elementare Lösung

Eine von [Lukovszki 98] vorgeschlagene Methode löst diese Amortisierungsaufgabe direkt: Sei r der Szenendurchmesser einer zweidimensionalen Szene. Um den Betrachter werden $\log_2 r$ Zylinder im Abstand von 2^n Einheiten, $n = 1...\log r$ plaziert, die jeweils alle Geometrie im Radius zwischen 2^n und 2^{n-1} darstellen. Sobald sich der Betrachter um mehr als $\Theta(2^j)$ Einheiten weiterbewegt, müssen alle Zylinder mit Abstand $\leq 2^j$ neu berechnet werden (der Aufwand sollte über den gesamten Bewegungszeitraum verteilt werden). Der mittlere Aufwand beträgt:

$$\sum_{i=1}^{\log r} O\left(\underbrace{\left(2^i\right)^2}_{\substack{\#\text{ Objekte in} \\ \text{Zylinder i}}} \cdot \underbrace{2^{-i}}_{\substack{\text{Update-} \\ \text{häufigkeit}}} \right) = \sum_{i=1}^{\log r} O\left(2^i\right) = O(r)$$

In jedem Schritt müssen $\log r$ gespeicherte Bilder zusammengefügt werden, was durch Texturhardware unterstützt werden könnte. Die Methode wurde noch nicht praktisch untersucht, zeigt aber gut das prinzipielle Vorgehen dynamischer Methoden. Die im folgenden beschriebene Methode ist in vielerlei Hinsicht optimiert, insbesondere wird auch die Verteilung der Objekte im Raum ausgenutzt, um weitere Einsparungen zu erreichen (Zelleneinteilung mit BSP-Baum):

2.3.4 Hierarchical Image Caching

[Shade et al. 96] beschreiben einen Algorithmus, der für Teilbäume eines räumlichen Unterteilungsbaumes dynamisch Ersatzbilder vorberechnet. Die Szene wird mit einem BSP-Baum hierarchisch in Regionen eingeteilt. Für jede Region kann ein Bild gespeichert werden, welches alle darin enthaltenen Objekte darstellt. Für jede Region wird dann mit Hilfe eines entsprechenden Fehlermaßes entschieden, ob ein evtl. vorhandenes Bild noch gültig ist. Wenn ja, so wird dieses beim Rendering verwendet. Wenn nicht, wird geprüft, ob das Abspeichern eines Bildes sich

amortisieren würde, und nur dann wird ein solches erstellt. Andernfalls ist es günstiger, die Geometrie direkt einzuzeichnen. Der Vorgang wird auf dem Baum rekursiv durchgeführt, so daß ein Caching von Bildern auf verschiedenen Ebenen im Baum möglich ist. Um die Gültigkeitsdauer zu erhöhen, wird ein Bild aus dem Cache per Texturemapping auf ein Viereck plaziert und entgegen der seit der Erstellung getätigten Kamerabewegung gedreht.

In einer Testszene ließen sich Beschleunigungen in der Größenordnung von Faktor 10 erreichen. Die Vorbereitungszeit für das erste Bild lag bei zwei Minuten gegenüber einigen Sekunden für alle weiteren.

2.3.5 Talisman

[Torborg und Kajiya 96] beschreiben eine Hardwarearchitektur zur effizienten Unterstützung von *„image-based rendering“* Verfahren. Zunächst wird eine Szene mit dem gewöhnlichen z-Buffer Verfahren gerendert, was von der Hardware unterstützt wird. Die Ergebnisse des Renderings können dann auf mehreren teilweise transparenten Bildebenen abgelegt werden. Diese Ebenen werden später durch einen speziellen Prozessor zur Verarbeitung zweidimensionaler Bilder in Echtzeit wieder zusammenmontiert. Dabei kann jede Ebene einer affinen Transformation unterzogen werden, um den Einfluß eines leicht geänderten Blickpunktes zu simulieren. Das Montieren der Ebenen ist um eine Größenordnung schneller möglich als die Neuberechnung auf Basis der Geometrie, so daß dieses mit voller Bildwiederholfrequenz erfolgen kann. Um keine riesigen Texturspeichermengen für die vielen Bildebenen zu benötigen, wird das Bild kachelweise bearbeitet und der Inhalt von zur Zeit nicht bearbeiteten Kacheln wird komprimiert gespeichert.

Ziel der Architektur war es, auf üblichen Heimcomputern komplexe dreidimensionale Szenen darstellen zu können, ohne die Speicherbandbreite und Menge einer Hochleistungsworkstation aufbringen zu müssen. Bis heute ist die beschriebene Hardware allerdings noch nicht auf dem Massenmarkt erhältlich. Die zum damaligen Zeitpunkt angestrebte amortisierte Leistung ist allerdings heutzutage bereits in Form konventioneller z-Buffer Hardware auf dem Massenmarkt erhältlich ([Bertuch 99]). Es bleibt daher unklar, ob die Architektur auch unter Berücksichtigung der Fortschritte der Chiptechnologie in den letzten Jahren ähnliche Vorteile erbringen könnte.

2.4 Andere Approximationsverfahren

Neben den bisher genannten sind noch eine Reihe weiterer Methoden bekannt. Als weitere Klasse von Algorithmen sei hier noch speziell die „prozedurale Methode“ (*„semantic driven simplification“* in [Erikson 96]) erwähnt: Für einen speziellen Typ von Objekten oder Objektgruppen können u.U. besondere Eigenschaften der Objekte ausgenutzt werden, um diese besonders gut, besonders schnell oder besonders einfach zu approximieren. Der Programmierer schreibt einen speziell auf die Anwendung zugeschnittenen Algorithmus, der solche Objekte in verschiedenen Größen darstellt. [Weber und Penn 95] benutzen einen solchen Ansatz zur Darstellung künstlich erzeugter Bäume.

Wie bereits in Kapitel 2.1.2 erläutert, führt die hohe Leistung heutiger Renderinghardware immer mehr zu Problemen mit dem großen Speicherbedarf von Szenen. Ein Ausweg daraus könnte die prozedurale Modellierung bilden, d.h., daß die Geometrie online aus Berechnungsvorschrif-

ten erzeugt wird. In dem Zusammenhang könnte auch eine Form von prozeduraler Approximation interessant werden.

2.5 „Visibility Culling"-Verfahren

„*visibility culling*"-Verfahren dienen dazu, einen möglichst großen Teil der im Bild unsichtbaren Geometrie vom Rendering auszuschließen. In einem ersten Schritt werden dazu zunächst Szenenteile, die außerhalb der aktuellen Sichtpyramide liegen, abgeschnitten. Dies läßt sich mit relativ geringem Aufwand durch einen räumlichen Unterteilungsbaum (z.B. mit einem Octree) realisieren: Bei einer Traversierung werden nur Knoten gezeichnet, die (fast) vollständig innerhalb liegen (siehe z.B. [Chamberlain et. al 95]).

Der schwierigere Teil der Aufgabe besteht darin, effizient Verdeckungen von Objekten untereinander festzustellen. Für eine rein geometrische Lösung läßt sich sehr einfach eine untere Schranke zeigen ([Foley et al. 96], Seite 717): Aus n Objekten können durch Verdeckung $\Omega(n^2)$ unabhängige sichtbare Fragmente entstehen. Nicht zuletzt dieses Problem hat zur Entwicklung von pixelbasierten Verfahren, wie dem z-Buffer Verfahren geführt, die zumindest eine lineare Laufzeit garantieren können. Wie schnell die Berechnung der Verdeckungen für Pixelbilder bei sinnvoller Vorberechnungszeit maximal möglich ist, insbesondere, ob dies im allgemeinen Fall schneller als mit dem z-Buffer Algorithmus möglich ist, ohne sehr große Speicherplatzmengen für vorberechnete Datenstrukturen zu benötigen, ist derzeit (meines Wissens nach) nicht bekannt. Für das verwandte Raytracingproblem sind untere Schranken bekannt (siehe Abschnitt 2.6).

Beim derzeitigen Stand der Technik kann das Verdeckungsproblem für große Szenen unter Echtzeitbedingungen also als schwierig gelten. Trotzdem können heuristische Lösungen für einige Klassen von Szenen das Problem erfolgreich behandeln:

2.5.1 „Potentially Visible Sets" (PVS) - Methoden

Die Idee bei PVS-Methoden besteht darin, für jeden Betrachtungsstandpunkt schnell eine Obermenge der sichtbaren Objekte zu bestimmen. Dazu wird der Raum in Zellen eingeteilt, und für jede Zelle wird in einer Vorberechnungsphase bestimmt, welche anderen Zellen von dort sichtbar sind. Die PVS Methoden speichern dadurch im Prinzip eine vorberechnete Approximation der tatsächlichen Sichtbarkeitsverhältnisse. Die Zellen approximieren sowohl die sichtbaren Objekte wie auch die Regionen gleicher Sichtbarkeit. Dieser Ansatz kann natürlich nur in Umgebungen mit einer entsprechenden Zellenstruktur der Verdeckung einen Gewinn bringen; ein typisches Einsatzgebiet sind daher architektonische Modelle von Innenräumen.

PVS-Verfahren

[Teller und Séquin 91] beschreiben ein Verfahren für architektonische Modelle. Die Zellen werden automatisch mit Hilfe eines k-d-Baumes berechnet. Für gegebene „Portale" zwischen den Zellen wird die Sichtbarkeit zwischen den Zellen durch einen linearen Optimierungsalgorithmus bestimmt (die Existenz einer Lösung entspricht der Sichtbarkeit). Diese Berechnungen werden offline durchgeführt. Während des Walkthroughs werden die so bestimmten PVSs noch blickpunktabhängig verfeinert: Für das Rendering werden die für die aktuelle Blickposition tatsächlich sichtbaren Zellen bestimmt.

Verfahren wie dieses können in entsprechend dicht verdeckten Umgebungen große Einsparungen an Rechenzeit erbringen. Da gerade die Kombination aus großen, in Zellen gegliederten, stark verdeckten statischen Szenen oft in Computerspielen benötigt wird, sind entsprechende Verfahren dort in vielen Produkten im Einsatz [Zerbe 99]. Anstelle eines k-d-Baumes wird in der Regel ein allgemeiner BSP-Baum zur Zelleneinteilung benutzt, da dieser eine flexiblere Gestaltung der „Räume“ zuläßt (keine Beschränkung auf achsenparallele Wände).

Exakte Verfahren

Es stellt sich die Frage, ob nicht eine genauere Vorabberechnung der Sichtbarkeitsinformationen möglich wäre. In [O'Rourke 87], S. 219-223, wird gezeigt, daß in einer zweidimensionalen Umgebung von n Liniensegmenten die Region, in der ein einzelnes Segment sichtbar ist, im schlimmsten Fall $\Omega(n^4)$ Eckpunkte benötigt. Da dies ein Spezialfall des allgemeinen dreidimensionalen Problems darstellt, erscheint damit zumindest eine explizite Speicherung der exakten Sichtbarkeitsregionen als unpraktikabel. In [Durand et al. 97] wird eine Datenstruktur entworfen, um exakte Sichtbarkeitsinformationen zu speichern. Mit Hilfe dieser können effizient verschiedene Anfragen beantwortet werden, darunter auch die Bestimmung der sichtbaren Objekte. Im worstcase benötigt die Datenstruktur ebenfalls $\Theta(n^4)$ Speicherplatz; in Experimenten mit kleineren Szenen ($n \leq 1.500$) wurde nur ein quadratischer Speicherplatzbedarf festgestellt, was allerdings immer noch ein ernstes Problem für größere Szenen darstellt.

2.5.2 Hierarchischer z-Buffer

[Green et al. 93] beschreiben eine Erweiterung des bekannten z-Buffer Verfahrens, die auch in allgemeineren Szenen effizient Verdeckungen vom Rendering ausschließen kann. Dazu wird die Szene mit einem Octree in Boxen unterteilt. Der z-Buffer wird zu einer „z-Pyramide“ erweitert: Zusätzlich zu dem bisherigen Puffer werden noch weitere Puffer angelegt, deren Durchmesser in Zweierpotenzen schrumpft. In diesen gröberen Puffer wird jeweils der maximale (d.h. am weitesten entfernte) Tiefenwert aus den vier darunterliegenden Pixeln gespeichert.

Der Octree wird nun von vorne nach hinten (vom Betrachter aus gesehen) traversiert. Bevor der Inhalt einer Box aus dem Octree gezeichnet wird, wird geprüft, ob diese bereits verdeckt ist. Dazu wird die Fläche der Box in die z-Pyramide projiziert und beginnend bei der gröbsten Ebene geprüft, ob die Entfernung bereits abgedeckt ist. Nur falls alle Tests negativ sind, wird die Geometrie eingezeichnet und die z-Pyramide aktualisiert. Wenn mehrere ähnliche Bilder (z.B. bei einer Walkthroughanwendung) nacheinander erzeugt werden sollen, kann der Algorithmus weiter beschleunigt werden, indem die im letzten Durchgang sichtbaren Boxen als Ausgangsbasis für die Überprüfung genutzt werden.

Der Algorithmus lieferte in Tests starke Beschleunigungen für Szenen, die mit PVS-Methoden nicht mehr handhabbar gewesen wären. Ein Nachteil, der wohl den Einsatz in vielen Anwendungen verhindert hat, ist, daß bestehende z-Buffer Hardware den Algorithmus nicht optimal unterstützen kann, so daß man wesentliche Teile des Algorithmus auf vergleichsweise niedrigem Geschwindigkeitsniveau in Software realisieren muß.

Es gibt verwandte Methoden wie z.B. *„Hierarchical Occlusion Maps“* [Zhang et al. 97], die diesen Nachteil nicht mehr haben, dafür aber nicht mehr so allgemeine Verdeckungen erkennen können.

2.6 Raytracing als algorithmische Alternative

Bei den hinlänglich bekannten Schwächen des z-Buffer Algorithmus stellt sich die Frage, ob nicht ein völlig anderer Algorithmus das Problem direkt lösen könnte. Unter den bekannten Standardalgorithmen (siehe [Foley et al. 96]) erscheint vor allem das Raytracing interessant, da dieses Verfahren explizit für jeden Pixel die Farbe eines darunter sichtbaren Objektes bestimmt, ohne alle Objekte dazu bearbeiten zu müssen und damit zumindest vom Prinzip her unabhängig von Verdeckung und geometrischer Komplexität sein könnte.

2.6.1 Der Raytracing Algorithmus

Raytracing tritt in zwei Varianten auf: Als reiner Renderingalgorithmus („*visible surface raytracing*“, „*raycasting*“, [Appel 68]) und als globales Beleuchtungsmodell („*recursive raytracing*“, [Kay 79], [Whitted 80]). Hier ist nur die erste Variante interessant.

Bei dieser wird durch jeden Pixel des Bildes ein Strahl in die Szene geschickt und der erste Schnittpunkt mit einem Objekt bestimmt. Für diesen wird eine Schattierung berechnet und in das Bild eingetragen.

Eine naive Implementierung dieser Idee würde zu einer Laufzeit von $O(A \cdot n)$ mit n als geometrischer Komplexität und A als Anzahl von Pixeln im Bild führen. Da die Schnittpunktberechnung zwischen einem Strahl und einem Dreieck aufwendige Fließkommaarithmetik erfordert, liegen die Konstanten dabei zudem relativ hoch. Insgesamt ist daher ein solcher Algorithmus bei Echtzeitaufgaben keine Alternative zum z-Buffer.

2.6.2 Verbesserungen der Laufzeit

Alle Verbesserungen des Raytracingalgorithmus zielen daher darauf ab, das Finden des ersten Schnittpunktes durch vorberechnete Suchdatenstrukturen möglichst stark zu beschleunigen. Falls eine solche Suche in stark sublinearer Zeit bei akzeptabler Vorberechnungszeit und vernünftigen Konstanten gelingen würde, könnte man das statische Walkthroughproblem wohl als gelöst ansehen.

[Agarwal und Sharir 91] geben einen Algorithmus an, der den ersten Schnittpunkt in $O(n^{4/5})$ Zeit mit $O(n^{1+\varepsilon})$ Vorberechnungszeit findet.

[de Berg et al. 94] geben einen Algorithmus an, der zu einem beliebigen Strahl den ersten Schnittpunkt mit einem Dreieck in $O(\log n)$ Zeit findet. Um dies erreichen zu können, wird allerdings eine Datenstruktur mit einem Platzbedarf von $O(n^{4+\varepsilon})$ und ebenfalls $O(n^{4+\varepsilon})$ Vorberechnungszeit benötigt, was den Einsatz für Szenen interessanter Größe verhindert. Weiterhin stellen sie als offenes Problem die Frage, welche untere Schranke für die Vorberechnungszeit einer Datenstruktur gilt, die Strahlschnittpunkte in polylogarithmischer Zeit finden kann.

[Szirmay-Kalos und Márton 98] zeigen, daß im algebraischen Entscheidungsbaummodell das Raytracing mindestens $\Omega(\log n)$ Zeit benötigt und daß jeder Algorithmus, der Raytracinganfragen in $O(\log n)$ Zeit beantworten kann, mindestens $\Omega(n^4)$ Speicherplatz (und damit auch Vorberechnungszeit) für Suchdatenstrukturen benötigt.

Sie zeigen weiterhin, daß für spezielle Szenen (gleichmäßig dicht verteilte, konstant große Kugeln mit asymptotisch wachsendem Szenendurchmesser), eine Reihe von bekannten heuristi-

schen Verfahren eine mittlere Suchzeit von O(1) aufweisen. Die Annahmen über die Szene sind allerdings ziemlich stark; im Prinzip müssen die Szenen im Schnitt eine vollständige Verdekkung in konstanter Entfernung aufweisen.

Bekannte Heuristiken zur Beschleunigung des Raytracings, die mit linearem Speicherplatz auskommen, verwenden z.B. Octrees, reguläre Gitter oder allgemeinere Hierarchien von umfassenden Volumen zur Beschleunigung der Schnittpunktsuche. Ein Überblick findet sich in [Arvo und Kirk 91]. Die Effizienz dieser Methoden hängt von der Szene ab, auf die sie angewendet werden.

In einer praktischen Arbeit von [Wimmer et al. 98] wird Raytracing zum Erstellen einer Fernbereichsapproximation eingesetzt. Sowohl durch die besondere Form der Szenen (regelmäßige Stadtszenen, wenig Pixelfläche im Fernbereich sichtbar) sowie durch eine Amortisierung des Aufwandes für die Fernbereichsapproximation über mehrere Bilder konnte der Aufwand dabei in den Echtzeitbereich gesenkt werden.

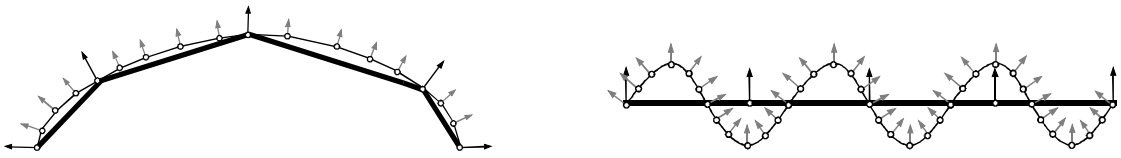
2.7 Grenzen der bekannten Verfahren

Effiziente Walkthroughanimation komplexer Szenen erfordern zwei Arten von Verbesserungen: die Reduzierung geometrischer Details und die Erkennung verdeckter Flächen.

„*mesh simplification*“ dient der Reduktion geometrischer Details. Das Verfahren liefert gute Ergebnisse, falls die Objekte eine einfache Topologie haben und die Feinstruktur der Oberfläche vernachlässigt werden kann.

In Walkthroughanimationen findet man allerdings in der Regel topologisch komplexe Szenen, die aus vielen einzelnen Objekten bestehen. Hier muß ein topologievereinfachendes Verfahren eingesetzt werden, das die Form des Objektes u.U. stark verändert. Das Jalousiebeispiel (Abbildung 8) kann z.B. überhaupt nicht sinnerhaltend in ein wesentlich vereinfachtes Dreiecksnetz überführt werden. Bei stark strukturierten Oberflächen sind ebenfalls Probleme zu erwarten: Um mit wenigen Dreiecken auskommen zu können, müssen kleinere Oberflächendetails vollständig entfernt werden. Dadurch werden Oberflächenattribute wie Farbe und Oberflächennormalen[8] u.U. stark verändert (siehe Abbildung 11) und die Schattierung später falsch berechnet. Für spezielle Fälle kann das Problem durch Oberflächentexturen ausgeglichen werden. Bei komplexerer Selbstverdeckung innerhalb der ersetzten Details oder auch bei blickpunktabhängiger Schattierung ist auch das nicht mehr möglich. Für topologisch komplexe Modelle und Modelle mit nicht unwesentlichen Oberflächendetails kann daher mit „*mesh simplification*“ oft keine ausreichende Vereinfachung erzielt werden.

[8] Die Normalen steuern in den meisten Schattierungsfunktionen wesentlich, wie Licht von der Oberfläche reflektiert wird.



günstiger Fall für „*mesh simplification*“ **ungünstiger Fall: viele Oberflächendetails**

Abbildung 11: Wiedergabe von Oberflächenattributen bei „*mesh simplification*“

Bildbasierte Verfahren sind unabhängig von der Topologie der zu ersetzenden Szenenteile. Es sind allerdings, wie ebenfalls das Jalousiebeispiel zeigt, Annahmen über die „Gutmütigkeit“ der Szene notwendig, um Grenzen für Parallaxefehler zu setzen. Statische Verfahren benötigen oft große Mengen an Speicherplatz und erlauben oft keine effiziente Handhabung von sich dynamisch verändernden Szenen. Dynamische Verfahren liefern eine gewisse Amortisierung, die aber das Problem auch unter günstigen Annahmen (zweidimensionale Szene gleichmäßig verteilter Objekte) nicht alleine lösen kann.

Das zweite Problem, die Erkennung verdeckter Flächen, erscheint schwieriger als die Anpassung der Detaillierung. Für brutale Ansätze, wie das Speichern aller Sichtbarkeitsregionen, lassen sich hohe polynomielle untere Schranken zeigen. Erfolgversprechender sind Verfahren, die die Sichtbarkeit approximativ vorberechnen und speichern. Die verbreiteten PVS-Methoden sind derzeit allerdings noch auf Szenentypen beschränkt, die eine günstige Zelleneinteilung erlauben.

Raytracingverfahren sind für große allgemeine Szenen nach dem heutigen Stand der Dinge nicht echtzeitfähig.

Der hierarchische z-Buffer liefert bei relativ allgemeinen Szenen gute Ergebnisse, ist aber darauf angewiesen, daß sich die verdeckte Geometrie gut durch hierarchische Boxen (z.B. eines Octrees) erfassen läßt. Der relativ hohe Grundaufwand macht zudem eine Hardwareimplementation wünschenswert, die derzeit nicht verfügbar ist.

Im allgemeinen Fall kann daher die Unterdrückung verdeckter Flächen als ungelöstes Problem angesehen werden. Für einige spezielle Strategien (wie das explizite Speichern exakter Sichtbarkeitsregionen oder Raytracing in logarithmischer Zeit) kann sogar gezeigt werden, daß das allgemeine Problem damit innerhalb annehmbarer Platzschranken nicht lösbar sein kann.

2.8 Randomisierte Algorithmen in der Computergrafik

Der randomisierte z-Buffer ist ein *„random sampling“*-Algorithmus: Er rekonstruiert ein Bild einer Szene aus einer Stichprobe von Punkten. Im Bereich der Computergraphik finden sich einige Verfahren, die nach diesem Paradigma arbeiten. Der Zufall wird dabei vor allem dazu eingesetzt, um die Bildqualität gegenüber deterministischen Verfahren, insbesondere in Bezug auf Aliasingartefakte, zu verbessern.

2.8.1 Monte Carlo Integration

[Glassner 95a] gibt eine Übersicht über randomisierte Verfahren zur numerischen Integration. Viele Renderingprobleme laufen im Kern auf die numerische Auswertung mehrdimensionaler Integrale hinaus. [Glassner 95a] zeigt, daß die arithmetische Mittelwertbildung aus n zufällig

über das Integrationsgebiet verteilten Funktionsauswertungen eine Konvergenzordnung von $O(\sigma/\sqrt{n})$ hat, wobei σ die Varianz der Stichproben angibt, also von der Stärke der Schwankungen in der zu integrierenden Funktion abhängt. Für viele Anwendungen ist dies allerdings zu langsam. Es werden verschiedene Methoden beschrieben, um die Konvergenz zu beschleunigen:

„Quasi Monte Carlo“: Statt einer rein zufälligen Auswahl von Stichprobenpunkten werden deterministische Punktfolgen verwendet, die eine gleichmäßige Verteilung der Stichproben garantieren.

„Adaptive Sampling“: Zunächst wird eine grobe Schätzung der Funktion aus wenigen Stichprobenpunkten erstellt, danach wird an Stellen, die eine größere Varianz aufweisen, die Stichprobe verfeinert.

„Weighted Monte Carlo“: Die Punkte werden mit dem Abstand zwischen ihnen gewichtet. Eine noch bessere Konvergenzordnung erzielt man, indem man den Flächeninhalt dadurch schätzt, daß man zwischen den zufälligen Stützpunkten linear interpoliert. Im mehrdimensionalen Fall können Voronoidiagramme der Stichprobenpunkte verwendet werden. Auf der Fläche jeder Voronoiregion wird ein konstanter Wert für den Funktionswert angenommen, d.h. die Funktionswerte werden mit der Größe der Voronoiregion gewichtet. Im eindimensionalen Fall verbessert der Ansatz der linearen Interpolation die Konvergenzrate auf $O(1/n^2)$ und die zweidimensionale Voronoimethode liefert $O(1/n)$, wenn angenommen werden kann, daß die zweite Ableitung der Funktion existiert und stetig ist. Diese Ergebnisse sind deutlich besser als bei der Standardmethode.

Weitere Verbesserungen sind möglich, wenn bestimmte Eigenschaften der zu integrierenden Funktion vorab bekannt sind (*„informed Monte Carlo“*). Für den randomisierten z-Buffer sind diese sowie auch die ersten beiden Methoden nicht interessant, da die Stichprobenauswahl nur bedingt gesteuert werden kann. In Abschnitt 3.2.3.4 wird eine Bildrekonstruktionsmethode beschrieben, die ähnlich wir das *„weighted Monte Carlo“* Voronoidiagramme benutzt, um den Einfluß von Stichprobenpunkten zu gewichten, allerdings werden diese direkt angezeigt und nicht zur Integration aufsummiert. Es wäre aber auch denkbar, einen ähnlichen Integrationsansatz zur Reduktion des Rauschen des erzeugten Bildes für das randomisierte z-Buffer Verfahren anzuwenden, dies wurde hier aber nicht weiter untersucht.

2.8.2 Anti-Aliasing, Distributed Raytracing und Bildrekonstruktion

Die meisten Renderingalgorithmen, wie Raytracing oder der z-Buffer Algorithmus, erzeugen ein Bild einer Szene, indem sie dieses punktweise abtasten und dann versuchen, aus den gewonnenen Informationen ein Bild zu rekonstruieren. Dies erfordert typischerweise die numerische Integration über die Bildfunktion gewichtet mit einem geeigneten Filter. Hier ist allerdings neben der reinen Integrationsgenauigkeit in jedem Pixel vor allem interessant, welche Effekte global im Bild durch die Bildrekonstruktion auftreten.

Meistens wird ein regelmäßiges Raster verwendet, auf dem die Abtastpunkte gewählt werden, typischerweise das Pixelraster des zu erzeugenden Bildes oder ein kleines Vielfaches davon (*„supersampling“*). Bilder, die im Fourierspektrum Frequenzen oberhalb der halben Rasterfrequenz enthalten, können durch die Abtastung nicht korrekt wiedergegeben werden (insbesondere kann kein Bild, das scharfe Kanten enthält, durch eine Abtastung mit endlicher Abtastfrequenz korrekt wiedergegeben werden [Foley et al. 96], Seite 627/628). Die Fehler, die dadurch entstehen, können bei einer regelmäßigen Abtastung als niederfrequente Spiegelfrequenzen („Aliasing“) wiedergegeben werden, die sich u.U. sehr grob strukturiert über das Bild

ziehen und den visuellen Eindruck stark stören können. In [Cook 91] wird ein Beispiel konstruiert, das zeigt, daß das Phänomen auch bei stärkerem „*supersampling*“ und anschließender Mittelwertbildung noch sehr störend zutage treten kann.

Eine Möglichkeit, die Auswirkungen der Störungen zu verhindern, ist die Abtastung an zufällig verstreuten Positionen anstatt auf einem regulären Gitter vorzunehmen. Der Fehler im damit konstruierten Bild wird dadurch zwar nicht kleiner, aber wie in [Cook 91] gezeigt wird, treten keine niederfrequenten Interferenzmuster auf, sondern die Fehler äußern sich in gleichmäßigem, unstrukturiertem Rauschen, das vom Menschen wesentlich weniger stark wahrgenommen wird als systematische Aliasingartefakte. Weiterhin wird gezeigt, daß unstrukturierte Abtastungsmuster mit einem garantierten Mindestabstand zwischen den Abtastpunkten zu hochfrequentem und damit weniger auffallendem, da weniger grob strukturiertem, Rauschen führt.

In [Cook 91] wird die Methode eingesetzt, um Aliasing beim Raytracing zu vermindern. Darüber hinaus wird gezeigt, wie beim rekursiven Raytracing vergleichsweise einfach eine Reihe von Effekten wie weiche Schatten, Tiefen- und Bewegungsunschärfe sowie rauhe Oberfläche durch zufällige Variation von Strahlparametern erreicht werden können (*„distributed raytracing“*). In den Beispielen werden pro Pixel 16-64 zufällig variierte Strahlen betrachtet und daraus der Mittelwert gebildet, um das Rauschen im Bild zu reduzieren.

[Glassner 95a] diskutiert in Kapitel 10 eine Reihe möglicher Abtastungsmuster, wobei ein Trade-Off zwischen Konstruktionsaufwand und Qualität gemacht werden muß. Für das randomisierte z-Buffer Verfahren sind diese nicht relevant, da bei der dort verwendeten Technik die Auswahl der Punkte im Gegensatz zum Raytracing nicht gesteuert werden kann (dies führt gerade zu dem Performancevorteil gegenüber dem Raytracing). Weiterhin wird die Rekonstruktion eines Bildes aus unstrukturierten Abtastungen diskutiert. Einige Methoden sind dabei auf von ihnen selbst vorgegebene Raster begrenzt und daher hier nicht anwendbar. Die allgemein anwendbaren Methoden lassen sich in drei Kategorien einteilen:

„warping“-Methoden: Sie versuchen die Abtastpunkte durch Verzerrung in ein reguläres Gitter zu überführen, dort das Bild zu rekonstruieren und dann die inverse Verzerrung darauf anzuwenden. Das Problem dabei ist die Konstruktion einer geeigneten Verzerrungsfunktion für zweidimensionale Signale.

„thin plate spline“ Methoden: Hier wird eine C^1 stetige, zweidimensionale Splinefunktion durch die Punkte gelegt. Die Stetigkeit des rekonstruierten Bildes ist allerdings für viele Bilder eher unnatürlich. Die Berechnung der Interpolation erfordert einen hohen Aufwand.

„local filtering“ Methoden: Die Punkte werden mit einer Filterfunktion gewichtet und der entsprechende Mittelwert wird im Bild eingetragen.

Von den Laufzeiten her kommt, wenn überhaupt, nur die letzte Verfahrensklasse für den Einsatz in Echtzeit in Frage. Die auch in [Cook 91] vorgeschlagene Variante dieser Klasse mit einem Gaußschen Rekonstruktionsfilter wird in Abschnitt 3.2.3.3 auf den randomisierten z-Buffer Algorithmus angewandt.

2.8.3 Weitere Algorithmen

Es finden sich noch weitere Einsatzgebiete für *„random sampling“*-Algorithmen in der Computergraphik. Ein Beispiel sind *„Monte Carlo Radiosity“*-Methoden [Glassner 95b]. Dabei soll eine physikalische Simulation der Interaktion von Photonen und Oberflächen einer Szene durchgeführt werden, um teilweise diffuse Spiegelungen und realistische Schatten zu berechnen. Der

Monte Carlo Algorithmus in der einfachsten Form sendet Photonen zufällig aus den Lichtquellen in die Szene aus und entscheidet beim Auftreffen auf einer Oberfläche zufällig nach vorgegebenen Verteilungen, ob diese absorbiert, und wenn nicht, in welche Richtungen sie abgelenkt werden. Das so entstehende Strahlungsbild wird in einer diskretisierten Version der Szene gespeichert und zum Erzeugen eines Bildes genutzt.

Die Methode erlaubt auf sehr einfache Weise, komplexe Vorgänge zu simulieren. Obwohl man relativ schnell einen sehr groben Eindruck der Lösung erhält, kann es sehr lange dauern, bis eine gute Lösung mit geringem Rauschen berechnet ist.

3 Randomisierter z-Buffer

In diesem Kapitel wird der in dieser Arbeit neu entwickelte randomisierte z-Buffer Algorithmus beschrieben. Das Verfahren beruht darauf, aus einer nach der projizierten Fläche verteilten, zufälligen Stichprobe von Oberflächenpunkte ein Bild der Szene zu rekonstruieren. Im ersten Abschnitt 3.1 wird eine Übersicht über den Algorithmus gegeben und eine Analyse der für das Verfahren wichtigen Eigenschaften der Zentralprojektion vorgenommen. In Abschnitt 3.2 wird die Konstruktion eines Bildes aus Stichprobenpunkten untersucht und in Abschnitt 3.3 wird die effiziente Auswahl von Stichprobenpunkten beschrieben.

3.1 Prinzip

3.1.1 Der Grobalgorithmus

Die Idee des randomisierten z-Buffers ist, eine zufällige Stichprobe von Punkten im Objektraum mit einer Verteilung und einem Stichprobenumfang so zu nehmen, daß sich daraus möglichst gut ein Bild in einer gegebenen Auflösung konstruieren läßt. Der Umfang der Stichprobe ist dabei unabhängig von der Detaillierung, mit der die Szene gespeichert ist. Das Verfahren führt automatisch dazu, daß sich die Detaillierung den Bedürfnissen des Standpunktes und der Bildauflösung anpaßt.

Damit eine Stichprobe den Bildinhalt repräsentativ wiedergibt, muß sie berücksichtigen, wieviel Fläche die Objekte auf dem Bild einnehmen. Kleine Objekte tragen weniger zum Bild bei als große Objekte. Daher wird die Stichprobe mit einer zur projizierten Fläche proportionalen Wahrscheinlichkeitsdichte ausgewählt, was dazu führt, daß die erfaßten Punkte gleichmäßig über das Bild verteilt sind. Die einzelnen Punkte werden unabhängig voneinander gewählt.

Zur Rekonstruktion eines Bildes aus diesen Punkten muß vor allem berücksichtigt werden, wie die Objekte sich untereinander verdecken. Dazu wird abgeschätzt, wie groß die Dichte der Stichprobe sein muß, um diese Frage relativ sicher beantworten zu können. Die Bildrekonstruktion wird in Kaptitel 3.2 genauer beschrieben.

Ein weiteres Problem besteht darin, wie man die Stichprobe effizient nehmen kann. Dazu müssen dynamisch die projizierten Flächen der Objekte geschätzt werden. Nur wenn dies deutlich schneller zu bewerkstelligen ist als der Bildaufbau mit klassischen Renderingverfahren, lohnt sich das Verfahren. Da die projizierte Fläche eines Objektes von seiner tatsächlichen Fläche, von der Entfernung zum Betrachter und von seiner Orientierung relativ zur Blickrichtung abhängt, werden entsprechende Suchdatenstrukturen eingesetzt, die den Raum bezüglich Position und Orientierung einteilen, um die Objekte schnell klassifizieren zu können. Die Verteilung der Stichprobenpunkte wird in Kaptitel 3.3 behandelt.

Der bisherige Grobalgorithmus stellt sich wie folgt dar:

```
1) Wähle eine Stichprobe Q von Oberflächenpunkten verteilt nach
   der projizierten Fläche
2) Entferne alle verdeckten Punkte aus Q
3) Rekonstruiere aus den verbliebenen Punkten das Bild
```

Algorithmus 1: Grobalgorithmus für den randomisierten z-Buffer

3.1.2 Die Verteilungsfunktion

Die Wahrscheinlichkeit für einen Punkt, in die Stichprobe aufgenommen zu werden, soll proportional zu der Größe sein, in der ein Oberflächenstück in seiner Umgebung auf dem Bildschirm erscheint. Dabei wird nicht berücksichtigt, ob dieses durch andere Szenenteile verdeckt wird, da diese Frage schwierig zu entscheiden sein kann. Im folgenden wird zunächst die Projektionsfunktion hergeleitet, die einem (infinitesimal) kleinen Oberflächenstück die projizierte Fläche auf dem Bildschirm zuordnet. Dazu wird angenommen, daß die Szene in einer Zentralprojektion, wie in Kapitel 1.4.2 beschrieben, dargestellt wird.

Zunächst sei dazu der Spezialfall angenommen, daß alle Flächen parallel zur Bildebene sind. Man erhält folgende Situation (Abbildung 12):

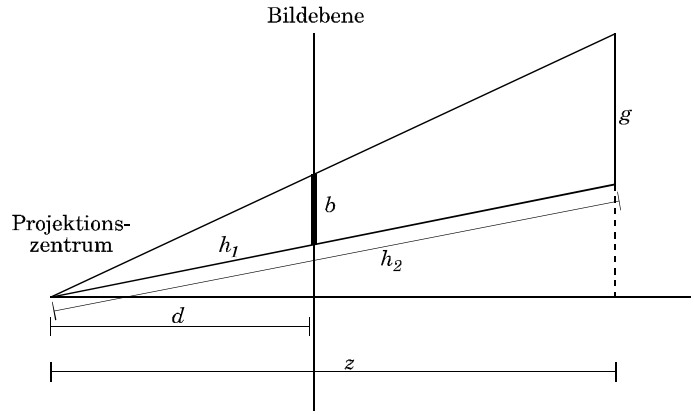


Abbildung 12: Projektion von zur Bildebene parallelen Flächen

Betrachte eine Schnittebene senkrecht durch die Bildebene. Sei g die Länge des Segmentes, das durch den Schnitt eines Dreieckes entsteht, und sei b die Länge der Projektion des Segmentes in die Bildebene, z die Tiefe des Segmentes, d.h. die Projektion des Abstandsvektors auf die senkrecht zur Bildebene stehende Blickrichtung und d der Brennweitenfaktor. Dann erhält man mit den Strahlensätzen:

$$\frac{g}{b} = \frac{h_2}{h_1} = \frac{z}{d}, \quad \text{also:} \quad b = g \cdot \frac{d}{z}$$

Für zweidimensionale Flächen tritt dieser Faktor in beiden Koordinatenrichtungen auf, so daß für die projizierte Fläche A_B eines planaren Oberflächenstückes der Oberfläche A_G gilt:

$$A_B = A_G \cdot \frac{d^2}{z^2}$$

Im nächsten Schritt sei nun angenommen, daß die Oberflächen senkrecht auf dem Verbindungsvektor v zwischen ihrem Mittelpunkt und dem Projektionszentrum stehen. Da infinitesimale Oberflächenstücke betrachtet werden, hat die Wahl des Mittelpunktes keinen Einfluß auf das Ergebnis, und das Projektionszentrum kann als unendlich weit entfernt angenommen werden. Die Situation wird in Abbildung 13 gezeigt. Um die projizierte Fläche zu bestimmen, wird zunächst das Segment g so gedreht, daß es parallel zur Bildebene steht (Segment g'). Man erhält mit α als Winkel zwischen Objekt und v:

$$g = g' \cos\alpha, \quad \text{also:} \quad g' = \frac{g}{\cos\alpha} \quad \text{und damit:} \quad b = g \cdot \frac{1}{\cos\alpha} \cdot \frac{d}{z}$$

Analog erhält man im Fall einer zweidimensionalen Oberfläche mit r_n als normiertem Blickrichtungsvektor und v_n als normiertem Verbindungsvektor:

$$A_B = A_G \cdot \frac{1}{v_n \circ r_n} \cdot \frac{d^2}{z^2} = A_G \cdot \frac{d^2}{(v_n \circ r_n) \cdot (v \circ r_n)^2}$$

Zum Schluß sei eine beliebige Orientierung der Oberflächen erlaubt (Segment g_2 in Abbildung 14). Dann muß die Oberfläche zunächst so gedreht werden, daß sie senkrecht auf der Verbindung zum Projektionszentrum steht, und danach kann in die bisher gewonnene Formel eingesetzt werden. Wenn n die Normale der Oberfläche in dem betrachteten Punkt ist, erhält man:

$$A_B = A_G \cdot d^2 \cdot \frac{\cos\beta}{(v_n \circ r_n) \cdot (v \circ r_n)^2} = A_G \cdot d^2 \cdot \frac{v_n \circ n}{(v_n \circ r_n) \cdot (v \circ r_n)^2}$$

$\cos\beta$ kann auch negativ sein, falls die Normale der Oberfläche nicht dem Betrachter zugewandt ist und damit ein Winkel größer als $\pi/2$ entsteht. Als Konvention werden Oberflächen, die dem Betrachter abgewandt sind, nicht dargestellt (siehe Abschnitt 1.4.1.1). Daher kann die projizierte Fläche eines solchen Oberflächenstückes als null angesehen werden. Formal wird dies ausgedrückt, indem die Funktion $cut_{[0,1]}$ auf den Kosinus bzw. das Skalarprodukt angewendet wird. Diese sei wie folgt definiert:

$$cut_{[a,b]}: \mathbb{R} \to \mathbb{R}, \quad cut_{[a,b]}(x) := \begin{cases} x, & \text{falls } x \in [a,b] \\ a, & \text{falls } x < a \\ b, & \text{falls } x > b \end{cases}$$

Der Faktor, um den die projizierte von der absoluten Fläche abweicht, wird im folgenden für alle $s \in S$ für eine Szene S mit $prj(s)$ bezeichnet, und ergibt sich insgesamt wie folgt:

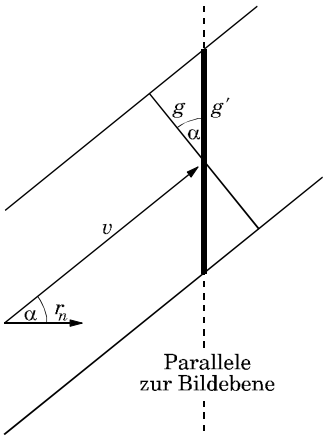


Abbildung 13: Projektion von Flächen senkrecht zur Verbindung zum Projektionszentrum

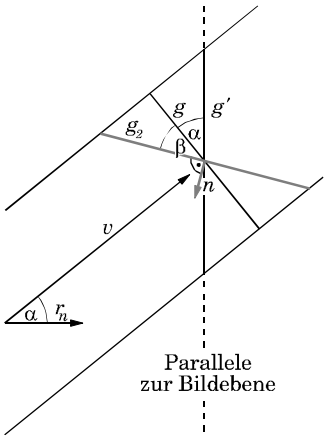


Abbildung 14: allgemeiner Fall

Definition 6: *Projektionsfaktor*

Sei S eine Szene, $s \in S$, v der Vektor vom Projektionszentrum zu s, n die Normale des Dreiecks, zu dem s gehört[9], r die Blickrichtung und d der Brennweitenfaktor der Zentralprojektion. Weiter seien v_n und r_n die normierten Vektoren zu v und r und n sei ebenfalls normiert. Dann sei

$$prj(s) := d^2 \frac{cut_{[0,1]}\left(v_n \circ n\right)}{\left(v_n \circ r_n\right) \cdot \left(v \circ r_n\right)^2}$$

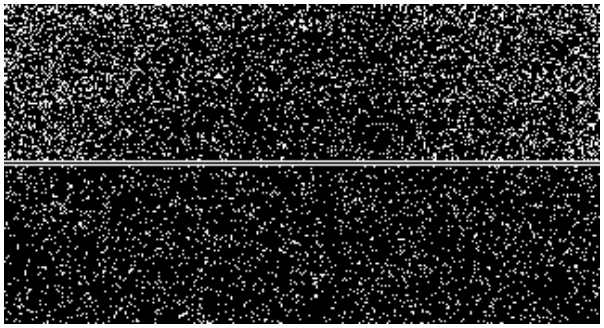
der *Projektionsfaktor an der Stelle* s.

Wenn man in den Skalarprodukten in Definition 6 die entsprechenden Winkel einsetzt, läßt sich der Projektionsfaktor auch als

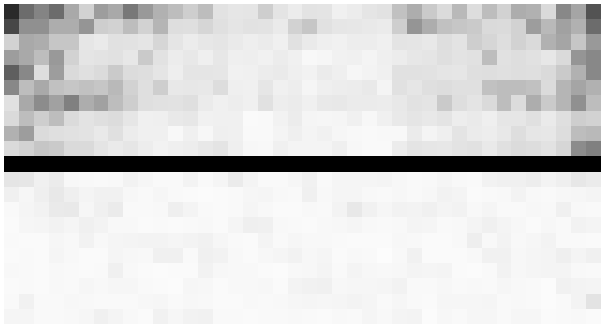
$$prj(s) := \frac{d^2 \cdot \cos\beta}{z^2 \cdot \cos\alpha}$$

ausdrücken, wobei β den Winkel, unter dem die Oberfläche gesehen wird (der Winkel zwischen der Normalen und der Verbindung zu s), z den Tiefenwert von s und α den Winkel, um den der Punkt im Bild von der Bildmitte abweicht, angibt. Der Projektionsfaktor besteht aus drei nicht konstanten Teilfaktoren: dem *Tiefenfaktor* $1/z^2$, dem *Orientierungsfaktor* $\cos\beta$ und dem *Verzerrungsfaktor* $1/\cos\alpha$. Der Tiefenfaktor trägt dem Umstand Rechnung, daß die projizierte Fläche mit der Entfernung quadratisch abnimmt. Der Orientierungsfaktor beschreibt die Verringerung der sichtbaren Fläche unter flachen Blickwinkeln. Der Verzerrungsfaktor beschreibt die besondere Eigenschaft der Zentralprojektion, daß Objekte zum Rand hin verzerrt, auseinandergezogen dargestellt werden. Dieser Faktor liegt für normale „Brennweiten“ allerdings in einem kleinen Bereich: Bei einem diagonalen Blickwinkel von 70° (der dem menschlichen Betrachter natürlich erscheinende Normalblickwinkel liegt etwa bei 45°) liegt dieser Faktor im Intervall [1...1,22].

Abbildung 15 zeigt eine senkrecht zur Blickrichtung stehende Fläche, auf der Punkte zufällig



oben ohne Berücksichtigung / unten mit Berücksichtigung des Verzerrungsfaktors, Blickwinkel 70° vertikal, 79° diagonal



dasselbe Bild nach Invertierung , Mittelwertbildung und Kontrastverstärkung

Abbildung 15: Einfluß des Verzerrungsfaktors

[9] Hier sei vorausgesetzt, daß dies auch an gemeinsamen Kanten und Schnitten von Dreiecken durch eine entsprechende Vereinbarung eindeutig festgelegt ist.

nach der oben definierten Wahrscheinlichkeitsdichte gewählt und schwarz gefärbt wurden, einmal mit und einmal ohne Berücksichtigung des Verzerrungsfaktors. Da der Einfluß des Verzerrungsfaktors relativ schwach ist und insbesondere das Verhältnis des größten zum kleinsten möglichen Wert für jeden festen Wert von d durch eine Konstante beschränkt ist, wird er für die Verteilung der Stichprobenpunkte im folgenden an einigen Stellen vernachlässigt.

Um die Wahrscheinlichkeitsverteilung für die Stichproben zu definieren, wird der Projektionsfaktor im sichtbaren Teilgebiet der Szene, d.h. in der Sichtpyramide der gegebenen Zentralprojektion, als Wahrscheinlichkeitsdichte aufgefaßt. Zunächst wird eine Funktion definiert, die die Sichtpyramide beschreibt:

Definition 7: *Sichtpyramide*

Sei *frustum*: $\mathbb{R}^3 \to \{0,1\}$ die Funktion, die jedem Punkt des Raumes genau dann eine 1 zuordnet, wenn er sich in der Sichtpyramide des aktuellen Betrachters befindet, mindestens eine vorgegebene Mindesttiefe $mindepth > 0$ und maximal eine vorgegebene Höchsttiefe vom $maxdepth > mindepth$ hat.

Wenn kein Wert für maxdepth angegeben ist, wird der Wert ∞ angenommen. Die Sichtpyramide hat ihre Spitze im Projektionszentrum, die Symetrieachse in Richtung der Blickrichtung und einen quadratischen Querschnitt mit dem Breiten / Höhenverhältnis des zu berechnenden Bildes. *mindepth* und *maxdepth* beschreiben Begrenzungsebenen senkrecht zur Blickrichtung. Die Vorgabe einer Mindesttiefe ist notwendig, da die Zentralprojektion für die Tiefe $z=0$ singulär wird (bei der Projektion wird durch z dividiert). Die Höchsttiefe wird nur eingeführt, um später die Beschreibung zu vereinfachen, sie wird im weiteren stets so gewählt, daß sie mindestens so groß wie die Tiefe aller sichtbaren Szenenteile ist.

Definition 8: *Wahrscheinlichkeitsdichte und projizierte Fläche*

Sei $p_{sample}(s) := \dfrac{frustum(s) \cdot prj(s)}{\int_S frustum(x) \cdot prj(x) dx}$ die *nach der projizierten Fläche gebildete Wahrscheinlichkeitsdichte*. Dann sei die Wahrscheinlichkeit, ein Stichprobenpunkt x in $G \subseteq S$ zu nehmen, gegeben durch $P(x \in G) = \int_G p_{sample}(t)\, dt$.

Das Integral $\int_G frustum(x) prj(x)\, dt$ über die Menge G aller Punkte eines Objektes $G \subseteq S$ wird im folgenden als *projizierte Fläche* des Objektes bezeichnet.

Es sei vorausgesetzt, daß die Mengen S und G stets so beschaffen sind, daß alle Integrale existieren.

3.2 Bildrekonstruktion

Die erste Frage, die sich nach Betrachten des Grobalgorithmus stellt, ist sicherlich, ob es überhaupt möglich ist, aus einer hinreichend kleinen Stichprobe ein sinnvolles Bild zu rekonstruieren. In den nächsten Unterkapiteln werden verschiedene Methoden für diese Aufgabe verglichen. Das wichtigste Ergebnis wird sein, daß der Stichprobenumfang, der notwendig ist, um mit hoher Wahrscheinlichkeit ein korrektes Bild zu erhalten, bei allen Methoden um einen logarithmischen Faktor superlinear mit der Auflösung und linear mit der projizierten Gesamtfläche, d.h. der Summe aus sichtbarer und verdeckter projizierter Fläche, wächst. Zusätzlicher Aufwand bei der Rekonstruktion des Bildes aus einer gegebenen Stichprobe kann eine Verbesserung der Bildqualität bringen, die Auflösung der Kanten der Objekte, d.h. der Linien, an denen sich die Verdeckung zwischen den Objekten ändert, ist aber im wesentlichen durch die Größe der Stichprobe festgelegt.

3.2.1 Pixelweise Rekonstruktion

Grundlage für alle weiteren Rekonstruktionsmethoden ist die pixelweise Rekonstruktion. Das Vorgehen ist kanonisch: Alle Punkte werden projiziert und als einzelne Pixel mit dem bekannten z-Buffer Algorithmus eingezeichnet, d.h. die Pixel werden nur dann in den Bildspeicher geschrieben, wenn ihre Tiefe kleiner als die des zuvor dort vorhandenen Pixels ist. Als Pseudocode:

```
Sei im[w][h] der Bildspeicher und zb[w][h] der Tiefenpuffer,
auf maximale Werte initialisiert
Wähle eine Stichprobe Q aus S, verteilt nach der proj. Fläche
Für alle Punkte q ∈ Q:
   p := projektion(q)
   falls zb[p.x][p.y] > p.tiefe dann
      im[p.x][p.y] := farbe(q)
      zb[p.x][p.y] := p.tiefe
```

Algorithmus 2: Pixelweise Rekonstruktion

Die Frage ist nun, wie groß die Stichprobe sein muß, damit durch den z-Buffer alle Verdeckungen richtig wiedergegeben werden oder anders ausgedrückt, wie viele zufällige Pixel man benötigt, um alle sichtbaren Flächen (mit hoher Wahrscheinlichkeit) vollständig zu bedecken.

3.2.1.1 Erkennung der Verdeckung innerhalb eines Pixels

Der z-Buffer behandelt alle Pixel des Bildes gleichartig und unabhängig voneinander. Daher wird zuerst die Konstruktion eines einzelnen Pixels betrachtet. Dazu werden alle Objektfragmente betrachtet, die in der durch den Pixel definierten Sichtpyramide liegen (siehe Abbildung 16).

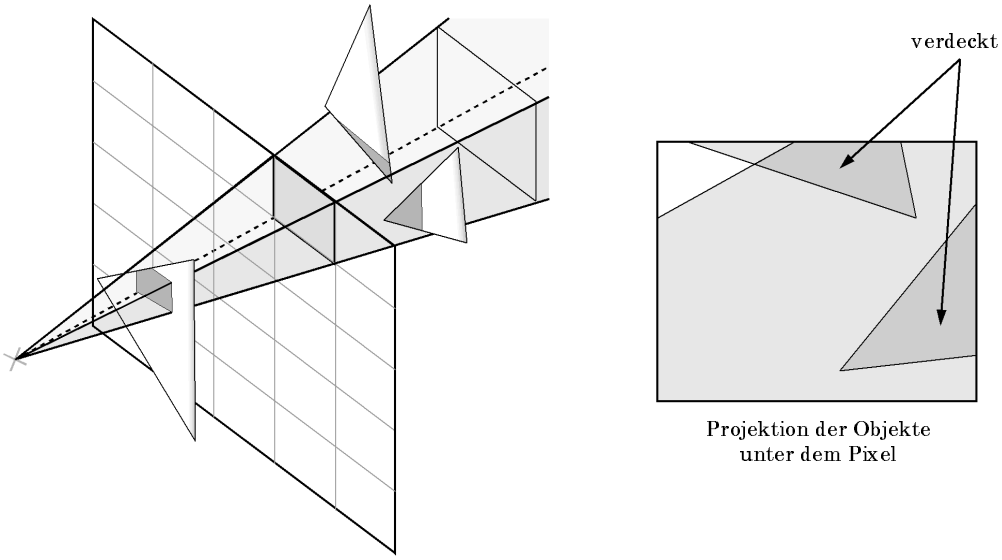


Abbildung 16: „Unter" einem Pixel sichtbare und verdeckte Objektfragmente

In diesem Abschnitt sei v die projizierte Fläche der sichtbaren Objektfragmente und h die der verdeckten Objektfragmente innerhalb eines Pixels, gemessen in Pixelbruchteilen. Es kann ohne Einschränkungen $v = 1$ angenommen werden, dazu kann die Szene z.B. in eine Kugel in Hintergrundfarbe eingeschlossen werden.

Es wird nun eine auf der projizierten Fläche gleichverteilte Stichprobe gewählt. Der Stichprobenumfang sei so gewählt, daß für alle Pixel des Bildes alle unter jedem Pixel liegenden Objektteile, die mindestens eine projizierte Fläche von 1 haben, mit hoher Wahrscheinlichkeit mindestens mit einem Punkt in der Stichprobe vertreten sind. Die Wahrscheinlichkeit für andere Werte steigt bzw. fällt proportional zur projizierten Fläche des Objektteiles. Im nächsten Abschnitt wird hergeleitet, wie groß die Stichprobe sein muß, um diese Annahmen erfüllen zu können, und die Wahrscheinlichkeiten werden genauer quantifiziert. Unterscheide nun drei Fälle:

1. Keine Verdeckung

Die vom Sichtvolumen des Pixels geschnittenen Objektteile verdecken sich nicht (Abbildung 17 (a)). In dem Fall erhält man mit hoher Wahrscheinlichkeit ein korrektes Bild nach Definition 3, da alle Stichprobenpunkte in der Farbe eines sichtbaren Objektteiles ausgewählt werden und die Stichprobe mit hoher Wahrscheinlichkeit mindestens 1 Punkt für den Pixel enthält.

2. Getrennte Verdeckung

Alle sichtbaren Objektteile haben eine Tiefe, die geringer ist als die kleinste Tiefe eines verdeckten Objektteils. Da $v = 1$ ist, bedeutet das, daß vor allen verdeckten Objektteilen eine den gesamten Pixel bedeckende Front von sichtbaren Objektteilen liegt (Abbildung 17 (b)). Mit hoher Wahrscheinlichkeit wird ein sichtbarer Objektpunkt gewählt, der alle anderen Punkte im z-Buffer überschreiben wird. Daher wird mit entsprechend hoher Wahrscheinlichkeit ein korrektes Bild produziert.

3. Die übrigen Fälle

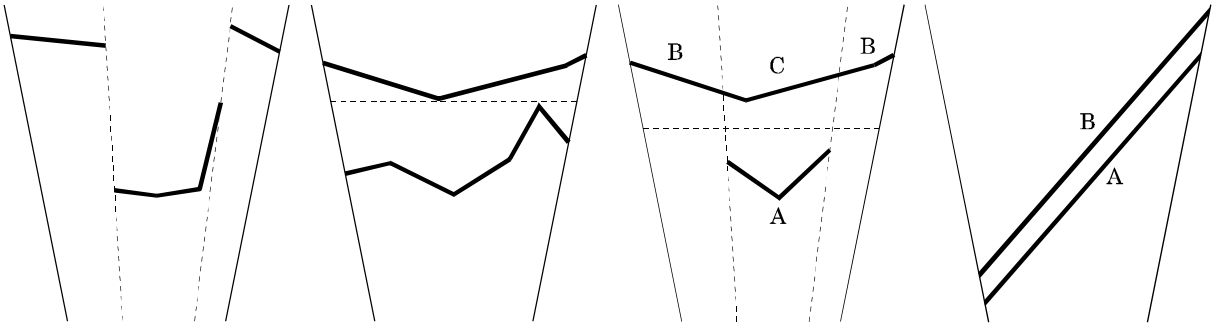
Wenn die Verdeckung innerhalb des Pixels nicht getrennt im Sinne von Fall 2 ist, kann es mit einer relevanten Wahrscheinlichkeit zu falschen Bildern kommen. Typische Situationen sind in Abbildung 17 (c) und (d) dargestellt.

In Abbildung 17 (c) wird ein Objektfragment C durch ein Fragment A verdeckt, welches nicht den gesamten Pixel überdeckt. Mit einer gewissen Wahrscheinlichkeit wird nun ein Punkt aus C in die Stichprobe aufgenommen, während die Wahrscheinlichkeit, daß auch ein verdeckender Punkt aus A aufgenommen wird, nicht hoch genug sein muß, um meistens ein korrektes Bild zu liefern, da die projizierte Fläche von A kleiner als 1 Pixel ist. Auch die weiteren sichtbaren Teile B können C nicht verdecken, da sie eine größere Tiefe haben.

Falls sich die Farbe von C nicht (oder nicht stark) von den in A oder B zu findenden Farben unterscheidet, erhält man immer noch ein korrektes (oder näherungsweise korrektes) Bild. Dies ist insbesondere zu erwarten, wenn C und B zu ein und dem selben Objekt gehören. Eine solche Situation findet sich an quasi allen Kanten von Objekten im Bild, da diese nur zufällig genau an Pixelgrenzen verlaufen werden und daher das Vordergrundobjekt den Hintergrund nur teilweise bedeckt. Der Fall, daß ein verdeckter sub-Pixel kleiner Bereich C sich in der Farbe völlig von seiner sichtbaren Umgebung im selben Pixel unterscheidet, ist nur in wenigen Ansichten zu erwarten, und selbst dann ist fraglich, ob ein solcher Fehler bemerkt würde. Daher wird dieser Fall in praktischen Anwendungen eher selten Probleme bereiten.

In Abbildung 17 (d) wird ein anderer Fall dargestellt, in dem zwei ausgedehnte Oberflächen A und B einen zu geringen Tiefenunterschied besitzen. Wenn A und B sich nur minimal in der Tiefe unterscheiden, verteilen sich ihre Tiefenwerte etwa im gleichen Intervall. Dadurch scheint mit bis zu 50% Wahrscheinlichkeit die Oberfläche B durch die Oberfläche A durch, da ein zufällig aus B gezogener Punkt mit dieser Wahrscheinlichkeit eine größere Tiefe als ein zufällig aus A gezogener Punkt haben kann. Wenn die von Objektteilen vom Typ B bedeckte Fläche größer als 1 Pixel ist, d.h. wenn mehrere verdeckte Ebenen vorhanden sind, verschlechtert sich das Verhältnis noch entsprechend.

Dieser Fall tritt vor allem auf, wenn Oberflächendetails durch aufgesetzte Polygone modelliert werden („Oberflächendetailpolygone“): z.B. an einem Haus, dargestellt durch einen Würfel, werden Fenster, dargestellt durch ein Viereck, angebracht, die nur einen sehr geringen Abstand zur Hauswand haben, damit dies bei seitlicher Betrachtung nicht auffällt. Bei derartigen Szenen liefert das randomisierte z-Buffer Verfahren deutlich sichtbare Fehler: Die Oberflächendetails erscheinen halbdurchsichtig. Trotzdem ist dieser Fehler keine schwere Einschränkung, da sich



a) keine Verdeckung **b) getrennte Verdeckung** **c) nicht getrennte Verdeckung (1)** **d) nicht getrennte Verdeckung (2)**

Abbildung 17: verschiedene Verdeckungssituationen innerhalb eines Pixels

diese Situation leicht vermeiden läßt, z.B. durch die Verwendung von Texturen zur Modellierung von Oberflächendetails oder das Auslassen einer entsprechenden Lücke im Hintergrund hinter dem Vordergrund.

Abbildung 18 zeigt ein Szene aus mehreren Straßenzügen. Die Asphaltflächen und die Fahrbahnmarkierungen wurden dabei mit Oberflächenpolygonen mit einem relativen Abstand von 1:2000 (gemessen am Szenendurchmesser) modelliert. Die mit dem randomisierten z-Buffer erstellte Version zeigt deutlich, daß der Untergrund unter der Straße durchscheint. Allerdings zeigt auch das mit dem konventionellen z-Buffer erstellte Bild Fehler, die durch Rundungsfehler durch die beschränkte Genauigkeit des z-Buffers auftreten. Diese sind jedoch schwächer als beim randomisierten Verfahren und leichter (z.B. durch Erhöhen der Tiefenauflösung) zu vermeiden. Beide Bilder wurden mit einem 16-Bit Integer-z-Buffer berechnet.

In Experimenten stellte sich heraus, daß Szenen vom in Abbildung 17 (d) gezeigten Typ die meisten Probleme verursachten, in anderen Situationen traten bei den untersuchten Testszenen keine nennenswerten Fehler auf (siehe Kapitel 5.2).

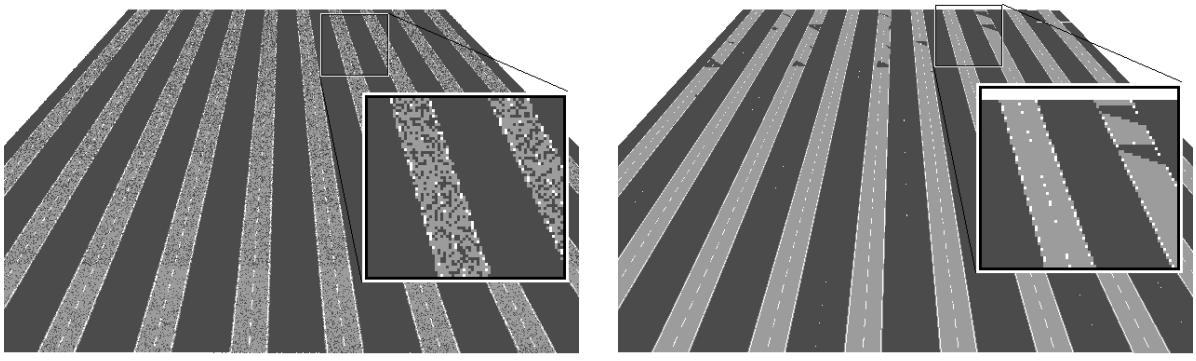
Zusammenfassend kann man festhalten:

Definition 9: *Ansicht einer Szene mit getrennter Verdeckung*

Falls eine Szene für eine feste Projektion und Rasterauflösung, zusammen als *Ansicht* bezeichnet, für alle Pixel die Annahmen (1) oder (2) aus obiger Fallunterscheidung erfüllt, d.h. es ist $v = 1$ in jedem Pixel und es existiert kein verdeckter Objektpunkt in geringerer Entfernung als ein sichtbarer im selben Pixel, heißt diese Ansicht eine *Ansicht mit getrennter Verdeckung*.

Lemma 1: *Pixelrekonstruktion*

In Algorithmus 2 sei mit Wahrscheinlichkeit von mindestens p garantiert, daß die sichtbaren Objektteile in jedem Pixel in der Stichprobe mit einem Punkt vertreten sind. Dann produziert der Algorithmus für jede Ansicht mit getrennter Verdeckung mit einer Wahrscheinlichkeit von mindestens p ein korrektes Bild im Sinne von Definition 3. Die Aussage trifft ebenfalls für Ansichten ohne getrennte Verdeckung zu, bei denen die Farben von verdeckten Objektpunkten, die zu nah am Betrachter sind, auch in jeweils einem sichtbaren Objektpunkt im selben Pixel zu finden sind.



randomisierter z-Buffer **konventioneller z-Buffer**

Abbildung 18: Artefakte durch nicht getrennte Verdeckung

3.2.1.2 Größe der Stichprobe

Als nächstes soll nun bestimmt werden, wie groß eine Stichprobe sein muß, um die im vorherigen Abschnitt angenommenen Voraussetzungen erfüllen zu können. In diesem Abschnitt sei, anders als im letzten Abschnitt, v die sichtbare projizierte Fläche und h die verdeckte projizierte Fläche im *gesamten* Bild in Pixeln, Objektteile außerhalb der Sichtpyramide nicht mitgezählt. Weiter sei $a := v + h$ die gesamte projizierte Fläche innerhalb der Sichtpyramide. Es sei wieder vorausgesetzt, daß in jedem Pixel des Bildes die Summe aller sichtbaren projizierten Flächen 1 ist (d.h. nirgendwo scheint der Hintergrund durch). v ist also damit gleich der Anzahl von Pixeln des Bildes. v und h seien ohne Einschränkungen ganzzahlig, der dadurch verursachte relative Fehler kann bei einer Auflösung von typischerweise mehreren hunderttausend Pixeln vernachlässigt werden.

Die Frage ist nun, welchen Umfang eine nach der projizierten Fläche verteilte Stichprobe haben muß, damit in die sichtbaren Oberflächenfragmente jedes Pixels mindestens 1 Punkt fällt. Sei zunächst $h = 0$ angenommen, d.h. die Szene enthalte keinerlei Verdeckungen. Dann reduziert sich die Auswahl von Objektpunkten auf die Auswahl von Pixeln mit Wahrscheinlichkeit $1/v$. Die Frage ist also, wie viele Pixel man unabhängig voneinander mit gleicher Wahrscheinlichkeit auswählen muß, um relativ sicher sein zu können, daß alle Pixel mindestens einmal berücksichtigt werden.

In der Literatur wird dieses Szenario unter dem Begriff *„occupancy problems"* geführt [Motwani und Raghavan 95]: In n Behälter werden unabhängig und zufällig gleichverteilt m (gleichartige) Bälle geworfen. Die hier gestellt Frage lautet dann in dieser Formulierung: Wie viele Würfe sind notwendig, damit mit hoher Sicherheit in jeden Behälter mindestens 1 Ball fällt? Diese spezielle Frage wird auch als *„coupon collector's problem"* bezeichnet. Hier soll zunächst eine elementare Schranke für dieses Problem hergeleitet werden, danach werden asymptotische Ergebnisse aus [Motwani und Raghavan 95] angewendet.

Eine elementare Rechnung

Im folgenden wird eine Formel hergeleitet, mit deren Hilfe die genaue Mindestanzahl von Bällen berechnet werden kann, für die alle Behälter mit vorgegebener Sicherheit einen Ball erhalten. Diese kann dazu eingesetzt werden, den Renderingalgorithmus zu steuern.

Seien n Behälter gegeben, die mit m zufälligen, gleichverteilten, unabhängigen Ballwürfen gefüllt werden. Weiter sei $g = 1/n$ die Wahrscheinlichkeit, mit der bei jedem Wurf ein fester Behälter gewählt wird. Die Wahrscheinlichkeit dafür, daß ein fester Behälter nicht gewählt wird, ist dann $(1\text{-}g)$. Da alle Würfe unabhängig voneinander getätigt werden, ist die Wahrscheinlichkeit dafür, daß ein Behälter auch nach m Würfen noch leer bleibt, $(1\text{-}g)^m$. Damit ein fester Behälter mit einer Sicherheit von mindestens $\bar{s}$ in der Stichprobe enthalten ist, muß also gelten:

$$1-(1-g)^m \geq \bar{s}, \text{ also } (1-g)^m \leq 1-\bar{s}, \text{ oder äquivalent } m\log(1-g) \leq \log(1-\bar{s}),$$

$$\text{was gegeben ist bei } m \geq \frac{\log(1-\bar{s})}{\log(1-g)}$$

$\bar{s}$ muß nun so gewählt werden, daß die Wahrscheinlichkeit dafür, daß alle bzw. zumindest viele Behälter einen Ball erhalten, insgesamt mindestens s ist.

Damit mindestens $l \in \{1 \ldots n\}$ Behälter mit einer Wahrscheinlichkeit von mindestens s einen Ball erhalten, muß $\bar{s}^l \geq s$ gelten, da die Wahrscheinlichkeit, mindestens einen Ball zu erhalten, nach

m Würfen für alle Behälter unabhängig voneinander gleich $\bar{s}$ ist. Also muß $\bar{s} \geq s^{1/l}$ sein und eingesetzt erhält man:

$$m \geq \frac{\log\left(1 - s^{1/l}\right)}{\log(1-g)}$$

Diese Schranke ist dicht, da keine Abschätzungen verwendet wurden. Man erhält:

Lemma 2: *Eine Schranke für das „coupon collector's problem" mit geg. Sicherheit*

Seien n Behälter gegeben, in die m Bälle zufällig, mit gleicher Wahrscheinlichkeit und unabhängig voneinander geworfen werden. Sei $g = 1/n$. Damit mit einer Wahrscheinlichkeit von mindestens s höchstens $k < n$ Behälter leer bleiben, sind

$$m \geq \frac{\log\left(1 - s^{\frac{1}{n-k}}\right)}{\log(1-g)}$$

Würfe notwendig. Insbesondere sind

$$m \geq \frac{\log\left(1 - s^{g}\right)}{\log(1-g)}$$

Würfe notwendig, damit mit einer Wahrscheinlichkeit von s kein Behälter leer bleibt. Diese Schranken sind dicht.

Der Ausdruck läßt sich unter der Annahme, daß n sehr groß und g damit sehr klein ist, noch vereinfachen. Nach [Bronstein et al. 97] läßt sich der Nenner in eine Taylorreihe entwickeln: Für $-1 \leq g < 1$ (und damit insbesondere in dem für g möglichen Bereich) gilt:

$$\log(1-g) = -\sum_{k=1}^{\infty} \frac{g^k}{k} = -\left[g + O(g^2)\right] \doteq -g \quad \text{(für kleine g)}$$

Damit erhält man:

Lemma 3: *Schranke für große Werte von n*

Für große Werte von n lassen sich die Schranken aus Lemma 2 auf

$$m \geq n \log\left(1 - s^{\frac{1}{n-k}}\right) \text{ (für k leere Behälter),}$$

$$\text{bzw. } m \geq n \log\left(1 - s^{1/n}\right) \text{ (keine leeren Behälter)}$$

vereinfachen.

Diese Formulierung läßt noch nicht klar den asymptotischen Verlauf der Stichprobengröße erkennen. Anhand von Beispielen (siehe Abbildung 21) läßt sich ein Wachstum mit $\Theta(n \cdot \log n)$ vermuten. Ein Antwort findet sich in [Motwani und Raghavan 95]:

Asymptotische Ergebnisse

[Motwani und Raghavan 95] zeigen folgende Sätze zu *„occupancy problems"*:

> **Satz 1:** [Motwani und Raghavan 95], S. 57f
>
> Der Erwartungswert für die Anzahl Bälle, die unabhängig, zufällig gleichverteilt in n Behälter geworfen werden müssen, bis jeder Behälter einen Ball enthält, ist $n \cdot H_n$, wobei $H_n := \sum_{k=1}^{n} 1/k$ die n-te Harmonische Zahl ist.
>
> **Satz 2:** [Motwani und Raghavan 95], S. 61ff
>
> Sei X die Zufallsvariable, die die Anzahl von unabhängig gleichverteilten Ballwürfen angibt, die notwendig sind, um in jeden von n Behältern einen Ball zu plazieren. Dann gilt für alle $c \in \mathbb{R}$:
>
> $$\lim_{n \to \infty} \text{Prob}[X > n \ln n + cn] = 1 - e^{-e^{-c}}$$
>
> **Satz 3:** [Motwani und Raghavan 95], S. 94ff
>
> Sei X die Zufallsvariable, die die Anzahl von Behältern angibt, die nach m unabhängig gleichverteilten Ballwürfen in n Behälter leer geblieben sind. Dann gilt:
>
> $$E(X) = n\left(1 - \frac{1}{n}\right)^m \approx n \cdot e^{-m/n}$$
>
> Für $\lambda > 0$ gilt:
>
> $$\text{Prob}\left[\left|X - E(X)\right| \geq \lambda\right] \leq 2e^{\left(-\frac{\lambda^2 (n-1/2)}{n^2 - E(X)^2}\right)}$$

Satz 1 besagt, daß die erwartete Anzahl von Bällen, für die alle Behälter einen Ball erhalten, etwa $n \cdot \ln n$ ist, da $H_n \approx \ln(n)$ und Satz 2, daß für eine große Anzahl n von Behältern die Wahrscheinlichkeit für eine größere relative Abweichung c von diesem Wert sehr klein wird. Die Wahrscheinlichkeit, um mehr als $5n$ nach oben vom Erwartungswert abzuweichen, ist (für hinreichend große n) bereits kleiner als 1%.

Satz 3 besagt, daß der Erwartungswert für die Anzahl leerer Behälter für eine Behälteranzahl n und eine Wurfanzahl m exponentiell mit m/n fällt und daß die Wahrscheinlichkeit für eine stärkere relative Abweichung $\lambda = c \cdot n$ schnell abfällt: Mit wachsendem n verstärkt sich der Abfall noch, da nach Einsetzen im Exponenten ein Term der Größenordnung $-c^2 \cdot n$ steht.

Auch dieser Satz zeigt, daß $\Theta(n \cdot \log n)$ Würfe benötigt werden, um eine konstante Anzahl leerer Behälter zu erreichen. Wenn man zuläßt, daß ein konstanter Anteil c von Behältern ungefüllt bleiben darf, daß also $c \cdot v$ Behälter nicht gefüllt sind, so erhält man als Bedingung für die Wurfanzahl m sogar:

$$ve^{-m/v} \le cv, \text{ bzw. } -m/v \le \ln c, \text{ also:}$$
$$m \ge -v \ln c$$

In dem Fall hängt die Anzahl der Würfe somit nur noch linear von der Behälteranzahl ab sowie logarithmisch vom zugelassenen relativen Anteil leerer Behälter.

Anwendung auf die pixelweise Bildrekonstruktion

Sei zunächst also $h = 0$ und vollständige Bedeckung der Pixel, d.h. $v = \textit{Anzahl Pixel im Bild,}$ angenommen. Für den Algorithmus zur pixelweisen Bildrekonstruktion entsprechen dann die Stichprobenpunkte den Bällen und die Pixelanzahl v den Behältern. Man erhält aus Satz 1 und Satz 2 und Lemma 1, daß die Stichprobe einen asymptotischen Umfang von $\Theta(v \cdot \log v)$ Punkten haben muß, um für ein Bild ohne Verdeckungen mit beliebig hoher, fester Wahrscheinlichkeit ein korrektes Bild zu produzieren. Die Wahrscheinlichkeit, für große Werte von v um mehr als $c \cdot v$ vom exakten Erwartungswert $v \cdot \ln v$ abzuweichen, fällt mit wachsendem c schnell gegen null ab. Wenn im Bild eine fester Anteil c von Pixeln nicht korrekt sein darf, so wächst der Aufwand mit $\Theta(v \cdot \log c)$.

Berücksichtigung von Verdeckungen

Nun zum Fall mit Verdeckungen ($h \neq 0$): Es sei vorausgesetzt, daß es sich um eine Ansicht der Szene mit getrennter Verdeckung (im Sinne von Definition 9) handelt und damit insbesondere, daß für alle Pixel die sichtbare projizierte Fläche gleich 1 ist. Wenn für jeden Pixel der darin sichtbare Teil der Oberfläche mindestens einen Stichprobenpunkt erhält, haben alle Stichprobenpunkte, die in verdeckte Flächenteile fallen, keinen Einfluß auf das Bild, da sie nach Lemma 1 beim z-Test herausfallen. Diejenigen Stichprobenpunkte, die in sichtbare Oberflächenteile fallen, werden gleichmäßig unter den Pixeln des Bildes verteilt, da angenommen wurde, daß alle Pixel vollständig mit sichtbarer Oberfläche abgedeckt sind. Zusätzlich fallen nun auch Stichprobenpunkte in nicht sichtbare Bereiche. Die nicht sichtbaren Bereiche können nun durch h weitere Behälter modelliert werden, in die Punkte mit gleicher Wahrscheinlichkeit wie in die v sichtbaren fallen, die aber keinen Einfluß auf das Bild haben. Dieses Modell gibt (bis auf die zu vernachlässigende Rundung auf eine ganzzahlige Behälteranzahl) genau die Wahrscheinlichkeit wieder, mit der die Stichprobenpunkte ausgewählt werden. Wenn alle v sichtbaren Behälter einen Stichprobenpunkt erhalten, entsteht nach Lemma 1 ein korrektes Bild.

Um nun garantieren zu können, daß die v Behälter, die für die sichtbaren Pixel stehen, mindestens einen Punkt erhalten, ist auf jeden Fall hinreichend, daß alle $a = v + h$ Behälter mindestens einen Punkt erhalten; man erhielte damit eine erwartete Stichprobengröße von $O(a \log a)$. Diese Schranke läßt sich allerdings noch leicht verbessern. Dies läßt sich zunächst an einer anschaulichen Überlegung verdeutlichen:

Aus Symmetriegründen ist zu erwarten, daß leer gebliebene Behälter sich unter den sichtbaren und den unsichtbaren im Mittel gemäß $v : h$ verteilen, also nur v/a leer gebliebene Behälter sichtbar werden. Wenn erwartete c sichtbare „Behälter“ ungefüllt bleiben dürfen, d.h. c Pixel zeigen potentiell ein falsches Bild, dürfen insgesamt $c \cdot (a/v)$ Behälter in der Stichprobe ungefüllt bleiben. Nach Satz 3 erhält man für den Stichprobenumfang m für erwartete c sichtbare Fehler:

$$ae^{-m/a} \le \frac{ca}{v}, \text{ bzw. } -m/a \le \ln\frac{c}{v}, \text{ oder äquivalent } m \ge -a\ln\frac{c}{v},$$
$$\text{was gegeben ist bei: } m \ge a(\ln v - \ln c) \in O(a \log v)$$

Dies gilt nur unter der Annahme, daß sich die Fehler gleichmäßig im Verhältnis $v:h$ verteilen, was zwar der erwartete Fall, aber nicht garantiert ist.

Sei daher nun allgemeiner angenommen, daß eine Stichprobengröße m_v notwendig ist, um alle v sichtbaren Pixel zu füllen. Nach den bisherigen Ergebnissen ist m_v mit hoher Sicherheit in $O(v \cdot \log v)$. Es wird eine Stichprobe der Größe m genommen, wobei Punkte mit Wahrscheinlichkeit $p_v := v/a$ in den sichtbaren und $p_h := h/a$ in den unsichtbaren Bereich fallen. Die Anzahl von Punkten im sichtbaren Bereich ist binomialverteilt mit den Parametern m und p_v. Im folgenden soll die Größe m der Stichprobe bestimmt werden, die notwendig ist, damit weiterhin mit beliebig vorgegebener Sicherheit s mindestens m_v Punkte in die v sichtbaren Behälter fallen, wenn nun auch auf unsichtbare Bereiche verteilt wird. Dazu muß gelten:

$$\sum_{k=m_v}^{m} \binom{m}{k} {p_v}^k {p_h}^{1-k} \ge s$$

Sei nun vorausgesetzt, daß m_v groß und h/a nicht zu groß ist, so daß $m \cdot p_v \cdot p_h \gg 9$ ist. Da die Bildauflösung in Anwendungen typischerweise im sechsstelligen Bereich angesiedelt ist und die verdeckte Fläche in sinnvollen Anwendungen nur äußerst selten um mehr als einen dreistelligen Faktor größer sein kann[10], ist dies keine starke Einschränkung. Allerdings kann h nicht beliebig wachsen, man erhält also keine asymptotische Abschätzung für festes v und beliebiges h. Unter den Voraussetzungen kann die Binomialverteilung durch die Normalverteilung angenähert und die Summe durch ein entsprechendes Integral ausgedrückt werden ([Pfanzagl 91], S. 175f). Sei im folgenden

$$\Phi(x) := \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{\tau^2}{2}} d\tau$$

das linksseitig unendliche Integral über die Dichtefunktion der Gaußschen Normalverteilung. Dann erhält man nach [Pfanzagl 91]:

$$\sum_{k=m_v}^{m} \binom{m}{k} {p_v}^k {p_h}^{1-k} \approx 1 - \Phi\left(\frac{m_v - mp_v}{\sqrt{mp_v p_h}}\right)$$

Dies muß mindestens die gegebene Sicherheit erreichen, d.h.:

$$1 - \Phi\left(\frac{m_v - mp_v}{\sqrt{mp_v p_h}}\right) \ge s, \text{ also } \Phi\left(\frac{m_v - mp_v}{\sqrt{mp_v p_h}}\right) \ge 1 - s,$$

und da Φ streng monoton wachsend ist, folgt dies aus $\frac{m_v - mp_v}{\sqrt{mp_v p_h}} \ge \Phi^{-1}(1-s) =: \varphi.$

Bezeichne nun die rechte Seite mit φ und erhalte:

$m_v - mp_v \ge \varphi\sqrt{mp_v p_h}$ bzw. $(m_v - mp_v)^2 \ge \varphi^2 mp_v p_h$, was gilt, wenn

[10] Da das Verfahren keine verdeckten Bildteile erkennt, wäre der Aufwand in einem solchen Fall unrealistisch hoch.

$$
\begin{aligned}
0 &\le {m_v}^2 - 2m_v m p_v + m^2 {p_v}^2 - \varphi^2 m p_v p_h \\
&= m^2 {p_v}^2 - 2m_v m p_v - \varphi^2 m p_v p_h + {m_v}^2, \text{ gleichbedeutend mit} \\
0 &\le m^2 - 2\frac{m_v m}{p_v} - \frac{\varphi^2 m p_h}{p_v} + \frac{{m_v}^2}{{p_v}^2} \\
&= m^2 - m\frac{2m_v + \varphi^2 p_h}{p_v} + \frac{{m_v}^2}{{p_v}^2} \\
&= \left(m - \frac{2m_v + \varphi^2 p_h}{2p_v} \right)^2 - \left(\frac{2m_v + \varphi^2 p_h}{2p_v} \right)^2 + \frac{{m_v}^2}{{p_v}^2}
\end{aligned}
$$

Dies ist äquivalent zu:

$$m \geq \frac{2m_v + \varphi^2 p_h}{2p_v} + \sqrt{\left(\frac{2m_v + \varphi^2 p_h}{2p_v}\right)^2 - \frac{{m_v}^2}{{p_v}^2}} \quad \vee \quad m \leq \frac{2m_v + \varphi^2 p_h}{2p_v} - \sqrt{\left(\frac{2m_v + \varphi^2 p_h}{2p_v}\right)^2 - \frac{{m_v}^2}{{p_v}^2}}$$

und dies ist auf jeden Fall erfüllt, wenn die erste Bedingung erfüllt ist, folgt also aus:

$$\begin{aligned}
m &\geq \frac{2m_v + \varphi^2 p_h}{2p_v} + \sqrt{\frac{1}{{p_v}^2}\left(\left(\frac{2m_v + \varphi^2 p_h}{2}\right)^2 - {m_v}^2\right)} \\
&= \frac{1}{p_v}\left(\frac{2m_v + \varphi^2 p_h}{2} + \sqrt{\left(\frac{2m_v + \varphi^2 p_h}{2}\right)^2 - {m_v}^2}\right) \\
&= \frac{1}{p_v}\left(m_v + \frac{\varphi^2 p_h}{2} + \sqrt{{m_v}^2 + \varphi^2 p_h m_v + \frac{\varphi^4 {p_h}^2}{4} - {m_v}^2}\right) \\
&= \frac{1}{p_v}\left(m_v + \frac{\varphi^2 p_h}{2} + \sqrt{\frac{\varphi^4 {p_h}^2}{4} + \varphi^2 p_h m_v}\right) \\
&= \frac{1}{p_v}\left(m_v + \frac{\varphi^2 p_h}{2} + \varphi\sqrt{\frac{\varphi^2 {p_h}^2}{4} + p_h m_v}\right)
\end{aligned}$$

Da $p_h < 1$ ist, läßt sich die Schranke abschätzen durch:

$$\begin{aligned}
&\frac{1}{p_v}\left(m_v + \frac{\varphi^2 p_h}{2} + \varphi\sqrt{\frac{\varphi^2 {p_h}^2}{4} + p_h m_v}\right) \\
&\leq \frac{1}{p_v}\left(m_v + \frac{\varphi^2}{2} + \varphi\sqrt{\frac{\varphi^2}{4} + m_v}\right) \\
&\approx \frac{1}{p_v}\left(m_v + \varphi\sqrt{m_v}\right)
\end{aligned}$$

Die Näherung in der letzten Zeile nutzt aus, daß $\varphi = \Phi^{-1}(1-s)$ sehr stark mit s fällt und daher für ausreichend kleine Wahrscheinlichkeiten gegenüber m_v vernachlässigt werden kann. So ist z.B. $\varphi \leq 2$ für $s = 97\%$ und $\varphi \leq 4$ für $s = 1\text{-}10^{-4}$ [Bronstein et al. 97].

Als Ergebnis läßt sich festhalten:

Lemma 4: *Verteilung auf zwei Sorten von Behältern*

Es seien zwei unterscheidbare Mengen V und H von Behältern gegeben, es seien $v := \#V$, $h := \#H$ und $a := (\#H + \#V)$. Um durch unabhängig gleichverteilte Würfe mindestens m_v Bälle mit hoher Sicherheitswahrscheinlichkeit in den Behältern vom Typ V zu plazieren, sind ungefähr

$$m \geq \frac{a}{v}\left(m_v + O\left(\sqrt{m_v}\right)\right)$$

Würfe in alle Behälter hinreichend. Die Näherung ist genau, wenn m_v groß ist (etwa fünfstellig), $\frac{vh}{a^2} m_v$ nicht zu klein ist (d.h. im zweistelligen Bereich) und die Sicherheitswahrscheinlichkeit nicht extrem groß ist ($1\text{-}10^{-4}$ ist unproblematisch).

Setzt man darin nun $m_v := O(v \cdot \ln v)$, so erhält man einen Gesamtaufwand von

$$m \in O\left(\frac{a}{v}\left(v \ln v + \sqrt{v \ln v}\right)\right) = O\left(a \ln v + \frac{a\sqrt{\ln v}}{\sqrt{v}}\right) = O(a \ln v),$$

wie das zuvor genannte Argument der „gleichmäßigen Verteilung" der Fehler bereits nahe gelegt hat. Analog erhält man $O(a \cdot \log c)$, wenn ein Anteil c von falschen Pixeln erlaubt ist.

Ergebnis

Die Ergebnisse können nun auf die pixelweise Rekonstruktion angewendet werden. Zusammenfassend erhält man:

Satz 4: *Stichprobenumfang*

Sei eine Ansicht einer Szene mit getrennter Verdeckungen gegeben. Sei v die sichtbare, h die verdeckte und a die gesamte projizierte Fläche in Pixeln. Sei weiter $g := 1/a$.

Um mit Sicherheit s keine falschen Pixel zu erhalten sind nicht mehr als

$$\frac{\log\left(1 - s^g\right)}{\log(1 - g)} \overset{a \text{ groß}}{\approx} a \log\left(1 - s^{1/a}\right)$$

Stichprobenpunkte notwendig.

Der Erwartungswert für die Größe einer Stichprobe, für die Algorithmus 2 ein korrektes Bild liefert, ist höchstens $a \cdot H_a \in \Theta(a \log a)$. Für große Werte von a ist die Wahrscheinlichkeit für ein $c > 0$ um mehr als $c \cdot a$ von diesem Wert nach oben abzuweichen $\leq 1 - e^{-e^{-c}}$.

Satz 5: *Schärfere Schätzung des Stichprobenumfangs*

Es sei dieselbe Situation wie in Satz 4 gegeben. Zusätzlich gelte: v ist groß und h weicht nicht um einen extremen Faktor von v ab, so daß $(v \cdot h / a^2) \cdot m_v$ nicht zu klein wird (d.h., im zweistelligen Bereich bleibt). Dann wächst der erwartete Stichprobenumfang, der für ein korrektes Bild notwendig ist, bei wachsendem a höchstens mit $O(a \cdot \log v)$.

Wenn zugelassen wird, daß $c \cdot v$ der sichtbaren Pixel kein korrektes Bild zeigen, so wächst der Aufwand mit $O(a \cdot \log c)$.

Um mit Sicherheit s keine falschen Pixel zu erhalten sind ungefähr

$$m \geq \frac{a}{v}\left(m_v + \Phi^{-1}(1-\bar{s})\sqrt{m_v}\right) \text{ Stichprobenpunkte}$$

$$\text{mit } m_v := \frac{\log\left(1-\bar{s}^{1/v}\right)}{\log\left(1-\frac{1}{v}\right)} \overset{v \text{ groß}}{\approx} v\log\left(1-\bar{s}^{1/v}\right)$$

$$\text{und } \bar{s} = \sqrt{s}$$

hinreichend. Dies gilt nur, wenn s nicht extrem klein gewählt wird (99% Sicherheit ist noch kein Problem).

Die letzte Aussage erhält man, indem man ein zweistufiges Zufallsexperiment betrachtet, indem mit Wahrscheinlichkeit $s^{1/2}$ zuerst genügend Punkte in den sichtbaren Bereich fallen (Lemma 4) und dann unabhängig davon dadurch wiederum mit Wahrscheinlichkeit $s^{1/2}$ alle v Pixel bedeckt werden. Im Produkt ist die Wahrscheinlichkeit dann s, daß beide Experimente erfolgreich sind.

Satz 5 liefert das wesentliche Ergebnis dieses Abschnittes: Die erwartete Stichprobengröße liegt für jede gegebene Sicherheit in $O(a \cdot \log v)$. Dies läßt sich noch etwas anders formulieren: Wenn man mit $H = a/v$ das Verhältnis der gesamten zur sichtbaren projizierten Fläche bezeichnet (beim z-Buffer-Rendering oft als „Tiefenkomplexität" bezeichnet), erhält man einen erwarteten Stichprobenumfang von $O(Hv \cdot \log v)$. Nimmt man nun noch an, daß das gesamte Bild mit sichtbarer Fläche vollständig überdeckt ist, so gibt v die Auflösung in Pixeln an. Der Aufwand wächst dann also (in den beschriebenen Grenzen für H) linear in der Tiefenkomplexität und in $\Theta(v \cdot \log v)$ mit der Auflösung.

Beispiele

Tabelle 1 vermittelt einen Eindruck von der Größenordnung der Stichprobe bei typischen Werten für die projizierte Fläche. Die Werte wurden, wie auch in den folgenden Beispielen, nach Lemma 3 und für $h = 0$ berechnet. Für Szenen mit Verdeckung skalieren die Werte nach Satz 5 annähernd linear. Abbildung 19 zeigt eine grafische Darstellung.

V	$s = 0{,}05$	$s = 0{,}5$	$s = 0{,}95$
320×240	779.652	892.063	1.092.024
640×480	3.544.489	3.994.137	4.793.986
800×600	5.752.485	6.455.060	7.704.826
1024×786	9.813.151	10.964.251	13.011.868

Tabelle 1: Stichprobenumfang bei verschiedenen Auflösungen, $h = 0$.

Abbildung 20 zeigt die Abhängigkeit von der gewünschten Sicherheit bei fester projizierter Fläche. Wie man sieht hat die der Parameter s bei einer Auflösung in für die Anwendung typischer Größenordnung nur einen schwachen Einfluß auf die Stichprobengröße.

Abbildung 21 zeigt den superlinearen Anteil (*Stichprobenumfang / projizierte Fläche*) am Wachstum des Stichprobenumfangs. Dieser Wert gibt auch das „Überfüllungsverhältnis" an, d.h. die Anzahl von Stichprobenpunkten, die im Durchschnitt pro Pixel verwendet werden. Wie man sieht, wächst dieses im dargestellten Bereich logarithmisch mit der projizierten Fläche.

Die absoluten Werte in Tabelle 1 und die Werte für das Überfüllungsverhältnis zeigen bereits eine Einschränkung für die Performance des randomisierten z-Buffer Verfahrens: Da für jeden Stichprobenpunkt eine ähnliche Transformation wie für einen Dreieckseckpunkt beim gewöhnlichen z-Buffer notwendig ist, kann sich das Verfahren, zumindest mit pixelweiser Bildrekonstruktion, nur für Szenen lohnen, deren geometrische Komplexität deutlich über der Größenordnung von einer Million Punkten liegt. Schuld daran ist die Tatsache, daß etwa $\ln v$ mehr Stichprobenpunkte als Pixel v benötigt werden, um alle Pixel aufzufüllen. Daher sind Methoden interessant, die bereits mit einer geringeren Anzahl von Stichprobenpunkten ein brauchbares Bild in einer ausreichenden Auflösung liefern. In den folgenden Abschnitten werden einige Möglichkeiten beschrieben.

Ein naheliegender Ansatz wäre, nicht alle Pixel aufzufüllen, sondern ein gewissen Anteil c an falschen Pixeln zu zulassen. Wie bereits gesehen, liefert dies asymptotisch eine Verbesserung der Laufzeit von $O(v \cdot \log v)$ auf $O(v \cdot \log c)$. Der Toleranzanteil muß allerdings exponentiell erhöht werden, um bei einer festen Auflösung den Aufwand linear zu verringern. Abbildung 22 zeigt den möglichen Gewinn durch erhöhen der Toleranz für typische Auflösungen: Selbst wenn in Kauf genommen wird, daß mit 50% Wahrscheinlichkeit bis zu 50% der Pixel nicht korrekt sind (was zu einem unbrauchbaren Bild führen würde), läßt sich der Stichprobenumfang nur um 5% reduzieren. Folglich ist durch reines Erhöhen des Toleranzanteils keine signifikante Verkleinerung des Stichprobenumfangs zu erzielen.

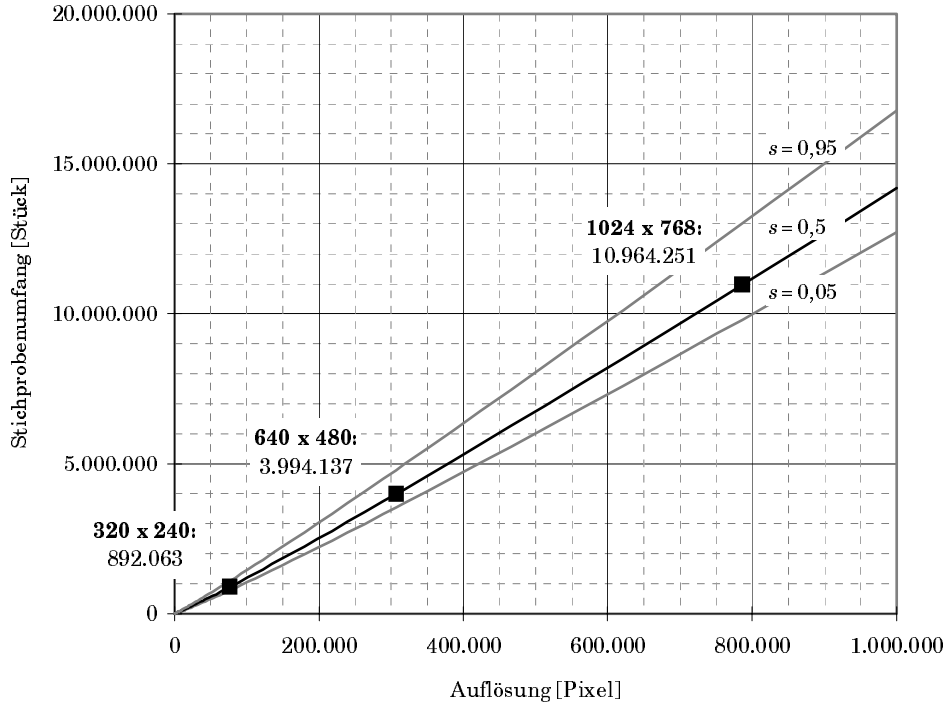


Abbildung 19: Stichprobenumfang bei verschiedenen Auflösungen

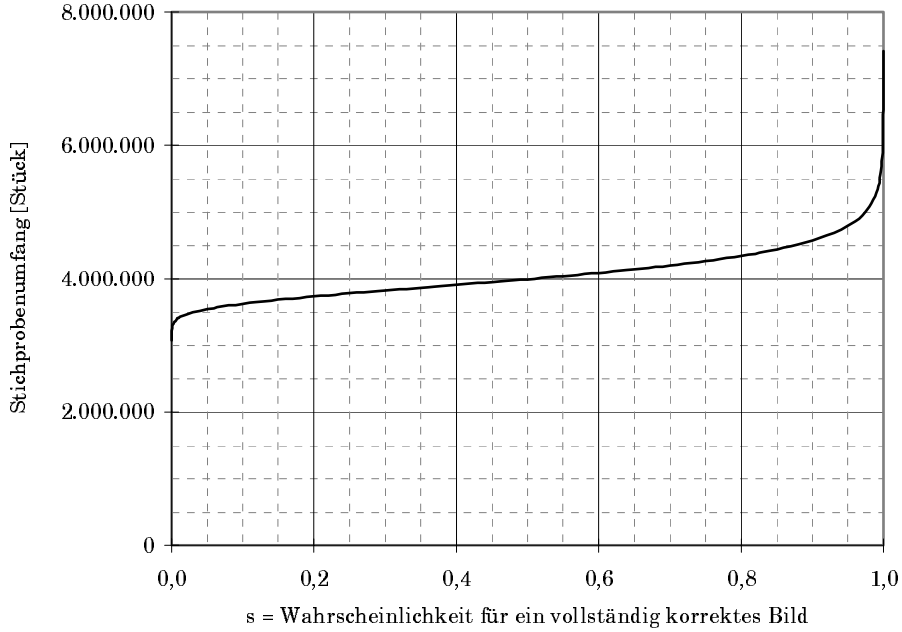


Abbildung 20: Stichprobenumfang in Abhängigkeit von s ($a = 640 \times 480$)

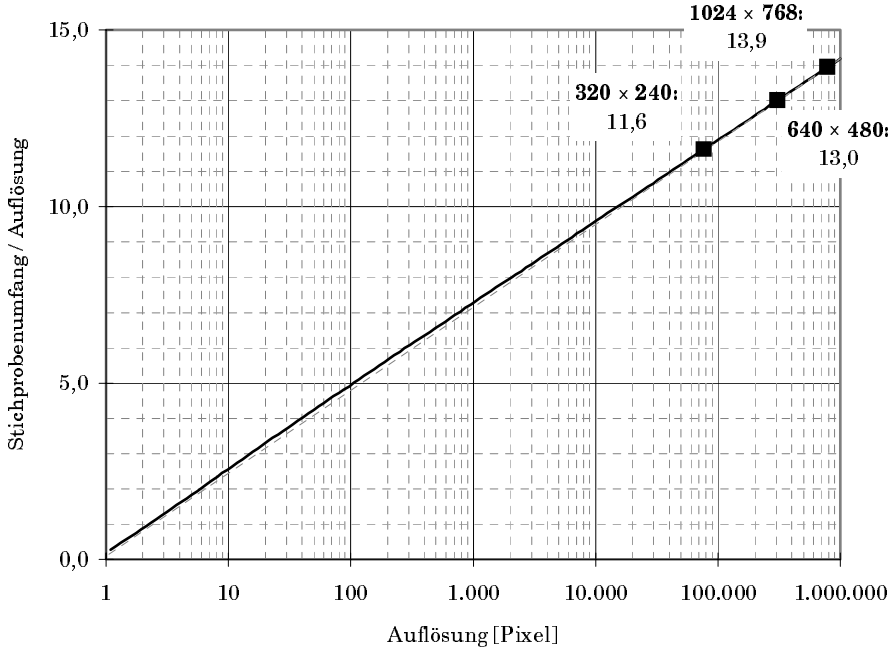


Abbildung 21: Überfüllungsverhältnis bei verschiedenen Auflösungen, $s = 0{,}5$ (gestrichelt: Gerade durch Ursprung und letzten Punkt)

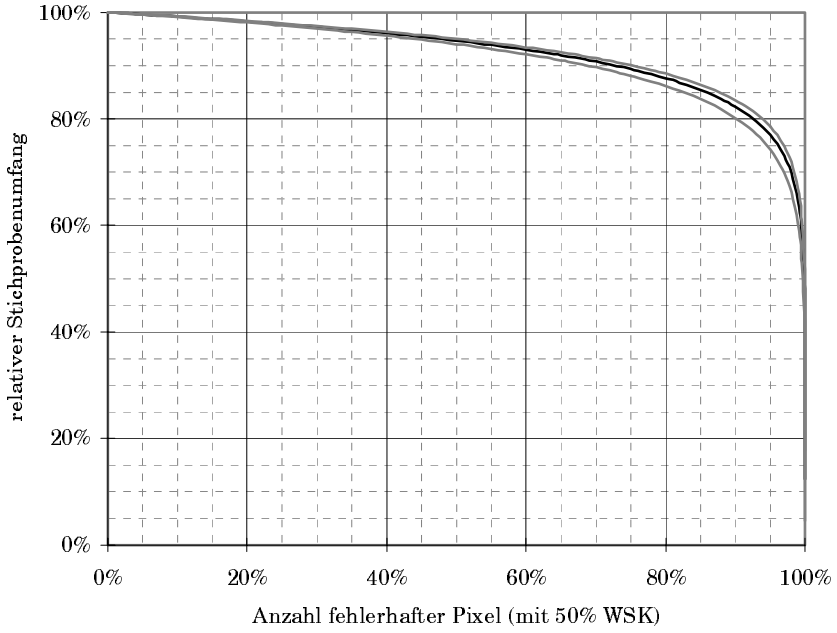


Abbildung 22: Verkleinerung des Stichprobenumfangs bei Erhöhung der Anzahl fehlerhafter Pixel für $s = 0.5$, $a = 640 \times 480$ (mittlere Kurve), $a = 320 \times 240$ (untere Kurve), $a = 1024 \times 768$ (obere Kurve)

3.2.1.3 Erwartungswert des Bildes

Ein Vorteil der Konstruktion eines Bildes aus einer zufälligen Stichprobe ist, daß ein Teil der Aliasingartefakte, die bei regelmäßiger Abtastung entstehen, in Form von Rauschen auftreten, das in der Regel als weniger störend wahrgenommen wird. Falls das Rauschen für die Anwendung störend ist, kann es durch Mittelwertbildung über mehrere Bilder vermindert werden, was allerdings entsprechend zusätzliche Rechenzeit kostet. Es stellt sich damit die Frage, wie schnell das Rauschen verschwindet und welches Bild im Mittel zu erwarten ist.

Betrachte nun einen Pixel in einem Graustufenbild (für Farbbilder kann alles auf jeden der drei Farbkanäle unabhängig voneinander angewendet werden). Der Einfachheit halber sei weiter angenommen, daß sich das Bild unter dem Pixel in n Regionen zerlegen läßt, die eine konstante Helligkeit I_i, $i \in \{1, ..., n\}$ haben. Da in der Computergrafik ohnehin mit beschränkter Farbauflösung gearbeitet wird, ist dies keine echte Einschränkung. Für jede dieser Regionen besteht eine feste Wahrscheinlichkeit p_i, $i \in \{1, ..., n\}$, daß die Helligkeit dieser Region für die des Pixels ausgewählt wird. Diese Wahrscheinlichkeit hängt von der Tiefe der Region und dem Stichprobenumfang ab. Daneben sei q_i, $i \in \{1, ..., n\}$ die sichtbare projizierte Fläche der Region.

Der Erwartungswert für die Helligkeit des Pixels ist $\mu_P = \sum_{i=1}^{n} p_i I_i$ und die Varianz ist $\sigma_P^2 = \sum_{i=1}^{n} p_i \left(I_i - \mu\right)^2 \leq \sum_{i=1}^{n} p_i \cdot 1 \leq 1$. Das Mittelwertbild (Definition 4) ist in dem Pixel gleich $\mu_k = \sum_{i=1}^{n} q_i I_i$.

Es stellen sich nun zwei Fragen: Inwieweit entsprechen sich der Erwartungswert des Renderings und das Mittelwertbild, und wie kann der Erwartungswert bestimmt werden?

Näherung des Erwartungswertes

Nach dem zentralen Grenzwertsatz ([Pfanzagl 91], S. 165ff) konvergiert das arithmetische Mittel von m Zufallsvariablen, die unabhängig und gleich verteilt sind, mit (endlichem) Erwartungswert μ und (endlicher) Standardabweichung σ, gegen eine normalverteilte Zufallsvariable mit Erwartungswert μ und Standardabweichung $\sigma/\sqrt{m}$. Da die Helligkeiten der Pixel bei mehreren Renderingläufen unabhängig voneinander gewählt werden und für jedes feste Pixel jeweils die gleiche Verteilung besitzen und weil Erwartungswert und Varianz der Helligkeiten endlich sind, liefert der Satz, daß das arithmetische Mittel eines festen Pixels über m Renderingläufe gegen eine normalverteilte Zufallsvariable mit Erwartungswert μ_P und Varianz $\sigma_P/\sqrt{m} \leq 1/\sqrt{m}$ konvergiert.

Für die Bilderzeugung bedeutet das, daß das Rauschen im Bild durch Mittelung über mehrere Bilder reduziert werden kann. Dabei nimmt das Rauschen ab einer gewissen Anzahl von Bilder (sobald genügend Konvergenz gegen die Normalverteilung eintritt) mit hoher Wahrscheinlichkeit mit $\Theta(\sigma_P / \sqrt{m}) \subseteq O(1/\sqrt{m})$ ab. Zum einen bedeutet dies, daß die Intensität des Rauschens im Mittelwertbild proportional zur ursprünglichen Standardabweichung der Helligkeit eines Pixels ist, d.h. daß in Bereichen, in denen die Farbe kaum variiert nur wenig Rauschen zu erwarten ist, während an scharfen Farbkanten stärkeres Rauschen auftritt. Zum anderen zeigt das, daß ein quadratischer Aufwand erforderlich ist, um das Rauschen linear um einen Faktor zu reduzieren.

Das Rauschen zu verringern, ist besonders für Walkthroughanimation wichtig: wenn sich der Betrachter längere Zeit nicht oder nur wenig bewegt, kann ein ständiges Rauschen in der Animation auf den menschlichen Beobachter sehr störend wirken. Da aber eine hinreichende Reduzierung des Rauschens durch das Bilden des arithmetischen Mittelwertes aus vielen (etwa mehr als 10) Bildern nicht in Echtzeit machbar ist, kann eine andere Strategie angewendet werden. Jedes neue Bild wird in einem festen Anteil ε mit dem bisherigen Bild gemischt („*exponential smoothing*“). In Experimenten zeigte sich, daß für einen Wert von ε in der Größenordnung von 0.1 die Differenz zwischen den Animationsphasen bereits so stark geschwächt wird, daß das Rauschen nicht mehr stark auffällt. Die Konvergenz gegen den Erwartungswert geht natürlich dadurch langsamer vonstatten und nicht über den Effekt des Mittels aus $1/\varepsilon$ Bilder hinaus. Wenn sich die Ansicht der Szene ändert, tritt nun eine Bewegungsunschärfe auf, die mit $1/\varepsilon$ und mit der Änderungsgeschwindigkeit wächst. Es wäre denkbar, den Wert von ε mit wachsender Geschwindigkeit zu erhöhen, um diesen Effekt zu verringern. Die Idee dabei wäre, daß Rauschen in einem stark bewegten Bild weniger auffällt, in Standbildern und nahezu-Standbildern würde es dagegen wirksam gedämpft, ohne allzu starke Artefakte. Diese Koppelung wurde hier allerdings nicht experimentell untersucht.

Zusammenhang mit dem Mittelwertbild

Wenn man annimmt, daß alle Punkte der Szene in einer Ebene senkrecht zur Blickrichtung liegen, d.h., daß die Tiefenwerte aller Punkte gleich groß sind, so ist die Wahrscheinlichkeit für alle Punkte innerhalb eines Pixels gleich groß, die Farbe des Pixels zu stellen. In Algorithmus 2 wird dann stets der erste Stichprobenpunkt in jedem Pixel benutzt und alle weiteren werden ignoriert. Dies bedeutet, daß für alle $i \in \{1, ..., n\}$ $p_i = q_i$ ist und das somit der Erwartungswert des Bildes gleich dem Mittelwertbild ist (also dem Bild, das sich als arithmetisches Mittel über alle sichtbaren Objektfragmente jedes jeweiligen Pixels ergibt).

Sobald sich innerhalb eines Pixels jedoch Objektfragmente unterschiedlicher Tiefe befinden, kann diese Aussage nicht mehr garantiert werden. Das Problem liegt darin, daß nach Abschnitt 3.2.1.2 im Mittel $\ln k$ Stichprobenpunkte aus sichtbaren Bereichen in jeden der k Pixel des Bildes

fallen. Übernommen wird allerdings nur derjenige Punkt, dessen Tiefe minimal ist. Dieser Effekt, im folgenden als *„z-Dominanz"* bezeichnet, führt dazu, daß Objektfragmente mit geringerer Tiefe bevorzugt werden, der Erwartungswert weicht vom Mittelwertbild ab. Der Effekt tritt besonders dann störend in Erscheinung, wenn ein Pixel nur teilweise von kleinen Objektdetails bedeckt ist und dahinter ein Hintergrund durchscheinen soll. Die durch das randomisierte z-Buffer Verfahren mit Mittelwertbildung berechnete Transparenz liegt deutlich zu Ungunsten des Hintergrundes. Auch im ungemittelten Bild, in dem die Transparenz durch ein zufälliges Muster aus Hintergrund und Vordergrundpixeln erscheint, sind die Hintergrundpixel deutlich unterrepräsentiert.

Abbildung 23 zeigt das Phänomen an einem künstlichen Beispiel: Der Vordergrund besteht aus 32 schwarzen Streifen, zwischen denen genau soviel Platz ist, wie diese breit sind. Der Hintergrund ist weiß. Wenn diese Szene aus großer Distanz betrachtet wird, sollte daher im Mittelwert (zweite Zeile) ein 50%tiger Grauton entstehen. In der Tat erhält man allerdings ein sehr tiefes, fast schwarzes grau.

In Abbildung 24 und Abbildung 25 wurde die gleiche Beispielszene verwendet, nur daß hier die Lücken zwischen den schwarzen Streifen mit weißen Streifen gefüllt wurden. In Abbildung 25 wurde das Testobjekt im Gegensatz zu den beiden anderen aus 128 Streifen zusammengesetzt.

In Abbildung 24 wurde eine Perspektive gewählt, in der das Testobjekt in unterschiedlicher Tiefe erscheint. In den hinteren Teilen treten dabei beim normalen z-Buffer starke Aliasingartefakte auf (Bild a)). Dieser Effekt wird, wie die Bilder b) und c) zeigen, beim randomisierten z-Buffer nicht wesentlich vermindert. Die Interferenzmuster entstehen hier durch die systematische Bevorzugung der vordersten Tiefe in jedem Pixel. Durch die starke mittlere Überfüllung der Pixel tritt in der randomisierten Version der Effekt fast genauso stark auf. Bild d) zeigt eine Näherung des Mittelwertbildes nach Definition 4. Es wurde erstellt, indem in einem in beiden Dimensionen zehnmal so hoch aufgelösten Bild das arithmetische Mittel aus Blöcken von jeweils 10×10 Pixeln gebildet wurde. Man sieht deutlich, daß das Mittelwertbild immer noch störende Aliasingartefakte aufweist, wenn auch abgeschwächt. Zum Vergleich sieht man in Bild f) das Ergebnis, wenn anstelle des arithmetischen Mittelwertes eine gewichtete Mittelung mit einem Gaußschen Tiefpaßfilter [Glassner 95a] verwendet wird. Die Filterbreite wurde nach optischem Eindruck gewählt, theoretisch können auf Kosten der Detailschärfe die Aliasingartefakte beliebig stark abgeschwächt werden. Sowohl Bild d) wie Bild f) wurden mit einem Bildverarbeitungsprogramm aus einem hochauflösenden z-Buffer Rendering erstellt.

Um die Erklärung zu untermauern, daß das Phänomen der z-Dominanz für die starke Abweichung vom Mittelwertbild verantwortlich ist, wurde in Bild e) der Mittelwert aus einem sehr kleinen Stichprobenumfang bestimmt: Die Anzahl von Stichproben wurde so weit verringert, daß nur selten zwei Stichprobenpunkte auf einen Pixel fallen. Da dies dazu führt, daß der 50% graue Hintergrund stark durchscheint, mußte der Kontrast des entstandenen Mittelwertbildes etwa 10fach angehoben werden. Da dies ebenfalls den relativen Fehler entsprechend verstärkt, war eine große Zahl von Bildern für die Mittelwertbildung notwendig. Man beachte, daß ein solches Vorgehen für allgemeine Szenen nicht möglich ist, da Verdeckungen dabei falsch dargestellt werden. Nur durch die passend gewählte Hintergrundfarbe ließ sich in diesem Fall der Effekt kompensieren. Das Ergebnis entspricht bis auf ein Restrauschen dem korrekten Mittelwertbild.

Abbildung 25 zeigt eine Ansicht senkrecht auf das Testobjekt. Auch hier liefert das z-Buffer Verfahren starke Aliasingartefakte: Es erscheinen nur 9 „Pseudostreifen" statt der tatsächlichen 128 (Bild (a)). In Bild (b) sieht man das Ergebnis mit dem randomisierten z-Buffer: Die Fläche erscheint in einem gleichmäßigen schwarzweiß Rauschen. Der Mittelwert aus vielen Bildern

(Bild (c)) beweist, daß der Erwartungswert dieses verrauschten Bildes gut dem Mittelwertbild entspricht.

Die Bedeutung der z-Dominanz für die Bildqualität läßt sich in Farbbild 10 (a) und (b) (abgebildet in Anhang C) nochmals besonders deutlich erkennen: Bild (a) zeigt eine z-Buffer Darstellung, Bild (b) eine randomisierte z-Buffer Darstellung einer Schachbrettszene. Im mittleren Tiefenbereich zeigen beide Verfahren ähnlich starke Aliasingartefakte, da hier die starke Überfüllung der Pixel beim randomisierten Verfahren dazu führt, daß dieses sich ähnlich wie das konventionelle Verfahren verhält: Da auf fast alle unter dem Pixel befindlichen Dreiecksfragmente ein Stichprobenpunkt entfällt, setzt sich auch im randomisierten Fall der vorderste Punkt durch. Mit zunehmendem Abstand wird die Anzahl von einem Pixel bedeckter Dreiecke allerdings immer größer und die Auswahl zwischen schwarzen und weißen Oberflächenpunkten zunehmend zufälliger, so daß im hinteren Tiefenbereich nur noch ein gleichmäßiges Rauschen anstelle der systematischen Aliasingartefakte entsteht, während das gewöhnliche z-Buffer Verfahren weiterhin systematische Artefakte produziert.

An den Beispielen sieht man, daß die z-Dominanz nicht nur dazu führt, daß teilweise bedeckte Flächen mit falscher Transparenz dargestellt werden, sondern auch die Hauptursache für die Anfälligkeit des randomisierten z-Buffer Verfahrens für Aliasing ist. Dies motiviert eine andere Rekonstruktion des Bildes aus der Stichprobe, wie sie im Abschnitt 3.2.3 beschrieben wird. Das Hauptproblem besteht dann darin, ein Kriterium für die Erkennung von Verdeckungen zu finden, welches bei nahe aneinanderliegenden z-Werten die vordersten nicht in der bisherigen Form systematisch bevorzugt.

Als Ergebnis läßt sich festhalten, daß das randomisierte z-Buffer Verfahren mit pixelweiser Rekonstruktion aufgrund des Effektes der z-Dominanz ähnlich empfindlich für Aliasing ist wie das gewöhnliche z-Buffer Verfahren, wenn die systematischen Details der Szene als Frequenz einen kleinen Bruchteil der Pixelfrequenz aufweisen. Nur in Fällen, bei denen alle Objektpunkte die gleiche Tiefe haben oder die Details der Szene hochfrequente Muster aufweisen, erhält man ein verbessertes Bild. Der Erwartungswert entspricht dem Mittelwertbild (welches aber seinerseits im allgemeinen nicht frei von Aliasing ist).

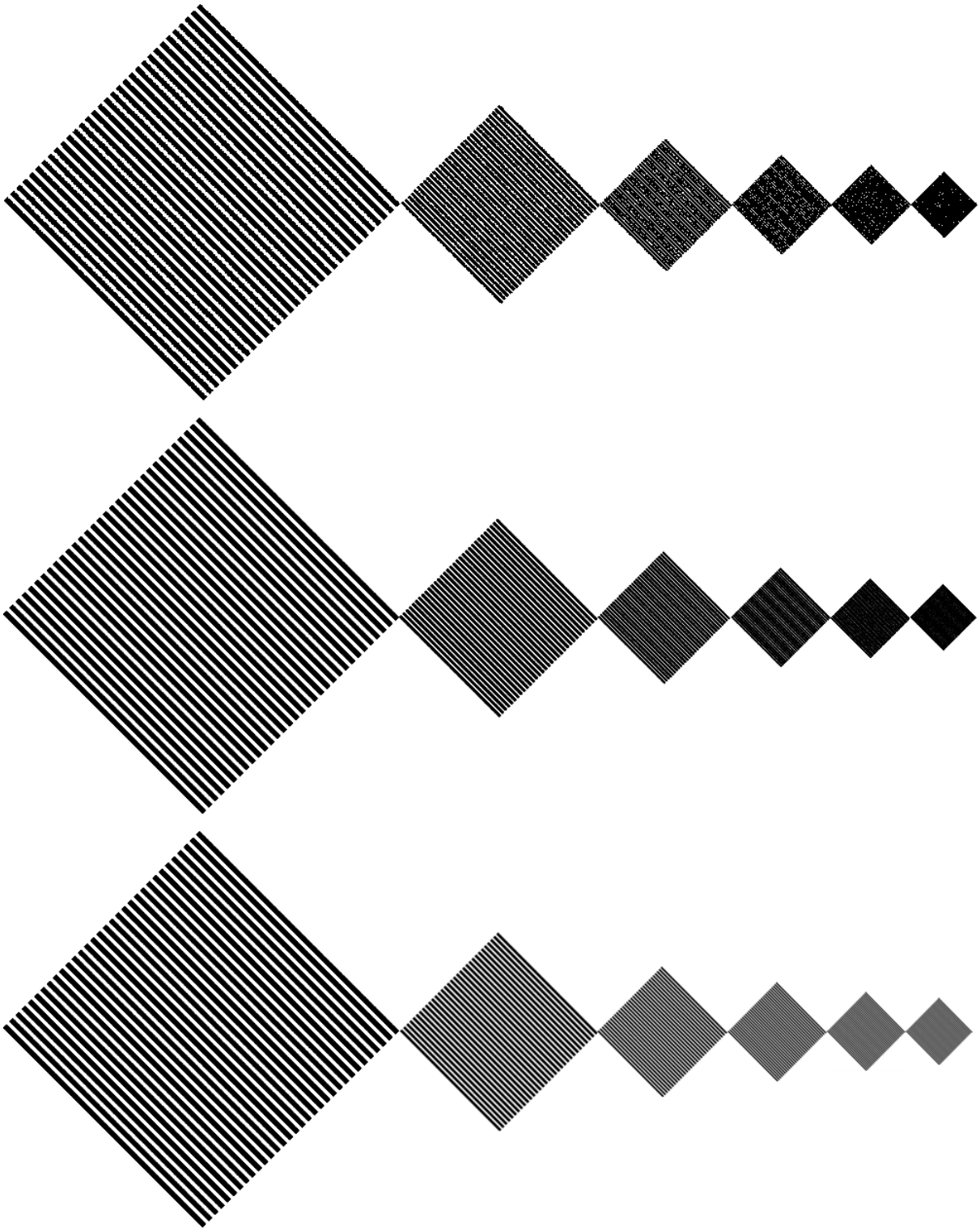
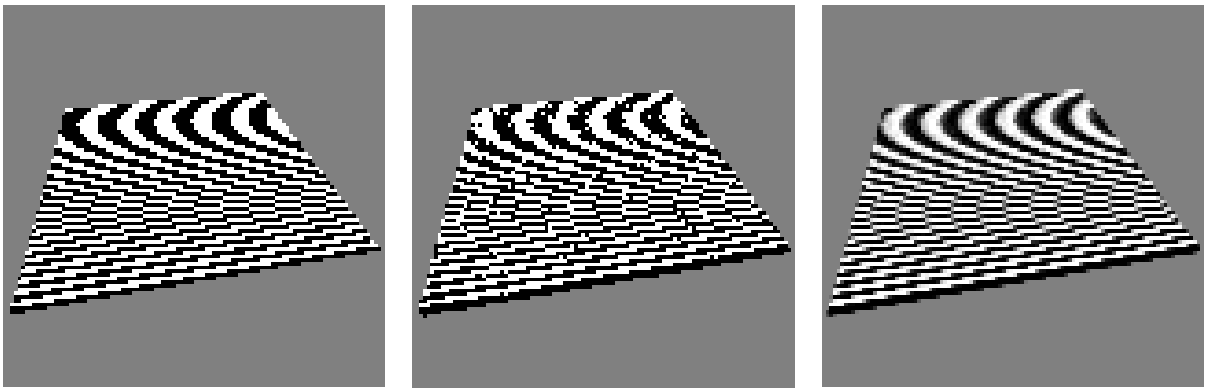
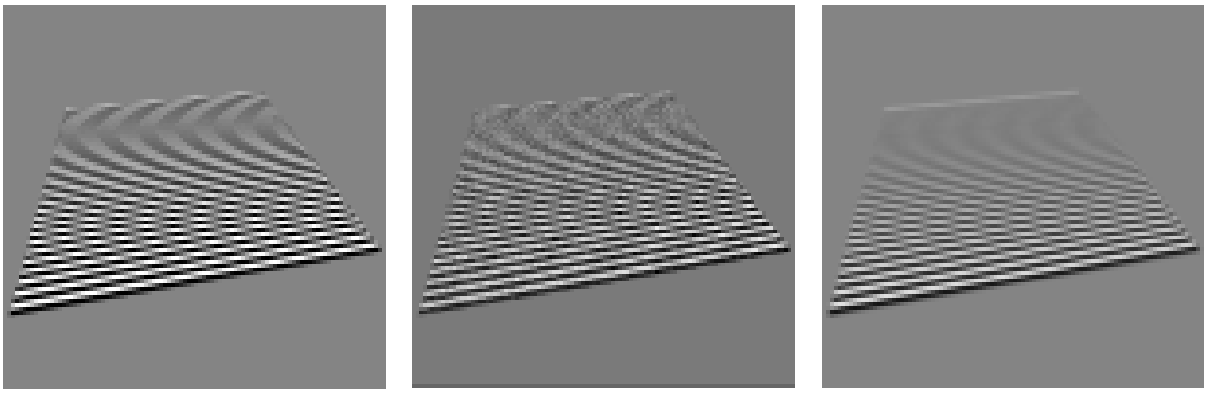


Abbildung 23: z-Dominanz: ein Quadrat aus schwarzen Streifen vor weißem Hintergrund in wachsendem Abstand. erste Zeile: randomisierter z-Buffer, zweite Zeile: Mittelwert über 16 Bilder, dritte Zeile: Darstellung in hoher Qualität zum Vergleich (mit Bildverarbeitungssoftware erstellt)

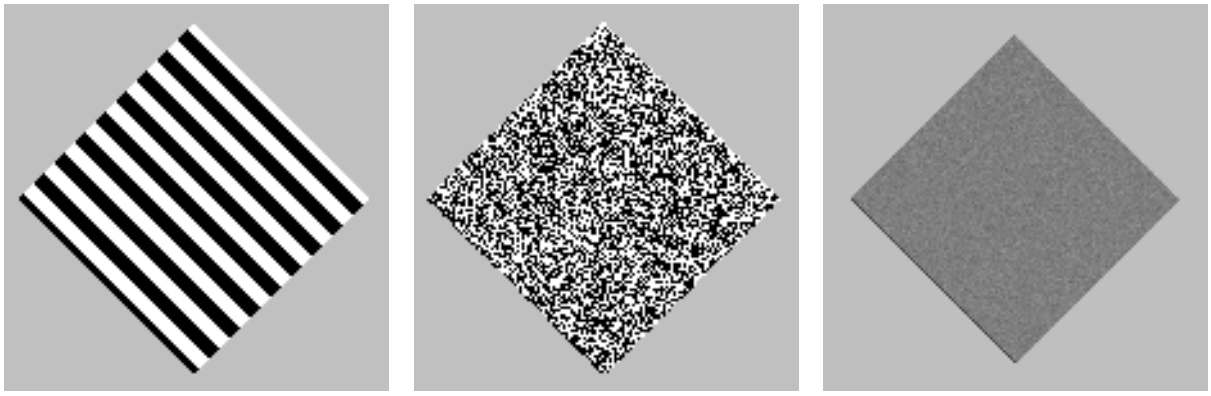


(a) z-Buffer **(b) randomisierter z-Buffer** **(c) randomisierter z-Buffer, arithm. Mittel aus 32 Bilder**



(d) Mittelwertbild (10×10 supersampling) **(e) 1/32 Stichprobenumfang, Mittelwert über 1000 Bilder, kontrastverstärkt** **(f) Gaußscher Tiefpaß nach 10×10 supersampling**

Abbildung 24: Aliasing Artefakte beim randomisierten z-Buffer durch unterschiedliche Tiefe



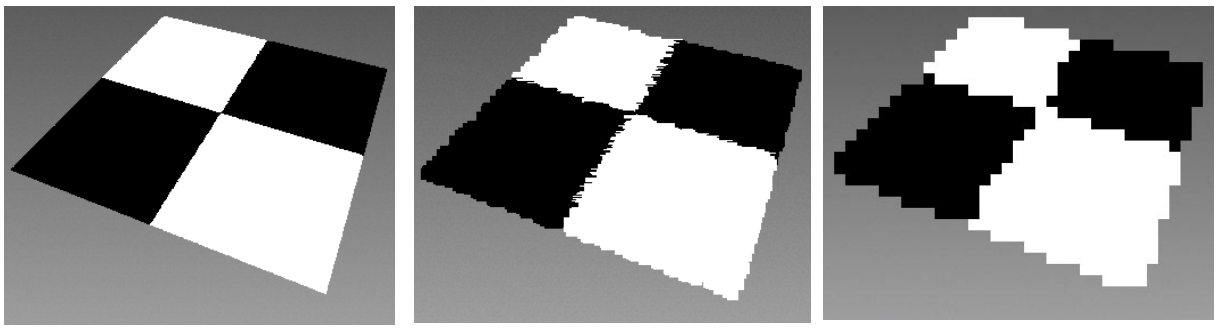
a) z-Buffer (Szene aus 128 Streifen!) **b) randomisierter z-Buffer** **c) randomisierter z-Buffer, arithm. Mittel aus 256 Bilder**

Abbildung 25: Aliasing bei gleicher Tiefe der Objektpunkte

3.2.2 Große Farbklötze

Um ein hochauflösendes Bild durch pixelweise Rekonstruktion zu erzeugen, ist ein sehr großer Stichprobenumfang notwendig. Für jeden Stichprobenpunkt ist eine Transformation in die Bildebene notwendig, die insbesondere eine 4×4-Matrix/Vektormultiplikation mit Fließkommazahlen erfordert. Zusätzlich sind je nach Schattierungsfunktion auch noch Beleuchtungsberechnungen notwendig, die relativ aufwendig sein können. Heutige Hardware kann diese Operation unter praktischen Bedingungen etwa 100.000 bis 1.000.000 mal pro Sekunde ausführen, wobei die größeren Werte mit Hardwareunterstützung erreicht werden (siehe Anhang A). Damit ist die Erzeugung eines Bildes in üblicher Auflösung kaum in Echtzeit möglich.

Die einfachste Möglichkeit, diese teuren, weil fließkommaintensiven Operationen einzusparen, besteht darin, an die Stelle jedes Stichprobenpunktes statt eines einzelnen Pixels einen größeren quadratischen Klotz in der entsprechenden Farbe zu zeichnen. D.h. in jedem Pixel des Klotzes wird ein z-Test durchgeführt und dann der Tiefenwert und die Farbe des Mittelpunktes des Klotzes eingetragen, falls der vorhandene Tiefenwert größer ist. Dadurch werden mehrere Bildpunkte auf einmal abgedeckt und der Stichprobenumfang kann kleiner ausfallen. Ein ähnlicher Effekt wäre zu erreichen, indem man einfach die Auflösung um den entsprechenden Wert verringert. Da die Klötze aber mit Pixelgenauigkeit positioniert werden, werden die Kanten und die groben Konturen der Objekte durch die Farbklotzmethode wesentlich besser wiedergegeben. Abbildung 26 zeigt dieses Phänomen an einem einfachen Beispiel.



pixelweise Rekonstruktion (740.000 Punkte) | **10 × 10 Klötze (5.500 Punkte)** | **1/10 × 1/10 der Auflösung (5.500 Punkte)**

Abbildung 26: pixelweise Bildrekonstruktion und Rekonstruktion mit großen Farbklötzen

Stichprobenumfang

Ein korrektes Bild im Sinne von Definition 3 läßt sich mit dieser Rekonstruktionsmethode nur in Spezialfällen erzielen, da jeder Farbklotz mehrere Pixel gleichzeitig beeinflußt. Daher wird im folgenden ein schwächerer Korrektheitsbegriff verwendet:

> **Definition 10:** *schwache Verdeckungserkennung*
>
> Sei eine Szene S und eine Ansicht von S gegeben. Die Menge aller Pixel im Bild sei in Regionen C_i, $i = 1...k$, partitioniert. Weiter seien Teilmengen T_j, $j = 1...l$, von S gegeben, die jeweils einen Bildbereich C_{i_j} für $i_j \in \{1...k\}$ vollständig so überdecken, daß kein Punkt der Szene im Bereich von C_{i_j}, der nicht in T_j ist, eine geringere Tiefe hat als ein Punkt aus T_j.

Ein Renderingverfahren *erkennt schwach die Verdeckungen bezüglich der Regionen* C_i, wenn es für jede Szene S und Teilmengen T_j daraus, die die obigen Bedingungen erfüllen, in jeder Region C_{i_j} nur Punkte aus T_j im Bild darstellt.

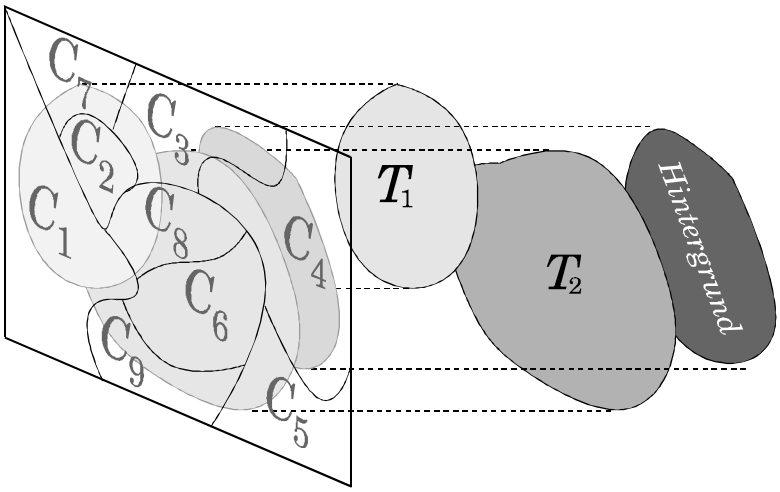


Abbildung 27: Einteilung des Bildes in Bereiche: T_1 überdeckt C_2 und T_2 überdeckt C_6, alle anderen Regionen werden nicht vollständig von einer Teilszene T_j überdeckt

Die schwache Verdeckungserkennung soll ausdrücken, daß das Renderingverfahren einen klar definierten Vordergrund auf einem vorgegebenen Genauigkeitsraster (in Form der Regionen C_i) als solchen erkennt: In einer Region dürfen keine Hintergrundpunkte angezeigt werden, wenn diese vollständig durch einen Vordergrund in geringerer Tiefe überdeckt ist.

Sei nun d der Durchmesser der Farbklötze, die zur Rekonstruktion verwendet werden, in Pixeln und $r := \lfloor d/2 \rfloor$ der auf ganze Pixel abgerundete Radius. Das Bild kann dann in ein Raster von Zellen mit Durchmesser r eingeteilt werden. Der Einfachheit halber sei angenommen, daß Breite und Höhe des Bildes ganzzahlig durch r teilbar sind. Jeder Farbklotz, dessen Mittelpunkt sich in einer Zelle befindet, deckt diese vollständig ab. Wenn nun in jede Zelle mindestens ein Punkt aus einem Vordergrundbereich fällt, so erhält man durch den z-Test im Tiefenpuffer eine schwache Verdeckungserkennung bezüglich des Zellrasters.

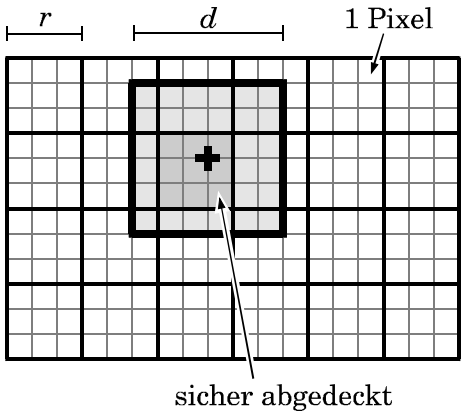


Abbildung 28: Einteilung des Bildes in Zellen

Da hier an die Zellen genau die gleichen Anforderungen wie an die Pixel bei der pixelweisen Rekonstruktion gestellt werden, lassen sich die Ergebnisse aus Satz 5 direkt übertragen, indem die projizierten Flächen a, v, h in Pixeln durch a/r^2, v/r^2, h/r^2, also die projizierten Flächen gemessen in Zellen, ersetzt werden. Man erhält damit:

Korollar 1: *Bildrekonstruktion mit großen Farbklötzen*

Es sei eine Situation wie in Satz 5 gegeben. Bezeichne wiederum mit a die gesamte und mit v die sichtbare projizierte Fläche in der Sichtpyramide in Pixeln. d sei der Durchmesser der Farbklötze in Pixeln. Weiter seien $r := \lfloor d/2 \rfloor$, $a' := a/r^2$ und $v' := v/r^2$. Teile das Bild in ein gleichmäßiges Raster von Zellen mit r Pixeln Breite und Höhe ein.

Dann wächst der Erwartungswert für die Größe einer Stichprobe, bei der die Bildrekonstruktion mit Farbklötzen der Größe d eine schwache Verdeckungserkennung bezüglich der durch das Raster gebildeten Zellen leistet, mit $O(a' \cdot \log v')$.

Um eine schwache Verdeckungserkennung bezüglich dieser Zellen mit Sicherheit s zu erhalten, sind ungefähr

$$m \geq \frac{a'}{v'}\left(m_v + \Phi^{-1}(1-\bar{s})\sqrt{m_v}\right) \text{ Stichprobenpunkte}$$

$$\text{mit } m_v := \frac{\log\left(1-\bar{s}^{1/v'}\right)}{\log\left(1-\frac{1}{v'}\right)} \overset{v' \text{ groß}}{\approx} v' \log\left(1-\bar{s}^{1/v'}\right)$$

$$\text{und } \bar{s} = \sqrt{s}$$

hinreichend, wenn s nicht extrem klein gewählt wird.

In der Praxis stellt sich allerdings heraus, daß diese Abschätzung zu pessimistisch ist: Die Zellengröße ist nur ein viertel so groß wie die tatsächliche von jedem Klotz bedeckte Fläche. Es genügt erfahrungsgemäß, in obiger Abschätzung statt r den Wert d zu verwenden, um Bilder ohne Verdeckungsfehler zu erhalten. Dies führt dazu, daß der Stichprobenumfang um etwas mehr als den Faktor 4 gegenüber der pessimistischen Abschätzung gesenkt werden kann.

Die Zellen, innerhalb derer die Verdeckung richtig erkannt werden sollen, sind nur ein künstliches Konstrukt, um diese Verdeckungseigenschaft einfach nachweisen zu können. Die Erfahrung mit dem Verfahren zeigt, daß generell Vordergrundbereiche mit Mindestdurchmesser d^2 in beide Richtungen mit hoher Wahrscheinlichkeit den Hintergrund im erzeugten Bild verdecken. Diese müssen nicht in das in Korollar 1 verwendete Raster fallen. Dies für allgemeine Formen und beliebige Positionen der Zellen zu beweisen, ist allerdings schwierig, da die Bereiche sich dann überschneiden und die in ihnen erscheinenden Szenenteile nicht mehr disjunkt sind. Die bisherigen Sätze erfordern aber disjunkte und gleich große „Behälter“, in die die Punkte plaziert werden. Da die Wahrscheinlichkeiten, daß Stichprobenpunkte in nichtdisjunkte Szenenteile fallen, nicht mehr unabhängig voneinander sind, läßt sich auch das Beweisprinzip nicht für Zellen in beliebiger Position aufrechterhalten: In der Herleitung von Lemma 2 wurde explizit Gebrauch davon gemacht, daß die Wahrscheinlichkeit für alle Behälter unabhängig ist, einen „Ball“ zu erhalten.

Als empirisches Ergebnis läßt sich festhalten, daß Vordergrundbereiche, die ein Quadrat der Größe d vollständig enthalten, bei der Rekonstruktion mit großen Farbklötzen in ihrem Inneren korrekt dargestellt werden, falls ein Stichprobenumfang gemäß Korollar 1 (auch mit $r=d$ statt $\lfloor d/2 \rfloor$) gewählt wird. An den Randbereichen, etwa in einem Abstand von bis zu r zum Rand, kann es häufiger zu Verdeckungsfehlern kommen.

Laufzeitgewinn

Im folgenden wird angenommen, daß der Stichprobenumfang bei der Verwendung von Farbklötzen mit Durchmesser d um $1/d^2$ reduziert werden kann. Man erhält dann folgendes Szenario: Anstatt erwarteten $a \cdot \ln v$ Stichprobenpunkte wird nur noch ein Stichprobenumfang von $(a/d^2)\ln(v/d^2)$ Punkten erwartet. Die insgesamt im Bild eingezeichneten Pixel sind $(d^2 \cdot a/d^2)\ln(v/d^2) = a\ln(v/d^2)$ Stück, das Überfüllungsverhältnis beträgt also $\ln(v/d^2)$ statt $\ln v$, wird also im Gegensatz zum Stichprobenumfang nur schwach reduziert. Wenn d relativ klein (etwa 2-5) und v relativ groß (etwa $\geq$ 100.000) ist, erreicht man eine Reduktion des Stichprobenumfangs um den Faktor $1/d^2$, muß aber etwa genau so viele Pixel (mit z-Test) füllen wie zuvor. Diese Operation ist allerdings auf den allermeisten Hardwareplattformen um Größenordnungen schneller, da keine aufwendigen Fließkommamultiplikationen benötigt werden, um die Pixel zu füllen.

Sei eine Zeit t_{pixel} angenommen, die benötigt wird, um einen Pixel zu füllen und eine Zeit $t_{projekt}$, die benötigt wird, um einen Stichprobenpunkt zu transformieren[11]. Dann ergibt sich als Laufzeit für die Bildrekonstruktion mit Farbklötzen des Durchmessers d ($d=1$ ist der Spezialfall der pixelweisen Rekonstruktion) $T(\mathrm{d}) = t_{projekt} \cdot (a/d^2)\ln(v/d^2) + t_{pixel} \cdot a\ln(v/d^2)$.

Bezogen auf den Aufwand bei pixelweiser Rekonstruktion ergibt sich eine Reduktion der Laufzeit $R(d) := T(d) / T(1)$ von:

$$
\begin{aligned}
R(d) &= \frac{t_{projekt} \cdot (a/d^2)\ln(v/d^2) + t_{pixel} \cdot a\ln(v/d^2)}{t_{projekt} \cdot a\ln v + t_{pixel} \cdot a\ln v} \\
&= \frac{\left((1/d^2)\,t_{projekt} + t_{pixel}\right) \cdot \left(\ln v - \ln d^2\right)}{\left(t_{projekt} + t_{pixel}\right) \cdot \ln v} \qquad \text{(a)} \\
&\leq \frac{\left((1/d^2)\,t_{projekt} + t_{pixel}\right) \cdot \ln v}{\left(t_{projekt} + t_{pixel}\right) \cdot \ln v} \\
&= \frac{(1/d^2)\,t_{projekt} + t_{pixel}}{t_{projekt} + t_{pixel}} \qquad \text{(b)}
\end{aligned}
$$

Es sei nun vorausgesetzt, daß $t_{projekt}$ deutlich größer als t_{pixel} ist, und daß d relativ klein ist. Man erhält dann aus (b), daß die Reduktion der Laufzeit in der Größenordnung d^2 liegt, wenn t_{pixel} deutlich kleiner als $(1/d^2) \cdot t_{projekt}$ ist und daher im Nenner und im Zähler vernachlässigt werden kann.

Wenn im anderen Extrem t_{pixel} deutlich größer als $(1/d^2) \cdot t_{projekt}$ ist, d.h. wenn d sehr groß wird, kann der Term $(1/d^2) \cdot t_{projekt}$ in (a) vernachlässigt werden. Ignoriert man weiter den kleinen Anteil t_{pixel} im Nenner, so erhält man näherungsweise eine Reduktion von

---

[11] Diese Annahme ist etwas ungenau, da größere Farbklötze größere lineare Speicherzugriffe am Stück produzieren und daher oft deutlich schneller durchführbar sind.

$$\frac{t_{pixel}}{t_{projekt}} \cdot \frac{\left(\ln v - \ln d^2\right)}{\ln v}.$$

Der Geschwindigkeitsgewinn reduziert sich also für große d auf das Geschwindigkeitsverhältnis der beiden elementaren Operationen mal der Reduktion des logarithmischen Überhangs, der zum Auffüllen aller Pixel erforderlich ist.

Bildqualität in Relation zum Laufzeitgewinn

Es können natürlich keine beliebigen Werte für d verwendet werden, da die Detailschärfe des Bildes dadurch abnimmt und Kanten beginnen „auszufransen". Farbbild 5 bis Farbbild 11 in Anhang C zeigen die Ergebnisse der Bildrekonstruktion mit verschieden großen Farbklötzen für verschiedene Szenen. Wenn man Qualität und Rechenzeit in den Beispielen vergleicht, stellt sich meistens ein Wert von $d = 2..5$ als akzeptabler Kompromiß heraus, man erhält einen deutlichen Geschwindigkeitsgewinn bei akzeptabler Bildqualität. Für $d = 2$ ist der Unterschied zur pixelweisen Rekonstruktion kaum erkennbar, die Rechenzeit ist aber nur etwa ein viertel so groß. Für $d = 5$ verschwinden bereits einige Details und die Kanten laufen aus, für Fernbereichsapproximationen kann dieser Wert aber noch durchaus eingesetzt werden. Die Problemfälle aus Abschnitt 3.2.1.1 treten bei der Rekonstruktion mit großen Farbklötzen ebenfalls in ähnlicher Form auf. Insbesondere werden sehr nahe zusammenliegende, ausgedehnte Objekte (Oberflächendetailpolygone u.ä.) falsch wiedergegeben, was mit zunehmendem Durchmesser der Farbklötze stärker wird. Dies ist vor allem am Beispiel der Stadtszenen (Farbbild 6) zu sehen. In Kapitel 5.2 werden die verschiedenen Ergebnisse für die Testszenen noch genauer beschrieben.

Abbildung 29 zeigt die Abhängigkeit der Laufzeit zur Detailauflösung graphisch. Die lineare Skalierung der d-Achse gibt mit dem Durchmesser der Farbklötze in etwa den Verlust an Kantenauflösung wieder, der in Kauf genommen werden muß. Es ist deutlich zu erkennen, daß für kleine Werte von d große Laufzeitgewinne ohne starken Verlust erzielt werden können, während für größere Werte die Gewinne immer schwächer wachsen, da der Aufwand zum Füllen der Pixel wesentlich langsamer abfällt als der Aufwand für die Transformation der Stichprobenpunkte.

Die schwarz eingezeichneten, gemessenen Werte wurden für die in Farbbild 1 dargestellte Testszene[12] bestimmt, die auch im nächsten Abschnitt verwendet wird. Die in grau eingezeichneten Kurven zeigen die Schätzungen (a) und (b) für $R(d)$ aus dem vorherigen Abschnitt. Der Wert für $t_{projekt}$ wurde aus der Laufzeit bei $d = 1$ durch Division durch den Stichprobenumfang bestimmt, der Wert t_{pixel} aus der Laufzeit bei $d = 80$ durch Division durch die Pixelanzahl. Im allgemeinen sind allerdings keine genauen Schätzungen zu erwarten: Der Wert für t_{pixel} schwankte bei dem Experiment um den Faktor 3, je nachdem, wie man ihn bestimmte (etwa mit größerem d oder mit einer Ansicht in der mehr Pixel in den Blöcken verdeckt sind). Dies liegt daran, daß sich bei dieser elementaren Operation bemerkbar machen kann, ob ein Pixel geschrieben oder im Tiefenpuffer abgeblockt wird. Außerdem treten bei größeren Werten für d längere lineare Speicherzugriffe auf, die schneller ausgeführt werden können. Der hier verwendete Wert t_{pixel} ist daher für die Meßreihe günstig gewählt, im allgemeinen sind keine so genauen Voraussagen möglich: mit verschiedenen Werten für t_{pixel} traten entsprechend Abweichungen bis zum Faktor 3 im Bereich größerer d auf. Dennoch gibt das einfache Modell zumindest qualitativ gut den Verlauf der Laufzeit wieder.

---

[12] Das dort verwendete Modell einer Kuh gehört zu einer Sammlung von Modellen, die in [Garland 99] dazu verwendet werden, um das dort vorgestellte Mesh Simplification Verfahren zu testen. Die Modelle wurden vom Autor im Internet zur Verfügung gestellt (die Adresse findet sich im Literaturverzeichnis unter [Garland 99]).

Man sieht, daß Abschätzung (b) für sehr große Werte von d zu pessimistisch ist. Durch die drastische Verringerung der Zellen, die gefüllt werden müssen, verringert sich das Überfüllungsverhältnis in dem Beispiel etwa um einen Faktor 3, was sich entsprechend deutlich in der Laufzeit niederschlägt.

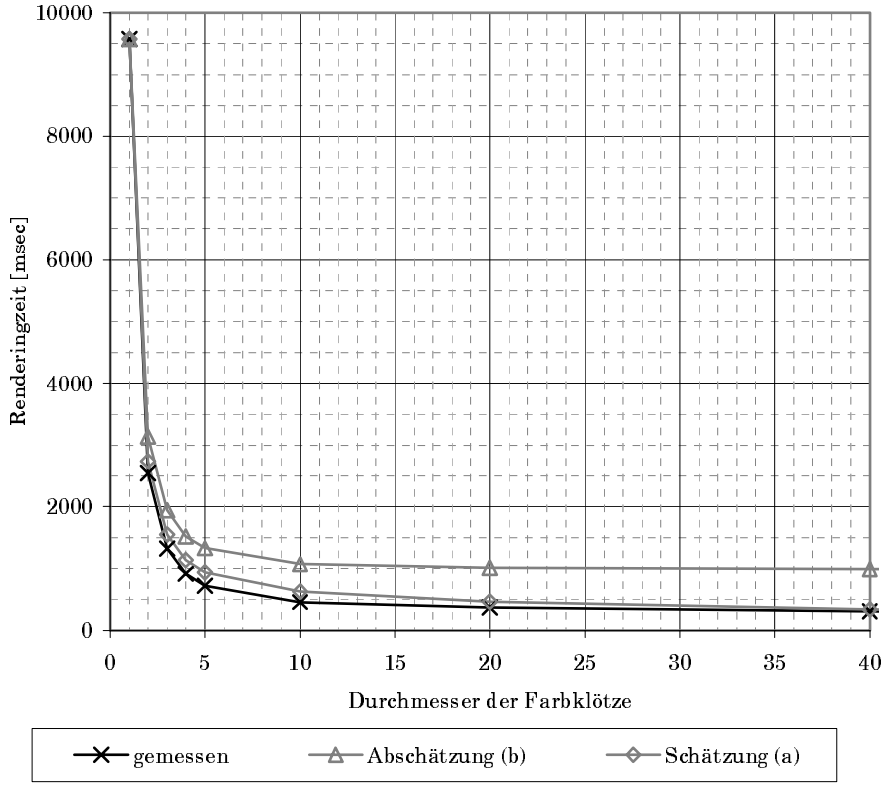


Abbildung 29: Laufzeit in Abhängigkeit von *d*

3.2.3 Bildrekonstruktion ohne Tiefenpuffer

Die Rekonstruktion eines Bildes aus der Stichprobe kann konzeptionell in zwei unabhängige Teile zerlegt werden. Zum einen müssen die verdeckten Stichprobenpunkte als solche erkannt und aus dem Bild entfernt werden und zum anderen muß aus den verbliebenen ein Bild rekonstruiert werden. Die beiden bisher gezeigten Methoden haben dies in einem Schritt durch die Verwendung des Tiefenpuffers erledigt. Indem man die Schritte trennt, können flexiblere Verfahren zur Bildrekonstruktion verwendet werden. Die in diesem Kapitel vorgestellten Verfahren sind zum Teil nur unvollständig implementiert und untersucht worden, da eine umfassende Behandlung den Rahmen dieser Arbeit gesprengt hätte. Das Ziel war vor allem, durch Experimentieren einen Eindruck zu bekommen, welche Verbesserungen mit allgemeineren Bildrekonstruktionsverfahren noch möglich sind.

3.2.3.1 Entfernen verdeckter Punkte

Das Entfernen verdeckter Punkte ist ein zentrales Problem beim randomisierten z-Buffer Verfahren. Für einen gegebenen Bereich, der nur größere (tiefere) z-Werte als seine Umgebung enthält, muß entschieden werden, ob es sich tatsächlich um ein „Loch“ in der Oberfläche handelt, oder ob nur zufällig kein näher gelegener Punkt in dem Bereich gesetzt wurde (schematisch in Abbildung 30).

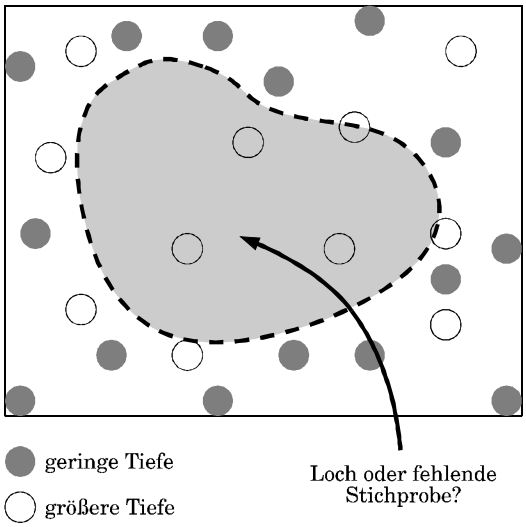


Abbildung 30: Erkennung der Verdeckungssituation

Die Herleitung von Korollar 1 gibt eine Antwort auf diese Frage: Wenn ein größerer Vordergrundbereich eine Zelle mit Durchmesser r vollständig bedeckt, dann findet sich innerhalb dieser Zelle auch mindestens ein Punkt aus diesem Bereich. Einer dieser Vordergrundpunkte besitzt dann auch jeweils einen minimalen Tiefenwert unter allen Stichprobenpunkten in der Zelle. Damit ergibt sich eine Strategie zur Entfernung verdeckter Punkte:

```
1) Es sei eine nach der projizierten Fläche gewichtet verteilte
   Menge von Stichprobenpunkten gegeben.
2) Teile die Punkte nach Korollar 1 in ein Raster ein.
3) Lösche in jeder Zelle alle Punkte, außer denjenigen mit
   minimaler Tiefe.
```

Algorithmus 3: Zellenweise Erkennung verdeckter Flächen

Unter den in Korollar 1 angegebenen Bedingungen leistet dieser eine schwache Verdeckungserkennung bezüglich eines Rasters von Zellen mit Durchmesser $r = \lfloor d/2 \rfloor$, in der Praxis kann r auch durch d ersetzt werden. Algorithmus 3 entspricht vom Vorgehen her im wesentlichen der pixelweisen Rekonstruktion. Die Ausgabe ist allerdings eine Menge von projizierten Punkten an exakten Koordinaten und kein Pixelbild. In Analogie zur Rekonstruktion mit großen Farbklötzen, erscheint folgende Strategie ebenfalls naheliegend:

```
1) wähle nach der projizierten Fläche gleichverteilte
   Stichprobenpunkte, mit einer Anzahl in Abhängigkeit von einem
   Parameter d, wie in Abschnitt 3.2.2 beschrieben.
2) Bestimme für jeden Punkt p alle Punkte, die sich in dem
   Quadrat mit Mittelpunkt p und Durchmesser d befinden.
3) Lösche unter diesen alle Punkte, die keine minimale Tiefe
   aufweisen.
```

Algorithmus 4: Erkennung verdeckter Flächen in quadratischen Nachbarbereichen

Das Ergebnis dieses Algorithmus entspricht dem der Farbklotzmethode, bis auf den Umstand, daß man als Ausgabe eine Menge von Stichprobenpunkten statt eines aufgefüllten Rasterbildes erhält.

3.2.3.2 Bessere Ausnutzung der Stichprobe

Typischerweise fallen bei obigem Vorgehen nach Algorithmus 3 bzw. Algorithmus 4 noch einige weitere Punkte der selben Oberfläche in jedes Quadrat, die ungenutzt gelöscht werden. Die erwartete Anzahl solcher Punkte ist $\ln(v)$-1, wenn v die sichtbare Oberfläche ist. Neben den dadurch möglichen Aliasingartefakten wird damit auch zusätzliche Bildinformation verschenkt. Dies läßt sich beheben, indem ein weniger strenges Kriterium für das Löschen von Punkten angewendet wird. Dabei sind verschiedene Heuristiken denkbar um zu entscheiden, ob die Punkte verdeckt sind oder noch auf der verdeckenden Oberfläche liegen. Eine naheliegende Möglichkeit besteht darin, einen Toleranzbereich für den Tiefenwert über dem Minimum anzugeben, innerhalb dessen die Punkte nicht gelöscht werden. Dies kann allerdings in vielen Fällen zu Fehlinterpretationen führen. Ein wichtiges Beispiel sind Kanten, an denen sich die Oberfläche von ihrer dem Betrachter zugewandten zu der dem Betrachter abgewandten Seite „umbiegt“ (schematisch in Abbildung 31, Abbildung 32 zeigt das Problem für eine Beispielszene).

Um diese Fälle auszuschließen, können auch die Oberflächennormalen betrachtet werden. Solange nur geschlossene Oberflächen betrachtet werden, können alle Punkte mit betrachterabgewandter Normale gelöscht werden. Als weitere Verbesserungen wären die Berücksichtigung anderer Oberflächenattribute (wie Farbe) oder der Vergleich der Tiefenwerte in benachbarten Zellen denkbar. Es ist allerdings nicht zu erwarten, daß diese Heuristiken ohne Annahmen über die Szene immer zu korrekten Ergebnissen führen können.

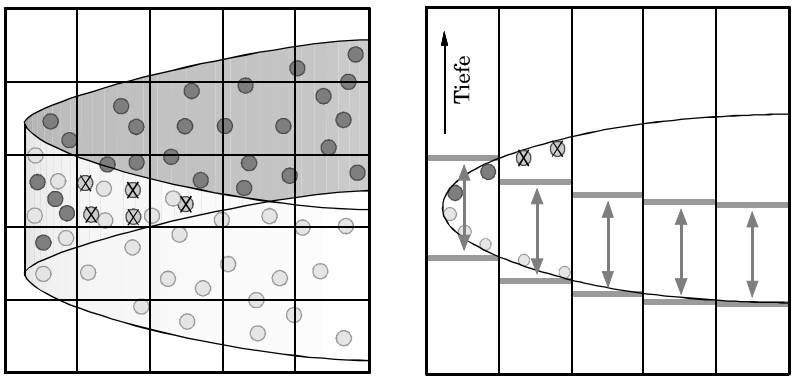
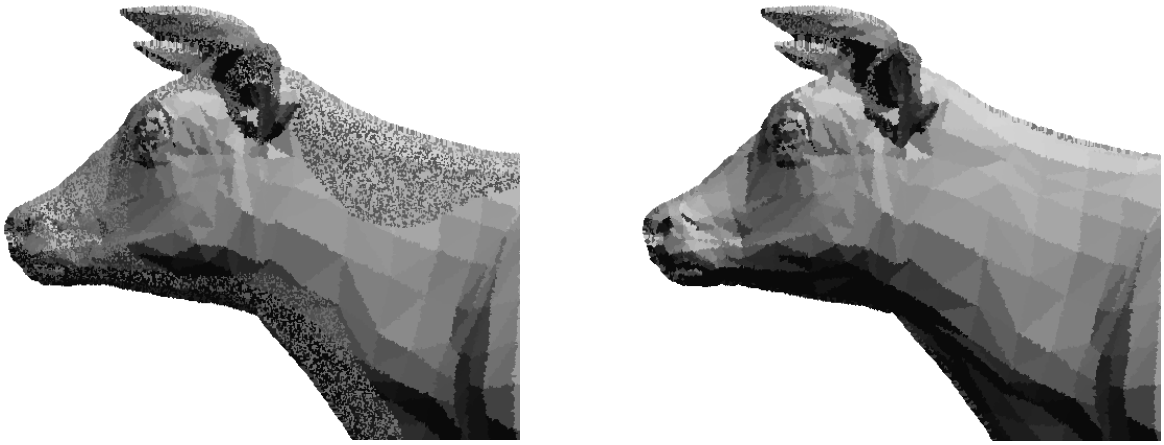


Abbildung 31: Problemfall für einen einfachen Tiefentoleranzbereich



(a) falscher Tiefenbereich **(b) besserer Wert für den Tiefentoleranzbereich**

Abbildung 32: Entfernen verdeckter Oberflächenpunkte ($d = 5$) nach Algorithmus 4 mit verschiedenen Tiefentoleranzbereichen, keine Unterdrückung abgewandter Normalen, Rekonstruktion mit diskreten Voronoiregionen (Abschnitt 3.2.3.4). In Bild (a) scheint die Rückseite des Kuhmodells teilweise durch.

In dieser Arbeit wurde eine Methode mit einem Tiefentoleranzbereich und Löschen abgewandter Normalen implementiert. Eine genaue Untersuchung der weiteren möglichen Heuristiken hätte den Rahmen dieser Arbeit überstiegen, es sollte nur grob ausgelotet werden, wie stark sich die Bildqualität durch die zusätzlichen Informationen steigern läßt. Der Einfachheit halber wurden die projizierten Punkte nicht exakt gespeichert, sondern aus einem höher aufgelösten, mit pixelweiser Rekonstruktion erstellten Rasterbild abgeleitet. Dies führte zwar zu einem etwas ungenaueren Ergebnis und schlechteren Laufzeiten, erforderte aber keine vollständige Neuimplementation der Projektionsroutine, die bei den anderen Verfahren komplett von der verwendeten Grafikbibliothek übernommen wurde (siehe auch Kapitel 4). Die im folgenden angegebenen Laufzeiten sind daher mit entsprechendem Vorbehalt zu betrachten.

Die Tiefenwerte wurden direkt aus dem Tiefenpuffer übernommen. Diese sind dort nicht linear abgelegt, sondern als projizierte Tiefenwerte ([Foley et al. 96], S. 671) nach $z \to d \cdot z/(d+z)$, d.h. größere z-Werte werden gestaucht, damit der Vordergrund eine höhere Auflösung hat. Dieser Effekt wurde nicht kompensiert, da er für einfache Szenen mit geringer Tiefe keinen großen Einfluß hat und im allgemeinen nicht klar ist, ob dies einen negativen oder einen positiven Effekt auf das Ergebnis haben könnte.

Der Verlust an Genauigkeit durch das ganzzahlige Pixelraster dürfte bei der verwendeten Auflösung zu vernachlässigen sein. Das Phänomen der z-Dominanz spielt bei geringem Füllungsgrad des Tiefenpuffers mit Stichprobenpunkten ebenfalls eine untergeordnete Rolle: Sei A die Anzahl von Pixeln und $p := 1/A$ die Wahrscheinlichkeit, daß ein Pixel mit einem Punkt gefüllt wird. Wenn nun k Punkte gleichmäßig über die A Pixel verteilt werden, ist für jeden festen Pixel die Wahrscheinlichkeit, daß sich zwei oder mehr Punkte darin finden, gleich:

$$p_2 = 1 - \left[\underbrace{(1-p)^k}_{\text{kein Punkt}} + \underbrace{k \cdot p \cdot (1-p)^{k-1}}_{\text{genau ein Punkt}} \right]$$

Dies ist auch der Erwartungswert für die Häufigkeit, mit der dieses Ereignis in einem Punkt eintritt. Der Erwartungswert für die Anzahl von Pixeln mit mehr als 2 Punkten im ganzen Bild ist daher (Linearität des Erwartungswertes) $k \cdot p_2$. Als Anteil am gesamten Bild erhält man somit wieder p_2. Abbildung 33 zeigt den erwarteten Anteil von Pixeln, die mehr als einem Stichprobenpunkt erhalten in Abhängigkeit des Stichprobenumfangs. Solange dieser die Hälfte der Vordergrundfläche nicht übersteigt, erhält man bei 640×480 Pixeln Vordergrundfläche höchstens 10% Pixel, in die mehr als ein Punkt fällt. In den weiteren Beispielen wurde stets eine deutlich geringere Stichprobenanzahl als die halbe Auflösung verwendet. Daher war zu erwarten, daß Artefakte durch z-Dominanz keine Rolle gespielt haben.

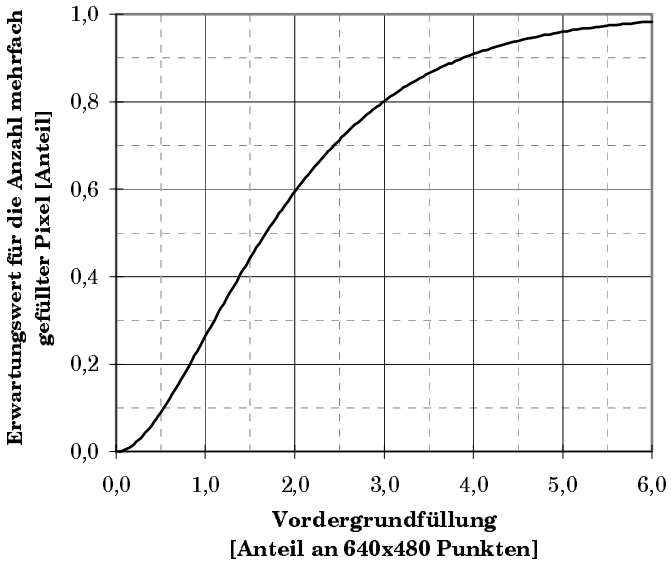


Abbildung 33: Anteil mehrfach gefüllter Pixel bei verschiedenem Stichprobenumfang

In den nächsten beiden Abschnitten sind einige Ergebnisse beschrieben, die sich mit dieser einfachen Implementation erzielen ließen.

3.2.3.3 Rekonstruktion durch Tiefpaßfilter

[Glassner 95a] beschreibt, wie in Abschnitt 2.8.2 kurz vorgestellt, einige Methoden, um aus unstrukturierten Punkten ein Bild zu rekonstruieren. Eine wichtige Klasse bilden die *„local filtering“* Methoden, bei denen die Bildpunkte als mit einer Filterfunktion gewichtetes Mittel von Punkten der Nachbarschaft gebildet werden. [Glassner 95a] diskutiert verschiedene Filter und Vor- und Nachteile. Als klassisches Beispiel wurde hier nur die Methode mit einem Gaußschen Tiefpaßfilter implementiert, die auch von [Cook 91] zur Bildrekonstruktion beim stochastischen Raytracing verwendet wurde. Ziel war es nur, einen Eindruck zu erhalten, inwieweit sich die Bildqualität verbessern läßt, insbesondere, inwieweit sich Aliasingartefakte dadurch verringern ließen.

Als Filterfunktion wurde, analog zu [Cook 91], die Funktion

$$f_c(x,y) = e^{\left(-\frac{(1{,}5x)^2+(1{,}5y)^2}{c^2}\right)}$$

verwendet, wobei die Funktion für Werte mit $|x/c| \geq 1$ oder $|y/c| \geq 1$ auf null gesetzt wurde, da die Exponentialfunktion dort bereits sehr kleine Werte annimmt. c gibt dabei die Ausdehnung des Filters in Pixeln an. Der Wert eines Pixels ergibt sich als gewichtetes Mittel aller Punkte mit dem Wert $f_c(x,y)$ als Gewicht, wobei der Vektor (x,y) die Distanz zwischen dem zur Gewichtung betrachteten Punkt und dem Pixel, dessen Intensität bestimmt werden soll, in Pixeln angibt. Durch die Beschränkung des Trägers der Filterfunktion auf $2c$ wird der Aufwand zur Berechnung dieses Mittelwertes drastisch reduziert.

[Glassner 95a] stellt die signaltheoretischen Hintergründe von Anti-Aliasing und Filterung ausführlich vor, im folgenden folgt eine sehr kurze Zusammenfassung für den hier betrachteten Fall:

Durch die Faltung eines Signals mit der Filterfunktion werden hohe Frequenzen im Fourierspektrum gedämpft. Je größer der Wert c ist, desto niedriger liegt der Frequenzbereich, in dem die Dämpfung einsetzt. Frequenzen oberhalb der halben Rasterfrequenz werden durch den Rasterungsprozeß an dem Wert der halben Rasterfrequenz in Richtung der Frequenzachse gespiegelt und treten als niederfrequente Störsignale (Aliasing) auf. Ein Gaußscher Tiefpaßfilter mit einem Wert c in der Größenordnung einiger Pixel, angewandt auf das Signal vor der Rasterung, dämpft die höheren Frequenzen so stark, daß nur noch wenig Aliasing auftritt. Da aber gleichzeitig auch höhere Frequenzen im Bereich unter der halben Rasterfrequenz abgeschwächt werden, nimmt die allgemeine Bildschärfe mit höheren Werten von c ebenfalls ab, daher ist die Wahl dieses Wertes ein Trade-Off zwischen Aliasing und Schärfe. Im Folgenden wurden verschiedene Werte für c nach optischem Eindruck gewählt.

Im Fall der zufälligen Abtastung kommen weitere Artefakte hinzu: Das Faltungsintegral der Filterfunktion mit dem Bildsignal wird hier nicht exakt berechnet, sondern nur aufgrund einer zufälligen Stichprobe geschätzt. Dies führt zu zufälligen Abweichungen von theoretischen Wert und damit zu Rauschen im Bild (soweit die Theorie nach [Glassner 95a]).

Der Algorithmus

Hier wurde folgender Algorithmus zur Berechnung des gefilterten Bildes verwendet:

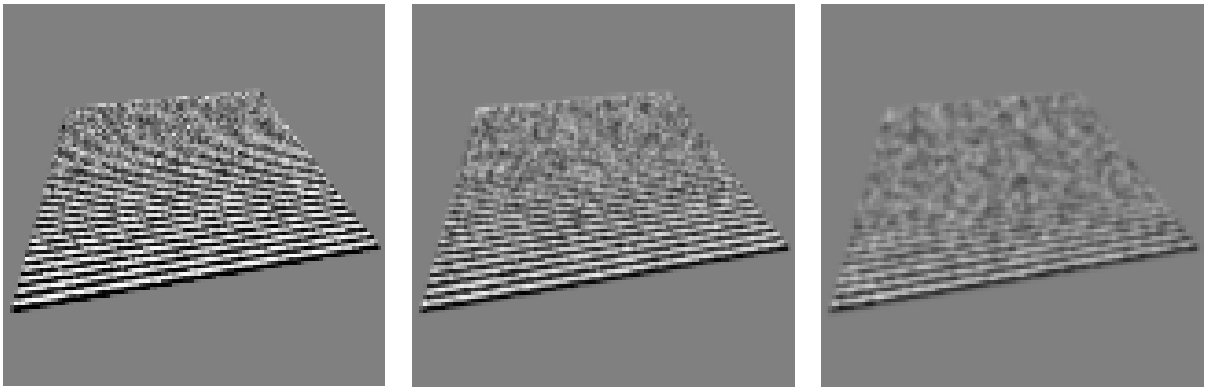
```
1) Bestimme eine Stichprobenmenge (aus einem Rasterbild und
   Tiefenpuffer mit Breite b und Höhe h)
2) berechne die Filtergewichte in einem Feld F von 2c x 2c
   Einträgen vor
3) bilde ein Feld B mit Breite b/c und Höhe h/c von Listen
4) Teile die Stichprobenpositionen in Zellen der Größe 2c ein
   und sortiere die Stichprobenpunkte entspr. in B ein.
5) Für alle Punkte (x,y) des Bildes:
6)    Bestimme die 4 Zellen in B, auf denen der Filter ≠ 0 ist.
7)    Summe = 0
8)    Für alle Punkte p = (a,b) der Zellen aus B:
9)       Falls |a-x| < 1.5c und |b-y| < 1.5c
10)         Summe += Farbe(p)*F[a-x][b-y]
11)   Färbe Punkt in Summenfarbe
```

Algorithmus 5: Rekonstruktion mit Filterfunktion

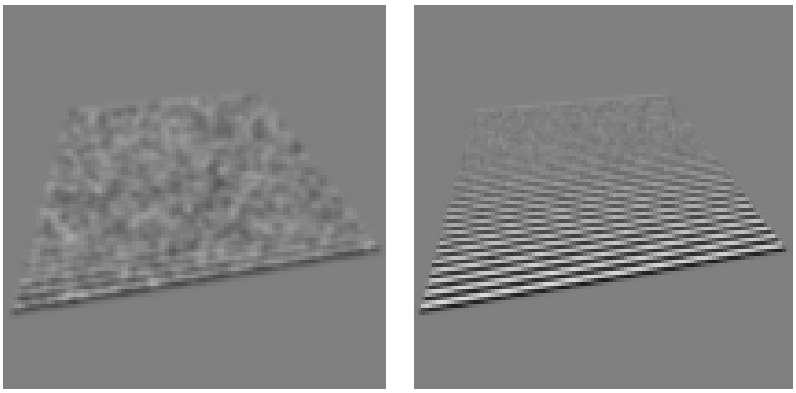
Die Berechnung des gewichteten Mittels eines Punktes erfordert, alle Punkte, für die das Gewicht nicht null ist, zu bestimmen. Diese Suche wird hier mit einem groben Raster („*buckets*“) der Größe $2c$ beschleunigt. Der Träger des Filters hat ebenfalls die Größe $2c$, und die sichtbaren Punkte sind über das Bild gleichverteilt. Daher ist nur ein konstanter Mehraufwand für die Suche zu erwarten, es werden im Mittel viermal so viele Punkte abgefragt (Zeile 9) wie zur Mittelwertbildung benötigt werden.

Ergebnisse

Abbildung 34 zeigt die aus Abbildung 24 bekannte Testszene. Diesmal wurde die oben beschriebene Rekonstruktionsmethode verwendet. Zur Entfernung verdeckter Flächen wurde Algorithmus 4 mit einem sehr großen Tiefentoleranzbereich verwendet, was möglich war, da die Szene (bis auf den Hintergrund) keine Verdeckung enthielt. Wie man sieht, lassen sich die Aliasingartefakte deutlich reduzieren. Als zusätzliches Artefakt erhält man allerdings ein grobgranulares Rauschen, das insbesondere in Bild c) und d) auffällt. Dieses Problem ist auch theoretisch zu erwarten: Nach [Cook 91] (Seite 189) hat ein rein zufälliges Punktraster ein Frequenzspektrum, in dem alle Frequenzen ungefähr gleichmäßig enthalten sind. Der Tiefpaßfilter filtert nun mit zunehmendem c ab einer immer geringeren Frequenz hochfrequente Anteile aus dem Bild und damit insbesondere aus dem Rauschanteil heraus. Das verbleibende Rauschen



a) $c = 1{,}5$ (8,1 s) **b) $c = 2{,}0$ (10,5 s)** **c) $c = 2{,}5$ (15,7 s)**



d) $c = 3{,}0$ (23,0 s) **e) $c = 2{,}0$, Mittelwert aus 64 Bildern (7m 9 s)**

Abbildung 34: Verminderung von Aliasing durch Rekonstruktion mit einem Gaußschem Tiefpaßfilter, Laufzeiten in Klammern, gemessen auf Testsystem 1 (siehe Anhang A)

besteht daher zunehmend aus niedrigen Frequenzen, welches für den menschlichen Beobachter störender wirkt als hochfrequentes Rauschen ([Glassner 95a], Seite 375). Auch deswegen sollte der Wert für c nicht zu groß gewählt werden. Das Rauschen läßt sich insgesamt reduzieren, indem die Stichprobe vergrößert wird. Dies wurde in Bild e) gemacht; anstatt die Stichprobe zu vergrößern, was aufgrund der einfachen Implementation zu Problemen wie z-Dominanz geführt hätte, wurde hier das arithmetische Mittel über mehrere Bilder gebildet. Dies führt prinzipiell zum gleichen Ergebnis, allerdings wird für jedes Bild die Summe der Punkte auf die Summe ihrer Gewichte normiert anstatt nur einmal für alle Stichprobenpunkte, so daß die Filtergewichte nicht so exakt eingehalten würden, wie es bei einem einzigen Durchgang der Fall wäre. Dies könnte u.U. eine schwächere Abnahme des Rauschens verursachen.

Die Laufzeiten liegen bei der derzeitigen Implementation in Bereichen, die für Echtzeitanwendungen nicht geeignet sind (die Auflösung der Testbilder betrug nur 104×104 Punkte!). Es ist unklar, ob sich diese durch Optimierungen der Implementation tatsächlich um mehrere Größenordnungen reduzieren lassen. Angesichts der $O(c^2 \cdot A)$ Multiplikationen, die für ein Bild mit A Pixeln notwendig sind, erscheint dies aber unwahrscheinlich.

3.2.3.4 Diskrete Voronoiregionen

Die Rekonstruktionsmethode mit großen Farbklötzen lieferte brauchbare Bilder mit einem geringeren Stichprobenumfang als die pixelweise Rekonstruktion. Die Idee, quadratische Bereiche fester Größe mit der Farbe des Stichprobenpunktes zu füllen, wirkt dabei etwas künstlich. Es erscheint naheliegender, so etwas wie die Nachbarschaft des Punktes, abhängig von der Position der Nachbarpunkte, zu füllen. Eine kanonische Definition von Nachbarschaft ist die Voronoiregion des Punktes ([de Berg et al. 94], S. 145ff): Sei eine Menge M von Punkten in der Ebene gegeben. Dann ist die Voronoiregion eines Punktes p diejenige Teilmenge der Ebene, die aus Punkten besteht, die näher zu p liegen als zu irgendeinem anderen Punkt aus M.

Bei der Bildrekonstruktion ist M die Menge der Stichprobenpunkte, die in sichtbaren Bereichen liegen. Es wird dann die Voronoiregion jedes Punktes bestimmt und in der Farbe des Punktes eingefärbt. Von dem entstandenen Bild erhofft man sich eine bessere Bildqualität, da Regionen, in denen mehr Stichprobenpunkte liegen, automatisch feiner aufgelöst werden. Zur Bestimmung der Sichtbarkeit kann eine der in Abschnitt 3.2.3.1 bzw. 3.2.3.2 beschriebenen Methoden angewandt werden. Die analytische Berechnung der Voronoiregionen ist allerdings vergleichsweise aufwendig. Es kann gezeigt werden, daß ein Aufwand von $\Theta(n \cdot \log n)$ mit $n = |M|$ erforderlich ist ([de Berg et al. 94], S. 150).

Eine exakte Bestimmung der Voronoiregionen ist hier allerdings gar nicht notwendig, da sowieso ein Rasterbild davon erstellt werden soll. Daher kann ein einfacherer und schnellerer Algorithmus benutzt werden, der nur diskrete Voronoiregionen berechnet, d.h. ein Rasterbild als Näherung des geometrischen Diagramms. In [Eck et al. 95] wird ein verwandtes Problem behandelt, die Bestimmung diskreter Nachbarschaften in Dreiecksnetzen, und ein kanonischer Algorithmus dafür angegeben. Der hier vorgestellt Algorithmus für den Fall eines Rasterbildes arbeitet ähnlich.

Zunächst ist es einfacher, statt den Voronoiregionen bezüglich der Euklidischen Norm, diejenigen bezüglich der 1-Norm $\|(x,y)\| := |x| + |y|$ zu berechnen. Dies verzerrt zwar etwas den Nachbarschaftsbegriff, läßt sich aber wesentlich einfacher und effizienter implementieren. Die Idee ist, alle Punkte anfangs (wie in [Eck et al. 95]) in eine Warteschlange einzufügen, und danach für alle darin befindlichen Punkte jeweils die vier direkten Nachbarn $(x \pm 1, y \pm 1)$ zu füllen und in die Warteschlange aufzunehmen, sofern diese noch frei sind:

```
1) Bestimme ein schwach aufgefülltes Rasterbild
   mit pixelweiser Rekonstruktion
2) Entferne verdeckte Punkte (wie gehabt)
3) Füge alle verbliebenen Punkte in Warteschlange Q ein
4) Solange Q nicht leer:
5)    entnehme p aus Q
6)    Für die vier direkten Nachbarn q von p:
7)       Falls q frei:
8)          färbe q wie p
9)          füge q in Q ein
```

Algorithmus 6: Rekonstruktion mit diskreten Voronoiregionen

Während des Algorithmus wachsen die Regionen als diskrete Kreise in 1-Norm um die Punkte. Das Wachstum hält an den Stellen an, an denen sie aneinander stoßen. Abbildung 35 zeigt dies an einem Beispiel.

Der eigentliche Rekonstruktionsalgorithmus läuft offensichtlich in linearer Zeit bezüglich der Auflösung des Bildes ab, d.h. in $\Theta(A)$, wenn das Bild aus A Pixeln besteht. Dazu kommt der Aufwand zum Erkennen der verdeckten Flächen. Dazu kann z.B. Algorithmus 4 aus Abschnitt 3.2.3.1 verwendet werden. Die Suche nach Punkten im Umkreis d kann direkt auf dem Rasterbild erfolgen. Man erhält damit einen Aufwand von $\Theta(A + S \cdot d^2) \subseteq \mathrm{O}(A \cdot d^2)$ mit S als kleinerem der beiden Werte aus Stichprobengröße und A. Dies benötigt bezüglich der Pixeloperationen größenordnungsmäßig also etwa den selben Aufwand wie die entsprechende Farbklotzrekonstruktion. Wenn man statt dessen den einfacheren, zellenweisen Algorithmus 3 verwendet, erhält man

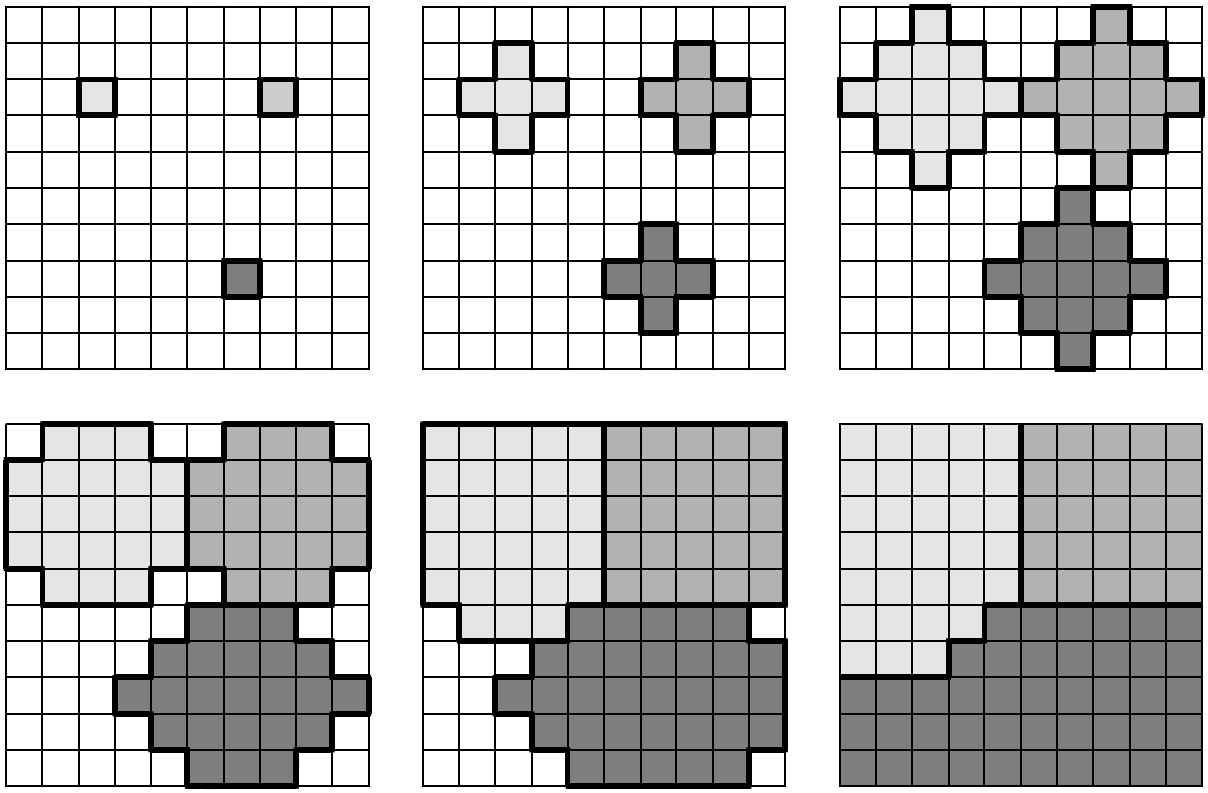


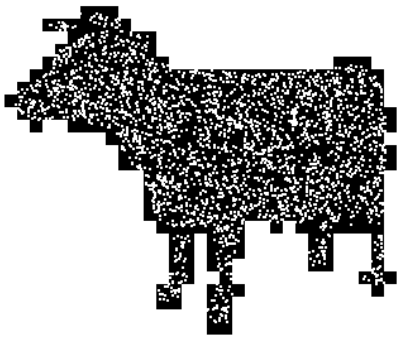
Abbildung 35: Wachstum der Voronoiregionen

einen Aufwand von $\Theta(A)$, ist also deutlich schneller. Durch die Verwendung von höheren Datenstrukturen (Prioritätenwarteschlange für Minimum im aktuellen Quadrat) könnte sich der asymptotische Aufwand von Algorithmus 4 vielleicht noch verringern lassen. Es ist jedoch fraglich, ob sich dies für kleine Werte von d wirklich lohnt oder ob der dadurch verursachte Überhang nicht sogar zu schlechteren Laufzeiten führen würde.

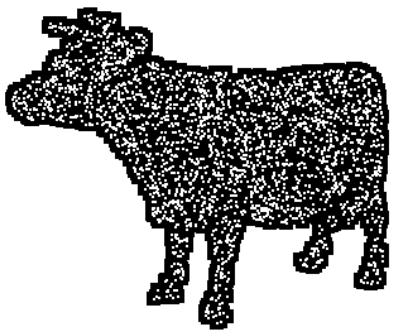
Ergebnisse

Farbbild 1 (in Anhang C) zeigt die Ergebnisse der Rekonstruktion mit diskreten Voronoiregionen. Die Entfernung verdeckter Oberflächenteile wurde einmal mit dem zellenweisen („Algorithmus 3“) und einmal mit dem Algorithmus mit frei beweglichen Verdeckungsquadraten („Algorithmus 4“) durchgeführt. Zum Vergleich wurde das Ergebnis der Rekonstruktion mit Farbklötzen daneben angegeben, bei allen drei Verfahren wurde jeweils der gleiche Durchmesser d verwendet. Die Implementation der Verdeckungsalgorithmen wurde nicht stark optimiert, so wurde z.B. auch für alle Hintergrundpixel unnötigerweise der Minimumtest durchgeführt. Dies führte vor allem bei Algorithmus 4 zu deutlich schlechteren Laufzeiten im Vergleich zu der im Prinzip genauso arbeitenden Farbklotzmethode. Insbesondere in allen Bildern, die viel Hintergrund enthalten, lieferte die Voronoimethode daher (zu Unrecht) schlechtere Laufzeiten als die Farbklotzmethode. In anderen Beispielen mit wenig Hintergrund fiel die Laufzeit der Voronoimethode (mit Algorithmus 4 zur Verdeckungserkennung) daher wesentlich schwächer gegenüber der Farbklotzmethode ab (siehe Farbbild 5, Farbbild 6, Farbbild 10), teilweise war sogar ein leichter Geschwindigkeitsgewinn zu erreichen.

Wie man in Farbbild 1 sieht, hat der verwendete Algorithmus zur Entfernung verdeckter Punkte einen starken Einfluß auf die Qualität der Ergebnisse: Der schnellere, blockweise Algorithmus 3 führt deutlich sichtbare Artefakte in das Bild ein; die Kanten von Objekten werden nur in der niedrigen Zellenauflösung wiedergegeben und die Rekonstruktion zeigt daher eine starke Blockstruktur (vor allem im Mittelwertbild (l) im Vergleich zu (k) sichtbar). Abbildung 36 zeigt die Ergebnisse der Entfernung verdeckter Punkte für diese Szene. Hier sieht man deutlich, daß der blockweise Algorithmus die Objektkanten mit geringer Auflösung wiedergibt. Bei der Methode mit frei positionierten Verdeckungsquadraten werden die Objektkanten dagegen wesentlich sauberer wiedergegeben, was zu einer wesentlich besseren Bildqualität



(a) Entfernen verdeckter Stichprobenpunkte mit Algorithmus 3 und Tiefentoleranz



(b) Entfernen verdeckter Stichprobenpunkte mit Algorithmus 4 und Tiefentoleranz

Abbildung 36: Entfernen verdeckter Stichprobenpunkte

führt. Im weiteren wird dieser daher stets zur Verdeckungserkennung bei der Rekonstruktion mit Voronoiregionen oder Tiefpaßfiltern verwendet.

Die Bildqualität bei den mit diskreten Voronoiregionen und frei positionierten Verdeckungsquadraten rekonstruierten Bildern erscheint insgesamt etwas besser als bei denen, die nach der Methode großer Farbklötze erstellt wurden. Farbbild 2 (f) (in Anhang C) zeigt die Differenz zwischen der Rekonstruktion mit diskreten Voronoiregionen und dem entsprechenden, mit dem gewöhnlichen z-Buffer bestimmten Bild. Im Vergleich zu Bild (d), welches die Abweichung bei der Farbklotzmethode zeigt, fallen die Fehler tatsächlich kleiner aus. Insbesondere das Auslaufen der Konturen fällt bei der Voronoimethode schwächer aus. Im selben Bild ist auch zum Vergleich eine Rekonstruktion mit einem Gaußschen Filter mit $c = 20$ (also der doppelten Verdeckungsauflösung, wie in Abbildung 34 (c)) angegeben. Das Bild zeigt die durch die grobe Abtastung entstandene Ungenauigkeit in Form von Unschärfe. Bei dem geringen Stichprobenumfang und dem grobstrukturierten Bild liefert diese Methode eher schlechte Ergebnisse. Sobald sich hochfrequente Anteile im Bild finden, liegt man mit der unscharfen Wiedergabe im Gegensatz zu den anderen Methoden auf der sicheren Seite in Bezug auf Aliasing (siehe auch die Untersuchung der Schachbrettszene in Kapitel 5.2), für eine hohe Bildqualität wird dann allerdings ein größerer Stichprobenumfang notwendig.

Anmerkung zu den angegebenen Laufzeiten: Die verwendete Grafikbibliothek gestattete keinen effizienten Zugriff auf den Bild- und Tiefenpuffer, dieser mußte vor einem Zugriff erst umkopiert und im Format konvertiert werden. Der dadurch zusätzlich entstehende Aufwand lag bei den oben angesprochenen Farbbildern in der Größenordnung von einer Sekunde. Da diese Operation prinzipiell nicht notwendig ist, wurde diese Zeit bei allen angegebenen Laufzeiten abgezogen. Dasselbe Problem trat auch bei den mit Gaußschen Filtern rekonstruierten Bildern auf, dort wurde dies ebenfalls entsprechend berücksichtigt.

3.2.4 Zusammenfassung der Ergebnisse

Unter relativ allgemeinen Bedingungen kann aus einer zufälligen Stichprobe ein näherungsweise korrektes Bild rekonstruiert werden. Das Hauptproblem liegt darin, die Verdeckungen zwischen den Stichprobenpunkten richtig zu rekonstruieren. Alle hier vorgestellten Verfahren teilen dazu daß Bild in feste oder bewegliche Zellen ein, in denen durch Minimumsuche und ggf. weitere, heuristische Kriterien die sichtbaren Punkte bestimmt werden. Problematisch für dieses Vorgehen sind ausgedehnte Oberflächen, die so nahe beieinander liegen, daß sich ihre Tiefenbereiche innerhalb eines Pixels (oder einer Zelle im allgemeineren Fall) stark überlappen. In solchen Fällen erscheint die verdeckende Oberfläche fälschlicherweise transparent, da mit erhöhter Wahrscheinlichkeit ein verdeckter Punkt als Minimum in einigen Zellen auftreten kann. Wenn in jeder Zelle die Tiefenwerte von verdeckten und sichtbaren Punkten in disjunkten Intervallen liegen, treten dagegen garantiert keine Probleme auf.

Um die Verdeckungsuntersuchung mit hoher Wahrscheinlichkeit korrekt durchführen zu können, ist ein Stichprobenumfang von $\Theta(a \cdot \log v)$ hinreichend, wobei a die gesamte und v die sichtbare projizierte Fläche, gemessen in Zellen, angibt.

Es wurden vier Verfahren zur Rekonstruktion vorgestellt:

- **pixelweise Rekonstruktion:** liefert Ergebnisse, die einem gewöhnlichen z-Buffer Rendering ähnlich sind. Für realistische Auflösungen ist der Stichprobenumfang zu groß für den Echtzeiteinsatz.

- **große Farbklötze:** liefert für eine kleine Farbklotzdurchmesser (≈2-5) eine gute Bildqualität bei deutlich reduziertem Stichprobenumfang.
- **Rekonstruktion mit einem Gaußschem Tiefpaßfilter:** liefert sehr gute Ergebnisse mit beliebig geringem Aliasing. Es ist ein größerer Stichprobenumfang notwendig, um niederfrequentes Rauschen zu vermeiden. Hohe Laufzeiten.
- **diskrete Voronoidiagramme:** leichte Verbesserung der Bildqualität gegenüber der Farbklotzmethode, schneller als die Filtermethode.

Die Methode, die für den Einsatz in Echtzeit am erfolgversprechendsten erscheint, ist die Methode mit großen Farbklötzen. Sie bietet einen guten Kompromiß zwischen Aufwand und Qualität und ist einfach zu implementieren, insbesondere wird das Füllen von Farbklötzen mit z-Buffer Test von Softwareschnittstellen zu Renderinghardware direkt unterstützt (siehe Kapitel 4.1). Die diskreten Voronoidiagramme liefern, gemessen am zusätzlichen Aufwand, wenig Gewinn an Bildqualität. Die Filtermethode liefert prinzipiell wesentlich bessere Ergebnisse (vor allem in Bezug auf Aliasing), ist aber noch relativ weit vom Echtzeitbereich entfernt. Die Ausnutzung der Stichprobe bei den beiden letzten Methoden kann durch Verdeckungsheuristiken deutlich verbessert werden, was zu höherer Bildqualität aber zu keiner höheren Verdeckungsauflösung führt. Die in dieser Arbeit untersuchte Tiefentoleranzmethode kann allerdings nicht ohne weiteres für beliebige Szenen angewendet werden.

3.3 Verteilung der Stichprobenpunkte

Im vorangegangenen Abschnitt wurde nur der Aufwand betrachtet, um aus einer nach der projizierten Fläche gleichverteilten Stichprobe von Objektpunkten ein Bild zu rekonstruieren. Die Frage, wie groß der Aufwand für die Auswahl einer solchen Stichprobe ist, wurde bisher nicht gestellt. Sollte dieser stark mit der geometrischen Komplexität wachsen, z.B. in der Größenordnung wie bei bekannten Raytracingalgorithmen, so würde das Verfahren wenig Sinn machen. Im folgenden wird ein Verfahren entwickelt, daß die Punkte in $O(\log n)$ Zeit pro Stichprobenpunkt auf n Dreiecke näherungsweise gleichverteilt verteilen kann und nur O(n) Speicherplatz für vorberechnete Datenstrukturen benötigt. Die Laufzeit wird zusätzlich von weiteren geometrischen Parametern der Gesamtszene abhängen. Im schlimmsten Fall können diese zu einem beliebig großen Aufwand führen, unter Annahmen über eine gleichmäßige Verteilung der Oberflächennormalen in der Szene läßt sich jedoch zeigen, daß der Mehraufwand im durchschnittlichen Fall klein ist.

3.3.1 Approximative Stichprobenauswahl

Eine exakt nach der projizierten Fläche gleichverteilte Auswahl von Stichprobenpunkten in sublinearer Zeit erscheint schwierig: Bei jeder Bewegung des Betrachters ändert sich die projizierte Fläche aller (sichtbaren) Dreiecke. Naheliegende Algorithmen zur exakten Berechnung der projizierten Flächen und anschließender Verteilung nach diesen, würden daher stets einen Zeitaufwand erfordern, der linear mit der Anzahl von Dreiecken wächst.

Um eine Bildrekonstruktion mit einem der beschriebenen Verfahren durchführen zu können, ist allerdings keine exakte Gleichverteilung notwendig, man kann sich mit einer Approximation begnügen: Die Bildqualität wird nicht negativ beeinflußt, wenn in einem Bereich mehr Stichpro-

benpunkte als nötig genommen werden, nur eine Unterschreitung der notwendigen Stichprobendichte kann zu falschen Bildern führen. Wenn die Wahrscheinlichkeitsdichte der tatsächlichen Stichprobenwahl in einem Gebiet um einen Faktor $1/\varepsilon$ mit $\varepsilon > 1$ nach unten von dem durch die projizierte Fläche gegebenen Wert abweicht, führt dies nicht zu Problemen bei der Bildrekonstruktion, wenn insgesamt (oder zumindest in einer Obermenge des abweichenden Gebietes) ε-mal mehr Punkte gewählt werden, da die Punktdichte den notwendigen Wert dann nicht unterschreitet. Wenn die Ungenauigkeit ε zu groß wird, kann der Aufwand durch den größeren Stichprobenumfang stark wachsen.

Im folgenden werden die verschiedenen Faktoren untersucht, die die Flächen bei der Projektion skalieren. Darauf aufbauend werden die Flächenstücke der Szene in Gruppen zusammengefaßt, in denen die Projektionsfaktoren für die Objektpunkte möglichst wenig voneinander abweichen. Wenn entsprechende Gruppen von Objekten bestimmt wurden, werden innerhalb jeder Gruppe die Stichprobenpunkte nur noch nach der absoluten (d.h. nicht projizierten) Fläche auf die Objekte verteilt, wobei die insgesamt verwendete Punktanzahl nach dem maximalen Projektionsfaktor gewählt wird, der innerhalb der Gruppe auftreten kann. Dadurch wird der Stichprobenumfang nie unterschätzt und es ist eine korrekte Bildrekonstruktion sichergestellt.

Es wäre wünschenswert, die Abweichung des Projektionsfaktors in jeder Gruppe maximal auf einen konstanten Faktor ε beschränken zu können. Dies läßt sich allerdings nur mit hohem Aufwand erreichen. Daher wird hier statt dessen versucht, den Fehler soweit zu beschränken, daß die erwartete Gesamtabweichung des Projektionsfaktors in einem „durchschnittlichen" Fall durch einen konstanten Faktor beschränkt bleibt.

3.3.2 Hilfsmittel

3.3.2.1 Der Projektionsfaktor

Um später die Szenenelemente nach ähnlichem Projektionsfaktor gruppieren zu können, soll in diesem Abschnitt zunächst der Einfluß der verschiedenen Parameter auf den Projektionsfaktor untersucht werden.

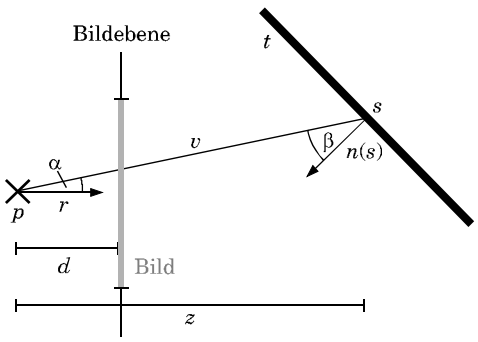


Abbildung 37: Variablen im Projektionsfaktor

Sei eine Szene S sowie ein Punkt $s \in S$ mit der Oberflächennormalen $n(s)$ gegeben. p sei das Projektionszentrum der Zentralprojektion und r die (normierte) Blickrichtung. v sei der Vektor s-p, und z sei die Projektion von v auf r, d.h. $z := v \circ r$. Dann ist der Projektionsfaktor (Definition 6) in s gegeben durch

$$prj(s) = \frac{d^2 \cdot cut_{[0,1]}(\cos\beta)}{z^2 \cdot \cos\alpha}$$

mit β als Winkel zwischen $n(s)$ und v, d als Brennweitenfaktor und α als Winkel zwischen r und v (Abbildung 37). Der Vektor v wird im folgenden auch als *Betrachtungsvektor*, der Winkel β als *Betrachtungswinkel* und der Wert z als *Tiefenwert* bezeichnet.

Einfluß der Tiefe

Seien nun alle Parameter konstant bis auf den Tiefenwert z. Es werden nur Punkte betrachtet, deren Tiefenabstand z vom Projektionszentrum größer als ein Mindestabstand *mindist* ist. Die anderen Parameter seien so vorgegeben, daß der Projektionsfaktor für diese nicht null wird. Sei z_{min} der kleinste und z_{max} der größte Tiefenwert im betrachteten Bereich. Dann gilt:

Die relative Abweichung ist höchstens ε, wenn ${z_{max}}^2 / {z_{min}}^2 \le \varepsilon$ ist, wenn also $z_{max} / z_{min} \le \sqrt{\varepsilon}$ bzw. $z_{max} \le \sqrt{\varepsilon} \cdot z_{min}$ ist.

Einfluß der Orientierung

Sei nun der Tiefenwert und der Winkel α konstant und nur die Betrachtungswinkel β veränderlich. Sei β_{min} der kleinste und β_{max} der größte Winkel in einer Gruppe von Objekten. Weiterhin seien $\beta_{min}, \beta_{max} \in [0, \pi/2]$, d.h. die Oberflächen seien alle dem Betrachter zugewandt. Damit die relative Abweichung des Projektionsfaktors durch den veränderlichen Winkel höchstens ε beträgt, muß gelten:

$$\frac{\cos\beta_{min}}{\cos\beta_{max}} \le \varepsilon$$

Der Kosinus verhält sich in der Nähe von $\pi/2$ annähernd linear mit Steigung -1 und im weiteren Verlauf liegt die Steigung im betrachteten Intervall in [-1, 0]. Daher gilt für $\alpha \in [0, \pi/2]$ und $\delta \in [0, \alpha]$ folgende lineare Abschätzung:

$$\begin{aligned}
\cos(\alpha - \delta) &= \cos\alpha + \int_{\alpha}^{\alpha-\delta} \frac{d}{dx}\cos x\,dx \\
&= \cos\alpha + \int_{\alpha}^{\alpha-\delta} -\sin x\,dx = \cos\alpha + \int_{\alpha-\delta}^{\alpha} \sin x\,dx \\
&\le \cos\alpha + \int_{\alpha-\delta}^{\alpha} 1\,dx \\
&= \cos\alpha + \delta
\end{aligned}$$

Angewandt auf obige Bedingung erhält man mit $\delta := \beta_{max} - \beta_{min}$

$$\frac{\cos\beta_{min}}{\cos\beta_{max}} = \frac{\cos(\beta_{max} - \delta)}{\cos\beta_{max}} \le \frac{\cos\beta_{max} + \delta}{\cos\beta_{max}} = 1 + \frac{\delta}{\cos\beta_{max}},$$

d.h. es ist hinreichend, daß

$$\frac{\delta}{\cos\beta_{max}} \le \varepsilon - 1, \text{ d.h. } \delta \le (\varepsilon - 1)\cos\beta_{max}$$

ist, damit der Projektionsfaktor eingeschränkt auf β um höchstens ε abweicht.

Einfluß der Position auf die Orientierung

Innerhalb eines größeren räumlichen Bereiches werden Oberflächen mit gleicher Orientierung unter einem unterschiedlichen Winkel gesehen, da sich der Winkel zum Betrachter ändert. Um eine Abweichung im Parameter β zu beschränken, muß also auch die räumliche Abweichung eingeschränkt werden. Betrachte einen Objektpunkt s_1 mit Normalvektor n. Sei β_1 der Winkel, den der Vektor n mit $v_1 := s_1\text{-}p$ einschließt. Wenn man s_1 in einen anderen Punkt s_2 verschiebt, erhält man einen neuen Winkel $\beta_2 = \beta_1 + \gamma$, wobei γ der Winkel ist, unter dem der Betrachter die beiden Punkte sieht, d.h. γ ist der Winkel zwischen v_1 und v_2 mit $v_2 := s_2\text{-}p$ (siehe Abbildung 38).

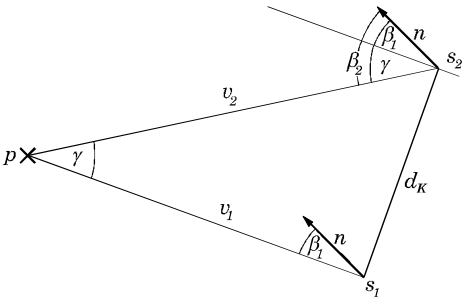


Abbildung 38: Veränderung des Winkels bei verschiedenen Positionen und festem Normalvektor

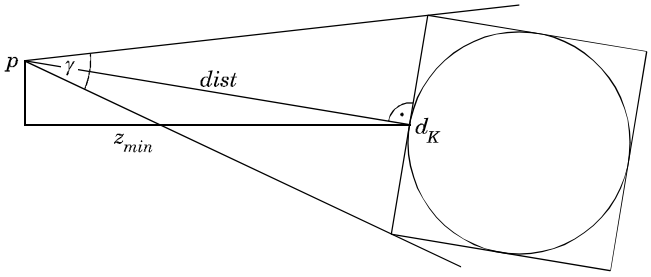


Abbildung 39: Beschränkung des Winkels für eine Gruppe von Objekten

Sei eine Gruppe von Punkten gegeben, die sich in eine Kugel mit Mindesttiefe z_{min} zu p und Durchmesser d_K einschließen läßt. Sei $dist$ die Länge der Strecke zwischen Mittelpunkt der Kugel und p abzüglich $d_K/2$. Die z-Komponente des entsprechenden Vektors wird nie größer als $dist$ und diese ist nie kleiner als z_{min}, daher gilt $z_{min} \le dist$. Der maximale Winkel γ, unter dem zwei Punkte aus dieser Gruppe gesehen werden können, läßt sich daher wie folgt beschränken (siehe Abbildung 39):

$$\gamma \le 2\arctan\left(\frac{d_K/2}{dist}\right) \overset{(1)}{\le} 2\arctan\left(\frac{d_K}{2\cdot z_{min}}\right) \overset{(2)}{\le} \frac{d_K}{z_{min}}$$

Ungleichung (1) nutzt aus, daß der Arcustangens streng monoton steigend ist und Ungleichung (2) gilt, weil der Arcustangens im Intervall [0, ∞) nicht größer als sein Argument ist. Dies erhält man, da für $\alpha \ge 0$ gilt:

$$\arctan(\alpha) = \arctan(0) + \int_0^\alpha \frac{d}{dx}\arctan x\,dx = \int_0^\alpha \frac{1}{1+x^2}dx \overset{\text{für } \alpha\ge 0}{\le} \int_0^\alpha 1dx = \alpha$$

Das Resultat kann nun in die Abschätzung des Winkelverhältnises aus dem vorangegangenen Abschnitt eingesetzt werden, wobei sich die Winkelabweichungen durch unterschiedliche Normalen und unterschiedliche Positionen (im schlimmsten Fall) addieren. Man erhält, daß die Variation des Projektionsfaktors innerhalb einer Gruppe von Objekten nicht größer als ε wird, wenn gilt

$$\frac{2\cdot\arctan\left(\frac{d_K}{2\cdot z_{min}}\right)+\delta}{\cos\beta_{max}} \le \varepsilon - 1, \quad \text{bzw.} \quad \arctan\left(\frac{d_K}{2\cdot z_{min}}\right)+\delta \le \frac{1}{2}(\varepsilon-1)\cos\beta_{max},$$

wobei β_{max} den größten Winkel einer Normalen zum Betrachter innerhalb der Gruppe angibt und δ den maximalen Winkel zwischen zwei Normalen der Gruppe. Nutzt man wiederum aus, daß der Arcustangens im betrachteten Intervall nicht größer als sein Argument ist, erhält man als hinreichende Bedingung:

$$\frac{d_K}{z_{min}}+\delta \le (\varepsilon-1)\cos\beta_{max}.$$

Im folgenden wird daß Verhältnis von Durchmesser d_K zur Mindesttiefe z_{min} einer Objektgruppe als ihr *relativer Durchmesser* bezeichnet. Die Abweichung bleibt also beschränkt, wenn der relative Durchmesser einer Objektgruppe unter einem Wert in $O(\cos\beta_{max})$ bleibt.

Einfluß des Verzerrungsfaktors

Sei nun der Tiefenwert und der Winkel zum Betrachter konstant. Sei α_{max} der maximale Winkel zwischen Blickrichtung und dem Vektor s-p zwischen einem Objektpunkt s und dem Betrachter p. Sei w die Breite und h die Höhe des Bildes in Pixeln, dann ist $D=\sqrt{w^2+h^2}$ der Durchmesser des Bildes in der Bildebene. Da die Bildebene in einem Abstand von d zum Projektionszentrum verläuft, ist der maximale Winkel α_{max} begrenzt durch $\arctan\left(\frac{D}{2d}\right)$.

Der minimale Winkel ist mindestens null. Der Fehler, der durch den Verzerrungsfaktor entsteht, ist daher höchstens

$$\frac{1}{\cos\alpha_{max}} \le \frac{1}{\cos\left(\arctan\frac{D}{2d}\right)}.$$

Wenn die gewünschte Fehlerschranke ε relativ groß und $D/(2d)$ nicht allzu klein ist, kann der Einfluß von α vernachlässigt werden. Typische diagonale Blickwinkel liegen zwischen 30 und 80°, der Wert der einer Normalbrennweite in der Photographie entspricht, liegt etwa bei 45°. In dem Bereich ergibt sich eine Abweichung zwischen 1,15 und 1,3. Wenn ε deutlich größer ist, spielt die Verzerrung eine untergeordnete Rolle. Insbesondere wird der Verzerrungsfaktor im Gegensatz zu den anderen Parametern nie null oder unendlich groß, so daß sich immer ein ε finden läßt, daß den Einfluß begrenzt.

Sei nun eine Gruppe von Objekten angenommen, die in eine Kugel mit Durchmesser d_K und Mindesttiefe z_{min} eingeschlossen werden kann. Sei α_{max} der größte und α_{min} der kleinste Winkel, den die Verbindung eines Punktes aus der Kugel zum Projektionszentrum mit der Blickrichtung einschließt. Dann muß dafür gelten:

$$\frac{\cos\alpha_{min}}{\cos\alpha_{max}} \le \varepsilon, \text{ was erfüllt ist, wenn } \frac{\alpha_{max}-\alpha_{min}}{\cos\alpha_{max}} \le \varepsilon - 1$$

ist (lineare Abschätzung für den Kosinus). Die maximale Winkeldifferenz zwischen α_{max} und α_{min} läßt sich mit der gleichen Konstruktion wie in Abbildung 39 abschätzen, so daß man erhält:

$$\frac{2\arctan\left(\frac{d_K}{2z_{min}}\right)}{\cos\alpha_{max}} \le \varepsilon - 1, \quad \text{bzw.} \quad \arctan\left(\frac{d_K}{2z_{min}}\right) \le \frac{1}{2}(\varepsilon - 1)\cos\alpha_{max}$$

Und da der Arcustangens im betrachteten Intervall nicht größer als sein Argument ist, ist dies sicher erfüllt, wenn gilt:

$$\frac{d_K}{z_{min}} \le (\varepsilon - 1)\cos\alpha_{max}$$

Man erhält also die gleiche Bedingung für den Gruppendurchmesser wie bei der Beschränkung des Einflusses der Position auf die Orientierung. Der Wert α_{max} ist nach oben beschränkt durch den halben Öffnungswinkel der Sichtpyramide $\arctan(D/(2d))$, daher kann die durch den Verzerrungsfaktor bedingte Schranke als konstante Maximalgröße der Objektgruppen angesehen werden. Im folgenden wird daher nur noch die Beschränkung

$$\frac{d_K}{z_{min}} \le (\varepsilon - 1)\cos\varphi \quad \text{mit} \quad \varphi = \max\left(\arctan\left(\frac{D}{2d}\right), \beta_{max}\right)$$

betrachtet. Man erhält dadurch eine Mindestunterteilung der Szene in kleine Gruppen, die dann für Orientierungen nahe $\pi/2$ noch weiter verfeinert werden muß. Der Verzerrungsfaktor wird nicht weiter betrachtet, es wird davon ausgegangen, daß die räumliche Unterteilung in Gruppen so fein vorgenommen wird, daß eine konstante maximale Gruppengröße nicht überschritten wird, so daß obige Bedingung stets für ein geeignetes ε erfüllt ist.

Zusammenfassung

Die Ergebnisse dieses Abschnitts werden durch das folgende Lemma zusammengefaßt:

Lemma 5: *Beschränkung der Abweichung des Projektionsfaktors*

Es sei eine Gruppe von Objekten gegeben, die eine positive Mindesttiefe von z_{min} und Maximaltiefe z_{max} hat, deren Normalvektoren mit dem Sichtvektor vom Objektpunkt zum Projektionszentrum mindestens einen Winkel $\beta_{min} \ge 0$ und höchstens einen Winkel $\beta_{max} < \pi/2$ einschließen und die sich in eine Kugel mit Durchmesser d_K einschließen läßt, die ebenfalls eine Mindesttiefe von z_{min} hat. Weiter sei d der Brennweitenparameter der Zentralprojektion und D der Durchmesser des Bildes in der Bildebene. Dann gilt:

Der Projektionsfaktor eingeschränkt auf die Variable z variiert innerhalb der Gruppe maximal um einen Faktor ε, falls gilt:

$$z_{max} \le \sqrt{\varepsilon} \cdot z_{min}$$

Der Projektionsfaktor eingeschränkt auf die Variable β variiert innerhalb der Gruppe maximal um einen Faktor ε, falls gilt:

$$\beta_{max} - \beta_{min} \le (\varepsilon - 1)\cos\beta_{max}$$

Wenn δ der größte Winkel zwischen zwei Normalvektoren der Gruppe ist, dann variiert der Projektionsfaktor eingeschränkt auf die Variablen α, β innerhalb der Gruppe nur um ε, falls gilt:

$$\frac{d_K}{z_{min}} + \delta \le (\varepsilon - 1)\cos\beta_{max} \quad \text{und} \quad \frac{d_K}{z_{min}} \le (\varepsilon - 1)\cos\arctan\left(\frac{D}{2d}\right)$$

Wenn der Verzerrungsfaktor unberücksichtigt bleiben soll, genügt die erste Bedingung.

Die maximale Differenz γ der Beobachtungswinkel zwischen zwei Punkten mit identischen Normalen innerhalb der Gruppe ist beschränkt durch:

$$\gamma \le 2\arctan\left(\frac{d_K}{2 \cdot z_{min}}\right) \le \frac{d_K}{z_{min}}$$

3.3.2.2 Auswahl von Objekten nach Gewicht

Ein Teilproblem, das im folgenden mehrmals auftritt, ist die Verteilung von Stichproben nach einer vorgegebenen Verteilung. Abstrakt läßt sich das Problem wie folgt modellieren: Es sind n Behälter gegeben, sowie n Gewichte g_i, $i = 1...n$. Nun sollen m Bälle zufällig so auf die n Behälter verteilt werden, daß in jedem Wurf die Wahrscheinlichkeit $p_k = g_k / \sum_{i=1}^{n} g_i$ für Behälter k ist, einen Ball zu erhalten. Vorausgesetzt wird, daß ein Zufallszahlengenerator zur Verfügung steht, der unabhängige, im Intervall [0,1] gleichverteilte Zufallszahlen produzieren kann. Pseudozufallszahlengeneratoren, die solche Zahlen näherungsweise produzieren können, stehen in den meisten Standardbibliotheken von Programmiersprachen und Betriebssystemen zur Verfügung; für die hier benötigten Zwecke reichen diese Generatoren völlig aus.

Nach [Pfanzagl 91] (Seite 27) können mit Hilfe von im Intervall [0,1] gleichverteilten Zufallszahlen Zahlen erzeugt werden, die nach einer beliebigen, invertierbaren Verteilungsfunktion ϕ verteilt sind. Dazu muß die Inverse ϕ^{-1} der Verteilungsfunktion auf die gleichverteilten Zufallszahlen angewendet werden. Dies läßt sich zur Lösung des oben gestellten Problems anwenden: Sei X die Zufallsvariable, die jedem Ballwurf die Nummer des getroffenen Behälters zuordnet. Die Verteilungsfunktion von X ist ϕ_X mit $\phi_X(k) = \sum_{i=1}^{k} g_i$. Sei eine gleichverteilt aus [0,1] gezogenen Zufallszahl z gegeben. Dann muß die Gleichung $\phi_X(i) - z = 0$ nach i aufgelöst werden, um den richtigen Behälter zu erhalten. Diese Aufgabe kann effizient durch binäre Suche gelöst werden: Vorab wird ein Feld von n Werten angelegt; in das i-te Feld wird die Summe der Gewichte $g_1 + ... + g_i$ und ein Verweis auf den i-ten Behälter eingetragen. Diese Vorberechnung benötigt $O(n)$ Zeit. Danach können Behälter gezogen werden, indem für das gegebene z der größte Eintrag in dieser Liste gesucht wird, der nicht größer als z ist. Diese Suche kann als binäre Suche ausgeführt werden, so daß eine Ziehung $O(\log n)$ Zeit benötigt.

Diese Datenstruktur wird im folgenden als *Verteilungsliste* bezeichnet.

3.3.2.3 Auswahl von Punkten auf einem Dreieck

Da die Szene durch Dreiecke beschrieben wird, ist es früher oder später erforderlich, auf einem Dreieck einen zufälligen Punkt auszuwählen. Sei ein Dreieck durch die Koordinaten $a, b, c \in \mathbb{R}^3$ seiner Eckpunkte gegeben. Der folgende Algorithmus wählt einen Punkt zufällig auf der projizierten Fläche des Dreiecks, `rnd[0..1]` bezeichne dabei eine Funktion, die eine (Pseudo-) Zufallszahl im Intervall [0..1] bestimmt:

```
A := projektion(a)
B := projektion(b)
C := projektion(c)

u := rnd[0..1]
v := rnd[0..1]
falls u+v > 1 dann
   h := 1-v
   v := 1-u
   u := h

w := 1 - u - v
ergebnis := u·A + v·B + w·C
```

Algorithmus 7: Wahl eines zufälligen Punktes auf der projizierten Fläche eines Dreiecks

Der Algorithmus projiziert die Koordinaten zuerst in die Bildebene. Danach werden zwei Parameter u und v so bestimmt, daß diese im Dreieck (0,0), (1,0), (0,1) gleichverteilt sind. Dazu werden zuerst die Werte von u und v im Quadrat $[0,1]^2$ gleichverteilt gewürfelt und dann diese an der Geraden durch (1,0) und (0,1) „umgeklappt“, falls die Werte oberhalb dieser Geraden und damit nicht mehr im Dreieck liegen (siehe Abbildung 40).

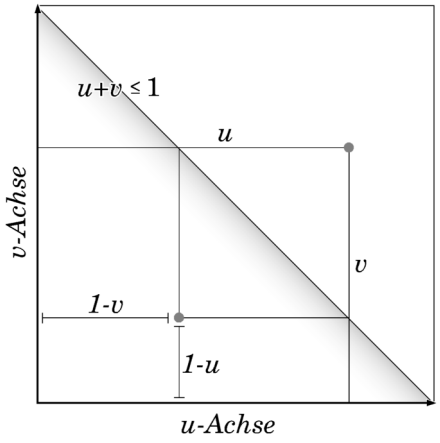


Abbildung 40: Zufällige Wahl von u und v im unteren Dreieck

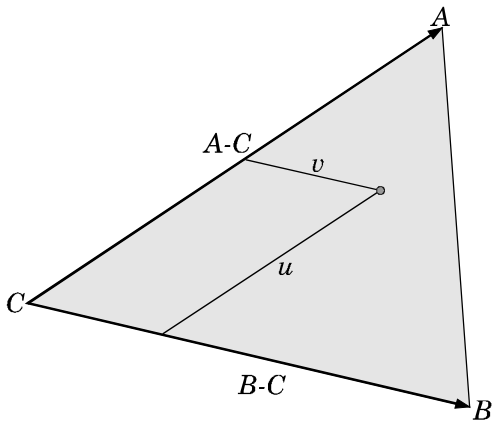


Abbildung 41: Verwendung im Dreieck

Danach wird zum Punkt C der Vektor $A-C$ gewichtet mit u und der Vektor $B-C$ gewichtet mit v hinzuaddiert (Abbildung 41), man erhält

$$u\cdot(A-C)+v\cdot(B-C)+C = u\cdot A-u\cdot C+v\cdot B-v\cdot C = u\cdot A+\mathrm{v}\cdot\mathrm{B}+(1-u-v)\cdot C = u\cdot A+\mathrm{v}\cdot\mathrm{B}+w\cdot C,$$

wie im Algorithmus angegeben.

Beschleunigung

Algorithmus 7 hat den Nachteil, daß für jeden Stichprobenpunkt 3 Transformationen in die Bildebene berechnet werden müssen. Da dies die aufwendigsten Operationen sind, sollten sie aber möglichst selten eingesetzt werden. Daher ist es sinnvoller, die Punkte im dreidimensionalen Raum zu interpolieren und danach den dreidimensionalen Stichprobenpunkt zu transformieren

(im Rahmen dieser Arbeit wurde auch nur dieses Verfahren implementiert). Man erhält damit folgenden Algorithmus:

```
u := rnd[0..1]
v := rnd[0..1]
falls u+v > 1 dann
   h := 1-v
   v := 1-u
   u := h
w := 1 - u - v
ergebnis := projektion(u·A + v·B + w·C)
```

Algorithmus 8: Wahl eines zufälligen Punktes auf der räumlichen Oberfläche eines Dreiecks

Um Algorithmus 8 verwenden zu können, muß vorausgesetzt werden, daß der Projektionsfaktor innerhalb des Dreiecks nicht zu stark variiert, da ansonsten die Punkte zu ungleichmäßig auf der vom Dreieck im Bild belegten Fläche verteilt werden. Da dabei für die Dreiecke prinzipiell dieselben Forderungen hinreichend für eine geringe Variation des Projektionsfaktors sind wie für Gruppen von Objekten, wird eine ungünstige Situation automatisch bei der Verteilung der Punkte auf Dreiecksgruppen erkannt: Wenn eine Gruppe ein zu großes Dreieck enthält, führt dies dazu, daß die Gruppe nicht weiter aufgeteilt werden kann und eine Ausweichstrategie gewählt wird. Die Details für die Strategien in solchen Sonderfällen werden daher in Abschnitt 3.3.6.1 im Zusammenhang mit der Gruppierung von Szenenteilen beschrieben.

Berücksichtigung von Oberflächenattributen

Oft sind an den Eckpunkten von Dreiecken neben den Koordinaten noch weitere Oberflächenattribute, wie Farbe, Texturkoordinaten oder Parameter für Materialeigenschaften gegeben, die innerhalb des Dreiecks linear interpoliert werden sollen. Allgemein seien Vektoren a_{attrib}, b_{attrib}, $c_{attrib} \in \mathbb{R}^k$ gegeben, die die Oberflächenattribute beschreiben, die Dimension k hängt dabei von der Anzahl an Attributen ab. Diese können genauso wie die Koordinaten mit den Faktoren u, v, w linear kombiniert werden. Die Interpolation von Farben mit Algorithmus 7 entspricht vom Ergebnis her der Gouraudschattierung und die Interpolation von Oberflächennormalen mit Algorithmus 8 entspricht der Phongschattierung ([Foley et al. 96]). Im Rahmen dieser Arbeit wurde die Interpolation für Oberflächennormalen, Farben und Materialeigenschaften implementiert. Auf Testsystem 1 (siehe Anhang A) benötigte die Interpolation von Farbwerten etwa 10% zusätzliche Rechenzeit ($k=4$ für RGB und Transparenz). Die Interpolation der Normalen benötigte etwa 2% und die Interpolation von Materialeigenschaften (mit Normalen) ($k=22$) etwa 25% mehr Rechenzeit, wobei diese nur bei dynamisch berechneter Beleuchtung (in diesem Fall gemessen für ein Phong Beleuchtungsmodell [Foley et al. 96] mit zwei Lichtquellen) möglich ist, die ihrerseits auf diesem System eine etwa achtmal höhere Rechenzeit benötigt. Der absolute Aufwand für die Interpolation der Normalen und der Farbwerte ist also, wie erwartet, etwa gleich groß. Die Interpolation von Oberflächenattributen ist also auf Systemen mit rein softwaremäßigem Rendering also mit wenig zusätzlichem Aufwand verbunden. Die Interpolation kompletter Sätze von Materialparametern, die relativ teuer ist, wird aber in der Praxis nur sehr selten benötigt.

3.3.3 Strategien zur Gruppierung der Objekte

In diesem Abschnitt soll kurz eine Übersicht über die weiteren Strategien zur Gruppeneinteilung der Objekte gegeben werden, die in den nächsten Abschnitten ausgearbeitet werden.

Zur Auswahl der Stichprobenpunkte sollen Gruppen von Objekten gebildet werden, innerhalb derer der Projektionsfaktor nur wenig variiert. Im wesentlichen ist der Projektionsfaktor durch den Betrachtungswinkel und die Betrachtungstiefe eines Objektes gegeben. Der Einfluß der Betrachtungstiefe läßt sich dadurch beschränken, daß die Tiefe der Objektgruppen beschränkt wird. Den Einfluß des Winkels zu beschränken, ist dagegen komplexer: Zum einen hängt der Betrachtungswinkel von den Oberflächennormalen ab, so daß die Abweichungen dieser voneinander innerhalb einer Gruppe beschränkt werden müssen. Zum anderen spielt auch die Position des Objektes relativ zum Betrachter eine Rolle, da der Betrachtungswinkel sich auch mit der Position ändert. Um diesen Einfluß zu beschränken, muß die räumliche Ausdehnung der Objektgruppen in einen Kegel um die Beobachtungsrichtung eingeschränkt werden (dies schränkt gleichfalls auch den Einfluß des Verzerrungsfaktors ein). Um die relative Abweichung des Projektionsfaktors bezüglich des Betrachtungswinkels beschränkt zu halten, muß die Beschränkung der Winkelabweichung immer kleiner ausfallen, je näher die Betrachtungswinkel einer Gruppe sich an $\pi/2$ annähern.

Um für jeden Betrachtungsstandpunkt effizient Gruppen von Objekten bilden zu können, in denen der Projektionsfaktor wenig abweicht, wird im folgenden so vorgegangen:

- Der Raum wird mit einem Octree in eine Hierarchie von Gittern eingeteilt, aus der sich würfelförmige, also in alle Raumrichtungen symmetrische Boxen in verschiedenen Größen entnehmen lassen. Der Octree wird in einem Vorberechnungsschritt aufgebaut, die Symmetrie der Boxen garantiert die Verwendbarkeit der vorberechneten Raumunterteilung von jedem Blickpunkt aus.
  Aus dem Octree werden dann dynamisch Boxen ausgewählt, die in der Tiefe so beschränkt sind, daß der Projektionsfaktor bezüglich der Objekttiefe nur um einen konstanten Faktor variiert. Es wird bewiesen, daß die dynamische Auswahl effizient möglich ist.
- Zusätzlich werden die Objekte im Raum in Gruppen ähnlicher Normalenrichtungen gruppiert. Dazu wird entweder ein auf den Polarkoordinaten der Normalen aufgebautes Gitter oder ein darauf aufgebauter Quadtree benutzt. Für jede solche Gruppe wird eine Raumunterteilung mit einem Octree angelegt, aus der dann wie gehabt dynamisch Objektgruppen beschränkter Tiefe ausgewählt werden.
  Für die in jeder Winkelgruppe ausgewählten Octree-Boxen wird mit allen zur Verfügung stehenden Informationen über Winkel, Position und Tiefe der Projektionsfaktor nach oben abgeschätzt und als konstant über die Box angenommen.

Um mit dieser Strategie den Projektionsfaktor soweit einzuschränken, daß er auch im schlimmsten Fall nur um einen konstanten Faktor vom richtigen Wert abweicht, ist eine so feine Unterteilung sowohl räumlich wie auch im Winkelbereich notwendig, daß das Verfahren zu ineffizient wird. Daher muß mit einer gröberen Schätzung gearbeitet werden. Diese garantiert nur die maximal konstante Abweichung bezüglich der Tiefe und vernachlässigt den Winkel völlig bzw. stellt nur eine grobe Schätzung des Winkeleinflusses an. Bei einem solchen Vorgehen sind natürlich ungünstige Szenen mit stark erhöhtem Stichprobenumfang durch eine falsche Schätzung des Projektionsfaktors möglich.

Für die Analyse wird daher angenommen, daß die Normalvektoren der Szenenelemente rein zufällig auftreten und gleichverteilt auf der Einheitskugel sind. Unter diesen Voraussetzungen läßt sich zeigen, daß alleine mit der räumlichen Unterteilung zur Beschränkung der Tiefe ein erwarteter mittlerer Fehler des Projektionsfaktors entsteht, der nur um eine Konstante vom exakten Wert abweicht. Die Konstante läßt sich dann, wie sich weiter zeigt, durch die zusätzliche Bildung von Gruppen ähnlicher Winkel auf den Polarkoordinaten noch verbessern.

Eine Stichprobenauswahl mit konstanten Fehlerschranken für die Stichprobendichte im Bild erscheint, ähnliche wie eine exakte Lösung, kaum möglich. Mit dem hier gewählten Ansatz müßte eine zu große Anzahl von Boxen für eine exakte Approximation gebildet werden. Ob andere Strategien das Problem besser lösen könnten, bleibt offen. Die Tatsache, daß eine Vernachlässigung der Betrachtungswinkel nicht unbedingt zu großen Fehlern führt, ist daher entscheidend dafür, daß das randomisierte z-Buffer Verfahren in der hier vorgestellten Form überhaupt realisiert werden konnte.

Im folgenden Abschnitt (3.3.4) wird zunächst die Raumunterteilung vorgestellt und analysiert und danach werden verschiedene Strategien zur Berücksichtigung der Winkelverhältnisse beschrieben (3.3.5). Im darauffolgenden Abschnitt (3.3.6) wird dann noch auf einige Sonderfälle und Erweiterungen eingegangen.

3.3.4 Raumunterteilung

3.3.4.1 Adaptive Unterteilung

Die Position der Objekte im Raum hat einen starken Einfluß auf die projizierte Fläche. Um dies zu kompensieren, soll der Raum nun in Zellen eingeteilt werden, innerhalb derer die Abweichung des Projektionsfaktors durch unterschiedliche Entfernungen nur noch maximal einen Unterschied von einem Faktor ε zwischen dem größten und dem kleinsten Wert annimmt. Dazu sei hier stets angenommen, daß eine Mindesttiefe *mindepth* vorgegeben ist, die ein Objektpunkt haben muß, um dargestellt werden zu können. Diese Annahme ist bei der Zentralprojektion stets notwendig, da die Projektion für den Tiefenwert $z = 0$ singulär wird, die projizierte Fläche strebt bei Annäherung an null gegen unendlich. Für eine feste Szene und einen festen Beobachterstandpunkt gibt es einen maximalen Tiefenwert *maxdepth*. Das Verhältnis dieser beiden Werte wird im folgenden als Tiefendynamik bezeichnet und hat Einfluß auf die Laufzeit. Offensichtlich ist die Tiefendynamik jeder Szene unabhängig vom Standpunkt durch das Verhältnis von Szenendurchmesser zu *mindepth* nach oben beschränkt.

Definition 11: *Tiefendynamik*

Sei *mindepth* der minimale Tiefenwert, für den Punkte aus der Szene noch dargestellt werden. Sei *maxdepth* der maximale Tiefenwert, den ein Objektpunkt der Szene für einen Beobachterstandpunkt (Projektionszentrum) p aufweist. Dann heißt

$$\tau(p) := \frac{maxdepth}{mindepth}$$

die *Tiefendynamik der Szene bezüglich des Standpunktes* p. Sei $d(S)$ der Durchmesser der Szene. Dann bezeichne $\tau := d(S) / mindepth$ die *maximale Tiefendynamik*.

Nach Lemma 5 darf das Verhältnis der größten zur kleinsten Tiefe in jeder Gruppe von Objekten den Wert $\varepsilon' := \sqrt{\varepsilon}$ nicht überschreiten, damit der Projektionsfaktor um nicht mehr als den Wert

ε innerhalb der Zelle variiert. Eine naheliegende Idee wäre daher, den Raum, den die Szene einnimmt, durch ein Gitter mit Gitterweite $\varepsilon' \cdot mindepth$ in Boxen einzuteilen. Dies würde allerdings im schlimmsten Fall die Bearbeitung von $\Theta(\tau^3)$ Boxen erfordern, was für Szenen mit größerer Ausdehnung nicht praktikabel erscheint. Durch eine hierarchische Datenstruktur läßt sich dieser Aufwand auf $O(\log \tau)$ verringern. Dazu wird eine hierarchische Unterteilung in Boxen benutzt, die es erlaubt, die Größe der Boxen ihrer Entfernung anzupassen:

Definition 12: *Quadtree und Octree*

Ein *d-dimensionaler Quadtree* (schlicht *Quadtree* im Fall $d=2$, *Octree* im Fall $d=3$) ist eine 2^d-ärer Baum, der den $\mathbb{R}^d$ wie folgt in achsenparallele Boxen aufteilt: Jedem Knoten k ist ein achsenparalleler d-dimensionaler Quader $Q(d)$ zugeordnet, jeder Knoten hat entweder keine oder genau 2^d Kindknoten. Die Boxen eines Kindknoten teilen den Elternquader in ein regelmäßiges Gitter auf, das dadurch entsteht, daß man den Elternknoten in allen Dimensionen genau in der Mitte teilt. Alle Quader seien halboffen, d.h. das kartesische Produkt von Intervallen $[a_i, b_i)$, $i=1...d$.

Jedem Knoten k kann eine Menge von geometrischen Objekten $Obj(k)$ zugeordnet sein. Dabei ist einem Elternknoten stets die Vereinigungsmenge aller den Kindknoten und ihm selber zugeordneten Objekte zugeordnet. $B(k)$ bezeichne den minimalen, achsenparallelen, geschlossenen Quader, der alle k zugeordneten Objekte vollständig enthält.

Quadtrees in verschiedenen Varianten werden ausführlich in [Samet 90] beschrieben. Definition 12 läßt noch offen, wie die Objekte, d.h. hier die Dreiecke der Szene, den Knoten des Baumes zugeordnet werden. Hier sei zunächst angenommen, daß alle Dreiecke so klein sind, daß sie als Punkte aufgefaßt werden können. Dann wird der *Raumunterteilungsbaum* wie folgt definiert:

Definition 13: *(einfacher) Raumunterteilungsbaum*

Sei eine Menge S von punktförmigen Objekten im $\mathbb{R}^3$ gegeben. Ein *Raumunterteilungsbaum für S* ist ein Octree, dessen Wurzel ein minimaler Würfel zugeordnet ist, in dem S vollständig enthalten ist und in dem jedem Blattknoten k die Objekte aus S zugeordnet sind, die vollständig in $Q(k)$ enthalten sind. Die Blätter haben eine minimale Tiefe, bei der jedes Blatt nur ein Objekt enthält.

In Abschnitt 3.3.6.1 werden Strategien, die mit beliebigen Szenen aus beliebigen Dreiecken arbeiten können, und die dabei auftretenden Einschränkungen erläutert. Zunächst soll für den eingeschränkten Fall gezeigt werden, daß aus einem solchen Baum eine geringe Anzahl von Knoten ausgewählt werden kann, die den sichtbaren Teil der Szene enthalten und in denen der Projektionsfaktor bezüglich des Tiefenwertes jeweils um nicht mehr als ε variiert.

Satz 6: *Gruppierung mit einem Raumunterteilungsbaum bezüglich des Tiefenwertes (mit Clipping)*

Sei eine Szene S aus punktförmigen Objekten und ein Raumunterteilungsbaum T dafür gegeben. Dann lassen sich für jeden Beobachterstandpunkt $O(\log \tau)$ Knoten $k_1...k_n$ aus T in Zeit $O(\log \tau)$ bestimmen, so daß der Projektionsfaktor eingeschränkt auf den Tiefenwert z innerhalb von $B(k_i)$ nur um eine Faktor ε variiert und daß $F \subseteq Obj(k_1) \cup ... \cup Obj(k_n)$ gilt, mit $F := \{x \in S \mid frustum(x) = 1\}$ als Sichtpyramide.

Beweis: Sei im folgenden wieder $\varepsilon' := \sqrt{\varepsilon}$. Teile nun die Sichtpyramide nach Tiefenwerten in Bereiche auf, in denen der Projektionsfaktor höchstens um ε variiert, d.h. in denen der Tiefenwert nicht um mehr als ε' variiert (Abbildung 42): Der erste Bereich beginnt bei $z_0 := mindist$ und endet bei $z_1 := \varepsilon' \cdot mindist$. Der zweite Bereich beginnt bei z_1 und endet bei $z_2 := \varepsilon'^2 \cdot mindist$. Bereich n beginnt bei $z_{n-1} := \varepsilon'^{n-1} \cdot mindist$ und endet bei $z_n := \varepsilon'^n \cdot mindist$. Der letzte Bereich ist derjenige, der mit einer Tiefe kleiner als $maxdepth$ beginnt und einer Tiefe größer als $maxdepth$ endet. D.h. die Anzahl der Bereiche n ist gegeben durch den kleinsten Wert, für den $\varepsilon'^n \cdot mindist \geq maxdist$ gilt, also $\varepsilon'^n \geq \tau(p)$. Dies ist der Wert $\lceil \log_{\varepsilon'} \tau(p) \rceil \in O(\log \tau / \log \varepsilon') = O(\log \tau)$. Die Anzahl von Bereichen ist somit logarithmisch in der Tiefendynamik. Nun wird noch gezeigt, daß sich jeder Bereich mit einer konstanten Anzahl von Boxen aus dem Octree überdecken läßt und daß diese effizient bestimmt werden können.

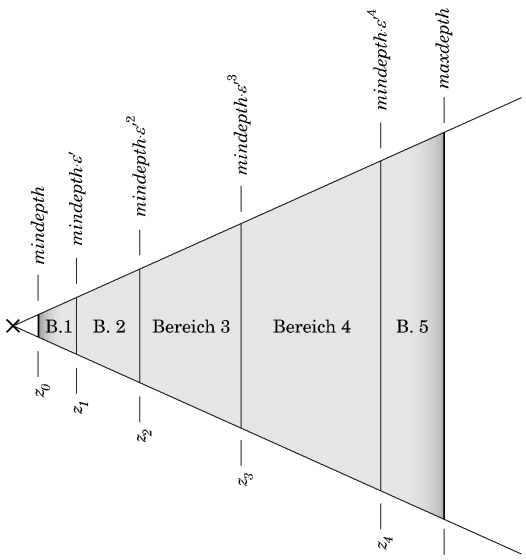


Abbildung 42: Tiefenbereiche

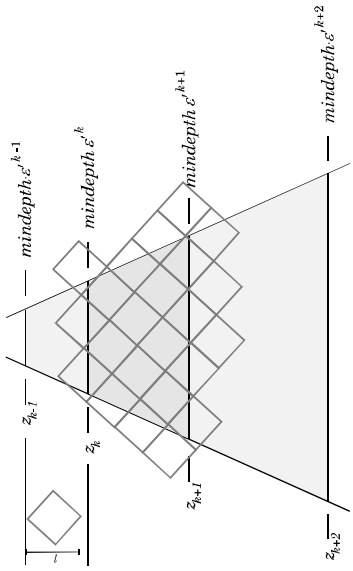


Abbildung 43: Überdeckung eines Tiefenbereiches mit Octreeboxen

Überdeckung eines Tiefenbereiches mit konstant vielen Boxen: Betrachte nun einen festen Tiefenbereich k. Dieser soll mit Boxen aus dem Raumunterteilungsbaum so überdeckt werden, daß keine Box einen Tiefenbereich schneidet, der eine kleinere Tiefe hat, als die maximale Tiefe der Box, die durch ihre Diagonale beschränkt ist. Nach Definition gilt für einen (einfachen) Raumunterteilungsbaum offensichtlich für jeden Knoten x: $B(x) \subseteq Q(x)$, daher wird die Überdeckung mit den Würfeln $Q(x)$ vorgenommen. Diese Würfel stehen in Größen jeder Zweierpotenz in dem festen Raster dieser Zweierpotenzen zur Verfügung, solange nur eine Größe gleichzeitig verwendet wird (der Einfluß der verschiedenen Größen für die verschiedenen Bereiche wird später berücksichtigt). Wenn eine Box nicht weiter verkleinert werden kann, weil sie zu einem Blattknoten gehört, so kann dieser Blattknoten ohne Einschränkungen verwendet werden, da er nur ein Objekt enthält, dessen Größe nach Voraussetzung verschwindend klein ist. Daher variiert der Projektionsfaktor innerhalb des Objektes nicht stark. Wenn eine zu große Box nötig ist, wird statt dessen die Wurzel des Baumes verwendet und wenn eine Box außerhalb des Octrees benötigt wird, so wird dies ignoriert. Diese Randbedingungen verringern höchstens die Anzahl von Boxen, die benötigt werden, und auch den Aufwand, um diese zu bestimmen. Im folgenden kann also o.B.d.A angenommen werden, daß Tiefenbereich k vollständig durch Boxen in Größen jeder Zweierpotenz in dem festen Raster dieser Zweierpotenzen überdeckt werden kann.

Um Tiefenbereich k zu überdecken, werden nur Boxen verwendet, die die größte mögliche Größe haben, so daß ihr diagonaler Durchmesser kleiner als die Tiefe von Tiefenbereich $k-1$ ist, also kleiner als $z_{k-1} - z_{k-2} = \varepsilon'^{k-1} \cdot mindist - \varepsilon'^{k-2} \cdot mindist = \varepsilon'^{k-2} \cdot mindist \cdot (\varepsilon' - 1)$ ist. Solche Boxen passen in jeder Orientierung vollständig in jeden der Bereiche $k-1$, k und $k+1$, daher verletzen auch Boxenteile, die über den Bereich k hinausgehen nicht die Tiefenbedingung (Abbildung 43). Der Durchmesser[13] dieser Boxen ist mindestens $q := (z_{k-1} - z_{k-2}) / 2 = \varepsilon'^{k-2} \cdot mindist \cdot (\varepsilon' - 1) / 2$ und höchstens $2q$, da Boxen in der Größe jeder Zweierpotenz zur Verfügung stehen. Die Kantenlänge der Boxen ist nochmals um den Faktor $\frac{1}{3}\sqrt{3}$ kleiner. Das Volumen einer Box ist daher mindestens

$$V_{Box} = \left(\frac{q}{\sqrt{3}}\right)^3 = \frac{1}{9}\sqrt{3}q^3 = \frac{1}{72}\sqrt{3}\left(\frac{z_{k-1} - z_{k-2}}{2}\right)^3$$

$$= \frac{1}{72}\sqrt{3}\left(\frac{\varepsilon' z_{k-2} - z_{k-2}}{2}\right)^3 = \frac{1}{72}\sqrt{3} \cdot z_{k-2}{}^3\left(\frac{\varepsilon' - 1}{2}\right)^3.$$

In der zweiten Zeile wurde benutzt, daß für alle $k \geq 2$ gilt, daß $z_k = \varepsilon' z_{k-1}$ ist.

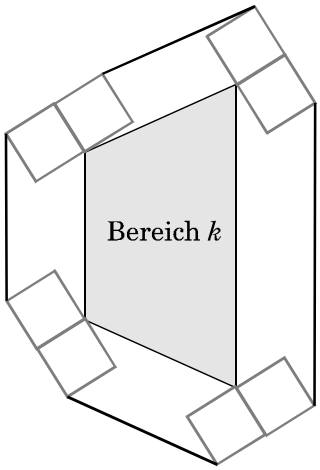


Abbildung 44: Erweiterung des Sichtkegels

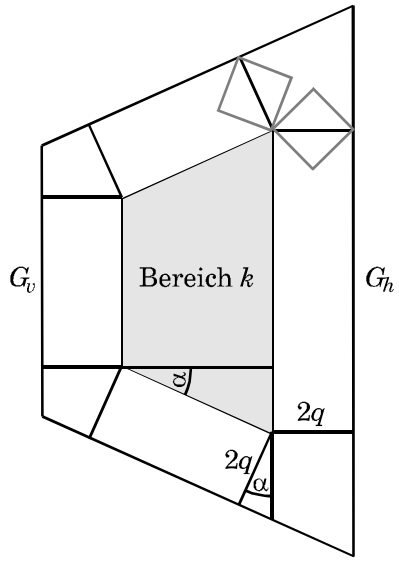


Abbildung 45: Obermenge der Erweiterung

Um die Anzahl von Boxen zu beschränken, die zur Überdeckung des Tiefenbereiches k notwendig sind, wird nun ein erweiterter Bereich so konstruiert, daß der Tiefenbereich dann vollständig überdeckt ist, wenn keine weiteren Boxen mehr in den erweiterten Bereich aufgenommen werden können, ohne teilweise außerhalb des erweiterten Bereiches zu liegen. Dann läßt sich die Anzahl der Boxen einfach durch das Volumenverhältnis des erweiterten Bereiches und der Boxen nach oben beschränken.

Sei w die Breite und h die Höhe des Bildes, $a = w \cdot h$ dessen Fläche und d die Brennweite der Zentralprojektion. Die Sichtpyramide hat im Abstand d eine Querschnittsfläche[14] von a. Der Bereich k ist ein Ausschnitt in Form eines Pyramidenstumpfes aus der Sichtpyramide, im Abstand z

[13] Mit Durchmesser wird hier immer die Länge der Diagonale bezeichnet.
[14] Die Projektion wurde so konstruiert, daß die Kantenlänge eines Pixels genau 1 ist.

beträgt die Querschnittsfläche also $a\left(\frac{z}{d}\right)^2$. Der Pyramidenstumpf von Bereich k wird nun um alle Punkte im Raum erweitert, die in einer verschobenen Octreebox mit Durchmesser $2q$ liegen, die Bereich k schneidet (siehe Abbildung 44). Sei m die maximale Anzahl Boxen aus dem Octree, die vollständig innerhalb des erweiterten Bereiches liegen. Diese haben mindestens den Durchmesser q und höchstens den Durchmesser $2q$. Die Boxen überdecken vollständig den nicht erweiterten Bereich k. *Beweis*: Falls Bereich k in einem Punkt x nicht überdeckt wird, so kann die Box, die x enthält, zu den Boxen hinzugefügt werden, da diese den erweiterten Bereich nach Konstruktion nicht überschreiten kann. Dies widerspräche der Maximalität von m.

m ist nach oben beschränkt durch das Verhältnis des Volumens des erweiterten Bereiches zum Volumen der überdeckenden Boxen, da diese alle disjunkt sind. Um das Volumen des erweiterten Bereiches zu beschränken, wird eine einfache Obermenge des erweiterten Bereiches gebildet (Abbildung 45): Die Begrenzungsebenen des Pyramidenstumpfes werden um $2q$ parallel nach außen verschoben. Dieses Gebilde hat eine Tiefe von $z_k - z_{k-1} + 4q$. Sei α_w der halbe Öffnungswinkel der Sichtpyramide in der Horizontalen und α_h der halbe Öffnungswinkel in der Vertikalen (siehe Abbildung 45, dort in der zweidimensionalen Ansicht nur mit α bezeichnet). Die hintere Grundfläche des entstehenden erweiterten Pyramidenstumpfes (die mit größerer Tiefe) hat dann eine Fläche von:

$$\begin{aligned} G_h &= \left(w\frac{z_k}{d} + \frac{4q}{\cos\alpha_w}\right)\left(h\frac{z_k}{d} + \frac{4q}{\cos\alpha_h}\right) \\ &= \frac{wh}{d^2}{z_k}^2 + \frac{4w}{d\cos\alpha_h}z_k q + \frac{4h}{d\cos\alpha_w}z_k q + \frac{16}{\cos\alpha_h\cos\alpha_w}q^2 \\ &= \underbrace{\frac{wh}{d^2}}_{A}{z_k}^2 + \underbrace{\frac{4}{d}\left(\frac{w}{\cos\alpha_h} + \frac{h}{\cos\alpha_w}\right)}_{B}z_k q + \underbrace{\frac{16}{\cos\alpha_h\cos\alpha_w}}_{C}q^2 \end{aligned}$$

Um die Notation zu vereinfachen, werden die konstanten Koeffizienten im folgenden mit A, B und C bezeichnet und man erhält:

$$G_h = A{z_k}^2 + Bz_k q + Cq^2$$

Die vordere Grundfläche (die mit kleinerer Tiefe) hat in dieser Schreibweise eine Fläche von

$$\begin{aligned} G_v &= \left(w\frac{z_{k-1}}{d} + \frac{4q}{\cos\alpha_w}\right)\left(h\frac{z_{k-1}}{d} + \frac{4q}{\cos\alpha_h}\right) \\ &= A{z_{k-1}}^2 + Bz_{k-1}q + Cq^2 \end{aligned}$$

Das Volumen des erweiterten Pyramidenstumpfes ergibt sich damit als:

$$\begin{aligned} V_{erw} &= \frac{G_v + G_h}{2}\,Tiefe \\ &= \frac{\left(A{z_{k-1}}^2 + Bz_{k-1}q + Cq^2\right) + \left(A{z_k}^2 + Bz_k q + Cq^2\right)}{2}\left(z_k - z_{k-1} + 2q\right) \\ &= \left(A\frac{{z_{k-1}}^2 + {z_k}^2}{2} + Bq\frac{z_{k-1} + z_k}{2} + Cq^2\right)\left(z_k - z_{k-1} + 2q\right) \end{aligned}$$

Nun wird die Rekursion $z_k = \varepsilon' z_{k-1}$ und die Abschätzung $q \le (z_k - z_{k-1})/2$ eingesetzt, und man erhält:

$$V_{erw} \le \left(A\frac{{z_{k-1}}^2 + {z_k}^2}{2} + B\frac{z_{k-1} - z_{k-2}}{2} \cdot \frac{z_{k-1} + z_k}{2} + C\left(\frac{z_{k-1} - z_{k-2}}{2}\right)^2 \right)\left(z_k - z_{k-1} + 2\frac{z_{k-1} - z_{k-2}}{2} \right)$$

$$= \left(A\frac{{z_{k-1}}^2 + {z_k}^2}{2} + B\frac{z_{k-1} - z_{k-2}}{2} \cdot \frac{z_{k-1} + z_k}{2} + C\left(\frac{z_{k-1} - z_{k-2}}{2}\right)^2 \right)\left(z_k - z_{k-2} \right)$$

$$= \left(A\frac{\left(\varepsilon' z_{k-2}\right)^2 + \left({\varepsilon'}^2 z_{k-2}\right)^2}{2} + B\frac{\varepsilon' z_{k-2} - z_{k-2}}{2} \cdot \frac{\varepsilon' z_{k-2} + {\varepsilon'}^2 z_{k-2}}{2} + C\left(\frac{\varepsilon' z_{k-2} - z_{k-2}}{2}\right)^2 \right)\left({\varepsilon'}^2 z_{k-2} - z_{k-2} \right)$$

$$= \left(A{z_{k-2}}^2\frac{{\varepsilon'}^2 + {\varepsilon'}^4}{2} + B{z_{k-2}}^2\frac{\varepsilon' - 1}{2} \cdot \frac{\varepsilon' + {\varepsilon'}^2}{2} + C{z_{k-2}}^2\left(\frac{\varepsilon' - 1}{2}\right)^2 \right) z_{k-2}\left({\varepsilon'}^2 - 1 \right)$$

$$= {z_{k-2}}^3\left(A\frac{{\varepsilon'}^2 + {\varepsilon'}^4}{2} + B\frac{{\varepsilon'}^2 + {\varepsilon'}^3 - \varepsilon' - {\varepsilon'}^2}{4} + C\frac{{\varepsilon'}^2 - 2\varepsilon' + 1}{4} \right)\left({\varepsilon'}^2 - 1 \right)$$

Das Volumen der Boxen, mit denen der erweiterte Bereich überdeckt werden soll, ist mindestens $V_{Box} = \frac{1}{72}\sqrt{3} \cdot {z_{k-2}}^3\left(\frac{\varepsilon' - 1}{2}\right)^3$. Man erhält als Volumenverhältnis:

$$\frac{V_{erw}}{V_{Box}} = \frac{{z_{k-2}}^3\left(A\frac{{\varepsilon'}^2 + {\varepsilon'}^4}{2} + B\frac{{\varepsilon'}^2 + {\varepsilon'}^3 - \varepsilon' - {\varepsilon'}^2}{4} + C\frac{{\varepsilon'}^2 - 2\varepsilon' + 1}{4} \right)\left({\varepsilon'}^2 - 1 \right)}{\frac{1}{72}\sqrt{3}{z_{k-2}}^3\left(\frac{\varepsilon' - 1}{2}\right)^3}$$

$$= \frac{\left(A\frac{{\varepsilon'}^2 + {\varepsilon'}^4}{2} + B\frac{{\varepsilon'}^2 + {\varepsilon'}^3 - \varepsilon' - {\varepsilon'}^2}{4} + C\frac{{\varepsilon'}^2 - 2\varepsilon' + 1}{4} \right)\left({\varepsilon'}^2 - 1 \right)}{\frac{1}{72}\sqrt{3}\left(\frac{\varepsilon' - 1}{2}\right)^3} \in \Theta\left(\frac{{\varepsilon'}^8}{{\varepsilon'}^3}\right) = \Theta\left({\varepsilon'}^5\right)$$

$$= \Theta\left(\varepsilon^{5/2}\right)$$

Das Verhältnis hängt nur noch von den Parametern der Zentralprojektion und von ε' ab und nicht von k. Folglich ist es für alle Tiefenbereiche konstant. Damit erhält man für alle Tiefenbereiche eine konstante obere Schranke für die Anzahl von Boxen, die zur Überdeckung des Bereiches benötigt werden.

Überdeckung mehrerer Bereiche: Das bisherige Ergebnis begrenzt allerdings noch nicht die Gesamtanzahl von Boxen, da nicht berücksichtigt wurde, daß disjunkte Boxen verschiedener Größe nicht wahllos aus dem Octree entnommen werden können. Wenn in dem Bereich einer großen Box schon eine kleinere Box gewählt wurde, so muß die große Box aufgeteilt werden. Da dies rekursiv angewendet werden muß, erhält man eine Situation wie in Abbildung 46:

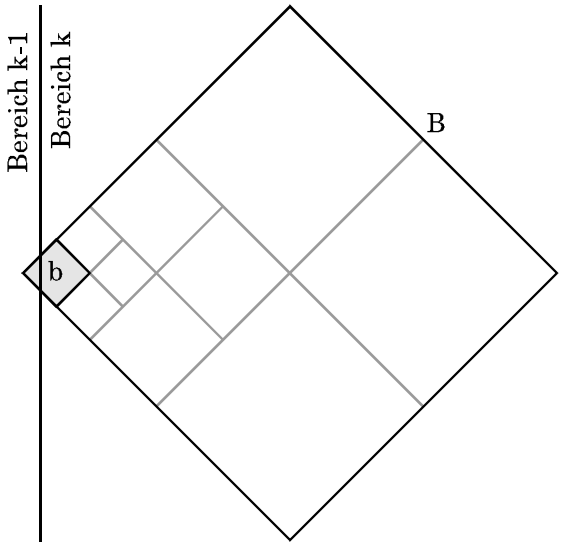


Abbildung 46: Einfluß von variierenden Boxengrößen

Wenn also innerhalb einer Box B in Tiefenbereich k noch eine kleinere Box b aus dem Tiefenbereich k-1 enthalten ist, so muß B über mehrere Ebenen wie in Abbildung 46 gezeigt aufgeteilt werden; für jede Ebene kommen nur konstant viele neue Boxen hinzu. Boxen aus einem noch kleineren Tiefenbereich k-2 oder kleiner müssen nicht berücksichtigt werden, die von ihnen verursachten Kosten werden entsprechend den Boxen aus k-1 und kleiner zugeordnet. Die Anzahl von Ebenen wächst mit dem Zweierlogarithmus des Größenunterschiedes der Boxen, also mit $O(\log \varepsilon')$. Man erhält also $O(\log \tau / \log \varepsilon' \cdot \varepsilon'^5 \cdot \log \varepsilon') = O(\log \tau)$ Boxen, wie behauptet.

Berechnung der Boxen: Der folgende Algorithmus wählt, gestartet bei der Wurzel des Baumes, Knoten im Blickfeld aus, in denen der Projektionsfaktor um höchstens ε variiert. Dazu teilt er rekursiv alle Knoten, die im Blickfeld liegen und noch zu groß sind, in ihre Kindknoten auf. Bei Blattknoten ist keine Überprüfung notwendig, da diese nach Voraussetzung klein sind.

```
Algorithmus wähleKnoten(k: Knoten im Raumunterteilungsbaum)
falls B(k) ∩ F ≠ ∅ dann
   sei z_min die kleinste und z_max die größte Tiefe in B(k)
   falls (z_max / z_min < ε' oder k ist Blattknoten) und z_min > 0 dann
      wähle Knoten k aus
   sonst
      falls k kein Blattknoten dann
         für alle Kinder c von k:
            wähleKnoten(c)
```

Algorithmus 9: Auswahl der Knoten nach dem Tiefenwert

Da die Knoten schrittweise verkleinert werden, wird eine Menge optimaler Größe von Knoten bestimmt, die diese Eigenschaft hat. Nach obigen Überlegungen hat diese Menge höchstens eine Größe von $O(\log \tau)$, denn dort wurde eine Menge dieser Größe konstruiert, bei der der Algorithmus spätestens die Tiefensuche stoppen muß. Zu zeigen ist noch, daß die Auswahl von Knoten nicht mehr als $O(\log \tau)$ Zeit benötigt.

Falls der gesamte Raumunterteilungsbaum im Blickfeld liegt, so ist dies einfach zu sehen: In allen Knoten werden stets alle acht Nachfahren weiterverfolgt oder der Knoten wird ausgegeben. Der so entstehende Teilbaum hat in jedem inneren Knoten Grad acht. Für jeden besuchten Knoten, der nicht ausgegeben wird, werden daher später mindestens acht Blätter ausgegeben. Somit ist die Anzahl der Blätter stets größer als die Anzahl der besuchten Knoten, die nicht ausgegeben werden, und diese ist daher auch durch $O(\log \tau)$ beschränkt.

Falls nicht der gesamte Raumunterteilungsbaum im Blickfeld liegt, kann es aber passieren, daß in einigen Stellen im Algorithmus nur jeweils ein Kindknoten weiterverfolgt wird, so daß der Aufwand nicht durch diese Argumentation beschränkt werden kann. Fasse nun die zuvor definierten Tiefenbereiche zusammen, so daß F_k die Vereinigung der Tiefenbereiche $1...k$ angibt.

Betrachte alle Boxen der Tiefe t, die vom Algorithmus weiter unterteilt werden und bezeichne den diagonalen Durchmesser dieser Boxen mit q. Sei k für dieses t minimal gewählt, so daß die Boxengröße, die in Tiefenbereich k zulässig ist, nicht kleiner als q ist. k sei also minimal unter der Bedingung $q \leq \varepsilon'^{k-2} \cdot mindist \cdot (\varepsilon' - 1)$. Die Anzahl von Ebenen im Baum (also von Werten t), für die der gleiche Wert k gewählt wird, ist in $O(\log \varepsilon')$, also konstant: Die zulässigen Boxengrößen nehmen jeweils um den konstanten Faktor ε' zu, wenn k um eins erhöht wird. Um diesen Faktor durch Halbieren bzw. Verdoppeln des Durchmessers zu erreichen sind nur $\lceil \log_2 \varepsilon' \rceil$ Schritte notwendig, d.h. es können nur $O(\log \varepsilon')$ Ebenen im Baum zum gleichen k führen.

Damit die betrachtete Box vom Algorithmus weiter unterteilt wird, muß sie F_{k-1} schneiden, da sie sonst entweder außerhalb der Sichtpyramide liegen müßte oder bereits klein genug ist, um direkt ausgewählt zu werden.

Für Boxen mit dem Durchmesser q gilt, wie bereits zuvor gezeigt wurde, daß der Tiefenbereich k durch konstant viele von ihnen überdeckt werden kann. Für alle Boxen, deren Größe nur um einen konstanten Faktor von q abweicht, gilt dies ebenfalls: Wenn das zuvor konstruierte erweiterte Volumen des Bereiches nur um eine Konstante größer ist als das von Boxen der Größe q, so gilt dies auch für Größen von Boxen, die nur um eine Konstante von q abweichen. Daher können konstant viele Boxen der Tiefe t den Tiefenbereich k überdecken. Betrachte nun die erweiterten Volumina der Tiefenbereiche $1...k-1$. Da das Volumen eines erweiterten Bereiches mit $(z_{k-2})^3 \in \Theta(\varepsilon'^{3k})$ zunimmt, wobei $\varepsilon' > 1$ ist, erhält man eine geometrische Reihe, die die Summe der erweiterten Volumina der Tiefenbereiche $1...k-1$ so beschränkt, daß diese nicht um mehr als einen konstanten Faktor nach oben von dem erweiterten Volumen von Tiefenbereich k abweichen können.

Die Folgerung ist, daß eine konstante Anzahl von Boxen der Tiefe t ausreicht, um F_{k-1} zu überdecken. Außerdem werden nur konstant viele verschiedene Tiefen t bei obiger Konstruktion auf das selbe k abgebildet, das heißt, jedes F_k wird nur konstant oft für die Überdeckungen von Boxen verschiedener Tiefe t benutzt. Die Anzahl verschiedener Bereiche F_k ist beschränkt; k ist eine natürliche Zahl und nach oben durch eine Konstante in $O(\log \tau)$ beschränkt. Die gesamte Anzahl von Boxen, die im Algorithmus weiter aufgeteilt werden, ist daher durch $O(\log \tau)$ beschränkt. Insgesamt werden maximal achtmal so viele besucht. Die Anzahl der besuchten Boxen ist also in $O(\log \tau)$ und damit auch der maximale Gesamtaufwand des Algorithmus, da jede Box nur genau einmal besucht wird. ■

[Chamberlain et al. 95] bzw. [Chamberlain et al. 96] verwenden ein ähnliches Beweisprinzip, um die Laufzeit der dort beschriebenen Methode zur beschränken.

Genauigkeit des *„viewing frustum culling“*

Algorithmus 9 führt bei der Auswahl der Boxen gleichzeitig ein *„viewing frustum culling“* durch, d.h. Szenenteile, die außerhalb des Sichtkegels liegen, werden abgeschnitten. Diese Aufgabe wird allerdings nicht exakt gelöst, sondern nur näherungsweise, da auch Teile der ausgewählten Boxen aus dem Sichtkegel herausragen können. Eine exakte Lösung würde die Verwendung sehr kleiner Boxen erfordern und damit die Anzahl von benötigten Boxen stark erhöhen, daher ist ein approximatives *„viewing frustum culling“* sicherlich zu bevorzugen; entsprechend findet sich eine solche Strategie in vielen Arbeiten in der Literatur (z.B. [Chamberlain et al. 95]).

Es stellt sich die Frage, ob das ungenaue Abschneiden den Aufwand bei der Berechnung des Bildes nicht übermäßig dadurch erhöht, daß Szenenteile bearbeitet werden, die eigentlich nicht sichtbar sind. Es ist sicherlich möglich, ein worst-case Szenario zu konstruieren, in dem nur Objekte in den Boxen enthalten sind, die gerade außerhalb der Sichtpyramide liegen. Der Mehraufwand wäre im schlimmsten Fall also beliebig groß. Für praktische Anwendungen kann man aber annehmen, daß solche bösartigen Konstellationen nur bei wenigen Szenen und nur für wenige Betrachterstandpunkte zu erwarten sind. Die Frage ist daher, wie groß der Mehraufwand ist, wenn die Objekte gleichmäßig im Bild verteilt sind.

Die Laufzeit des randomisierten z-Buffer Verfahrens wächst linear mit der projizierten Fläche. Um diese zu schätzen, wird angenommen, daß der Raum gleichmäßig mit kleinen Objekten mit fester Orientierung zum Blickpunkt gefüllt ist, so daß nur die Entfernung Einfluß auf die Projektion hat. Diese Idealisierung liegt der Definition des „projizierten Volumens“ zugrunde: jeder Punkt im Raum wird mit $1/z^2$ gewichtet, also mit dem Faktor, um den ein Oberflächenelement an dieser Stelle bei der Projektion verkleinert wird. Für Szenen mit gleichmäßig verteilten Objekten und zufälligen Oberflächennormalen und sollte dieses Modell eine Abschätzung des Wachstums der projizierten Fläche bei einer Vergrößerung des Volumens erlauben.

Definition 14: *projiziertes Volumen*

Sei ein Volumen $V \subseteq \mathbb{R}^3$ gegeben. Sei p das Projektionszentrum der Zentralprojektion und r die Blickrichtung. Dann wird das Integral

$$prj(V) := \int_V \frac{1}{Tiefe(x)^2}\,dx = \int_V \frac{1}{\left(r \circ (x - p)\right)^2}\,dx$$

als *projiziertes Volumen von* V bezeichnet. (V sei so gewählt, daß das Integral existiert.)

Nun kann gezeigt werden, daß das projizierte Volumen durch das approximative *„viewing frustum culling“* nur um eine Konstanten Faktor erhöht wird:

Satz 7: *Genauigkeit des „viewing frustum culling“*

Wenn Boxen wie im Beweis von Satz 6 ausgewählt werden, dann wächst das projizierte Volumen der ausgewählten Boxen bei wachsender Tiefe *maxdepth* der Szene höchstens um einen konstanten Faktor stärker als das projizierte Volumen der Sichtpyramide, die durch *maxdepth* in der Tiefe beschränkt wird. Die Konstante hängt von ε und den Projektionsparametern ab.

Beweis: Das projizierte Volumen der Sichtpyramide ist gegeben durch:

$$\begin{aligned} prj(F) &= \int_F \frac{1}{x^2}\,dx \\ &= \int_{mindepth}^{maxdepth} a\left(\frac{z}{d}\right)^2 \cdot \frac{1}{z^2}\,dz \\ &= \frac{a}{d^2}\int_{mindepth}^{maxdepth} 1dz \\ &= \frac{a}{d^2}\left(maxdepth - mindepth\right) \end{aligned}$$

Bei der Überdeckung der Sichtpyramide mit Octreeboxen werden auch Bereiche außerhalb der Sichtpyramide ausgewählt. Betrachte nun einen Tiefenbereich k wie im Beweis von Satz 6 definiert. Die maximale Abweichung entsteht hier durch eine Box zur Überdeckung von Tiefenbereich k, die in Tiefenbereich k-1 hineinragt und teilweise außerhalb der Sichtpyramide liegt. Nach der Konstruktion kann die Box nicht in kleinere Tiefenbereiche hineinragen.

Im Abstand z_{k-2} erhält man also eine maximale Höhe des Gebildes, das durch die Überdeckung der Sichtpyramide mit Boxen entsteht, von

$$h\frac{z_{k-2}}{d} + 2q = h\frac{z_{k-2}}{d} + z_{k-1} - z_{k-2} = h\frac{z_{k-2}}{d} + \varepsilon' z_{k-2} - z_{k-2} = z_{k-2}\left(h\frac{1}{d} + \varepsilon' - 1\right)$$

und entsprechend eine Breite von

$$w\frac{z_{k-2}}{d} + 2q = z_{k-2}\left(w\frac{1}{d} + \varepsilon' - 1\right).$$

Die maximale Höhe und Breite des von den Boxen überdeckten Volumens weicht in jedem Tiefenbereich an der Stelle maximaler Abweichung also nur um einen konstanten Faktor vom bisherigen Wert ab. Daher können die Boxen in eine Pyramide mit erweiterter Querschnittsfläche eingeschlossen werden. Die erweiterte Querschnittsfläche beträgt im Abstand z

$$z\cdot\left(\frac{w}{d} + \varepsilon' - 1\right)\cdot z\cdot\left(\frac{h}{d} + \varepsilon' - 1\right) = z^2\cdot\left(\frac{a}{d^2} + \underbrace{\frac{w(\varepsilon'-1)}{d} + \frac{h(\varepsilon'-1)}{d} + (\varepsilon'-1)^2}_{C}\right) = z^2\cdot\left(\frac{a}{d^2} + C\right)$$

wenn man die zusätzlichen Konstanten unter dem Namen C zusammenfaßt.

Die minimale Tiefe wird ebenfalls durch das näherungsweise Clipping verringert: Zur Überdekkung von Tiefenbereich 1 werden Boxen der Größe $mindist/\varepsilon'$ verwendet. Daher reduziert sich die neue minimale Tiefe im schlimmsten Fall auf $mindist - mindist/\varepsilon' = mindist\cdot(1-1/\varepsilon') > 0$. Als projiziertes Volumen erhält man damit:

$$
\begin{aligned}
prj(Boxen) &= \int_{mindepth\cdot(1-1/\varepsilon')}^{maxdepth} z^2\left(\frac{a}{d^2}+C\right)^2 \frac{1}{z^2}\,dz \\
&= \left(\frac{a}{d^2}+C\right)^2 \cdot \int_{mindepth\cdot(1-1/\varepsilon')}^{maxdepth} 1\,dz \\
&= \left(\frac{a}{d^2}+C\right)^2 \left(maxdepth - mindepth\cdot\left(1-1/\varepsilon'\right)\right)
\end{aligned}
$$

Als Verhältnis erhält man:

$$
\begin{aligned}
\frac{prj(Boxen)}{prj(F)} &= \frac{\left(\frac{a}{d}+C\right)^2}{\frac{a}{d^2}} \cdot \frac{maxdepth - mindepth\cdot\left(1-1/\varepsilon'\right)}{maxdepth - mindepth} \\
&\in \mathrm{O}\left(\frac{maxdepth - mindepth + mindepth/\varepsilon'}{maxdepth - mindepth}\right) \\
&= \mathrm{O}\left(1+\frac{mindepth}{\varepsilon'\left(maxdepth - mindepth\right)}\right) \\
&= \mathrm{O}(1).
\end{aligned}
$$

■

Abbildung 47 zeigt die Auswahl von Boxen aus einem Octree an einem Beispiel.

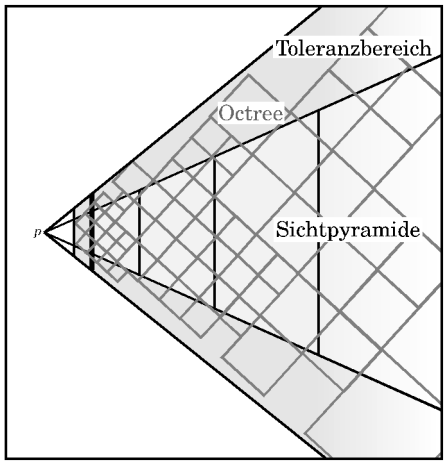


Abbildung 47: Auswahl von Boxen mit beschränktem Tiefenverhältnis und Clipping mit Toleranz

Wenn es wünschenswert ist, die Genauigkeit des *„viewing frustum culling“* unabhängig vom Parameter ε zu steuern, kann ein zusätzlicher Toleranzwert c in Algorithmus 9 vorgesehen werden. Boxen, die nicht vollständig in der Sichtpyramide liegen und einen größeren Durchmesser als $c \cdot z$ haben, werden dann in jedem Fall aufgeteilt. Damit ist es möglich, das Abschneiden am Sichtvolumen genauer als durch ε vorgegeben vorzunehmen. Für Szenen mit großer projizierter Fläche könnte dies Vorteile haben.

3.3.4.2 Der Verteilungsbaum

Bis hierhin wurde eine Strategie zur Einteilung der Szene in Gruppen entwickelt, die die Abweichungen des Projektionsfaktors bezüglich der Tiefe beschränkt, die Orientierung der Dreiecke wurde noch nicht berücksichtigt. Nun sollen als erstes zwei Methoden erläutert werden, die mit Hilfe einer solchen Gruppeneinteilung eine Auswahl von Stichprobenpunkten vornehmen. Diese Methoden werden auch in allen weiteren Strategien, die auch die Orientierungen berücksichtigen, genauso eingesetzt, da diese dieselbe Boxenauswahlmethode für Gruppen von Objekten mit ähnlichen Winkeln einsetzen.

Auswahl der Stichprobenpunkte

Um bei gegebenen Gruppen von Dreiecken die Stichprobenpunkte auszuwählen, sind zwei verschiedene Algorithmen denkbar. Wenn man tatsächlich eine zufällige Gleichverteilung simulieren möchte, müssen die Stichprobenpunkte zunächst zufällig auf die Gruppen verteilt werden. Die Produkte aus dem jeweiligen maximalen Projektionsfaktor und des in der jeweiligen Gruppe enthaltenen Flächeninhaltes der Objekte dienen dabei als Gewichte. Dann werden zufällige Dreiecke aus den Gruppen ausgewählt, wobei nur noch die (blickpunktunabhängigen) absoluten Flächeninhalte als Gewichte verwendet werden. Danach wird für jedes Dreieck ein zufälliger Punkt bestimmt. Man erhält dadurch folgenden Grobalgorithmus:

```
1) Wähle Gruppen G_1...G_n aus, so daß G_1 ∪ ... ∪ G_n alle sichtbaren
   Objekte enthält und der Projektionsfaktor innerhalb jeder
   Gruppe nur wenig variiert
2) Bestimme die Gesamtanzahl m der Stichprobenpunkte aus der
   geschätzten projizierten Fläche
3) Für 1..m wiederhole:
4)    bestimme zufällige Objektgruppe k nach den Gewichten
      absoluteFläche(G_i) * prj_max(G_i)
5)    wähle ein Dreieck t aus G_k, gewichtet nach der
      absoluten Fläche
6)    wähle einen zufälligen Punkt auf t
```

Algorithmus 10: rein zufälliger Grobalgorithmus zur approximativen Stichprobenauswahl

Zur Verteilung der Stichprobenpunkte auf die Objektgruppen wird dynamisch (für jeden Renderinglauf neu) eine Verteilungsliste aus den Objektgruppen und ihren Gewichten bestimmt und dann damit die Auswahl der Boxen für die Stichprobenpunkte vorgenommen.

Dieser Aufwand ist allerdings nicht notwendig, da für die Bildrekonstruktion keine völlige Zufälligkeit benötigt wird: Anstatt die Stichprobenpunkte zufällig auf die Boxen zu verteilen, können diese auch den Boxen fest zugeordnet werden: Für jede Box wird der maximale Projektionsfaktor bestimmt und mit dem enthaltenen Flächeninhalt multipliziert. Der so erhaltene Stichprobenumfang wird auf die nächst höhere ganze Zahl aufgerundet damit kleine Boxen nicht im Mittel unterrepräsentiert sind. Danach wird eine solche Anzahl von Punkten wie gehabt auf die Dreiecke verteilt. Dafür erhält man folgenden Grobalgorithmus:

```
1) Wähle Gruppen G1...Gn aus, so daß G1 ∪ ... ∪ Gn alle sichtbaren
   Objekte enthält und der Projektionsfaktor innerhalb jeder
   Gruppe nur wenig variiert
2) Für alle Gruppen Gi:
3)    mi = absolute Fläche(Gi) * prjmax(Gi)
4)    Für 1..mi wiederhole:
5)       Wähle ein Dreieck t zufällig aus Gi, gewichtet nach den
         absoluten Flächen der Dreiecke
6)       wähle einen zufälligen Punkt aus t
```

Algorithmus 11: „halbdeterministischer" Grobalgorithmus zur approximativen Stichprobenauswahl

In der Praxis stellte sich bei allen untersuchten Beispielen heraus, daß sich die beiden Varianten hinsichtlich des notwendigen Stichprobenumfangs, welcher im deterministischen Fall eventuell kleiner ausfallen könnte, nicht sichtbar unterschieden. Je nach Szene arbeitete die „halbdeterministische" Variante allerdings bei gleichem Stichprobenumfang schneller (bis zu 20% auf Testsystem 1 und bis zu 50% auf Testsystem 2, siehe Anhang A) und im schlimmsten Fall genauso schnell wie die rein zufällige, so daß man dieser Variante den Vorzug geben sollte.

Speicherung der Verteilungslisten für die Boxen

In beiden Varianten müssen Dreiecke aus den Objektgruppen gewichtet nach den Flächeninhalten zufällig ausgewählt werden. Die Objektgruppen sind hier die ausgewählten Boxen aus dem Raumunterteilungsbaum. Eine Möglichkeit, die Dreiecksauswahl zu bestimmen, besteht darin, für jeden Knoten im Baum eine Liste aller darin enthaltenen Objekte und deren Flächeninhalte bereitzustellen. Daraus kann wie in Abschnitt 3.3.2.2 eine Verteilungsliste gebildet werden. Wenn diese vorberechnet vorliegt, können beim Rendering Dreiecke per binärer Suche nach ihrer Fläche gewichtet ausgewählt werden.

Es würde relativ viel Speicher erfordern, für jeden Knoten des Baumes eine eigene Verteilungsliste zu speichern. Durch eine Verschachtelung der Teillisten kann man jedoch mit einer einzigen Liste auskommen: Für alle Knoten im Baum wird rekursiv gleich verfahren: Zuerst werden die Flächen der Dreiecke, die dem aktuellen Knoten zugeordnet sind[15], bestimmt und in die Liste eingetragen, d.h. der Wert wird zum vorangegangenen Wert hinzuaddiert und an die Liste angehängt. Danach wird für alle Kindknoten, falls vorhanden, in einer beliebigen Reihenfolge rekursiv das gleiche gemacht. Im Knoten wird dann der Tabellenindex für den ersten, vom Knoten selbst vorgenommenen Eintrag und der Tabellenindex des letzten Eintrages des letzten Kindknotens gespeichert. Dieser zusammenhängende Abschnitt der Liste stellt dann die Verteilungsliste für den Knoten dar. Im Unterschied zur zuvor definierten Liste fangen die summierten Flächenwerte in der Teilliste nicht bei null an und enden nicht bei eins. Dies kann aber bei der binären Suche leicht durch eine Transformation des Suchwertes ausgeglichen werden (der Suchwert wird mit der Differenz der beiden Randwerte multipliziert und zum Anfangswert addiert).

Als Alternative könnte die Verteilungsliste auch implizit im Baum gespeichert werden, dies wird in Abschnitt 3.3.6.3 benutzt.

Man erhält folgende Aussage:

---

[15] Bei dem bisher verwendeten Unterteilungsbaum können inneren Knoten keine Dreiecke zugeordnet sein, im allgemeinen Fall wird dies aber benötigt, siehe Abschnitt 3.3.6.1.

Satz 8: *räumlicher Verteilungsbaum*

Mit dem zuvor beschriebenen Verteilungsbaum können m Stichprobenpunkte für eine Szene von n Dreiecken in Zeit $\Theta(m \cdot (\log n + \log K) + K)$ mit der zufälligen bzw. $\Theta((m+K) \cdot \log n)$ mit der halbdeterministischen Boxenauswahlvariante ausgewählt werden. K sei dabei die Anzahl von Boxen, die zur Einschränkung des Projektionsfaktors benötigt werden. Die Datenstruktur kann in $O(n \cdot \log n)$ Zeit aufgebaut und in $\Theta(n)$ Speicherplatz abgelegt werden.

Beweis: Die Variante mit zufälliger Boxenauswahl würfelt für jeden der m Punkte zunächst eine Box, was $O(\log K)$ Zeit kostet und dann ein Dreieck, was $O(\log n)$ Zeit für die binäre Suche kostet. Die Auswahl eines Punktes auf dem Dreieck erfolgt in konstanter Zeit. Zusätzlich muß in $O(K)$ Zeit erst eine Verteilungsliste für die Boxen erstellt werden. Insgesamt erhält man daher die behauptete Zeitschranke.

Die Variante des Algorithmus, die die Stichprobenpunkte deterministisch auf die Boxen verteilt, verteilt auf die K Boxen insgesamt m Dreiecke und für jede Box mindestens ein Dreieck. Jede Auswahl eines zufälligen Dreiecks kostet wieder $O(\log n)$ Zeit.

Nun zur Vorberechnung. Wenn der Raumunterteilungsbaum so implementiert wird, wie er definiert wurde, ist nicht garantiert, daß dieser nur linearen Speicherplatz benötigt. Es ist möglich, daß ein Knoten aufgeteilt wird, ohne daß zwei Objekte voneinander getrennt werden, d.h. ohne daß sie in verschiedene Kindknoten gelangen. Daher muß, wenn diese Situation eintritt, ein „*shortcut*“ [Bern et al. 90] in den Baum eingebaut werden: Es wird der kleinste umfassende, achsenparallele Quader für die Menge von Objekten bestimmt. Die längste Kantenlänge wird auf die nächstliegende Zweierpotenz an der Untergrenze ab und an der Obergrenze aufgerundet. Dazu kann der Zweierlogarithmus berechnet und entsprechend gerundet werden und anschließend wieder zwei mit diesen Werten potenziert werden. Mit dieser Kantenlänge wird nun ein Würfel im Raster der Zweierpotenz gebildet, der den ursprünglichen Quader komplett enthält. Dieser Quader wird als einziges Kind in den Baum aufgenommen. Dadurch bleibt exakt die ursprüngliche Form des Baumes erhalten, es werden lediglich ein ganze Anzahl von leeren Boxen nicht gespeichert. Außerdem sind anstelle einiger Boxen nur noch kleinere Boxen als im vollen Baum vertreten, die dieselben Objekte enthalten. Da dabei die Variation des Projektionsfaktors höchstens kleiner wird und die Anzahl von bearbeiteten und ausgewählten Boxen nur sinken kann, werden die in Satz 6 gezeigten Schranken dadurch nicht erhöht. In dem Baum mit „*shortcuts*“ werden in jedem Knoten mindestens zwei Objekte voneinander getrennt, außer in Knoten mit „*shortcuts*“, auf die aber immer ein gewöhnlicher Knoten folgt. Daher gibt es maximal $O(n)$ Knoten.

Um den Baum in $O(n \cdot \log n)$ Zeit aufzubauen, wird eine Technik aus [Callahan 95] (S. 31) verwendet, die dort für eine ähnliche Konstruktion benutzt wurde: Die Koordinaten aller Objekte, die hier als Punkte angenommen wurden, werden in drei nach allen drei Koordinatenrichtungen sortierten Listen gespeichert. Dann werden rekursiv die Listen durchgegangen und die Punkte werden danach markiert, ob sie oberhalb oder unterhalb der mittleren Teilungsebene in allen drei Dimensionen liegen. Danach werden alle Listen ein weiteres mal durchlaufen und die Punkte gemäß ihrer Markierung auf die Kindknoten aufgeteilt. Da die Punkte dabei in sortierter Reihenfolge aufgegriffen werden, können aus den alten Listen direkt neue Listen für alle Kindknoten erstellt werden, die die in ihnen enthaltenen Punkte wiederum für alle Dimensionen in sortierter Reihenfolge enthalten, so daß die selbe Situation wie vor dem Rekursionsschritt herrscht. Falls alle Objekte in eine Box fallen, so muß ein „*shortcut*“ eingefügt werden. Dies

kann in konstanter Zeit wie oben beschrieben erfolgen, da sich der kleinste umfassende Quader direkt aus den Randpunkten der sortierten Listen bestimmen läßt.

Dieser Prozeß benötigt nur lineare Zeit, das anfängliche Sortieren zur Bereitstellung des ersten Satzes von Listen benötigt $O(n \cdot \log n)$ Zeit, so daß die behauptete Zeitschranke eingehalten werden kann. ■

3.3.4.3 Erwarteter Aufwand

Läßt man zunächst den Einfluß der Orientierung der Dreiecke und den des Verzerrungsfaktors auf den Projektionsfaktor außer acht, kann der im vorherigen Abschnitt beschriebene rein räumliche Verteilungsbaum bereits zur Auswahl von Stichprobenpunkten genutzt werden, wobei der Stichprobenumfang allerdings überschätzt wird, u.U. auch um mehr als den für den räumlichen Baum vorgegebenen Faktor ε. Im schlimmsten Fall könnte der Mehraufwand beliebig groß werden: Es ist vorstellbar, daß in einer Szene die Normalen aller Dreiecke senkrecht zum Beobachter stehen. Der notwendige Stichprobenumfang wäre dann null. Je nach Größe der Dreiecke können aber beliebig viele Stichprobenpunkte gewählt werden, wenn die Orientierung der Dreiecke nicht berücksichtigt wird.

Dieses Verhalten kann auch in typischen Anwendungsbeispielen auftreten: Häufig ist die Szene in Form einer großen, zweidimensionalen Ebene angeordnet (z.B. bei Stadt und Landschaftsszenen). Die projizierte Fläche (mit Berücksichtigung der Sichtpyramide) einer einfachen, beliebig großen Ebene ist durch eine Konstante beschränkt (maximal füllt sie den ganzen Bildschirm). Das Integral über den Projektionsfaktor ist dies aber ohne Berücksichtigung der Orientierung nicht: Sei K ein Ausschnitt aus einer unendlich großen Ebene in Form eines Kreissegmentes mit Radius r, der als Schnitt der Sichtpyramide mit der Ebene entsteht. Dann erhält man eine projizierte Fläche von

$$\int_K \frac{1}{z^2} dz = \int_{\Theta(1)}^{r} \Theta(z) \frac{1}{z^2} dz = \Theta\left(\int_{\Theta(1)}^{r} \frac{1}{z} dz \right) \in \Theta(\log r) ,$$

was offensichtlich unbeschränkt mit r wächst, wenn auch so schwach, daß hier noch keine größeren Probleme zu erwarten wären.

Es bleibt die Frage, wie sich der Mehraufwand verhält, wenn eine „durchschnittliche“ Szene vorliegt, in der keine Orientierung systematisch bevorzugt wird (bei der Szene mit der Ebene ist dies die Umgebung um die senkrechte Orientierung, daher die unbeschränkte Abweichung). Um diese Frage zu untersuchen, wird angenommen, daß die Betrachtungswinkel auf Oberflächennormalen in der Szene rein zufällig auftreten. Dann wird der erwartete Wert des Projektionsfaktors bestimmt und mit dem Erwartungswert für den Projektionsfaktor verglichen, den man erhält, wenn immer der Blickwinkel null angenommen wird, was bei einer Strategie, die die Orientierungen nicht kennt, gemacht werden muß, um überall eine ausreichende Stichprobendichte zu garantieren.

Es erscheint aus Symmetriegründen sinnvoll, für die Verteilung der Blickwinkel anzunehmen, daß die Normalvektoren auf der Einheitskugel, unabhängig vom Betrachtungsvektor, gleichverteilt sind. Für Szenen mit vielen unterschiedlichen, gleichmäßig verteilten Orientierungen der Normalen, in denen keine einzelnen Elemente besonders groß im Bild abgebildet werden, kann man dadurch eine relativ genaue Schätzung des Mehraufwandes erwarten. Für dieses Szenario erhält man:

Lemma 6: *durchschnittlicher Projektionsfaktor*

Es sei ein Punkt auf einer Oberfläche an fester Position unter einer festen Projektion gegeben. Der Normalvektor der Oberfläche sei zufällig gleichverteilt aus der Einheitskugel gewählt. Dann ist die Wahrscheinlichkeitsdichte der Zufallsvariablen, die einer solchen zufällig gezogenen Normalen den Betrachtungswinkel φ zuordnet, durch $\varphi \mapsto (\sin \varphi)/2$ gegeben. Der Erwartungswert für den Projektionsfaktor ist 1/4 so groß wie der maximale Projektionsfaktor an dieser Stelle.

Beweis: Der Projektionsfaktor ist gegeben durch:

$$prj(s) = \frac{d^2 \cdot cut_{[0,1]}(\cos\beta)}{z^2 \cdot \cos\alpha}$$

Dieser wird maximal bei $\beta = 0$, so daß der maximale Wert

$$prj_{max} = \frac{d^2}{z^2 \cdot \cos\alpha}$$

beträgt. Der erwartete Wert beträgt

$$prj_{av} = E\left(\frac{d^2 \cdot cut_{[0,1]}(\cos\beta)}{z^2 \cdot \cos\alpha}\right) = \frac{d^2}{z^2 \cdot \cos\alpha} E\left(cut_{[0,1]}(\cos\beta)\right),$$

wobei E den Erwartungswert bei gleichmäßiger Verteilung der Normalen auf der Einheitskugel angibt. Wie man sieht, ist das Verhältnis gegeben durch

$$\frac{prj_{av}}{prj_{max}} = E\left(cut_{[0,1]}(\cos\beta)\right).$$

Daß die Normalen auf der Einheitskugel gleichverteilt sind, bedeutet, daß die Wahrscheinlichkeit, daß eine Normale innerhalb eines Oberflächenstückes dieser Kugel mit Flächeninhalt A liegt, $A/4\pi$ beträgt. Sei nun φ der Winkel zwischen Blickrichtung und Normalen. Die Wahrscheinlichkeit, daß φ zwischen 0 und φ_0 liegt, beträgt dann

$$P(\varphi \le \varphi_0) = \frac{2\pi h}{4\pi} = \frac{1 - \cos\varphi}{2}$$

mit h als Höhe des Kugelabschnittes (siehe Abbildung 48; die Formel für die Mantelfläche des Kugelabschnittes findet sich in [Bronstein et al. 97], S. 140).

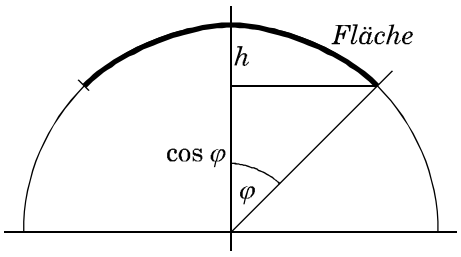


Abbildung 48: Fläche auf der Einheitskugel in Abhängigkeit des Winkels φ

Die Wahrscheinlichkeitsdichte f zu dieser Verteilungsfunktion erhält man durch Ableiten als:

$$f(\varphi) = \frac{d}{d\varphi}\frac{1-\cos\varphi}{2} = \frac{\sin\varphi}{2}$$

Der Erwartungswert ist dann gegeben durch:

$$\begin{aligned} E\left(cut_{[0,1]}(\cos\beta)\right) &= \frac{1}{2}\int_0^{\pi/2} \sin\varphi\cdot\cos\varphi\, d\varphi \\ &= \frac{1}{2}\left[\frac{1}{2}\sin^2\varphi\right]_{\varphi=0}^{\varphi=\pi/2} \\ &= \frac{1}{4} \end{aligned}$$

■

Wenn man also Stichproben nur anhand der bisher entwickelten räumlichen Unterteilung in eine Szene verteilt, bei der alle Orientierungen im projizierten Bild etwa gleich häufig auftreten, so ist zu erwarten, daß der notwendige Stichprobenumfang nur 4ε so groß ist, wie der bei einer exakten Gleichverteilung benötigte. Es ist zudem relativ effizient möglich, Punkte, in denen die Normale von Betrachter abgewandt ist, frühzeitig zu erkennen und nicht weiter zu verarbeiten. Dazu ist im wesentlichen nur die Berechnung einer Vektordifferenz und eines Skalarproduktes notwendig. Dies reduziert den Faktor beim Stichprobenumfang sogar auf 2ε.

3.3.5 Berücksichtigung der Orientierung

Um auch in ungünstigeren Fällen eine starke Erhöhung des Stichprobenumfangs vermeiden zu können, muß der Einfluß der Orientierung der Oberfläche auf den Projektionsfaktor berücksichtigt werden. Im folgenden werden Strategien zur Stichprobenauswahl untersucht, die neben der Tiefe auch den Winkel der Oberfläche zum Betrachter berücksichtigen. Dabei lassen sich Typen von Strategien unterscheiden: Die einen können eine ε-Approximation der idealen Verteilung bestimmen für ein konstantes ε bestimmen, d.h., die Stichprobendichte ist keiner an keiner Stelle um mehr als einem Faktor ε nach oben vom idealen Wert entfernt und unterschreitet diesen nie. Die anderen Strategien können den Fehler bei der Approximation der Verteilung nur verringern, aber nicht beschränken können: Die Stichprobendichte liegt auch nie unter dem idealen Wert, kann aber (stellenweise) beliebig weit oberhalb des idealen Wertes liegen.

Die hier vorgestellten Verfahren der ersten Art haben damit zu kämpfen, daß der Aufwand zur Klassifizierung der Objekte sehr groß werden kann. Daher sind die Strategien der zweiten Art für die praktische Anwendung vorzuziehen. Obwohl sie keine Schranken für die Güte der Approximation garantieren, liefern sie im Gesamtergebnis akzeptable Laufzeiten, wie sich in praktischen Beispielen (siehe Kapitel 5) zeigt. Um diese Beobachtung auch analytisch untermauern zu können, müssen stärkere Annahmen über die Szene gemacht werden: Auch hier wird wie im vorherigen Abschnitt wieder angenommen, daß die Oberflächennormalen unabhängig vom Betrachtungsvektor auf der Einheitskugel gleichverteilt sind. Dann wird untersucht, wie genau die Strategie im Durchschnitt den Projektionsfaktor schätzen kann, so daß man auch für diese Art von Strategien ein quantitatives Maß dafür erhält, wie gut sie ihre Aufgabe bei in diesem Sinne „durchschnittlichen“ Szenen erfüllen.

3.3.5.1 Klassifizierung der Normalen

Ein erster Schritt bei der Berücksichtigung der Orientierung ist die Klassifizierung der Dreiecke in Gruppen mit ähnlichen Oberflächennormalen. Die Menge aller möglichen Normalen bildet die Einheitskugel. Diese Menge sollte nun in möglichst wenige Bereiche aufgeteilt werden, in denen die Winkelunterschiede zwischen den darin enthaltenen Normalen möglichst klein sind. Für diese Aufteilung kommen verschiedene Strategien in Betracht, die unterschiedlich gute Aufteilungen liefern können. [Samet 90] gibt einen Überblick (S. 21ff) über verschiedene Möglichkeiten, eine Kugel hierarchisch in Bereiche aufzuteilen. Eine verbreitete Methode beginnt damit, einen der 5 platonischen Körper als Ausgangspunkt zu wählen. Die Punkte auf den Flächen des Körpers werden Punkten auf der Kugel durch Normieren der zugehörigen Vektoren zugeordnet. Wenn die so entstandenen Gebiete noch zu groß sind, werden die Flächen durch ein rekursives Unterteilungsschema bei Bedarf tiefer unterteilt. Dabei wird eine Fläche durch mehrere kleinere gleich oder ähnlich geformte ersetzt, die die ursprüngliche Fläche abdecken. Die Oberfläche läßt sich dadurch relativ gleichmäßig einteilen, da die Grundgebiete bereits sehr regelmäßig sind.

In dieser Arbeit soll eine einfachere Methode benutzt werden: Die Punkte der Kugel werden in dreidimensionale Polarkoordinaten umgerechnet und dann in einem rechteckigen Parameterbereich nach Längen und Breitengrad aufgeteilt. Im Falle der Winkelklassen (nächster Abschnitt) werden Längen und Breitengrade durch ein äquidistantes Gitter aufgeteilt und im hierarchischen Fall (übernächster Abschnitt) wird ein Quadtree benutzt. Die Transformation von Polarkoordinaten auf Normalen wird durch die Funktion

$$(\lambda,\varphi)\mapsto\begin{pmatrix}\sin\varphi\cdot\cos\lambda\\ \sin\varphi\cdot\sin\lambda\\ \cos\varphi\end{pmatrix}$$

vorgenommen [Bronstein et al. 97], wobei sich λ im Bereich $[0,2\pi]$ bewegen kann und den „Längengrad“ angibt und φ im Bereich $[0,\pi]$ verläuft und den „Breitengrad“ angibt (siehe Abbildung 49).

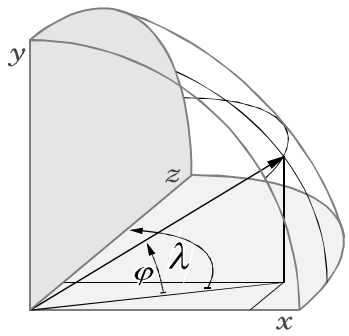


Abbildung 49: Polarkoordinaten

Wenn zwei Normalen n_1 und n_2 mit Polarkoordinaten (λ_1, φ_1) und (λ_2, φ_2) gegeben sind, dann ist der Winkel im Raum zwischen n_1 und n_2 nicht größer als $|\lambda_2-\lambda_1|+|\varphi_2-\varphi_1|$. Das Bogenstück zwischen den beiden Normalen liegt auf einem Großkreis der Einheitskugel. Die Summe setzt sich aus zwei Bogenstücken zusammen, die von n_1 zum Punkt mit den Polarkoordinaten (λ_1, φ_2) verlaufen und von dort aus zu n_2. Diese Stücke bilden ebenfalls einen geschlossenen Weg von n_1 zu n_2 auf der Oberfläche der Einheitskugel, der länger ist als das Bogenstück zwischen n_1 und n_2,

da die kürzesten Verbindungen auf der Kugeloberfläche genau diejenigen sind, die auf einem Großkreis liegen.

Diese Abschätzung ist relativ grob. Der genaue Wert für den Winkel zwischen n_1 und n_2 ist $\arccos(\cos(\lambda_2 - \lambda_1)\cdot\cos(\varphi_2 - \varphi_1))$: nach [Bronstein et al. 97] (Seite 152) ist dies die Bogenlänge der Hypothenuse in einem rechtwinkligen, sphärischen Dreieck. Für kleine Winkel können die Bogenstücke auch näherungsweise durch Geradenstücke ersetzt werden, so daß man ein rechtwinkliges Dreieck mit Kathetenlängen $|\lambda_2 - \lambda_1|$ und $|\varphi_2 - \varphi_1|$ erhält. Der diagonale Winkel ist dann näherungsweise $\sqrt{(\lambda_2 - \lambda_1)^2 + (\varphi_2 - \varphi_1)^2}$. In der Implementation wurde die genaue Formel verwendet, um unnötigen Aufwand zu vermeiden.

Seien nun λ_1 und φ_1 die jeweils kleinsten und λ_2 und φ_2 die größten Polarkoordinaten innerhalb einer Gruppe von Normalvektoren. Die Gruppe wird dann durch einen mittleren Normalvektor

$$n_G := \frac{1}{2}\left((\lambda_1, \varphi_1) + (\lambda_2, \varphi_2)\right) \quad \text{(in Polarkoordinaten)}$$

und ein maximalem Abweichungswinkel

$$\mu_G := 2\arccos\left(\cos\frac{\lambda_2 - \lambda_1}{2}\cos\frac{\varphi_2 - \varphi_1}{2}\right) \leq (|\lambda_2 - \lambda_1| + |\varphi_2 - \varphi_1|)$$

repräsentiert, um später den maximalen Projektionsfaktor abzuschätzen. Die folgende Analyse berücksichtigt nur die Abschätzung durch die Summe der Kantenlängen der „Winkelbox". $\mu_G/2$ beschränkt dann die maximale Abweichung von der repräsentativen Normalen und μ_G die maximale Winkelabweichung innerhalb der Gruppe.

3.3.5.2 Winkelklassen

Eine naheliegende Methode zur Berücksichtigung der Oberflächenorientierung bei der Schätzung des Projektionsfaktors besteht darin, die gesamte Szene in feste Klassen ähnlicher Normalen zu gruppieren und für jede Klasse von Objekten mit ähnlichen Normalen einen räumlichen Verteilungsbaum aufzubauen. Die Klassen sind hier unabhängig vom Betrachtungswinkel, daher sollten sie für alle Winkelbereiche möglichst gleich groß sein. Dazu werden alle Normalen wie beschrieben in Polarkoordinaten umgerechnet und dann in ein regelmäßiges Gitter aus qua-

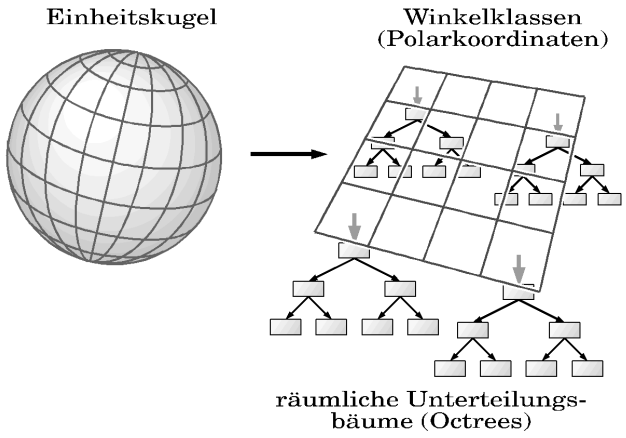


Abbildung 50: Datenstruktur für das Winkelklassenverfahren (schematisch)

dratischen Teilbereichen eingeteilt. Für jede Zelle des Gitters wird ein räumlicher Verteilungsbaum aufgebaut (schematisch in Abbildung 50). Die Frage ist nun, wie fein das Gitter gewählt werden sollte und wie die räumliche Unterteilung in den verschiedenen Zellen vorgenommen werden soll.

Approximation der Verteilung ohne konstante Fehlerschranke

Eine realistische Strategie kann zwangsläufig das Gitter nur relativ grob wählen (größenordnungsmäßig höchstens einige hundert Zellen) damit die Anzahl zu bearbeitender Boxen nicht zu groß wird. Der Einfluß der Orientierung kann dann nur näherungsweise abgeschätzt werden: Nachdem für alle Winkelklassen aus den räumlichen Unterteilungsbäumen Boxen ausgewählt wurden, deren relative Tiefe nicht zu groß ist, muß für jede eine obere Schranke für den Projektionsfaktor bestimmt werden, mit der dann die in der Box enthaltene Fläche gewichtet wird.

Dazu wird in die Formel für den Projektionsfaktor neben der minimalen Tiefe eine untere Schranke für den Winkel zwischen der Oberflächennormalen und dem Vektor zum Betrachter eingesetzt (also eine obere Schranke für den Projektionsfaktor, da der Kosinus im betrachteten Bereich streng monoton fällt). Vernachlässigt man zunächst den Einfluß der Position auf diesen Winkel, so erhält man diese, indem man den Winkel zwischen Beobachter und mittlerer Normalen n_G in der betrachteten Winkelklasse berechnet und davon die maximale Abweichung $\mu_G/2$ abzieht. Falls der Wert null unterschreitet, so wird er auf null gesetzt, da bei diesem Winkel der größte Projektionsfaktor auftritt. Falls dieser Wert $\pi/2$ oder größer ist, kann die Winkelklasse ignoriert werden, da sie nur Rückseiten von Objekten enthalten kann.

Um auch den Einfluß der Position auf den Winkel beschränken zu können, muß die räumliche Ausdehnung der Boxen u.U. weiter eingeschränkt werden: Es erscheint aus Symmetriegründen wünschenswert, die Abweichung des Winkels durch verschiedene Positionen im Raum in der gleichen Größenordnung zu halten wie die Eingrenzung des Winkels durch die Winkelklassen (eine empirische Untersuchung zur Wahl dieser Parameter findet sich in Abschnitt 5.1.2). Wenn Boxen ausgewählt werden, in denen der Projektionsfaktor eingeschränkt auf die Tiefe höchstens um ε variiert, dürfen Boxen in einem minimalen Abstand von z_{min} höchstens eine Tiefe von $z_{min}\sqrt{\varepsilon} - z_{min} = z_{min}(\sqrt{\varepsilon} - 1)$ haben. Die relative Größe der Boxen, d.h. das Verhältnis zwischen diagonalem Durchmesser und minimaler Tiefe, ist also durch $r := \left(\sqrt{3\varepsilon} - 1\right)$ nach oben beschränkt. Der Faktor $\sqrt{3}$ ist dabei das maximale Verhältnis zwischen der Diagonalen und der Tiefe einer Box. Die Box läßt sich also in eine Kugel mit Durchmesser $d_K = z_{min}(\sqrt{3\varepsilon} - 1)$ einschließen und nach Lemma 5 erhält man, daß die Abweichung γ des Betrachtungswinkels aufgrund unterschiedlicher Positionen durch

$$\gamma \le 2\arctan\left(\frac{d_K}{2z_{min}}\right) = 2\arctan\left(\frac{r}{2}\right) \le r$$

beschränkt ist. Damit γ nicht größer als $\mu_G/2$ wird, ist hinreichend, daß gilt:

$$\frac{\mu_G}{2} = \sqrt{3\varepsilon} - 1, \text{ also } \frac{\mu_G}{2} + 1 = \sqrt{3\varepsilon}, \text{ d.h. } \left(\frac{\mu_G}{2} + 1\right)^2 = 3\varepsilon, \text{ bzw. } \varepsilon = \frac{1}{3}\left(\frac{\mu_G}{2} + 1\right)^2$$

Die relative Größe der Boxen beträgt dann ebenfalls höchstens $\frac{\mu_G}{2}$.

Bei der Berechnung des Projektionsfaktors für die Boxen wird nun die Summe der direkten Winkelabweichung $\mu_G/2$ von der repräsentativen Normalen n_G und der räumlich bedingten Abweichung $2\arctan\left(\frac{d_K}{2z_{min}}\right) \leq \frac{\mu_G}{2}$ gebildet und vom aus n_G berechneten Winkel abgezogen. Die maximale Winkelabweichung beträgt damit μ_G. Dadurch, daß für jede Winkelklasse der kleinstmögliche Betrachtungswinkel angenommen wird, der zu einem größtmöglichen Projektionsfaktor führt, wird garantiert, daß der Projektionsfaktor nie unterschätzt und somit in allen Boxen ein ausreichender Stichprobenumfang gewählt wird.

Erwarteter Aufwand

Um die Abweichung des Stichprobenumfangs bei der Verteilung mittels grober Winkelklassen vom Stichprobenumfang bei idealer Gleichverteilung zu bestimmen, ist eine worst-case Analyse nicht sehr hilfreich: Im schlimmsten Fall enthält die Szene große Flächen, die senkrecht zum Betrachter stehen und daher keine Stichprobenpunkte erhalten sollten. Wenn in der selben Winkelklasse aber auch Dreiecke vorhanden sind, die nicht absolut senkrecht zum Betrachter stehen, muß für alle ein maximaler Projektionsfaktor angenommen werden, in den der kleinste mögliche Betrachtungswinkel eingeht, welcher dann echt kleiner als $\pi/2$ ist. Dadurch werden Stichprobenpunkte auf Flächen verteilt, die eigentlich gar keine erhalten dürften, und der Stichprobenumfang der angenäherten Verteilung kann sich um einen beliebigen Faktor von dem bei exakter Verteilung notwendigen entfernen.

Abbildung 51 zeigt eine Beispielszene: Nach Winkelklassenbildung und räumlicher Unterteilung befindet sich in einer Box ein Stapel von Dreiecken, die senkrecht zum Betrachter stehen sowie ein weiteres Dreieck, das nicht senkrecht zum Betrachter steht. Die projizierte Fläche der Szene ist durch das letzte Dreieck gegeben, so daß nur eine kleine Stichprobe von Punkten genommen werden müßte. Dadurch, daß auch auf die senkrecht zum Beobachter stehenden Objekte Punkte verteilt werden, kann durch Erhöhen der Anzahl dieser Objekte der Aufwand unbeschränkt in die Höhe getrieben werden.

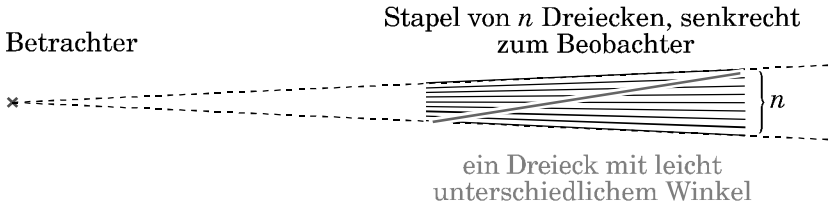


Abbildung 51: worst-case Szene für das grobe Winkelklassenverfahren

Die Existenz einer worst-case-Szene, die eine unbeschränkte Steigerung des Mehraufwandes erlaubt, macht eine weitere worst-case Untersuchung des groben Winkelklassenverfahrens uninteressant. Um dennoch eine Aussage über den Gewinn durch die Verwendung grober Winkelklassen machen zu können, wird nun wiederum eine „durchschnittliche“ Szene angenommen, in der alle Normalenrichtungen im projizierten Bild etwa gleich stark vertreten sind. Um den Mehraufwand für eine derartige Szene zu schätzen, wird wieder der Erwartungswert des exakten und des durch die Winkelklassen geschätzten Projektionsfaktors verglichen.

Seien die Normalen gleichverteilt auf der Einheitskugel und sei φ der Winkel, unter dem der Betrachter die Oberflächennormale sieht. Dann ist die Wahrscheinlichkeitsdichte der Zufallsvariablen, die der Normalen den Betrachtungswinkel φ zuordnet, nach Lemma 6 gegeben durch:

$$f(\varphi) = \frac{\sin\varphi}{2}$$

Nun sei angenommen, daß die Polarwinkel in $k \times 2k$ quadratische Bereiche der Kantenlänge $\lambda = \pi/k$ eingeteilt wurden. Aufgrund der Größe der Winkelklassen kann der Betrachtungswinkel damit bis auf $\mu_G \le 2\lambda$ genau geschätzt werden. Wird die räumliche Unterteilung wie oben beschrieben gewählt, erhält man eine zusätzliche Winkelungenauigkeit von $\mu_G/2$ durch den Einfluß der Position auf den Betrachtungswinkel. Insgesamt erhält man damit 3λ Genauigkeit, d.h. für jeden Punkt der Szene wird der Winkel zum Betrachter nie um mehr als den Wert 3λ unterschätzt. Der Verteilungsalgorithmus begrenzt außerdem Winkel unter 0° auf diesen Wert und verwirft alle Winkelklassen, die unter Berücksichtigung der Ungenauigkeit einen Wert von $\pi/2$ sicher überschreiten. Daher läßt sich die so produzierte Schätzung der projizierten Fläche wie folgt nach oben begrenzen:

$$\begin{aligned}
prj_{est} &\le \int_0^{\pi} \frac{1}{2}\sin(\varphi)\cdot cut_{[0,1]}\Big(\cos\big(cut_{[0,1]}(\varphi - 3\lambda)\big)\Big)d\varphi \\
&= \int_0^{3\lambda} \frac{1}{2}\sin(\varphi)d\varphi + \int_{3\lambda}^{\pi/2+3\lambda} \frac{1}{2}\sin(\varphi)\cdot\cos(\varphi - 3\lambda)d\varphi \\
&= \left[-\frac{1}{2}\cos\varphi\right]_{\varphi=0}^{\varphi=3\lambda} + \left[-\frac{1}{8}\cos(2\varphi - 3\lambda) + \frac{1}{4}\varphi\sin(3\lambda)\right]_{\varphi=3\lambda}^{\varphi=\pi/2+3\lambda} \\
&= -\frac{1}{2}\cos 3\lambda + \frac{1}{2}\cos 0 - \frac{1}{8}\cos\big(2(\pi/2+3\lambda) - 3\lambda\big) + \frac{1}{4}(\pi/2+3\lambda)\sin 3\lambda + \frac{1}{8}\cos(6\lambda - 3\lambda) - \frac{1}{4}3\lambda\sin 3\lambda \\
&= \frac{1}{2} - \frac{1}{2}\cos 3\lambda - \frac{1}{8}\cos(\pi + 3\lambda) + \left(\frac{\pi}{8} + \frac{3}{4}\lambda\right)\sin 3\lambda + \frac{1}{8}\cos 3\lambda - \frac{3\lambda}{4}\sin 3\lambda \\
&= \frac{1}{2} - \frac{1}{2}\cos 3\lambda + \frac{1}{4}\cos 3\lambda + \frac{\pi}{8}\sin 3\lambda \\
&= \frac{1}{2}\left(1 - \frac{1}{2}\cos\frac{3\pi}{k} + \frac{\pi}{4}\sin\frac{3\pi}{k}\right)
\end{aligned}$$

Da der erwartete Wert für den tatsächlichen Projektionsfaktor $1/4$ beträgt, erhält man eine erwartete Überschätzung von

$$2 - \cos\frac{3\pi}{k} + \frac{\pi}{2}\sin\frac{3\pi}{k} \doteq 1 + \frac{3\pi^2}{2k} \quad \text{(für große Werte von k)}$$

In der Herleitung bleibt noch die Korrektheit der Stammfunktion für das zweite Integral zu zeigen, die mit Hilfe des Computeralgebrasystems MAPLE bestimmt wurde. Durch Differenzieren erhält man hier:

$$\begin{aligned}
\frac{d}{d\varphi}\left(-\frac{1}{8}\cos(2\varphi - 3\lambda) + \frac{1}{4}\varphi\sin(3\lambda)\right) &= \frac{1}{4}\sin(2\varphi - 3\lambda) + \frac{1}{4}\sin(3\lambda) \\
&= \frac{1}{8i}\left(e^{i(2\varphi-3\lambda)} - e^{-i(2\varphi-3\lambda)}\right) + \frac{1}{8i}\left(e^{3i\lambda} - e^{-3i\lambda}\right) \\
&= -\frac{ie^{2i\varphi}}{8e^{3i\lambda}} + \frac{ie^{3i\lambda}}{8e^{2i\varphi}} - \frac{ie^{3i\lambda}}{8} + \frac{i}{8e^{3i\lambda}}
\end{aligned}$$

$$= \frac{1}{4i}\left(e^{i\varphi} - e^{-i\varphi}\right)\frac{1}{2}\left(e^{i(\varphi-3\lambda)} + e^{-i(\varphi-3\lambda)}\right)$$
$$= \frac{1}{2}\sin(\varphi)\cos(\varphi - 3\lambda)$$

Dabei wurde die komplexe Darstellung des Sinus und Kosinus als

$$\sin x = \frac{1}{2i}\left(e^{ix} - e^{-ix}\right) \quad \text{und} \quad \cos x = \frac{1}{2}\left(e^{ix} + e^{-ix}\right)$$

benutzt.

Tabelle 2 zeigt die Werte für diese Schranke bei verschiedenen Werten von λ. Als „Mehraufwand“ wurde die zuvor hergeleitete Schranke geteilt durch den Erwartungswert im exakten Fall (1/4) angegeben. Abbildung 52 zeigt eine grafische Darstellung.

K	6	8	12	16	32	64
λ	0,52	0,39	0,26	0,20	0,10	0,05
Winkelklassen	72	128	288	512	2048	8192
Mehraufwand	3,57	3,07	2,40	2,04	1,50	1,24

Tabelle 2: obere Schranken für den erwarteten Mehraufwand bei verschieden vielen Winkelklassen

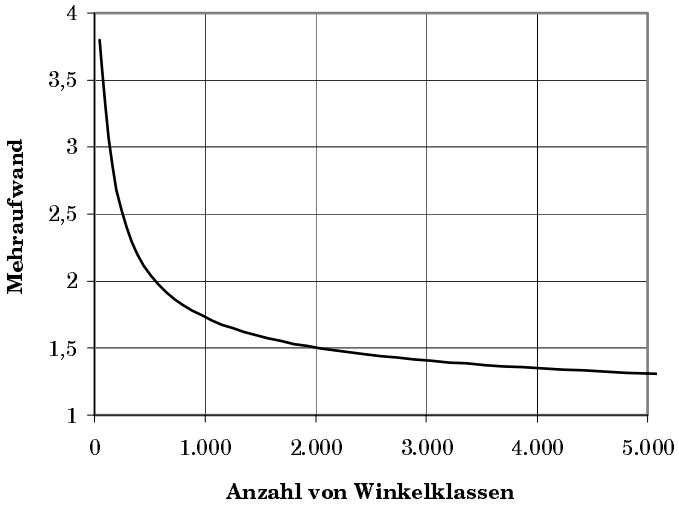


Abbildung 52: die obere Schranke für den erwarteten Mehraufwand in Abhängigkeit der Anzahl von Winkelklassen

Der Aufwand bei der Einteilung der Szene in Gruppen nimmt mit steigender Genauigkeit zu. Hier kommen zwei Faktoren zusammen: Zum einen wächst der Aufwand dadurch, daß $2k^2$ Winkelklassen gebildet werden, quadratisch mit k. Die Einschränkung der Boxengröße, die notwendig ist, um die Winkelabweichung durch die verschiedenen Positionen innerhalb der Box zu beschränken, erhöht die Anzahl von Boxen zusätzlich. Um die Berechnung der Boxenanzahl zu vereinfachen, sei angenommen, daß der Algorithmus wie folgt vorgeht: Zuerst werden mit Algorithmus 9 für ein festes ε Boxen ausgewählt, so daß die Abweichung des Projektionsfaktors

aufgrund der Tiefe gering ist. Danach werden diese Boxen solange aufgeteilt, bis das Verhältnis von Durchmesser zu minimaler Tiefe kleiner als $\mu_G/2=\lambda$ ist. Da Boxen in Zweierpotenzen geteilt werden können, ist die Anzahl von Boxen bis auf einen Faktor $8=2^3$ genau wählbar. Am Rand der Sichtpyramide entstehen u.U. Boxen, die vollständig außerhalb des Sichtfeldes liegen und die verworfen werden könnten. Da aber $\Theta(k^3)$ Boxen im Inneren der Sichtpyramide liegen und nur $\Theta(k^2)$ den Rand schneiden, fällt dieser Fehler asymptotisch nicht ins Gewicht. Die Kantenlänge der Boxen schrumpft mit $\Theta(\lambda)=\Theta(1/k)$, also werden für die Unterteilung $\Theta(k^3)$ Boxen benötigt, da das selbe Volumen wie zuvor überdeckt werden muß. Insgesamt wächst die Anzahl von Boxen bei schrumpfender Größe der Winkelklassen, also mit $\Theta(k^5)$. Damit erhält man:

Satz 9: *Aufwand und Stichprobenumfang beim Winkelklassenverfahren*

Es sei ein Punkt auf einer Oberfläche an fester Position unter einer festen Projektion gegeben. Der Normalvektor der Oberfläche sei zufällig gleichverteilt aus der Einheitskugel gewählt.

Dann ist das Verhältnis zwischen der erwarteten Schätzung der projizierten Fläche durch das oben beschriebene Winkelklassenverfahren und dem Erwartungswert des Projektionsfaktors höchstens

$$2-\cos\left(\frac{3\pi}{k}\right)+\frac{1}{2}\pi\sin\left(\frac{3\pi}{k}\right),$$

wenn die Polarkoordinaten in mindestens $k\times 2k$ gleich große, quadratische Bereiche eingeteilt werden. Für diese Einteilung ist die Bearbeitung von $\Theta(k^5)$ Boxen notwendig.

Insbesondere zeigt die Herleitung noch:

Lemma 7: *Wachstum der Boxenanzahl mit dem relativen Durchmesser*

Sei r der relative Durchmesser der Boxen im Tiefenunterteilungsbaum. Dann wächst der Aufwand zur Unterteilung mit $\Theta(r^{-3})$.

Approximation der Verteilung mit konstanter Fehlerschranke

Aus den bisherigen Ergebnissen kann eine Strategie abgeleitet werden, um mit Winkelklassen eine ε-Approximation der Verteilung mit konstantem ε zu erreichen. Sei β_{max} der größte Winkel echt unter $\pi/2$ zwischen dem Betrachtungsvektor und der Normalen eines Punktes. Da die Szene aus endlich vielen ebenen Dreiecken besteht, muß ein solcher Wert stets existieren, falls die Ansicht nicht nur abgewandte oder gar keine Oberflächen der Szene zeigt, was ein einfacher Sonderfall wäre. Bezeichne im folgenden mit $\omega := \pi/2 - \beta_{max}$ die *Winkelauflösung* einer Ansicht einer Szene.

Dann kann das gerade beschriebene Verfahren mit $\lambda = \omega\cdot(\varepsilon - 1)/6$ eingesetzt werden. Die maximale Winkelabweichung innerhalb jeder Box beträgt dann $3\lambda = \omega\cdot(\varepsilon - 1)/2$. Nach Lemma 5 ist hinreichend, daß

$$\beta_{max}-\beta_{min}\le(\varepsilon-1)\cos\beta_{max}$$

ist, damit der Projektionsfaktor bezüglich des Blickwinkels nicht stärker als ε variiert. Durch die Beschränkung der Winkelklassen gilt nun:

$$\beta_{max} - \beta_{min} \le 3\lambda = (\varepsilon - 1)\frac{\omega}{2} \overset{(*)}{\le} (\varepsilon - 1)\cos\left(\frac{\pi}{2} - \omega\right) = (\varepsilon - 1)\cos\beta_{max}$$

Die Ungleichung (*) erhält man, da die Kosinusfunktion im Intervall $[0, \pi/2]$ stets oberhalb der Geraden zu $x \mapsto \pi/2 - x/2$ verläuft (siehe Abbildung 53).

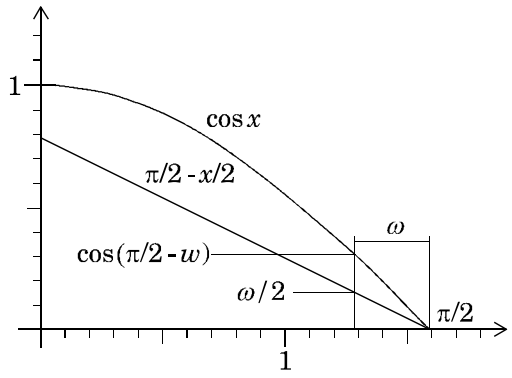


Abbildung 53: Abschätzung des Kosinus

Nicht berücksichtigt wurde dabei bisher der Verzerrungsfaktor. Nach Lemma 5 ist hinreichend, daß

$$r \le (\varepsilon - 1)\cos\arctan\left(\frac{D}{2d}\right)$$

ist, damit dieser auf einen Wert von ε beschränkt bleibt. Dies ist eine gegenüber der Szene konstante obere Schranke für die relative Boxengröße, die für jede Box geprüft werden kann, so daß diese ggf. weiter unterteilt werden kann.

Um nun eine ε_0-Approximation der Verteilung bestimmen zu können, muß in dem zuvor beschriebenen Verfahren der Wert $\varepsilon := \sqrt[3]{\varepsilon_0}$ für die räumliche Abweichung, die Beschränkung der Winkelabweichung und die Beschränkung des Verzerrungsfaktors eingesetzt werden. Da der Projektionsfaktor ein Produkt aus diesen drei unabhängigen Faktoren ist, bleibt so die Gesamtabweichung stets unter ε.

Der Aufwand bei diesem Vorgehen wird durch die Anzahl der Winkelklassen und die Tiefe der räumlichen Unterteilung bestimmt: Es gibt $\Theta(k^2) = \Theta(\omega^{-2})$ Winkelklassen und in jeder muß eine räumliche Unterteilung vorgenommen werden, die sowohl die Winkelabweichung durch unterschiedliche Positionen im Raum, wie auch die Abweichung des Projektionsfaktors durch unterschiedliche Tiefe begrenzen. Für die erste Bedingung ist eine Mindestunterteilungstiefe notwendig, die zu $\Theta(k^3) = \Theta(\omega^3)$ Boxen in jeder Winkelklasse führt, und für die zweite Bedingung ist eine Mindestunterteilungstiefe notwendig, die $\Theta(\log \tau)$ Boxen erfordert. Man erhält daher insgesamt:

Satz 10: *Aufwand für eine ε-Approximation mit dem Winkelklassenverfahren*

Um mit dem oben beschriebenen Verfahren eine nach einer ε-Approximation der in Definition 8 definierten Verteilung verteilte Stichprobe zu bestimmen, ist die Bearbeitung von $\Theta(\omega^{-2} \cdot T)$ mit $T = \max(\omega^{-3}, \log \tau)$, also insbesondere von $\Omega(\omega^{-5})$ Boxen notwendig.

Dieses Ergebnis ist ungünstig, da der Wert ω beliebig klein sein kann. Für eine praktische Anwendung ist daher eine garantierte ε-Approximation der Verteilung, zumindest mit diesem Verfahren, nicht interessant. Der hohe Aufwand wird hier zum einen durch die große Anzahl Winkelklassen bedingt und zum anderen durch die notwendige feine räumliche Unterteilung, um eine Abweichung des Winkels durch unterschiedliche Positionen zu beschränken. Für das erste Problem sind relativ einfache Verbesserungen möglich, wie im nächsten Abschnitt gezeigt wird. Das zweite Problem ist allerdings deutlich schwieriger: Der maximale Winkel, um den sich Punkte im Raum unterscheiden dürfen, fällt mit $\cos \beta_{max} \approx \pi/2 - \beta_{max}$, also linear mit dem Winkel zur Oberfläche. Bereiche, die unter einem kleinen Winkel gesehen werden, müssen also in sehr kleine Kegel mit Ursprung im Projektionszentrum eingeschlossen werden, um auch im worstcase eine grobe Fehlschätzung des Projektionsfaktors zu vermeiden. Es erscheint naheliegend, daß mit einer uniformen, blickpunktunabhängigen Raumaufteilungsstrategie stets mit einem Aufwand in der Größenordnung von $\Omega(\omega^{-3})$ Boxen für diese Einschränkung zu rechnen ist.

3.3.5.3 Zwei-Ebenen-Baum

Die uniforme Aufteilung der Oberflächenorientierungen beim Winkelklassenverfahren führt zu unnötigen Performanceverlusten. Um den gleichen relativen Fehler zu erhalten, können für Bereiche, in denen die Oberflächennormalen unter einem kleinen Winkel (d.h. mit einem großen Kosinuswert im Projektionsfaktor) gesehen werden, größere Abweichungen toleriert werden als für Bereiche, in denen der Betrachtungswinkel nahe an $\pi/2$ liegt. Es erscheint daher wünschenswert, die Winkelklassen abhängig vom Blickwinkel, unter dem die gesamte Szene betrachtet wird, so zu wählen, daß Oberflächen, die eher senkrecht zur Blickrichtung stehen, in größere Winkelklassen fallen, als solche, die nahezu parallel zur Blickrichtung verlaufen. Um dynamisch eine entsprechende Aufteilung auswählen zu können, wird ein Quadtree auf den in Polarkoordinaten transformierten Normalvektoren aufgebaut und für jeden Knoten in diesem Baum ein räumlicher Unterteilungsbaum erstellt (Abbildung 54 zeigt den Aufbau schematisch).

Man betrachte die Menge aller Normalen der Dreiecke der Szene, transformiert in Polarkoordinaten. Auf dieser Menge wird ein Quadtree aufgebaut, dessen Wurzelknoten w das Rechteck $Q(w) = [0, 2\pi] \times [0, \pi]$ zugeordnet ist. Jedem Blattknoten k im Baum werden diejenigen Dreiecke zugeordnet, deren Normalvektor in Polarkoordinaten im Rechteck $Q(k)$ liegt. Jeder Knoten wird solange weiter unterteilt, bis er nur noch Dreiecke mit höchstens einem verschiedenen Normalvektor enthält oder bis der Knoten eine vorgegebene Maximaltiefe h_{max} erreicht hat[16].

Danach wird für jeden Knoten k im Baum ein Raumunterteilungsbaum für die Dreiecke $Obj(k)$ aufgebaut und mit diesem Knoten gespeichert. Eine solche Datenstruktur wird im folgenden als

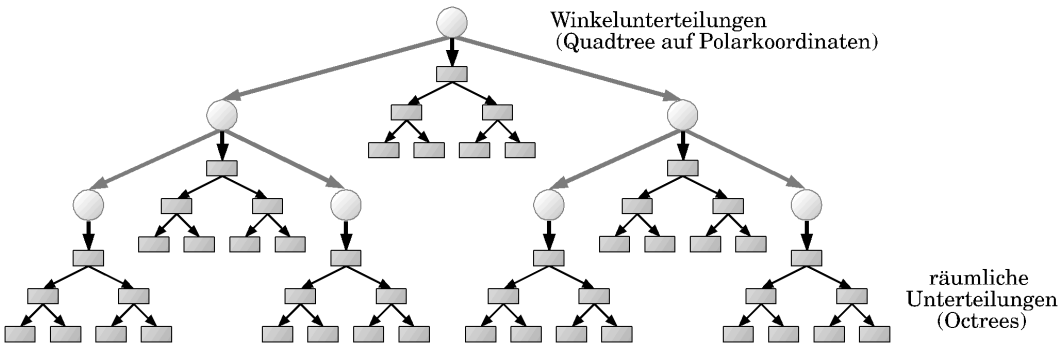


Abbildung 54: Zwei-Ebenen-Baum (schematisch)

[16] Die Tiefe sei hier so angegeben, daß die Wurzel die Tiefe 1 hat.

Zwei-Ebenen-Verteilungsbaum bezeichnet, der übergeordnete Baum wird dabei als *Winkelbaum* bezeichnet. Diese Datenstruktur ähnelt dem Range Tree [de Berg et al. 97] und man erhält ähnliche Resultate zur Konstruktionszeit und zum Speicherbedarf:

Satz 11: *Konstruktion des Zwei-Ebenen-Verteilungsbaumes*

Ein *Zwei-Ebenen-Verteilungsbaum* benötigt $O(h \cdot n)$ Speicher und kann in $O(h \cdot n \cdot \log n)$ Zeit aufgebaut werden, wenn n die Anzahl von Dreiecken der Szene und h die maximale Tiefe des Winkelbaumes ist. h ist auf $O(\log \omega^{-1})$ beschränkt, wenn ω der kleinste Winkel zwischen zwei Normalen der Szene ist.

Beweis: Alle Kindknoten eines Knotens im Winkelbaum beschreiben disjunkte Mengen von Dreiecken. Daher wird jedes Dreieck auf jeder Ebene im Baum maximal einmal gespeichert und man benötigt $O(h \cdot n)$ Speicherplatz. Der Winkelbaum kann mit der selben Technik wie der Raumunterteilungsbaum in Zeit $O(n \cdot \log n)$ aufgebaut werden. In jeder Ebene werden Teilbäume der Größe $n_1 ... n_j$ in Zeit

$$O\left(\sum_{i=1}^{k} n_i \log n_i\right) \subseteq O\left(\sum_{i=1}^{k} n_i \log n\right) = O\left(n \log n\right)$$

aufgebaut, so daß insgesamt ein Zeitbedarf von $O(h \cdot n \cdot \log n)$ entsteht.

Da die Boxengröße sich bei jeder Unterteilung halbiert, können nach $O(\log \omega^{-1})$ Schritten keine zwei Punkte bzw. Normalen in Polarkoordinaten mehr in einer Box liegen. ■

Bestimmung der Boxen

Die Strategie zur Auswahl der Boxen aus dem Zwei-Ebenen-Verteilungsbaum verläuft ähnlich wie beim Winkelklassenverfahren. Als erster Schritt werden hier zusätzlich Boxen geeigneter Größe aus dem Winkelbaum ausgewählt, die die gesamte Szene enthalten. Für diese wird die räumliche Unterteilungstiefe festgelegt und dann wie gehabt in den räumlichen Unterteilungsbäumen aller ausgewählter Winkelboxen solche Boxen bestimmt, die die räumliche Mindestunterteilung einhalten und in denen der Projektionsfaktor eingeschränkt auf die Tiefe nicht zu stark variiert.

Zur Auswahl der Boxen aus dem Winkelbaum wird die aktuelle Blickrichtung als zweidimensionaler Punkt b in Polarkoordinaten dargestellt, und der Baum wird beginnend bei der Wurzel rekursiv traversiert: Falls der aktuelle Knoten k ein Blattknoten ist, wird er in die Liste der ausgewählten Knoten übernommen. Anderenfalls sei β_{min} der kleinste Winkel zwischen b und einem Punkt aus $B(k)$ und β_{max} der größte solche Winkel. Von β_{min} wird dann ein Wert $\delta(k)$ abgezogen, der später die Winkelabweichung durch verschiedene Positionen angeben wird. Falls β_{min} dadurch kleiner als null wird, so wird β_{min} auf null gesetzt. Entsprechend wird β_{max} um $\delta(k)$ erhöht, wobei der Wert π nicht überschritten wird.

Falls β_{min} bereits größer als $\pi/2$ ist, wird der Knoten ignoriert. Falls nur β_{max} größer als $\pi/2$ ist, wird die Unterteilung fortgesetzt. Anderenfalls wird die Unterteilung genau dann gestoppt und der Knoten in die Liste der ausgewählten Knoten aufgenommen, wenn

$$\frac{\cos\beta_{min}}{\cos\beta_{max}} \leq \varepsilon$$

ist. Um die Winkelabweichung auf Grund unterschiedlicher Positionen zu beschränken, wird $\delta(k)$ auf den Wert des diagonalen Durchmessers l der Boxen im Winkelbereich festgelegt. Dazu wird die relative Boxengröße bei der räumlichen Unterteilung auf den Wert l eingeschränkt, da dann nach Lemma 5 auch die Winkelabweichung den Wert l nicht überschreitet.

Nun soll die Anzahl von Boxen abgeschätzt werden, die diese Strategie auswählt. Dafür wird im folgenden nur eine obere und eine untere Schranke angegeben. Im Rahmen dieser Arbeit war es nicht mehr möglich eine genauere Analyse durchzuführen.

Zur Vereinfachung sei zunächst angenommen, daß die Szene so viele verschiedene Orientierungen enthält, daß der Algorithmus zur Konstruktion stets beim Wert h_{max} die weitere Unterteilung abbricht. Dies ist quasi der ungünstigste Fall. Die minimale Kantenlänge einer Box im Winkelbaum ist dann im Winkelbereich $\pi \cdot 2^{-(h_{max}-1)}$, die maximale Kantenlänge einer Box mit minimaler Kantenlänge ist zweimal so groß, da die Boxen, aufgrund des asymmetrischen Parameterbereiches der Polarkoordinaten, alle ein Breiten/Höhenverhältnis von 2 aufweisen. Der diagonale Durchmesser einer kleinsten Box ist daher

$$u := \sqrt{5}\pi 2^{1-h_{max}} \in O\left(2^{-h_{max}}\right).$$

Um die Bedingung

$$\frac{\cos\beta_{min}}{\cos\beta_{max}} = \varepsilon$$

einzuhalten, muß gelten, daß

$$\cos\beta_{max} = \frac{1}{\varepsilon}\cos\beta_{min}, \quad \text{also} \quad \beta_{max} = \arccos\left(\frac{1}{\varepsilon}\cos\beta_{min}\right)$$

ist. Damit läßt sich der Winkelbereich in Gebiete einteilen. Bezeichnet man mit β_i den Winkel, der Gebiet i von Gebiet $i+1$ trennt, so erhält man mit $\beta_0 = 0$ induktiv:

$$\beta_i = \arccos\left(\varepsilon^{-i}\right) \approx \varepsilon^{-i} \quad \text{(für große Werte von } i\text{)}$$

D.h., daß β_i gegen $\pi/2$ konvergiert, wobei die Abstände zwischen den Werten immer kleiner werden. Abbildung 55 zeigt die Einteilung des Winkelbereiches für ein Beispiel. Wie man sieht, entstehen in der Nähe der Blickrichtung größere und in der Nähe von einem Winkel von $\pi/2$ (im Bild als Hilfslinie eingezeichnet) zunehmend kleinere Bereiche. Das Diagramm wurde mit dem Computeralgebrasystem MAPLE durch numerisches Lösen der Gleichungen $\cos(b-\lambda)\cdot\cos(b-\varphi) = \cos(\beta_i)$ erstellt.

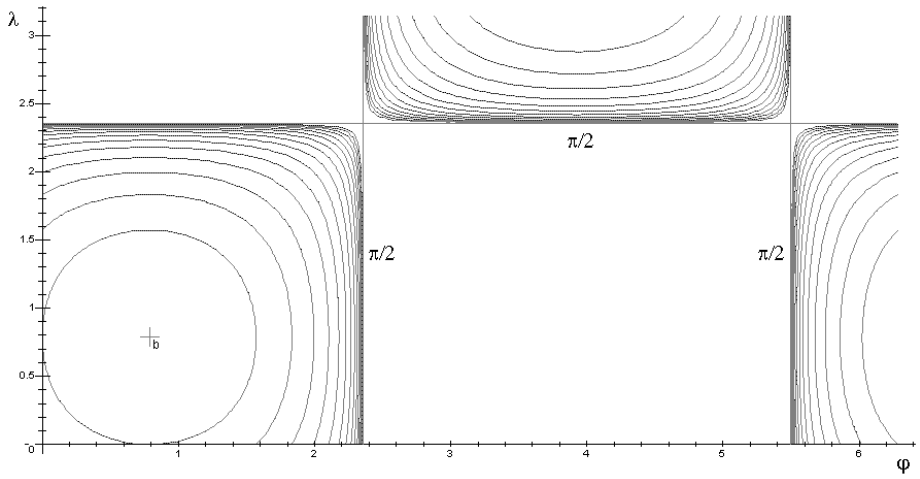


Abbildung 55: Winkelbereiche für $b = (\pi/4, \pi/4)$ in Polarkoordinaten

Betrachtet man nun die Einteilung des Gebietes in Boxen, so entstehen in der Nähe von $\pi/2$ die kleinsten Boxen. Die minimale Größe ist auf u beschränkt, da keine kleineren Boxen zur Verfügung stehen. Die Kurve im Winkelbereich, die alle Normalenwinkel beschreibt, die senkrecht zur Blickrichtung stehen, hat für alle Blickrichtungen eine Länge von $\Theta(1)$. Daher werden mindestens $\Omega(u^{-1}) = \Omega(2^{h_{max}})$ Boxen der Größe u ausgewählt, die den Rand schneiden. Andererseits wird der Algorithmus nie mehr als alle Blattknoten auswählen, so daß $O(u^{-2}) = O(4^{h_{max}})$ eine obere Schranke für die Anzahl der Boxen darstellt, die dem Ergebnis im Falle der Winkelklassenmethode aus dem vorangegangenen Abschnitt entspricht.

Es ist zu vermuten, daß man (für gemessen an ε relativ kleines u) auch eine Schranke von $\Theta(u^{-1})$ erhält, da die Größe der Boxen mit wachsendem Abstand zum „Rand", d.h. zur $\pi/2$ Kurve, exponentiell größer werden kann. Ein Beweis dieser Vermutung war im Rahmen dieser Arbeit nicht mehr möglich.

Berücksichtigt man nun auch die räumliche Aufteilung, so entstehen in den $\Omega(u^{-1})$ Boxen am „Rand" nochmals $\Theta(u^{-3})$ Boxen durch die räumliche Unterteilung, da auch die relative räumliche Boxengröße auf den Wert u festgelegt wird, was, wie bereits im vorigen Abschnitt dargestellt, $\Theta(u^{-3})$ Boxen erfordert. Insgesamt werden also $\Omega(u^{-4})$ Boxen erzeugt. Da andererseits das Verfahren im schlimmsten Fall nicht mehr Boxen auswählen kann als das Winkelklassenverfahren, da dort die Bedingungen an die Boxen strenger sind, können auch nicht mehr als $O(u^{-5})$ Boxen zur Einschränkung der Winkelabweichung entstehen. Berücksichtigt man noch den Einfluß der Tiefendynamik, die unabhängig von der Winkeleinteilung den Aufwand der räumlichen Aufteilung erhöhen kann, so erhält man als vollständige Schranken $\Omega(u^{-1} \cdot T)$ und $O(u^{-2} \cdot T)$ ausgewählte Boxen mit $T = \max(u^{-3}, \log \tau)$.

Die Genauigkeit der Verteilung und insbesondere die erwartete Überschätzung des Projektionsfaktors, hängt hier sowohl vom Parameter ε wie auch von u (bzw. der maximalen Tiefe h_{max} des Winkelbaumes) ab. Durch eine geeignete Wahl von ε läßt sich mindestens die Genauigkeit des Winkelklassenverfahrens erreichen, dazu genügt z.B. ε auf 1 zu setzen und $u := 1/k$ zu wählen. Ein genauerer Trade-Off zwischen den beiden Parametern konnte im Rahmen dieser Arbeit nicht bestimmt werden.

Es ist allerdings klar, daß auch mit der hier vorgestellten Strategie keine ε-Approximation der in Definition 8 definierten Verteilung bestimmt werden kann, ohne einen sehr großen Aufwand in Kauf nehmen zu müssen: Es ist auf jeden Fall notwendig, in der Nähe vom Betrachtungswin-

kel $\pi/2$ die räumliche Abweichung so stark einzuschränken, daß der größte Betrachtungswinkel ω unter $\pi/2$ im ganzen Bild noch vom Wert $\pi/2$ selbst unterschieden werden kann, denn für eine Gruppe von Objekten, die einen Winkel von $\pi/2$ und noch mindestens einen Winkel $< \pi/2$ enthalten können, kann eine unbeschränkte Abweichung des Projektionsfaktors nicht ausgeschlossen werden. Eine worst-case Szene wäre auch hier wieder ein „Stapel" dicht beieinander liegender Flächen senkrecht zum Beobachter sowie eine weitere Fläche, die einen etwas kleineren Betrachtungswinkel aufweist. Die räumliche Unterteilung sowie die Winkelunterteilung müssen also fein genug gewählt werden, um Winkelunterschiede deutlich unter ω zu liefern (abhängig von ε). Nach den bisherigen Ergebnissen steht bereits fest, daß für eine solche Unterteilung $\Omega(\omega^{-1} \cdot \omega^{-3}) = \Omega(\omega^{-4})$ Boxen notwendig sind, was wohl in den meisten Fällen nicht praktikabel wäre.

3.3.6 Erweiterungen

3.3.6.1 Allgemeine Szenen

Für die Konstruktion des Raumunterteilungsbaumes wurde angenommen, daß die Dreiecke der Szene als unendlich kleine Punkte aufgefaßt werden können. Sobald ausgedehnte Dreiecke zugelassen werden, können Probleme auftreten: Alle Boxen des Baumes werden durch achsenparallele Ebenen voneinander getrennt. Es ist aber nicht immer möglich, zwei beliebige Dreiecke durch eine achsenparallele Ebene voneinander zu trennen. Daher kann ein Baum in der angenommenen Form, mit einem Dreieck pro Blattknoten, nicht immer erzeugt werden, für stetig aneinandergefügte Dreiecksoberflächen ist dies sogar eher unwahrscheinlich.

Die Strategie im allgemeinen Fall

Um das Problem zu vermeiden, daß Dreiecke, die die Grenzen der Boxen schneiden, die weitere Unterteilung verhindern, wird um die Box jedes Knotens k eine Toleranzzone eingerichtet: Auch Objekte, die in einer Box $Q'(k)$ liegen, die um bis zu einem Faktor $c > 1$ größer ist als die ursprüngliche Box $Q(k)$, können dem Knoten zugeordnet werden. Falls dies nicht eindeutig ist, wird eine Pattregelung benutzt, es wird z.B. diejenige Box gewählt, deren Mittelpunkt näher am Mittelpunkt des Dreiecks liegt. Falls sich ein Dreieck auch dann nicht zuordnen läßt, so muß dieses groß sein, d.h. es muß in einer Dimension mindestens den Durchmesser $c-1$ aufweisen. Im Falle $c = 1{,}5$ wäre dies z.B. die halbe Boxengröße. Diese Dreiecke werden dann nicht mehr an Kindknoten weiterverteilt, sondern einem inneren Knoten zugeordnet. Diese Strategie isoliert die großen Dreiecke von den kleineren, die sich noch aufteilen lassen.

Wenn der Algorithmus zur Boxenauswahl (Algorithmus 9) dann auf einen inneren Knoten trifft, dem Dreiecke zugeordnet sind, in denen der Projektionsfaktor noch zu stark variiert, so werden diese Dreiecke in einer speziellen Liste gespeichert und später bei der Bilderzeugung gesondert behandelt.

Dadurch, daß die Boxen nun nicht mehr disjunkt sein müssen, kann sich der Aufwand zur Bestimmung der Boxen natürlich erhöhen: Dadurch, daß die Box $Q'(k)$ maximal eine um einen Faktor c größere Kantenlänge als die ursprüngliche Box $Q(k)$ besitzt, muß sie u.U. im Gegensatz zur nicht erweiterten Box noch weiter aufgeteilt werden. Um diese dann soweit zu unterteilen, daß die Teilboxen wieder so groß wie die ursprünglichen Boxen sind, sind $O(\log c)$ Unterteilungen notwendig, die $O(c^3)$ zusätzliche Boxen produzieren. Da für den Faktor c allerdings keine

größeren Werte als 2 sinnvoll sind (bei einer Abweichung um den Faktor 2 könnte man auch direkt die nächst größere Box verwenden), bleibt der Mehraufwand insgesamt konstant.

Somit wird nun nur noch eine Strategie benötigt, um mit „zu großen" Dreiecken umzugehen. Dies sind alle Dreiecke, die in Knoten gespeichert wurden, die vom Boxenauswahlalgorithmus (Algorithmus 9) noch weiter unterteilt werden mußten. Andere Dreiecke, die in inneren Knoten gespeichert sind, die nicht unterteilt werden müssen, verursachen dagegen keine Probleme. Sie werden über die Verteilungslisten automatisch in die Stichprobenauswahl mit einbezogen.

Sinnvoll erscheint z.B., die Dreiecke direkt mit dem gewöhnlichem z-Buffer Verfahren darzustellen, da dieses für große Dreiecke ohnehin effizienter ist. Falls der Rekonstruktionsalgorithmus dies nicht zuläßt, können auch die Stichprobenpunkte mit Hilfe von Algorithmus 7 ausgewählt werden. Beide Ausweichstrategien können allerdings den Aufwand stark erhöhen, da in ungünstigen Fällen viele solcher „großen" Dreiecke auftreten können, die alle einzeln bearbeitet werden müssen. Eine ungünstige Szene besteht auch hier z.B. aus vielen Dreiecken mit großem Flächeninhalt, die fast senkrecht zum Beobachter stehen. Obwohl die projizierte Fläche relativ klein ist, ist dann unter Umständen die Bearbeitung einer sehr großen Zahl von Dreiekken notwendig. In praktischen Anwendungen sind solche Fälle wohl nur sehr selten zu erwarten. Bei den in dieser Arbeit durchgeführten Experimenten traten keine Probleme mit der hier vorgeschlagenen Strategie auf.

Eine alternative Strategie

Eine anderer naheliegender Ansatz, um alle Dreiecke voneinander zu trennen, bestünde darin, die Dreiecke an den Teilungsebenen durchzuschneiden. Die Anzahl von Schnitten könnte dadurch reduziert werden, daß beliebig orientierte Teilungsebenen zugelassen werden[17]. Eine Datenstruktur zur Unterteilung von Dreiecksszenen im $\mathbb{R}^3$, der *BSP-Baum*, wird in [de Berg et al. 97] (S. 249ff) beschrieben. Es wird gezeigt, daß durch das Zerschneiden von n Dreiecken im schlimmsten Fall $\Theta(n^2)$ Fragmente entstehen können; der Speicherbedarf dieser Datenstruktur wäre damit deutlich zu hoch. Es wird aber darauf hingewiesen, daß ein derart schlechtes Verhalten in vielen praktischen Anwendungsbeispielen nicht zu beobachten ist. Eine Szene mit vielen „zu großen" Dreiecken ließe sich aber auch damit, also mit einer vollständigen Trennung der Dreiecke, nicht immer effizient behandeln. Hier läßt sich ein Beispiel eines „Stapels" von vielen, (fast) senkrecht gesehenen Dreiecken mit größerer Ausdehnung anführen. Obwohl sich diese vollständig trennen ließen, müßten sie immer noch einzeln gesondert behandelt werden, weil ihre Tiefe u.U. noch zu groß für die Stichprobenverteilung ist.

3.3.6.2 Instantiierungen von Szenenteilen

Der Speicherplatzbedarf für Szenen großer Komplexität entwickelt sich im Gebiet der dreidimensionalen Computergrafik zunehmend zu einem Problem. Insbesondere wenn Szenen, wie hier vorausgesetzt wurde, explizit aus einzelnen Dreiecken modelliert werden, kann die Größe eines solchen Modells beachtliche Ausmaße annehmen. Eine Szene mit einer Million Dreiecken benötigt mindestens Speicher im zweistelligen Megabytebereich. Andererseits ist erst dies in etwa die Komplexität von Szenen, die sich mit heutiger Hardware nicht mehr in Echtzeit handhaben läßt. Bei der schnellen Entwicklung der Hardware in diesem Bereich ist sogar damit zu rechnen, daß sich diese Grenze in allernächster Zeit noch deutlich weiter nach oben verschiebt.

---

[17] Solange die Zellen nicht zu asymmetrisch werden und auch die Aufteilung relativ symmetrisch erfolgt, ließen sich sicherlich die bisherigen Resultate zur benötigten Anzahl von Zellen übertragen.

Das randomisierte z-Buffer Verfahren hat einen beträchtlichen Grundaufwand, der allerdings nur noch schwach mit einer zunehmenden Szenenkomplexität steigt. In der Praxis lohnt sich das Verfahren daher auf heutiger Hardware erst für Szenen, die mehr als eine Million Dreiecke enthalten (siehe Kapitel 5). Große Gewinne sind erst bei Szenen zu erzielen, die ein Vielfaches dieser Komplexität aufweisen und daher bei expliziter Speicherung aller Dreiecke enorme Mengen an Hauptspeicher benötigen würden.

Das Verfahren macht also überhaupt nur Sinn, wenn eine effizientere Kodierung der Szene verwendet werden kann. Dazu kommen vor allem zwei Strategien in Betracht, die im Bereich der Modellierung häufig eingesetzt werden:

- Verwendung höherer Primitive statt Dreiecken (z.B. Splines u.ä.). Im Extremfall: Prozedurale Modellierung, d.h. Objekte werden durch Eingabedaten für u.U. komplexe Algorithmen beschrieben.
- Instantiierung von Szenenteilen (wie im Abschnitt 1.4.1.2 als „High-Level-Modell" beschrieben): ähnliche Objekte werden nur einmal gespeichert.

Die erste Variante wurde hier nicht weiter untersucht, da dazu für jede Klasse von Primitiven ein neuer Verteilungsalgorithmus sowie eine Abschätzung der projizierten Fläche entwickelt werden müßte. Für die zweite Methode läßt sich jedoch ein universelles Schema implementieren.

Sei nun ein Szenengraph, wie in Abschnitt 1.4.1.2 beschrieben, gegeben (d.h. manuell vom Modellierer der Szene erstellt), der aus Grundobjekten, Listen und Referenzen besteht. Die Referenzen in diesem Graphen erzeugen zusätzliche Instanzen von einer Gruppe von Objekten, die auch ihrerseits wieder Referenzen enthalten kann. Jeder Knoten ist mit einer homogenen 4×4 Transformationsmatrix versehen, die den relativen Unterschied zur Transformation der übergeordneten Knoten beschreibt. Dabei muß gefordert werden, daß alle Transformationen sich als Hintereinanderausführung von Rotationen, Spiegelungen, Translationen und (für alle Achsen gleichförmigen, nicht singulären) Skalierungen darstellen lassen. Das bedeutet, daß die Transformation stets auch durch Verschieben der Kamera bzw. des Blickwinkels in der Szene realisierbar ist. Berechnet werden kann dies, indem die Kameraposition und ihre definierenden Bezugspunkte jeweils mit der inversen absoluten Transformationsmatrix transformiert werden.

Das Ziel ist nun, zu ermöglichen, daß alle Szenenteile, die in mehreren Instanzen auftreten, nur einmal gespeichert zu werden brauchen. Für ein konventionelles z-Buffer-Rendering ist dies einfach zu erreichen, da der Szenengraph direkt als Datenstruktur zum Rendering eingesetzt werden kann. Die Dreiecke werden bei der rekursiven Traversierung des Szenenbaums dynamisch erzeugt, direkt eingezeichnet und nicht explizit gespeichert. Für das randomisierte z-Buffer Verfahren ist dies nicht möglich, da entsprechende Verteilungsbäume benötigt werden, die Zugriff auf alle Dreiecke der Szene gestatten[18].

Die Grundidee besteht hier nun darin, für jeden Szenenteil, der in mehreren Instanzen auftritt, genau einen Verteilungsbaum aufzubauen, der alle die Dreiecke des Szenenteils explizit speichert, die nicht ihrerseits wieder in mehreren Instanzen auftreten. Für alle weiteren Instanzen wird nur noch ein Platzhalter gespeichert, der einen Verweis auf den bereits vorhandenen Verteilungsbaum sowie eine Transformationsmatrix enthält, die den Unterschied zwischen den Objekten beschreibt. Insgesamt benötigen die Verteilungsbäume dadurch größenordnungsmäßig nicht mehr Platz als der ursprüngliche Szenengraph, wenn man annimmt, daß dieser nur Drei-

[18] Da auch viele andere Renderingverfahren ähnliche Hilfsdatenstrukturen benötigen (hier im wesentlichen einen Octree der Szene) könnten sich die Methoden dieses Abschnittes auch für andere Algorithmen als nützlich erweisen.

ecke als Grundobjekte erlaubt. Die Kennzeichnung der Szenenteile als Referenzen muß dabei manuell, also vom menschlichen Modellierer der Szene selbst vorgenommen werden. Die dadurch möglichen Einsparungen sind oft drastisch. Fast alle Szenen, die in Kapitel 5 vorgestellt werden, benutzen diesen Mechanismus. Abbildung 56 zeigt das Prinzip an dem Beispiel aus Abschnitt 1.4.1.2 / Abbildung 4:

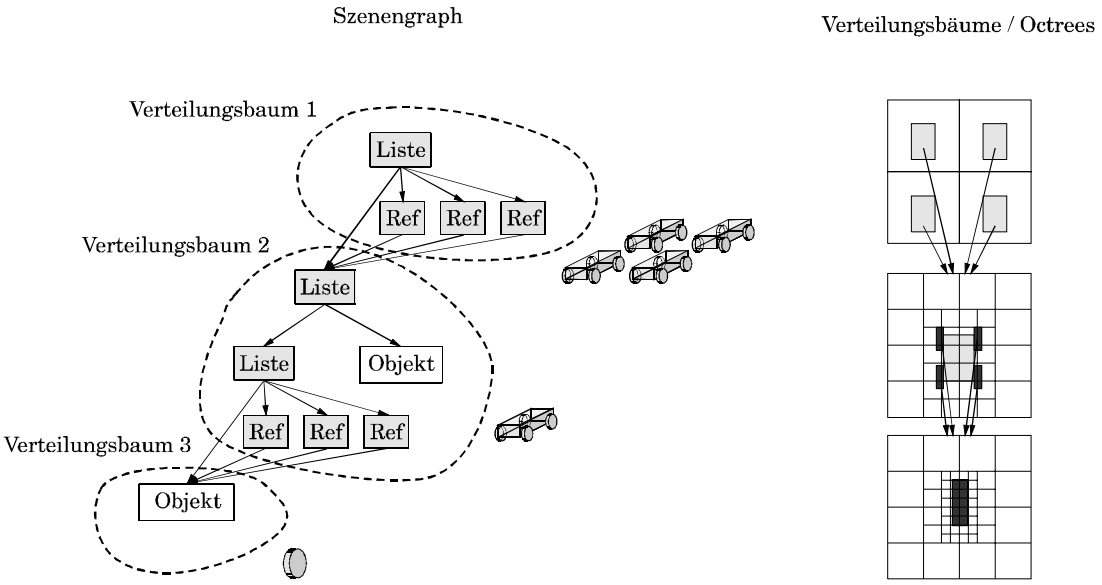


Abbildung 56: Mehrfachbenutzung von Verteilungsbäumen

Zum Erzeugen der Verteilungsdatenstruktur wird der Szenengraph per Tiefensuche durchlaufen. Die Tiefensuche stoppt bei allen Knoten K, die eingehende Referenzen aufweisen, und die (vom Modellierer) mit einem Kennzeichen versehen wurden, daß für den von ihnen ausgehenden Teilgraphen eine Unterverteilungsstruktur angefertigt werden soll. Für alle übrigen Knoten werden Dreiecke erzeugt und in den Verteilungsbaum eingefügt. Für die Knoten K und alle Referenzen darauf werden „Dummyknoten“ in den Verteilungsbaum eingefügt. Diese speichern die Transformation, die man erhält, indem man auf dem Pfad zur Wurzel des übergeordneten Teilgraphens alle Transformationen akkumuliert, sowie einen Zeiger auf die Unterverteilungsstruktur. Danach wird derselbe Algorithmus auf die Knoten K mit den jeweiligen Knoten als Wurzel des übergeordneten Teilgraphens angewendet und die entsprechenden Zeiger auf die so rekursiv bestimmte Unterverteilungsstruktur in die Dummyknoten eingetragen.

Falls Winkelklassen verwendet werden, so wird dieses Verfahren für alle räumlichen Unterteilungsbäume aller Winkelklassen nacheinander angewendet, wobei die Transformationen beachtet werden müssen (hier insbesondere Drehungen). Entsprechend wird beim Zwei-Ebenen-Verteilungsbaum verfahren, der in diesem Zusammenhang als eine Pyramide verschieden großer Winkelklassen aufgefaßt werden kann.

Eine Feinheit tritt auf, wenn Unterverteilungsstrukturen angelegt werden, deren Transformationen Skalierungen enthalten. In diesem Fall muß der Wert für die absolute Fläche, die darin enthalten ist und in die Verteilungsliste der übergeordneten Verteilungsstruktur eingetragen werden muß, zuvor mit dem entsprechenden Skalierungswert multipliziert werden. Dieser läßt sich einfach aus der Transformationsmatrix bestimmen, da nur eine eingeschränkte Klassen von Transformationen zugelassen wurde, die prinzipiell alle Dimensionen gleichmäßig skalieren. Die eigentlich kompliziertere Eigenwertaufgabe läßt sich in diesem Fall einfach dadurch lösen, die

Längen eines beliebigen Vektors (nicht null) vor und nach der Transformation (ohne Translation) zu vergleichen.

Zur Stichprobenverteilung kann die zuvor erzeugte Datenstruktur dann wie folgt genutzt werden: Zur Auswahl der Boxen wird die Struktur wie ein gewöhnlicher Baum traversiert. Falls ein Dummyknoten angetroffen wird, so wird die aktuelle Kameraposition entsprechend der dort gespeicherten Transformationen geändert (d.h. die inverse Transformation wird auf Kameraposition, Blickpunkt etc. angewendet) und danach wird die Bearbeitung wie gewohnt fortgesetzt.

Nachdem die Boxen bestimmt wurden, werden Stichprobenpunkte darauf verteilt. Dabei sind zwei Fälle möglich: Zum einen kann die Verteilungsliste direkt zu einem Dreieck führen, das wie gewohnt behandelt werden kann. Im anderen Fall führt die Liste zu einem Dummyknoten. An diesem wird die Verteilung erneut für die Unterverteilungsstruktur gestartet, wobei eine Liste von Zeigern auf alle auf dem Weg zum Dreieck angetroffenen Dummyknoten mitgeführt wird. Wenn ein Dreieck erreicht ist, wird ein Punkt darauf zufällig bestimmt und dieser Punkt dann mit den Transformationsmatrizen aller angetroffenen Dummyknoten transformiert, um die korrekte Position des Punktes zu erhalten[19].

Da die Matrix/Vektormultiplikation mit die teuerste Operation bei der Bearbeitung der Punkte ist, sollten die Instanzen nicht allzu tief geschachtelt sein, da dies die Geschwindigkeit deutlich reduzieren kann. In der Praxis haben sich, zumindest bei einer reinen Softwareimplementation, ein bis drei Instantiierungsebenen als akzeptabel herausgestellt, man erhält einen Mehraufwand von ca. 30% bis 100%.

Ein weiterer Faktor, der die Geschwindigkeit u.U. negativ beeinflussen kann, ist, daß die manuell zusammengesetzten Octrees nicht solchen entsprechen, die man erhalten würde, wenn man einen gewöhnlichen Octree für die gesamte Szene aufbauen würde. Solange die Bereiche für alle Teildatenstrukturen disjunkt sind, sind keine Verluste zu erwarten, da die Boxen dadurch höchstens kleiner und weniger werden als in einem gewöhnlichen Octree. Falls sich von den Unterbäumen dargestellte Volumina überlappen, entsteht ein Mehraufwand, da nun einige Volumenanteile mehrfach bearbeitet werden, so daß insgesamt eine größere Anzahl von Boxen entsteht (der Stichprobenumfang bleibt unberührt). Der so entstehende Mehraufwand wächst offensichtlich (höchstens) linear mit dem mehrfach überdeckten Volumen. Wenn dies beim Entwurf des Szenengraphen und der Kennzeichnung der Knoten für die Unterverteilungsstrukturen berücksichtigt wird, sind keine schwerwiegenden Probleme zu befürchten.

3.3.6.3 Dynamik

In vielen interaktiven Anwendungen soll die Szene online, während eines Walkthrough, verändert werden. Das randomisierte z-Buffer Verfahren ermöglicht nach einigen kleineren Änderungen der Datenstrukturen auch beliebige dynamische Veränderungen der Szene.

Ein einfacher Spezialfall ist, daß die relative Transformationsmatrix eines Knotens im Szenengraphen (d.h. im High-Level-Modell) geändert wird. Falls für diesen auch direkt eine Unterverteilungsstruktur angelegt wurde, genügt es, die entsprechenden Transformationen neu zu berechnen, was sich in kurzer Zeit erledigen läßt (proportional zur Länge eines Pfades zur Wurzel im Szenengraphen). Dies trifft allerdings nicht zu, wenn die Skalierung eines Knotens geändert wird, da dann die Verteilungslisten geändert werden müssen, um der Veränderung der

[19] Für Beleuchtungsberechnungen sind auch noch oft Oberflächennormalen mit den Punkten assoziiert, diese müssen nach [Foley et al. 96], Seite 1107f mit dem Transponierten der inversen Transformationsmatrix, aus der die Translationsanteile gestrichen wurden, multipliziert werden.

absoluten Fläche Rechnung zu tragen. Drehungen und Translationen bereiten dagegen keine Probleme. In einem Szenario, wie einer Stadt mit einigen fahrenden Autos, ließe sich diese einfache Methode sicherlich gut einsetzen. Da die Änderungszeit pro Objekt dabei konstant ist, wäre dieses Vorgehen in einem solchen Fall sogar die Methode der Wahl.

Im allgemeineren Fall ändern sich Teile der Szene in beliebiger Weise, derartige Änderungen lassen sich stets als Folge von Löschungen und Einfügungen von Dreiecken in die Low-Level-Darstellung auffassen. Da die Dreiecke in einem Octree gespeichert sind, können entsprechende Dynamikoperationen auf dem Octree ausgeführt werden. Zum Löschen eines Dreiecks muß der entsprechende Eintrag in einem Knoten des Baumes gelöscht werden und danach noch unnötige Knoten oberhalb gelöscht werden. Unter Umständen müssen auch noch entsprechende „*shortcuts*“ in diesen Weg eingebaut werden. Zum Einfügen eines Dreiecks muß zunächst der Blattknoten lokalisiert werden, in den das Dreieck fällt und dann dieser aufgeteilt werden. „*shortcuts*“ müssen dabei u.U. ebenfalls in zwei Teilstücke aufgeteilt werden.

Beide Operationen benötigen $O(h)$ Zeit, wenn h die Höhe des Baumes angibt. Diese ist in den meisten Szenen aus praktischen Anwendungen relativ klein, größenordnungsmäßig häufig nur logarithmisch in der Objektanzahl [Chamberlain et al. 95], so daß sich einzelne Änderungen effizient durchführen lassen.

Als Problem bleibt allerdings noch, die Verteilungslisten dynamisch anzupassen. Dies würde eine in der Objektanzahl lineare Zeit benötigen und somit zu einer schlechten Laufzeit führen. An dieser Stelle muß die bisherige Datenstruktur geändert werden. Anstatt Verteilungslisten zu benutzen, werden Verteilungsbäume verwendet, die wie folgt aufgebaut sind:

In jedem Knoten wird die absolute Fläche aller in dem jeweiligen Teilbaum gespeicherten Dreiecke gespeichert. Die Wurzel enthält damit die Gesamtsumme. Der Algorithmus zur Auswahl der Stichproben bestimmt nun zunächst zufällig eine Zahl, gleichverteilt zwischen null und der Gesamtsumme der Flächen. Dann traversiert er den Baum von der Wurzel ausgehend nach unten und entscheidet an jedem Knoten, ob der linke oder der rechte Teilbaum weiterverfolgt wird. Falls der rechte Teilbaum gewählt wird, wird eine mitgeführte Hilfsvariable, die die aktuelle Fläche „links“ von dem aktuellen Knoten speichert, um den Wert des linken Teilbaumes erhöht. Falls der linke Teilbaum gewählt wird, wird dort keine Änderung vorgenommen. Initialisiert wird die Hilfsvariable mit null. Um die Entscheidung für den linken oder den rechten Teilbaum treffen zu können, wird betrachtet, ob die Summe in der Hilfsvariablen zuzüglich des Wertes im linken Teilbaum über oder unter dem zufällig bestimmten Zielwert liegt. Im ersten Fall muß im linken anderenfalls im rechten Teilbaum weiter gesucht werden.

In einem solchen Baum können nun effizient Knoten gelöscht, eingefügt oder geändert werden. In jedem Fall muß nur der Pfad im Baum vom Blatt zur Wurzel bearbeitetet werden, auf dem die Summen der Flächeninhalte korrigiert werden.

Der Baum kann als vom Verteilungsbaum abgesetzte Datenstruktur implementiert werden, wobei anstelle der Indizes der Einträge in der Verteilungsliste Zeiger auf die entsprechenden Knoten des Baumes auf die entsprechenden Mengen von Dreiecken verweisen. Dieser Baum muß dann dynamisch balanciert werden, wobei ein Schema wie bei der Balancierung von AVL-Bäumen [Manber 89] verwendet werden kann. Da dazu nur lokale Änderungen notwendig sind, bleibt auch hier der Aufwand zur Korrektur der Flächensummen auf die Höhe des Baumes und damit in diesem Fall auf $O(\log n)$ (mit n als Objektanzahl) beschränkt.

Eine einfachere Möglichkeit ist, die notwendigen Informationen direkt in den Octree zu integrieren, anstatt einen abgesetzten Baum zu verwenden. Da dieser in vielen Szenen in praktischen Anwendungen relativ gut balanciert ist, sind gute Suchzeiten zu erwarten. Der Ausgangsgrad

jedes Knotens ist hier allerdings acht, daher muß die Regel für die Berechnung der Hilfsvariablen geändert werden: Die Knoten werden durchnumeriert, wenn Knoten i gewählt wird, wird die Fläche von Knoten $1 \ldots i$-1 sowie der dem Knoten selbst zugeordneten Objekte zur Hilfsvariablen hinzuaddiert. Die Regel entspricht der zur Generierung der zuvor verwendeten Verteilungslisten, es wird quasi der benötigte Ausschnitt aus der Liste dynamisch erzeugt.

Als Ergebnis läßt sich festhalten, daß es möglich ist, die vorberechneten Datenstrukturen für das randomisierte z-Buffer Verfahren dynamisch zu aktualisieren, wobei für die Änderung an einem Dreieck ein Aufwand von $O(h)$ notwendig wird, wobei h die Höhe des Octrees der Szene angibt. Wenn der Knoten des Raumunterteilungsbaumes bekannt ist, in dem die Objekte eingefügt oder gelöscht werden sollen, und ein dynamisch balancierter, abgesetzter Verteilungsbaum verwendet wird, genügen $O(\log n)$ Schritte. Die asymptotische Laufzeit für die Stichprobenauswahl ändert sich dabei nicht.

3.3.6.4 Weitere Optimierungsmöglichkeiten

Eine weitergehende Optimierung, die allerdings im Rahmen dieser Arbeit nicht weiter verfolgt werden konnte, bestünde darin, je nach Größe der Dreiecke unterschiedliche Strategien zu ihrer Darstellung zu verfolgen: Alle Dreiecke, die eine projizierte Fläche von mehreren Pixeln (bzw. Farbklotzdurchmessern) aufweisen, können bei einer tiefenpufferbasierten Bildrekonstruktion direkt mit dem konventionellen z-Buffer Verfahren eingezeichnet werden, anstatt sie mit zufälligen Stichproben abzutasten. Für Dreiecke, die mehrere Zellen der Bildrekonstruktion bedekken, ist die Verwendung des randomisierten z-Buffers immer mit Sicherheit langsamer als die Verwendung des konventionellen Verfahrens, da mehr Transformationen und mehr Pixelfüllungen im randomisierten Fall benötigt werden.

Für Dreiecke, deren Ausmaße auf dem Bildschirm garantiert unter einem Pixel liegen, könnte andererseits direkt ein zufällig gewählter Eckpunkt projiziert werden, anstatt einen Punkt auf dem Dreieck zu würfeln, was die relativ aufwendigen Fließkommaoperationen für die zufälligen Linearkombinationen sparen würde.

Um derartige Optimierungen ausnutzen zu können, müßten jedoch alle Dreiecke mit großer projizierter Fläche, bzw. mit kleiner projizierter Ausdehnung schnell gefunden werden. Eine exakte Lösung erscheint hier kaum möglich: schon einfachere exakte Bereichsanfragen ohne Berücksichtigung von Objektgrößen und Winkelverhältnissen bereiten einige Probleme (wie superlinearen Platzbedarf bei polylogarithmischer Abfragezeit [de Berg et al. 97], Seite 114). Es wäre aber eine approximative Lösung, ähnlich wie bei der Verteilung der Stichprobenpunkte, denkbar.

Eine Möglichkeit für eine approximative Bestimmung aller Dreiecke mit großer projizierter Fläche bestünde darin, die Verteilungsbäume für mehrere Größenklassen von Dreiecken aufzubauen, wobei in jeder Klasse der Flächeninhalt der enthaltenen Dreiecke bis auf maximal einen konstanten Faktor fest ist. Dadurch erhält man zusätzlich zur räumlichen und zur Winkelunterteilung nochmals k-mal so viele Klassen für Boxen wie beim gewöhnlichen Winkelklassenverfahren, wobei k logarithmisch mit dem Verhältnis des kleinsten zum größten Flächeninhalt eines Dreiecks in der Szene wächst. Für jede Kombination aus Winkel- und Größenklasse könnte dann eine approximative räumliche Bereichsanfrage (s.u.) vorgenommen werden mit einem Radius der entsprechend der Größenklasse und des Winkels gewählt wird. Die dabei gefundenen Boxen enthalten Objekte, für die sich die randomisierte Methode nicht lohnt und werden entsprechend markiert, so daß sie beim Rendering direkt eingezeichnet anstatt zufällig abgetastet werden.

Durch die zusätzlichen Größenklassen würde sich allerdings die Boxenanzahl, die schon jetzt die erreichbare Geschwindigkeit in bestimmten Fällen beschränkt (siehe Kapitel 5), nochmals vervielfachen, so daß zweifelhaft ist, ob sich die Strategie wirklich auszahlen würde. Falls eine Szene gegeben ist, in der die Objekte alle ungefähr gleich groß sind, ist die Klasseneinteilung allerdings nicht notwendig, und man kann mit einer Bereichsanfrage auskommen, die sich hinreichend effizient beantworten läßt.

Approximative Bereichsanfragen mit einem Octree

Eine mögliche Strategie zur approximativen Bereichsanfrage mit Hilfe eines Octrees verläuft wie folgt: Es wird ein quadratischer Bereich um den Betrachter vorgegeben und die Boxen des Baumes werden rekursiv durchsucht: Boxen die vollständig innerhalb liegen werden direkt mit dem konventionellen z-Buffer Verfahren eingezeichnet, Boxen die außerhalb liegen werden nicht bearbeitet, und Boxen die teilweise innerhalb liegen, werden nur dann weiter aufgeteilt, wenn deren Durchmesser höchstens um einen konstanten Faktor kleiner als der Durchmesser des Suchbereiches ist. Damit werden nur konstant viele Boxen zur Überdeckung des Suchbereiches benötigt. Hinzu kommt eine lineare Suche durch Octreeknoten, die den Bereich enthalten, deren Zeitbedarf maximal logarithmisch im Verhältnis von Szenendurchmesser zu Suchbereichsdurchmesser ist.

Falls die Suche für mehrere Größenklassen mit fallendem Durchmesser durchgeführt werden soll, kann dieser anfängliche Aufwand bei weiteren Klassen reduziert werden, indem alle Boxen, die gleiche räumliche Bereiche repräsentieren, miteinander durch Zeiger in eine lineare Liste (geordnet nach den Größen) verbunden werden und die Suche in der für die letzte (größere) Klasse erreichten Tiefe gestartet wird. Für die Suche in mehreren Winkelklassen kann analog vorgegangen werden.

Die Idee der Sonderbehandlung für große Dreiecke konnte im Rahmen dieser Arbeit nicht mehr im Detail weiter verfolgt werden. Als lediglich einen ersten Schritt in diese Richtung wird in Abschnitt 5.1.4 eine Kombination mit konventionellem z-Buffer Rendering beschrieben, in der unterschiedliche Größen und Betrachtungswinkel der Dreiecke völlig ignoriert werden und immer nur ein Bereich mit festem Radius mit dem konventionellen Verfahren dargestellt wird. Da in der verwendeten Testszene die Verhältnisse der Objektgrößen nicht extrem waren, ließ sich damit tatsächlich eine gewisse Beschleunigung erzielen.

3.3.7 Zusammenfassung der Ergebnisse

Zur Bestimmung einer Stichprobe von Szenenpunkten können verschiedene Strategien verfolgt werden. Die hier vorgestellten Strategien basieren auf einer Unterteilung des Raumes mittels eines Octrees und einer groben Vorklassifizierung der Normalenrichtungen über ein Gitter oder ein Quadtree auf den Polarkoordinaten der Normalen. Aus diesen Datenstrukturen werden dynamisch Objektgruppen mit ähnlichem Projektionsfaktor bestimmt und Punkte nach dem Flächeninhalt der enthaltenen Dreiecke zufällig ausgewählt. Der Aufwand hängt dabei zum einen von geometrischen Eigenschaften der Szene ab und zum anderen von der Anzahl der in der Szene vorhandenen Objekte.

Der erste geometrische Faktor ist die Tiefendynamik Szene, also das Verhältnis zwischen minimalem Betrachtungsabstand und dem Durchmesser der Szene. Der Aufwand bei der Stichprobenverteilung wächst logarithmisch mit diesem Wert, wobei nur eine konstante Überschätzung des Einflusses der Objekttiefe auf die projizierte Fläche gemacht wird. Dabei werden nur Objek-

te berücksichtigt, die in einer um einen konstanten Faktor im Querschnitt erweiterten Pyramide um die Sichtpyramide liegen.

Der zweite Faktor ist die Verteilung der Winkel, unter denen die Oberflächen gesehen werden. Für worst-case Szenen ist mit den hier verwendeten Methoden eine effiziente Verteilung nicht möglich: Entweder wird die projizierte Fläche beliebig stark überschätzt, was den Aufwand zur Stichprobenwahl und Bildrekonstruktion beliebig erhöht, oder der Aufwand wächst mit der vierten Potenz des Kehrwertes des kleinsten Winkels, unter dem ein Objekt in der Szene gesehen wird, welcher selber auch unbeschränkt ist.

Diese schlechten Ergebnisse sind allerdings in der Praxis nicht zu erwarten: Wenn angenommen wird, daß die Oberflächennormalen auf der Einheitskugel unabhängig vom Betrachtungsvektor gleichverteilt sind, erhält man einen erwarteten Wert für die Überschätzung des Projektionsfaktors, der nur um einen kleinen konstanten Faktor über dem erwarteten Wert für den genauen Projektionsfaktor liegt.

Falls die Orientierungen der Dreiecke völlig unberücksichtigt bleiben, beträgt dieser Faktor 4. Durch die Einführung von Winkelklassen kann der Faktor weiter gesenkt werden (mit einer realistischen Anzahl von Winkelklassen etwa in den Bereich von knapp über 2). Mit einem Zwei-Ebenen-Baum ist eine adaptive Auswahl von Winkelklassen möglich, die den Aufwand etwas verbessern kann, dafür aber mehr Speicher benötigt und im dynamischen Fall entsprechend langsamer ist.

Falls ein Szenenteil näherungsweise durch eine ebene Scheibe beschrieben werden kann, muß auch bei Vernachlässigung aller Winkeleinflüsse der Durchmesser exponentiell erhöht werden, um die Überschätzung des Projektionsfaktors linear zu vergrößern. Daher ist auch in vielen ungünstigen Fällen kein allzu starkes Anwachsen des Aufwandes zu befürchten.

Der Einfluß der Anzahl von Objekten n in der Szene auf die Zeit zur Bestimmung der Stichprobe ist sehr schwach, der Aufwand wächst nur logarithmisch mit n. Wenn also eine feste Szene angenommen wird, in die zunehmend mehr Details eingefügt werden, ohne die projizierte Fläche oder die genannten geometrischen Eigenschaften stark zu verändern, wächst der Aufwand nur logarithmisch mit dem Detaillierungsgrad. In diesem Sinne ist das randomisierte z-Buffer Verfahren auch für extrem komplexe Szenen geeignet, da der Aufwand nur sehr schwach mit dem Detaillierungsgrad wächst. Ein solches Ergebnis ist für Szenen in der hier möglichen Allgemeinheit mit den bekannten, in Kapitel 2 beschriebenen Verfahren nicht zu erreichen.

Der Speicheraufwand wächst bei den hier beschriebenen Verfahren linear mit der Szenengröße, was für komplexe Szenen bereits Probleme bereitet. Daher wurde ein Verfahren vorgestellt, um Redundanzen in der Szene auszunutzen: Szenenteile, die sich nur durch eine Translation, Rotation, Skalierung oder Spiegelung voneinander unterscheiden, brauchen nur einmal abgespeichert zu werden. Diese Instantiierung ist auch auf mehreren Ebenen, d.h. mit Teilen von instantiierten Objekten, möglich.

4 Implementation

Um das im vorherigen Kapitel beschriebene randomisierte z-Buffer Verfahren an praktischen Beispielen zu testen und den Einfluß der verschiedenen Parameter zu überprüfen, wurde ein Prototyp implementiert. Die Implementation besteht aus einer C++-Bibliothek, die es gestattet, einfache Modelle zu erstellen und diese mit dem randomisiertem und dem konventionellen z-Buffer Verfahren darzustellen. Darauf aufbauend wurde ein minimales Walkthroughsystem implementiert, um diese Verfahren an verschiedenen Szenen zu testen.

In diesem Kapitel sollen die wesentlichen Merkmale der Implementation des Verfahrens beschrieben werden. Zunächst wird die Systemumgebung vorgestellt, innerhalb derer die Beispielimplementation entwickelt wurde. Danach wird auf einige Besonderheiten der Implementation der beschriebenen Algorithmen eingegangen und zuletzt wird kurz die softwaretechnische Umsetzung beschrieben.

4.1 Plattform und Bibliotheken

Das Kernsystem wurde in C++ implementiert. Als 3D-Grafikbibliothek kam die ursprünglich von Silicon Graphics entwickelte Bibliothek „OpenGL“ zum Einsatz, die inzwischen als offener Standard definiert wurde und für eine Vielzahl von Plattformen zur Verfügung steht [Woo et al. 97]. Für die Implementation der Benutzerschnittstelle wurde die Bibliothek „VCL“ der Firma Borland genutzt, die derzeit nur für Microsoft Windows zur Verfügung steht. Da sich die Benutzerschnittstelle auf die Eingabe einiger Parameter und die Navigation durch die Szene beschränkt, wird darauf im weiteren nicht mehr eingegangen. Alle Testprogramme wurden mit Borland C++ Builder Version 4.0 unter Microsoft Windows NT 4.0 erstellt und kompiliert. Genauere Angaben zur verwendeten Hardware für die Testläufe finden sich in Anhang A.

Für die Umsetzung der Algorithmen ist vor allem die Struktur der 3D-Grafikbibliothek OpenGL interessant, diese soll daher kurz im folgenden beschrieben werden. Ein umfassender Überblick findet sich in [Woo et al. 97].

Die Bibliothek wurde entwickelt, um hardwareunabhängig interaktive Echtzeitanwendungen im Bereich der dreidimensionalen Computergrafik entwickeln zu können. Es handelt sich um eine Low-Level Bibliothek, die im wesentlichen nur die Transformation und Rasterung von Zeichenprimitiven sowie für eine einfache, lokale Beleuchtungsberechnung (ein Phong Beleuchtungsmodell wie in [Foley et al. 96] beschrieben) zur Verfügung stellt. Standardmäßig wird ein z-Buffer Algorithmus zur Unterdrückung verdeckter Flächenteile bereitgestellt. Eine besondere Stärke von OpenGL ist, daß alle bereitgestellten Operationen auch von Spezialhardware unterstützt werden können, soweit vorhanden. Hier wurden sowohl eine Softwareimplementation (siehe Anhang A, Testsystem 1) wie auch eine weitgehend hardwareunterstützte Variante verwendet (Testsystem 2). Da das entsprechende System sehr kurzfristig zur Verfügung stand, wurde es allerdings nur für einige komplexe Szenen eingesetzt[20].

Als Zeichenprimitive stehen in OpenGL Punkte (Quadrate variabler Größe), Liniensegmente und Dreiecke zur Verfügung. Um eine Menge von Primitiven einzuzeichnen, geht man wie folgt vor: Die Bibliothek merkt sich in einer internen Datenstruktur einen aktuellen „Zustand“, der

[20] Das „Happy Buddha“ Modell und das große Stadtmodell hätten zuviel Speicher für Testsystem 1 benötigt und die Landschaftsszenen mit dynamischer Beleuchtung hätten zu zu langen Renderingzeiten geführt.

alle Parameter des Renderings wie Farben, Materialeigenschaften, Lichtquellen, die aktuelle Projektionsmatrix und ähnliches speichert. Durch Prozeduraufrufe lassen sich alle dieses Zustandseigenschaften einzeln einstellen und werden dann beim Rendering des nächsten Primitivs verwendet. Das Zeichnen eines Punktes wird durch einen Prozeduraufruf ausgelöst, der die (absoluten, noch nicht projizierten) Koordinaten des Punktes übergibt. Für das Zeichnen von Linien und Dreiecken wird ähnlich vorgegangen, wobei allerdings verschiedene Möglichkeiten für die Reihenfolge bestehen, in der die Punkte von mehreren zusammenhängenden Liniensegmenten oder Dreiecken angegeben werden. Bei der auf Testsystem 1 benutzten Softwareimplementation der Bibliothek spielte diese allerdings keine wesentliche Rolle. Bei der Verwendung von Hardware, die geometrische Transformationen unterstützt (wie Testsystem 2), kann dies jedoch eine größere Rolle spielen. Auch die Anzahl von Zustandsänderungen (z.B. Ändern von Farben oder Materialparametern) kann einen signifikanten Einfluß auf die Performance haben [Woo et al. 97]. Es war allerdings im Rahmen dieser Arbeit nicht mehr möglich, hier Optimierungen vorzunehmen. Soweit die Laufzeiten und der Speicherbedarf dies zuließen, wurden daher alle Messungen mit der Softwareimplementation auf Testsystem 1 vorgenommen, die sich relativ unempfindlich gegenüber diesen Details zeigte und einen ausgewogene Beurteilung des Einfluß der verschiedenen Operationen auf die Laufzeit erlaubte.

Zur Implementation des randomisierten z-Buffer Verfahrens wurden direkt die bereitgestellten Funktionen zum Zeichnen von Punkten mit verschiedenem Durchmesser verwendet. Die entsprechenden Transformationen, Beleuchtungsberechnungen und der z-Buffer Test wurden dadurch direkt von der Grafikbibliothek vorgenommen und waren daher gut für die verwendete Hardware optimiert. Die direkte Unterstützung der elementaren Operationen für die Bildrekonstruktionsmethode mit „großen Farbklötzen“ ist ein großer Vorteil für die Implementation des Verfahrens, sowohl bezüglich des Implementationsaufwandes als auch für die erhaltenen Laufzeiten. Auf Testsystem 2 wurde diese Operation sogar größtenteils in Hardware ausgeführt.

Das konventionelle z-Buffer Rendering wurde soweit möglich mit *„triangle strips“* vorgenommen, die in der Regel eine gute Performance liefern. Bei einigen Modellen, die nur als ungeordnete Dreiecke vorlagen, mußte jedoch auf das einzelne Rendering der Dreiecke zurückgegriffen werden, was auf Testsystem 2 zur gewissen Performanceeinbußen (etwa Faktor 1,7) führte[21]. Auf Testsystem 1 zeigten sich hier keine wesentlichen Unterschiede.

4.2 Umsetzung der Algorithmen

Im Rahmen dieser Arbeit wurden alle in Kapitel 3.2 beschriebenen Verfahren zur Bildrekonstruktion implementiert, wobei für die Filtermethoden allerdings die dort beschriebenen Einschränkungen zu machen sind (es wurde mit Pixeln statt mit exakten Punkten gerechnet und eine schwache z-Dominanz mußte daher auch in Kauf genommen werden). In allen Fällen wurden die Transformationen und die z-Buffer Tests direkt mit den von OpenGL bereitgestellten Funktionen durchgeführt.

Von den in Kapitel 3.3 beschriebenen Methoden zur Auswahl von Stichprobenpunkten wurde nur das Winkelklassenverfahren implementiert. Dabei konnte die Anzahl von Winkelklassen und die maximale Variation des Projektionsfaktors ε durch die Tiefe vorgegeben werden. Die in Kapitel 3.3.6.2 beschriebene Methode, um mehrere Instanzen eines Szenenteils nur einmal spei-

[21] Dies mußte u.a. beim z-Buffer Rendering des „Happy Buddha“ Modells (siehe Kapitel 5.2) angewandt werden und führte dort zu einer gemessen an der verwendeten Hardware relativ langen Laufzeit.

chern zu müssen, wurde wie dort beschrieben implementiert. Auf eine Implementation der Zwei-Ebenen-Verteilungsbäume sowie auf eine dynamische Version mußte im Rahmen dieser Arbeit verzichtet werden.

Bei der Implementation der Raumunterteilungsbäume wurde eine wichtige Änderung vorgenommen. Anstatt der in Kapitel 3.3 beschriebenen Octrees wurden die eng verwandten, in [Callahan 95] eingeführten *„fair-split-trees“* verwendet, welche im statischen Fall leichter zu implementieren sind als Octrees mit *„shortcuts“* und sich besser an die Verteilung der Objekte in der Szene anpassen:

Ein *„fair-split-tree“* ist ein binärer Baum, der (in der hier verwendeten Form, die ursprüngliche Definition in [Callahan 95] war etwas allgemeiner) wie folgt gebildet wird:

Die Objekte sind den Blättern des Baumes zugeordnet. Den inneren Knoten sind jeweils alle den beiden Kindknoten zugeordneten Objekte zugeordnet, so daß die Wurzel die gesamte Szene repräsentiert. Zusätzlich ist jedem Knoten k der kleinste Quader $B(k)$ zugeordnet, der alle Objekte vollständig enthält. In jedem inneren Knoten k werden die Objekte auf die Kindknoten dadurch aufgeteilt, daß die Box $B(k)$ in ihrer längsten Dimension genau in der Mitte aufgeteilt wird. Im Gegensatz zum d-dimensionalen Quadtree werden die Boxen der Kindknoten für die in ihnen enthaltenen Objekte neu bestimmt; es werden nicht einfach die Hälften von $B(k)$ verwendet. Dadurch werden automatisch größere leere Bereiche durch „Schrumpfen“ der Boxen übersprungen, so daß keine *„shortcuts“* o.ä. notwendig werden (siehe Abbildung 57).

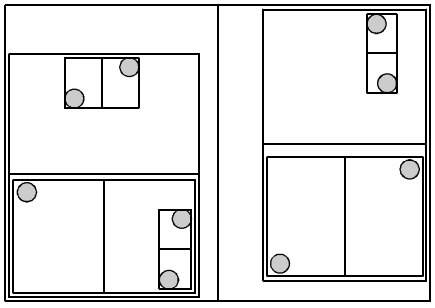


Abbildung 57: Ein *„fair-split-tree“* für eine Menge punktförmiger Objekte

Falls die Szene würfelförmig und dicht mit Objekten gefüllt ist, erhält man mit dem Fair-Split-Tree genau einen Octree, da immer alle drei Teilungsdimensionen nacheinander durchlaufen werden. Der Octree ist zwar kein binärer Baum, jeder Knoten kann aber ohne Einschränkung als gemeinsame Speicherung von drei aufeinanderfolgenden, binären Teilungen in allen drei Dimensionen aufgefaßt werden.

Falls die Szene auch unbesetzte Gebiete enthält, schrumpfen die Boxen schneller als bei einem Octree mit *„shortcuts“*, so daß insgesamt weniger Boxen entstehen, die, auf den selben Ebenen betrachtet, immer kleiner als die im Octree sind. Dadurch lassen sich alle Ergebnisse aus Kapitel 3.3, insbesondere Satz 6, direkt auf die *„fair-split-trees“* übertragen.

Die *„fair-split-trees“* bieten eine Reihe von Vorteilen, da sie sich besser und automatisch an die Szene anpassen. So muß z.B. nicht zwischen „zweidimensionalen“ und „dreidimensionalen“ Szenen unterschieden werden, da die Datenstruktur automatisch in der Ebene angeordnete Szenen erkennt und die groben Einteilungen zunächst nur in der Ebene vornimmt, bis die Boxengröße auf die Höhe der Szene geschrumpft ist. Der einzige Nachteil ist, daß eine dynamische Version dieser Datenstruktur deutlich schwieriger zu implementieren ist als bei gewöhnlichen Octrees

(die entsprechenden Algorithmen finden sich in [Callahan 95]), es können aber die selben Zeitschranken erreicht werden.

Die Strategie zur Behandlung ausgedehnter Dreiecke kann für einen *„fair-split-tree“* genauso wie in Abschnitt 3.3.6.1 für einen Octree beschrieben angewendet werden. Für große Dreiecke konnte ausgewählt werden, ob diese mit dem konventionellen z-Buffer Verfahren gerastert werden sollten oder (trotz der Abweichung) in das Stichprobenverfahren mit einbezogen werden sollten. Die erste Methode erreichte meistens kürzere Laufzeiten. Die zweite Methode wurde in allen Beispielen zur Beurteilung der Bildqualität (Kapitel 5.2) gewählt.

Um den Speicherbedarf der Verteilungsbäume in handhabbaren Grenzen zu halten, wurde ein zusätzlicher Parameter eingeführt, der eine Mindestanzahl von Objekten pro Blattknoten angibt. In der Regel wurden so 16 bis 256 Objekte (je nach Szenengröße) auf einmal in einem Knoten gespeichert, was den Speicherbedarf gegenüber einer vollständigen Auflösung um ein vielfaches reduzierte, da jeder Knoten im Raumunterteilungsbaum relativ viele Verwaltungsinformationen, insbesondere Angaben über die Boxen $B(k)$, benötigt.

Um zu verhindern, daß durch die Vorsortierung in Winkelklassen weit voneinander entfernte Objekte, die nur zufällig als einzige einen ähnlichen Winkel in der Szene haben, in einen Knoten zusammengefaßt werden, wurde ein weiterer Parameter eingeführt, der das Verhältnis von Knotendurchmesser zum Durchmesser des größten darin enthaltenen Objekts beschränkt. Wird der entsprechende Wert überschritten, so wird der Knoten auch bei wenigen Objekten weiter aufgeteilt. Ohne diese Maßnahme traten bei einigen Szenen (z.B. bei der Landschaftszene aus Kapitel 5.2) Probleme mit zu großen Boxen auf. Für diesen Parameter wurde hier ein Wert in der Größenordnung der Wurzel aus der maximalen Objektanzahl verwendet, was der Annahme folgt, daß die Objekte der Szene lokal etwa ähnlich groß und näherungsweise in einer Ebene angeordnet sind.

Mit diesen Maßnahmen traten keine wesentlichen Probleme mit der Datenstruktur auf. Bei großen Werten für die Minimalanzahl von Objekten konnten bei einer starken Nahansicht zum Teil die allernächsten Dreiecke nicht weit genug räumlich unterteilt werden, was durch die oben genannten Ausweichstrategien (Umschalten auf konventionellen z-Buffer) ohne Probleme kompensiert wurde. Bei den Messungen in Kapitel 5.1 wurde stets darauf geachtet, daß alle Boxen vom gewählten Standpunkt aus weit genug unterteilt werden konnten, so daß dieses Phänomen keinen Einfluß auf die gemessenen Laufzeiten oder die Boxenanzahl hatte.

Die *„fair-split-trees“* wurden auch zur Erzeugung der Winkelklassen verwendet. Anstatt ein festes Raster vorzugeben, wurde ein *„fair-split-tree“* über allen Normalvektoren in Polarkoordinaten aufgebaut, wobei sowohl eine maximale Unterteilungstiefe wie auch ein minimaler Durchmesser der Boxen vorgegeben werden konnte. Die Boxen der Blattknoten dieses Baumes wurden als Winkelklassen verwendet. Auch dies ist eine naheliegende Optimierung des zuvor beschriebenen Winkelklassenverfahrens, da dadurch die Boxen nur die Bereiche abdecken, in denen sich tatsächlich Normalen befinden. Falls sich die Normalen gleichmäßig über den möglichen Winkelbereich verteilen, entsteht ein gewöhnliches Gitter von Boxen, so daß auch hier im worst-case kein Unterschied zur in Kapitel 3.3.5.2 analysierten Methode besteht.

Im Falle einer statischen Szene ist ein *„fair-split-tree“* eine sehr naheliegende Optimierung des konventionellen Octrees, so daß es Sinn machte, direkt diese Datenstruktur anstelle der Octrees zu implementieren, um praxisgerechte Resultate zu erhalten. In der Analyse konnte die Optimierung jedoch nicht berücksichtigt werden, da ihr Effekt stark von der Form der Szene abhängt. Im worst-case tritt kein Unterschied zum Octree auf.

4.3 Systemarchitektur

Das Kernsystem wurde als objektorientierte C++-Bibliothek realisiert. Diese gliedert sich grob in die drei Bereiche *Szenengraphenverwaltung*, *Modellierung* und *geometrische Datenstrukturen*. Beim Entwurf des Systems wurde besonderen Wert auf Flexibilität und Erweiterbarkeit gelegt, insbesondere da am Anfang der Entwicklung noch nicht völlig klar war, welche Funktionen später benötigt würden und wie die Algorithmen und Datenstrukturen für das randomisierte z-Buffer Rendering genau aussehen würden. Dadurch ließen sich notwendige Erweiterungen, wie z.B. das Mehrfachinstantiieren von Szenenteilen, relativ einfach integrieren. Abbildung 58 zeigt einen Überblick über die wichtigsten Klassen.

Szenengraphenverwaltung

Die Oberklasse aller Elemente des Szenengraphen ist die Klasse `Object3D`. Diese Klasse sieht eine relative Transformationsmatrix vor, verwaltet Assoziationen mit referenzierenden Objekten und geometrischen Datenstrukturen und speichert allgemeine Optionen, wie die Verwendung von Untersuchstrukturen oder die Sichtbarkeit (Objekte können als unsichtbar markiert werden, um das Modellieren mit Referenzen zu erleichtern).

Davon abgeleitet sind die Klassen `Object3DList` und `Object3DInstance`. `Object3DList` ist eine Liste, in der beliebige Objekte vom Typ `Object3D` eingetragen werden können. Von `Object3DInstance` sind die Unterklassen `SingleObject3D` und `Object3DReference` abgeleitet. Die Oberklasse verwaltet die Assoziation der Objektinstanzen mit ihren Einträgen in geometrische Datenstrukturen, d.h. es werden die entsprechenden Informationen zum effizienten dynamischen Ändern und Löschen der Instanzen bereitgehalten. `Object3DReference` stellt einen Knoten im Szenengraphen dar, der als Referenz auf einen beliebigen anderen Szenenteil vom Typ `Object3D` fungiert. `SingleObject3D` ist der Vorfahre von allen konkreten Objekten. Die Schnittstellen zum Zugriff auf eine Dreiecksnetzdarstellung der Geometrie sowie auf Materialparameter ist dabei bereits in der Oberklasse implementiert, so daß sich das Ableiten eines neuen Objekttyps durch das Überschreiben einer Methode realisieren läßt.

Die Listen und Referenzen stellen Methoden bereit, um einzelne Knoten zu einem Szenengraphen zusammenzufügen. Dabei werden die wichtigsten Konsistenzen überprüft (keine zykli-

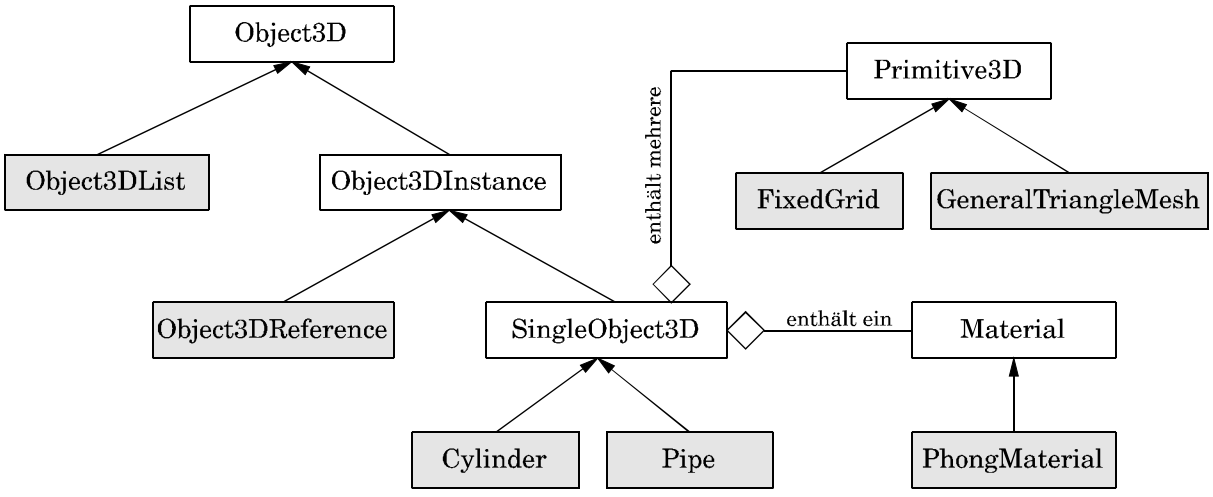


Abbildung 58: Vererbungshierarchie für die Szenengraphen und Modellierungsobjekte (abstrakte Klassen mit weißem Hintergrund, Pfeile für Vererbung)

schen Referenzen, keine Knoten, die in zwei Listen enthalten sind etc.), was sich in der Testphase als nützlich erwiesen hat. Zusätzlich gibt es Methoden, die den Szenengraphen traversieren und für jedes Objekt im Graphen oder für jede Instanz (wenn man alle Referenzen vollständig verfolgt) eine Callbackfunktion aufrufen. Dies wird z.B. für die Implementierung des z-Buffer Renderings benutzt.

Modellierung

Die von `SingleObject3D` abgeleiteten Objekte implementieren nicht direkt das Zeichnen der Objekte mit OpenGL-Befehlen. Statt dessen werden zwei zusätzliche Hilfsklassen verwendet. Die erste ist die abstrakte Klasse `Primitive3D`. Diese stellt Methoden bereit, um Dreiecksnetze zu verwalten. Als konkrete Nachfahren wurden hier allgemeine Netze (als Eckpunkt- und Dreieckslisten) sowie parametrische Oberflächen mit quadratischem Parameterbereich und gleichförmiger, gitterförmiger Unterteilung des Parameterbereiches implementiert. Diese lassen über eine Callbackfunktion unter Angabe der Parameterwerte automatisch die Koordinaten der Dreieckseckpunkte vom definierenden Objekt erzeugen. Eine elementare Netzvereinfachung durch Auslassen von Parameterwerten im Gitter wurde ebenfalls zur Verfügung gestellt; an dieser Stelle könnten bei Bedarf auch aufwendigere „*mesh simplification*“ Verfahren integriert werden. Die Trennung von der Verwaltung der Dreiecksnetze und der Definition der Geometrie vermeidet, daß die Verwaltungsalgorithmen (insbesondere die Netzvereinfachung) mehrfach implementiert werden müssen.

Die zweite Klasse ist die abstrakte Klasse `Material`, sie setzt vor der eigentlichen Übergabe der Dreiecke an OpenGL die benötigten Zustandsparameter, um so die Erscheinung der Oberfläche zu definieren. Da die Klasse selber das Rendering steuert, sind auch mehrere Renderingdurchgänge mit der selben Oberfläche möglich, wie sie für komplexere Schattierungsalgorithmen benötigt werden. Davon wurde hier allerdings kein Gebrauch gemacht, als einziger Nachfolger wurde eine Klasse implementiert, die ein einfaches Phong Beleuchtungsmodell realisiert. Auf Wunsch konnte die Beleuchtung auch vor dem Rendering statisch vorberechnet werden, so daß nur noch eine feste Farbe gespeichert und beim Rendering berücksichtigt werden muß, was aus Laufzeitgründen in den meisten Beispielen auch verwendet wurde.

Als von `SingleObject3D` abgeleitete Grundprimitive wurden nur einige, für die Implementation der Testszenen benötigte Klassen implementiert. Neben allgemeinen Dreiecksnetzen waren dies Zylinder und Röhren (mehrere in verschiedenen Winkeln und mit verschiedenen Querschnitten stetig aneinander gesetzte Zylinder), sowie Höhenfelder (in einem regelmäßigen zweidimensionalen Gitter gegebene Höhenwerte).

Die Zylinder und Röhren wurden zur Darstellung der Stämme und Äste der Bäume in den Waldszenen und die Höhenfelder für den Boden und die Landschaftsszenen benutzt. Alle anderen Objekte, wie die Blätter der Bäume, die Objekte aus den Stadtszenen sowie die aus dem Internet geladenen, von Garland [Garland 99] bereitgestellten Modelle, wurden als allgemeine Dreiecksnetze modelliert.

Geometrische Datenstrukturen

An einen Szenengraphen können geometrische Datenstrukturen gebunden werden. Dazu wurde eine abstrakte Oberklasse `SearchStructure` definiert, die die dazu notwendigen Verwaltungsoperationen bereitstellt. Nachdem festgelegt wurde, ob einzelne Dreiecke oder ganze Objektinstanzen gespeichert werden sollen (für effizientes z-Buffer Rendering), wird die Datenstruktur an den Szenengraphen gebunden. Dazu werden alle gewünschten Objekte in den Szenengraphen eingefügt und bei Bedarf automatisch Unterdatenstrukturen (Abschnitt 3.3.6.2) angelegt.

Mit den eingefügten Objekten werden Informationen über alle Datenstrukturen, in der sie enthalten sind sowie dem Knoten der Datenstruktur, in der sie gespeichert sind, abgelegt. Bei einer Änderung im Szenengraph wird eine entsprechende Nachricht an die geometrische Datenstruktur geschickt, so daß effizient eine dynamische Aktualisierung vorgenommen werden kann. Dieses kann von der Datenstruktur bis zum nächsten Zugriff darauf verzögert werden. Dieser Mechanismus kann bei Bedarf abgeschaltet werden, da er zusätzlichen Speicherplatz kostet.

Da hier nur die statische Version implementiert wurde, wurde diese Vorgaben der abstrakten Basisklasse in den konkreten Nachfahren so realisiert, daß nach einer Änderung beim ersten Zugriff auf die Datenstruktur diese vollständig neu aufgebaut wird. Eine Integration von voll dynamischen Datenstrukturen wäre allerdings später einfach möglich.

In einer geometrischen Datenstruktur können drei verschiedene Typen von Objekten abgelegt werden: Dreiecke, Objektinstanzen sowie Dummyknoten für vollständige Unterdatenstrukturen (siehe Abschnitt 3.3.6.2). Um diese gleichartig und ohne Fallunterscheidungen und Mehrfachimplementationen behandeln zu können, wurde ein abstraktes Interface (d.h. eine Basisklasse, die nur abstrakte Methoden enthält und per Mehrfachvererbung in verschiedenen konkreten Objekten implementiert werden kann) `SamplingPrimitive` definiert, daß alle Eigenschaften liefern kann, die zum Aufbau einer geometrischen Datenstruktur benötigt werden. Dazu zählen Methoden, um den kleinsten umfassenden, achsenparallelen Quader oder die repräsentative Normale und die maximale Winkelabweichung abzufragen. Die verschiedenen konkreten Klassen liefern hier so genaue Angaben, wie ihnen möglich ist. Zusätzlich gibt es Methoden, die einen zufälligen Punkt auf dem Objekt einzeichnen und die überprüfen, ob sich ein Objekt noch weiter unterteilen läßt. Damit können ganze Untersuchstrukturen konzeptionell wie einzelne Objekte behandelt werden, was die Implementation deutlich vereinfacht. Auch eine Integration von anderen Grundprimitiven als Dreiecken würde dadurch erleichtert.

5 Experimentelle Ergebnisse

Dieses Kapitel gliedert sich in zwei Teile: Im ersten Teil wird experimentell untersucht, welchen Einfluß die verschiedenen Verfahrens- und Szenenparameter auf die Laufzeit haben. Im zweiten Teil wird daß Verfahren auf mehrere komplexe Szenen angewandt. Hier soll insbesondere untersucht werden, welche Bildqualität für typische Anwendungsszenen erreicht werden kann, speziell unter Berücksichtigung der verschiedenen Alternativen bei der Bildrekonstruktion. Dazu werden auch ungünstige Szenen, die zu Aliasing neigen, scharfe Kanten aufweisen oder großflächig die Annahme der getrennten Verdeckung verletzen, eingesetzt. Außerdem wird hier deutlich werden, daß in Szenen, die mit konventionellen Verfahren kaum darstellbar gewesen wären, große Laufzeitgewinne mit dem randomisierten z-Buffer Verfahren erreicht werden können.

5.1 Laufzeitverhalten in Abhängigkeit der verschiedenen Parameter

In diesem Abschnitt sollen der Einfluß einiger wesentlicher Szenen- und Verfahrensparameter auf die Performance des Verfahrens experimentell untersucht werden und die Ergebnisse mit denen der theoretischen Voraussagen verglichen werden. Alle Untersuchungen in diesem Abschnitt wurden auf Testsystem 1 (reines Softwarerendering) durchgeführt, um die Geschwindigkeitsrelationen zwischen den verschiedenen Operationen nicht zu verzerren. Die Meßwerte zu den hier abgebildeten Diagrammen finden sich in Anhang B.

5.1.1 Wachstum des Aufwandes mit der Szenenkomplexität

Die wichtigste Eigenschaft des randomisierten z-Buffer Verfahrens ist, daß der Aufwand zur Darstellung einer Szene nur logarithmisch mit der Anzahl von Dreiecken in der Szene wächst. Um dies in der Praxis nachvollziehen zu können, muß eine Szene gewählt werden, in der sich der Detaillierungsgrad variieren läßt, ohne andere Parameter, wie die projizierte Fläche oder die Tiefendynamik, zu verändern.

Als Testszene wurde daher eine Schachbrettszene (siehe Farbbild 3 (a)) verwendet, bei der die Anzahl der (explizit aus Dreiecken modellierten) Felder auf dem Schachbrett schrittweise erhöht wurde, ohne die Ausdehnung der Gesamtszene zu verändern. Es wurde eine weiterentfernte Perspektive mit geringer Tiefendynamik und geringerer projizierter Fläche verwendet, um auch bei kleinen Szenengrößen und damit relativ großen Dreiecken eine vollständige Tiefenunterteilung vornehmen zu können.

Da sich nur einige hunderttausend Dreiecke im Speicher ablegen ließen, mußten für größere Szenen ein oder mehrere Instantiierungsebenen eingefügt werden, um eine größere Szene hierarchisch aus auf einem Gitter angeordneten kleineren Teilschachbrettern zusammenzufügen. Da die Benutzung von zusätzlichen Instantiierungsebenen eine zusätzliche Matrix-Vektor-Multiplikationen für jeden Stichprobenpunkt und jede Ebene erfordert, erhöht sich die benötigte Rechenzeit sprungartig an den Stellen, an denen zusätzliche Instantiierungsebenen eingefügt wurden. Auf die Benutzung von Instanzen konnte nicht verzichtet werden, da sich nur so Szenen in der Größenordnung von 10^{13} Dreiecken kodieren ließen.

Insgesamt wurden maximal 4 Instantiierungsebenen benutzt, die Anzahl von Ebenen ist jeweils über den Kurvensegmenten angegeben. Beim reinen z-Buffer Rendering spielt die Anzahl von Instantiierungsebenen keine Rolle für die Laufzeit, da hier für alle Dreiecke, die unter der selben Projektion gezeichnet wurden, nur eine einzige Matrizenmultiplikation notwendig ist. Die Ergebnisse finden sich in Abbildung 59 (siehe auch Tabelle 3 in Anhang B).

Wie man sieht, wächst der Aufwand des randomisierten z-Buffer Verfahrens bei exponentiell ansteigender Dreiecksanzahl annähernd linear (wenn man von den durch die Instantiierungen verursachten Sprüngen absieht), was die theoretische Voraussage bestätigt. Das konventionelle z-Buffer Verfahren wächst dagegen wie erwartet linear mit der Dreiecksanzahl und daher in der hier verwendeten Darstellung exponentiell mit einer exponentiell wachsenden Dreiecksanzahl.

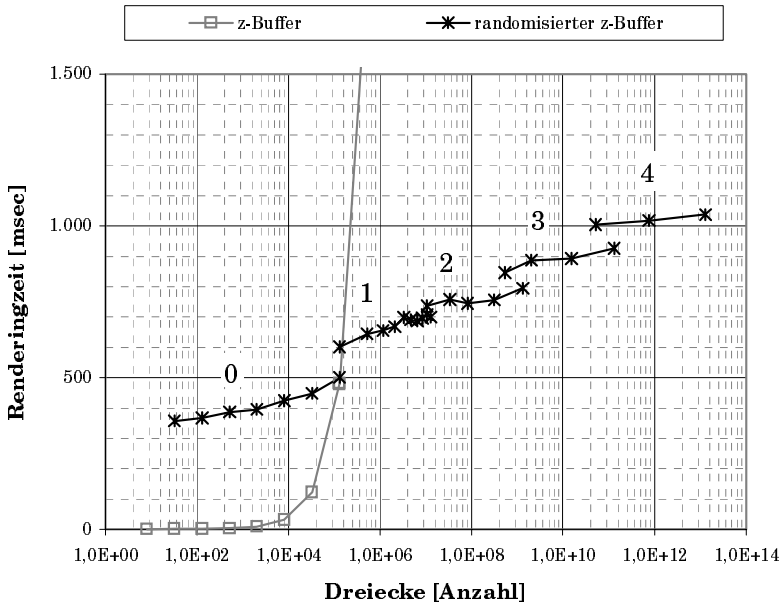


Abbildung 59: Laufzeit bei wachsender Dreiecksanzahl, über den Kurvensegmenten ist die Anzahl von Instantiierungsebenen angegeben.

Auffallend sind kleinere Schwankungen im Verlauf der Meßwerte beim randomisierten Verfahren. Diese sind dadurch zu erklären, daß die Lage der Felder auf dem Schachbrett die räumliche Unterteilung und damit die Schätzung des Projektionsfaktors beeinflußt. Zunächst wurde die Anzahl von Feldern von 2×2 bis auf 256×256 Feldern in Zweierpotenzen erhöht. Auf den höheren Instantiierungsebenen wurden dann auf jeweils der gleichen Ebene zunächst ein Schachbrett, dann 2×2 Instanzen, dann 3×3 bis maximal 10×10 Instanzen verwendet. Durch die Lage der Felder im Raum bzw. durch die Lage der Dummyknoten der Instanzen war die grobe Lage der Boxen auf der obersten Ebene vorgegeben. Je nachdem, ob diese noch zu groß oder bereits klein genug waren, um die Tiefenbedingungen zu erfüllen, mußten diese ggf. weiter unterteilt werden. Daher schwankte die Genauigkeit der Schätzung der projizierten Fläche leicht, je nachdem, ob die weitere Unterteilung die Genauigkeit erhöhte oder nicht. Da die räumliche Abweichung allerdings auf den Faktor 1,4 beschränkt war (die Winkelverhältnisse spielten bei der verwendeten Perspektive keine Rolle), blieben die Störungen beschränkt.

Als Ergebnis läßt sich festhalten, daß der Aufwand zur Behandlung von $1{,}3\cdot10^{13}$ Dreiecken nur etwa drei mal so groß war wie der für 32 Dreiecke. Die Verwendung einer zusätzlichen Instan-

tiierungsebene erhöht den Aufwand hier um 100 msec bei einem Grundaufwand von 360 msec, also im schlimmsten Fall (wenige Dreiecke) um etwa 27%.

Sinngemäß gelten diese Ergebnisse auch für andere Szenen, wie Bäume mit zusätzlichen Blättern oder Landschaften mit zusätzlichen Oberflächendetails. Eine solche Szene wurde hier nicht verwendet, da bei einer solchen Szene sich zusätzlich die projizierte Fläche (leicht) ändern kann und dieser Einfluß nur schwer herausgerechnet werden könnte.

5.1.2 Optimierung der Parameter zur Stichprobenauswahl

Beim hier implementierten Winkelklassenverfahren zur Auswahl von Stichprobenpunkten bleiben noch einige Parameter offen. Es stellt sich die Frage, wie diese gewählt werden sollen, um eine möglichst günstige Laufzeit zu erzielen. Die beiden wichtigsten Parameter sind die Anzahl von Winkelklassen und die maximale Abweichung des Projektionsfaktors bezüglich der Tiefe, im folgenden als Tiefengenauigkeit bezeichnet.

Eine Verbesserung der Tiefengenauigkeit oder der Winkelgenauigkeit verbessert die Schätzung der projizierten Fläche und führt dazu, daß weniger Stichprobenpunkte verwendet werden müssen. Andererseits erhöht sich dadurch gleichzeitig die Anzahl von Boxen, die für die Schätzung bearbeitet werden müssen. Die Auswahl geeigneter Parameter ist damit ein Kompromiß zwischen der Zeit zur Bearbeitung der Boxen und der zur Bearbeitung der Stichprobenpunkte. Die Lage eines Optimums ist dabei von der Szene abhängig: Je nach Tiefendynamik, projizierter Fläche und Winkelverhältnissen ist damit zu rechnen, daß sich eine Erhöhung der Tiefengenauigkeit, eine Erhöhung der Winkelgenauigkeit oder eine generelle Verringerung der Boxenanzahl eher lohnt.

Als Szene wurde hier eine Waldszene aus 400 fraktal erzeugten, auf einem Gitter angeordneten Bäumen verwendet (siehe Farbbild 3 (b)/(c)). Eine solche Szene weist eine relativ stark streuende Verteilung der Oberflächenorientierungen auf, so daß sie der Annahme gleichverteilter Oberflächennormalen nahekommt. Von dieser Szene wurden zwei verschiedene Ansichten mit geringer und größerer Tiefendynamik (Faktor 1,035 bzw. Faktor 18) untersucht, die ein sehr unterschiedliches Verhalten zeigten. Zusätzlich wurde zum Vergleich eine Stadtszene herangezogen (Farbbild 3 (d)).

Für die beiden Ansichten der ersten Testszene wurden verschiedene Werte für die Tiefengenauigkeit und für die Größe der Winkelklassen gewählt. Letztere wurde dadurch gesteuert, daß die Unterteilungstiefe des Winkelunterteilungsbaumes, aus dem die Klassen gebildet wurden (siehe Abschnitt 4.2) auf einen Wert zwischen 0 und 8 festgelegt wurde. Dadurch entstanden entsprechend 1 bis 256 Winkelklassen. Zur Bildrekonstruktion wurde in allen Fällen die Farbklotzmethode mit einem Durchmesser von 2 Pixeln und einer entsprechend gewählten Mindeststichprobendichte verwendet.

Abbildung 60 zeigt das Ergebnis für die Waldszene mit großer Tiefendynamik (siehe auch Tabelle 4 in Anhang B). Wie man sieht, bildet sich für jede Winkelunterteilungstiefe bei variierender Tiefengenauigkeit ein deutliches Minimum aus: Bei sehr kleiner Tiefengenauigkeit wird eine sehr große Boxenanzahl bearbeitet, bei sehr großer Tiefengenauigkeit wird die projizierte Fläche zu stark überschätzt (Die Zusammensetzung der Rechenzeit wird im nächsten Abschnitt noch genauer untersucht). Das Minimum liegt mit abnehmender Anzahl von Winkelklassen immer weiter links, d.h. im Bereich feinerer Tiefenunterteilung. Dies liegt daran, daß die Tiefenunterteilungen für jede Winkelklasse durchgeführt werden müssen, und daher die Boxenanzahlen für Tiefen- und Winkelunterteilung sich multiplizieren. Daher können bei gleichem

Aufwand für die Bearbeitung der Boxen bei einer geringeren Anzahl von Winkelklassen mehr Tiefenunterteilungen vorgenommen werden.

In der Analyse des Winkelklassenverfahrens in Abschnitt 3.3.5.2 wurde eine genau gegenteilige Steuerung der Parameter angenommen, um die verschiedenen Einflüsse auf den Projektionsfaktor beschränken zu können. In der Praxis scheint dagegen das andere Vorgehen bessere Resultate zu liefern, zumindest bei der hier verwendeten Testszene, die insbesondere keine extrem große projizierte Fläche im Verhältnis zur Tiefendynamik aufweist.

Das auffallendste Ergebnis ist, daß bei dieser Testszene die besten Laufzeiten bei einer Winkelunterteilungstiefe von null, also vollständig ohne die Verwendung von Winkelklassen, erzielt werden. Offensichtlich ist der Überhang zur Berechnung der Boxen in der derzeitigen Implementation so groß[22], daß sich eine Verringerung der Winkelklassenanzahl und die damit verbundene Möglichkeit, eine bessere Tiefengenauigkeit zu benutzen, eher lohnt, als die genauere Winkelschätzung: In den hier vorgenommenen Messungen wurden betrachterabgewandte Oberflächennormalen erkannt und frühzeitig nicht weiter bearbeitet. Für völlig zufällige Oberflächenorientierungen würde damit nach Lemma 6 ein Mehraufwand beim Stichprobenumfang durch die Vernachlässigung des Winkeleinflusses von einem Faktor 2 auftreten. Der starke Anstieg des Aufwandes zur Boxenauswahl tritt bei einer größeren Winkelunterteilung allerdings schon ein, bevor die Tiefengenauigkeit diesen Wert deutlich unterschreitet. Daher erscheint plausibel, daß der Verzicht auf die Winkelgenauigkeit zu Gunsten einer größeren Tiefengenauigkeit eher lohnend ist.

Dieses Ergebnis ist noch insofern mit Vorsicht zu behandeln, als daß eine Optimierung der Implementation der Boxenauswahl sowie auch die Verwendung einer anders strukturierten Tests-

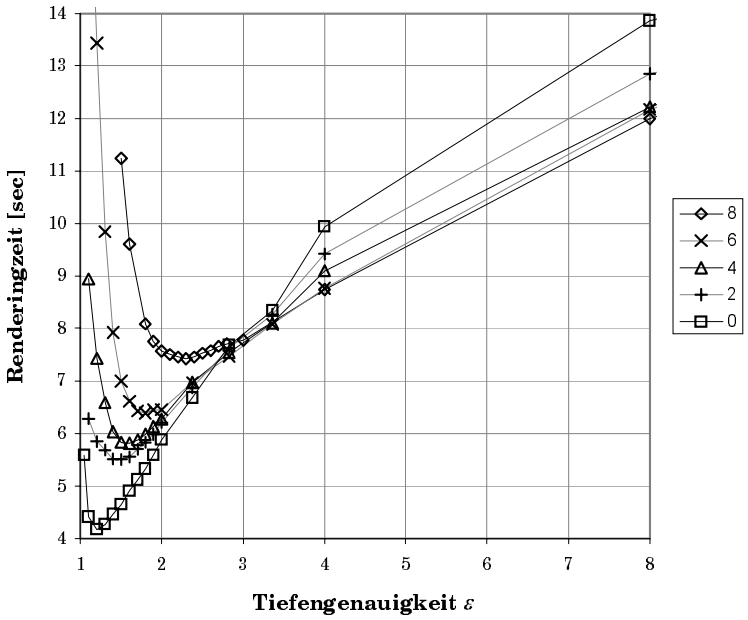


Abbildung 60: Laufzeiten bei verschiedener Tiefengenauigkeit und verschieden großer Anzahl von Winkelklassen (große Tiefendynamik)

[22] Es sollte angemerkt werden, daß die Routine zur Boxenauswahl nicht sehr stark optimiert wurde, insbesondere werden für jede Box aufwendige trigonometrische Berechnungen (arccos) angestellt und dynamische Listen für alle Boxen verwaltet.

zene starken Einfluß darauf haben könnten.

Abbildung 61 zeigt die Ergebnisse bei einer Ansicht der selben Szene mit sehr geringer Tiefendynamik (Ansicht von oben, vgl. Farbbild 3 (c), siehe auch Tabelle 5 in Anhang B). Hier ergibt sich ein völlig anderes Bild: Der Aufwand hängt zunächst kaum von der Tiefengenauigkeit ab und explodiert, sobald der Wert 1,069 unterschritten wird. Dies liegt daran, daß bei diesem Wert die Unterteilung so fein gewählt werden muß, daß auch die relativ gesehen sehr kleine Höhe der Bäume nicht mehr vollständig in eine Box paßt. Um eine bessere Tiefengenauigkeit erreichen zu können, muß aber zunächst der gesamte Wald in entsprechend kleine Boxen unterteilt werden. Der dadurch verursachte Aufwand lohnt sich dann nicht mehr und die Laufzeit schnellt in die Höhe, und zwar um so drastischer, je größer die Anzahl von Winkelklassen ist, da die Boxenanzahl noch damit multipliziert wird. Lediglich bei einer Winkelunterteilungstiefe von null konnte der (in diesem Fall recht hohe) Aufwand durch die weitere Unterteilung noch leicht gesenkt werden, da hier die Boxenanzahl noch in einem unkritischen Bereich verblieb.

Da bei dieser Szene im Bereich einer Tiefengenauigkeit von 1,069 oder größer nur sehr wenige Boxen zur Tiefenunterteilung notwendig sind, lohnt es sich hier eher, eine stärkere Winkelunterteilung zu benutzen. Bei dieser Ansicht der Szene verringert sich die Laufzeit tatsächlich mit der Erhöhung der Winkelunterteilungstiefe, wobei die ersten vier Unterteilungsebenen den größten Gewinn liefern. Dies könnte daran liegen, daß die Szene immer noch Vorzugsrichtungen enthält (die Blätter sind größtenteils nach oben orientiert, die Baumstämme zur Seite), die durch den auf den Winkelwerten aufgebauten *„fair-split-tree“* als erstes voneinander getrennt werden, so daß der Anfangsgewinn am höchsten ausfällt.

Daß der Aufwand mit schwindender Tiefengenauigkeit leicht ansteigt, obwohl bereits eine minimale Anzahl von Boxen verwendet wird, liegt an einer Ungenauigkeit der Implementation: Die positionsabhängige Winkelabweichung wurde hier pauschal nach der vorgegebenen Tiefengenauigkeit und nicht dynamisch nach der tatsächlichen Boxengröße berechnet. Daher tritt der Effekt nicht auf, wenn keine Winkelklassen angelegt werden (Unterteilungstiefe 0).

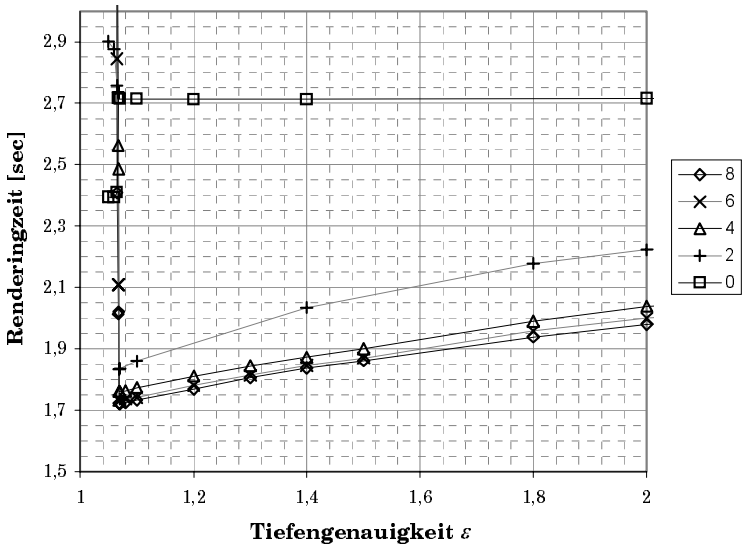


Abbildung 61: Laufzeiten bei verschiedener Tiefengenauigkeit und verschieden großer Anzahl von Winkelklassen (geringe Tiefendynamik)

Abbildung 62 zeigt den Zusammenhang zwischen Laufzeit und Winkelunterteilung noch mal für verschiedene Szenen. Als Wert für die Laufzeit wurde jeweils der beste bestimmte benutzt und auf der x-Achse wurde die Wurzel aus der Anzahl von Winkelklassen eingetragen, also ungefähr die Anzahl von Unterteilungen für jede Polarkoordinatenachse. Zusätzlich zu den beiden Ansichten der Waldszene wurde noch eine Stadtszene in einer ähnlichen Ansicht wie die Waldszene mit großer Tiefendynamik aufgenommen. Es wäre zu vermuten, daß sich hier Winkelunterteilungen eher lohnen, da in dieser Szene nur sehr wenige diskrete Winkel vorkommen, an die sich der *„fair-split-tree“* schnell anpassen kann. Es zeigt sich allerdings ein ähnliches Verhalten wie bei der Waldszene, der Verzicht auf Winkelklassen liefert auch hier die beste Laufzeit. Allerdings wächst die Laufzeit nicht so stark mit wachsender Winkelunterteilung. Dies liegt wahrscheinlich eher daran, daß die Tiefendynamik etwas geringer als bei der Waldszene ist und das Verhältnis der Szenenhöhe zum Szenendurchmesser viel kleiner ist. Dadurch kann der verwendete *„fair-split-tree“* viele Unterteilungen in der Höhe einsparen und daher ein schwächeres Anwachsen der Boxenanzahl mit der Tiefengenauigkeit erreichen: Eine Verkleinerung einer nicht allzu kleinen Box vervierfacht den Aufwand dann nur, anstatt ihn zu verachtfachen.

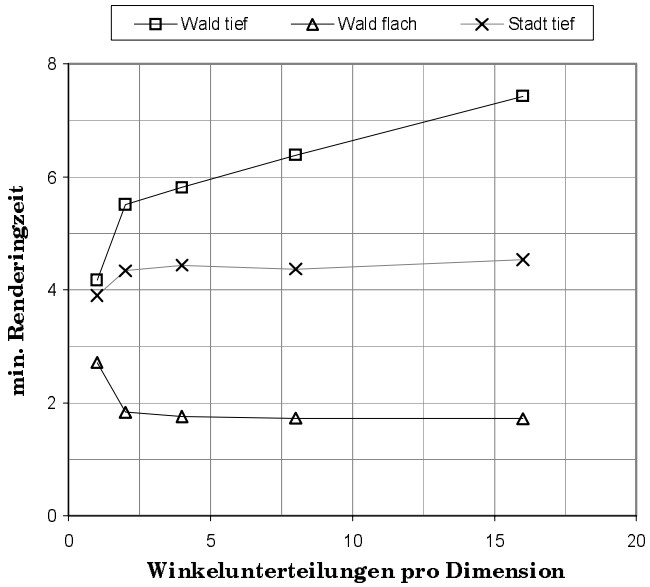


Abbildung 62: Günstigste Laufzeit für verschiedene Winkelunterteilungen und verschiedene Szenen

Zusammenfassend läßt sich festhalten, daß der Gewinn durch die Verwendung von Winkelklassen in Szenen mit keiner stark ausgeprägten Asymmetrie in der Winkelverteilung und größerer Tiefendynamik so gering ist, daß der dadurch verursachte zusätzliche Aufwand diesen übersteigt. Die genaue Wahl der Parameter bleibt damit szenenabhängig. Als Richtlinie empfiehlt sich jedoch für die oben genannte Art von Szenen eher auf Winkelklassen zu verzichten oder nur eine kleine Anzahl von Klassen (etwa Unterteilungstiefe 2-4, entspricht 4-16 Klassen) zu verwenden, da dies, falls es nützt, bereits den größten Teil des Gewinns liefert.

5.1.3 Zusammensetzung des Aufwandes

In diesem Abschnitt soll der Aufwand für das Rendering genauer nach dem Anteil zur Auswahl der Boxen und dem Anteil der Bearbeitung der Stichprobenpunkte aufgeschlüsselt werden. Außerdem wird das Verhalten bei Änderung der Parameter mit den theoretischen Voraussagen verglichen. Als Beispielszene dient hier die Waldszene mit großer Tiefendynamik aus dem vorangegangenen Abschnitt (Farbbild 3 (b)). Als Winkelunterteilungstiefe wurde fest der Wert 4 verwendet, für andere Werte sollte sich im Prinzip ein ähnliches Bild ergeben.

Tabelle 7 (Anhang B) zeigt eine genauere Aufschlüsselung der Rechenzeitanteile. Neben der vollständigen Renderingzeit wurde auch die Zeit gemessen, die zur Bearbeitung der Boxen benötigt wurde. Dazu wurde ein normales Rendering durchgeführt, wobei die Stichprobendichte auf den Wert null gesetzt wurde, so daß keine Stichprobenpunkte genommen und verarbeitet wurden. Zusätzlich wurde die Anzahl von Boxen und die Anzahl von Stichprobenpunkten bei einer korrekten Stichprobendichte bestimmt. Aus der Differenz der Bearbeitungszeit für die Boxen und der Gesamtrenderingzeit konnte so die Zeit für die Bildrekonstruktion, also für die Transformation der Stichprobenpunkte und das Einzeichnen der Farbklötze, bestimmt werden.

Bei dieser Meßmethode wird aller Aufwand, der nicht zum Transformieren und Einzeichnen der Punkte verwendet wird, als Boxenbearbeitungszeit gewertet. Dieser Wert muß nicht zwangsläufig genau proportional zur Anzahl der ausgewählten Boxen sein, da je nach Lage der Boxen und Tiefengenauigkeit mehr oder weniger Boxen traversiert werden müssen, bis die endgültige Menge an Boxen gefunden wird. Auch die Überschreitung von Instantiierungsebenen kann dabei zusätzlichen Zeitaufwand verursachen. Nach Satz 6 ist lediglich garantiert, daß die Bearbeitungszeit und die Anzahl ausgewählter Boxen asymptotisch in der selben Aufwandsklasse (bei wachsender Tiefendynamik) liegen. Abbildung 63 zeigt die Verhältnisse von Stichprobenumfang zur berechneten Rekonstruktionszeit und von der Anzahl ausgewählter Boxen zur zu deren Auswahl benötigten Zeit. Wie erwartet schwankt der Boxendurchsatz relativ stark bei verschiedener Tiefengenauigkeit, während der Durchsatz an Stichprobenpunkten relativ konstant bleibt.

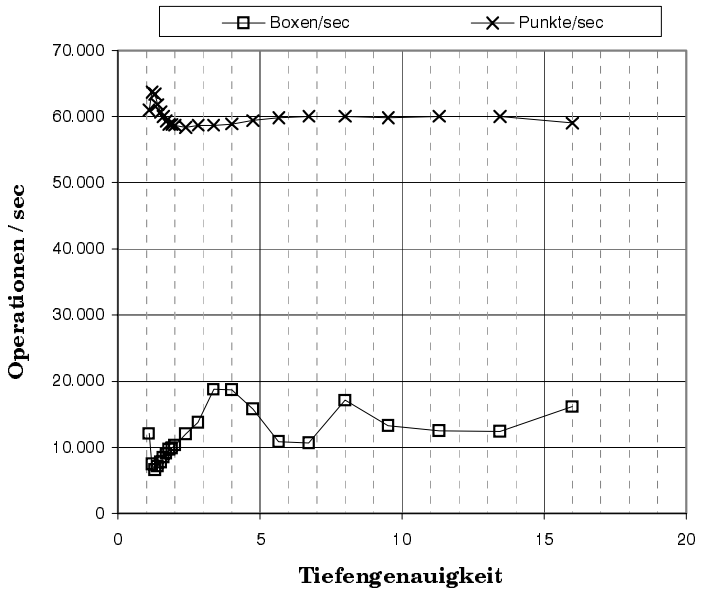


Abbildung 63: Durchsatz an Boxen und Stichprobenumfang bei verschiedener Tiefengenauigkeit

Im Bereich kleiner Tiefengenauigkeit tritt allerdings eine Abweichung des Stichprobendurchsatzes nach oben auf, der mit einer Abweichung des Boxendurchsatzes nach unten korreliert. Dies ist dadurch zu erklären, daß bei einer entsprechend feinen Unterteilung zunehmend Boxen aus der unteren Instantiierungsebene benutzt werden, so daß der Aufwand für die zusätzliche Transformation der Punkte beim Überschreiten der Instantiierungsebene entfällt und der Durchsatz an Stichprobenpunkten damit steigt. Gleichzeitig wird die Auswahl von Boxen dabei teurer, da nun hier zusätzliche Transformationsinformationen beim Übergang zwischen den Ebenen ausgewertet werden müssen. Um einen Eindruck von den Rechenzeitanteilen zu erhalten, sollten die so bestimmten Werte allerdings genau genug sein. Es ist ohnehin zu erwarten, daß der Aufwand zur Boxenauswahl auch noch stark von der Szene selbst abhängen kann.

Trägt man die verschiedenen Zeitanteile gegeneinander auf, so erhält man folgendes Bild (Abbildung 64), das gut den Trade-Off zwischen den beiden Aufwandsparametern veranschaulicht:

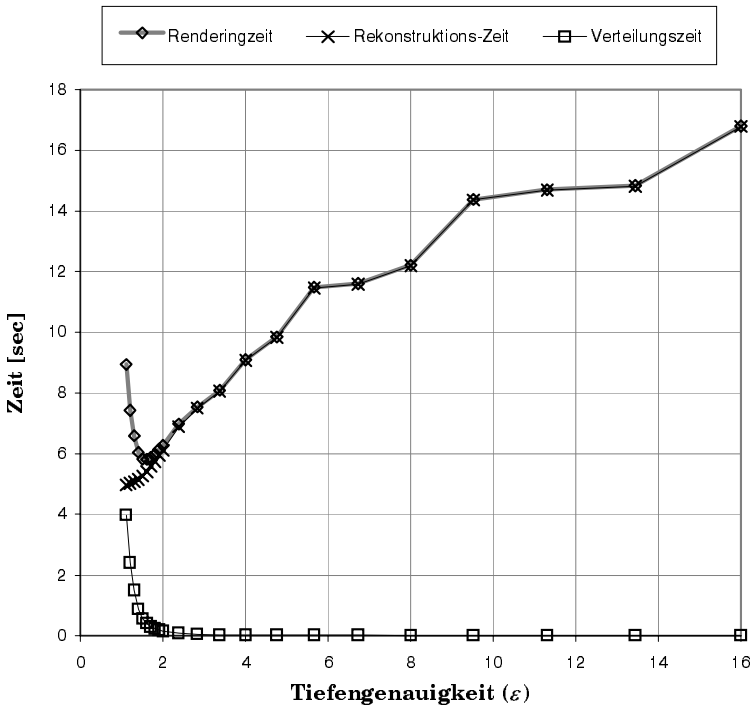


Abbildung 64: Laufzeitanteile beim Rendering mit randomisiertem z-Buffer

Bei großen Werten für die Tiefengenauigkeit dominiert die Zeit zur Bearbeitung der Stichprobenpunkte klar die Laufzeit, während bei geringer Tiefengenauigkeit diese nicht mehr stark reduziert werden kann, statt dessen wächst der Zeitbedarf zur Boxenauswahl stark an. Interessant erscheint hier ein Vergleich des gemessenen Aufwandes mit den theoretischen Voraussagen. Nach Lemma 7 wächst die Anzahl ausgewählter Boxen mit dem Kehrwert der dritten Potenz der relativen Boxengröße, also mit

$$Anz.Boxen \sim \left(\sqrt{\varepsilon} - 1\right)^{-3}$$

Abbildung 65 zeigt diesen Schätzwert im Verhältnis zum gemessenen Wert. Die Schätzung wurde dabei so skaliert, daß die Anfangswerte übereinstimmen. Wie man sieht, stimmen die Schätzung und die gemessenen Werte recht gut überein.

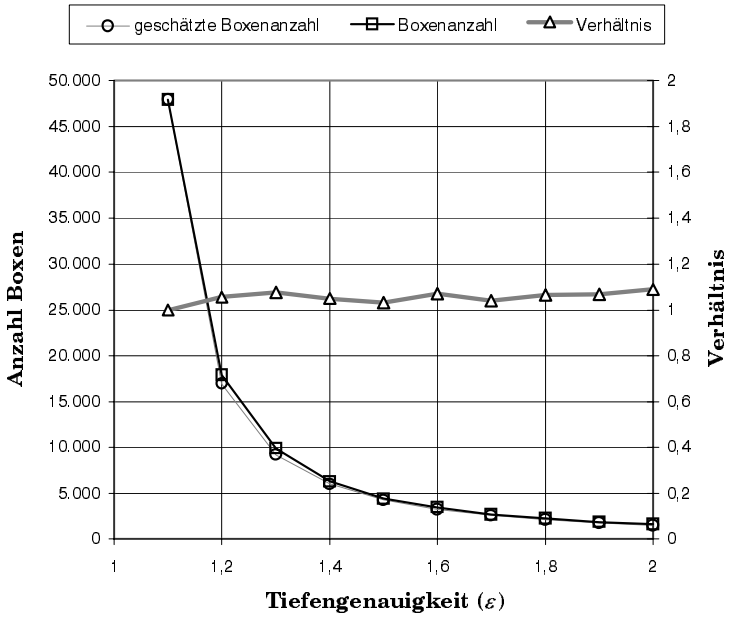


Abbildung 65: Zusammenhang zwischen Boxenanzahl und Tiefengenauigkeit

Wenn man dies für größere Werte der Tiefengenauigkeit (größer 2) durchführt, erhält man eine deutlich geringere gemessene Boxenanzahl. Dies liegt daran, daß die Szene scheibenförmig aufgebaut ist. Wenn die Boxengröße die Baumhöhe überschreitet, sind keine Unterteilungen in der Höhenrichtung mehr notwendig, so daß sich die Boxenanzahl von größenordnungsmäßig $\left(\sqrt{\varepsilon}-1\right)^{-3}$ auf $\left(\sqrt{\varepsilon}-1\right)^{-2}$ reduziert, so daß die Voraussage hier zu pessimistische Werte liefert.

Interessant ist ebenfalls die Abhängigkeit des Stichprobenumfangs von der Tiefengenauigkeit. Zunächst würde man erwarten, daß dieser linear mit der Genauigkeit steigt, vielleicht sogar etwas stärker, da eine feinere Tiefenunterteilung auch eine genauere Schätzung der Winkelverhältnisse ermöglicht. Tatsächlich wächst der Stichprobenumfang nur sublinear mit steigenden Werten für ε. Abbildung 66 zeigt den gemessenen Stichprobenumfang, die gestrichelte Linie gibt den aus dem ersten Meßwert geschätzten linearen Zusammenhang wieder.

Ein Grund für dieses Verhalten ist darin zu finden, daß die Überschätzung des Projektionsfaktors nicht über den gesamten Tiefenbereich der Box konstant ist, sondern erst mit zunehmender Tiefe in jeder Box wächst. Um eine genauere Schätzung zu erhalten, sei nun angenommen, daß der Raum wie im Beweis von Satz 6 in Tiefenbereiche eingeteilt ist, in denen jeweils der maximale Projektionsfaktor als Schätzwert für den tatsächlichen Projektionsfaktor verwendet wird. Die Tiefenbereiche vergrößern sich, beginnend beim minimalen Tiefenwert, mit jedem Bereich um den Faktor $\sqrt{\varepsilon}$. Unter dieser Annahme soll dann das projizierte Volumen (Definition 14) bestimmt und mit dem exakten Wert verglichen werden.

Sei also a die minimale und b die maximale Tiefe der Szene, sowie ε der Faktor für die Tiefengenauigkeit. Der geschätzte Wert ergibt sich dagegen als:

$$\sum_{k=0}^{\frac{2\ln\frac{a}{b}}{\ln\varepsilon}} \frac{a\left(\sqrt{\varepsilon}\right)^{k+1} - a\left(\sqrt{\varepsilon}\right)^{k}}{a^2\left(\sqrt{\varepsilon}\right)^{2k}} = \frac{b\sqrt{\varepsilon}-a}{ab} \in \Theta\left(\sqrt{\varepsilon}\right)$$

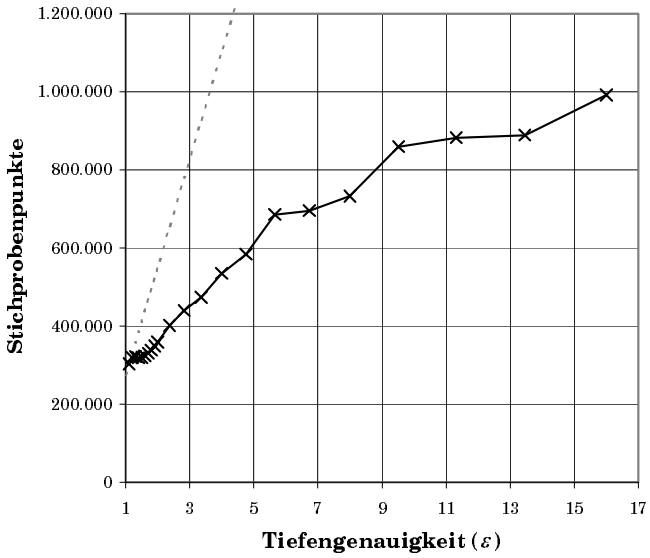


Abbildung 66: Wachstum des Stichprobenaufwandes in Abhängigkeit vom Tiefengenauigkeitsfaktor ε

Dabei gibt der Nenner unter der Summe den geschätzten Projektionsfaktor wieder, der Zähler die Tiefe des Tiefenbereiches. Die Obergrenze der Summe ist so gewählt, daß der Tiefenbereich von a bis b vollständig überdeckt wird. Die Summe wurde mit dem Computeralgebrasystem MAPLE berechnet[23].

Die Überschätzung des Projektionsfaktors nimmt unter diesen Annahmen also nur proportional zu $\sqrt{\varepsilon}$ zu. Tatsächlich wird dies durch die Meßwerte bestätigt: Abbildung 67 zeigt das Quadrat des gemessenen Stichprobenumfangs der hier betrachteten Waldszene aufgetragen über der Tiefengenauigkeit. Es ergibt sich ein annähernd linearer Zusammenhang, allerdings fallen einige Ausreißer auf. Diese sind dadurch zu erklären, daß die Unterteilung der Szene in diskreten Schritten erfolgt. Da hier ein „*fair-split-tree*“ verwendet wurde, schrumpfen die Boxen bei bestimmten Verkleinerungsschritten besonders stark, da z.B. mehrere Bäume vollständig voneinander getrennt werden konnten, was die Schätzung der Stichprobe genauer macht, als bei einem vollständig gefüllten Raum anzunehmen wäre. Um diesen Einfluß auszuschließen, wurde das Experiment noch einmal mit einer Schachbrettszene (aus der gleichen Perspektive) wiederholt. Die Szene war so gebildet, daß keine „Lücken“ zwischen den Objekten blieben und daher o.g. Sprünge nicht auftreten konnten. Das Ergebnis findet sich in Abbildung 68. Hier treten kaum noch Abweichungen auf (die verbleibenden Abweichungen sind wohl auf die diskrete Natur der Unterteilung zurückzuführen). Der analytisch vorausgesagte Zusammenhang wird hier gut bestätigt.

Zusammenfassend läßt sich festhalten, daß sich der Aufwand zum Rendering in zwei Anteile zerlegen läßt: die Boxenauswahl und die Stichprobenbearbeitung. Der Aufwand zur Boxenauswahl wächst in der Praxis für Szenen, deren Ausdehnung in allen Richtungen noch größer als die relative Boxengröße ist, grob mit $\Theta\left(\left(\sqrt{\varepsilon}-1\right)^{-3}\right)$. Der Mehraufwand durch die Überschätzung

[23] Dies ist natürlich nicht als formaler Beweis sondern, lediglich als Stütze der Interpretation der Meßergebnisse zu sehen.

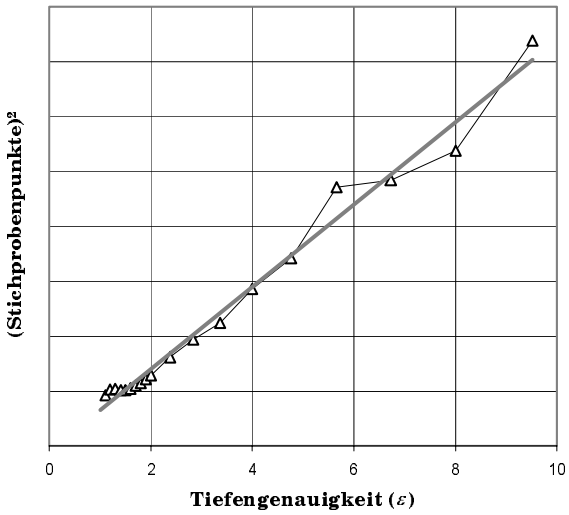


Abbildung 67: Quadrat des Stichprobenumfangs bei verschiedener Tiefengenauigkeit, Waldszene

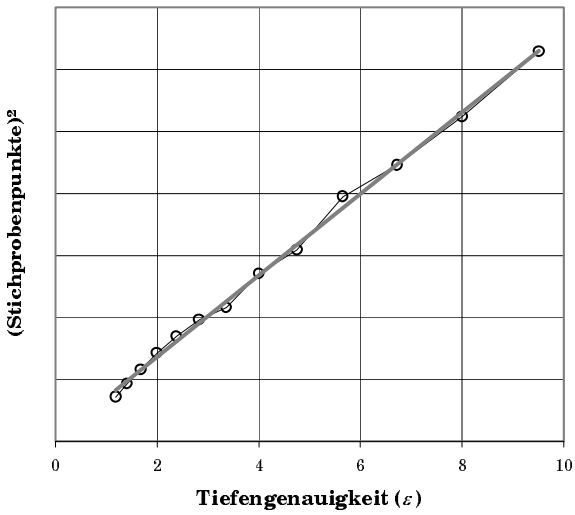


Abbildung 68: Quadrat des Stichprobenumfangs bei verschiedener Tiefengenauigkeit, Schachbrettszene

der Stichproben wächst bei einem wachsenden Tiefengenauigkeitsfaktor ε nur mit $\Theta\left(\sqrt{\varepsilon}\right)$, falls die Objekte gleichmäßig in der Tiefe verteilt sind. Dadurch gibt sich in allen hier untersuchten Szenen ein ausgeprägtes Minimum für den Aufwand. Die genaue Lage ist allerdings von der Szene und der Winkelunterteilungstiefe abhängig. Das relativ schwache Wachstum des Mehraufwandes mit wachsendem ε läßt es für die Praxis ratsam erscheinen, bei Unkenntnis der optimalen Parameter den Wert für ε eher etwas zu groß als zu klein anzunehmen.

5.1.4 Kombination mit konventionellem z-Buffer Rendering

Ein wesentlicher Faktor, der beim randomisierten z-Buffer Rendering zu hohen Laufzeiten führt, ist die starke Überfüllung der Flächen. Damit alle Zellen (d.h. Pixel bzw. von Farbklötzen abgedeckte Bereiche) gefüllt werden, müssen im Durchschnitt logarithmisch (in der Anzahl Zellen) mal mehr Zellen gefüllt werden als vorhanden sind, damit alle sicher berücksichtigt werden. Für jede Operation muß dabei eine vollständige perspektivische Transformation vorgenommen werden. Es ist daher nicht lohnend, ausgedehnte Oberflächen aus einzelnen Dreiecken mit dem randomisierten Verfahren aufzufüllen, da das konventionelle z-Buffer Verfahren hierzu nur maximal drei Transformationen und für jeden Pixel nur eine, sehr schnelle Pixelfülloperation benötigt.

Eine naheliegende Idee ist daher, zumindest die nähere Umgebung des Betrachters in der Szene mit dem konventionellen z-Buffer Verfahren darzustellen und nur weiter entfernte Objekte mit dem randomisierten Verfahren zu behandeln. Im allgemeinen müßten, wie in Abschnitt 3.3.6.4 beschrieben, die Größen der Objekte und die Betrachtungswinkel mit einbezogen werden. Falls die Szene aus relativ grobdetaillierten Objekten mit nicht allzu extremen Größenunterschieden besteht, ist allerdings auch mit dem einfachen Ansatz einer reinen „Fernapproximation“ ein Laufzeitgewinn zu erwarten.

In dem folgenden Experiment wurde eine Kombination von konventionellem z-Buffer Rendering innerhalb eines Würfels mit einem vorgegebenen Durchmesser und randomisiertem z-Buffer

Rendering außerhalb implementiert. Innerhalb des Quadrates wurde ein approximatives Clipping auf den von der Sichtpyramide abgedeckten Bereich wie in Algorithmus 9 durchgeführt. Als Szene wurde eine Stadtszene verwendet, in der die meisten Objekte eine relativ ähnliche Größe aufwiesen und alle Objekte aus wenigen Dreiecken bestanden. Dadurch trat nur am Horizont eine Überdetaillierung durch eine große Anzahl der an sich einfachen Objekte auf, und ein z-Buffer Rendering der näheren Umgebung erschien lohnend. Von der Szene wurden 10×10 Instanzen auf einem regulären Gitter angeordnet, so daß insgesamt 1.894.600 Dreiecke entstanden. Farbbild 4 (a) zeigt die für die Messungen verwendete Ansicht der Szene.

Die Meßergebnisse für verschiedene Durchmesser des z-Bufferbereiches findet sich in Abbildung 69 (siehe auch Tabelle 8 in Anhang B):

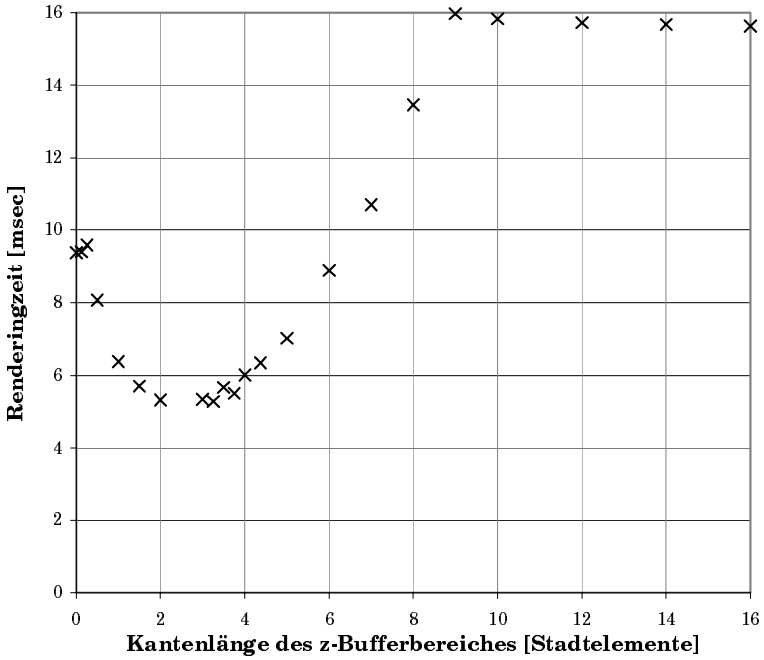


Abbildung 69: Renderingzeit bei kombiniertem z-Buffer / rnd. z-Buffer Rendering in einem festen Umkreis bei verschiedenem Radius

Die Laufzeit konnte durch die Verwendung des konventionellen z-Buffers im Nahbereich auf 57% des Wertes bei einem Rendering ausschließlich mit dem randomisierten z-Buffer verringert werden, gegenüber einem reinen konventionellen z-Buffer Rendering erzielt die gemischte Strategie im besten Fall eine Verbesserung um den Faktor 3 (für größere Szenen wäre hier natürlich noch ein stärkerer Gewinn zu erwarten). Für Szenen dieser Art erscheint die kombinierte Strategie also durchaus lohnend.

Ein Echtzeitszenario

Das randomisierte z-Buffer Verfahren wurde mit dem Ziel entwickelt, die Darstellung komplexer Szenen in Echtzeit zu ermöglichen. Der hohe Überhang durch die starke Überfüllung der Pixel führt allerdings, wie die Zeitmessungen in Kapitel 5.2 noch deutlich belegen werden, meistens zu Laufzeiten im Sekundenbereich.

Die in diesem Abschnitt verwendete Testszene weist allerdings eine relativ kleine projizierte Fläche auf, so daß unter zusätzlicher Ausnutzung aller verfügbaren Verbesserungen (Rekon-

struktion mit Farbklötzen, optimal gewählte Verteilungsparameter, Kombination mit z-Buffer Rendering, hardwareunterstützte geometrische Transformationen) tatsächlich Laufzeiten in der Nähe des Echtzeitbereiches erzielt werden können.

Die Szene wurde auf Testsystem 2 dargestellt, es wurden keine separaten Verteilungsbäume für Instanzen in der Szene angelegt, da das Einfügen von einer Ebene von Dummyknoten auf diesem System die Laufzeit bereits um 50% erhöht, da die hierzu *zusätzlich* erforderlichen Matrix/Vektormultiplikationen unter der Benutzung von OpenGL als Schnittstelle nicht von der Hardware durchgeführt werden können. Daher mußte die Anzahl von Instanzen auf 5×5 reduziert werden (473.650 Dreiecke), um den Speicherbedarf zu begrenzen.

Das z-Buffer Rendering des Nahbereiches wurde mit einem randomisierten z-Buffer Rendering mit 5 Pixel breiten Farbklötzen kombiniert, was eine für eine Fernapproximation gerade noch tolerable Bildqualität lieferte. In dieser Konstellation konnte eine Renderingzeit von 367 msec für eine Auflösung von 640×480 Punkten (siehe Farbbild 4 (b)) und 187 msec für eine Auflösung von 400×300 Punkten erreicht werden, gegenüber 1,56 sec für ein vollständiges z-Buffer Rendering. Dies liegt zumindest so weit in der Nähe des Echtzeitbereiches (20-100 msec), daß dieser durch weitere Optimierungen von Hard- und Software erreichbar scheint. Insbesondere eine Optimierung der zusätzlichen Transformationen bei der Verwendung von Instanzen könnte entsprechende Ergebnisse auch für komplexere Szenen ermöglichen, bei denen ein reines z-Buffer Rendering nicht mehr effizient wäre.

Entscheidend für eine kleine Laufzeit bleibt aber eine kleine projizierte Fläche. Auf derzeit verfügbarer PC-Hardware ist eine projizierte Fläche in der Größenordnung eines vollständig bedeckten 640×480 Punkte großen Fensters nicht in Echtzeit handhabbar. Denkbar wäre in einem solchen Fall lediglich eine Amortisierung, bei der eine Fernapproximation für eine ganze Reihe von Bildern mit konventionell erstelltem (wenig detaillierten) Vordergrund verwendet wird.

5.2 Bildqualität und Performance bei komplexen Szenen

Neben den bisher gezeigten Bildern sollen in diesem Kapitel einige Ergebnisse mit komplexeren Szenen dargestellt werden. Die Frage ist hier, wie stark sich Artefakte des randomisierten z-Buffer Verfahrens in solchen Szenen zeigen und wie die Qualitätsunterschiede zwischen den verschiedenen Alternativen zur Bildrekonstruktion ausfallen. Es wurde versucht, Szenen zusammenzustellen, die verschiedene Charakteristika von typischen Anwendungsgebieten aufweisen. Da die Modellierung von Szenen ein sehr breites Feld ist, kann die vorliegende Auswahl in keiner Weise vollständig sein. Verzichtet wurde auf komplexe Gestaltung der Oberflächenschattierung, insbesondere wurden keine Texturen verwendet. Dies erschien weniger notwendig, da diese auch häufig eingesetzt werden, um Details einzusparen, und das hier beschriebene Verfahren darauf weniger stark angewiesen ist.

Auf die Szenen wurden jeweils die folgenden Methoden angewendet, jeweils für eine Bildauflösung von 640×480 Punkten:

- **Konventionelles z-Buffer Rendering**: Diese Methode dient als Referenz für Bildqualität und Rechenzeit.
- **Pixelweise Bildrekonstruktion:** Diese Methode liefert die höchste Auflösung, die mit dem randomisierten Verfahren erreichbar ist.

- **Große Farbklötze:** Um die recht hohen Rechenzeiten bei der pixelweisen Bildrekonstruktion zu verringern, wurden dieselben Bilder mit größeren Farbklötzen mit 2, 3, 5 und 10 Pixeln Durchmesser berechnet. An der Bilderfolge kann man erkennen, welches Bildmaterial für gröbere Darstellungen besser geeignet ist und welches eher empfindlich darauf reagiert.
- **Mittelwertbildung:** Aus einer Folge von 10 Bildern, die mit Farbklötzen mit 2 Pixeln Durchmesser erstellt wurden, wurde der arithmetische Mittelwert gebildet. In den entsprechenden Bildern tritt ein vermindertes Rauschen auf, systematische Artefakte (im Erwartungswert des Bildes) sind dadurch deutlicher zu erkennen.
- **Diskrete Voronoiregionen:** Mit jeweils dem selben Stichprobenumfang wie bei der Farbklotzmethode mit Durchmesser 5 wurde eine Rekonstruktion mit diskreten Voronoiregionen vorgenommen. Die Ergebnisse sollten daher mit diesen Bildern verglichen werden. Zur Verdeckungserkennung wurde Algorithmus 4 (frei bewegliche Zellen) mit einem jeweils per Hand gewählten Tiefentoleranzbereich und der Unterdrückung von Punkten mit abgewandten Normalen eingesetzt.
- **Gaußscher Tiefpaßfilter:** Unter den selben Bedingungen wurde auch eine Rekonstruktion mit einem Gaußschen Tiefpaßfilter vorgenommen, wobei ein relativ großer Filterdurchmesser ($c = 10$, doppelte Verdeckungsauflösung) verwendet wurde. In diesen Bildern erscheinen die fehlenden Informationen durch den geringen Stichprobenumfang als Unschärfe. Die Bilder sollten mit den Ergebnissen der diskreten Voronoiregionen und der Farbklotzmethode mit Durchmesser 5 verglichen werden. Um absolut gesehen für die verwendete Auflösung eine gute Bildqualität zu erzielen, waren der Stichprobenumfang zu klein und der dadurch bedingte Filterdurchmesser zu groß.

Die folgenden fünf Szenen wurden als Testszenen ausgewählt; die ersten beiden zeigen Fälle, die mit vielen der bekannten Methoden nicht in den hier erreichten Zeitschranken darstellbar gewesen wären. Die drei weiteren Szenen sind nicht komplex genug, um Geschwindigkeitsvorteile aus dem randomisierten z-Buffer Rendering zu ziehen, sie sollen lediglich bestimmte Eigenschaften des Verfahrens zeigen. Die jeweils für alle Szenen und Verfahren entstandenen Bilder finden sich zusammen mit der Angabe der zugehörigen Laufzeiten im Farbteil in Anhang C.

Stadtszenen

Farbbild 5 und Farbbild 6 zeigen Szenen aus Häusern, Bäumen, Straßen, Autos und Laternen. Die Szenenelemente wurden an der Universität-GH Paderborn im Rahmen der Projektgruppe *„Realzeitalgorithmen für Walkthrough-Animationen“* erstellt. Die Modelle wurden auf einem vorgegebenen Raster per Hand angeordnet. Zusätzlich wurden mehrere Instanzen der so definierten Stadtteile auf einem Gitter angeordnet. Bei der Übersichtsszene wurden insgesamt 100×100 Instanzen mit Hilfe von 2 Instantiierungsebenen (ein 10×10 Gitter von 10×10 Gittern) verwendet. Daneben wurde noch eine Ansicht aus der Nähe berechnet, die nur 10×10 Instanzen einer halb so großen Stadt zeigt. Die Übersichtsszene besteht aus 3.434.780.000 Dreiecken, von denen 343.478 explizit gespeichert wurden. Die kleinere Szene besteht aus 17.731.500 Dreiekken, von denen 177.315 explizit gespeichert wurden.

Die Stadtszene hat die Eigenschaft, daß die einzelnen Objekte relativ wenige Details enthalten, die Menge aller Objekte zusammen aber nicht mehr in Echtzeit handhabbar ist. Da sich eine solche Szene kaum durch *„mesh simplification“* vereinfachen läßt und aus erhöhter Perspektive auch nur wenig Verdeckung aufweist, stellt sie ein typisches Anwendungsgebiet für das rando-

misierte z-Buffer Verfahren dar. Eine Besonderheit ist, daß die meisten Modelle Oberflächendetails durch Oberflächendetailpolygone modellieren. Daher treten hier Artefakte durch die fehlende „Trennung der Verdeckung“ (siehe Abschnitt 3.2.1) zu Tage.

Ergebnisse: Die große Szene (Farbbild 5) läßt sich mit Hilfe des randomisierten z-Buffer Verfahrens wesentlich schneller als mit dem gewöhnlichen z-Buffer Verfahren darstellen. Das randomisierte Verfahren benötigt für eine Darstellung in guter Qualität (Bild (c), Farbklötze mit $d=2$) 33 Sekunden, während ein vollständiges Rendering[24] mit dem konventionellen z-Buffer Verfahren etwa 2 Stunden benötigt hätte (geschätzt nach Transformationsleistung). Bei der kleineren Szene (Farbbild 6) läßt sich erst bei verminderter Bildqualität (größere Farbklötze) ein Gewinn erzielen, da diese im Verhältnis zur projizierten Fläche noch relativ wenige Details enthält.

Im Vergleich der Bildqualität zum z-Buffer Verfahren fällt auf, daß beim randomisierten Verfahren Bereiche mit extrem vielen Details durch ein Farbrauschen dargestellt werden, während das z-Buffer Verfahren dazu neigt, systematische Aliasingartefakte zu produzieren. Gut sichtbar wird dies im Bereich nahe des Horizontes in Farbbild 5 (a) bzw. (b), vor allem an den Fassaden der Hochhäuser. Das Rauschen kann beim randomisierten z-Buffer Verfahren durch Mittelwertbildung reduziert werden (Bild (d)), dabei tritt ein leichtes Aliasing an den Fassaden der Hochhäuser im mittleren Entfernungsbereich zutage, gegenüber dem konventionellen z-Buffer Rendering fällt die Bildqualität jedoch deutlich besser aus.

Da die Übersichtsansicht sehr viele feine Details (vor allem die vertikalen Linien der Hochhäuser) aufweist, nimmt die Bildqualität sehr deutlich mit zunehmendem Radius der Farbklötze ab. Die Rekonstruktion mit Voronoiregionen liefert ein Bild, in dem Vordergrunddetails nicht so stark „auslaufen“ und den Hintergrund fälschlicherweise überdecken. Es entsteht allerdings ein neues Problem: Dadurch, daß zur Verdeckungserkennung ein für das ganze Bild einheitlicher Tiefentoleranzbereich benutzt wurde, werden in einigen Bereichen die Verdeckungen falsch wiedergegeben (unten rechts in Farbbild 5 (g) scheinen unter den roten Dächern der Häuser z.B. die weißen Wände durch). Wenn man dagegen den Toleranzbereich verkleinert, werden in vielen Bereichen des Bildes viele nützliche Stichprobenpunkte gelöscht, so daß sich dadurch die Bildqualität ebenfalls verringert. Für das hier gezeigte Bild wurde ein Kompromißwert nach optischem Eindruck gewählt. Dies gilt gleichermaßen für die Rekonstruktion mit einem Gaußschen Tiefpaßfilter (Farbbild 5 (h)). Aufgrund der relativ starken Unschärfe fällt der Fehler hier allerdings weniger auf.

In der Nahansicht (Farbbild 6) werden die in Abschnitt 3.2.1 beschriebenen Probleme mit Oberflächendetailpolygonen deutlich sichtbar: Die Straßen, sowie Fenster und Türen von Häusern und Autos in dieser Szene wurden durch in geringem Abstand aufgesetzte Polygone modelliert. Wie erwartet scheint unter diesen der Untergrund durch. Der Effekt nimmt mit steigender Entfernung von Beobachter zu, da größere Tiefenbereiche auf einen Pixel fallen und die relativen Abstände zwischen den beiden jeweils beteiligten Schichten damit kleiner werden. Gleichermaßen nimmt der Effekt mit wachsendem Durchmesser der Farbklötze zu, da dadurch ebenfalls ein größerer Tiefenbereich auf einen Farbklotz fällt.

Ein ähnlicher Effekt ist an den Stellen zu erkennen, an denen verschiedenfarbige Flächen an einer verdeckten Kante aneinander stoßen: Betrachtet man das Haus im Vordergrund (vor allem in Bild (f), (i)), so erkennt man, daß die obere Kante des weißen Hauses durch das rote Dach fälschlicherweise durchscheint. Dies tritt ebenfalls auf, weil für den jeden Farbklotz ein konstan-

[24] Bild (a) zeigt nur einen Ausschnitt, bei dem die Szene nur einem kleineren Umkreis dargestellt wurde; ein kleiner Streifen am Horizont wurde abgeschnitten, dieser enthält jedoch den größeren Teil der Geometrie.

ter Tiefenwert angenommen werden muß und die Verdeckungssituation an den aneinanderstoßenden Kanten nicht mehr über die gesamte Fläche eines Pixels bzw. Farbklotzes eindeutig ist. Entsprechend tritt der Effekt bei einem größeren Durchmesser der Farbklötze stärker zutage.

Betrachtet man scharfe Farbkanten im Bild (z.B. die Kanten der Häuser im Vordergrund), so erkennt man, daß wie erwartet hier ein starkes Rauschen (bzw. ein zufälliges Ausfransen der Kanten im Standbild) auftritt. Dies fällt, zumindest bei Standbildern, stärker auf als das Rauschen in überdetaillierten Bereichen, da hier keine so klare (Kanten-) Struktur erkannt wird und das Rauschen daher natürlicher erscheint.

Insgesamt zeigt sich, daß Szenen in der Art dieser Stadtszenen bei entsprechend großer Komplexität mit einem großen Gewinn an Rechenzeit dargestellt werden können. Einen guten Kompromiß zwischen Rechenzeit und Bildqualität erhält man etwa mit der Farbklotzmethode mit einem Farbklotzdurchmesser von $d=2$. Bei der Modellierung sollte beachtet werden, Oberflächendetails lieber durch Texturen darzustellen oder die darunterliegenden Flächen nicht mit in das Modell aufzunehmen. Aufgrund der relativ großen projizierten Fläche einer solchen Szene ist allerdings eine Darstellung in Echtzeit auch auf einer sehr leistungsfähigen Plattform (Testsystem 2) nicht zu erreichen. Trotzdem können erhebliche Laufzeitgewinne gegenüber dem konventionellen z-Buffer Verfahren (z.B. Faktor 220 für die große Szene und $d=2$) erreicht werden.

Für Stadtszenen mit größerem Durchmesser fallen die relativen Gewinne, gemessen am konventionellen z-Buffer Rendering, stärker aus als für Szenen mit kleinerem Durchmesser. Dies liegt daran, daß die projizierte Fläche wesentlich schwächer mit dem Durchmesser wächst als die geometrische Komplexität, wie die folgende Analyse zeigt. Das schwache Wachstum der projizierte Fläche ermöglicht überhaupt erst die Darstellung großer Szenen dieser Art alleine mit dem randomisierte z-Buffer Verfahren.

Wenn man für eine „Stadtszene“ vereinfachend annimmt, daß Objekte innerhalb eines ebenen, scheibenförmigen Bereiches gleichmäßig verteilt und zufällig orientiert sind, kann die projizierte Fläche, die beim randomisierten z-Buffer Algorithmus die Laufzeit wesentlich bestimmt, durch das projizierte Volumen (Definition 14) des sichtbaren Sektors S der Scheibe geschätzt werden. Man erhält als projiziertes Volumen:

$$prj(S) = \int_S \frac{1}{Tiefe(x)^2}\,dx = \int_{\Theta(1)}^{r} \frac{Querschnittsfläche(t)}{t^2}\,dt$$

$$= \int_{\Theta(1)}^{r} \frac{\Theta(t)}{t^2}\,dt = \int_{\Theta(1)}^{r} \Theta\left(\frac{1}{t}\right)dt = \Theta\left(\log r\right)$$

Da die Querschnittsfläche einer „scheibenförmigen“ Szene ab einer festen Tiefe nur linear mit der Entfernung zunimmt, während der Projektionsfaktor quadratisch abnimmt, erhält man im Integral nur eine logarithmische Zunahme des projizierten Volumens mit zunehmendem Radius.

Nach Satz 5 wächst der Stichprobenumfang linear mit der projizierten Fläche, also logarithmisch mit dem Durchmesser der Szene. Zusätzlich wächst der Aufwand zur Einteilung der Szene in Boxen beim Winkelklassenverfahren nach Satz 6 logarithmisch mit dem Szenendurchmesser. Die Verteilung und Bearbeitung der Stichprobenpunkte benötigt nach Satz 8 $\Theta((m+K)\cdot\log n)$ Zeit mit K als Anzahl von Boxen, m als Stichprobenumfang und n als Anzahl von Dreiecken. Eingesetzt erhält man einen Aufwand von $\Theta((\log r+\log r)\cdot\log r^2) = \Theta(\log r\cdot 2\cdot\log r) = \Theta(\log^2 r)$, da n mit r^2 wächst.

Dies bedeutet, daß in diesem Sinne „stadtartige“ Szenen alleine mit dem randomisierten z-Buffer Verfahren, ohne weitergehende Erkennung von Verdeckungen, mit einem Aufwand dargestellt werden können, der nur polylogarithmisch mit dem Durchmesser der Szene wächst, während der Aufwand für ein konventionelles z-Buffer Rendering quadratisch mit dem Durchmesser wächst. Die Ergebnisse mit der Stadtszene belegen experimentell, daß sich sehr große Szenen mit diesen Eigenschaften handhaben lassen. Die hier angestellten Überlegungen gelten im übrigen auch für andere zweidimensional ausgedehnte „scheibenförmige“ Szenen mit gleichmäßig verteilten Oberflächen, wie z.B. für die im folgenden Abschnitt betrachteten Waldszenen.

Waldszenen

Die Szenen bestehen aus vielen Instanzen einiger fraktal erzeugter Bäume. Zur Modellierung der Bäume wurde eine vereinfachte Variante des von [Weber und Penn 95] beschriebenen Verfahrens implementiert. Auch diese Szenen haben die Eigenschaft, daß sich die einzelnen Objekte kaum noch geometrisch vereinfachen lassen und daß wenig Verdeckung auftritt. Anders als bei der Stadtszene sind hier die einzelnen Objekte allerdings schon so komplex, daß auch im Nahbereich eine Approximation für eine effiziente Darstellung sinnvoll ist.

Ergebnisse: Das randomisierte z-Buffer Verfahren liefert für Szenen dieser Art, gemessen an einem konventionellen z-Buffer Rendering, in sehr kurzer Zeit eine ansprechende Darstellung (siehe Farbbild 7). Während eine Darstellung der insgesamt 93.177.000.000 Dreiecke mit dem z-Buffer Verfahren auch auf dem leistungsfähigen Testsystem 2 etwa 25 Stunden benötigt hätte, liefert die randomisierte Variante bereits in etwa 4 Sekunden eine akzeptable Approximation (siehe Bild (e)). Im Vergleich zur Stadtszene erhält man aufgrund der geringeren projizierten Fläche eine deutlich kürzere Renderingzeit.

Die große Flut an Details im oberen Bereich des Bildes, nahe des Horizontes, wird beim randomisierten z-Buffer Rendering auch hier wieder als Farbrauschen wiedergegeben, was einem natürlichen Eindruck relativ nahekommt. Durch Mittelwertbildung läßt sich dieses in einen Durchschnittsfarbton überführen, allerdings auf Kosten deutlich höherer Rechenzeiten. Ebenfalls gut sichtbar ist, daß Details unterhalb der Pixelgröße (z.B. einige Äste von Bäumen im mittleren Bereich) im Mittelwertbild berücksichtigt werden, während im nicht gemittelten Bild nur ein entsprechend kleiner zufälliger Anteil von Pixeln dieser Details dargestellt wird.

Auffallend sind noch einige schwarze Punkte auf den Blättern der Bäume im vorderen Bereich, die in allen Bildern mehr oder weniger stark auftreten. Diese stammen von den dunkel dargestellten Unterseiten der Blätter. Bei den Bäumen im vorderen Bereich, insbesondere bei den ersten acht Bäumen, sind diese von der gewählten Perspektive aus allerdings eigentlich nicht sichtbar (Bei den hinteren Bäumen sind durchaus auch Unterseiten von Blättern zu sehen, da hier ein flacherer Blickwinkel herrscht). Die Punkte sind Fehler, die dadurch entstehen, daß durch die nahe beieinander liegenden Schichten (Ober- und Unterseiten der Blätter) keine getrennte Verdeckung mehr vorliegt, und so die verdeckten Teile wie zuvor bei den Oberflächendetailpolygonen durchscheinen können. Der Effekt nimmt dementsprechend mit wachsendem Farbklotzdurchmesser zu. Aus technischen Gründen konnte in dieser Szene die Unterdrückung von Oberflächen mit betrachterabgewandter Normalen, die das Problem zumindest an dieser Stelle behoben hätte, nicht angewendet werden.

Farbbild 8 zeigt die gleiche Szene, die zusätzlich mit einem Waldboden und Grashalmen versehen wurde. Durch die hinzugekommene Geometrie steigt die Gesamtanzahl an Dreiecken auf 678.905.000.000. Die Szene wurde auf Testsystem 1 berechnet, wodurch die Renderingzeiten etwas höher ausfielen. Zusätzlich wurde die Renderingzeit wesentlich erhöht durch die größere projizierte Fläche, die durch den zusätzlichen „Boden“ entstand.

Ein konventionelles z-Buffer Rendering aller Dreiecke hätte schätzungsweise 39 Tage für jedes dieser Bilder benötigt (bei einer angenommenen Projektionsleistung von maximal 200.000 Dreiecke pro Sekunde auf Testsystem 1), so daß die hier erreichten Zeiten im zweistelligen Sekundenbereich einen großen Fortschritt darstellen.

Die Bilderfolge zeigt die Szene aus 4 verschiedenen Blickwinkeln mit zunehmendem Abstand. An diesem Beispiel sieht man, daß das randomisierte z-Buffer Verfahren adaptiv für jeden Standpunkt die notwendigen Details auswählt. Während in der Nahaufnahme einzelne Blätter und Grashalme korrekt wiedergegeben werden, wird in der Übersichtsansicht nur noch ein zufälliges Farbrauschen wiedergegeben, das nur noch statistisch die Anteile dieser Elemente am Bild wiedergibt, so daß die gröberen Strukturen erkennbar werden.

Die Fehler durch fälschlicherweise dargestellte dunkle Blattunterseiten treten bei dieser Szene nicht auf, da hier die Unterdrückung abgewandter Oberflächennormalen eingesetzt werden konnte.

„Happy Buddha“

Das in Farbbild 9 dargestellte Modell einer Buddha Statue ist eine Standardszene zum Testen von *„mesh simplification“*-Methoden: Die Oberfläche wurde mit einem 3D-Scanner erzeugt, so daß in großen Bereichen viele fast komplanare Dreiecke auftreten. Der Datensatz wurde u.a. in [Garland 99] verwendet, um das dort beschriebene *„mesh simplification“*-Verfahren zu testen. Der Datensatz wird vom Autor unter der im Literaturverzeichnis angegebenen Adresse zusammen mit weiteren Modellen (u.a. dem Kuhmodell) im Internet bereitgestellt. Die Szene enthält 1.085.634 Dreiecke, was sich durch *„mesh simplification“* ohne starken Verlust bis auf 2% reduzieren läßt ([Garland 99], Seite 12). Das Modell soll hier dazu dienen, die Bildqualität mit der von *„mesh simplification“*-Verfahren vergleichen zu können. Die Laufzeiten dieser Methoden lassen sich, zumindest bei einer vollformatigen Darstellung, mit dem randomisierten z-Buffer Verfahren nicht erreichen, da bei diesem Modell, im Gegensatz zu den beiden zuvor beschriebenen Szenen, die meisten Dreiecke redundant sind und daher starke Vereinfachungen mit *„mesh simplification“* möglich sind.

Ergebnisse: Die Darstellung mit kleinem Durchmesser der Farbklötze ist von einem z-Buffer Rendering praktisch nicht zu unterscheiden. Insbesondere treten keine Verdeckungsfehler auf der Oberfläche auf, und die Schattierung der Oberfläche wird korrekt wiedergegeben. Im Gegensatz zu *„mesh simplification“*-Verfahren tritt eine zunehmend gröbere Approximation hier durch ein zunehmende, gleichmäßige Unschärfe im Bild zu Tage, während erstere in jedem Fall ein scharfes Bild liefern, in dem zunehmend geometrische Details fehlen.

Es bleibt festzuhalten, daß sich auch einfach strukturierte, zusammenhängende Oberflächen mit wenigen wesentlichen Details mit dem randomisierten z-Buffer Verfahren in hoher Qualität darstellen lassen, im Gegensatz zu anderen Methoden wie der nach [Chamberlain et al. 95] (Ersetzung von Würfeln im Octree durch vorberechnete Farbwerte). Der Laufzeitgewinn fällt hier allerdings eher gering aus, mit *„mesh simplification“*-Verfahren sind deutlich größere Einsparungen möglich. Das vereinfachte Modell aus [Garland 99], Figure 1.2 (b) (Seite 12) hätte, geschätzt nach der verfügbaren Transformationsleistung auf der hier verwendeten Plattform, weniger als etwa 0,04 Sekunden für ein z-Buffer Rendering benötigt, was bei höherer Qualität immer noch schneller als die Berechnung der stark vereinfachte Darstellung in Bild (i) wäre.

Schachbrett

Die Szene besteht aus zu einem Schachbrettmuster angeordneten weißen und schwarzen Quadraten, die explizit aus Dreiecken modelliert wurden. Eine Darstellung in Echtzeit wäre problemlos mit Texturen möglich. Hier soll die Szene vor allem dazu dienen, Rauschen, Kantenschärfe und Aliasing bei den verschiedenen Verfahren zu vergleichen.

Ergebnisse: Die Darstellung mit dem z-Buffer Verfahren in Farbbild 10 (a) zeigt mit zunehmender Entfernung vom Betrachter deutliche Aliasingartefakte. Im Vergleich dazu liefert das randomisierte Verfahren im mittleren Tiefenbereich praktisch dieselben Artefakte, während im hinteren Bereich ein unstrukturiertes Rauschen entsteht. Die Fehler im mittleren Bereich erklären sich, wie in Abschnitt 3.2.1.3 beschrieben, dadurch, daß durch die starke „Überfüllung" der Pixel das randomisierte z-Buffer Rendering im wesentlichen die selben Ergebnisse wie ein konventionelles z-Buffer Rendering produziert. Mit zunehmender Entfernung wird die Anzahl von Dreiecken, aus denen zufällig ausgewählt wird jedoch immer größer, ohne daß sich die Anzahl von Stichprobenpunkten, die auf jeden Pixel fallen, erhöht. Die Auswahl zwischen schwarzen und weißen Dreiecken wird dadurch zunehmend zufällig und damit verschwinden im hinteren Bereich die systematischen Artefakte und es tritt ein gleichförmiges Rauschen auf. Das konventionelle z-Buffer Verfahren berücksichtigt dagegen auch im hinteren Bereich immer alle Dreiecke, so daß hier auch in diesem Bereich starke, systematische Artefakte erhalten bleiben.

Mit wachsendem Durchmesser der Farbklötze nehmen die systematischen Fehler ab, da dadurch größere Mengen von Dreiecken von jedem Klotz repräsentiert werden. Die systematischen Aliasingartefakte lassen sich beim randomisierten Verfahren auch dadurch vermindern, daß die Überfüllung der Pixel verringert wird. So zeigt Bild (g), bei dem das Bild durch Voronoiregionen anstatt durch große Farbklötze aufgefüllt wurde, bei Verwendung einer relativ kleinen Punktdichte (1/16 gegenüber Bild (b)), nur noch wenige systematische Aliasingartefakte, die erst im gemittelten Bild (i) deutlicher hervortreten. Diese treten vor allem deswegen auf, weil die Methode lediglich ein *„unweighted area sampling"* annähert. Die Verwendung von Tiefpaßfiltern bei der Bildrekonstruktion führt zu besseren Ergebnissen: In Bild (h) treten keine systematischen Aliasingartefakte mehr auf. Der geringe Stichprobenumfang (genauso groß wie in Bild (g)) führt jedoch zu einer starken Unschärfe und einem auffallenden, niederfrequenten Rauschen. Mit einem größeren Stichprobenumfang lassen sich wesentlich bessere Ergebnisse erzielen, die praktisch frei von Aliasing, Rauschen und Unschärfe sind: In Bild (j) wurde der Stichprobenumfang 160mal so groß wie in Bild (h) gewählt[25] wodurch ein sehr gutes, fast störungsfreies Bild bei gleichzeitig sehr großem Zeitaufwand entsteht. Für das *„distributed raytracing"* sind Techniken entwickelt worden, um eine derartige Bildrekonstruktion schneller durchführen zu können (siehe z.B. [Glassner 95a] für eine Übersicht). Diese können allerdings auf das randomisierte z-Buffering zumeist nicht angewendet werden, da sie eine freie Auswahl der Stichprobenpunkte in der Bildebene voraussetzen.

Es läßt sich festhalten, daß das randomisierte Verfahren weniger anfällig für Aliasing als das konventionelle z-Buffer Verfahren ist, viele systematische Aliasingartefakte lassen sich in weniger störendes Rauschen umwandeln. Die Erzeugung hochqualitativer Bilder mit beliebig geringem Aliasing und Rauschen ist zwar prinzipiell möglich, der hohe Stichprobenaufwand, der dazu notwendig ist, läßt dies allerdings in vielen Fällen unpraktikabel erscheinen.

---

[25] Zur Berechnung mußte, wie schon bei den Beispielen aus Abschnitt 3.2.3.3 eine Mittelwertbildung über mehrere Bilder mit geringerem Stichprobenumfang durchgeführt werden, die gemessene Laufzeit ist daher relativ ungünstig. Die Auflösung des Bildes mußte ebenfalls aus technischen Gründen etwas verringert werden.

Fraktale Landschaft

Die Szene in Farbbild 11 zeigt ein fraktal erzeugtes Höhenfeld (im Groben nach der Methode aus [Foley et al. 96], Seite 1022ff erstellt). Die Szene enthält viele relativ weiche Farbübergänge und praktisch keine harten Farbkontraste an Kanten außer am Horizont. Für eine solche Szene kann der Durchmesser der Farbklötze durchaus bis auf 10 Pixel erhöht werden, ohne daß allzu starke Qualitätsverluste sichtbar werden. Lediglich ein Auslaufen von Kanten (speziell an den Graten der Berge sichtbar) fällt stärker ins Auge. Im Vergleich zur Stadtszene aus Farbbild 5 zeigt sich also, daß Szenen dieser Art wesentlich besser für starke Vereinfachungen geeignet sind. Da harte Farbkontraste fehlen, tritt auch praktisch kein Rauschen im Bild auf. Die hier verwendete Szene enthält allerdings zu wenige Dreiecke, um das randomisierte z-Buffer Verfahren in bezug auf die Laufzeit gewinnbringend einsetzen zu können. Eine weitere Erhöhung der Detaillierung war aus Speicherplatzgründen nicht möglich. Dies unterstreicht nicht zuletzt, wie wichtig eine speichereffiziente Repräsentation der Szene ist. Ohne die Verwendung von Instanzen (wie in diesem Fall) ist es praktisch unmöglich mit derzeit üblichen Speichergrößen Szenen zu verwalten, bei denen das randomisierte z-Buffer Verfahren größere Laufzeitgewinne erbringen kann.

Die Szene wurde, im Gegensatz zu allen anderen hier gezeigten, mit dynamischer Beleuchtung berechnet (man beachte die Glanzlichter der Lichtquelle auf dem Wasser). Das randomisierte z-Buffer Verfahren gibt hierbei insbesondere die Interaktion der Lichtquelle mit der rauhen Wasseroberfläche gut wieder. Beim randomisierten z-Buffering wurden sowohl die Interpolation von Normalvektoren als auch die der Materialparameter aktiviert. Das z-Buffer Rendering wurde mit Gouraudinterpolation (zweidimensionale, lineare Interpolation der Farbwerte auf den Dreiecken, siehe [Foley et al. 96]) berechnet. Die Glanzlichter auf dem Wasser werden durch die Interpolation der Normalvektoren, die im Prinzip der Phonginterpolation [Foley et al. 96] entspricht, etwas klarer wiedergegeben als durch die einfache Farbinterpolation. Bei der relativ hoch detaillierten Szene fällt der Effekt allerdings nicht sehr stark auf. Deutlicher sichtbar werden die Vorteile in Szenen mit wenigen Dreiecken, wie in Farbbild 12 gezeigt: Auf dem roten Zylinder werden die Glanzlichter der beiden Lichtquellen wesentlich besser durch die Normaleninterpolation als durch eine einfache Farbinterpolation wiedergegeben. Da diese Szenen mit anderen Renderingverfahren wesentlich effizienter wiedergegeben werden können als mit dem randomisierten z-Buffer, sind die Vorteile der besseren Interpolation allerdings begrenzt.

Gesamtergebnis

Laufzeit: Die randomisierte Methode erreicht bei extrem hoch detaillierten Szenen wie Waldszenen oder Übersichtsansichten von Städten starke Laufzeitgewinne beim Rendering.

Mit herkömmlichen Methoden wäre dies kaum zu erreichen gewesen: Szenen dieses Typs lassen sich mit „*mesh simplification*"-Methoden nicht vereinfachen und bei bildbasierten Methoden wäre ebenfalls damit zu rechnen, daß z.B. die komplexen Verdeckungen der Blätter und Äste untereinander nur unzureichend wiedergegeben würden.

Auch bei vielen Beschleunigungsmethoden für Raytracingverfahren wäre für solche Szenen mit Performanceproblemen zu rechnen. Insbesondere in den Waldszenen laufen viele Sehstrahlen knapp an einer großen Zahl von Blättern und Ästen vorbei, bis sie auf eine Oberfläche auftreffen. Die häufig verwendeten Octree- oder Volumenhierarchiemethoden müßten daher für jeden Strahl eine größere Anzahl von Boxen bzw. Teilvolumina erfolglos nach Schnittpunkten absuchen, bis sie den tatsächlichen Schnittpunkt finden. Dies bleibt allerdings eine Vermutung, da eine experimenteller Vergleich mit Raytracingmethoden im Rahmen dieser Arbeit nicht möglich war.

Die einzige Methode unter den in Kapitel 2 vorgestellten, die hochdetaillierte Waldszenen in ähnlich kurzer Zeit darstellen kann, ist die Methode nach [Chamberlain et al. 95], die Szenenteiel durch farbige, halbtransparente Octreeboxen ersetzt. Während diese für solche unstrukturierte Waldszenen ebenfalls eine gute Bildqualität liefert, stößt sie bei großflächigen Oberflächen in der Art des „Happy Buddha"-Modells (die Autoren zeigen das Beispiel eines Büstenmodells, ebenfalls aus einem 3D-Scanner) auf Probleme, die beim randomisierten z-Buffer Verfahren nicht auftreten. Damit behandelt das neue Verfahren eine größere Klasse von Szenen mit ähnlicher Effizienz.

Insgesamt erschließt das randomisierte z-Buffer Verfahren neue Klassen von Szenen für eine Darstellung in kurzer Zeit, bei denen dies mit den zuvor genannten Verfahren nicht möglich gewesen wäre. Die Darstellung in Echtzeit ist beim derzeitigen Stand der Technik allerdings aufgrund des notwendigen großen Stichprobenumfangs noch nicht zu erreichen, zumindest nicht, solange Szenen formatfüllend in einer üblichen Auflösung (hier 640×480 Punkte) dargestellt werden sollen.

Bildqualität: Das randomisierte z-Buffer Verfahren liefert bei pixelweiser Bildrekonstruktion bzw. bei der Rekonstruktion mit sehr kleinen Farbklötzen (2-3 Pixel) eine Qualität, die mit der eines gewöhnlichen z-Buffer Renderings durchaus vergleichbar, in bezug auf Aliasingartefakte sogar teilweise besser ist. Die größten Probleme treten bei sehr nahe beieinander liegenden Oberflächen auf (speziell bei Oberflächendetailpolygonen). Hier führt die mangelnde Trennung der Verdeckung zu Pixeln mit falsch rekonstruierter Verdeckung. Diese Probleme nehmen mit wachsendem Durchmesser der Farbklötze zu.

Eine Vergrößerung der Farbklötze bzw. eine Verringerung des Stichprobenumfangs führt zu erhöhtem Rauschen bzw. Unschärfe an allen scharfen Kanten im Bild. Inwieweit dies tolerierbar ist, hängt stark vom Bildmaterial ab. Der Einsatz aufwendigerer Bildrekonstruktionsmethoden (Voronoiregionen, Tiefpaßfilter) erbringt nach optischem Eindruck nur relativ wenig Verbesserung. Speziell die Tiefpaßfiltermethoden stellen allerdings eine Möglichkeit dar, auch in ungünstigen Szenen Aliasingartefakte zu unterdrücken, wobei allerdings ein sehr großer Stichprobenumfang notwendig wird. Ohne die Verwendung von Filtern kann auch das randomisierte z-Buffer Verfahren Aliasingartefakte produzieren. In der Regel treten allerdings weniger Artefakte auf als beim konventionellen z-Buffer Verfahren. Lediglich in dem Bereich, in dem noch relativ wenige Details auf jeden Pixel fallen (typischerweise in einem mittleren Entfernungsbereich), treten ähnliche starke Artefakte wie beim konventionellen Verfahren auf.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung der Ergebnisse

In dieser Arbeit wurde ein neues Renderingverfahren zur Darstellung hochdetaillierter Szenen, das „randomisierte z-Buffer Verfahren", vorgestellt. Das Verfahren erzeugt ein Bild einer gegebenen Szene aus einer zufällig gewählten Stichprobe von im Bild gleichverteilten Oberflächenpunkten. Die Laufzeit ist linear in der Stichprobengröße, hinzu kommt der Aufwand zur Auswahl der Stichproben.

Die erwartete Stichprobengröße, die zu einer erfolgreichen Konstruktion eines Bildes notwendig ist, beträgt $a \cdot \ln v$, wenn a die gesamte und v die sichtbare projizierte Fläche in Pixeln der gegebenen Auflösung des zu erzeugenden Bildes ist. Das erhaltene Bild ist mit (beliebig) hoher Wahrscheinlichkeit korrekt, wenn für jeden Pixel die starke Annahme gemacht werden kann, daß verdeckende und verdeckte Flächen unter dem Pixel in disjunkten Tiefenintervallen liegen. In der Praxis treten nachweislich dann Probleme auf, wenn ausgedehnte Oberflächen sehr dicht beieinander liegen.

Ein unschärferes Bild kann auch in höherer Auflösung schneller konstruiert werden, wenn statt Pixeln Farbklötze verwendet werden, was die Bildqualität je nach Durchmesser der Klötze entsprechend reduziert. Mit aufwendigeren Verfahren zur Konstruktion des Bildes (Tiefpaßfilter, Voronoiregionen) und Heuristiken zur Erkennung von Verdeckungen können bei gleichem Stichprobenumfang bessere Bilder als mit der Farbklotzmethode erzeugt werden und Aliasingeffekte vermieden werden. In der Praxis dominiert der Laufzeitaufwand für die aufwendigere Bildrekonstruktion oft die Gewinne durch den verringerten Stichprobenumfang. Die Heuristiken zur Verdeckungserkennung bereiten in allgemeinen Szenen Probleme, die hier nicht zufriedenstellend gelöst werden konnten.

Es wurde ein Algorithmus vorgestellt, der mit Hilfe eines Octrees den Raum unterteilt und näherungsweise gleichverteilte Stichprobenpunkte in logarithmischer Zeit bezüglich der Anzahl von Dreiecken in der Szene liefern kann. Der Octree wird vorberechnet und dann werden dynamisch daraus geeignete Boxen mit Objekten ausgewählt. Der Aufwand für diese Auswahl wächst dabei zusätzlich maximal logarithmisch mit dem Verhältnis vom Szenendurchmesser zum minimalem Betrachtungsabstand. Da der Algorithmus keine exakte Gleichverteilung liefert, muß dies durch eine ggf. größere Stichprobenanzahl kompensiert werden. Im schlimmsten Fall kann der dadurch verursachte Mehraufwand beliebig groß werden. Unter der Annahme, daß die Orientierungen in der Szene zufällig und gleichmäßig verteilt sind, kann der erwartete Aufwand allerdings auf einen konstanten Faktor (4ε für $\varepsilon > 1$) begrenzt werden. Die Einteilung der Szene in Klassen ähnlicher Oberflächenorientierung erlaubt es, den Faktor mit vertretbarem Aufwand für die Einteilung noch leicht zu verbessern. Die Beobachtung, daß der erwartete Fall unter praxisnahen Annahmen über die Verteilung der Normalenrichtungen der Dreiecke im Raum deutlich günstiger als der schlimmste Fall ausfällt, ist wesentlich für die Praktikabilität des Verfahrens verantwortlich, da alle Versuche, eine Approximation der Verteilung mit garantierten Fehlerschranken auch im worst-case zu erhalten, zu ernsten Komplexitätsproblemen führten.

Bei der Wahl der Anzahl von Winkelklassen und der Tiefe der räumlichen Unterteilung ergibt sich ein Trade-Off zwischen der Genauigkeit der Stichprobenauswahl und dem dazu erforderlichen Aufwand. Eine optimale Wahl der möglichen Parameter hängt stark von der verwendeten

Szene ab, in praktischen Beispielen erwies sich allerdings tendenziell eine geringe Anzahl von Winkelklassen und eine nicht übertrieben starke räumliche Unterteilung als unkritisch.

Unter der Annahme einer gleichmäßigen Verteilung der Oberflächennormalen erhält man als Gesamtergebnis (mit dem „halbdeterministischen“ Algorithmus 11 zur Verteilung) eine erwartete Laufzeit von

$$\Theta\Big(\big(a \log v + \log \tau\big) \cdot \log n\Big)$$

wenn a die gesamte und v die sichtbare projizierte Fläche ist, τ die Tiefendynamik angibt und n die Anzahl von Dreiecken der Szene bezeichnet.

Das Verfahren wurde auf eine Reihe von Szenen angewandt, die mit vielen anderen, bisher bekannten Verfahren nur unter hohem Aufwand darstellbar gewesen wären. Da der Renderingaufwand des randomisierten z-Buffer Verfahrens nur logarithmisch mit der Dreiecksanzahl wächst, waren für Szenen, die z.B. eine Übersichtsansicht einer sehr großen Stadt oder einer Waldlandschaft aus mehreren hundert Milliarden Dreiecken zeigten, Renderingzeiten im Sekundenbereich bei akzeptabler Bildqualität erreichbar. Ein konventionelles z-Buffer Rendering hätte hier unter Umständen Tage gedauert, und viele bekannte Beschleunigungsmethoden, vor allem die in anderen Fällen sehr erfolgreiche automatische Vereinfachung von Dreiecksnetzen (*„mesh simplification“*) wären nicht anwendbar gewesen.

Eine Darstellung in (nahe-) Echtzeit war nur für Szenen mit geringer projizierter Fläche zu erreichen, ein vollständig bedecktes Bild in üblicher Auflösung (etwa 640×480 Punkte) benötigte stets einige Sekunden zur Berechnung.

Als Ergebnis erhält man ein Verfahren, daß für extrem hochdetaillierte Szenen mit moderater Verdeckung innerhalb der Szene in vergleichsweise sehr kurzer Zeit, wenn auch nicht unbedingt in Echtzeit, ein approximative Darstellung in guter Qualität liefert. Dabei ist sowohl das algorithmische Prinzip wie auch dessen Implementation vergleichsweise einfach.

Betrachtet man die Konsequenzen dieser Ergebnisse für das Walkthrough-Problem an sich, so sieht man, daß eine Reduktion von überschüssigen Details aus einer überdetaillierten Szene in für praktische Anwendungen sehr allgemeinen Fällen mit Hilfe eines recht elementaren Prinzips möglich ist. Die effiziente Behandlung von worst-case Szenen scheitert allerdings an der Komplexität der Aufgabe, geometrische Objekte bis auf einen konstanten Fehler genau nach ihrer projizierten Fläche zu klassifizieren.

Das Verfahren bringt keinen Nutzen in Szenen mit sehr starker Verdeckung, da hier der Stichprobenumfang zu groß wird. Wenn man die anderen Einschränkungen des Verfahrens für praktische Anwendungen hinnehmen kann, reduziert sich die Lösung des Walkthroughproblems auf die Lösung des Problems der Unterdrückung verdeckter Flächen. Das neue Verfahren zeigt ein Stück weit, daß das Kernproblem des Walkthroughs im Bereich der Verdeckung von Oberflächen und ihrer effizienten Erkennung liegt, während sich die Reduktion von Details für eine relativ große Klasse von Szenen mit geringen Einschränkungen universell behandeln läßt.

6.2 Weiterführende Fragen und Ausblick

Die in dieser Arbeit vorgestellten Ergebnisse werfen eine Reihe weiterführender Fragen auf, die zu weiteren Untersuchungen auf diesem Gebiet führen könnten.

Im Bereich der Analyse und Implementation der Algorithmen bleibt noch offen:

- Eine genauere Analyse des 2-Ebenen-Winkelbaumes, insbesondere eine genauere Einordnung des erwarteten Aufwandes.
- Eine Implementation dieser Datenstruktur und ein Vergleich ihrer Effizienz mit den hier implementierten Verfahren. Insbesondere die Adaptivität der Unterteilung des Winkelbereiches durch den Quadtree kann vermutlich nur in einer experimentellen Untersuchung voll berücksichtigt werden.
- Eine Implementation einer dynamischen Version der Verteilungsbäume.
- Eine Implementation eines Größenklassenverfahrens oder eines anderen Ansatzes, um große Dreiecke effizienter behandeln zu können und eine analytische und eine experimentelle Untersuchung der Effizienz.
- Eine weitergehende Untersuchung der verschiedenen Bildrekonstruktionsverfahren. Hier könnten vor allem die Heuristiken zur Verdeckungserkennung für eine bessere Ausnutzung der Stichprobe verfeinert werden, um auch in allgemeineren Szenen korrekte Ergebnisse ohne eine manuelle Parameterwahl durch den Benutzer zu ermöglichen.

Darüber hinaus sind verschiedene Erweiterungen des Verfahrens in hier noch nicht beschriebene Richtungen möglich:

- Dynamische Erkennung verdeckter Flächen: Beim Einzeichnen der Stichprobenpunkte in den Tiefenpuffer stehen bisher nicht genutzte statistische Informationen zur Verfügung: Für jede Box kann (auch hierarchisch) eine „Trefferquote“ definiert werden, die angibt, wie viele Punkte von Tiefenpuffer zurückgewiesen wurden. Eine Strategie könnte darin bestehen, Boxen mit schlechter Trefferquote nicht vollständig einzuzeichnen, sondern statt dessen zunächst auf Boxen mit besserer Quote auszuweichen. Wenn dieser Vorgang so gesteuert wird, daß Boxen tendenziell von vorne nach hinten gezeichnet werden, kann der Prozeß abgebrochen werden, wenn die Anzahl der gezeichneten Punkte in der Größenordnung der Zellen des Bildes liegt.
  Eine Realisierung dieses Prinzips, das quasi zu einer randomisierten Variante des hierarchischen z-Buffer Verfahrens [Green et al. 93] führen würde, würde zunächst eine genaue Untersuchung der statistischen Eigenschaften des dabei ablaufenden Prozesses erfordern. Anschaulich ist unklar, ob und inwieweit die statistischen Informationen über die „Treffer“ im Tiefenpuffer gewinnbringend eingesetzt werden können. Danach wird insbesondere ein Algorithmus benötigt, der während der laufenden Stichprobenauswertung dynamisch schätzt, wie stark die Fläche zur Zeit bedeckt ist; es ist zu vermuten, daß dieses Problem nicht einfach zu lösen wäre.
- Es stellt sich weiter die Frage, ob durch eine geschickte Stichprobenauswahl der logarithmische Überhang bezüglich der Auflösung verringert oder vermieden werden könnte. Denkbar wäre eine Vorauswahl „günstiger“ aus einer größeren Menge rein zufälliger Stichprobenpunkte um zumindest einen Teil des Berechnungsaufwandes zu sparen.
- Eine weitere interessante Frage, die hier nicht beantwortet werden konnte, wäre, ob auch eine rein deterministische Version dieses Verfahrens mit ähnlichen Garantien für die Bildqualität denkbar wäre, und wie diese aussehen könnte. Vielleicht ließe sich damit der bisher notwendige Überhang weiter reduzieren.

In Hinblick auf eine praktische Anwendung des Verfahrens würden ebenfalls noch ein Reihe von Aufgaben anfallen, darunter:

- Eine Implementation einer Fernapproximation, die über jeweils mehrere Bilder gültig bleibt und in Kombination mit einem anderen Renderingverfahren für den Nahbereich verwendet werden kann. Dabei müßten insbesondere noch Probleme wie das Rauschen der randomisierten Approximation behandelt werden, dazu kommt z.B. das beschriebene Verfahren mit exponentieller Mittelung in Frage.
- Für bestimmte Klassen von Szenen wäre auch eine Integration mit bekannten Verfahren zur Unterdrückung verdeckter Flächen, z.B. mit *„potentially visible sets“*-Methoden interessant. Hierzu müßte u.U. der verwendete Unterteilungsbaum angepaßt werden (z.B. als BSP-Baum mit beschränktem Verhältnis von kleinstem zu größtem orthogonalen Durchmesser, um die Ergebnisse für den Octree sinngemäß übertragen zu können und trotzdem eine flexible Zelleneinteilung zu ermöglichen).
- Für einen praktischen Einsatz wichtig wäre eine automatische Bestimmung der Parameter Tiefengenauigkeit und Winkelklassenanzahl. Dies könnte bei aufeinanderfolgenden Bildern einer stetigen Bewegungssequenz u.U. dynamisch erfolgen, indem eine Gradientenabstiegsmethode eingesetzt wird. Um auch die Winkelunterteilung dynamisch zu ändern, wäre ein 2-Ebenen-Baum notwendig.

Neben diesen sind sicherlich noch einige weitere Ansätze und Fragestellungen denkbar, die weiter verfolgt werden könnten.

Anhang A: Verwendete Hardware

Alle Messungen in dieser Arbeit wurden auf gewöhnlicher PC-Hardware durchgeführt. Als Betriebsystem wurde Microsoft WindowNT 4.0 verwendet. Als Hardwareplattform kamen zwei verschiedene Systeme zum Einsatz:

Testsystem 1: Ein etwas älterer Arbeitsplatzrechner mittlerer Leistung

Hardware:

Prozessor:	Intel Pentium II, 233 Mhz
System:	Asus LX 97 Motherboard mit Intel LX Chipsatz, 66 Mhz Speicher und Systemtakt.
Speicher:	96 MB
Grafikkarte:	Asus V3000 AGP Grafikkarte, nVidia RIVA-128 Chipsatz

gemessene Leistungsdaten (unter OpenGL):

Transformations-leistung:	ca. 100.000 Punkte/sec (Transformationen ohne Beleuchtung, entspricht 200.000 Dreiecken/sec im günstigsten Fall) ca. 38.000 Punkte/sec bei zwei Lichtquellen (Phong-Modell)
Fülleistung:	je nach Verdeckung etwa 600.000 - 1.800.000 Pixel/sec für Farbklötze („`GL_POINTS`“, $d = 100$), etwa 13.000.000 Pixel/sec bei größtenteils verdeckten, großflächigen Dreiecken (ohne Hardwareunterstützung)

Die Grafikkarte ist in der Lage, das Füllen von texturierten Dreiecken mit relativ hoher Leistung in Hardware zu unterstützen, was hier aber nicht ausgenutzt wurde. Geometrische Transformationen werden von der Hardware nicht unterstützt. Wie Messungen in verschiedenen Modi belegten, wurden in allen hier auftretenden Fällen alle Berechnungen vollständig von Prozessor vorgenommen. Alle Messungen, bei denen nichts anderes angegeben wurde, wurden auf diesem System durchgeführt.

Testsystem 2: Aktuelle PC-Hardware mit Hardwareunterstützung für geometrische Transformationen

Hardware:

Prozessor:	AMD Athlon, 500 Mhz
System:	AMD Irongate Chipsatz, 100 Mhz Speichertakt
Speicher:	256 MB
Grafikkarte:	Elsa Erazor X Grafikkarte mit nVidia GeForce 256 Chipsatz

gemessene Leistungsdaten (unter OpenGL):

Transformations-leistung:	416.000 Punkte/sec („`GL_POINTS`“, mit randomisiertem z-Buffer Algorithmus, ohne Beleuchtung) 1.050.000 Punkte/sec („`GL_TRIANGLE_STRIP`“, ohne Beleuchtung, entspricht max. 2.100.000 Dreiecken/sec) 126.000 Punkte/sec („`GL_POINTS`“, zwei Lichtquellen, offensichtlich *ohne* Hardwareunterstützung, bei Dreiecksrendering trat nur ein minmaler Leistungsabfall bei zwei Lichtquellen auf)
Fülleistung:	ca. 340.000.000 Pixel/sec für Farbklötze („`GL_POINTS`“, $d = 50$)

Die Prozessorleistung dieses Systems ist etwa doppelt so groß wie bei Testsystem 1 und die Grafikkarte bietet wesentlich weitreichendere Unterstützung beim Rendering. Neben dem Füllen von Dreiecken und Farbklötzen wird auch die geometrische Transformation von Punkten in Hardware unterstützt, so daß sich hier gegenüber Testsystem 1 die 5-10fache Leistung ergibt. Die Implementation des randomisierten z-Buffer Verfahrens konnte allerdings nicht mehr stärker auf die Anforderungen der geometriebeschleunigten Hardware angepaßt werden, so daß die volle Leistung der Plattform nicht ausgeschöpft werden konnte. Wie man an den Meßwerten sieht, fiel die Transformationsleistung beim Zeichnen von Punkten in der gegenwärtigen Implementation (gemessen an einer sehr kleinen Szene mit dem randomisierten z-Buffer Verfahren) deutlich gegenüber dem Zeichnen von Dreiecken (gemessen mit punktweise übergebenen Dreiecksstreifen, auch dies ist noch nicht optimal gelöst) ab. Verantwortlich dafür sind vermutlich die relativ aufwendigen Vorbereitungen, insbesondere das Bestimmen einer zufälligen Linearkombination der Dreieckseckpunkte. Hierfür sind ähnlich viele Fließkommaoperationen wie für die eigentliche Transformation des Punktes notwendig, die nicht in Spezialhardware durchgeführt werden können. Eine für solche Hardware optimierte Version sollte daher für kleine Dreiecke möglichst auf diesen Schritt verzichten (sofern diese erkannt werden, siehe Abschnitt 3.3.6.4). Insgesamt sollten die auf diesem System gemessenen absoluten Zeiten mit einer gewissen Vorsicht betrachtet werden, da diese Plattform durch die hohe Leistung der Grafikhardware wesentlich sensibler auf mangelnde Optimierungen im Detail reagiert.

Anhang B: Meßwerte

Wachstum des Aufwandes mit der Szenenkomplexität (5.1.1)

Dreiecke / Instanz:	4	16	64	256	1.024	4.096	16.384	65.536	16.384
Instanzen ($n \times n$):	1	1	1	1	1	1	1	1	2
Ebenen:	0	0	0	0	0	0	0	0	1
Dreiecke insgesamt:	8	32	128	512	2,05E+3	8,19E+3	3,28E+4	1,31E+5	1,31E+5
z-Buffer [sec]:	0,002	0,002	0,003	0,004	0,010	0,033	0,124	0,482	0,480
rnd. z-Buffer [sec]:	0,364	0,358	0,368	0,387	0,396	0,424	0,449	0,501	0,601

65.536	65.536	65.536	65.536	65.536	65.536	65.536	65.536	65.536	65.536	65.536
2	3	4	5	6	7	8	9	10	3	4
1	1	1	1	1	1	1	1	1	2	2
5,24E+5	1,18E+6	2,10E+6	3,28E+6	4,72E+6	6,42E+6	8,39E+6	1,06E+7	1,31E+7	1,06E+7	3,36E+7
1,838	3,539	5,702	8,309	11,423	15,5	20,3	25,7	31,7	25,7	81,2
0,644	0,655	0,668	0,698	0,691	0,687	0,697	0,708	0,700	0,736	0,758

65.536	65.536	65.536	65.536	65.536	65.536	65.536	65.536	65.536	65.536
5	7	10	4	5	7	10	5	7	10
2	2	2	3	3	3	3	4	4	4
8,19E+7	3,15E+8	1,31E+9	5,37E+8	2,05E+9	1,54E+10	1,31E+11	5,12E+10	7,56E+11	1,31E+13
198	762	3,17E+3	1,30E+3	4,96E+3	3,73E+4	3,17E+5	1,24E+5	1,83E+6	3,17E+7
0,745	0,756	0,795	0,846	0,887	0,893	0,927	1,005	1,017	1,037

Tabelle 3: Aufwand bei verschiedener Dreiecksanzahl. Die grau hinterlegten Zeiten für das z-Buffer Rendering sind geschätzt (Anzahl Dreiecke / Transformationsleistung).

Optimierung der Parameter zur Stichprobenauswahl (5.1.2)

Waldszene mit großer Tiefendynamik

ε	0 - Zeit	Boxen	2 - Zeit	Boxen	4 - Zeit	Boxen	6 - Zeit	Boxen	8 - Zeit
1,1	4,406	5192	18157	6,279	47933	8,943	17,309	-	-
1,2	4,169	1406	5581	5,848	17876	7,435	13,435	-	-
1,3	4,269	713	2858	5,679	9926	6,59	9,839	-	-
1,4	4,458	442	1746	5,515	6289	6,038	7,918	-	-
1,5	4,644	321	1245	5,504	4424	5,833	7,001	50674	11,242
1,6	4,908	226	916	5,554	3493	5,811	6,614	39958	9,603
1,7	5,116	187	712	5,7	2694	5,879	6,43	-	-
1,8	5,323	163	648	5,831	2254	5,983	6,382	27341	8,088
1,9	5,588	128	508	5,984	1898	6,135	6,447	23231	7,754
2	5,883	109	440	6,203	1650	6,271	6,453	21148	7,569
2,1	-	-	-	-	-	-	-	19231	7,503
2,2	-	-	-	-	-	-	-	17165	7,458
2,3	-	-	-	-	-	-	-	14414	7,426
2,378	6,682	77	293	6,878	1036	6,971	6,968	-	-
2,4	-	-	-	-	-	-	-	13351	7,452
2,5	-	-	-	-	-	-	-	12634	7,527
2,6	-	-	-	-	-	-	-	11870	7,578
2,7	-	-	-	-	-	-	-	11087	7,661
2,8	-	-	-	-	-	-	-	10541	7,707
2,828	7,679	51	195	7,633	758	7,548	7,472	-	-
3	-	-	-	-	-	-	-	9733	7,774
3,363	8,338	43	153	8,267	600	8,101	8,078	-	-
4	9,937	27	107	9,42	429	9,095	8,761	6128	8,736
8	13,86	15	57	12,849	222	12,215	12,168	3055	11,993
16	16,888	11	41	15,69	145	16,798	16,415	2063	15,995

Tabelle 4: Laufzeiten und Boxenanzahl bei verschiedener Wahl der Unterteilungsparameter, (Waldszene mit großer Tiefendynamik). In den Spalten: gleiche Winkelunterteilungstiefe, in den Zeilen: gleiche Tiefengenauigkeit. Einträge: Renderingzeiten in Sekunden, gefolgt von der Anzahl ausgewählter Boxen.

Waldszene mit geringer Tiefendynamik

ε	0 - Zeit	Boxen	2 - Zeit	Boxen	4 - Zeit	Boxen	6 - Zeit	Boxen	8 - Zeit	Boxen
1,05	2,395	901	2,901	7201	4,59	21504	8,839	49506	26,631	81369
1,06	2,393	901	2,876	6601	3,973	15805	5,694	26264	6,59	29986
1,065	2,41	501	2,757	3901	3,364	8709	2,845	5899	2,409	4041
1,0675	2,717	2	2,736	3501	2,562	4419	2,108	2310	2,013	2147
1,068	-	-	2,732	3501	2,486	3920	2,108	2310	2,02	2147
1,069	-	-	1,834	9	1,761	29	1,731	115	1,722	451
1,07	2,714	2	1,836	9	1,761	29	1,731	115	1,721	451
1,08	-	-	-	-	1,763	29	1,736	115	1,725	452
1,1	2,714	2	1,86	9	1,773	29	1,742	115	1,733	452
1,2	2,712	2	-	-	1,811	29	1,781	116	1,768	464
1,3	-	-	-	-	1,844	29	1,814	117	1,805	472
1,4	2,713	2	2,034	9	1,873	29	1,846	122	1,837	477
1,5	-	-	2,178	-	1,901	31	1,87	124	1,862	483
1,8	-	-	-	9	1,99	33	1,957	125	1,938	496
2	2,715	2	2,223	9	2,039	33	2	126	1,978	502
4	2,716	2	2,424	9	2,259	33	2,207	129	2,172	510
16	2,711	2	2,462	9	2,363	33	2,295	129	2,267	510

Tabelle 5: Laufzeiten und Boxenanzahl bei verschiedener Wahl der Unterteilungsparameter, (Waldszene mit geringer Tiefendynamik). In den Spalten: gleiche Winkelunterteilungstiefe, in den Zeilen: gleiche Tiefengenauigkeit. Einträge: Renderingzeiten in Sekunden, gefolgt von der Anzahl ausgewählter Boxen.

Stadtszene

Winkelunterteilungen	0	2	4	6	8
Renderingzeit [sec]	3,9	4,334	4,439	4,37	4,536
ausgewählte Boxen	2285	3873	3409	6195	8396
opt. Tiefengenauigkeit	1,1	1,2	1,4	1,4	1,4

Tabelle 6: Meßwerte für die Stadtszene

Zusammensetzung des Aufwandes (5.1.3)

	gemessen				berechnet		
Tiefen-genauigkeit	Rendering-zeit	Verteilungs-zeit	Punkte	Boxen	Rekonstr.-zeit	Boxen/sec	Punkte/sec
1,1	8,943	3,971	303.275	47933	4,972	12.071	60.997
1,2	7,435	2,409	320.262	17876	5,026	7.421	63.721
1,3	6,59	1,504	322.475	9926	5,086	6.600	63.404
1,4	6,038	0,883	318.809	6289	5,155	7.122	61.845
1,5	5,833	0,569	319.572	4424	5,264	7.775	60.709
1,6	5,811	0,412	323.851	3493	5,399	8.478	59.984
1,7	5,879	0,296	331.358	2694	5,583	9.101	59.351
1,8	5,983	0,234	338.130	2254	5,749	9.632	58.815
1,9	6,135	0,193	349.729	1898	5,942	9.834	58.857
2	6,271	0,16	358.801	1650	6,111	10.313	58.714
2,378	6,971	0,086	402.008	1036	6,885	12.047	58.389
2,828	7,548	0,055	439.724	758	7,493	13.782	58.685
3,363	8,101	0,032	473.474	600	8,069	18.750	58.678
4	9,095	0,023	534.615	429	9,072	18.652	58.930
4,756	9,855	0,023	584.013	363	9,832	15.783	59.399
5,656	11,491	0,023	686.187	249	11,468	10.826	59.835
6,727	11,607	0,023	695.581	245	11,584	10.652	60.047
8	12,215	0,013	732.778	222	12,202	17.077	60.054
9,513	14,375	0,013	858.934	172	14,362	13.231	59.806
11,31	14,698	0,013	882.446	162	14,685	12.462	60.092
13,45	14,826	0,013	888.560	161	14,813	12.385	59.985
16	16,798	0,009	991.534	145	16,789	16.111	59.059
					Durchschnitt:	11.823	59.971

Tabelle 7: genauere Aufschlüsselung der Rechenzeitanteile für die Waldszene mit großer Tiefendynamik und einer Winkelunterteilungstiefe von 4. Die rechten drei Spalten sind aus den übrigen Meßwerten berechnet und nicht unabhängig gemessen worden.

Kombination mit konventionellem z-Buffer Rendering (5.1.4)

z-Buffer Bereich [Instanzdurchmesser]:	0	0,125	0,25	0,5	1	1,5	2
Laufzeit:	9,375	9,393	9,593	8,076	6,376	5,705	5,316

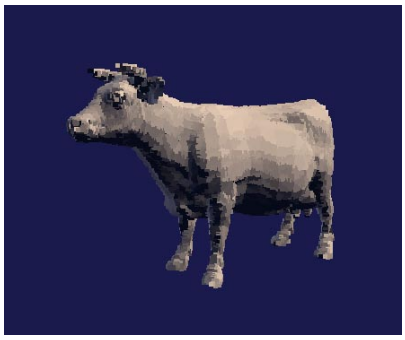
3	3,25	3,5	3,75	4	6	8	16	4,375	7
5,337	5,27	5,67	5,501	6,008	8,891	13,449	15,622	6,345	10,701

12	10	14	5	9
15,715	15,821	15,671	7,012	15,96

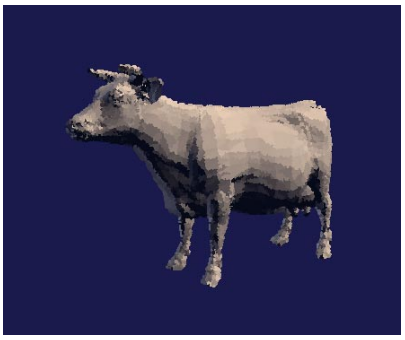
Tabelle 8: Laufzeiten bei einem z-Buffer Rendering des Nahbereiches

Anhang C: Farbbilder

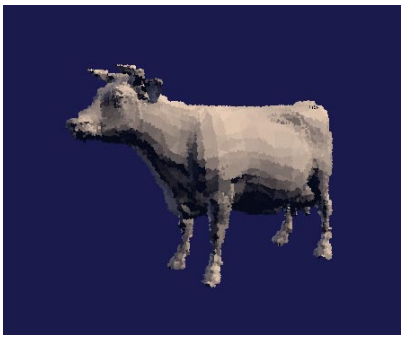
Diskrete Voronoiregionen (3.2.3.4)



(a) Farbklötze, $d = 5$ (0,46 s)

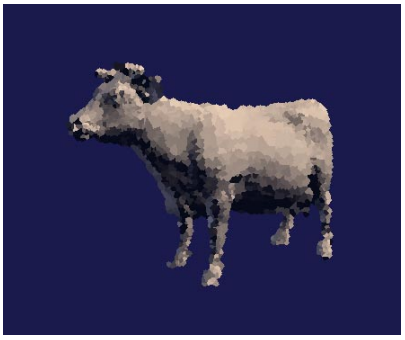


(b) Algorithmus 4, $d = 5$ (2,4 s)

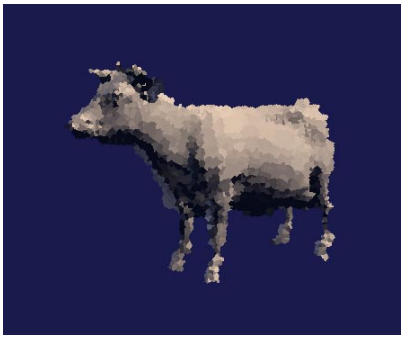


(c) Algorithmus 3, $d = 5$ (1,3 s)

(d) Farbklötze, $d = 10$ (0,36 s)



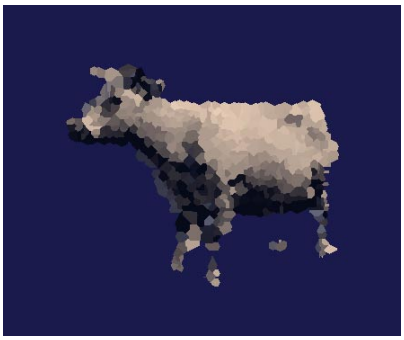
(e) Algorithmus 4, $d = 10$ (4,7 s)



(f) Algorithmus 3, $d = 10$ (1,2 s)

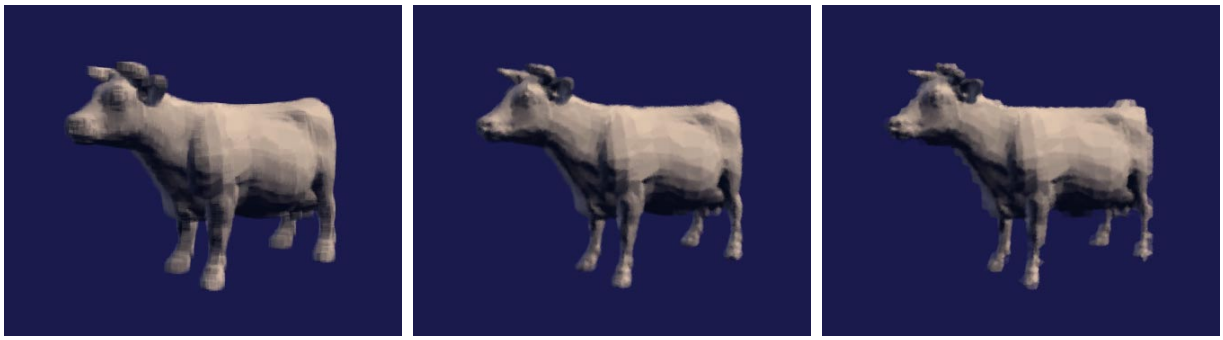
(g) Farbklötze, $d = 20$ (0,32 s)

(h) Algorithmus 4, $d = 20$ (13,4 s)



(i) Algorithmus 3, $d = 20$ (1,1 s)

Farbbild 1: Rekonstruktion mit Farbklötzen und diskreten Voronoiregionen. Linke Spalte: „große Farbklötze“, mittlere und rechte Spalte: diskrete Voronoidiagramme. In der mittleren Spalte wurden die Verdeckungen aus beliebig plazierten Quadraten bestimmt (Algorithmus 4), in der rechten wurde die Verdeckung zellenweise in einem festen Raster bestimmt (Algorithmus 3). In beiden Fällen wurde ein Tiefentoleranzbereich benutzt. Laufzeiten in Klammern, gemessen auf Testsystem 1 (siehe Anhang A)



(j) wie (d), Mittelwert über 20 Bilder (k) wie (e), Mittelwert über 20 Bilder (l) wie (f), Mittelwert über 20 Bilder

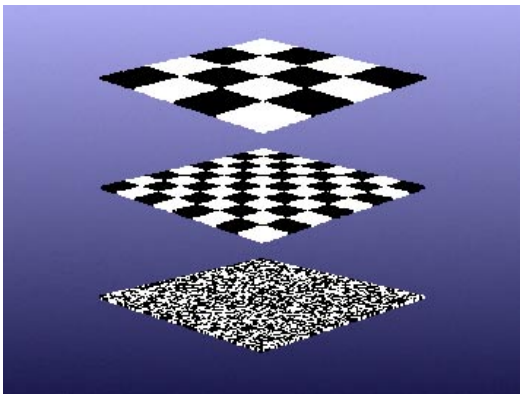
Farbbild 1 (Fortsetzung): Mittelwerte über 20 Bilder für alle drei Verfahren, $d = 10$.

(a) Farbklötze, $d = 10$, 2920 Stichprobenpunkte (0,36 s) (b) Gaußsche Rekonstruktion, Algorithmus 4, $d = 10$, 2920 Stichprobenpunkte (4m 5s) (c) diskrete Voronoidiagramme, Algorithmus 4, $d = 10$, 2920 Stichprobenpunkte (4,7 s)

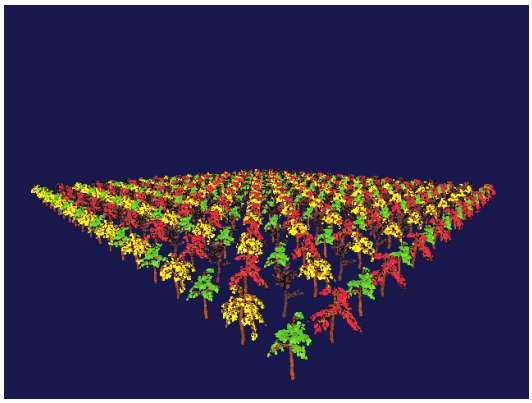
(d) Differenz zwischen (a) und z-Bufferbild (verstärkt) (e) Differenz zwischen (b) und z-Bufferbild (verstärkt) (f) Differenz zwischen (c) und z-Bufferbild (verstärkt)

Farbbild 2: Vergleich der vorgestellten Bildrekonstruktionsverfahren (Testsystem 1)

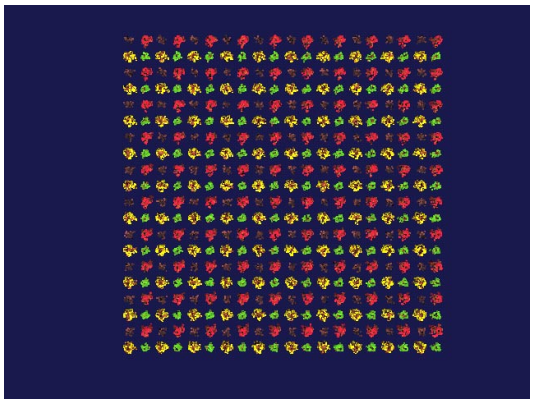
Testszenen zur Bestimmung des Parametereinflusses (5.1)



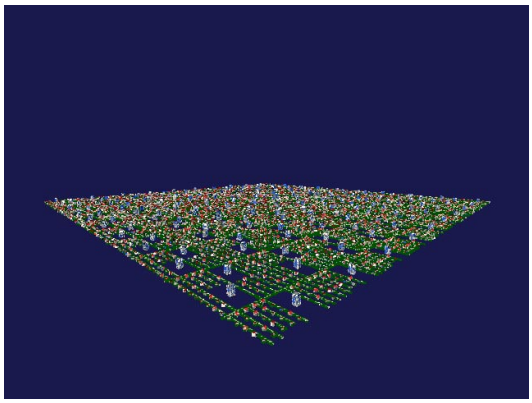
(a) Testszene zur Bestimmung Abhängigkeit der Laufzeit von der Dreiecksanzahl



(b) Waldszene, Ansicht mit großer Tiefendynamik

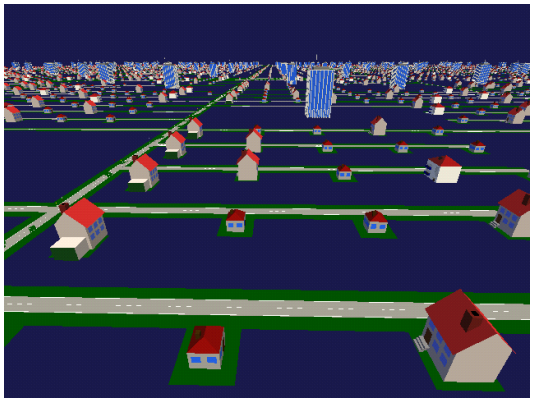


(c) Waldszene, Ansicht mit geringer Tiefendynamik

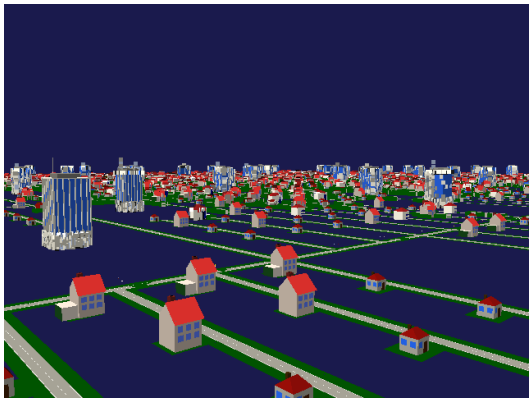


(d) Stadtszene

Farbbild 3: Testszenen zur Bestimmung des Parametereinflusses beim Winkelklassenverfahren



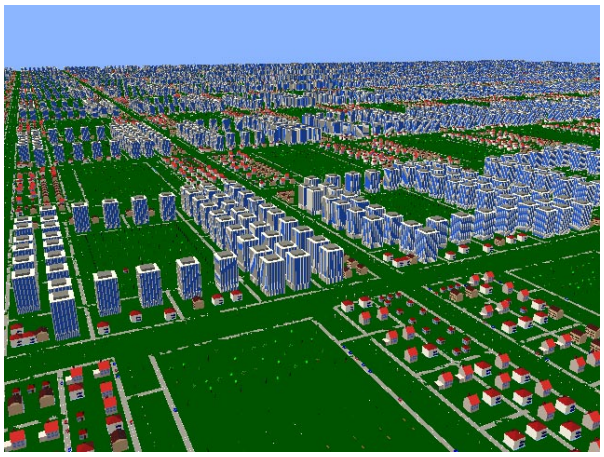
(a) Testszene, z-Buffer Rendering



(b) Kombiniertes Rendering, $d = 5$, 640×480 Punkte (367 msec, Testsystem 2)

Farbbild 4: Stadtszene mit z-Buffer Rendering des Vordergrundes

Komplexe Szenen (5.2)



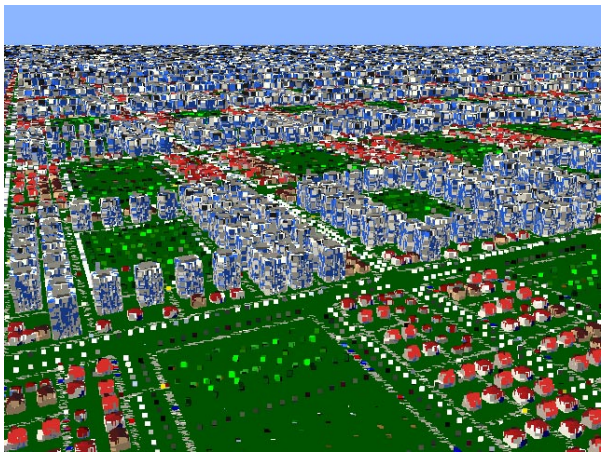
(a) z-Buffer, Teilszene (123 sec, voll ca. 2h)

(b) rnd. z-Buffer, $d = 1$, (124 sec)

(c) rnd. z-Buffer, $d = 2$ (33 sec)

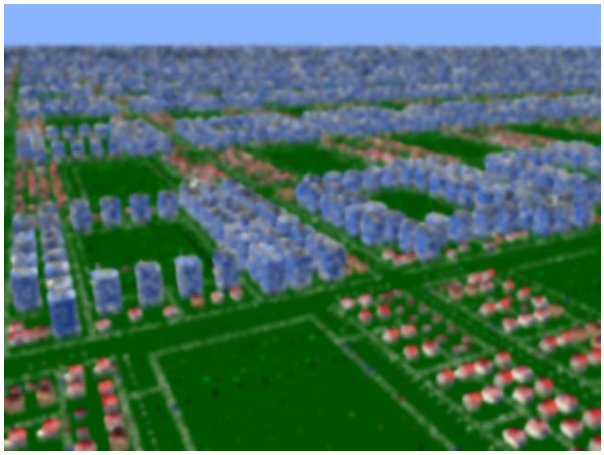
(d) rnd. z-Buffer, $d = 2$, arithmetischer Mittelwert über 10 Bilder (340 sec)

(e) rnd. z-Buffer, $d = 3$ (17 sec)

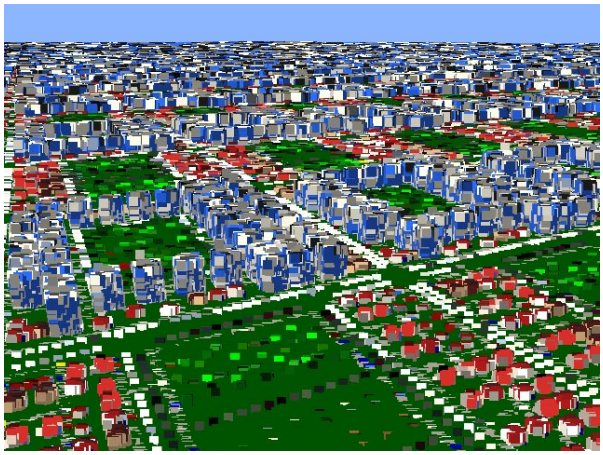


(f) rnd. z-Buffer, $d = 5$ (9,5 sec)

(g) rnd. z-Buffer, $d = 5$, Voronoiregionen (10,4 sec)

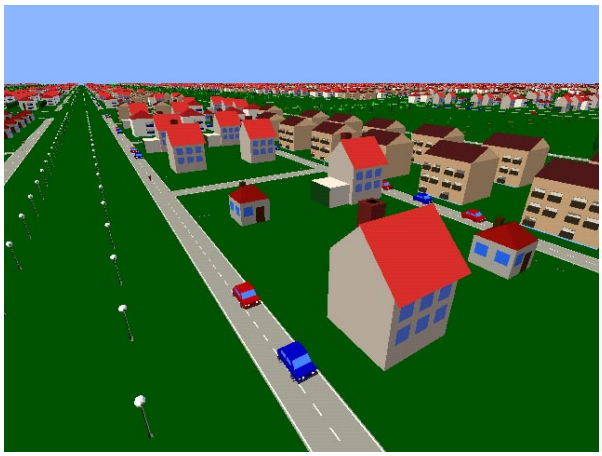


(h) rnd. z-Buffer, $d = 5$, Gaußscher Tiefpaß (27,5 sec)

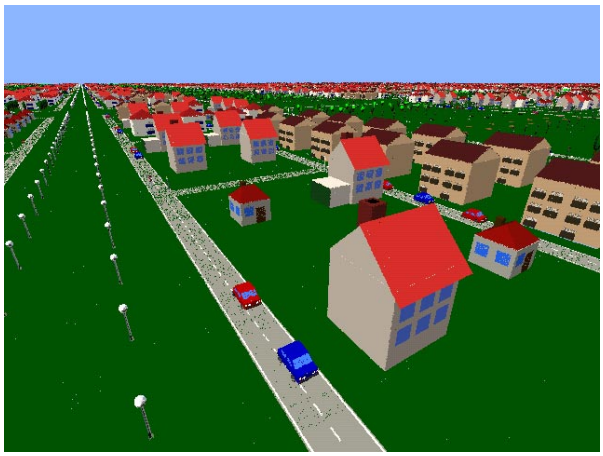


(i) rnd. z-Buffer, $d = 10$ (3,5 sec)

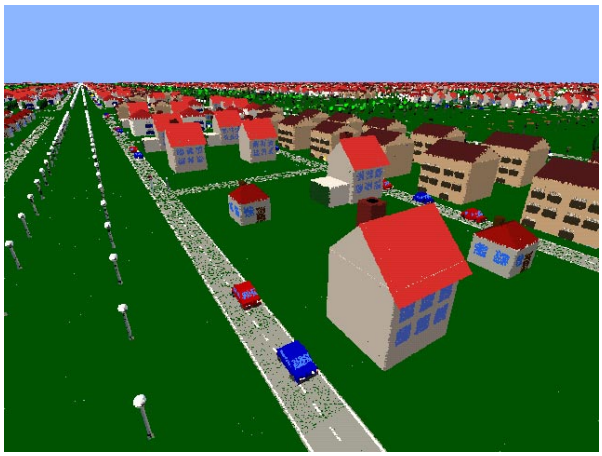
Farbbild 5: große Stadtszene (3.434.780.000 Dreiecke), Übersichtsansicht, alle Zeiten gemessen auf Testsystem 2



(a) z-Buffer (161 sec)



(b) rnd. z-Buffer, $d = 1$ (211 sec)

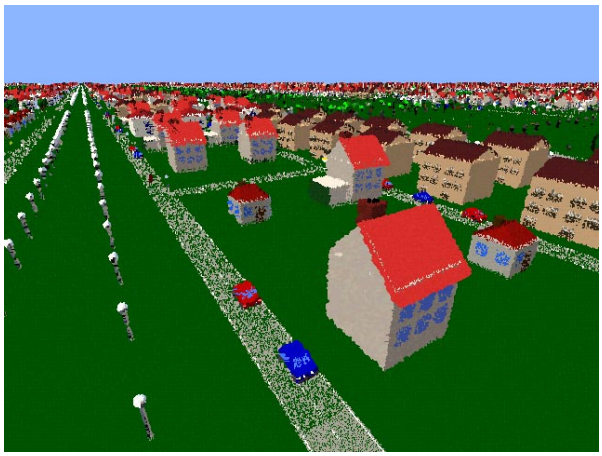


(c) rnd. z-Buffer, $d = 2$ (59 sec)

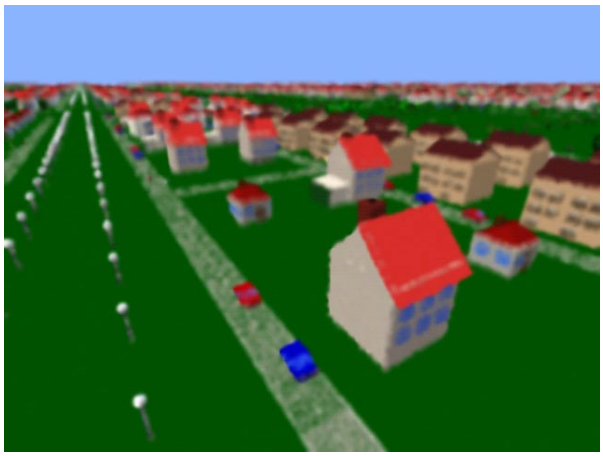
(d) rnd. z-Buffer, $d = 2$, arithmetischer Mittelwert über 10 Bilder (608 sec)

(e) rnd. z-Buffer, $d = 3$ (35 sec)

(f) rnd. z-Buffer, $d = 5$ (24 sec)



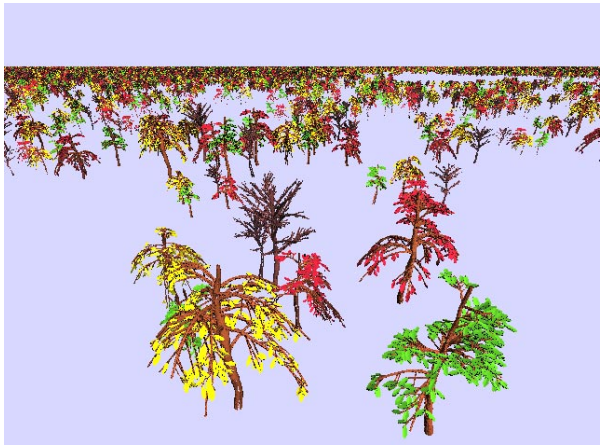
(g) rnd. z-Buffer, $d = 5$, Voronoiregionen (19 sec)



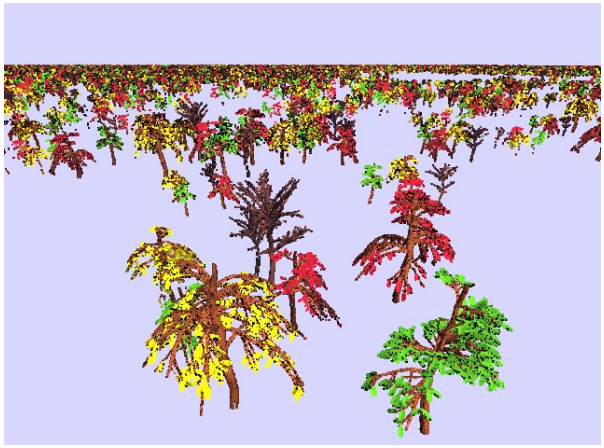
(h) rnd. z-Buffer, $d = 5$, Gaußscher Tiefpaß (62 sec)

(i) rnd. z-Buffer, $d = 10$ (15 sec)

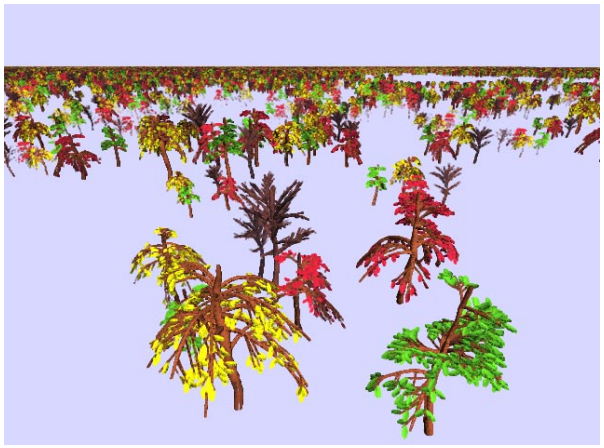
Farbbild 6: kleine Stadtszene (17.731.500 Dreiecke, Vorberechnungszeit 76 sec), niedrigerer Betrachtungsstandpunkt, alle Zeiten gemessen auf Testsystem 1



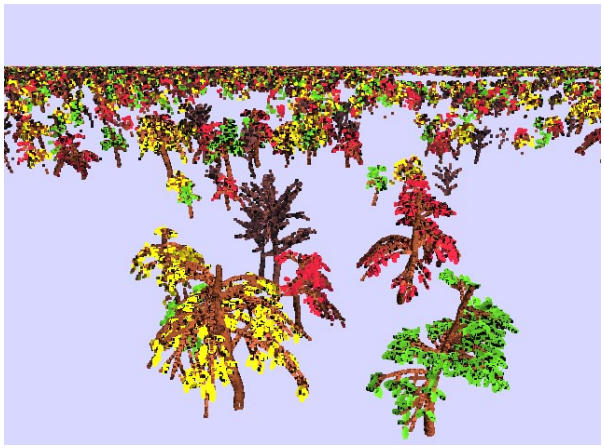
(a) rnd. z-Buffer, $d = 1$ (23 sec)



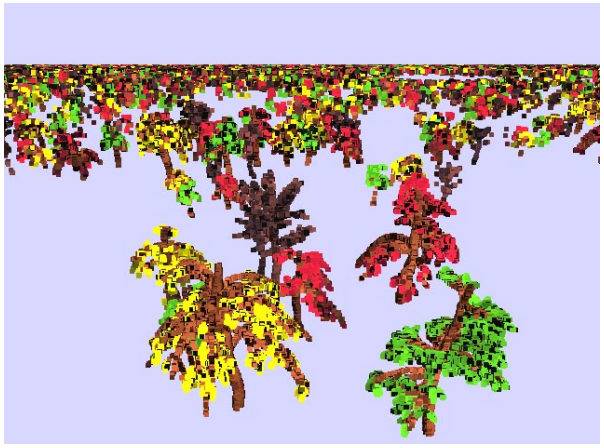
(b) rnd. z-Buffer, $d = 2$ (6,5 sec)



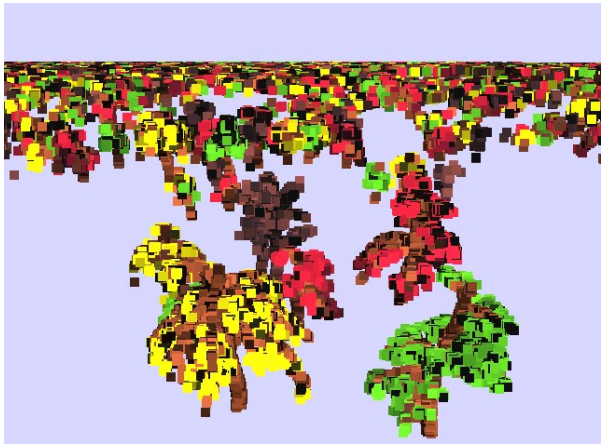
(c) rnd. z-Buffer, $d = 2$, arithmetischer Mittelwert über 10 Bilder (69 sec)



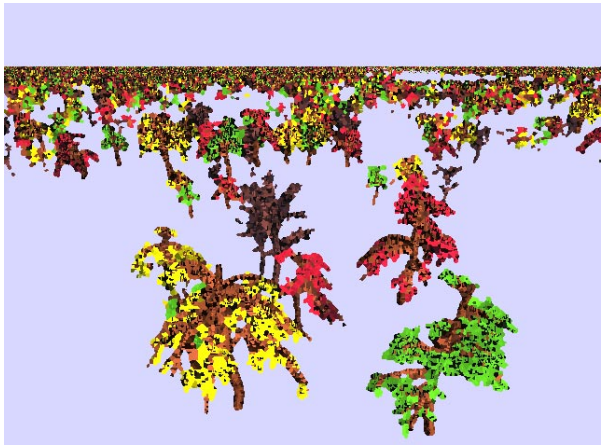
(e) rnd. z-Buffer, $d = 3$ (3,9 sec)



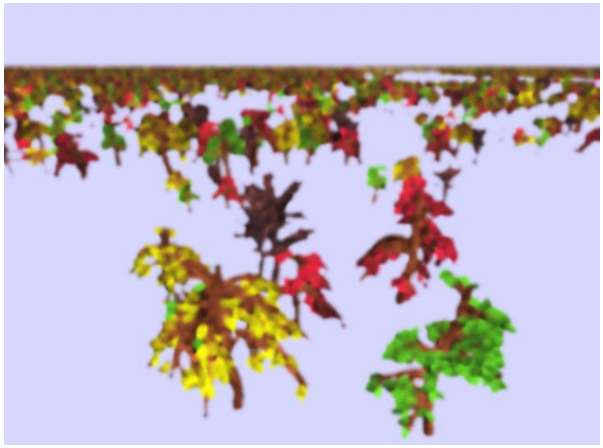
(f) rnd. z-Buffer, $d = 5$ (2,3 sec)



(g) rnd. z-Buffer, $d = 10$ (1,2 sec)

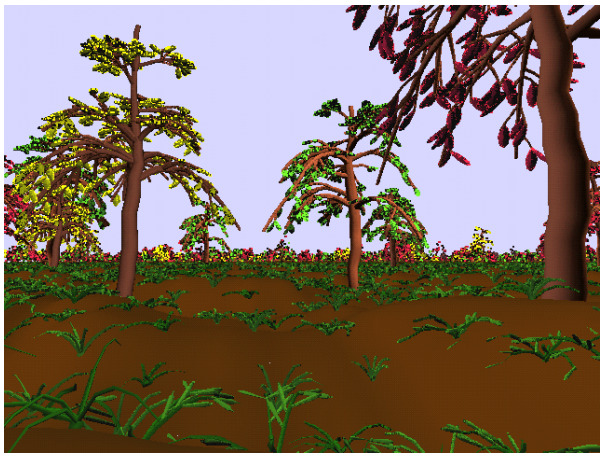


(h) rnd. z-Buffer, $d = 5$, Voronoiregionen (2,9 sec)

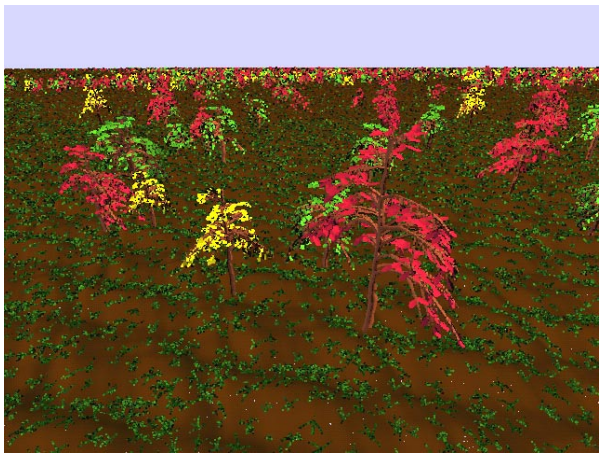


(i) rnd. z-Buffer, $d = 5$, Gaußscher Tiefpaß (27 sec)

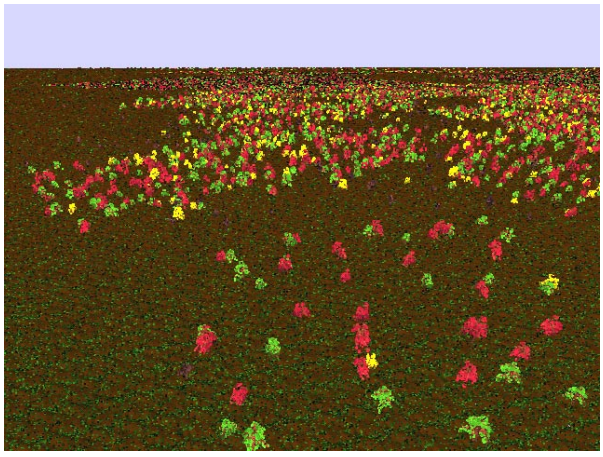
Farbbild 7: Waldszene (93.177.000.000 Dreiecke, Vorberechnungszeit 55 sec), alle Zeiten gemessen auf Testsystem 2, auf ein z-Buffer Rendering wurde verzichtet, da es auch auf Testsystem 2 ca. 25h benötigt hätte.



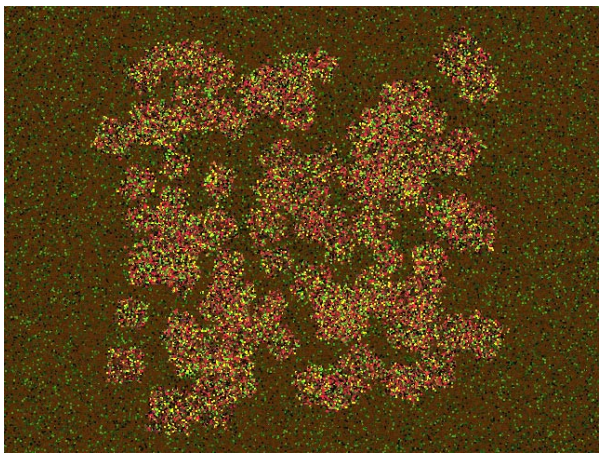
(a) rnd. z-Buffer, $d = 2$ (103 sec)



(b) rnd. z-Buffer, $d = 2$ (43 sec)



(c) rnd. z-Buffer, $d = 2$ (46 sec)



(d) rnd. z-Buffer, $d = 2$ (32 sec)

Farbbild 8: Waldszene mit Boden und Gras (678.905.000.000 Dreiecke, Vorberechnugszeit 50 sec), alle Zeiten gemessen auf Testsystem 1, ein konventionelles z-Buffer Rendering hätte hier jeweils ca. 39 Tage gedauert.

(a) z-Buffer (1,8 sec)

(b) rnd. z-Buffer, $d = 1$ (2,0 sec)

(c) rnd. z-Buffer, $d = 2$ (0,65 sec)

(d) rnd. z-Buffer, $d = 2$, arithmetischer Mittelwert aus 10 Bildern (12,5 sec)

(e) rnd. z-Buffer, $d = 3$ (0,30 sec)

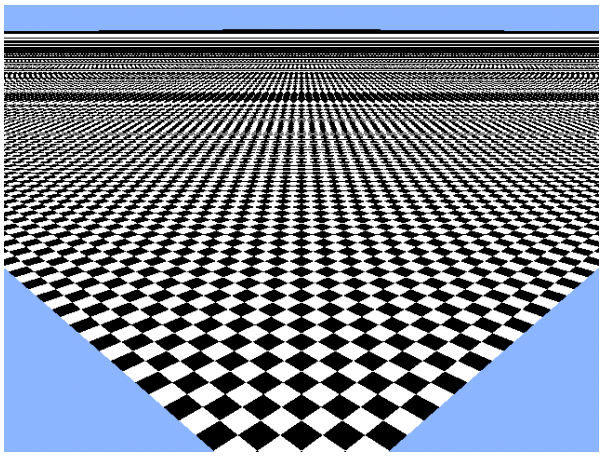
(f) rnd. z-Buffer, $d = 5$ (0,14 sec)

(g) rnd. z-Buffer, $d = 5$, Voronoiregionen (0,85 sec)

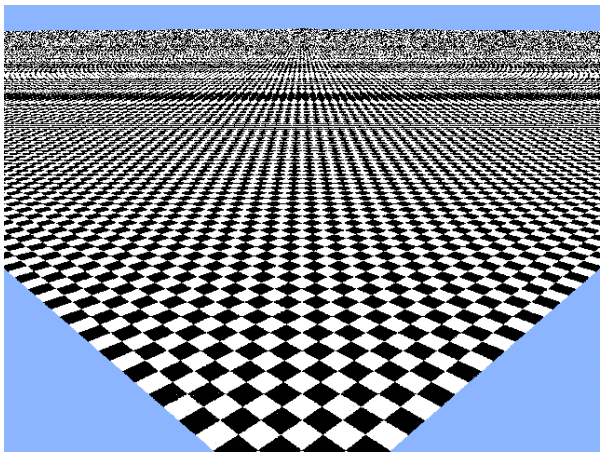
(h) rnd. z-Buffer, $d = 5$, Gaußscher Tiefpaß (27 sec)

(i) rnd. z-Buffer, $d = 10$ (0,073 sec)

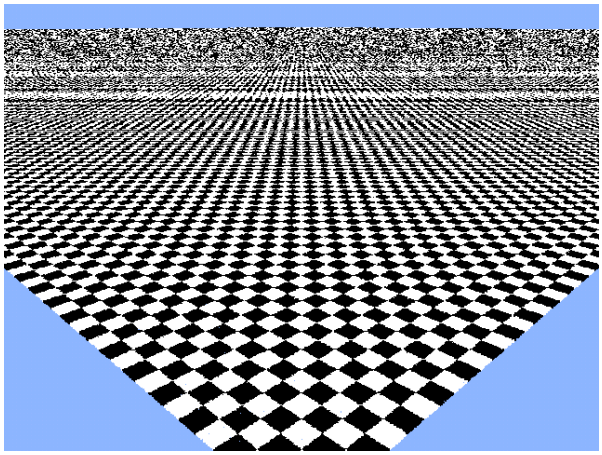
Farbbild 9: „Happy Buddha“ Modell [Garland 99] (1.085.634 Dreiecke, Vorberechnungszeit 230 sec), alle Zeiten gemessen auf Testsystem 2



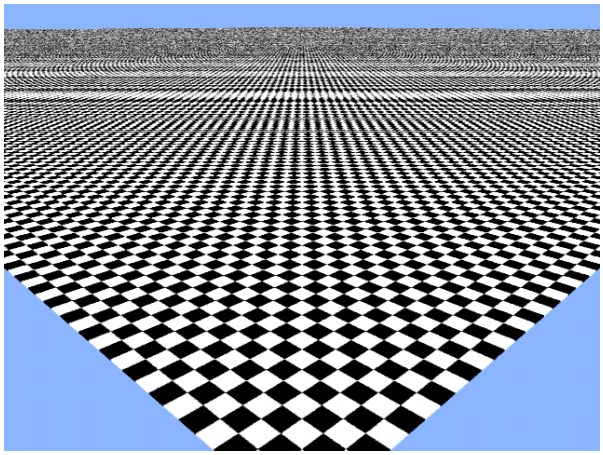
(a) z-Buffer (6,4 sec)



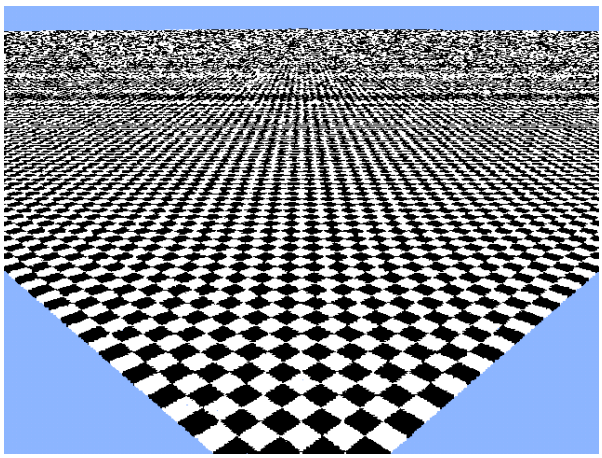
(b) rnd. z-Buffer, $d = 1$ (55 sec)



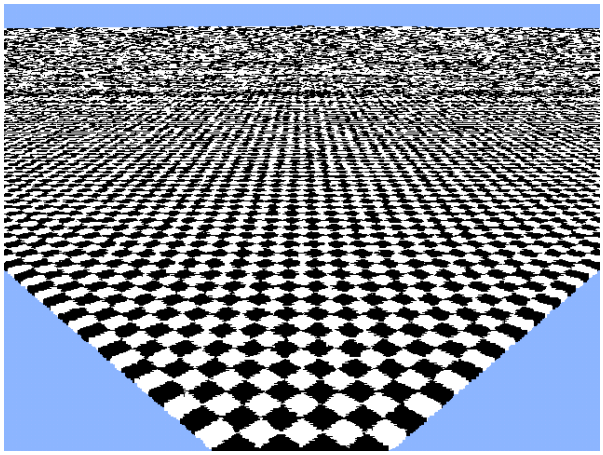
(c) rnd. z-Buffer, $d = 2$ (16 sec)



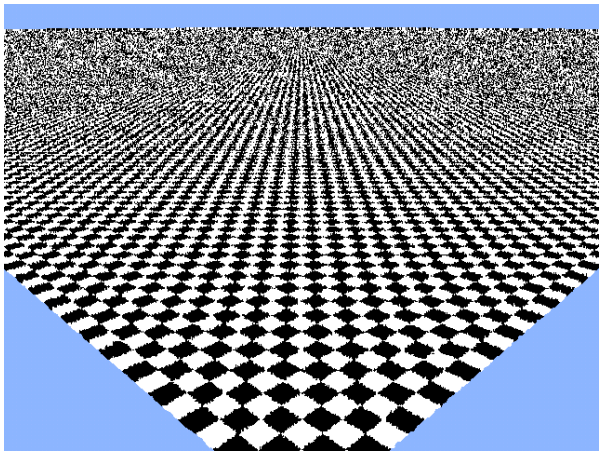
(d) rnd. z-Buffer, $d = 2$, arithmetischer Mittelwert über 10 Bilder (170 sec)



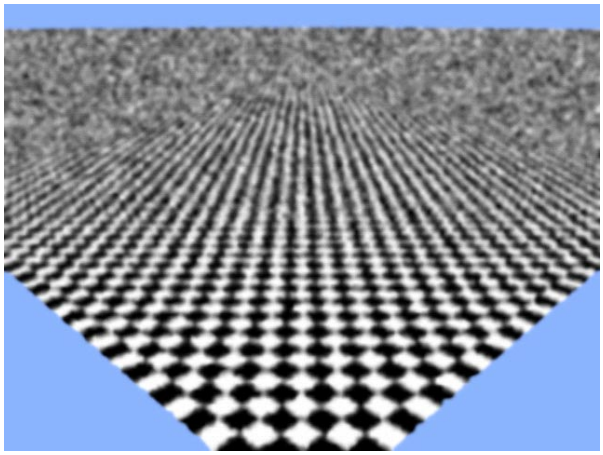
(e) rnd. z-Buffer, $d = 3$ (9,8 sec)



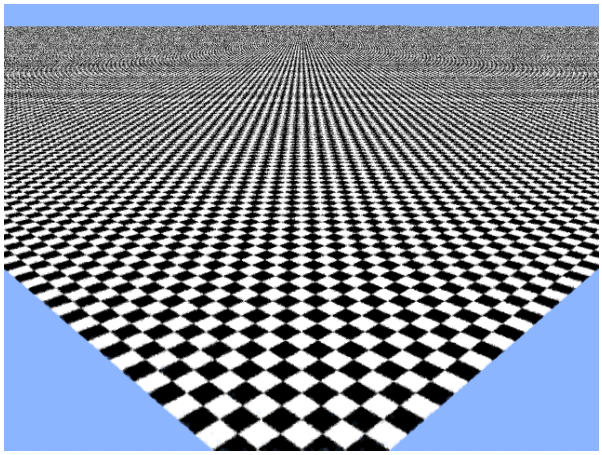
(f) rnd. z-Buffer, $d = 5$ (7,8 sec)



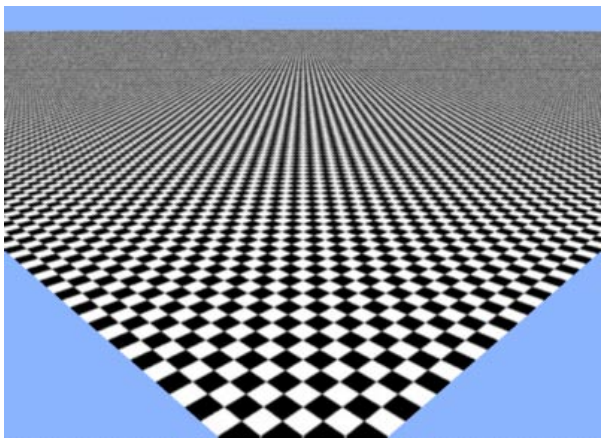
(g) rnd. z-Buffer, $d = 5$, Voronoiregionen (6,7 sec)



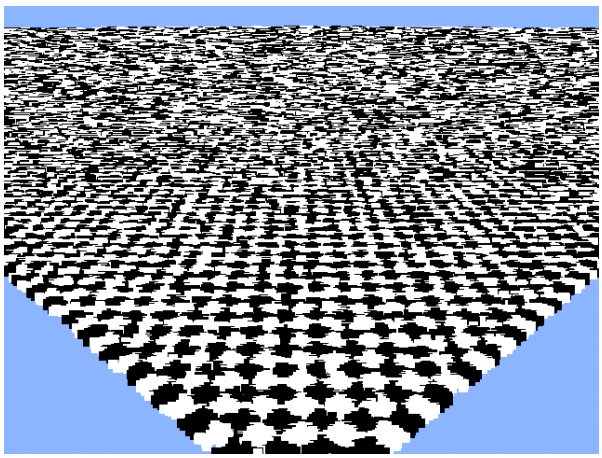
(h) rnd. z-Buffer, $d = 5$, Gaußscher Tiefpaßfilter (43 sec)



(i) wie (g), Mittelwert aus 10 Bildern (142 sec)

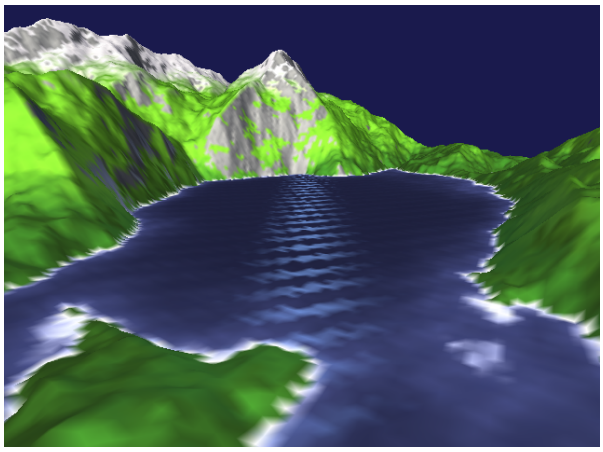


(j) rnd. z-Buffer, etwa 10fach größerer Stichprobenumfang als in (b), Bildrekonstruktion mit einem Gaußschen Tiefpaßfilter, 400×289 Pixel (20 Min.)

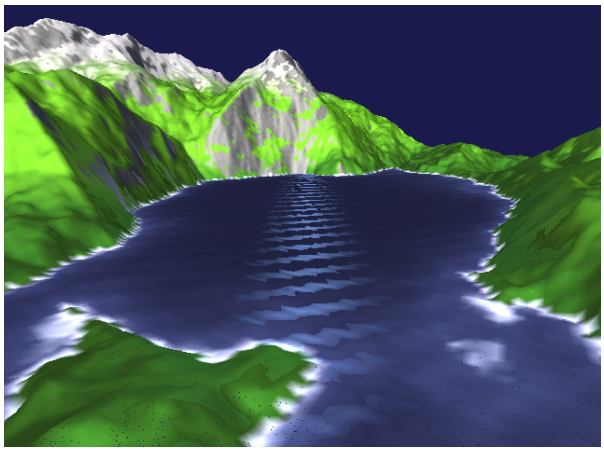


(k) rnd. z-Buffer, $d = 10$ (2,8 sec)

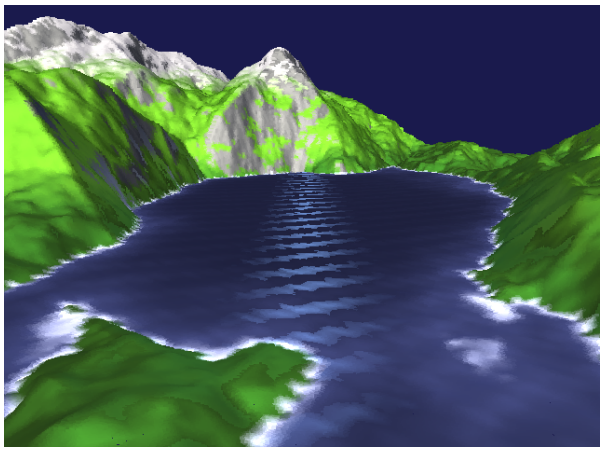
Farbbild 10: Schachbrett (3.276.800 Dreiecke, Vorberechnungszeit 78 sec), alle Zeiten gemessen auf Testsystem 1



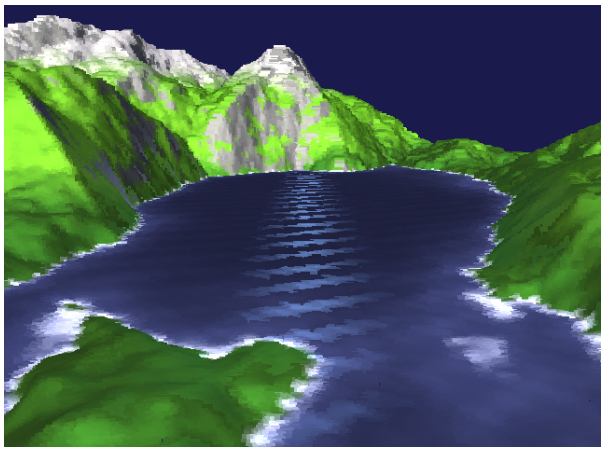
(a) z-Buffer (0,82 sec)



(b) rnd. z-Buffer, $d = 1$ (72 sec)

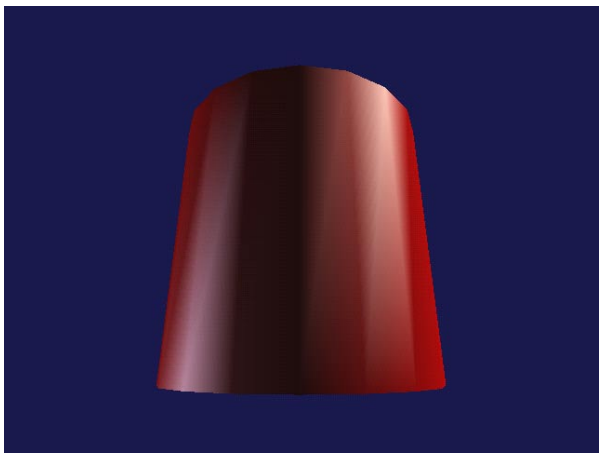


(b) rnd. z-Buffer, $d = 5$ (4,7 sec)

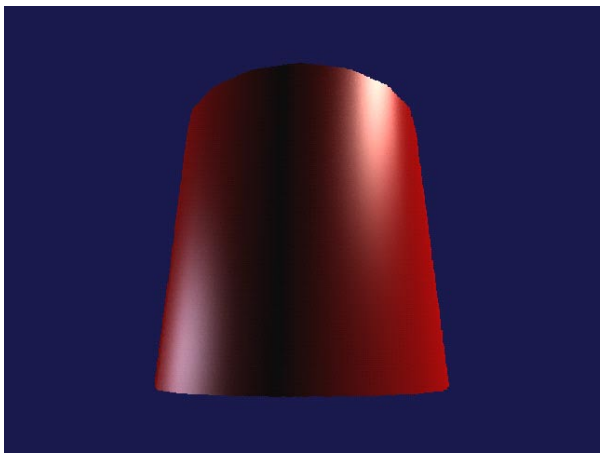


(c) rnd. z-Buffer, $d = 10$ (1,3 sec)

Farbbild 11: Fraktal erzeugte Landschaft (131.072 Dreiecke, Vorberechnungszeit 52 sec) mit dynamisch berechneter Beleuchtung (Phong Modell, 2 Lichtquellen). Alle Zeiten gemessen auf Testsystem 2.



(a) z-Buffer (Gouraudinterpolation)



(b) rnd. z-Buffer, $d = 5$ (Normalvektorinterpolation)

Farbbild 12: roter Zylinder (12 Dreiecke sichtbar) beleuchtet von 2 Lichtquellen (Phong Modell)

Literatur

[Agarwal und Sharir 91] **Agarwal, P.K., Sharir, M.** Applications of a New Space Partitioning Technique. In: *Proc. Workshop on Algorithms and Data Structures,* Lecture Notes in Computer Science 519, 379-391, 1991.

[Appel 68] **Appel, A.** Some Techniques for Shading Mashine Renderings of Solids. In: *Proceedings of the Spring Joint Computer Conference,* 37-45, 1968.

[Arvo und Kirk 91] **Arvo, J., Kirk, D.** A Survey of Ray Tracing Acceleration Techniques. In: Glassner, A. (editor): *An Introduction to Ray Tracing,* Academic Press, 4th printing, pp. 201-262, 1991.

[Bern et al. 90] **Bern, M., Eppstein, D., Gilbert, J.** Provably good mesh generation. In: *Proc. 31st Annu. IEEE Sympos. Found. Compt. Sci.,* pages 231-241, 1990.

[Bertuch 99] **Bertuch, M.** 3D-Poker: 15 Grafikkarten für 3D-Spiele und Anwendungen. In: *c't Magazin für Computertechnik 19/99,* 188-205. Verlag Heinz Heise, 1999.

[Bronstein et al. 97] **Bronstein, I.N., Semendjajew, K.A., Musiol, G., Mühlig, H.** *Taschenbuch der Mathematik.* 3., überarbeitete und erweiterte Auflage der Neubearbeitung, Verlag Harri Deutsch 1997.

[Callahan 95] **Callahan, P.B.** *Dealing with Higher Dimensions: The Well-Separated Pair Decomposition and Its Applications.* Ph.D. thesis, John Hopkins University, 1995.

[Cattmull 74] **Catmull, E.** *A Subdivision Algorithm for Computer Display of Curved Surfaces.* Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, 1974.

[Chamberlain et al. 95] **Chamberlain, B., DeRose, T., Lischinski, D., Salesin, D., Snyder, J.** *Fast Rendering of Complex Environments Using a Spatial Hierarchy.* Technical Report, UW-CSE-95-05-02, University of Washington, 1995.

[Chamberlain et al. 96] **Chamberlain, B., DeRose, T., Lischinski, D., Salesin, D., Snyder, J.** Fast Rendering of Complex Environments Using a Spatial Hierarchy. In: *Proc. Graphics Interface '96*, 132-141, 1996.

[Cook 91] **Cook, R.L.** Stochastik Sampling and Distributed Ray Tracing. In: Glassner, A. (editor): *An Introduction to Ray Tracing.* Academic Press, 4th printing, pp. 161-199, 1991.

[de Berg et al. 94] **de Berg, M., Halperin, D., Overmars, M., Snoeyink, J., van Kreveld, M.** Efficient Ray Shooting and Hidden Surface Removal. In: *Algorithmica, 12*, 30-53, 1994.

[de Berg et al. 97] **de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.** *Computational Geometry - Algorithms and Applications,* Springer-Verlag 1997.

[Durand et al. 97] **Durand, F., Drettakis, G., Puech, C.** The Visibility Skeleton: A Powerful And Efficient Multi-Purpose Global Visibility Tool In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '97 Proceedings),* 1997.

[Ebbesmeyer 98] **Ebbesmeyer, P.** Textured Virtual Walls - Achieving Interactive Frame Rates During Walkthroughs of Complex Indoor Environments. In: *IEEE Virtual Reality Annual International Symposium, VRAIS '98*, 1998

[Eck et al. 95] **Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., Stuetzle, W.** Multiresolution Analysis of Arbitrary Meshes. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '95 Proceedings),* 173-182, 1995.

[Erikson 96] **Erikson, C.** *Polygonal Simplification: An Overview*. Technical Report, TR96-016, Department of Computer Science, UNC-Chapel Hill, Chapel Hill, NC, 1996.

[Foley et al. 96] **Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F.** *Computer Graphics: principals and practice, second edition in C*. Addison-Wesley, 1996.

[Garland 99] **Garland, M.** *Quadric-Based Polygonal Surface Simplification*, Ph.D. thesis, Technical Report CMU-CS-99-105, Carnegie Mellon University, 1999.
`Http://www.cs.cmu.edu/~garland/CMU/quadrics/quadrics.html`

[Glassner 95a] **Glassner, A. S.** *Principles of Digital Image Synthesis, Volume One.* Morgen Kaufmann Publishers, 1995.

[Glassner 95b] **Glassner, A. S.** *Principles of Digital Image Synthesis, Volume Two.* Morgen Kaufmann Publishers, 1995.

[Green et al. 93] **Green, N., Kass, M., Miller, G.** Hierarchical Z-Buffer Visibility. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '93 Proceedings)*, 231-238, 1993.

[He et al. 95] **He, T., Hong, L., Kaufman, A., Varshney, A., Wang, S.** Voxel Based Object Simplification. In: *Visualization '95 Conference Proceedings,* 296-303, 1995.

[Heckbert und Garland 94] **Heckbert, P. S., Garland, M.** Multiresolution Modelling for Fast Rendering In: *Proceedings of Graphics Interface '94,* 43-50, 1994.

[Heckbert und Garland 97] **Heckbert, P. S., Garland, M.** Survey of Polygonal Surface Simplification Algorithms, *SIGGRAPH 97 course notes*, 1997.

[Hoppe 96] **Hoppe, H.** Progressive meshes. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '96 Proceedings),* 99-108, 1996.

[Hoppe 97] **Hoppe, H.** View-dependent refinement of progressive meshes. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '97 Proceedings)*, 189-198, 1997.

[Kay 79] **Kay, D. S.** *Transparency, Refraction and Ray Tracing for Computer Synthesized Images.* M.S. Thesis, Program of Computer Graphics, Cornell University, Ithaca, NY, 1979.

[Klein und Straßer 97] **Klein, R., Straßer, W.** Generation of multiresolution models from cad data for realtime rendering. In: R. Klein, W. Straßer and R.Rau, editors, *Geometric Modeling: Theory and Practice, The State of the Art.* Springer-Verlag, pp. 324-334, 1997.

[Lorensen und Cline 87] **Lorensen, W.E., Cline, H.E.** Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '87 Proceedings),* 21 (4), 163-169, 1987.

[Lounsbery et al. 94] **Lounsbery, M. DeRose, T. Warren, J.** *Multiresolution Analysis for Surfaces of Arbitrary Topological Type.* Technical Report 93-10-05b, Department of Computer Science and Egineering, University of Washington, 1994.

[Lukovszki 98] **Lukovszki, T.** persönliches Gespräch.

[Maciel und Shirley 95] **Maciel, P.W.C., Shirley, P.:** Visual Navigation of Large Environments Using Textured Clusters. In: *Proceedings of the 1995 ACM SIGGRAPH Symposium on Interactive 3D Graphics*, 1995.

[Manber 89] **Manber, U.:** *Introduction to Algorithms, A creative Approach.* Addison-Wesley 1989.

[Motwani und Raghavan 95] **Motwani, R., Raghavan, P.** *Randomized Algorithms.* Cambridge University Press, 1995.

[O'Rourke 87] **O'Rourke, J.** *Art Gallery Theorems and Algorithms.* Oxford University Press, 1987.

[Pfanzagl 91] **Pfanzagl, J.** *Elementare Wahrscheinlichkeitsrechnung.* 2., überarbeitete und erweiterte Auflage, de Gruyter 1991.

[Popovic und Hoppe 97] **Popovic, J., Hoppe, H.** Progressive simplicial complexes. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '97 Proceedings)*, 217-224, 1997.

[Puppo und Scopigno 97] **Puppo, E., Scopigno, R.** Simplification, LOD and Multiresolution Principals and Applications. In: *EUROGRAPHICS '97 Tutorial Notes, 1997.*

[Rossignac und Borrel 93] **Rossignac, J., Borrel, P.** Multi-resolution 3D approximations for rendering complex scenes. In: B. Falcidieno and T.Kunii, editors, *Modeling in Computer Graphics: Methods and Applications,* pages 455-465, Springer-Verlag 1993.

[Samet 90] **Samet, H.** *The Design and Analysis of Spatial Data Structures.* Addison-Wesley 1990.

[Schröder et al. 92] **Schroeder, W. J., Zarge, J. A., Lorensen, W.E.** Decimation of Triangle Meshes. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '92 Proceedings), 26 (2),* 65-70, 1992

[Shade et al. 96] **Shade, J., Lischinski, D., Salesin, D. H., DeRose, T., Snyder, J.** *Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments.* In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '96 Proceedings),* 1996.

[Sillion et al. 97] **Sillion, F., Drettakis, G., Bodelet, B.** Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery. In: *EUROGRAPHICS '97,* 207-218, 1997.

[Stollnitz et al. 96] **Stollnitz, E. J., DeRose, T. D., Salesin, D. H.** *Wavelets for Computer Graphics.* Morgen Kaufmann Publishers, 1996.

[Szirmay-Kalos und Márton 98] **Szirmay-Kalos, L., Márton, G.** Worst-Case Versus Average Case Complexity of Ray-Shooting, In: *Journal of Computing.*, 1998

[Teller und Séquin 91] **Teller, S.J., Séquin, C.H.** Visibility Preprocessing For Interactive Walkthroughs. In: *Computer Graphics Proc., Annual Conference Series (SIGGRAPH '91 Proceedings), 25 (4)*, 61-69, 1991.

[Torborg und Kajiya 96] **Torborg, J., Kajiya, J.** Talisman: Commodity realtime 3D graphics for the PC. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '96 Proceedings),* 353-363, 1996.

[Turk 92] **Turk, G.** Re-Tiling Polygonal Surfaces. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '92 Proceedings), 26 (2),* 55-64, 1992

[Varshney 94] **Varshney, A.** *Hierarchical Geometric Approximations.* Ph.D. thesis, Department of Computer Science, University of North Carolina at Chapel Hill, NC-27599-3175, TR-050-1994,1994

[Weber und Penn 95] **Weber, J., Penn, J.** Creation and Rendering of Realistic Trees. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '95 Proceedings),* 119-128, 1995.

[Wernecke 94] **Wernecke, J.** The Inventor Mentor: Programming Object-Oriented 3d Graphics With Open Inventor, Release 2. Addison Wesley 1994.

[Whitted 80] **Whitted, T.** „An Improved Illumination Model for Shaded Display". In: *Communications of the ACM,* 23(6), 343-349, 1980.

[Wimmer et al. 98] **Wimmer, M., Giegl, M., Schmalstieg, D.** *Fast Walkthroughs with Image Caches and Ray Casting.* Technische Universität Wien, Technical Report TR-186-2-98-30, 1998.

[Woo et al. 97] **Woo, M., Neider, J., Davis, T.** *OpenGL Programming Guide: the official guide to learning OpenGL version 1.1, second edition.* Addison-Wesley 1997.

[Zerbe 99] **Zerbe, K.** Virtuelle Welten: 3D-Spiele-Programmierern über die Schulter geschaut. In: *c't Magazin für Computertechnik 12/99,* 194-203. Verlag Heinz Heise, 1999.

[Zhang et al. 97] **Zhang, H., Manocha, D., Hudson, T., Hoff, K.** Visibility Culling using Hierarchical Occlusion Maps. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH '97 Proceedings), 31(3A), 77-88,* 1997.

Erklärung

Ich versichere, daß ich diese Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Paderborn, den 29.03.2000

Danksagung

Ich möchte an dieser Stelle allen Personen danken, die mich bei der Erstellung der Arbeit unterstützt haben. Insbesondere danke ich Herrn Prof. Dr. Friedhelm Meyer auf der Heide, der mir stets mit Ratschlägen und Anregungen zur Seite stand, und Matthias Fischer, der mit seinem besonderen Engagement bei der Betreuung der Arbeit viel zu deren Gelingen beigetragen hat. Ich möchte ebenfalls Tamás Lukovszki und Christian Sohler für ihre Unterstützung und ihre Anregungen in Diskussionen danken.