Effiziente Lösungsverfahren für Sichtbarkeitsprobleme in der realitätsnahen Bildsynthese

Dissertation
zur Erlangung des Grades des
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik
von

Dipl.-Inform. André Hinkenjann

Lehrstuhl VII - Graphische Systeme
Fachbereich Informatik
Universität Dortmund

Dortmund
1999

Tag der mündlichen Prüfung: 31.03.2000

Dekan: Prof. Dr. Bernd Reusch

Gutachter: Prof. Dr. Heinrich Müller, Prof. Dr. Dieter W. Fellner, PD Dr. Paul Fischer

Vorwort

Die vorliegende Arbeit ist das Resultat von viereinhalb Jahren Auseinandersetzung mit der Sichtbarkeitsproblematik. Ganz sicher wäre sie nicht so schnell erstellt worden ohne die Unterstützung, die ich erhalten habe. Ich danke dem Betreuer dieser Arbeit, Herrn Prof. Dr. Heinrich Müller für die sehr gute Zusammenarbeit. Ich danke den weiteren Gutachtern, Herrn Prof. Dieter W. Fellner und Herrn PD Dr. Paul Fischer. Meinen Diplomanden Markus Kukuk, Christoph Reuling und Norbert Haiduk möchte ich auf diesem Wege ebenfalls meinen Dank aussprechen. Ebenso gilt mein Dank Frank Weller und meiner Frau Christiane für das Korrekturlesen der Arbeit und meinen Eltern, die mich auf das Informatikstudium aufmerksam gemacht haben.

Zusammenfassung

Eine prinzipielle Schwierigkeit bei der Bildsynthese ist die Sichtbarkeitsbestimmung. Sie nimmt einen großen Anteil des Rechenaufwands praktisch aller heutigen Verfahren ein. In dieser Dissertation werden Sichtbarkeitsprobleme in anderen Ausprägungen als der üblichen Sichtbarkeit von einem Punkt aus betrachtet, nämlich wie sie etwa beim hierarchischen Radiosity-Verfahren auftreten. Als Hauptproblem wird die Berechnung der Sichtbarkeitsmenge zweier Elemente behandelt. Dieses besteht darin, eine gegebene Szene von Elementen so in eine Datenstruktur vorzuverarbeiten, daß die Sichtbarkeit zwischen zwei disjunkten Elementen **a** und **b** (desselben Typs und disjunkt vom Rest der Szene) effizient bestimmt werden kann.

Dieses Problem wird zunächst im Hinblick auf eine worst-case-effiziente, exakte Lösung betrachtet. Auf der Grundlage von Datenstrukturen zur Halbraumanfrage wird eine Lösung präsentiert, die eine Anfrage asymptotisch schneller als durch systematisches Testen, d.h. in sublinearer Zeit in der Szenengröße, beantwortet und dabei mit nahezu linearem Speicherplatzbedarf auskommt.

Begründet durch die bekannte Beobachtung, daß sich Sichtbarkeitsprobleme trotz ihres unangenehmen Verhaltens im schlechtesten Fall heuristisch gut lösen lassen, konzentriert sich die Arbeit dann auf diesen Aspekt. Als ein Lösungsvorschlag werden zunächst hierarchische Verdeckungsbäume eingeführt. Sie erlauben einen Trade-off zwischen Speicherbedarf und Rechenzeit sowie Exaktheit der Lösung.

Als ganz anderer Zugang wird dann eine Lösung durch „Abtastung" entwickelt, die sich auch auf andere Anfrageprobleme übertragen läßt. In einem Vorverarbeitungsschritt werden Abtastanfragen zusammen mit ihren Antworten in einer Antwortdatenbank gespeichert. Aus diesen Abtastanfragen werden zur Laufzeit für beliebige Anfragen Antworten (re)konstruiert. Es werden worst-case-Schranken für Laufzeit und Speicherbedarf der Datenstruktur für die Abtastanfragen in Abhängigkeit von einem vorgegebenen Fehler hergeleitet. Dabei werden die zwangsläufig auftretenden Fragen der Stichprobenanfragen und der Komprimierung der Stichprobenantworten behandelt.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Gegenstand der realitätsnahen Bildsynthese ist die Berechnung realistisch wirkender Rasterbilder aus einer dreidimensionalen Szenenbeschreibung nach Modellen, die aus der Optik abgeleitet sind. Die realitätsnahe Bildsynthese stellt heute ein wichtiges Teilgebiet der graphischen Datenverarbeitung dar, die Methoden werden aber auch in der Beleuchtungsplanung eingesetzt. Ein Ziel aktueller Anstrengungen ist das Lösen der allgemeinen Bildsynthesegleichung von Kajiya [Kaj86]. Spezialfälle wie die Simulation von Spiegelungen, Brechungen und Schlagschatten durch das Strahlverfolgungsverfahren (ray tracing) [Whi79] oder die diffuse Reflexion durch das Strahlungsverfahren (radiosity) [GTGB84, NN85] sind allgemein bekannt und gut untersucht. In jüngerer Zeit wurde mit dem hierarchischen Radiosity-Verfahren ein Durchbruch erzielt [HSA91]. Dieses Verfahren ist auch auf die volle Bildsynthesegleichung übertragbar. Zum Themenbereich der realitätsnahen Bildsynthese gibt es umfassende Darstellungen, die den Stand der Forschung weitgehend wiedergeben [CW93, Gla95].

Eine prinzipielle Schwierigkeit bei der Bildsynthese ist die Sichtbarkeitsbestimmung. Sie nimmt einen großen Anteil des Rechenaufwands praktisch aller heutigen Verfahren ein. Je nach Verfahren tritt sie in unterschiedlichen Formulierungen auf, beispielsweise in Form des Hidden-Line-Eliminationsproblems oder des Strahlanfrageproblems. Obwohl beide Versionen schon sehr umfassend untersucht wurden (vgl. z.B. [Mül88, deB93]), sind sie nach wie vor Gegenstand regen Forschungsinteresses.

In dieser Dissertation werden Sichtbarkeitsprobleme in anderen Ausprägungen betrachtet, nämlich wie sie etwa beim hierarchischen Radiosity-Verfahren [HSA91] auftreten. Die Grundform dieser Sichtbarkeitsprobleme ist:

Sichtbarkeit zwischen zwei Elementen

Eingabe: Eine Szene, bestehend aus disjunkten konvexen Elementen.

Ausgabe: Die Sichtbarkeitsregion zweier Elemente **a** und **b**, die entweder Szenenelemente sind oder vom selben Typ wie die Szenenelemente sind, diese aber nicht schneiden.

Elemente sind dabei Flächen oder Körper im Raum, z.B. Dreiecke oder Kugeln. Die *Sichtbarkeitsregion* kann dabei als die Menge aller Strecken **s** definiert werden, deren einer Endpunkt auf der Oberfläche von **a** und deren anderer Endpunkt auf der Oberfläche von **b** liegt. Die Strecke **s** darf in ihrem Inneren keinen Punkt mit einem Element der Szene gemeinsam haben. Die Konvexität der Elemente hat zur Folge, daß die Sichtbarkeitsregion eines Elements mit sich selbst leer ist.

Ein etwas umfangreicheres, verwandtes Sichtbarkeitsproblem ist:

Sichtbare Umgebung eines Elements

Eingabe: Eine Szene, bestehend aus disjunkten konvexen Elementen.

Ausgabe: Die Sichtbarkeitsregionen eines Anfrageelementes **a** mit allen anderen Szenenelementen, die von **a** aus sichtbar sind. **a** ist entweder ein Szenenelement oder vom selben Typ wie die Szenenelemente, schneidet diese aber nicht.

Ein Szenenelement **b** heißt *sichtbar* von **a**, wenn die Sichtbarkeitsregion von **a** und **b** nicht leer ist, d.h. , wenn mindestens eine Strecke zwischen den Oberflächen der beiden Elemente kein Szenenelement schneidet. Die Forderung der Konvexität trägt zur Vereinfachung bei. Für die große Klasse von Szenen, die durch ebene Polygone oder konvexe Polytope beschrieben werden, bedeutet sie keine Einschränkung.

Sichtbarkeit in dieser allgemeinen Weise wurde in der Literatur bisher wenig betrachtet. Ein Konzept in diesem Zusammenhang ist der *Ansichtsgraph* (engl. *Aspect Graph*) [GCS88]. Der Ansichtsgraph eines Polyeders zerlegt die Menge der Punkte, von denen aus das Polyeder sichtbar ist, in Gebiete gleicher Sichtbarkeit. Ein Gebiet gleicher Sichtbarkeit ist dadurch festgelegt, daß von den darin liegenden Augenpunkten dieselbe Menge von Facetten des Polyeders sichtbar ist. Für ein Polyeder der Größe $O(n)$, dessen Partition der Augenpunkte die Größe m hat, kann der Ansichtsgraph mit $O(n^4 \log n + m \log m)$ Rechenzeitaufwand berechnet werden [GCS88]. Die Anzahl m kann im schlechtesten Fall recht hoch sein.

Ein weiteres bekanntes Konzept ist der *Sichtbarkeitskomplex* (*Visibility Complex*) [PV93]. Der Sichtbarkeitskomplex beschreibt die Sichtbarkeit zwischen zwei Elementen einer Szene von Strecken in der Ebene. Er repräsentiert Strecken, die zwei Punkte auf zwei Elementen verbinden und kein anderes Element der Szene schneiden. Der Sichtbarkeitskomplex ergibt sich als die Menge der Äquivalenzklassen dieser Sichtstrecken, wobei zwei Sichtstrecken äquivalent genannt werden, wenn man durch kontinuierliches Verschieben der Endpunkte der einen Sichtstrecke auf den beiden Szenenstrecken zu der anderen Sichtstrecke gelangen kann, ohne daß ein Szenenelement geschnitten wird. Der Sichtbarkeitskomlpex wurde auf die Radiosity-Berechnung für zweidimensionale Szenen [ORDP96] angewendet.

Der Ansichtsgraph und der Sichtbarkeitskomplex sind mit viel Speicherbedarf verbunden. Bei der praktischen Bildberechnung gibt man sich daher häufig mit der näherungsweisen Sichtbarkeitsberechnung zufrieden. Eine gängige Vorgehensweise ist etwa, zwischen zwei Szenenelementen eine Reihe von Sichtstrecken auf Schnitt mit anderen Szenenelementen zu testen. Das Verhältnis von nicht schneidenden zu schneidenden Strecken gibt Aufschluß über den Grad der Sichtbarkeit zwischen den beiden Elementen. In diesem Zusammenhang ist auch der *Durchsichtigkeitsansatz* (engl. *transmittance*) [Sil94] zu nennen. Dort wird der Raum der Szene in Zellen unterteilt, wobei jede Zelle eine transmittance erhält, die aus der statistischen Verteilung der Elemente der Zelle berechnet wird. Die Sichtbarkeit zwischen Punktemengen in der Szene wird aus der transmittance der dazwischen liegenden Zellen berechnet.

Eine Vorberechnung der Sichtbarkeitsverhältnisse liegt auch dem Verfahren der *konservativen Sichtbarkeit* [TS91] zugrunde. Die Eingabeszene wird in Zellen unterteilt und die Sichtbarkeit zwischen Zellen abgeschätzt. Die Abschätzung der Sichtbarkeit wird durch die Überabschätzung der sichtbaren Polygone beschleunigt. In der zugrundeliegenden Anwendung des Durchlaufens und Betrachtens von Szenen wird dort, wo es sich aufgrund potentieller Verdeckungen als notwendig erweist, eine exakte Berechnung der traditionellen Sichtbarkeit vom betrachtenden Augenpunkt aus durchgeführt. Die konservative Sichtbarkeit dient damit zunächst der Beschleunigung der klassischen Sichtbarkeitsberechnung, die Abschätzung der hier betrachteten allgemeineren Sichtbarkeitsverhältnisse ist hierfür ein Hilfsmittel.

In dieser Arbeit werden neue Lösungsverfahren für verschiedene Versionen der Probleme der Berechnung der Sichtbarkeit zwischen zwei Elementen und Berechnung der sichtbaren Umgebung eines Elements entwickelt. In Kapitel 2 wird zunächst die Anfrageversion des Problems der Berechnung der Sichtbarkeit zwischen zwei Elementen behandelt. Dabei wird die gegebene Szene in eine Datenstruktur vorverarbeitet, so daß anschließend die Frage nach Sichtbarkeit für beliebige Anfrageelemente effizient beantwortet werden kann. Dies ist im gewissen Sinne vergleichbar mit dem Raytracing-Verfahren [Whi79]. Dort wird die Szene in eine Datenstruktur vorverarbeitet (Gitter, Octrees o.ä.), so daß die Anfrage nach einem Element, das von einem beliebigen Strahl geschnitten wird, effizient beantwortet werden kann [AK89]. Ein weiteres Beispiel ist die Vorverarbeitung einer Szene (z.B. in einen BSP-Baum) für die anschließende, effiziente Begehung (engl. *walk through*), so daß alle Elemente, die von einem gegebenen *Punkt* sichtbar sind, gefunden werden können [TS91]. Der Unterschied zwischen diesen Beispielen und den Problemen, die hier behandelt werden, ist, daß hier der Teil der Szene gesucht wird, der von einem *ausgedehnten Element* aus sichtbar ist.

Die Anfrageversion vermeidet es, die Sichtbarkeit explizit zu speichern, so wie es beim Ansichtsgraphen und Sichtbarkeitskomplex der Fall ist. Dadurch ist es möglich, den Speicherbedarf zu reduzieren, der bei expliziter Speicherung sehr groß sein kann. Dies wird durch etwas zusätzlichen Zeitbedarf erkauft, da die Antwort auf eine Anfrage nun aufwendiger zu berechnen ist als in dem Fall, in dem sie quasi schon explizit vorliegt. Diese Vorgehensweise ist in der Bilderzeugung überall dort zweckmäßig, wo nicht ohnehin die Lichtverteilung in Form von Lichtwegen zu speichern ist. Dies gilt insbesondere für den Fall, daß nur diffus reflektierende Elemente auftreten, da dann das emmitierte Licht als skalare Funktion auf der Elementoberfläche beschrieben werden kann. Diese skalare Funktion wird oftmals iterativ berechnet. Während eines Schritts dieser Iteration wird das Licht, das von einem Element in die Szene geschossen wird oder von dem Element aufgesammelt wird, bestimmt und für die Neubestimmung der Verteilung des Lichts genutzt. Für diese Bestimmung müssen die Teile der Szene berechnet werden, die von dem Element aus sichtbar sind.

Für zweidimensionale Szenen, die aus disjunkten Strecken bestehen, wird gezeigt, daß die Vorverarbeitung in eine Datenstruktur mit *ungefähr linearem Speicherbedarf* möglich ist, welche die Beantwortung der Anfrage nach Sichtbarkeit zwischen zwei Strecken asymptotisch schneller als durch systematisches Testen, d.h. in sublinearer Zeit in der Szenengröße, erlaubt. Methodisch wird auf Dualisierung und Partitionsbäume (engl. *partition trees*) [Mat91] zurückgegriffen, die sich auch schon bei der Behandlung des Strahlverfolgungsverfahrens bewährt haben [AK87, Mül88, deB93]. Die Einschränkung auf beinahe linearen Speicherplatzbedarf ist aus der Sicht der Praxis wichtig. Obwohl die hier vorgestellte Lösung theoretischer Natur ist, zeigt sie aber, daß eine solche Lösung existiert. Das Lösungsverfahren bildet die Grundlage für ein Verfahren zur Bestimmung der sichtbaren Umgebung eines Elements, das ebenfalls in Kapitel 2 vorgestellt wird.

Die Betrachtungen aus Sicht des schlechtesten Falls weisen die behandelten Sichtbarkeitsprobleme als aufwendig aus, wie es aufgrund der Kenntnisse bei den traditionellen Sichtbarkeitsberechnungen nicht anders zu erwarten war. Bei diesen traditionellen Problemen hat es sich jedoch gezeigt, daß es aus praktischer Sicht gute heuristische Lösungsverfahren gibt, d.h. Verfahren, die häufig, aber nicht in allen Fällen einen guten Aufwand liefern. Beispiele, die im Zusammenhang mit der Sichtbarkeitsberechnung, etwa durch Strahlverfolgung, Verwendung fanden, sind Hierarchien von Hüllkörpern oder Raumzerlegungen durch Gitter oder Octrees [AK89]. Entsprechendes wird in der Arbeit [TH93] gemacht, wo BSP-Bäume und Strahlverfolgung zu einer heuristischen Gesamtlösung des Sichtbarkeitsproblems im Rahmen des hierarchischen Radiosity-Ansatzes herangezogen werden.

Zur näherungsweisen, dafür aber praktikableren Lösung des Auffindens der Sichtbarkeit zwischen zwei beliebigen Anfrageelementen wird in Kapitel 3 dieser Dissertation der *hierarchische Verdeckungsbaum* eingeführt. Die Approximation besteht in der unvollständigen Beschreibung der gefundenen Sichtbarkeitsregionen. Die Sichtbarkeit wird im Prinzip durch drei Teilmengen der Menge aller Strecken zwischen den beiden Anfrageelementen beschrieben. Die Teilmenge vom Typ „frei” enthält nur Sichtstrecken zwischen den beiden Anfrageelementen. Die Teilmenge „blockiert” besteht aus Strecken, deren Inneres irgendein Szenenelement schneidet. Die Teilmenge „partiell" enthält Strecken, bei denen aufgrund des hierarchischen Verdeckungsbaums nicht entschieden werden kann, ob sie frei oder blockiert sind. Methodisch basiert der hierarchische Verdeckungsbaum auf regulären hierarchischen Zerlegungen etwa in Quad/Octree-Form, die sowohl im Szenenraum als auch im Streckenraum durchgeführt werden.

Hierarchische Verdeckungsbäume erlauben einen Trade-off zwischen Speicherbedarf und Laufzeit sowie Exaktheit der Beschreibung. Die Tiefe der Rekursion kann genutzt werden, um im Rahmen der Ressourcen beliebig genau zu rechnen oder eine weitere Auswertung der Sichtbarkeit bei Bedarf durch ein exaktes Verfahren durchzuführen. Das rechnerische Verhalten des Konzepts der hierarchischen Verdeckungsbäume wird in Implementierungen mit verschiedenen Baumformen empirisch analysiert.

In Kapitel 4 wird eine völlig andere Sichtweise für die Berechnung der Sichtbarkeit zwischen zwei Elementen in der Anfrageversion eingeführt, die sich auch auf andere Anfrageprobleme übertragen läßt und der die Idee der *Abtastung* (engl. *sampling*) zugrunde liegt. In einer Vorverarbeitungsphase ist zunächst eine endliche Anzahl von Anfragen mit den zugehörigen Antworten als Lösung zu bestimmen. Damit liegt eine endliche Anzahl von Stützstellen vor. In der Anfragephase wird die Antwort für eine beliebige Anfrage aus denjenigen vorausberechneten Anfragen berechnet, die in der „Nähe" der gestellten Anfrage liegen. Üblicherweise wird diese Vorgehensweise nicht immer eine korrekte Antwort liefern. Dies kann jedoch, wie schon beim hierarchischen Verdeckungsbaum bei der Bildsynthese, tolerierbar sein, bzw. die gefundene Lösung kann Ausgangspunkt für weitere Berechnungen sein, falls eine genauere Lösung notwendig ist.

Dieser Ansatz wird in Kapitel 4 ausgearbeitet. Es werden Aussagen über seine Leistungsfähigkeit hergeleitet werden, die die zwangsläufig auftretenden Fragen der Auswahl der Stichproben und der Komprimierung der Stichprobendatenbank zum Gegenstand haben. Dabei wird besonderes Augenmerk auf die Möglichkeit der Effizienzsteigerung durch korrigierbare näherungsweise Lösungen gelegt. Korrigierbarkeit ohne signifkanten Mehraufwand ist dann gegeben, wenn die gelieferte Antwort eine Obermenge der korrekten Antwort ist, deren Mächtigkeit sich nur durch einen konstanten Faktor von der Größe der korrekten Antwort unterscheidet. Die richtige Antwort kann aus dieser Obermenge durch eine nachgeschaltete Filterung in Zeit proportional zur Größe der korrekten Antwort bestimmt werden. Diese Vorgehensweise wird auf das Streckenanfrageproblem, das auch bei den anderen vorgestellten Lösungen Grundlage war, angewandt. Es werden Ergebnisse von Messungen von Speicherbedarf und Laufzeit für verschiedene Vorgehensweisen präsentiert und miteinander verglichen.

Kapitel 2

Exakte Element-Elementsichtbarkeit

Im folgenden werden zwei Arten von Sichtbarkeitsproblemen behandelt. Das erste Problem, die *Berechnung der Sichtbarkeitsregion zweier Elemente*, besteht darin, eine gegebene Szene von Elementen so in eine Datenstruktur vorzuverarbeiten, daß die Sichtbarkeitsregion zwischen zwei disjunkten konvexen Elementen **a** und **b**, desselben Typs und disjunkt vom Rest der Szene, effizient bestimmt werden kann. Das zweite Problem, *Berechnung der sichtbaren Umgebung für ein Element*, besteht darin, eine gegebene Szene von Elementen so in eine Datenstruktur vorzuverarbeiten, daß die Sichtbarkeit zwischen einem Element **a**, wiederum desselben Typs und disjunkt vom Rest der Szene, und allen anderen Szenenelementen effizient bestimmt werden kann. Das Ziel ist die Vorverarbeitung in eine Datenstruktur mit ungefähr *linearem Speicherbedarf*, welche die Beantwortung der Sichtbarkeitsanfrage schneller als durch systematisches Testen, d.h. in sublinearer Zeit in der Szenengröße, erlaubt.

Die Einschränkung auf beinahe linearen Speicherplatzbedarf ist aus der Sicht der Praxis wichtig. Obwohl die hier vorgestellte Lösung theoretischer Natur ist, zeigt sie aber, daß eine solche Lösung existiert. Dies unterscheidet sie vom *Sichtbarkeitskomplex* (*visibility complex*) [PV93, DDP96], der eine explizite Beschreibung der Sichtbarkeit innerhalb einer Szene gibt, die im schlechtesten Fall aber signifikant mehr als linearen Speicherplatz belegt.

Im folgenden Kapitel 2.1 werden zunächst die notwendigen formalen Begriffe für die hier verwendete Repräsentation von Sichtbarkeit eingeführt. Kapitel 2.2 beschreibt das Verfahren zur Beantwortung von Sichtbarkeitsanfragen an zwei ausgedehnte Elemente, Kapitel 2.3 das Verfahren zur Berechnung der sichtbaren Umgebung.

2.1 Repräsentation von Sichtbarkeit

Die Repräsentation der Sichtbarkeit, die hier benutzt wird, basiert auf Geraden. Eine nicht-vertikale Gerade in der Ebene kann durch die Gleichung $y = a \cdot x + b$ beschrieben werden. Die Menge dieser Geraden ist durch die Abbildung von $y = a \cdot x + b$ auf den Punkt (a, b) äquivalent zu $\mathbb{R}^2$.

Definition 2.1 (Steigungs-Achsenabschnitt-Koordinaten (SA-Koordinaten))
Die Steigungs-Achsenabschnitt-Koordinaten (SA-Koordinaten) (a, b) *einer nicht-vertikalen Geraden in der Ebene ergeben sich als die Koeffizienten der Punkt-Steigungsdarstellung* $y = a \cdot x + b$ *der Geraden.*

Durch die SA-Koordinaten wird die Menge der nicht-vertikalen Geraden in einen ebenen Punktraum, den SA-Dualraum *der Geraden abgebildet. Die Darstellung einer Geraden durch ihre SA-Koordinaten wird auch als* duale Repräsentation *der Geraden bezeichnet.*

Ein Punkt $\mathbf{p} = (p_x, p_y)$ in der x-y-Ebene kann auch durch die Menge aller Geraden beschrieben werden, die durch ihn gehen. Die Menge dieser Geraden wird im SA-Dualraum durch die Gerade $b = -p_x \cdot a + p_y$ beschrieben. Unter der SA-Dualisierung werden also Geraden im Originalraum in Punkte im Dualraum und Punkte im Originalraum in Geraden im Dualraum abgebildet (vgl. Abbildung 2.1). Die 2D-Szenen, die in diesem Kapitel

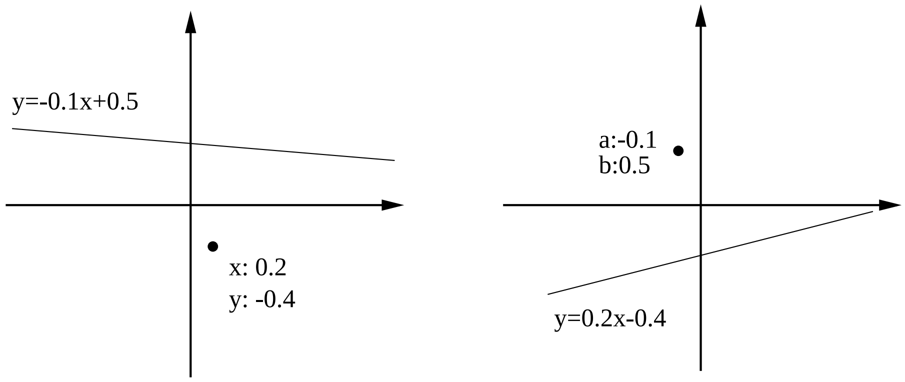


Abbildung 2.1: Dualisierung von Punkten bzw. Geraden des Originalraums (links) in Geraden bzw. Punkte des Dualraums (rechts)

betrachtet werden, bestehen aus n disjunkten Strecken, die durch ihre Endpunkte $\mathbf{p}_i$ und $\mathbf{q}_i$, $i = 1, \ldots, n$, gegeben sind. Sei P die Menge dieser Punkte. Die Menge der Strecken, die von einer Geraden $\mathbf{l}:\ y = a \cdot x + b$ geschnitten werden, ist (vgl. Abbildung 2.2):

$$\begin{aligned} S(\mathbf{l}) \ := \ & \{(\mathbf{p}_i, \mathbf{q}_i) \mid \\ & p_{i,y} \le a \cdot p_{i,x} + b \text{ und } q_{i,y} \ge a \cdot q_{i,x} + b, \\ & \text{oder } q_{i,y} \le a \cdot q_{i,x} + b \text{ und } p_{i,y} \ge a \cdot p_{i,x} + b\}. \end{aligned} \tag{2.1}$$

Das bedeutet, daß $S(\mathbf{l})$ aus den Punkten eines von $\mathbf{l}$ induzierten Halbraums besteht, deren Partnerpunkte in dem anderen von $\mathbf{l}$ induzierten Halbraum liegen (vgl. Abbildung 2.3).

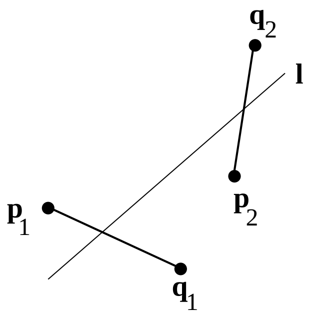


Abbildung 2.2: Strecken, die von einer Geraden aufgespießt werden

Diese Schachtelung zweier Halbräume wird später genutzt werden. Die Menge der Geraden, die von einer Strecke $\mathbf{s} : \overline{\mathbf{p}, \mathbf{p}'}$ geschnitten werden, ist

$$\begin{aligned} L(\mathbf{s}) \quad := \quad & \{y = a \cdot x + b \mid \\ & p_y \leq a \cdot p_x + b \text{ und } {p'}_y \geq a \cdot {p'}_x + b, \\ & \text{oder } p_y \geq a \cdot p_x + b \text{ und } {p'}_y \leq a \cdot {p'}_x + b\}. \end{aligned} \tag{2.2}$$

Diese Darstellung zeigt, daß sich die Menge aller Geraden, die eine gegebene Strecke schneiden, im Dualraum als ein Paar von Schnitten von zwei Halbebenen ergibt, die von dem Anfangs– und Endpunkt der Strecke induziert werden. Zur Definition von sogenannten

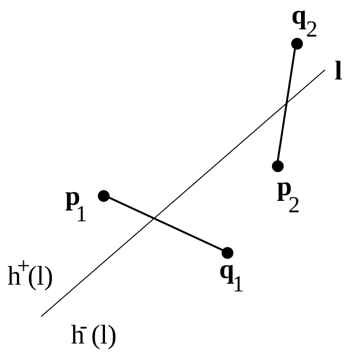


Abbildung 2.3: Geschachtelte Halbebenenanfrage

Sichtbarkeitsregionen von Szenenstrecken werden Stabber-Mengen benutzt.

Definition 2.2 (Stabber-Menge)
Gegeben sei eine Szene von disjunkten Strecken in der Ebene. Eine Stabber-Menge *ist eine zusammenhängende Menge von Geraden, die dieselben Szenenstrecken in derselben Reihenfolge schneiden. Eine Menge von Geraden ist* zusammenhängend*, wenn es zu zwei beliebigen ihrer Geraden einen Pfad im SA-Dualraum gibt, der komplett in der dualisierten Menge liegt.*

Abbildung 2.4 zeigt eine Szene, bestehend aus vier Szenenstrecken, und drei Geraden. Die Geraden schneiden paarweise unterschiedliche Mengen von Strecken und gehören daher drei unterschiedlichen Stabber-Mengen an.

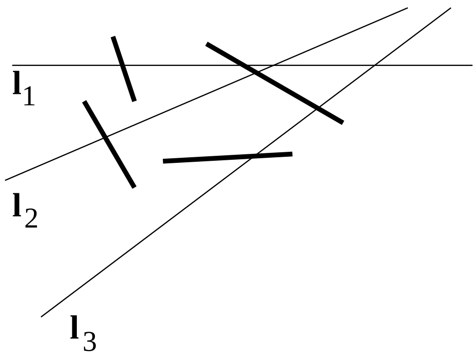


Abbildung 2.4: Drei Geraden $\mathbf{l}_1$, $\mathbf{l}_2$ und $\mathbf{l}_3$, die zu drei unterschiedlichen Stabber-Mengen gehören

Die Reihenfolge, in der die Szenenstrecken geschnitten werden, ist für alle Geraden der Stabber-Menge dieselbe. Andernfalls gäbe es auf jedem Pfad zwischen zwei Geraden mit unterschiedlicher Schnittpunktreihenfolge eine Gerade, auf der die zwei Schnittpunkte aufeinander fallen. Dies steht im Widerspruch zu der Annahme, daß die Szenenstrecken disjunkt sind.

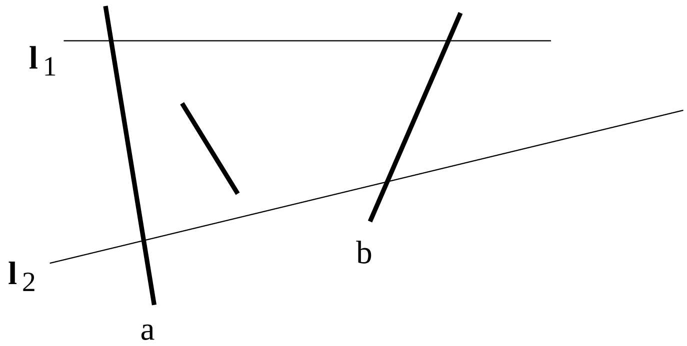


Abbildung 2.5: Zwei Geraden $\mathbf{l}_1$ und $\mathbf{l}_2$ der Sichtbarkeitsregion von **a** und **b**

Definition 2.3 (Sichtbarkeitsregion zwischen zwei Szenenstrecken)
Gegeben sei eine Szene von disjunkten Strecken in der Ebene. Zwei Szenenstrecken heißen gegenseitig sichtbar, *falls eine Stabber-Menge existiert, in der die zwei Szenenstrecken benachbart sind. Zwei Szenenstrecken heißen* benachbart, *falls ihre Schnittpunkte mit einer Geraden der Stabber-Menge auf der Geraden benachbart sind.*

Die Sichtbarkeitsregion $V(\mathbf{s}_1, \mathbf{s}_2)$ *zweier Szenenstrecken* $\mathbf{s}_1$*,* $\mathbf{s}_2$ *ist die Vereinigung aller Stabber-Mengen, in der die beiden Szenenstrecken benachbart sind. Die* Verdeckungsregion $B(\mathbf{s}_1, \mathbf{s}_2)$ *zweier Szenenstrecken ist die Vereinigung aller Stabber-Mengen, in der die zwei Szenenstrecken nicht benachbart sind.*

Die Sichtbarkeitsregion zweier Szenenstrecken ist nicht notwendigerweise zusammenhängend, denn die Sicht zwischen zwei Szenenstrecken kann teilweise durch andere Szenenstrecken verdeckt sein, siehe Abbildung 2.5. Der in der Einleitung erwähnte Sichtbarkeitskomplex beschreibt die Sichtbarkeit durch zusammenhängende Komponenten der Sichtbarkeitsregion [PV93, DDP96].

Im dreidimensionalen Fall ist die Dualisierung der Geraden über *Plücker-Koordinaten* nützlich [CEGS89, Sto91, Plü65]. Dies liegt daran, daß sie für Szenen aus disjunkten Dreiecken im Raum eine entsprechende Darstellung wie für zweidimensionale Szenen durch die SA-Koordinaten erlaubt, wobei die Dualdarstellung von Stabber-Mengen ebenfalls in linearer Weise beschrieben werden kann, im Unterschied etwa zu unmittelbareren Übertragungen der SA-Darstellung auf den dreidimensionalen Fall.

Plücker-Koordinaten repräsentieren eine gerichtete Gerade als sechsdimensionalen dualen Punkt:

Definition 2.4 (Plücker-Koordinaten)
Sei $\mathbf{l}$ *eine Gerade in* $I\!R^3$ *sowie* $\mathbf{a} = (a_0, a_1, a_2, a_3)$ *und* $\mathbf{b} = (b_0, b_1, b_2, b_3)$ *zwei Punkte auf* $\mathbf{l}$*, gegeben in homogenen Koordinaten.* $\mathbf{l}$ *sei von* $\mathbf{a}$ *nach* $\mathbf{b}$ *gerichtet. Seien*

$$\xi_{ij} = \begin{vmatrix} a_i & a_j \\ b_i & b_j \end{vmatrix}$$

die Determinanten der 2×2*-Untermatrizen der Matrix*

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \end{pmatrix}, \quad a_0, b_0 > 0.$$

Dann sind die homogenen Plücker-Koordinaten *von* $\mathbf{l}$ *gegeben durch*

$$\boldsymbol{\pi}(\mathbf{l}) = (\xi_{01}, \xi_{02}, \xi_{03}, \xi_{12}, \xi_{31}, \xi_{23}).$$

$\boldsymbol{\pi}(\boldsymbol{l})$ *ist der* Plückerpunkt *von* $\mathbf{l}$ *im projektiven fünfdimensionalen Raum* $I\!P^5$*.*

Definition 2.5 (Plücker-Hyperebene)
Die Hyperebene $\mathbf{p}(\mathbf{l})$ *in* $I\!P^5$ *mit dem Koeffizientenvektor* $\mathbf{v}(\mathbf{l}) = (\xi_{23}, \xi_{31}, \xi_{12}, \xi_{03}, \xi_{02}, \xi_{01})$ *ist die* Plücker-Hyperebene *von* $\mathbf{l}$*.*

$$\mathbf{p}(\mathbf{l}) = \{\mathbf{p} \in I\!P^5 \mid \mathbf{v}(\mathbf{l}) \cdot \mathbf{p} = 0\}$$

Für jede Plücker-Hyperebene kann ein positiver und negativer Halbraum

$$\begin{aligned}\mathbf{h}^+(\mathbf{l}) &= \{\mathbf{p} \in I\!P^5 \mid \mathbf{v}(\mathbf{l}) \cdot \mathbf{p} \geq 0\}, \\ \mathbf{h}^-(\mathbf{l}) &= \{\mathbf{p} \in I\!P^5 \mid \mathbf{v}(\mathbf{l}) \cdot \mathbf{p} \leq 0\}\end{aligned}$$

definiert werden.

Nicht jedes 6-Tupel beschreibt eine Gerade $\mathbf{l}$ aus $I\!R^3$. Vielmehr muß für die Tupel die Bedingung

$$\xi_{01}\xi_{23} + \xi_{02}\xi_{31} + \xi_{03}\xi_{12} = 0 \tag{2.3}$$

gelten. Alle Punkte, die Gleichung 2.3 erfüllen, beschreiben die *Plücker-Hyperfläche* Π, die auch *Graßmann-Mannigfaltigkeit* genannt wird.

Die Position des Plücker-Punktes einer gerichteten Geraden $\mathbf{l}$ in Bezug auf die Plücker-Ebene einer anderen gerichteten Geraden $\mathbf{l}'$ charakterisiert die gegenseitige Orientierung von $\mathbf{l}$ und $\mathbf{l}'$. Dies kann für den Schnitt von Geraden mit Dreiecken genutzt werden. Dazu werden die Geraden $\mathbf{l}'$, $\mathbf{l}''$ und $\mathbf{l}'''$ entlang der Seiten des Dreiecks $\mathbf{t}$ zyklisch ausgerichtet. Eine gerichtete Gerade $\mathbf{l}$ schneidet $\mathbf{t}$, wenn der Plücker-Punkt von $\mathbf{l}$ entweder auf der positiven oder negativen Seite aller Plücker-Hyperebenen von $\mathbf{l}'$, $\mathbf{l}''$ und $\mathbf{l}'''$ liegt. Wie im 2-D Fall kann also die Menge der Geraden, die ein gegebenes Dreieck schneiden, als der Schnitt einer konstanten Anzahl von linearen Halbräumen im Dualraum beschrieben werden. Weiterhin können die Dreiecke, die von einer gerichteten Geraden geschnitten werden, durch eine Schachtelung von Halbräumen im Dualraum beschrieben werden. Einige weitere geometrische Tests mit Plückerkoordinaten und Plückerkoeffizienten werden in [YN97] beschrieben.

Der auf den ersten Blick existierende Nachteil der Fünfdimensionalität des Plücker-Raumes gegenüber den vier Dimensionen anderer Dualisierungen, wie etwa die Repräsentation von Geraden über Schnitte mit 2 parallelen Ebenen bei $z = 0$ und $z = 1$, wird durch folgende Beobachtung ausgeräumt:

Theorem 2.1 *[APS93] Gegeben sei eine Menge H von Hyperebenen in $I\!P^5$. Dann ist die Anzahl der Zellen des Durchschnitts des Arrangements $\mathcal{A}(H)$, das von diesen Hyperebenen gebildet wird, mit der Plücker-Hyperfläche Π durch $O(n^4 log(n))$ beschränkt.*

Die folgenden Abschnitte dieses Kapitels beziehen sich auf zweidimensionale Szenen. Dreidimensionale Szenen aus disjunkten Dreiecken können aufgrund der beschriebenen Eigenschaften durch Anwendung der Dualisierung über Plücker-Koordinaten analog behandelt werden.

2.2 Sichtbarkeitsanfragen mit zwei ausgedehnten Elementen

2.2.1 Das Problem

Mit den Begriffen des vorigen Abschnitts läßt sich das in diesem Abschnitt behandelte Problem so formulieren:

Definition 2.6 (Sichtbarkeit zwischen zwei Strecken)
Eingabe: *Eine Szene aus endlich vielen disjunkten Strecken in der Ebene.*
Ausgabe: *Die Sichtbarkeitsregion* $V(\mathbf{s}_1, \mathbf{s}_2)$ *zweier Strecken* $\mathbf{s}_1$ *und* $\mathbf{s}_2$*, die entweder Szenenstrecken sind oder diese nicht schneiden.*

2.2.2 Lösung

Die hier beschriebene Lösung des Problems der Bestimmung der Sichtbarkeit zwischen zwei Elementen benutzt die Repräsentation der Sichtbarkeitsregionen als Vereinigung von Stabber-Mengen. Die Lösung verwendet *Halbraumunterteilungs-Schemata* [DE87, AS91, Mat92]. Ein Halbraumunterteilungs-Schema ist eine Datenstruktur, die aus geschachtelten *Partitionen* zusammengesetzt ist. Die Schachtelung der Partitionen reflektiert eine Abfolge von Halbraum-Anfragen, die die Antwort zum ursprünglichen Problem liefert. Die Partitionen werden über endlichen Mengen von Punkten aufgebaut. Es sind Datenstrukturen, in die diese Punkte für eine effiziente Beantwortung der Halbraumanfragen vorverarbeitet werden. Eine *Halbraumanfrage* resultiert in der Auffindung aller Punkte aus einer endlichen Menge von Punkten, die in dem Anfragehalbraum liegen.

Definition 2.7 (Halbraumanfrage)
Eingabe: *Eine endliche Menge* S *von Punkten in* $\mathbb{R}^d$*.*
Ausgabe: *Für einen beliebigen Halbraum* $\mathbf{h}$*, der durch eine Hyperebene begrenzt ist, die Punkte aus* S*, die in* $\mathbf{h}$ *enthalten sind.*

Die Partitionen, die für das *Halbraum-Anfrageproblem* benötigt werden, entstehen durch die Unterteilung der gegebenen Menge von Punkten in eine konstante Anzahl von Teilmengen, die die günstige Eigenschaft haben, daß nur ein bestimmter Teil von ihnen von irgendwelchen Anfragegeraden geschnitten wird. Die resultierenden Partitionselemente werden rekursiv auf gleiche Art und Weise unterteilt, bis eine konstante Größe erreicht ist (vgl. Abbildung 2.6). Für die hier genannte Anwendung sollten die Partitionen wie folgt beschaffen sein:

Definition 2.8 (Geeignete r-Partition)
Eine geeignete r-Partition *einer endlichen Menge von Punkten in* $\mathbb{R}^d$ *ist ein Baum mit den folgenden Eigenschaften:*

1. *Der Grad der Knoten ist begrenzt durch einen Wert* $g(r) = \Theta(r)$.
2. *Jeder Knoten* s *repräsentiert eine Untermenge* $P(s)$ *der gegebenen Menge von Punkten. Die Wurzel repräsentiert die komplette Eingabemenge. Die Untermengen von Nachfolgern* s' *eines inneren Knotens bilden eine Partition von* $P(s)$. *Es existiert ein* $\rho(r)$, $0 < \rho(r) < 1$, *so daß* $|P(s')| \leq \rho(r)|P(s)|$. *Die Größe der Untermenge eines Blattes ist durch eine Konstante nach oben begrenzt.*
3. *Jeder Knoten* s *besitzt ein konvexes Polytop* $\mathcal{C}(s)$ *mit* $P(s) \subset \mathcal{C}(s)$. *Die Anzahl der Elemente (Ecken, Kanten, Flächen usw.) von* $\mathcal{C}(s)$ *ist durch einen Wert* $c(d)$ *begrenzt, der nicht von* r *abhängt.*
4. *Die Anzahl der Polytope* $\mathcal{C}(s')$ *der Nachfolger* s' *eines Knotens* s, *die eine beliebige Halbebene schneiden, ist durch einen Wert* $f(r) < g(r)$ *begrenzt.*

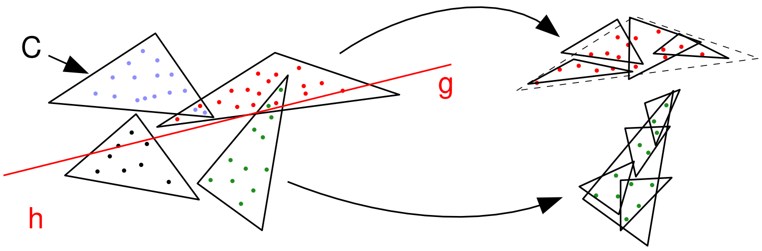


Abbildung 2.6: Untermengenbildung der gegebenen Punktmenge (schematisch)

Ein Beispiel einer solchen geeigneten Partition sind die *partition trees* aus [Mat91], mit $g(r) = \Theta(r)$, $\rho(r) = 2/g(r)$, $f(r) = O(r^{1-1/d})$, für $r \leq n^{1-\delta}$, $\delta > 0$ konstant. Diese Definition ist konservativ, aber ausreichend für das hier verfolgte Ziel, das Prinzip zu demonstrieren. Eine allgemeinere Version, die entworfen wurde, um die Komplexität weiter zu verringern, kann in [Mat92] gefunden werden.

Der Speicherbedarf $S(n)$ einer r-Partition von n Punkten genügt

$$S(n) = O(g(r)) + g(r) \cdot S(\rho(r) \cdot n). \tag{2.4}$$

Mit $g(r) = \Theta(r)$ und $\rho(r) = 2/g(r)$ gilt

$$S(n) = O(n^{1+\varepsilon}) \tag{2.5}$$

für ein beliebiges, festes $\varepsilon > 0$ und eine ausreichend große Konstante r.

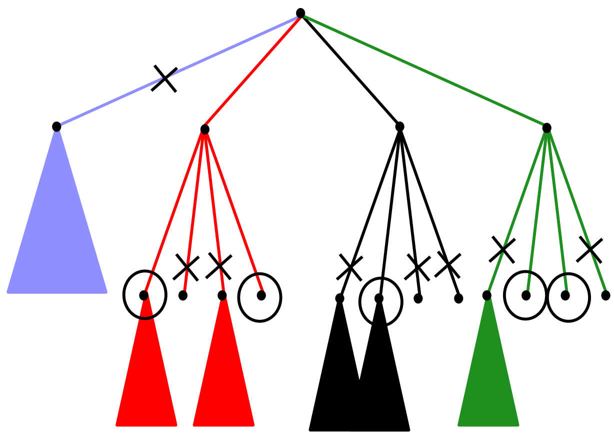


Abbildung 2.7: Besuchte Knoten eines Partitionsbaums während einer Halbraumanfrage

Eine *Halbraumanfrage* wird durch Traversierung einer Teilmenge der Knoten der Partitionen beantwortet (vgl. Abbildung 2.7). Für Knoten, dessen Polytope komplett innerhalb des Anfragehalbraums liegen, werden die Punkte ausgegeben, die im entsprechenden Teilbaum gespeichert sind.

Bei Knoten, die komplett außerhalb des Anfragehalbraums liegen, wird die Suche abgebrochen. Für solche Knoten, deren Polytope von der Ebene, die den Anfragehalbraum begrenzt, geschnitten werden, wird die Suche bei deren Söhnen fortgeführt. Wenn der Knoten ein Blatt ist, wird entweder eine Referenz auf den Knoten oder die Menge der Punkte, die innerhalb des Anfrageraums liegen, zurückgegeben. Die Anfragezeit für diese Prozedur ist

$$Q(n) \quad = \quad O(g(r)) + f(r) \cdot Q(\rho(r) \cdot n). \tag{2.6}$$

Für $g(r) = \Theta(r)$, $f(r) = O(r^{1-1/d})$, und $\rho(r) = 2/g(r)$ ergibt sich

$$Q(n) \quad = \quad O(n^{\alpha}) \tag{2.7}$$

mit $\alpha = (1 - 1/d)(\log r / \log(r-1)) \leq (1 - 1/d)(1 + \varepsilon)$, für eine beliebige Konstante $\varepsilon > 0$ und ein genügend großes r.

Es ist bekannt, daß geeignete r-Partitionen existieren, die einen Speicherbedarf von $S(n) = O(n^{1+\varepsilon})$ haben ($\varepsilon > 0$ beliebig, aber konstant) oder $S(n) = O(n \text{ polylog } n)$ (wobei „polylog" einen Faktor repräsentiert, der ein Polynom in $\log n$ ist), so daß die I Strecken, die von einer Geraden geschnitten werden, in Zeit $O(n^{\alpha} + I)$ (α ist eine Konstante zwischen 0 und 1) bestimmt werden können (vgl. [Wil82, Wel88, Mat91, AS91]). Insbesondere existieren r-Partitionen mit $g(r) = \Theta(r)$, $f(r) = O(r^{1-1/d})$ und $\rho(r) = 2/g(r)$. Das Ergebnis besteht aus $O(n^{\alpha})$ Teilszenen, die in der Datenstruktur gespeichert sind.

Hierarchische Dekomposition ist wie folgt definiert:

Definition 2.9 (Hierarchisches Dekompositionsschema)
Ein hierarchisches Dekompositionsschema *für eine endliche Menge von Punkten aus* $I\!R^{d_1} \times \ldots \times I\!R^{d_m}$ *ist ein Baum* T *mit den folgenden Eigenschaften:*

1. *T enthält Teilbäume, die je zu einer von m Ebenen der Dekomposition gehören.*
2. *Die d_1-dimensionale r-Partition der Projektion der gegebenen m-Tupel auf deren erste Komponente ist ein Teilbaum von T. Dessen Wurzel ist die Wurzel von T.*
3. *Jeder Knoten s von T ist ein Knoten einer d_k-dimensionalen r-Partition einer Projektion einer Untermenge der gegebenen m-Tupel auf deren k-te Komponente.*
4. *Jeder Knoten s, der kein Blatt ist, auf einer Dekompositionsebene $k < m$ zeigt auf die Wurzel einer d_{k+1}-dimensionalen r-Partition der Projektion von $P(s)$ auf deren $(k+1)$-te Komponente. $P(s)$ ist die Menge der m-Tupel, die durch s repräsentiert werden.*
5. *Für jeden Knoten s bezeichnet $S(s)$ die Menge der m-Tupel, aus deren Projektion $P(s)$ besteht.*

Dies soll am Beispiel des *Geradenschnittproblems* in zweidimensionalen Szenen aus Strecken illustriert werden. Das Problem hier ist, die gegebene Szene so in eine Datenstruktur vorzuverarbeiten, daß die Strecken, die von einer Anfragegeraden geschnitten werden, effizient gefunden werden können.

Definition 2.10 (Geradenschnittproblem)
Eingabe: *Eine endliche Menge S disjunkter Strecken aus $I\!R^2$.*
Ausgabe: *Für eine beliebige Gerade $\mathbf{l}$, die Strecken aus S, die von $\mathbf{l}$ geschnitten werden.*

Zur Lösung des Problems kann die Tatsache aus 2.1 genutzt werden, daß die Menge der geschnittenen Strecken mit zwei geschachtelten Halbraumanfragen ermittelt werden kann. Dies führt zu einem hierarchischen Dekompositionsschema mit zwei Ebenen. Die 2-Tupel sind $(\mathbf{p}, \mathbf{p}')$, wobei $\overline{\mathbf{pp}'}$ die gegebenen Strecken sind. Die resultierende Halbraumdekomposition besteht aus einer primären r-Partition der Menge der Punkte $\mathbf{p}$ der gegebenen Strecken. Jeder Knoten s der primären Partition referenziert den Knoten s' einer sekundären Partition der Menge der Endpunkte $\mathbf{p}'$ der gegebenen Strecken $S(s)$, die von s repräsentiert werden (vgl. Abbildung 2.8). s' ist die Wurzel der r-Partition der sekundären Ebene dieser Punkte. Der Speicherbedarf dieses Dekompositionsschemas, das auf geeigneten r-Partitionen basiert, kann rekursiv wie folgt beschrieben werden:

$$S_k(n) \quad = \quad O(g(r)) + g(r) \cdot S_k(\rho(r) \cdot n) + S_{k+1}(n). \tag{2.8}$$

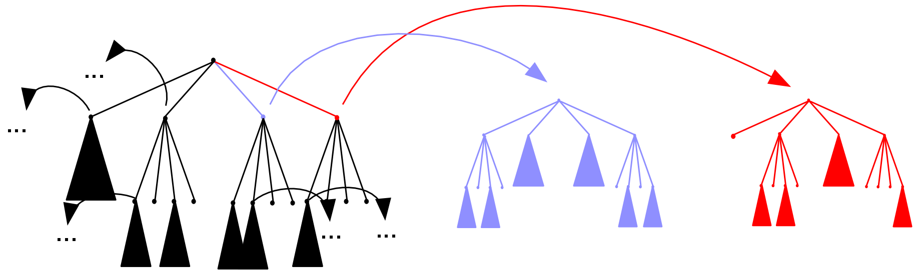


Abbildung 2.8: Hierarchische Partitionsbäume

S_k ist der Speicherbedarf der Struktur auf Ebene k. $k = 0$ ist die Anfangsebene der Struktur. Der Speicherbedarf der höchsten Ebene $k = m$ kann mit Hilfe der Formel 2.5 abgeschätzt werden. Für $g(r) = \Theta(r)$ und $\rho(r) = 2/g(r)$ ergibt sich

$$S_0(n) \quad = \quad O(n^{1+\varepsilon}) \tag{2.9}$$

für ein beliebiges, aber konstantes $\varepsilon > 0$ und ein genügend großes r.

Eine *geschachtelte Halbraumanfrage* ist ein m-Tupel von Halbraumanfragen, wobei sich das k-te Element auf die k-te Projektion der gegebenen m-Tupel bezieht. Es müssen alle m-Tupel gefunden werden, die allen Halbraumanfragen auf allen m Projektionen genügen. Eine geschachtelte Halbraumanfrage $\mathbf{q}$ wird durch die Traversierung einer Untermenge der Knoten der r-Partitionen beantwortet, wobei mit der Halbraumanfrage der ersten Komponente von $\mathbf{q}$ an der Wurzel der hierarchischen Dekomposition begonnen wird. Für Knoten, dessen Polytope komplett im Anfragehalbraum liegen, wird die Suche mit der Halbraumanfrage der zweiten Komponente von $\mathbf{q}$ in der zugehörigen r-Partition der zweiten Ebene fortgeführt. Dieses wird über alle Ebenen der Dekomposition fortgeführt. Für Knoten, deren Polytope von der Ebene geschnitten werden, die den Anfragehalbraum begrenzt, wird die Suche in ihren Nachfolgern auf derselben Ebene der Dekomposition fortgeführt. Wenn der Knoten ein Blatt ist, wird entweder eine Referenz auf den Knoten oder die Antwort auf die Anfrage $\mathbf{q}$ durch den Knoten zurückgegeben. Die Zeit für die Beantwortung der Anfrage kann rekursiv wie folgt abgeschätzt werden:

$$Q_k(n) \quad = \quad O(g(r)) + f(r) \cdot Q_k(\rho(r) \cdot n) + Q_{k+1}(n). \tag{2.10}$$

Für den höchsten Level $k = m$, wird die Anfrage durch 2.7 abgeschätzt.

Für $g(r) = \Theta(r)$, $f(r) = O(r^{1-1/d})$, und $\rho(r) = 2/g(r)$ ergibt sich

$$Q_0(n) \quad = \quad O(n^{(1-1/d)(1+\varepsilon)}) \tag{2.11}$$

für ein beliebiges, aber konstantes $\varepsilon > 0$ und ein genügend großes r.

Für das Geradenschnittproblem ergibt sich eine geschachtelte Halbraumanfrage, die aus zwei klassischen Halbraumanfragen besteht. Die Begrenzung der Halbräume beider Anfragen ist durch die Anfragegerade $\mathbf{l}$ definiert. Einer der beiden Halbräume ist der untere Halbraum, der andere der obere Halbraum, wie in Formel 2.1.

Im Gegensatz zur Sichtbarkeitsanfrage mit nur einer Geraden besteht bei den hier betrachteten Sichtbarkeitsproblemen eine Sichtbarkeitsanfrage aus einer Menge von unendlich vielen Geraden. Für diese Anfragen muß die Anfrageprozedur wie im nächsten Abschnitt beschrieben modifiziert werden.

2.2.3 Sichtbarkeit zwischen zwei Elementen

Für die Lösung des zweidimensionalen Problems der Sichtbarkeit zwischen zwei Strecken wird die soeben vorgestellte Anfragedatenstruktur verwendet. Im Raum werden die Partitionen über die drei gerichteten Geraden, die das Dreieck begrenzen, aufgebaut. Die Partitionen werden dann in drei Ebenen statt in zwei geteilt. Im folgenden wird das Hauptaugenmerk auf den zweidimensionalen Fall gelegt. Die Lösung wird aber so beschrieben, daß sie auf den dreidimensionalen Fall übertragen werden kann.

Die Datenstruktur für die Geradenanfrage wird etwas modifiziert. Für jeden Knoten s, dessen entsprechende Teilszene $S(s)$ von einer Geraden aufgespießt werden kann, d.h. es existiert eine Gerade, die alle Strecken von $S(s)$ schneidet, werden die Strecken in der Reihenfolge, in der sie von $\mathbf{l}$ geschnitten werden, bei s gespeichert. Der asymptotische Speicherbedarf ist derselbe wie zuvor, mit etwas schlechterem ε.

Sei $L[\mathbf{s}_1, \mathbf{s}_2]$ die Menge der Geraden, die die beiden Strecken $\mathbf{s}_1$ und $\mathbf{s}_2$ gleichzeitig schneiden. Während der Traversierung der hierarchischen Partitionen wird $L[\mathbf{s}_1, \mathbf{s}_2]$ in Stabber-Mengen unterteilt. Dafür wird der Partitions-Baum in post-order Reihenfolge traversiert. Algorithmus 1 zeigt den rekursiven Pseudocode *GenerateStabbers*, der hierfür verwendet wird.

GenerateStabbers hat zwei Argumente: Eine Anfrage $\mathbf{q}$ und ein Knoten s der hierarchischen Dekomposition. Eine Anfrage besteht aus zwei Elementen: der Menge $L(\mathbf{q})$ der Strecken, die die Anfrage definieren, und einer Datenstruktur $A(\mathbf{q})$, die die Antwort speichert. *GenerateStabbers* gibt eine neue Menge *GenerateStabbers* $(\mathbf{q}, s)$ von Anfragen zurück. Jede Anfrage $\mathbf{q}'$ dieser Menge besteht aus einer Stabber-Menge $L(\mathbf{q}')$ bezüglich der Teilszene $S(s)$, die durch den Teilbaum mit der Wurzel bei s repräsentiert wird und aus einer Beschreibung $A(\mathbf{q}')$ der Strecken von $S(s)$, die durch $L(\mathbf{q}')$ aufgespießt werden. Die Vereinigung aller $L(\mathbf{q}')$ ist $L(\mathbf{q})$. Daher ist die Menge *GenerateStabbers*$(\mathbf{q}, s)$ eine komplette Beschreibung der Stabber-Situation aller Geraden in $L(\mathbf{q})$ bezüglich der Unterszene $S(s)$.

Die Traversierung der hierarchischen Dekomposition in *GenerateStabbers* wird hauptsächlich durch den Algorithmus *BelongsTo* kontrolliert. *BelongsTo* hat zwei Parameter:

Algorithmus 1 Rekursiver Algorithmus zur Aufzählung der Stabber-Mengen für eine Sichtbarkeitsanfrage an ausgedehnte Elemente

Parameter: s: Wurzel eines hierarchischen Verdeckungsbaums, $\mathbf{q}$: Anfrage

GenerateStabbers($\mathbf{q}, s$)

```
if BelongsTo(q, s) = not then
  return q
end if
if BelongsTo(q, s) = completely then
  if NotFinalPartion(s) then
    return GenerateStabbers(q,NextLevel(s))
  else
    return ReportStabber(q, s);
  end if
end if
if BelongsTo(q, s) = partially then
  if s leaf then
    return StabberDecomposition(q, s)
  else
    Q = {};
    for q' in Split(q, s) do
      P = {q'}
      for s' in Successor(s) do
        R = {};
        for q" in P do
          R = R ∪ GenerateStabbers(q'', s')
        end for
        P = R
      end for
      Q = Q ∪ P;
    end for
    return Q
  end if
end if
```

eine Anfrage $\mathbf{q}$ und einen Knoten s der hierarchischen Dekomposition. Die Rückgabe kann drei mögliche Werte annehmen: „*not*", „*completely*" und „*partially*".

„*not*" steht für den Fall, daß das Polytop $\mathcal{C}(s)$ keinen Halbraum der Anfragehalbräume schneidet, die durch $L(\mathbf{q})$ repräsentiert werden. „*completely*" bedeutet, daß das Polytop $\mathcal{C}(S)$ eine Teilmenge des Anfragehalbraums aller Halbraumanfragen ist, die durch $L(\mathbf{q})$ repräsentiert werden. In allen anderen Fällen wird $L(\mathbf{q})$ als „*partially*" klassifiziert.

Der einfachste Fall ist „*not*". In diesem Fall wird $\mathbf{q}$ ohne Modifikation zurückgegeben.

Bei Rückgabe von „*completely*" müssen zwei Fälle unterschieden werden. Falls s kein Knoten der zweiten Ebene der Partitionen ist, dann wird *GenerateStabbers* für $\mathbf{q}$ und den Knoten s' (=*Nextlevel*(s) in Algorithmus 1) der zweiten Ebene, der von s referenziert wird, ausgeführt. D.h., daß die Partnerpunkte der Punkte von $P(s)$ bezüglich der zweiten Halbebenenanfragen, die durch die Geraden in $L(\mathbf{q})$ induziert werden, untersucht werden. Das Ergebnis *GenerateStabbers*$(\mathbf{q},s')$ dieser Untersuchung ergibt die Stabber-Mengen, die durch den Teilbaum mit Wurzel s' induziert werden, welche als Teil der kompletten Lösung zurückgegeben werden.

Wenn s ein Knoten der zweiten Ebene ist, dann wird eine Referenz auf s in $A(\mathbf{q})$ gespeichert. In Algorithmus 1 geschieht dies durch *ReportStabber*. *ReportStabber* gibt diese modifizierte Anfrage zurück, welche wiederum von *GenerateStabbers* als Teil der kompletten Lösung weitergegeben wird.

Der komplizierteste Fall ist „*partially*". Es gibt hier zwei Fälle: Falls s ein Blatt ist, dann wird $L(\mathbf{q})$ explizit in Stabber-Mengen $L(\mathbf{q}')$ bezüglich der Teilszene $S(s)$ zerlegt, die durch s repräsentiert wird. In $A(\mathbf{q}')$ werden die Strecken von $S(s)$ gespeichert, die von $L(\mathbf{q}')$ aufgespießt werden, sortiert nach Aufspießreihenfolge. In Algorithmus 1 wird dieser Fall durch *StabberDecomposition* behandelt. Das Ergebnis wird durch *GenerateStabbers* als Teil der kompletten Lösung weitergereicht.

Die explizite Zerlegung von $L(\mathbf{q})$ für $S(s)$ kann im Dualraum durchgeführt werden. Das *Arrangement* der dualen Geraden zu den Endpunkten der Strecken von $S(s)$ induziert eine Zellzerlegung im dualen Raum. Die zu den Punkten in einer Zelle c dualen Geraden haben alle dieselbe Antwort A auf jede der zwei Halbraumanfragen, die durch sie induziert werden. Die dualen Geraden der gegenüberliegenden Punkte der Strecken von $S(s)$ induzieren eine andere Zellzerlegung des dualen Raums. Die nichtleeren Schnitte von $L(\mathbf{q})$ mit den Zellen, die aus dem Schnitt dieser Zellzerlegung mit c entstehen, werden als die gewünschten Stabber-Mengen $L(\mathbf{q}')$ genommen.

Falls s kein Blatt ist, wird $L(\mathbf{q})$ in Teilmengen $L(\mathbf{q}')$ von Geraden zerlegt (Algorithmus *Split*), so daß die Halbräume der Halbraumanfragen von zwei beliebigen von ihnen das gleiche Schnittverhalten haben bzgl. der Polytope $\mathcal{C}(s')$ der Nachfolger von s in deren r-Partition. D.h., daß der Halbraum aller Geraden für jedes $\mathcal{C}(s')$ entweder $\mathcal{C}(s')$ enthält, oder sein Komplement enthält $\mathcal{C}(s')$, oder $\mathcal{C}(s')$ hat einen nichtleeren Schnitt mit dem Halbraum sowie mit seinem Komplement.

Die Zerlegung wird durch die Zellzerlegung des Dualraums durchgeführt, die durch die dualen Geraden der Polytope $\mathcal{C}(s')$ der Nachfolger s' von s induziert wird. Die $L(\mathbf{q}')$ sind als die Schnitte von $L(\mathbf{q})$ mit Zellen der Zellzerlegung definiert, die nichtleer sind.

Die Zerlegung von $L(\mathbf{q}')$ in Stabber-Mengen bezüglich der Teilszene $S(s)$ des Teilbaums mit der Wurzel bei s wird durch sequentielle Abarbeitung der Nachfolgeknoten von s bestimmt (vgl. Abbildung 2.9). Für den ersten dieser Knoten s' werden die Stabber-Mengen von $L(\mathbf{q}')$ bezüglich $S(s')$ durch den rekursiven Aufruf von *GenerateStabbers*($\mathbf{q}'$,s') bestimmt. Sie werden einer Menge P hinzugefügt. Dieses ist die anfängliche Situation, die nun angenommen wird während der Abarbeitung aller weiteren Nachfolger s' von s. Jedes $L(\mathbf{q}'') \in P$ wird wie gerade beschrieben behandelt und wird in P durch die von ihm abgeleiteten Stabber-Mengen ersetzt. Nach der Abarbeitung aller Nachfolger s' von s enthält P die gewünschte Zerlegung von $L(\mathbf{q}')$.

Jede der Mengen $L(\mathbf{q}')$ wird auf diese Weise in Stabbermengen zerlegt. Die einzelnen Zerlegungen werden in einer Menge Q vereinigt und diese als Teil der kompletten Lösung von *GenerateStabbers* zurückgegeben.

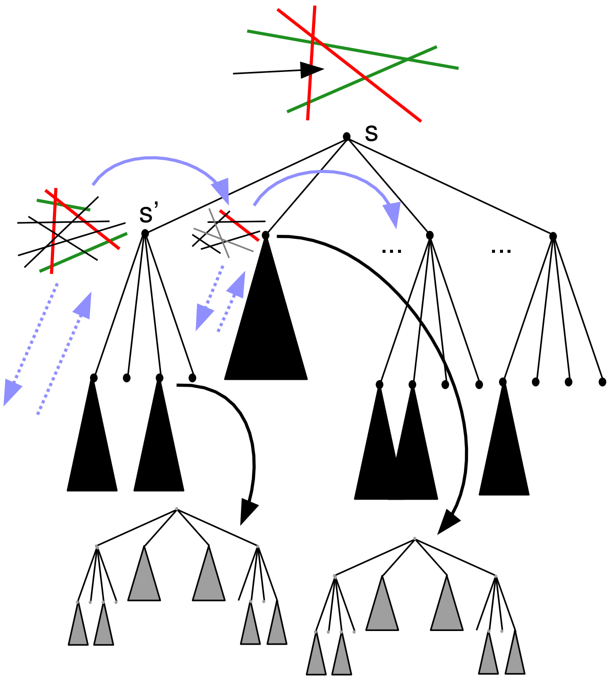


Abbildung 2.9: Zerlegung der Stabber-Mengen in Teilstabbermengen

Der Hauptalgorithmus besteht nun einfach aus der Ausführung von *GenerateStabbers* mit $L(\mathbf{q}) = L[\mathbf{s}_1, \mathbf{s}_2]$ und $A(\mathbf{q}) = \emptyset$. $L(\mathbf{q})$ ist der Bereich im Dualraum, der durch den Schnitt von vier Halbebenen entsteht, die durch die vier zu den Endpunkten der Strecken $\mathbf{s}_1$ und $\mathbf{s}_2$ dualen Geraden induziert werden. Jede der zurückgegebenen Stabber-Mengen wird darauf getestet, ob eine Szenenstrecke zwischen $\mathbf{s}_1$ und $\mathbf{s}_2$ liegt. Ist das nicht der Fall, so wird die Stabber-Menge zur Sichtbarkeitsregion hinzugefügt, ansonsten zur Verdeckungsmenge. Hiermit wird das Problem der Sichtbarkeit zwischen $\mathbf{s}_1$ und $\mathbf{s}_2$ gelöst.

Eine explizite Repräsentation der aufgespießten Strecken einer Stabber-Menge $L(\mathbf{q})$ kann berechnet werden durch die Verschmelzung der sortierten Listen von aufgespießten Strecken, die $A(\mathbf{q})$ referenziert.

2.2.4 Aufwandsabschätzung

Theorem 2.2 *Die Anfragezeit für die Nennung der Stabber-Mengen $L(\mathbf{q}')$ und die explizite Repräsentation $A(\mathbf{q}')$ der aufgespießten Strecken für eine Anfrage $[\mathbf{s}_1, \mathbf{s}_2]$, wobei $\mathbf{s}_1$ und $\mathbf{s}_2$ zwei Strecken sind, kann abgeschätzt werden durch*

$$Q(n) \quad = \quad O((t(n) + c(n)) \cdot K), \tag{2.12}$$

mit

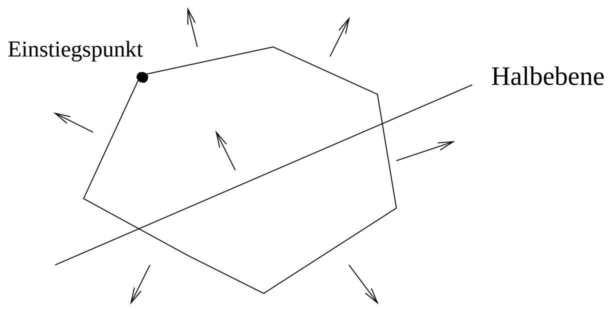
$$\begin{aligned} \alpha &= (1+\varepsilon)/2, \\ t(n) &= O(n^\alpha), \\ c(n) &= O(n^\alpha \log n). \end{aligned}$$

K ist die Anzahl der genannten Stabber-Mengen, $\varepsilon > 0$ eine Konstante, $t(n)$ die „worst case"-Zeit, die benötigt wird, um die hierarchische Partition zu traversieren, und $c(n)$ die Zeit, um eine bestimmte Stabber-Menge zu konstruieren.

Beweis: Für die folgenden Betrachtungen soll r ein genügend großer konstanter Wert sein. $g(r) = \Theta(r)$, $f(r) = O(r^{1-1/d})$, und $\rho(r) = 2/g(r)$.

Für jede Stabber-Menge $L(\mathbf{q}')$, die von dem Algorithmus ausgegeben wird, wird dieselbe Traversierung der hierarchischen Dekomposition durchgeführt wie für jede ihrer Geraden. Nach (2.7) ist die dafür benötigte Zeit $t(n) = O(n^\alpha)$, vorausgesetzt, daß die Entscheidung, welchem Pfad gefolgt wird, an jedem Knoten innerhalb derselben Zeit getroffen werden kann. Das ist unter Verwendung der Resultate der Berechnungen der Stabber-Mengen möglich.

Jede während der Traversierung auftretende Stabber-Menge $L(\mathbf{q}')$ wird aus der initialen Menge $L(\mathbf{q}) = L[\mathbf{s}_1, \mathbf{s}_2]$ durch den iterativen Schnitt ihrer entsprechenden konvexen Zelle (mit konstanter kombinatorischer Komplexität) mit konvexen Zellen des Dualraums berechnet. Jede konvexe Zelle entsteht aus dem Schnitt einer konstanten Anzahl (nur von r abhängig) von Halbräumen. Die Anzahl der konvexen Zellen, die geschnitten werden, ist höchstens so groß wie die Anzahl der besuchten Zellen und ist damit durch $O(n^\alpha)$ beschränkt. Daher ist die Anzahl der Halbräume, die geschnitten werden, auch $O(n^\alpha)$. Es kann ein Algorithmus gefunden werden für die inkrementelle Konstruktion des Schnittes von m Halbebenen in Zeit $O(m \log m)$, woraus sich $c(n) = O(n^\alpha \log n)$ ergibt. Die Idee für den inkrementellen Algorithmus



ist die Verwaltung der Menge N der Normalen der die Halbebenen begrenzenden Geraden, die schon abgearbeitet wurden, sortiert nach ihrem Winkel mit der x-Achse. Die Normale der nächsten zu bearbeitenden Halbebene wird in N eingefügt, und die benachbarten Normalen werden benutzt, um einen Einstiegspunkt in die konvexe Menge zu finden, von wo aus der Teil ihres Randes abgesucht werden kann, welcher von der neuen Halbebene abgeschnitten wird. Die Normalen der Strecken, die komplett abgeschnitten wurden, werden aus N entfernt. Das Einfügen einer Normalen kostet auf diese Weise $O(\log m)$ Zeit. Die Zeit, um den Rand abzusuchen und die Normalen zu entfernen, amortisiert über alle m Einfügeoperationen, ist $O(m)$. □

Theorem 2.3 *Der Zeitbedarf, um die I aufgespießten Strecken einer Stabber-Menge $L(\mathbf{q}')$ in Aufspießreihenfolge aufzuzählen, ist begrenzt durch*

$$s_r(n) \quad = \quad O(n^\alpha \log n + I). \tag{2.13}$$

Beweis: $A(\mathbf{q}')$ besteht aus $O(n^\alpha)$ sortierten Listen, die in dieser Zeit verschmolzen werden können. □

Ein schnellerer Algorithmus könnte durch Ausnutzung der Koherenz zwischen den Stabber-Klassen konstruiert werden. Die asymptotische Komplexität des dreidimensionalen Falles könnte wegen des teureren inkrementellen Halbebenenschnittes größer sein.

2.3 Berechnung der sichtbaren Umgebung für ein Element

2.3.1 Das Problem

Das Verfahren des vorigen Kapitels wird zur Lösung des folgenden Anfrageproblems verwendet:

Definition 2.11 (Sichtbarkeitsverhältnisse für eine Strecke)
Eingabe: *Eine Szene aus endlich vielen disjunkten Strecken in der Ebene.*
Ausgabe: *Die Sichtbarkeitsregionen* $V(\mathbf{s}_1, \mathbf{s}_2)$ *einer Anfragestrecke* $\mathbf{s}_1$ *und aller anderen Szenenstrecken* $\mathbf{s}_2$*, die von* $\mathbf{s}_1$ *aus sichtbar sind.* $\mathbf{s}_1$ *ist entweder identisch mit einer Szenenstrecke oder schneidet keine der Szenenstrecken.*

2.3.2 Lösung

Um die Sichtbarkeit bei einer Strecke $\mathbf{s}_1$ zu bestimmen, werden die Stabber-Untermengen der Menge $L[\mathbf{s}_1, \mathbf{s}_2]$ aller Geraden berechnet, die $\mathbf{s}_1$ und eine beliebige Szenenstrecke $\mathbf{s}_2$ schneiden. Mit dem Algorithmus *GenerateStabbers* des vorhergehenden Abschnitts werden die Stabber-Mengen von $L[\mathbf{s}_1, \mathbf{s}_2]$ bestimmt. Die Stabber-Mengen, in denen $\mathbf{s}_2$ mit $\mathbf{s}_1$ benachbart ist, d.h., in denen $\mathbf{s}_2$ von $\mathbf{s}_1$ aus sichtbar ist, werden als Teil der kompletten Lösung zur Lösung addiert. Die anderen werden in eine Menge E von noch nicht bearbeiteten Anfragen eingefügt. Die Gerade (Ebene im 3d-Fall), die $\mathbf{s}_2$ affin aufspannt, wird durch die beiden Endpunkte (die drei durch die Kanten des Dreiecks aufgespannten Geraden) in zwei Halbgeraden (sechs konvexe, offene Polygone) unterteilt. Die zwei Halbgeraden (sechs konvexen, offenen Polygone) werden als *Komplement* von $\mathbf{s}_2$ bezeichnet. Geraden, die nicht komplett im affinen Raum liegen, der von $\mathbf{s}_2$ aufgespannt wird, und die das Komplement von $\mathbf{s}_2$ schneiden, schneiden $\mathbf{s}_2$ nicht. Für jede Komponente $\mathbf{s}'_2$ des Komplements wird die Menge $L[\mathbf{s}_1, \mathbf{s}'_2]$ der Geraden, die sowohl $\mathbf{s}'_2$ als auch $\mathbf{s}_1$ schneiden, der Menge E als Anfrage hinzugefügt.

Im Gegensatz zu den bisherigen Gegebenheiten ist $\mathbf{s}'_2$ nun unbegrenzt. Weiterhin kann $\mathbf{s}'_2$ $\mathbf{s}_1$ und Strecken der gegebenen Szene schneiden. Diese Fälle können jedoch durch den Algorithmus *GenerateStabbers* behandelt werden.

Die Anfragen in E werden abgearbeitet, bis E leer ist. Wenn für eine Anfrage aus E die Menge der geschnittenen Szenenstrecken leer ist, wird diese Anfrage aus der Menge E gelöscht. Falls nicht, wird eine Szenenstrecke, die am nächsten bei $\mathbf{s}_1$ liegt, aus einer nichtleeren Stabber-Menge in E als neue Strecke $\mathbf{s}_2$ bestimmt.

Das Komplement von $\mathbf{s}_2$ wird wie oben beschrieben berechnet, und die Anfragen $L[\mathbf{s}_1, \mathbf{s}'_2]$ werden aus den Geraden, die $\mathbf{s}_1$ schneiden, und aus den Komponenten $\mathbf{s}'_2$ des Komplements von $\mathbf{s}_2$ konstruiert. Diese Anfragen werden mit den in E verbliebenen Anfragen geschnitten, was zu eingeschränkten Anfragen führt, die die Anfragen in E ersetzen. Weiterhin werden die Stabber-Untermengen von $L[\mathbf{s}_1, \mathbf{s}_2]$ bestimmt. Wiederum werden solche Stabber-Mengen, in denen $\mathbf{s}_2$ und $\mathbf{s}_1$ benachbart sind, d.h., in denen $\mathbf{s}_2$ von $\mathbf{s}_1$ aus sichtbar ist, als Teillösung zur Gesamtlösung addiert. Die anderen werden in die Menge E der noch nicht behandelten Anfragen eingefügt.

Algorithmus 2 zeigt den Pseudocode für die Berechnung der Sichtbarkeitsverhältnisse für eine Strecke.

Der Vorteil dieser Vorgehensweise ist, daß generell nicht alle Paare von $\mathbf{s}_1$ und Szenenstrecken auf der Sichtseite von $\mathbf{s}_1$ behandelt werden müssen. Es werden nur solche Strecken der Szene behandelt, die von $\mathbf{s}_1$ aus sichtbar sind, und die Komplemente der resultierenden Schnitte. Falls nicht viele Szenenstrecken von $\mathbf{s}_1$ aus sichtbar sind, kann eine geringe Anzahl von Schnitten erwartet werden. Eine worst-case Abschätzung der Anzahl der Schnitte bleibt offen.

2.4 Reduktion der Anzahl der Stabber-Mengen

Der bislang beschriebene Algorithmus hat den nachteiligen Effekt, daß eine große Anzahl Stabber-Mengen von Szenenstrecken erzeugt werden kann, die nicht zwischen $\mathbf{s}_1$ und $\mathbf{s}_2$ liegen, da solche Szenenstrecken auch Stabber-Mengen generieren. Diese Schwierigkeit kann für den Fall der Sichtbarkeitsberechnung zwischen zwei Strecken $\mathbf{s}_1$ und $\mathbf{s}_2$ vermindert werden, indem die Teile der Szene ausgeschlossen werden (*clipping*), die nicht die *Sichtbarkeitshülle* von $\mathbf{s}_1$ und $\mathbf{s}_2$ schneiden (in Abbildung 2.10 dünn gezeichnet). Die Sichtbarkeitshülle ist so definiert:

Definition 2.12 (Sichtbarkeitshülle)
Die Sichtbarkeitshülle $V^*(\mathbf{s}_1, \mathbf{s}_2)$ *zweier Strecken* $\mathbf{s}_1$ *und* $\mathbf{s}_2$ *ist die Menge aller Punkte, die auf einer gerichteten Geraden, die* $\mathbf{s}_1$ *und* $\mathbf{s}_2$ *schneidet, zwischen* $\mathbf{s}_1$ *und* $\mathbf{s}_2$ *liegen.*

Es könnte der Eindruck entstehen, daß die Einschränkung auf die Sichtbarkeitshülle den Test auf die Position von $\mathbf{s}_1$ und $\mathbf{s}_2$ in den Stabber-Klassen unnötig macht. Allerdings ist das nicht korrekt. Der Grund ist, daß es Szenenstrecken geben kann, die die Sichtbarkeitshülle schneiden, die auch außerhalb der Sichtbarkeitshülle von einer Geraden, die $\mathbf{s}_1$ und $\mathbf{s}_2$ schneidet, geschnitten werden.

Das Clipping wird durch folgendes Problem formalisiert:

Definition 2.13 (Konvexe Anfrage)
Eingabe: *Eine Szene aus endlich vielen disjunkten Strecken in der Ebene.*
Ausgabe: *Die Menge aller Szenenstrecken, die durch die Sichtbarkeitshülle von zwei nicht kolinearen Anfragestrecken* $\mathbf{s}_1$ *und* $\mathbf{s}_2$ *geschnitten werden.*

Algorithmus 2 Berechnung der Sichtbarkeitsverhältnisse für eine Strecke

Variablen: S: die gegebene Szene,
s: die Wurzel des hierarchischen Verdeckungsbaums zu S
Parameter: $\mathbf{s}_1$: eine Strecke der Szene S

VisibleSet($\mathbf{s}_1$)

```
Solution = {}
E = {}
s2 = SelectFrom(S)
for G in GenerateStabbers(L(s1, s2), s) do
  if s2 in NeighborOf(s1, G) then
    Solution = Solution ∪ {G}
  else
    E = E ∪ {G}
  end if
end for
for s2' in Complement(s2) do
  E = E ∪ {L(s1, s2')}
end for
while E not empty do
  G = SelectFrom(E)
  if NumberOfIntersectedObjects(G) = 0 then
    E = E - {G}
  else
    s2 = SelectFrom(NeighborOf(s1,SelectFrom(GenerateStabbers(G, s))))
    for s2' in Complement(s2) do
      for G' in GenerateStabbers(L(s1, s2'), s) do
        for G'' in E do
          E = E - {G''}
          E = E ∪ {IntersectionOf(G', G'')}
        end for
      end for
    end for
    for G' in GenerateStabbers(L(s1, s2), s) do
      if s2 in NeighborOf(s1, G') then
        Solution = Solution ∪ {G'}
      else
        E = E ∪ {G'}
      end if
    end for
  end if
end while
return Solution
```

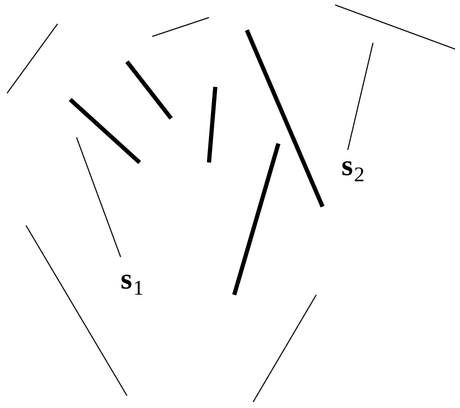


Abbildung 2.10: Strecken, die die Sichtbarkeitshülle von $\mathbf{s}_1$ und $\mathbf{s}_2$ schneiden (fett), bzw. nicht schneiden (dünn)

Das konvexe Anfrageproblem kann durch fortgesetzte Fallunterscheidung gelöst werden. Im zweidimensionalen Fall müssen die Strecken bestimmt werden, die die konvexe Hülle $\mathcal{V}$ der Anfangs- und Endpunkte von $\mathbf{s}_1$ und $\mathbf{s}_2$ schneiden. $\mathcal{V}$ hat dabei die Form eines Vierecks oder eines Dreiecks, je nach gegenseitiger Ausrichtung von $\mathbf{s}_1$ und $\mathbf{s}_2$ (siehe Abbildung 2.11). Seien $\mathbf{l}$ und $\mathbf{u}$ die beiden Seiten von $\mathcal{V}$, die nicht mit $\mathbf{s}_1$ und $\mathbf{s}_2$ identisch sind. Die Menge

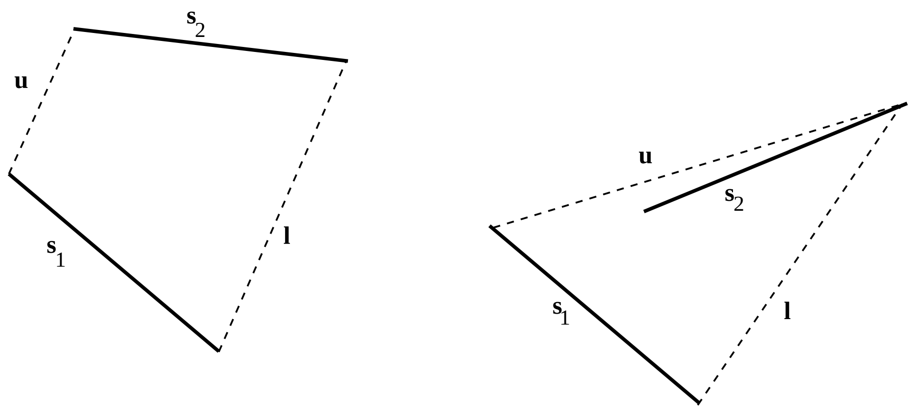


Abbildung 2.11: Zwei mögliche Konfigurationen der konvexen Hülle zweier Strecken in der Ebene

$I(\mathcal{V})$ aller Strecken $\overline{\mathbf{ab}}$, die $\mathcal{V}$ schneiden, kann als disjunkte Vereinigung von Mengen wie folgt repräsentiert werden:

$$\begin{aligned} I_1 &= \{\overline{\mathbf{ab}} \mid \overline{\mathbf{ab}} \cap \mathbf{l} \neq \emptyset\}; \\ I_2 &= \{\overline{\mathbf{ab}} \mid \overline{\mathbf{ab}} \cap \mathbf{u} \neq \emptyset\}; \\ I_{21} &= \overline{I_1} \cap \overline{I_2}; \end{aligned}$$

$$
\begin{aligned}
I_{22} &= \{\overline{\mathbf{ab}} \mid \mathbf{a} \in \mathcal{V}\}; \\
I_{23} &= \{\overline{\mathbf{ab}} \mid \mathbf{b} \in \mathcal{V}\}; \\
I(\mathcal{V}) &= I_1 \dot{\cup} (I_2 \cap \overline{I_1}) \dot{\cup} (I_{21} \cap I_{22} \cap I_{23})
\end{aligned}
$$

I_1 und I_2 können durch eine Anfrage an eine Menge von Strecken mit einer Anfragestrecke berechnet werden. Dieses kann auf ein Anfrageproblem mit einer Geraden und einer gegebenen Menge von Strecken und zwei Anfrageprobleme der Lokalisation eines Punktes bezüglich einer gegebenen Menge von Geraden reduziert werden. Diese Probleme können ebenfalls durch Halbraum-Dekomposition gelöst werden.

I_{22} und I_{23} resultieren aus einer Anfrage mit einer konvexen Menge an die gegebene Szene. Dieses Problem kann durch Halbraum-Dekomposition gelöst werden. Die anderen Mengen können durch Schnittbildung erzeugt werden, was auch durch Halbraum-Dekompostion implementiert werden kann.

Im dreidimensionalen Fall muß bestimmt werden, welche Dreiecke $\mathbf{t}$ von einem konvexen Anfrage-Polytop $\mathcal{V}$ geschnitten werden. $\mathcal{V}$ ist die Sichtbarkeitshülle der beiden Dreiecke $\mathbf{t}_1$ und $\mathbf{t}_2$ und hat daher konstante Größe. Seien $\mathbf{e}_i$, $i = 1, \ldots, m$, die Kanten von $\mathcal{V}$, die keine Kanten von $\mathbf{t}_1$ und $\mathbf{t}_2$ sind. Die $\mathbf{e}_i$ seien gerichtet von $\mathbf{t}_1$ nach $\mathbf{t}_2$. Seien $\mathbf{p}(\mathbf{t})$, $\mathbf{q}(\mathbf{t})$ und $\mathbf{r}(\mathbf{t})$ die Ecken eines gegebenen Dreiecks $\mathbf{t}$. Es können drei Schnittkonfigurationen auftreten, die getrennt behandelt werden:

Eine Kante $\mathbf{e}_i$ schneidet das Dreieck $\mathbf{t}$

Abbildung 2.12 zeigt eine solche Konfiguration. Das Anfragedreieck wird durch eine Kante einer Seitenfläche von $\mathcal{V}$, die keine Kante von $\mathbf{t_1}$ oder $\mathbf{t_2}$ ist, aufgespießt. Die

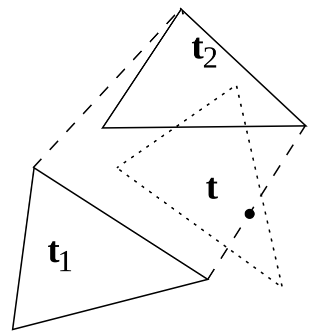


Abbildung 2.12: Konvexe-Hülle-Kanten-Anfrage an Dreiecke. Das Dreieck $\mathbf{t}$ schneidet eine Kante der Seitenfläche der konvexen Hülle, die durch $\mathbf{t_1}$ und $\mathbf{t_2}$ aufgespannt wird.

betreffenden Dreiecke $\mathbf{t}$ können durch eine Anfrage an eine Menge von gegebenen Dreiecken mit einer Strecke berechnet werden. Dieses Problem kann in ein Anfrageproblem mit einer Geraden an eine gegebene Menge von Dreiecken und zwei Anfrageprobleme der Lokalisation eines Punktes bezüglich einer gegebenen Menge von Ebenen zerlegt werden [deB93]. Dieses kann ebenfalls durch Halbraum-Dekomposition gelöst werden.

Eine oder mehrere Ecken von t liegen innerhalb von $\mathcal{V}$

Abbildung 2.13 zeigt eine solche Konfiguration. Die Ecke $\mathbf{p}(\mathbf{t})$ liegt innerhalb der konvexen Hülle von $\mathbf{t_1}$ und $\mathbf{t_2}$. Die betreffenden Dreiecke $\mathbf{t}$ resultieren aus einer

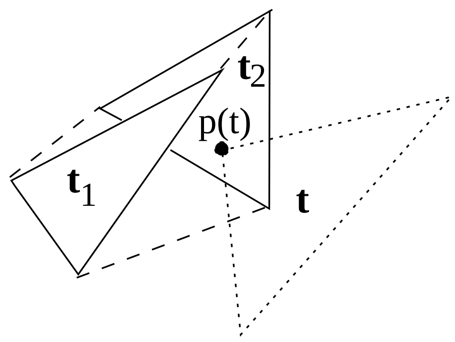


Abbildung 2.13: Konvexe-Hülle-Anfrage an Ecken. Die Ecke $\mathbf{p}(\mathbf{t})$ liegt innerhalb der konvexen Hülle von $\mathbf{t_1}$ und $\mathbf{t_2}$.

Anfrage mit einer konvexen Menge an die Menge von Dreieckseckpunkten. Dieses Problem kann durch Halbraum-Dekomposition gelöst werden.

Eine oder mehrere Kanten von t schneiden eine oder mehrere Seitenflächen von $\mathcal{V}$

Abbildung 2.14 zeigt eine solche Konfiguration. Eine Kante von $\mathbf{t}$ schneidet die Seitenflächen von $\mathcal{V}$ in zwei Punkten. Die betreffenden Dreiecke $\mathbf{t}$ können durch An-

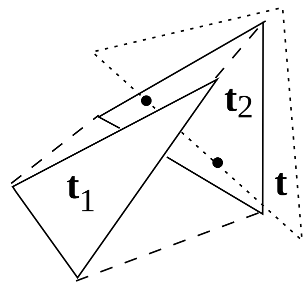


Abbildung 2.14: Seitenflächen-Anfrage an Dreieckskanten. Eine Kante von $\mathbf{t}$ schneidet die Seitenflächen von $\mathcal{V}$ in zwei Punkten.

fragen an die Menge aller gegebenen Dreieckskanten mit einer Seitenfläche berechnet werden. Dieses Problem kann in ein Anfrageproblem mit einem Dreieck an eine gegebene Menge von Geraden und zwei Anfrageprobleme der Lokalisation einer Ebene bezüglich einer gegebenen Menge von Punkten zerlegt werden. Die Geraden werden von den Dreieckskanten induziert. Die Punkte sind die Kantenendpunkte. Die Anfrage mit einem Dreieck kann durch eine Kombination von Halbebenenanfragen im Plückerraum gelöst werden [deB93].

Jede der konstant vielen Anfragen wird durch geschachtelte hierarchische Partitionen beantwortet (siehe Kapitel 2.2.2). Die Gesamtanfragezeit ist nach Formel 2.11 durch

$$Q_0(n) \quad = \quad O(n^{(1-1/d)(1+\varepsilon)})$$

beschränkt, für ein beliebiges, aber konstantes $\varepsilon > 0$ und d die größte auftretende Dimension, d.h. $d = 6$ für die Anfragen im Plückerraum.

Das Sichtbarkeitsproblem wird dadurch gelöst, daß über allen Teilszenen der Knoten der Suchbäume des Konvexe-Anfrage-Problems in einem Vorverarbeitungsschritt die Datenstruktur für *GenerateStabbers* aufgebaut wird. Für eine Anfrage des Sichtbarkeitsproblems wird zunächst die entsprechende konvexe Anfrage durchgeführt. Das Ergebnis setzt sich aus Teilszenen der genannten Art zusammen. Für jede dieser Teilszenen werden der Reihe nach Stabbermengen generiert. Dabei wird eine Stabbermenge, die sich als Ausgabe einer Teilszene der Folge ergibt, als Eingabe für die nächste Teilszene verwendet. Auf diese Weise ergibt sich am Ende eine Stabbermenge, die alle Dreiecke der Antwort auf die konvexe Anfrage berücksichtigt.

Es ist möglich, daß dabei ein Dreieck mehrmals auftritt, da die Antwortmengen auf die Teilanfragen, aus denen sich die konvexe Anfrage zusammensetzt, nicht notwendigerweise disjunkt sind. Da es sich jedoch um konstant viele Teilanfragen handelt, wird jedes Dreieck nicht mehr als konstant oft bearbeitet.

Kapitel 3

Näherungsweise Sichtberechnung durch hierarchische Verdeckungsbäume

In diesem Kapitel wird ein Verfahren zur näherungsweisen Beantwortung der Frage nach der Sichtbarkeitsregion zwischen zwei Anfrageelementen entwickelt. Kern der Lösung ist der sogenannte *hierarchische Verdeckungsbaum*. Die Approximation besteht darin, daß die Sichtbarkeitsregion nicht nur durch die frei sichtbaren und vollständig blockierten Teile beschrieben wird, sondern auch aus einer Teilregion besteht, für die die Sichtbarkeit ungeklärt ist. Dies ist ähnlich wie bei der Approximation geometrischer Szenen durch eine Raumunterteilung mit Octrees, bei der neben den Zellen, die vollständig im Inneren eines Elements liegen, und Zellen, die vollständig im freien Raum liegen, noch Zellen auftreten, die zwar ein Szenenelement schneiden, aber auch einen Teil des Freiraums enthalten. Eine Realisierungsmöglichkeit für Verdeckungsbäume, die im folgenden im wesentlichen genutzt wird, sind Zerlegungen in Octree-Art, wodurch sich diese Ähnlichkeit erklärt. Wie Octrees erlauben hierarchische Verdeckungsbäume einen Trade-off zwischen Speicherbedarf und Rechenzeit sowie Exaktheit der Beschreibung.

In Kapitel 3.1 wird die Repräsentation von Sichtbarkeit beschrieben, die dem hierarchischen Verdeckungsbaum zugrunde liegt. Diese basiert wie im vorigen Kapitel auf Geraden, die jedoch etwas anders dualisiert werden. In Kapitel 3.2 wird der Verdeckungsbaum als Vorstufe des hierarchischen Verdeckungsbaums eingeführt. Verdeckungsbäume beschreiben die Sichtbarkeitsverhältnisse in ganzen Szenen. Die in Kapitel 3.3 definierten hierarchischen Verdeckungsbäume enthalten darüber hinaus Information über die Sichtbarkeitsverhältnisse auch in Unterszenen. Die Berechnung der Sichtbarkeitsregion für eine Anfrage wird durch Zusammensetzen aus den Sichtbarkeitsregionen solcher Unterszenen erreicht, die für die Sichtbarkeitsverhältnisse der Anfrage relevant sind.

3.1 Repräsentation von Sichtbarkeit

Die folgenden Ausführungen konzentrieren sich im wesentlichen auf den zweidimensionalen Fall. Eine Übertragung auf dreidimensionale Gegebenheiten ist meist auf direkte Weise möglich, so daß diese Vorgehensweise keine besondere Einschränkung darstellt.

Zur Vereinfachung der Beschreibung wird ohne Beschränkung der Allgemeinheit angenommen, daß die gegebene ebene Szene aus Szenenelementen besteht, die vollständig im Einheitsquadrat liegen. Die Beschreibung der Sichtbarkeit geschieht über Geraden, die durch sogenannte *Randkoordinaten* repräsentiert werden, die für beliebige Dimensionen so definiert sind:

Definition 3.1 (Randkoordinaten (R-Koordinaten))
Gegeben seien zwei voneinander verschiedene der $(d-1)$-dimensionalen abgeschlossenen Seitenflächen F_1, F_2 des Einheitswürfels im d-dimensionalen Raum, jeweils in einer $(d-1)$-dimensionalen Parametrisierung über dem $(d-1)$-dimensionalen Einheitswürfel. Ferner sei g eine Gerade, die F_1 und F_2 schneidet und im Inneren des Würfels verläuft. Das $2(d-1)$-Tupel, dessen erste $(d-1)$ Komponenten von Parametern des Schnittpunkts mit F_1 und dessen zweite $(d-1)$ Komponenten von den Parametern des Schnittpunkts von F_2 gebildet werden, heißt Randkoordinate (R-Koordinate) *der Geraden g bezüglich F_1 und F_2.*

*Die Darstellung von Geraden durch R-Koordinaten wird als R-*Dualisierung *bezeichnet.*

Alle Geraden, die das Innere des d-dimensionalen Einheitswürfels schneiden, schneiden mindestens zwei der abgeschlossenen Seitenflächen F_i und F_j des Würfels, $i, j = 1, \ldots, 2d$. Die Menge aller dieser Geraden läßt sich also als Vereinigung der Mengen $R(i,j)$, $i, j = 1, \ldots, 2d$, $i \neq j$, darstellen, die aus den Geraden bestehen, die F_i und F_j in genau zwei unterschiedlichen Punkten schneiden. Dabei kann eine Gerade in mehreren dieser Mengen auftreten, nämlich dann, wenn einer der Schnittpunkte auf dem Rand einer der Seitenflächen F_i liegt. Eine disjunkte Darstellung könnte erreicht werden, indem offene Seitenflächen des Würfels beliebiger Dimension herangezogen würden. Da dies zu einer größeren Zahl von Paaren führt, wird aus Effizienzgründen die Mehrdeutigkeit in Kauf genommen, die für das folgende irrelevant ist. Für $d = 2$ ist die R-Dualisierung zweidimensional, es sind $\binom{2d}{2} = \binom{4}{2} = 6$ Mengen $R(i,j)$ zu betrachten. Für $d = 3$ ist die Dualisierung vierdimensional, die Anzahl der Mengen $R(i,j)$ ist $\binom{6}{2} = 15$.

Der grundlegende Ansatz dieses Kapitels ist, die gegebene Szene S aus disjunkten, konvexen Elementen in disjunkte Unterszenen S_j, $j = 1, \ldots, m$, zu zerlegen. Für jede Unterszene wird ihre Sichtbarkeitsregion $V(S_j)$ bestimmt, die so definiert ist:

Definition 3.2 (Sichtbarkeitsregion einer Szene)
Sei S eine im Einheitsquadrat liegende Szene. Die Menge aller Geraden $V(S)$, die das Einheitsquadrat, aber nicht die Szene S schneiden, wird als Sichtbarkeitsregion *von S bezeichnet.*

Die Parametrisierung der Geraden geschieht in R-Koordinaten. Das bedeutet, daß Operationen zwischen Mengen $V(S_i)$ und $V(S_j)$ als Punktmengenoperationen in der R-Dualisierung durchgeführt werden können.

Seien $\mathbf{q}_1$ und $\mathbf{q}_2$ zwei disjunkte Anfragemengen, die entweder Szenenelemente sind oder keines der Szenenelemente schneiden.

Definition 3.3 (Sichtbarkeitshülle und Sichtbarkeitsregion zwischen Anfragemengen)
Seien $\mathbf{q}_1$ und $\mathbf{q}_2$ zwei Anfragemengen, S eine im Einheitsquadrat liegende Szene, die in Unterszenen S_j, $j \in J$, zerlegt ist.

Die Sichtbarkeitshülle $V^*(\mathbf{q}_1, \mathbf{q}_2)$ *von $\mathbf{q}_1$ und $\mathbf{q}_2$ ist die Menge aller Punkte, die auf einer Geraden, die $\mathbf{q}_1$ und $\mathbf{q}_2$ schneidet, zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ liegen. Die* Sichtbarkeitshülle $V^*(S_j)$ *einer Unterszene S_j ist die Menge aller Punkte, die auf Geraden, die Elemente in S_j schneiden, zwischen zwei Schnittpunkten mit Szenenelementen liegen.*

Die Sichtbarkeitsregion $V(\mathbf{q}_1, \mathbf{q}_2)$ *von $\mathbf{q}_1$ und $\mathbf{q}_2$ ist die Menge aller Geraden, die $\mathbf{q}_1$ und $\mathbf{q}_2$ schneiden und keinen Schnittpunkt mit der gegebenen Szene S gemeinsam haben, der auf dem Geradenabschnitt zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ liegt.*

Sei $R(\mathbf{q}_1, \mathbf{q}_2)$ die Menge der Geraden, die $\mathbf{q}_1$ und $\mathbf{q}_2$ schneiden. Eine Unterszene S_j liegt zwischen $\mathbf{q}_1$ *und* $\mathbf{q}_2$*, falls alle Schnittpunkte von Geraden aus $R(\mathbf{q}_1, \mathbf{q}_2)$ mit $V^*(S_j)$ in $V^*(\mathbf{q}_1, \mathbf{q}_2)$ liegen.*

Mit diesen Definitionen läßt sich die Sichtbarkeitsregion $V(\mathbf{q}_1, \mathbf{q}_2)$ so darstellen:

Theorem 3.1 *Seien S_j, $j \in J$, die Unterszenen, die die Sichtbarkeitshülle von $\mathbf{q}_1$ und $\mathbf{q}_2$ schneiden. Sei $J_1 \subseteq J$ die Menge der Indizes der Unterszenen, die zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ liegen. Dann läßt sich die Sichtbarkeitsregion $V(\mathbf{q}_1, \mathbf{q}_2)$ von $\mathbf{q}_1$ und $\mathbf{q}_2$ darstellen durch*

$$V(\mathbf{q}_1, \mathbf{q}_2) = R(\mathbf{q}_1, \mathbf{q}_2) \cap \bigcap_{j \in J_1} V(S_j) \setminus \bigcup_{j \in J \setminus J_1} \{\mathbf{l} \mid \mathbf{l} \cap \mathbf{s} \cap V^*(\mathbf{q}_1, \mathbf{q}_2) \neq \emptyset,\ \mathbf{s} \in S_j\}.$$

Beweis: Zur Konstruktion der Sichtbarkeitsregion zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ muß zunächst die Menge $R(\mathbf{q}_1, \mathbf{q}_2)$ der Geraden bestimmt werden, die beide Anfragemengen schneidet. Diese Menge wird dann auf die Geraden eingeschränkt, die alle diejenigen Unterszenen nicht schneiden, die zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ liegen:

$$R(\mathbf{q}_1, \mathbf{q}_2) \cap \bigcap_{j \in J_1} V(S_j).$$

Von diesen Geraden müssen solche eliminiert werden, die Elemente $\mathbf{s}$ von Unterszenen S_j, $j \in J_2 := J \setminus J_1$, in mindestens einem Punkt schneiden, der zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ liegt, was zum abschließenden Ergebnis

$$V(\mathbf{q}_1, \mathbf{q}_2) = R(\mathbf{q}_1, \mathbf{q}_2) \cap \bigcap_{j \in J_1} V(S_j) \setminus \bigcup_{j \in J_2} \{\mathbf{l} \mid \mathbf{l} \cap \mathbf{s} \cap V^*(\mathbf{q}_1, \mathbf{q}_2) \neq \emptyset,\ \mathbf{s} \in S_j\}$$

führt. $\square$

Aus Sicht der Verdeckung stellt sich die Situation so dar:

Definition 3.4 (Verdeckungsregion zwischen Anfragemengen)
Seien $\mathbf{q}_1$ und $\mathbf{q}_2$ zwei Anfragemengen, S eine im Einheitsquadrat liegende Szene. Die Menge aller Geraden $B(S)$, die das Einheitsquadrat und die Szene S schneiden, wird als Verdeckungsregion *von S bezeichnet. Die* Verdeckungsregion *$B(\mathbf{q}_1, \mathbf{q}_2)$ von $\mathbf{q}_1$ und $\mathbf{q}_2$ ist die Menge aller Geraden, die $\mathbf{q}_1$ und $\mathbf{q}_2$ schneiden und einen Schnittpunkt mit der gegebenen Szene S gemeinsam haben, der auf dem Geradenabschnitt zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ liegt.*

Für die Verdeckungsregion $B(\mathbf{q}_1, \mathbf{q}_2)$ gilt dann:

Theorem 3.2 *Sei S_j, $j \in J$, die Menge der Unterszenen, die die Sichtbarkeitshülle von $\mathbf{q}_1$ und $\mathbf{q}_2$ schneiden. Sei $J_1 \subseteq J$ die Menge der Indizes der Unterszenen, die zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ liegen. Dann läßt sich die Verdeckungsregion $B(\mathbf{q}_1, \mathbf{q}_2)$ von $\mathbf{q}_1$ und $\mathbf{q}_2$ darstellen durch*

$$B(\mathbf{q}_1, \mathbf{q}_2) = R(\mathbf{q}_1, \mathbf{q}_2) \cap (\bigcup_{j \in J_1} B(S_j) \cup \bigcup_{j \in J \setminus J_1} \{\mathbf{l} \mid \mathbf{l} \cap \mathbf{s} \cap V^*(\mathbf{q}_1, \mathbf{q}_2) \neq \emptyset,\ \mathbf{s} \in S_j\}).$$

Beweis: Zur Berechnung der Verdeckungsregion zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ muß zunächst die Menge $R(\mathbf{q}_1, \mathbf{q}_2)$ der Geraden bestimmt werden, die beide Anfragemengen schneidet. Diese Menge wird dann auf die Geraden eingeschränkt, die eine der Unterszenen schneiden, die zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ liegen:

$$R(\mathbf{q}_1, \mathbf{q}_2) \cap (\bigcup_{j \in J_1} B(S_j)).$$

Zu diesen Geraden kommen noch diejenigen hinzu, die Elemente $\mathbf{s}$ von Unterszenen S_j, $j \in J_2 := J - J_1$, in mindestens einem Punkt schneiden, der zwischen $\mathbf{q}_1$ und $\mathbf{q}_2$ liegt, was zum abschließenden Ergebnis

$$B(\mathbf{q}_1, \mathbf{q}_2) = R(\mathbf{q}_1, \mathbf{q}_2) \cap (\bigcup_{j \in J_1} B(S_j) \cup \bigcup_{j \in J_2} \{\mathbf{l} \mid \mathbf{l} \cap \mathbf{s} \cap V^*(\mathbf{q}_1, \mathbf{q}_2) \neq \emptyset,\ \mathbf{s} \in S_j\})$$

führt. $\square$

Das Auswerteprinzip, das diesem Satz zugrunde liegt, kann auf unterschiedliche Art und Weise implementiert werden. Im folgenden werden *Quadtrees* [Sam84, Sam89b, Sam89a] als Grundlage benutzt. Dies führt zwar nur zu einer approximativen Lösung, dafür ist die Implementierung relativ einfach und es werden numerische Instabilitäten vermieden, wie sie bei exakter geometrischer Berechnung auftreten können.

Zunächst wird das Konzept der *Verdeckungsbäume* beschrieben, das zur approximativen Repräsentation der Mengen $V(S_i)$ genutzt wird. Danach werden die Verdeckungsbäume benutzt, um Anfragen nach Sichtbarkeitsmengen zu beantworten. Das führt zu den *hierarchischen Verdeckungsbäumen*, die aus einer hierarchischen Dekomposition der gegebenen Szene resultieren. Für diesen Zweck werden ebenfalls Quadtrees benutzt.

3.2 Verdeckungsbäume

Wie im vorigen Kapitel dargestellt, läßt sich die Menge der Geraden, die das Einheitsquadrat durchlaufen, als Vereinigung von Mengen $R(i,j)$, $i,j = 1,\ldots,4$, darstellen, die alle Geraden enthalten, die zwei Kanten E_i und E_j des Einheitsquadrats schneiden. Durch Parametrisierung einer solchen Kante über dem Einheitsintervall $[0,1]$, wie sie bei den R-Koordinaten von Geraden vorausgesetzt wird, werden die Geraden in $R(i,j)$ eindeutig auf die Punkte eines *dualen Einheitsquadrats* abgebildet. Eine natürliche Parametrisierung einer Kante ergibt sich durch Streichen derjenigen Koordinate ihrer Punkte, die für die ganze Kante identisch ist. In Abbildung 3.1 a) sind zwei Kanten eines Quadrates fett hervorgehoben. Jede Gerade, die beide Kanten des Quadrats schneidet, korrespondiert mit einem Punkt im Quadrat des dualen Koordinatensystems, siehe Abbildung 3.1 b), und umgekehrt, durch die Repräsentation der beiden Kanten auf den Achsen eines zweidimensionalen dualen Koordinatensystems. Dieses wird in der Abbildung rechts dargestellt. Einer Menge $R(i,j)$ wird nun auf folgende Weise ein Verdeckungsbaum zugeordnet:

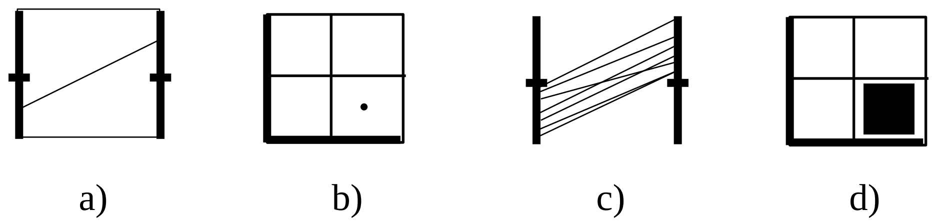


Abbildung 3.1: Dualisierung der Geraden. a) Strecke vom linken zum rechten Rand der Gesamt-boundingbox. b) Dieselbe Strecke als Punkt im Dualraum. c) Menge aller Strecken von unterer Hälfte linker Rand zu oberer Hälfte rechter Rand. d) Dieselbe Menge als rechteckiges Gebiet im Dualraum. Schwarze Flächen entsprechen blockierten Geraden, weiße Flächen entsprechen nicht blockierten Geraden.

Definition 3.5 (Verdeckungsbaum)
Sei S eine zweidimensionale Szene im Einheitsquadrat, $R(i,j)$, $i,j = 1,\ldots,6$, die von zwei Kanten E_i, E_j aufgespannte Menge von Geraden. Der Verdeckungsbaum *von S bezüglich $R(i,j)$ ist ein Quadtree, d.h. ein Baum mit vier Nachfolgern für jeden inneren Knoten.*

Die Wurzel des Verdeckungsbaums repräsentiert das duale Quadrat der gegebenen Szene. Alle anderen inneren Knoten repräsentieren Unterquadrate des dualen Quadrats. Das Unterquadrat eines Knotens ist eins von vier kongruenten Unterquadraten des Quadrates, das deren Vater zugeordnet ist.

Die Blätter des Verdeckungsbaums sind mit einer von drei verschiedenen Typen von Sichtbarkeitsinformation beschriftet.

transparent: *Keine der Geraden, die durch das Quadrat dieses Blattes repräsentiert werden, schneidet ein Element der Szene.*

blockiert: *Es existiert eine Unterteilung des Quadrates dieses Blattes in vier kongruente Unterquadrate, so daß mindestens ein Element der Szene von allen Geraden geschnitten wird, die durch das Unterquadrat repräsentiert werden.*

partiell blockiert: *Sonst.*

Ein Verdeckungsbaum wird gerundet *genannt, falls seine Blätter entweder transparent oder blockiert sind.*

Eine mögliche Erweiterung des Verdeckungsbaumes wäre es, die *partiell blockierten* Blätter durch Verweise zu ergänzen, die auf die Elemente der Szene zeigen, die für die partielle Blockierung verantwortlich sind. Bei Bedarf kann so mit diesen Elementen eine exakte Berechnung der Sichtbarkeit vorgenommen werden.

Die Abbildungen 3.1 c) und d) zeigen die Korrespondenz zwischen Punkten eines Unterquadrats (schwarz gefüllt) des dualen Quadrats und Geraden im Szenenraum. Um die gesamte Sichtbarkeit einer Szene zu speichern, sind sechs Verdeckungsbäume nötig, einer für jedes Paar von Quadratkanten.

3.2.1 Algorithmische Konstruktion eines Verdeckungsbaums

Der hier beschriebene Algorithmus konstruiert die Verdeckungsbäume einer Szene durch die Berechnung eines Verdeckungsbaumes für jedes Element der Szene und die anschließende Kombination dieser Bäume in einen Verdeckungsbaum für die ganze Szene.

Für ein gegebenes Paar von Kanten E_i, E_j des umschreibenden Einheitsquadrats testet der Algorithmus, ob sie wechselseitig komplett sichtbar sind oder nicht, d. h. daß jeder Punkt der einen Kante jeden Punkt der anderen Kante sieht. Falls die Kanten nicht komplett sichtbar sind, werden sie jeweils in zwei Kanten gleicher Länge unterteilt. Falls sich die Kanten komplett sehen oder komplett nicht sehen, stoppt der Algorithmus. Falls die Kantenlänge während der Konstruktion eine vorgegebene Schranke ε unterschreitet, stoppt der Algorithmus ebenfalls. In einem solchen Fall wird das Blatt als partiell blockiert gekennzeichnet.

Abbildung 3.2 zeigt die Konstruktion eines Verdeckungsbaums für eine Strecke als Szenenelement. Algorithmus 3 zeigt den Pseudocode für die Konstruktion.

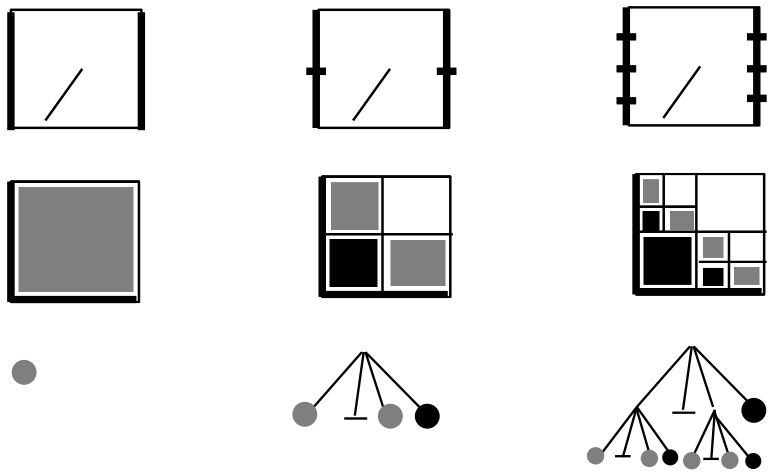


Abbildung 3.2: Konstruktion eines Verdeckungsbaums für eine Strecke, Unterteilung der linken und rechten Kante des umschreibenden Einheitsquadrats. Oben: Szene mit einer Strecke. Mitte: Dualraum mit markierten Regionen. Unten: Baumstruktur. Schwarze Flächen entsprechen blockierten Geraden, graue Flächen entsprechen partiell blockierten Geraden und weiße Flächen entsprechen nicht blockierten Geraden.

Die Vereinigungsoperation zweier Verdeckungsbäume wird, wie später noch die Durchschnitts- und Differenzoperation, über einen simultanen Durchlauf beider zu verknüpfender Bäume definiert. Hierfür werden beide Bäume implizit auf dieselbe Form gebracht. Dies geschieht durch die Expansion von Blättern in dem Fall, daß zu einem Blatt in einem der Bäume ein innerer Knoten im anderen Baum korrespondiert. Das Blatt wird dann durch einen Teilbaum ersetzt, dessen Form mit dem Teilbaum übereinstimmt, dessen Wurzel der erwähnte innere Knoten ist. Die Blätter des neuen Teilbaums erhalten die Markierung des entfernten Blatts.

Definition 3.6 (Vereinigung von Verdeckungsbäumen)
Die Vereinigung $T_1 \cup T_2$ *von Verdeckungsbaum* T_1 *mit* T_2 *wird mittels eines simultanen Durchlaufs durch beide Bäume definiert. Seien* t_1 *und* t_2 *zwei korrespondierende Knoten der Bäume* T_1 *und* T_2*. Abhängig von der Sichtbarkeitsinformation der Knoten wird der Vereinigungsbaum wie folgt ergänzt, wobei innere Knoten als partiell blockiert behandelt werden:*

t_1 **oder** t_2 **blockiert:** *Füge einen blockierten Knoten in den resultierenden Baum ein.*

Algorithmus 3 Algorithmus zur Konstruktion des Verdeckungsbaums einer Strecke.

Parameter: Kanten $\mathbf{s}_1$ und $\mathbf{s}_2$ der boundingbox, verdeckende Strecke **occc**, zu generierender Verdeckungsbaum **qtree**.

MakeBlockerQuadtree(Segment $\mathbf{s}_1$, Segment $\mathbf{s}_2$, Segment occ, BlockerQuadtree qtree)

```
if s1 and s2 are completely mutually visible then
  qtree does not block visibility
  return
end if
if s1 and s2 are completely mutually not visible then
  qtree blocks visibility
  return
end if
if s1 and s2 are long enough (≥ ε) then
  assign qtree to new BlockerQuadtree node;
  qtree is partially blocking;
  half s1 into s11, s12
  half s2 into s21, s22
  MakeBlockerQuadtree(s12, s21, occ, qtree.NW);
  MakeBlockerQuadtree(s12, s22, occ, qtree.NE);
  MakeBlockerQuadtree(s11, s22, occ, qtree.SE);
  MakeBlockerQuadtree(s11, s21, occ, qtree.SW);
  if all sons of qtree are tagged partially blocking then
    assign qtree tag of sons
    delete sons
    return
  end if
else
  qtree is partially blocking
  return
end if
```

t_1 **und** t_2 **transparent:** *Füge einen transparenten Knoten in den resultierenden Baum ein.*

t_1 **oder** t_2 **partiell blockiert:** *Falls* t_1 *oder* t_2 *blockiert ist, dann gilt Fall 1. Ansonsten: Wenn ein Knoten der Wurzelknoten eines Unterbaumes ist, klassifiziere die nachfolgenden Knoten rekursiv. Füge einen inneren Knoten ein, der Wurzelknoten der klassifizierten Nachfolgeknoten ist.*

Falls t_1 *und* t_2 *Blätter sind, füge ein partiell blockiertes Blatt in den resultierenden Baum ein.*

Abbildung 3.3, oben, zeigt ein Beispiel für eine Vereinigung zweier Verdeckungsbäume. Algorithmus 4 zeigt den Pseudocode der Vereinigungsoperation. Abbildung 3.4 zeigt eine

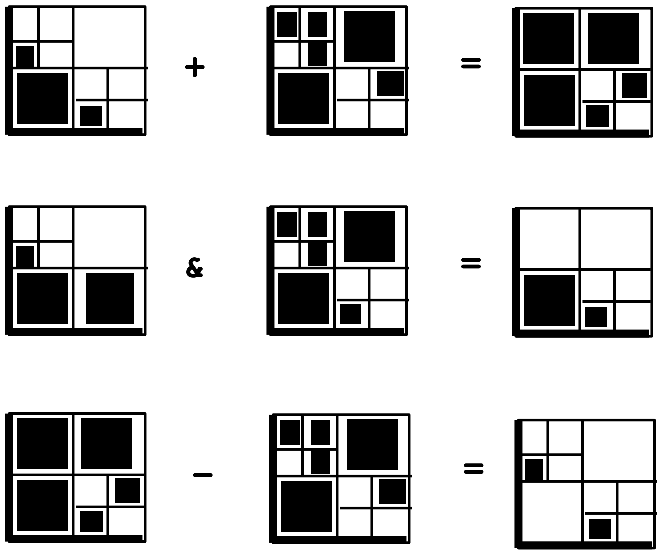


Abbildung 3.3: Operationen auf Verdeckungsbäumen: Vereinigung (oben), Durchschnitt (Mitte) und Differenz (unten). Schwarze Flächen entsprechen blockierten Geraden, weiße Flächen entsprechen nicht blockierten Geraden.

Szene aus Strecken und den zugehörigen Verdeckungsbaum für die Strecken, die die linke und die untere Kante des Quadrats schneiden.

3.2.2 Runden in Verdeckungsbäumen

Das Ziel des Rundens in Verdeckungsbäumen ist die Ersetzung der partiell blockierten Blätter durch transparente oder blockierte Blätter.

Algorithmus 4 Algorithmus zur Vereinigung zweier Verdeckungsbäume

Parameter: Verdeckungsbaum $\mathbf{t}_1$, Verdeckungsbaum $\mathbf{t}_2$, Verdeckungsbaum $\mathbf{t}$: Vereinigung von $\mathbf{t}_1$ und $\mathbf{t}_2$.
Rückgabe: Vereinigung $\mathbf{t}$ der Verdeckungsbäume $\mathbf{t}_1$ und $\mathbf{t}_2$.

UnionOfBlockerQuadtrees(BlockerQuadtree $\mathbf{t}_1$, BlockerQuadtree $\mathbf{t}_2$, BlockerQuadtree t)

```
if (t1 = NULL) or (t2 = NULL) then
  t=NULL;
  return;
end if
assign t to new BlockerQuadtree node;
if t1 or t2 blocking then
  t is blocking;
  return;
end if
if t1 and t2 transparent then
  t is transparent;
  return;
end if
if t1 or t2 partially blocking then
  if t1 and t2 leaves then
    t is partially blocking
    return
  else
    t is partially blocking;
    UnionOfBlockerQuadtrees(t1.NW, t2.NW, t.NW);
    UnionOfBlockerQuadtrees(t1.NE, t2.NE, t.NE);
    UnionOfBlockerQuadtrees(t1.SE, t2.SE, t.SE);
    UnionOfBlockerQuadtrees(t1.SW, t2.SW, t.SW);
    return;
  end if
end if
```

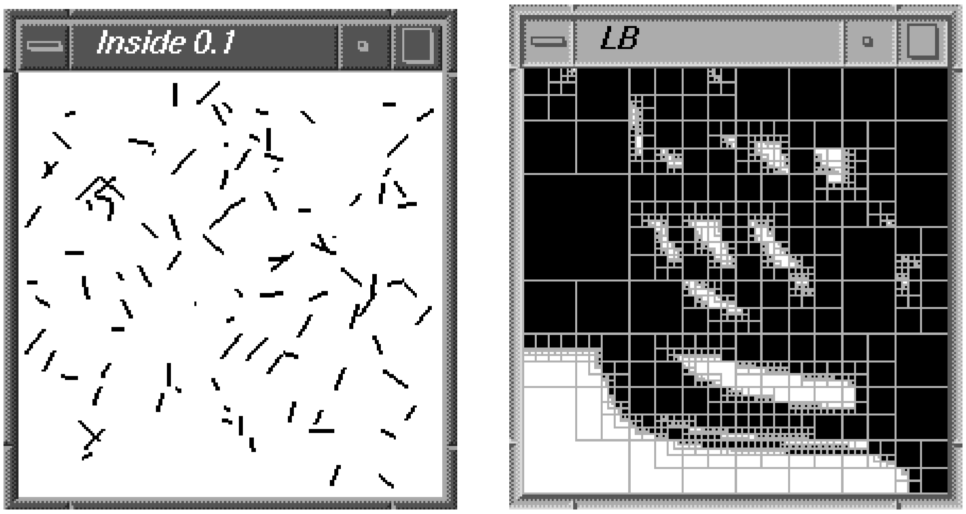


Abbildung 3.4: Eine Szene mit 100 Strecken und ihr Verdeckungsbaum für die Geraden, die die linke und die untere Kante schneiden. Schwarze Flächen entsprechen blockierten Geraden, weiße Flächen entsprechen nicht blockierten Geraden.

Definition 3.7 (Runden in Verdeckungsbäumen)
Das Ersetzen von partiell blockierten Blättern eines Verdeckungsbaums durch blockierte wird als Aufrunden *bezeichnet.* Abrunden *bedeutet, die partiell blockierten Blätter eines Verdeckungsbaums durch transparente Blätter zu ersetzen. Falls es beim Auf– oder Abrunden passiert, daß alle vier Blätter eines gemeinsamen Vorgängers entweder transparent oder blockiert werden, wird der gemeinsame Vorgänger als transparent oder blockiert gekennzeichnet und die vier Blätter werden gelöscht.*

Abbildung 3.5, oben, zeigt einen vergrößerten Ausschnitt eines Verdeckungsbaums, der aufgerundet wurde. Zum Vergleich zeigt dieselbe Abbildung im unteren Teil das Ergebnis einer Abrundung.

Experimente mit Rundungsoperationen haben den interessanten Effekt gezeigt, daß das Aufrunden den Rand der blockierten Bereiche üblicherweise mit weniger Quadtree-Zellen approximiert als das Abrunden. Aufgrund der Tatsache, daß Randzellen wesentlich zum Speicherbedarf des Verdeckungsbaums beitragen, bedeutet Aufrunden üblicherweise weniger Speicherbedarf als Abrunden.

Experiment 3.1 (Messung des Speicherbedarfs von Verdeckungsbäumen für Auf– und Abrunden)

Eingabe: *Zufällig generierte Strecken, wie in Abbildung 3.4, links.*

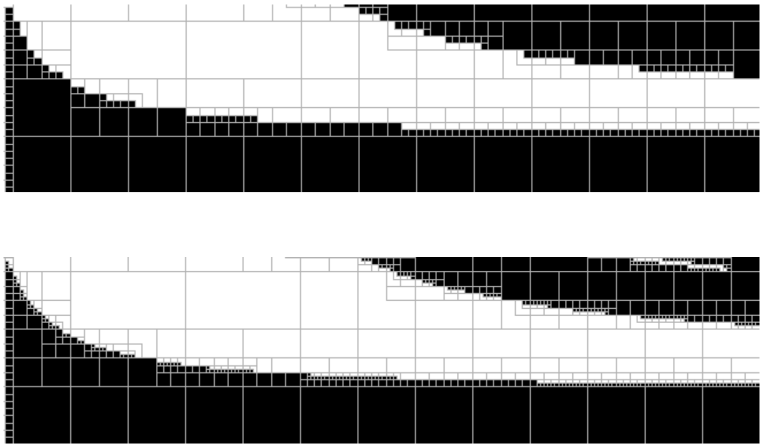


Abbildung 3.5: Vergleich zwischen dem Aufrunden (oben) und dem Abrunden (unten) von partiell blockierten Blättern. Schwarze Flächen entsprechen blockierten Geraden, weiße Flächen entsprechen nicht blockierten Geraden.

Parameter: *Auf– oder Abrunden.*

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Speicherbedarf des Verdeckungsbaumes in Abhängigkeit von der Anzahl der Originalstrecken für Auf– und Abrunden.*

Beobachtung: *Vgl. Abbildung 3.6. Der qualitative Verlauf der beiden Kurven ist für beide gemessenen Rundungsverfahren ähnlich. Die Kurve für das Aufrunden verläuft jedoch ab ca. 20 Strecken bis ca. 300 Strecken deutlich unter der Kurve für das Abrunden.*

Eine weitere Beobachtung deutet darauf hin, daß Aufrunden die bessere Lösung sein könnte. Wenn Unterszenen durch eine lange vertikale Strecke getrennt werden, beispielsweise durch eine Wand, dann wird diese Strecke sehr wahrscheinlich beim Aufbau der Teilszenenhierarchien (s.u.) in Teilstrecken unterteilt. Diese Teilstrecken werden in separaten Quadranten des Quadtrees gespeichert. Während der Kombination von Verdeckungsbäumen, die unterschiedlichen Quadranten zugeordnet sind, kann es aufgrund der Diskretisierung passieren, daß die Originalstrecke mit kleinen Unterbrechungen repräsentiert wird und damit nicht mehr komplett die Unterszenen blockiert. Dieser Effekt wird durch das Aufrunden gemindert.

Manchmal macht es Sinn, in Abhängigkeit von der lokalen Situation auf- oder abzurunden. Ein Beispiel wird in Abbildung 3.7 gezeigt. Falls ein Blatt aufgerundet wurde und alle anderen Brüder transparent sind, dann wäre es aus Speichereffizienzgründen besser, das Blatt abzurunden. Dann könnte der Vater der vier Knoten als transparent gekennzeichnet werden und die vier Söhne könnten gelöscht werden.

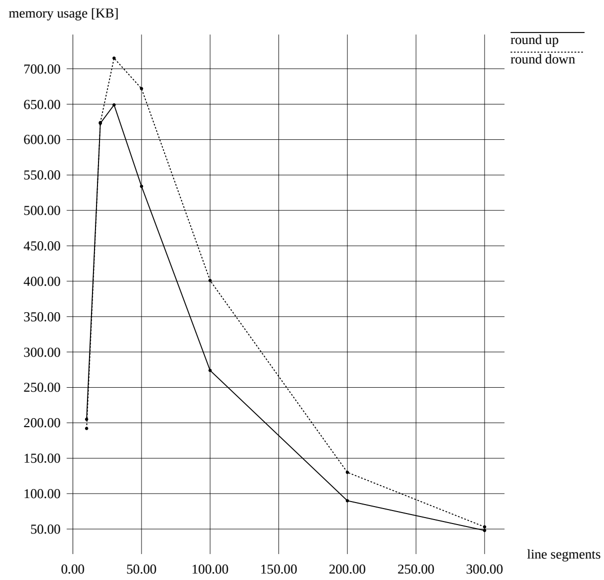


Abbildung 3.6: Speicherbedarf von Verdeckungsbäumen (y-Achse) in Abhängigkeit von der Anzahl der Strecken (x-Achse) für Auf– und Abrunden

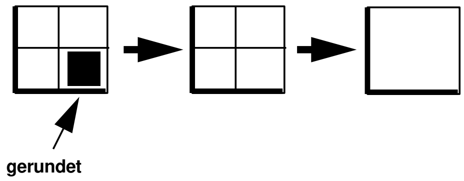


Abbildung 3.7: Lokale Wahl der Rundungsoperation. Schwarze Flächen entsprechen blockierten Geraden, weiße Flächen entsprechen nicht blockierten Geraden.

3.2.3 Anfragen an Verdeckungsbäume

Verdeckungsbäume werden nun genutzt, um für eine gegebene Menge von Sichtgeraden die Untermenge der Geraden zu bestimmen, die von der Szene blockiert werden. Die Menge der Sichtgeraden ist in Form eines Verdeckungsbaums T_Q gegeben. Die Anfragemenge besteht aus solchen Geraden, die als „blockiert" repräsentiert sind. Der Grund für diese etwas ungewöhnliche Spezifikation der Anfragemenge wird später klar.

Definition 3.8 (Anfrage nach Blockiermengen)
Gegeben: *Ein Verdeckungsbaum T_S der gegebenen Szene, ein Anfragebaum T_Q über demselben Kantenpaar von T_S.*
Gesucht: *Der* Schnitt *von T_S und T_Q.*

Die Schnittoperation von Verdeckungsbäumen ist dabei so definiert:

Definition 3.9 (Schnitt von Verdeckungsbäumen)
Der Schnitt *von Verdeckungsbaum T_1 mit T_2, $T_1 \cap T_2$, wird durch den simultanen, rekursiven Durchlauf der beiden Bäume erzeugt. Seien t_1 und t_2 die beiden korrespondierenden Knoten der Bäume T_1 und T_2. Abhängig vom Typ der Sichtbarkeit dieser Knoten wird der Schnittbaum wie folgt definiert, wobei wiederum die inneren Knoten als partiell blockiert behandelt werden.*

t_1 **xor** t_2 **blockiert:** *Füge einen Knoten in den resultierenden Baum ein und hänge den Teilbaum mit der Wurzel des nicht blockierten Baums an diesen Knoten, falls es einen gibt. Andernfalls markiere den neuen Knoten mit der Markierung des nicht blockierten Knotens.*

t_1 **und** t_2 **blockiert:** *Füge ein blockiertes Blatt in den resultierenden Baum an die entsprechende Stelle ein.*

t_1 **oder** t_2 **transparent:** *Füge ein transparentes Blatt in den resultierenden Baum an die entsprechende Stelle ein.*

andernfalls: *Klassifiziere die Nachfolger rekursiv. Füge einen inneren Knoten ein und füge die klassifizierten Nachfolger an.*

Falls t_1 und t_2 Blätter sind, füge ein partiell blockiertes Blatt in den resultierenden Baum ein.

In Abbildung 3.3, Mitte, wird das Resultat der Schnittbildung gezeigt. Algorithmus 5 zeigt den zugehörigen Pseudocode.

Die blockierten Knoten im Anfragebaum T_Q charakterisieren eine Teilmenge von Geraden, für die herausgefunden werden soll, welche davon blockiert oder transparent sind. Die Schnittbildung kopiert diejenigen Unterbäume des Szenenbaums T_S in den Schnittbaum, deren Wurzel zu einem blockierten Knoten im Anfragebaum korrespondiert. Der Schnittbaum gibt auf diese Weise Aufschluß darüber, wie es mit Blockierung und Transparenz in den Geradenmengen aussieht, die durch die blockierten Knoten im Anfragebaum T_Q repräsentiert werden.

3.3 Hierarchische Verdeckungsbäume

Nun wird der Verdeckungsbaum genutzt, um Verdeckungsmengen zwischen Anfrageelementen zu berechnen. Zu diesem Zweck wird die Szene in Unterszenen unterteilt, wobei ein Quadtree benutzt wird. Die Größe der Unterszenen kann durch die Vorgabe einer maximalen Anzahl von Elementen gesteuert werden, die den Quadranten einer Blatt-Szene

Algorithmus 5 Algorithmus zur Schnittbildung zweier Verdeckungsbäume

Parameter: Verdeckungsbaum $\mathbf{t}_1$, Verdeckungsbaum $\mathbf{t}_2$, Verdeckungsbaum $\mathbf{t}$: Schnittmenge von $\mathbf{t}_1$ und $\mathbf{t}_2$
Rückgabe: Schnitt $\mathbf{t}$ der Verdeckungsbäume $\mathbf{t}_1$ und $\mathbf{t}_2$.
IntersectionOfBlockerQuadtrees(BlockerQuadtree $\mathbf{t}_1$, BlockerQuadtree $\mathbf{t}_2$, BlockerQuadtree t)

```
if (t1 = NULL) or (t2 = NULL) then
  t=NULL;
  return;
end if
assign t new BlockerQuadtree node;
if t1 xor t2 blocking then
  if nonblocking node is subtree then
    add non blocking subtree to t
  else
    t is assigned tag of non blocking node
  end if
  return;
end if
if t1 and t2 blocking then
  t is blocking
  return
end if
if t1 or t2 transparent then
  t is transparent
  return
end if
if t1 and t2 leaves then
  t is partially blocking
  return
else
  t is partially blocking
  IntersectionOfBlockerQuadtrees(t1.NW, t2.NW, t.NW);
  IntersectionOfBlockerQuadtrees(t1.NE, t2.NE, t.NE);
  IntersectionOfBlockerQuadtrees(t1.SE, t2.SE, t.SE);
  IntersectionOfBlockerQuadtrees(t1.SW, t2.SW, t.SW);
  return
end if
```

schneiden dürfen. Das bedeutet, daß die Anzahl der Unterszenen besonders dort hoch ist, wo die Streckendichte hoch ist.

Definition 3.10 (Szenen-Quadtree)
Sei S eine ebene Szene aus Elementen im Einheitsquadrat. Der Szenen-Quadtree *ist ein Baum mit vier Nachfolgern an jedem inneren Knoten. Die Wurzel des Szenen-Quadtree repräsentiert das Einheitsquadrat. Alle anderen inneren Knoten repräsentieren Unterquadrate des Einheitsquadrats. Das Unterquadrat eines Knotens ist eins von vier kongruenten Unterquadraten des Quadrates, das deren Vater zugeordnet ist. Den Blättern des Szenen-Quadtrees sind Unterszenen zugeordnet, die aus den Elementen bestehen, die sich als nichtleerer Schnitt der Elemente der gegebenen Szene S mit dem Quadrat des Blatts ergeben.*

Abbildung 3.8, links, zeigt eine Szene von 30 zufällig plazierten Strecken. Im rechten Teil wird der Quadtree der Szene gezeigt. Die maximale Anzahl von Strecken in den Blättern ist in diesem Beispiel 1. Jeder Knoten des Quadtrees der Szene repräsentiert eine Unterszene,

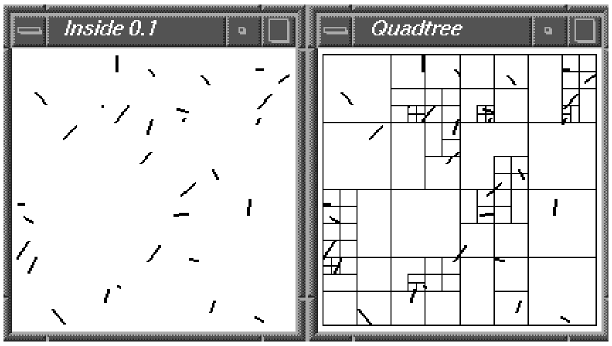


Abbildung 3.8: 30 Szenenstrecken (links), vorverarbeitet in einen Szenen-Quadtree (rechts)

die durch Clipping mit dem Quadranten des Knoten entsteht. Die Quadranten und die in ihnen enthaltenen Unterszenen sind in der Abbildung zu sehen. Die Unterszenen der inneren Knoten bestehen aus den korrespondierenden Unterquadranten. Algorithmus 6 zeigt den Pseudocode zur Konstruktion des Szene-Quadtrees.

Basierend auf einem Szenen-Quadtree wird ein hierarchischer Verdeckungsbaum wie folgt definiert:

Definition 3.11 (Hierarchischer Verdeckungsbaum)
Gegeben sei ein Quadtree einer Szene von Elementen im Einheitsquadrat sowie zwei verschiedene Kanten E_i und E_j des Einheitsquadrats. Ein hierarchischer Verdeckungsbaum zu E_i und E_j entsteht durch Verweis eines jeden Knotens des Szene-Quadtrees auf die Wurzel eines Verdeckungsbaums zu E_i und E_j, der zu der Unterszene gehört, die durch den Knoten repräsentiert wird.

Algorithmus 6 Algorithmus zur Konstruktion des Szenen-Quadtrees

Parameter: Szene S und der umschließende Quadrant $\mathbf{q}$.
Rückgabe: Zeiger auf einen Quadtree, der die Szene beschreibt.
Datenstrukturen:

```
structure Quadtree {
segments: Lineare Liste aller Strecken des Knotens;
quadrant: Boundingbox aller Strecken;
father: Vaterknoten;
NE,SE,SW,NW: Söhne;
iLR,iLT,iLB,iRB,iRT,iTB: Verdeckungsbäume dieses Knotens;
}
structure Segments {
Segment: Strecke (Start/Endpunkt);
next: Zeiger auf nächste Strecke;
}
structure Quadrant {
left, top, right, bottom : Eckpunkte des Quadranten
}
Quadtree *MakeQuadtree(Segments S, Quadrant q)
  assign t new quadtree node
  if S contains enough segments and q > ε then
    subdivide S into subscenes S_NE, S_NW, S_SE, S_SW
    subdivide q into subquadrants q_NE, q_NW, q_SE, q_SW
    if S_NE exists then
      t_NE=MakeQuadtree(S_NE, q_NE)
    end if
    if S_NW exists then
      t_NW=MakeQuadtree(S_NW, q_NW)
    end if
    if S_SE exists then
      t_SE=MakeQuadtree(S_SE, q_SE)
    end if
    if S_SW exists then
      t_SW=MakeQuadtree(S_SW, q_SW)
    end if
  end if
  return t
```

Abbildung 3.9 zeigt die Struktur der hierarchischen Verdeckungsbäume. Die Verdeckungsbäume der inneren Knoten können als Vereinigung der Verdeckungsbäume ihrer Nachfolger berechnet werden. Manchmal können Teilbäume der Nachfolger ohne Modifikation kopiert

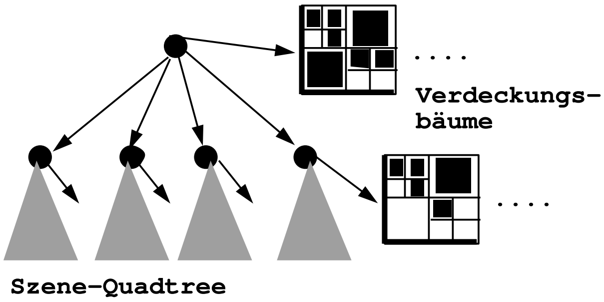


Abbildung 3.9: Die Struktur der hierarchischen Verdeckungsbäume. Schwarze Flächen entsprechen blockierten Geraden, weiße Flächen entsprechen nicht blockierten Geraden.

werden. Die *komprimierte Repräsentation* eines hierarchischen Verdeckungsbaums entsteht durch den Ersatz der Kopien dieser Teilbäume durch Referenzen.

Algorithmus 7 zeigt den Pseudocode zur Konstruktion des hierarchischen Verdeckungsbaumes. Der Algorithmus durchläuft den Szenenquadtree in post-order Reihenfolge bis zu dessen Blättern. Dort generiert er die Verdeckungsbäume der einzelnen Strecken, die den Blättern zugeordnet sind und faßt sie zu einem Verdeckungsbaum zusammen (*GenerateAndCombineBlockerQuadtrees* in Algorithmus 7). Anschließend werden jeweils rekursiv bis zur Wurzel die Verdeckungsbäume der Sohnknoten an den Vaterknoten zusammengefaßt (*CombineBlockerQuadtrees* in Algorithmus 7).

3.3.1 Anfragen an hierarchische Verdeckungsbäume

Der hierarchische Verdeckungsbaum wird nun zur Beantwortung von Anfragen nach der Verdeckungsmenge zwischen zwei Anfrageunterszenen verwendet. Die zu lösende Aufgabe ist folgende:

Definition 3.12 (Verdeckungsmenge zwischen zwei Unterszenen in einem hierarchischen Verdeckungsbaum)
Gegeben: *Ein hierarchischer Verdeckungsbaum T_S einer Szene S, Verdeckungsbäume T_{Q_1} und T_{Q_2} zweier disjunkter Unterszenen S_1 und S_2 im Szene-Quadtree von S.*
Gesucht: *Eine Approximation der Verdeckungsmenge zwischen T_{Q_1} und T_{Q_2} bezüglich T_S, dargestellt als Verdeckungsbaum T_A.*

Die Anfrage wird in drei Schritten beantwortet:

Algorithmus 7 Algorithmus zur Konstruktion des hierarchischen Verdeckungsbaums

Parameter: Szenen-Quadtree **t**.
Rückgabe: Hierarchischer Verdeckungsbaum ergänzt in **t**.
Datenstrukturen: Vgl. Algorithmus 6.

```
TraverseTreeForBlockerQuadtrees(Quadtree t)
  if t is empty then
    return
  end if
  if t is a leaf then
    GenerateAndCombineBlockerQuadtrees(t)
  else
    TraverseTreeForBlockerQuadtrees(t_NW)
    TraverseTreeForBlockerQuadtrees(t_NE)
    TraverseTreeForBlockerQuadtrees(t_SW)
    TraverseTreeForBlockerQuadtrees(t_SE)
    CombineBlockerQuadtrees(t)
  end if
```

1. Berechne den Schnitt T_Q der Verdeckungsbäume der beiden Verdeckungsbäume T_{Q_1} und T_{Q_2}.

2. Bestimme die maximalen Unterszenen des Szenenbaums, die von der Sichtbarkeitshülle von T_{Q_1} und T_{Q_2} geschnitten werden, d.h. von Geraden aus T_Q zwischen den beiden Anfrageunterszenen S_1 und S_2.

3. Berechne T_A aus T_Q und den Verdeckungsbäumen der Unterszenen aus 2.

Der zweite Schritt wird durch eine Bereichsanfrage an den Szenen-Quadtree mit der Sichtbarkeitshülle durchgeführt. Die Unterszenen, die als Antwort auf die Anfrage zurückgegeben werden, sind Kandidaten dafür, die Region zwischen den beiden Anfrageszenen zu blockieren.

Bei der Berechnung der Verdeckungsmenge im dritten Schritt sind zwei Fälle zu unterscheiden, vgl. Kapitel 3.1. Der erste Fall besteht darin, daß sich die Szenenquadrate der beiden Anfrageunterszenen S_1 und S_2 in keiner der Projektionen auf die Koordinatenachsen überlappen, vgl. Abbildung 3.10, links. In diesem Fall ergibt sich T_A als Schnitt von T_Q mit der Vereinigung der Verdeckungsbäume der im zweiten Schritt gefundenen Unterszenen, d.h. denen, die das Gebiet schneiden, das in der Abbildung mit a' gekennzeichnet ist.

Der zweite Fall liegt vor, wenn eine überlappende orthogonale Projektion auftritt. Abbildung 3.10, rechts, zeigt eine solche Situation. Die im folgenden beschriebene Behandlung bezieht sich auf diese, die anderen Überlappungssituationen lassen sich entsprechend bearbeiten. In diesem Fall müssen solche Geraden ausgeschlossen werden, die eine der potentiell

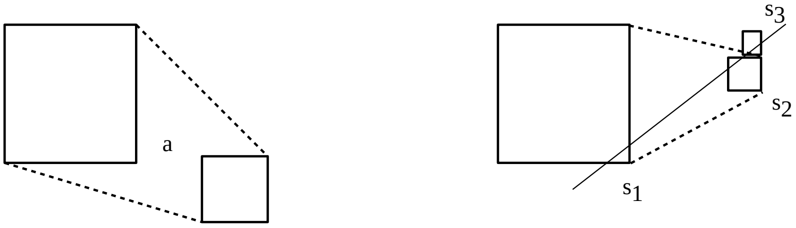


Abbildung 3.10: Mögliche relative Positionen der Anfragequadranten: nicht überlappend (links) und überlappend (rechts). Im überlappenden Fall müssen solche Geraden ausgeschlossen werden, die ein Element $\mathbf{S}_3$ schneiden, aber kein Element zwischen $\mathbf{S_1}$ und $\mathbf{S}_2$.

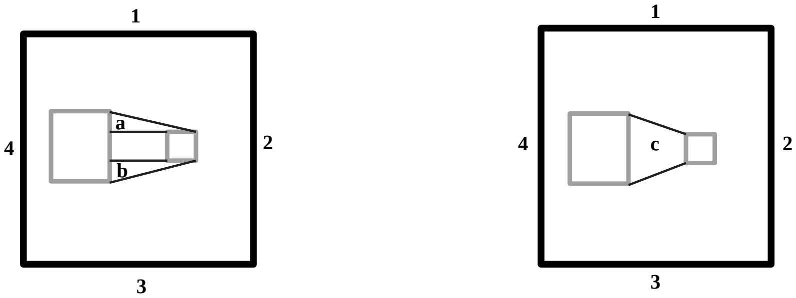


Abbildung 3.11: Dekomposition der Sichtbarkeitshülle zwischen den Anfragequadranten in drei Regionen a, b und c, die separat behandelt werden. Zur Übersichtlichkeit sind sie in zwei Abbildungen eingezeichnet.

blockierenden Unterszenen S_i schneiden, jedoch keinen Schnittpunkt mit irgendeiner Unterszene haben, der zwischen S_1 und S_2 liegt. Die in Abbildung 3.10, rechts, eingezeichnete Gerade ist ein solches Beispiel, wenn S_3 als einzige Unterszene angenommen wird, die im zweiten Schritt des Algorithmus geliefert wird.

Zur Bearbeitung dieses Falls wird zwischen den drei Regionen a, b und c unterschieden, welche in Abbildung 3.11 zum besseren Verständnis in zwei Teilabbildungen gezeichnet wurden. Für jede dieser drei Regionen werden die maximalen geschnittenen Unterszenen bestimmt.

Für die Region a werden nun nur die Verdeckungsbäume der Kantenpaare 1/2, 3/4 (vgl. Abbildung 3.11) unverändert berücksichtigt. Die Bäume 1/3 und 2/4 werden auf die Teilmenge unterhalb der Diagonale beschränkt, wie es in Abbildung 3.12 zu sehen ist. Die Beschränkung wird mit der gewünschten Genauigkeit berechnet. Die Bäume 1/4 und 2/3 werden nicht berücksichtigt. Die Geraden mit nicht-negativer Steigung, die für den kritischen Fall in Abbildung 3.10, rechts, verantwortlich sind, werden also ausgeschlossen.

Für die Region b wird dasselbe mit den Unterszenen, die b schneiden, gemacht. Hier werden die Verdeckungsbäume der Kantenpaare 2/3, 1/4 und die „Hälfte" der Bäume 1/3 und 2/4 berücksichtigt. Für die Region c gibt es keine Restriktionen.

Die gesuchte Verdeckungsmenge T_A ergibt sich als Schnitt von T_Q mit den so gefundenen Verdeckungsbäumen.

Abbildung 3.12: Nur der Teil des Verdeckungsbaums unter der Diagonale wird für die Anfrage im Fall der Bäume 1/3 und 2/4 benutzt.

Eine Pseudocode-Formulierung dieses Verfahrens zur Bestimmung der Verdeckungsregion für zwei Quadranten des Szene-Quadtrees für den ersten Fall findet sich in Algorithmus 8.

Die blockierten Blätter des resultierenden Baums T_A beschreiben die Sichtgeraden der Anfrage, die durch die blockierten Blätter in T_Q gegeben sind, die blockiert werden, d.h. die Verdeckungsregion. Die Sichtbarkeitsregion wird durch die blockierten Blätter der Differenz $\overline{T}_A = T_Q - T_A$ beschrieben.

Definition 3.13 (Differenz von Verdeckungsbäumen)
Die Differenz $T_1 - T_2$ *zweier Verdeckungsbäume* T_1 *und* T_2*, wird durch den simultanen, rekursiven Durchlauf beider Bäume erzeugt. Seien* t_1 *und* t_2 *zwei korrespondierende Knoten der Bäume* T_1 *und* T_2*. Abhängig vom Typ der Sichtbarkeit dieser Knoten wird der Differenzbaum wie folgt definiert, wobei wiederum innere Knoten als partiell blockiert behandelt werden.*

t_1 **ist ein blockiertes Blatt:** *Füge einen Knoten in den resultierenden Baum ein und hänge das Komplement des Teilbaums mit der Wurzel* t_2 *an diesen Knoten, falls es einen gibt. Andernfalls markiere den neuen Knoten mit dem Komplement der Markierung von* t_2*.*

t_1 **ist ein transparentes Blatt oder** t_2 **ist ein blockiertes Blatt:** *Füge einen Knoten in den resultierenden Baum ein und markiere ihn als transparent.*

t_2 **ist ein transparentes Blatt:** *Füge einen Knoten in den resultierenden Baum ein und hänge den Teilbaum mit der Wurzel* t_1 *daran, falls er existiert. Andernfalls markiere den neuen Knoten mit der Markierung von* t_1*.*

Algorithmus 8 Algorithmus zur Berechnung der Verdeckungsregion zweier Unterszenen

globale Variable: **rays**: Der Durchschnitt der hierarchischen Verdeckungsbäume (bzgl. einer Randkantenkombination) von $\mathbf{q}_1$ und $\mathbf{q}_2$.
Parameter: Szenen-Quadtree **q**, Quadranten $\mathbf{q}_1$ und $\mathbf{q}_2$, **bb**: Boundingbox von $\mathbf{q}_1$ und $\mathbf{q}_2$.
Rückgabe: TRANSP: Die Quadranten sehen sich. BLOCKED: Die Quadranten sehen sich nicht. DISJOINT: Die Quadranten von q und bb sind disjunkt. **rays** enthält die duale Darstellung der Geraden, die zur Verdeckungsregion der Unterszenen $\mathbf{q}_1$ und $\mathbf{q}_2$ gehören.

```
CheckForBlock(Quadtree q, Quadtree q_1, q_2, Quadrant bb)
  if q is empty or q equals q_1 or q_2 then
    return TRANSP
  end if
  if bb in q then
    if CheckForBlock(q_NW,q_1,q_2,bb) returns BLOCKED or
    CheckForBlock(q_NE,q_1,q_2,bb) returns BLOCKED or
    CheckForBlock(q_SE,q_1,q_2,bb) returns BLOCKED or
    CheckForBlock(q_SW,q_1,q_2,bb) returns BLOCKED then
      return BLOCKED
    else
      return TRANSP
    end if
  end if
  if bb and q are disjoint then
    return DISJOINT
  else
    if q is not a leaf then
      if q_1 or q_2 in q then
        if CheckForBlock(q_NW,q_1,q_2,bb) returns BLOCKED or
        CheckForBlock(q_NE,q_1,q_2,bb) returns BLOCKED or
        CheckForBlock(q_SE,q_1,q_2,bb) returns BLOCKED or
        CheckForBlock(q_SW,q_1,q_2,bb) returns BLOCKED then
          return BLOCKED
        else
          return TRANSP
        end if
      else
        goto LABEL
      end if
    else
      LABEL: BlockerQuadtree newrays
      DifferenceOfBlockerQuadtrees(rays, q_innerQuadtree, newrays)
      rays = newrays
      if rays not empty then
        return TRANSP
      else
        return BLOCKED
      end if
    end if
  end if
```

Andernfalls: *Falls mindestens ein Knoten der Wurzelknoten eines Unterbaums ist, klassifiziere den nachfolgenden Knoten rekursiv. Füge einen inneren Knoten ein, der die Wurzel der klassifizierten Nachfolger ist.*

Das Komplement von „blockiert” ist „transparent” und umgekehrt. Das Komplement von „partiell blockiert” ist wiederum „partiell blockiert”.

Falls t_1 und t_2 Blätter sind, füge ein partiell blockiertes Blatt in den resultierenden Baum ein.

Abbildung 3.3, unten, stellt ein Beispiel für die Differenz dar. Algorithmus 9 zeigt eine Pseudocode-Formulierung für die Berechnung der Differenz.

Falls das einzige Blatt des reduzierten T_A seine Wurzel ist und diese als „transparent” markiert ist, dann ist die Sicht zwischen den beiden Anfrageteilszenen komplett transparent. Ein Quadtree heißt *reduziert*, falls er keine vier Blätter enthält, die einen gemeinsamen Vater mit derselben Markierung haben. Falls das gleiche für $\overline{T}_A$ gilt, dann ist die Sichtbarkeit zwischen den beiden Anfrageunterszenen komplett blockiert.

3.4 Experimentelle Analyse des 2D-Verdeckungsbaums

Für die experimentelle Analyse der zweidimensionalen Verdeckungsbäume wurden die Datenstruktur und die Operationen auf den Datenstrukturen implementiert und Messungen auf einer SGI Indy R4400 vorgenommen. Die graphische Darstellung der Ergebnisse auf dem Bildschirm erfolgt mit Hilfe von OpenGL und die Eingabe geschieht über eine TCL/TK-Benutzerschnittstelle [Ous93]. Weitere Hinweise zur Implementierung finden sich in [HM97]. Abbildung 3.4, links, zeigt eine Szene aus 100 gleichmäßig verteilten Strecken. Ein zugehöriger Verdeckungsbaum wird rechts gezeigt. Er wird durch die linke und untere Kante des die Szene umgebenden Quadrats induziert.

Da die Szenenstrecken an den Rändern der Unterszenen-Quadtrees abgeschnitten werden, ist die Anzahl der Strecken, die in die Berechnung der Sichtbarkeit einbezogen werden, höher als die Anzahl der ursprünglichen Szenenstrecken. Dazu wurde folgendes Experiment durchgeführt:

Experiment 3.2 (Messung der durch die Quadtree-Unterteilung generierten Strecken)

Verfahren: *Rekursive Unterteilung der Szene in Quadranten, mit Generierung neuer Strecken durch Schnitt mit Quadrantenumrandung.*

Eingabe: *Zufällig generierte Strecken, wie in Abbildung 3.4, links.*

Algorithmus 9 Algorithmus zur Subtraktion zweier Verdeckungsbäume

Parameter: Verdeckungsbaum $\mathbf{t}_1$,$\mathbf{t}_2$, Verdeckungsbaum $\mathbf{t}$: Differenz,
Rückgabe: Differenz $\mathbf{t}$ der Verdeckungsbäume $\mathbf{t}_1$ und $\mathbf{t}_2$.
DifferenceOfBlockerQuadtrees(BlockerQuadtree $\mathbf{t}_1$, BlockerQuadtree $\mathbf{t}_2$, BlockerQuadtree t)

if ($\mathbf{t}_1$ = NULL) or ($\mathbf{t}_2$ = NULL) **then**
  t=NULL;
  return;
end if
assign **t** new BlockerQuadtree node;
if $\mathbf{t}_1$ is blocking **then**
  if $\mathbf{t}_2$ is a subtree **then**
    add complement of $\mathbf{t}_2$ to **t**;
  else
    mark **t** with complement of tag of $\mathbf{t}_2$
  end if
  return;
end if
if $\mathbf{t}_1$ is transparent or $\mathbf{t}_2$ is blocking **then**
  t is transparent
  return;
end if
if $\mathbf{t}_2$ is transparent **then**
  if $\mathbf{t}_1$ is a subtree **then**
    add $\mathbf{t}_1$ to **t**;
  else
    mark **t** with tag of $\mathbf{t}_1$
  end if
  return;
end if
if $\mathbf{t}_1$ and $\mathbf{t}_2$ leaves **then**
  t is partially blocking
  return
else
  t is partially blocking
  DifferenceOfBlockerQuadtrees($\mathbf{t}_{1NW}$,$\mathbf{t}_{2NW}$, $\mathbf{t}_{NW}$)
  DifferenceOfBlockerQuadtrees($\mathbf{t}_{1NE}$,$\mathbf{t}_{2NE}$, $\mathbf{t}_{NE}$)
  DifferenceOfBlockerQuadtrees($\mathbf{t}_{1SE}$,$\mathbf{t}_{2SE}$, $\mathbf{t}_{SE}$)
  DifferenceOfBlockerQuadtrees($\mathbf{t}_{1SW}$,$\mathbf{t}_{2SW}$, $\mathbf{t}_{SW}$)
  return;
end if

Parameter: *Maximal 5, 10, 20 und 50 Strecken pro Quadtree-Blatt.*

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Vgl. Abbildung 3.13: die Anzahl der durch die Quadtree-Unterteilung generierten Strecken (y-Achse) in Abhängigkeit von der Anzahl der Originalstrecken (x-Achse).*

Beobachtung: *Abbildung 3.13 zeigt, daß die Anzahl der durch das Verfahren*

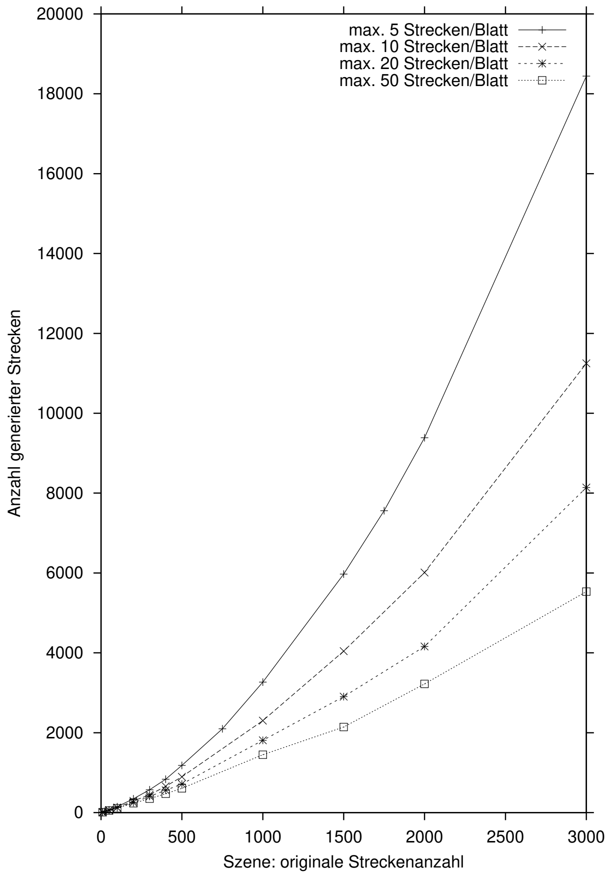


Abbildung 3.13: Anzahl der generierten Strecken abhängig von der Anzahl der Eingabestrecken

generierten Strecken (y-Achse) deutlich stärker als linear in der Anzahl der Originalstrecken (x-Achse) wächst.

Experiment 3.3 (Messung der Berechnungszeiten für den Szene-Quadtree)

Verfahren: *Rekursive Unterteilung der Szene in Quadranten, mit Generierung neuer Strecken durch Schnitt mit Quadrantenumrandung.*

Eingabe: *Zufällig generierte Strecken, wie in Abbildung 3.4, links.*

Parameter: *Maximal 5 Strecken pro Quadtree-Blatt.*

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Vgl. Abbildung 3.14: für die Quadtree-Unterteilung benötigte Zeit (y-Achse) in Abhängigkeit von der Anzahl der Originalstrecken (x-Achse).*

Beobachtung: *Die Berechnung des Quadtrees der Szene benötigt nur wenig Zeit. Für 1000 Strecken sind ungefähr 100ms Rechenzeit notwendig.*

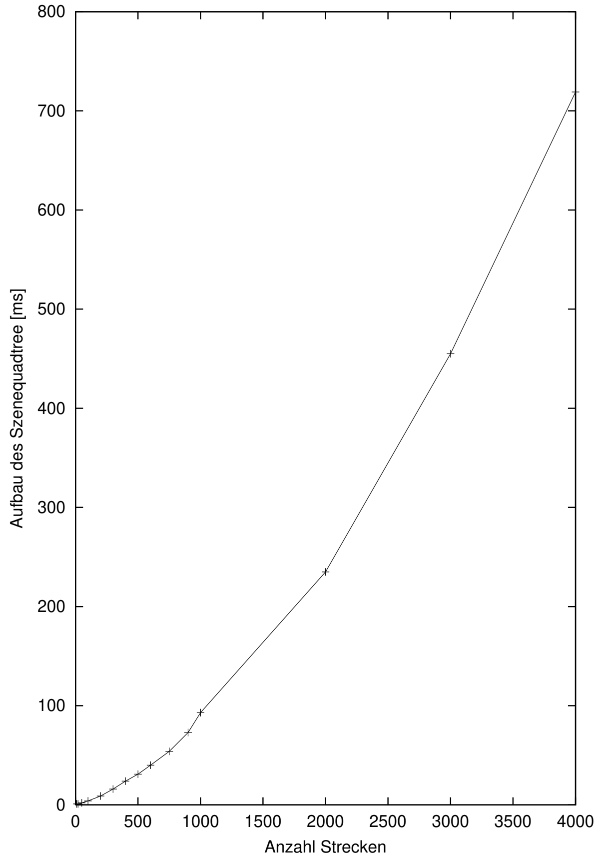


Abbildung 3.14: Berechnungszeit zum Aufbau des Szenen-Quadtrees (y-Achse) in Abhängigkeit von der Anzahl der Originalstrecken (x-Achse)

Die Anzahl von maximal 5 Strecken pro Quadtree-Blatt im vorigen Experiment hat eine feine Unterteilung der Szene zur Folge. Daraus folgt eine zeitaufwendige Berechnung der Sichtbarkeit, wie das folgende Experiment belegt.

Experiment 3.4 (Messung der Berechnungszeiten für den hierarchischen Verdeckungsbaum)

Verfahren: *Rekursive Konstruktion des hierarchischen Verdeckungsbaums aus den Verdeckungsbäumen der einzelnen Quadranten des Szenen-Quadtrees.*

Eingabe: *Zufällig generierte Strecken, wie in Abbildung 3.4, links.*

Parameter: *Maximal 5 Strecken pro Quadtree-Blatt.*

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Vgl. Abbildung 3.15: für die Konstruktion des hierarchischen Verdeckungsbaumes benötigte Zeit (y-Achse) in Abhängigkeit von der Anzahl der Originalstrecken (x-Achse).*

Beobachtung: *Bedingt durch das Clippen der Szenenstrecken bei der Berechnung des Quadtrees der Szene werden bei der Wahl von maximal 5 Strecken*

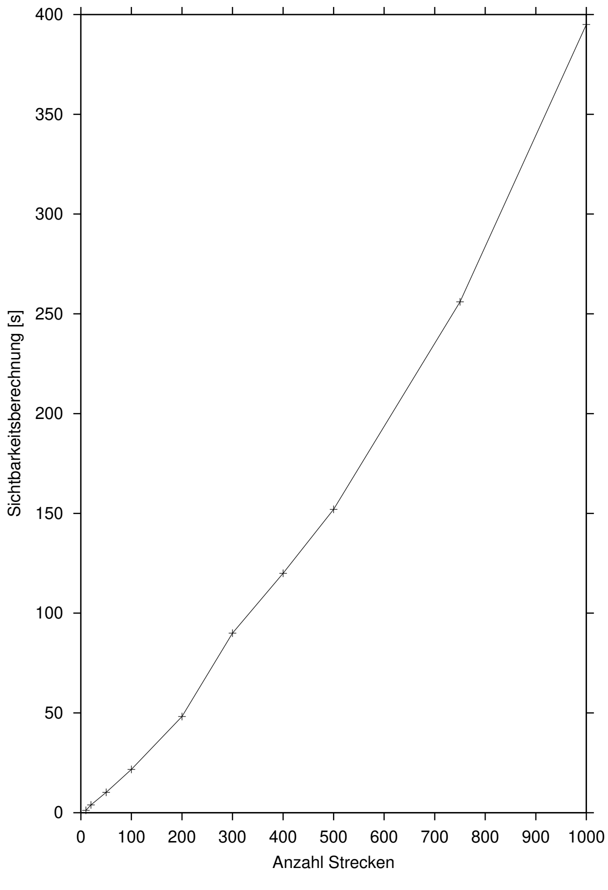


Abbildung 3.15: Berechnungszeit (y-Achse) zum Aufbau des hierarchischen Verdeckungsbaumes in Abhängigkeit von der Anzahl der Originalstrecken (x-Achse)

pro Blatt des Quadtrees der Szene deutlich mehr Strecken generiert als die Anzahl der Originalstrecken (vgl. Abbildung 3.13). Daraus folgt eine stärker als linear in der Anzahl der Originalstrecken wachsende Berechnung des hierarchischen Verdeckungsbaums der geclippten Originalstrecken.

Die Zeit, die zur Berechnung einer Anfrage der Sichtbarkeit zwischen zwei Unterszenen benötigt wird, liegt im Bereich von Millisekunden und ist für die hier betrachteten Szenen relativ unabhängig von der Größe der Eingabeszene. Das folgende Experiment analysiert die Abhängigkeit genauer.

Experiment 3.5 (Messung der Berechnungszeiten für Sichtbarkeitsanfragen)

Verfahren: *Sichtbarkeitsanfrage zwischen zwei Unterszenen des hierarchischen Verdeckungsbaumes. Es wird jeweils die obere linke und die untere rechte Unterszene im Szenen-Quadtree gewählt.*

Eingabe: *Zufällig generierte Strecken, wie in Abbildung 3.4, links.*

Parameter: *Maximal 10 Strecken pro Quadtree-Blatt.*

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Vgl. Abbildung 3.16: für die Sichtbarkeitsanfrage benötigte Zeit (y-Achse) in Abhängigkeit von der Anzahl der Originalstrecken (x-Achse).*

Beobachtung: *Bis zu 500 Strecken steigt die Anfragezeit an. Danach fällt die Zeit auf einen Level von 0.2 bis 0.8 msek. Der Grund hierfür ist wahrscheinlich der folgende: Wenn der Raum der Szene mit vielen Strecken gefüllt ist, dann befinden sich viele Blätter auf den oberen Ebenen des Verdeckungsbaumes. Daher können viele Anfragen schon sehr weit oben in der Hierarchie entschieden werden, d.h. zu einem frühen Zeitpunkt.*

Zur Analyse des Speicherbedarfs wurde das folgende Experiment durchgeführt:

Experiment 3.6 (Messung des Speicherbedarfs der hierarchischen Verdeckungsbäume)

Verfahren: *Rekursive Konstruktion des hierarchischen Verdeckungsbaums aus den Verdeckungsbäumen der einzelnen Quadranten des Szenen-Quadtrees.*

Eingabe: *Zufällig generierte Strecken, wie in Abbildung 3.4, links.*

Parameter: *Maximal 5, 20 und 50 Strecken pro Quadtree-Blatt. Hierbei werden die beiden Varianten „copy" (Kopieren kompletter Unterbäume) und „ptr" (identische Unterbäume werden referenziert) unterschieden.*

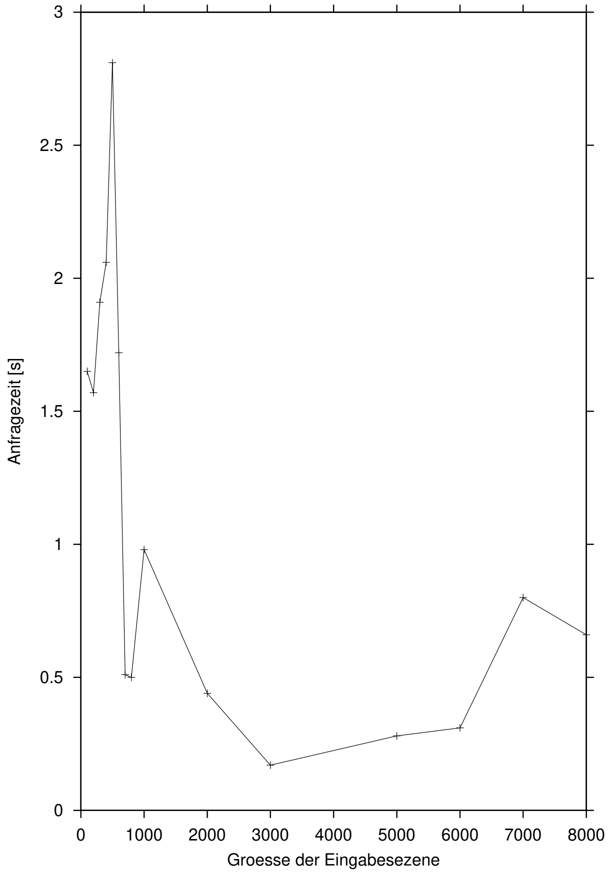


Abbildung 3.16: Anfragezeit der Sichtbarkeitsberechnung zwischen 2 Quadranten (y-Achse) in Abhängigkeit von der Anzahl der Originalstrecken (x-Achse)

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Vgl. Abbildung 3.17: für die hierarchischen Verdeckungsbäume benötigter Speicher (y-Achse) in Abhängigkeit von der Anzahl der Originalstrecken (x-Achse).*

Beobachtung: *Die Kurven zeigen, daß der Speicherbedarf drastisch von der Anzahl der Strecken pro Blatt abhängt. An der Kurve für maximal 50 Strecken pro Blatt kann beobachtet werden, daß der benötigte Speicher nicht unbedingt immer mit der Anzahl der Strecken wächst. Durch Effekte, wie die Zusammenfassung von Sichtregionen gleicher Sichtbarkeit, ergibt sich manchmal trotz steigender Streckenzahl ein geringerer Speicherbedarf.*

Im folgenden Experiment werden der Speicherbedarf und die Anzahl der durch Clipping generierten Strecken verglichen.

Experiment 3.7 (Messung der durch Clipping generierten Strecken)

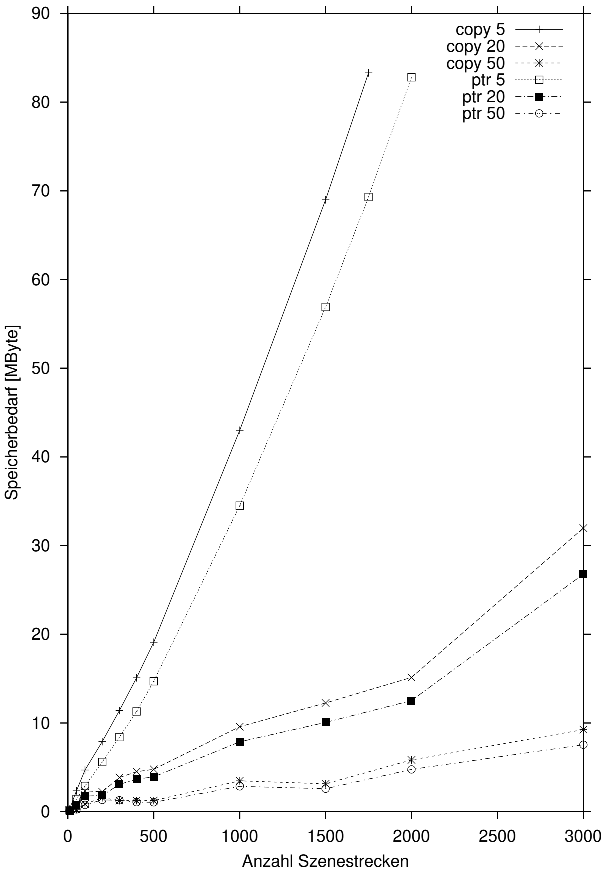


Abbildung 3.17: Speicherbedarf für hierarchische Verdeckungsbäume (y-Achse) mit kopierten oder referenzierten Teilbäumen in Abhängigkeit von der Größe der Eingabeszene (x-Achse)

Verfahren: *Rekursive Unterteilung der Szene in Quadranten, mit Generierung neuer Strecken durch Schnitt mit Quadrantenumrandung.*

Eingabe: *Zufällig generierte Strecken, wie in Abbildung 3.4, links.*

Parameter: *Maximal 5 Strecken pro Quadtree-Blatt.*

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Vgl. Abbildung 3.18: für die hierarchischen Verdeckungsbäume benötigter Speicher (y-Achse) in Abhängigkeit von der Anzahl der durch das Clipping entstehenden Strecken (x-Achse).*

Beobachtung: *Im Gegensatz zu Abbildung 3.17 kann hier eine (sub)lineare Abhängigkeit zu der Anzahl der generierten Strecken vermutet werden.*

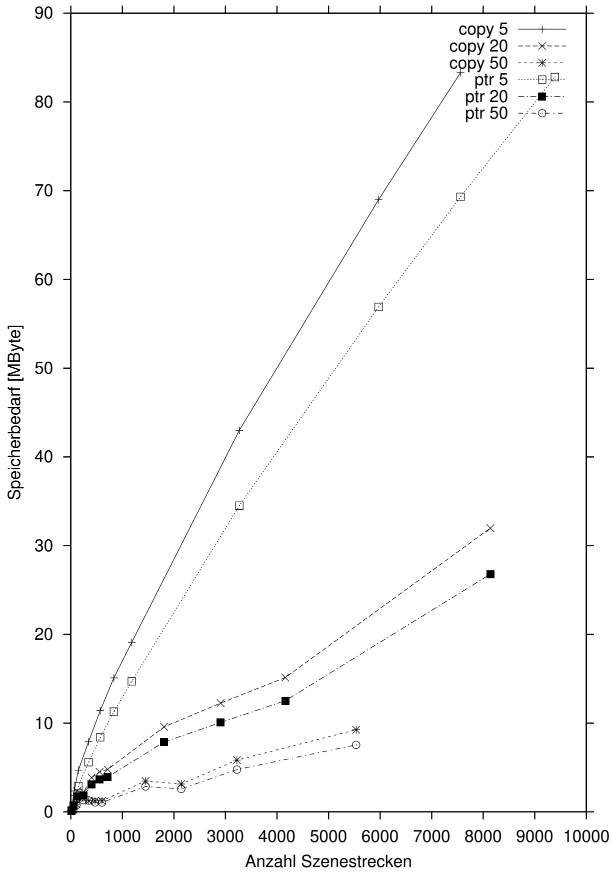


Abbildung 3.18: Für die hierarchischen Verdeckungsbäume benötigter Speicher (y-Achse), in Abhängigkeit von der Anzahl der durch das Clipping entstehenden Strecken (x-Achse)

3.5 Speicheraspekte

Ein kritischer Aspekt des vorgestellten Ansatzes der hierarchischen Verdeckungsbäume könnte der Speicherbedarf für den dreidimensionalen Fall sein. Im schlechtesten Fall kann ein Quadtree der Höhe k eine Blattanzahl von 4^k haben. Wird die Information „blockiert", „transparent" oder „partiell blockiert" an den Blättern mit 2 Bits codiert, bedeutet dies für $k = 10$ einen Speicherbedarf von $2 \cdot 4^{10}$ Bit $=$ 2 MBit $=$ 256 KByte. Dies entspricht einer Unterteilung einer Kante des Einheitsquadrats in 1024 Teilintervalle.

Diesem akzeptablen Wert steht erheblich mehr Speicherbedarf im dreidimensionalen Fall gegenüber. Das Prinzip der zweidimensionalen Verdeckungsbäume ist direkt auf den dreidimensionalen Anwendungsbereich erweiterbar. Der aus der R-Dualisierung der Geraden resultierende vierdimensionale Dualraum kann in diesem Fall durch einen vierdimensionalen Hyperoctree unterteilt werden. Bezüglich des Szenenraums korrespondiert dies mit einer Quadtree-Unterteilung der beiden Seitenflächen des die Szene umhüllenden Einheitswürfels, welche dem entsprechenden Verdeckungsbaum zugrunde liegen. Insgesamt werden $6 \cdot 5/2 = 15$ Verdeckungsbäume benötigt. Ein Verdeckungsbaum der Höhe k hat im schlechtesten Fall 16^k Blätter. Bei zwei Bit pro Blatt und $k = 10$ führt dies auf $2 \cdot 16^{10}$ Bit = 2048 Gbit = 256 GByte. Dem korrespondiert wiederum eine Unterteilung einer Kante des Einheitquadrats in 1024 Intervalle. Bei einer Unterteilung in nur 32 Intervalle genügen 256 KByte. Dies zeigt die Grenzen bei heutigen Rechnern auf, läßt aber angesichts weiter steigender Speicherkapazitäten Praktikabilität erwarten.

Eine Speicherplatzersparnis ist durch die Verwendung *hybrider Verdeckungsbäume* möglich. Dem liegt die Beobachtung zugrunde, daß Verdeckungsbäume häufig sehr viele Blätter zur genauen Approximation der Grenze zwischen transparenten und blockierenden Gebieten brauchen. Die Grenze selbst wird durch partiell blockierende Blätter repräsentiert. Die Vorgehensweise bei hybriden Verdeckungsbäumen ist, bei einem transparenten Blatt auf die Elemente der Szene zu verweisen, die eine Blockierung von Geraden, die dem Blatt entsprechen, hervorrufen. Falls bei einer Anfrage eine genauere Approximation der Verdeckungsregion an diesem Blatt notwendig ist, ist gegebenenfalls eine genauere Berechnung „on the fly” durchzuführen. Die Möglichkeit der effizienten Nachberechnung macht es möglich, die Grenze auch durch größere partiell blockierende Gebiete zu überdecken und so Bäume mit weniger Blättern zu haben. Diese erlaubt einen Trade-off zwischen Speicherbedarf und Rechenzeit. Ein weiterer Aspekt ist, daß in der realitätsnahen Bilderzeugung die Verdeckungsbäume nur für solche Blickrichtungen vorverarbeitet werden sollten, für die Anfragen erwartet werden. Beispielsweise wird beim Radiosity-Verfahren die Sichtbarkeit nur zwischen Oberflächen untersucht.

Die bisherigen Betrachtungen bezogen sich auf die Anzahl der Blätter. Es wird jedoch zusätzlicher Speicher für die inneren Knoten der Bäume benötigt. Zwei bekannte Speicherformen für Quadtrees sind die als Zeigerbaum und die als zeigerloser Baum. Daneben gibt es eine hybride Form, die beides kombiniert.

Definition 3.14 (Zeigerbaum)
Der Verweis der inneren Knoten des Baums auf ihre Nachfolger geschieht durch jeweils einen Zeiger. Die Blätter des Baums repräsentieren die Verdeckungsinformation „transparent”, „blockiert” oder „partiell blockiert” (Abbildung 3.19).

Diese Datenstruktur ist intuitiv und einfach zu implementieren. Allerdings bringt die Verzeigerung in der Praxis einen nicht unerheblichen Speicherbedarf mit sich, gerade bei dicht besetzten Bäumen. Ein Zeiger benötigt in einem 32-Bit Adreßraum 4 Bytes. Ein vollbesetzer Quadtree der Höhe k hat $\sum_{i=1}^{k} 4^k = \frac{4^{k+1}-1}{3} - 1$ Zeiger. Der damit verbundene Speicherbedarf der Zeiger ist $4 \cdot (\frac{4^{k+1}-1}{3} - 1)$ Bytes.

Definition 3.15 (Zeigerloser Baum)
Die Knoten jeder Ebene des Baumes werden in eine Zeichenfolge kodiert. Die verwendeten

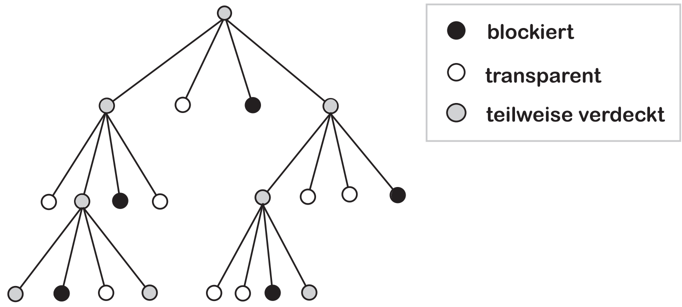


Abbildung 3.19: Explizite Speicherung als verzeigerter Baum

Zeichen sind „innerer Knoten", „blockiertes Blatt", „transparentes Blatt" und „partiell blockiertes Blatt".

Die vier Zeichen erlauben eine Kodierung mit 2 Bit pro Zeichen. Der Vorteil des zeigerlosen Baums gegenüber dem Zeigerbaum ist der geringere Speicherbedarf durch den Wegfall der Zeiger. Der Nachteil ist der hohe Zeitaufwand zum Auffinden einzelner Knoten. Dieser Nachteil kann durch eine hybride Version, bei der zusätzlich Zeiger verwendet werden, gemindert werden, allerdings auf Kosten des Speicherbedarfs.

Definition 3.16 (Teilweise verzeigerter Baum)
Ein hybrider Baum entsteht dadurch, daß die Zeichenkette einer Ebene des zeigerlosen Baums in Pakete aufgeteilt wird, wobei die Pakete einer Ebene untereinander verbunden werden. Jedes Paket enthält ein ganzzahliges Vielfaches der Anzahl der Nachfolgerknoten eines inneren Knotens als Knoten. Jedes Paket wird mit einem Zeiger auf die Nachfolger des ersten inneren Knotens des Pakets versehen.

Abbildung 3.20 zeigt Pakete, die ebenso viele Knoten besitzen wie ein innerer Knoten Nachfolger hat (hier vier), d.h., das erwähnte ganzzahlige Vielfache ist 1.

Weitere Datenstrukturen zur effizienten Speicherung von Bäumen sind in [Samet89] zu finden.

Zum Vergleich des Speicherbedarfs wurden folgende Experimente anhand einer Implementierung des Algorithmus in C++ durchgeführt. Dabei handelt es sich um Messungen für den 3D-Verdeckungsbaum, da gerade der 3D-Fall einen großen Speicherbedarf mit sich bringt. Die verschiedenen Strategien können sich hier besonders gegeneinander abgrenzen.

Experiment 3.8 (Messung des Speicherbedarfs des 3D-Verdeckungsbaumes)

> **Verfahren:** *Rekursive Konstruktion des 3D-Verdeckungsbaumes.*
>
> **Eingabe:** *200 zufällig verteilte Dreiecke.*

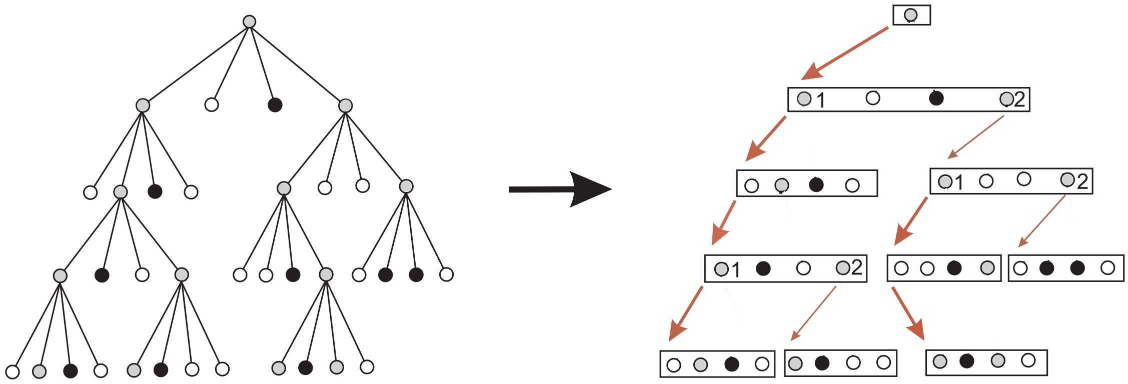


Abbildung 3.20: Komprimierung durch wenig verzeigerten Baum

Parameter: *Die verschiedenen Datenstrukturen. Maximale Unterteilungstiefe 4.*

Plattform: *SGI* O^2*, R5000, 180MHz.*

Gemessene Größe(n): *Vgl. Abbildung 3.21: für den 3D-Verdeckungsbaum benötigter Speicher in Abhängigkeit von der Anzahl der Dreiecke.*

Beobachtung: *Obgleich der Speicherbedarf der Datenstrukturen bis auf einen konstanten Faktor gleich zu sein scheint, ergibt sich durch die Nutzung des*

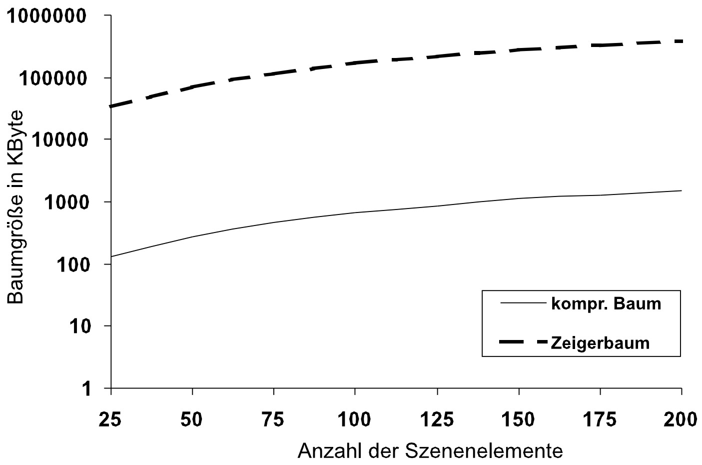


Abbildung 3.21: Speicherbedarf der Datenstrukturen (y-Achse) für verschiedene Szenengrößen in Abhängigkeit von der Anzahl der Dreiecke (x-Achse)

komprimierten Baumes in der Praxis ein deutlicher Vorteil. Dieser wird durch das Entfernen der Verzeigerung des Zeigerbaumes erreicht.

Nicht nur die Komprimierung, sondern auch die Auswahl des Unterteilungsverfahrens kann zu einem geringeren Speicherbedarf beitragen. Bekannte Unterteilungsverfahren, bezogen auf ein Quadrat, sind:

Definition 3.17 (Unterteilungsverfahren für ein Quadrat)

Symmetrische Quadranten-Unterteilung: *Unterteilung des Quadrats in vier kongruente Unterquadrate, wie sie beim Quadtree verwendet wird (Abbildung 3.22).*

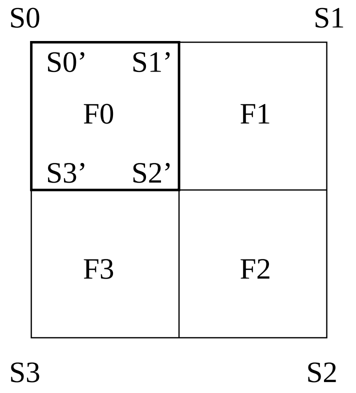


Abbildung 3.22: Unterteilung der Seitenflächen in Quadranten

Binäre Unterteilung: *Unterteilung des Quadrats in zwei kongruente Rechtecke, die, iteriert, auf Bintrees führt (Abbildung 3.23).*

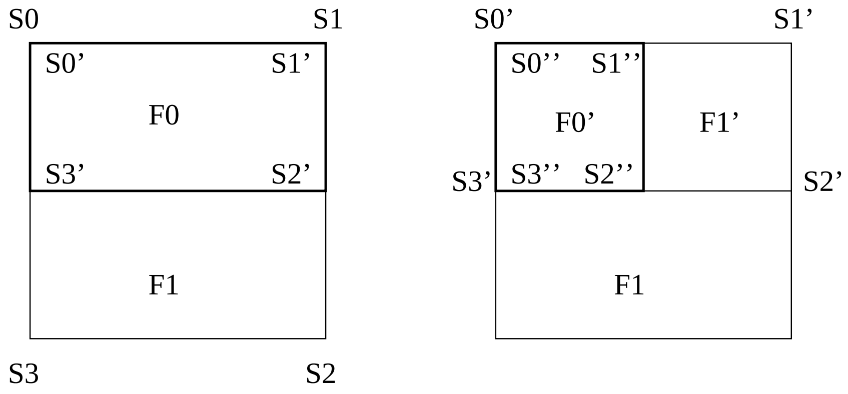


Abbildung 3.23: Unterteilung der Seitenflächen in Hälften

Dreiecks-Unterteilung: *Unterteilung des Quadrats in zwei kongruente Dreiecke, die iteriert halbiert werden (Abbildung 3.24). Alle folgenden Unterteilungen teilen die jeweils längste Kante eines Dreiecks in der Mitte so auf, daß eine Strecke vom Mittelpunkt der Seite zum gegenüberliegenden Punkt des Dreiecks eingefügt wird (Abbildung 3.24, Mitte und rechts).*

Die Dreiecksunterteilung versucht, durch nicht zu den Kanten der Seitenflächen parallele Unterteilungskanten die „starre" Unterteilung zu umgehen und sich den in der Szene beliebig ausgerichteten Elementen besser anzupassen. Abbildung 3.25 erlaubt einen opti-

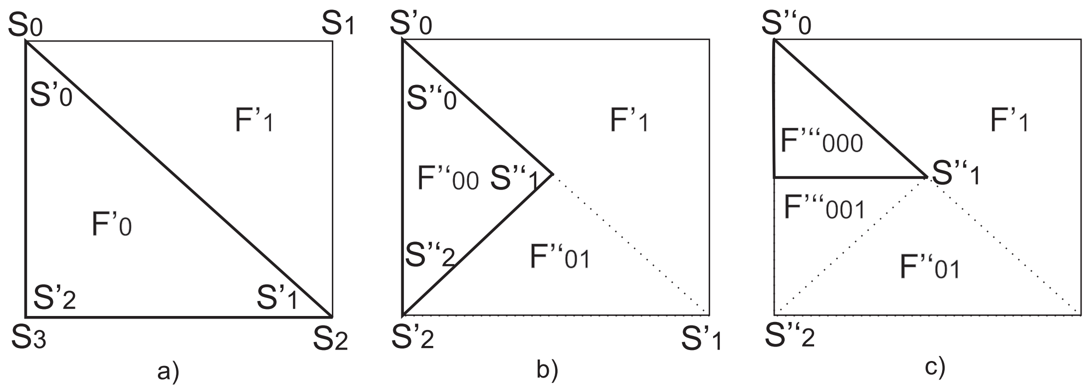


Abbildung 3.24: Unterteilung der Seitenflächen in Dreiecke

schen Vergleich der Verfahren. Sie zeigt die rekursive Unterteilung des Verdeckungsbaumes für die Quadranten-Unterteilung (links), die binäre Unterteilung (Mitte) und für die Dreiecks-Unterteilung (rechts). Zu erkennen ist, daß die binäre Unterteilung einige Un-

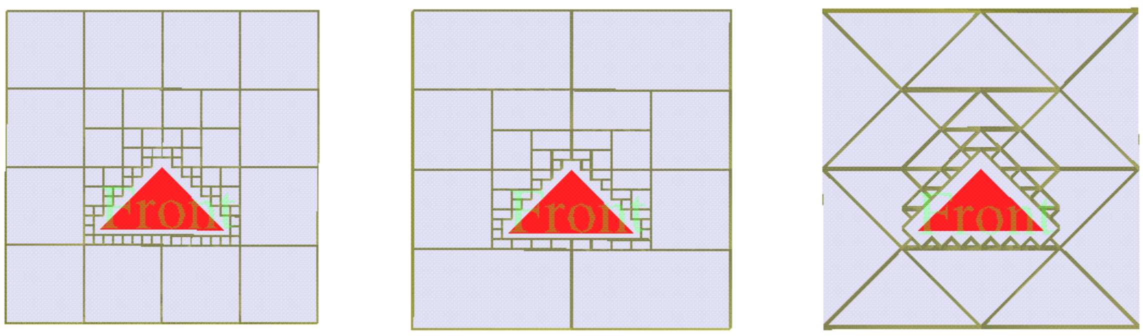


Abbildung 3.25: Optischer Vergleich der unterschiedlichen Unterteilungsverfahren

terteilungen ausläßt, die die Quadranten-Unterteilung aufgrund des starren Unterteilungsschemas ausführen muß. Die wenigsten Unterteilungen führt der Dreiecks-Unterteilungs-Algorithmus durch.

Für die folgenden Experimente, die für dreidimensionale Szenen ausgeführt wurden, wurden die beiden quadratischen Seitenflächen des umgebenden Einheitswürfels, die einem Verdeckungsbaum zugrunde liegen, jeweils nach einer dieser Strategien unterteilt.

Experiment 3.9 (Messung des Speicherbedarfs bzw. der Berechnungszeit der hierarchischen Verdeckungsbäume)

> **Verfahren:** *Rekursive Konstruktion der hierarchischen Verdeckungsbäume.*
>
> **Eingabe:** *1 Dreieck.*
>
> **Parameter:** *Die verschiedenen Unterteilungsverfahren.*
>
> **Plattform:** *SGI O^2, R5000, 180MHz.*
>
> **Gemessene Größe(n):** *Vgl. Abbildung 3.26. Rechts: für die hierarchischen Verdeckungsbäume benötigter Speicher (y-Achse), in Abhängigkeit von der Überdeckung der Sichtbarkeit durch das Dreieck (x-Achse). Links: für die Konstruktion der hierarchischen Verdeckungsbäume benötigte Zeit (y-Achse), in Abhängigkeit von der Überdeckung der Sichtbarkeit durch das Dreieck (x-Achse).*
>
> **Beobachtung:** *Durch die geringere Anzahl der Unterteilungsschritte im Vergleich zu den anderen Verfahren resultiert die Dreiecksunterteilung in dem geringsten Speicherbedarf und der geringsten Berechnungszeit für alle Ver-*

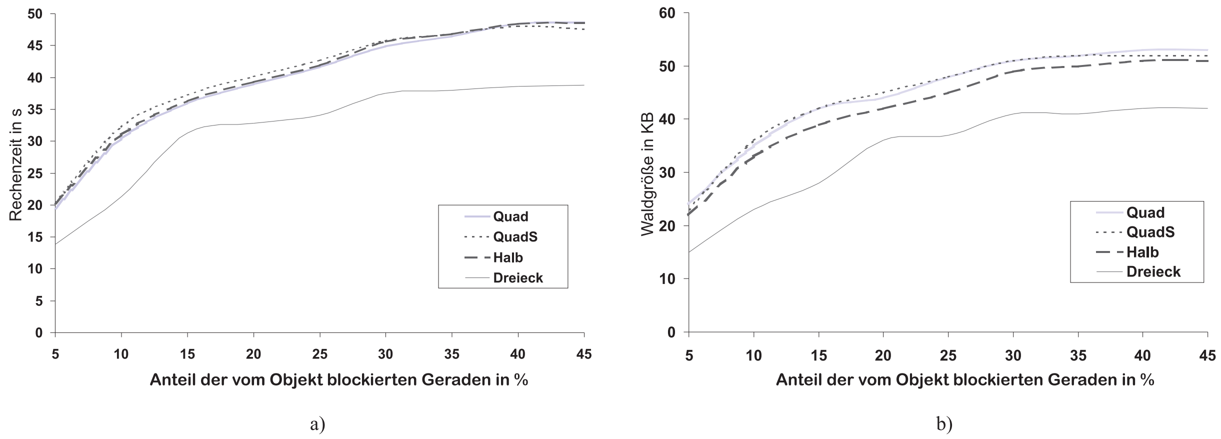


Abbildung 3.26: Vergleich der Unterteilungsverfahren anhand der Laufzeit und des Speicherbedarfs für unterschiedlich große Dreiecke

> *deckungsbäume. Alle anderen Verfahren haben einen zueinander grob ähnlichen Verlauf.*

Zum Abschluß sei noch auf einen weiteren interessanten Aspekt dieses Ansatzes hingewiesen. Die grundlegenden Operationen sind nämlich ähnlich denen, die schon in der Computergraphik–Rendering–Hardware implementiert sind. Dies könnte vorteilhaft für die Beschleunigung der Berechnung von Verdeckungsbäumen genutzt werden.

Kapitel 4

Approximative Lösung von Anfrageproblemen durch Abtastanfragen

Geometrische Anfrageprobleme können beispielsweise dadurch approximativ gelöst werden, daß zunächst die Antworten für eine Menge repräsentativer Anfragen vorberechnet werden. Für eine beliebige Anfrage wird dann die Antwort des nächstliegenden Repräsentanten zurückgegeben. Diese Antwort wird im allgemeinen nicht korrekt sein. In diesem Kapitel wird ein Rahmen zur Charakterisierung der Approximationsqualität dieser und ähnlicher Ansätze angegeben. In diesem Rahmen werden Aussagen für den Zeit- und Speicherplatzbedarf gemacht, die den Trade-off zwischen Approximationsqualität und Aufwand beschreiben. Ferner werden Verfahren zur effizienten Speicherung der Abtastantworten vorgeschlagen. Es werden empirische Untersuchungen für Anfragen mit Strecken beziehungsweise Geraden zur Analyse des Verhaltens dieses Lösungsansatzes angestellt.

4.1 Anfrageprobleme

Die vorangegangenen Kapitel konzentrierten sich auf die Aufgabe, bei einer gegebenen Szene die Sichtbarkeits- oder Verdeckungsregion zwischen zwei beliebigen Elementen **a** und **b** herauszufinden, die zur Szene gehören oder vom gleichen Typ wie die Szenenelemente sind, diese aber nicht schneiden. Die Vorgehensweise war dabei im wesentlichen, den Geradenraum in eine Datenstruktur vorzuverarbeiten, so daß für die Menge der Geraden, die **a** und **b** schneiden, effizient herausgefunden werden kann, welche von Szenenelementen blockiert werden und welche nicht. Ausgangspunkt für die Datenstrukturen, die zur Vorverarbeitung verwendet wurden, waren dabei im Prinzip solche zur Beantwortung der Anfrage mit einer einzelnen Geraden, d.h. für eine beliebige Anfragegerade herauszufinden, ob sie von einem Szenenelement geschnitten wird oder nicht, oder etwas umfassender, welche Szenenelemente die Gerade schneiden:

Definition 4.1 (Geraden-Anfrageproblem)
Eingabe: *Eine endliche Szene S, bestehend aus Strecken in der Ebene.*
Ausgabe: *Die Menge der Strecken aus S, die von einer beliebigen Anfragegeraden l geschnitten werden.*

Zur Lösung des allgemeinen Sichtbarkeitsproblems wurden dann geeignete Suchstrategien in diesen Datenstrukturen entwickelt.

Die Verwendung von Geraden zur Beschreibung von Sichtbarkeit machte es notwendig, die Szenenelemente bei der Beantwortung der Anfrage auszuschließen, mit denen Geraden nur Schnittpunkte gemeinsam haben, die nicht zwischen den beiden Anfrageelementen liegen. Das erforderte etwa, die Verdeckungsbäume hierarchisch über Teilszenen zu konstruieren. Dies kann vermieden werden, wenn anstelle von Geraden Strecken verwendet werden. Zur Beschreibung einer Sichtbarkeitsregion zwischen zwei Anfrageelementen werden dann die Strecken herangezogen, die Endpunkte auf den Oberflächen beider Elemente haben. Die Beschränkung der Anfrage auf eine Strecke führt auf das folgende Streckenanfrageproblem:

Definition 4.2 (Strecken-Anfrageproblem)
Eingabe: *Eine endliche Szene S, bestehend aus Strecken in der Ebene.*
Ausgabe: *Die Menge der Strecken aus S, die von einer beliebigen Anfragestrecke s geschnitten werden.*

Der genannte Vorteil wird durch eine höhere Dimension des Anfrageraums erkauft. Eine Strecke in der Ebene ist durch vier Werte festgelegt: die je zwei Koordinaten ihrer beiden Endpunkte. Diesem vierdimensionalen Raum steht der zweidimensionale Raum der R-Dualisierung von Geraden gegenüber. Im dreidimensionalen Raum sind dies sechs Dimensionen gegenüber vier Dimensionen bei der R-Dualisierung.

Das Geraden- und das Strecken-Anfrageproblem sind klassische Probleme der algorithmischen Geometrie. Es gibt viele andere Anfrageprobleme ähnlichen Typs, die aus anderen Anwendungsbereichen stammen. Dieses wird deutlich, wenn man sich an die Bedingung für den Schnitt einer Geraden mit einer Strecke aus Kapitel 2.1 erinnert beziehungsweise die folgende mathematische Formulierung für den Schnitt einer Anfragestrecke $\mathbf{q}_1, \mathbf{q}_2$ mit einer Strecke $\mathbf{p}_1, \mathbf{p}_2$ der Szene betrachtet:

$$\mathbf{q}_1 + \lambda(\mathbf{q}_2 - \mathbf{q}_1) = \mathbf{p}_1 + \mu(\mathbf{p}_2 - \mathbf{p}_1),\ 0 \leq \lambda \leq 1,\ 0 \leq \mu \leq 1.$$

Dazu äquivalent ist

$$f_i(\mathbf{p}_1, \mathbf{p}_2, \mathbf{q}_1, \mathbf{q}_2) \geq 0,\ i = 1, \ldots, 4,$$

mit

$$f_1 := -h_1/h_0,\ f_2 := h_1/h_0 + 1,\ f_3 := -h_2/h_1,\ f_4 := h_2/h_1 + 1,$$

$$h_0 := |(\mathbf{q}_2 - \mathbf{q}_1) \cdot (\mathbf{p}_2 - \mathbf{p}_1)|, h_1 := |(\mathbf{p}_1 - \mathbf{q}_1) \cdot (\mathbf{p}_2 - \mathbf{p}_1)|, h_2 := |(\mathbf{q}_2 - \mathbf{q}_1) \cdot (\mathbf{p}_1 - \mathbf{q}_1)|.$$

Die entsprechende allgemeine Klasse von Anfrageproblemen ist:

Definition 4.3 (Anfrageproblem)
Eingabe: *Eine Menge von Szenenelementen* $\mathbf{p}_i,\ i = 1, \ldots, n$,
ein Anfragetyp $\mathbf{f} : \mathbb{R}^{p+q} \to \mathbb{R}^r,\ \mathbf{p} \in \mathbb{R}^p,\ \mathbf{q} \in \mathbb{R}^q$. $\mathbb{R}^q$ *heißt* Anfrageraum.

Ausgabe: *Für eine beliebige* Anfrage $\mathbf{q}$ *die dazugehörende* Antwortmenge $R(\mathbf{q})$*, bestehend aus den Elementen* $\mathbf{p}_i$ *von* S*, für die*

$$\mathbf{f}(\mathbf{p}_i, \mathbf{q}) \geq \mathbf{0},\ i = 1, \ldots, n$$

gilt, wobei die "$\geq$*"-Relation komponentenweise anzuwenden ist.*

Für das Geradenanfrageproblem ist $p = 4$, $q = 2$ und $r = 2$, für das Streckenanfrageproblem $p = q = r = 4$. $\mathbf{f}(\mathbf{p}_i, \mathbf{q})$ legt den Typ des Anfrageproblems fest. $\mathbf{p}_i$ ist der Szenenelementparameter, $\mathbf{q}$ der Anfrageparameter.

Für viele dieser Probleme existieren spezielle effiziente algorithmische Lösungen. Diese verarbeiten die Eingabe während der Vorverarbeitung in eine Datenstruktur, die es erlaubt, beliebige Anfragen schnell zu beantworten. Für das Streckenanfrageproblem ist eine Lösung mit Speicherbedarf $S(n) = O(n^2)$ und Anfragezeit $Q(n) = O(\log n + k)$ für sich nicht schneidende Szenenstrecken bekannt [Cha82].

Bei der Betrachtung des Problems in der allgemeinen Form können zwei divergierende Ansätze zur Lösung unterschieden werden. Beim ersten wird die Szene, beim zweiten werden die Anfragen im Parameterraum repräsentiert:

Definition 4.4 (Repräsentation von Anfrageproblemen)
Parameterraum-Repräsentation der Szene: *Die Elemente der Szene werden als Punkte* $\mathbf{p}_i$ *in* $\mathbb{R}^p$ *repräsentiert. Eine Anfrage* $\mathbf{q}^*$ *definiert einen Bereich*

$$Q(\mathbf{q}^*) := \{\mathbf{p} \mid \mathbf{f}(\mathbf{p}, \mathbf{q}^*) \geq 0\}.$$

Von Interesse sind die Punkte $\mathbf{p}_i$*, die in* $Q(\mathbf{q}^*)$ *liegen (vgl. Abbildung 4.1).*

Parameterraum-Repräsentation der Anfragen: *Die Elemente der Szene werden als Regionen*

$$S(\mathbf{p}_i) := \{\mathbf{q} \mid \mathbf{f}(\mathbf{p}_i, \mathbf{q}) \geq 0\}$$

in $\mathbb{R}^q$ *repräsentiert. Eine Anfrage definiert einen Punkt* $\mathbf{q}^*$*. Von Interesse sind die Regionen, die die Bedingung* $\mathbf{q}^* \in S(\mathbf{p}_i)$ *erfüllen (vgl. Abbildung 4.2).*

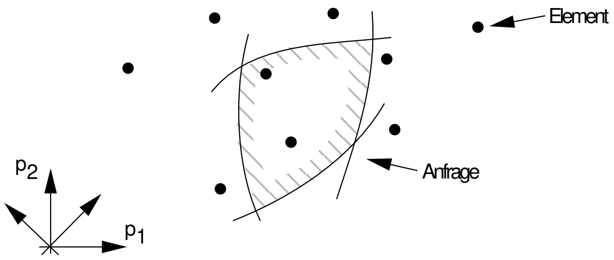


Abbildung 4.1: Parameterraum-Repräsentation der Szene

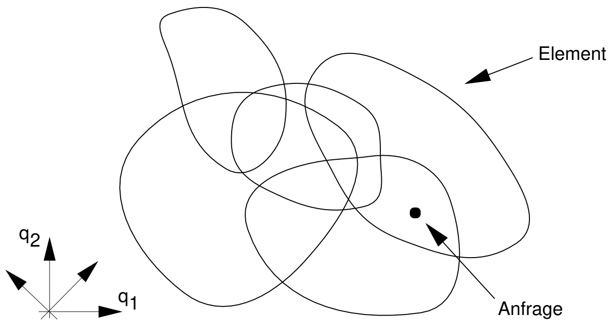


Abbildung 4.2: Parameterraum-Repräsentation der Anfragen

Der erste Ansatz führt bei einer großen Klasse von Problemtypen auf Lösungsverfahren mit geringem Speicherbedarf, d.h. nahezu linearem Speicherbedarf in der Größe n der Szene, bei korrespondierenden Anfragezeiten in $O(n^\alpha)$, α eine Konstante zwischen 0 und 1 [YY85]. Die zweite Version ist die Basis für Lösungen mit polylogarithmischen Anfragezeiten, d.h. $O(\log^i n)$ für eine Konstante $i \geq 1$, die allerdings deutlich mehr als linear viel Speicher benötigen, nämlich polynomiell mit einem Grad gößer als 1 [Cha85]. Ein kontinuierlicher trade-off zwischen beiden Extremen ist möglich.

Alle solche Ansätze sind aufgrund der komplexen geometrischen Strukturen, die involviert sind, schwer zu implementieren. Im folgenden wird ein heuristischer Ansatz diskutiert, der auf Abtastung (engl. *sampling*) beruht. Abtastung bedeutet in diesem Fall, daß zunächst die Antworten für eine Menge möglichst repräsentativer Anfragen vorberechnet werden. Diese sogenannten *Abtastanfragen* werden dann so in eine Datenstruktur vorverarbeitet, daß anschließend für eine beliebige Anfrage aus einem oder mehreren guten Repräsentanten schnell eine Antwortmenge für die Anfrage zu finden ist.

4.2 Abtastung

Die folgenden Betrachtungen zur Abtastung basieren auf der Parameterraum-Repräsentation des Anfrageraums eines Anfrageproblems. Es werden zwei Abtastarten unterschieden, die *Punktabtastung* und die *Gebietsabtastung*:

Definition 4.5 (Punktabtastung und Gebietsabtastung)
Die Aufgabe, für ein Gebiet G des Anfrageraums eines Anfrageproblems die Menge aller Elemente zu finden, die zur Antwortmenge irgendeiner Anfrage in G gehört, wird als Gebietsanfrage *bezeichnet. Diese Menge wird als* Antwortmenge *der Gebietsanfrage bezeichnet. Ein Element der Antwortmenge einer Gebietsanfrage heißt* überdeckend, *wenn es zur Antwortmenge aller Anfragen des Anfragegebietes gehört. Zur Unterscheidung wird eine Anfrage entsprechend der Definition eines Anfrageproblems* Punktanfrage *genannt.*

Unter einer Punktabtastung *wird eine endliche Menge Q von Punktanfragen verstanden. Die Familie*

$$\mathcal{P} = \{R(\mathbf{q}) \mid \mathbf{q} \in Q\}, \textit{mit } R(\mathbf{q}) \textit{ die Antwortmenge einer Anfrage } \mathbf{q},$$

der Antworten wird Abtastantwort *der Punktabtastung genannt.*

Eine Gebietsabtastung *ist durch eine endliche Menge $\mathcal{Q}$ von Gebietsanfragen gegeben. Die* Abtastantwort *einer Gebietsabtastung $\mathcal{Q}$ setzt sich aus den Antworten der Gebietsanfragen in $\mathcal{Q}$ zusammen:*

$$\mathcal{G} = \{R(G) \mid G \in \mathcal{Q}\}, \text{mit } R(G) = \bigcup_{\mathbf{q} \in G} R(\mathbf{q}).$$

Ein Gebiet im q-dimensionalen Raum ist eine zusammenhängende offene Menge. Ein nichtleeres Gebiet enthält dementsprechend unendlich viele Punkte. Damit ist es ausgeschlossen, die Antwortmenge einer Gebietsanfrage algorithmisch als Vereinigung der Antwortmengen aller in ihm enthaltenen Punktanfragen zu bestimmen. Dies macht es zunächst unmöglich, Verfahren zur Beantwortung von Punktanfragen unmittelbar dafür einzusetzen. Für den Typ von Anfrageproblemen, der hier betrachtet wird, ist es jedoch prinzipiell möglich, auch Gebietsanfragen zu behandeln. Sie bedeuten letztendlich nur, daß in der Parameterrepräsentation des Anfrageraums das Gebiet einer Punktanfrage durch das Gebiet einer Gebietsanfrage ersetzt wird. Dabei kann es möglich sein, Datenstrukturen für die Beantwortung von Punktanfragen auch für die Beantwortung von Gebietsanfragen effizient einzusetzen. Dies war die Vorgehensweise in Kapitel 2 dieser Arbeit.

4.3 Antwortgenerierung

Die prinzipielle Vorgehensweise bei der Lösung von Anfrageproblemen durch Abtastung besteht darin, zunächst eine Abtastung festzulegen und die entsprechende Abtastantwort zu berechnen. Aus den Abtastantworten wird dann für eine beliebige Anfrage eine Antwort generiert. Hierfür sind verschiedene Strategien möglich, die im folgenden auf der Grundlage der Punktabtastung (Kapitel 4.3.1) und der Gebietsabtastung (Kapitel 4.3.2) dargestellt werden.

4.3.1 Punktabtastung

Die wohl naheliegendste Vorgehensweise zur Beantwortung von Punktanfragen für den Fall der Punktabtastung ist, die Antwortmengen der Abtastanfragen heranzuziehen, die in der Nachbarschaft der Punktanfrage liegen. Hier lassen sich etwa folgende zweckmäßige Vorgehensweisen unterscheiden:

Definition 4.6 (Approximative Antworttypen für Punktanfragen bei Punktabtastung)
Die Nächste-Nachbar-Antwort *einer Punktanfrage bei Punktabtastung ist die Antwortmenge einer Abtastanfrage, die der Punktanfrage bezüglich der Metrik des Anfrageraums am nächsten liegt.*

Die k-Nächster-Nachbar-Antwort *einer Punktanfrage ist die Vereinigung der Antwortmengen von k Abtastanfragen, die der Punktanfrage am nächsten liegen.*

Die auf eine dieser Arten gewonnene Antwort wird im allgemeinen nicht korrekt sein. Abbildung 4.3 zeigt eine Szene aus sechs Elementen, numeriert mit $1, \dots, 6$. Ferner sind vier Abtastanfragen, repräsentiert durch Punkte $\mathbf{a}, \dots, \mathbf{d}$, dargestellt. Wird nun mit dem beliebigen Anfragepunkt $\mathbf{q}$ aus Abbildung 4.3 angefragt, so wird als nächstliegende Abtastanfrage die Anfrage $\mathbf{c}$ gefunden. Als Nächste-Nachbar-Antwort auf die Anfrage $\mathbf{q}$ wird die Antwortmenge von $\mathbf{c}$, nämlich $\{4, 6\}$ ausgegeben. Es kann jedoch leicht festgestellt werden, daß die korrekte Antwort $\{2, 4\}$ ist.

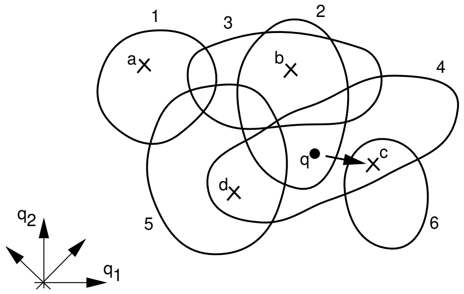


Abbildung 4.3: Vorverarbeitung der Abtastanfragen

Es gibt zwei Wege, mit diesem Problem umzugehen. Der erste ist, zu versuchen, die falschen Antworten zu korrigieren; der zweite, die falschen Antworten zu akzeptieren. Letzteres ist etwa im Zusammenhang mit Sichtbarkeitsproblemen durchaus möglich, da es bei der Bildsynthese meist nur auf den visuellen Eindruck ankommt, der häufig auch leichte Abweichungen von der korrekten Lösung kaum erkennen läßt.

Unter den Elementen einer näherungsweisen Antwort können folgende Typen unterschieden werden:

Definition 4.7 (Typen von Elementen einer approximativen Antwort)
Gegeben seien eine korrekte Antwort A und eine approximative Antwort $\tilde{A}$. Dann heißen die Elemente in $A \cap \tilde{A}$ korrekt ausgegeben*, die Elemente in $\tilde{A} \setminus A$* falsch ausgegeben *und die Elemente in $A \setminus \tilde{A}$* nicht ausgegeben*.*

Unter der Annahme, daß für ein Element getestet werden kann, ob es zur Antwortmenge einer Anfrage gehört, lassen sich die falsch ausgegebenen Elemente einer approximativen Antwort eliminieren:

Definition 4.8 (Korrigierte approximative Antwort)
Gegeben seien eine korrekte Antwort A und eine approximative Antwort $\tilde{A}$. Dann wird die Menge der korrekten Elemente $A \cap \tilde{A}$ in $\tilde{A}$ als korrigierte Antwort *zu $\tilde{A}$ bezeichnet. $\tilde{A}$ heißt* korrigierbar*, wenn $\tilde{A} \supseteq A$.*

Die korrigierte Antwort zur Anfrage in Abbildung 4.3 ist $\{4\}$. Die approximative Antwort ist also in diesem Fall nicht korrigierbar.

Die Qualität einer approximativen Antwort kann wie folgt quantifiziert werden:

Definition 4.9 (Approximationsfehler)
Sei A die korrekte Antwortmenge einer Anfrage, $\tilde{A}$ eine approximative Antwortmenge. Dann heißt

$$e_a(A, \tilde{A}) := |A \Delta \tilde{A}|$$

absoluter Approximationsfehler*,*

$$e_r(A, \tilde{A}) := |A \Delta \tilde{A}| / |A|$$

relativer Approximationsfehler*. Δ bezeichnet dabei die symmetrische Differenz.*

Um einen Eindruck vom Verhalten des Abtastansatzes zu erhalten, wurden die folgenden Experimente für das Streckenanfrageproblem durchgeführt.

Experiment 4.1 (Vergleich von Abtastantworten)

Eingabe: *Eine Szene von 50 zufällig generierten Strecken innerhalb eines Quadrates. Die Koordinaten der Endpunkte der Strecken sind gleichverteilt.*

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Vgl. Abbildung 4.4: mittlere Anzahl der unterschiedlichen Antworten über die Anfragen (y-Achse) in Abhängigkeit von der Anzahl der Abtastanfragen (x-Achse).*

Beobachtung: *Bis zu ungefähr 10 000 Abtastanfragen beträgt die Steigung der Kurve der Streckenanfrage 1. Das heißt, daß alle Anfragen eine un-*

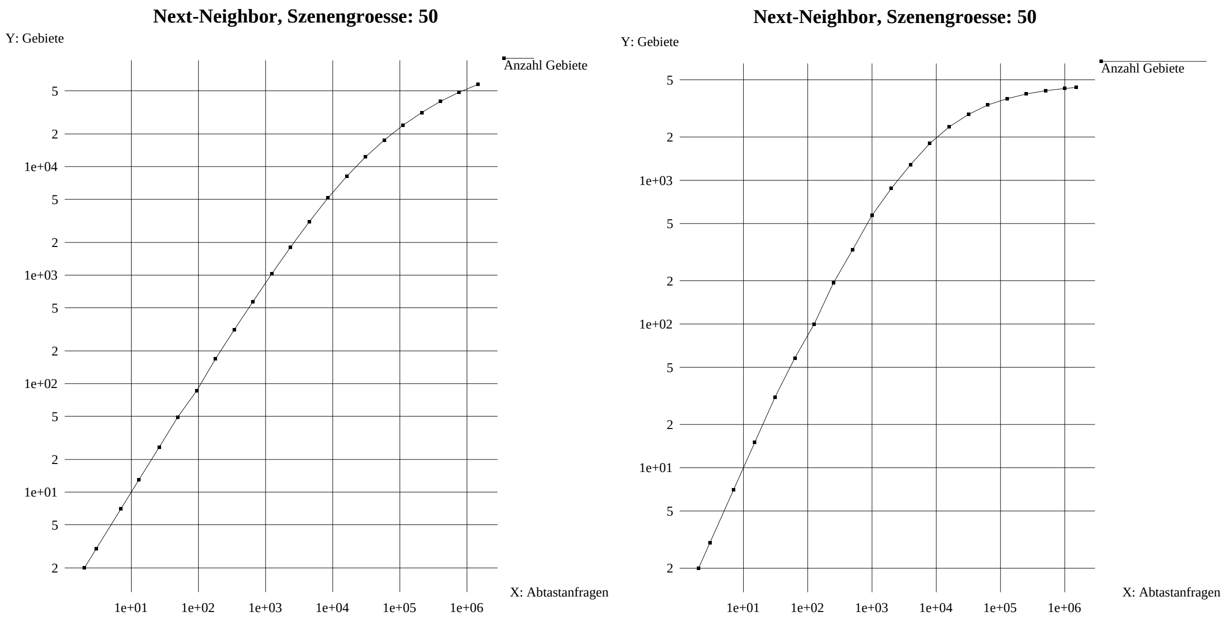


Abbildung 4.4: Anzahl der unterschiedlichen Antworten auf die Abtastanfragen, abhängig von der Anzahl der Abtastanfragen für die Anfrage mit Strecken (links) bzw. mit Geraden (rechts)

terschiedliche Antwortmenge haben. Daher ist die Chance klein, daß ein beliebiger Anfragepunkt dieselbe Antwortmenge hat wie eine der Abtastanfragen. Also ist die Chance, eine korrekte Antwort zu bekommen, ebenso klein. Diese Chance erhöht sich in Regionen der Kurve mit abnehmender Steigung.

Die Kurve der Geradenanfragen verläuft flacher als die der Streckenanfragen. Das spiegelt die unterschiedliche Komplexität der beiden Anfragetypen wider.

Experiment 4.2 (Vergleich von exakter und korrigierter approximativer Antwort)

Eingabe: *Eine Szene wie zuvor sowie 200 zufällige Abtastanfragen.*

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Vgl. Abbildung 4.5: Verhältnis des absoluten Approximationsfehlers der korrigierten Antwort zum absoluten Approximationsfehler der nicht korrigierten Antwort, gemittelt über die 200 (links) beziehungsweise 100 (rechts) Anfragen (y-Achse, skaliert mit* $1/10$*) in Abhängigkeit von der Anzahl der Abtastanfragen (x-Achse).*

Beobachtung: *Obwohl die Kurven ein wenig nervös erscheinen, können sie als oszillierend um den Wert* $1/2$ *beschrieben werden. Das heißt, daß un-*

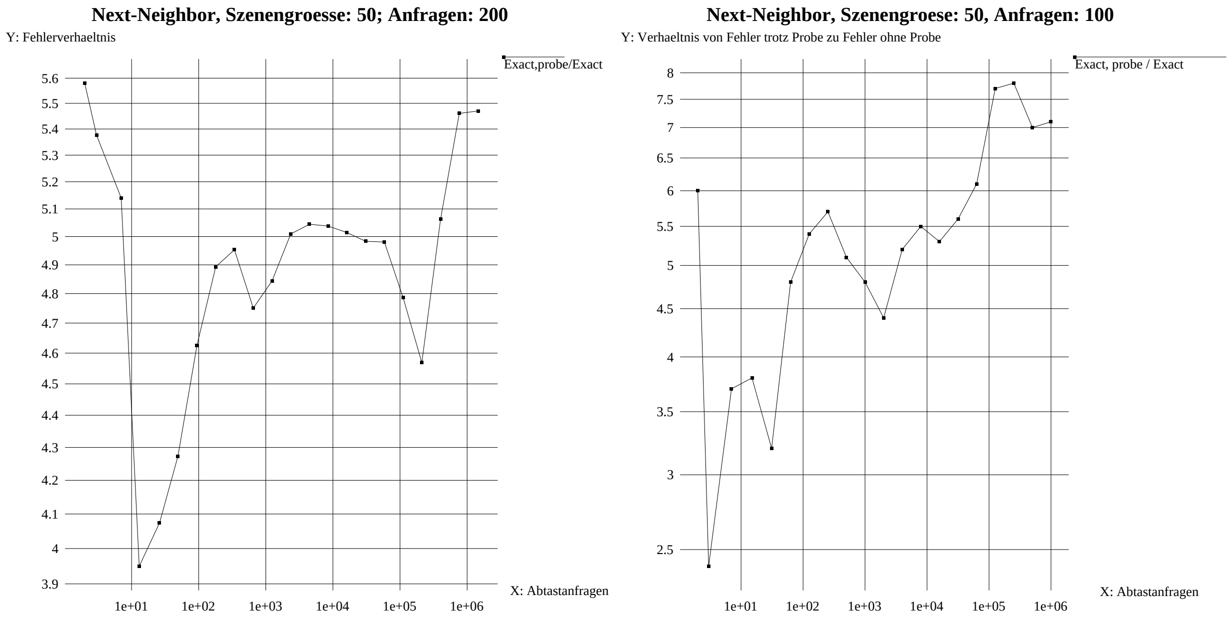


Abbildung 4.5: Verhältnis des absoluten Approximationsfehlers der korrigierten Antwort zum absoluten Approximationsfehler der nichtkorrigierten Antwort (y-Achse, skaliert mit 1/10) in Abhängigkeit von der Anzahl der Abtastanfragen (x-Achse), für die Anfrage mit Strecken (links) und Geraden (rechts)

gefähr 50 Prozent des Fehlers durch das Testen der Antwortmenge auf falsch genannte Elemente eliminiert werden kann. Die gleiche Anzahl an falsch ausgegebenen und nicht ausgegebenen Elementen könnte in diesem Beispiel durch die zufällig generierte Szene verursacht werden. In diesem Fall sollten sich Anzahl der Szenenelemente, die zur Anfrageantwort, aber nicht zur nächst-benachbarten Abtastanfrageantwort gehören, und die Anzahl der Szenenelemente, die zur nächsten Abtastanfrageantwort, aber nicht zur Anfrageantwort gehören, ungefähr gleich verhalten – aufgrund der Zufälligkeit gibt es keinen Grund, warum eine der Anfragen bevorzugt sein sollte.

Experiment 4.3 (Durchschnittliche Anzahl an Fehlerstrecken)

Eingabe: *Wie zuvor.*

Plattform: *SGI Indy, R4000, 100MHz.*

Gemessene Größe(n): *Vgl. Abbildung 4.6: durchschnittlicher absoluter Approximationsfehler der nichtkorrigierten Antwort (obere Kurve) und der korrigierten Antwort (untere Kurve) (y-Achse) in Abhängigkeit von der Anzahl der Abtastanfragen (x-Achse).*

Beobachtung: *Bei 10 000 Abtastanfragen bleibt bei der Streckenanfrage durchschnittlich eine nicht ausgegebene Strecke übrig. Bei der Geradenanfrage sind es nur noch durchschnittlich ca.* 0.18 *Strecken. Gleichzeitig bleiben jeweils ca. genauso viele „absolut" falsch ausgegebene Strecken zurück.*

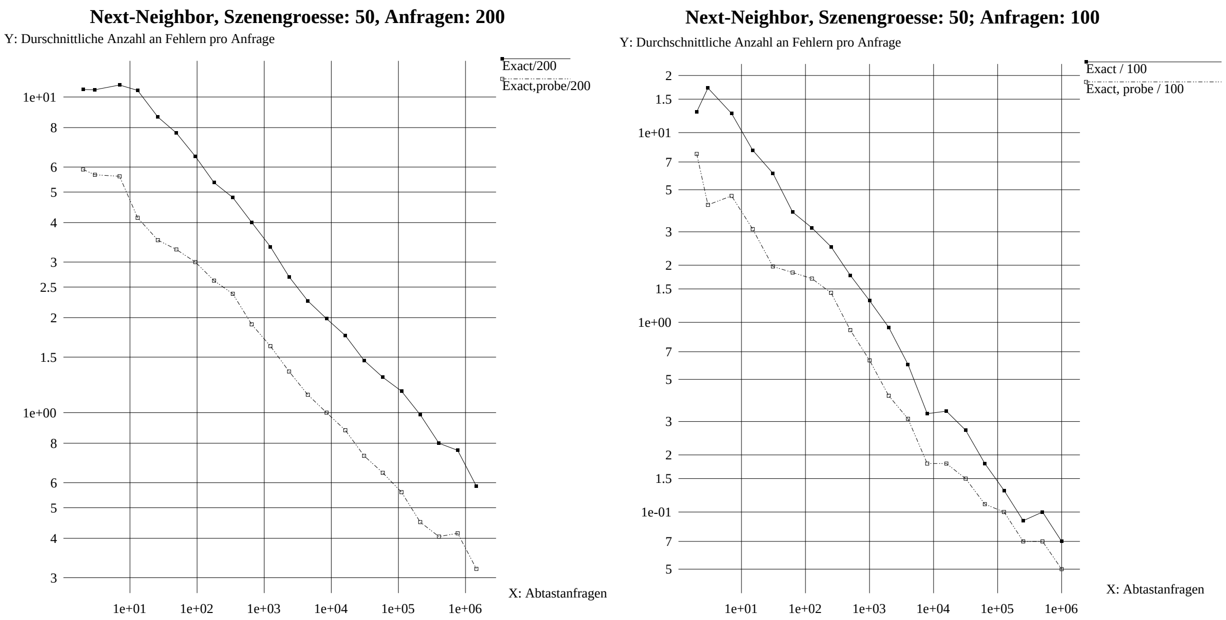


Abbildung 4.6: Durchschnittlicher Approximationsfehler in Abhängigkeit von der Anzahl der Abtastanfragen, für Anfragen mit Strecken (links) und Geraden (rechts). Die untere Kurve repräsentiert jeweils den absoluten Fehler für die korrigierte Antwort, die obere für die nicht korrigierte Antwort.

Experiment 4.4 (Relative Anzahl an Fehlerstrecken)

Eingabe: *Wie zuvor.*

Plattform: *SGI Indy, R4000, 100MHz.*

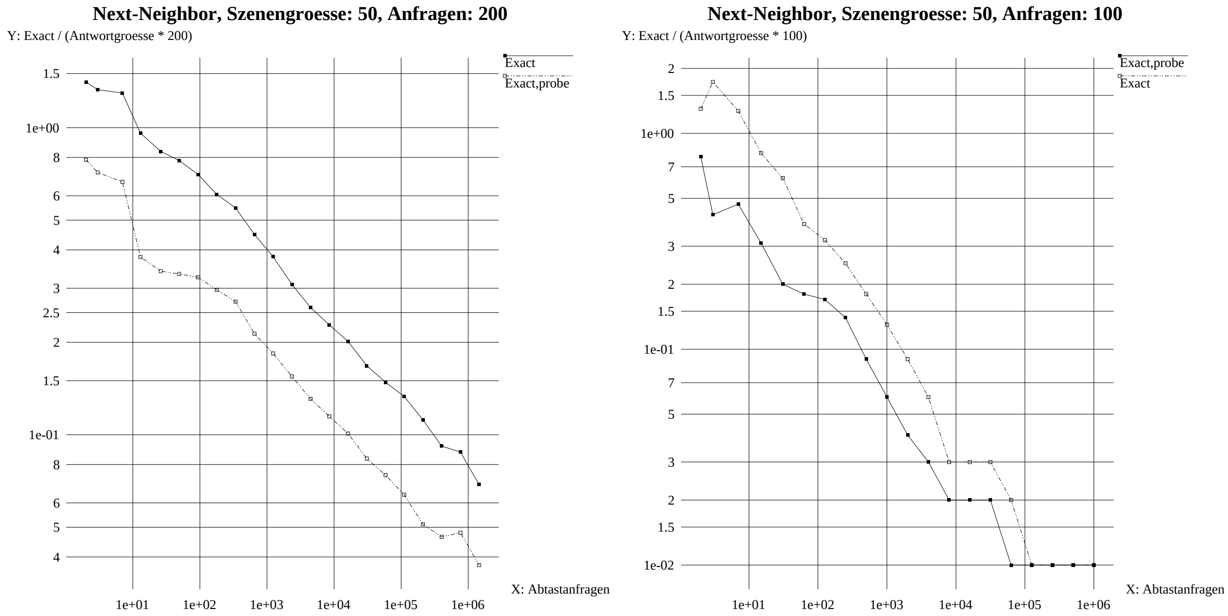


Abbildung 4.7: Durchschnittlicher relativer Approximationsfehler für die nicht korrigierte Antwort (obere Kurve) und die korrigierte Antwort (untere Kurve) in Abhängigkeit von der Anzahl der Abtastanfragen, für Anfragen mit Geraden (links) und Strecken (rechts)

Gemessene Größe(n): *Vgl. Abbildung 4.7: durchschnittlicher relativer Approximationsfehler für die nicht korrigierte Antwort (obere Kurve) und die korrigierte Antwort (untere Kurve) (y-Achse) in Abhängigkeit von der Anzahl der Abtastanfragen (x-Achse).*

Beobachtung: *Bei 10 000 Abtastanfragen liegt die relative Anzahl nach Korrektur nahe bei 10%, bei 100 Abtastanfragen liegt sie bei* 1/3.

Der maximale Approximationsfehler ist durch die Unterschiede der Antworten innerhalb eines Bereichs des Voronoidiagramms [Ede87] der Punkte bestimmt, die die Abtastanfrage repräsentieren. Das Voronoigebiet einer Abtastanfrage **q** ist das Gebiet aller Punkte des Anfrageraums, deren Abstand zu **q** nicht größer als zu allen anderen Abtastanfragen ist. Das Voronoidiagramm besteht aus den Voronoigebieten aller Abtastpunkte. Abbildung 4.8 zeigt das Voronoidiagramm der Abtastpunkte des Beispiels.

Mit dieser Terminologie gilt nun folgende Beobachtung:

Theorem 4.1 (Einflußmenge) *Nur Szenenelemente, die das Voronoigebiet der nächstgelegenen Abtastanfrage* **q** *einer gegebenen Anfrage* $\mathbf{q}'$ *schneiden, tragen zum Fehler der Nächste-Nachbar-Antwortmenge von* $\mathbf{q}'$ *bei.*

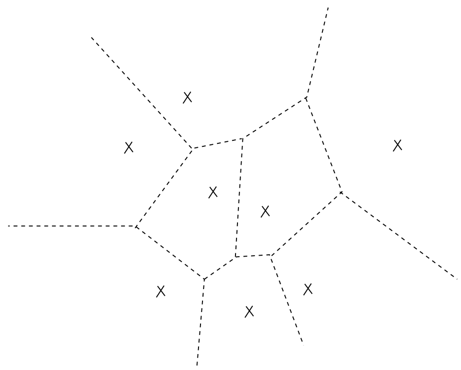


Abbildung 4.8: Eine Szene mit Abtastanfragen und dem überlagerten Voronoidiagramm der Abtastanfragen

Beweis: Zum Fehler einer Antwortmenge tragen die falsch ausgegebenen und die nicht ausgegebenen Szenenelemente bei. Die falsch ausgegebenen Szenenelemente enthalten $\mathbf{q}$ und schneiden damit das Voronoigebiet von $\mathbf{q}$, da $\mathbf{q}$ in seinem Voronoigebiet liegt. Die nicht ausgegebenen Szenenelemente enthalten $\mathbf{q}'$, da sie zur Antwort von $\mathbf{q}'$ gehören. Da $\mathbf{q}'$ im Voronoigebiet von $\mathbf{q}$ liegt, schneiden auch diese Szenenelemente dieses Voronoigebiet. □

Damit kann die Qualität einer Punktabtastung wie folgt formal beschrieben werden:

Definition 4.10 (ε-Optimalität)
Sei $\varepsilon > 0$. Eine Punktabtastung eines Anfrageproblems wird absolut *bzw.* relativ *ε-optimal genannt, wenn sie von minimaler Größe unter allen endlichen Teilmengen X des Anfrageraums ist, deren Voronoigebiete einen ε-Fehler garantieren, d.h.*

$$\max\{e(R(\mathbf{q}), R(\mathbf{x})) \mid \mathbf{q} \in V(\mathbf{x}),\ \mathbf{x} \in X\} \leq \varepsilon,$$

mit

$V(\mathbf{x})$ *das Voronoigebiet von* $\mathbf{x}$

$R(\mathbf{x})$ *die Szenenelemente, die* $\mathbf{x}$ *enthalten*

$R(\mathbf{q})$ *die Antwortmenge einer Anfrage* $\mathbf{q}$

$e(.,.)$ *der absolute bzw. relative Fehler.*

Der Aufwand zum Auffinden einer ε-optimalen Punktabtastung ist unbekannt. Im folgenden wird statt dessen eine approximierende Lösung eines ähnlich gelagerten Problems entwickelt. Die Idee ist, anstelle von Voronoigebieten (Hyper)kugeln um die Abtastanfragen zu legen und nach dem größten Radius zu fragen, der einen ε-Fehler garantiert. Da Kugeln im Unterschied zu Voronoigebieten beschränkt sind, wird das Gebiet, aus dem die Anfragen stammen können, als beschränkt angenommen. Als Punktabtastung wird die Menge der Mittelpunkte der Kugeln einer minimalen Überdeckung des Anfragegebiets mit solchen (Hyper)Kugeln genommen.

Der Zusammenhang dieser Punktabtastung mit dem Originalproblem wird über das Potenzdiagramm [Ede87, Aur87] klar. Ein *Potenzdiagramm* einer endlichen Menge von Punkten $\mathbf{p}_i$ mit nichtnegativen reellen *Potenz*-Werten w_i besteht aus Zellen, die solche Punkte enthalten, für die die Potenz $d(\mathbf{p}, \mathbf{p}_j, w_j) := d(\mathbf{p}, \mathbf{p}_j)^2 - {w_j}^2$ zu einem speziellen Punkt $\mathbf{p}_j$ kleiner oder gleich als die Potenz zu allen anderen Punkten $\mathbf{p}_i$ ist.

Eine Kugel einer Kugelüberdeckung wird *Randkugel* genannt, falls ihr Rand nicht komplett von anderen Kugeln der Überdeckung überdeckt ist.

Mit diesen Definitionen ergibt sich:

Theorem 4.2 *Seien eine Überdeckung mit Kugeln sowie das Potenzdiagramm der Mittelpunkte der Kugeln, mit den Radii der Kugeln als Potenz, gegeben. Dann ist das Potenzgebiet des Mittelpunktes einer Kugel, die nicht Randkugel ist, eine Untermenge dieser Kugel.*

Beweis: Eine Kugel ist keine Randkugel, wenn ihr Rand komplett von anderen Kugeln überdeckt ist. Sei P die Menge aller Mittelpunkte aller Kugeln, die eine hier betrachtete Kugel $U(\mathbf{q})$ schneiden. Das Voronoigebiet ihres Mittelpunktes $\mathbf{q}$ resultiert mindestens aus dem Schnitt der Halbräume, die durch $\mathbf{q}$ und die Punkte $\mathbf{p} \in P$ definiert werden. Der Schnitt der Kugeln $U(\mathbf{p})$ mit dem Rand von $U(\mathbf{q})$ liegt komplett außerhalb des betreffenden Halbraums. Daher liegt der Rand von $U(\mathbf{q})$ komplett außerhalb des Schnitts aller Halbräume. Also liegt das Voronoigebiet als eine Teilmenge dieses Schnitts komplett innerhalb von $U(\mathbf{q})$. $\square$

Als unmittelbare Konsequenz ergibt sich:

Korollar 4.1 *Sei eine Überdeckung mit Kugeln gegeben, die einen ε-Fehler garantiert. Dann garantiert das Potenzgebiet derjenigen Kugelmittelpunkte, die nicht zu einem Rand-Voronoigebiet gehören, ebenfalls einen ε-Fehler.*

Eine weitere Konsequenz dieser Beobachtungen ist, daß für die Abtastmengen, die aus Überdeckungen mit Kugeln mit variablem Radius gewonnen werden, die Anfrage nach der nächstgelegenen Abtastanfrage durch die Anfrage nach der „nächstgelegenen gewichteten Abtastanfrage" ersetzt werden muß.

Die Frage ist nun, wie eine minimale Überdeckung mit Kugeln berechnet wird, die einen ε-Fehler garantiert. Dieses soll hier in einer allgemeineren Art und Weise diskutiert werden.

Definition 4.11 (f-Überdeckung)
Sei $M \subset I\!R^d$, $f : M \to I\!R_+$. Eine Menge $M_f \subseteq M$ wird f-Überdeckung *von M genannt, wenn die Vereinigung der Kugeln mit den Mittelpunkten $\mathbf{p} \in M_f$ und Radii $f(\mathbf{p})$ die Menge M überdecken.*

Die Beziehung mit dem Originalproblem wird durch die folgende Funktion für f einsichtig:

Definition 4.12 (Überdeckungsfunktion $r_\varepsilon(\mathbf{q})$)

$$r_\varepsilon(\mathbf{q}) := \sup\{r \mid U_r(\mathbf{q}) \text{ garantiert einen } \varepsilon - \text{Fehler}\} \text{ für } \mathbf{q} \in Q,$$

wobei Q der betrachtete Anfrageraum ist. $U_r(\mathbf{q})$ bezeichnet die offene Kugel mit Radius r und Mittelpunkt $\mathbf{q}$.

$U_r(\mathbf{q})$ garantiert einen ε-Fehler, wenn

$$\max\{e(R(\mathbf{q}), R(\mathbf{x})) \mid \mathbf{q} \textit{ Mittelpunkt von } U_r(\mathbf{q}),\ \mathbf{x} \in U_r(\mathbf{q})\} \le \varepsilon,$$

mit

$R(\mathbf{q})$ die Szenenelemente, die $\mathbf{q}$ enthalten

$R(\mathbf{x})$ die Antwortmenge einer Anfrage $\mathbf{x}$

$e(.,.)$ der absolute bzw. relative Fehler.

Die Struktur und algorithmische Berechnung von kleinen f-Überdeckungen hängen von den konkreten Eigenschaften von M und f ab. Abbildung 4.9 zeigt ein Beispiel für $f := r_\varepsilon$ und das absolute Fehlermaß für den eindimensionalen Anfrageraum $M = [0, 21]$. Wie zu sehen ist, ist r_ε im allgemeinen nicht stetig.

Bezüglich der Existenz der f-Überdeckung gilt das folgende:

Theorem 4.3 *Falls M eine kompakte Menge ist und $M \subseteq \bigcup_{\mathbf{q} \in M} U_r(\mathbf{q})$, dann existiert eine endliche f-Überdeckung.*

Beweis: Es ist eine bekannte Tatsache aus der Topologie, daß jede offene Überdeckung einer kompakten Menge eine endliche Teilüberdeckung besitzt. $\square$

Die Bedingung $M \subseteq \bigcup_{\mathbf{q} \in M}$ ist nicht immer erfüllt. Würde beispielsweise in dem Beispiel aus Abbildung 4.9 ε kleiner als 1 gewählt, würden Anfragen $\mathbf{q} \in M$ auftreten, für die $r_\varepsilon(\mathbf{q}) = 0$ ist. Solche Anfragen liegen in keiner der offenen Kugeln. Bei Anfrageräumen mit Dimension $d > 1$ kann es an Anfragen, in denen sich die Berandungsflächen von d Szenenobjekten schneiden, zu entsprechenden Schwierigkeiten mit $\varepsilon < \lceil d/2 \rceil$ kommen.

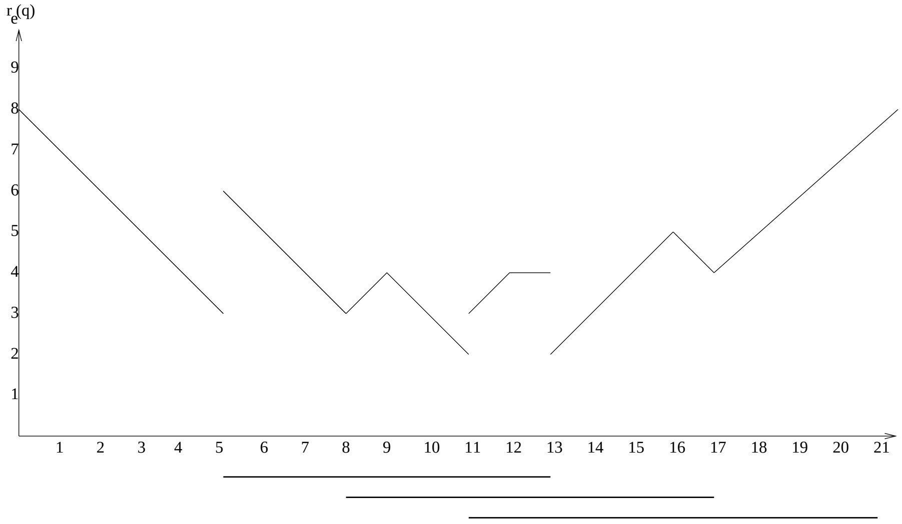


Abbildung 4.9: Ein eindimensionaler Anfrageraum und die korrespondierende Funktion r_ε, $\varepsilon = 1$, für das absolute Fehlermaß

Dies liegt an Anfragen $\mathbf{q}$, in denen sich die Berandungsflächen von d Szenenelementen schneiden. Diese Situation kann in d-dimensionalen Anfrageräumen in nichtdegenerierter Weise auftreten. Dort kann in jeder Umgebung von $\mathbf{q}$ jede beliebige Kombination dieser d Szenenelemente als Antwort auftreten.

Eine direkte Idee, um eine gute Überdeckung für den Fall zu bekommen, daß sie existiert, ist, Kugeln $U_r(\mathbf{q})$ zu benutzen, die maximal sind, d.h., es existiert keine andere Kugel $U_{r'}(\mathbf{q}')$, so daß gilt $U_r(\mathbf{q}) \subset U_{r'}(\mathbf{q}')$. Nicht-maximale Kugeln können wie folgt charakterisiert werden.

Theorem 4.4 *Sei $\mathbf{q}^* \in M$, $f \in C^1(U(\mathbf{q}^*))$ für eine offene Umgebung $U(\mathbf{q}^*) \subseteq M$. Falls es einen Pfad $\mathbf{q}(t) \subset U(\mathbf{q}^*)$, $t \in [-1,1]$, mit $\mathbf{q}(0) = \mathbf{q}^*$, $\mathbf{q}(t) \in C^1[-1,1]$, in Kurven-Längenparametrisierung gibt, so daß die gerichtete Ableitung von f bei $\mathbf{q}^*$ für die Richtung $\mathbf{q}'$ größer gleich 1 ist, dann ist die Kugel $U_{f(\mathbf{q})}$ nicht maximal.*

Beweis: Jede der Kugeln $U_{f(\mathbf{q}(t))}(\mathbf{q}(t))$, $t \in [0,1]$, überdeckt $U(\mathbf{q}^*)$. $\square$

Eine Konsequenz dieses Theorems ist, daß es ausreicht, nur die Mittelpunkte $\mathbf{q}$ für eine minimale Überdeckung zu betrachten, die nicht die Bedingung des Theorems erfüllen. Im eindimensionalen Fall, wie dem der Abbildung 4.9, wird die Bedingung des Satzes überall außer an den isolierten Punkten erfüllt. An einigen von ihnen können also die Kugeln einer minimalen Überdeckung zentriert werden. In höheren Dimensionen ist die Menge der interessierenden Mittelpunkte der minimalen Überdeckung eine Art „Retraktion".

Eine approximierende algorithmische Lösung kann damit so skizziert werden:

Definition 4.13 (Iterative Konstruktion einer Kugelüberdeckung über iterierte lokale Maxima)
Die iterative Konstruktion einer Kugelüberdeckung über iterierte lokale Maxima *besteht aus folgenden Schritten:*

1. *Füge maximale Kugeln um lokale Maxima von f in die anfangs leere Überdeckung ein.*

2. *Iteriere den ersten Schritt, wobei f auf das Gebiet beschränkt wird, das nicht von Kugeln der bisher konstruierten Überdeckung geschnitten wird.*

Ein zweiter Ansatz besteht darin, einen hierarchischen Gitter-Scan wie folgt durchzuführen:

Definition 4.14 (Konstruktion einer Kugelüberdeckung durch einen iterativen Gitter-Scan)
Die Konstruktion einer Kugelüberdeckung durch einen iterativen Gitter-Scan *besteht in der iterativen Unterteilung eines initialen Hüllquaders des Anfrageraums nach Octree-Art, wobei der Baum in Präorder-Strategie durchlaufen wird. Die Unterteilung wird gestoppt, d.h., ein Blatt wird erreicht, wenn die Anzahl der Szenenelemente, deren Rand von der entsprechenden Zelle geschnitten wird, unter eine vorgegebene Schranke fällt. Für den Mittelpunkt der Blattzelle als Abtastanfrage wird der größte Radius einer umgebenen Kugel bestimmt, die einen ε-Fehler garantiert. Dieses geschieht anhand der Berechnung der Entfernung des Abtastpunktes zu allen Szenenelementen der betrachteten Zelle und möglicherweise von umgebenen Zellen, die* on demand *berechnet werden, solange das ε-Fehlerkriterium noch gilt. Dann wird der größte Vorgänger der Zelle (unter Beachtung der Zelle selbst) bestimmt, der komplett von der resultierenden Kugel umschlossen ist, falls er existiert. In diesem Fall werden alle Blätter nach diesem Knoten ausgelassen, und der Algorithmus wird mit einem anderen Blatt, das im Baum benachbart ist, fortgesetzt. Falls kein Vorgänger existiert, wird das Blatt weiter unterteilt, und die Mittelpunkte der neuen Zellen werden auf eine umschließende Kugel getestet. Falls notwendig, wird der Prozeß iteriert.*

Der Octree spielt in diesem Algorithmus zwei wichtige Rollen. Die erste ist, daß er den Rahmen für die Lage der Abtastanfragen vorgibt. Die zweite ist, den Suchraum für die Berechnung des maximalen Radius der Kugeln zu begrenzen.

Während des Prozesses kann es passieren, daß im Baum Vorgänger gefunden werden, die Vorgänger von vorher bestimmten Zellen sind. In diesem Fall werden die letzteren überflüssig und werden entfernt.

Der skizzierte Algorithmus beinhaltet zwei neue Operationen, die *Entfernungsberechnung* und den *Schnittest einer rechteckigen Box mit dem Rand eines Szenenelements.*

Die Distanzberechnung ist ein Optimierungsproblem unter Nebenbedingungen:

$$\min\{d(\mathbf{q}^*, \mathbf{q}) \mid \mathbf{f}(\mathbf{p}_i, \mathbf{q}) \geq 0\}.$$

Es kann mit bekannten Methoden gelöst werden [Fou81, McC83].

Der Schnittest einer rechteckigen Box mit dem Rand eines Szenenelements wird durch den Schnitt des Elements mit jeder der sechs achsenparallelen Ebenen ausgeführt, die von den Seiten der Box induziert werden. Für jedes dieser sechs eingeschränkten Probleme wird auf Schnitt des Szenenelements und der Seitenfläche der Box getestet. Dieses ist äquivalent zur Existenz einer Lösung eines Systems von Ungleichungen, das sich aus den Ungleichungen des Anfrageproblems durch Festhalten einer Koordinate von $\mathbf{q}$ auf die entsprechende Koordinate der schneidenden Seitenebene und Addieren der Ungleichungen der vier Halbräume, deren Schnitt die betrachtete Seite der Box beschreibt, ergibt. Die Halbräume sind durch die Ebenen der Seiten senkrecht zu der schneidenen Ebene definiert.

Der Abschluß dieses Kapitels setzt sich mit der Frage auseinander, in welcher Größenordnung die Anzahl von Abtastanfragen liegen muß, um eine vorgegebene Approximationsschranke zu erreichen. Eine allgemeine Antwort auf diese Frage kann nicht gegeben werden. Um ein Gefühl für diese Abhängigkeit zu bekommen, wird ein spezielles Beispiel mit ungünstigem Verhalten konstruiert.

Dazu wird ein q-dimensionaler Einheitshyperwürfel betrachtet, der durch n/q äquidistante Ebenen in jede Koordinatenrichtung in ein reguläres Gitter von $(n/q+1)^q$ Unterhyperwürfel unterteilt wird. Jede dieser Ebenen definiert eines von n Elementen einer Szene im Anfrageraum, durch $x_i \leq k/(n/q+1)$ oder $x_i \geq k/(n/q+1)$, $i = 1, \ldots, q$, $k = 1, \ldots n/q$, fest.

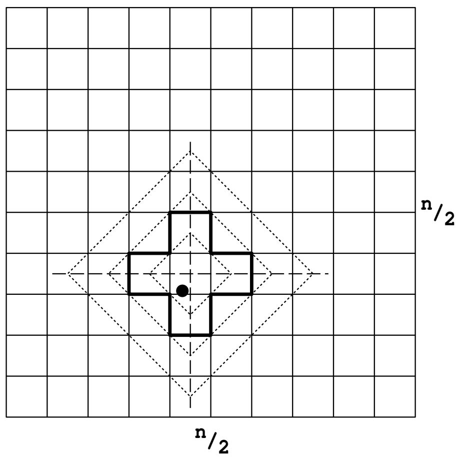


Abbildung 4.10: Abschätzung der Größe der Gebiete mit 0, 1 und 2 Fehlern um einen Abtastpunkt. Der Rand des Gebietes eines Fehlers ist fett gezeichnet. Das Gebiet wird durch die Vereinigung von vier Dreiecken überdeckt.

Lemma 4.1 *Sei $\varepsilon \in I\!N$ eine Schranke für den absoluten Fehler und $\mathbf{q}_0$ eine Abtastanfrage,* *die in einem der Gitter-Hyperwürfel liegt. Jedes Gebiet, das* $\mathbf{q}_0$ *umgibt und das einen* ε*-Fehler garantiert, enthält höchstens* $2^q(\varepsilon+1)^q/q!$ *Gitterpunkte.*

Beweis: Vgl. Abbildung 4.10. Die Antwort eines Anfragepunktes $\mathbf{q}$ differiert um höchstens ε Szenenelemente genau dann, wenn höchstens ε Hyperebenen zwischen $\mathbf{q}$ und $\mathbf{q}_0$ liegen. Daher muß die Anzahl i_k, $k = 1, \ldots, q$, an Hyperebenen zwischen den zwei Punkten in jede Koordinatenrichtung $\sum_{k=1}^{q} i_k \leq \varepsilon$ genügen. Zusammen mit den Bedingungen $i_k \geq 0$, $k = 1, \ldots, q$, berandet diese Ebene $\sum_{k=1}^{q} x_k = \varepsilon$ ein q-dimensionales Simplex. Das Volumen dieses Simplex ist $(\varepsilon+1)^q/q!$. Da die Differenz zwischen den k-ten Koordinaten von $\mathbf{q}$ und $\mathbf{q}_0$ positiv oder negativ sein kann, ist das Gebiet um $\mathbf{q}_0$, in dem $\mathbf{q}$ liegen kann, aus 2^q Simplizes dieses Volumens zusammengesetzt. Daher ist das Gesamtvolumen dieses Gebiets $2^q(\varepsilon+1)^q/q!$. Da dieses Volumen auch eine obere Schranke für die Anzahl der Gitterzellen angibt, die von der Fehlerregion überdeckt werden, ergibt sich die obere Schranke, die im Satz genannt wurde. □

Theorem 4.5 *Sei* $\varepsilon \in I\!N$. *Die Anzahl der Abtastanfragen* $s(n, \varepsilon)$, *die benötigt wird, um eine absolute* ε*-Schranke zu garantieren, ist im schlechtesten Fall mindestens* $\Omega((n/\varepsilon)^q)$, *wobei* n *die Anzahl der Szenenelemente und* q *die Dimension des Anfrageraums ist.*

Beweis: Da insgesamt $(n/q)^q$ innere Gitterpunkte vorhanden sind, ist die Anzahl s an benötigten Abtastanfragen

$$s \approx (n/q)^q / 2^q(\varepsilon+1)^q/q! \geq (q!n^q)/((2q)^q(\varepsilon+1)^q) = \Omega((n/\varepsilon)^q).$$

□

Beim Streckenanfrageproblem kann diese unerwünschte Situation in der Tat auftreten.

Theorem 4.6 *Beim Streckenanfrageproblem für eine Szene von* n *sich nicht schneidenden Strecken in der Ebene können im schlechtesten Fall* $\Theta(n^4)$ *Antwortmengen auftreten.*

Beweis: Siehe Beispiel in Abbildung 4.11. Die Szene besteht aus drei horizontal und vertikal arrangierten Unterszenen. Die Unterszene in der Mitte besteht aus $n/3$ vielen horizontalen Strecken. Die linke und rechte Szene sind identisch, jede von ihnen setzt sich aus $n/3$ vielen vertikalen Strecken zusammen.

Die Anfragestrecken gehen durch einen linken und einen rechten Endpunkt der horizontalen Strecken. Die Anfragestrecken beginnen zwischen zwei aufeinanderfolgenden Strecken der linken Szene und enden zwischen zwei aufeinanderfolgenden Strecken der rechten Szene.

Es gibt eine Menge von $\Theta(n^2)$ vielen Geraden, die durch die Endpunkte der horizontalen Strecken laufen, wobei jede eine andere Menge von horizontalen Strecken

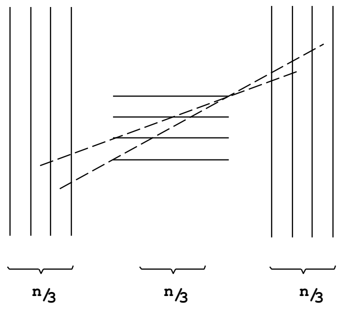


Abbildung 4.11: Beispiel einer Szene mit $\Theta(n^4)$ Antwortmengen für das Streckenschnitt-Anfrageproblem

schneidet. Auf jeder dieser Geraden gibt es eine Menge von $\Theta(n^2)$ vielen Strecken, wobei jede eine andere Menge von vertikalen Strecken schneidet (es gibt $\Theta(n)$ Orte von Endpunkten zwischen Strecken der linken Unterszene. Jeder von ihnen wird kombiniert mit den $\Theta(n)$ vielen unterschiedlichen Orten der Endpunkte in der rechten Unterszene). Also existieren $\Theta(n^4)$ viel Strecken mit unterschiedlichen Antwortmengen.

Bei beliebigen Szenen von sich nicht schneidenden Strecken kann es nicht mehr als $O(n^4)$ Anfragestrecken mit unterschiedlichen Antwortmengen geben. Um dieses zu verdeutlichen, wird eine Gerade durch jeden Endpunkt eines Elements der Szene betrachtet, wobei alle Geraden dieselbe Richtung haben. Diese Geraden unterteilen die Szene in Streifen. Jeder Streifen wird von Strecken der Szene traversiert und jede Menge von traversierenden Strecken kann unterschiedlich sein. Wenn die Ausrichtung der Geraden nur wenig verändert wird, dann werden die Streifen normalerweise von denselben Szenenstrecken traversiert. Das ändert sich bei solchen Ausrichtungen, bei denen die Geraden zwei Endpunkte der Strecken verbinden. In diesem Fall ändert ein Streifen die Menge der traversierenden Strecken. Da es $O(n^2)$ viele dieser Ausrichtungen gibt, existieren höchstens $O(n^2)$ viele Streifen mit unterschiedlichen Mengen.

In jedem Streifen gibt es höchstens $O(n^2)$ Anfragestrecken mit unterschiedlichen Antwortmengen (es gibt höchstens $O(n^2)$ viele Gebiete in dem Streifen für jeden der beiden Endpunkte). Daher ist die Gesamtanzahl der Anfragestrecken mit unterschiedlicher Antwortmenge in $O(n^4)$. $\square$

Korollar 4.2 *Beim Geradenanfrageproblem für eine Szene von n sich nicht schneidenden Strecken in der Ebene können im schlechtesten Fall $\Theta(n^2)$ Antwortmengen auftreten.*

Beweis: Über eine ähnliche Argumentation wie oben, mit Abbildung 4.11 ohne die beiden vertikalen Teilszenen links und rechts. $\square$

Die letzte Abschätzung ist recht großzügig. Oftmals ist die Anzahl der geschnittenen Szenenstrecken erheblich kleiner. Als Beispiel seien Szenenstrecken genannt, die auf einem konvexen Polygon plaziert werden. In diesem Fall wird ein Streifen höchstens von zwei Szenenstrecken traversiert. Daher können hier nur $O(n^2)$ Anfragestrecken mit unterschiedlichen Antwortmengen beobachtet werden.

Das schlechte Verhalten im schlechtesten Fall kann durch spezielle Situationen in der Praxis relativiert werden. Es kann passieren, daß Anfragen aus Applikationen nicht über den ganzen Anfrageraum verteilt sind. In diesem Fall ist es sinnvoll, die Abtastanfragen aus den Anfragen der späteren Applikation zu wählen. Die Vorverarbeitung ist dann auf die Gebiete beschränkt, in die diese Abtastanfragen fallen.

4.3.2 Flächenabtastung

Bei einer Flächenabtastung lassen sich auf folgende Weise approximative Antwortmengen erzeugen:

Definition 4.15 (Approximative Antworttypen für Punktanfragen bei Gebietsabtastung)
Gegeben sei eine Gebietsabtastung $\mathcal{G}$ eines Anfrageproblems. Die approximative Antwort *auf eine Punktanfrage bezüglich $\mathcal{G}$ ist die Vereinigung der Antwortmengen aller Abtastgebietsanfragen in $\mathcal{G}$, in die die Punktanfrage fällt.*

Die korrigierte approximative Antwort *besteht aus allen Elementen der approximativen Antwort, die die zu beantwortende Punktanfrage enthalten.*

Falls die Punktanfrage in kein Gebiet fällt, ist die Antwortmenge leer. Üblicherweise wird die Gebietsabfrage eine Partition des Anfrageraums in disjunkte Abtastgebiete darstellen. In diesem Fall fällt jede Punktanfrage in genau ein Gebiet.

Die Qualität einer approximativen Antwort läßt sich wie mit Definition 4.9 charakterisieren.

Für die korrigierte Antwort gilt folgende unmittelbare Beobachtung:

Theorem 4.7 (Korrektheit der korrigierten Antwort bei Gebietsabtastung)
Die korrigierte approximative Antwort, die über eine Gebietsabtastung erhalten wurde, ist die korrekte Antwort auf die Anfrage. Der Zeitaufwand zur Korrektur einer approximativen Antwort mit absolutem bzw. relativem Fehler kleiner als $\varepsilon > 0$ ist $O((I + \varepsilon) \cdot t))$ bzw. $O(\varepsilon \cdot I \cdot t)$, wobei I die Größe der korrekten Antwort und t die Zeit zum Testen eines Szenenelements auf Zugehörigkeit zur Antwortmenge einer Punktanfrage ist.

Beweis: Die approximative Antwort enthält insbesondere alle Szenenelemente, in die die Punktanfrage fällt. Die Korrektur selektiert genau diese. Sie bilden die korrekte Antwort auf die Punktanfrage.

Zur Korrektur sind so viele Szenenelemente zu testen wie in der approximativen Antwort liegen. Ausgehend von der Größe I der korrekten Antwort sind dies maximal $I + \varepsilon$ bzw. $\varepsilon \cdot I$ viele, im absoluten bzw. relativen Fehlermaß. $\square$

Bei vorgegebener Fehlerschranke ist man aus Effizienzgründen an Gebietsabtastungen interessiert, die möglichst wenige Abtastgebiete enthalten:

Definition 4.16 (ε-Optimalität)
Sei $\varepsilon > 0$. Eine Gebietsabtastung eines Anfrageproblems wird absolut *bzw.* relativ ε-optimal *genannt, wenn sie von minimaler Größe aller in Frage kommenden endlichen Überdeckungen $\mathcal{G}$ des Anfrageraums ist, deren Gebiete einen ε-Fehler garantieren, d.h.*

$$\max\{e(R(\mathbf{q}), \bigcup_{\mathbf{x} \in G} R(G)) \mid \mathbf{q} \in \bigcup_{\mathbf{x} \in G} R(G),\ \mathbf{x} \in \bigcup_{G \in \mathcal{G}} G\} \leq \varepsilon,$$

wobei

$R(\mathbf{q})$ *die Antwortmenge einer Anfrage* $\mathbf{q}$

$R(G)$ *die Menge der Szenenelemente, die G schneiden*

$e(.,.)$ *der absolute bzw. relative Fehler ist.*

Das Optimierungsproblem kann mit denselben Ansätzen wie im Fall der Punktabtastung behandelt werden. Üblicherweise wird man sich bei der Wahl der Abtastanfragen auf eine spezielle Klasse von Gebieten beschränken. Ein typisches Beispiel ist die Abtastung durch hierarchische Zerlegung.

Definition 4.17 (Gebietsabtastung durch hierarchische Zerlegung)
Eine Gebietsabtastung durch hierarchische Zerlegung *ergibt sich durch iteriertes Anwenden einer Zerlegungsregel auf die resultierenden Gebiete, die angibt, wie ein gegebenes Gebiet in endlich viele Teilgebiete zu zerlegen ist. Die Anwendung der Regel endet bei Gebieten, die ein vorgegebenes Approximationskriterium erfüllen. Diese Gebiete definieren die Gebietsabtastung.*

Typische Beispiele für hierarchische Zerlegungen sind die Zerlegungen nach dem (Hyper)-Octree-Schema und nach dem Bintree-Schema. Beim *Hyper-Octree-Schema* wird der d-dimensionale Anfrageraum rekursiv in 2^d kongruente Hyperquader unterteilt. Beim *Bintree-Schema* erfolgt die Unterteilung mittels einer achsenparallelen Hyperebene in zwei Quader, wobei die Richtung der Ebene alternierend geändert wird [BF79].

Das Approximationskriterium, das sich aus dem ε-Optimierungsproblem ergibt, ist, daß für jeden Punkt eines Abtastgebiets der absolute bzw. relative Fehler kleiner als ε ist.

Dieses kann durch Bestimmung der Zerlegung des Abtastgebiets geschehen, die durch die schneidenden Szenenelemente induziert wird. Um in der Umgebung einer Anfrage, die gleichzeitig auf dem Rand mehrerer Szenenelemente liegt, zu verhindern, daß die hierarchische Zerlegung nicht terminiert, ist eine Schranke für die Größe des Abtastgebiets vorzugeben, unter der das Gebiet nicht weiter zerlegt wird. In einem solchen Gebiet kann es passieren, daß die vorgegebene Fehlerschranke nicht unterboten wird. Dies ändert an der Korrigierbarkeit jedoch nichts, sondern nur an dem Aufwand dafür, der dann größer sein kann.

Die Berechnung der Zerlegung des Abtastgebiets kann aufwendig sein. Das folgende Approximationskriterium kann einfacher auswertbar sein:

Definition 4.18 (Überdeckungsfehler)
Sei G ein Gebiet des Anfrageraums, $R(G)$ die Menge der G schneidenden Szenenelemente, $C(G)$ die Menge der Szenenelemente, die G als Teilmenge haben. Dann heißt

$$e_a^c(C(G), R(G)) := |R(G) \setminus C(G)|$$

absoluter Überdeckungsfehler *und*

$$e_r^c(C(G), R(G)) := |R(G) \setminus C(G)|/|C(G)|$$

relativer Überdeckungsfehler.

Abbildung 4.12 illustriert diese Terminologie. In diesem Beispiel ist $e_a^c = 3$ und $e_r^c = 3$.

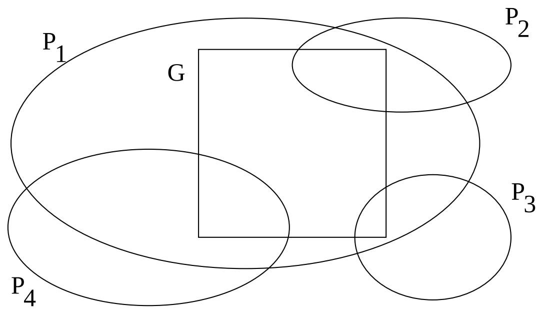


Abbildung 4.12: Gebiet G im Anfrageraum mit vier Szenenelementen $P_1, \ldots, P_4$

Nun gilt:

Theorem 4.8 (Überdeckungsfehler und Abtastfehler) *Ein Überdeckungsfehler kleiner als ε garantiert auch einen Approximationsfehler kleiner als ε für alle Punkte in einem Abtastgebiet einer Gebietsabtastung mit disjunkten Abtastgebieten.*

Beweis: Im schlimmsten Fall umfaßt die Menge aller Szenenelemente, die das Abtastgebiet, in das eine Punktanfrage fällt, schneiden, aber nicht zur Antwortmenge der Punktanfrage gehören, alle Szenenelemente, die das Abtastgebiet schneiden, aber nicht überdecken. Der Überdeckungsfehler ist also mindestens so groß wie der Approximationsfehler. $\square$

In Abbildung 4.12 ist $e_a^c \leq 3$ und $e_r^c \leq 3$ für alle Anfragen in Z.

Die beiden folgenden Definitionen geben weitere Beispiele für Gebietsabtastungen, die sich aus den Überlegungen zur Punktabtastung ergeben.

Definition 4.19 (Voronoiabtastung)
Gegeben sei eine endliche Menge X von Punktanfragen. Die von X induzierte Voronoiabtastung *besteht aus den Voronoigebieten zu den Punkten in X.*

Die Voronoiabtastung ist im Unterschied zu der Punktabtastung, die durch X induziert wird, stets korrigierbar.

Definition 4.20 (Kugelabtastung)
Eine Kugelabtastung *besteht aus einer endlichen Überdeckung des Anfrageraums mit Kugeln. Dabei wird der Anfrageraum als kompakt vorausgesetzt.*

Im Unterabschnitt 4.3.1 wurde die Überdeckungsfunktion $r_\varepsilon(\mathbf{q})$ für Kugeln definiert:

$$r_\varepsilon(\mathbf{q}) := \sup\{r \mid U_r(\mathbf{q}) \text{ garantiert einen } \varepsilon - \text{Fehler}\} \text{ für } \mathbf{q} \in Q,$$

wobei Q der betrachtete Anfrageraum ist und $U_r(\mathbf{q})$ die offene Kugel mit Radius r und Mittelpunkt $\mathbf{q}$. Abbildung 4.13 zeigt die korrespondierende Funktion r_ε, $\varepsilon = 1$, für den Fall der Gebietsabtastung über dem Anfrageraum $M = [0, 21]$. Im Gegensatz zur Punktabtastung ist diese Funktion stetig.

Zur Untersuchung des quantitativen Verhaltens der Gebietsabtastung wurden eine Reihe von Experimenten ausgeführt, die im folgenden zusammengestellt sind.

Experiment 4.5 (Anzahl notwendiger Abtastanfragen bei vorgegebener Fehlerschranke)

Eingabe: *Szenen aus 50 Strecken im Einheitsquadrat mit unterschiedlicher Verteilung: zufällige Verteilung im ganzen Einheitsquadrat (*Zuf*) sowie in vier (*4Q*), drei (*3Q*), zwei (*2Q*) und einem (*1Q*) Quadranten des Einheitsquadrats.*

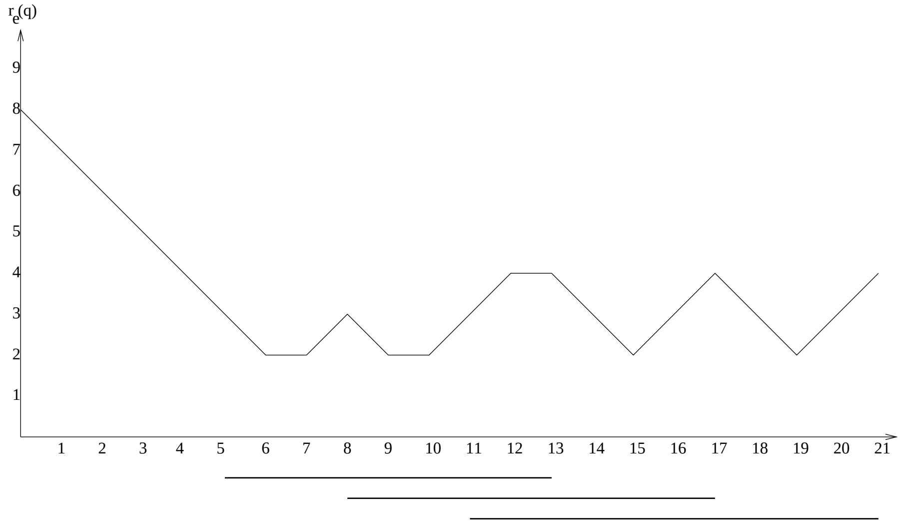


Abbildung 4.13: Ein eindimensionaler Anfrageraum und die korrespondierende Funktion r_ε, $\varepsilon = 1$, für das absolute Fehlermaß bei der Flächenabtastung

Parameter: *Der absolute Überdeckungsfehler* e_a^c *wird durch* 3, *der relative Überdeckungsfehler* e_r^c *durch* 0.5 *beschränkt, d.h., die Unterteilung wird abgebrochen, wenn* $e_a^c \leq 3$ *ist oder wenn* $e_r^c \leq 0.5$ *ist. Die Unterteilung erfolgt nach dem Quadtree-Schema, die Unterteilungstiefe wird durch* 15 *beschränkt.*

Plattform: *Linux-PC, 486DX2.*

Gemessene Größe(n): *Vgl. Abbildung 4.14: Anzahl der notwendigen Abtastanfragen (y-Achse), in Abhängigkeit von der Verteilung der Strecken in der Szene (x-Achse).*

Beobachtung: *Es ist deutlich zu sehen, daß die Anzahl der Abtastpunkte um so höher ist, je stärker sich die Strecken in der Szene ballen. Ein Grund dafür ist die Tatsache, daß die Wahrscheinlichkeit dafür, daß eine Anfragegerade die Endpunkte mehrerer Szenenstrecken schneidet, steigt, je dichter diese Strecken beieinander liegen (die Strecken haben alle eine maximale Länge von* 1/10 *der Breite des umschließenden Quadrats). In diesem ungünstigen Fall wird die Unterteilung eventuell bis zur maximalen Rekursionstiefe d fortgeführt.*

Den direkten Vergleich zwischen den beiden Unterteilungsverfahren zeigt Abbildung 4.15.

Experiment 4.6 (Speicherbedarf der Unterteilungsverfahren)

Eingabe: *Wie zuvor.*

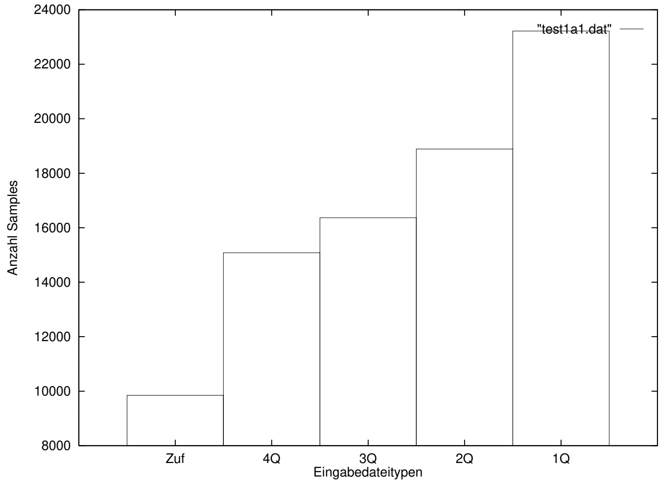


Abbildung 4.14: Vergleich der Anzahl der Abtastanfragen in Abhängigkeit von der Verteilung der Elemente der Szene, von links nach rechts: zufällige Verteilung im ganzen Einheitsquadrat (*Zuf*) sowie in vier (*4Q*), drei (*3Q*), zwei (*2Q*) und einem (*1Q*) Quadranten des Einheitsquadrats.

Parameter: *Der absolute Fehler* e_a^c *wird durch* 3, *der relative Fehler* e_r^c *durch* 0.5 *beschränkt, d.h., die Unterteilung wird abgebrochen, wenn* $e_a^c \leq 3$ *ist oder wenn* $e_r^c \leq 0.5$ *ist. Die Unterteilung erfolgt nach dem Quadtree-Schema und dem Bintree-Schema, die Unterteilungstiefe wird durch* 15 *beschränkt.*

Plattform: *Linux-PC, 486DX2.*

Gemessene Größe(n): *Vgl. Abbildung 4.15: benötigter Speicher (y-Achse) in Abhängigkeit von der Verteilung der Strecken in der Szene (x-Achse), für das Quadtree-Schema (jeweils linker Balken) und das Bintree-Schema (jeweils rechter Balken).*

Beobachtung: *Bei allen Szenentypen zeigt das Bintree-Schema ein deutlich speichereffizienteres Verhalten. Das bedeutet, daß beim Bintree-Schema zur Garantie eines bestimmten Überdeckungsfehlers weniger Abtastanfragen erzeugt und gespeichert werden müssen als beim Quadtree-Schema. Die Ersparnis liegt bei* 27–29%.

Interessant ist, ab welcher Tiefe der Unterteilung der e_a^c-Fehler kleiner oder gleich einem vorgegebenen Wert wird. Im folgenden wird $e_a^c \leq \varepsilon$, mit $\varepsilon > 0$, das e_a^c-*Kriterium* genannt. Daher wurde der prozentuale Anteil der Abtastpunkte, die dieses Kriterium nicht erfüllen, in Abhängigkeit von der maximalen Tiefe d ermittelt.

Experiment 4.7 (Anteil der Anfragen, die das e_a^c-Kriterium nicht erfüllen)

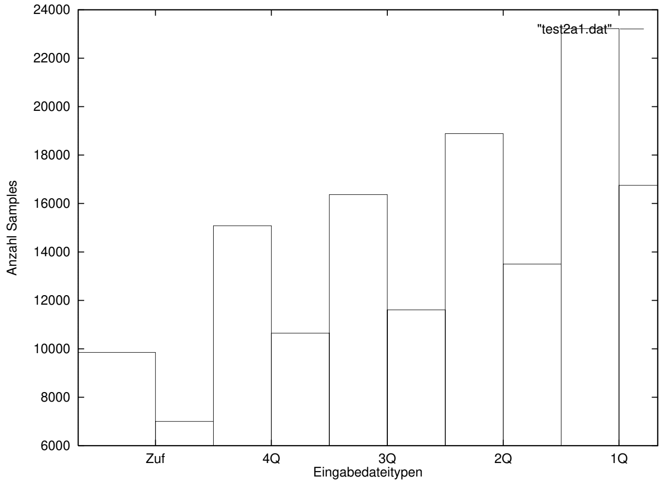


Abbildung 4.15: Direkter Vergleich der Unterteilungsverfahren

Eingabe: 200, 300 *und* 400 *zufällig generierte Strecken.*

Parameter: *Beschränkung des absoluten Fehlers* e_a^c *auf* 2 *bei* 200 *Strecken, auf* 3 *bei* 300 *Strecken und auf* 4 *bei* 400 *Strecken.*

Plattform: *Linux-PC, 486DX2.*

Gemessene Größe(n): *Vgl. Abbildung 4.16: Anteil der Anfragen (in Prozent), die das* e_a^c*-Kriterium nicht erfüllen (y-Achse), in Abhängigkeit von der maximalen Baumtiefe (x-Achse).*

Beobachtung: *Bei* $d = 5$ *erfüllen ca.* 95% *aller Zellen nicht das* e_a^c*-Kriterium. Zwischen* $d = 6$ *und* $d = 9$ *fällt die Kurve rapide ab, so daß bei* $d = 10$ *die*

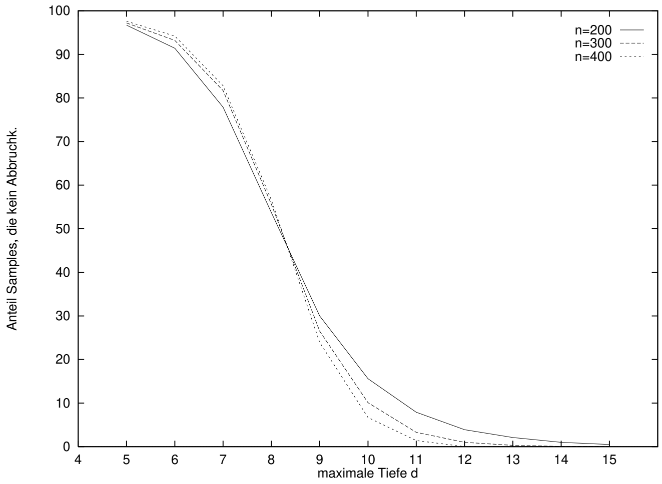


Abbildung 4.16: Anteil der Abtastpunkte, die nicht das e_a^c-Kriterium erfüllen

Anzahl der Zellen, die das Kriterium nicht erfüllen, auf ca. 10% *gefallen* *ist. Ab $d = 10$ verlaufen die Kurven flacher, d.h., es gibt nur noch wenige Zellen der Unterteilung, die das Kriterium nicht erfüllen und daher bis zur maximalen Tiefe unterteilt werden.*

4.4 Speicherung von Abtastungen

Da von der Anfragedatenstruktur, die in der Abtastphase aufgebaut wird, für jede Abtastanfrage die entsprechende Antwortmenge bereitgestellt werden muß, ist zu erwarten, daß der Speicherbedarf nicht unerheblich ist. Dieser Aspekt wird im folgenden behandelt.

Der Typ der Anfragedatenstruktur, der hier betrachtet wird, ist wie folgt aufgebaut:

Definition 4.21 (Anfragedatenstruktur)
Eine Anfragedatenstruktur *besteht aus zwei Teilen, der* Antwortdatenbank *und dem* Anfrageverzeichnis.

Das Anfrageverzeichnis *liefert für eine Anfrage diejenigen Abtastanfragen, aus denen die Antwortmenge einer Anfrage bestimmt wird. Die* Antwortdatenbank *liefert für eine Abtastanfrage die zu ihr gehörende Abtastantwort.*

4.4.1 Anfrageverzeichnis der Anfragedatenstruktur

Bei Punktabtastungen handelt es sich beim Anfrageverzeichnis um eine Datenstruktur, in der endlich viele Punkte abgelegt sind. Je nach Typ der Antwortkonstruktion hat sie für einen beliebigen Anfragepunkt den nächsten oder die k nächsten Nachbarn unter den abgespeicherten Punkten zu liefern. Für diese Aufgaben sind effiziente Lösungen in der algorithmischen Geometrie bekannt, auf die hierfür zurückgegriffen werden kann und auf die hier verwiesen wird [AMN95, GO97].

Bei der Flächenabtastung speichert das Anfrageverzeichnis eine Menge von Gebieten, so daß für einen Anfragepunkt effizient die Gebiete gefunden werden, in die er fällt. Je nachdem, ob die Gebiete überlappend sind oder eine Partition bilden, handelt es sich hier um ein Aufspießproblem (engl. *stabbing problem*) oder ein Punktlokalisierungsproblem. Auch für dieses Problem sind für diverse Typen von Regionen effiziente Lösungen bekannt, so daß darauf nicht weiter eingegangen wird [DL76, CS90, GO97].

4.4.2 Antwortdatenbank der Anfragedatenstruktur

Die einfachste Form der Antwortdatenbank ist, jede Antwortmenge explizit zu repräsentieren. Hierfür bieten sich zunächst die folgenden beiden elementaren Möglichkeiten an:

s1	1	1	0
s2	1	1	1
s3	0	0	0
s4	1	0	0
s5	0	1	0
	q_1	q_2	q_3

Abbildung 4.17: Beispiel einer Antwortdatenbank in der Darstellung als binäres Feld

Definition 4.22 (Listendarstellung einer Antwortdatenbank)
Jede Antwortmenge wird durch eine linear verzeigerte Liste der in ihr enthaltenen Elemente dargestellt. Ein Eintrag des Anfrageverzeichnisses verweist dann auf den Kopf derjenigen Liste, die seine Antwortmenge repräsentiert.

Definition 4.23 (Binärfelddarstellung einer Antwortdatenbank)
Der Zeilenindex eines binären Feldes repräsentiert die Szenenelemente, der Spaltenindex die Abtastantwort, die in der Datenbank gespeichert sind. Ein Feldelement mit Index $[i, j]$ ist genau dann gleich 1, wenn Szenenelement i in der Abtastantwort j enthalten ist. Ein Eintrag des Anfrageverzeichnisses enthält den Index derjenigen Abtastantwort, die seine Antwortmenge repräsentiert.

Falls mehrere Abtastantworten dieselbe Antwortmenge haben, genügt es, die Liste beziehungsweise die Spalte nur einmal abzuspeichern.

Abbildung 4.17 zeigt ein Beispiel mit drei Abtastanfragen an fünf Strecken. Spalte 2 speichert die Antwort auf die zweite Geradenanfrage: die Strecken 1, 2 und 5 werden von der Anfragegeraden geschnitten, die Strecken 3 und 4 werden nicht geschnitten.

Bei dichteren Abtastungen ist es sehr wahrscheinlich, daß sich viele Antwortmengen nur wenig voneinander unterscheiden. Im folgenden werden Überlegungen dazu angestellt, die Redundanz zu mindern.

Ein erster Vorschlag ist die wie folgt definierte Intervallrepräsentation der Antwortdatenbank:

Definition 4.24 (Intervalldarstellung einer Antwortdatenbank)
Gegeben sei eine lineare Anordnung der Abtastantworten. Die Intervalldarstellung einer Antwortdatenbank *besteht aus einer Menge von Intervallen von Abtastantworten, die folgende Eigenschaft hat:*

1. *Jedem Intervall ist ein Szenenelement zugeordnet.*

2. *Die Vereinigung zweier Intervalle mit dem gleichen zugeordneten Szenenelement bildet kein Intervall.*

3. *Die Antwortmenge einer Abtastanfrage ergibt sich als die Menge aller Szenenelemente, die den Intervallen zugeordnet sind, in die die Abtastanfrage fällt.*

Die Intervalle sind in einer Datenstruktur organisiert, die das effiziente Auffinden der Intervalle ermöglicht, in die ein solcher Index fällt. Ein Eintrag des Anfrageverzeichnisses enthält den Index derjenigen Abtastantwort, die seine Antwortmenge repräsentiert.

Die Intervalldarstellung kann aus der Darstellung durch ein binäres Feld dadurch erhalten werden, daß lückenlose Folgen von Einsen auf einer Zeile zu einem Intervall zusammengefaßt werden. Wenn ein Intervall durch seinen Anfangs- und Endindex kürzer als durch die entsprechenden Folgen von Einsen repräsentiert werden kann, ergibt sich ein Kompressionseffekt.

Eine effiziente Implementierung der Intervalldarstellung einer Antwortdatenbank erfordert, Intervalle so in eine Datenstruktur vorzuverarbeiten, daß für einen beliebigen Anfragepunkt diejenigen Intervalle effizient gefunden werden, in die der Anfragepunkt fällt. Für dieses Problem ist eine Reihe von Löungen bekannt: *Intervallbäume* [Ede80, McC80], *Segmentbäume* [Ben77], *Intervall-Skip-Listen* [Han91], *Prioritätssuchbäume* [McC82] und *Filtering search* [Cha82]. Dabei kann die Information zur geometrischen Lokalisierung in den Datenstrukturen weggelassen werden. Anstelle dessen wird vom Anfrageverzeichnis auf diejenigen Blätter verwiesen, die eine geometrische Region repräsentieren, in die die entsprechende Abtastanfrage fällt. Die Datenstruktur wird dann in umgekehrter Weise, also vom Blatt zur Wurzel und nicht, wie üblich, umgekehrt durchlaufen, um auf diesem Pfad die Information über die Intervallzugehörigkeit aufzusammeln, die an inneren Knoten gespeichert ist.

Die lineare Anordnung der Antwortmengen ist natürlich, aber dennoch willkürlich. Es ist im Prinzip auch eine mehrdimensionale Anordnung möglich. So bietet es sich an, die Anordnung der Abtastanfragen im Anfrageraum zu verwenden. Dies führt zu Darstellungen der Antwortdatenbank durch approximative Elementgebiete:

Definition 4.25 (Elementgebietdarstellungen einer Antwortdatenbank)
Gegeben sei eine Antwortdatenbank aus einer Punktabtastung. Eine Elementgebietdarstellung *der Antwortdatenbank besteht aus einer Menge von Gebieten im Anfrageraum, von denen jedes einem Szenenelement entspricht, das in einer Antwort auftritt. Es werden folgende Arten der Gebietsdarstellung unterschieden:*

Voronoidarstellung: *Ein Gebiet eines Szenenelements besteht aus der Vereinigung derjenigen Voronoizellen des Voronoidiagramms der Menge der Abtastanfragen, deren zugehörige Abtastanfrage eine Antwort hat, die das Szenenelement enthält.*

Delaunay-Darstellung: *Das Gebiet eines Szenenelements ergibt sich als Vereinigung derjenigen Delaunay-Simplizes, deren Ecken Abtastanfragen sind, deren Antwortmenge das Szenenelement enthält.*

Hyperoctree-Darstellung: *Der Anfrageraum wird so durch einen Hyperoctree zerlegt, daß alle Blattzellen mindestens eine Abtastanfrage enthalten und deren weitere Unterteilung zu einer leeren Blattzelle führen würde. Das Gebiet eines Szenenelements ergibt sich als Vereinigung derjenigen Blattzellen, in denen eine Abtastanfrage liegt, zu deren Antwortmenge das Szenenelement gehört.*

Die Zellen sind in einen Anfragebaum organisiert, der aufgebaut wird, indem sukzessive Gebiete zusammengefaßt werden. Im Fall der Hyperoctreezerlegung kann der Hyperoctree natürlicherweise dafür verwendet werden. Jeder Knoten des Baums wird mit den Szenenelementen markiert, zu denen eines seiner Blätter beiträgt. Diese Markierung kann dadurch komprimiert werden, daß sie durch eine Kantenmarkierung ersetzt wird. Die Kantenmarkierung enthält die Elemente, die beim Übergang von einem Knoten zu seinem Nachfolger nicht mehr auftreten.

Ein Eintrag des Anfrageverzeichnisses enthält einen Verweis auf diejenigen Blätter des Anfragebaums, in deren Zellen die Abtastanfrage liegt.

Ein dritter Vorschlag nützt die Tatsache, daß Antworten, die auf der Grundlage der Abtastdatenstruktur gewonnen werden, in der Regel approximativ sind und daß es zweckmäßig ist, auf den approximativen Antworten eine Korrektur durchzuführen. Falls die Korrektur nicht durchgeführt wird, liefert das Verfahren eine etwas schlechtere Approximation der Antworten.

Definition 4.26 (δ-Containerdarstellung einer Antwortdatenbank)
Gegeben sei eine Konstante $\delta > 0$. Ein absoluter *beziehungsweise* relativer Container C *ist eine Menge von Abtastantworten R, zu denen es eine Menge $R(C)$ von Szenenelementen gibt, so daß $R(C) \supseteq R$ für alle $R \in C$ und der absolute beziehungsweise relative Fehler von $R(C)$ bezüglich aller R kleiner als δ ist. $R(C)$ heißt* Repräsentant *des Containers.*

Eine δ-Containerdarstellung *einer Antwortdatenbank besteht aus Repräsentanten von Containern, die eine Überdeckung der in der Antwortdatenbank vorhandenen Abtastantworten liefert. Diese Repräsentanten können in irgendeiner Darstellung für Antwortdatenbanken gespeichert werden. Diese Datenstruktur wird Repräsentantendatenbank genannt.*

Ein Eintrag des Anfrageverzeichnisses enthält einen Verweis auf den Repräsentanten eines Containers, der seine Antwortmenge überdeckt. Der Repräsentant wird über ihn über das Anfrageverzeichnis der Repräsentantendatenbank gefunden.

Die Beantwortung einer Anfrage geschieht dadurch, daß der gefundene Repräsentant bezüglich der Anfrage korrigiert wird.

s1	1	1	0	**1**
s2	1	1	1	**1**
s3	0	0	0	**0**
s4	1	0	1	**1**
s5	0	1	1	**1**
	q_1	q_2	q_3	$R(C)$

Abbildung 4.18: Beispiel eines Containerrepräsentanten $R(C)$ mit $\delta = 2$

Abbildung 4.18 zeigt ein Beispiel der δ-Containerdarstellung mit $\delta = 2$, $C = \{\{1,1,0,1,0\},\{1,1,0,0,1\},\{0,1,0,1,1\}\}$ und $R(C) = \{1,1,0,1,1\}$.

Für die δ-Containerdarstellung gilt:

Theorem 4.9 (Approximationsverhalten und Effizienz der δ-Containerdarstellung) *Gegeben sei eine Abtastung, die eine absolute beziehungsweise relative ε-approximative korrigierte Anfragebeantwortung erlaubt. Dann erlaubt die δ-Containerdarstellung ebenfalls die ε-approximative korrigierte Anfragebeantwortung. Der zusätzliche Zeitaufwand hierfür ist durch $O(I' + \delta \cdot t)$ bzw. $O((1+\delta) \cdot I' \cdot t)$ beschränkt, wobei I' die Mächtigkeit der unkorrigierten approximativen Antwortmenge ist und t die Zeit, die benötigt wird, um ein Szenenelement darauf hin zu testen, ob es zur korrekten Antwortmenge gehört.*

Falls die Abtastung eine absolute beziehungsweise relative ε-approximative Anfragebeantwortung erlaubt, dann erlaubt die δ-Containerdarstellung eine $(\varepsilon + \delta)$-approximative absolute und eine $(\varepsilon + \delta + \varepsilon \cdot \delta)$-approximative relative Anfragebeantwortung.

Beweis: Da der Containerrepräsentant für eine Antwort eine Obermenge der Antwort ist, läßt sich durch dessen Korrektur auch die Korrektur der Antwort erreichen. Der Zusatzzeitaufwand ist durch die Anzahl der zu testenden Szenenelemente beschränkt, die nach Definition der δ-Containerdarstellung die angegebene Größe hat.

Sei $\tilde{A}$ die approximative Antwort, A die korrekte Antwort und $R(\tilde{A})$ der Containerrepräsentant von $\tilde{A}$. Für die absolute Approximation gilt dann

$$|R(\tilde{A}) \Delta A| \leq |R(\tilde{A}) - \tilde{A}| + |\tilde{A} \Delta A| \leq \delta + \varepsilon.$$

Für die relative Approximation gilt (vgl. Abbildung 4.19)

$$\begin{aligned} |R(\tilde{A}) \Delta A| / |A| &= \big(|R(\tilde{A}) - A| + |A - R(\tilde{A})|\big)/|A| \\ &= \big(|R(\tilde{A}) - \tilde{A} - A| + |\tilde{A} - A| + |A - R(\tilde{A})|\big)/|A| \\ &\leq \big(|R(\tilde{A}) - \tilde{A} - A| + |\tilde{A} - A| + |A - \tilde{A}|\big)/|A| \end{aligned}$$

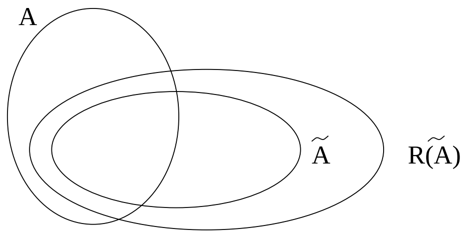


Abbildung 4.19: Korrekte Antwort A, approximierte Antwort $\tilde{A}$ und der Containerrepräsentant $R(\tilde{A})$

$$\begin{aligned}
&\leq \big(|R(\tilde{A}) - \tilde{A}| + |\tilde{A}\Delta A|\big)/|A| \\
&\leq (1+\varepsilon) \cdot |R(\tilde{A}) - \tilde{A}|/|\tilde{A}| + |\tilde{A}\Delta A|/|A| \leq (1+\varepsilon)\cdot\delta + \varepsilon \\
&= \varepsilon + \delta + \varepsilon \cdot \delta,
\end{aligned}$$

mit

$$|\tilde{A}| = |\tilde{A} - A| + |\tilde{A} \cap A| \leq \varepsilon|A| + |A| = (1+\varepsilon)\cdot|A|. \quad \square$$

Falls die korrigierte Antwort auch beim ursprünglichen Abfrageproblem durch Enthaltenseintest der approximativen Antwort erhalten wird, reduziert sich der Zusatzaufwand auf $O(\delta \cdot t)$ bzw. $O(\delta \cdot I' \cdot t)$.

4.4.3 Speicheroptimierung

Im allgemeinen werden die genannten komprimierten Darstellungen nicht eindeutig sein. Es stellt sich daher die Frage nach einer kleinsten derartigen Darstellung, d.h. einer Darstellung mit kleinstem Speicherbedarf.

Für die Intervalldarstellung der Antwortdatenbank läßt sich die Suche nach einer kleinsten Darstellung auf das Problem des Handlungsreisenden reduzieren. Das Problem des Handlungsreisenden (engl. *traveling salesman problem (TSP)*) besteht darin, in einem kantengewichteten Graphen eine Anordnung der Knoten so zu finden, daß diese Anordnung einen Pfad induziert, dessen Summe der Kantengewichte minimal ist. Ein solcher Pfad heißt *minimale Rundreise.*

Theorem 4.10 (Reduktion der Intervalldarstellung auf ein TSP) *Gegeben sei eine Antwortdatenbank. Sei $G = (V, E)$ ein kantengewichteter, vollständiger Graph, dessen Knoten jeweils eine Antwortmenge als binären Vektor repräsentieren. Die Kanten seien mit dem Hamming-Abstand, d.h. der Anzahl unterschiedlicher Bits, der binären Vektoren ihrer Knoten gewichtet. Dann liefert eine minimale Rundreise eine Anordnung der Antwortmengen, für die die Intervalldarstellung eine minimale Anzahl von Intervallen enthält und umgekehrt.*

Beweis: Das Auffinden einer Anordnung der Antwortmengen, für die die Anzahl der Intervalle bei einer Intervalldarstellung der Antwortdatenbank minimal wird, bedeutet, eine Anordnung der Antwortmengen so zu finden, daß die Anzahl der 0/1 und 1/0-Wechsel in jeder Zeile der entsprechend permutierten Darstellung als binäres Feld minimal ist. Die Anzahl der 0/1 und 1/0-Wechsel zwischen zwei Spalten des binären Feldes ist gleich dem Hamming-Abstand dieser Spalten. Aus dieser Beobachtung ergibt sich die Behauptung. □

Die Version des TSP, die Gegenstand des Satzes ist, wird auch als *Hamming-TSP* bezeichnet. Es ist bekannt, daß das Hamming-TSP NP-schwer ist [EKP85]. Es kann also unter der Annahme, daß $P \neq NP$ ist, kein deterministischer Algorithmus erwartet werden, der eine Lösung in polynomieller Zeit findet.

Aufgrund der Komplexität der gestellten Aufgabe bieten sich näherungsweise optimale Lösungen an. Interessant sind in diesem Zusammenhang polynomielle Approximationsschemata.

Definition 4.27 (Polynomielles Approximationsschema für TSP)
Ein polynomielles Approximationsschema *ist ein polynomieller Algorithmus, der für jede Konstante $c > 1$ das Problem innerhalb eines Faktors $1 + 1/c$ approximativ lösen kann. Dabei kann die Laufzeit von c abhängen, aber für jede Konstante c ist sie polynomiell in der Eingabegröße.*

Für die Variante des *euklidischen* TSP, d.h. dort wo die Gewichte der Kanten den Abständen der beteiligten Knoten nach der l_2-Norm entsprechen, sind polynomielle oder sogar nahezu lineare Approximationsschemata bekannt [Aro96, Aro97, AGK$^+$98]. Unglücklicherweise ist das Hamming TSP aber kein euklidisches TSP, sondern ein *metrisches* TSP, bei dem die Gewichte der Dreiecksungleichung gehorchen. Es ist leicht einzusehen, daß dieses für die Hammingabstände gilt. Unter der Annahme, daß $P \neq NP$ gilt, wurde gezeigt, daß es für die Klasse der metrischen TSPs *keine* polynomiellen Approximationsschemata gibt [ALM$^+$92]. Das bedeutet aber nicht, daß es ein solches polynomielles Approximationsschema für das Hamming TSP nicht gibt. Es vermittelt allerdings einen Eindruck, wo die Komplexität der Approximation des Hamming TSP angesiedelt ist.

Eine Lösung, die eine Tour findet, die höchstens um den Faktor zwei länger ist als die gesuchte optimale Tour, beruht auf dem Besuchen des minimalen Spannbaums von G (siehe Abbildung 4.20). Dazu wird zunächst der minimale Spannbaum S von G konstruiert. Anschließend werden alle Kanten von S verdoppelt. Danach werden alle Knoten von S durch Ablaufen „außen herum" besucht (Abbildung 4.20, Mitte). Dabei werden einige Knoten mehr als einmal besucht. Dies kann jedoch vermieden werden, wenn beim Besuchen von S nach der genannten Strategie immer zu dem nächsten noch nicht besuchten Knoten gesprungen wird (Abbildung 4.20, rechts). Die Dreiecksungleichung stellt beim metrischen TSP sicher, daß sich durch diese „Abkürzungen" die Tour nicht verlängert.

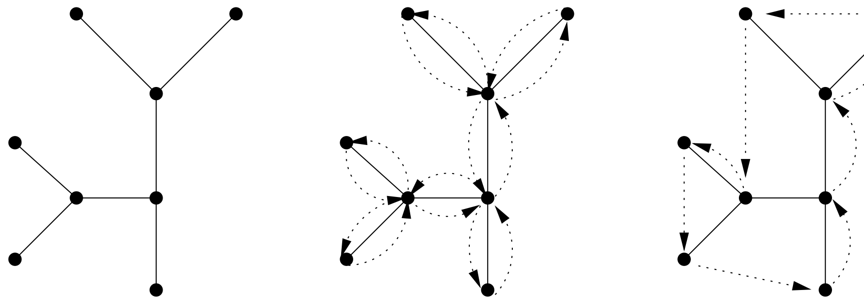


Abbildung 4.20: Konvertierung des minimalen Spannbaumes in eine approximierte TSP-Lösung

Seit längerem ist für die metrische Variante ein Approximationsalgorithmus bekannt, der in polynomieller Zeit läuft und Zyklen generiert, die maximal um den Faktor 1.5 schlechter sind als die optimale Lösung [Chr76]. Auch hierbei wird zunächst der minimale Spannbaum S von G konstruiert. Allerdings entfällt die Verdopplung der Kanten des vorherigen Algorithmus'. Statt dessen werden die Kanten von S durch die Kanten des perfekten Matchings minimalen Gewichts (engl. *minimum weight perfect matching*) auf dem vollständigen Graphen der Knoten ungeraden Grades von S und der entsprechenden Kantengewichte von G ergänzt. Ein perfektes Matching einer Knotenmenge V, $|V| = 2k$, besteht aus einer Menge von k Kanten, wobei jeder Knoten aus V nur zu genau einer dieser Kanten benachbart ist. Danach wird der ergänzte Baum S wie beim vorherigen Algorithmus besucht.

Zahlreiche andere approximative Lösungen, wie z.B. lokale Optimierungen durch Löschen von zwei oder mehreren Kanten und Neuverbindung der beteiligten Knoten (sogenannte *2-Opt* bzw. *3-Opt-Verfahren* [Lin65]), der *Lin-Kernighan-Algorithmus* [LK73] oder die Lösung durch *Simulated Annealing* [KGV83] sowie durch genetische Algorithmen [Bra85] sind bekannt und untersucht [JM97].

Der folgende Satz macht eine absolute Aussage über den Aufwand für die Intervalldarstellung einer Antwortdatenbank. Er nutzt dabei geometrische Information der betrachteten Anfrageprobleme, die über die rein kombinatorischen Betrachtungen hinausgeht, die bisher angestellt wurden.

Theorem 4.11 *Eine Szene S mit n Elementen, die mit s Abtastanfragen punktabgetastet wurde, die einen absoluten Fehler ε bei Nächster-Nachbar-Antwort garantieren, kann in eine Intervalldarstellung mit $O(\varepsilon s + n)$ Intervallen vorverarbeitet werden.*

Beweis: Es wird der minimale Spannbaum der Abtastanfragen betrachtet. Da der minimale Spannbaum Teilgraph der Delaunaytriangulierung der Abtastanfragen ist, verbindet jede seiner Kanten Abtastanfragen, die zu zwei benachbarten Voronoigebieten gehören. Die beiden Abtastanfragen lassen sich durch einen Pfad verbinden, der bis auf genau einen Schnittpunkt mit der gemeinsamen $(q-1)$-dimensionalen Trennwand im Inneren der beiden Voronoigebiete verläuft, q die Dimension des Anfrageraums. Bei einem absoluten Fehler von ε differieren die Antwortmengen der beiden Abtastanfragen höchstens um 2ε Szenenelemente, da für diesen Schnittpunkt die Differenz seiner Antwort zu den Antworten der Abtastantworten in beiden Fällen höchstens ε beträgt. Daraus folgt, daß zwischen zwei aufeinanderfolgenden Abtastpunkten höchstens 2ε Intervalle beginnen oder enden. Das führt zu insgesamt höchstens $2\varepsilon s' + n$ Intervallen, wobei s' die Anzahl der Punkte des Spannbaums ist, die beim Durchlauf aufgetreten sind. Dieser Durchlauf läßt sich durch Überspringen bereits durchlaufener Knoten zu einer Rundreise verkürzen, längs der sich die Anzahl der Intervalle nicht erhöht. Da $s' \leq 2s$, ergibt sich, daß die Anzahl der Intervalle auf der Rundreise durch $O(\varepsilon s + n)$ beschränkt ist. $\square$

Die Minimierung der δ-Containerdarstellung kann auf das Problem der Bestimmung einer minimalen Überdeckung in einem Unabhängigkeitssystem [LLK80, NT74] zurückgeführt werden, das wiederum als Hypergraphenfärbungsproblem interpretiert werden kann.

Definition 4.28 (Unabhängigkeitssystem, (un)abhängige Mengen)
Sei X eine endliche Menge und sei $\mathcal{E}$ eine nichtleere Familie von Teilmengen von X, die die folgende Bedingung erfüllen: Falls $E \in \mathcal{E}$ und $E' \subseteq E$, dann $E' \in \mathcal{E}$. Unter diesen Voraussetzungen wird $I = (X, \mathcal{E})$ ein Unabhängigkeitssystem *genannt. $\mathcal{E}$ ist die Familie der* unabhängigen Mengen *von I. Die Teilmengen von X, die nicht in $\mathcal{E}$ enthalten sind, sind* abhängige Mengen.

Das *korrespondierende Abhängigkeitssystem* $D(I)$ eines Unabhängigkeitssystems besteht aus den minimalen Teilmengen von X, die bzgl. I nicht unabhängig sind. Unabhängigkeits- und Abhängigkeitssysteme sind Beispiele für Hypergraphen [Ber73].

Definition 4.29 (Hypergraph)
Sei $X = \{x_1, x_2, \ldots, x_n\}$ eine endliche Menge, und sei $\mathcal{E} = \{E_i \mid i \in I\}$ eine Familie von Teilmengen von X. Die Familie $\mathcal{E}$ heißt Hypergraph auf X, *falls*

$$\begin{aligned} E_i &\neq \emptyset \quad (i \in I) \\ \bigcup_{i \in I} E_i &= X. \end{aligned}$$

Das Paar $H = (X, \mathcal{E})$ heißt Hypergraph. *Die Elemente $x_1, x_2, \ldots, x_n$ werden* Knoten, *die Mengen $E_1, E_2, \ldots, E_m$ werden* Kanten *von H genannt.*

Definition 4.30 (Knotenfärbung eines Hypergraphen)
Eine (Knoten-)Färbung *eines Hypergraphen ist eine Zuweisung von je einer Farbe an jeden Knoten, so daß keine Kante des Hypergraphs, die mehr als einen Knoten verbindet, Knoten gleicher Farbe verbindet. Eine* minimale (Knoten-)Färbung *eines Hypergraphen ist eine (Knoten-)Färbung mit der geringsten Anzahl an Farben.*

Aus diesen Definitionen folgt, daß die Mengen gleicher Farbe eines Abhängigkeitssystems unabhängige Mengen sind. Mit dieser Terminologie gilt:

Theorem 4.12 (Reduktion der δ-Containerdarstellung auf ein Färbungsproblem) *Die Menge der δ-Container einer Antwortdatenbank ist ein Unabhängigkeitssystem über der Menge der Antworten. Die Bestimmung einer minimalen δ-Containerdarstellung ist damit äquivalent zur Bestimmung einer minimalen Färbung im korrespondierenden Abhängigkeitssystem.*

Beweis: An der Eigenschaft, Container zu sein, ändert sich nichts, wenn aus einem Container Antworten entfernt werden. Also sind Container unabhängige Mengen. Aus einer minimalen Containerdarstellung ergibt sich unmittelbar eine zulässige Färbung des entsprechenden Abhängigkeitssystems mit einer Anzahl von Farben gleich der Anzahl der Container. Umgekehrt läßt sich aus einer Färbung unmittelbar eine Containerdarstellung mit der gleichen Containeranzahl wie die Farbanzahl ableiten. Daraus ergibt sich die Äquivalenz von minimaler Färbung und minimaler Containerdarstellung. $\square$

Das Problem der Berechnung einer minimalen Färbung von Hypergraphen ist NP-schwer [GJ79]. Das ergibt sich aus der entsprechenden Eigenschaft für Graphen. Allerdings ist im hier betrachteten Fall im Gegensatz zum klassischen Färbungsproblem der Hypergraph nicht explizit durch die übliche explizite Beschreibung der Kantenmenge gegeben. Daher ist die letzte Beobachtung nur von begrenztem Nutzen und eine spezifischere Analyse des Problems der Minimierung der δ-Container-Repräsentation wird benötigt.

Für die minimale Färbung von Graphen und Hypergraphen sind eine Reihe von approximativen Algorithmen bekannt, die hier angewendet werden können. Die wohl einfachste Vorgehensweise ist der *sequentielle Algorithmus.*

Definition 4.31 (Sequentielle Konstruktion von Containern)
Bei der sequentiellen Konstruktion *werden die Container aus den Abtastantworten in der Reihenfolge aufgebaut, in der diese generiert werden. Für jeden Container werden die Menge C aller Szenenelemente, die in Antworten auftreten, die in ihm enthalten sind, sowie die Mächtigkeit $k_{min}(C)$ der kleinsten in ihm enthaltenden Antwortmenge gespeichert. Für jede neue Antwort S' werden die Container daraufhin überprüft, ob die δ-Bedingung nach ihrer Hinzunahme erfüllt ist, d.h. $|R(C) \cup S'| \leq \delta \cdot \min\{k_{\min}(C), |S'|\}$. Falls ein Container gefunden wird, für den dies erfüllt ist, wird ihm die Antwort zugeordnet, $C := C \cup \{S'\}$ und $k_{\min}(C) := \min\{k_{\min}(C), |S'|\}$ gesetzt. Falls die Antwort in keinen Container paßt, wird ein neuer Container angelegt, S' in ihn eingefügt und $R(C) := S'$, $k_{\min}(C) := |S'|$ gesetzt.*

Die Komprimierung, die mit dieser Lösung erreicht werden kann, ist stark davon abhängig, in welcher Reihenfolge die Abtastantworten übergeben werden. Verschiedene Permutationen derselben Abtastantworten generieren eine unterschiedliche Anzahl an Containern, deren Repräsentanten eine unterschiedliche Anzahl von Szenenelementen umfassen können.

Definition 4.32 (Sequentielle Konstruktion mit Containerklassen)
Bei der sequentiellen Konstruktion von Containern mit Containerklassen sind Containerklassen unterschiedlicher Größe vorgegeben. Die k-te Klasse, $k \geq 0$, enthält alle Container, deren Antwortmengen minimal $\sum_{i=0}^{k-1} \delta^i$ und maximal $\sum_{i=1}^{k} \delta^i - 1$ Elemente enthalten.

Die Container werden aus den Abtastantworten in der Reihenfolge konstruiert, in der diese generiert werden. Für jede einzufügende Abtastantwort S' werden zunächst alle Container bestimmt, für die $k_{min}(C) \leq |S'| \leq k_{\max}(C)$ gilt, wobei $k_{\max}(C)$ die Mächtigkeit der größten Antwort von C ist. Falls es keinen solchen Container gibt, wird ein neuer Container der kleinsten Größe $k_{\max}(C)$ generiert, der die Obermengenbedingung erfüllt, und der Algorithmus stoppt. Falls ein oder mehrere Container die Bedingung erfüllen, wird der erste Container bestimmt, dessen Repräsentant $R(C)$ die Menge S' komplett enthält, und der Algorithmus stoppt. Falls kein solcher Container existiert, wird ein neuer Container der kleinsten Größe $k_{\max}(C)$ generiert, der die Obermengenbedingung erfüllt, und der Algorithmus stoppt.

Unglücklicherweise ist die Güte der Komprimierung, d.h. die Anzahl der erzeugten Container, von der Strategie abhängig, wie neue Container generiert werden. Bei dieser Generierung gibt es bei der Bestimmung der in der Abtastantwort fehlenden Einsen Freiheiten für die Positionierung innerhalb des Containers. Ist beispielsweise $\delta = 2$, so haben die zweitkleinsten Container die minimale Größe 3 und die maximale Größe 6. Falls eine Abtastantwort der Größe 3 in die Antwortdatenbasis eingefügt werden soll und falls noch kein Container der Größe 6 existiert, so muß ein neuer Container dieser Größe generiert werden. Dabei sind drei Einsen neben den schon in der Antwort vorhandenen drei Einsen frei zu positionieren. Eventuell entsteht dabei ein Container, der eine folgende Antwort der Größe 3 nicht aufnehmen kann, da die oben genannten drei freien Einsen dafür falsch positioniert wurden.

Weitere Verfahren zur Konstruktion von Containerdarstellungen ergeben sich aus den beiden anderen, nichtapproximativen Darstellungsverfahren für Antwortdatenbanken, der Intervalldarstellung und der Gebietsdarstellung.

Definition 4.33 (Container-Intervalldarstellung)
Gegeben sei eine Abtastung für ein Anfrageproblem. Eine Container-Intervalldarstellung *ist eine Intervalldarstellung von Antwortmengen, die sich aus denen der Abtastung dadurch ergeben, daß eine Antwortmenge durch irgendeinen für sie möglichen Containerrepräsentanten ersetzt wird.*

Definition 4.34 (Container-Gebietsdarstellung)
Gegeben sei eine Abtastung für ein Anfrageproblem. Eine Container-Gebietsdarstellung

ist eine Gebietsdarstellung von Antwortmengen, die sich aus denen der Abtastung dadurch ergeben, daß eine Antwortmenge durch irgendeinen für sie möglichen Containerrepräsentanten ersetzt wird.

Die Möglichkeit, eine Abtastantwort durch irgendeinen Containerrepräsentanten zu ersetzen, ermöglicht es im allgemeinen, längere Intervalle und größere Gebiete zu bilden. Eine Möglichkeit der Konstruktion ist, von einer Intervall- bzw. Gebietsdarstellung auszugehen und durch sukzessives Ersetzen von Antwortmengen durch passende Containerrepräsentanten größere Intervalle bzw. Gebiete zu bilden. Die Menge der so gewonnenen Containerrepräsentanten definiert unabhängig von der speziellen Vorgehensweise eine Containerdarstellung. In diesem Sinn können die Container-Intervall und -Gebietsdarstellung auch als Konstruktionsverfahren für Container unabhängig von der Darstellung angesehen werden. Der Vorteil der Intervall- und Gebietsdarstellung besteht darin, daß diese zugleich eine Darstellung der Repräsentantendatenbank liefern.

4.4.4 Experimentelle Untersuchung des Streckenanfrageproblems mit Geraden

Die im folgenden dargestellten beiden experimentellen Untersuchungen des Containerverfahrens für die Anfrage mit Geraden an eine Szene von Strecken in der Ebene verwenden im Prinzip die sequentielle Konstruktion von Containern. Allerdings werden die Abtastanfragen und die Reihenfolge ihrer Bearbeitung nach speziellen Heuristiken gewählt. Die Untersuchungen wurden im Rahmen von [Kuk97] durchgeführt.

Experiment 4.8 (Speicherbedarf des Containeransatzes im Vergleich zu der unkomprimierten Speicherung für Abtastanfragen am Rand)

Eingabe: *Eine ebene Szene aus zufällig verteilten Strecken. Als Abtastanfragen werden Geraden genommen, die Eckpunkte von Szenenstrecken verbinden. Die Punkte, die die Abtastanfragen repräsentieren, werden durch eine Delaunay-Triangulierung verbunden. Die Delaunay-Triangulierung wird in Breitensuche durchlaufen, um aus den Antworten in dieser Reihenfolge einen δ-Container zu konstruieren.*

Parameter: $\delta = 2$. *Die Anzahl der Abtastanfragen ist so gewählt, daß der Fehler der korrigierten Antwort kleiner als* 5% *ist.*

Plattform: *SGI Indigo2, 200MHz R4400 Prozessor.*

Gemessene Größe(n): *Vgl. Abbildung 4.21: Speicherbedarf bei nicht komprimierter Durchführung der Punktabtastung mit Antwortkonstruktion* *nach dem Nächster-Nachbar- (NN=1) bzw. dem Zwei-Nächste-Nachbarn-Verfahren (NN=2) sowie bei Komprimierung mit dem Containeransatz (StarCon2D), gemittelt über 100 zufällige Anfragen (y-Achse), in Abhängigkeit von der Anzahl der Strecken in der Szene (x-Achse).*

Beobachtung: *Deutlich zu sehen ist der Verlauf der Kurve der komprimierten Antwortdatenbank unterhalb der Kurve der unkomprimierten Datenbank ab einer Szenengröße von etwa 10 Strecken. Danach orientiert sich der*

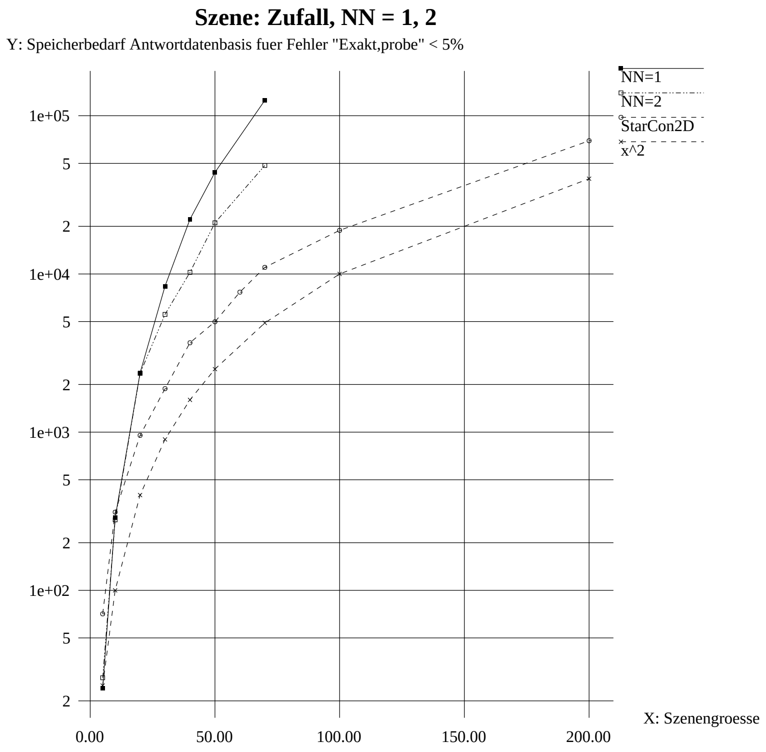


Abbildung 4.21: Vergleich des Speicherbedarfs des Containeransatzes mit der unkomprimierten Speicherung der Abtastantworten

Verlauf der Kurve der komprimierten Datenbank eher an der Kurve der quadratischen Abhängigkeit, während sich die Kurve der unkomprimierten Datenbank mit zunehmender Szenengröße weiter von letzterer Kurve entfernt. Zur Beurteilung des Wachstums ist ferner eine Kurve mit quadratischer Abhängigkeit in das Diagramm eingezeichnet (untere Kurve).

Der Vorteil der gewählten Abtastanfragen liegt darin, daß in ihrer Umgebung besonders viele Szenenobjekte liegen. Betrachtet man die Szenenobjekte im Dualraum, bedeutet dies nämlich, daß sie auf dem Rand mehrerer Szenenobjekte liegen. Es ist daher davon auszugehen, daß auf diese Weise eine bessere Approximation als mit der gleichen Anzahl an Abtastanfragen von anderen Abtaststrategien erzielt werden kann. Werden in dem Anwendungsbeispiel von Strecken alle $O(n^2)$ Geraden, die Streckenendpunkte verbinden, genommen, werden alle möglichen Antworten überdeckt.

Durch die Delaunay-Triangulierung und die Breitensuche wird eine Anordnung der Abtastantworten angestrebt, bei denen aufeinanderfolgende Antwortmengen einen kleinen Hamming-Abstand besitzen.

Experiment 4.9 (Speicherbedarf des Containeransatzes im Vergleich zu der unkomprimierten Speicherung für Abtastanfragen aus einer Quadtree-Unterteilung)

Eingabe: *Eine ebene Szene aus zufällig verteilten Strecken. Als Abtastanfragen werden die Mittelpunkte der Zellen eines vollständigen Quadtrees über dem Anfrageraum einer gewählten Tiefe d genommen. Zur Konstruktion der $\delta -$ Container werden die Knoten von den Blättern zur Wurzel* (bottom-up) *zu Containern zusammengefaßt, solange die Antwort des Mittelpunktes eines Knotens zusammen mit den Containerrepräsentanten seiner Nachfolgerknoten zu einem Container zusammenfaßbar sind.*

Parameter: $d = 7$, $\delta = 2$. *Die Anzahl der Abtastanfragen ist so gewählt, daß der Fehler der korrigierten Antwort kleiner als 5% ist.*

Plattform: *SGI Indigo2, 200MHz R4400 Prozessor.*

Gemessene Größe(n): *Vgl. Abbildung 4.22: Speicherbedarf bei nicht komprimierter Durchführung der Punktabtastung mit Antwortkonstruktion nach dem Nächster-Nachbar- (NN=1) bzw. dem Zwei-Nächste-Nachbarn-Verfahren (NN=2) sowie bei Komprimierung mit dem Containeransatz (QuadConUp), gemittelt über 100 zufällige Anfragen (y-Achse), in Abhängigkeit von der Anzahl der Strecken in der Szene (x-Achse).*

Beobachtung: *Deutlich ist der Verlauf der Kurve der komprimierten Antwortdatenbank unterhalb der Kurve der unkomprimierten Datenbank ab einer Szenengröße von 20 Strecken zu sehen. Zur Beurteilung des Wachstums ist ferner eine Kurve mit quadratischer Abhängigkeit in das Diagramm eingezeichnet (untere Kurve).*

Das Verfahren hat die Eigenschaft, daß die initialen, kleinen Container auf dem Weg zur Wurzel größer werden und damit größere Gebiete im Anfrageraum überdecken. Die so entstehenden Container sind adaptiv an die Verhältnisse im Anfrageraum angepaßt.

Der Vergleich der beiden Experimente zeigt, daß das Kompressionsverhalten des ersten Experiments für kleinere Szenen besser als das des zweiten Experiments ist. Dies ist wohl auf eine geringere Anzahl von Abtastanfragen wegen der besseren Überdeckungseigenschaft zurückzuführen.

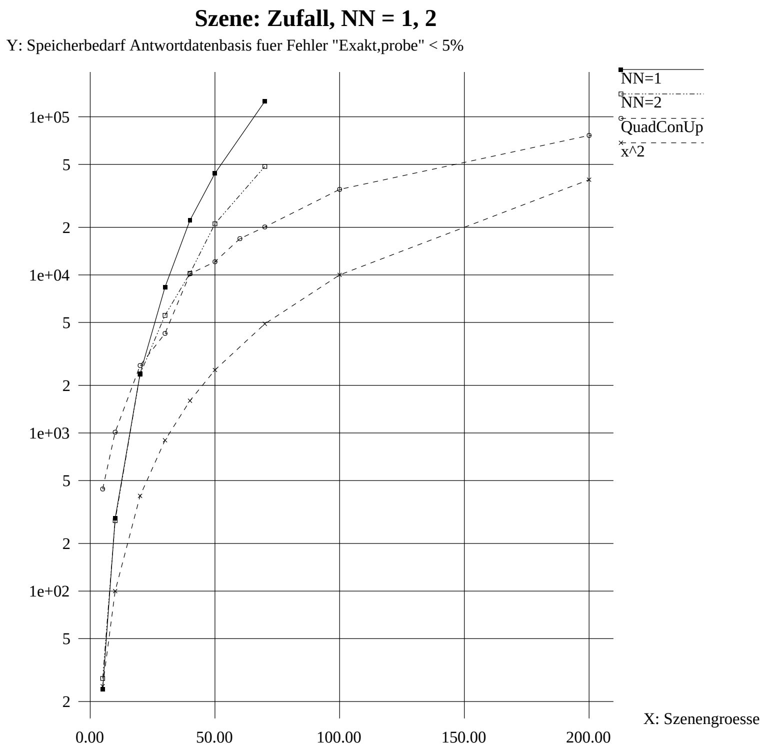


Abbildung 4.22: Vergleich des Speicherbedarfs des bottom-up Ansatzes mit der unkomprimierten Speicherung der Abtastantworten

Literaturverzeichnis

[AGK+98] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 33–41, San Francisco, California, 25–27 January 1998.

[AK87] J. Arvo and D. Kirk. Fast ray tracing by ray classification. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 55–64, July 1987.

[AK89] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In A. S. Glassner, editor, *An Introduction to Ray Tracing*, pages 201–262. Academic Press, 1989.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 1992.

[AMN95] S. Arya, D. Mount, and O. Narayan. Accounting for boundary effects in nearest neighbor searching. In *Proceedings of the 11th Annual Symposium on Computational Geometry*, pages 336–344, New York, NY, USA, June 1995. ACM Press.

[APS93] B. Aronov, M. Pellegrini, and M. Sharir. On the zone of an algebraic surface in a hyperplane arrangement. *Discrete and Computational Geometry*, 9:177–188, 1993.

[Aro96] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *37th Annual Symposium on Foundations of Computer Science*, pages 2–11, Burlington, Vermont, 14–16 October 1996. IEEE.

[Aro97] S. Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *38th Annual Symposium on Foundations of Computer Science*, pages 554–563, Miami Beach, Florida, 20–22 October 1997. IEEE.

[AS91] P. K. Agarwal and M. Sharir. Applications of a new space partitioning technique. In F. Dehne, J.-R. Sack, and N. Santoro, editors, *Algorithms and Data Structures, 2nd Workshop WADS '91*, volume 519 of *Lecture Notes in Computer Science*, pages 379–391, Ottawa, Canada, 14–16 August 1991. Springer-Verlag.

[Aur87] Franz Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):77–96, February 1987.

[Ben77] J. L. Bentley. Solutions to Klee's rectangles problems. *(Unpublished notes)*, pages 1–15, September 1977.

[Ber73] C. Berge. *Graphs and Hypergraphs.* North Holland Publ. Comp., Amsterdam, 1973.

[BF79] J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACM Comput. Surv.*, 11:397–409, 1979.

[Bra85] R. M. Brady. Optimization strategies gleaned from biological evolution. *Nature*, 317:804–806, 1985.

[CEGS89] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Lines in space—combinatorics, algorithms and applications. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 382–393, Seattle, Washington, 15–17 May 1989.

[Cha82] B. Chazelle. Filtering search: A new approach to query-answering. In *24th Annual Symposium on Foundations of Computer Science*, pages 122–132, Los Alamitos, Ca., USA, November 1982. IEEE Computer Society Press.

[Cha85] B. Chazelle. Fast searching in a real algebraic manifold with applications to geometric complexity. In Maurice Nivat Hartmut Ehrig, Christiane Floyd and James Thatcher, editors, *Mathematical foundations of software development: Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT 85): volume 1 - Colloquium on Trees in Algebra and Programming (CAAP '85)*, volume 185 of *LNCS*, pages 145–156, Berlin, FRG, March 1985. Springer.

[Chr76] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegy–Mellon University, 1976.

[CS90] B. Chazelle and M. Sharir. An algorithm for generalized point location and its application. *J. Symbolic Comput.*, 10:281–309, 1990.

[CW93] M. F. Cohen and J. R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Boston, MA, 1993.

[DDP96] F. Durand, G. Drettakis, and C. Puech. The 3D visibility complex: A new approach to the problems of accurate visibility. In X. Pueyo and P. Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 245–256, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

[DE87] D. P. Dobkin and H. Edelsbrunner. Space searching for intersecting objects. *J. Algorithms*, 8:348–361, 1987.

[deB93] M. deBerg. Ray shooting, depth orders and hidden surface removal. In *Lecture Notes in Computer Science*, volume 703. Springer Verlag, New York, 1993.

[DL76] D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM Journal on Computing*, 5(2):181–186, June 1976.

[Ede80] H. Edelsbrunner. Dynamic data structures for orthogonal intersection queries. Research Report F59, Universität Graz, Inst. für Informationsverarbeitung, Graz, Österreich, 1980.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, November 1987.

[EKP85] J. Ernvall, J. Katajainen, and M. Penttonen. NP-completeness of the Hamming salesman problem. *BIT*, 25(1):289–292, 1985.

[Fou81] L. R. Foulds. *Optimization Techniques*. Springer, New York, 1981.

[GCS88] Z. Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. In *Second International Conference on Computer Vision (Tampa,, FL, December 5–8, 1988)*, pages 30–39, Washington, DC,, 1988. Computer Society Press.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, New York, 1979.

[Gla95] A. S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann, San Francisco, CA, 1995.

[GO97] J. E. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry*, volume 6 of *CRC Press Series on Discrete Mathematics and its Applications*. CRC Press, 1997.

[GTGB84] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modelling the Interaction of Light Between Diffuse Surfaces. In *Computer Graphics (ACM SIGGRAPH '84 Proceedings)*, volume 18, pages 212–222, July 1984.

[Han91] E. N. Hanson. The interval skip list: A data structure for finding all intervals that overlap a point. In F. Dehne, J.-R. Sack, and N. Santoro, editors, *Algorithms and Data Structures, 2nd Workshop WADS '91*, volume 519 of *Lecture Notes in Computer Science*, pages 153–164, Ottawa, Canada, 14–16 August 1991. Springer-Verlag.

[HM97] A. Hinkenjann and H. Müller. General visibility. In *Intelligent Robots – Sensing, Modelling and Planning*, volume 27 of *Series in Machine Perception and Artificial Intelligence*. World Scientific Publishing, 1997.

[HSA91] P. Hanrahan, D. Salzman, and L. Aupperle. A Rapid Hierarchical Radiosity Algorithm. In *Computer Graphics (ACM SIGGRAPH '91 Proceedings)*, volume 25, pages 197–206, July 1991.

[JM97] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization, Wiley*. 1997.

[Kaj86] J. T. Kajiya. The Rendering Equation. In *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, volume 20, pages 143–150, August 1986.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimisation by simulated annealing. *Science*, 220:671–680, 1983.

[Kuk97] M. Kukuk. Kompakte Antwortdatenbasen für die Lösung von geometrischen Anfrageproblemen durch Abtastung. Diplomarbeit, Informatik VII, Universität Dortmund, September 1997. Betreuer: A. Hinkenjann, H. Müller.

[Lin65] S. Lin. Computer Solutions of the Traveling Salesman Problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[LK73] S. Lin and W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Op. Res.*, 21(2), 1973.

[LLK80] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

[Mat91] J. Matoušek. Efficient partition trees. In *Proceedings of the 7th Annual Symposium on Computational Geometry (SCG '91)*, pages 1–9, North Conway, NH, USA, June 1991. ACM Press.

[Mat92] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proceedings of the 8th Annual Symposium on Computational Geometry (SCG '92)*, pages 276–285, Berlin, FRG, June 1992. ACM Press.

[McC80] E. M. McCreight. Efficient algorithms for enumerating intersecting intervals and rectangles. Technical Report CSL-80-9, Xerox Palo Alto Research Center Technical Report, 1980.

[McC82] E. M. McCreight. Priority search trees. Technical Report CSL-81-5, Xerox Palo Alto Research Center Technical Report, January 1982.

[McC83] G. P. McCORMICK. *Nonlinear Programming: Theory, Algorithms, and Applications.* Wiley-Interscience, New York, NY, 1983.

[Mül88] H. Müller. *Realistische Computergraphik.* Springer-Verlag, Berlin, Heidelberg, New York, 1988.

[NN85] T. Nishita and E. Nakamae. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. In *Computer Graphics (ACM SIGGRAPH '85 Proceedings)*, volume 19, pages 23–30, July 1985.

[NT74] G. L. Nemhauser and L. E. Trotter Jr. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6:48–61, 1974.

[ORDP96] R. Orti, S. Riviere, F. Durand, and C. Puech. Using the Visibility Complex for Radiosity Computation. In *Lecture Notes in Computer Science (Applied Computational Geometry: Towards Geometric Engineering)*, volume 1148, pages 177–190, Berlin, Germany, May 1996. Springer-Verlag.

[Ous93] J. K. Ousterhout. *An Introduction to Tcl and Tk.* Addison-Wesley, Reading, MA, USA, 1993.

[Plü65] J. Plücker. On a new geometry of space. *Philos. Trans. Royal Soc. London*, 155:725–791, 1865.

[PV93] M. Pocchiola and G. Vegter. The visibility complex. In *Proceedings of the 9th Annual Symposium on Computational Geometry (SCG '93)*, pages 328–337, San Diego, CA, USA, May 1993. ACM Press.

[Sam84] H. J. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[Sam89a] H. J. Samet. *Applications of Spatial Data structures: Computer Graphics, Image Processing, and GIS.* Addison–Wesley, Redding, MA, 1989.

[Sam89b] H. J. Samet. *Design and analysis of Spatial Data Structures: Quadtrees, Octrees, and other Hierarchical Methods.* Addison–Wesley, Redding, MA, 1989.

[Sil94] F. Sillion. Clustering and Volume Scattering for Hierarchical Radiosity Calculations. In *Fifth Eurographics Workshop on Rendering*, pages 105–117, Darmstadt, Germany, June 1994.

[Sto91] J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations.* Academic Press, 1991.

[TH93] S. Teller and P. Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 239–246, 1993.

[TS91] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In T. W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 61–69, July 1991.

[Wel88] E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proceedings of the Fourth Annual Symposium on Computational Geometry (Urbana-Champaign, IL, June 6–8, 1988)*, pages 23–33, New York, 1988. ACM, ACM Press.

[Whi79] T. Whitted. An improved illumination model for shaded display. *Computer Graphics*, 13(2):14–14, August 1979.

[Wil82] D. E. Willard. Polygon retrieval. *SIAM Journal on Computing*, 11(1):149–165, 1982.

[YN97] F. Yamaguchi and M. Niizeki. Some basic geometric test conditions in terms of plücker coordinates and plücker coefficients. *The Visual Computer*, 13(1):29–41, 1997. ISSN 0178-2789.

[YY85] A. C. Yao and F. Frances Yao. A general approach to d-dimensional geometric queries (extended abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 163–168, Providence, Rhode Island, 6–8 May 1985.