Globale Sichtbarkeitsalgorithmen

Dissertation
zur
Erlangung des Doktorgrades
der Naturwissenschaften
(Dr. rer. nat.)

dem

Fachbereich Mathematik und Informatik

der Philipps-Universität Marburg
vorgelegt von

Axel Schröder

aus Eschwege

Marburg/Lahn 2003

Vom Fachbereich Mathematik und Informatik der Philipps-Universität Marburg
als Dissertation am ***6.6.2003*** angenommen.

Erstgutachter: ***Prof. Dr. Manfred Sommer***

Zweitgutachter: ***Prof. Dr. Wolfgang Hesse***

Tag der mündlichen Prüfung am ***2.7.2003***

Inhaltsverzeichnis

1 Einleitung

Wenn man im Kontext von Computer-Grafiksystemen von „Sichtbarkeit“ spricht, ist meist ein Spezialfall gemeint: die Sichtbarkeit von einem Punkt (Augpunkt) zu den 3D-Objekten einer 3D-Szene. Beispiele hierfür sind *CAD-Anwendungen*, *Raytracer* oder auch die in den letzten Jahren beliebten Computerspiele mit „*First-Person-Interface*“[1].

In dieser Arbeit wird jedoch ein allgemeinerer und sehr viel komplexerer Begriff der Sichtbarkeit betrachtet: die Sichtbarkeit zwischen zwei beliebigen 3D-Objekten bzw. die globale Sichtbarkeit innerhalb einer kompletten 3D-Szene.

Die Information über die gegenseitige Sichtbarkeit wird von globalen Beleuchtungsmodellen wie dem Radiosity-Beleuchtungsmodell benötigt, um den Lichtfluss innerhalb einer 3D-Szene, d.h. die Beleuchtung dieser 3D-Szene, zu berechnen. Anschaulich entsprechen Sichtbarkeitsinformationen „Schattenlinien“ in der 3D-Szene. Umgangssprachlich kann man die Berechnung der globalen Sichtbarkeit innerhalb einer 3D-Szene also als Berechnung aller Schatten in dieser 3D-Szene beschreiben.

In dieser Arbeit werden zwei grundsätzlich verschiedene Ansätze zur Berechnung der globalen Sichtbarkeit innerhalb einer 3D-Szene betrachtet: Näherungs- und exakte Verfahren. Der Schwerpunkt der Arbeit liegt auf den exakten Verfahren und deren wichtigstem Vertreter, dem so genannten Visibility Skeleton einer 3D-Szene.

In den folgenden Kapiteln wird gezeigt, dass viele für die Näherungsverfahren entwickelten Ansätze und Algorithmen auch für die Berechnung des Visibility Skeletons verwendet werden können.

Da der Aufwand zur Konstruktion des Visibility Skeletons einer 3D-Szene auch mit den in dieser Arbeit vorgeschlagenen Verbesserungen sehr groß ist, wird abschließend ein Verfahren vorgeschlagen, dass nur den für globale Beleuchtungsmodelle „interessanten“ Teil des Visibility Skeletons berechnet.

---

[1] Dies bedeutet einfach, dass der Spieler die Spielwelt aus der Perspektive einer virtuellen Spielperson sieht und sich durch Tastatur- oder Mausinteraktion durch diese Spielwelt bewegen kann.

1.1 Übersicht

Die folgende Übersicht soll dem Leser einen ersten Eindruck der in dieser Arbeit diskutierten Themen geben. An dieser Stelle soll betont werden, dass die einzelnen (Teil-) Kapitel aufeinander aufbauen und somit die Lektüre in der vorgegebenen Reihenfolge angeraten ist!

Die Arbeit ist wie folgt gegliedert:

- ***Kapitel 2: Grundlagen***

  In diesem Kapitel werden wichtige Begriffe, Probleme und Lösungsansätze für den Rest der Arbeit vorgestellt. Dazu werden zunächst einfache und komplexe geometrische Objekte sowie Beziehungen zwischen diesen betrachtet und die bereits erwähnten Begriffe 3D-Objekt und 3D-Szene definiert. Den Abschluss des Kapitels bildet eine Einführung in das Radiosity-Beleuchtungsmodell, das die diffuse Lichtausbreitung in 3D-Szenen beschreibt.

- ***Kapitel 3: Globale Sichtbarkeitsalgorithmen***

  In diesem Kapitel werden die beiden wichtigsten Vertreter von Verfahren zur Berechnung der Sichtbarkeit in 3D-Szenen betrachtet: Näherungs- und exakte Verfahren. Für beide Verfahren werden Ansätze zur Reduktion des Rechenaufwands vorgeschlagen und ihre Eignung für das Radiosity-Beleuchtungsmodell diskutiert. Die theoretischen Betrachtungen werden durch Messungen anhand konkreter 3D-Szenen untermauert.

- ***Kapitel 4: Direct Illumination Visibility Skeleton***

  Den Anfang des Kapitels bildet ein Vergleich der Näherungs- und exakten Verfahren. Dieser Vergleich ergibt, dass die Näherungsverfahren gut für indirekte Beleuchtung und die exakten Verfahren gut für direkte Beleuchtung geeignet sind. Aus den Ergebnissen von Kapitel 3 folgt, dass die Trennung in indirekte und direkte Beleuchtung zwar für die Näherungsverfahren trivial möglich ist, bei den exakten Verfahren jedoch nur mit großem Aufwand durchgeführt werden kann.

Daher ist das zentrale Problem dieses Kapitels die Anpassung der exakten Verfahren an die direkte Beleuchtung in 3D-Szenen. Hierfür wird eine neue Datenstruktur, das so genannte *Direct Illumination Visibility Skeleton*, vorgestellt und die Konstruktion desselben ausführlich diskutiert.

Den Abschluss des Kapitels bilden wieder Messungen anhand konkreter 3D-Szenen.

- ***Kapitel 5: GloVe – Global Visibility Environment***

  Da die in Kapitel 3 und 4 betrachteten Algorithmen sehr komplex und aus vielen, nicht-trivialen Einzelschritten aufgebaut sind, können theoretische Aussagen zum Rechen- und Speicheraufwand nur sehr grob getroffen werden. Daher wurden alle Algorithmen in einer im Rahmen dieser Arbeit implementierten Testumgebung namens GloVe untersucht und bewertet. Das Programm GloVe wird in seiner grundlegenden Struktur und seinen wesentlichen Eigenschaften vorgestellt.

- ***Kapitel 6: Zusammenfassung***

  Hier werden die wichtigsten Ergebnisse der Arbeit zusammengestellt und bewertet. Insbesondere wird dabei aufgezeigt, an welchen Stellen diese Arbeit fortgeführt werden könnte.

1.2 Bemerkungen

Jede wissenschaftliche Arbeit hat ihren eigenen Stil und ihre eigene Art, Dinge darzustellen und zu veranschaulichen. Eine verbindliche Regelung, was „guter Stil“ oder was eine „gute Darstellung“ ist, gibt es nicht. Daher sollen an dieser Stelle einige Aussagen zur verwendeten Form der Arbeit gemacht werden:

- ***Stil: Passiv und unpersönlich vs. aktiv und persönlich***

  Für die Anrede des Lesers gibt es viele Möglichkeiten. Hier wurde der passive und unpersönliche Stil gewählt, d.h. der Leser wird nicht mit „Sie“ oder gar „Du“ angesprochen und der Autor kommt nicht explizit als „Ich“ oder „Wir“ vor.

- ***Stil: Deutsch vs. Englisch***

  In deutschen Texten zur Informatik werden deutsche und englische Begriffe oft vermischt, da viele Begriffe in der Informatik ursprünglich aus dem Englischen stammen. Daraus ergibt sich ein unschönes Kauderwelsch, das zwar korrekt, aber nur schwer lesbar ist. Daher wurden in dieser Arbeit englische Begriffe wenn möglich ins Deutsche übersetzt. Begriffe, deren deutsche Übersetzung „holprig" klingt, wurden in der ursprünglichen Form belassen.

- ***Darstellung: Anschaulichkeit vs. Formalismus***

  Diese Arbeit beschäftigt sich zum größten Teil mit geometrischen Problemstellungen. Diese Probleme können durch mathematische Formulierungen exakt gefasst werden, sind aber oft nur wenig anschaulich. Daher werden an vielen Stellen anschauliche Begriffe wie „links", „rechts", „über", „unter" oder „in" verwendet. Diese beziehen sich meist auf eine konkrete Abbildung, in der dann z.B. ein Punkt tatsächlich „über" einer Ebene liegt.

- ***Darstellung: Algorithmen vs. Prosa***

  Algorithmen können auf viele verschiedene Arten dargestellt werden: Flussdiagramme, Ablaufdiagramme, (Pseudo-) Code, … In dieser Arbeit werden Algorithmen anhand von Code-Beispielen angegeben. Diese Code-Beispiele sind in einer Mischung aus Java-Syntax und „Umgangssprache" abgefasst. Dadurch ist einerseits ein gewisser Formalismus gewahrt, der aber andererseits nicht in „Bandwurm-Code" ausartet und den Blick auf das Wesentliche verstellt.

- ***Darstellung: 3D vs. 2D***

  Diese Arbeit beschäftigt sich hauptsächlich mit Problemen im 3D. Für viele Abbildungen und Verfahren ist es jedoch hilfreich, das Problem zunächst im 2D zu betrachten, da sich dort viele Eigenschaften und Zusammenhänge besser darstellen lassen und einfacher zu verstehen sind.

  Dies führt allerdings zum Problem, dass der Leser an vielen Stellen zwischen 2D und 3D umdenken muss. Dies wird in den folgenden Kapiteln meist implizit vorausgesetzt, d.h. es wird nicht jedes Mal darauf hingewiesen, dass eine Abbildung eine 2D-Repräsentation eines 3D-Problems ist.

2 Grundlagen

In diesem Kapitel sollen die wesentlichen Grundlagen für den Hauptteil der Arbeit eingeführt werden. Viele der dabei behandelten Themen werden ausführlich in verschiedenen (populär-) wissenschaftlichen Büchern behandelt (vgl. z.B. [58] oder [59]) und dürften dem an der 3D-Grafik interessierten Leser daher vertraut sein.

Hier geht es nicht um eine vollständige Darstellung dieser Themen, sondern lediglich um eine Zusammenstellung der wichtigsten Begriffe, Probleme und Lösungsansätze. Besonderer Wert wird dabei auf einheitliche Darstellung und Schreibweise gelegt, da sich eine solche bisher in den einschlägigen Veröffentlichungen nicht durchgesetzt hat.

Zunächst werden elementare *geometrische Primitive* wie Linien, Ebenen, Polygone und Polyeder erklärt. Dies führt zum speziellen Problem der Berechnung von Schnitten zwischen diesen Primitiven und den allgemeineren *Klassifikationen von Primitiven*. Anschaulich ist eine Klassifikation eine qualitative Aussage über die relative Lage zweier Primitive, z.B. ob ein Punkt „über", in oder „unter" einer Ebene liegt.

Der mathematische Begriff der geometrischen Primitiven wird dann in die informatischen Begriffe der *3D-Entitäten*, *3D-Objekte* und (3D-)*Elemente* überführt, d.h. es werden Datenstrukturen zur Speicherung von Primitiven vorgestellt.

Eine Menge von 3D-Objekten wird als *3D-Szene* bezeichnet. Zunächst wird die in dieser Arbeit betrachtete Klasse von 3D-Szenen genau definiert. Dann werden zwei Möglichkeiten zur Speicherung von 3D-Szenen diskutiert: Listen und Hierarchien von 3D-Objekten.

Für die im Hauptteil der Arbeit betrachteten Verfahren zur Sichtbarkeitsberechnung spielen *konvexe Hüllen von 3D-Entitäten* eine zentrale Rolle, daher werden die in diesem Kontext wichtigsten Begriffe und Verfahren kurz vorgestellt.

Den Abschluss dieses Kapitels bildet die Diskussion des *Radiosity-Beleuchtungsmodells* zur Berechnung der diffusen Beleuchtung einer 3D-Szene. Dieses Modell ist der Ausgangspunkt für alle weiteren Betrachtungen der Sichtbarkeit in 3D-Szenen. Da Details, Varianten und Optimierungen von Radiosity-Berechnungen im weiteren Verlauf nicht wichtig sind, wird das Radiosity-Beleuchtungsmodell lediglich im Ansatz und anhand von einfachen Beispielen diskutiert.

2.1 Geometrische Primitive

Wesentliche Teile dieser Arbeit beschäftigen sich mit geometrischen Objekten und Beziehungen. Daher sollen an dieser Stelle einige Grundlagen der Geometrie eingeführt werden. Zunächst werden die *geometrischen Primitive* (elementare Objekte) einzeln diskutiert und anschließend in Kap. 2.2 die für diese Arbeit grundlegenden Klassifikationen von Primitiven erklärt und bewertet.

2.1.1 Punkte und Vektoren

Sei $\mathbf{R}$ der Körper der *reellen Zahlen*. Ein $\mathbf{n}$-dimensionaler Punkt oder Vektor ist das Koordinaten-$\mathbf{n}$-Tupel $\mathbf{p} = (p_1, p_2, \ldots, p_n)^T \in \mathbf{R}^n$ des über $\mathbf{R}$ definierten $\mathbf{n}$-dimensionalen Vektorraumes. Syntaktisch sind Punkte und Vektoren also gleich, semantisch gibt es jedoch einen wesentlichen Unterschied: Punkte beschreiben einen *Ort* relativ zum Ursprung des Vektorraums, Vektoren dagegen geben eine ortsunabhängige *Richtung* in diesem Vektorraum an.

2.1.2 Linien und Ebenen

Die im Folgenden betrachteten geometrischen Konstrukte sind aus *Linien* und *Ebenen* aufgebaut. Zunächst werden Darstellungen von Linien und Ebenen betrachtet: die *implizite Darstellung* gibt eine Bedingung an, wann ein Punkt zu einer Linie bzw. Ebene gehört, die *explizite Darstellung* gibt eine Konstruktionsvorschrift für diese Punkte an.

Definition 2.1: Implitzite und explizite Darstellung von Linien und Ebenen

Gegeben seien drei paarweise verschiedene Punkte $\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2} \in \mathbf{R^3}$. Dann gilt mit

$$\mathbf{v_1} = \mathbf{p_1} - \mathbf{p_0},\ \mathbf{v_2} = \mathbf{p_2} - \mathbf{p_0} \text{ und } \mathbf{n} = \frac{\mathbf{v_1} \times \mathbf{v_2}}{\|\mathbf{v_1} \times \mathbf{v_2}\|_2}$$

	implizite Darstellung	*explizite Darstellung*
Linie	$L(p_0,p_1) = \{ p \in R^3 : \|v_1 \times (p - p_0)\|_2 = 0 \}$	$l(t) = p_0 + t \cdot v_1 \ (t \in R)$
Ebene	$E(p_0,p_1,p_2) = \{ p \in R^3 : n \cdot (p - p_0) = 0 \}$	$e(s,t) = p_0 + sv_1 + tv_2$ $s, t \in R$

Die Beschränkung des Parameters **t** auf ein Intervall **[a, b]** bei der expliziten Darstellung einer Linie ergibt ein *Liniensegment*. Z.B. entspricht die Beschränkung einer Linie $\mathbf{L(p_0, p_1)}$ auf das Intervall **[0, 1]** dem Liniensegment „zwischen" den Punkten $\mathbf{p_0}$ und $\mathbf{p_1}$.

Eine häufig verwendete Darstellung von Linien ist die *Plücker-Repräsentation* (vgl. [55] und [61]):

Definition 2.2: Plücker-Repräsentation von Linien

Die Plücker-Repräsentation $\tilde{\mathbf{L}}$ einer Linie $\mathbf{L(p_0, p_1)}$ ist gegeben durch

$$\tilde{\mathbf{L}} : \mathbf{R}^3 \times \mathbf{R}^3 \to \mathbf{R}^6 : (\mathbf{p}_0, \mathbf{p}_1) \to \begin{pmatrix} \mathbf{p}_0^x \mathbf{p}_1^y - \mathbf{p}_1^x \mathbf{p}_0^y \\ \mathbf{p}_0^x \mathbf{p}_1^z - \mathbf{p}_1^x \mathbf{p}_0^z \\ \mathbf{p}_0^x - \mathbf{p}_1^x \\ \mathbf{p}_0^y \mathbf{p}_1^z - \mathbf{p}_1^y \mathbf{p}_0^z \\ \mathbf{p}_0^z - \mathbf{p}_1^z \\ \mathbf{p}_1^y - \mathbf{p}_0^y \end{pmatrix}$$

Die Plücker-Repräsentation ist eine Mischung aus dem *Kreuzprodukt* $\mathbf{c = p_0 \times p_1}$ der Ortsvektoren $\mathbf{p_0}$ und $\mathbf{p_1}$ sowie dem *Richtungsvektor* $\mathbf{d = p_1 - p_0}$. Damit kann man $\tilde{\mathbf{L}}$ auch als Tupel $\mathbf{(c_z, -c_y, -d_x, c_x, -d_z, d_y)^T}$ schreiben.

Das *Plücker-Skalarprodukt* erlaubt Aussagen über den *relativen Drehsinn* zweier Linien:

Definition 2.3: Plücker-Skalarprodukt

Das (permutierte) Plücker-Skalarprodukt der Plücker-Repräsentationen $\tilde{\mathbf{K}}$ und $\tilde{\mathbf{L}}$ zweier Linien **K** und **L** ist gegeben durch

$$\cdot : \mathbf{R}^6 \times \mathbf{R}^6 \to \mathbf{R} : (\tilde{\mathbf{K}}, \tilde{\mathbf{L}}) \to \tilde{\mathbf{K}}_1 \tilde{\mathbf{L}}_5 + \tilde{\mathbf{K}}_2 \tilde{\mathbf{L}}_6 + \tilde{\mathbf{K}}_3 \tilde{\mathbf{L}}_4 + \tilde{\mathbf{K}}_4 \tilde{\mathbf{L}}_3 + \tilde{\mathbf{K}}_5 \tilde{\mathbf{L}}_1 + \tilde{\mathbf{K}}_6 \tilde{\mathbf{L}}_2$$

und es gilt in „Blickrichtung" $\mathbf{K_d}$:

$$\tilde{\mathbf{K}} \cdot \tilde{\mathbf{L}} \begin{cases} < 0 & \mathbf{L} \textit{ "dreht sich" gegen den Uhrzeigersinn um } \mathbf{K} \\ = 0 & \mathbf{L} \textit{ und } \mathbf{K} \textit{ schneiden sich oder sind parallel} \\ > 0 & \mathbf{L} \textit{ "dreht sich" mit dem Uhrzeigersinn um } \mathbf{K} \end{cases}$$

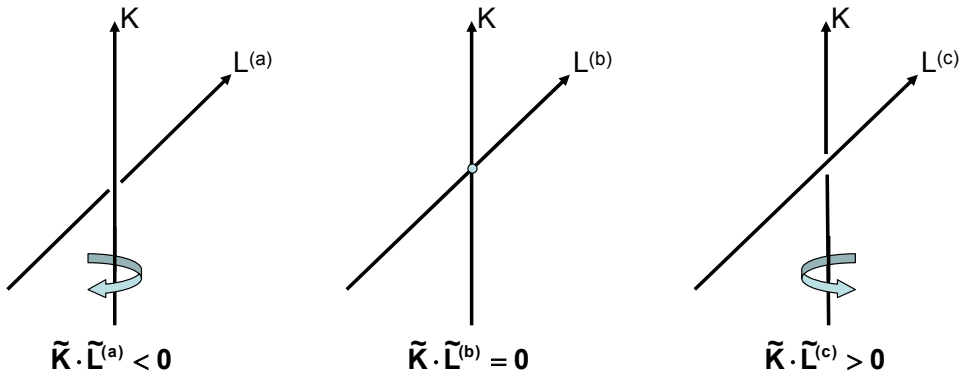


Abb. 2.1
Das Plücker-Skalarprodukt macht Aussagen über den relativen Drehsinn zweier Linien

Dies kann man sich anhand von Abb. 2.1 veranschaulichen: Wenn man sich „auf" der Linie $\mathbf{K}$ befindet und in Richtung von $\mathbf{K}$ schaut, würde sich Linie $\mathbf{L}^{(a)}$ gegen den Uhrzeigersinn, die Linie $\mathbf{L}^{(c)}$ dagegen mit dem Uhrzeigersinn um $\mathbf{K}$ „drehen".

Auch für Ebenen sollen hier noch einige wichtige Beziehungen angegeben werden:

Definition 2.4: Halbraum-Klassifikation

Für eine Ebene $\mathbf{E}$ heißt

$\mathbf{E^- = \{ p \in R^3 \mid E_n \cdot p + E_d \leq 0 \}}$ *negativer Halbraum* und

$\mathbf{E^+ = \{ p \in R^3 \mid E_n \cdot p + E_d > 0 \}}$ *positiver Halbraum* von E.

Anschaulich ist der *negative Halbraum* also der Ausschnitt des $\mathbf{R^3}$ „hinter" der Ebene und analog *der positive Halbraum* der Ausschnitt des $\mathbf{R^3}$ „vor" der Ebene.

2.1.3 Polygone und Polyeder

Polygone und *Polyeder* werden in den folgenden Kapiteln eine große Rolle spielen und sollen daher in ihren wesentlichen Eigenschaften und Beziehungen vorgestellt werden.

Definition 2.5: Polygone

Ein *Polygon* **P** ist durch $n > 2$ koplanare Punkte (oder Vertices) p_0, p_1, ..., p_{n-1} definiert, wobei zwei aufeinander folgende Vertices p_i und $p_{(i+1) \bmod n}$ ein Linien-Segment (oder Kante) K_i definieren.

Mehrere Polygone können adjazent zueinander sein:

Definition 2.6: Adjazenzen von Polygonen

Zwei Polygone **P** und **Q** heißen *adjazent*, wenn sie beide zu einer Kante **K** adjazent sind. Ein Menge **Z** von Polygonen heißt *zusammenhängend*, wenn es für jedes Paar von Polygonen $(P, Q) \in Z \times Z$ gilt, dass eine Folge $p = (P, P_1, P_2, \ldots, P_n, Q)$ von Polygonen existiert, für die P_i adjazent zu P_{i+1} für $0 \leq i \leq n$ ist.

Eine Menge **G** von Polygonen heißt *geschlossen*, wenn es für jede Kante **K** jedes Polygons $P \in G$ ein zu **K** adjazentes Polygon $P \neq Q \in G$ gibt.

Für Polygone mit drei oder vier Vertices existiert das folgende Zerlegungsschema (vgl. auch [42] und [43]):

Definition 2.7: Quadtree-Zerlegungen von Drei- und Vierecken

Die *Quadtree-Zerlegung* eines Drei- oder Vierecks **P** ist eine Zerlegung von **P** in vier kongruente Kind-Drei- bzw. Vierecke nach folgendem Schema:

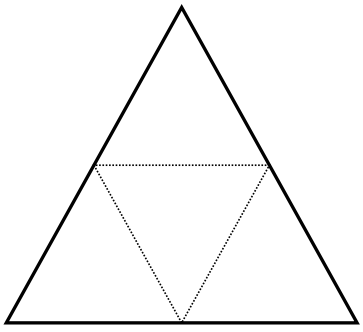


Abb. 2.2
Dreieck-Quadtree

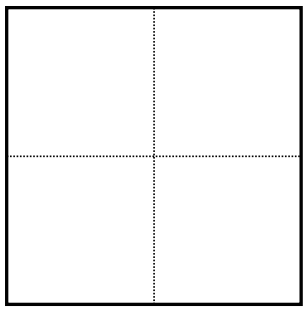


Abb. 2.3
Viereck-Quadtree

Das Verfahren kann rekursiv auf die Kind-Drei- und Vierecke erweitert werden, dadurch entsteht ein Baum von Kind-Polygonen: der Quadtree.

Definition 2.8: Polyeder

Ein *Polyeder* $\mathbf{K}$ ist eine geschlossene Menge von Polygonen $\mathbf{P_0}$, $\mathbf{P_1}$, ..., $\mathbf{P_{n-1}}$. Die Polygone werden als *Facetten* des Polyeders bezeichnet, die Vereinigung der Polygon-Vertices bzw. –Kanten als *Vertices* bzw. *Kanten* des Polyeders.

Ein Polyeder $\mathbf{K}$ heißt *konvex*, wenn für jede Facette $\mathbf{F_i}$ der Polyeder $\mathbf{K}$ im negativen Halbraum von $\mathbf{F_i}$ liegt.

Satz 2.1: Eulerscher Polyedersatz

Für jeden konvexen Polyeder $\mathbf{K}$ mit $\mathbf{f}$ Facetten, $\mathbf{k}$ Kanten und $\mathbf{v}$ Vertices gilt:

$$\mathbf{v - k + f = 2}$$

Beweis: siehe z.B. [8] oder [13]

Dieser lineare Zusammenhang erlaubt es, die Zahl der Facetten, Kanten und Vertices einer Menge von Polyedern durch eine gemeinsame Kennzahl $\mathbf{N}$, z.B. die Zahl der Polyeder, auszudrücken, was im weiteren Verlauf einige Komplexitätsabschätzungen vereinfachen wird.

Ein im Weiteren wichtiger Spezialfall eines Polyeders ist die *Bounding Box*:

Definition 2.9: Bounding Box

Eine *Bounding Box* $\mathbf{B = (p_{min}, p_{max})}$ mit $\mathbf{p_{min}, p_{max} \in R^3}$ ist ein achsenparalleler Quader, wobei $\mathbf{p_{min}}$ den Eckpunkt mit minimalen und $\mathbf{p_{max}}$ den Eckpunkt mit maximalen Koordinaten angibt.

Für einen Polyeder $\mathbf{K}$ heißt die kleinste Bounding Box $\mathbf{B_K}$ mit $\mathbf{K \subset B_K}$ Bounding Box von $\mathbf{K}$. Analog heißt für zwei Bounding Boxen $\mathbf{C}$ und $\mathbf{D}$ die kleinste Bounding Box $\mathbf{B_{CD}}$ mit $\mathbf{C, D \subset B_{CD}}$ Bounding Box von $\mathbf{C}$ und $\mathbf{D}$.

Als Kurzschreibweise wird in dieser Arbeit statt Bounding Box der Begriff *B-Box* verwendet.

2.2 Schnittberechnungen und Klassifikationen

Schnittberechnungen zwischen geometrischen Primitiven sind grundlegend für die Beschäftigung mit geometrischen Problemen. Eine Verallgemeinerung von Schnittberechnungen sind die *Klassifikationen anhand von geometrischen Primitiven*. Eine Klassifikation sagt aus, wie zwei geometrische Primitive zueinander „stehen", z.B. ob ein Polyeder eine Ebene schneidet oder ob er vollständig „über" oder „unter" der Ebene liegt.

Besonderes Augenmerk liegt dabei auf den Klassifikationen von B-Boxen anhand von Linien und Ebenen. Hierfür werden jeweils vereinfachte Klassifikationsschemata angegeben.

Zunächst soll aber der Begriff Klassifikation genauer gefasst werden, da er für alle folgenden Betrachtungen eine große Rolle spielt:

Definition 2.10: Klassifikationen von geometrischen Primitiven

Für zwei geometrische Primitive **A** und **B** ist eine *Klassifikation* eine Abbildung

k: (A, B) → { OUT, INS, OCC, PAR },

die die relative Lage von **B** bezüglich **A** beschreibt:

Klassifikation	*Erklärung*	*Beispiel*
OUT *(„außerhalb")*	**B** liegt vollständig außerhalb von **A**	Wenn zwei Polyeder **A** und **B** sich nicht schneiden, liegt **A** außerhalb von **B**
INS *(„innerhalb")*	**B** liegt vollständig innerhalb von **A**	Wenn eine B-Box **A** einen Polyeder **B** umfasst, so liegt **B** „innerhalb" von **A**
OCC *(„verdeckend")*	**B** verdeckt **A** voll-ständig	Wenn eine Linie **B** einen Polyeder **A** schneidet, wird **B** von **A** verdeckt
PAR *(„partiell")*	**B** liegt teilweise außer- und teilweise innerhalb von **A**	Wenn sich zwei Polyeder **A** und **B** schneiden, so liegt **A** teilweise außer-/ innerhalb von **B**

Tabelle 2.1

Im Folgenden werden die Klassifikationen **OUT** und **INS** oft durch im Kontext anschaulichere Begriffe **POS** und **NEG** ersetzt. Dies bedeutet z.B. für Ebenen **A** und geometrische Primitive **B**, dass **B** vollständig im positiven (**POS**) oder negativen (**NEG**) Halbraum der Ebene **A** liegt.

2.2.1 Klassifikationen anhand von Linien

Bei dem in Definition 2.3 betrachteten definierten Plücker-Skalarprodukt handelt es sich bereits um eine *Klassifikation einer Linie anhand einer zweiten Linie* (vgl. [25]):

Definition 2.11: Klassifikation von Linien anhand von Linien

Für zwei Linien $\mathbf{K}$ und $\mathbf{L}$ ist die Linie-Linie-Klassifikation definiert als

$$k(K,L) = \begin{cases} \tilde{K} \cdot \tilde{L} < 0 & \text{NEG} \\ \tilde{K} \cdot \tilde{L} = 0 & \text{PAR} \\ \tilde{K} \cdot \tilde{L} > 0 & \text{POS} \end{cases}$$

Dies kann zunächst nicht auf Polygone oder Polyeder verallgemeinert werden: die Aussage, dass sich z.B. ein Polyeder gegen den Uhrzeigersinn um eine Linie „dreht", macht keinen Sinn.

Eine nahe liegende Methode zur Berechnung eines *Linie-Polygon-Schnitts* ist, zunächst den Schnittpunkt $\mathbf{p}$ der Linie $\mathbf{L}$ mit der Ebene des Polygons $\mathbf{P}$ zu berechnen und anschließend zu testen, ob dieser Schnittpunkt in $\mathbf{P}$ liegt. Eine elegantere Möglichkeit ist die Verwendung der *Plücker-Repräsentationen* der Linie $\mathbf{L}$ und der Kanten $\mathbf{K_i}$ des Polygons (vgl. [56]).

Satz 2.2: Linie-Polygon-Schnitte

Eine Linie $\mathbf{L}$ schneidet ein konvexes Polygon $\mathbf{P}$ mit Kanten $\mathbf{K_i}$ genau dann, wenn

$$L_d \cdot P_n \neq 0 \text{ und } \forall (K_i, K_j) : \left(\tilde{L} \cdot \tilde{K}_i\right) \cdot \left(\tilde{L} \cdot \tilde{K}_j\right) \geq 0$$

gilt

Beweis:

Gilt $L_d \cdot P_n = 0$, so sind Linie und Polygon-Ebene parallel und es gibt keinen bzw. unendlich viele Schnittpunkte. Wie man in Abb. 2.4 erkennt, haben im Falle eines Linie-Polygon-Schnitts alle Kanten $\mathbf{K_i}$ bezüglich $\mathbf{L}$ denselben relativen Drehsinn, d.h. das Plücker-Skalarprodukt $\tilde{L} \cdot \tilde{K}_i$ hat immer positives oder immer negatives Vorzeichen. Schneidet $\mathbf{L}$ eine Kante oder einen Vertex von $\mathbf{P}$, so ist $\tilde{L} \cdot \tilde{K}_i = 0$.

Verfehlt $\mathbf{L}$ dagegen das Polygon $\mathbf{P}$ (vgl. Abb. 2.5), so muss es mindestens zwei Kanten $\mathbf{K_i}$ und $\mathbf{K_j}$ mit unterschiedlichem relativem Drehsinn bezüglich $\mathbf{L}$ geben.

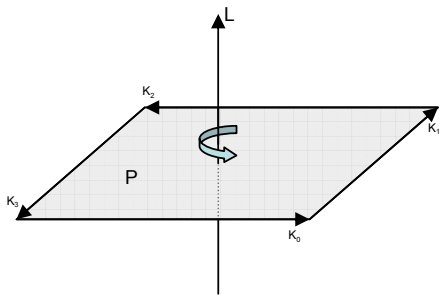


Abb. 2.4
Alle Kanten haben denselben relativen Drehsinn bzgl. **L**

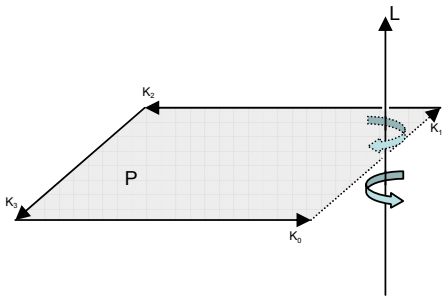


Abb. 2.5
Mindestens zwei Kanten haben unterschiedlichen Drehsinn bzgl. **L**

Man beachte, dass mit diesem Ansatz lediglich das Vorkommen eines Linie-Polygon-Schnitts erkannt wird, der konkrete Schnittpunkt muss dann mit der Schnittgleichung für Linie-Ebenen-Schnitte berechnet werden. Als Klassifikation ergibt sich somit:

$$\mathbf{k(L,F)} = \begin{cases} \mathbf{OCC} & \left(\forall \mathbf{K_i} \in \mathbf{P} : \tilde{\mathbf{L}} \cdot \tilde{\mathbf{K}}_\mathbf{i} \le \mathbf{0}\right) \vee \left(\forall \mathbf{K_i} \in \mathbf{P} : \tilde{\mathbf{L}} \cdot \tilde{\mathbf{K}}_\mathbf{i} \ge \mathbf{0}\right) \\ \mathbf{OUT} & \mathbf{sonst} \end{cases}$$

Da ein Polyeder nach Kap. 2.1.3 einfach eine zusammenhängende Menge von Polygonen ist, kann damit ein einfaches Verfahren zur Erkennung eines Linie-Polyeder-Schnitts formuliert werden: eine Linie **L** schneidet einen Polyeder **K**, wenn **L** mindestens eine Facette **F** von **K** schneidet. Als Klassifikation eines Polyeders **K** anhand einer Linie **L** ergibt sich so:

$$\mathbf{k(L,K)} = \begin{cases} \mathbf{OCC} & \exists \mathbf{F} \in \mathbf{K} : \mathbf{k(F,L)} = \mathbf{OCC} \\ \mathbf{OUT} & \mathbf{sonst} \end{cases}$$

Oft ist man bei der Berechnung eines Linie-Polyeder-Schnitts nur an einem Schnittpunkt interessiert: dem Eintrittspunkt, d.h. dem ersten Schnittpunkt mit dem Polyeder entlang der Linie. Dazu könnte man natürlich alle Schnittpunkte mit dem Polyeder berechnen und daraus dann den nächstliegenden auswählen.

Eine elegantere Möglichkeit bietet die Einteilung der Polyeder-Facetten in *Front-* und *Rückseiten-Facetten* bezüglich einer Linie (vgl. Abb. 2.6):

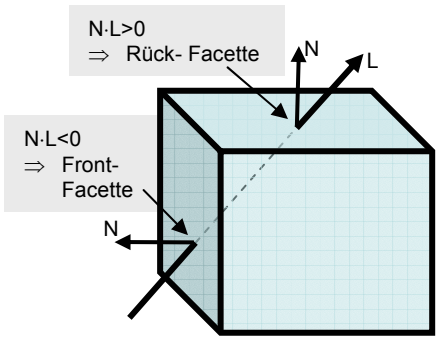


Abb. 2.6
Front- und Rückseiten-Facetten bzgl. einer Linie

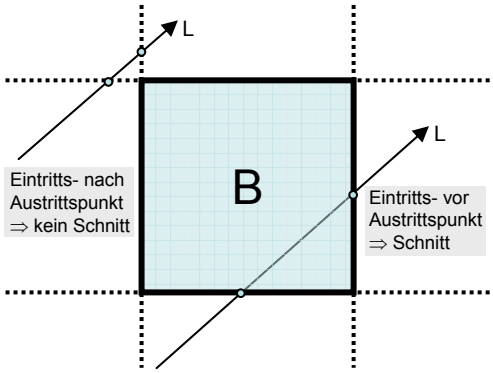


Abb. 2.7
Schnitt von Linie und B-Box

Definition 2.12: Front- und Rück-Facetten

Für einen Polyeder **K** und eine Linie **L** heißen die Facetten $\mathbf{F^i}$ von **K** *Front-Facetten* bezüglich **L**, wenn $\mathbf{L_d \cdot F_n^i < 0}$ und *Rück-Facetten*, wenn $\mathbf{L_d \cdot F_n^i \geq 0}$ gilt.

Um also den nächstliegenden Schnittpunkt einer Linie mit einem Polyeder zu finden, genügt es die Front-Facetten des Polyeders bezüglich der Linie zu betrachten.

Der Schnitt einer Linie **L** mit einer B-Box $\mathbf{B = (p_{min}, p_{max})}$ kann wesentlich einfacher berechnet werden (vgl. [60]): Die Facetten von **B** definieren sechs achsenparallele Ebenen, die Schnitte von **L** mit diesen Ebenen ergeben also sechs potentielle Ein- und Austrittspunkte. Im Fall eines Schnitts müssen alle Eintrittspunkte bezüglich aller Koordinatenachsen vor den Austrittspunkten liegen (vgl. Abb. 2.7). Gilt für eine Kombination von Ein- und Austrittspunkten, dass der Austrittspunkt vor dem Eintrittspunkt liegt, verfehlt die Linie die Box.

Eine Verallgemeinerung von Linie-Polyeder-Schnitten bzw. –Klassifikationen ist die Strahlanfrage an eine Menge von Polyedern (vgl. [3]:

Definition 2.13: Strahlanfrage an Mengen von Polyedern

Für eine Linie **L** und Menge **M** von Polyedern liefert die Strahlanfrage

$$\mathbf{R(L, M) = \{ P \in M: k(L, P) = OCC \}}$$

die Menge der Polyeder $\mathbf{P \in M}$, die **L** schneiden, zurück.

Mit dieser Strahlanfrage wird die gesamte Linie betrachtet. Für die weiteren Betrachtungen ist es sinnvoll, zusätzlich die folgenden Strahlanfragen zu unterscheiden:

Definition 2.14: Beschränkung von Strahlanfragen

Für eine Linie **L** mit Aufpunkten **a** und **b** und eine Menge **M** von Polyedern liefert…

- die untere Strahlanfrage $\mathbf{R_a(L,M)}$ die Elemente von **R(L, M)** zurück, die entlang der Linie „vor" dem Aufpunkt **a** liegen.
- die obere Strahlanfrage $\mathbf{R^b(L,M)}$ die Elemente von **R(L, M)** zurück, die entlang der Linie „hinter" dem Aufpunkt **b** liegen.
- die innere Strahlanfrage $\mathbf{R_a^b(L,M)}$ die Elemente von **R(L, M)** zurück, die „zwischen" den Aufpunkten **a** und **b** liegen.

In Abb. 2.8 sind für eine 2D-Linie **L** und eine Menge von 2D-Polyedern (d.h. Polygonen) die Ergebnisse der unteren, oberen und inneren Strahlanfragen dargestellt.

Diese Strahlanfragen liefern jeweils eine Menge von Polyedern zurück. Bei den Anwendungen der Strahlanfragen in Kap. 3 und 4 werden jedoch nur Teile dieser Information benötigt. Dies wird in den folgenden Definitionen konkretisiert:

Definition 2.15: Innere Verdeckungs-Strahlanfrage an Mengen von Polyedern

Die innere Verdeckungs-Strahlanfrage an eine Menge von Polyedern ist definiert als

$$\mathbf{r_a^b(L,M)} = \begin{cases} \mathbf{OUT} & \mathbf{R_a^b(L,M)} = \varnothing \\ \mathbf{OCC} & \mathbf{sonst} \end{cases}$$

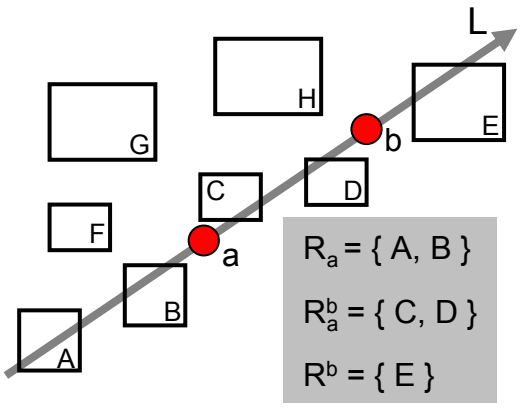


Abb. 2.8
Ergebnis der unteren, oberen und inneren Strahlanfragen

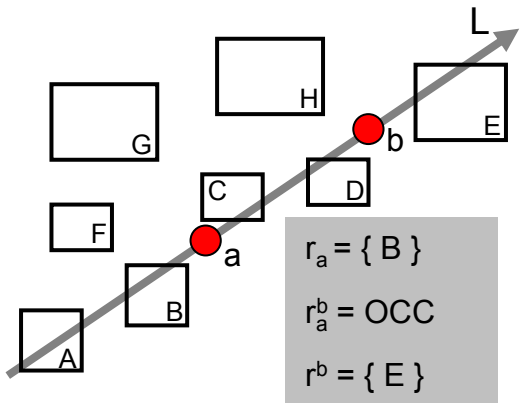


Abb. 2.9
Ergebnis der unteren und oberen Extremal-Strahlanfrage und der inneren Verdeckungs-Strahlanfrage

Definition 2.16: Untere und obere Extremal-Strahlanfrage an Mengen von Polyedern

Die untere (bzw. obere) Extremal-Strahlanfrage $\mathbf{r_a(L, M)}$ (bzw. $\mathbf{r^b(L, M)}$) liefert den Polyeder aus der Menge $\mathbf{R_a(L, M)}$ (bzw. $\mathbf{R^b(L, M)}$) zurück, für den der Schnittpunkt mit **L** den kleinsten Abstand zum Aufpunkt **a** (bzw. **b**) hat.

Die Verdeckungs-Strahlanfrage gibt also an, ob die Linie zwischen den Aufpunkten **a** und **b** geschnitten wird und die untere bzw. obere Extremal-Strahlanfrage liefert den Polyeder zurück, der als nächstes „vor" bzw. „hinter" den Aufpunkten geschnitten wird (vgl. Abb. 2.9).

Damit ist die Betrachtung der Klassifikationen anhand von Linien zunächst abgeschlossen. Die performante Umsetzung der in diesem Teilkapitel definierten Strahlanfragen wird in Kapitel 3.4.6 ausführlich diskutiert.

2.2.2 Klassifikationen anhand von Ebenen

Die Klassifikation anhand einer Ebene entspricht der Frage, „auf welcher Seite" der Ebene ein Punkt (oder Linie, Polygon, Polyeder) liegt. Die folgenden Ausführungen beziehen sich nur auf die Klassifikation von Polyedern anhand von Ebenen, die anderen Klassifikationen können direkt aus diesen Betrachtungen abgeleitet werden.

Definition 2.17: Klassifikation eines Polyeders anhand einer Ebene

Für eine Ebene **E** und einen Polyeder **K** ist die Ebene-Polyeder-Klassifikation definiert als:

$$\mathbf{side(E,K)} = \begin{cases} \mathbf{NEG} & \mathbf{K} \in \mathbf{E}^- \\ \mathbf{POS} & \mathbf{K} \in \mathbf{E}^+ \\ \mathbf{PAR} & \mathbf{sonst} \end{cases}$$

Ein einfacher Ansatz zur Berechnung von **side(E, K)** ist die Klassifikation aller Vertices von **K** anhand **E**: Liegen alle Vertices im negativen (positiven) Halbraum von **E**, liegt auch **K** vollständig im negativen (positiven) Halbraum von **E**, sonst schneidet die Ebene **E** den Polyeder **K**.

In [29] wird eine Optimierung für die Klassifikation einer B-Box anhand einer Ebene vorgeschlagen: Statt alle acht Eckpunkte des Quaders zu klassifizieren, reichen hier nur zwei Eckpunkte aus: der Eckpunkt $\mathbf{v}^-$, der am weitesten im negativen und der Eckpunkt $\mathbf{v}^+$, der am weitesten im positiven Halbraum von **E** liegt (vgl. Abb. 2.10):

$$\mathbf{v}^- = \mathbf{min}_{\mathbf{v}\,Vertex\,von\,\mathbf{B}}\, \mathbf{d}_{\mathbf{E}}^{\pm}(\mathbf{v}) \text{ und } \mathbf{v}^+ = \mathbf{max}_{\mathbf{v}\,Vertex\,von\,\mathbf{B}}\, \mathbf{d}_{\mathbf{E}}^{\pm}(\mathbf{v})$$

Wenn $\mathbf{v}^-$ im positiven (bzw. $\mathbf{v}^+$ im negativen) Halbraum von **E** liegt, dann liegt der Quader vollständig im positiven (bzw. negativen) Halbraum von **E**, sonst schneidet **E** den Quader.

Die Vertices $\mathbf{v}^-$ bzw. $\mathbf{v}^+$ können direkt aus den Komponenten des Normalenvektors $\mathbf{E_n}$ bestimmt werden:

$$\mathbf{v}_i^- = \begin{cases} \mathbf{p}_{max}^i & \mathbf{E}_n^i \le 0 \\ \mathbf{p}_{min}^i & \mathbf{E}_n^i > 0 \end{cases} \quad \text{und} \quad \mathbf{v}_i^+ = \begin{cases} \mathbf{p}_{min}^i & \mathbf{E}_n^i \le 0 \\ \mathbf{p}_{max}^i & \mathbf{E}_n^i > 0 \end{cases}$$

Dies sei anschaulich für die **x**- und **y**-Komponente von $\mathbf{v}^-$ begründet (vgl. Abb. 2.11):

Da der Normalenvektor von **E** eine negative **x**-Komponente hat, muss der Vertex $\mathbf{v}^-$, der am weitesten im negativen Halbraum liegt, ein Vertex mit maximaler **x**-Komponente $\mathbf{p}_{max}^x$ sein. Analog dazu muss $\mathbf{v}^-$ der Vertex mit minimaler **y**-Komponente sein, da der Normalenvektor von **E** eine positive **y**-Komponente hat.

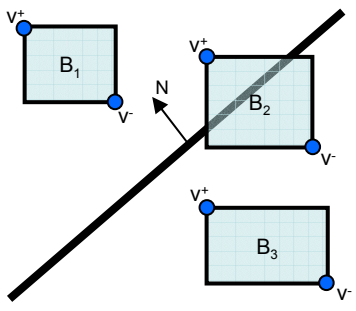


Abb. 2.10
Klassifikation anhand der Vertices $\mathbf{v}^-$ und $\mathbf{v}^+$

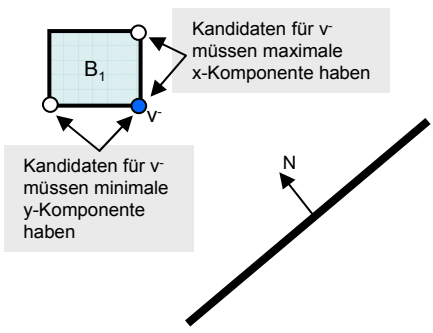


Abb. 2.11
Wahl der **x**-Komponente für $\mathbf{v}^-$ und $\mathbf{v}^+$

2.3 3D-Entitäten und 3D-Szenen

Die intuitive Vorstellung der Begriffe *3D-Objekt* und *3D-Szene* soll in diesem Kapitel formalisiert werden. Insbesondere sind dabei die Eigenschaften und möglichen Einteilungen von 3D-Szenen interessant. Abschließend werden noch zwei mögliche Datenstrukturen zur Speicherung von 3D-Szenen diskutiert.

2.3.1 3D-Entitäten: 3D-Objekte und Elemente

Unter dem Begriff *3D-Objekt* werden in der Fachliteratur (vgl. z.B. [X]) eine große Zahl von unterschiedlichen Entitäten im $\mathbf{R}^3$ zusammengefasst: Punkte, Linien, Ebenen, Polygone, *Volumenmodelle*, usw.

In dieser Arbeit wird dagegen nur eine kleine Teilmenge möglicher 3D-Objekte betrachtet, die so genannten *polygonalen 3D-Objekte* oder einfach *Polyeder*. Wenn also im Folgenden von 3D-Objekten die Rede ist, sind stets Polyeder gemeint.

Definition 2.18: 3D-Entitäten, 3D-Objekte und Elemente

Ein *polygonales 3D-Objekt* **O** ist ein Polyeder. Die Facetten, Kanten und Vertices von **O** werden *Elemente* genannt. Ein 3D-Objekt heißt *vernetzt*, wenn jedes seiner Elemente Referenzen auf alle benachbarten Elemente und auf das 3D-Objekt, welches das Element enthält, speichert.

Der Oberbegriff für 3D-Objekte und Elemente ist *3D-Entität*.

Die folgende Tabelle fasst die Adjazenzen von Elementen eines 3D-Objekts zusammen, z.B. gibt die erste Beschreibungszelle oben links an, wann ein Vertex **v'** adjazent zu einem Vertex **v** ist. Da die Tabelle symmetrisch ist, sind nur die Zellen des rechten oberen Dreiecks angegeben.

	Vertex **v**	Kante **k**	Facette **f**
Vertex **v'**	**v'** ist über eine Kante **k** von **v** aus erreichbar	**v'** ist Start- oder End-Vertex von **k**	**v'** ist ein Vertex des Polygons **f**
Kante **k'**		**k** und **k'** haben einen gemeinsamen Vertex	**k'** ist eine Kante des Polygons **f**
Facette **f'**			**f** und **f'** haben eine gemeinsame Kante

Tabelle 2.2

Eine mögliche Klassenhierarchie für 3D-Entitäten, 3D-Objekte und Elemente ist die folgende:

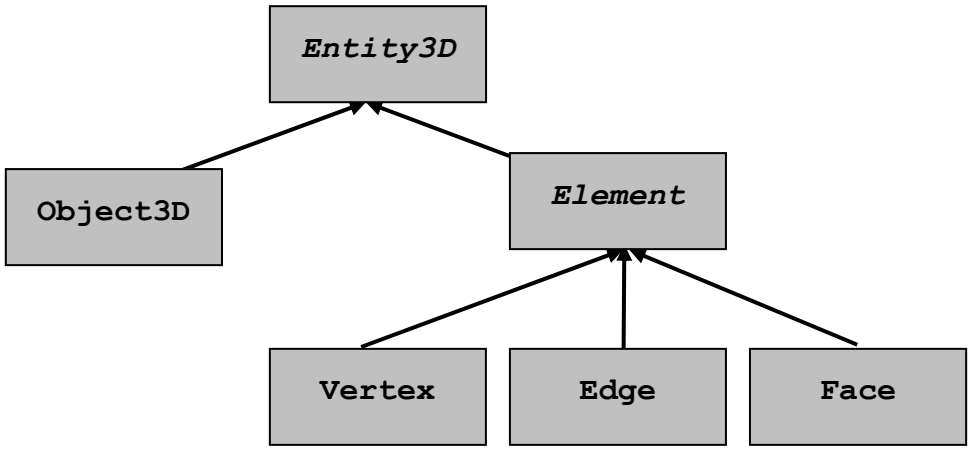


Abb. 2.12

Klassendiagramm für 3D-Entitäten, 3D-Objekte und Elementen

Die abstrakte Klasse **`Entity3D`** fasst alle *3D-Entitäten* zusammen, die dadurch gekennzeichnet sind, dass sie benachbarte Elemente haben bzw. Elemente enthalten:

```
abstract class Entity3D
    int         id;         // Typweise eindeutige ID
    Vertex[]    vert;       // Liste adjazenter bzw. enthaltener Vertices
    Edge[]      edge;       // Liste adjazenter bzw. enthaltener Kanten
    Face[]      face;       // Liste adjazenter bzw. enthaltener Facetten
```

Code-Beispiel 2.1

Die Klasse **`Object3D`** erweitert **`Entity3D`** lediglich um ein Feld **`box`**, das einen Verweis auf die Bounding-Box (*B-Box*) des Objekts enthält:

```
class Object3D extends Entity3D
     BBox           box;
```

Code-Beispiel 2.2

Die abstrakte Klasse **`Element`** erweitert **`Entity3D`** um einen Verweis auf das übergeordnete 3D-Objekt, d.h. das 3D-Object, welches das Element enthält:

```
abstract class Element extends Entity3D
     Object3D       parObj;      // Verweis auf übergeordnetes Objekt
```

Code-Beispiel 2.3

Die konkreten Element-Typen erweitern dann jeweils die Klasse **`Element`** wie folgt:

```
class Vertex extends Element
     Point3d        p;          // Punkt, der den Vertex definiert

class Edge extends Element
     Line           line;       // durch Kante definiertes Liniensegment

class Face extends Element
     Plane          plane;      // Ebene des Polygons
     Material       material;   // Materialeigenschaften
```

Code-Beispiel 2.4

Materialeigenschaften (Datenfeld **`material`** der Klasse **`Face`**) beschreiben die Fähigkeiten eines 3D-Objekts, Lichtenergie zu *reflektieren*, *transmittieren* oder zu *emittieren*. Anschaulich kann man sich dies auch als „Farbe“ des Objekts vorstellen: ein 3D-Objekt, das rotes Licht emittiert oder nur den Rot-Anteil einfallenden Lichts reflektiert oder transmittiert, wirkt auf den Betrachter rot.

Auf den ersten Blick wäre es also sinnvoll, dieses Datenfeld der Klasse **`Object3D`** zuzuordnen. Dies würde jedoch bedeuten, dass z.B. alle Facetten eines Objekts nur rotes Licht reflektieren oder dass alle Facetten Licht emittieren, d.h. Lichtquellen sind. Dies wäre eine unnötige Einschränkung möglicher 3D-Objekte, daher wird das Datenfeld in die Klasse **`Face`** übernommen.

2.3.2 3D-Szenen

Die Definition einer 3D-Szene ist mit den bisherigen Definitionen offensichtlich:

Definition 2.19: 3D-Szene

Eine 3D-Szene **S** ist eine endliche, nicht-leere Menge von 3D-Objekten $\mathbf{O_i}$.

Die Eigenschaften einer 3D-Szene **S** werden durch die der enthaltenen 3D-Objekte und deren Beziehungen bestimmt. Die folgende Tabelle enthält in der ersten Spalte eine Eigenschaft von 3D-Szenen, in der zweiten Spalte eine Beschreibung und in der dritten Spalte die Begründung, warum diese Eigenschaft im Weiteren vorausgesetzt wird:

Eigenschaft	*Beschreibung*	*Begründung*
polygonal	$\forall \mathbf{O_i} \in \mathbf{S}$: $\mathbf{O_i}$ *polygonal*	Alle Verfahren in Kap. 3 und 4 gehen von polygonalen 3D-Objekten aus.
konvex	$\forall \mathbf{O_i} \in \mathbf{S}$: $\mathbf{O_i}$ *konvex*	Jedes nicht-konvexe Objekt $\mathbf{O_i}$ kann in konvexe Teilobjekte zerlegt werden.
geschlossen[2]	$\exists\, \mathbf{R} \in \mathbf{S}: \forall \mathbf{O_i} \in \mathbf{S}: \mathbf{O_i} \subset \mathbf{R}$	Radiosity-Beleuchtungsmodell (Kap. 2.4) setzt geschlossene Szenen voraus.
schnittfrei	$\forall \mathbf{O_i}, \mathbf{O_j} \in \mathbf{S}$ mit $\mathbf{O_i}, \mathbf{O_j} \neq \mathbf{R}$: $\mathbf{O_i} \cap \mathbf{O_j} = \varnothing$	Jede nicht-schnittfreie 3D-Szene kann in eine schnittfreie 3D-Szene überführt werden.
beleuchtet	$\exists \mathbf{O_i} \in \mathbf{S}$: $\mathbf{O_i}$ *Lichtquelle*	Ohne Lichtquellen ist die Beschäftigung mit Beleuchtungsmodellen sinnlos.

Tabelle 2.3

Der Zugriff auf die in einer 3D-Szene enthaltenen 3D-Objekte $\mathbf{O_i}$ erfolgt über eine Liste. Für die weiteren Betrachtungen empfiehlt sich der folgende Aufbau der Liste:

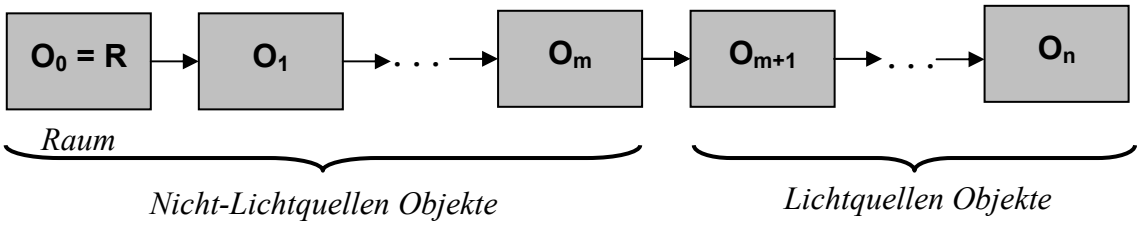


Abb. 2.13
Aufbau der Objektliste einer 3D-Szene

Die Liste ist in drei Abschnitte aufgeteilt: Zuerst kommt das umgebende Raumobjekt[3], dann alle anderen nicht-Lichtquellen-Objekte und abschließend alle Lichtquellen-Objekte. Die Liste ist insgesamt nach aufsteigenden Objekt-IDs geordnet, diese aufsteigende Ordnung gilt auch Typweise für die IDs der Elemente.

Oft ist es nützlich, wenn die verwendete Zugriffsstruktur in irgendeiner Form die *räumliche Anordnung* der 3D-Objekte wiedergibt. Dies kann durch eine Erweiterung der Objektliste zu einem Objektbaum, auch *Bounding-Box-Hierarchie* (kurz: *BB-Hierarchie*) oder *R-Baum* genannt, erreicht werden:

[2] **R** wird auch *Raum* genannt und entspricht der intuitiven Vorstellung: der Raum umgibt die Szene.
[3] Im Folgenden wird vorausgesetzt, dass das Raumobjekt **R** keine Lichtquelle ist.

Jedes 3D-Objekt enthält einen Verweis auf seine umgebende B-Box. Nun kann man Paare von B-Boxen, die „nahe“ beieinander liegen, zu neuen B-Boxen zusammenfassen und diesen Vorgang rekursiv wiederholen, bis nur noch eine einzige B-Box – die Wurzel des Objekt-Baums – übrig bleibt.

Für den Aufbau und die Bewertung der „Qualität“ solcher R-Bäume gibt es sehr viele Verfahren, deren Beschreibung hier jedoch den Rahmen sprengen würde. Es sei daher nur bemerkt, dass der hier verwendete Konstruktionsalgorithmus aus [7] versucht, das Volumen der einzelnen B-Boxen und das Volumen des Schnitts zweier B-Boxen zu minimieren.

Weiterhin wird analog zur Objektliste gefordert, dass die 3D-Objekte und deren Elemente in den Blättern des Objektbaums „von links nach rechts“ nach aufsteigenden IDs geordnet sind und dass zuerst das Raumobjekt, dann die nicht-Lichtquellen-Objekte und abschließend die Lichtquellen-Objekte kommen. Man beachte, dass durch diese Forderung die Struktur des Objektbaums die Vergabe der Objekt- und Element-IDs induziert.

Abb. 2.14 zeigt eine einfache Szene im 2D mit einem Raum, vier Nicht-Lichtquellen-Objekten und zwei Lichtquellen. Dazu sind die B-Boxen sowie die darüber liegende Baumstruktur gezeigt. In Abb. 2.15 ist der dazu gehörende Objektbaum dargestellt.

Man erkennt, dass der Objektbaum durch die zusätzlichen Forderungen in den ersten beiden Ebenen eine feste Struktur hat: der linke Sohn **R** der Wurzel ist das Raumobjekt, der rechte Sohn **OBJ** enthält als linken Sohn den Teilbaum der nicht-Lichtquellen-Objekte und als rechten Sohn den Teilbaum der Lichtquellen-Objekte.

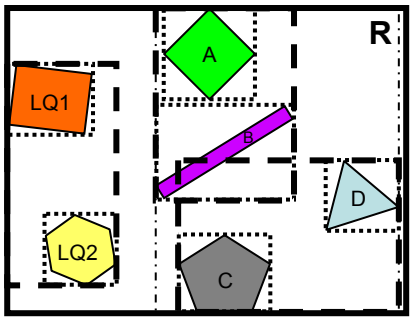


Abb. 2.14
Beispiel für eine 2D-Szene

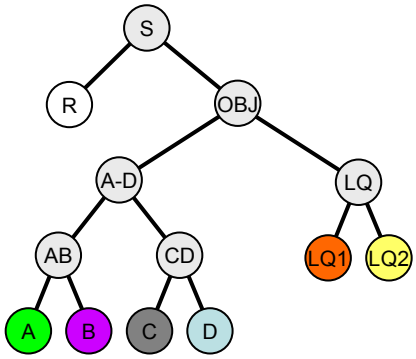


Abb. 2.15
Objektbaum dieser Szene

Aufgrund der zentralen Bedeutung der BB-Hierarchien für den Rest der Arbeit sollen hier die wesentlichen Eigenschaften und Aussagen zusammengestellt werden:

- eine BB-Hierarchie ist ein Binärbaum:
  - die Knoten der BB-Hierarchie sind B-Boxen
  - die inneren Knoten der BB-Hierarchie enthalten Verweise auf zwei (Kind-) B-Boxen
  - die Blätter der BB-Hierarchie enthalten Verweise auf 3D-Objekte
  - die Wurzel der BB-Hierarchie ist eine B-Box, die die gesamte 3D-Szene umschließt
- die in den Blättern der BB-Hierarchie gespeicherten 3D-Objekte und Elemente sind mit IDs versehen. Diese IDs sind in den Blättern der Baumdarstellung „von links nach rechts“ aufsteigend geordnet.
- Eine Pre-Order-Traversierung der BB-Hierarchie liefert die Blätter in aufsteigender Reihenfolge, d.h. in der Baumdarstellung „von links nach rechts“.
- Viele Klassifikationen von B-Boxen anhand von Ebenen oder Linien übertragen sich transitiv auf die enthaltenen Kind-Boxen bzw. 3D-Objekte. Wenn z.B. eine B-Box vollständig „über“ einer Ebenen liegt, gilt dies auch für alle enthaltenen Kind-Boxen bzw. 3D-Objekte.

Mit den obigen Punkten sowie den Überlegungen aus Kap. 2.2 ergibt sich so folgende Motivation für die Verwendung der BB-Hierarchie statt der einfachen Objekt-Liste:

In Kap. 2.2.1 und 2.2.2 wurde gezeigt, dass viele Klassifikationen für B-Boxen einfacher zu berechnen sind als für allgemeine 3D-Entitäten. Einige (nicht alle) Klassifikationen einer B-Box übertragen sich transitiv auf die enthaltenen Kind-Boxen bzw. 3D-Objekte.

Um also z.B. alle 3D-Objekte einer 3D-Szene anhand einer Ebene **E** zu klassifizieren, kann man die BB-Hierarchie in Pre-Order-Reihenfolge traversieren und jeden Knoten (d.h. jede B-Box) anhand der Ebene **E** klassifizieren. Liegt der Knoten vollständig „unter“ bzw. „über“ der Ebene **E**, so liegen auch alle in diesem Knoten enthaltenen Kind-Knoten und 3D-Objekte „unter“ bzw. „über“ der Ebene und müssen nicht mehr einzeln klassifiziert werden (vgl. auch [51] und [62]).

2.4 Das Radiosity-Beleuchtungsmodell

Um die Lichtausbreitung in einer 3D-Szene algorithmisch zu beschreiben, benötigt man globale Beleuchtungsmodelle. Diese beschreiben, wie Licht, das auf einer Oberfläche auftrifft, von ihr reflektiert, transmittiert und absorbiert wird.

Es gibt sehr viele verschiedene globale Beleuchtungsmodelle, die unterschiedliche Aspekte der Lichtausbreitung modellieren. Insbesondere unterscheidet man dabei spiegelnde und diffuse Lichtausbreitung. Der Unterschied zwischen diesen beiden Ausbreitungsarten soll anhand von zwei Beispielen veranschaulicht werden:

In Abb. 2.16 ist spiegelnde Lichtausbreitung dargestellt: ein Lichtstrahl **S** trifft auf die Fläche **F** auf und wird nach der einfachen Regel „Einfallswinkel = Ausfallswinkel“ von **F** als einzelner Lichtstrahl **S'** gespiegelt. Abb. 2.17 zeigt dieselbe Ausgangssituation bei diffuser Lichtausbreitung: der Lichtstrahl **S** trifft auf die Fläche **F** und wird von dort als Strahlbündel gleichmäßig in alle Richtungen reflektiert.

Im Rahmen dieser Arbeit wird nur die diffuse Lichtausbreitung betrachtet, die vom *Radiosity-Beleuchtungsmodell* beschrieben wird. Erste Arbeiten hierzu stammen von Goral, Greenberg und Wallace (vgl. [27], [11], [9] und [10]). Für weitere Informationen sei [4], [45] und [47] empfohlen.

In [27] wird Radiosity wie folgt definiert:

Definition 2.20: Radiosity

Radiosity ist das Verhältnis von abgestrahlter Energie (oder Fluss) **dE** zur Größe des abstrahlenden Flächenstücks **dA**: $\mathbf{R} = \frac{\mathbf{dE}}{\mathbf{dA}}$

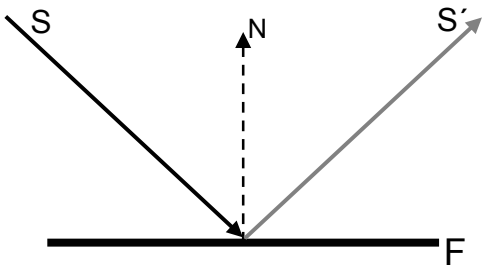


Abb. 2.16
Spiegelnde Lichtausbreitung

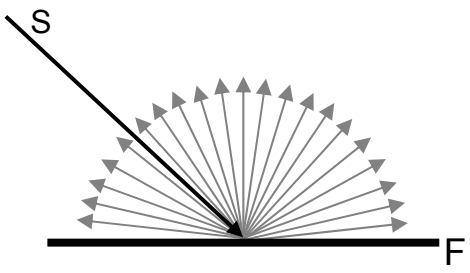


Abb. 2.17
Diffuse Lichtausbreitung

Bereits aus dieser Definition ist die Grundidee des Radiosity-Beleuchtungsmodells klar: Die physikalische Größe Radiosity beschreibt keinen statischen Zustand, sondern einen Energiefluss. Ist der Energiefluss zwischen allen Oberflächen einer Szene bekannt, kennt man auch die zu einem beliebigen Zeitpunkt auf einem beliebig kleinen Flächenstück (auch *Patch* genannt) konzentrierte Energie, also die diffuse Beleuchtung der Szene.

An dieser Stelle soll nun keine vollständige Beschreibung des Radiosity-Beleuchtungsmodells folgen, sondern lediglich eine Veranschaulichung der für den Rest der Arbeit wesentlichen Eigenschaften.

Vorraussetzung für eine sinnvolle Anwendung des Radiosity-Beleuchtungsmodells ist eine Zerlegung der 3D-Szene bzw. der enthaltenen 3D-Objekte in genügend kleine Patches. Betrachtet man z.B. die gesamte Decke eines Raums als ein Patch, so liefert das Radiosity-Beleuchtungsmodell für dieses Patch, also die gesamte Decke, nur einen diffusen Energiewert. Informationen wie Schatten- oder Helligkeitsverläufe entlang der Decke gehen verloren.

Ein einfacher Ansatz für eine genauere Lösung ist, alle Oberflächen der Szene in „sehr kleine“ Patches aufzuteilen und somit auch eine „sehr genaue“ Abbildung von Helligkeits- und Schattenverläufen zu garantieren. Der offensichtliche Nachteil dieses Ansatzes ist, dass zur Berechnung des Energieflusses zwischen allen $\mathbf{n}$ Patches ein quadratischer Aufwand notwendig ist, da potentiell zwischen allen $\mathbf{n^2}$ Paaren von Patches Energie fließen kann.

Dieser Nachteil wird durch das hierarchische Radiosity-Verfahren behoben (vgl. [30] und [31]): Beginnend mit maximal großen Start-Patches (anschaulich z.B. wieder gesamte Decke eines Raums) wird der Formfaktor (anschaulich: der potentielle Energiefluss) zwischen diesen Patches geschätzt. Ergibt diese Schätzung einen „großen“ Formfaktor, so werden die Patches in Kind-Patches zerlegt und die Berechnung rekursiv mit den Kind-Patches wiederholt. Das Verfahren bricht ab, sobald der Formfaktor zwischen Kind-Patches klein genug oder eine minimale Patchgröße erreicht ist.

Dies ist in Abb. 2.18 für eine punktförmige Lichtquelle **L** und ein Patch **P** dargestellt: für das Paar **(L, P)** ergibt sich ein großer Energiefluss, also wird **P** in vier Kind-Patches zerlegt und der Energiefluss für die Paare **(L, P_i)** erneut ausgewertet.

In Abb. 2.19 – Abb. 2.21 ist die daraus resultierende Beleuchtung von **P** (bzw. der Kind-Patches) bei Schrittweiser Unterteilung von **P** dargestellt. Abb. 2.20 zeigt die zweite Stufe der Unterteilung, die nur einen sehr groben Eindruck der Beleuchtung vermittelt. Durch weitere Unterteilung in Kind-Patches nähert sich die Beleuchtung schrittweise dem aus der Realität vertrauten Bild (vgl. Abb. 2.21): direkt unter der Lichtquelle **L** (d.h. in der Mitte von **P**) ist das Patch **P** am hellsten, zu den Rändern von **P** hin nimmt die Helligkeit ab.

Ein Problem wurde dabei bisher ignoriert: wenn zwischen **L** und **P** ein anderes Patch **P'** liegt, so beeinflusst **P'** den Energiefluss zwischen **L** und **P**, anschaulich zeichnet sich auf Patch **P** der Schatten von **P'** ab. Dies ist in Abb. 2.22 und Abb. 2.23 dargestellt.

Die Berechnung dieses Schattens, d.h. die Unterteilung der Kind-Patches anhand ihrer gegenseitigen Sichtbarkeit, ist das zentrale Thema dieser Arbeit und wird in Kap. 3 und 4 ausführlich diskutiert.

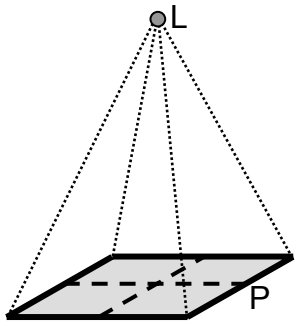


Abb. 2.18

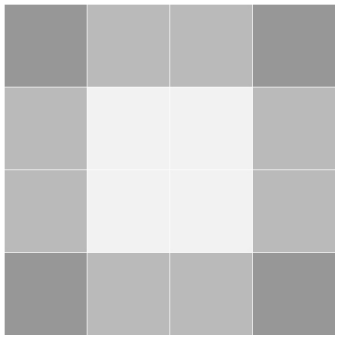


Abb. 2.19

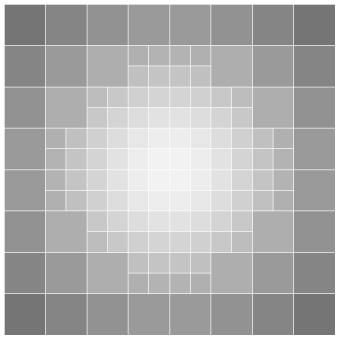


Abb. 2.20

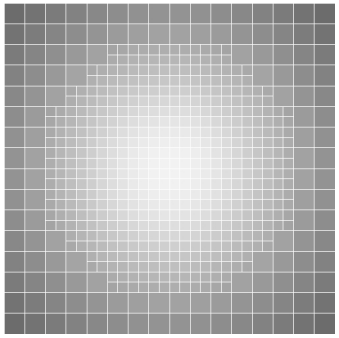


Abb. 2.21

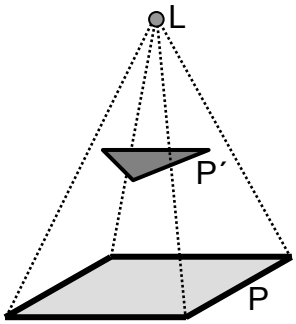


Abb. 2.22

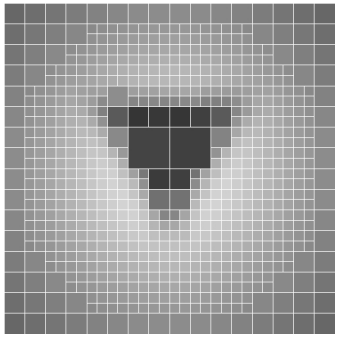


Abb. 2.23

2.5 Zusammenfassung

In diesem Kapitel wurden die grundlegenden Begriffe, Datenstrukturen und Verfahren für den Rest der Arbeit eingeführt. Da sehr viele verschiedene Themen diskutiert wurden, soll an dieser Stelle ein grober Überblick der wichtigsten Begriffe und Aussagen gegeben werden:

- ***Geometrische Primitive:*** Die wichtigsten geometrischen Elemente in dieser Arbeit sind Punkte, Vektoren, Linien, Ebenen, Polygone und Polyeder.

  Für die Linien wurde die Plücker-Darstellung definiert, das Plücker-Skalarprodukt macht Aussagen über den relativen Drehsinn zweier Linien.

  Ein im Weiteren wichtiger Spezialfall eines Polyeders ist die B-Box, die einem achsenparallelen Quader entspricht.

- ***Klassifikationen von geometrischen Primitiven:*** Eine Klassifikation ist eine Abbildung, die für zwei geometrische Primitive Informationen über ihre relative Lage liefert, z.B. ob eine Punkt „unter“, in oder „über“ einer Ebene liegt.

- ***3D-Objekte und 3D-Szenen:*** Ein polygonales 3D-Objekt ist ein Polyeder. Die Elemente eines 3D-Objekts sind die Vertices, Kanten und Facetten des Polyeders. Der Oberbegriff für Elemente und 3D-Objekte ist 3D-Entität.

  Eine 3D-Szene ist eine (nicht-leere) Menge von 3D-Objekten. 3D-Szenen können als Liste von 3D-Objekten oder als BB-Hierarchie gespeichert werden.

  Eine BB-Hierarchie ist ein Baum von B-Boxen: Die Wurzel des Baums ist eine B-Box, die die gesamte 3D-Szene umfasst und die Blätter sind B-Boxen, die 3D-Objekte enthalten.

- ***Konvexe Hüllen von 3D-Objekten:*** Die konvexe Hülle zweier 3D-Objekte ist aus Linien und Ebenen aufgebaut. Diese Linien und Ebenen werden durch die Kontur-Elemente der beiden 3D-Objekte definiert.

- ***Das Radiosity-Beleuchtungsmodell:*** Das Radiosity-Beleuchtungsmodell berechnet die diffuse Beleuchtung einer 3D-Szenen. Die Grundidee dabei ist, die 3D-Objekte der 3D-Szene anhand des Lichtflusses zu unterteilen.

  Das zentrale Problem hierbei ist die Einbeziehung der gegenseitigen Sichtbarkeit von 3D-Objekten in die Radiosity-Berechnung.

3 Verfahren zur Sichtbarkeitsberechnung

In diesem Kapitel sollen Algorithmen vorgestellt und untersucht werden, die das globale Sichtbarkeitsproblem lösen, d.h. für alle Paare von 3D-Entitäten einer 3D-Szene die gegenseitige Sichtbarkeit bestimmen. Diese Sichtbarkeitsinformation kann dann im Radiosity-Verfahren zur Berechnung des Energieflusses zwischen diesen 3D-Entitäten verwendet werden.

Zunächst wird die Problemstellung konkretisiert und dabei insbesondere beschrieben, wie das Radiosity-Beleuchtungsmodell verschiedene Arten von Sichtbarkeits-informationen zur Berechnung der globalen Beleuchtung einer 3D-Szene nutzen kann.

Anschließend werden verschiedene Typen von Sichtbarkeitsanfragen betrachtet, die sich im Wesentlichen in zwei Punkten unterscheiden:

- Informationsgehalt, d.h. „Wie aussagekräftig ist eine Sichtbarkeitsanfrage?“
  - „Wird die Sichtbarkeit der 3D-Entitäten ‚irgendwie’ beeinflusst?“
  - „Wie viel (Prozent) sehen die 3D-Entitäten von einander?“
  - „Welche Teile der 3D-Entitäten sind gegenseitig ganz oder teilweise sichtbar?“
- Richtung, d.h. „Was ist Ausgangspunkt und was ist Ziel der Anfrage?“
  - „Wer beeinflusst die Sichtbarkeit gegebener 3D-Entitäten?“
  - „Wessen Sichtbarkeit wird durch gegebene 3D-Entitäten beeinflusst?“

Im Hauptteil des Kapitels werden zwei Ansätze zur Lösung des globalen Sichtbarkeitsproblems vorgestellt: Näherungs- und exakte Verfahren. Der Schwerpunkt liegt auf den exakten Verfahren, die Näherungsverfahren werden nur betrachtet, um einige der dort erarbeiteten Hilfsmittel für die exakten Verfahren anwenden zu können.

Die Idee der Näherungsverfahren ist, 3D-Entitäten so lange rekursiv in Kind-Entitäten zu unterteilen, bis alle Kind-Entitäten gegenseitig entweder vollständig oder überhaupt nicht sichtbar sind oder eine vorgegeben Unterteilungsgrenze erreicht ist. Im letzten Fall kann die Sichtbarkeit zwischen diesen Kind-Entitäten nur angenähert werden.

Die Idee der exakten Verfahren ist, die 3D-Entitäten anhand von Sichtbarkeitswechseln (anschaulich: Schattenlinien) zu unterteilen. Diese Unterteilung ergibt Kind-Entitäten, die entweder gegenseitig vollständig (nicht) sichtbar oder teilweise sichtbar sind.

Die von den exakten Verfahren berechneten Sichtbarkeitsinformationen werden in Form des Visibility Skeletons gespeichert. Das *Visibility Skeleton* ist ein Graph, dessen Kanten alle globalen Sichtbarkeitsinformationen (anschaulich: die Schattenlinien) der 3D-Szene repräsentieren. Das Visibility Skeleton ist die zentrale Datenstruktur dieser Arbeit und wird daher ausführlich definiert, konstruiert und optimiert.

Zu allen vorgestellten Verfahren werden im Rahmen der Arbeit entwickelte Vorschläge zur Verbesserung angegeben. Da sich für die meisten Verfahren selbst in der einfachsten Variante keine oder nur sehr grobe Aussagen über den Aufwand treffen lassen, werden alle betrachteten Verfahren und die zugehörigen Verbesserungen anhand von Experimenten mit konkreten 3D-Szenen diskutiert.

3.1 Problemstellung

Das in Kap. 2.4 vorgestellte Radiosity-Verfahren berechnet die globale Beleuchtung einer 3D-Szene nach einem einfachen Grundprinzip: Für jedes Paar von 3D-Entitäten wird der potentielle Energieaustausch zwischen ihnen berechnet. Ist er größer als eine vorgegebene Schranke, werden die 3D-Entitäten in Kind-Entitäten aufgeteilt und die Berechnung rekursiv für die Kind-Entitäten wiederholt. Ist er kleiner als die vorgegeben Schranke, so ist die Aufteilung genau genug und der Energieaustausch kann stattfinden. Für eine korrekte Berechnung dieses Energieaustauschs ist Information über die gegenseitige Sichtbarkeit der 3D-Entitäten notwendig.

An dieser Stelle empfiehlt es sich, zunächst einige grundsätzliche Eigenschaften des Radiosity-Verfahrens zu betrachten und daraus die wünschenswerten Eigenschaften von Algorithmen zur Berechnung der Sichtbarkeit abzuleiten.

Zunächst gilt, dass das Radiosity-Verfahren progressiv arbeitet (vgl. [31]), d.h. ausgehend von den Lichtquellen die Lichtenergie schrittweise in der 3D-Szene verteilt und so die globale Beleuchtung der 3D-Szene berechnet. Dies bedeutet insbesondere, dass bei der Radiosity-Berechnung 3D-Entitäten nur dann betrachtet werden, wenn sie auch Radiosity tragen, d.h. anschaulich „leuchten“. Umgekehrt kann Radiosity nur zwischen 3D-Entitäten fließen, die gegenseitig (teilweise) sichtbar sind.

Diese beiden Abhängigkeiten machen also eine Rückkopplung zwischen dem Radiosity-Verfahren und der Sichtbarkeitsberechnung wünschenswert. In Kap. 3.4 und Kap. 3.5 wird gezeigt, dass nur die Näherungsverfahren mit dem Radiosity-Verfahren rückgekoppelt werden können, die exakten Verfahren sind dagegen eine Vorstufe des Radiosity-Verfahrens und können von diesem nicht wieder beeinflusst werden.

Als zweiter wichtiger Punkt soll der Informationsgehalt der Sichtbarkeitsanfragen betrachtet werden. Das Radiosity-Verfahren lässt sich in zwei Phasen einteilen: in der ersten Phase werden die 3D-Entitäten **A** und **B** in Kind-Entitäten **a** und **b** unterteilt, in der zweiten Phase wird für jedes Paar **(a, b)** von Kind-Entitäten der Energieaustausch unter Berücksichtigung der Sichtbarkeit berechnet. Damit folgt, dass in der ersten Phase des Radiosity-Verfahrens nur *qualitative Sichtbarkeitsinformation* benötigt werden, d.h.

- „Sind **(a, b)** gegenseitig vollständig sichtbar?“ (Für das Paar **(a, b)** müssen dann keine weiteren Sichtbarkeitsberechnungen durchgeführt werden)
- „Sind **(a, b)** gegenseitig vollständig nicht sichtbar?“ (Im Radiosity-Verfahren kann die rekursive Unterteilung für das Paar **(a, b)** abgebrochen werden)
- „Welche Generator-Entitäten $\mathbf{G_i}$ beeinflussen die Sichtbarkeit zwischen **a** und **b**?“ (Nur diese können die Sichtbarkeit zwischen Kind-Entitäten von **(a, b)** beeinflussen)

Bei den Näherungsverfahren müssen diese Sichtbarkeitsinformationen in jedem Unterteilungsschritt des Radiosity-Verfahrens neu berechnet werden. Bei Verwendung der exakten Verfahren liegt diese Information dagegen bereits in Form des (vorberechneten) Visibility Skeletons vor. Umgekehrt gilt, dass die Näherungsverfahren die Sichtbarkeit nur für 3D-Entitäten auswerten, zwischen denen auch Energie fließen kann. Die exakten Verfahren dagegen berechnen die globale Sichtbarkeit innerhalb der gesamten 3D-Szene vorab, egal ob dort (später) Energie fließen kann.

Beide Verfahren haben also wechselseitige Vor- und Nachteile: die Stärken der Näherungsverfahren sind die Schwächen der exakten Verfahren und umgekehrt. In Kap. 4 wird eine Kombination der beiden Verfahren vorgeschlagen, die die Stärken der beiden Verfahren miteinander verbindet.

3.2 Grundlegende Begriffe

In diesem Kapitel werden die für Sichtbarkeitsberechnungen grundlegenden Begriffe definiert. Zuerst werden die 3D-Entitäten in Extremal- und Generator-Entitäten eingeteilt und die umgangssprachlichen Beschreibungen der vollständigen oder teilweisen Sichtbarkeit formalisiert.

Dann wird die Unterscheidung zwischen lokaler und globaler Sichtbarkeit erklärt und ein grundlegendes Schema zur Behandlung der lokalen Sichtbarkeit erklärt. Dies führt zur besonders in Kap. 3.5 häufig benötigten Klassifikation von 3D-Entitäten anhand von Elementen (Vertices, Kanten oder Facetten).

Zuerst werden die Rollen von 3D-Entitäten bei der Betrachtung der Sichtbarkeit beschrieben (vgl. Abb. 3.1):

Definition 3.1: Extremal- und Generator-Entitäten

3D-Entitäten, deren gegenseitige Sichtbarkeit (von anderen 3D-Entitäten) beeinflusst wird, heißen *Extremal-Entitäten* (kurz: *Extrema*).

3D-Entitäten, die die gegenseitige Sichtbarkeit gegebener Extremal-Entitäten **(A, B)** beeinflussen, heißen *Generator-Entitäten* (kurz: *Generatoren*) **(A, B)**.

Es sei betont, dass diese Einteilung einen Rollencharakter hat, d.h. dass eine 3D-Entität je nach Kontext sowohl Extremal- als auch Generator-Entität sein kann.

Die folgenden Definitionen formalisieren die umgangssprachlichen Beschreibungen von vollständiger und teilweiser Sichtbarkeit von Extremal-Entitäten. Dazu wird zunächst nur die Sichtbarkeit zwischen zwei Punkten betrachtet:

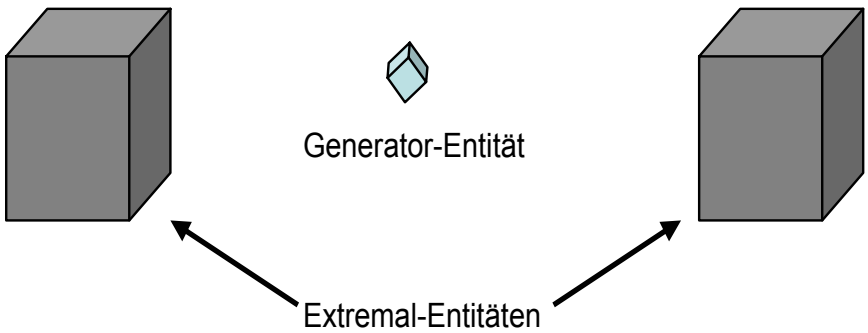


Abb. 3.1
Extremal- und Generator-Entitäten

Definition 3.2: Sichtbarkeit Punkt-Punkt

Zwei Punkte $\mathbf{p}, \mathbf{q} \in \mathbf{R}^3$ in einer 3D-Szene **S** heißen *gegenseitig sichtbar*, wenn $\mathbf{r}_p^q(\mathbf{L(p,q),S}) = \mathbf{OUT}$ gilt. Sonst heißen **p** und **q** *gegenseitig nicht sichtbar*.

Darauf aufbauend ergibt sich die Definition für die Sichtbarkeit zwischen 3D-Entitäten:

Definition 3.3: Sichtbarkeit 3D-Entität – 3D-Entität

Zwei 3D-Entitäten **A** und **B** heißen *gegenseitig vollständig* (*nicht*) *sichtbar*, wenn für

$\forall(\mathbf{p, q}) \in \mathbf{A} \times \mathbf{B}$: **(p, q)** *gegenseitig (nicht) sichtbar*

gilt. Sonst heißen **A** und **B** *gegenseitig teilweise sichtbar*.

Zur Definition der gegenseitigen Sichtbarkeit von Punkten wurde die Verdeckungs-Strahlanfrage $\mathbf{r}_p^q(\mathbf{L(p,q),S})$ verwendet. Diese liefert **OCC** (*verdeckt*) zurück, wenn der Strahl **L(p, q)** „zwischen" den Aufpunkten **p** und **q** ein 3D-Objekt **G** schneidet.

In Abb. 3.2 ist zu erkennen, dass für Paare **(p, q)** von Punkten auf der Oberfläche zweier Extremal-Objekte **P** und **Q** auch **G = P** oder **G = Q** gelten kann. Dies bedeutet, dass die Extremal-Objekte selber ihre gegenseitige Sichtbarkeit beeinflussen.

Dies führt zur Definition der *lokalen Sichtbarkeit*:

Definition 3.4: Lokale Sichtbarkeit

Seien $\mathbf{(p, q)} \in \mathbf{P} \times \mathbf{Q}$ zwei Punkte auf 3D-Entittäten **P** und **Q.**

Gilt $\mathbf{R}_p^q(\mathbf{L(p,q),\{P,Q\}}) \cap \mathbf{\{P,Q\}} = \varnothing$, so heißen die Punkte **p** und **q** gegenseitig lokal sichtbar. Sonst heißen **p** und **q** lokal nicht sichtbar.

Gilt $\mathbf{P} \notin \mathbf{R}_p^q(\mathbf{L(p,q),\{P,Q\}})$, so heißt **q** von **p** lokal sichtbar.

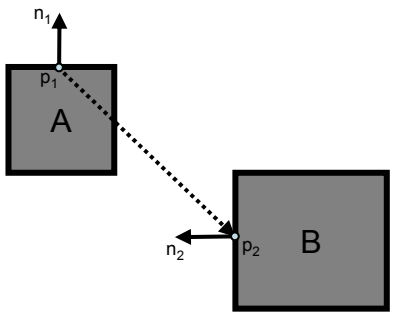


Abb. 3.2
Lokal (nicht) sichtbare Punkte

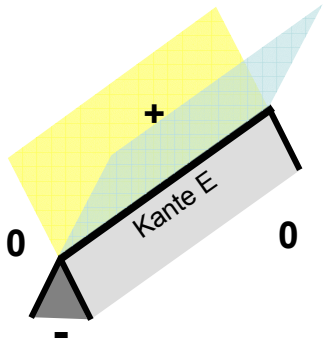


Abb. 3.3
Halbräume einer Kante

Dies kann wieder auf 3D-Entitäten verallgemeinert werden:

Definition 3.5: Lokale Sichtbarkeit von 3D-Entitäten

Seien **P** und **Q** zwei 3D-Entitäten. Dann heißen **P** und **Q** gegenseitig lokal (nicht) sichtbar, wenn alle Paare von Punkten $(p, q) \in P \times Q$ gegenseitig lokal (nicht) sichtbar sind. Sonst heißen **P** und **Q** gegenseitig teilweise lokal sichtbar.

Gilt für alle Paare (p, q) $P \notin R_p^q(L(p,q),\{P,Q\})$, so heißt **Q** von **P** lokal sichtbar.

Für eine Facette und eine 3D-Entität kann lokale Sichtbarkeit so ausgedrückt werden:

Satz 3.1: Lokale Sichtbarkeit von Facetten und 3D-Entitäten

Sei **F** eine Facette und **Q** eine 3D-Entität. Dann ist **Q** von **F** vollständig lokal (nicht) sichtbar, wenn. **Q** vollständig im positiven (negativen) Halbraum von **F** liegt.

Sonst ist **Q** von **F** teilweise lokal sichtbar.

[Ohne Beweis]

Diese Aussage kann direkt auf Kanten und Vertices verallgemeinert werden:

Satz 3.2: Lokale Sichtbarkeit von Kanten bzw. Vertices und Facetten

Sei **E** eine Kante (bzw. ein Vertex) mit adjazenten Facetten F_i und **Q** eine 3D-Entität. Dann ist **Q** von **E** lokal sichtbar, wenn **Q** von mindestens einer Facette F_i lokal sichtbar ist.

[Ohne Beweis]

Die folgende Definition verallgemeinert diesen lokalen Sichtbarkeitstest zur Klassifikation von 3D-Entitäten anhand von Vertices und Kanten (vgl. Abb. 3.6):

Definition 3.6: Positive, negative und indefinite Halbräume von Vertices und Kanten

Sei **E** ein Vertex oder eine Kante mit adjazenten Facetten F_i. Dann heißt

$E^+ = \bigcap_{F_i} F_i^+$ *positiver Halbraum* von **E**

$E^- = \bigcap_{F_i} F_i^-$ *negativer Halbraum* von **E**

$E^0 = R^3 - (E^+ \cup E^-)$ *indefiniter Halbraum* von **E**

3.3 Sichtbarkeitsanfragen

Eine Sichtbarkeitsanfrage ist eine Abbildung, die jedem Paar (bzw. n-Tupel) von Extremal- (bzw. Generator-) Entitäten Sichtbarkeitsinformationen zuordnet. Diese noch ungenaue Beschreibung wird im Folgenden konkretisiert.

Besonderer Augenmerk liegt hierbei auf einer einheitlichen Darstellung der verschiedenen Sichtbarkeitsanfragen: zwar sind alle Sichtbarkeitsanfragen in der verwendeten Literatur bereits definiert, allerdings fehlt eine gemeinsame Nomenklatur und insbesondere auch eine saubere Abgrenzung der einzelnen Anfragen voneinander.

Zunächst wird die Richtung einer Sichtbarkeitsanfrage definiert:

Definition 3.7: Richtung von Sichtbarkeitsanfragen

Eine Sichtbarkeitsanfrage heißt *Extremal-basiert*, wenn sie zu gegebenen Extremal-Entitäten Informationen über die zugehörigen Generator-Entitäten liefert.

Eine Sichtbarkeitsanfrage heißt *Generator-basiert*, wenn sie zu gegebenen Generator-Entitäten Informationen über die zugehörigen Extremal-Entitäten liefert.

Bisher wurden noch keine Aussagen darüber getroffen, welche Art von Informationen eine konkrete Sichtbarkeitsanfrage zurückliefert. In dieser Arbeit wird zwischen folgenden Typen von Sichtbarkeitsinformationen unterschieden:

(1) ***qualitative Sichtbarkeitsinformationen*:** „Welche 3D-Entitäten beeinflussen die Sichtbarkeit gegebener Extremal-Entitäten?“ oder „Die Sichtbarkeit welcher Extremal-Entitäten wird durch die gegebenen Generator-Entitäten beeinflusst?“

(2) ***quantitative Sichtbarkeitsinformationen*:** „Wie viel (Prozent) sehen gegebene Extremal-Emtitäten voneinander?“

(3) ***volle Sichtbarkeitsinformationen*:** „Welche Teile gegebener Extremal-Entitäten sind gegenseitig teilweise oder vollständig (nicht) sichtbar?“

Diese umgangssprachlichen Beschreibungen werden nun in den folgenden Definitionen konkretisiert. Dabei werden zunächst nur Extremal-basierte Sichtbarkeitsanfragen betrachtet, da die Generator-basierten Sichtbarkeitsanfragen erst mit den in Kap. 3.4.6 erarbeiteten Begriffen sauber und verständlich formuliert werden können.

Definition 3.8: Qualitative Sichtbarkeitsanfragen

In einer 3D-Szene **S** mit enthaltenen 3D-Entitäten **A≠B≠G ∈ S** unterscheidet man folgende Extremal-basierte qualitative Sichtbarkeitsanfragen (vgl. Abb. 3.4):

(1) Die qualitative Extremal-Sichtbarkeitsanfrage $\mathbf{V_{qual}(A, B)}$ unterscheidet für das Extremal-Entitäten-Paar **(A, B)** lediglich zwischen den in Definition 3.3 erarbeiteten Begriffen:

$$\mathbf{V_{qual}} : \mathbf{(A,B)} \rightarrow \begin{cases} \mathbf{OCC} & \mathbf{(A, B)} \text{ gegenseitig vollständig nicht sichtbar} \\ \mathbf{PAR} & \mathbf{(A, B)} \text{ gegenseitig teilweise sichtbar} \\ \mathbf{VIS} & \mathbf{(A,B)} \text{ gegenseitig vollständig sichtbar} \end{cases}$$

(2) Die kleine Generatoranfrage **g(A, B, G)** gibt an, ob die 3D-Entität **G** bezüglich des Extremal-Entitäten-Paars **(A, B)** ein Generator ist:

$$\mathbf{g} : \mathbf{(A,B,G)} \rightarrow \begin{cases} \mathbf{OCC} & \mathbf{G} \text{ ist vollständig verdeckender Generator bzgl. } \mathbf{(A, B)} \\ \mathbf{OUT} & \mathbf{G} \text{ ist kein Generator bzgl. } \mathbf{(A, B)} \\ \mathbf{PAR} & \text{sonst} \end{cases}$$

(3) Die große Generatoranfrage **G(A, B)** gibt zu den Extremal-Entitäten **(A, B)** die Menge aller Generator-Entitäten bezüglich **(A, B)** zurück:

$\mathbf{G: (A, B) \rightarrow \mathcal{P}(S)}$ mit $\mathbf{G \in G(A, B) \Leftrightarrow g(A, B, G) \neq OUT}$

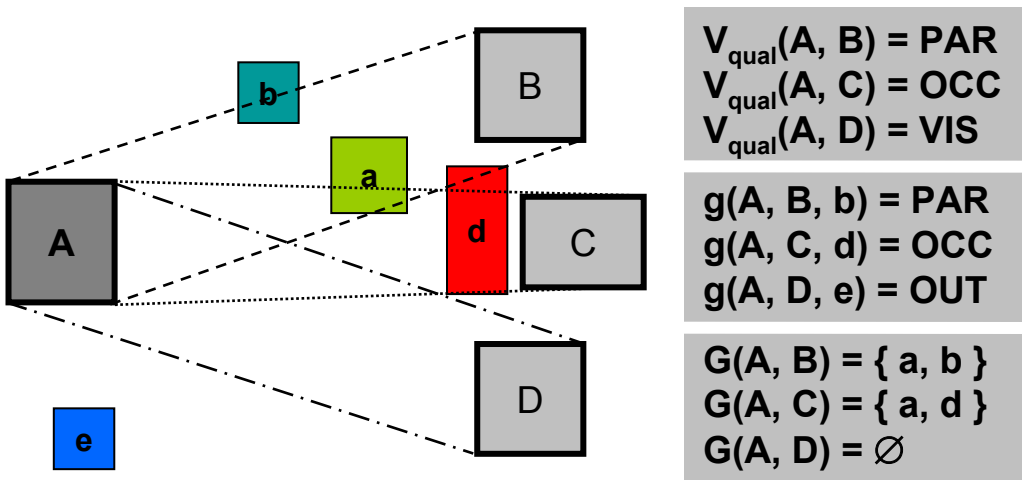


Abb. 3.4
Qualitative sowie kleine und große Generatoranfragen

Eine Variante der qualitativen Sichtbarkeitsanfragen sind die *qualitativen Sichtbarkeitsschätzungen*, auch *konservative Sichtbarkeitsanfragen* genannt. Da die Entscheidung, ob eine gegebene 3D-Entität bezüglich zweier Extremal-Entitäten ein Generator ist, oft nur mit hohem Rechenaufwand getroffen werden kann, begnügt man sich mit einer Schätzung oder Vermutung der Generator-Eigenschaft (vgl. [29] und [16].

Da dies natürlich zu Fehleinschätzungen führen kann, muss man ein Kriterium definieren, welche Art von Fehlern fatal und welche vernachlässigbar ist. Dies wird in der folgenden Definition für die kleine Generatoranfrage formuliert und kann sinngemäß auf die anderen qualitativen Sichtbarkeitsanfragen übertragen werden:

Definition 3.9: Konservative kleine Generatoranfrage

Die konservative kleine Generatoranfrage $\tilde{\mathbf{g}}(\mathbf{A},\mathbf{B},\mathbf{G})$ ist wie folgt definiert:

(1) $\tilde{\mathbf{g}}(\mathbf{A},\mathbf{B},\mathbf{G}) = \mathbf{OCC} \Leftrightarrow \mathbf{g}(\mathbf{A},\mathbf{B},\mathbf{G}) = \mathbf{OCC}$

(2) $\tilde{\mathbf{g}}(\mathbf{A},\mathbf{B},\mathbf{G}) = \mathbf{OUT} \Rightarrow \mathbf{g}(\mathbf{A},\mathbf{B},\mathbf{G}) = \mathbf{OUT}$

(3) $\tilde{\mathbf{g}}(\mathbf{A},\mathbf{B},\mathbf{G}) = \mathbf{PAR} \Rightarrow \mathbf{g}(\mathbf{A},\mathbf{B},\mathbf{G}) = \mathbf{PAR} \vee \mathbf{g}(\mathbf{A},\mathbf{B},\mathbf{G}) = \mathbf{OUT}$

Konservativ bedeutet in diesem Zusammenhang also nicht anderes, als dass die konservativen Klassifikationen **OCC** und **OUT** auf jedem Fall richtig sind, im Falle der konservativen Klassifikation **PAR** das richtige Ergebnis aber auch **OUT** lauten könnte.

Die *konservative qualitative Sichtbarkeitsanfrage* $\hat{\mathbf{V}}_{\mathbf{qual}}(\mathbf{A},\mathbf{B})$ kann demzufolge also die Klassifikation **PAR** liefern, obwohl die richtige Klassifikation **VIS** ergeben würde. Weiterhin kann bei der *konservativen großen Generatoranfrage* $\hat{\mathbf{G}}(\mathbf{A}, \mathbf{B})$ die Liste der Generatoren unnötige 3D-Entitäten **e** (d.h. $\mathbf{g}(\mathbf{A}, \mathbf{B}, \mathbf{e}) = \mathbf{OUT}$) enthalten, jedoch auf keinem Fall tatsächliche Generatoren verfehlen.

Zusammenfassend kann man also festhalten, dass qualitative Sichtbarkeitsschätzungen niemals eine Klassifikation liefern, die nicht durch eine anschließende qualitative Sichtbarkeitsanfrage korrigiert werden könnte.

Diese obigen qualitativen Aussagen können durch die Einführung eines *Sichtbarkeitsmaßes* quantisiert werden:

Definition 3.10: Quantitative Sichtbarkeitsanfragen

In einer 3D-Szene $\mathbf{S}$ mit enthaltenen 3D-Entitäten $\mathbf{A \neq B \in S}$ und Punkten $\mathbf{a \in A}$ und $\mathbf{b \in B}$ ist die *quantitative Sichtbarkeitsanfrage* $\mathbf{V_{quant}}$ wie folgt definiert:

$$\mathbf{V_{quant} : (A,B) \rightarrow [0,1]} \quad \text{mit} \quad \mathbf{V_{quant}(A,B)} = \frac{\mathbf{1}}{\mathbf{s_A \cdot s_B}} \int\int_{\mathbf{AB}} \delta(\mathbf{a,b}) \, dadb$$

$$\text{und } \delta(\mathbf{a,b}) = \begin{cases} \mathbf{0} & \mathbf{(a,b)} \text{ gegenseitig nicht sichtbar} \\ \mathbf{1} & \mathbf{(a,b)} \text{ gegenseitig sichtbar} \end{cases}$$

$$\text{und } \mathbf{s_X} = \begin{cases} \mathbf{Fläche(X)} & \mathbf{X} \text{ 3D - Objekt oder Polygon} \\ \mathbf{Länge(X)} & \mathbf{X} \text{ Liniensegment} \\ \mathbf{1} & \mathbf{X} \text{ Punkt} \end{cases}$$

Das Ergebnis der quantitativen Sichtbarkeitsanfrage wird auch als Sichtbarkeitskoeffizient $\mathbf{v_{AB}}$ bezeichnet.

Anschaulich entspricht das obige Integral der „Summe“ der gegenseitig sichtbaren Punkte auf den Extremal-Entitäten **A** und **B** und der Faktor $\mathbf{(s_A \cdot s_B)^{-1}}$ einer Normierung dieser Summe auf das Intervall **[0, 1]**. Damit lässt sich der ungenaue Begriff der teilweisen Sichtbarkeit präzisieren: statt zu sagen, dass die Entitäten **A** und **B** gegenseitig teilweise sichtbar sind, kann man nun z.B. sagen, dass **A** und **B** zu 80% gegenseitig sichtbar sind.

Die in den folgenden Kapiteln betrachteten Verfahren zur Berechnung der Sichtbarkeit arbeiten alle nach einem ähnlichen Prinzip: Extremal-Entitäten **(A, B)**, die gegenseitig teilweise sichtbar sind, werden in Kind-Entitäten $\mathbf{a_i}$ und $\mathbf{b_j}$ zerlegt, so dass für „möglichst viele“ Paare $\mathbf{(a_i, b_j)}$ gegenseitig vollständig (nicht) sichtbar gilt.

Auch dies kann als Sichtbarkeitsanfrage formuliert werden:

Definition 3.11: Zerlegungsanfrage

Für zwei Extremal-Entitäten **A** und **B** liefert die Zerlegungsanfrage **Z(A, B)** eine Zerlegung von **A** und **B** in Kind-Entitäten $\mathbf{a_i}$ und $\mathbf{b_j}$. Dabei wird jedes Paar $\mathbf{(a_i, b_j)}$ durch seine gegenseitige Sichtbarkeit gewichtet, d.h. zu einem Tupel $\mathbf{(a_i, b_j, V_{quant}(a_i, b_j))}$ erweitert.

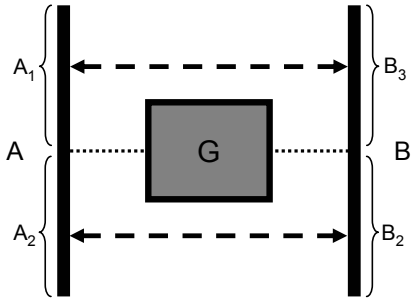


Abb. 3.5
Schlechte Zerlegung

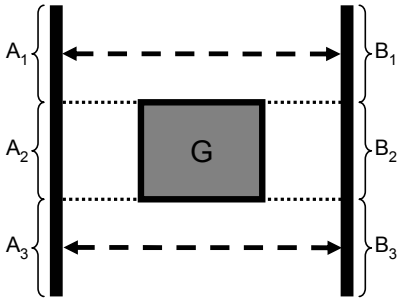


Abb. 3.6
Bessere Zerlegung

Man beachte, dass die Zerlegungsanfrage in der obigen Form keinerlei Aussagen über die Qualität der Zerlegungen bzw. der Gewichtung macht:

- $\mathbf{V_{quant}(A, B)}$ kann auch als Zerlegungsanfrage formuliert werden, indem man als triviale „Zerlegungen" die Extremal-Entitäten **A** und **B** selber wählt und das Paar **(A, B)** mit der quantitativen Sichtbarkeitsanfrage $\mathbf{V_{quant}(A, B)}$ gewichtet
- es ist nicht sicher gestellt, dass die Zerlegung sinnvoll ist, d.h. z.B. gegenseitig vollständig (nicht) sichtbare Kind-Entitäten in den Zerlegungen auftauchen (vgl. Abb. 3.5 und Abb. 3.6)

Alle bisher betrachteten Sichtbarkeitsanfragen sind Extremal-basiert, die Generator-basierten Sichtbarkeitsanfragen werden erst in Kap. 3.5 in Form des *Visibility Skeletons* einer 3D-Szenen betrachtet. Zumindest anschaulich können sie jedoch bereits hier den Extremal-basierten gegenüber gestellt werden:

	Extremal-basiert	***Generator-basiert***
Qualitativ	"Welche Generator-Entitäten beeinflussen (wie) die Sichtbarkeit gegebener Extremal-Entitäten?"	„Die Sichtbarkeit welcher Extremal-Entitäten wird durch die geg. Generator-Entitäten beeinflusst?"
Quantitativ	"Wie viel (Prozent) sehen die geg. Extremal-Entitäten voneinander?"	*„Wie stark wird die Sichtbarkeit von Extremal-Entitäten durch geg. Generator-Entitäten beeinflusst?"*
Voll	"Welche Zerlegung der gegebenen Extremal-Entitäten wird durch die zugehörigen Generator-Entitäten induziert?"	„Welche Zerlegung induzieren gegebene Generator-Entitäten auf den zugehörigen Extremal-Entitäten?"

Tabelle 3.1

3.4 Näherungsverfahren

Dieses Teilkapitel beschäftigt sich mit Näherungsverfahren zur Lösung des globalen Sichtbarkeitsproblems. In Kap. 3.1 wurde bereits ausgeführt, dass die Näherungsverfahren im Prinzip nur eine Erweiterung des Radiosity-Verfahrens sind:

In jedem Unterteilungsschritt des Radiosity-Verfahrens wird mittels der qualitativen Sichtbarkeitsanfrage $\mathbf{V_{qual}(a, b)}$ entschieden, ob die aktuell betrachteten Kind-Entitäten **(a, b)** gegenseitig vollständig (nicht) oder teilweise sichtbar sind. Ist die Unterteilung genau genug, wird mittels der quantitativen Sichtbarkeitsanfrage $\mathbf{V_{quant}(a, b)}$ der Sichtbarkeitskoeffizient $\mathbf{v_{ab}}$ berechnet bzw. geschätzt.

In diesem Kapitel muss also die Umsetzung der quantitativen und qualitativen Sichtbarkeitsanfragen diskutiert werden. Zuerst wird jedoch die Idee der Näherungsverfahren anhand eines einfachen 2D-Beispiels veranschaulicht.

Dann werden Verfahren zur Umsetzung der quantitativen und qualitativen Sichtbarkeitsanfragen vorgestellt. Der Schwerpunkt hierbei liegt auf den qualitativen Sichtbarkeitsanfragen. Die quantitativen Sichtbarkeitsanfragen werden nur im Ansatz diskutiert, da der Schwerpunkt im weiteren Verlauf der Arbeit auf den qualitativen Sichtbarkeitsanfragen und Zerlegungsanfragen liegt.

Grundlegend für die Umsetzung der qualitativen Sichtbarkeitsanfragen ist das in diesem Teilkapitel eingeführte Konzept des *Shafts* zweier 3D-Objekte: alle 3D-Objekte „im“ *Shaft* beeinflussen die gegenseitige Sichtbarkeit der Extremal-Entitäten. Die Entscheidung, ob ein 3D-Objekt „im“ *Shaft* liegt, ist das zentrale Problem dieses Teilkapitels und wird ausführlich diskutiert.

Hierzu werden zwei Ansätze betrachtet, die beide konservative, d.h. nicht notwendig korrekte, Ergebnisse liefern. Um diesen Nachteil zu beheben, wird eine Kombination dieser beiden Ansätze vorgeschlagen, die immer das korrekte Ergebnis liefert.

Anschließend werden einige Optimierungen für die Berechnung der qualitativen Sichtbarkeitsanfragen vorgestellt und anhand von Experimenten und Messungen mit konkreten 3D-Szenen bewertet.

3.4.1 Ein einfaches Beispiel

Das folgende 2D-Beispiel soll die grundsätzliche Idee der Näherungsverfahren veranschaulichen: zwischen den Extremal-Objekten **A** und **B** befindet sich ein Generator-Objekt **G** und beeinflusst die gegenseitige Sichtbarkeit von **A** und **B** (vgl. Abb. 3.7). In den folgenden Schritten werden die Extremal-Entitäten **(A, B)** so lange in Kind-Entitäten $\mathbf{A_i}$ und $\mathbf{B_i}$ aufgeteilt, bis jedes Paar $\mathbf{(A_i, B_j)}$ von Kind-Entitäten gegenseitig vollständig (nicht) sichtbar ist oder eine vorgegebene Unterteilungstiefe erreicht ist.

In Abb. 3.7 sind **A** und **B** gegenseitig teilweise sichtbar, also wird **B** in zwei Kind-Objekte $\mathbf{B_0}$ und $\mathbf{B_1}$ unterteilt und zunächst das Paar $\mathbf{(A, B_0)}$ betrachtet (vgl. Abb. 3.8). Auch **A** und $\mathbf{B_0}$ sind gegenseitig nur teilweise sichtbar, also wird **A** in Kind-Objekte $\mathbf{A_0}$ und $\mathbf{A_1}$ unterteilt und mit den Paaren $\mathbf{(A_0, B_0)}$ und $\mathbf{(A_1, B_0)}$ fortgefahren (vgl. Abb. 3.9). Dieses Verfahren wird fortgesetzt, bis alle Paare $\mathbf{(A_i, B_j)}$ gegenseitig vollständig (nicht) sichtbar sind (vgl. Abb. 3.10) oder die vorgegebene Unterteilungstiefe erreicht ist.

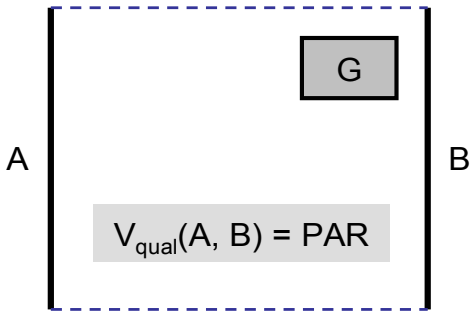


Abb. 3.7

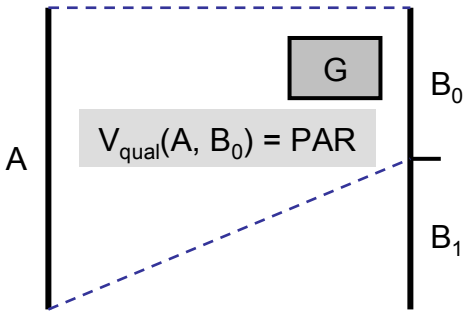


Abb. 3.8

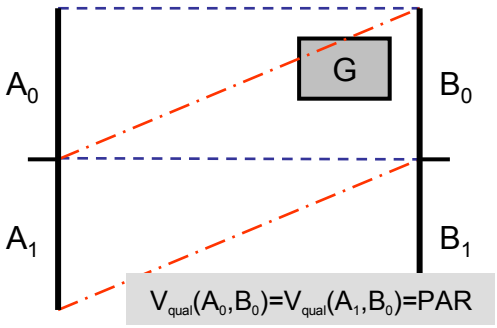


Abb. 3.9

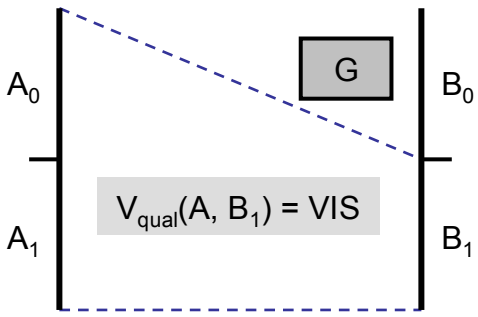


Abb. 3.10

In Abb. 3.11 ist eine vergleichbare Situation im 3D dargestellt, Abb. 3.12 zeigt den ersten Schritt der Unterteilung des Polygons **B**.

In Abb. 3.13 – Abb. 3.15 ist für das Polygon **B** aus die schrittweise Unterteilung zu sehen. Die Helligkeit der Kind-Polygone $\mathbf{B_i}$ entspricht dabei dem Wert der quantitativen Sichtbarkeitsanfrage $\mathbf{V_{quant}(A, B_i)}$, d.h. je heller $\mathbf{B_i}$ ist, desto „sichtbarer" ist es vom Polygon **A** bzw. dessen Kind-Polygonen.

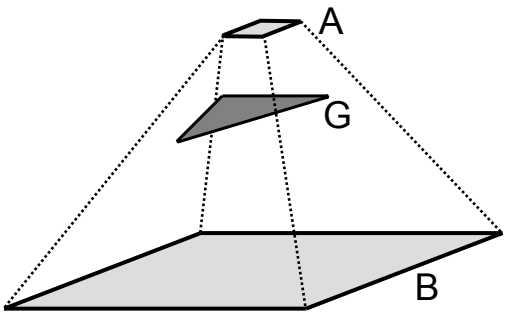


Abb. 3.11
Sichtbarkeit von **A** und **B** wird von **G** beeinflusst

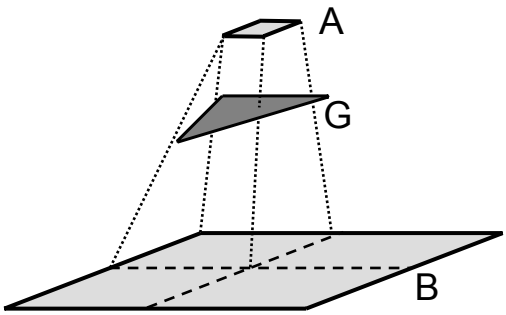


Abb. 3.12
Erste Unterteilung von **B**

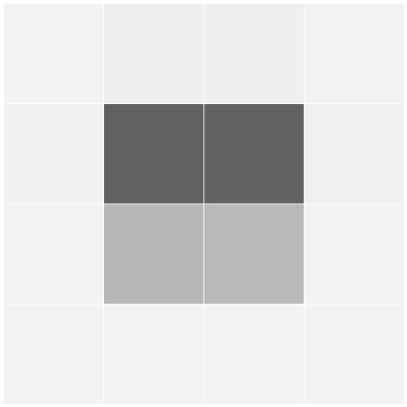


Abb. 3.13
Unterteilung von **B** bei Tiefe 2

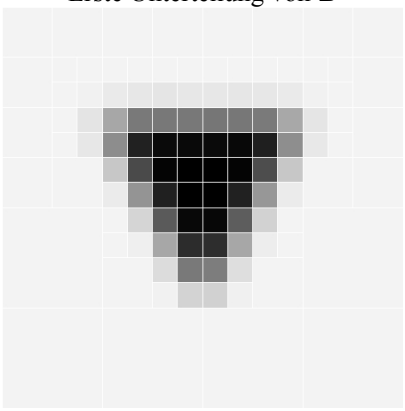


Abb. 3.14
Unterteilung von **B** bei Tiefe 4

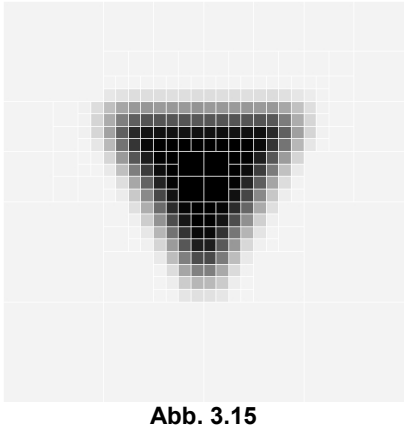


Abb. 3.15
Unterteilung von **B** bei Tiefe 6

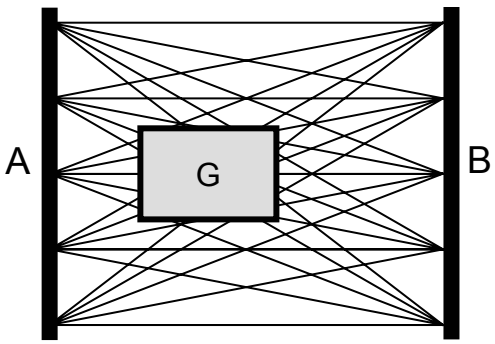


Abb. 3.16
Schätzung von $\mathbf{v_{AB}}$ durch Raycasting

3.4.2 Quantitative Sichtbarkeitsanfragen durch Strahlanfragen

Die quantitative Sichtbarkeitsanfrage $\mathbf{V_{quant}(A, B)}$ bzw. der Sichtbarkeitskoeffizient $\mathbf{v_{AB}}$ gibt an, wie viel die 3D-Entitäten **A** und **B** voneinander „sehen" können. Zur Berechnung bzw. Schätzung dieses Sichtbarkeitskoeffizienten gibt es sehr viele verschiedene Ansätze, von denen hier nur der einfachste diskutiert werden soll: die quantitative Sichtbarkeitsanfrage mit Strahlanfragen (vgl. [3] und [9]).

In Definition 3.10 wurde der Sichtbarkeitskoeffizient zweier Extremal-Entitäten als Integral über die Sichtbarkeitskoeffizienten der Punkte dieser Extremal-Entitäten definiert. Dieses Integral kann nicht direkt gelöst werden, da der nicht-triviale Integrand $\delta\mathbf{(a, b)}$ für unendlich viele Paare von Extremal-Punkten ausgewertet werden müsste.

Eine Lösung dieses Problems bilden die aus der Numerik bekannten Sampling-Verfahren, d.h. der Integrand wird nur in endlich vielen (Paaren von) Sampling-Punkten oder –Regionen ausgewertet und damit der Wert des Integrals angenähert.

Einen einfachen Ansatz für die Näherung der quantitativen Sichtbarkeitsanfrage bilden die in Kapitel 2.2.1 vorgestellte Strahlanfragen: man wählt eine Menge von Sampling-Punkten $\mathbf{(p, q) \in A \times B}$ auf den Extremal-Entitäten **A** und **B**, schießt zwischen allen Paaren **(p, q)** Strahlen und zählt, wie viele dieser Strahlen ein anderes Objekt, d.h. einen Generator, geschnitten haben (vgl. Abb. 3.16). Wenn $\mathbf{n_A}$ (bzw. $\mathbf{n_B}$) die Anzahl der Sampling Punkte auf **A** (bzw. **B**) und $\mathbf{n_i}$ die Anzahl der geschnittenen Strahlen ist, kann daraus eine Schätzung $\hat{\mathbf{v}}_{\mathbf{AB}}$ für den Sichtbarkeitskoeffizienten $\mathbf{v_{AB}}$ berechnet werden:

$$\hat{v}_{AB} = 1 - \frac{n_i}{n_A \cdot n_B}$$ (in Abb. 3.16 ergibt sich z.B. $\hat{v}_{AB} = 1 - \frac{18}{5 \cdot 5} = 0.28$)

Dabei entspricht anschaulich der Faktor $\mathbf{(n_A \cdot n_B - n_i)}$ dem Integral und $\mathbf{(n_A \cdot n_B)^{-1}}$ dem Normierungsfaktor aus Definition 3.10.

Hierzu werden an dieser Stelle keine weiteren Optimierungen vorgeschlagen. Die Optimierung einer einzelnen Strahlanfrage wird in Kap. 3.5.7.1 diskutiert, die Optimierung der Anzahl bzw. der Auswahl der Strahlen führt in die Details des Radiosity-Verfahrens bzw. in die Sampling-Theorie und würde den Rahmen der Arbeit sprengen. Im Folgenden wird die quantitative Sichtbarkeitsanfrage nicht mehr benötigt.

3.4.3 Qualitative Sichtbarkeitsanfragen durch Shaft-Klassifikationen

Im Radiosity-Verfahren wird für ein Paar **(A, B)** von Extremal-Entitäten zunächst nur qualitative Sichtbarkeitsinformation benötigt, d.h. es interessiert nur, welche 3D-Objekte $\mathbf{O_i}$ die Sichtbarkeit zwischen **A** und **B** beeinflussen und ob eines dieser 3D-Objekte $\mathbf{O_i}$ das Paar **(A, B)** vollständig verdeckt.

In Kap. 3.3 wurden zwei qualitative Sichtbarkeitsanfragen unterschieden: Die kleine Generatoranfrage $\mathbf{g(A, B, O_i)}$ gibt an, ob …

- $\mathbf{O_i}$ die gegenseitige Sichtbarkeit von **A** und **B** gar nicht beeinflusst (**OUT**)
- $\mathbf{O_i}$ die Extremal-Entitäten **(A, B)** vollständig verdeckt (**OCC**)
- $\mathbf{O_i}$ die Extremal-Entitäten **(A, B)** teilweise verdeckt (**PAR**)

Die große Generator-Anfrage **G(A, B, S)** liefert eine Liste aller 3D-Objekte $\mathbf{O_i}$ in der 3D-Szene **S** mit $\mathbf{g(A, B, O_i) \neq OUT}$ zurück. Formal bedeuten die Klassifikationen **OUT**, **OCC** und **PAR** für das Tupel $\mathbf{(A, B, O_i)}$ folgendes:

- **OUT**: keine Linie zwischen **A** und **B** schneidet $\mathbf{O_i}$
- **OCC**: alle Linien zwischen **A** und **B** schneiden $\mathbf{O_i}$
- **PAR**: einige (aber nicht alle) Linien zwischen **A** und **B** schneiden $\mathbf{O_i}$

Sei **S** die Menge der Linien zwischen **A** und **B**, wobei **S** als Menge von Punkten betrachtet werden soll. Dann gilt: $\mathbf{S \subset conv(A, B)}$, d.h. die konvexe Hülle der Extremal-Entitäten **A** und **B** umschließt die Menge der Linien zwischen **A** und **B**.

Diese Überlegungen werden in der folgenden Definition formalisiert:

Definition 3.12: Shaft zwischen 3D-Entitäten

Für zwei 3D-Entitäten **A** und **B** ist der Shaft **S(A, B)** definiert als

$$\mathbf{S(A, B) = conv(A, B)}$$

Der Shaft kann dabei auf zwei Weisen repräsentiert werden:

(1) Ebenen-Shaft: Der Ebenen-Shaft $\mathbf{S_E(A, B)}$ wird durch die Menge der Ebenen **E** der konvexen Hülle von **A** und **B** repräsentiert.

(2) Linien-Shaft: Der Linien-Shaft $\mathbf{S_L(A, B)}$ wird durch die Kanten der konvexen Hülle, die einen Vertex von **A** mit einem Vertex von **B** verbinden, repräsentiert.

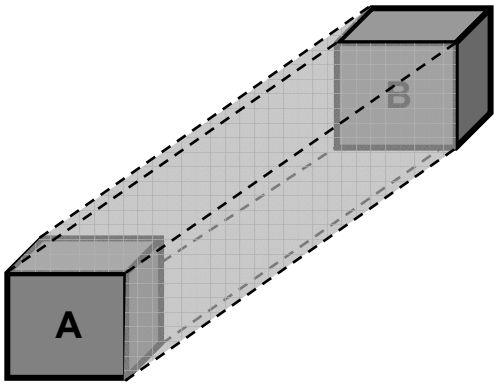


Abb. 3.17
Ebenen-Shaft von **A** und **B**

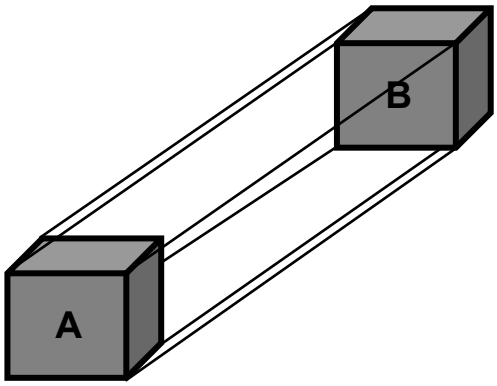


Abb. 3.18
Linien-Shaft von **A** und **B**

Ein Spezialfall dieser allgemeinen Shaft-Struktur ist der Shaft zwischen zwei B-Boxen. Da die B-Boxen laut Definition achsenparallel sind, können die Shaft-Ebenen und –Linien hier sehr viel einfacher berechnet werden. In [29] wird ein einfacher Algorithmus zur Berechnung dieser B-Box-Shafts mit Aufwand **O(1)** vorgeschlagen, der aber hier nicht näher diskutiert wird.

Alle 3D-Objekte, die (teilweise) „im“ Shaft **S(A, B)** der Extremal-Entitäten **(A, B)** liegen, beeinflussen die Sichtbarkeit zwischen **A** und **B**. Es wird also ein Klassifikations-Verfahren benötigt, das zu einem Shaft **S(A, B)** entscheidet, ob ein 3D-Objekt $\mathbf{O_i}$ den Shaft vollständig (**OCC**) oder teilweise (**PAR**) verdeckt oder ob es außerhalb des Shafts liegt (**OUT**).

Die folgenden Sätze zeigen, dass die einzelne Betrachtung des Ebenen- bzw. Linien-Shafts für eine korrekte Klassifikation nicht ausreicht, da einige Klassifikationen nicht eindeutig sind oder gar nicht zurückgeliefert werden. Zunächst wird die in [29] vorgeschlagene Klassifikation anhand der Shaft-Ebenen betrachtet:

Satz 3.3: Klassifikation von 3D-Entitäten anhand eines Ebenen-Shafts

Eine 3D-Entität **G** kann anhand eines abgeschlossenen Ebenen-Shafts $\mathbf{S_E(A, B)}$ im Sinne der qualitativen Sichtbarkeitsanfrage konservativ klassifiziert werden.

Dabei ist die Klassifikation **OUT** auf jedem Fall richtig, die Klassifikation **PAR** kann aber auch **OUT** oder **OCC** bedeuten, wobei **OCC** nicht zurückgeliefert werden kann. Als zusätzliche Klassifikation kann **INS** geliefert werden, was bedeutet, dass **G** vollständig im Shaft $\mathbf{S_E(A, B)}$ enthalten ist.

Beweis:

Zu zeigen ist, dass allein anhand der Klassifikationen von **G** an Ebenen **E** des Shafts $\mathbf{S_E(A, B)}$ die (konservativen) Klassifikationen **OUT**, **PAR** und **INS** gewonnen werden können. Dies zeigt die folgende Fallunterscheidung:

Fall 1: $\exists \mathbf{E}: \mathbf{G} \subset \mathbf{E}^+$: Dann gilt $\mathbf{G} \cap \mathbf{S_E(A, B)} = \varnothing$, d.h. **G** wird korrekt als **OUT** klassifiziert. (vgl. Abb. 3.19).

Fall 2: $\forall \mathbf{E}: \mathbf{G} \subset \mathbf{E}^-$: Dann gilt $\mathbf{G} \subset \mathbf{S_E(A, B)}$, d.h. **G** wird korrekt als **INS** klassifiziert (vgl. Abb. 3.20).

Fall 3: $\exists \mathbf{E}: (\mathbf{G} \cap \mathbf{E}^- \neq \varnothing) \wedge (\mathbf{G} \cap \mathbf{E}^+ \neq \varnothing)$: Dann werden eine oder mehrere Shaft-Ebenen von **G** geschnitten. Dies ergibt folgende Unterfälle:

Fall 3.1 (Abb. 3.21): Alle Ebenen geschnitten, Klassifikation: **INV**
Fall 3.2 (Abb. 3.22): Alle Ebenen geschnitten, Klassifikation **PAR**
Fall 3.3 (Abb. 3.23): Nicht alle Ebenen geschnitten, Klassifikation **PAR**
Fall 3.4 (Abb. 3.24): Nicht alle Ebenen geschnitten, Klassifikation **OUT**

Die Ebenen-Klassifikationen ergeben keine eindeutige Shaft-Klassifikation, daher kann nur das konservative Ergebnis **PAR** zurückgeliefert werden.

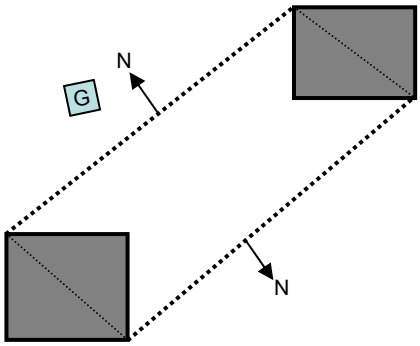


Abb. 3.19
G im positiven Halbraum einer Shaft-Ebene: **OUT**

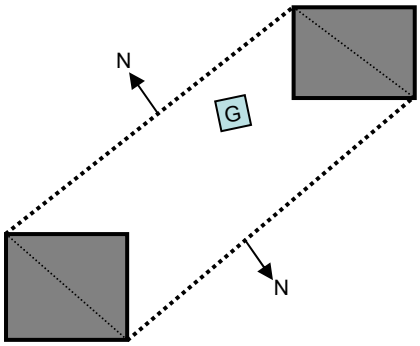


Abb. 3.20
G im negativen Halbraum aller Shaft-Ebenen: **INS**

Die Abbildungen unten zeigen den Shaft-Querschnitt, d.h. man schaut „durch" den Shaft

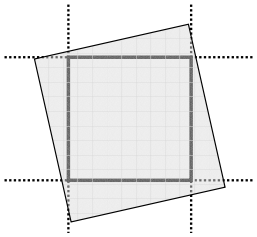


Abb. 3.21
Klassifikation **OCC**

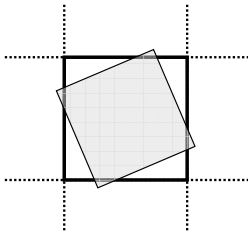


Abb. 3.22
Klassifikation **PAR**

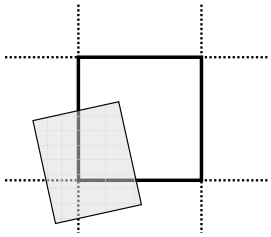


Abb. 3.23
Klassifikation **PAR**

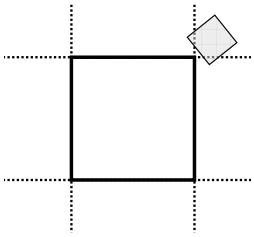


Abb. 3.24
Klassifikation **OUT**

Die konservative Klassifikation einer 3D-Entität anhand eines Ebenen-Shafts kann also **PAR** ergeben, obwohl die richtige Klassifikation **OUT** ist. Dadurch können 3D-Entitäten fälschlich in die Liste der Generatoren-Entitäten aufgenommen werden und müssen im nächsten Unterteilungsschritt erneut (eventuell wieder falsch) klassifiziert werden. Im schlimmsten Fall ändert sich das Ergebnis der qualitativen Sichtbarkeitsanfrage damit von **VIS** nach **PAR**, wodurch im nächsten Unterteilungsschritt wieder die Sichtbarkeit der Extremal-Entitäten ausgewertet wird, obwohl dies eigentlich nicht notwendig wäre.

Weiterhin kann die Sichtbarkeitsberechnung für zwei Extremal-Entitäten abgebrochen werden, sobald eine Generator-Entität als verdeckend (**OCC**) klassifiziert wurde. Da diese Klassifikation jedoch von einem Ebenen-Shaft nicht geliefert werden kann, wird die Unterteilung weitergeführt, bis die Unterteilungsgrenze erreicht ist. Erst dort kann dann mittels der quantitativen Sichtbarkeitsanfrage festgestellt werden, dass für alle Paare **(a, b)** von Kind-Entitäten $\mathbf{v_{ab} = 0}$ gilt.

Diese Probleme können zumindest teilweise durch zusätzliche Betrachtung des Linien-Shafts $\mathbf{S_L(A, B)}$ gelöst werden. Dazu wird zunächst der Spezialfall der Klassifikation eines konvexen Polygons anhand des Linien-Shafts betrachtet, anschließend wird dies wieder auf 3D-Entitäten verallgemeinert (vgl. [56]).

Satz 3.4: Klassifikation von konvexen Polygonen anhand eines Linien-Shafts

Ein konvexes Polygon **P** kann anhand eines Linien-Shafts $\mathbf{S_L(A, B)}$ im Sinne der qualitativen Sichtbarkeitsanfrage konservativ klassifiziert werden.

Dabei sind die Klassifikationen **OUT** und **OCC** auf jedem Fall richtig, die Klassifikation **PAR** kann aber auch **OUT** oder **INS** bedeuten, wobei **INS** nicht als Klassifikation zurückgeliefert werden kann.

Beweis:

O.B.d.A sei **P** Front-Facette bezüglich der Extremal-Entität **A**. Wir betrachten die Plücker-Repräsentationen der Shaft-Linien **L** und der Polygon-Kanten **K**:

Fall 1: $\exists \mathbf{K} : \forall \mathbf{L} : \tilde{\mathbf{K}} \cdot \tilde{\mathbf{L}} < \mathbf{0}$: Es existiert eine Kante **K** von **P**, so dass alle Shaft-´ Linien „rechts“ dieser Kante liegen (vgl. Abb. 3.25). Mit der Konvexität von **K** und **P** bedeutet dies, dass auch der Shaft vollständig „rechts“ von **K** und damit von **P** liegt, d.h. das Polygon wird korrekt als **OUT** klassifiziert.

Fall 2: $\forall \mathbf{K} : \forall \mathbf{L} : \tilde{\mathbf{K}} \cdot \tilde{\mathbf{L}} > \mathbf{0}$: Bezüglich jeder Kante **K** des Polygons liegen jeweils alle Shaft-Linien „links“ von **K** (vgl. Abb. 3.26), d.h. alle Shaft-Linien schneiden das Polygon. Wiederum gilt dann mit der Konvexität des Polygons und des Shafts, dass alle Linien zwischen **A** und **B** das Polygon **P** schneiden, d.h. **P** wird korrekt als **OCC** klassifiziert.

Fall 3: $\left(\exists \mathbf{K} : \exists \mathbf{L} : \tilde{\mathbf{K}} \cdot \tilde{\mathbf{L}} < \mathbf{0}\right) \wedge \left(\exists \mathbf{K} : \exists \mathbf{L} : \tilde{\mathbf{K}} \cdot \tilde{\mathbf{L}} > \mathbf{0}\right)$: Shaft-Linien liegen links und rechts bezüglich der Polygon-Kanten. Abb. 3.27 – Abb. 3.30 zeigen die verschiedenen Unterfälle, die hierbei auftreten können.

Es ist also auch hier nicht möglich, eine eindeutige Klassifikation zurück zu liefern. Daher kann auch hier nur das konservative Ergebnis **PAR** zurückgeliefert werden.

Die Verallgemeinerung auf 3D-Objekte **G** wird hier ohne Beweis angegeben. Dabei werden nur die Front-Facetten von **G** bezüglich **A** betrachtet. Kanten, die adjazent zu einer Front- und einer Rückseiten-Facette sind, werden als Kontur-Kanten bezeichnet.

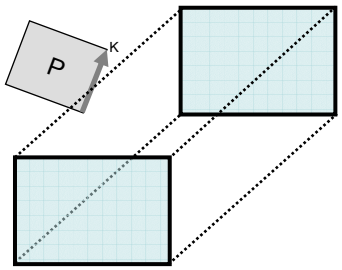


Abb. 3.25
Alle Shaft-Linien liegen „rechts“ einer Kante von **P**

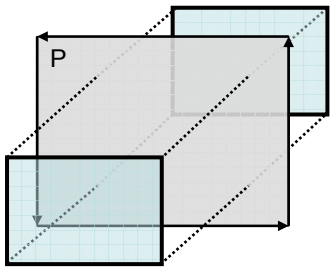


Abb. 3.26
Alle Shaft-Linien liegen „links“ von allen Kanten von **P**

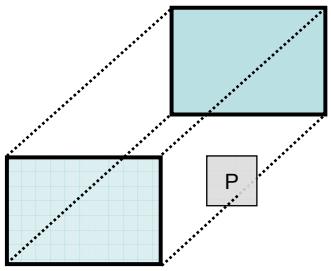


Abb. 3.27
Shaft-Linie schneidet Polygon (**PAR**)

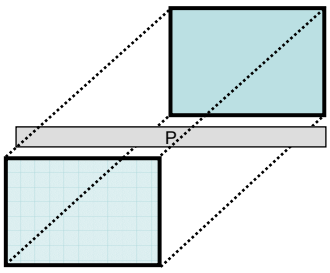


Abb. 3.28
Polygon-Kanten „zwischen“ Shaft-Linien (**PAR**)

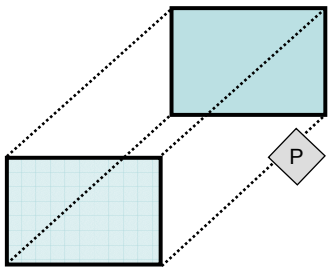


Abb. 3.29
Polygon-Kanten „zwischen“ Shaft-Linien (**OUT**)

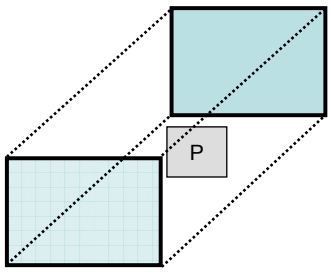


Abb. 3.30
Polygon-Kanten „zwischen“ Shaft-Linien (**INS**)

Klassifikation eines 3D-Objekts **G** anhand eines Linien-Shafts $\mathbf{S_L(A, B)}$:

Beschreibung	*Klassifikation von G*
OUT bezüglich einer Kontur-Kante	**OUT**
Alle Shaft-Linien schneiden eine beliebige Facette von **G**	**OCC**
sonst	**PAR**

Tabelle 3.2

Mit dem vollständigen Shaft liegen also zwei Möglichkeiten zur Klassifikation einer 3D-Entität vor: Klassifikation anhand der Shaft-Ebenen und -Linien.

Die folgende Tabelle zeigt, welche dieser Klassifikationen als richtig übernommen werden können (✓), welche eine weitere Überprüfung notwendig machen (*?*) und welche nicht zurück geliefert werden können (✗):

	OUT	**OCC**	**INS**	**PAR**
Shaft-Ebenen	✓	✗	✓	*?*
Shaft-Linien	✓	✓	✗	*?*

Tabelle 3.3

Alle Klassifikationen bis auf **PAR** können also von einer der Shaft-Repräsentationen eindeutig zurückgeliefert werden. Der folgende Satz zeigt, dass auch die Klassifikation **PAR** eindeutig ist, wenn man sowohl den Ebenen- als auch den Linien-Shaft betrachtet.

Satz 3.5: Klassifikation von konvexen Polygonen anhand des vollständigen Shafts

Ein konvexes Polygon **P** kann anhand des Shafts **S(A, B)** im Sinne der qualitativen Sichtbarkeitsanfrage klassifiziert werden.

Beweis:

Nach Tabelle 3.3 sind die Klassifikationen **OUT**, **OCC** und **INS** korrekt. Es ist zu zeigen, dass $\left(\hat{\mathbf{g}}_{\mathbf{E}}\mathbf{(A,B,G)} = \mathbf{PAR} \wedge \hat{\mathbf{g}}_{\mathbf{L}}\mathbf{(A,B,G)} = \mathbf{PAR}\right) \Rightarrow \mathbf{g(A,B,G)} = \mathbf{PAR}$ gilt.

Wir nehmen also an, dass **P** sowohl anhand des Ebenen- als auch anhand des Linien-Shafts mit konservativem Ergebnis **PAR** klassifiziert wurde. Zur Vereinfachung betrachten wir dazu das Polygon **P** sowie das Polygon **S'**, das durch den Schnitt des Shafts mit der Ebene von **P** definiert wird (vgl. Abb. 3.31 und Abb. 3.32).

Damit reduziert sich das Problem darauf, ob sich zwei Polygone **P** und **S'** im 2D schneiden. Dies ist der Fall, wenn es keine Kante von **P** oder **S'** gibt, so dass das jeweils andere Polygon vollständig rechts dieser Kante liegt (vgl. z.B. [33] oder [36]).

Dies entspricht genau den Klassifikationsbedingungen von Ebenen- und Linien-Shaft: Es gibt keine Shaft-Ebene **E**, so dass **P** vollständig „rechts" (d.h. im positiven Halbraum) von **E** liegt und es gibt keine Polygon-Kante **K**, so dass der Shaft vollständig „rechts" von **K** liegt (d.h. die Shaft-Linien sich gegen den Uhrzeigersinn um die Kante „drehen").

Damit kann also nun ein 3D-Objekt eindeutig anhand des vollständigen Shafts klassifiziert werden, d.h. es kann sicher entschieden werden, ob ein 3D-Objekt die gegenseitige Sichtbarkeit gegebener Extremal-Entitäten beeinflusst oder nicht.

Bis jetzt wurden Shafts als konvexe Hüllen von 3D-Objekten betrachtet. In den folgenden Betrachtungen werden aber auch Shafts zwischen beliebigen Elementen (Vertices, Kanten und Facetten) benötigt. Dies wird in der folgenden Definition formalisiert:

Definition 3.13: Verallgemeinerte Shafts

Ein verallgemeinerter Shaft besteht aus einer Menge **E** von **m** Ebenen $\mathbf{e_i}$, einer Menge **L** von **n** Liniensegmenten $\mathbf{l_j}$ und einer Abbildung **K**, die jedem **(m+n)**-Tupel von Ebenen- bzw. Linien-Klassifikationen (**POS**, **PAR**, **NEG** bzw. **OUT**, **OCC**) einer 3D-Entität eine Shaft-Klassifikation (**OUT**, **INS**, **PAR**, **OCC**) zuordnet.

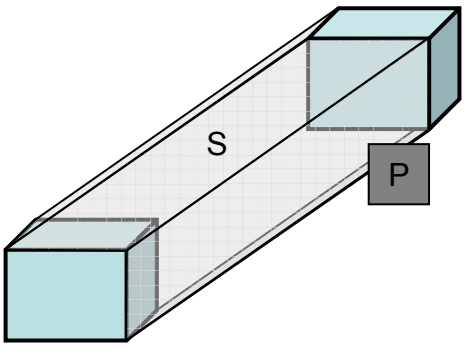


Abb. 3.31
3D-Shaft mit Polygon

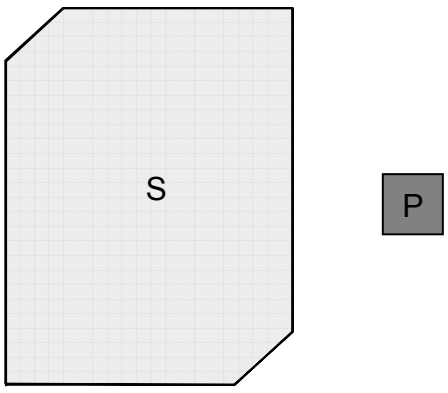


Abb. 3.32
Schnitt des Shafts mit der Ebenen von **P**

3.4.4 Verbesserungen

In diesem Kapitel werden einige Verbesserungen für die Berechnung der qualitativen Sichtbarkeitsanfragen vorgeschlagen. Im Prinzip handelt es sich schon bei der in Satz 3.5 erarbeiteten Betrachtung von Ebenen- und Linien-Shaft um eine „Optimierung" von einer „fast richtigen" (d.h. konservativen) zur richtigen Klassifikation. In diesem Teilkapitel geht es dagegen um die Reduzierung des Aufwands zur Gewinnung dieser (korrekten) Klassifikationen.

Die Verbesserung der qualitativen Sichtbarkeitsanfrage geschieht in drei Schritten: Als erstes wird gezeigt, dass es Abhängigkeiten zwischen der Klassifikation anhand des Ebenen- und des Linien-Shafts gibt und daher nicht immer alle Klassifikationen vollständig ausgeführt werden müssen. Danach wird erklärt, wie man die BB-Hierarchie der 3D-Szene (vgl. Abb. 2.14 und Abb. 2.15) zur weiteren Beschleunigung der Klassifikationen nutzen kann. Abschließend wird diese hierarchische Klassifikation anhand des gesamten Shafts auf die hierarchische Klassifikation anhand einzelner Shaft-Elemente verallgemeinert.

3.4.4.1 Abhängigkeiten von Klassifikationen

Für eine eindeutige Klassifikation einer 3D-Entität **G** anhand eines Shafts müssen sowohl der Ebenen- und der Linien-Shaft betrachtet werden. Mit dem naiven Ansatz, **G** nacheinander anhand des Ebenen- und Linien-Shaft zu klassifizieren, ignoriert man aber zwei offensichtliche Abhängigkeiten zwischen Ebenen- und Linien-Shaft:

Der Shaft **S(A, B)** besteht aus Shaft-Ebenen und Shaft-Linien, wobei jede Shaft-Ebene von zwei Shaft-Linien eingeschlossen wird und umgekehrt von jeder Shaft-Linie zwei Shaft-Ebenen ausgehen. Anschaulich besteht der Shaft also aus einer Liste **S** von **n** Shaft-Elementen s_i, in der immer abwechselnd Shaft-Linien $s_{2j} = l_j$ und Shaft-Ebenen $s_{(2j+1) \bmod n} = e_j$ $(0 \leq j \leq n/2)$ vorkommen (vgl. Abb. 3.33). Dann gilt:

- Wenn **G** eine Shaft-Ebene e_j nicht schneidet, so kann **G** auch keine der zu e_j adjazenten Shaft-Linien l_j und l_{j+1} schneiden (vgl. oberer Teil von Abb. 3.34)
- Wenn **G** eine Shaft-Linie l_j schneidet, so schneidet **G** auch beide zu l_j adjazenten Shaft-Ebenen e_{j-1} und e_j (vgl. unterer Teil von Abb. 3.34)

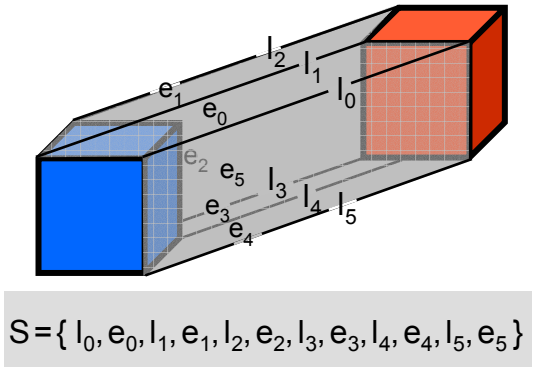


Abb. 3.33
Shaft ist abwechselnd aus Shaft-Ebenen und –Linien aufgebaut

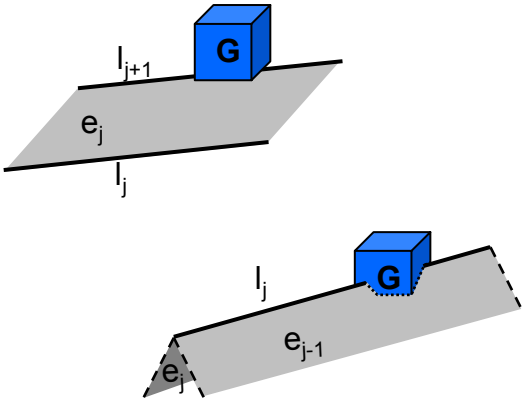


Abb. 3.34
Abhängigkeiten bei der Klassifikation anhand von Shaft-Ebenen und Linien

Um dies auszunutzen, muss jede Shaft-Ebene und -Linie ein Datenfeld **`state`** mit dem aktuellen Klassifikationsergebnis speichern. Dieses Datenfeld wird mit dem Platzhalterwert **`UND`** (*undefiniert*) initialisiert. Bei jeder Klassifikation anhand eines Shaft-Elements wird dann abgefragt, ob für dieses Shaft-Element schon eine gültige Klassifikation (**`state ≠ UND`**) vorliegt oder ob diese explizit berechnet werden muss. Bei expliziter Berechnung werden die Klassifikationen der adjazenten Elemente angepasst:

- Shaft-Ebene e_j nicht geschnitten: $e_j.state \neq PAR \Rightarrow l_j.state = l_{j+1}.state = OUT$
- Shaft-Linie l_j geschnitten: $l_j.state = OCC \Rightarrow e_{j-1}.state = e_j.state = PAR$

Diese Verbesserung beseitigt lediglich einige unnötige Klassifikationen, die Komplexität des gesamten Algorithmus ändert sich dadurch nicht. An dieser Stelle setzt nun der zweite Schritt der Optimierung an.

3.4.4.2 Shaft-Klassifikationen von Bounding-Box-Hierarchien

In Kap. 2.3.2 wurde erklärt, dass die 3D-Szene als BB-Hierarchie vorliegt, d.h. jede 3D-Entität wird von einer B-Box umschlossen und diese B-Boxen sind rekursiv zu einer Baumstruktur zusammengefasst (vgl. Abb. 2.14 und Abb. 2.15).

Für zwei Extremal-Entitäten **A** und **B** sowie eine B-Box $\mathbf{B_{CD}}$, die zwei Kind-Boxen $\mathbf{B_C}$ und $\mathbf{B_D}$ enthält, gilt (vgl. Abb. 3.35):

$$\mathbf{g(A, B, B_{CD}) = OUT \Rightarrow g(A, B, B_C) = g(A, B, B_D) = OUT}$$

$$\mathbf{\wedge\ g(A, B, B_{CD}) = INS \Rightarrow g(A, B, B_C) = g(A, B, B_D) = INS}$$

d.h. wenn eine B-Box vollständig außerhalb (innerhalb) des Shafts **S(A, B)** liegt, so liegen auch alle ihre Kind-Boxen außerhalb (innerhalb) des Shafts.

Eine bei der Wurzel beginnende Pre-Order-Traversierung der BB-Hierarchie muss also in jedem Knoten **B** nur fortgesetzt werden, wenn **B** als **PAR** oder **OCC** klassifiziert wird. Wird **B** als **INS** klassifiziert, kann der gesamte Teilbaum **B** in die Liste der Generator-Objekte übernommen werden. Für die Klassifikation **OUT** kann die Traversierung des Teilbaums **B** abgebrochen werden.

Zu beachten ist, dass eine als **OCC** klassifizierte B-Box die Sichtbarkeit zwischen den Extremal-Entitäten nicht vollständig verdeckt, diese Aussage gilt nur für 3D-Objekte.

3.4.4.3 Zwischenspeicherung von Klassifikationen

Als letzten Schritt der Optimierung wird nun die Klassifikation der B-Boxen bzw. 3D-Objekte anhand des gesamten Shafts auf die Klassifikation anhand einzelner Shaft-Elemente erweitert. Dies ist in Abb. 3.36 veranschaulicht: Die B-Box $\mathbf{B_{CD}}$ wird anhand des Shafts **S(A, B)** als **PAR** klassifiziert. Bezüglich fünf der sechs Shaft-Ebenen liegt $\mathbf{B_{CD}}$ im negativen Halbraum (d.h. „im“ Shaft“), nur die sechste Ebene wird von **G** geschnitten. Weiterhin wird keine der sechs Shaft-Linien geschnitten.

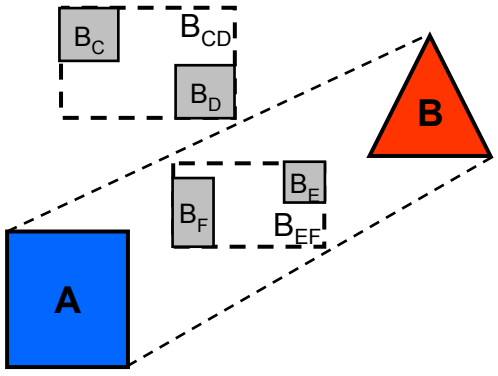


Abb. 3.35
Klassifikation von $\mathbf{B_{CD}}$ impliziert Klassifikationen von $\mathbf{B_C}$ und $\mathbf{B_D}$

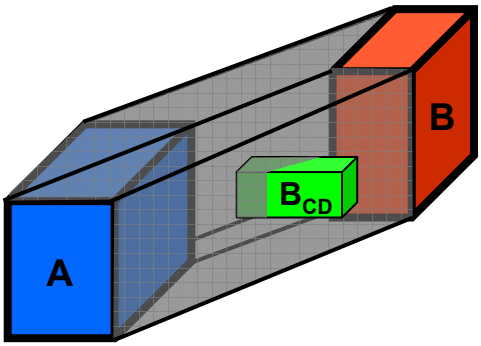


Abb. 3.36
Klassifikation von **B** ist für fünf von sechs Shaft-Ebenen eindeutig

Mit dem bisherigen Ansatz werden nun im nächsten Schritt die Kind-Boxen $\mathbf{B_C}$ und $\mathbf{B_D}$ anhand jeweils aller sechs Shaft-Ebenen und -Linien klassifiziert, obwohl das Ergebnis für fünf von sechs Shaft-Ebenen und alle sechs Shaft-Linien bereits feststeht.

Um diese redundanten Klassifikationen zu vermeiden, wird die Verwendung des Datenfelds **`state`**, das die Klassifikationen anhand einzelner Shaft-Elemente speichert, auf BB-Hierarchien verallgemeinert:

Um eine BB-Hierarchie anhand einer Shaft-Ebene **e** zu klassifizieren, muss die BB-Hierarchie von der Wurzel **W** aus in Pre-Order-Reihenfolge traversiert werden. Für die Wurzel **W** der BB-Hierarchie muss das Klassifikationsergebnis **PAR** lauten, da die Wurzel die gesamte 3D-Szene umfasst. Die Klassifikation, d.h. der Inhalt des Datenfelds **`e.state`**, kann sich entlang jedes Pfades von der Wurzel zu einem Blatt genau einmal von **PAR** nach **NEG** oder **POS** ändern.

Dies ist in Abb. 3.39 und Abb. 3.40 dargestellt: Abb. 3.39 zeigt eine BB-Hierarchie als Baum und einen Pfad $\mathbf{P = (W,\ldots,B_{CD},B_C,\ldots,B_0)}$ von der Wurzel **W** zu einem Blatt $\mathbf{B_0}$. Auf dem Teilpfad $\mathbf{(B_{CD}, B_C)}$ ändert sich das Datenfeld **`e.state`** von **PAR** nach **NEG**. Abb. 3.38 zeigt den Teilpfad $\mathbf{(B_{CD}, B_C)}$ in Form der B-Boxen: $\mathbf{B_{CD}}$ schneidet die Ebene und wird als **PAR** klassifiziert, $\mathbf{B_C}$ liegt „unter“ der Ebene und wird als **NEG** klassifiziert.

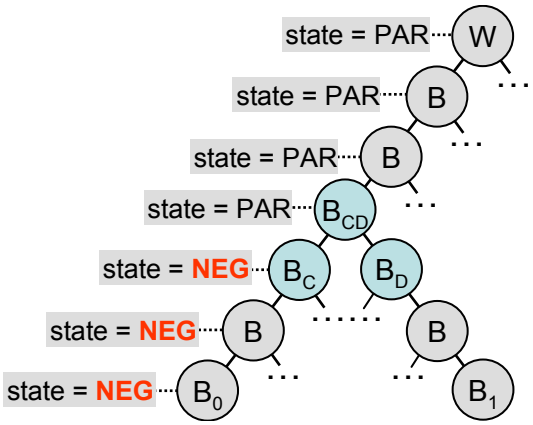


Abb. 3.37
Baumdarstellung der BB-Hierarchie mit Pfad $\mathbf{(W,\ldots,B_{CD},B_C,\ldots,B_0)}$ und Datenfeld **`state`**

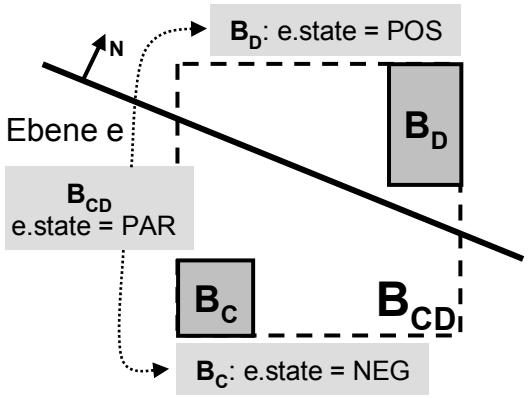


Abb. 3.38
Datenfeld **`state`** ändert sich auf den Teilpfaden $\mathbf{(B_{CD}, B_C)}$ und $\mathbf{(B_{CD}, B_D)}$

Ist die Klassifikation des Teilbaums B_C abgeschlossen, muss der Teilbaum B_D klassifiziert werden. Zuvor muss aber das Datenfeld **e.state** wieder auf **PAR** gesetzt werden, da für den Teilbaum B_D nicht notwendig **e.state = NEG** gilt.

Diese Änderung der Variablen **state** kann mit einem Zähler verfolgt werden. Für eine Shaft-Ebene **e** und eine B-Box **B** werden die Werte des Zählers **c** wie folgt interpretiert:

$c < 0 \rightarrow B \subset e^-$	**(state = NEG)**
$c = 0 \rightarrow (B \cap e^- \neq \emptyset) \wedge (B \cap e^- \neq \emptyset)$	**(state = PAR)**
$c > 0 \rightarrow B \subset e^+$	**(state = POS)**

In jedem Schritt der Traversierung wird der Zähler dann wie folgt angepasst:

Klassifikation **NEG** → `c = c - 1`
Klassifikation **PAR** → --- *(keine Änderung)*
Klassifikation **POS** → `c = c + 1`

Der Zähler zählt, wie oft B-Boxen als **NEG** oder **POS** klassifiziert wurden. Damit nach Traversierung des Teilbaums B_C wieder **c = 0** gilt, muss der Zähler auch auf dem "Rückweg" durch die Hierarchie angepasst werden, d.h. das De- bzw. Inkrementieren des Zählers muss rückgängig gemacht werden.

Dazu muss man nicht erneut die Klassifikation betrachten, es reicht das Vorzeichen des Zählers aus: Ist der Zähler negativ, so wurde die B-Box auf dem „Hinweg" durch die BB-Hierarchie als **NEG** klassifiziert und der Zähler dekrementiert. Also muss der Zähler auf dem „Rückweg" durch die BB-Hierarchie inkrementiert werden:

c < 0 → **c = c + 1**
c = 0 → --- *(keine Änderung)*
c > 0 → **c = c -1**

Dieses Vorgehen ist in Abb. 3.39 und Abb. 3.40 veranschaulicht: Abb. 3.41 zeigt die BB-Hierarchie als Baum, für jeden Knoten auf den Pfaden $P_0 = (W,...,B_{CD},B_C,...,B_0)$ und $P_1 = (W,...,B_{CD},B_D,...,B_1)$ sind die Werte des Zähler **c** angegeben. Abb. 3.42 zeigt die Änderung des Zählers auf den Teilpfaden (B_{CD}, B_C) und (B_{CD}, B_D).

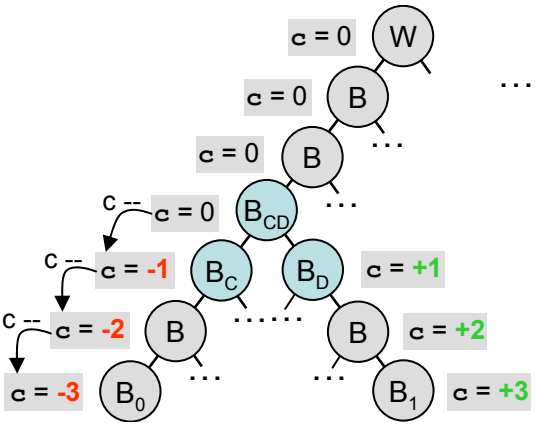


Abb. 3.39
Baumdarstellung der BB-Hierarchie mit Pfad **(W,…,B_{CD},B_C,…,B_0)** und Zähler **c**

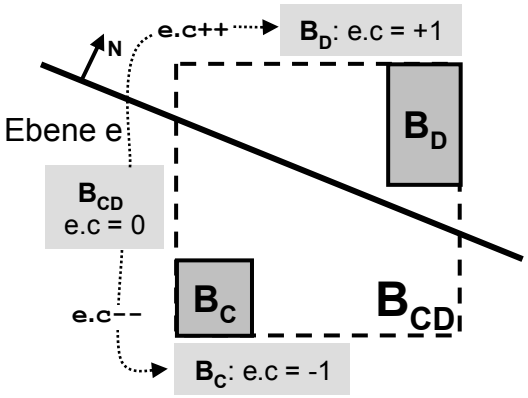


Abb. 3.40
Zähler **c** ändert sich auf den Pfaden **(B_{CD}, B_C)** und **(B_{CD}, B_D)**

Das folgende Code-Beispiel zeigt die Klassifikation einer B-Box **b** anhand einer Shaft-Ebene mit Zähler:

```
class CounterShaftPlane extends ShaftPlane
     int c = 0 // Klassifikationszähler

     int classify(BBox b)
       if (c < 0) c--   return NEG
       if (c > 0) c++   return POS
       int state = plane.classify(b)
       if (state == NEG)      c--
       else if (state == POS) c++
       return state

     void popClassify()
       if (c < 0)       c++
       else if (c > 0)  c--
```

Code-Beispiel 3.1

Bei jedem Aufruf der Methode **classify** wird der Zähler **c** betrachtet: Nur für **c = 0** wird die Klassifikation explizit berechnet, für **c ≠ 0** wird der Zähler de- bzw. inkrementiert und das Vorzeichen des Zählers als Klassifikation zurückgeliefert. Die Methode **popClassify()** macht diese Änderung des Zählers wieder rückgängig: Wurde **b** als **NEG** klassifiziert und **c** dekrementiert, ist **c** hinterher negativ. Um die Klassifkation von **b** rückgängig zu machen, muss **c** also inkrementiert werden.

Auf diese Weise wird die Klassifikation anhand einzelner Shaft-Elemente nur dann berechnet, wenn sie aus den letzten Klassifikationen noch nicht eindeutig vorliegt.

Für Shafts mit „sehr vielen“ Shaft-Elementen ergibt sich auch mit dieser Optimierung noch ein Nachteil: in jedem Klassifikationsschritt muss für alle **n** Shaft-Elemente die Methode **`classify`** aufgerufen werden. Auch wenn diese durch den obigen Ansatz optimiert wurden bleibt der Aufwand bei **O(n)**. Ein besserer Ansatz muss in jedem Klassifikationsschritt die Methode **`classify`** nur für die Shaft-Elemente aufrufen, die noch nicht eindeutig klassifiziert sind.

Dazu kann man für die Shaft-Ebenen wie folgt vorgehen: Die Liste **E** der Shaft-Ebenen wird in drei Abschnitte **NEG**, **PAR** und **POS** eingeteilt. In jedem Klassifikationsschritt muss die aktuelle B-Box **B** nur anhand der Shaft-Ebenen im **PAR**-Abschnitt der Liste neu klassifiziert werden. Wird **B** dann anhand einer dieser Ebenen als **NEG** bzw. **POS** klassifiziert, wird diese Ebene in den **NEG**- bzw. **POS**-Abschnitt der Liste verschoben.

Die drei Abschnitte der Liste werden mittels zweier Indices **`parStart`** und **`parEnd`** verwaltet: der **NEG**-Bereich geht von **`0`** bis **`parStart-1`**, der **PAR**-Bereich von **`parStart`** bis **`parEnd`** und der **POS**-Bereich von **`parEnd`** bis **`n-1`**.

An der Wurzel der BB-Hierarchie ist der **NEG**- und **POS**-Abschnitt der Liste leer, da alle Shaft-Ebenen die Wurzel schneiden, d.h. **`parStart=0`** und **`parEnd=n`**.

In jedem Klassifikationsschritt auf einem Pfad von der Wurzel zu einem Blatt der BB-Hierarchie kann sich der Wert von **`parStart`** bzw. **`parEnd`** durch Verschiebung von Shaft-Elementen in den **NEG-** bzw. **POS-**Abschnitt der Liste ändern. Dies wird mittels zweier Stacks verwaltet: vor jedem Klassifikationsschritt werden die Werte von **`parStart`** und **`parEnd`** auf die Stacks gelegt und auf dem „Rückweg“ durch die BB-Hierarchie werden die Werte wieder vom Stack genommen.

Die Verschiebungen von Shaft-Elementen in den **NEG**- bzw. **POS**-Abschnitts der Liste müssen dabei nicht rückgängig gemacht werden: es reicht aus, dass der **NEG**- bzw. **POS**-Abschnitt der Liste kleiner und der **PAR**-Abschnitt damit größer wird, die Position des Shaft-Elements innerhalb des **PAR**-Abschnitts spielt keine Rolle.

Mit dieser Optimierung sinkt der Aufwand in jedem Klassifikationsschritt von **O(n)** auf **$O(n_{PAR})$** mit **$n_{PAR} = parEnd - parStart$**.

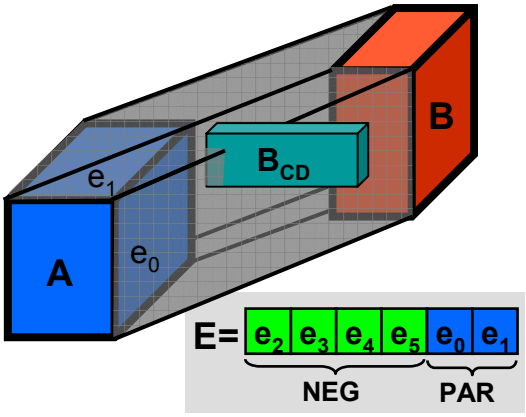


Abb. 3.41

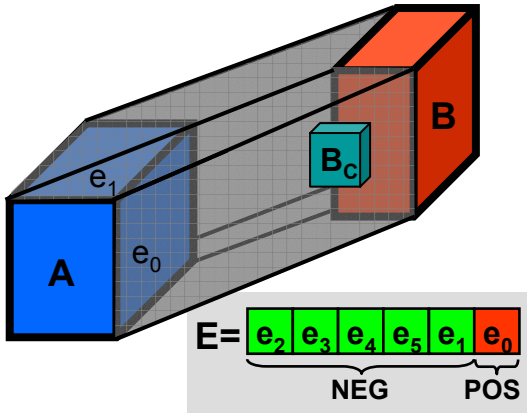


Abb. 3.42

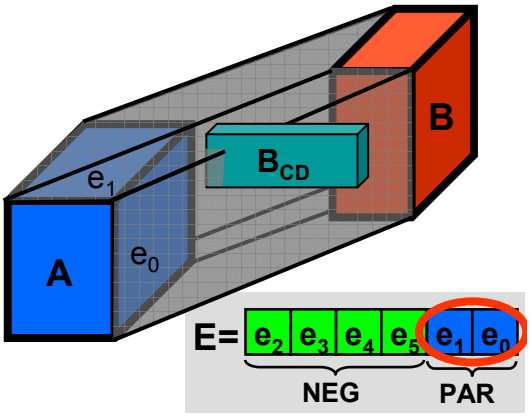


Abb. 3.43

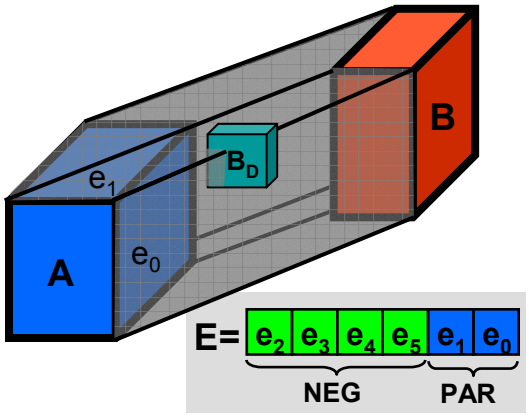


Abb. 3.44

Dies ist in Abb. 3.41– Abb. 3.44 veranschaulicht: In Abb. 3.41 wurde die B-Box $\mathbf{B_{CD}}$ als **PAR** klassifiziert. Bezüglich vier der sechs Shaft-Ebenen liegt $\mathbf{B_{CD}}$ im negativen Halbraum, zwei Shaft-Ebenen $\mathbf{e_0}$ und $\mathbf{e_1}$ werden geschnitten. Dies zeigt sich auch in den **NEG**- und **PAR**-Abschnitten der ebenfalls angegebenen Liste **E**.

In Abb. 3.42 wurde die Traversierung mit der Kind-Box $\mathbf{B_C}$ von $\mathbf{B_{CD}}$ fortgesetzt. Diese Box muss nur bezüglich der Shaft-Ebenen $\mathbf{e_0}$ und $\mathbf{e_1}$ im **PAR**-Abschnitt neu klassifiziert werden. Bezüglich der Shaft-Ebene $\mathbf{e_0}$ wird $\mathbf{B_C}$ als **POS** klassifiziert und somit in dem **POS**-Abschnitt der Liste verschoben.

Abb. 3.43 zeigt den Stand nach der Klassifikation von $\mathbf{B_C}$ aber vor der von $\mathbf{B_D}$: Die Anordnung der Shaft-Elemente entspricht der aus Abb. 3.42, Start- und End-Index des **PAR**-Abschnitts denen aus Abb. 3.41. Dies macht jedoch keinen Unterschied für die in Abb. 3.44 dargestellte Klassifikation von $\mathbf{B_D}$: es reicht aus, dass die Shaft-Ebene $\mathbf{e_0}$ im **PAR**-Abschnitt der Liste steht, wo spielt keine Rolle.

Zusammenfassend wurden also folgende Optimierungen durchgeführt:

- Abhängigkeiten zwischen Klassifikationen anhand verschiedener (Typen von) Shaft-Elementen wurden ausgenutzt. Somit werden für einen Shaft **S(A, B)** und eine 3D-Entität **G** nur so viele Element-Klassifikationen wie unbedingt nötig durchgeführt.
- Statt alle 3D-Entitäte nacheinander anhand des Shafts zu klassifizieren, wurde die gesamte BB-Hierarchie klassifiziert. Alle Teilbäume, die vollständig innerhalb oder außerhalb des Shafts liegen, müssen dabei nicht weiter klassifiziert werden.
- Diese Ausnutzung der BB-Hierarchie wurde auf die einzelnen Shaft-Elemente erweitert, so dass die zu klassifizierenden Teilbäume bzw. 3D-Entitäten nur anhand der Shaft-Elemente neu klassifiziert werden, für die noch kein eindeutiges Ergebnis vorliegt.

  Die wurde für Shaft mit „wenigen“ Shaft-Elementen durch einen einfache Zähler und für Shafts mit „vielen“ Shaft-Elementen durch Umordnungen der Shaft-Element-Listen realisiert.

3.4.5 Experimente und Messungen

Abschließend soll noch der Aufwand der qualitativen Sichtbarkeitsanfragen betrachtet werden. Da sich theoretische Aussagen hier nur als grobe Abschätzungen treffen lassen, wird der Aufwand anhand beispielhafter Messungen betrachtet. Dazu wurden jeweils 10 – 1000 zufällige 3D-Objekte anhand von jeweils 100 zufälligen Shafts mit jeweils 32 Shaft-Elementen (16 Shaft-Ebenen und 16 Shaft-Linien) klassifiziert.

Man beachte, dass diese Messungen allein nur wenige Rückschlüsse auf den Aufwand bei Rückkopplung mit dem Radiosity-Verfahren(vgl. Kap. 3.1) erlauben. Da die in Kap. 3.5 und 4 diskutierten exakten Verfahren jedoch nicht mit dem Radiosity-Verfahren rückgekoppelt werden können, wird zunächst nur der Aufwand für einzelne qualitative Sichtbarkeitsanfragen betrachtet.

Dazu werden zunächst die Auswirkungen der konservativen und nicht-konservativen Klassifikationen aus Kap. 3.4.3 betrachtet. Dann werden die Verbesserungen durch die drei Schritte aus Kap. 3.4.4 untersucht.

Als erstes wird betrachtet, wie sich die konservativen Fehlklassifikationen auswirken (vgl. Abb. 3.45) . Dazu wird verglichen, wie viele 3D-Objekte bei…

- konservativer Klassifikation anhand der Shaft-Ebenen (Kurve **k-E**)
- konservativer Klassifikation anhand der Shaft-Linien (Kurve **k-L**)
- nicht-konservativer Klassifikation anhand der Shaft-Ebenen/–Linien (Kurve **nk**)

als **OCC**, **PAR** oder **INS** (d.h. „im“ Shaft) klassifiziert wurden. Es ist zu erkennen, dass bei den konservativen Klassifikationen deutlich mehr Objekte als „im“ Shaft klassifiziert werden, d.h. dass es eine große Zahl von Fehl-Klassifikationen gibt.

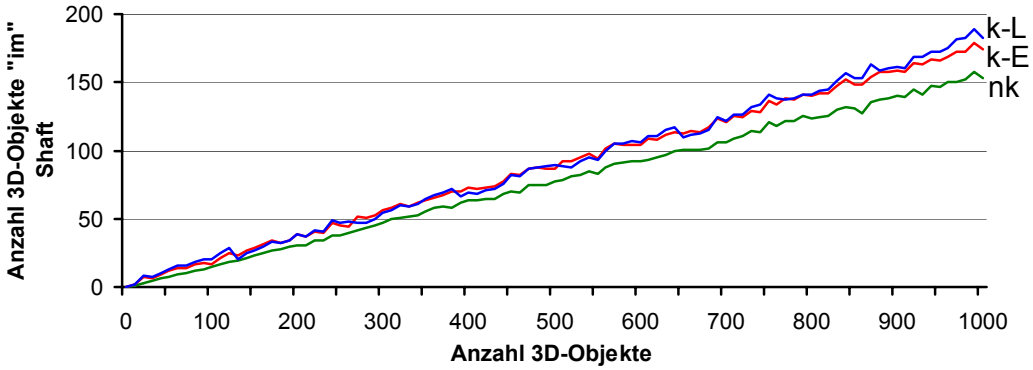


Abb. 3.45
Anzahl von 3D-Objekten „im“ Shaft bei konservativen und nicht-konservativen Klassifikationen

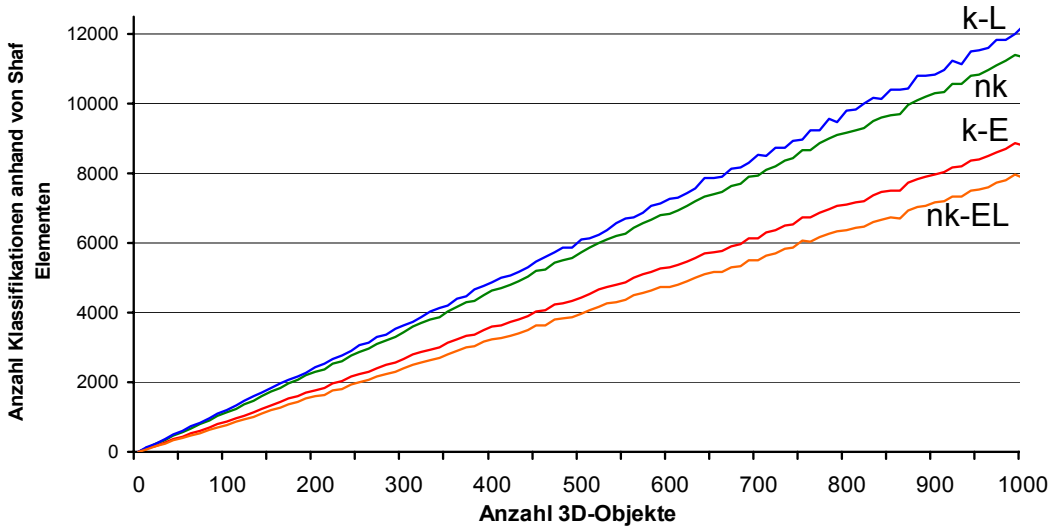


Abb. 3.46
Zahl der Klassifikationen anhand von einzelnen Shaft-Elementen

In Abb. 3.46 ist dargestellt, wie viele Klassifikationen anhand von einzelnen Shaft-Elementen notwendig waren, um Klassifikationen anhand des ganzen Shafts zu gewinnen. Die Kurve für die nicht-konservative Klassifikation (**nk**) liegt zwischen den Kurven für die beiden konservativen (**n-E** und **n-L**). Dies bedeutet, dass die nicht-konservative Klassifikation anhand des vollständigen Shafts teurer als die konservative Klassifikation anhand des Ebenen-Shafts, aber billiger als die konservative Klassifikation anhand des Linien-Shafts ist.

Durch zusätzliche Ausnutzung der Abhängigkeiten zwischen den Klassifikationen (Kurve **nk-EL**) anhand der Shaft-Ebenen und –Linien ist die ***nicht-konservative Klassifikation anhand des vollständigen Shafts billiger als die beiden konservativen Klassifikationen***!

Nun sollen die eigentlichen Optimierungen betrachtet werden: Ausnutzung der BB-Hierarchie mit und ohne Zwischenspeicherung von Klassifikationen.

In Abb. 3.47 und Abb. 3.48 zeigen die Kurven die Zahl der Klassifikationen bei Verwendung von:

- Objekt-Liste (Kurve **OL**)
- BB-Hierarchie (Kurve **BBH**)
- BB-Hierarchie mit Zwischenspeicherung von Klassifikationen (Kurve **ZSK**)

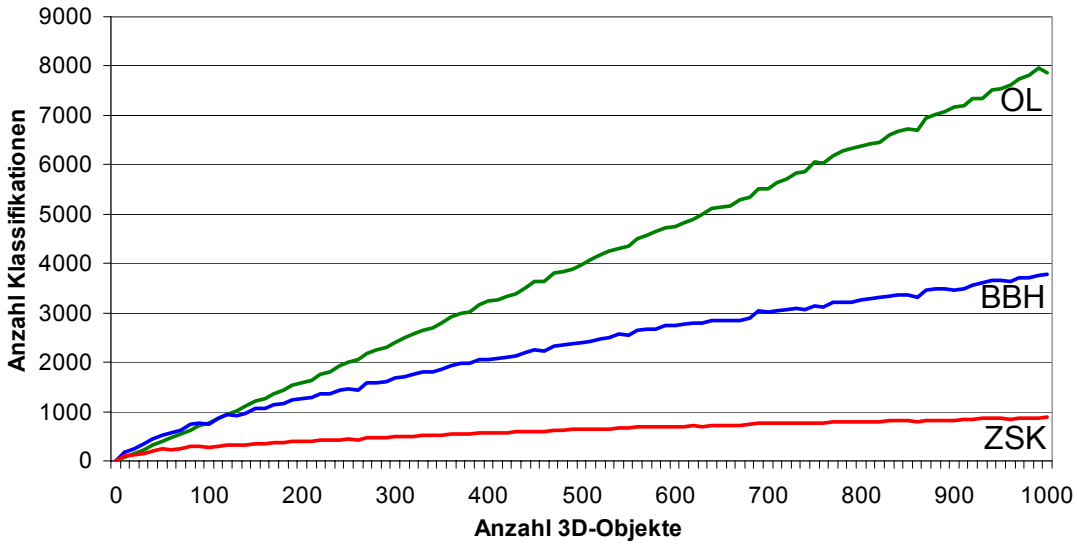


Abb. 3.47
Anzahl Klassifikationen bei Objekt-Liste und BB-Hierarchie mit und ohne Zwischenspeicherung

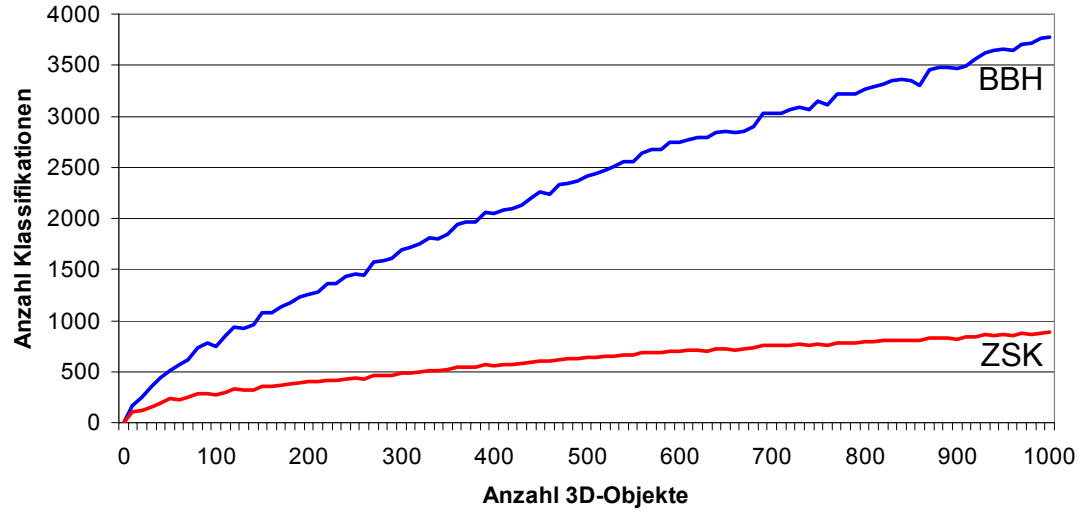


Abb. 3.48
Anzahl Klassifikationen BB-Hierarchie mit und ohne Zwischenspeicherung (Ausschnitt aus Abb. 3.47)

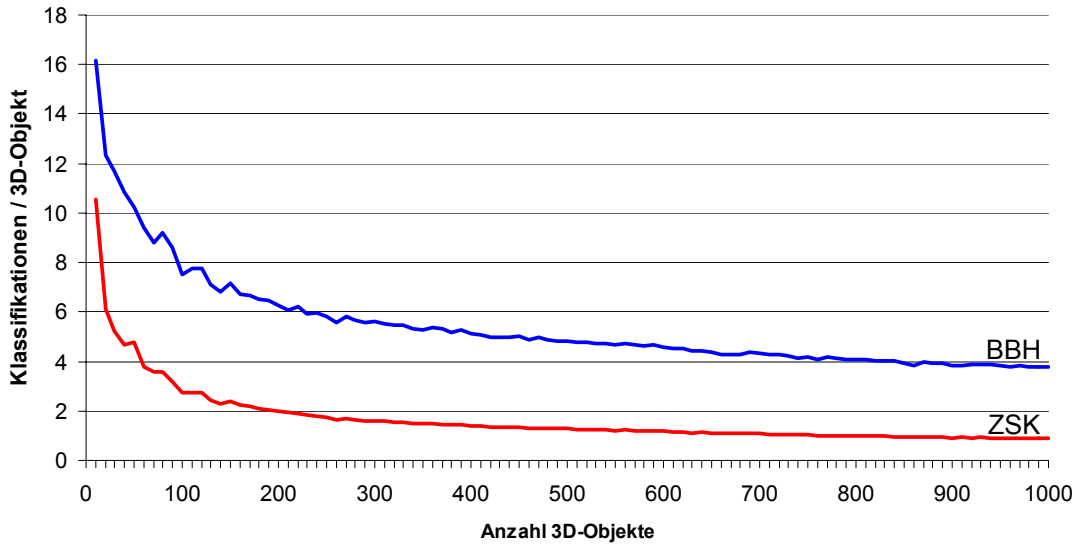


Abb. 3.49
Verhältnisse **BBH/OL** und **ZSK/BBH** der Kurven **OL**, **BBH** und **ZSK**

Man erkennt zunächst die sehr große Einsparung von Klassifikationen durch Ausnutzung der BB-Hierarchie: Während die Kurve **OL** im Wesentlichen linear verläuft, sind die Kurven **BBH** und **ZSK** deutlich sublinear. Weiterhin ist zu erkennen, dass durch die Zwischenspeicherung von Klassifikationen nochmals eine große Zahl von Klassifikationen eingespart werden kann.

In Abb. 3.49 ist die Anzahl der Klassifikationen pro 3D-Objekte dargestellt. Mit zunehmender Zahl von 3D-Objekten fallen beide Kurven ab, wobei die Kurve **ZSK** deutlich unterhalb der Kurve **BBH** verläuft.

3.4.6 Zusammenfassung

In diesem Kapitel wurden Näherungsverfahren zur Berechnung der globalen Sichtbarkeit in 3D-Szenen untersucht. Der Schwerpunkt lag hierbei auf der Betrachtung der qualitativen Sichtbarkeitsanfrage, diese liefert qualitative Informationen über die gegenseitige Sichtbarkeit von Extremal-Elementen.

Zur Umsetzung dieser Sichtbarkeitsanfrage wurden Shafts zwischen 3D-Entitäten betrachtet: jedes 3D-Objekt „im“ Shaft zweier Extremal-Entitäten beeinflusst die Sichtbarkeit zwischen diesen. Die Frage, ob eine 3D-Entität „in“ einem Shaft liegt, wurde als Klassifikation anhand des Ebenen- und Linien-Shafts formuliert.

Ein Problem dieser Klassifikation war, dass sie zunächst nur konservative, d.h. nicht notwendig korrekte, Ergebnisse liefert. Daher wurde die konservative Klassifikation anhand des Ebenen- oder Linien-Shafts zur nicht-konservativen Klassifiaktion anhand des vollständigen Shafts erweitert.

Anschließend wurde die Klassifikation von 3D-Entitäten anhand eines vollständigen Shafts in drei Schritten verbessert:

- Durch Ausnutzung der Abhängigkeiten zwischen Klassifikationen anhand einzelner Shaft-Elemente ist die nicht-konservative Klassifikation anhand eines vollständigen Shafts billiger als die beiden konservativen Klassifikationen.
- Durch Verwendung der BB-Hierarchie statt der einfachen Objekt-Liste ist der Aufwand zur Klassifikation einer kompletten 3D-Szene anhand eines vollständigen Shafts sublinear.
- Durch Verallgemeinerung der Klassifikation anhand eines vollständigen Shafts zur Klassifikation anhand einzelner Shaft-Elemente mit Zwischenspeicherung von Klassifikationen kann nochmals eine große Zahl von Klassifikationen eingespart werden.

Damit kann man abschließend festhalten, dass die Shaft-Klassifikation einer 3D-Szene mit n 3D-Objekten mit sublinearem Aufwand möglich ist und somit eine performante Methode zur Umsetzung der qualitativen Sichtbarkeitsanfragen ist.

3.5 Exakte Verfahren

In diesem Kapitel wird ein Verfahren zur exakten Lösung des globalen Sichtbarkeitsproblems vorgestellt. Anschaulich bedeutet dies, dass die Extremal-Elemente anhand von Schattenlinien zerlegt werden. Diese Schattenlinien auf den Extremal-Elementen entstehen durch Schnitte der Extremal-Elemente mit so genannten Sichtbarkeitswechseln, d.h. Flächen im 3D, an denen sich die gegenseitige Sichtbarkeit der Extremal-Elemente „in irgendeiner Form ändert".

Ziel dieses Kapitels ist, alle Sichtbarkeitswechsel einer 3D-Szene zu konstruieren und die Extremal-Elemente anhand dieser Sichtbarkeitswechsel zu zerlegen. Es wird sich zeigen, dass diese Zerlegung die Eigenschaft hat, dass jedes Paar von (Kind-) Extremal-Elementen entweder gegenseitig vollständig (nicht) sichtbar ist oder sich die Sichtbarkeit für dieses Paar entlang der (Kind-) Extremal-Elementen linear ändert.

Da die direkte Konstruktion der Sichtbarkeitswechsel nur schwer möglich ist, betrachtet man stattdessen Schnitte von Sichtbarkeitswechseln. Da Sichtbarkeitswechsel Flächen im 3D sind, entspricht der Schnitt zweier solcher Flächen einer Linie im 3D, einem so genannten maximal freien Liniensegment. Diese maximal freien Liniensegmente können nach einem recht einfachen Verfahren mit elementaren geometrischen Methoden konstruiert werden.

Diese Schnitte von Sichtbarkeitswechseln können auch als Adjazenzen interpretiert werden: wenn durch den Schnitt zweier Sichtbarkeitswechsel $\mathbf{SW_0}$ und $\mathbf{SW_1}$ eine maximal freies Liniensegment **L** definiert wird, so kann man auch sagen, dass $\mathbf{SW_0}$ und $\mathbf{SW_1}$ adjazent zu **L** sind. Die Gesamtheit aller Adjazenzen (d.h. aller Sichtbarkeitswechsel und aller maximal freier Liniensegmente) induziert eine Graphenstruktur. Dieser Graph wird Visibility Skeleton einer 3D-Szene genannt und enthält alle globalen Sichtbarkeitsinformationen der 3D-Szene.

Soll das Visibility Skeleton im Radiosity-Verfahren genutzt werden, müssen im Verlauf des Radiosity-Verfahrens keine weiteren Sichtbarkeitsberechnungen durchgeführt werden, alle benötigten Sichtbarkeitsinformationen sind im (vorberechneten) Visibility Skeleton bzw. in der vom Visibility Skeleton induzierten Zerlegung gespeichert .

Um das Visibility Skeleton einer 3D-Szene zu konstruieren müssen also zunächst die Sichtbarkeitswechsel und maximal freien Liniensegmente definiert werden. Anschließend muss die Konstruktion der maximal freien Liniensegmente erklärt und es muss aufgezeigt werden, wie man aus diesen die Sichtbarkeitswechsel konstruieren kann. Insbesondere wird an dieser Stelle ein einfacheres Konstruktionsverfahren für die Sichtbarkeitswechsel als das in der Literatur angegebene, vorgeschlagen.

Da diese Konstruktion der maximal freien Liniensegmente sehr aufwändig ist, werden abschließend noch einige Vorschläge zur Optimierung dieser Konstruktion gemacht.

Als erstes soll an dieser Stelle jedoch ein einfaches Beispiel im 2D diskutiert werden, um die grundlegende Idee der exakten Verfahren zu verdeutlichen.

3.5.1 Ein einfaches Beispiel im 2D

In Abb. 3.50 ist eine einfache 2D-Szene dargestellt: zwischen den Extremal-Elementen **A** und **B** befindet sich ein Generator-Objekt **G** und beeinflusst somit die gegenseitige Sichtbarkeit von **A** und **B**.

In Abb. 3.51 ist die einfachste Zerlegung der Extremal-Elemente dargestellt: die „Ober-“ und „Unterseite“ von **G** wurden zu Linien verlängert und die Extremal-Elemente anhand der Schnittpunkte mit diesen Linien zerlegt. Wie man leicht sieht, gilt $\mathbf{V_{qual}(A_0, B_0) = V_{qual}(A_2, B_2) = VIS}$ und $\mathbf{V_{qual}(A_0, B_2) = V_{qual}(A_1, B_1) = V_{qual}(A_2, B_0) = OCC}$, d.h. für diese Paare müssen keine weiteren Sichtbarkeitsberechnungen durchgeführt werden.

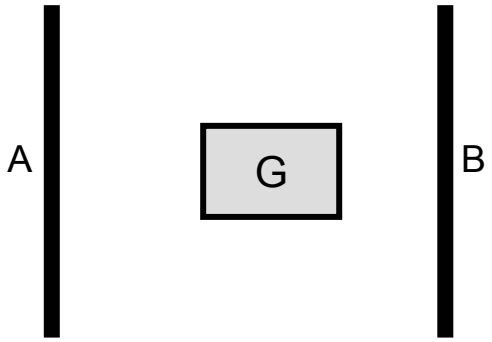


Abb. 3.50
2D-Szenen mit Extremal-Elementen **(A, B)** und Generator-Objekt **G**

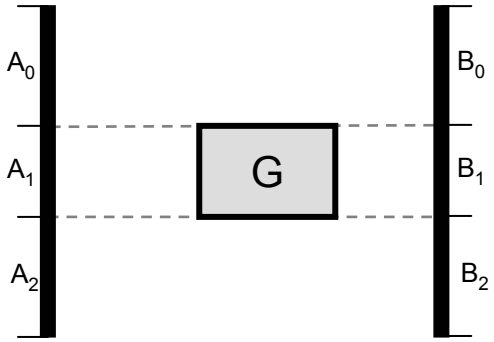


Abb. 3.51
Zerlegung der Extremal-Elemente anhand von „Ober-“ und „Unterseite“ von **G**

Es bleibt also die gegenseitige Sichtbarkeit der folgenden Paare von Kind-Elementen zu betrachten: **(A_0, B_1)**, **(A_1, B_0)**, **(A_1, B_2)** und **(A_2, B_1)**. Exemplarisch wird hier das Paar **(A_0, B_1)** betrachtet (vgl. Abb. 3.52), das Vorgehen für die anderen Paare ist analog.

Als erstes stellt sich die Frage, ob es eine Zerlegung von **B_1** gibt, so dass ein Kind-Element von **B_1** von **A_0** aus vollständig nicht sichtbar ist. In Abb. 3.53 ist diese Zerlegung dargestellt, **B_1** wurde in zwei Kind-Elemente **B_{10}** und **B_{11}** zerlegt. Diese Zerlegung wurde wieder vom Schnittpunkt einer Linie mit **B_1** induziert, diese Linie ergibt sich durch die Verbindung des Start-Vertex von **A_0** und dem Vertex **G_0** des Generator-Objekts **G**.

Für die Paare **(A_0, B_{10})** und **(A_0, B_{11})** gilt nun **$V_{qual}(A_0, B_{10}) = PAR$** und **$V_{qual}(A_0, B_{11}) = OCC$**, d.h. nur das Paar **$(A_0, B_{10})$** muss weiter betrachtet werden. Für dieses Paar kann nun das bisherige Verfahren nicht fortgesetzt werden, d.h. es gibt kein Kind-Element von **B_{10}**, von dem aus **A_0** vollständig (nicht) sichtbar ist.

Eine einfache Lösung wäre, für dieses Paar **(A_0, B_{10})** die gegenseitige Sichtbarkeit mittels der quantitativen Sichtbarkeitsanfrage **$V_{quant}(A_0, B_{10})$** aus Kap. 3.4.2 zu bestimmen bzw. bei „großen“ Extremal-Element **A_0** und **B_{10}** diese mittels des Näherungsverfahrens aus Kap. 3.4.3 zu zerlegen.

Eine bessere Möglichkeit bietet die Betrachtung der quantitativen Sichtbarkeitsanfrage **$V_{quant}(a, B_{10})$** zwischen einem Punkt **a** auf **A_0** und **B_{10}**. Es gilt (vgl. Abb. 3.53)

$$V_{quant}(A_0^{P0}, B_{10}) = 1 \text{ und } V_{quant}(A_0^{P1}, B_{10}) = 0,$$

d.h. **B_{10}** ist von **A_0^{P0}** vollständig sichtbar und von **A_0^{P1}** vollständig nicht sichtbar.

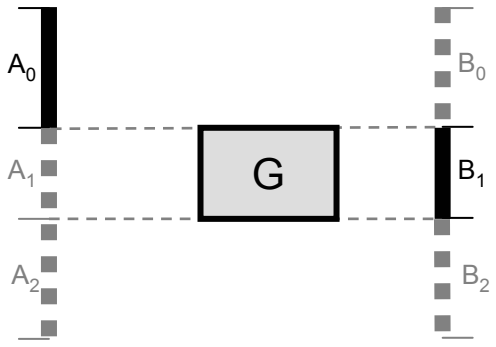


Abb. 3.52
Betrachtung des Paars **(A_0, B_1)**

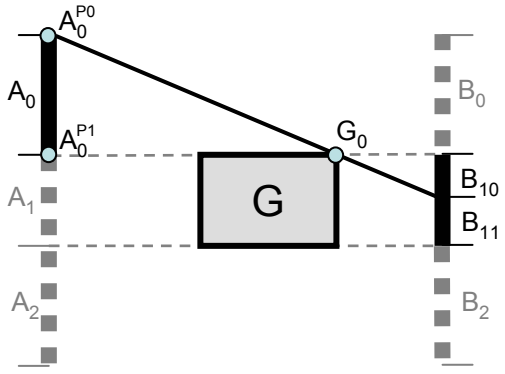


Abb. 3.53
Zerlegung von **B_1** in Kind-Elemente **B_{10}** und **B_{11}**

Für die Punkte $\mathbf{a_0(t) = A_0^{P0} + t \cdot \left(A_0^{P1} - A_0^{P0}\right)}$ ($\mathbf{t \in [0,1]}$) auf dem Liniensegment $\mathbf{A_0}$ gilt

$$\mathbf{V_{quant}(a_0(t), B_{10}) = (1-t) \cdot V_{quant}(A_0^{P0}, B_{10}) + t \cdot V_{quant}(A_0^{P1}, B_{10}) = 1 - t},$$

d.h. das Ergebnis der quantitativen Sichtbarkeitsanfrage für die inneren Punkte von $\mathbf{A_0}$ kann aus den Werten der quantitativen Sichtbarkeitsanfragen der Endpunkte von $\mathbf{A_0}$ linear interpoliert werden. Insbesondere bedeutet dies, dass $\mathbf{V_{quant}(p(t), B_{10})}$ ($\mathbf{t \in [0,1]}$) stetig und differenzierbar ist.

Betrachtet man die quantitative Sichtbarkeitsanfrage für die vollen Liniensegmente **A** und **B**, d.h. $\mathbf{V_{quant}(a(t), B)}$ mit $\mathbf{a(t) = A^{P0} + t \cdot \left(A^{P1} - A^{P0}\right)}$, so stellt man weiterhin fest, dass $\mathbf{V_{quant}(a(t), B)}$ nur in den Unterteilungstellen nicht differenzierbar ist. Dies kann man sich anhand des folgenden Beispiels klar machen:

In Abb. 3.54 ist für einen Punkt **a(t)** auf **A** der sichtbare Bereich auf **B** dargestellt. Dieser Bereich wird von zwei Linien begrenzt: die erste Linie verbindet **a(t)** mit dem Extremal-Vertex $\mathbf{B^{P0}}$ und die zweite Linie verbindet **a(t)** mit dem Generator-Vertex $\mathbf{G_0}$. Bewegt man nun den Punkt **a(t)** entlang **A** „nach unten", wird der von **a(t)** aus sichtbare Bereich von **B** immer kleiner, da sich die zweite Linie um den Generator-Vertex $\mathbf{G_0}$ „dreht" und der Schnittpunkt mit **B** somit „nach oben" wandert (vgl. Abb. 3.55).

Diese Änderung des sichtbaren Bereichs von **B** ist stetig und differenzierbar, bis **a(t)** auf gleicher „Höhe" wie $\mathbf{G_0}$ liegt. An dieser Stelle wechselt nun der „Drehpunkt" der zweiten Linie von $\mathbf{G_0}$ zu $\mathbf{G_1}$ (vgl. Abb. 3.56). Dadurch ist $\mathbf{V_{quant}(a(t), B)}$ an diesem Punkt nicht mehr differenzierbar, anschaulich ändert sich die Steigung von $\mathbf{V_{quant}(a(t), B)}$ in diesem Punkt.

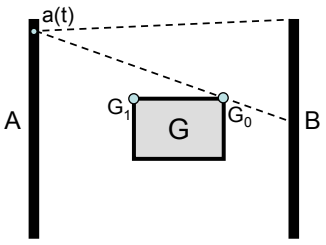


Abb. 3.54

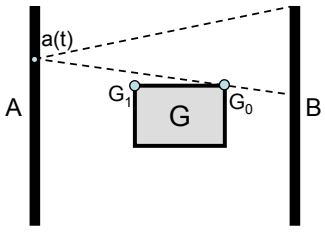


Abb. 3.55

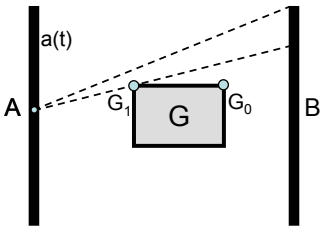


Abb. 3.56

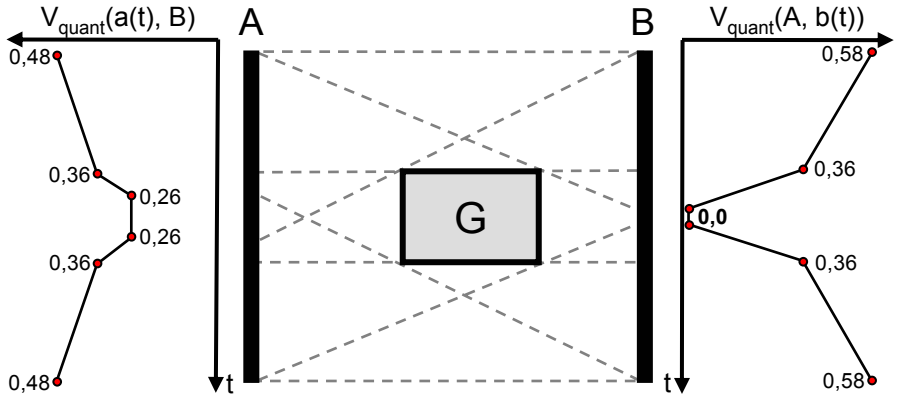


Abb. 3.57
2D-Szene mit allen Zerlegungslinien sowie Verlauf von $\mathbf{V_{quant}(a(t), B)}$ und $\mathbf{V_{quant}(A, b(t))}$

Zur Veranschaulichung dieser Aussage ist in Abb. 3.57 die 2D-Szene mit allen Zerlegungslinien dargestellt. Am linken bzw. rechten Rand ist $\mathbf{V_{quant}(a(t), B)}$ bzw. $\mathbf{V_{quant}(A, b(t))}$ aufgetragen. Wie man sieht, ändern sich die Steigungen von $\mathbf{V_{quant}(a(t), B)}$ bzw. $\mathbf{V_{quant}(A, b(t))}$ nur an den Unterteilungsstellen.

An dieser Stelle sollten nun die wesentlichen Ergebnisse aus diesem Beispiel zusammengestellt werden:

- Die Extremal-Elemente **A** und **B** wurden anhand von (Schnittpunkten mit) Linien in Kind-Elemente $\mathbf{A_i}$ und $\mathbf{B_i}$ unterteilt
- Diese Linien verbinden jeweils zwei global sichtbare Vertices der 2D-Szene
- Für jedes Paar $\mathbf{(A_i, B_j)}$ von Kind-Elementen von **A** und **B** gilt $\mathbf{V_{qual}(A_i, B_j) = VIS}$ oder $\mathbf{V_{qual}(A_i, B_j) = OCC}$ oder $\mathbf{V_{qual}(A_i, B_j) = PAR}$.
- Im Falle $\mathbf{V_{qual}(A_i, B_j) = VIS}$ oder $\mathbf{V_{qual}(A_i, B_j) = OCC}$ müssen keine weiteren Sichtbarkeitsberechnungen durchgeführt werden.
- Im Falle $\mathbf{V_{qual}(A_i, B_j) = PAR}$ gilt, dass $\mathbf{V_{quant}(A_i, B_j)}$ linear (also insbesondere stetig und differenzierbar) ist. Weiterhin können die Werte von $\mathbf{V_{quant}(A_i, B_j)}$ aus den Werten der Endpunkte linear interpoliert werden.
- An den Unterteilungsstellen sind $\mathbf{V_{quant}(a_i(t), B_j)}$ bzw. $\mathbf{V_{quant}(A, b(t))}$ nicht differenzierbar.

Diese Aussagen werden in den folgenden Teilkapiteln formalisiert und teilweise auch bewiesen. Eine vollständige Herleitung wird jedoch nicht angegeben.

Zuvor soll aber noch das schon bei den Näherungsverfahren betrachtete 3D-Beispiel (vgl. Kap. 3.4.1) auf die exakten Verfahren angewendet werden. In Abb. 3.58 ist die Situation dargestellt: die gegenseitige Sichtbarkeit der Extremal-Entitäten **A** und **B** wird von der Generator-Entität **G** beeinflusst. Weiterhin zeigt Abb. 3.58 eine von **G** verursachte Schattenlinie auf dem Polygon **B**.

In Abb. 3.59 ist die Zerlegung von **B** anhand aller Schattenlinien dargestellt. Je heller ein Kind-Polygon von **B** ist, desto „sichtbarer" ist es von (Kind-Polygonen von) **A**.

Schließlich zeigt Abb. 3.60 die Werte der quantitativen Sichtbarkeitsanfrage $\mathbf{V_{quant}(A, B_i)}$. Auch hier erkennt man wieder, dass $\mathbf{V_{quant}(A, B_i)}$ nur an den „Schattenlinien" nicht differenzierbar ist.

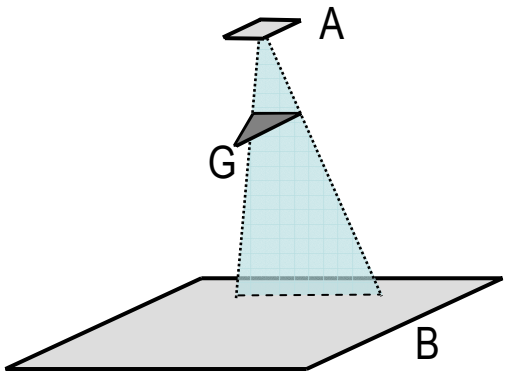


Abb. 3.58
Von **G** verursachte Schattenlinie auf **B**

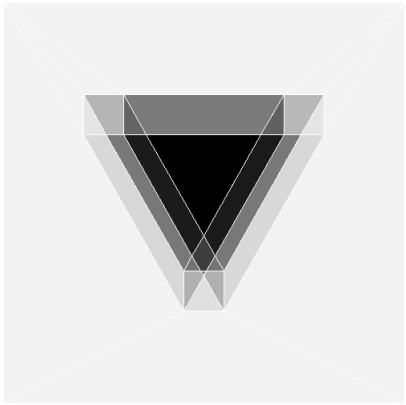


Abb. 3.59
Zerlegung von **B** anhand aller Schattenlinien

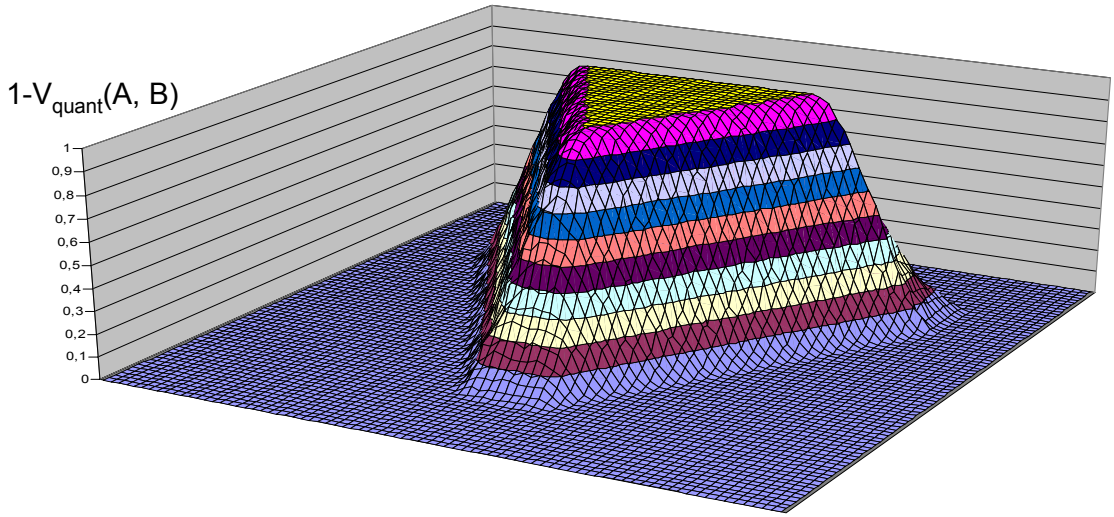


Abb. 3.60
Sichtbarkeitskoeffizient $\mathbf{V(A, B_i)}$

3.5.2 Sichtbarkeitswechsel

Am letzten Beispiel kann man die grundsätzliche Idee der exakten Verfahren erkennen: Finde alle Stellen auf den Extremal-Elementen, an denen die quantitative Sichtbarkeitsanfrage nicht unendlich oft stetig differenzierbar ist. Diese Stellen entsprechen anschaulich Schattenlinien auf den Extremal-Elementen und werden durch Schnitte der Extremal-Elemente mit 3D-Flächen (im 2D-Beispiel: Linien) induziert. Diese Flächen werden Sichtbarkeitswechsel genannt.

Definition 3.14: Sichtbarkeitswechsel

Für zwei Extremal-Elemente **A** und **B** heißen alle 3D-Flächen **F** Sichtbarkeitswechsel **n**-ter Ordnung, wenn für jedes Liniensegment $\mathbf{L(s)} \subset \mathbf{A}$ bzw. $\mathbf{L(s)} \subset \mathbf{B}$, das **F** genau einmal schneidet, gilt, dass $\mathbf{V_{quant}(L(s),\ B)}$ bzw. $\mathbf{V_{quant}(A,\ L(s))}$ im Schnittpunkt genau $\mathbf{(n-1)}$-mal stetig differenzierbar ist.

Man beachte, dass in der vorangegangenen Definition mit Flächen nicht notwendig Ebenen gemeint sind. Im Folgenden wird u.a. der **EEE**-Sichtbarkeitswechsel vorgestellt, der durch eine Fläche 2. Ordnung[4] repräsentiert wird.

Nun soll die wichtigste Eigenschaft der Sichtbarkeitswechsel aufgezeigt werden, nämlich dass Sichtbarkeitswechsel allein anhand geeignet gewählter Kanten der Generator-Entitäten konstruiert werden können.

Satz 3.6: Konstruktion von Sichtbarkeitswechseln

Ein Sichtbarkeitswechsel ist durch drei Generator-Kanten eindeutig bestimmt, d.h. er schneidet alle drei Kanten und mindestens eine in einem Liniensegment.

Beweis: Siehe [14] und [50].

Die folgende Definition zeigt die konkreten Ausprägungen von Sichtbarkeitswechseln. Der allgemeinste Fall ist der **EEE**-Sichtbarkeitswechsel, der durch drei disjunkte Kanten geht. Schneiden sich zwei oder mehr der drei Kanten, so entstehen spezielle Ausprägungen von Sichtbarkeitswechseln (siehe Erklärungen in Definition 3.15).

---

[4] Die Ordung der Fläche bezieht sich hier nicht auf die Differenzierbarkeit des Sichtbarkeitskoeffizienten, sondern auf den Grad des Polygons der impliziten Darstellung der Fläche. Ebenen sind z.B. Flächen 1. Ordung, da ihre implizite Darstellung $\mathbf{n \cdot p + d = 0}$ ein Polygon vom Grad 1 ist.

Definition 3.15: Ausprägungen von Sichtbarkeitswechseln

Man unterscheidet die folgenden vier Ausprägungen von Sichtbarkeitswechseln:

Bezeichnung	*Generator-Elemente*	*Erklärung*
EEE	3 Kanten (**E**dges)	Entspricht genau der Def. der Sichtbarkeitswechsel (vgl. Abb. 3.61)
EV	1 Kante (**E**dge), 1 **V**ertex	Ein Vertex wird durch mind. zwei Kanten bestimmt (vgl. Abb. 3.62)
Fv	1 **F**acette, 1 **V**ertex dieser Facette	Eine Facette wird durch mind. 2 Kanten bestimmt, der Vertex dieser Facette durch eine dritte Kante (vgl. Abb. 3.63)
FE	1 **F**acette, 1 Kante (**E**dge), die die Ebene dieser Facette schneidet	Analog: Facette wird durch mind. 2 Kanten definiert, die dritte Kante ist gegeben (vgl. Abb. 3.64)

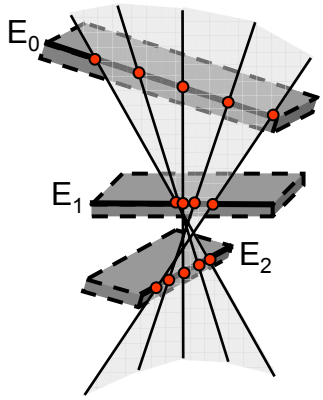


Abb. 3.61
EEE-Sichtbarkeitswechsel

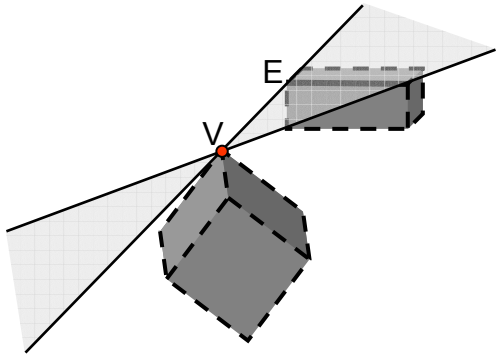


Abb. 3.62
EV-Sichtbarkeitswechsel

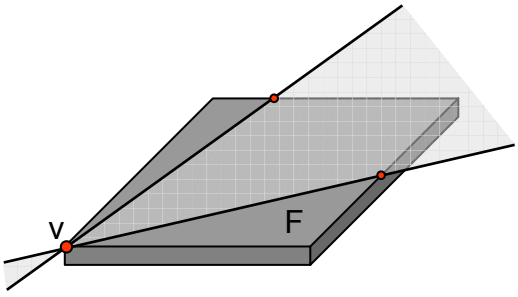


Abb. 3.63
Fv-Sichtbarkeitswechsel

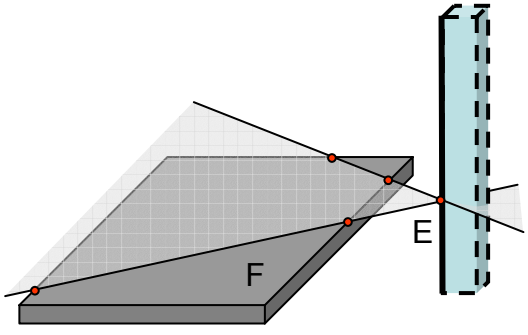


Abb. 3.64
FE-Sichtbarkeitswechsel

3.5.3 Maximale freie Liniensegmente

Will man die vorangegangenen theoretischen Ausführungen in die Praxis umsetzen, so stößt man schnell auf das Problem, dass die Ausdehnung (d.h. der „Wirkungsbereich“) der Sichtbarkeitswechsel noch nicht vernünftig definiert wurde und nur durch eine Vielzahl von Sonderfallbetrachtungen behandelt werden kann. Dazu kann man wieder ein einfaches Beispiel betrachten:

Abb. 3.65 zeigt einen **EV**-Sichtbarkeitswechsel $\mathbf{EV_A}$, der durch die End-Vertices der Kante **A** begrenzt wird. Abb. 3.66 ergänzt dies um eine zweite Kante **B**, die einen weiteren **EV**-Sichtbarkeitswechsel $\mathbf{EV_B}$ definiert. Nun werden die beiden Sichtbarkeitswechsel sowohl durch einen End-Vertex der definierenden Kante als auch durch den jeweils anderen Sichtbarkeitswechsel bzw. seine definierende Kante beschränkt.

Auch für die anderen Typen von Sichtbarkeitswechseln bzw. Kombinationen derselben ergeben sich so eine Vielzahl von analogen Beschränkungen und Nebenbedingungen (vgl. [40], und [55]). Um dies allgemein zu fassen, kann man wieder Abb. 3.66 betrachten:

Beide Sichtbarkeitswechsel werden „auf der einen Seite“ durch einen End-Vertex der Kante **A** bzw. **B** und „auf der anderen Seite“ durch die Schnittlinie **L** der beiden Sichtbarkeitswechsel beschränkt. Die Schnittlinie **L** geht dabei durch die Generator-Elemente **V**, **A** und **B**, dies entspricht genau der Vereinigung der Generator-Elemente der Sichtbarkeitswechsel $\mathbf{EV_A}$ und $\mathbf{EV_B}$.

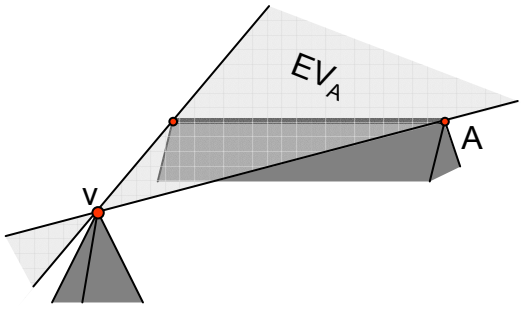


Abb. 3.65
EV-Sichtbarkeitswechsel wird durch End-Vertices der Kante **A** beschränkt

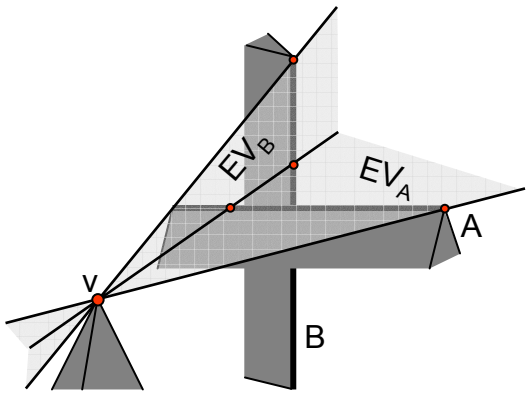


Abb. 3.66
EV-Sichtbarkeitswechsel werden durch End-Vertices der Kanten **A** und **B** sowie die durch die Kanten **A** und **B** selber beschränkt.

Die folgende Definition verallgemeinert diese beispielhaften Ausführungen:

Definition 3.16: Maximale freie Liniensegmente

Eine Linie **L**, die vier Generator-Kanten schneidet, heißt *maximal freies Liniensegment* (MFL).

Definition 3.17: Ausprägungen von maximal freien Liniensegmenten

Man unterscheidet folgende Ausprägungen von maximal freien Liniensegmenten:

Bezeichnung	*Generatoren*	*Erklärung*
VV	2 **V**ertices, die nicht zur selben Facette gehören	Vertices werden jeweils durch mind. 2 Generator-Kanten definiert (vgl. Abb. 3.67)
VEE	1 **V**ertex, 2 nicht koplanare Kanten (**E**dges)	MFL schneidet Generator-Kanten in der Reihenfolge **VEE** (vgl. Abb. 3.68)
EVE	analog **VEE**	MFL schneidet Generator-Kanten in der Reihenfolge **EVE** (vgl. Abb. 3.69)
Fvv	2 **V**ertices einer **F**acette, die keine gemeinsame Kante bilden	MFL ist eine „Diagonale“ der Facette F (vgl. Abb. 3.70)
FvE	1 **V**ertex einer **F**acette und eine Kante (**E**dge), die die Ebene der Facette schneidet	MFL ist Verbindungslinie Vertex-Schnittpunkt Kante (vgl. Abb. 3.71)
FEE	1 **F**acette, zwei Kanten (**E**dges), die die Ebene der Facette schneiden	MFL schneidet Generator-Kanten in der Reihenfolge **FEE** (vgl. Abb. 3.72)
EFE	analog **FEE**	MFL schneidet Generator-Kanten in der Reihenfolge **EFE** (vgl. Abb. 3.73)
FF	2 **F**acetten, wobei jede Facette die Ebene der anderen schneidet	MFL ist Verbindungslinie der Schnittpunkte (vgl. Abb. 3.74)
E	1 Kante (**E**dge)	MFL ist die durch Kante definierte Linie (vgl. Abb. 3.75)
E4	4 Kanten (**E**dges)	MFL entspricht genau der Definition (vgl. Abb. 3.76)

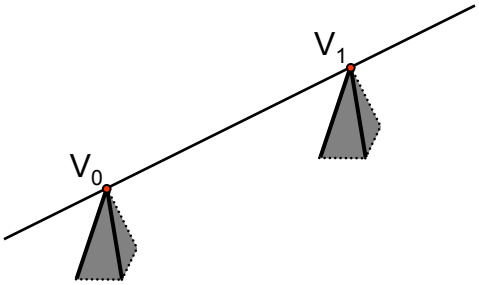


Abb. 3.67
VV-MFL

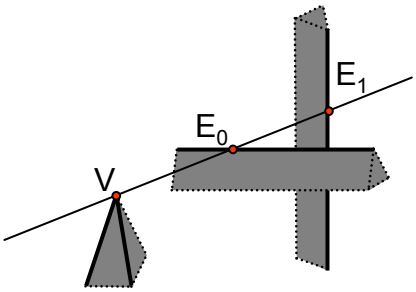


Abb. 3.68
VEE-MFL

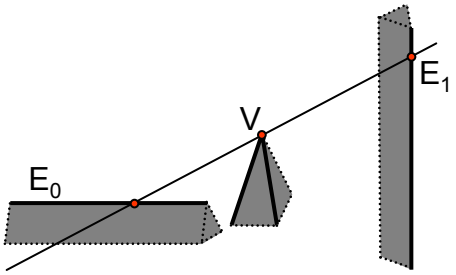


Abb. 3.69
EVE-MFL

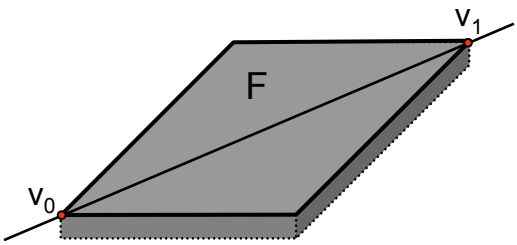


Abb. 3.70
Fvv-MFL

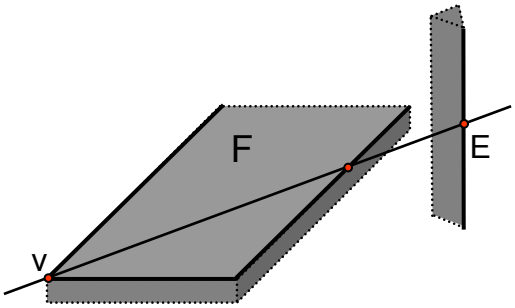


Abb. 3.71
FvE-MFL

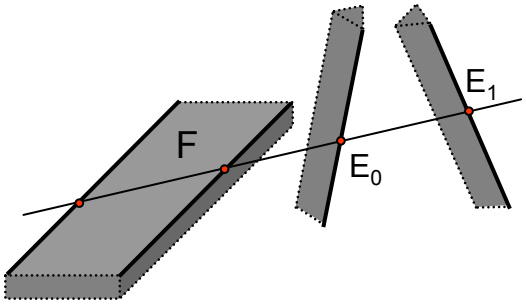


Abb. 3.72
FEE-MFL

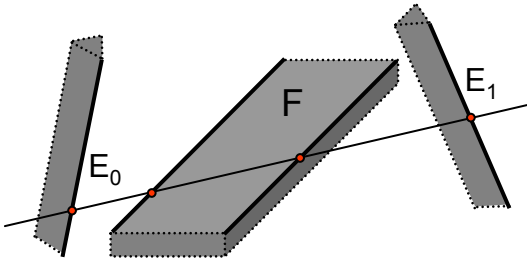


Abb. 3.73
EFE-MFL

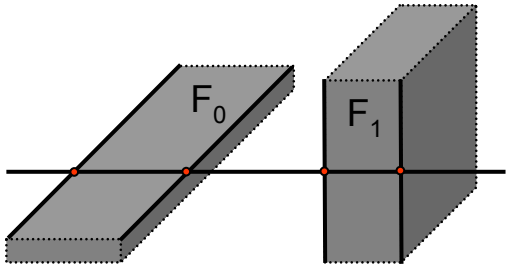


Abb. 3.74
FF-MFL

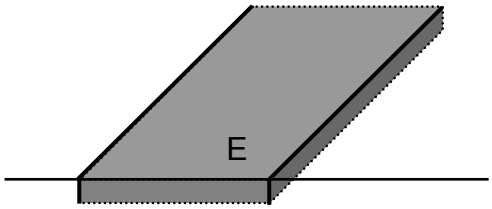


Abb. 3.75
E-MFL

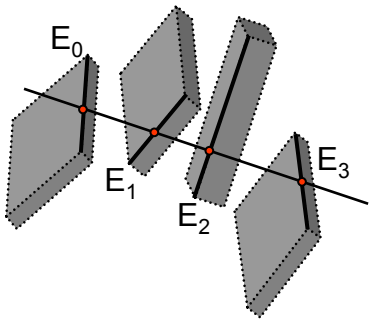


Abb. 3.76
E4-MFL

Mit dieser Definition kann der Zusammenhang zwischen Sichtbarkeitswechseln und MFLs genauer betrachtet werden. Im letzten Beispiel (vgl. Abb. 3.66) war zu erkennen, dass die **VEE**-MFL der Schnitt zweier **EV**-Sichtbarkeitswechsel ist und ihre Generator-Elemente der Vereinigung der Generator-Elemente der beiden Sichtbarkeitswechsel entsprechen.

Dies könnte nun auf beliebige Sichtbarkeitswechsel verallgemeinert werden, führt allerdings zu einer großen Zahl von Ausnahmen (z.B. Sichtbarkeitswechsel schneiden sich nicht) und Sonderfällen (z.B. Sichtbarkeitswechsel sind koplanar). Daher soll das Problem hier aus der umgekehrten Richtung betrachtet werden: Statt zu zeigen, dass Schnitte von Sichtbarkeitswechseln MFLs definieren, wird gezeigt, dass durch Variation von MFLs entlang einer oder mehrerer ihrer Generator-Kanten Sichtbarkeitswechsel entstehen:

Satz 3.7: Konstruktion von Sichtbarkeitswechseln aus maximal freien Liniensegmenten

Gegeben sei eine MFL **L** durch die Generator-Kanten $\mathbf{K = \{K_0, K_1, K_2, K_3\}}$, d.h. jede Generator-Kante $\mathbf{K_i}$ wird von **L** geschnitten.

Dann gilt, dass alle 3-elementigen Teilmengen $\{\mathbf{K_{i_0}, K_{i_1}, K_{i_2}}\} \subset \mathbf{K}$ Sichtbarkeitswechsel $\mathbf{VE_{K_{i_0} K_{i_1} K_{i_2}}}$ definieren und für alle Paare $\mathbf{(VE_1, VE_2)}$ gilt:

$$\mathbf{VE_1 \cap VE_2 = L}$$

Beweis:

Dass alle 3-elementigen Teilmengen der MFL-Generatoren Sichtbarkeitswechsel definieren, folgt aus der Definition der Sichtbarkeitswechsel. Ebenfalls trivial ist die Schnittbedingung: Die Vereinigung der Generator-Kanten jedes Paars von Sichtbarkeitswechseln ergibt wiederum **K**, also die Generator-Kanten der MFL.

In Abb. 3.77 sind die Adjazenzen der Sichtbarkeitswechsel und MFLs in Form eines Graphen dargestellt. Die Knoten entsprechen den MFLs und die Kanten den Sichtbarkeitswechseln. Dabei gilt, dass zwei MFL-Knoten durch eine Kante verbunden sind, wenn der (die) zugehörige(n) Sichtbarkeitswechsel zu den Knoten adjazent sein kann (können). Sind umgekehrt zwei Kanten (d.h. Sichtbarkeitswechsel) zu einem Knoten (d.h. MFL) adjazent, so entspricht diese MFL dem Schnitt dieser beiden Sichtbarkeitswechsel. Weiterhin gilt:

- Für eine konkrete 3D-Szene müssen bestimmte Adjazenzen nicht unbedingt vorliegen. So ist z.B. ein **EV**-Sichtbarkeitswechsel nur dann adjazent zu einem **Fvv**-MFL, wenn die Kante **E** im positiven Halbraum der Facette **F** liegt.
- Eine Kante des Graphen kann mehrere Sichtbarkeitswechsel repräsentieren. So können z.B. zu einem **VV**-MFL so viele **EV**-Sichtbarkeitswechsel adjazent sein, wie es lokal sichtbare Kanten gibt.

Weiterhin sollte noch angemerkt werden, dass selbstverständlich von jedem Knoten des Graphen noch selbstbezügliche Kanten abgehen müssten. Dies wurde aus Gründen der Übersichtlichkeit in Abb. 3.77 ausgelassen.

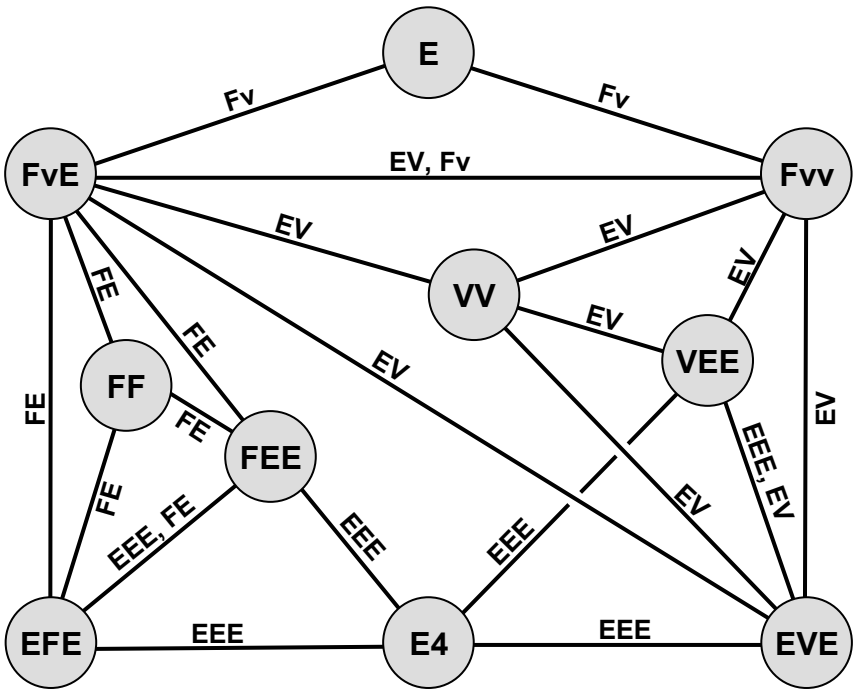


Abb. 3.77

3.5.4 Extremal-Elemente

In den bisherigen Ausführungen zu Sichtbarkeitswechseln und MFLs wurde nur die Konstruktion (d.h. die Generator-Elemente) betrachtet. Die eigentlich wichtigste Information wurde dagegen bisher nicht genannt, da sie ohne Kenntnis der MFLs nur recht umständlich zu fassen ist:

Welches sind die Extremal-Elemente, deren Sichtbarkeit durch einen Sichtbarkeitswechsel bzw. ein MFL beschrieben wird?

Für MFLs ist dies sehr einfach und macht nur eine kleine Sonderfallbetrachtung nötig:

Ein MFL ist eine Linie durch vier Generator-Kanten. Die Extremal-Elemente eines MFL sind die Elemente der Szene, die als nächstes „vor" der ersten bzw. als nächstes „hinter" der letzten Generator-Kante geschnitten werden. Ein Sonderfall ergibt sich, wenn das MFL „stumpf" auf die erste bzw. letzte Generator-Kante auftrifft, in diesem Fall sind diese Generator-Kante(n) gleichzeitig Extremal-Element(e) (vgl. Abb. 3.78).

Da Sichtbarkeitswechsel durch MFLs begrenzt werden (vgl. z.B. Abb. 3.66), können die Extremal-Elemente eines Sichtbarkeitswechsels direkt aus den Extremal-Elementen der begrenzenden MFLs abgelesen werden. Abb. 3.79 zeigt ein einfaches Beispiel: Ein $\mathbf{V_0V_1}$-MFL und ein $\mathbf{V_0E_0E_1}$-MFL schließen einen **EV**-Sichtbarkeitswechsel ein. Die Extremal-Elemente der MFLs sind die Facetten $\mathbf{F_0}$ und $\mathbf{F_1}$, also sind $\mathbf{F_0}$ und $\mathbf{F_1}$ auch die Extremal-Elemente des Sichtbarkeitswechsels.

In Kap. 3.5.6.4 werden weitere Bedingungen für die Konstruktion der Extremal-Elemente eines Sichtbarkeitswechsels aus den Extremal-Elementen der begrenzenden MFLs angegeben.

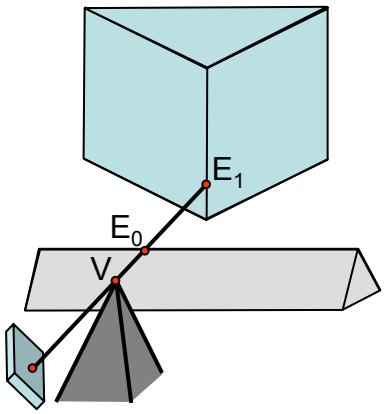


Abb. 3.78
Extremal-Element ist Generator-Element

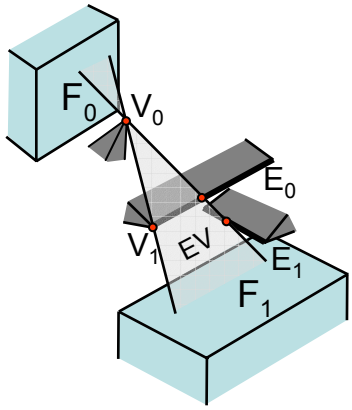


Abb. 3.79
Extremal-Elemente eines Sichtbarkeitswechsels

3.5.5 Überblick: Sichtbarkeitswechsel und maximal freie Liniensegmente

An dieser Stelle ist es sinnvoll, einen kurzen Überblick der Eigenschaften der Sichtbarkeitswechsel und maximal freien Liniensegmente zu geben. Zunächst gibt es einige Eigenschaften, die Sichtbarkeitswechseln und MFLs gemein sind:

- Sichtbarkeitswechsel und MFLs werden von Generator-Kanten aufgespannt.
- Die Benennung entspricht den geschnittenen Generator-Elementen, z.B. **VEE** für ein MFL, das einen **V**ertex und zwei Kanten (**E**dges) schneidet oder **EV** für einen Sichtbarkeitswechsel, der eine Kante (**E**dge) und einen **V**ertex schneidet.
- Sichtbarkeitswechsel und MFLs werden von Extremal-Elementen begrenzt.

Die folgende Tabelle zeigt die unterschiedlichen Eigenschaften von Sichtbarkeitswechseln und MFLs:

Eigenschaft	***Sichtbarkeitswechsel***	***Maximal freies Liniensegment***
Geometrie	3D-Fläche (nicht notwendig Ebene!)	3D-Linie
Generatoren	3 Generator-Kanten	4 Generator-Kanten
Extrema	Kurvensegmente auf Extremal-Elementen, Endpunkte der Kurvensegmente sind Extrema der adjazenten MFLs	Schnittpunkte der MFL mit den Extremal-Elementen
Adjazenzen	Sichtbarkeitswechsel entstehen durch Variation einer MFL entlang einer Generator-Kante. Ein Sichtbarkeitswechsel ist adjazent zu genau zwei MFLs.	MFLs entstehen durch Schnitte von (mindestens) zwei Sichtbarkeitswechseln. Ein MFL ist adjazent zu mindestens zwei Sichtbarkeitswechseln

Tabelle 3.4

Die wichtigste Eigenschaft der Sichtbarkeitswechsel ist, dass der Schnitt mit den Extremal-Elementen Kurvensegmente definiert, an denen die quantitative Sichtbarkeitsanfrage zwischen den Extremal-Elementen nicht unendlich oft stetig differenzierbar ist. Anschaulich entspricht dieses Kurvensegment also einer Schattenlinie.

3.5.6 Visibility Skeleton

Nach dieser notwendigerweise eher ausführlichen Einleitung kann die zentrale Datenstruktur dieses Kapitels bzw. der ganzen Arbeit definiert werden (vgl. [21], [18] und [19]): ***Das Visibility Skeleton einer 3D-Szene ist ein aus maximal freien Liniensegmenten und Sichtbarkeitswechseln aufgebauter Graph, der alle globalen Sichtbarkeitsinformationen der 3D-Szene enthält.***

Im Folgenden werden Definition und Aufbau des Visibility Skeletons diskutiert und dann Möglichkeiten zur Konstruktion der MFLs und Sichtbarkeitswechsel aufgezeigt. Abschließend werden Optimierungen für die MFL-Konstruktion vorgestellt.

3.5.6.1 Definition und Aufbau

Mit Satz 3.7 ist der Zusammenhang zwischen MFLs und Sichtbarkeitswechseln klar: Schnitte von Sichtbarkeitswechseln definieren MFLs, jedes MFL ist adjazent zu mehreren Sichtbarkeitswechsel. Diese Adjazenzen induzieren eine Graphenstruktur:

Definition 3.18: Visibility Skeleton einer 3D-Szene

Das *Visibility Skeleton* einer 3D-Szene ist ein Graph, dessen Knoten die MFLs und dessen Kanten die Sichtbarkeitswechsel der 3D-Szene sind.

Die Datenstrukturen für das Visibility Skeleton können z.B. wie folgt aussehen:

```
class VisibilitySkeleton          // Graph-Repräsentation des Visibility Skeletons
   List arc, node                 // Listen der MFLs und Sichtbarkeitswechsel
   void insertNode(Node l)        // füge MFL l in Visibility Skeleton ein
   void insertArc (Arc  a)        // füge Sichtbarkeitswechsel in Vis. Skeleton ein

class VisSkelElem                 // Oberklasse für Elemente des Graphen
   Element[] gen;                 // Array von 1-4 Generator-Elementen
   Element[] ext;                 // Array von genau zwei Extremal-Elementen
   void calcExtremities()         // Berechne Extremal-Elemente

class Node extends VisSkelElem    // Klasse für MFLs
   Arc[] arc;                     // Array von adjazenten Sichtbarkeitswechseln

class Arc extends VisSkelElem     // Klasse für Sichtbarkeitswechsel
   Node  [] node;                 // Array von adjazenten MFLs
   double[] para;                 // Parameter der adj. MFLs (siehe unten)
   void insertArc(Arc a)          // füge Sichtbarkeitswechsel a in diesen SW ein
```

Code-Beispiel 3.2

Das Datenfeld **`para`** in der Klasse **`Arc`** beschreibt die Position der im Sichtbarkeitswechsel enthaltenen MFLs. Dazu kann man Abb. 3.80 betrachten: der dargestellte **EV**-Sichtbarkeitswechsel **(v, e)** enthält vier MFLs: Die Start- und End-**VV**-MFLs und zwei **VEE**-MFLs „in der Mitte“. Dies Position dieser MFLs innerhalb des Sichtbarkeitswechsels kann hier einfach durch den Schnittparameter mit der Kante **e** beschrieben werden: der „linke“ Knoten hat Parameter 0.0, der „rechte“ Parameter 1.0 und die beiden „mittleren“ haben Parameter 0.3 bzw. 0.8.

Ähnliche Parametrisierungen findet man für alle Typen von Sichtbarkeitswechseln:

EV Schnittparameter von **L** mit Kante **E**

EEE Schnittparameter von **L** mit Kante $\mathbf{E_1}$ (d.h. „mittlere“ Kante von **EEE**)

Fv Winkel $\alpha = \angle(\mathbf{L}, \mathbf{L'})$ mit **L'** koplanar zu **F** und Aufpunkt **v**, z.B. **L'** ist Kante von **F** mit Aufpunkt **v** (vgl. Abb. 3.81)

FE Winkel $\alpha = \angle(\mathbf{L}, \mathbf{L'})$ mit **L'** koplanar zu **F** und Schnittpunkt **p** von Facetten-Ebene und **E** als Aufpunkt, z.B. $\mathbf{L' = (F.v_0 – p)}$ (vgl. Abb. 3.82)

Eine Parametrisierung anhand eines Winkels wirft zunächst Probleme auf: Winkel folgen einer Periode von $\mathbf{2\pi}$, daher kann man eigentlich keine sinnvolle Ordnung definieren. Da aber zwei MFLs innerhalb eines **Fv**- bzw. **FE**-Sichtbarkeitswechsel maximal einen Winkel von $\boldsymbol{\pi}$ **(= 180°)** aufspannen können (vgl. Abb. 3.83), wird diese Periode niemals vollständig durchlaufen.

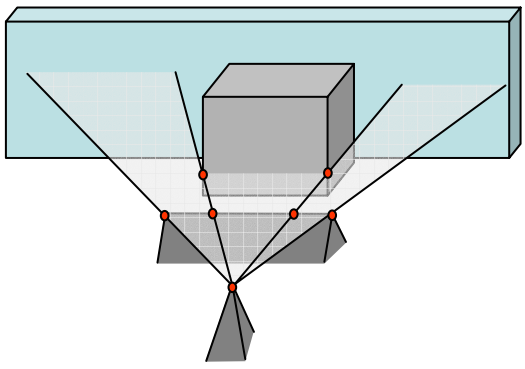


Abb. 3.80
Sichtbarkeitswechsel enthalten MFLs

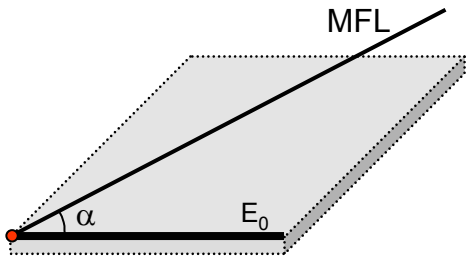


Abb. 3.81
Parametrisierung von MFLs bezüglich **Fv**-Sichtbarkeitswechsel

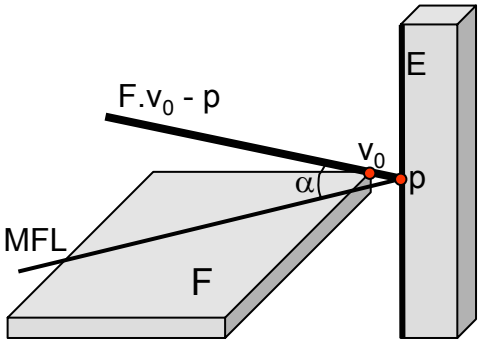


Abb. 3.82
Parametrisierung von MFLs bezüglich **FE**-Sichtbarkeitswechsel

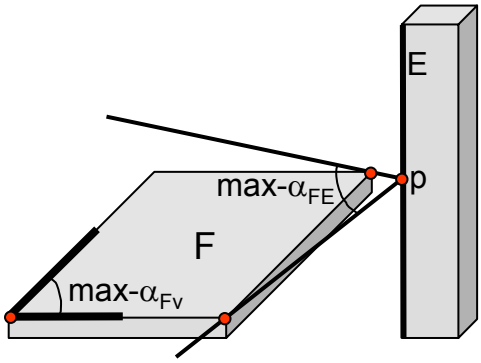


Abb. 3.83
Maximale Winkel für **Fv**- und **FE**-MFLs

Das folgende Beispiel soll nochmals den Aufbau des Visibility Skeletons verdeutlichen (vgl. [21]): In Abb. 3.84 ist eine einfache 3D-Szene mit sechs MFLs dargestellt. Rechts daneben sind diese MFLs als Knoten des Visibility Skeletons repräsentiert. In Abb. 3.85 sind die zugehörigen Sichtbarkeitswechsel eingezeichnet, rechts daneben die Repräsentation der Sichtbarkeitswechsel als Kanten des Visibility Skeletons.

Auch hier erkennt man die wesentlichen Eigenschaften des Visibility Skeletons:

- Jeder Sichtbarkeitswechsel wird von zwei MFLs begrenzt, von jeder MFLs gehen mehrere Sichtbarkeitswechsel aus (dargestellt sind hier jeweils maximal zwei).
- Extremal-Elemente der Sichtbarkeitswechsel ergeben sich aus denen der MFLs
- Auf Extremal-Elementen der Sichtbarkeitswechsel entstehen Schattenlinien

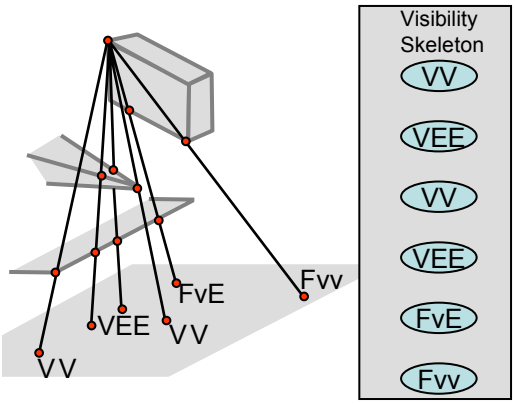


Abb. 3.84
3D-Szene mit MFLs

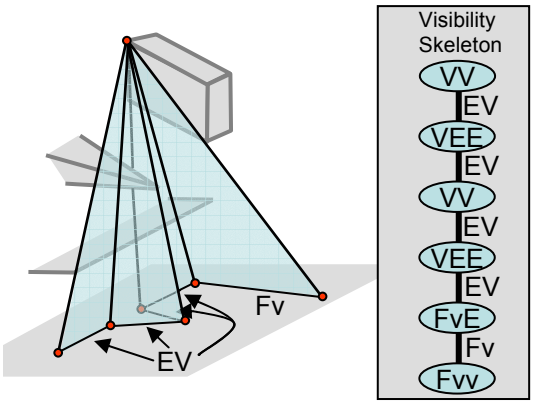


Abb. 3.85
3D-Szene mit Sichtbarkeitswechseln

3.5.6.2 Prinzipielle Konstruktion des Visibility Skeletons

In diesem Kapitel wird der prinzipielle Algorithmus zur Konstruktion des Visibility Skeletons einer 3D-Szenen vorgestellt. Dabei werden zwei Varianten angegeben, wobei die erste aus [21] bzw. [19] stammt und die zweite im Rahmen dieser Arbeit erstellt wurde.

Gemeinsam ist beiden Varianten, dass die Knoten (d.h. die MFLs) durch Bildung aller geeigneten Tupel von Generator-Kanten konstruiert werden und dann die adjazenten Kanten (d.h. die Sichtbarkeitswechsel) berechnet und in das Visibility Skeleton eingefügt werden.

Dabei wird hier der Übersichtlichkeit halber noch nicht auf die konkreten Ausprägungen der MFLs bzw. Sichtbarkeitswechsel eingegangen, d.h. sie werden streng nach Definition als die Menge von Linien durch vier bzw. drei Generator-Kanten betrachtet.

Konstruktion des Visibility Skeletons (Variante 1):

```
VisibilitySkeleton vs = new VisibilitySkeleton

for each combination of generator edges G = (g0, g1, g2, g3)
     if (G defines a valid MFL L)
        L.calcExtremities()      // Extremal-Elemente des MFL berechnen
        vs.insert(L)             // MFL in Visibility Skeleton einfügen

        for each arc A = (gi0, gi1, gi2) adjacent to L
          if (vs contains arc A' with same generators as A)
             A'.insert(A)        // Sichtbarkeitswechsel A in bestehenden Sichtbar-
          else                   // keitswechsel A' einfügen
             A.calcExtremities(L)
             vs.insert(A)        // Sichtbarkeitswechsel A' in Vis. Skel. einfügen
```

Code-Beispiel 3.3

Auf den ersten Blick erscheint dieser Code sehr einfach: für jede MFL werden die adjazenten Sichtbarkeitswechsel berechnet. Ist ein Sichtbarkeitswechsel mit denselben Generatoren schon im Visibility Skeleton enthalten, wird er mit dem neuen Sichtbarkeitswechsel verschmolzen. Andernfalls werden die Extremal-Elemente des neuen Sichtbarkeitswechsels berechnet und der Sichtbarkeitswechsel wird in das Visibility Skeleton eingefügt.

Bei genauerer Betrachtung stellt man jedoch fest, dass die Implementierung der Methoden **`calcExtremities`** und **`insertArc`** sehr aufwändig ist und weiterhin einige nicht-triviale geometrische Tests nötig macht. Dies soll hier nicht vollständig ausgeführt, sondern an einigen Beispielen veranschaulicht werden:

Abb. 3.86 zeigt zwei **VV**-MFLs $\mathbf{v_0v_1}$ und $\mathbf{v_0v_2}$, die einen **EV**-Sichtbarkeitswechsel $\mathbf{v_0e_0}$ bilden. Die Extremal-Elemente der **VV**-Knoten sind $\mathbf{(v_0, v_1)}$ bzw. $\mathbf{(v_0, f_0)}$, die Extremal-Elemente des Sichtbarkeitswechsels sind $\mathbf{(v_0, e_0)}$. Diese können also nicht immer direkt aus den Extremal-Elementen der adjazenten MFLs berechnet werden.

Ein weiterer Problemfall ist in Abb. 3.87 dargestellt: „zwischen“ den beiden **VEE**-MFLs $\mathbf{v_0e_0e_2}$ und $\mathbf{v_0e_1e_2}$ ist der **EV**-Sichtbarkeitswechsel $\mathbf{v_0e_2}$ unterbrochen, da Vertex und Kante in diesem Bereich gegenseitig global nicht sichtbar sind. Um zu testen, welches der beiden **VEE**-MFLs „Anfang“ und welches „Ende“ des unterbrochenen Bereichs ist, muss die relative Lage der beiden beteiligten Kanten $\mathbf{e_0}$ und $\mathbf{e_1}$ untersucht werden. Dies ist in Abb. 3.87 durch Pfeile an den Kanten dargestellt, als Testbedingung würde sich hier z.B. ergeben:

$$\left(\vec{e} \times \vec{e}_2\right) \cdot \vec{l} \begin{cases} < 0 & \text{"Start"} \\ > 0 & \text{"Ende"} \end{cases} \quad \text{mit } (e, l) \in \{(e_0, v_0e_0e_2),\ (e_1, v_0e_1e_2)\}$$

Dies setzt allerdings eine geeignete Ausrichtung der Kanten voraus, die nur durch eine Fallunterscheidung bestimmt werden kann, da jede Kante entlang der zu ihr adjazenten und von $\mathbf{v_0}$ lokal sichtbaren Facette ausgerichtet sein muss. Dies wiederum macht eine Klassifikation des Vertex anhand der Halbräume der betrachteten Kanten nötig.

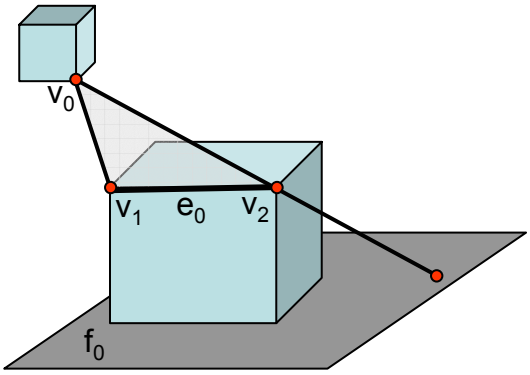


Abb. 3.86
Extremal-Elemente des Sichtbarkeitswechsels lassen sich nicht immer direkt aus den Extremal-Elementen der adjazenten MFLs bestimmen

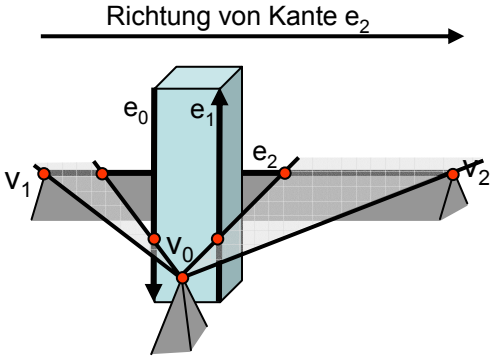


Abb. 3.87
Klassifikation von Start- und End-Knoten ist abhängig von der Ausrichtung der betrachteten Kanten

Die notwendigen Testbedingungen sollen hier nicht im Detail diskutiert werden. Als Ergebnis sollte lediglich festgehalten werden, dass bei diesem Ansatz die Bestimmung der Extremal-Elemente nicht-triviale geometrische Tests notwendig macht.

Auch die Implementierung der Methode **`insertArc(a)`** ist recht aufwändig. Abb. 3.88–Abb. 3.90 zeigen die drei Fälle, die dabei möglich sind:

`add`	Neuer Sichtbarkeitswechsel wird einfach hinzugefügt
`merge`	Neuer Sichtbarkeitswechsel wird mit einem vorhandenen verschmolzen
`split`	Neuer Sichtbarkeitswechsel spaltet einen vorhandenen

Wie man sieht, hängt der durchzuführende Schritt (**`add`**, **`merge`** oder **`split`**) in erster Linie von der Reihenfolge des Einfügens ab.

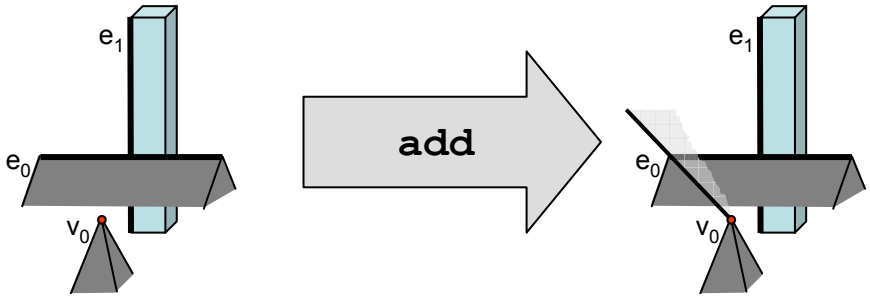


Abb. 3.88
`insertArc(a)` mit **`add`**-Schritt

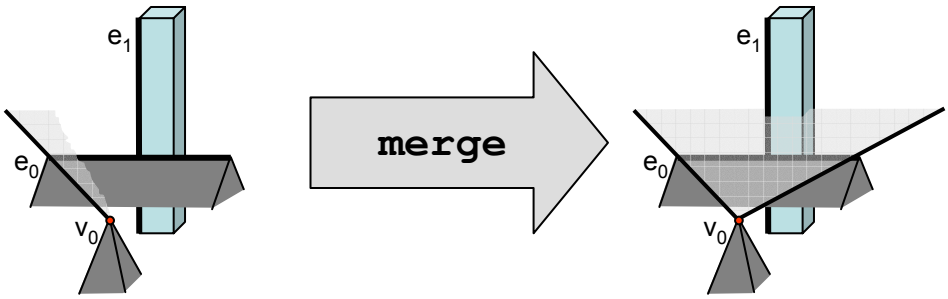


Abb. 3.89
`insertArc(a)` mit **`merge`**-Schritt

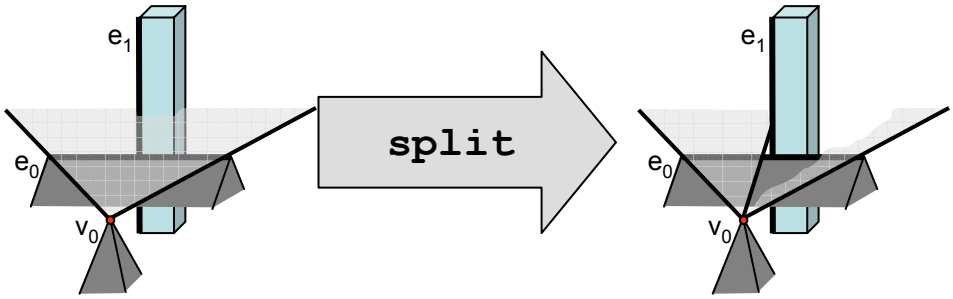


Abb. 3.90
`insertArc(a)` mit **`split`**-Schritt

Auch hier interessieren wieder weniger die Details dieser Schritte als vielmehr der offensichtliche Aufwand bei der Implementierung und der Ausführung des Algorithmus.

Diese sehr grobe Sicht auf das in [21] vorgeschlagene Verfahren zum Aufbau des Visibility Skeletons reicht jedoch aus, um die wesentlichen Vor- und Nachteile aufzuzeigen und so den neuen Algorithmus zum Aufbau des Visibility Skeletons zu motivieren:

Als wichtigster Vorteil ergibt sich die natürliche Konstruktion des Visibility Skeletons: Schrittweise werden MFLs und adjazente Sichtbarkeitswechsel eingefügt und das Visibility Skeleton wird dabei immer konsistent gehalten. Durch die geometrischen Tests ist weiterhin sichergestellt, dass Fehler bei der Konstruktion der Knoten schnell erkannt werden können.

Die oben aufgezählten Nachteile und Problemfälle lassen sich durch eine Eigenschaft des vorgestellten Konstruktions-Verfahrens erklären: anschaulich kann das Verfahren nicht „in die Zukunft blicken“, d.h. zum Zeitpunkt des Einfügens sind eben nur die bisher eingefügten und der einzufügende Sichtbarkeitswechsel bekannt. Dadurch ergibt sich die umständliche Bestimmung der Extremal-Elemente und die komplizierte Folge von **`add`**-, **`merge`**- und **`split`**-Schritten.

Konstruktion des Visibility Skeletons (Variante 2)

Ein besserer Ansatz zur Konstruktion des Visibility Skeletons muss bei den Nachteilen der Variante 1 zur Konstruktion ansetzen. Zur Verdeutlichung kann man Abb. 3.91 betrachten: dort sind vier MFLs dargestellt, die Nummerierung entspricht der Reihenfolge, in der die MFLs berechnet werden.

In den ersten beiden Schritten werden die beiden **VV**-MFLs berechnet und ein Sichtbarkeitswechsel zwischen ihnen aufgespannt (vgl. Abb. 3.92). Für diesen Sichtbarkeitswechsel wird bei Variante 1 kein eindeutiges Extremal-Element gefunden, da der Sichtbarkeitswechsel sowohl die Facette $\mathbf{F_0}$ als auch die Facette $\mathbf{F_1}$ schneidet.

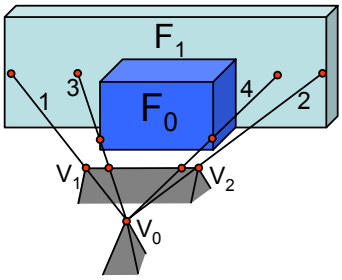


Abb. 3.91
Vier MFLs mit Berechnungs-Reihenfolge

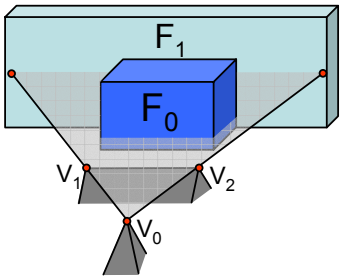


Abb. 3.92
Sichtbarkeitswechsel ohne eindeutiges Extremal-Element

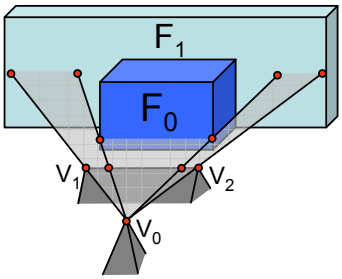


Abb. 3.93
Drei Sichtbarkeitswechsel mit eindeutigen Extremal-Elementen

Erst wenn auch die beiden mittleren **VEE**-MFLs berechnet wurden, ist der Sichtbarkeitswechsel aus Abb. 3.92 in drei Sichtbarkeitswechsel aufgespalten (vgl. Abb. 3.93). Die beiden äußeren Sichtbarkeitswechsel haben dann das eindeutige Extremal-Element $\mathbf{F_1}$, der mittlere Sichtbarkeitswechsel das eindeutige Extremal-Element $\mathbf{F_0}$.

Um dieses Ergebnis zu erreichen, mussten bei Variante 1 insgesamt ein **`add`**-Schritt (MFL 1), ein **`merge`**-Schritte (MFL 2) und zwei **`split`**-Schritte (MFLs 3 und 4) durchgeführt werden (vgl. dazu auch Abb. 3.88–Abb. 3.90). Für die MFLs 3 und 4 musste dabei noch getestet werden, ob es sich um den Start- oder End-Knoten des jeweiligen Sichtbarkeitswechsels handelt (vgl. Abb. 3.87).

Um diese aufwändige Konstruktion zu vereinfachen, kann man zunächst alle MFLs berechnen und in die jeweiligen Sichtbarkeitswechsel einfügen, ohne jedoch an dieser Stelle bereits die Extremal-Elemente zu berechnen. Damit ergibt sich die in Abb. 3.91 dargestellte Situation: alle vier MFLs wurden berechnet und entlang des Sichtbarkeitswechsels angeordnet.

Da dies nicht mehr der Definition des Sichtbarkeitswechsels entspricht, wird an dieser Stelle ein neuer Typ von Sichtbarkeitswechseln eingeführt:

```
class MasterArc
   Element[] gen;              // Array von Generator-Elementen
   List      nodeList;         // Liste der MFLs im Master-Sichtbarkeitswechsel
   void insertNode(Node l)     // füge MFL l in Master-Sichtbarkeitswechsel ein
```

Code-Beispiel 3.4

Im Unterschied zum "normalen" Sichtbarkeitswechsel, der von zwei MFLs begrenzt wird und eindeutige Extremal-Elemente besitzt, enthält der Master-Sichtbarkeitswechsel eine beliebige Menge von MFLs und besitzt dadurch auch keine eindeutigen Extremal-Elemente. Daher ist **`MasterArc`** auch keine Unterklasse von **`VisSkelElem`**, da dort bereits ein Array von Extremal-Elementen deklariert wird.

Weiterhin ist zu bemerken, dass die Liste der MFLs zunächst keinem Ordnungskriterium genügt, d.h. der Inhalt der Liste wird einfach durch die Reihenfolge des Einfügens bestimmt.

Mit diesem Hilfsmittel lässt sich nun ein neuer Algorithmus zum Aufbau des Visibility Skeletons formulieren:

```
VisibilitySkeleton vs   = new VisibilitySkeleton()
List               list = ∅   // Liste von Master-Sichtbarkeitswechseln

for each combination of generator edges G = (g0, g1, g2, g3)
   if (G defines a valid MFL L)
      L.calcExtremities()
      for each master arc A adjacent to L
        if (list contains master arc A' with same generators as A)
            A'.insertNode(L)  // füge MFL L in Master-SW. A' ein
        else
            list.add(A)       // füge Master-SW.  A in Liste der Master-SW. ein

for each master arc A in list
   sort nodes in node list of A by parameter
   for each pair (ni, ni+1) of nodes in node list of A
      Element[] ext = calcExtremities(A.gen, {ni, ni+1})
      if (ext != null)
         a = new Arc(A.gen, ni, ni+1, ext)
         vs.insert(a)
```

Code-Beispiel 3.5

Dieser Algorithmus zerfällt grob in zwei Teile: im oberen Teil werden die MFLs berechnet und in die entsprechenden Master-Sichtbarkeitswechsel eingefügt.

Im unteren Teil werden für jeden Master-Sichtbarkeitswechsel zunächst die enthaltenen MFLs nach dem Ordnungs-Parameter (vgl. Kap. 3.5.6.1) sortiert. Dann wird für jedes aufeinander folgende Paar von MFLs ein neuer Sichtbarkeitswechsel angelegt, wobei die Extremal-Elemente ohne weitere Tests direkt aus den Extremal- und Generator-Elementen der MFLs berechnet werden können (vgl. Kap. 3.5.6.4). Ist der so

konstruierte Sichtbarkeitswechsel gültig, wird er ins Visibility Skeleton eingefügt. Ein Beispiel für einen ungültigen Sichtbarkeitswechsel ist in Abb. 3.87 dargestellt: „zwischen“ den beiden mittleren **VEE**-MFLs ist die Kante des **EV**-Sichtbarkeitswechsels global nicht sichtbar, also ist die eingeschlossene Facette kein Extremal-Element und der Sichtbarkeitswechsel damit ungültig.

In den folgenden beiden Teilkapiteln wird die Konstruktion der MFLs und Sichtbarkeitswechsel genauer betrachtet, insbesondere werden auch die sehr einfachen Regeln zur Berechnung der Extremal-Elemente formuliert. Alle Ausführungen beziehen sich auf die zweite vorgestellte Möglichkeit zur Konstruktion des Visibility Skeletons.

3.5.6.3 Konstruktion der maximal freien Liniensegmente

Um alle MFLs einer 3D-Szene zu konstruieren, müssen alle Kombinationen von Generator-Elementen durchlaufen und daraufhin getestet werden, ob es eine Linie durch diese Generator-Elemente gibt. In den bisherigen Code-Beispielen wurden dabei jeweils streng nach Definition der MFLs Tupel von Generator-Kanten betrachtet. Da aber die konkreten Ausprägungen bekannt sind (vgl. Definition 3.16), empfiehlt es sich, dies bei der Konstruktion auszunutzen.

Da detaillierte Konstruktions- und Code-Beispiele für alle zehn Typen von MFLs zu aufwändig und auch redundant wären, sollen hier stellvertretend nur die MFL-Typen **VV**, **VEE/EVE** und **E4** genauer betrachtet werden. Dazu wird zunächst das allgemeine Konstruktionsprinzip vorgestellt und dann auf die betrachteten MFL-Typen angewandt. In Kap. 3.5.7 werden ein gemeinsamer Rahmen und Optimierungen für die Konstruktion aller MFL-Typen vorgeschlagen.

Die Konstruktion von MFLs läuft in vier Schritten bzw. Tests ab:

(1) ***Kombination von Generator-Elementen auf gültige MFL prüfen***

Hier wird getestet, ob es eine Linie durch die gegebenen Generator-Elemente geben kann. Für einige MFL-Typen ist dies trivial (z.B. gibt es immer eine Linie zwischen zwei Vertices (**VV**)), für andere recht kompliziert (**E4**).

(2) ***Test auf lokale Sichtbarkeit***

Eine MFL ist nur dann gültig, wenn sie von den Generator-Elementen lokal sichtbar ist. Dies ist gleichbedeutend mit der Bedingung, dass die Schnittpunkte der MFL mit den Generator-Elementen von den jeweils anderen Generator-Elementen lokal sichtbar sind.

(3) ***Test auf globale Sichtbarkeit***

Analog zu (2) muss die MFL von den Generator-Elementen auch global sichtbar sein, d.h. die MFL darf „zwischen" den Generator-Elementen keine anderen 3D-Objekte schneiden.

(4) ***Berechnung der Extremal-Elemente***

Für eine nach (1) – (3) gültige MFL müssen abschließend die Extremal-Elemente gefunden werden.

Die Behandlung der Punkte (3) und (4) wird auf Kap. 3.5.7.1 verschoben. Dort wird ein auf den Strahlanfragen aus Kap. 2.2.1 aufbauender Algorithmus zum Test der globalen Sichtbarkeit und zur Berechnung der Extremal-Elemente vorgestellt.

Die Schritte (1) und (2) können zusammengefasst bzw. verschränkt werden: sind zwei Generator-Elemente gegenseitig lokal nicht sichtbar, muss die Existenz eines MFL durch sie nicht geprüft werden.

Das folgende Code-Beispiel zeigt die sehr einfache Konstruktion eines **VV**-MFL:

```
NodeVV createVV(Vertex v0, Vertex v1)
     if (!v0.isLocVis(v1) || !v1.isLocVis(v0))
          return null

     if (v0.id > v1.id)
          swap(v0, v1)

     Ray       ray = new Ray(v0.p, v1.p)
     Element[] ext = getExtremities(r)

     if (ext != null)
          return new NodeVV(v0, v1, ext)
```

Code-Beispiel 3.6

Zunächst wird auf gegenseitige lokale Sichtbarkeit der beiden Vertices getestet. Sind die Vertices gegenseitig lokal sichtbar, werden sie nach der eindeutigen ID geordnet. Dann wird ein Strahl zwischen beiden Vertices gebildet und die zugehörigen Extremal-Elemente berechnet. Sind die Vertices gegenseitig global nicht sichtbar, wird **NULL** für die Extremal-Elemente zurückgeliefert, ansonsten kann eine gültige **VV**-MFL aus den Generator-Vertices und den Extremal-Elementen gebildet werden.

Analog dazu kann der Code für die Konstruktion eines **VEE**/**EVE**-MFLs formuliert werden. Hinzu kommen in diesem Fall eine stärkere Verschränkung der Konstruktion der Linie und der lokalen Sichtbarkeitstests sowie die Anordnung der Generator-Elemente:

```
Node createVEE_EVE(Vertex v, Edge e0, Edge e1)
     // Test: Vertex von beiden Kanten lokal sichtbar?
     if (!e0.isLocVis(v) || !e1.isLocVis(v))
          return null

     // Berechne Schnittpunkt/-parameter Ebene (v, e0) und e1, teste auf lok. Sichtbarkeit
     double  t1 = intersection parameter of e1 with plane (v, e0)
     Point3d p1 = intersection point of e1 with plane (v, e0)
     if ((t1 ∉ [0,1]) || !v.isLocVis(p0) || !e1.isLocVis(p0))
          return null

     // Berechne Schnittpunkt/-parameter Ebene (v, e1) und e0, teste auf lok. Sichtbarkeit
     double  t0 = intersection parameter of e0 with plane (v, e1)
     Point3d p0 = intersection point of e0 with plane (v, e1)
     if ((t1 ∉ [0,1]) || !v.isLocVis(p0) || !e1.isLocVis(p0))
          return null

     if (Vertex v lies between Edges e0 and e1) // Prüfe auf VEE-/EVE-MFL
          // EVE: ordne Kanten nach ID
          if (e0.id > e1.id) swap(e0,e1,t0,t1,p0,p1)
          Element[] ext = getExtremities(new Ray(p0, p1))
          if (ext != null)
               return  new NodeEVE(e0, v, e1, t0, t1, ext)
     else
          // VEE: ordne Kanten nach Entfernung vom Vertex
          if (||v.p − p0||₂ > ||v.p − p1||) swap(e0,e1,t0,t1,p0,p1)
          Element[] ext = getExtremities(new Ray(v.p, p1))
          if (ext != null)
               return  new NodeVEE(v, e0, e1, t0, t1, ext)

     return null
```

Code-Beispiel 3.7

Als erstes wird getestet, ob der Vertex von beiden Kanten aus lokal sichtbar ist. Dann werden jeweils die durch den Vertex und eine der Kanten definierte Ebene gebildet, die andere Kante mit ihr geschnitten, auf gültigen Schnittpunkt getestet und geprüft, ob der Schnittpunkt vom Vertex bzw. der anderen Kante aus lokal sichtbar ist.

Gilt dies für beide Schnittpunkte, existiert eine Linie durch die Generator-Elemente. Nun ist zu prüfen, ob die Generator-Elemente eine **VEE**- oder eine **EVE**-MFL bilden. In beiden Fällen müssen die Generator-Elemente entlang der MFL angeordnet werden. Im Falle einer **VEE**-MFL muss getestet werden, ob die zweite Kante die vom Vertex weiter entfernte ist, ansonsten müssen die Kanten (und alle abhängigen Parameter) vertauscht werden. Für die **EVE**-MFL sollte gelten, dass die erste Kante die mit der kleineren ID ist. Die weiteren Schritte (Bildung des Strahls, Berechnung der Extremal-Elemente und Anlegen der MFL) entsprechen denen aus Code-Beispiel 3.6.

Die mit Abstand aufwändigste Konstruktion ist die der **E4**-MFLs. In [54] wird ein numerisches Verfahren zur Lösung dieses Problems vorgeschlagen, das auf der Plücker-Darstellung der Kanten aufbaut: Das Skalarprodukt zweier Plücker-Linien ist **0**, wenn die beiden Linien sich schneiden oder parallel sind. Mit dieser Eigenschaft kann sofort das folgende lineare Gleichungssystem für die Kanten $\mathbf{e}_i$ und das gesuchte MFL $\mathbf{l}$ aufgestellt werden:

$$\begin{pmatrix} \tilde{e}_0^1 & \tilde{e}_0^2 & \tilde{e}_0^3 & \tilde{e}_0^4 & \tilde{e}_0^5 & \tilde{e}_0^6 \\ \tilde{e}_1^1 & \tilde{e}_1^2 & \tilde{e}_1^3 & \tilde{e}_1^4 & \tilde{e}_1^5 & \tilde{e}_1^6 \\ \tilde{e}_2^1 & \tilde{e}_2^2 & \tilde{e}_2^3 & \tilde{e}_2^4 & \tilde{e}_2^5 & \tilde{e}_2^6 \\ \tilde{e}_3^1 & \tilde{e}_3^2 & \tilde{e}_3^3 & \tilde{e}_3^4 & \tilde{e}_3^5 & \tilde{e}_3^6 \end{pmatrix} \cdot \begin{pmatrix} \tilde{l}_5 \\ \tilde{l}_6 \\ \tilde{l}_4 \\ \tilde{l}_3 \\ \tilde{l}_1 \\ \tilde{l}_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Dieses unterbestimmte Gleichungssystem kann nur mittels einer aufwändigen Singulärwertzerlegung gelöst werden, die weiteren Details sind [54] zu entnehmen.

Hier sollen stattdessen zwei Verfahren vorgestellt werden, die mit elementaren geometrischen Operationen auskommen: Das erste, in [21] vorgeschlagene, Verfahren dient lediglich dazu, Kombinationen von mindestens zwei Kanten und höchstens zwei 3D-Entitäten daraufhin zu testen, ob es überhaupt eine **E4**-MFL durch diese 3D-Entiäten geben kann.

Das zweite, im Rahmen dieser Arbeit erstellte, Verfahren kann auf anschauliche und einfache Weise für ein 4-Tupel von Kanten die Existenz der **E4**-MFLs ausschliessen oder nachweisen und dann die konkrete MFL bestimmen.

Zunächst soll es also darum gehen, für ein 2- oder 3-Tupel von Kanten zu entscheiden, ob eine 3D-Entität (B-Box, 3D-Objekt oder Kante) zusammen mit den gegebenen Kanten eine **E4**-MFL bilden kann (vgl [21]). Dazu kann man die Menge aller Linien durch zwei Kanten $\mathbf{e_0}$ und $\mathbf{e_1}$ betrachten. Diese Menge wird vom so genannten **EE**-Keil eingeschlossen, der sich aus den vom Vertex einer Kante und der jeweils anderen Kante gebildeten Ebenen ergibt (vgl. Abb. 3.94).

Liegt nun eine 3D-Entität **K** „zwischen“ diesen Ebenen, gibt es eine Linie durch $\mathbf{e_0}$, $\mathbf{e_1}$ und **K**. Ist **K** selber eine Kante, so kann sie auf das „Innere“ des **EE**-Keil zugeschnitten werden, dadurch ergibt sich eine Teilkante **K'**. Analog dazu kann jetzt die Kante $\mathbf{e_0}$ ($\mathbf{e_1}$) auf das „Innere“ des **EE**-Keils von $\mathbf{e_1}$ ($\mathbf{e_0}$) und **K'** geclippt werden, dadurch ergeben sich Teilkanten $\mathbf{e_0'}$ und $\mathbf{e_1'}$.

Dann gilt, dass jede 3D-Entität **L**, die in den **EE**-Keilen $\mathbf{(e_0', e_1')}$, $\mathbf{(e_0', K')}$ und $\mathbf{(e_1', K')}$ liegt, eine **E4**-MFL mit $\mathbf{e_0'}$, $\mathbf{e_1'}$ und **K'** bilden kann. Umgekehrt gilt, dass 3D-Entitäten, die nicht in allen **EE**-Keilen liegen, keine **E4**-MFL mit $\mathbf{e_0'}$, $\mathbf{e_1'}$ und **K'** bilden können.

Die Zusammenfassung der **EE**-Keile $\mathbf{(e_0', e_1')}$, $\mathbf{(e_0', K')}$ und $\mathbf{(e_1', K')}$ wird im Folgenden **EEE**-Keil von $\mathbf{(e_0, e_1, K)}$ genannt. In Kap. 3.5.7 werden einige weitere Optimierungen für die Berechnung und Anwendung von **EE(E)**-Keilen vorgestellt, hier soll zunächst ausreichen, dass mit den **EE(E)**-Keilen auf einfache Weise Kandidaten für **E4**-MFLs ausgeschlossen werden können.

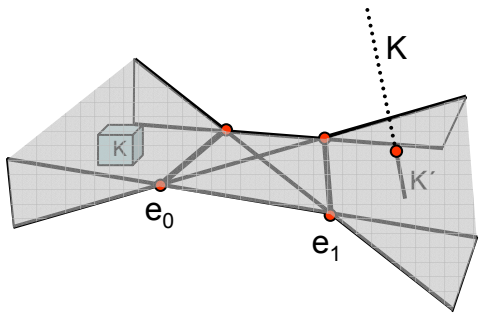


Abb. 3.94
EE-Keil zweier Kanten

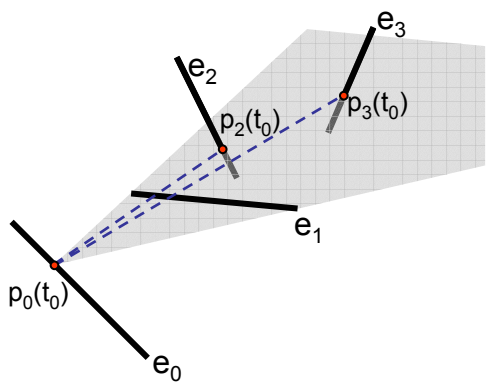


Abb. 3.95
Berechnung von **E4**-MFLs

Abschließend soll nun noch die konkrete Konstruktion von **E4**-MFLs für ein 4-Tupel **(e_0, e_1, e_2, e_3)** von Kanten vorgestellt werden (vgl. Abb. 3.95). Dabei werden im Folgenden diese Bezeichnungen verwendet:

Kante e_i: Start-/Endpunkt $p_{i0/1}$ und Richtungsvektor $d_i = p_{i1} - p_{i0}$

Wir betrachten die Punkte auf der Kante e_0:

(1) $$p_0(t_0) = p_{00} + t_0 \cdot d_0 \qquad (t_0 \in [0, 1])$$

Jeder Punkt $p_0(t_0)$ definiert mit der Kante e_1 eine Ebene $E_{01}(t_0)$ mit Normalenvektor

(2) $$\begin{aligned} n(t_0) &= (p_{11} - p_0(t_0)) \times (p_{10} - p_0(t_0)) \\ &= p_{11} \times p_{10} + p_0(t_0) \times p_{11} - p_0(t_0) \times p_{10} + \underbrace{p_0(t_0) \times p_0(t_0)}_{=(0,0,0)^T} \\ &= p_0(t_0) \times d_1 + p_{11} \times p_{10} \end{aligned}$$

Die Schnittparameter und –punkte der Kanten e_2 und e_3 mit $E_{01}(t_0)$ ergeben sich zu

(3) $$t_j(t_0) = -\frac{n(t_0) \cdot p_{10} - n(t_0) \cdot p_{j0}}{n(t_0) \cdot d_j} \quad \text{und} \quad p_j(t_0) = p_{j0} + t_j(t_0) \cdot d_j \quad (j \in \{2, 3\}).$$

Die beiden Schnittpunkte $p_2(t_0)$ und $p_3(t_0)$ definieren mit dem Punkt $p_0(t_0)$ zwei Vektoren $v_2(t_0) = (p_2(t_0) - p_0(t_0))$ und $v_3(t_0) = (p_3(t_0) - p_0(t_0))$. Sind $v_2(t_0)$ und $v_3(t_0)$ kollinear, so ist eine Linie mit Aufpunkt $p_0(t_0)$ und und Richtungsvektor $v_2(t_0)$ durch die vier Kanten gefunden.

Zwei Vektoren sind kollinear, wenn die Länge des Kreuzprodukt-Vektors 0 ist. Dies ist gleichbedeutend mit der Aussage, dass der Kreuzprodukt-Vektor $v(t_0)$ komponentenweise 0 sein muss:

(4) $$v(t_0) = (p_2(t_0) - p_0(t_0)) \times (p_3(t_0) - p_0(t_0)) = (0,0,0)^T$$

Durch Einsetzen der Terme für $p_j(t_0)$ ($j \in \{0, 2, 3\}$) ergibt sich:

(5) $$\begin{aligned} v(t_0) = \; & (p_2(t_0) - p_0(t_0)) \times (p_3(t_0) - p_0(t_0)) \\ = \; & p_2(t_0) \times p_3(t_0) + p_0(t_0) \times p_2(t_0) - p_0(t_0) \times p_3(t_0) \\ = \; & \underbrace{p_{00} \times (p_{20} - p_{30}) + p_{20} \times p_{30}}_{=a} \\ & + t_0 \cdot \underbrace{(d_0 \times (p_{20} - p_{30}))}_{=b} + t_2 \underbrace{(d_2 \times (p_{30} - p_{00}))}_{=c} + t_3 \underbrace{(d_3 \times (p_{00} - p_{20}))}_{=d} \\ & + t_0 t_2 \underbrace{(d_0 x d_2)}_{=e} - t_0 t_3 \underbrace{(d_0 \times d_3)}_{=f} + t_2 t_3 \underbrace{(d_2 \times d_3)}_{=g} \\ = \; & a + t_0 \cdot b + t_2 \cdot c + t_3 \cdot d + t_0 t_2 \cdot e - t_0 t_3 \cdot f + t_2 t_3 \cdot g \end{aligned}$$

Durch weitere Einsetzung der Terme für $\mathbf{t_j(t_0)}$ ($\mathbf{j \in \{2, 3\}}$) folgt:

$$
\begin{aligned}
\mathbf{v(t_0)} = \quad & \mathbf{a + t_0 \cdot b + t_2 \cdot c + t_3 \cdot d + t_0 t_2 \cdot e - t_0 t_3 \cdot f + t_2 t_3 \cdot g} \\
= \quad & \mathbf{a + t_0 \cdot b - \frac{n(t_0) \cdot p_{10} - n(t_0) \cdot p_{20}}{n(t_0) \cdot d_2} c - \frac{n(t_0) \cdot p_{10} - n(t_0) \cdot p_{30}}{n(t_0) \cdot d_3} d} \\
& \mathbf{- \frac{n(t_0) \cdot p_{10} - n(t_0) \cdot p_{20}}{n(t_0) \cdot d_2} t_0 e + \frac{n(t_0) \cdot p_{10} - n(t_0) \cdot p_{30}}{n(t_0) \cdot d_3} t_0 f} \\
& \mathbf{+ \frac{n(t_0) \cdot p_{10} - n(t_0) \cdot p_{20}}{n(t_0) \cdot d_2} \cdot \frac{n(t_0) \cdot p_{10} - n(t_0) \cdot p_{30}}{n(t_0) \cdot d_3} g}
\end{aligned}
\tag{6}
$$

Mit der Bedingung $\mathbf{v(t_0) = (0, 0, 0)^T}$ kann (6) mit den Nennern multipliziert werden:

$$
\begin{aligned}
\mathbf{(0,0,0)^T} = \quad & \mathbf{n(t_0) \cdot d_2 \cdot n(t_0) \cdot d_3 \cdot a + n(t_0) \cdot d_2 \cdot n(t_0) \cdot d_3 \cdot t_0 b} \\
& \mathbf{- n(t_0) \cdot d_3 \cdot n(t_0) \cdot d_{12} \cdot c - n(t_0) \cdot d_2 \cdot n(t_0) \cdot d_{13} \cdot d} \\
& \mathbf{- n(t_0) \cdot d_3 \cdot n(t_0) \cdot d_{12} \cdot t_0 \cdot e + n(t_0) \cdot d_2 \cdot n(t_0) \cdot d_{13} \cdot t_0 \cdot f} \\
& \mathbf{+ n(t_0) \cdot (p_{10} - p_{20}) \cdot n(t_0) \cdot (p_{10} - p_{30}) \cdot g}
\end{aligned}
\tag{7}
$$

Diese Rechnung wird nicht explizit fortgeführt, da sich die Auflösung der Gleichung über mehrere Seiten erstrecken würde. Am bisherigen Ergebnis erkennt man aber bereits die wesentliche Eigenschaft: die Ausdrücke $\mathbf{p_0(t_0)}$ und $\mathbf{n(t_0)}$ sind linear in $\mathbf{t_0}$, durch Ausmultiplizieren des obigen Ausdrucks ergibt sich ein kubisches Polynom in $\mathbf{t_0}$.

Da die Variablen **a**-**g** Vektoren sind, kann man dies als Gleichungssystem formulieren:

$$
\mathbf{q_0 \cdot t_0^3 + q_1 \cdot t_0^2 + q_2 \cdot t_0 + q_3} = \begin{pmatrix} q_0^X \cdot t_0^3 + q_1^X \cdot t_0^2 + q_2^X \cdot t_0 + q_3^X \\ q_0^Y \cdot t_0^3 + q_1^Y \cdot t_0^2 + q_2^Y \cdot t_0 + q_3^Y \\ q_0^Z \cdot t_0^3 + q_1^Z \cdot t_0^2 + q_2^Z \cdot t_0 + q_3^Z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{8}
$$

Gesucht sind die Werte von $\mathbf{t_0}$, für die alle drei Gleichungen eine Nullstelle haben. Dazu kann man alle Nullstellen einer der drei Gleichungen berechnen und testen, ob sie auch Nullstellen der beiden anderen Gleichungen sind.

Eine Möglichkeit zur Berechnung ist die Folgende (vgl. [8]): Sei o.B.d.A. $\mathbf{i = q_0^X}$, $\mathbf{j = q_1^X}$, $\mathbf{k = q_2^X}$ und $\mathbf{l = q_3^X}$. Für $\mathbf{i = 0}$ bzw. $\mathbf{l = 0}$ reduziert sich das Problem auf das Lösen einer quadratischen Gleichung, daher wird im Folgenden o.B.dA. $\mathbf{i \neq 0}$ und $\mathbf{l \neq 0}$ vorausgesetzt.

Weiterhin sei

$$\mathbf{m} = \frac{1}{3}\left(\frac{3k}{i} - \left(\frac{j}{i}\right)^2\right),\ \mathbf{n} = \frac{1}{27}\left(2\left(\frac{j}{i}\right)^3 - 9\frac{jk}{i^2} + 27\frac{l}{i}\right) \text{ und } \mathbf{o} = \frac{m^2}{4} + \frac{n^3}{27}.$$

Die Nullstellen können dann über folgende Fallunterscheidung gefunden werden:

o > 0: Sei $\mathbf{A} = \sqrt[3]{-\frac{n}{2} + \sqrt{o}}$ und $\mathbf{B} = \sqrt[3]{-\frac{n}{2} - \sqrt{o}}$.

Die Nullstellen ergeben sich zu:

$$\mathbf{x} = \mathbf{A} + \mathbf{B} - \frac{j}{3i} \text{ und } \mathbf{x} = -\frac{\mathbf{A}+\mathbf{B}}{2} - \frac{j}{3i} \pm \frac{\mathbf{A}-\mathbf{B}}{2}\sqrt{-3}$$

o = 0: Die Nullstellen ergeben sich zu:

$$\mathbf{x} = 2\sqrt[3]{-\frac{n}{2}} - \frac{j}{3i} \text{ und } \mathbf{x} = -\sqrt[3]{-\frac{n}{2}} - \frac{j}{3i}$$

o < 0: Transformiere in Polarkoordinaten:

$$\mathbf{r} = \sqrt{\frac{n^2}{4} - o} \text{ und } \theta = \cos^{-1}\left(-\frac{n}{2r}\right)$$

Die Nullstellen ergeben sich zu:

$$\mathbf{x} = 2\sqrt[3]{r}\cos\left(\frac{\theta}{3}\right) - \frac{j}{3i} \text{ und } \mathbf{x} = -\sqrt[3]{r}\left(\cos\left(\frac{\theta}{3}\right) \pm \sqrt{3}\sin\left(\frac{\theta}{3}\right)\right) - \frac{j}{3i}$$

Damit liegt nun insgesamt ein einfaches Verfahren zur Berechnung von **E4**-MFLs vor:

- Mit Hilfe der **EE**(**E**)-Keile werden Tupel $(\mathbf{e_0, e_1, e_2, e_3})$ von Kandidaten-Kanten gefunden.
- Für jedes solche Tupel kann das Gleichungssystem (8) aufgestellt werden
- Gilt für eine Gleichung aus (8) $\mathbf{q_0^W = 0}$ oder $\mathbf{q_3^W = 0}$ ($\mathbf{W \in \{ X, Y, Z\}}$), können die Nullstellen direkt bestimmt werden.
- Sonst können die Nullstellen mit dem obigen Verfahren bestimmt werden
- Sind alle Nullstellen für eine Gleichung gefunden, kann getestet werden, ob es auch Nullstellen der beiden anderen Gleichungen sind.

3.5.6.4 Konstruktion der Sichtbarkeitswechsel

Wie bereits in Kap. 3.5.6.1 ausgeführt, wird hier nur die zweite Möglichkeit zur Konstruktion des Visibility Skeletons betrachtet, da sie die Konstruktion der MFLs und Sichtbarkeitswechsel sauber voneinander trennt und (fast) keine Sonderfallbetrachtungen nötig macht.

Zunächst wird während der Konstruktion der MFLs jede erfolgreich konstruierte MFL in die zugehörigen Master-Sichtbarkeitswechsel eingefügt. Sind alle MFLs der 3D-Szene konstruiert, durchläuft man alle Master-Sichtbarkeitswechsel und sortiert die enthaltenen MFLs nach dem Ordnungs-Parameter.

Dieser Teil wurde bereits in Code-Beispiel 3.5 formuliert, hier soll es nun um die innere Schleife, d.h. die eigentliche Konstruktion der Sichtbarkeitswechsel, gehen:

```
for each pair (n_i, n_i+1) of nodes in node list of master arc A
   ext = calcExtremities(A, n_i,n_i+1)
   if (ext != null)
      a = new Arc(A.gen, n_i, n_i+1, ext)
      visSkeleton.insert(a)
```

Code-Beispiel 3.8

Als Beispiel wird hier die Konstruktion von **EV**-Sichtbarkeitswechseln betrachtet, die Konstruktion der anderen Typen von Sichtbarkeitswechseln verläuft weitgehend analog.

Zunächst muss geprüft werden, ob das MFL-Paar einen gültigen Sichtbarkeitswechsel definiert. In Abb. 3.87 ist ein Beispiel für einen ungültigen Sichtbarkeitswechsel zwischen einem Paar von MFLs zu sehen: der Bereich, der von den beiden **VEE**-MFLs eingeschlossen wird, ist vom Vertex aus global nicht sichtbar, definiert also auch keinen gültigen Sichtbarkeitswechsel.

Beim ersten Ansatz zur Konstruktion des Visibility Skeletons mussten ungültige (d.h. verdeckte) Sichtbarkeitswechsel durch komplizierte geometrische Tests erkannt werden, hier reicht dagegen ein einfacher Vergleich der Generator-Elemente der MFLs aus:

Wenn die zweite Kante einer **VEE**-MFL der Kante des **EV**-Sichtbarkeitswechsels entspricht (vgl. Abb. 3.87), ist die **VEE**-MFL der „Beginn“ oder das „Ende“ eines ungültigen Sichtbarkeitswechsels.

Um zwischen „Beginn“ und „Ende“ eines ungültigen Sichtbarkeitswechsels zu unterscheiden, kann man eine triviale Eigenschaft ausnutzen: Wenn die letzte

betrachtete MFL Beginn eines ungültigen Sichtbarkeitswechsels war, so ist die nächste MFL das zugehörige Ende. Es reicht also aus, mittels eines Flags zu speichern, ob die letzte MFL Beginn oder Ende eines ungültigen Sichtbarkeitswechsels war.

Als Sonderfall ergibt sich hier lediglich, dass die erste MFL eines Master-Sichtbarkeitswechsels nie Beginn eines ungültigen Sichtbarkeitswechsels sein kann.

Mit diesen Erkenntnissen kann Code-Beispiel 3.8 um die Unterscheidung zwischen gültigen und ungültigen Sichtbarkeitswechseln erweitert werden:

```
boolean bBlocked = false;

for each pair (n_i, n_i+1) of nodes in node list of master arc a'
   if (i > 0 && n_i is VEE && n_i.gen[2] == a'.gen[0])
      bBlocked = !bBlocked

   if (!bBlocked)
      ext = calcExtremities(a', n_i,n_i+1)
      a = new Arc(a'.gen, n_i, n_i+1, ext)
      visSkeleton.insert(a)
```

Code-Beispiel 3.9

Die Abfrage des **`bBlocked`**-Flags ersetzt hier die Abfrage auf gültige Extremal-Elemente (**`ext≠null`**). Anzumerken ist noch, dass hier im Gegensatz zu den vorigen Code-Beispielen implizit davon ausgegangen wird, dass die Schleife die MFL-Paare in aufsteigender Reihenfolge durchläuft.

Wurde so ein MFL-Paar gefunden, das einen gültigen Sichtbarkeitswechsel definiert, müssen die Extremal-Elemente des Sichtbarkeitswechsels berechnet werden. Auf den ersten Blick ergeben sich hier sehr viele verschiedene (Sonder-)Fälle, die sich jedoch alle einheitlich behandeln lassen. Die folgende Tabelle listet diese Fälle auf, der Übersichtlichkeit halber werden dabei nur **VV-** und **VEE**-MFLs betrachtet:

Extremal-Element des Sichtbarkeitswechsels ergibt sich aus…	***vgl.***
…Extremal-Elementen der adjazenten MFLs	Abb. 3.96
…Generator-Elementen der adjazenten MFLs	Abb. 3.97
…Extremal- und Generator-Elementen der adjazenten MFLs	Abb. 3.98 und Abb. 3.99

Tabelle 3.5

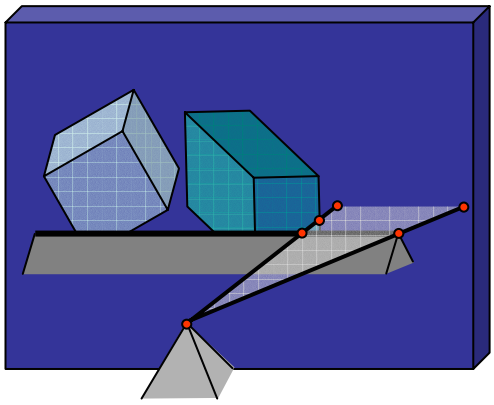


Abb. 3.96
Extremal-Element des Sichtbarkeitswechsels ergibt sich aus Extremal-Elementen der MFLs

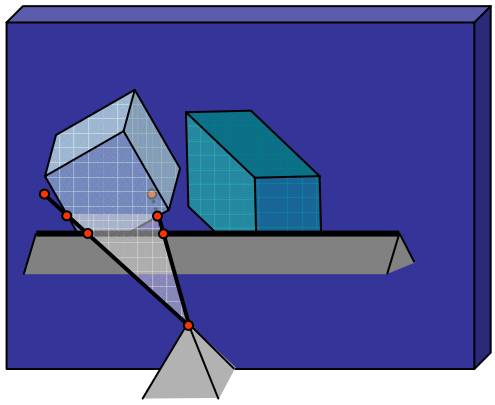


Abb. 3.97
Extremal-Element des Sichtbarkeitswechsels ergibt sich aus Generator-Elementen der MFLs

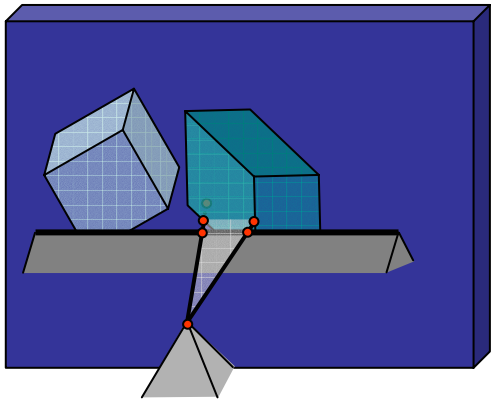


Abb. 3.98

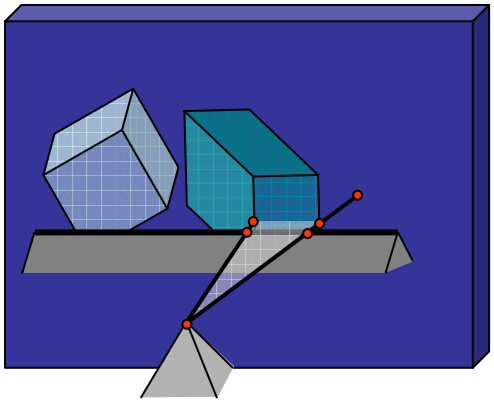


Abb. 3.99

Extremal-Element des Sichtbarkeitswechsels ergibt sich aus Extremal-/Generator-Elementen der MFLs

Der einfachste Fall ist offensichtlich der erste: Das Extremal-Element des Sichtbarkeitswechsels kann direkt aus den Extremal-Elementen der adjazenten MFLs konstruiert werden.

Abb. 3.100 – Abb. 3.108 listen die dabei möglichen Kombinationen von MFL-Extremal-Elementen **(A, B)** und das sich daraus ergebende Extremal-Element **C** des Sichtbarkeitswechsels in der Form **(A, B) → C** auf (der triviale Fall **(V, V) → V** wurde dabei ausgelassen). Die Punkte repräsentieren Schnittpunkte der MFLs mit dem Extremal-Element (d.h. die Punkte **A** bzw. **B**) und die gestrichelten Linien die Schnittlinie des Sichtbarkeitswechsels mit dem Extremal-Element (d.h. das Liniensegment **C**).

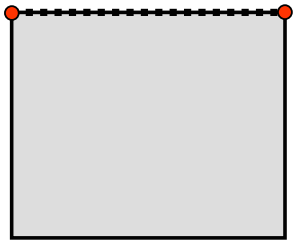


Abb. 3.100
(V, V) → E

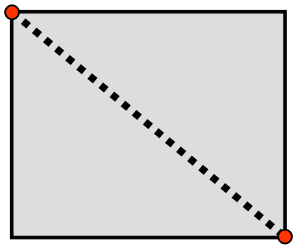


Abb. 3.101
(V, V) → F

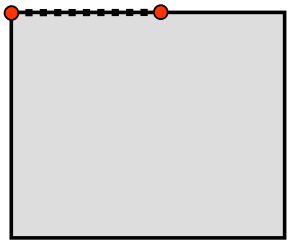


Abb. 3.102
(V, E) → E

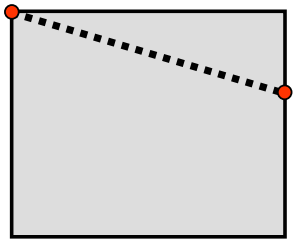


Abb. 3.103
(V, E) → F

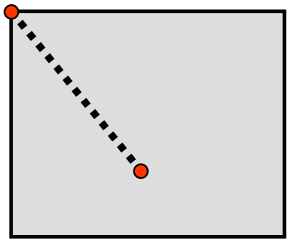


Abb. 3.104
(V, F) → F

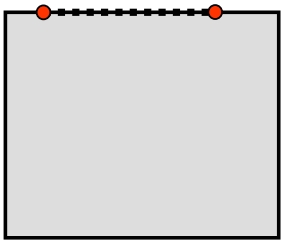


Abb. 3.105
(E, E) → E

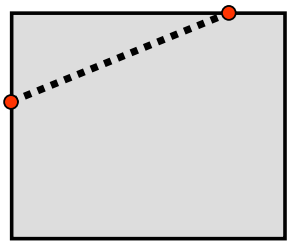


Abb. 3.106
(E, E) → F

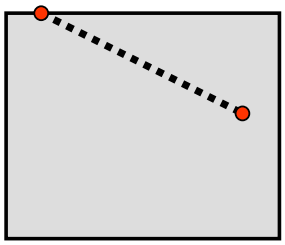


Abb. 3.107
(E, F) → F

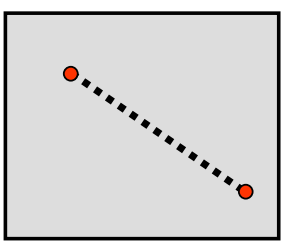


Abb. 3.108
(F, F) → F

Die Methode **`getSimpleExtremity`** ist die Umsetzung dieser Überlegungen:

```
Element getSimpleExtremity(Element ext0, Element Ext1)
     if (ext0 == ext1) return ext0

     if ((ext0 is Vertex) || (ext1 is Vertex))
          Element sharedEdge = sharedEdge(ext0, ext1)
          if (sharedEdge != null) return sharedEdge

     return sharedFace(ext0, ext1)
```

Code-Beispiel 3.10

Wenn sich das Extremal-Element des Sichtbarkeitswechsels nicht direkt aus den Extremal-Elementen der MFLs rekonstruieren lässt (Abb. 3.97 - Abb. 3.99), kann man für alle (sinnvollen) Paare von Generator- und Extremal-Elementen der MFLs testen, ob **`getSimpleExtremity`** ein gültiges Extremal-Element findet.

Die Umsetzung dieser Idee erfolgt in zwei Schritten. Zunächst wird der Algorithmus zur Konstruktion der Extremal-Elemente aus Paaren von Generator- und Extremal-Elementen formuliert. Danach wird die Auswahl dieser Elemente beschrieben.

```
Element[] calcExtremities(MasterArc A, Node[] node)
     Element[][][] pot    = getExtremalCandidates(A, n)
     Element[]     arcExt = new Element[2];

     for (int i = 0; i < 2; i++)
        for (int c0 = 0; (arcExt[i] == null) && (c0 < 2); c0++)
          for (int c1 = 0; (arcExt[i] == null) && (c1 < 2; c1++)
             arcExt[i]=getSimpleExtremity(pot[i][0][c0],
                                          pot[i][1][c1])
     return arcExt;
```

Code-Beispiel 3.11

Das Array **pot** enthält die Kandidaten für die Extremal-Elemente und ist wie folgt indiziert: Die erste Dimension gibt das gesuchte Extremal-Element des Sichtbarkeitswechsels an, die zweite Dimension die betrachtete MFL und die dritte Dimension schließlich den Kandidaten für das Extremal-Element.

In den beiden inneren Schleifen werden nun für ein Extremal-Element des Sichtbarkeitswechsels alle Kombinationen von Kandidaten durchgespielt, die Schleifen brechen ab, sobald ein gültiges Extremal-Element gefunden wurde.

Die eigentliche Auswahl der Kandidaten erfolgt anhand einfacher Regeln, die im Wesentlichen nur von den Typen des betrachteten Knotenpaars abhängen. Der Übersichtlichkeit wegen wird im folgenden Code-Beispiel nur die Kandidaten-Auswahl für **VEE**-MFLs durchgeführt, die anderen Fälle werden ähnlich behandelt:

```
Element[][][] getExtremalCandidates(MasterArc a', Node[]  node)
     Element[][][] pot = new Element[2][2][2]

     for (int i = 0; i < 2; i++)
       for (int j = 0; j < 2; j++)
         pot[j][i][1] = node[j].ext[i].elem;
       if (node[i] is VEE) && (node[i].gen[1] == a'.gen[0]))
         pot[0][i][0] = node[i].gen[2];
       else if …

     return pot;
```

Code-Beispiel 3.12

In der äußeren Schleife (`i`) werden die beiden MFLS durchlaufen. Zunächst werden die zweiten Kandidaten für die Extremal-Elemente einfach auf die Extremal-Elemente des MFL gesetzt. Dann werden die ersten Kandidaten für die Extremal-Elemente gesetzt:

Ist das MFL ein **VEE**-MFL und entspricht weiterhin die erste Kante dieses MFL der Generator-Kante des Sichtbarkeitswechsels, so ist die zweite Kante des **VEE**-MFL ein Kandidat für das Extremal-Element des Sichtbarkeitswechsels.

Diese Überlegung kann man sich leicht an Abb. 3.99 klarmachen: Beide MFLs des Sichtbarkeitswechsels sind **VEE**-MFLs, also wird auch für beide MFLs die „hintere" Kante als Kandidat gespeichert. Die Methode **`getSimpleExtremities`** liefert dann für diese beiden Kandidaten das gemeinsame Element, d.h. die von den MFLs eingeschlossene Facette. Diese ist ein Extremal-Element des dargestellten Sichtbarkeitswechsels.

Wie bereits ausgeführt, werden durch diese Art der Konstruktion komplizierte und ungenaue geometrische Tests vermieden. Allerdings erscheint das Durchspielen aller Kombinationen von Extremal-Elementen zunächst sehr umständlich. Dazu ist folgendes anzumerken:

- Der Test auf ein gemeinsames Element zweier gegebener Elemente E_0 und E_1 ist quadratisch in der Zahl der zu E_0 und E_1 adjazenten Elemente, da getestet werden muss, ob es ein Element gibt, das zu beiden adjazent ist.

  Allerdings gilt, dass zwei Elemente nur dann ein gemeinsames Element einschließen können, wenn sie beide zum selben 3D-Objekt gehören, dies kann mit Aufwand **O(1)** getestet werden. „Unsinnige" Kombinationen von Kandidaten sind dadurch gekennzeichnet, dass sie zu unterschiedlichen 3D-Objekten gehören. Man kann also davon ausgehen, dass alle „unsinnigen" Kombinationen mit Aufwand **O(1)** verworfen werden können.

- Der Sonderfall, dass ein Extremal-Element eines MFLs gleichzeitig Generator-Element ist, kann dazu führen, dass dieses Element doppelt in der Kandidatenliste auftaucht. Dies ist jedoch kein Problem, sondern eher ein Vorteil: Die Kombinationen mit diesem Element werden einfach früher getestet, nämlich nicht erst im zweiten, sondern bereits im ersten Durchlauf der **`c0/c1`**-Schleifen aus Code-Beispiel 3.11.

Mit diesen Überlegungen liegt nun eine einfache und performante Methode zur Konstruktion der Sichtbarkeitswechsel aus den MFLs vor. Dieser Ansatz kommt im Gegensatz zu dem in [21] vorgeschlagenen ohne komplizierte und ungenaue geometrische Tests aus. Weiterhin werden die in einem Master-Sichtbarkeitswechsel enthaltenen Sichtbarkeitswechsel Schritt für Schritt konstruiert, ohne dass ein teueres Verschmelzen und Splitten von Sichtbarkeitswechseln notwendig ist.

Der Aufwand hierfür ist linear in der Zahl **n** der in einem Master-Sichtbarkeitswechsel enthaltenen MFLs. Je nach Implementierung erhöht sich der Aufwand durch die Sortierung der MFLs eventuell auf **O(n log n)**. Da die Zahl der MFLs in einem Master-Sichtbarkeitswechsel jedoch meist „klein“ ist, fällt dies nicht weiter ins Gewicht.

3.5.7 Verbesserungen

Der Aufwand zur Konstruktion des Visibility Skeletons einer 3D-Szene ist sehr groß: Um die MFLs zu berechnen, müssen Kombinationen von Generator-Elementen daraufhin getestet werden, ob sie eine gültige MFL definieren. Im Falle der **E4**-MFLs müssen im schlimmsten Fall alle 4-Tupel von Generator-Kanten betrachtet werden. Dies ergibt eine Komplexität von $\mathbf{O(n_e^4)}$, wenn $\mathbf{n_e}$ die Zahl der Generator-Kanten ist. In Kap. 3.5.6.3 wurde bereits ausgeführt, dass viele 3-Tupel von Generator-Kanten ohne weitere Tests verworfen werden können, wenn sie nicht gegenseitig in ihren **EE**-Keilen liegen, dies ändert jedoch nichts an der theoretischen Komplexität des Algorithmus.

In diesem Kapitel werden einige Optimierungsvorschläge für die Konstruktion der MFLs vorgestellt. Zunächst werden der globale Sichtbarkeitstest und die Berechnung der Extremal-Elemente eines beliebigen MFL(-Typs) durch eine Strahlanfrage an die BB-Hierarchie der Szene umgesetzt.

Dann wird die Optimierung der Konstruktion aller MFL-Typen einzeln diskutiert. Dabei wird das Konzept der verallgemeinerten Shaft und der Shaft-Klassifikation von BB-Hierarchien aus Kap. 3.4.3 aufgegriffen. Damit übertragen sich alle dortigen Überlegungen zur effizienten Shaft-Klassifikation direkt auf die Optimierung der MFL-Konstruktion!

3.5.7.1 Globaler Sichtbarkeitstest und Berechnung der Extremal-Elemente

Für jedes MFL muss getestet werden, ob es global sichtbar ist (d.h. ob es zwischen seinen Generator-Elementen ein 3D-Objekt schneidet) und für jedes global sichtbare MFL müssen die Extremal-Elemente gefunden werden (d.h. die Elemente, die die MFL „vor“ und „hinter“ den Generator-Elementen begrenzen).

Beide Schritte entsprechen Strahlanfragen an eine 3D-Szene (vgl. Kap. 2.2.1): der globale Sichtbarkeitstest der inneren Verdeckungs-Strahlanfrage und die Berechnung der Extremal-Elemente der unteren bzw. oberen Extremal-Strahlanfrage (vgl. Abb. 3.109).

Der einfachste Ansatz zur Umsetzung dieser Strahlanfragen ist, die Objektsliste der Szene linear zu durchlaufen und für jedes 3D-Objekt **O** die drei Strahlanfragen auszuwerten. Wird **O** von der inneren Verdeckungsanfrage als **OCC** klassifiziert, kann die Auswertung der beiden Extremal-Anfragen abgebrochen werden.

Der erste Schritt der Optimierung ist die Ausnutzung der BB-Hierarchie der Szene (vgl. [3]): wenn ein Strahl eine B-Box **B** nicht schneidet, kann er auch keine(s) der in **B** enthaltenen B-Boxen oder 3D-Objekte schneiden. Eine bei der Wurzel beginnende Pre-Order-Traversierung der BB-Hierarchie muss also in jedem Knoten **B** nur fortgesetzt werden, wenn **B** den Strahl schneidet. Wird ein 3D-Objekt von der inneren Verdeckungsanfrage als **OCC** klassifiziert, kann die Traversierung der BB-Hierarchie abgebrochen werden.

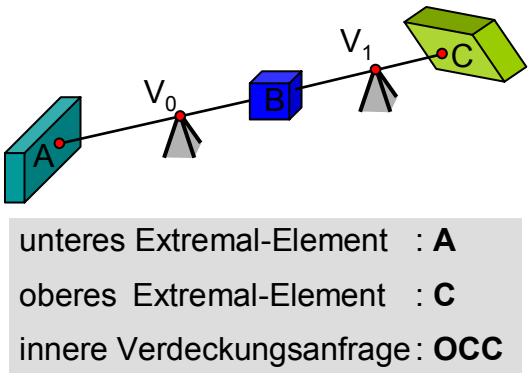


Abb. 3.109
Innere Verdeckungs- und Extremal-Strahlanfrage für ein **VV**-MFL

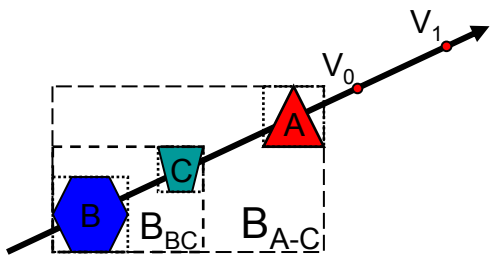


Abb. 3.110
B- Box $\mathbf{B_{BC}}$ liegt vor Extremal-Element **A**

Weiterhin muss die Traversierung nur für B-Boxen fortgesetzt werden, die „zwischen“ den Extremal-Elementen liegen. Dies kann man sich anhand von Abb. 3.110 für das untere Extremal-Element eines **VV**-MFL klarmachen: Die BB-Hierarchie wird in der Inorder-Reihenfolge $\mathbf{B_{A\text{-}C}}$, $\mathbf{B_A}$, $\mathbf{B_{BC}}$, … traversiert. Zuerst wird das Extremal-Element **A** gefunden. Die B-Box $\mathbf{B_{BC}}$ liegt entlang des Strahls „vor“ **A**, kann also kein Extremal-Element enthalten, das näher als **A** am Generator-Element $\mathbf{V_0}$ liegt.

Da die Extremal-Elemente nur interessieren, wenn der Strahl „zwischen“ den Generator-Elementen kein 3D-Objekt schneidet, sollten zuerst die B-Boxen traversiert werden, die zwischen den Generator-Elementen liegen. Nur wenn keins der in diesen B-Boxen enthaltenen 3D-Objekte den Strahl schneidet, müssen die Extremal-Elemente berechnet werden.

Dies kann man durch Verwendung zweier Stacks erreichen: bei der Traversierung werden nur B-Boxen weiterverfolgt, die zwischen den Generator-Elementen liegen, sonst werden sie auf die beiden Stacks verteilt: die B-Boxen „vor“ den Generator-Elementen auf den ersten Stack, die „hinter“ den Generator-Elementen auf den zweiten. Damit enthält der erste Stack die Kandidaten für das untere Extremal-Element und der zweite die Kandidaten die für das obere (vgl. Abb. 3.111).

Wird bei der Traversierung kein verdeckendes 3D-Objekt gefunden, werden die Extremal-Elemente berechnet. Dazu werden die Stack abgearbeitet, d.h. die Traversierung mit jeder B-Box auf den Stacks fortgesetzt.

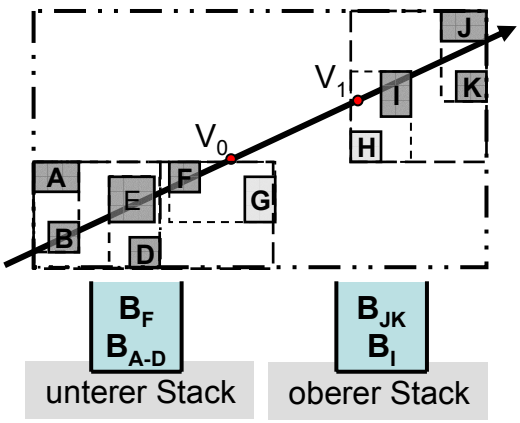


Abb. 3.111
Stacks nach Traversierung der BB-Hierarchie

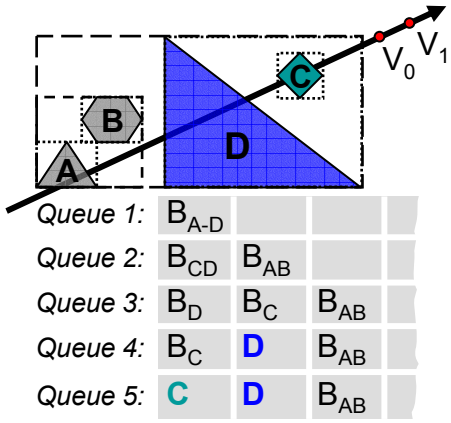


Abb. 3.112
Abarbeitung der Priority Queue für das untere Extremal-Element

Die Extremal-Elemente sind die Elemente, die den Strahl jeweils direkt „vor“ bzw. „hinter“ den Generator-Elementen schneiden. Für die Berechnung der Extremal-Elemente ist es also wünschenswert, die B-Boxen auf den Stacks auch genau in dieser Reihenfolge zu traversieren, d.h. für das untere Extremal-Element in absteigender, für das obere Extremal-Element in aufsteigender Reihenfolge.

Dies kann durch Ersetzung der Stacks durch Priority Queues erreicht werden. Die folgenden Ausführungen beziehen sich nur auf das untere Extremal-Element, können aber direkt auf das obere übertragen werden. Für das untere Extremal-Element entspricht die Ordnung der Priority Queue einer aufsteigenden Sortierung entlang des Strahls, d.h. das vordere Element der Queue ist das den Generator-Elementen am nächsten liegende.

Die Inorder-Traversierung der BB-Hierarchie wird durch eine der Level-Order-Traversierung vergleichbare ersetzt: in jedem Schritt wird das erste Element aus der Queue entnommen und auf Schnitt mit dem Strahl geprüft. Ist das Element eine B-Box, werden die Kind-Boxen bzw. das enthaltene 3D-Objekt in die Queue eingefügt. Ist das Element ein 3D-Objekt, ist das Extremal-Element gefunden.

In Abb. 3.112 sind diese Überlegungen dargestellt. Dort ist auch zu erkennen, warum die 3D-Objekte nochmals in die Priority Queue eingefügt werden müssen: zuerst wird das Extremal-Element **D** gefunden, da die B-Box $\mathbf{B_D}$ entlang des Strahls weiter „hinten“ als $\mathbf{B_C}$ liegt. Trotzdem liegt der Schnitt mit $\mathbf{B_C}$ und mit **C** entlang des Strahls „hinter“ dem Schnitt mit **D**, d.h. **C** ist das richtige Extremal-Element.

Damit ist die Verbesserung des globalen Sichtbarkeitstests und der Berechnung der Extremal-Elemente für ein maximal freies Liniensegment abgeschlossen. Zusammenfassend wurden folgende Verbesserungen eingebracht:

- Der globale Sichtbarkeitstest und die Berechnung der Extremal-Elemente wurden durch Strahlanfragen umgesetzt
- Diese Strahlanfragen wurden an die BB-Hierarchie der Szene angepasst.
- Durch die Verwendung von zwei Priority Queues wird zuerst der globale Sichtbarkeitstest durchgeführt. Ist dieser erfolgreich, werden die beiden Queues abgearbeitet und so die Extremal-Elemente bestimmt.

3.5.7.2 Konstruktion der maximal freien Liniensegmente

In Kap. 3.5.6.3 wurden zur Konstruktion der maximal freien Liniensegmente Tupel von Generator-Elementen gebildet und auf MFL-Eigenchaften geprüft, d.h. getestet, ob es eine Linie durch die Generator-Elemente gibt. Für die **E4**-MFLs bedeutet dies einen Aufwand von $\mathbf{O(n^4)}$ zum Testen aller Kombinationen von **n** Generator-Kanten.

Stattdessen kann man mit einem einzelnen Generator-Element $\mathbf{e_0}$ beginnen und für dieses zunächst alle B-Boxen zu suchen, die Generator-Elemente $\mathbf{e_1}$ enthalten, die mit $\mathbf{e_0}$ eine MFL bilden könnten. Dieses Verfahren wird dann mit dem Tupel $\mathbf{(e_0, e_1)}$ fortgesetzt, d.h. als nächstes werden alle B-Boxen gesucht, die Generator-Elemente $\mathbf{e_2}$ enthalten, die mit $\mathbf{(e_0, e_1)}$ eine MFL bilden könnten usw.

Zur Auswahl dieser B-Boxen wird das Konzept der verallgemeinerten Shafts aus Kap. wieder aufgegriffen. Ein verallgemeinerter Shaft war als Menge von Ebenen und Linien mit einer Klassifikation, wann eine 3D-Entität „im“ Shaft ist, definiert. Bisher bedeutete „im Shaft“, dass eine 3D-Entität die Sichtbarkeit gegebener Extremal-Elemente beeinflusst. In diesem Kapitel bedeutet „im Shaft“ nun, dass eine 3D-Entität mit den Generator-Elementen des Shafts ein MFL bilden kann.

In diesem Teilkapitel wird diese Idee zur Optimierung für alle MFL-Typen ausführlich erklärt. Für eine konkrete Umsetzung sollten die MFLs zuvor nach ihrem Typ eingeteilt werden, um doppelte Berechnungen zu vermeiden:

Definition 3.19: Einteilung der MFL-Typen

MFLs können in folgende Grundtypen aufgeteilt werden:

Typ	*MFL-Typen*
Vertex-MFLs	**VV**, **VEE**, **EVE**
Kanten-MFLs	**E**, **E4**
Facetten-MFLs	**Fvv**, **FvE**, **FEE**, **EFE**, **FF**

Tabelle 3.6

Zunächst wird anhand der Vertex-MFLs (**VV** und **VEE**/**EVE**) schrittweise die Anwendung der verallgemeinerten Shafts für die MFL-Konstruktion anhand von einfachen Code-Beispielen erklärt.

Dann wird ein Schema zur rekursiven Aufspaltung dieser Shafts in einen Baum von Kind-Shafts angegeben. Eine 3D-Entität wird nur dann anhand dieses Shaft-Baum als „im“ Shaft klassifiziert, wenn die 3D-Entität von den Generator-Elementen des Shafts global sichtbar ist. Dies bedeutet, dass für so gefundene MFLs keine globalen Sichtbarkeitstests mehr durchgeführt werden müssen und die Extremal-Elemente aus den (Kind-) Shafts rekonstruiert werden können!

Die Verwendung der verallgemeinerten Shafts und die Aufspaltung in Kind-Shafts werden mit einigen Anpassungen auch für die Konstruktion der Facetten-MFLs übernommen. Für die Konstruktion der Kanten-MFLs kann dieses Konzept nur teilweise übernommen werden, statt einer rekursiven Aufspaltung in Kind-Shafts ist hier nur eine Beschränkung der ursprünglichen Shafts möglich.

3.5.7.3 Vertex-MFLs

Ausgangspunkt der Konstruktion der Vertex-MFLs ist das folgende Code-Beispiel:

```
void createVertexMFLs(Scene s)
     for each vertex v in scene s
       createVertexMFLs(s, v, s.getBBHRoot())
```

Code-Beispiel 3.13

Der Algorithmus durchläuft alle Vertices der 3D-Szene und ruft die Methode **`createVertexMFLs`** mit der 3D-Szene **`s`**, dem aktuellen Vertex **`v`** und der Wurzel der BB-Hierarchie auf. In dieser Methode müssen nun die anderen Kandidaten für die **VV**-, **VEE**- und **EVE**-MFLs gefunden werden.

Dabei gilt, dass nur Elemente betrachtet werden müsen, die von **v** lokal sichtbar sind, d.h. die nicht im negativen Halbraum von **v** liegen. Dies kann als Klassifikation anhand eines verallgemeinerten Shafts (vgl. Definition 3.13) formuliert werden:

$$\mathbf{vs(o)} = \begin{cases} \forall \mathbf{e} \in \mathbf{E} : \mathbf{o} \subset \mathbf{e}^- & \mathbf{OUT} \\ \mathbf{sonst} & \mathbf{PAR} \end{cases}$$

Mit dieser Klassifikation werden 3D-Entitäten, die vom Vertex **v** lokal nicht sichtbar sind, als **OUT** klassifiziert.

Mit diesen Überlegungen kann Code-Beispiel 3.15 angepasst werden:

```
void createVertexMFLs(Scene s)
     for each vertex v in scene s
       VertexShaft vs = new VertexShaft(v)
       createVertexMFLs(s, vs, s.getBBHRoot())

void createVertexMFLs(Scene s, VertexShaft vs, Box b)
     if (vs.classify(b) ≠ OUT) // Box b vom Vertex v lokal sichtbar?
       if (b.isLeaf())
         createVertexMFLs(s, vs, b.getObject())
       else
         for each child box c of b
           createVertexMFLs(s, vs, c)
```

Code-Beispiel 3.14

Kommt die Rekursion im obigen Code-Beispiel so an ein (vom Vertex **v** lokal sichtbares) Blatt **b** der BB-Hierarchie, wird eine dritte **`createVertexMFLs`**-Methode mit der 3D-Szene **s**, dem Vertex **v** und dem in der Blatt-Box gespeicherten 3D-Objekt aufgerufen. In dieser Methode müssen nun die **VV**-MFLs konstruiert und die Konstruktion der **VEE**-/**EVE**-MFLs angestoßen werden:

```
void createVertexMFLs(Scene s, VertexShaft vs, Object3D O)
     for each vertex v1 in 3D-object o // VV-MFLs konstruieren
       createVV(vs.getVertex(), v1)
     for each edge e in 3D-Object o // Konstr. VEE-/EVE-MFLs anstoßen
       // Vertex und Kante gegenseitig lokal sichtbar?
       if (vs.classify(e) ≠ OUT && e.getState(v) ≠ NEG)
         EVShaft evs = new EVShaft(vs, e)
         createVEE_EVE_MFLs(s, evs, s.getBBHRoot())
```

Code-Beispiel 3.15

In der ersten Schleife werden alle **VV**-MFLs zwischen dem im Vertex-Shaft **vs** gespeicherten Vertex und den Vertices des 3D-Objekts **O** konstruiert. Die Methode **`createVV`** kann dabei unverändert aus Code-Beispiel 3.6 übernommen werden.

In der zweiten Schleife wird für alle Kanten **e** des 3D-Objekts **O** geprüft, ob Vertex und Kante gegenseitig (teilweise) lokal sichtbar sind. Ist dies der Fall, wird ein **EV**-Shaft aus Vertex und Kante definiert und die Methode **`calcVEE_EVE_MFLs`** aufgerufen.

Der **EV**-Shaft muss so aufgebaut sein, dass eine 3D-Entität „im“ **EV**-Shaft mit dem Vertex und der Kante eine **VEE**-/**EVE**-MFL bilden kann. In Kap. 3.5.3 wurde dies nicht allgemein für 3D-Entitäten, sondern nur für Kanten betrachtet. Diese Überlegungen können aber direkt auf 3D-Entitäten übertragen werden:

Eine Kante $\mathbf{e_1}$ kann mit dem Kante-Vertex-Paar $\mathbf{(e_0, v)}$ ein **VEE**-/**EVE**-MFL definieren, wenn…

(1) **v**, $\mathbf{e_0}$ und $\mathbf{e_1}$ gegenseitig (teilweise) lokal sichtbar sind

(2) $\mathbf{e_1}$ die durch $\mathbf{(e_0, v)}$ definierte Ebene $\mathbf{E_{ev}}$ im Punkt $\mathbf{p_1}$ schneidet (vgl. Abb. 3.113)

(3) $\mathbf{e_0}$ die durch **v** und $\mathbf{p_1}$ definierte Linie schneidet, d.h. $\mathbf{p_1}$ liegt „zwischen“ den durch $\mathbf{(v, e_0.p_0)}$ und $\mathbf{(v, e_0.p_1)}$ definierten Seitenlinien $\mathbf{L_{S0}}$ und $\mathbf{L_{S1}}$ (vgl. Abb. 3.113)

(4) Wenn **v** im positiven Halbraum von $\mathbf{e_0}$ liegt: $\mathbf{p_1}$ muss im positiven Halbraum von $\mathbf{e_0}$ liegen, d.h. vom Vertex **v** aus gesehen „vor“ der Kante $\mathbf{e_0}$.

(5) Wenn $\mathbf{e_0}$ im positiven Halbraum von **v** liegt: $\mathbf{p_1}$ muss im positiven Halbraum von **v** liegen, d.h. von der Kante $\mathbf{e_0}$ aus gesehen „vor“ dem Vertex **v**

Übertragen auf 3D-Entitäten bedeutet dies, dass für eine 3D-Entität **G**, die Kanten $\mathbf{e_1}$ enthält, die mit $\mathbf{(e_0, v)}$ eine **VEE**-/**EVE**-MFL bilden können, gelten muss:

(1) **G** ist von $\mathbf{e_0}$ und **v** (teilweise) lokal sichtbar

(2) **G** schneidet die Ebene $\mathbf{E_{ev}}$

(3) **G** liegt (teilweise) „zwischen“ den Seitenlinien $\mathbf{L_{S0}}$ und $\mathbf{L_{S1}}$

(4) Wenn **v** im positiven Halbraum von $\mathbf{e_0}$ liegt: **G** muss (teilweise) im positiven Halbraum von $\mathbf{e_0}$ liegen, d.h. vom Vertex **v** aus gesehen (teilweise) „vor“ der Kante $\mathbf{e_0}$

(5) Wenn $\mathbf{e_0}$ im positiven Halbraum von **v** liegt: **G** muss (teilweise) im positiven Halbraum von **v** liegen, d.h. von Kante $\mathbf{e_0}$ aus gesehen (teilweise) „vor“ dem Vertex **v**.

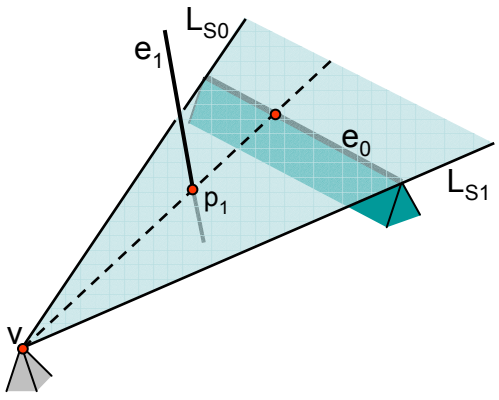


Abb. 3.113
Kante $\mathbf{e_1}$ muss Ebene $\mathbf{E_{ev}}$ schneiden
Linie $\mathbf{(v, p_1)}$ muss Kante $\mathbf{e_0}$ schneiden

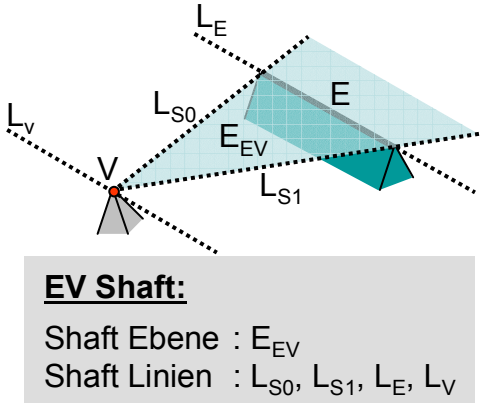


Abb. 3.114
Shaft-Ebene und Shaft-Linien des **EV**-Shafts

Aus diesen Bedingungen ergeben sich die Elemente des **EV**-Shafts (vgl. Abb. 3.114). Nur 3D-Entitäten „im“ **EV**-Shaft können mit dem Kante-Vertex-Paar **(e_0, v)** **VEE**-/**EVE**-MFLs definieren.

Damit kann der Rest der Konstruktion der Vertex-MFLs angegeben werden:

```
void createVEE_EVE_MFLs(Scene s, EVShaft evs, Box b)
     if (evs.classify(b) ≠ OUT) // Box b „im“ EV-Shaft?
       if (b.isLeaf())
         createVEE_EVE_MFLs(evs, b.getObject())
       else
         for each child box c of b
           createVertexMFLs(s, vs, c)

void createVEE_EVE_MFLS(EVShaft evs, Object3D o)
     Vertex v  = evs.getVertex()
     Edge   e0 = evs.getEdge()

     for each edge edge e1 in 3D-object o
       if (e1.getState(v) ≠ NEG && e1.getState(e0) ≠ NEG)
         createVEE_EVE(v, e0, e1)
```

Code-Beispiel 3.16

Zunächst wird die B-Box **b** anhand des EV-Shafts **evs** klassifiziert. Liegt **b** „im“ **EV**-Shaft, so wird die Traversierung der BB-Hiearchie mit den Kind-Boxen von **b** fortgesetzt. Ist **b** eine Blatt-Box, so werden für alle Kanten des in **b** enthaltenen 3D-Objekts die **VEE**-/**EVE**-MFLs berechnet. Dieser letzte Schritt der Konstruktion verläuft völlig analog zu Code-Beispiel 3.7 und wird daher hier nicht mehr diskutiert.

Durch Verwendung der verallgemeinerten Shafts wird also in jedem Schritt der MFL-Konstruktion nur der Teil der Bounding-Box-Hierarchie traversiert, der Kandidaten für die **VV**- und **VEE**-/**EVE**-MFLs enthalten kann. Bereits mit dieser einfachen Optimierung kann eine große Zahl von Kandidaten-Tupeln ausgeschlossen werden. In Kap. 3.5.8 wird diese Verbesserung anhand konkreter Messwerte untersucht.

Bisher wurden nur „lokale“ Eigenschaften des **EV**-Shafts betrachtet, d.h. der **EV**-Shaft hängt nur von der Generator-Kante e_0 und dem Generator-Vertex **v** ab. Es gilt aber weiterhin noch eine „globale“ Eigenschaft des **EV**-Shafts, die bisher noch nicht ausgenutzt wurde: Jedes Generator-Objekt **O**, das (teilweise) im **EV**-Shaft liegt, d.h. **VEE**-/**EVE**-MFLs mit dem Kante-Vertex-Paar **(e, v)** definieren kann, beschränkt den Bereich, in dem andere Generator-Objekte liegen können (vgl. [44] und [22]).

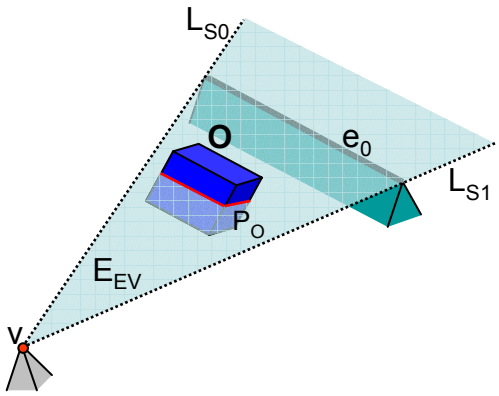


Abb. 3.115
Schnitt von 3D-Objekt **O** mit Ebene $\mathbf{E_{ev}}$ definiert Schnittpolygon $\mathbf{P_O}$

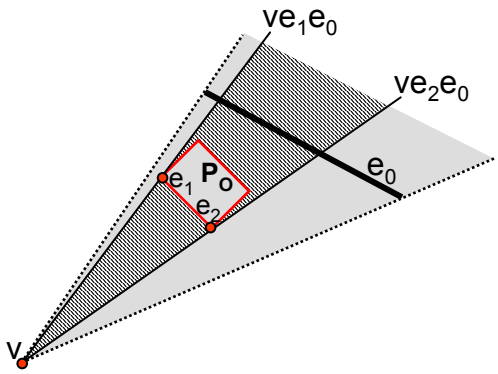


Abb. 3.116
Schnittpolygon **P** definiert Bereich für **VEE**- und **EVE**-MFLs

Dazu kann man zunächst Abb. 3.115 betrachten: das Generator-Objekt **O** schneidet die Ebene $\mathbf{E_{EV}}$ und liegt „zwischen“ den Seitenlinien $\mathbf{L_{S0}}$ uns $\mathbf{L_{S1}}$. Die Schnitte der Kanten des Generator-Objekts mit der Ebene definieren ein Schnittpolygon $\mathbf{P_O = (p_0, \ldots, p_n)}$. In diesem Beispiel werden dadurch zwei **VEE**-MFLs $\mathbf{ve_0e_1}$ und $\mathbf{ve_0e_2}$ definiert.

Diese beiden MFLs beschränken nun den Bereich für andere MFLs (vgl. Abb. 3.116), anschaulich können „hinter“ dem Schnittpolygon $\mathbf{P_0}$ und „zwischen“ den beiden MFLs (in Abb. 3.116 schraffiert) keine weiteren (Generator-Kanten für) **VEE**-/**EVE**-MFLs liegen. Dies bedeutet, dass der **EV**-Shaft neben den „normalen“ Shaft-Ebenen und –Linien nun durch drei weitere Shaft-Linien $\mathbf{ve_0e_1}$, $\mathbf{ve_0e_2}$ und $\mathbf{(ep_1, ep_2)}$ beschränkt ist.

Diese Beschränkung kann man als Aufspaltung des **EV**-Shafts auffassen: der **EV**-Shaft wird in drei Kind-Shafts $\mathbf{EV_0}$ – $\mathbf{EV_2}$ aufgespalten, die von den Shaft-Linien-Tupeln $\mathbf{(L_{S0}, ve_0e_1)}$, $\mathbf{(ve_0e_1, (ep_1, ep_2), ve_0e_2)}$ und $\mathbf{(ve_0e_1, L_{S1})}$ eingeschlossen werden.

Bei der Aufspaltung eines ***EV***-Shafts in Kind-Shafts können insgesamt sechs Fälle auftreten, die in der folgenden Tabelle anhand der Lage des Generator-Objekts **O** bezüglich der Shaft-Linien aufgelistet sind:

Generator-Objekt O…	***liegt „zwischen“ Seitenlinien***	***schneidet eine Seitenlinie***	***schneidet beide Seitenlinien***
liegt „zwischen“ Kante und Vertex	Abb. 3.117	Abb. 3.118	Abb. 3.119
liegt „vor“ Kante oder „hinter“ Vertex	Abb. 3.120	Abb. 3.121	Abb. 3.122

Tabelle 3.7

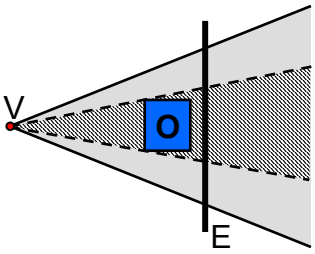


Abb. 3.117

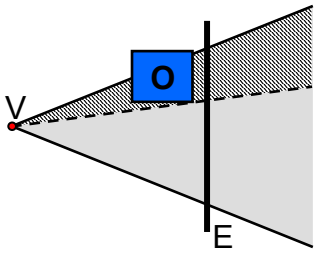


Abb. 3.118

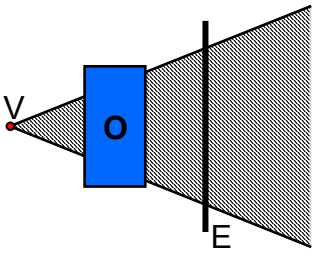


Abb. 3.119

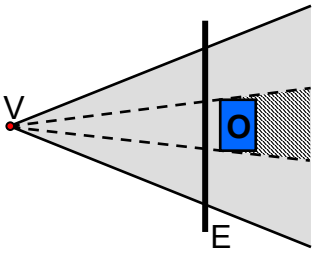


Abb. 3.120

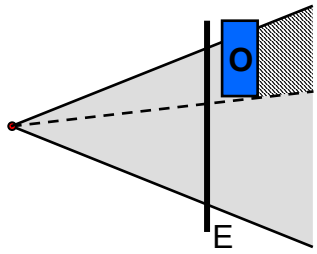


Abb. 3.121

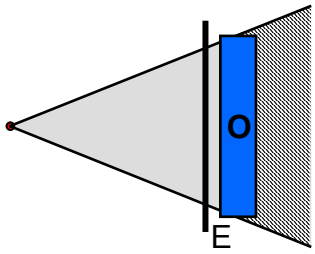


Abb. 3.122

In Abb. 3.117 – Abb. 3.122 ist jeweils der Teil des Shafts, in dem 3D-Entitäten als **OUT** klassifiziert werden, schraffiert dargestellt. Man erkennt folgende Eigenschaften:

- Generator-Objekte „zwischen“ Kante und Vertex spalten den **EV**-Shaft in maximal zwei Kind-Shafts auf (vgl. Abb. 3.117 und Abb. 3.118). Werden beide Seitenlinien geschnitten, so ist der ganze Shaft verdeckt (Klassifikation **OCC**, vgl. Abb. 3.119).
- Generator-Objekte „vor“ der Kante oder „hinter“ dem Vertex spalten den **EV**-Shaft in maximal drei Kind-Shafts auf (vgl. Abb. 3.120 und Abb. 3.121). Werden beide Seitenlinien geschnitten, so entstehen keine neuen Kind-Shafts, sondern der gesamte **EV**-Shaft wird durch eine weitere Shaft-Linie beschränkt.

Diese Beschränkung bzw. Aufspaltung des **EV**-Shafts kann mit anderen Generator-Objekten fortgesetzt werden (vgl. Abb. 3.123), diese rekursive Aufspaltung in Kind-Shafts ergibt einen Baum von **EV**-Shafts. In Abb. 3.124 ist der zu Abb. 3.123 gehörige Shaft-Baum dargestellt. Jeder Knoten des Baums entspricht einem (Kind-) **EV**-Shaft, die Knoten sind mit den begrenzenden Shaft-Linien attributiert.

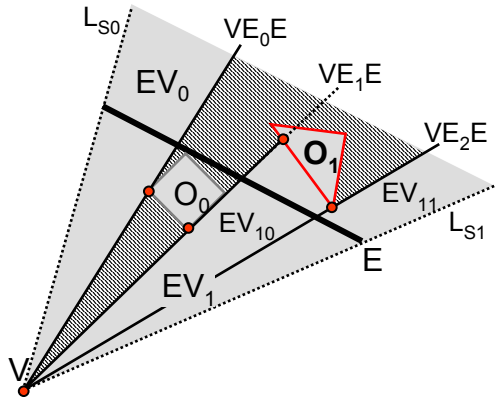


Abb. 3.123
Generator-Objekt $\mathbf{O_1}$ spaltet Kind-Shaft $\mathbf{EV_1}$ in Kind-Shafts $\mathbf{EV_{10}}$ und $\mathbf{EV_{19}}$ auf

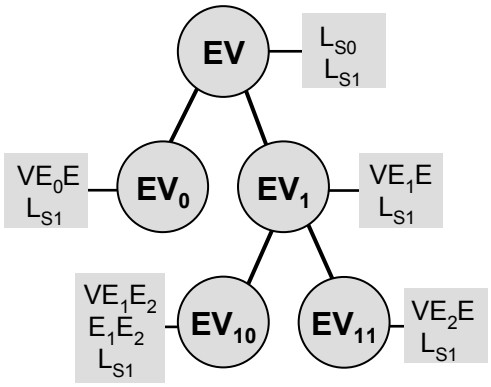


Abb. 3.124
Shaft-Baum für Aufspaltung links

Durch fortgesetzte Klassifikation von Generator-Objekten anhand eines **EV**-Shafts entsteht also ein Baum von **EV**-Shafts, wobei jeder innere Knoten entweder zwei oder drei Kind-Knoten hat. Jeder **EV**-Shaft wird von mindestens zwei Shaft-Linien beschränkt, diese Shaft-Linien entsprechen gleichzeitig **VEE**-/**EVE**-MFLs.

Von der Wurzel zu den Blättern dieses Baumes wird der **EV**-Shaft immer weiter beschränkt, d.h. eine zu klassifizierende 3D-Entität **G** wird nur dann als **INS** oder **PAR** klassifiziert, wenn sie tatsächlich vom Kante-Vertex-Paar $\mathbf{(e_0, v)}$ global sichtbar ist. Dies bedeutet insbesondere, dass für so gefundene **VEE**-/**EVE**-MFLs kein Raycasting mehr zum Nachweis der globalen Sichtbarkeit und dem Auffinden der Extremal-Elemente durchgeführt werden muss.

Im Prinzip wird durch diese Aufspaltung in Kind-Shafts auch schon die Konstruktion der Sichtbarkeitswechsel aus den MFLs vorweggenommen: die Wurzel des **EV**-Shaft-Baums entspricht dem Master-Sichtbarkeitswechsel aus Kap. 3.5.6.2 und die Blätter des **EV**-Shaft-Baums entsprechen einzelnen **EV**-Sichtbarkeitswechseln.

Die Extremal-Elemente dieser Sichtbarkeitswechsel können aus den „abschließenden“ Shaft-Linien rekonstruiert werden. Dazu kann man nochmals Abb. 3.123 betrachten: die Shaft-Linie $\mathbf{(E_1, E_2)}$ des Kind-Shafts $\mathbf{EV_{10}}$ gehört zu Generator-Objekt **O**, also ist **O** das untere Extremal-Element des **EV**-Sichtbarkeitswechsels $\mathbf{EV_{10}}$.

An dieser Stelle bleibt noch die Frage offen, wie eine BB-Hierarchie anhand eines **EV**-Shaft-Baums klassifiziert werden kann. Bis jetzt wurden nur einseitige Hierarchien betrachtet, d.h. es wurde eine B-Box anhand eines einzelnen Shafts klassifiziert. Die Beantwortung dieser Frage wird auf Kapitel 4.3.3 verschoben. Dort wird die hierarchische Shaft-Klassifikation allgemein für Shafts aus Shaft-Ebenen und –Linien, die nicht alle in einer (**EV**-) Ebene liegen, diskutiert.

Zusammenfassend wurden also folgende Schritte zur Optimierung der Konstruktion der Vertex-MFLs durchgeführt:

- Die lokalen Sichtbarkeitstests wurden durch Klassifikationen anhand verallgemeinerter Vertex- und Kanten-Shafts ersetzt. Dadurch werden zu klassifizierende 3D-Entitäten in jedem Klassifikationsschritt nur anhand der Shaft-Ebenen neu klassifiziert, für die noch kein eindeutiges Ergebnis vorliegt.
- Für (gegenseitig lokal sichtbare) Kante-Vertex-Paare wurde der **EV**-Shaft definiert. Nur 3D-Entitäten „im“ **EV**-Shaft können mit dem Vertex-Kante-Paar **VEE**-/**EVE**-MFLs definieren. Auch hier wird die Klassifikation anhand der Shaft-Elemente nur ausgewertet, wenn die noch nicht eindeutig ist.
- Durch aufeinander folgende Klassifikationen von 3D-Objekten wird der **EV**-Shaft in einen Baum von Kind-Shafts aufgespalten und dadurch immer weiter beschränkt. Damit wird der Bereich, in dem andere 3D-Entitäten als „im“ **EV**-Shaft klassifiziert werden, immer kleiner.
- Jeder Kind-Shaft in diesem Baum enthält **VEE**-/**EVE**-MFLs. Für diese müssen keine Strahlanfragen mehr durchgeführt werden, da durch die Konstruktion des Baums die globale Sichtbarkeit garantiert und die Extremal-Elemente der MFLs bzw. Sichtbarkeitswechsel berechnet sind.

Dieses Optimierungsschema wird nun auch auf die Facetten- und Kanten-MFLs übertragen: lokale Sichtbarkeitstests werden durch Klassifikationen anhand von Element- Shafts ersetzt und für jedes Paar von Generator-Elementen wird der zugehörige verallgemeinerte Shaft definiert.

3.5.7.4 Facetten-MFLs

Um die Facetten-MFLs zu konstruieren, müssen für jede Facette der 3D-Szene zunächst die 3D-Objekte bzw. Kanten gefunden werden, die die Ebene der Facette schneiden. Dabei gibt es zunächst keine weiteren Einschränkungen, d.h. eine 3D-Entität liegt „im“ Facetten-Shaft, wenn sie die Ebene der Facette schneidet.

In Abb. 3.125 ist für eine Facette **F** die Menge der Schnittpolygone bzw. Kanten-Schnittpunkte von Generator-Objekten $\mathbf{O_i}$ mit der Ebene der Facette **F** dargestellt. Zur Konstruktion der Facetten-MFLs (**FvE**, **FEE**/**EFE** und **FF**) müssen…

- alle Vertices der Facette **F** mit allen Schnitt-Kanten auf **FvE**-Eigenschaft und
- alle Paare von Schnitt-Kanten auf **FEE**-/**EFE**-/**FF**-Eigenschaft getestet werden.

Bevor auf die konkrete Konstruktion der Facetten-MFLs eingegangen wird, sollte noch ein wesentlicher Unterschied zum Vorgehen bei den Vertex-MFLs hervorgehoben werden: bei den Vertex-MFLs muss die BB-Hierarchie für einen (Baum von) **EV**-Shaft(s) genau einmal durchlaufen werden, um alle Generator-Objekte anhand des (Baums von) **EV**-Shafts zu klassifizieren.

Bei den Facetten-MFLs müssen Paare von Schnittkanten betrachtet werden, d.h. ein analoges Vorgehen zu den Vertex-MFLs würde in einer zweifachen Traversierung und Klassifizierung der BB-Hierarchie sowie einer doppelten Berechnung der Schnitt-Kanten und –Polygone resultieren.

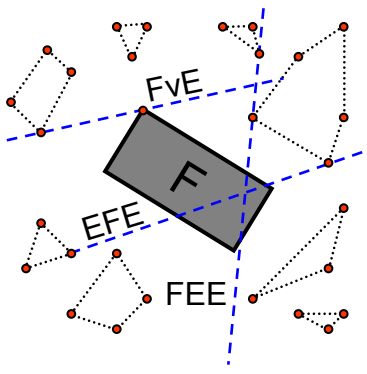


Abb. 3.125
Facette **F** mit Schnitt-Polygonen/-Kanten und einigen MFLs

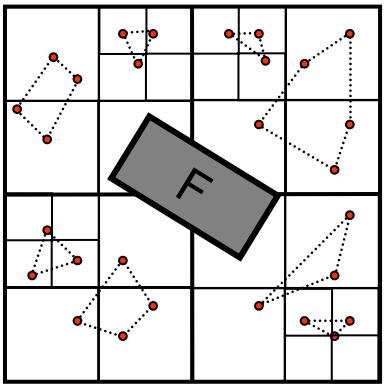


Abb. 3.126
Facette mit Schnitt-Polygonen und -Kanten sowie Quadtree (**m = 2**)

Daher wird hier ein anderer Ansatz gewählt: zuerst werden alle Schnitt-Kanten und –Polygone bezüglich der Facette **F** berechnet und in einem Quadtree **Q**, der in der Ebene der Facette **F** liegt, gespeichert. Eine Quadtree-Zelle wird dabei unterteilt, wenn sie mehr als **m** Kanten-Schnittpunkte enthält (vgl. Abb. 3.126).

Der Quadtree ersetzt in den folgenden Überlegungen quasi die bisher immer verwendete BB-Hierarchie. Prinzipiell kann man auch für die 2D-B-Boxen der Schnittpolygone jedes Mal eine neue 2D-BB-Hierarchie konstruieren. Der Vorteil des Quadtrees ist jedoch, dass er mit einer genügend großen Tiefe vorberechnet werden kann. Beim Einfügen der Schnitt-Polygone bzw. –Kanten in den Quadtree muss dann die Unterteilung des Quadtrees nicht neu berechnet werden, sondern lediglich traversiert werden.

Das folgende Code-Beispiel zeigt die Konstruktion der Facetten-MFLs:

```
private static QuadTree quadTree = new QuadTree(MAX_DEPTH)
void createFaceMFLs(Scene s)
    for each face f in Scene s
      List interPoly = (new FaceShaft(f)).getIntersectionPolys(s)
      quadTree.set(intPoly)
      for each polygon p in list interPoly
        for each vertex v of face f
           for each vertex e of polygon p: createFvE(f, v, e)
        for each edge f1 of polygon p
          if (f.id < f1.id) createFF(f, f1)
        createFEE_EFE_MFLs(f, quadTree)
```

Code-Beispiel 3.17

Für alle Facetten **f** der 3D-Szene wird zunächst der Facetten-Shaft gebildet und dann die 3D-Szene anhand dieses Shafts klassifiziert. Das Ergebnis der Klassifikation ist eine Liste **`interPoly`** von Schnittpolygonen. Diese Schnittpolygone werden in den (vorberechneten) Quadtree eingefügt. Dabei kann noch eine zusätzliche Optimierung angewendet werden (vgl. Abb. 3.127):

Jedes Schnitt-Polygon definiert einen von der Facette **F** global nicht sichtbaren Bereich (in Abb. 3.127 schraffiert), Quadtree-Zellen, die in diesem Bereich liegen, müssen nicht weiter untersucht werden, d.h. ihr Inhalt kann gelöscht werden, in sie kann nicht eingefügt werden und sie müssen in Zukunft auch nicht mehr traversiert werden.

Nach der Initialisierung des Quadtrees werden in Code-Beispiel 3.17 für jedes Schnitt-Polygon in der Liste **`interPoly`** (vgl. Abb. 3.128)…

- die **FvE**-MFLs berechnet (Kombinationen von Facetten-Vertices **v** und Polygon-Vertices (= Schnittkanten) **e** auf **FvE**-Eigenschaft prüfen)
- die **FF**-MFLs berechnet (Kanten **f1** des Schnittpolygons entsprechen Facetten des zugehörigen Schnitt-Objekts). Um doppelte MFLs zu vermeiden, werden nur Facetten-Paare **(f, f1)** mit **f.id < f1.id** betrachtet.
- die Berechnung der **FEE**-/**EFE**-MFLs angestoßen

Die Implementierung der Methoden zur Konstruktion der **FvE**- und **FF**-MFLs ist mit den bisherigen Überlegungen trivial und wird hier nicht angegeben. Die folgenden Überlegungen beziehen sich auf die Konstruktion der **FEE**-/**EFE**-MFLs unter Ausnutzung des Quadtrees.

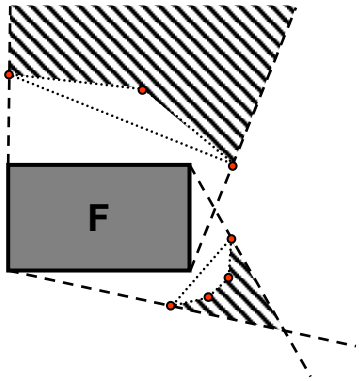


Abb. 3.127
Schnittpolygone definieren von Facette **F** global nicht sichtbare Bereiche

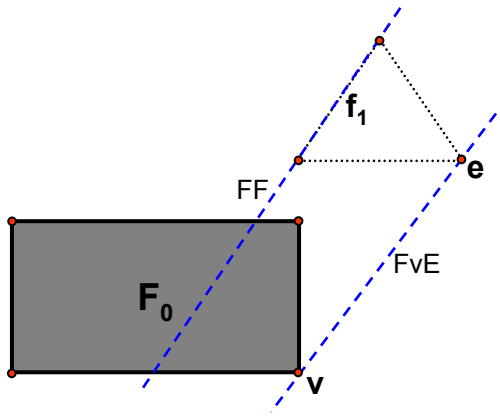


Abb. 3.128
Konstruktion von **FvE**- und **FF**-MFLs anhand eines Schnitt-Polygons

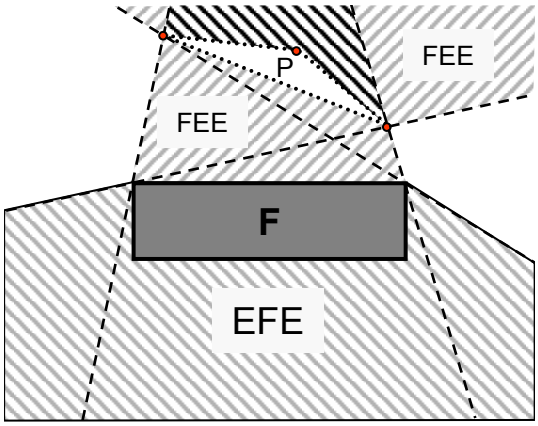


Abb. 3.129
Bereich für **FEE**-/**EFE**-MFLs für eine Facette **F** und ein Schnitt-Polygon **P**

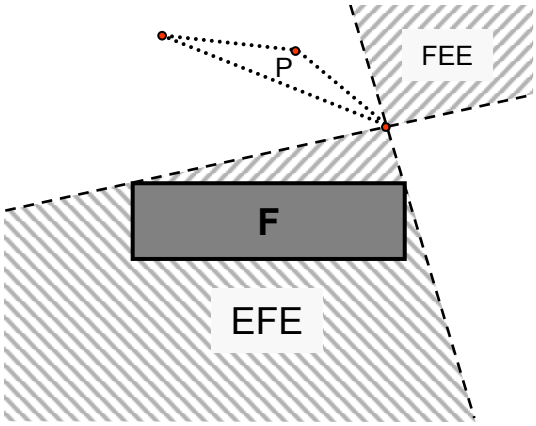


Abb. 3.130
Bereich für **FEE**-/**EFE**-MFLs für eine Facette **F** und eine Schnittkante **e**

Zunächst stellt man fest, dass jedes Schnitt-Polygon **P** mit der Facette **F** einen Bereich bildet, in dem **FEE**-/**EFE**-MFLs liegen können. Dieser Bereich entspricht der Menge aller Linien, die das Schnitt-Polygon **P** und die Facette **F** schneiden. In Abb. 3.129 ist dieser Bereich schraffiert dargestellt: der Bereich wird von insgesamt vier Linien begrenzt. Alle diese Linien haben die Eigenschaft, dass sie die Facette **F** nur in einem Vertex und das Schnitt-Polygon **P** nur in einem Vertex (d.h. einer Schnittkante des zugehörigen 3D-Objekts) schneiden.

Diese Linien definieren wieder einen verallgemeinerter Shaft, den Facette-3D-Objekt-Shaft (**FO**-Shaft). Nur Quadtree-Zellen, die (teilweise) im **FO**-Shaft liegen, können Schnittkanten enthalten, die mit der Facette **F** und den Schnitt-Kanten des Schnitt-Objekts **FEE**-/**EFE**-MFLS definieren können.

Diese Überlegung kann für die einzelnen Schnitt-Kanten des Schnitt-Objekts fortgeführt werden (vgl. Abb. 3.130): jede Schnittkante definiert mit der Facette **F** einen Bereich, in dem **FEE**-/**EFE**-MFLs liegen können, den Facetten-Kante-Shaft (**FE**-Shaft).

Damit ergibt sich eine zweistufige Hierarchie: Nur 3D-Objekte „im“ **FO**-Shaft können mit den Kanten des Schnittpolygons **FEE**-/**EFE**-MFLs definieren und für jede einzelne Kante muss das 3D-Objekt „im“ **FE**-Shaft liegen. Die Hierarchie kann wie folgt erweitert werden:

Die Liste der Schnittkanten des **FO**-Shafts wird halbiert und für jede Hälfte der **FE***-Shaft gebildet (vgl. Abb. 3.131). Dies wird rekursiv wiederholt, dadurch entsteht ein Baum von Facetten-Shafts (vgl. Abb. 3.132): die Wurzel des Baums ist der **FO**-Shaft, die inneren Knoten sind **FE***-Shafts und die Blätter sind **FE**-Shafts.

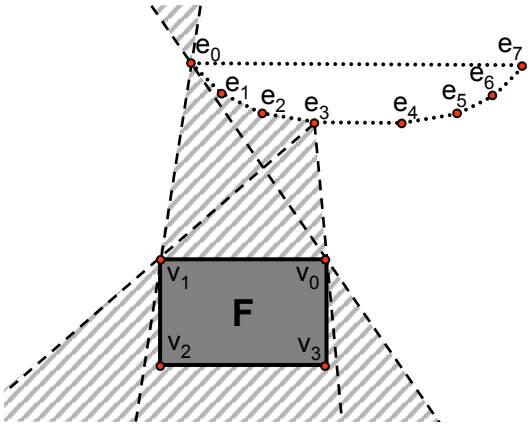


Abb. 3.131
FE*-Shaft der Schnittkanten **e0**, …, **e3**

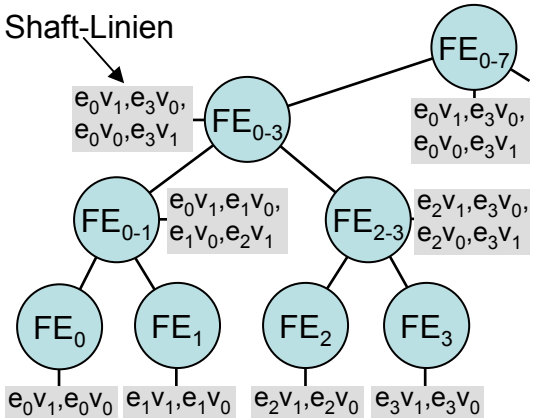


Abb. 3.132
Baum von Facetten-Shafts

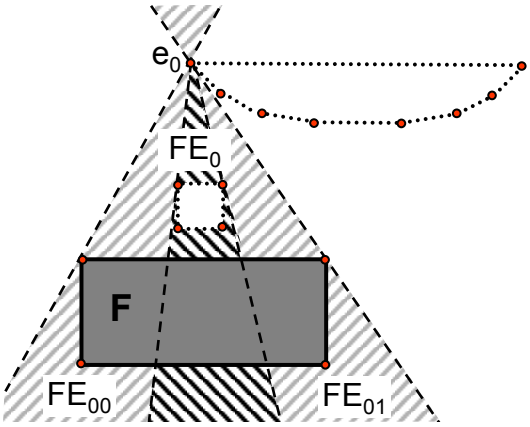


Abb. 3.133
Aufspaltung eines **FE**-Shafts in Kind-Shafts

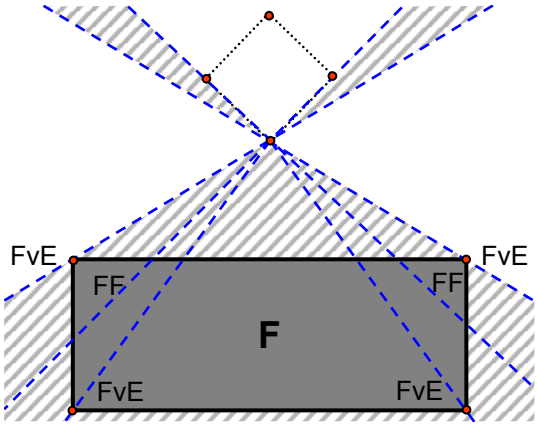


Abb. 3.134
Einfügen von **FvE**-/**FF**-MFLs in **FE**-Baum

Für die Blatt-Shafts kann man völlig analog zu den **EV**-Shafts vorgehen, d.h. der **FE**-Shaft wird durch Klassifikationen von 3D-Objekten in einen Baum von Kind-**FE**-Shafts aufgespalten (vgl. Abb. 3.133). Analog zu den Vertex-MFLs entsprechen die Begrenzungslinien der (Kind-) **FE**-Shafts den **FEE**-/**EFE**-MFLs. Für diese müssen keine weiteren Operationen zur Feststellung der globalen Sichtbarkeit und der Berechnung der Extremal-Elemente durchgeführt werden.

Die Klassifikation von 3D-Entitäten anhand dieses Baums von $\mathbf{FE}^{(*)}$-Shafts wird erst bei der Betrachtung des hierarchischen Shaft-Cullings in Kap. 4.3.3 diskutiert, da dort der allgemeinere Fall behandelt wird.

Als letzten Schritt der Optimierung können auch die in Code-Beispiel 3.17 noch getrennt berechneten **FvE**- und **FF**-MFLs in die zugehörigen **FE**-Shafts eingefügt und bei der Aufspaltung mitgeführt werden. Auch für die so berechneten **FvE**- und **FF**-MFLs müssen dann keine weiteren Raycasting-Operationen durchgeführt werden.

Zusammenfassend wurden also folgende Schritte zur Optimierung der Konstruktion der Facetten-MFLs unternommen:

- Die Klassifikation anhand der Ebene der Generator-Facette **F** und die Konstruktion der Schnitt-Polygone wurden durch einen Facetten-Shaft realisiert. Dadurch kann bei der Klassifikation die BB-Hierarchie ausgenutzt werden.
- Die Schnitt-Polygone wurden in einem Quadtree gespeichert, der die bisher verwendete BB-Hierarchie ersetzt. Jedes Polygon im Quadtree definiert …

- einen von der Facette **F** global nicht sichtbaren Bereich. Andere Schnitt-Polygone in diesem Bereich müssen nicht weiter betrachtet werden.
- einen Bereich, in dem **FEE**-/**EFE**-MFLs mit den Schnitt-Kanten des Schnitt-Polygons liegen können

- Diese Überlegungen wurden auf den Quadtree übertragen. Es werden nur Zellen traversiert, die **FEE**-/**EFE**-MFLs mit den Schnitt-Kanten des Schnitt-Polygons definieren können.
- Die Klassifikation anhand einzelner **FE**-Shafts wurde durch eine rekursive Klassifikation anhand einer Hierarchie von **FO**-, **FE***- und **FE**-Shafts ersetzt.
- Die **FE**-Shafts können analog zu den **EV**-Shafts rekursiv in Kind-Shafts aufgespalten werden. Die Begrenzungslinien dieser (Kind-) Shafts sind **FEE**-/**EFE**-MFLs. Für diese müssen keine weiteren Strahlanfragen durchgeführt werden.
- Auch die **FvE**- und **FF**-MFLs durch Einfügen in die **FE**-Shafts konstruiert werden.

Damit liegt eine performante Methode zur Konstruktion aller Facetten-MFLs einer 3D-Szene vor. In Kap. 3.5.8.2 wird der Aufwand für die Aufspaltung in Kind-Shafts für die **VE**- und **FE***-Shafts anhand von Experimenten mit konkreten 3D-Szenen untersucht.

3.5.7.5 Kanten-MFLs

Die Optimierung der Konstruktion der Kanten-MFLs ist nur für die **E4**-MFLs sinnvoll, die Konstruktion der **E**-MFLs ist trivial und wird hier nicht weiter diskutiert.

Ein **E4**-MFL ist eine Linie, die durch vier Kanten der 3D-Szene geht. Mit den Überlegungen aus Kap. 3.5.6.3 gilt, dass vier Kanten $\mathbf{e_0}$, …, $\mathbf{e_3}$ nur dann ein **E4**-MFL bilden können, wenn sie gegenseitig in ihren **EE**(**E**)-Keilen $\mathbf{e_i e_j}$ bzw. $\mathbf{e_i e_j e_k}$ liegen. Diese **EE**(**E**)-Keilen bestehen aus einer Menge von Ebenen, die jeweils von einem Vertex einer Kante $\mathbf{e_i}$ und einer zweiten Kante $\mathbf{e_j}$ aufgespannt werden.

Als ersten Schritt der Optimierung können diese **EE**(**E**)-Keile als verallgemeinerte **EE**(**E**)-Shafts realisiert werden. Dadurch werden 3D-Entitäten anhand der Shaft-Ebenen nur neu klassifiziert, wenn für sie noch kein eindeutiges Ergebnis vorliegt.

Im Gegensatz zu den bisher betrachteten Shafts sind diese **EE**(**E**)-Shafts dynamisch, d.h. sie müssen nicht mit allen Generator-Elementen initialisiert werden, sondern können schrittweise durch Hinzufügen von Kanten von **E**- zu **EE** - und weiter zu **EEE**-Shafts erweitert werden. Sie werden daher hier als **E***-Shafts bezeichnet.

Diese Erweiterung des **E***-Shafts entspricht einer Schrittweisen Beschränkung des Bereichs, der als „im“ Shaft klassifiziert wird: anhand eines **E***-Shafts mit **0 < n < 4** Generator-Kanten $\mathbf{e_i}$ wird eine 3D-Entität **G** als „im“ Shaft klassifiziert, wenn…

- **G** von allen **n** Generator-Kanten $\mathbf{e_i}$ (teilweise) lokal sichtbar ist
- Es eine Linie gibt, die durch alle **n** Generator-Kanten $\mathbf{e_i}$ und die 3D-Entität **G** geht

Weiterhin können die **E***-Shafts im Gegensatz zu den oben diskutierten Vertex- und Kanten-Shafts nicht in Kind-Shafts aufgespalten werden. Der Grund hierfür ist leicht einzusehen: Sowohl bei den Vertex- als auch den Facetten-MFLs hatte jeder Shaft ein eindeutiges Projektionszentrum, d.h. einen einzelnen Punkt, durch den alle MFLs verlaufen. Damit konnten diese Shafts auch eindeutig in Kind-Shafts aufgespalten werden, da für jeden Punkt im Shaft bestimmt werden kann, ob er vom Projektionszentrum sichtbar ist oder nicht. Im Falle der **E***-Shafts existiert jedoch kein eindeutiger Punkt durch den alle **E4**-MFLs verlaufen.

Daher wird an dieser Stelle vom bisherigen Optimierungsschema abgewichen und eine eigenständige Behandlung für die **E***-Shafts bzw. **E4**-MFLs vorgeschlagen. Wesentliches Ziel dabei ist den globalen Sichtbarkeitstest so früh wie möglich, d.h. noch vor der Konstruktion der eigentlichen MFLs, durchzuführen und somit die Konstruktion ungültiger MFLs zu vermeiden. Diese Optimierung wird in drei Schritten durchgeführt, die zunächst einzeln erläutert und dann in ein gemeinsames Schema eingebunden werden:

(1) Erkennung vollständiger oder teilweiser Verdeckung von **EE**-Shafts

(2) Erkennung vollständiger oder teilweiser Verdeckung von **EEE**-Shafts

(3) Auswahl der Generator-Kandidaten für **EEE**-Shafts

Für die folgenden Ausführungen wird nur der „innere“ Teil des **EE**-Shafts $\mathbf{S(e_i, e_j)}$ betrachtet, d.h. der Tetraeder $\mathbf{T(e_i, e_j)}$, der von den Kanten $\mathbf{e_i}$ und $\mathbf{e_j}$ aufgespannt wird (vgl. Abb. 3.135).

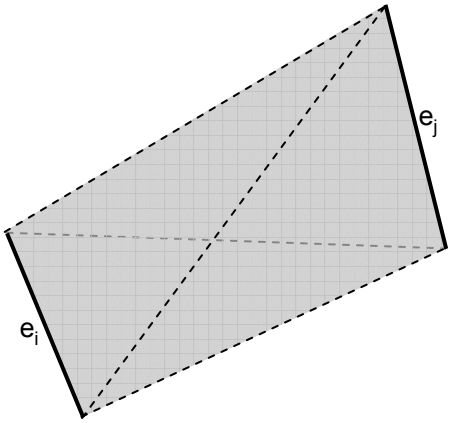


Abb. 3.135
„Innerer“ Teil des **EE**-Shafts **$S(e_i, e_j)$** entspricht Tetraeder **$T(e_i, e_j)$**

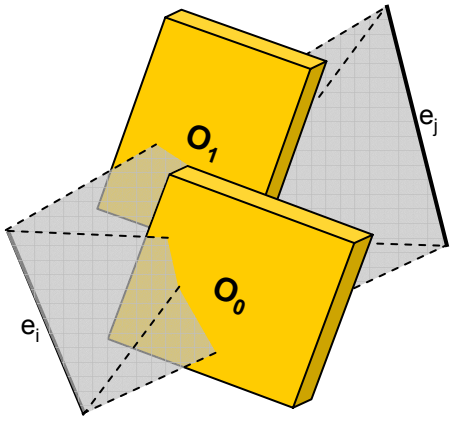


Abb. 3.136
EE-Shaft **$T(e_i, e_j)$** wird durch zwei 3D-Objekte vollständig verdeckt

Der einfachste Schritt ist offensichtlich die Erkennung der vollständigen Verdeckung von **EE**-Shafts: Wenn ein einzelnes 3D-Objekt **O** einen **EE**-Shaft **$T(e_i, e_j)$** vollständig verdeckt, d.h. alle seine Shaft-Linien schneidet, so sind die Kanten **e_i** und **e_j** gegenseitig global nicht sichtbar und können somit auch keine gültigen **E4**-MFLs definieren.

Wurde kein 3D-Objekt gefunden, das alleine den **EE**-Shaft vollständig verdeckt, so muss getestet werden, ob der **EE**-Shaft durch Kombinationen von 3D-Objekten verdeckt wird (vgl. Abb. 3.136).

Schneidet ein 3D-Objekt **O** zwei Shaft-Linien des Tetraeders, so beschränkt dieses 3D-Objekt den Tetraeder. Anschaulich bedeutet dies, dass eine der Kanten **e_i** oder **e_j** verkürzt werden kann und somit der Tetraeder kleiner wird, also auch weniger 3D-Objekte als „im“ Tetraeder klassifiziert werden. Dies ist in Abb. 3.137 dargestellt: zwei Shaft-Linien des Tetraeders **$T(e_i, e_j)$** werden vom 3D-Objekt **O** geschnitten. Diese Beschränkung führt zu einer Verkürzung der Kante **e_i** zu einer Kante **e_i'** und somit zu einer Verkleinerung des Tetraeders.

Der Anteil der verkürzten Kante **e_i'** entspricht dabei dem von der Kante **e_j** sichtbaren Bereich der ursprünglichen Kante **e_i**. Dieser Anteil ergibt sich durch Projektion des 3D-Objekts **O** vom Start- und Endpunkt der Kante **e_i** auf die Kante **e_j**.

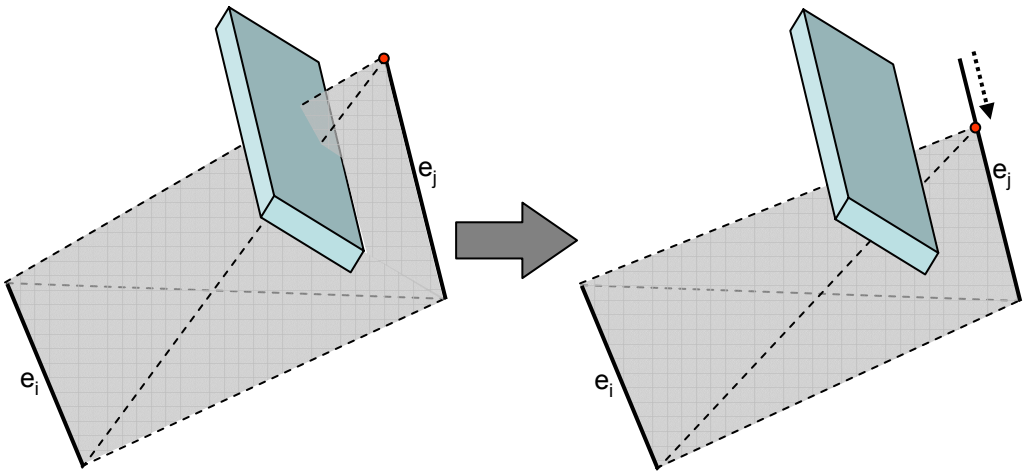


Abb. 3.137
3D-Objekt **O** beschränkt Tetraeder **T(e_i, e_j)**

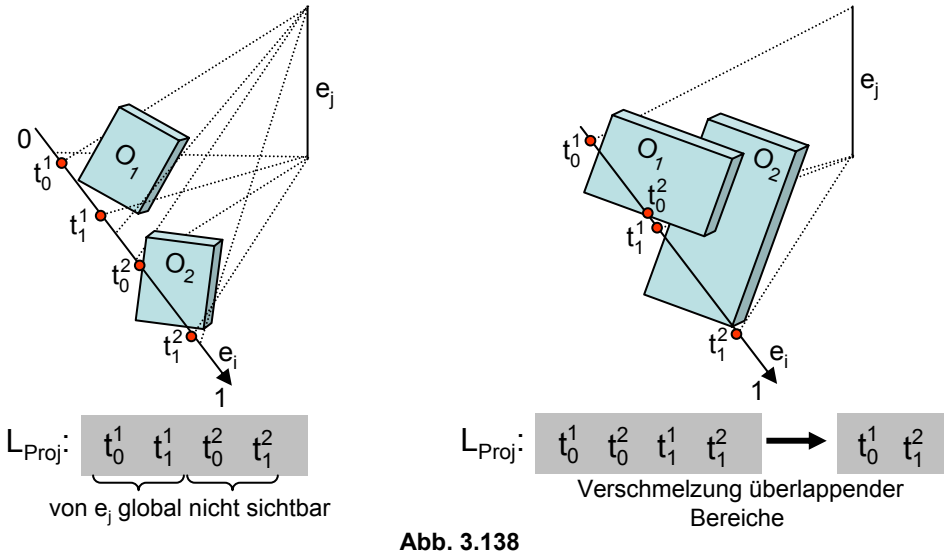


Abb. 3.138
3D-Objekte, die gegenüberliegende Ebenen von **T(e_i, e_j)** schneiden, definieren nicht sichtbare Bereiche

Auf diese Weise wird also der **EE**-Shaft zweier Kanten e_i und e_j durch schrittweises Einfügen von 3D-Objekten immer weiter beschränkt. Im besten Fall wird der **EE**-Shaft so ungültig, d.h. vollständig verdeckt und muss nicht weiter betrachtet werden.

Schneidet das 3D-Objekt zwei gegenüberliegende Ebenen des **EE**-Shafts (vgl. Abb. 3.138), so wird damit ein von der Kante e_j global nicht sichtbarer Teil auf der Kante e_i definiert, d.h. die Kante e_i wird in zwei Teilkanten e_i' und e_i'' aufgespalten. Analog zu oben kann der global nicht sichtbare Teil der Kante e_i durch Schnitt der Projektionen des 3D-Objekts **O** vom Start- und End-Vertex von e_j berechnet werden. Werden auf diese Weise **k** 3D-Objekte auf die Kante e_i projiziert, so entsteht eine (sortierte) Liste L^i_{Proj} von Projektionen (vgl. Abb. 3.138). Die Einträge der Liste sind Tupel der Form `(bStart, para)`, d.h. ein boolscher Wert, der angibt, ob der Eintrag den Beginn oder

das Ende eines global nicht sichtbaren Bereichs definiert und einem Parameter, der den Projektionspunkt auf der Kante beschreibt. Durch eine einfache Traversierung dieser Liste können mit linearem Aufwand überlappende global nicht sichtbare Bereiche zusammengefasst werden und somit eine minimale Repräsentation der global nicht sichtbaren Bereiche gewonnen werden.

Auf ähnliche Weise kann die teilweise oder vollständige Verdeckung von **EEE**-Shafts behandelt werden. Dazu kann zunächst die Ausgangssituation betrachtet werden: in einem (evtl. zuvor beschränkten) **EE**-Shaft $\mathbf{T(e_i, e_j)}$ liegt eine Kante $\mathbf{e_k}$. Der erste Schritt ist eine Beschränkung des **EE**-Shafts auf diese Kante (vgl. Abb. 3.139). Durch diese Beschränkung entspricht die Kante $\mathbf{e_k}$ einer Diagonalen durch den Tetraeder $\mathbf{T(e_i, e_j)}$, d.h. $\mathbf{e_k}$ schneidet alle vier Ebenen des Tetraeders und zwei seiner Kanten.

In Abb. 3.140 sind die Shaft-Linien des **EEE**-Shafts $\mathbf{S(e_i, e_j, e_k)}$ dargestellt. Wie man sieht, entspricht der **EEE**-Shaft $\mathbf{S(e_i, e_j, e_k)}$ zwei **EE**-Shafts $\mathbf{T(e_i, e_k)}$ und $\mathbf{T(e_k, e_j)}$.

Die Beschränkung dieses **EEE**-Shafts verläuft ähnlich wie die des **EE**-Shafts: jedes 3D-Objekt, dass zwei Shaft-Linien des **EEE**-Shafts schneidet, verkleinert den **EEE**-Shaft. Im Unterschied zu den **EE**-Shafts wird der **EEE**-Shaft jedoch auf diese Weise „auf beiden Seiten“ verkleinert, d.h. es werden alle drei Kanten $\mathbf{e_i}$, $\mathbf{e_j}$ und $\mathbf{e_k}$ verkürzt.

Dies liegt am Konstruktionsprinzip der **EEE**-Shafts: wenn drei Kanten einen **EEE**-Shaft bilden, so müssen sie gegenseitig in ihren **EE**-Shafts liegen. Eine Beschränkung eines dieser **EE**-Shafts führt direkt zu einer Beschränkung der anderen **EE**-Shafts.

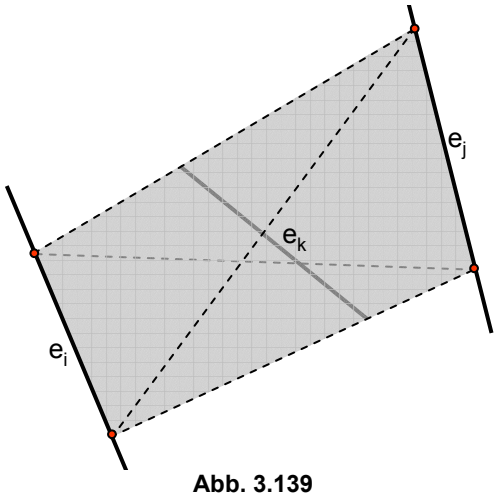


Abb. 3.139
Beschränkung von $\mathbf{T(e_i, e_j)}$ auf eine Kante $\mathbf{e_k}$

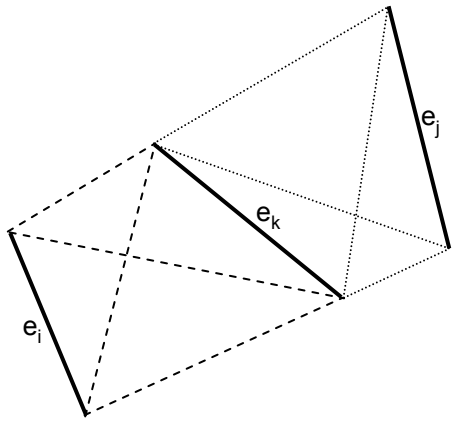


Abb. 3.140
Shaft-Linien des **EEE**-Shafts $\mathbf{S(e_i, e_j, e_k)}$

Dies ist in Abb. 3.141 dargestellt: der **EEE**-Shaft **S(e_i, e_j, e_k)** wird von einem 3D-Objekt **O** geschnitten und so zunächst analog zu den **EE**-Shafts die Kante e_i verkürzt. Anschaulich hat sich so zunächst der **EE**-Shaft **T(e_i, e_j)** verkleinert. Dies führt im nächsten Schritt zu einer Verkürzung der Kante e_k, da diese vollständig im **EE**-Shaft **T(e_i, e_j)** liegen muss. Dies wiederum führt zu einer Verkürzung der Kante e_j, da sie vollständig im **EE**-Shaft **T(e_i, e_k)** liegen muss.

Mit diesen Schritten können also die **E***-Shafts ähnlich zu den Vertex- und Facetten-Shafts schrittweise beschränkt und somit Generator-Kandidaten ausgeschlossen werden. Damit bleibt allerdings noch die Frage, wie, d.h. in welcher Reihenfolge, diese schrittweise Beschränkung durchgeführt werden sollte. Eine sinnvolle Beschränkungs-Reihenfolge ist, zuerst die 3D-Objekte in den **EE**-Shaft einzufügen, die diesen maximal beschränken oder sogar vollständig verdecken.

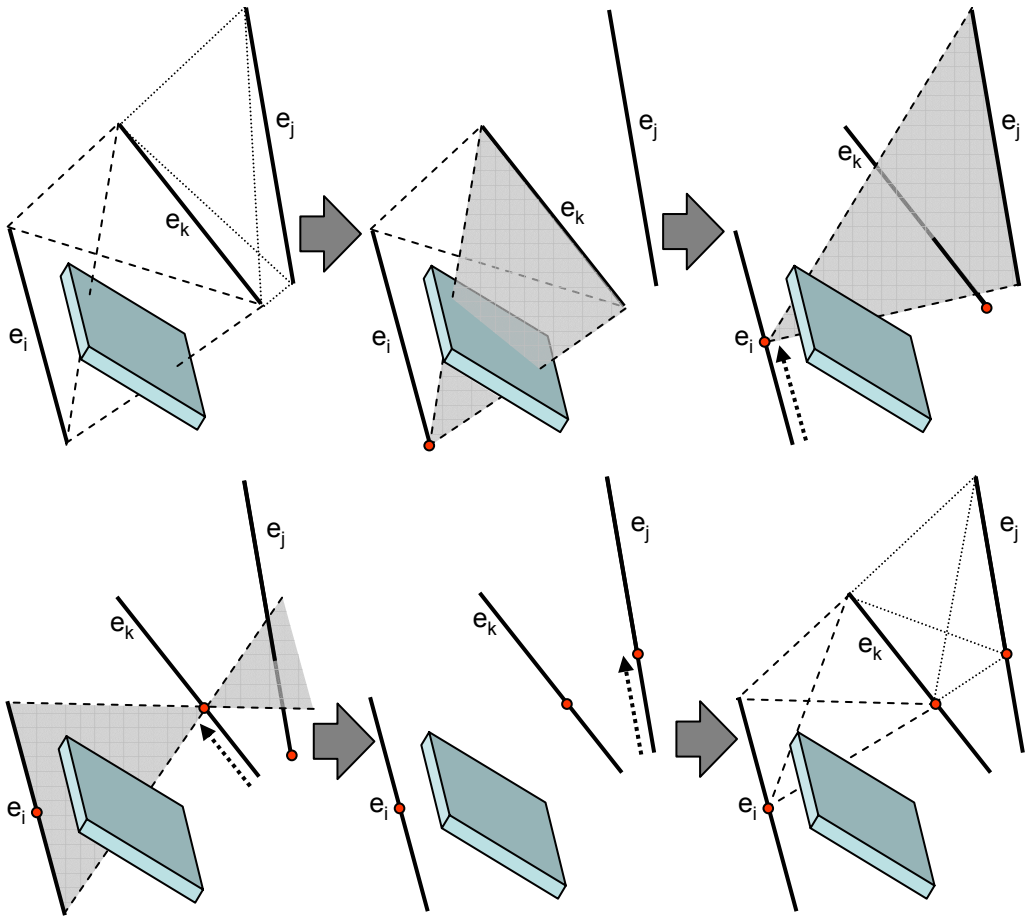


Abb. 3.141
Verkürzung einer Kante eines **EEE**-Shafts führt zu Verkürzungen der beiden anderen Kanten

Bisher wurde bei der Shaft-Klassifikation die BB-Hierarchie einfach in Pre-Order-Reihenfolge traversiert, d.h. die B-Boxen bzw. 3D-Objekte wurden in fester, durch die Struktur der BB-Hierarchie vorgegebener Reihenfolge anhand des Shafts klassifiziert. Für eine effiziente Beschränkung der **EE**-Shafts sollte die BB-Hierarchie dagegen in einer Reihenfolge traversiert werden, die die Zahl der jeweils geschnittenen Shaft-Linien maximiert.

Da ein **EE**-Shaft vier Shaft-Linien enthält, kann diese Traversierung mit Hilfe einer Liste **`stack`** von fünf Stacks umgesetzt werden, wobei jeweils der **k**-te Stack **`stack[k]`** nur B-Boxen bzw. 3D-Objekte enthält, die **k** Shaft-Linien schneiden. Das folgende Code-Beispiel zeigt eine mögliche Umsetzung dieser Traversierung:

```
void traverse(EEShaft shaft, BBox root)
     Stack[] stack      = new Stack[5]
     List    candidates = new List();

     stack[4].push(root)

     while (stacks [1] – [4] not empty)
        for (int sIndex = 4; sIndex > 0; sIndex--)
          while (stack[sIndex] not empty)
            box      = stack[sIndex].pop()
            numLines = shaft.getIntersectedShaftLines(box).length

            if (numLines > 1)
              if (box.isLeaf())
                if (shaft.restrict(box.getObject()))
                  shaft[0].push(box)
                else return
              else shaft[numLines].pushChildren(box)
            else shaft[numLines].push(box)

     while(stack[0] not empty)
        box = stack[0].pop()
        if (shaft.restrict(box.getObject()))
          candidates.add(box)
        else return
```

Code-Beispiel 3.18

Am Ende dieser Traversierung wurde der Shaft von allen 3D-Objekten, die zwei oder mehr Shaft-Linien schneiden, beschränkt und **`stack[0]`** enthält alle 3D-Objekte bzw. Bounding-Boxen, die keine Shaft-Linien schneiden. Zu beachten ist hierbei, dass durch die schrittweise Beschränkung der **EE**-Shafts und die daraus folgende Änderung der Shaft-Linien die Belegung der Stacks inkonsistent werden kann, d.h. eine B-Box bzw. ein 3D-Objekt mehr oder weniger Shaft-Linien schneidet, als durch die Stack-Position angezeigt wird.

Daher werden im obigen Code-Beispiel die Stacks so lange abgearbeitet, bis die Stacks 1 – 4 leer sind, d.h. keine weiteren Beschränkungen vorkommen können. Als letzter Schritt wird dann nochmals `stack[0]` abgearbeitet werden, da sich durch die Beschränkung des Shafts auch hier neue Schnitte mit den Shaft-Linien ergeben können.

Die so definierte Beschränkung der **E***-Shafts kann nur dann sinnvoll angewendet werden, wenn es genügend viele 3D-Objekte gibt, die zwei oder mehr Shaft-Linien schneiden. Sind die Generator-Kanten des **EE**-Shafts dagegen sehr groß gegenüber den 3D-Objekten, so bietet das obige Verfahren keine Vorteile. Da die Zahl der potentiellen **E4**-MFLs in einem **EE**-Shaft aber quadratisch mit der Zahl der Kandidaten-Kanten wächst, wird im Folgenden auch für den Sonderfall „großer" Generator-Kanten eine einfache Optimierung vorgeschlagen.

Jede Kante $\mathbf{e_k}$ in einem **EE**-Shaft $\mathbf{T(e_i,\ e_j)}$ definiert Intervalle $\mathbf{I_k^i = [a_i, b_i]}$ und $\mathbf{I_k^j = [a_j, b_j]}$ auf den Kanten $\mathbf{e_i}$ und $\mathbf{e_j}$, durch die **E4**-MFLs verlaufen können (vgl. Abb. 3.139). Wenn eine vierte Kante $\mathbf{e_l}$ mit $\mathbf{e_i}$, $\mathbf{e_j}$ und $\mathbf{e_k}$ ein **E4**-MFL definieren kann, so müssen die Intervalle $\mathbf{I_k^i}$ und $\mathbf{I_l^i}$ sowie $\mathbf{I_k^j}$ und $\mathbf{I_l^i}$ einen nicht-leeren Schnitt bilden.

Zwei eindimensionale Intervalle $\mathbf{I_k^i}$ und $\mathbf{I_k^j}$ können als zweidimensionales Intervall $\mathbf{R_k^{ij} = I_k^i \times I_k^j}$ betrachtet werden, d.h. anschaulich als ein Rechteck im 2D. Jede Kante $\mathbf{e_k}$ im **EE**-Shaft $\mathbf{T(e_i,\ e_j)}$ definiert also ein Rechteck $\mathbf{R_k^{ij}}$ im 2D und zwei Kanten $\mathbf{e_k}$ und $\mathbf{e_l}$ können nur dann mit $\mathbf{e_i}$ und $\mathbf{e_j}$ ein **E4**-MFL bilden, wenn sich die zugehörigen Rechtecke $\mathbf{R_k^{ij}}$ und $\mathbf{R_l^{ij}}$ schneiden (vgl. Abb. 3.142).

Das Problem, alle Paare von Kanten $\mathbf{(e_k,\ e_l)}$ im **EE**-Shaft $\mathbf{T(e_i,\ e_k)}$ zu finden, die **E4**-MFLs definieren können, reduziert sich somit also auf die Bestimmung aller Schnitte von Rechtecken $\mathbf{R_k^{ij}}$ und $\mathbf{R_l^{ij}}$. Zur Lösung dieses Schnittproblems existieren sehr viele verschiedene Ansätze (vgl. [43]), zu den bekanntesten gehören die Verwendung von R-Bäumen, Quadtrees oder Plane-Sweep-Verfahren.

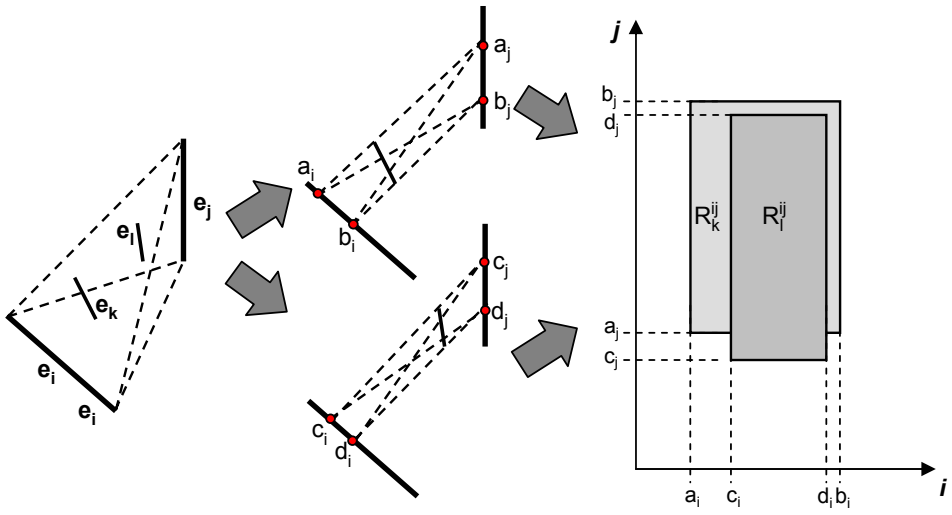


Abb. 3.142
Beschränkung von $\mathbf{T(e_i, e_j)}$ auf zwei Kanten $\mathbf{e_k}$ und $\mathbf{e_l}$ ergibt zwei Rechtecke im 2D

Hier soll nur das Plane-Sweep-Verfahren diskutiert werden, da es mit wenigen, elementaren Operationen auskommt und somit auch bei relativ kleinen Anzahlen von Rechtecken recht gute absolute Laufzeiten ergibt. Die folgende Darstellung des Plane-Sweep-Verfahresn ist im Wesentlichen [43] entnommen, dort werden auch Details der Implementierung diskutiert.

Gegeben sei also eine Menge **R** von **m** Rechtecken $\mathbf{R_k}$. Die grundlegende Idee des Plane-Sweep-Verfahrens ist, eine senkrechte Linie **L** „von links nach rechts“ über die Ebene der Rechtecke wandern zu lassen und dabei in jedem Schritt eine Menge von aktiven Rechtecken konsistent zu halten. Ein Rechteck $\mathbf{R_k}$ ist aktiv, wenn **L** das Rechteck schneidet (vgl. Abb. 3.143). Schnitte zwischen zwei Rechtecken können dabei nur vorkommen, wenn beide Rechtecke gleichzeitig aktiv sind.

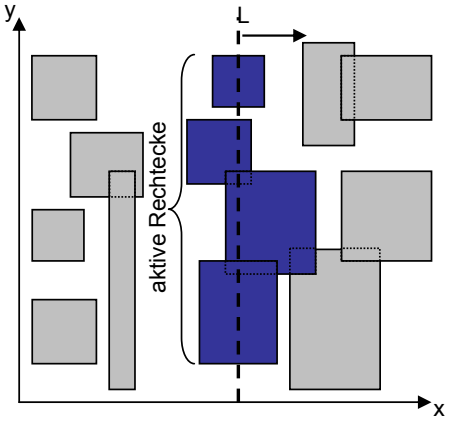


Abb. 3.143
Plane-Sweep mit Menge aktiver Rechtecke

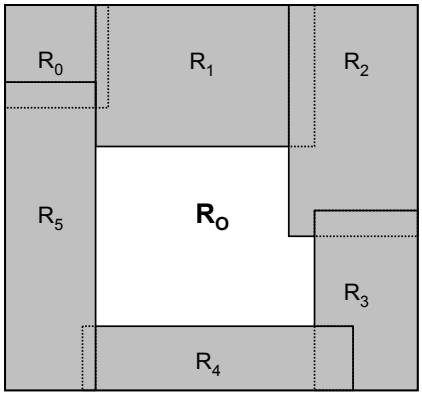


Abb. 3.144
Objekt-Rechteck $\mathbf{R_O}$ enthält Kanten-Rechtecke $\mathbf{R_k}$

Zur Umsetzung des Plane-Sweep-Verfahrens ist zunächst eine Sortierung der Rechtecke nach ihrer minimalen **x**-Koordinate mit Aufwand **O(m·ld m)** nötig. Bei der anschließenden Traversierung der Rechtecke „hält“ die Linie **L** bei jedem Ein- bzw. Austritt eines Rechtecks $\mathbf{R_k}$ an und fügt $\mathbf{R_k}$ der Menge der aktiven Rechtecke hinzu bzw. entfernt $\mathbf{R_k}$ aus der Menge der aktiven Rechtecke.

Um die Schnittberechnungen zwischen aktiven Rechtecken zu beschleunigen, ist die Menge der aktiven Rechtecke mittels eines binären Suchbaums nach der minimalen **y**-Koordinate sortiert. Jedes Rechteck, das neu in die Menge der aktiven Rechtecke aufgenommen wird, muss somit in den Suchbaum eingefügt und gleichzeitig mit den Rechtecken im Suchbaum auf Schnitte geprüft werden. Für das Einfügen in den Suchbaum ist insgesamt ein Aufwand von **O(m·ld m)** notwendig, für den Test auf Schnitte ein Aufwand von **O(m·ld m + M)**, wobei **M** die Gesamtzahl der Rechteckschnitte ist.

Jedes so gefundene Paar von sich schneidenden Rechtecken kann dann mit den bisher erarbeiteten Verfahren auf Existenz eines **E4**-MFL geprüft werden. Um die globale Sichtbarkeit dieses MFL zu testen, kann das Plane-Sweep-Verfahren auf einfache Weise erweitert werden: Bisher wurden nur (die Rechtecke) einzelne(r) Kanten betrachtet. Dies kann direkt auf komplette 3D-Objekte verallgemeinert werden, indem für jedes 3D-Objekt „im“ **EE**-Shaft die Menge der Kanten-Rechtecke $\mathbf{R_k}$ gebildet und dann das diese Rechtecke umfassende Objekt-Rechteck $\mathbf{R_O}$ bestimmt wird (vgl. Abb. 3.144).

Jeder Schnitt eines Kanten-Rechtecks mit einem Objekt-Rechteck entspricht dann einem Schnitt eines 3D-Objekts mit einem **EEE**-Shaft, d.h. die zuvor erarbeitete Beschränkung von **EEE**-Shafts kann ohne Änderungen auf das Plane-Sweep-Verfahren übertragen werden.

Dabei können weiterhin auch die globalen Sichtbarkeitsinformationen aus den Listen $\mathbf{L^i_{Proj}}$ und $\mathbf{L^j_{Proj}}$ (vgl. Abb. 3.138) verwertet werden: Jeder Eintrag in den beiden Listen definiert einen global nicht sichtbaren Bereich auf einer der beiden Kanten, dieser Bereich entspricht wiederum einem Rechteck im 2D. Liegt eines der Recktecke $\mathbf{R_k}$ vollständig in einem global nicht sichtbaren Bereich, so muss es nicht weiter bearbeitet werden.

Damit ist die Optimierung der Konstruktion der Kanten-MFLs abgeschlossen. Zusammenfassend wurden folgende Schritte durchgeführt:

- Die **EE**- bzw. **EEE**-Keile wurden als verallgemeinerte **E***-Shafts umgesetzt. Dadurch werden Klassifikationen anhand der Shaft-Ebenen und Linien nur dann ausgewertet, wenn sie noch nicht eindeutig bestimmt sind.
- Die **EE**-Shafts wurden durch schrittweises Einfügen der 3D-Objekte beschränkt. Dadurch werden im besten Fall ganze **EE**-Shafts ungültig, ansonsten enthalten die beschränkten **EE**-Shafts weniger 3D-Objekte als die unbeschränkten.
- Die Generator-Kanten der **EE**-Shafts wurden durch schrittweises Einfügen der 3D-Objekte in global (nicht) sichtbare Teilkanten aufgespalten. Die global nicht sichtbaren Teilkanten können mit linearem Aufwand verschmolzen werden.
- Das Beschränkungsschema wurde auf die **EEE**-Shafts übertragen. Durch den Aufbau der **EEE**-Shafts wirkt sich die Beschränkung einer Kante auf die beiden anderen Kanten aus und verkleinert somit die **EEE**-Shafts schneller als die **EE**-Shafts.
- Die feste Traversierungsreihenfolge der BB-Hierarchie wurde an die obige Problemstellung angepasst. Dadurch werden zuerst die B-Boxen bzw. 3D-Objekte traversiert, die die **EE**(**E**)-Shafts maximal beschränken.
- Für **EE**-Shafts, die sehr viele 3D-Objekte bzw. Kanten enthalten, wurde ein eigenes Optimierungsschema erarbeitet. Grundlegend hierfür ist, dass jede Kante „in“ einem **EE**-Shaft auf den Generator-Kanten des **EE**-Shafts Intervalle definiert, durch die **E4**-MFLs verlaufen können. Diese Intervalle entsprechen Rechtecken im 2D und können mit einem angepassten Plane-Sweep-Verfahren auf Schnitte, d.h. Existenz von **E4**-MFLs getestet werden. Der Aufwand für dieses Verfahren bei **m** Kanten bzw. Kanten-Rechtecken „im“ **EE**-Shaft ist **O(m·ld m + M)**, wenn **M** die Gesamtzahl der schneidenden Rechtecke ist.

  Auch für global nicht sichtbare Teilkanten wurden Rechtecke definiert. Intervall-Rechtecke, die vollständig in diesen liegen, müssen nicht weiter bearbeitet werden.

Die Auswirkungen dieser Optimierungsschritte werden im nächsten Kapitel anhand von Experimenten und Messungen mit konkreten 3D-Szenen näher untersucht und beurteilt.

3.5.8 Experimente und Messungen

In diesem Teilkapitel sollen die erarbeiteten Optimierungen für die Konstruktion des Visibility Skeletons einer 3D-Szene untersucht werden. Dazu werden zunächst die Verbesserung des globalen Sichtbarkeitstests und die Berechnung der Extremal-Elemente betrachtet. Anschließend werden die Optimierungen für die Konstruktion der einzelnen MFL-Typen diskutiert. Zuletzt wird der Aufwand für die Konstruktion eines kompletten Visibility Skeletons einer 3D-Szene untersucht.

3.5.8.1 Globaler Sichtbarkeitstest und Berechnung der Extremal-Elemente

Für die Optimierung des globalen Sichtbarkeitstests und der Berechnung der Extremal-Elemente wurden in Kap. 3.5.7.1 Strahlanfragen an 3D-Szenen verwendet. Dabei wurde unterschieden zwischen

- Strahlanfrage mit Objektliste (kurz: **OL**)
- Strahlanfrage mit BB-Hierarchie (kurz: **BBH**)
- Strahlanfrage mit Priority Queues (kurz: **PQ**)

Um die Auswirkungen der Verwendung der BB-Hierarchie und der Priority Queue gegenüber der einfachen Objektliste zu untersuchen, wurden jeweils 10000 zufällige Strahlen in 3D-Szenen mit 100 – 10000 Zufalls-Objekten geschossen.

Abb. 3.145 zeigt die durchschnittliche Zahl der jeweils insgesamt von einem Strahl geschnittenen 3D-Objekte. Wie zu erwarten, nimmt mit zunehmender Zahl von 3D-Objekten auch die Zahl der von einem zufälligen Strahl geschnittenen 3D-Objekte zu.

Abb. 3.146 zeigt die Zahl der durchschnittlichen Zahl von Strahl-B-Box-Schnitten zur Auswertung der Strahlanfragen. Interessant ist hier zunächst der Verlauf der Kurve **OL** für die Strahlanfragen an Objektlisten: mit zunehmender Zahl von 3D-Objekten in der Szene steigt auch die Wahrscheinlichkeit, dass eines dieser 3D-Objekte den Strahl zwischen den Aufpunkten schneidet, d.h. verdeckt. In diesem Fall wird die Strahlanfrage abgebrochen. Daher steigt der Aufwand für die Strahlanfrage an eine 3D-Szene nicht linear mit der Zahl der 3D-Objekte in der Szene.

Weiterhin ist in Abb. 3.146 deutlich die große Ersparnis von Strahl-Box-Schnitten durch Verwendung der BB-Hierarchie und der Priority Queues zu erkennen.

In Abb. 3.147 sind die Kurven **BBH** und **PQ** nochmals allein dargestellt. Man sieht, dass die Verwendung von Priority Queues die Zahl der Strahl-B-Box-Schnitte nochmals deutlich reduziert.

Insbesondere zeigt die Kurve **PQ** auch, dass verdeckte Strahlen sehr schnell erkannt werden und somit ***der Aufwand für einzelne Strahlanfragen bei steigender Zahl von 3D-Objekten abnimmt***!

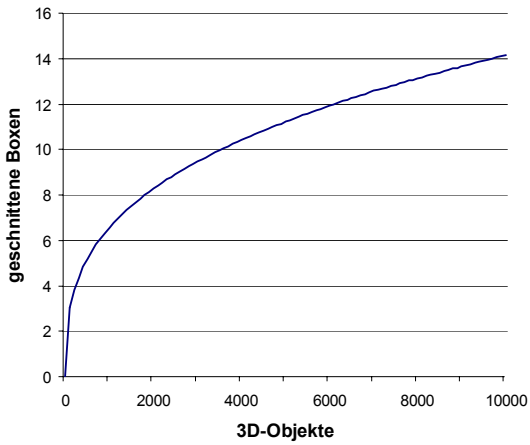


Abb. 3.145
Durchschnittliche Anzahl der von einem Strahl geschnittenen 3D-Objekte

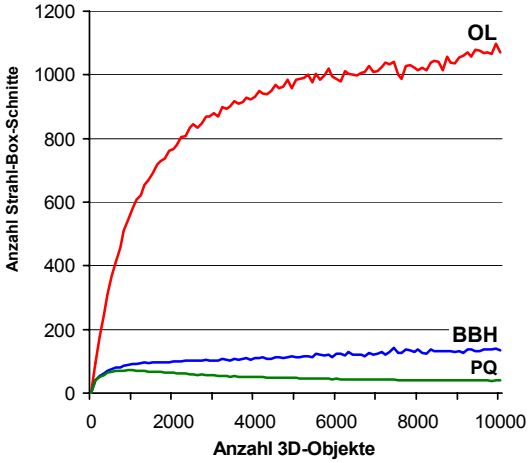


Abb. 3.146
Durchschnittliche Anzahl der Strahl-Box-Schnitte für Objektliste (**OL**), BB-Hierarchie (**BBH**) und Priority Queues (**PQ**)

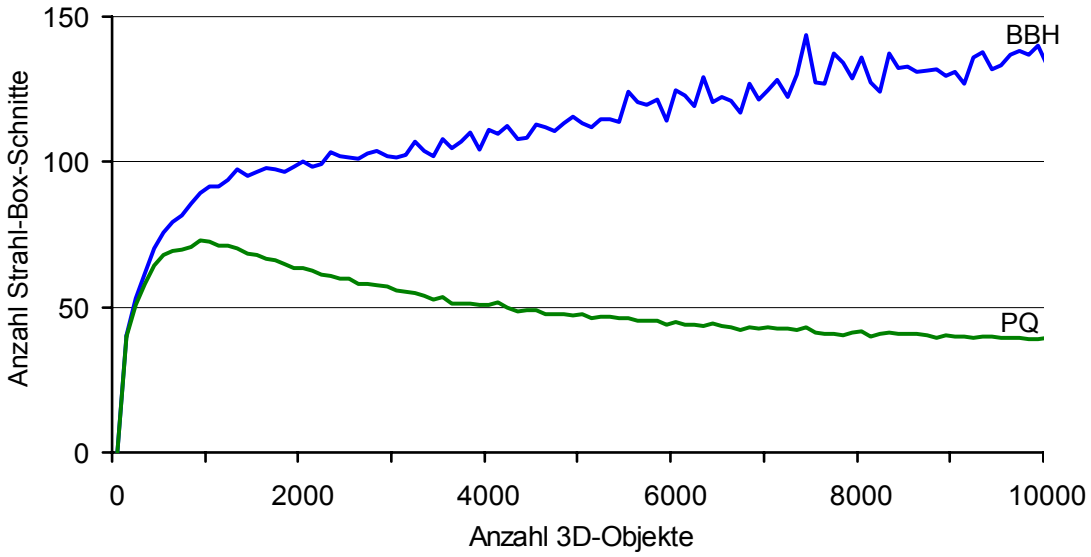


Abb. 3.147
Durchschnittliche Anzahl der Strahl-Box-Schnitte für BB-Hierarchie (**BBH**) und Priority Queues (**PQ**)

3.5.8.2 Konstruktion der maximal freien Liniensegmente

In diesem Teilkapitel wird der Aufwand zur Konstruktion der maximal freien Liniensegmente in jeweils einem Generator-Shaft untersucht, die Konstruktion aller MFLs in einer kompletten 3D-Szene wird im nächsten Teilkapitel betrachtet.

Für einen einzelnen Generator-Shaft wurden in Kap. 3.5.7.2 drei Ansätze zur Konstruktion der MFLs vorgestellt: Verwendung von Objektliste, BB-Hierarchie und BB-Hierarchie mit Aufspaltung in Kind-Shafts. Für die folgenden Experimente wurden in 10 3D-Szenen mit jeweils 10 – 1000 3D-Objekten jeweils 100 zufällige **EV**-Shafts gebildet und alle darin enthaltenen **VEE-**/**EVE-**MFLs konstruiert.

Abb. 3.148 zeigt die durchschnittliche Zahl der Kandidaten für **VEE-**/**EVE-**MFLs und die durchschnittliche Zahl der davon gültigen, d.h. global sichtbaren, MFLs. Man erkennt, dass die Zahl der Kandidaten im Wesentlichen linear mit der Zahl der 3D-Objekte steigt, während die demgegenüber recht kleine Zahl der gültigen MFLs mit zunehmender Zahl von 3D-Objekten kaum noch steigt (vgl. auch Abb. 3.149).

Dieses Verhalten tauchte schon bei der Betrachtung des globalen Sichtbarkeitstests im letzen Teilkapitel auf: Mit zunehmender Zahl von 3D-Objekten steigt auch die Zahl verdeckter Strahlen, d.h. die Zahl global sichtbarer MFLs sinkt.

Abb. 3.150 zeigt die durchschnittliche Zahl der notwendigen Klassifikationen anhand der **EV**-Shafts für die Konstruktion der **VEE**-/**EVE**-MFLs unter Verwendung von

- Objektliste (Kurve **OL**)
- BB-Hierarchie (Kurve **BBH**)
- BB-Hierarchie mit Aufspaltung in Kind-Shafts (**AKS**)

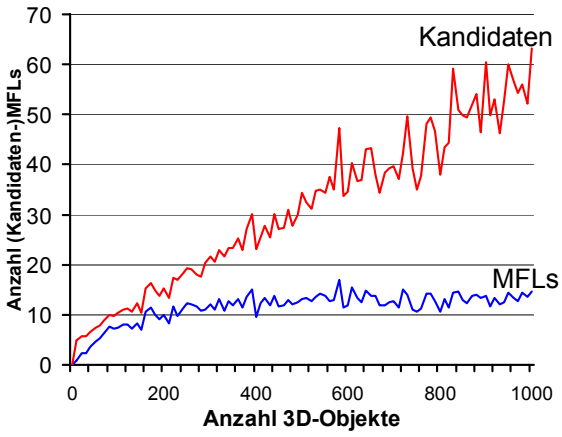


Abb. 3.148
Anzahl (Kandidaten-)MFLs

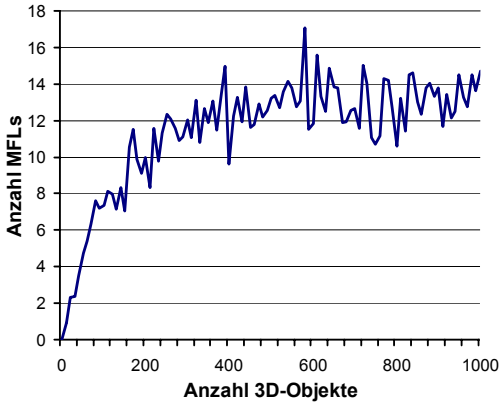


Abb. 3.149
Anzahl MFLs

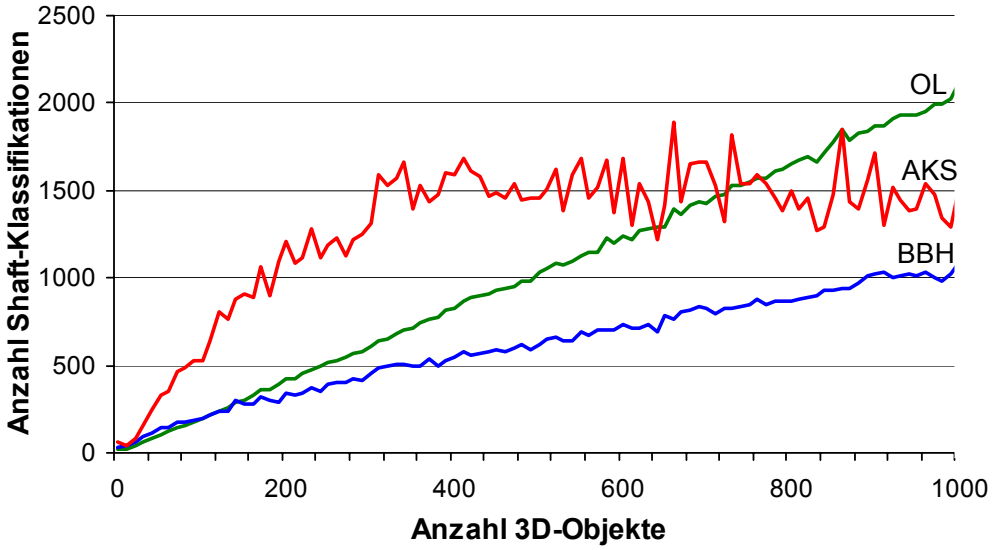


Abb. 3.150

Anzahl **EV**-Shaft-Klassifikationen zur Auswahl der Kandidaten für **VEE**-/**EVE**-MFLs

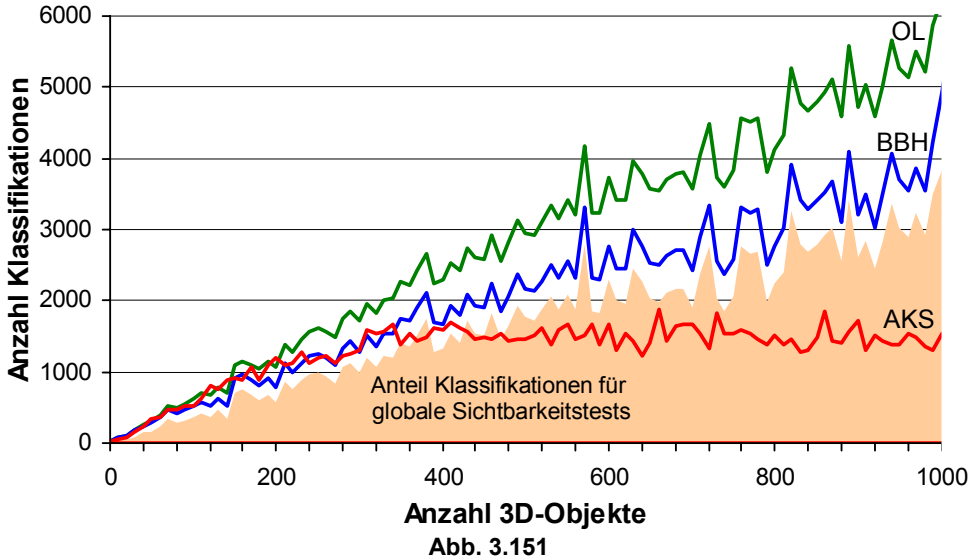


Abb. 3.151

Gesamtzahl der Klassifikationen für Konstruktion der **VEE**-/**EVE**-MFLs

In Abb. 3.150 ist zu erkennen, dass die Kurve **AKS** deutlich über der Kurve **BBH** verläuft. Damit scheint die in Kap. 3.5.7.3 vorgeschlagenen Aufteilung in Kind-Shafts zunächst nicht sinnvoll. Betrachtet man dagegen den Gesamtaufwand für die Konstruktion der MFLs, d.h. den Aufwand für die Shaft-Klassifikation und den Aufwand für die globalen Sichtbarkeitstests, so ergibt sich ein anderes Bild:

In Abb. 3.151 ist die Gesamtzahl aller Klassifikationen und der Anteil der globalen Sichtbarkeitstests daran dargestellt. Durch die Aufteilung in Kind-Shafts sind keine weiteren globalen Sichtbarkeitstests nötig, daher liegt die Kurve **AKS** deutlich unter den Kurven **OL** und **BBH**.

Weiterhin erkennt man, dass die Kurve **AKS** mit zunehmender Zahl von 3D-Objekten flacher wird, da die Kind-Shafts durch die Klassifikationen immer weiter beschränkt werden und somit immer weniger 3D-Objekte als „im“ Shaft klassifiziert werden.

Die Aufspaltung in Kind-Shafts konnte in Kap. 3.5.7.2 nur für die Vertex- und Facetten-MFLs durchgeführt werden, bei den Kanten-MFLs war nur eine Beschränkung der **EE**(**E**)-Shafts möglich. Diese wird im Folgenden genauer betrachtet. Dazu werden zunächst drei 3D-Szenen ***S1*** – ***S3*** mit jeweils 2000 – 10000 ungefähr gleich großen 3D-Objekten untersucht, wobei in jeder 3D-Szene die Seitenlänge der 3D-Objekte jeweils halbiert wurde. Anschaulich bedeutet dies, dass die 3D-Szene ***S1*** sehr „dicht“ ist, d.h. ein hohes Maß gegenseitiger Verdeckung herrscht, während die 3D-Szene ***S3*** „licht“ ist, d.h. die gegenseitige Sichtbarkeit der 3D-Objekte nur wenig beeinflusst wird.

Zunächst zeigt Abb. 3.152 die durchschnittliche Anzahl von 3D-Objekten in den **EE**-Shafts der 3D-Szenen ***S1*** - ***S3***, wobei nur **EE**-Shafts mit mindestens zwei 3D-Objekten gezählt wurden, da nur diese **E4**-MFLs enthalten können.. Diese Anzahl steigt mit $\mathbf{O(\sqrt[3]{n})}$, was wie folgt erklärt werden kann (vgl. auch [20]): Abb. 3.153 zeigt eine 2D-Szene mit $\mathbf{n = 64}$ 2D-Objekten. Für jedes Paar von 2D-Objekten können im Average Case $\mathbf{O(\sqrt{n})}$ 2D-Objekte die Sichtbarkeit dieses Paars beeinflussen. Übertragen auf den 3D bedeutet dies, dass $\mathbf{O(\sqrt[3]{n})}$ 3D-Objekte in einem Shaft liegen können.

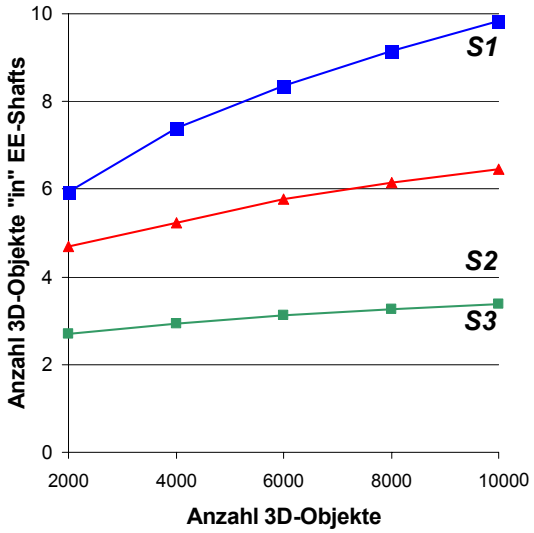


Abb. 3.152
Durchschnittliche Anzahl von 3D-Objekten „in“ den **EE**-Shafts der 3D-Szenen ***S1-S3***

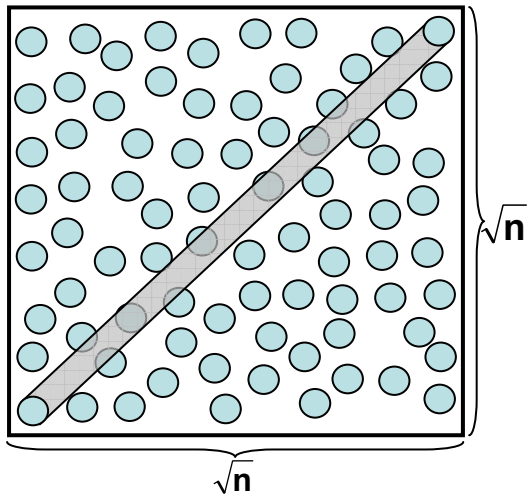


Abb. 3.153
2D-Szene mit 64 2D-Objekten und „maximalem“ Shaft

Die 3D-Objekte „in“ den **EE**-Shafts können diese beschränken, wenn Sie mindestens zwei Shaft-Linien des **EE**-Shafts schneiden (vgl. Kap. 3.5.7.5). Schneidet ein 3D-Objekt alle vier Shaft-Linien, so ist der **EE**-Shaft vollständig verdeckt und muss nicht weiter betrachtet werden. Abb. 3.154 – Abb. 3.156 zeigt für die 3D-Szenen ***S1-S3*** den durchschnittlichen Anteil der 3D-Objekte „in“ den **EE**-Shafts, die 0-4 Shaft-Linien schneiden bzw. durch Beschränkung außerhalb des **EE**-Shafts liegen.

Es ergibt sich, dass nahezu unabhängig von der „Dichte“ der 3D-Szene bereits vor dem ersten Beschränkungsschritt jeweils ca. 8% der 3D-Objekte vollständig verdeckend sind und mehr als 50% der 3D-Objekte den **EE**-Shaft beschränken können, da sie zwei oder drei Shaft-Linien schneiden. In den folgenden Beschränkungsschritten steigt sowohl der Anteil der vollständig verdeckenden 3D-Objekte als auch der Anteil der 3D-Objekte, die außerhalb des Shafts liegen.

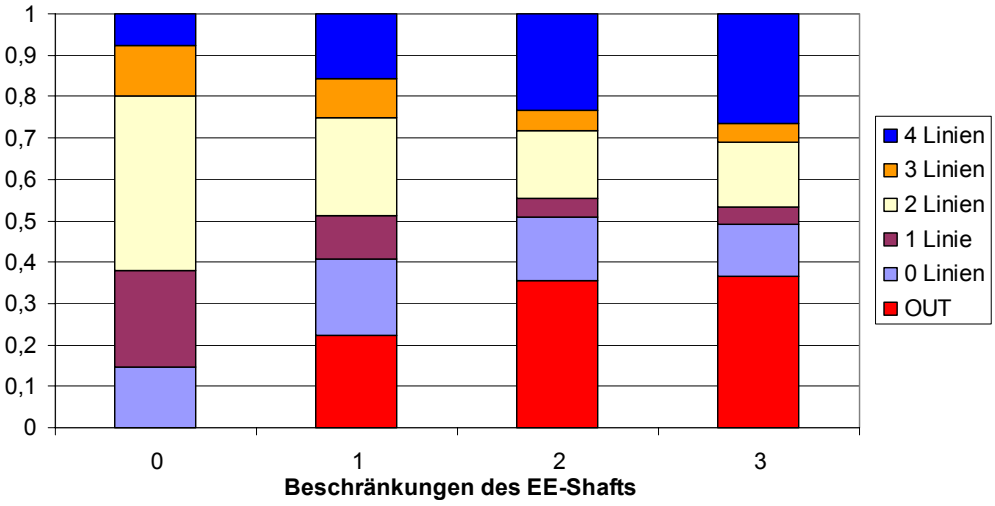


Abb. 3.154
Durchschnittliche Anteile von 3D-Objekten in 3D-Szene ***S1***, die 0-4 Shaft-Linien schneiden

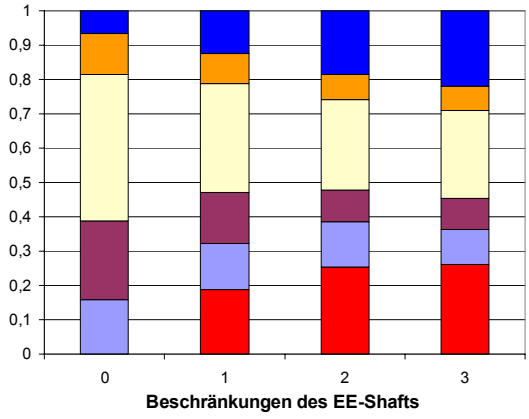


Abb. 3.155
Durchschnittliche Anteile von 3D-Objekten in 3D-Szene ***S2***, die 0-4 Shaft-Linien schneiden

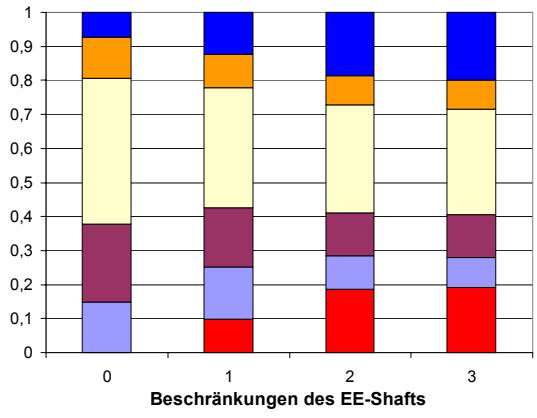


Abb. 3.156
Durchschnittliche Anteile von 3D-Objekten in 3D-Szene ***S3***, die 0-4 Shaft-Linien schneiden

An dieser Stelle soll kurz die Systematik hinter diesem Ansatz diskutiert werden, d.h. eine Begründung bzw. Veranschaulichung für die durchschnittliche Zahl geschnittener Shaft-Linien gegeben werden. Dazu wird ein stark vereinfachtes Modell im 2D betrachtet: Die 3D-Objekte der 3D-Szene werden als gleich große Kreise **K** mit Radius **R** und die beiden Kanten, die den **EE**-Shaft aufspannen, als gleich große Kreise $\mathbf{k_1}$ und $\mathbf{k_2}$ mit Radius $\mathbf{r = sR}$ ($\mathbf{s > 0}$) und Abstand **d** modelliert (vgl. Abb. 3.157).

Zunächst zeigt Abb. 3.157 den Shaft **S** zwischen den Kanten-Kreisen sowie zwei weitere Shafts $\mathbf{S_{PAR}}$ und $\mathbf{S_{OCC}}$ (hier für den Fall $\mathbf{s = {}^2/_3}$). Liegt der Mittelpunkt eines Objekt-Kreises **K** im Shaft $\mathbf{S_{PAR}}$, so schneidet **K** den Shaft **S**, liegt der Mittelpunkt im Shaft $\mathbf{S_{OCC}}$, so verdeckt **K** den Shaft **S**.

Dieses Modell kann direkt auf den 3D übertragen werden: Die Kreise **K**, $\mathbf{k_1}$ und $\mathbf{k_2}$ werden zu Kugeln und die Shafts zu Zylindern. Für die Radien der Zylinder $\mathbf{Z_{PAR}}$ und $\mathbf{Z_{OCC}}$ gilt (vgl. Abb. 3.157):

$$\mathbf{r_{PAR} = R + r = R(1 + s)} \text{ und } \mathbf{r_{OCC} = R - r = R(1 - s)} \text{ für } \mathbf{0<s\leq 1}.$$

Man beachte, dass $\mathbf{S_{OCC}}$ für $\mathbf{s > 1}$ nicht definiert ist, im Folgenden wird zunächst der Fall $\mathbf{s \leq 1}$ betrachtet. Die Wahrscheinlichkeit, dass ein 3D-Objekt „im“ Shaft **S** diesen auch vollständig verdeckt, entspricht dem Verhältnis der Volumen der beiden Zylinder:

$$\mathbf{P_{OCC}(s) = \frac{Z_{OCC}}{Z_{PAR}} = \frac{\pi dr_{OCC}^2}{\pi dr_{PAR}^2} = \frac{(1-s)^2}{(1+s)^2} = \left(\frac{1-s}{1+s}\right)^2}$$

In Abb. 3.158 ist der Verlauf von $\mathbf{P_{OCC}(s)}$ für $\mathbf{0 \leq s \leq 1}$ dargestellt. Man erkennt, dass nur 3D-Objekte, die „groß“ gegenüber dem Shaft sind, diesen verdecken können.

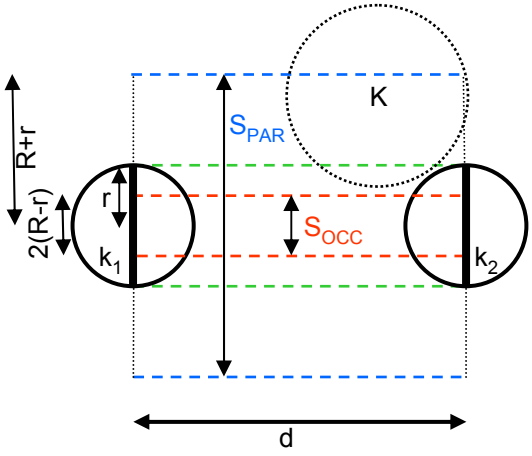


Abb. 3.157
Shafts zwischen Kanten-Kreisen

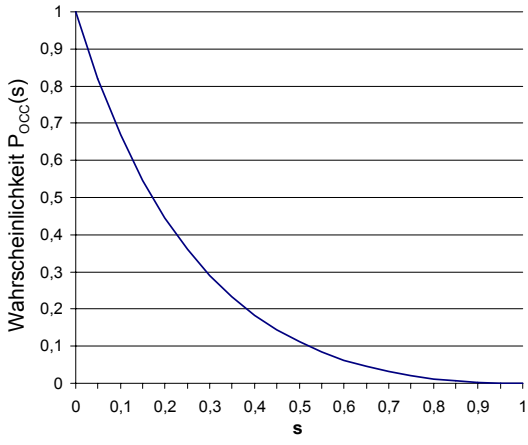


Abb. 3.158
Wahrscheinlichkeiten für vollständige Verdeckung

Als nächstes soll nun die Wahrscheinlichkeit betrachtet werden, dass ein 3D-Objekt im **PAR**-Shaft mindestens zwei Shaft-Linien schneidet, d.h. den Shaft beschränken kann. In Abb. 3.159 ist für die beiden Kanten-Kreise $\mathbf{k_1}$ und $\mathbf{k_2}$ ein Vertex-Kanten-Shaft $\mathbf{S_{VK}}$ eingezeichnet. Zusätzlich ist noch der Shaft $\mathbf{S_{VK\text{-}OCC}}$ angegeben (analog zu oben für $\mathbf{s = {}^2/_3}$), d.h. der Shaft, für den 3D-Objekte den Shaft $\mathbf{S_{VK}}$ vollständig verdecken können. Abb. 3.160 zeigt die Gesamtheit aller Vertex-Kanten-Shafts zwischen den Kanten-Kreisen $\mathbf{k_1}$ und $\mathbf{k_2}$ sowie fett eingezeichnet ihren gemeinsamen Umriss. Alle Objekt-Kreise, deren Mittelpunkt im Umriss liegt, schneiden mindestens zwei Shaft-Linien.

Dieses 2D-Modell kann nicht direkt auf den 3D übertragen werden, als grobe Näherung soll es jedoch ausreichen: Analog zu oben entsprechen die beiden 2D-Trapeze $\mathbf{T_1}$ und $\mathbf{T_2}$ in Abb. 3.160 zwei Kegelstümpfen $\mathbf{KS_1}$ und $\mathbf{KS_2}$ mit Radien $\mathbf{r_1 = R + r = R(1 + s)}$, $\mathbf{r_2 = R}$ und Höhe $\mathbf{½d}$. Die Wahrscheinlichkeit, dass ein 3D-Objekt „im“ Shaft **S** mindestens zwei Shaft-Linien schneidet, entspricht dem Verhältnis der Volumen der Kegelstümpfe $\mathbf{KS_1}$ und $\mathbf{KS_2}$ zum Volumen des **PAR**-Zylinders:

$$\mathbf{P_{VK-OCC}(s)} = \frac{\mathbf{KS_1 + KS_2}}{\mathbf{Z_{PAR}}} = \frac{\frac{2}{3}\cdot\pi\cdot\frac{d}{2}\left(r_1^2 + r_1 r_2 + r_2^2\right)}{\pi d R^2 (1+s)^2} = \frac{\frac{s^2}{3} + s + 1}{(1+s)^2}$$

Wie weiter oben bereits ausgeführt, gelten diese Überlegungen zunächst nur für $\mathbf{s \le 1}$. Für $\mathbf{1 < s < 2}$ und $\mathbf{2 < s}$ sind jeweils eigene Behandlungen erforderlich, die hier jedoch nicht vollständig aufgeführt werden sollen. Zur Veranschaulichung sind in Abb. 3.161 die Shafts $\mathbf{S_{VK}}$ und $\mathbf{S_{VK\text{-}OCC}}$ für $\mathbf{s = 2}$ dargestellt. Abb. 3.162 zeigt den Verlauf der Wahrscheinlichkeiten $\mathbf{P_{VK\text{-}OCC}(s)}$ für $\mathbf{0 \le s < 3}$.

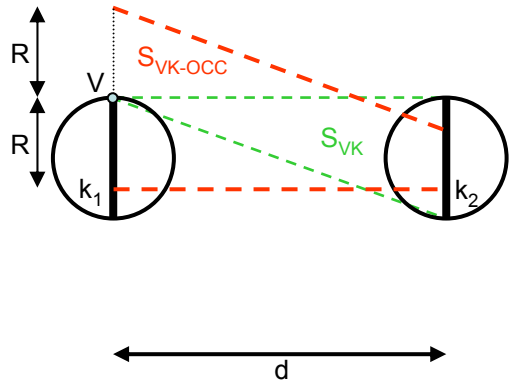


Abb. 3.159
Einzelner Vertex-Kante-Shaft $\mathbf{S_{VK}}$ und $\mathbf{S_{VK\text{-}OCC}}$

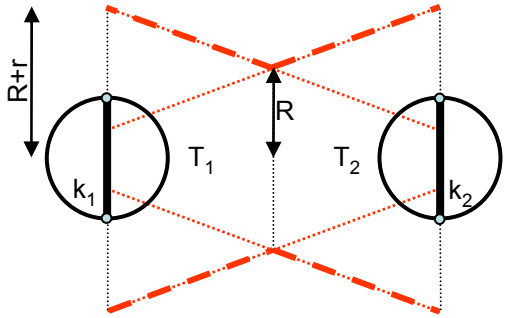


Abb. 3.160
Gesamtheit aller Vertex-Kanten-Shafts

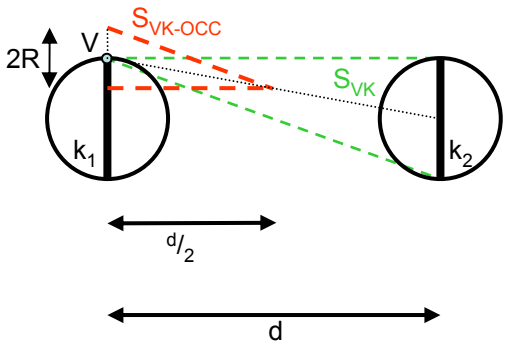


Abb. 3.161
Vertex-Kanten-Shafts S_{VK} und $S_{VK\text{-}OCC}$ für **s = 2**

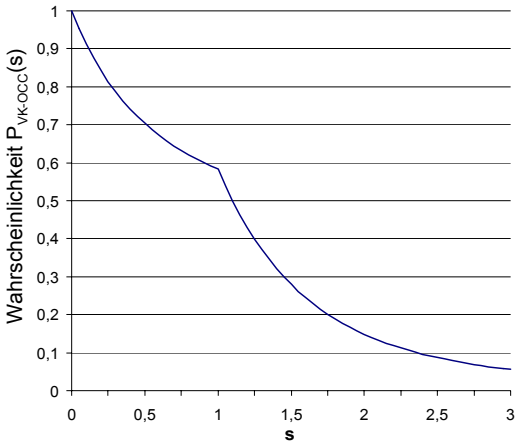


Abb. 3.162
Wahrscheinlichkeiten für vollständige Verdeckung von mindestens zwei Shaft-Linien

Wie man Abb. 3.162 entnehmen kann, beträgt die Wahrscheinlichkeit für mindestens zwei verdeckte Shaft-Linien für **s < 1** deutlich über 60%, selbst für **s = 2** ergibt sich noch ein Wert von ungefähr 15%. Dies entspricht den in Abb. 3.154 – Abb. 3.156 gemessenen Werten für die 3D-Szenen ***S1–S3*** mit ungefähr gleich großen 3D-Objekten: jeweils ca. 60-70% der 3D-Objekte verdecken mindestens zwei Shaft-Linien.

Damit bleibt zuletzt noch die Frage offen, wie der Faktor **s** interpretiert werden kann: Zunächst beschreibt **s** natürlich einfach die Größenverhältnisse zwischen den 3D-Objekten, die den Shaft aufspannen und den enthaltenen 3D-Objekten.

Weiterhin kann **s** jedoch auch zusätzlich wie folgt interpretiert werden:

- *Verhältnis zwischen Kantenlänge und maximalen Objekt-Querschnitt*: Für einen Würfel würde sich so z.B ein maximales Verhältnis $\mathbf{s = 1/\sqrt{3} \approx 0{,}577}$ ergeben.
- *„Verdrehung" der Kanten zueinander*: In den obigen Abbildungen waren die beiden Kanten stets parallel. Sind sie dagegen „verdreht", so entspricht der Radius der Kanten-Kreise nicht mehr der halben Kantenlänge, sondern der Projektion der Kante auf eine zur Verbindunglinie der Kantenmittelpunkte senkrechten Ebene. Für zwei Kanten, die jeweils um 45° gegeneinander verdreht sind, ergibt sich so z.B. $\mathbf{s = 1/\sqrt{2} \approx 0{,}707}$.

Mit diesen Überlegungen gilt also, dass der Faktor **s** auch bei gleich großen 3D-Objekten oder sogar bei 3D-Objekten unterschiedlicher Größe kleiner als **1** sein kann

und somit mit den Werten aus Abb. 3.158 und Abb. 3.162 die **EE**-Shafts mit hoher Wahrscheinlichkeit von den enthaltenen 3D-Objekten beschränkt werden können. Damit kann man also zunächst festhalten, dass die Beschränkung der **EE**-Shafts nach dem in Kap. 3.5.7.5 vorgeschlagenen Verfahren ein sinnvoller Ansatz ist, der sowohl die Anzahl der insgesamt gültigen, d.h. unverdeckten, **EE**-Shafts als auch die Anzahl der 3D-Objekte in den verbleibenden gültigen **EE**-Shafts deutlich senkt.

Diese Aussage ist alleine jedoch nicht ausreichend für eine echte Verbesserung: eine Halbierung der Zahl der gültigen **EE**-Shafts und der darin enthaltenen 3D-Objekte ändert zunächst nichts am Aufwand des Verfahrens. Daher wird im nächsten Schritt gezeigt, dass durch die Beschränkung der **EE**-Shafts praktisch alle **EE**-Shafts mit mehr als vier 3D-Objekten wegfallen. Abb. 3.165 – Abb. 3.170 zeigen jeweils auf der linken Seite die Verteilungen der 3D-Objekte pro **EE**-Shaft vor der Beschränkung und auf der rechten Seite nach der Beschränkung. Es ist deutlich zu erkennen, dass in allen drei 3D-Szenen ***S1-S3*** nach der Beschränkung im Wesentlichen nur noch **EE**-Shafts mit maximal 4-5 3D-Objekten übrig bleiben.

Damit bleibt noch die durchschnittliche Anzahl von 3D-Objekten in den **EE**-Shafts vor und nach der Beschränkung zu betrachten. Abb. 3.163 zeigt dies im Vergleich zu den ursprünglichen Anzahlen vor der Beschränkung und Abb. 3.164 nur die Anzahlen nach der Beschränkung. Die Anzahl der 3D-Objekte in den **EE**-Shafts ist nach der Beschränkung deutlich kleiner als vorher und ändert sich kaum noch in Abhängigkeit von der Gesamtzahl der 3D-Objekte.

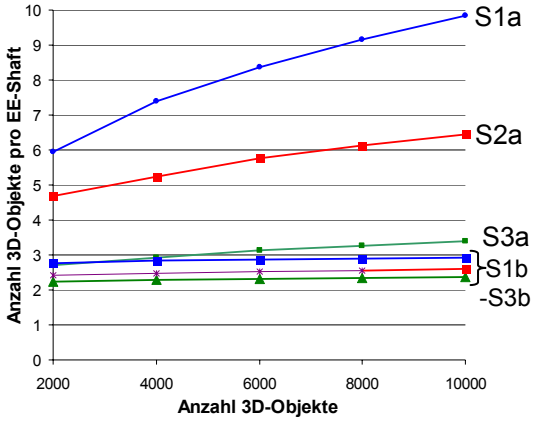


Abb. 3.163
Anzahl 3D-Objekte in **EE**-Shafts (a) vor und (b) nach Beschränkung

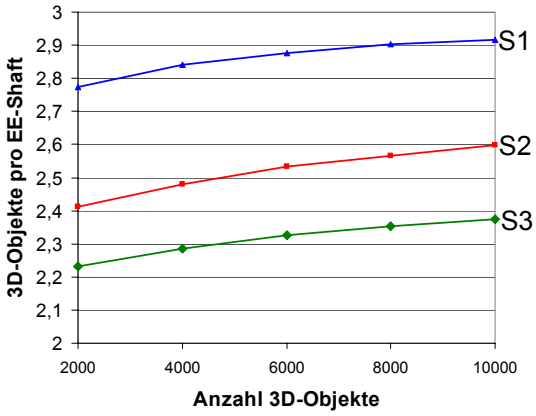


Abb. 3.164
Anzahl 3D-Objekte in **EE**-Shafts nach Beschränkung

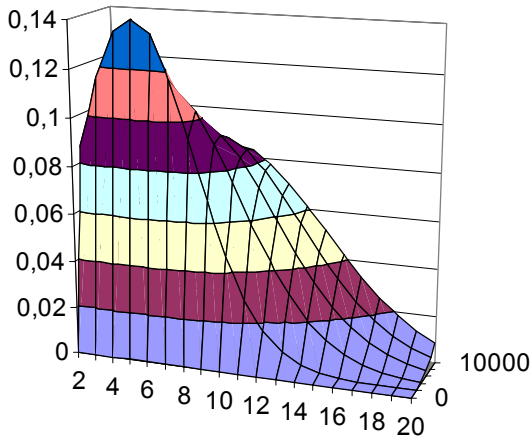


Abb. 3.165
3D-Objekte pro Shaft vor Beschränkung (***S1***)

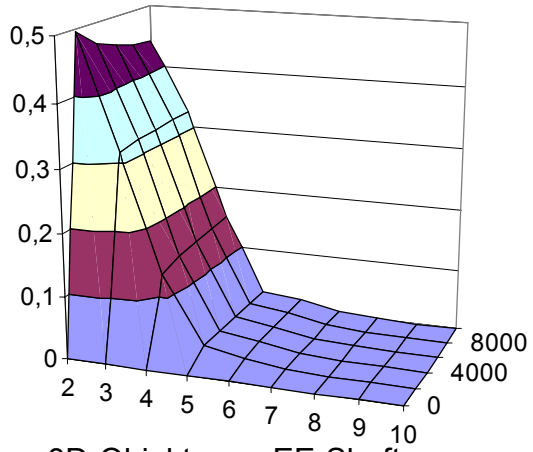


Abb. 3.166
3D-Objekte pro Shaft nach Beschränkung (***S1***)

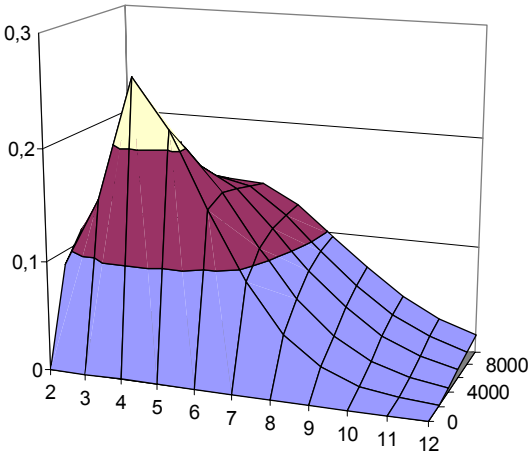


Abb. 3.167
3D-Objekte pro Shaft vor Beschränkung (***S2***)

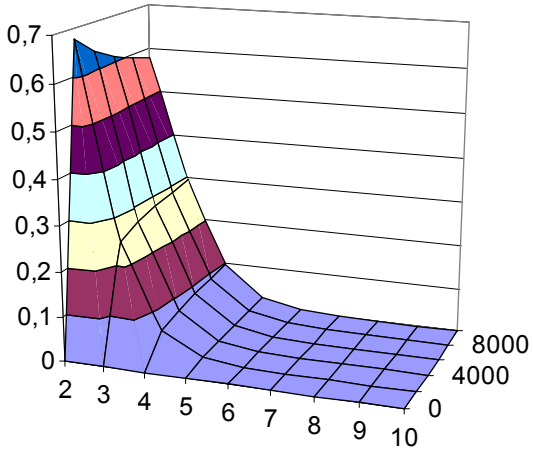


Abb. 3.168
3D-Objekte pro Shaft nach Beschränkung (***S2***)

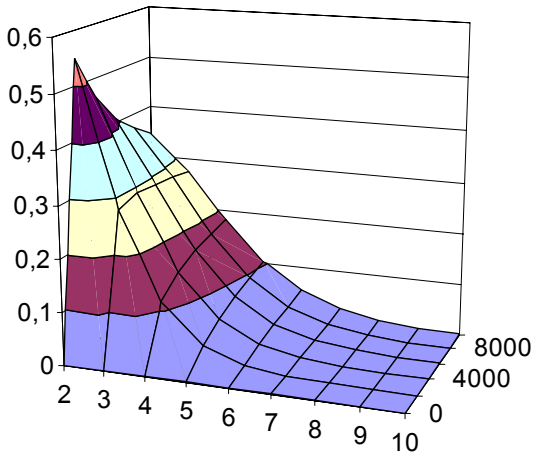


Abb. 3.169
3D-Objekte pro Shaft vor Beschränkung (***S3***)

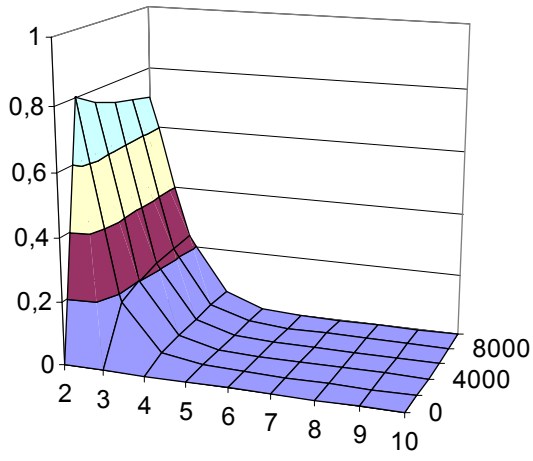


Abb. 3.170
3D-Objekte pro Shaft nach Beschränkung (***S3***)

Noch deutlicher wird die Auswirkung der Beschränkungen der **EE**-Shafts, wenn man nicht nur die Anzahl der 3D-Objekte in den gültigen **EE**-Shafts, sondern die Anzahl der zu überprüfenden Kanten-Paare in Bezug auf alle **EE**-Shafts (d.h. auch verdeckte **EE**-Shafts oder solche mit weniger als zwei enthaltenen Kanten) betrachtet. Abb. 3.171 zeigt für die 3D-Szene ***S1*** die Anzahlen der Kanten-Paare in den **EE**-Shafts (a) vor und (b) nach der Beschränkung und Abb. 3.172 nur die Anzahlen nach der Beschränkung. Während ohne die Beschränkung der **EE**-Shafts bei 3D-Szene ***S1*** zwischen 70 und 240 Kanten-Paare betrachtet werden müssen, sinkt die Zahl der Kanten-Paare pro **EE**-Shaft nach der Beschränkung von etwas mehr als 4 auf weniger als 3 Paare! Zum Vergleich zeigt Abb. 3.173 noch den Anteil der unverdeckten **EE**-Shafts an allen **EE**-Shafts.

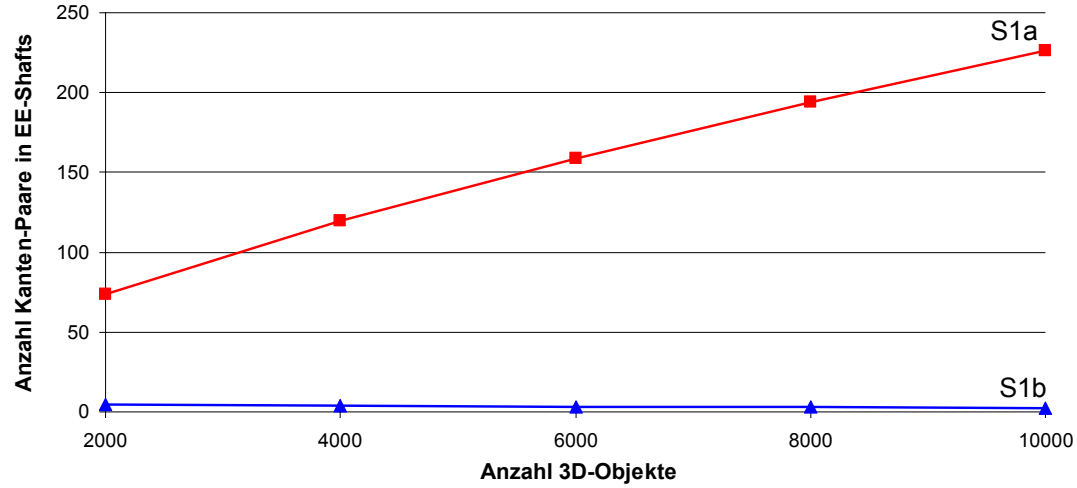


Abb. 3.171
Anzahl von Kanten-Paaren in **EE**-Shafts (a) vor und (b) nach Beschränkung für 3D-Szene ***S1***

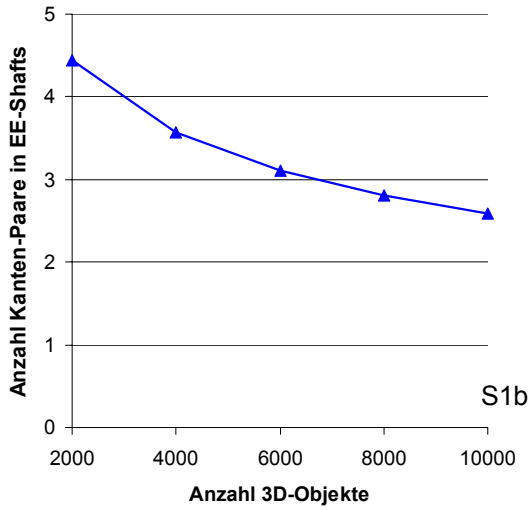


Abb. 3.172
Anzahl von Kanten-Paare in **EE**-Shafts nach Beschränkung für 3D-Szene ***S1***

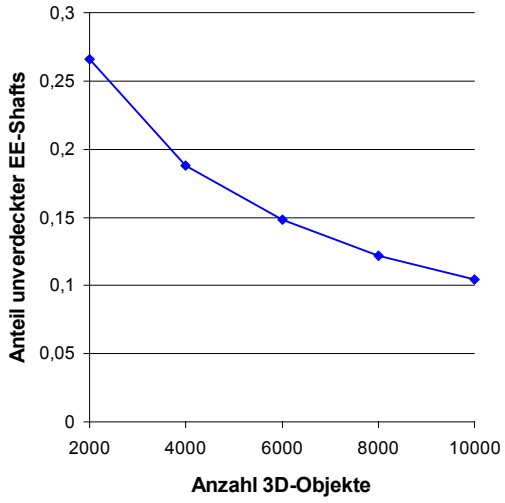


Abb. 3.173
Anteil unverdeckter **EE**-Shafts an allen **EE**-Shafts für 3D-Szene ***S1***

Die obigen Messungen beziehen sich auf **EE**-Shafts zwischen 3D-Objekten ungefähr gleicher Größe. Unterscheiden sich die Größen(ordnungen) der 3D-Objekte dagegen stark, so führt die Beschränkung der **EE**-Shafts zu keiner echten Verbesserung, da kaum 3D-Objekte zwei oder mehr Shaft-Linien des **EE**-Shafts schneiden (vgl. auch Abb. 3.162).

Für die Behandlung dieses Sonderfalls wurde in Kap. 3.5.7.5 ein Plane-Sweep-Verfahren vorgeschlagen, dass die Betrachtung aller $\mathbf{O(m^2)}$ Paare von Kanten in **EE**-Shafts mit **m** enthaltenen Kanten vermeidet. Um diese Verbesserung zu beurteilen, wird im folgenden eine 3D-Szene mit 50-500 3D-Objekten betrachtet, wobei nur die „größten“ **EE**-Shafts der 3D-Szene ausgewertet wurden, d.h. die **EE**-Shafts zwischen Kanten des die 3D-Szene umgebenden Raums.

In Abb. 3.174 ist dargestellt, wie viele Paare von Kanten durchschnittlich in diesen **EE**-Shafts liegen und wie viele dieser Paare nach dem Plane Sweep als Kandidaten für **E4**-MFLs ausgewählt wurden. Abb. 3.175 zeigt nur die Anzahl der Paare nach dem Plane Sweep.

Währen die Gesamtzahl der Paare mit $\mathbf{O(n^2)}$ steigt, steigt die Zahl der Paare nach dem Plane Sweep nur noch mit ungefähr $\mathbf{O(n^{1,33})}$, was wiederum die die Abschätzung aus [20] bestätigt: Die Zahl der Kanten „in“ den **EE**-Shafts steigt linear mit der Zahl der 3D-Objekte und für jede Kante „im“ **EE**-Shaft können $\mathbf{O(n^{1/3})}$ Kanten mit dieser Kante **E4**-MFLs definieren.

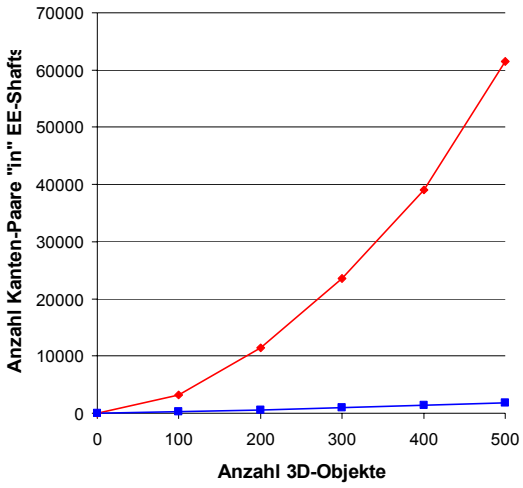


Abb. 3.174
Anzahl Kanten-Paare „in“ **EE**-Shafts vor und nach Plane Sweep

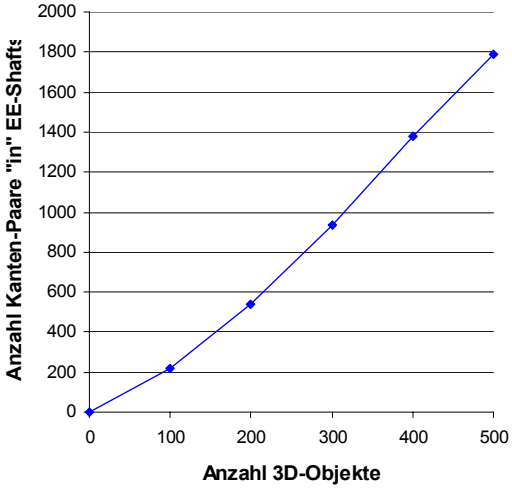


Abb. 3.175
Anzahl Kanten-Paare „in“ **EE**-Shafts nach Plane-Sweep

3.5.8.3 Konstruktion des Visibility Skeletons einer 3D-Szene

Im letzten Teilkapitel wurde der Aufwand zur Konstruktion aller MFLs in einzelnen Generator-Shafts untersucht. Nun wird der Aufwand zur Konstruktion aller MFLs in einer kompletten 3D-Szene, d.h. in $\mathbf{O(n^2)}$ Generator-Shafts, betrachtet. Dazu werden im Folgenden 10 3D-Szenen mit jeweils 25 – 500 zufälligen 3D-Objekten verwendet.

Zunächst zeigt Abb. 3.176 die durchschnittliche Gesamtzahl der Vertex-, Facetten- und Kanten-MFLs in den 10 3D-Szenen. Offensichtlich steigt die Zahl **m** der MFLs ungefähr quadratisch mit der Zahl **n** der 3D-Objekte. Um dies zu verdeutlichen, zeigt Abb. 3.177 den annähernd linearen Verlauf von $\sqrt{\mathbf{m}}$.

Der Anteil der Vertex-MFLs **VV**, **VEE** und **EVE** an allen MFLs ist ca. 70%, die verbleibenden 30% teilen sich ungefähr gleich auf die Facetten- und Kanten-MFLs auf.

In Abb. 3.178 sind die durchschnittlichen Zeiten zur Konstruktion des Visibility Skeletons der zehn 3D-Szenen bei Verwendung von Objektliste (**OL**), BB-Hierarchie (**BBH**) und BB-Hierarchie mit Aufspaltung in Kind-Shafts (**AKS**) dargestellt.

Kurve **OL** verläuft ungefähr mit $\mathbf{O(n^3)}$, da für jeden der $\mathbf{O(n^2)}$ Generator-Shafts **O(n)** 3D-Objekte klassifiziert werden müssen. In Abb. 3.179 sind die Kurven **BBH** und **AKS** nochmals alleine dargestellt. Kurve **BBH** steigt ungefähr mit $\mathbf{O(n^{2,8})}$, während Kurve **AKS** nur mit ungefähr $\mathbf{O(n^{2,4})}$ steigt.

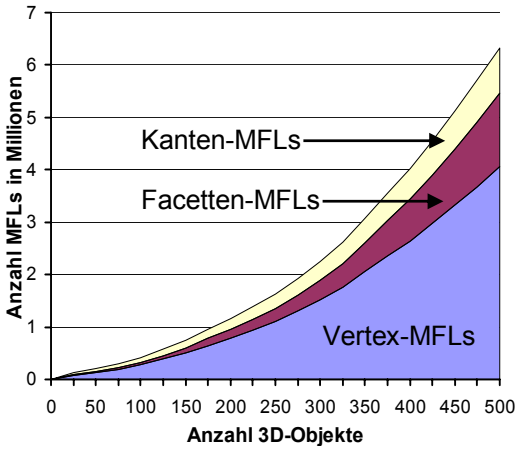


Abb. 3.176
Durchschnittliche Zahl von MFLs

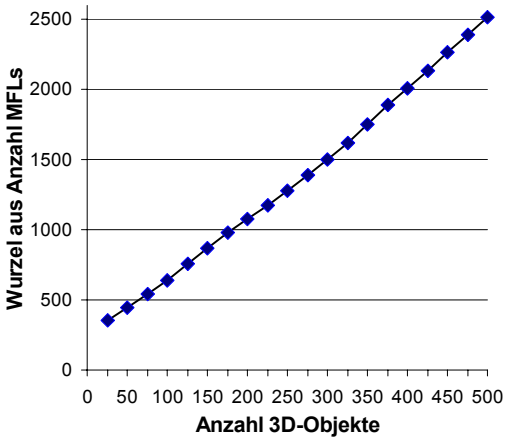


Abb. 3.177
Durchschnittliches Verhältnis $\mathbf{m/n^2}$
(**m** = Anzahl MFLs, **n** = Anzahl 3D-Objekte)

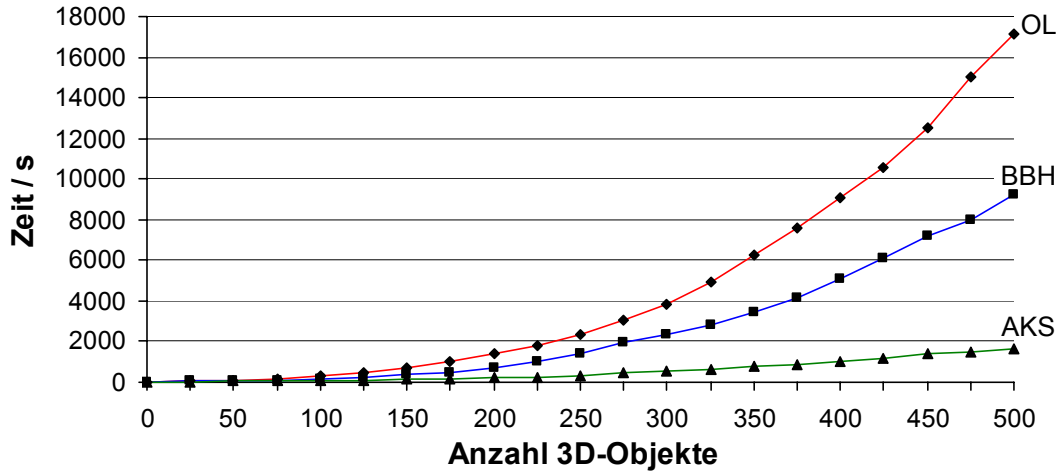


Abb. 3.178
Durchschnittliche Zeiten zur Konstruktion des Visibility Skeletons bei Verwendung von Objektliste, BB-Hierarchie und BB-Hierarchie mit Aufspaltung in Kind-Shafts

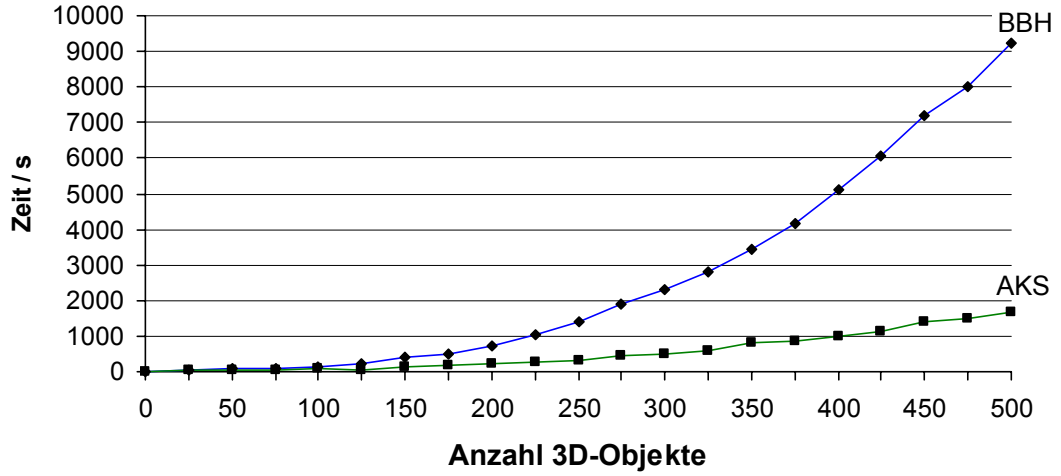


Abb. 3.179
Durchschnittliche Zeiten zur Konstruktion des Visibility Skeletons bei Verwendung von BB-Hierarchie und BB-Hierarchie mit Aufspaltung in Kind-Shafts

Der Verlauf der Kurven **BBH** und **AKS** erklärt sich dabei wie folgt: Mit den Ergebnissen des letzten Teilkapitels gilt, dass der Aufwand zur Konstruktion des Visibility Skeletons vom Aufwand zur Konstruktion der **E4**-MFLs dominiert wird, da für diese die meisten (Kombinationen von) Generator-Kandidaten betrachtet werden müssen. In 3D-Szenen, die im Wesentlichen aus 3D-Objekten gleicher Größenordnung bestehen, steigt die Zahl der Generator-Kanten in den **EE**-Shafts mit $\mathbf{O(\sqrt[3]{n})}$ und somit die Zahl der Paare von Generator-Kanten mit $\mathbf{O(n^{2/3})}$. Durch die Beschränkung der **EE**-Shafts bleibt die durchschnittliche Zahl der Generator-Kanten pro **EE**-Shaft dagegen fast konstant oder sinkt im besten Fall sogar.

An dieser Stelle sollte betont werden, dass dieser Effekt wesentlich vom Aufbau der betrachteten 3D-Szenen abhängt. Dazu zeigt Abb. 3.180 die Laufzeiten zur Konstruktion des Visibility Skeletons mit den Algorithmen **BBH** und **AKS** in drei 3D-Szenen mit jeweils 100–1000 3D-Objekten und folgenden Eigenschaften:

- 3D-Szene ***A*** *(Best Case)*: alle 3D-Objekte haben (ungefähr) die gleiche Größe
- 3D-Szene ***B*** *(Average Case)*: 50% „kleine“ 3D-Objekte, 40% „mittlere“ 3D-Objekte und 10% „große“ 3D-Objekte (jeweils Skalierung um Faktor 3).
- 3D-Szene ***C*** *(Worst Case)*: alle 3D-Objekte sind „Balken“, d.h. Quader, bei denen eine Seitenlänge um den Faktor 10 skaliert wurde.

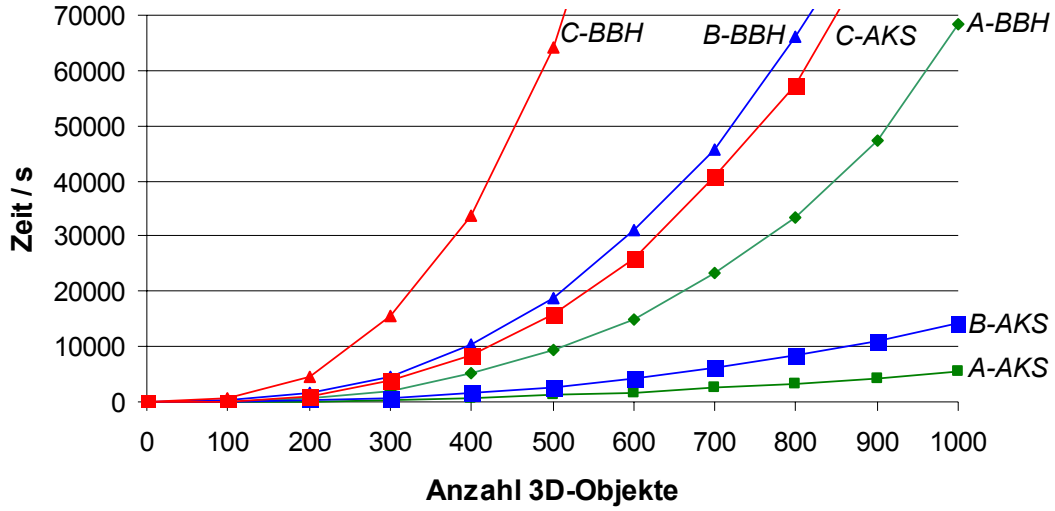


Abb. 3.180

Laufzeiten für 3D-Szenen ***A***, ***B*** und ***C*** bei Verwendung von BB-Hierarchie (**BBH**) und Aufspaltung in Kind-Shafts (**AKS**)

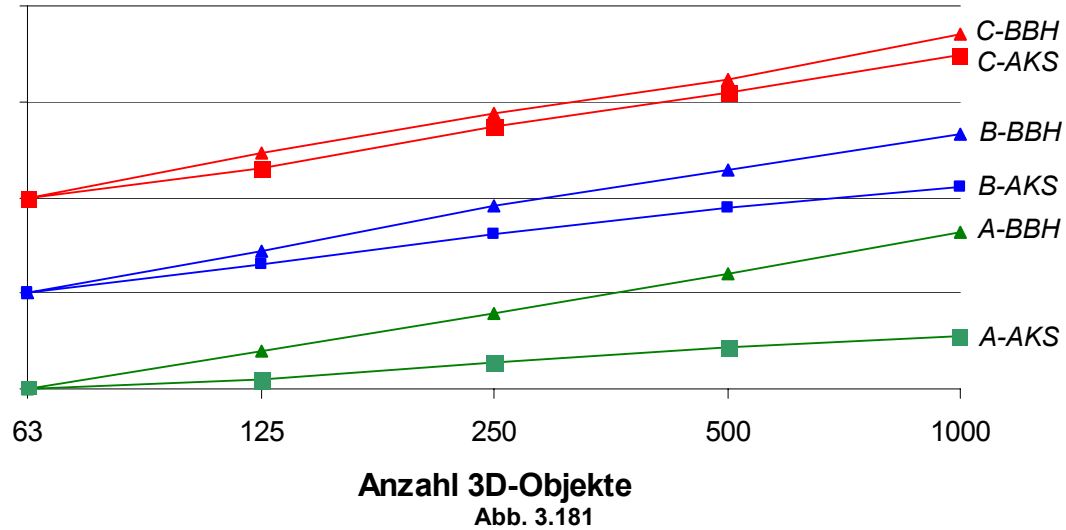


Abb. 3.181

Laufzeiten pro **EE**-Shaft für 3D-Szenen ***A***, ***B*** und ***C*** in doppelt logarithmischer Darstellung

Sowohl die absoluten Laufzeiten als auch die relative Veränderung der Laufzeiten unterscheiden sich stark zwischen den 3D-Szenen. Deutlich zu erkennen ist jedoch, dass durch die Aufspaltung in Kind-Shafts bzw. Beschränkung von Shafts in allen drei 3D-Szenen sowohl die absoluten Laufzeiten als auch deren relative Änderungen sinken.

Zur Verdeutlichung zeigt Abb. 3.181 die Laufzeiten in einer doppelt logarithmischen Darstellung[5], wobei der quadratische Anteil an den Laufzeiten eliminiert wurde. Die folgende Tabelle stellt die Abb. 3.181 entnommen Steigungen, d.h. die Exponenten **x** von $\mathbf{O(n^{2+x})}$, sowie deren Differenzen und Verhältnisse dar:

3D-Szene	*Exponent BBH*	*Exponent AKS*	*Differenz*	*Prozent*
A	≈0,82	≈0,27	≈0,54	≈33%
B	≈0,83	≈0,44	≈0,39	≈53%
C	≈0,85	≈0,75	≈0,10	≈88%

Tabelle 3.8

Die Verläufe der Kurven können für die 3D-Szenen ***A*** und ***B*** mit den bisherigen Ergebnissen direkt erklärt werden: In 3D-Szene ***A*** haben alle 3D-Objekte die gleiche Größe, daher ist mit den Ergebnissen des letzten Teilkapitels die durchschnittliche Zahl der 3D-Objekte in den **EE**-Shafts beschränkt bzw. sinkt im besten Fall.

In 3D-Szene ***B*** können je nach Aufbau der einzelnen **EE**-Shafts und den enthaltenen 3D-Objekten beide Verbesserungsvorschläge ansetzen: „Kleine“ **EE**-Shafts können durch die größeren 3D-Objekte effektiv beschränkt werden und für „große“ **EE**-Shafts können durch den Plane-Sweep viele Kombinationen „kleiner“ Generator-Kanten eliminert werden.

In 3D-Szene ***C*** greifen die vorgeschlagenen Verbesserungen nicht: Durch die stark unterschiedliche Seitenlänge der 3D-Objekte gibt es kaum **EE**-Shafts, die beschränkt werden können. Damit bleibt nur noch die Ausnutzung des Plane-Sweep-Verfahrens, das aufgrund der „langen“ Kanten in den **EE**-Shafts jedoch auch nicht greift. Als „Verbesserung“ bleibt hier lediglich ein linearer Faktor von ungefähr zwei, die Reduzierung des Exponenten um ≈0,1 ist marginal.

[5] Der Übersichtlichkeit wegen wurden die Aufpunkte der Geraden entlang der **y**-Achse so verschoben, dass Geraden, die zur selben 3D-Szene gehören, den gleichen Aufpunkt haben und Geraden verschiedener 3D-Szenen sich nicht schneiden.

Weitere Worst Cases ergeben sich, wenn es „viele“ 3D-Objekte gibt, die in der Größenordnung des umgebenden Raums liegen: Hier kann die oben verwendete Abschätzung von $\mathbf{O(\sqrt[3]{n})}$ nicht mehr angewendet werden, im schlimmsten Fall ergeben sich für $\mathbf{O(n^2)}$ Paare von „großen“ 3D-Objekten jeweils $\mathbf{O(n)}$ 3D-Objekte „in“ den Shafts (vgl. Abb. 3.174) und somit ein potentieller Gesamtaufwand von $\mathbf{O(n^4)}$.

Die oben vorgeschlagenen Verbesserungen durch die Aufpaltung in Kind-Shafts bzw. die Beschränkung von Shafts sowie den Plane-Sweep können in diesen Fällen nur greifen, wenn es sich bei diesen „großen“ 3D-Objekten nicht um „Balken“ handelt.

3.5.9 Zusammenfassung der exakten Verfahren

In diesem Teilkapitel wurde Verfahren zur exakten Berechnung der globalen Sichtbarkeit in einer 3D-Szene diskutiert. Anschaulich entspricht die exakte Sichtbarkeit der Berechnung aller „Schattenlinien“ in der 3D-Szene.

Diese „Schattenlinien“ werden durch Sichtbarkeitswechsel definiert. Ein Sichtbarkeitswechsel ist eine Fläche im 3D, die drei Generator-Kanten schneidet und von zwei Extremal-Elementen begrenzt wird. Schnitte von Sichtbarkeitswechseln definieren maximal freie Liniensegmente, die vier Generator-Kanten schneiden und von zwei Extremal-Elementen begrenzt werden. Die Gesamtheit der Sichtbarkeitswechsel und maximal freien Liniensegmente ergibt einen Graph: das Visibility Skeleton einer 3D-Szene.

Zunächst wurde die prinzipielle Konstruktion des Visibility Skeletons einer 3D-Szene diskutiert. Statt des in [21] definierten Verfahrens zum Schrittweisen Aufbau des Visibility Skeletons wurde ein Verfahren vorgeschlagen, das die Konstruktion der maximal freien Liniensegmente und der Sichtbarkeitswechsel sauber voneinander trennt. Mit diesem Verfahren könne die Sichtbarkeitswechsel ohne weitere geometrische Tests direkt aus den maximal freien Liniensegmenten konstruiert werden.

Anschließend wurde die Berechnung der maximal freien Liniensegmente anhand ausgewählter MFL-Typen erklärt. Insbesondere wurde dabei ein einfaches Verfahren zur Berechnung der **E4**-MFLs vorgeschlagen.

Da die Konstruktion aller maximal freien Liniensegmente einer 3D-Szene im Worst Case Aufwand $\mathbf{O(n^4)}$ hat, wurde die Konstruktion für alle MFL-Typen schrittweise verbessert. Dazu wurde das Konzept der Shaft-Klassifikation aus Kap. 3.4.3 aufgegriffen und an das Problem der MFL-Konstruktion angepasst.

Die grundlegende Idee hierbei war, durch Einfügen der 3D-Objekte in die Shafts diese schrittweise in Kind-Shafts aufzuspalten bzw. im Falle der **EE**-Shafts schrittweise zu beschränken. Das Ergebnis dieser Aufspaltung bzw. Beschränkung der Shafts ist eine Beschränkung der Zahl von Generator-Kandidaten in den Shafts. Für die Vertex- und Facetten-MFLs entfallen weiterhin die globale Sichtbarkeitstests und die Berechnung der Extremal-Elemente, da diese Informationen direkt aus den Kind-Shafts gewonnen werden können.

Mit all diesen Überlegungen konnte der Aufwand zur Konstruktion des Visibility Skeletons einer 3D-Szene im Average Case (d.h. 3D-Szenen mit $\mathbf{n}$ 3D-Objekten, die im Wesentlichen aus 3D-Objekten ungefähr gleicher Größenordnung $\mathbf{O(\sqrt[3]{n})}$ bestehen) von $\mathbf{O(n^{2,8})}$ auf ungefähr $\mathbf{O(n^{2,45})}$ reduziert werden, wobei in beiden Fällen $\mathbf{O(n^2)}$ Shafts betrachtet werden müssen. Für einen einzelnen Shaft bedeutet dies also, dass der durchschnittliche Aufwand von $\mathbf{O(n^{0,8})}$ auf $\mathbf{O(n^{0,45})}$ sinkt.

Der Aufwand für konkrete 3D-Szenen hängt jedoch in hohen Maßen von der Anordung und auch den Größen der 3D-Objekte ab: Für 3D-Szenen mit 3D-Objekten ungefähr gleicher Größe und Seitenverhältnisse ergibt sich mit den obigen Verbesserungen ein Aufwand von $\mathbf{O(n^{2,3})}$, während bei 3D-Objekten mit stark unterschiedlichen Seitenverhältnissen („Balken“) der Aufwand lediglich auf ungefähr $\mathbf{O(n^{2,75})}$ sinkt.

Der Worst Case ergibt sich für 3D-Szenen, die viele 3D-Objekte in der Größenordnung des umgebenden Raums enthalten. Der potentielle Gesamtaufwand ist hier $\mathbf{O(n^4)}$, wobei die vorgeschlagenen Verbesserungen auch hier greifen können, solange es sich bei diesen „großen“ 3D-Objekten nicht um „Balken“ handelt.

Zusammenfassend kann man also festhalten, dass die in diesem Kapitel erarbeiteten Verbesserungen zur Konstruktion der MFLs den Aufwand im besten Fall deutlich reduzieren, wobei konkrete Aussagen jedoch nur im Zusammenhang mit konkreten 3D-Szenen getroffen werden können.

4 Direct Illumination Visibility Skeleton

Im letzten Kapitel wurden zwei Verfahren zur Lösung des globalen Sichtbarkeitsproblems vorgestellt. Zunächst wird gezeigt, dass sich die Näherungsverfahren eher für kleine Energietransporte (indirekte Beleuchtung) und die exakten Verfahren eher für großen Energietransport (direkte Beleuchtung) eignen.

Aus dieser Erkenntnis ergibt sich eine neue Anforderung für das Visibility Skeleton einer 3D-Szene: es muss möglich sein, zwischen Sichtbarkeitswechseln bzw. MFLs für direkte und indirekte Beleuchtung zu unterscheiden und beide Teile getrennt zu berechnen. Der Teil des Visibility Skeletons, der zur direkten Beleuchtung gehört, wird in diesem Kapitel als *Direct Illumination Visibility Skeleton* (*DIVS*) bezeichnet.

Diese Problemstellung wird konkretisiert und dann in den folgenden Teilkapiteln schrittweise umgesetzt. Dazu wird zunächst das Problem der lokalen Sichtbarkeit von Lichtquellen betrachtet: 3D-Objekte, die von einer Lichtquelle direkt beleuchtet werden, müssen von der Lichtquelle lokal sichtbar sein.

Anschließend wird die Auswahl der 3D-Objekte, die zu direkten Beleuchtung gehören (d.h. direkt beleuchtet werden oder die direkte Beleuchtung anderer 3D-Objekte beeinflussen), diskutiert. Dazu werden die bisher betrachteten einfachen Shafts zu hierarchischen Shafts erweitert und darauf aufbauend die einfache Shaft-Klassifikation zur hierarchischen Shaft-Klassifikation verallgemeinert.

Auf diese Weise werden n-Tupel von Generator-Objekten gefunden, die Kandidaten für die Konstruktion von MFLs bzw. Sichtbarkeitswechseln sind, die zur direkten Beleuchtung gehören. Die eigentliche Konstruktion dieser MFLs und Sichtbarkeitswechsel verläuft dann im Wesentlichen analog zu den Überlegungen aus Kap. 3.4.6, d.h. aus Paaren von Generator-Elementen werden verallgemeinerte Shafts erstellt und die Kandidaten-Objekte anhand dieser Shafts klassifiziert.

Abschließend wird das vorgestellte Verfahren anhand konkreter Experimente und Messungen bewertet. Dabei wird insbesondere untersucht, wie sich verschiedene Anzahlen und Konstellationen von Lichtquellen in der 3D-Szene auf den Aufwand zur Konstruktion des Direct Illumination Visibility Skeletons auswirken.

4.1 Motivation und Problemstellung

Die Eigenschaften und der Aufwand zur Berechnung der Näherungs- und exakten Lösung des globalen Sichtbarkeitsproblems lassen sich prinzipiell nicht direkt vergleichen: die exakten Verfahren sind eine Vorstufe des Radiosity-Verfahrens während die Näherungsverfahren eine Rückkopplung mit dem Radiosity-Verfahren notwendig machen.

Um trotzdem Eigenschaften der beiden Verfahren vergleichen zu können, wird der folgende einfache Fall betrachtet: die gegenseitige Sichtbarkeit zweier Extremal-Elemente **A** und **B** wird von einem Generator-Objekt **G** beeinflusst. In Abb. 4.1 und Abb. 4.2 ist nochmals die Ausgangssituation für die betrachteten Verfahren dargestellt:

- Abb. 4.1: Beeinflussung der Sichtbarkeit von **A** und **B**, uniforme Zerlegung von **B** anhand der qualitativen Sichtbarkeitsanfrage (vgl. Kap. 3.4)
- Abb. 4.2: Beeinflussung der Sichtbarkeit von **A** und **B**, Zerlegung von **B** nach Schattenlinien (d.h. Sichtbarkeitswechseln) (vgl. Kap. 3.4.6)

Abb. 4.3 zeigt die Zerlegung von **B** bei steigender Radiosity (d.h. Lichtenergie) von **A**. Anschaulich kann man sich das Extremal-Element **A** als Lichtquelle mit Dimmer vorstellen, der schrittweise weiter aufgedreht wird.

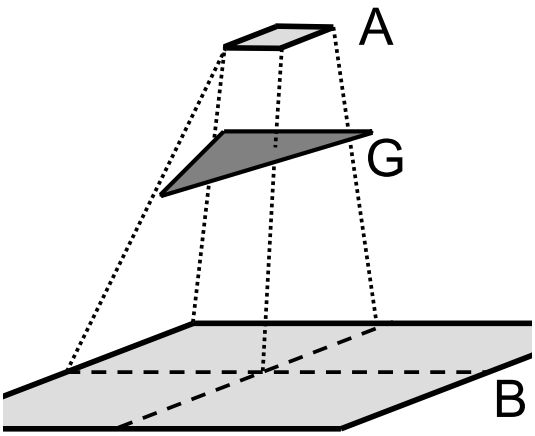


Abb. 4.1
Beeinflussung der Sichtbarkeit zwischen **A** und **B**, uniforme Zerlegung von **B**

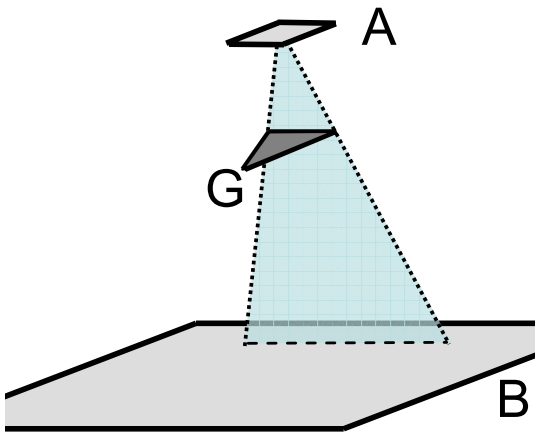


Abb. 4.2
Beeinflussung der Sichtbarkeit zwischen **A** und **B**, Zerlegung von **B** anhand von „Schattenlinien“

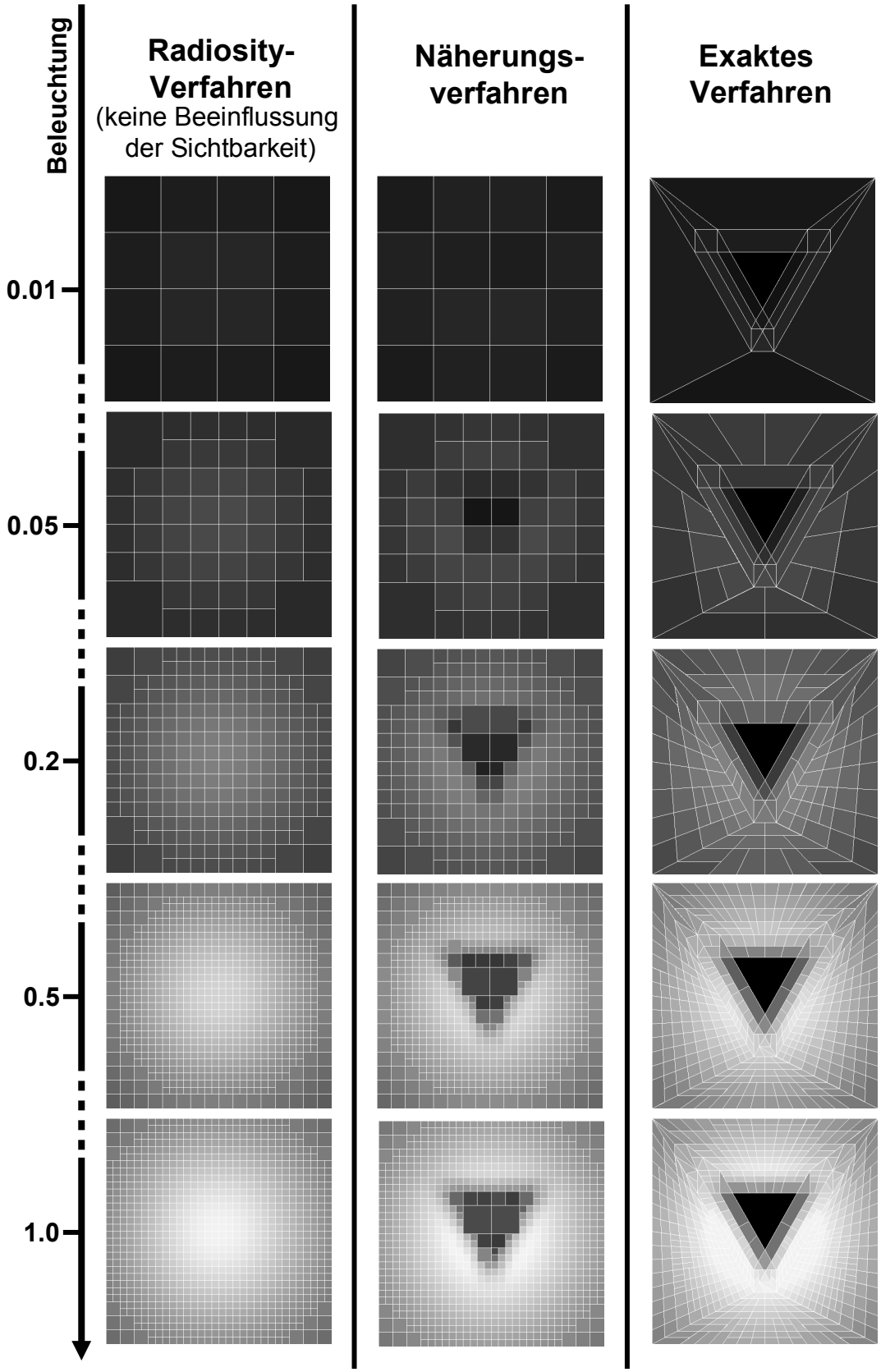


Abb. 4.3
Zerlegung von **B** anhand des Radiosity- sowie der Näherungs- und exakten Verfahren

Zunächst ist in der linken Spalte die vom Radiosity-Verfahren erzeugte, uniforme Aufteilung dargestellt: von oben nach unten steigt die Radiosity des Extremal-Elements **A**, also wird auch das Extremal-Element **B** immer feiner unterteilt, um die Änderung der Beleuchtung von **B** wiederzugeben.

In der mittleren Spalte ist die von den Näherungsverfahren erzeugte, uniforme Unterteilung dargestellt. Die Unterteilung entspricht der des Radiosity-Verfahrens mit dem Zusatz, dass gegenseitig vollständig nicht sichtbare Kind-Elemente von **A** und **B** nicht weiter unterteilt werden.

In der rechten Spalte ist die von den exakten Verfahren erzeugte Unterteilung anhand der Schattenlinien dargestellt. Bereits in der ersten Zeile, d.h. bei „kleiner" Beleuchtung von **B**, liegt die exakte Unterteilung anhand der Schattenlinien vor. Bei steigender Beleuchtung von **B** wird diese Unterteilung vom Radiosity-Verfahren weiter unterteilt.

Bereits an diesem einfachen Beispiel erkennt man jeweils für die Näherungs- und exakten Verfahren einen wesentlichen Nachteil:

(1) Nachteil der Näherungsverfahren

In jedem Unterteilungsschritt der Näherungsverfahren muss gestestet werden, ob die Kind-Elemente gegenseitig vollständig (nicht) oder teilweise sichtbar sind. Sind die Kind-Elemente vollständig (nicht) sichtbar, müssen für dieses Paar bzw. seine Kind-Elemente keine weiteren Sichtbarkeitsberechnungen durchgeführt werden.

Alle Paare $(\mathbf{a_i, b_i})$ von Kind-Elementen, die im „Halbschatten" von **G** liegen, sind gegenseitig teilweise sichtbar, d.h. für diese Paare müssen die Sichtbarkeitsanfragen bis zu den Blättern der Unterteilung ausgewertet werden. Dies bedeutet, dass bei immer feinerer Unterteilung von **B** aufgrund höheren Energieflusses zwischen **A** und **B** auch der (Halb-) Schatten immer feiner unterteilt wird, d.h. der Aufwand für die Näherungsverfahren steigt bei „wachsender Helligkeit".

Dies ist auch in Abb. 4.4 veranschaulicht. Die „Höhe" des Gitternetzes gibt dabei an, wie viele Sichtbarkeitsanfragen für ein Blatt der Unterteilung von **B** ausgewertet werden mussten.

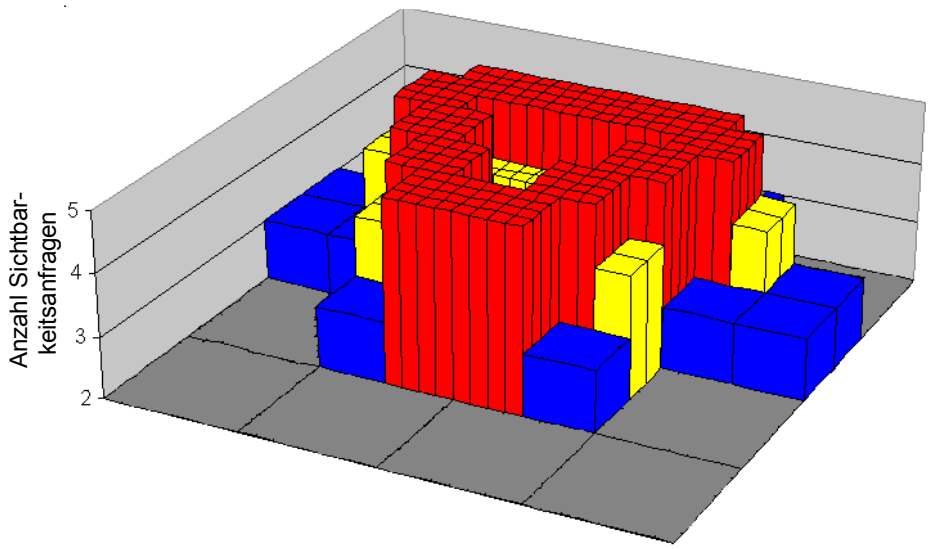


Abb. 4.4
Zahl der qualitativen Sichtbarkeitsanfragen für die Blätter der Unterteilung von **B**

(2) Nachteil der exakten Verfahren

Die exakten Verfahren sind eine Vorstufe des Radiosity-Verfahrens, d.h. die Unterteilung anhand der Sichtbarkeit wird unabhängig von der tatsächlichen Beleuchtung berechnet. In Abb. 4.3 ist zu erkennen, dass selbst bei minimaler Beleuchtung von **B** bereits eine sehr feine Unterteilung von **B** vorliegt.

Überspitzt ausgedrückt steckt in dieser feinen Unterteilung aber nur die Information, dass es „überall dunkel ist“, d.h. bei kleiner Beleuchtung liefern die exakten Verfahren viel zu genaue, also überflüssige Sichtbarkeitsinformationen.

Anschaulich sind die Näherungsverfahren also schlecht für „großen“ Energiefluss (d.h. „helle“ Extremal-Elemente) geeignet und die exakten Verfahren schlecht für „kleinen“ Energiefluss (d.h. „dunkle“ Extremal-Elemente). Der Energiefluss zwischen Extremal-Elementen wird erst im Radiosity-Verfahren berechnet, kann also nicht bereits in der Vorstufe der exakten Sichtbarkeitsunterteilung ausgenutzt werden. Daher muss man eine a priori Aufteilung der (Paare von) Extremal-Elemente(n) in „kleinen“ und „großen“ Energiefluss vornehmen.

Die einfachste Unterscheidung, die man hier finden kann, ist die Aufteilung in direkte Beleuchtung (d.h. eines der Extremal-Elemente ist eine Lichtquelle) und indirekte Beleuchtung (d.h. beide Extremal-Elemente sind keine Lichtquellen).

Diese Motivation der Einteilung in direkte und indirekte Beleuchtung ergibt also die folgende Problemstellung:

- Bei der Berechnung der gegenseitigen Sichtbarkeit von Extremal-Elementen muss zwischen direkter und indirekter Beleuchtung unterschieden werden.
- Für die direkte Beleuchtung muss die Sichtbarkeit mittels der exakten Verfahren berechnet werden, da hier auch in den „feinen" Unterteilungsstufen sinnvolle Information steckt.
  - Ein Paar **(A, B)** von Extremal-Elementen gehört zur direkten Beleuchtung, wenn eins der Extremal-Elemente eine Lichtquelle ist.
  - Für alle diese Paare **(A, B)** müssen die Sichtbarkeitswechsel (d.h. die „Schattenlinien") „zwischen" **A** und **B** gefunden werden.
- Für die indirekte Beleuchtung muss die Sichtbarkeit mittels der Näherungsverfahren berechnet werden, da hier bei „kleinem" Energiefluss auch eine grobe Berechnung der Sichtbarkeit ausreicht.
  - Ein Paar **(A, B)** von Extremal-Elementen gehört zur indirekten Beleuchtung, wenn keins der Extremal-Elemente eine Lichtquelle ist.
  - Für diese Paare sind keine weiteren Überlegungen notwendig. Alle Ergebnisse aus Kap. 3.4 können direkt auf dieses Teilproblem übertragen werden

Das primäre Ziel dieses Kapitels ist also die Berechnung aller Sichtbarkeitswechsel zwischen den Lichtquellen und den anderen 3D-Entitäten einer 3D-Szene.

Das grundlegende Problem dabei ist, dass die exakten Verfahren Generator-basiert arbeiten, d.h., dass sie ausgehend von Tupeln von Generator-Elementen maximal freie Liniensegmente bzw. Sichtbarkeitswechsel konstruieren und erst dann die zugehörigen Extremal-Elemente berechnen. Für die Berechnung des Direct Illumination Visibility Skeletons müssen die Sichtbarkeitswechsel bzw. MFLs aber ausgehend von den Extremal-Elementen berechnet werden.

4.2 Definition des $\mathcal{DIVS}$ und seiner Elemente

Mit den Erkenntnissen aus dem letzten Teilkapitel ist die Aufgabenstellung klar: es soll nur der Teil des Visibility Skeletons berechnet werden, der zur direkten Beleuchtung gehört. Bevor dies jedoch konkret umgesetzt wird, müssen zuvor noch die notwendigen Begriffe erarbeitet und definiert werden.

Dazu wird zunächst untersucht, welche maximal freien Liniensegmente zur direkten Beleuchtung gehören. Daraus ergeben sich dann auf natürliche Weise die zugehörigen Sichtbarkeitswechsel. Abschließend wird auf dieser Gesamtheit von MFLs und Sichtbarkeitswechseln ein Teilgraph des Visibility Skeletons, das Direct Illumination Visibility Skeleton, definiert.

Zunächst gilt mit der einfachen Definition der MFLs, dass ein MFL auf jedem Fall zur direkten Beleuchtung gehört, wenn eins seiner Extremal-Elemente eine Lichtquelle ist. Diese Bedingung reicht jedoch noch nicht aus. Dazu kann man Abb. 4.5 betrachten:

- **VEE$_1$**: die Lichtquelle **L** ist ein Extremal-Element von **VEE$_1$**, also gehört **VEE$_1$** zur direkten Beleuchtung
- **VEE$_2$**: die Lichtquelle **L** ist „hinteres“ Generator-Element von **VEE$_2$**. **VEE$_2$** gehört ebenfalls zur direkten Beleuchtung, da der adjazente Sichtbarkeitswechsel **EV** die Lichtquelle **L** als Extremal-Element hat
- **VEE$_3$**: die Lichtquelle **L** ist „mittleres“ Generator-Element von **VEE$_3$**. **VEE$_3$** gehört nicht zur direkten Beleuchtung, da kein zu **VEE$_3$** adjazenter Sichtbarkeitswechsel die Lichtquelle **L** als Extremal-Element haben kann.

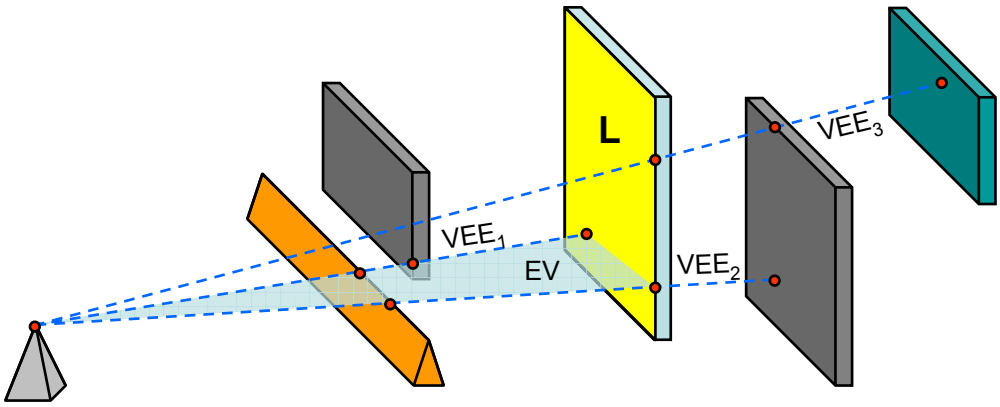


Abb. 4.5
VEE-MFLs haben Lichtquelle **L** als Extremal- bzw. „äußeres“ oder „mittleres“ Generator-Element

Betrachtet werden müssen also für die direkte Beleuchtung alle MFLs, die eine Lichtquelle als Extremal- oder „äußeres“ Generator-Element haben. Die folgenden Tabellen präzisieren den bisher noch vagen Begriff des „äußeren“ Generator-Elements:

Vertex-MFLs

MFL	*„äußere“ Gen.*
V_0V_1	V_0, V_1
VE_0E_1	V, E_1
E_0VE_1	E_0, E_1

Tabelle 4.1

Kanten-MFLs

MFL	*„äußere“ Gen.*
E	E
E4	E_0, E_3

Tabelle 4.2

Facetten-MFLs

MFL	*„äußere“ Gen.*
Fvv	F
FvE	F, E
FE_0E_1	F, E_1
E_0FE_1	E_0, E_1
F_0F_1	F_0, F_1

Tabelle 4.3

Damit können die maximal freien Liniensegmente, die zur direkten Beleuchtung gehören, definiert werden (vgl. Abb. 4.6):

Definition 4.1: α- und β-MFLs

Sei **M** ein beliebiges MFL. Dann heißt **M**…

- α-MFL, wenn mindestens ein Extremal-Element von **M** eine Lichtquelle ist.
- β-MFL, wenn ein „äußeres“ Generator-Element von **L** eine Lichtquelle ist.

Ist ein „äußeres“ Generator-Element eine Lichtquelle und gleichzeitig auch Extremal-Element von **L**, so heißt **L** ebenfalls β-MFL.

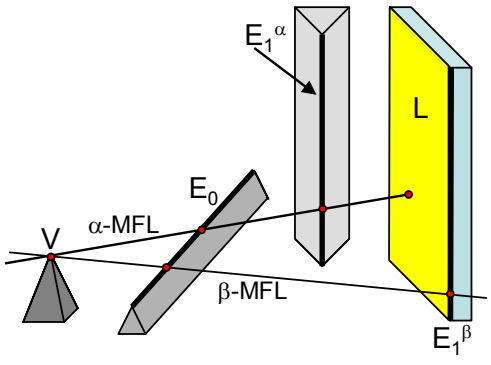


Abb. 4.6
α- und β-MFLs

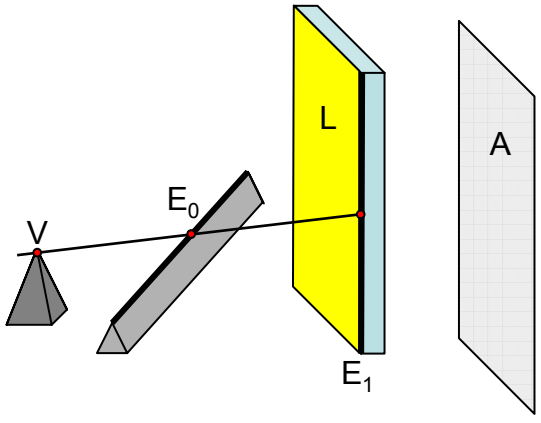


Abb. 4.7
β-MFL mit Abschluss-Generator-Element

Bevor auf die konkrete Konstruktion der α- und β-MFLs eingegangen wird, sollten die Extremal-Elemente der β-MFLs näher betrachtet werden: β-MFLs haben nach Definition eine Lichtquelle **L** als äußeres Generator-Element. Ist dieses Generator-Element gleichzeitig Extremal-Element, sind keine weiteren Überlegungen nötig. Andernfalls kann **L** als Extremal-Element gesetzt werden, da Sichtbarkeits-informationen „hinter" der Lichtquelle nicht benötigt werden (vgl. Abb. 4.7):

Satz 4.1: Extremal-Elemente von β-MFLs

Sei **M** eine β-MFL mit Lichtquellen-Generator-Element **L** und Extremal-Elementen **(E_0, E_1)** und sei o.B.d.A. **E_0** das Extremal-Element „hinter" **L**. Weiterhin sei **E_0** selber keine Lichtquelle.

Dann gilt, dass das Extremal-Element **E_0** ohne Informationsverlust für die direkte Beleuchtung durch **L** ersetzt werden kann, **M** also Extremal-Elemente **(L, E_1)** hat.

[Ohne Beweis]

Durch diese „Verkürzung" der MFL wird einerseits unwichtige Information unterdrückt, andererseits kann so auch die Strahlanfrage für das MFL beschleunigt werden, da Schnitte des Strahls „hinter" **L** nicht betrachtet werden müssen.

Definition 4.1 und Satz 4.1 beziehen sich zunächst nur auf MFLs, können aber direkt auf die Sichtbarkeitswechsel übertragen werden und werden daher nicht nochmals angegeben.

Abschließend ergibt sich aus den α- und β-MFLs bzw. Sichtbarkeitswechseln die zentrale Datenstruktur dieses Kapitels:

Definition 4.2: Direct Illumination Visibility Skeleton

Gegeben seien alle α- und β-MFLs bzw. Sichtbarkeitswechsel einer 3D-Szene. Dann heißt die durch diese MFLs und Sichtbarkeitswechsel induzierte Graphenstruktur

Direct Illumination Visibility Skeleton

(*DIVS*)

4.3 Konstruktion des $\mathcal{DIVS}$

Die Konstruktion des $\mathcal{DIVS}$ ist wesentlich komplizierter als die des einfachen Visibility Skeletons. Zunächst wird gezeigt, warum einfache Ansätze hierfür nicht geeignet sind. Anschließend wird die prinzipielle Vorgehensweise zur Konstruktion des $\mathcal{DIVS}$ geschildert und in drei Schritte aufgeteilt:

(1) Berechnung der lokalen Sichtbarkeit von den Lichtquellen

(2) Auswahl der Generator-Kandidaten, d.h. der (Tupel von) 3D-Objekte(n), die α- oder β-MFLs bzw. Sichtbarkeitswechsel definieren können.

(3) Konstruktion der α- und β-MFLs bzw. Sichtbarkeitswechsel

Dann wird die Konstruktion des $\mathcal{DIVS}$ anhand dieser drei Schritte vorgestellt. Zur Umsetzung jedes einzelnen Schritts wird auf die in Kap. 3.4 und Kap. 3.4.6 erarbeiteten Verfahren zurückgegriffen. Insbesondere wird für die Schritte (1) und (2) wieder das Konzept der verallgemeinerten Shafts aus Kap. 3.4.3 verwendet: Die Berechnung der lokalen Sichtbarkeit kann durch Umordnungen von Listen von Shaft-Elementen (vgl. Kap. 3.4.4.3) umgesetzt werden. Da die so gewonnenen lokalen Sichtbarkeitsinformationen in allen folgenden Schritten benötigt werden, werden sie in der BB-Hierarchie der 3D-Szene zwischengespeichert.

Für die Auswahl der Generator-Kandidaten werden die verallgemeinerten Shafts zu hierarchischen Shafts, d.h. Bäumen von Shafts, erweitert. Dies macht eine Anpassung der in Kap. 3.4.4.2 entwickelten Shaft-Klassifikation zur hierarchischen Shaft-Klassifikation, d.h. der Klassifikation einer BB-Hierarchie anhand einer Hierarchie von Shafts, notwendig.

Die eigentliche Konstruktion der α- und β-MFLs bzw. Sichtbarkeitswechsel wird mit einigen einfachen Anpassungen nahezu unverändert aus Kap. 3.5.7.2 übernommen.

Abschließend wird der so erarbeitete Algorithmus zur Konstruktion des $\mathcal{DIVS}$ anhand konkreter Experimente und Messungen untersucht und bewertet.

4.3.1 Prinzipielle Vorgehensweise

In diesem Teilkapitel wird die prinzipielle Vorgehensweise zur Konstruktion des $\mathcal{DIVS}$ einer 3D-Szenen vorgestellt. Da dies einige nicht-triviale zusätzliche Überlegungen notwendig macht, soll zuvor kurz aufgezeigt werden, warum einfache Ansätze zur Konstruktion nicht geeignet sind.

Dazu wird den Experimenten und Messungen in Kap. 4.4 vorgegriffen: dort werden vier 3D-Szenen mit jeweils 100 – 500 3D-Objekten betrachtet, die sich im Wesentlichen in der Zahl und Anordnung der Lichtquellen unterscheiden.

Die Details des Aufbaus dieser 3D-Szenen sind hier zunächst nicht wichtig. Betrachtet werden soll nur der Anteil des Direct Illumination Visibility Skeletons am vollständigen Visibility Skeleton. Dazu zeigt Abb. 4.8 den Prozentsatz der α- und β-MFLs (d.h. der MFLs, die zur direkten Beleuchtung gehören) in den vier 3D-Szenen.

Es ist deutlich zu erkennen, dass dieser Anteil mit steigender Zahl von 3D-Objekten immer kleiner wird. Der einfache Ansatz, alle MFLs zu berechnen und dann aus diesen die α- und β-MFLs auszuwählen, kann also als zu aufwändig verworfen werden.

Ein besserer Ansatz zur Konstruktion des $\mathcal{DIVS}$ sollte von den Lichtquellen der Szene ausgehen, denn sie sind die allen α- und β-MFLs bzw. Sichtbarkeitswechseln gemeinsamen Elemente.

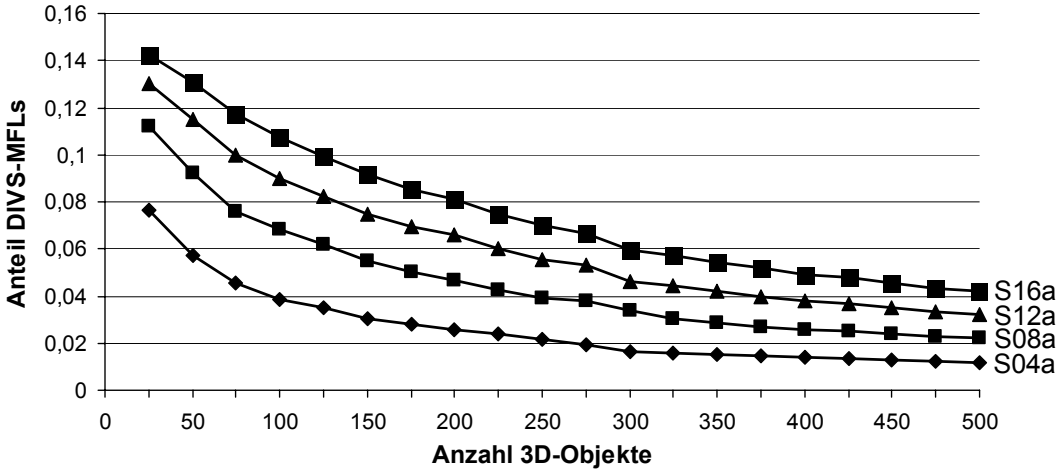


Abb. 4.8
Anteil von α- und β-MFLs in den in Kap. 4.4 betrachteten 3D-Szenen

Der wesentliche Aufwand zur Konstruktion des (Direct Illumination) Visibility Skeletons liegt in der Bildung von Kombinationen von Generator-Elementen. Daher ist es sicherlich eine schlechte Idee, das *DIVS* nacheinander für jede einzelne Lichtquelle zu berechnen. Bei diesem Ansatz müssten dieselben Kombinationen von Generator-Elementen mehrfach gebildet und auf *DIVS*-Eigenschaften untersucht werden. Weiterhin kann man so nicht ausnutzen, dass Lichtquellen in „realen“ Szenen oft in Clustern (anschaulich z.B. ein Kronleuchter) auftreten und somit wiederum anschaulich zunächst getestet werden kann, ob eine MFL den ganzen Kronleuchter schneidet.

Um die folgenden Ausführungen etwas zu vereinfachen, werden zunächst nur Szenen mit genau einer Lichtquelle **L** (d.h. einem 3D-Objekt mit mindestens einer Lichtquellen-Facette) betrachtet.

Gegeben sei also eine 3D-Szene **S** mit 3D-Objekten $\mathbf{O_i}$ und genau einer Lichtquelle **L** ($\mathbf{=O_0}$, vgl. Abb. 2.13). Gesucht sind alle 3D-Objekte, die mit **L** ein α- oder β-MFL bilden können. Dies sind zunächst die Objekte, die von **L** aus lokal sichtbar sind, d.h. (teilweise) im positiven Halbraum mindestens einer Lichtquellen-Facette von **L** liegen (vgl. Abb. 4.9).

Für ein Objekt $\mathbf{O_j}$, das lokal von einer Lichtquelle sichtbar ist, müssen nun alle anderen Objekte $\mathbf{O_k}$ gefunden werden, die mit **L** und $\mathbf{O_j}$ ein β-MFL (d.h. ein MFL, für das die Lichtquelle **L** selber ein Generator-Element ist) bilden können.

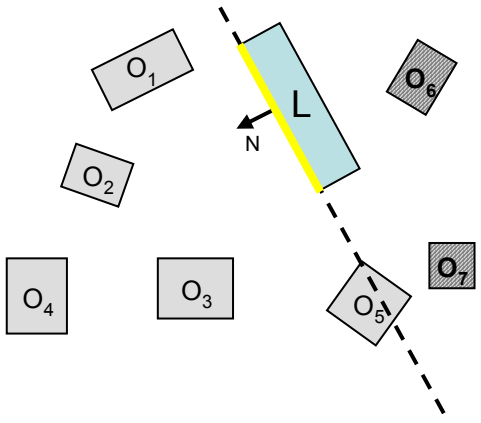


Abb. 4.9
Von Lichtquelle lokal sichtbare Objekte

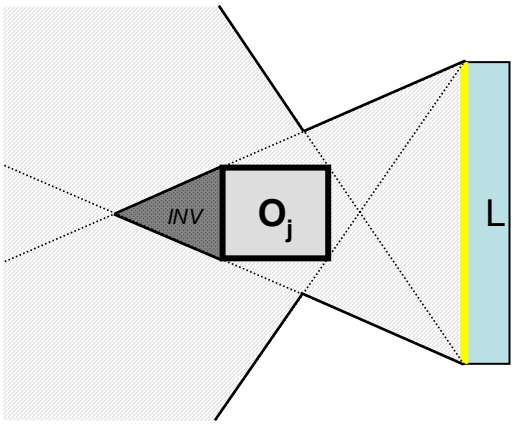


Abb. 4.10
Bereich für α- und β-MFLs

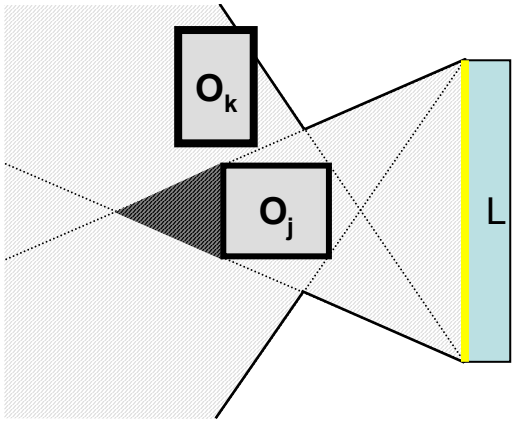


Abb. 4.11
Objekt O_k im Bereich für α- und β-MFLs

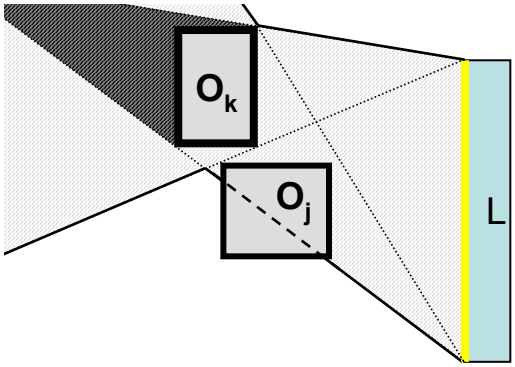


Abb. 4.12
Objekt O_j im Bereich für α- und β-MFLs

Weiterhin müssen für festes O_k alle 3D-Objekte O_l gefunden werden, für die (O_j, O_k, O_l) ein α-MFL (d.h. ein MFL, für das die Lichtquelle Extremal-Element ist) bilden können. Für **E4**-MFLs kommt sogar noch ein dritter Kandidat für die β-MFLs und ein vierter Kandidat für die α-MFLs hinzu.

In Abb. 4.10 ist für eine Lichtquelle **L** und ein festes Objekt O_j der Bereich, in dem Kandidaten für α- und β-MFLs liegen können, schraffiert dargestellt. Eine Anfrage für diesen Bereich kann ohne weiteres in Form eines verallgemeinerten Shafts formuliert werden, ergibt allerdings ein Problem:

In Abb. 4.11 ist ein 3D-Objekt O_k dargestellt, dass im Bereich für α- und β-MFLs liegt. Wie man in Abb. 4.12 leicht sieht, liegt umgekehrt O_j im Bereich der α- und β-MFLs für das Paar (O_k, L), d.h. das Kandidatenpaar (O_j, O_k) wird doppelt betrachtet. Natürlich kann dieses Problem ohne weiteres durch einen Vergleich der eindeutigen Objekt-IDs gelöst werden, indem man nur Paare (O_j, O_k) mit $j < k$ betrachtet.

Dies ändert jedoch nichts an der relativ komplizierten Ausprägung des Shafts: Nach Abb. 4.10 besteht er aus dem „normalen" Shaft $S(O_j, O_k)$, dem inversen Shaft $S^-(O_j, O_k)$ sowie der „Verlängerung" des Shafts $S(O_j, O_k)$ für den global von **L** nicht sichtbaren Bereich.

Beide Probleme kann man vermeiden, indem man einfach den Shaft $S(O_j, L)$ betrachtet. Alle Objekte in diesem Shaft können mit O_j (und **L**) α- bzw. β-MFLs definieren und bis auf wenige Ausnahmen (vgl. Abb. 4.14) gilt

$$O_k \cap S(O_j, L) \neq \varnothing \Rightarrow O_j \cap S(O_k, L) \neq \varnothing$$ (vgl. Abb. 4.13).

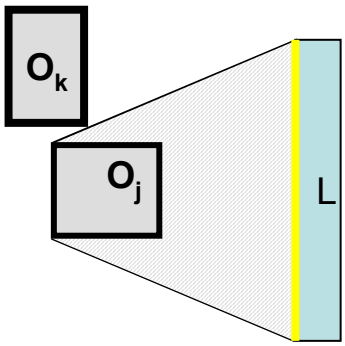


Abb. 4.13
Objekte liegen i.a. nicht gegenseitig in ihrem Lichtquellen-Shafts

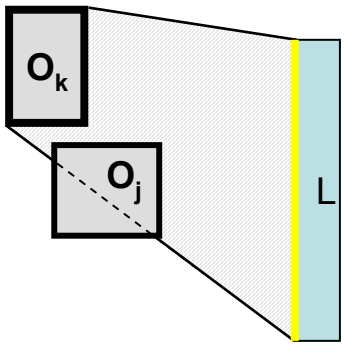


Abb. 4.14
Sonderfall: Objekte „nahe bei einander“ können gegenseitig in ihren Lichtquellen-Shafts liegen.

Die Eigenschaft, dass Objekte im Shaft **S(O_j, L)** α- bzw. β-MFLs definieren können, ist mit den bisherigen Ausführungen zwar offensichtlich, soll aber aufgrund der zentralen Bedeutung für den Rest des Kapitel kurz bewiesen werden:

Satz 4.2: Shaft-Bedingung für α- und β-MFLs

Gegeben sei eine Lichtquelle **L**, ein von **L** lokal sichtbares 3D-Objekt **O** sowie der Shaft **S(O, L)**. Dann gilt, dass alle α- bzw. β-MFLs, die **O** als Generator-Element enthalten, **S(O, L)** schneiden.

Beweis:

Sei **M** eine α- bzw. β-MFL, die **O** als Generator-Element enthält. Die Definition der α- bzw. β-MFLs besagt, dass **M** ein Generator- und/oder Extremal-Element aus **L** haben muss, d.h. **M** schneidet **L**. Weiterhin gilt mit der Definition der MFLs, dass die Schnittpunkte von **M** mit **O** und **L** vom jeweils anderen Objekt aus lokal sichtbar sein müssen.

Damit gilt also, dass **M** ein Liniensegment ist, das zwischen lokal sichtbaren Facetten von **O** und **L** verläuft. Da der Shaft **S(O, L)** die konvexe Hülle aller Linien mit dieser Eigenschaft ist, gilt trivial, dass **M** zumindest teilweise innerhalb von **S(O, L)** verläuft, d.h. **S(O, L)** schneidet.

Mit diesem Satz gilt also auch, dass alle Generator-Kandidaten für α- bzw. β-MFLs mit O_j und **L** im Shaft **S(O_j, L)** liegen müssen. Weiterhin gilt somit auch, dass ein Objekt, dass den Shaft **S(O_j, L)** vollständig verdeckt, auch alle α- bzw. β-MFLs zwischen O_j und **L** blockiert, d.h. O_j muss nicht weiter betrachtet werden.

Gibt es kein solches, vollständig verdeckendes 3D-Objekt, können mit den Generator-Kandidaten im Shaft $\mathbf{S(O_j, L)}$ die einzelnen (Gruppen von) MFL-Typen konstruiert werden. In Kap. 4.3.3.1 und Kap. 4.3.4.3 werden einige Ausnahmen von dieser Regel angegeben.

Mit diesen Überlegungen ergibt sich der Arbeitsplan für dieses Kapitel:

KONSTRUKTION DES DIRECT ILLUMINATION VISIBILITY SKELETONS:

(1) **Finde alle von Lichtquellen L_j lokal sichtbaren 3D-Objekte O_i** (Kap. 4.3.2)

(2) **Für jedes dieser 3D-Objekte O_i:**

- *a.* **Bilde Shafts $S(O_i, L_j)$ mit allen Lichtquellen** (Kap. 4.3.3.1 und 4.3.3.3)
- *b.* **Finde 3D-Objekte O_k, die „in" Shafts $S(O_i, L_j)$ liegen** (Kap. 4.3.3.3)
- *c.* **Bilde n-Tupel von Generator-Elementen „in" Shafts $S(O_i, L_j)$ und teste, ob sie ein gültiges α- bzw. β-MFL definieren** (Kap. 4.3.4)

(3) **Konstruiere α- bzw. β-Sichtbarkeitswechsel aus den MFLs** (Kap. 4.3.5)

Die folgenden Teilkapitel werden zeigen, dass bei der Konstruktion des *DIVS* nach dem obigen Verfahren immer wieder Teilergebnisse und temporäre Datenstrukturen wieder verwendet werden können. Diese eigentlich wünschenswerte Eigenschaft der *DIVS*-Konstruktion wirft jedoch einige Probleme bei der Diskussion der einzelnen Konstruktionsschritte auf: oft wird erst später klar, warum an einer bestimmten Stelle Teilergebnisse zwischengespeichert werden sollten, bzw. warum an einer anderen Stelle Teilergebnisse wieder verwendet werden können.

Daher ist jedem der folgenden Teilkapitel ein kurzer Überblick über den jeweils aktuellen Stand der *DIVS*-Konstruktion in der folgenden Form vorangestellt:

Stand	*Was wurde bisher erreicht, welche Daten wurden zwischengespeichert?*
Teilziel	*Welcher Schritt der Konstruktion wird in diesem Teilkapitel behandelt?*
Ansatz	*Wie wird dieses Teilziel prinzipiell erreicht?*
Ausblick	*Warum ist dieser Schritt für die folgenden Schritte notwendig?*

4.3.2 Lokale Sichtbarkeit von den Lichtquellen

Stand	Es liegt eine 3D-Szene mit mindestens einer Lichtquelle vor. Die Szene ist in Form einer BB-Hierarchie gespeichert, dabei liegen Nicht-Lichtquellen und Lichtquellen in getrennten Teilbäumen (vgl. Kap. 2.3.2, Abb. 2.14 und Abb. 2.15)
Teilziel	Es sollen alle von mindestens einer Lichtquelle lokal sichtbaren 3D-Objekte gefunden werden. Weiterhin sollen diese lokalen Sichtbarkeitsinformationen in der BB-Hierarchie zwischengespeichert werden
Ansatz	Die BB-Hierarchie wird anhand der Lichtquellen-Facetten klassifiziert, d.h. getestet, ob eine B-Box im positiven Halbraum der Lichtquellen-Facetten liegt.
Ausblick	Nur 3D-Objekte, die von mindestens einer Lichtquelle lokal sichtbar sind, können α- oder β-MFLs mit den Lichtquellen definieren.

Die Berechnung der lokalen Sichtbarkeit von den Lichtquellen ist der mit Abstand einfachste Schritt der *DIVS*-Konstruktion. In den folgenden Schritten können diese lokalen Sichtbarkeitsinformationen auf vielfältige Weise ausgenutzt werden:

- Teilbäume der BB-Hierarchie, die von allen Lichtquellen lokal nicht sichtbar sind, müssen in den folgenden Schritten nicht mehr betrachtet werden.
- Ein 3D-Objekt, das die Ebene einer Lichtquellen-Facette schneidet (d.h. von dieser teilweise lokal sichtbar ist), kann mit der Facette β-Facetten-MFLs bilden (vgl. Kap. 4.3.4.3).
- Die lokalen Sichtbarkeitsinformationen zwischen einer Lichtquelle $\mathbf{L_j}$ und einem 3D-Objekt $\mathbf{O_i}$ können zur Konstruktion des Shafts $\mathbf{S(O_i, L_j)}$ verwendet werden (vgl. Kap. 4.3.3.1).

Die BB-Hierarchie der 3D-Szene sollte also einmal vollständig traversiert werden und dabei die lokalen Sichtbarkeitsinformationen bezüglich der Lichtquellen in den 3D-Objekten bzw. der BB-Hierarchie gespeichert werden.

In einer Szene mit $\mathbf{n}$ 3D-Objekten, von den $\mathbf{m}$ Lichtquellen sind, würde dies zunächst einen Rechen- bzw. Speicheraufwand von $\mathbf{O(n \cdot m)}$ notwendig machen, da jedes der $\mathbf{n}$ 3D-Objekte (bzw. der $\mathbf{O(n)}$ B-Boxen) den Sichtbarkeitsstatus bezüglich $\mathbf{O(m)}$ Lichtquellen (-Facetten) berechnen bzw. speichern müsste.

Hier kann jedoch eine bereits in Kap. 3.4.4.3 betrachtete Eigenschaft ausgenutzt werden: wenn für eine Lichtquellen-Facette **F** gilt, dass eine B-Box **B** vollständig in ihrem positiven bzw. negativen Halbraum liegt, so liegen auch alle Kind-Boxen von **B** im positiven bzw. negativen Halbraum.

Damit ergibt sich der folgende Ansatz für die Speicherung der lokalen Sichtbarkeitsinformationen bezüglich Lichtquellen-Facetten: jede B-Box **B** speichert die Liste der Lichtquellen-Facetten, bezüglich derer sie vollständig im positiven Halbraum liegt, d.h. von denen aus sie vollständig lokal sichtbar ist. Die Lichtquellen-Facetten in dieser Liste müssen dann in keiner Kind-Box von **B** mehr gespeichert werden. Lichtquellen-Facetten, bezüglich derer die B-Box **B** vollständig im negativen Halbraum liegen, müssen nicht gespeichert werden.

In den Blättern der BB-Hierarchie, d.h. den 3D-Objekten **O**, wird dann jeweils die Liste der Lichtquellen-Facetten gespeichert, für die keine eindeutige Klassifikation vorliegt, d.h. für die das 3D-Objekt **O** teilweise lokal sichtbar ist.

Abb. 4.15 zeigt ein einfaches Beispiel: eine 2D-Szene mit fünf 2D-Objekten und zwei Lichtquellen mit insgesamt drei Lichtquellen-Facetten, sowie die BB-Hierarchie. In Abb. 4.16 ist der zugehörige Baum dargestellt. Jeder Knoten hält einen Verweis auf die Liste der Lichtquellen-Facetten, bezüglich derer er vollständig im positiven Halbraum liegt. Die Blätter des Baums halten Verweise auf die Lichtquellen-Facetten, deren Ebene das im jeweiligen Blatt gespeicherte 2D-Objekt schneiden.

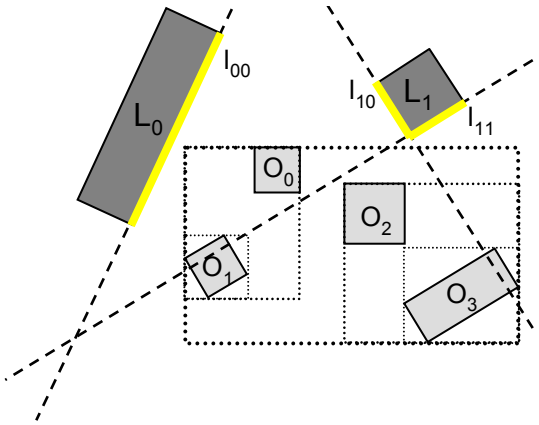


Abb. 4.15
3D-Szene mit Lichtquellen

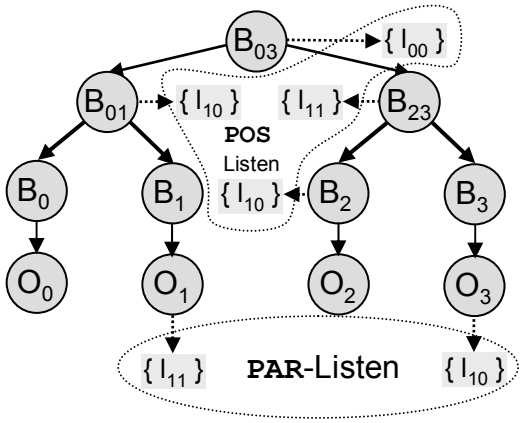


Abb. 4.16
Szene-Baum mit **POS**- und **PAR**-Listen

Für die effiziente Berechnung dieser Klassifikationen kann das Konzept der Umordnung von Shaft-Element-Listen aus Kap. 3.4.4.3 aufgegriffen werden:

Die Liste **L** aller Lichtquellen-Facetten wird in drei Abschnitte **NEG**, **PAR** und **POS** unterteilt. Der **NEG**- (**POS**-) Abschnitt der Liste enthält alle Lichtquellen-Facetten, bezüglich derer die aktuelle B-Box vollständig im negativen (positiven) Halbraum liegt. Der **PAR**-Abschnitt der Liste enthält alle Lichtquellen-Facetten, deren Ebene die aktuelle B-Box schneidet. Weiterhin gilt, dass die Wurzel der BB-Hierarchie alle Lichtquellen-Facetten schneidet, da die Wurzel die gesamte Szene, also auch alle Lichtquellen-Facetten, umfasst.

Bei der Traversierung der BB-Hierarchie müssen in jedem Schritt nur die Lichtquellen-Facetten im **PAR**-Abschnitt der Liste betrachtet werden, die Lichtquellen-Facetten im **NEG**-/**POS**-Abschnitt der Liste sind bereits eindeutig klassifiziert.

Wird die aktuelle B-Box anhand einer Lichtquellen-Facette im **PAR**-Abschnitt der Liste als **NEG** bzw. **POS** klassifiziert, so wird die Lichtquellen-Facette in den **NEG**- bzw. **POS**-Abschnitt der Liste verschoben. Zusätzlich wird jede als **POS** klassifizierte Lichtquellen-Facette der **POS**-Liste der aktuellen B-Box hinzugefügt.

Damit liegt nun eine effiziente Methode zur Berechnung und Speicherung der lokalen Sichtbarkeitsinformationen von B-Boxen bzw. 3D-Objekten bezüglich der Lichtquellen (-Facetten) vor. In allen folgenden Schritten…

- …müssen keine weiteren Klassifikationen anhand der Lichtquellen-Facetten vorgenommen werden, da alle lokalen Sichtbarkeitsinformationen in der BB-Hierarchie gespeichert sind.
- …müssen nur die Teilbäume der BB-Hierarchie traversiert werden, die nicht-leere **POS**- bzw. **PAR**-Listen enthalten, d.h. von mindestens einer Lichtquelle lokal sichtbar sind.

In Kap. 4.4 wird der Aufwand zur Berechnung und Speicherung der lokalen Sichtbarkeitsinformationen anhand konkreter Messungen untersucht.

4.3.3 Auswahl der Generator-Kandidaten

Stand	Die BB-Hierarchie der 3D-Szene ist mit lokalen Sichtbarkeits-informationen bezüglich der Lichtquellen (-Facetten) attributiert, d.h. jede B-Box „weiß“, von welchen Lichtquellen (-Facetten) sie lokal sichtbar ist.
Teilziel	Es sollen für ein festes 3D-Objekt $\mathbf{O_i}$ alle 3D-Objekte $\mathbf{O_k}$ gefunden werden, die mit $\mathbf{O_i}$ und den Lichtquellen $\mathbf{L_j}$ α- oder β-MFLs definieren können, d.h. MFLs, die eine Lichtquelle als Extremal- oder „äußeres“ Generator-Element enthalten.
Ansatz	Nach Satz 4.2 können nur Objekte $\mathbf{O_k}$ „im“ Shaft $\mathbf{S(O_i, L_j)}$ α- oder β-MFLs definieren. Statt alle 3D Objekte $\mathbf{O_k}$ nacheinander anhand aller Shafts $\mathbf{S(O_i, L_j)}$ zu klassifizieren, werden die Shafts zu hierarchischen Shafts erweitert und die BB-Hierarchie anhand dieser hierarchischen Shafts klassifiziert.
Ausblick	Alle 3D-Objekte, die in (mindestens) einem Shaft $\mathbf{S(O_i, L_j)}$ liegen, können mit $\mathbf{O_i}$ und $\mathbf{L_j}$ α- oder β-MFLs definieren.

Mit den Ergebnissen des letzten Teilkapitels kann nun auf einfache Weise nur der Teil der BB-Hierarchie traversiert werden, der von mindestens einer Lichtquelle lokal sichtbar ist. Der nächste Schritt der *DIVS*-Konstruktion besteht darin, für jedes von mindestens einer Lichtquelle lokal sichtbare 3D-Objekt $\mathbf{O_i}$ diejenigen Kandidaten-Objekte $\mathbf{O_k}$ zu bestimmen, die mit $\mathbf{O_i}$ eine α- oder β-MFL bilden können.

Mit Satz 4.2 ergibt sich, dass dies genau die Objekte $\mathbf{O_k}$ sind, die in den Shafts $\mathbf{S(O_i, L_j)}$ liegen. Zunächst ist also die Berechnung dieser Lichtquellen-Shafts $\mathbf{S(O_i, L_i)}$ zu diskutieren. Auf diesen Lichtquellen-Shafts kann eine Hierarchie definiert werden. Dies macht weiterhin eine Anpassung der Shaft-Klassifikationen aus Kap. 3.4.3 notwendig.

Da die Kandidaten-Objekte $\mathbf{O_k}$ eventuell mehrfach traversiert werden müssen (z.B. für eine **E4**-MFL bis zu dreimal), ist es sinnvoll, die Shaft-Klassifikationen analog zu den lokalen Sichtbarkeitseigenschaften in der BB-Hierarchie zu speichern. Dies ist hier jedoch wesentlich komplizierter und aufwändiger und führt zu den so genannten Shaft-Signaturen, die ebenfalls kurz diskutiert werden.

4.3.3.1 Berechnung der Lichtquellen-Shafts

Stand	Wie Kap. 4.3.3: die BB-Hierarchie der 3D-Szene ist mit lokalen Sichtbarkeitsinformationen bezüglich der Lichtquellen-Facetten attributiert.
Teilziel	Für ein festes 3D-Objekt $\mathbf{O_i}$ und eine feste Lichtquelle $\mathbf{L_j}$ soll der Lichtquellen-Shaft $\mathbf{S(O_i, L_j)}$ konstruiert werden.
Ansatz	Ausgangspunkt für die Konstruktion von Shafts sind die Konturkanten der 3D-Objekte, die den Shaft definieren. Diese Konturkanten können aus den lokalen Sichtbarkeitsinformationen zwischen dem 3D-Objekt $\mathbf{O_i}$ und der Licht-quelle $\mathbf{L_j}$ berechnet werden.
Ausblick	Klassifikation von 3D-Objekten anhand der Lichtquellen-Shafts liefert Kandidaten-Objekte für die Konstruktion der α- und β-MFLs.

Um den Shaft eines 3D-Objekts $\mathbf{O_i}$ mit einer Lichtquelle $\mathbf{L_j}$ zu berechnen, müssen zunächst die Konturkanten der beiden 3D-Objekte berechnet werden. Mit den Ergebnissen aus Kap. 4.3.2 ist für jedes 3D-Objekt bekannt, von welchen Lichtquellen-Facetten es (teilweise) lokal sichtbar ist, d.h. bezüglich welcher Lichtquellen-Facetten es ganz oder teilweise im negativen bzw. positiven Halbraum liegt.

Aus dieser Information lassen sich direkt die Konturkanten von $\mathbf{L_j}$ rekonstruieren: eine Kante **k** von $\mathbf{L_j}$ ist Konturkante bezüglich $\mathbf{O_i}$, wenn $\mathbf{O_i}$ bezüglich einer zu **k** adjazenten Facette vollständig im negativen Halbraum und bezüglich der anderen Facette vollständig oder teilweise im positiven Halbraum liegt (vgl. Abb. 4.17).

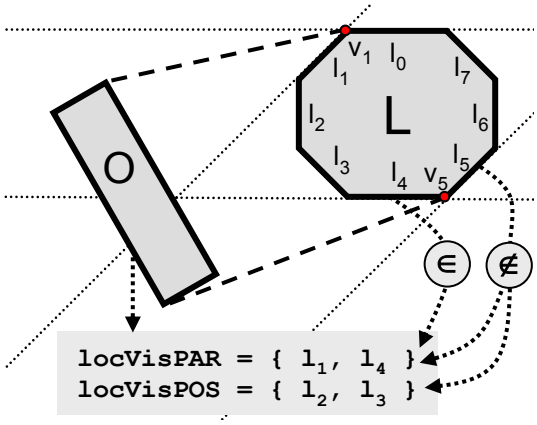


Abb. 4.17
Konturkanten von **L** ergeben sich aus den `locVisPos/Par`-Listen von **O**

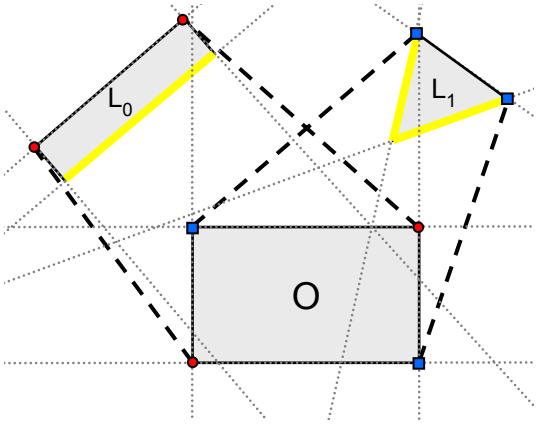


Abb. 4.18
2D-Szene mit Facetten-Ebenen und Konturvertices

Um die Konturkanten von $\mathbf{O_i}$ bezüglich $\mathbf{L_j}$ zu berechnen, kann man das Verfahren zur Berechnung der lokalen Sichtbarkeit aus Kap. 4.3.2 einfach „umdrehen“, d.h. man beginnt mit einer Liste der Facetten von $\mathbf{O_i}$ und klassifiziert die BB-Hierarchie der Lichtquellen anhand dieser Liste von Facetten. Damit erhält man die Klassifikation aller Lichtquellen anhand der Facetten des 3D-Objekts $\mathbf{O_i}$ und kann daraus die Konturkanten von $\mathbf{O_i}$ bezüglich $\mathbf{L_j}$ berechnen.

Mit den Konturkanten des 3D-Objekts $\mathbf{O_i}$ bezüglich der Lichtquelle $\mathbf{L_j}$ und denen von $\mathbf{L_j}$ bezüglich $\mathbf{O_i}$ kann der Lichtquellen-Shaft $\mathbf{S(O_i, L_j)}$ konstruiert werden.

Abb. 4.18 zeigt zur Veranschaulichung dieser Überlegungen eine 2D-Szene mit einem 2D-Objekt und zwei 2D-Lichtquellen. Zusätzlich sind die Ebenen der Objekt- und Lichtquellen-Facetten sowie die sich daraus ergebenden Konturkanten (im 2D: Konturvertices) eingezeichnet.

Anzumerken ist hier noch, dass auch bei dieser Konstruktion der Lichtquellen-Shafts wieder zusätzliche Information für die weiteren Schritte zur Konstruktion des $\mathcal{DIVS}$ abfällt:

- Objekt-Facetten, von denen eine Lichtquelle lokal nicht sichtbar ist, können keine α- oder β-MFLs mit dieser Lichtquelle bilden. Gleiches gilt für Kanten und Vertices, von deren adjazenten Facetten eine Lichtquelle lokal nicht sichtbar ist.
- Objekt-Facetten, von denen keine Lichtquelle lokal sichtbar ist, müssen nicht weiter betrachtet werden
- Der Schnitt der Lichtquellen mit den Ebenen der Objekt-Facetten kann α- oder β-Facetten-MFLs definieren.

Mit diesen Überlegungen können für ein festes 3D-Objekt $\mathbf{O_i}$ und alle Lichtquellen $\mathbf{L_j}$ die Lichtquellen-Shafts $\mathbf{S(O_i, L_j)}$ konstruiert werden. Der Aufwand für diese Konstruktion hängt im Wesentlichen vom Aufwand zum Finden der Konturkanten ab, der wiederum dem Aufwand zur Berechnung der lokalen Sichtbarkeit von 3D-Objekten und Lichtquellen entspricht.

4.3.3.2 Hierarchische Shafts

Stand	Für ein festes 3D-Objekt $\mathbf{O_i}$ und alle **m** Lichtquellen $\mathbf{L_j}$ sind die Lichtquellen-Shafts $\mathbf{S(O_i, L_j)}$ berechnet und in den Lichtquellen gespeichert.
Teilziel	Die **m** Lichtquellen-Shafts $\mathbf{S(O_i, L_j)}$ sollen zu einer Hierarchie von Lichtquellen-Shafts zusammengefasst werden.
Ansatz	Die Lichtquellen $\mathbf{L_j}$ sind in einem separaten Teilbaum der BB-Hierarchie gespeichert. Für jeden Knoten dieses Teilbaums werden alle Shaft-Ebenen und –Linien gesucht, die alle Kind-Knoten dieses Knotens umfassen.
Ausblick	Klassifikation von 3D-Objekten anhand der hierarchischen Lichtquellen-Shafts liefert Kandidaten für die Konstruktion der α- und β-MFLs.

Die einzelnen Shafts $\mathbf{S(O, L_j)}$ können mit dem oben geschilderten Verfahren auf einfache und effektive Weise konstruiert werden. Im nächsten Schritt müssen die 3D-Objekte der 3D-Szenen anhand aller Lichtquellen-Shafts klassifiziert werden, um so die Kandidaten-Objekte Konstruktion der α- und β-MFLs zu erhalten.

In Abb. 4.19 ist diese Situation dargestellt: das 3D-Objekt **O** definiert mit den zehn Lichtquellen $\mathbf{L_0}$, …, $\mathbf{L_9}$ zehn Lichtquellen-Shafts $\mathbf{S(O, L_j)}$, die BB-Hierarchie muss also zehnmal anhand dieser Shafts klassifiziert werden.

Auf den Lichtquellen-Shafts kann eine Hierarchie definiert werden: in Abb. 4.19 ist die konvexe Hülle alle Lichtquellen-Shafts durch dick gestrichelte Linien angedeutet, dies ist die Wurzel $\mathbf{S(O_i, \{L_0,\ldots,L_9\})}$ des Lichtquellen-Shaft-Baums. Die Kind-Shafts $\mathbf{S(O_i, \{L_0,\ldots,L_4\})}$ und $\mathbf{S(O_i, \{L_5,\ldots,L_9\})}$ des Wurzel-Shafts sind in Abb. 4.19 durch blau bzw. rot gestrichelte Linien dargestellt.

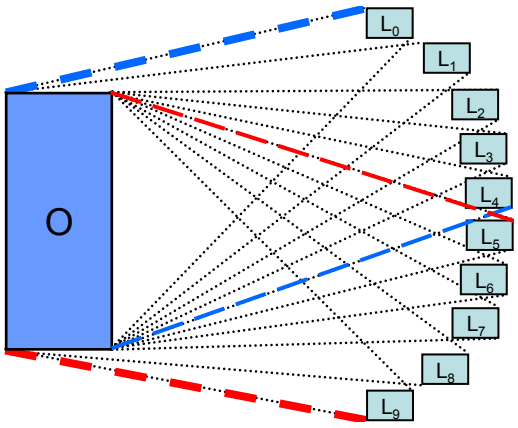


Abb. 4.19
Szene mit (Hierarchie von) Lichtquellen-Shafts

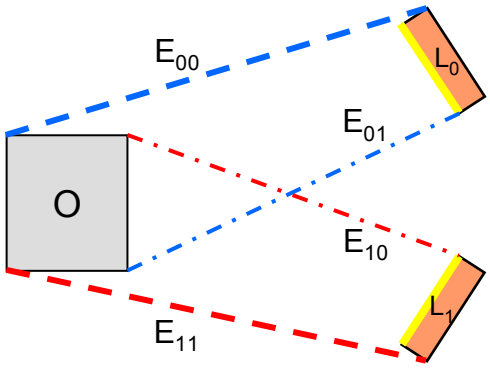


Abb. 4.20
Einfache Shaft-Hierarchie

Wird ein Teilbaum der BB-Hierarchie bezüglich des Shafts **$S(O_i, L_{0-9})$** als **OUT** (d.h. nicht „im“ Shaft“) klassifiziert, müssen die Kind-Shafts für die Lichtquellen **L_0**, …, **L_4** nicht mehr betrachtet werden.

Gesucht ist also ein Verfahren, das aus den einzelnen Lichtquellen-Shafts (bzw. deren Ebenen und Linien) eine Hierarchie von Shafts konstruiert. Dazu kann man Abb. 4.20 betrachten: das 2D-Objekt **O** definiert mit **L_0** und **L_1** zwei Shafts **$S(O, L_0)$** und **$S(O, L_1)$** mit Ebenen **E_{ij}** (zu lesen als: **j**-te Ebene des **i**-ten Shafts).

Für den Shaft **$S(O, \{L_0, L_1\})$** mit Shaft-Ebenen (im 2D: Linien) **$\{E_{00}, E_{11}\}$** gilt:

$S(O, L_0) \subset S(O, \{L_0, L_1\})$ und **$S(O, L_1) \subset S(O, \{L_0, L_1\})$**

Für die Ebenen des Shafts **$S(O, \{L_0, L_1\})$** gilt, dass alle enthaltenen Shafts (oder einfacher: Lichtquellen) im negativen Halbraum der Ebenen liegen müssen. Wie man leicht sieht, gilt dies für die Ebenen **E_{01}** und **E_{10}** nicht, hier liegen die Lichtquellen **L_0** und **L_1** jeweils auf verschiedenen Seiten der Ebenen bzw. schneiden diese.

Im 2D findet man für jedes Paar von Lichtquellen(-Shafts) immer genau zwei einschließende Ebenen. Im 3D ist dies i.a. nicht der Fall. In Abb. 4.21 ist ein solcher Problemfall im 3D dargestellt (der Übersichtlichkeit halber wird das 3D-Objekt **O** hier durch einen einzelnen Punkt repräsentiert): für keine Ebene des Shafts **$S(O, L_1)$** liegt **L_0** vollständig im negativen Halbraum.

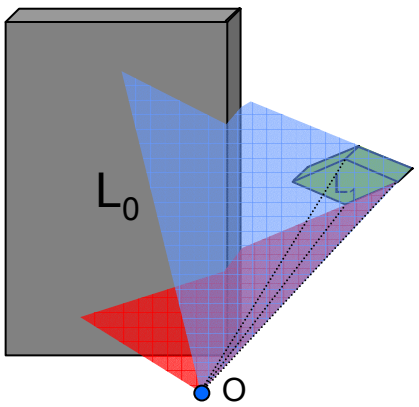


Abb. 4.21
Problemfall: Keine Shaft-Ebene von **L_1** kann übernommen werden

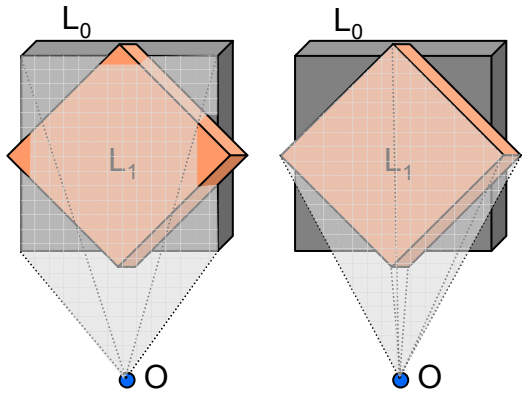


Abb. 4.22
Problemfall: Weder Shaft-Ebenen von **L_0** noch von **L_1** können übernommen werden

Der Shaft **S(O, { L_0, L_1 })** ist also nicht notwendig abgeschlossen. Im Extremfall kann es sogar vorkommen, dass für den Shaft **S(O, { L_0, L_1 })** gar keine Ebenen aus **S(O, L_0)** und **S(O, L_1)** übernommen werden können. Ein solcher Fall ist in Abb. 4.22 dargestellt. Die Behandlung dieser Sonderfälle wird erst am Ende dieses Kapitels diskutiert, bis dahin werden diese Fälle ignoriert.

Mit diesen Ausführungen ist klar, wie der Shaft für zwei Blätter der BB-Hierarchie konstruiert wird: seien L_0 und L_1 zwei Lichtquellen, S_0 und S_1 Listen der Shaft-Ebenen von **S(O, L_0)** und **S(O, L_1)** und S_{01} die (zunächst leere) Liste der Shaft-Ebenen von **S(O, {L_0, L_1})**. Dann gilt, dass jede Ebene $E \in S_0$ mit $L_1 \subset E^-$ von S_0 nach S_{01} verschoben werden kann, umgekehrt wird jede Ebene $E \in S_1$ mit $L_0 \subset E^-$ von S_1 nach S_{01} verschoben.

Darauf aufbauend kann man die Shaft-Linien von **S(O, L_0)** und **S(O, L_1)** behandeln: eine Shaft-Linie wird von S_0 bzw. S_1 in die Liste S_{01} verschoben, wenn mindestens eine adjazente Shaft-Ebene in die überliegende Liste verschoben wurde. Dabei kann man ausnutzen, dass die Liste der Shaft-Elemente eines (Lichtquellen-) Shafts immer abwechselnd aus Shaft-Ebenen und –Linien besteht (vgl. Kap. 3.4.4.1).

Dies ist in Abb. 4.23 und Abb. 4.24 für den 2D dargestellt: die Lichtquellen-Shafts **S(O_i, L_0)**, …, **S(O_i, L_3)** werden zunächst zu zwei Lichtquellen-Shafts **S(O_i, {L_0, L_1})** und **S(O_i, {L_2, L_3})** zusammengefasst, im nächsten Schritt dann zu einem Lichtquellen-Shaft **S(O_i, {L_0, …, L_3})**.

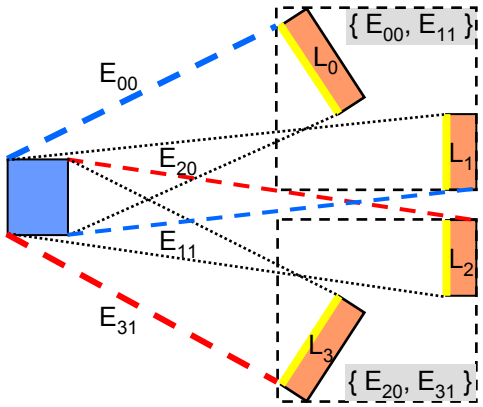


Abb. 4.23
Konstruktion der Listen S_{01} und S_{23}

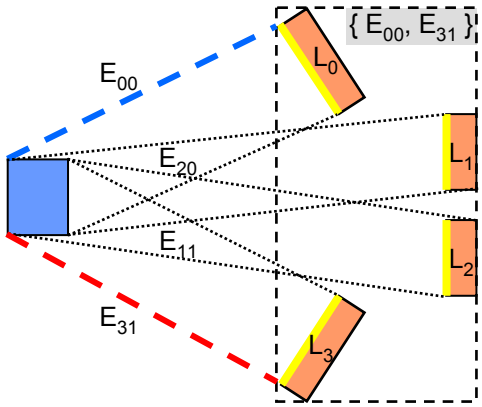


Abb. 4.24
Konstruktion der Liste S_{03}

Der folgende Algorithmus gibt zu Shafts **S(O, L_0)**, **S(O, L_1)** die Liste **S_{01}** zurück:

```
List hierShaft(Shaft[] s)
     List s01 = ∅
     for (int i = 0; i < 2; i++)
        j = (i+1) % 2 // Index des anderen Shafts, i=0 → j=1, i=1 → j=0
        for each shaft element e[k] in shaft s[i]
           if (e[k] is Plane)
              if (e[k].getState(s[j].getObject()) == NEG)
                move(s[i], e[k], s01)
                if (e[k-1] is Line) move(s[i], e[k-1], s01)
                if (e[k+1] is Line) move(s[i], e[k+1], s01)
     return s01
```

Code-Beispiel 4.1

Der Algorithmus prüft für alle Shaft-Ebenen beider 3D-Objekte, ob das jeweils andere 3D-Objekt im negativen Halbraum liegt. Ist dies der Fall, werden die beiden der Ebene adjazenten Shaft-Linien und die Ebene selber in die Liste **S_{01}** verschoben.

Dieser Algorithmus für die Blätter der BB-Hierarchie kann auf die inneren Knoten der Hierarchie erweitert werden. Im 2D reicht es aus, nur die (beiden) Lichtquellen zu testen, die die Shaft-Ebenen des jeweils anderen Unterbaums definieren (für Ebene **E_{00}** in Abb. 4.23 also die Lichtquellen **L_2** und **L_3**). Im 3D ist dies i.a. nicht der Fall. In Abb. 4.25 ist zu sehen, dass für die Ebene **E_{01}** zwar **L_5**, **$L_6 \subset E_{01}^-$** gilt, nicht aber **$L_4 \subset E_{01}^-$**. Um im 3D zu testen, ob eine Ebene aus der Liste **S_i** in die Liste **S_{ij}** ver-schoben werden kann, müssen alle 3D-Objekte im Unterbaum **B_j** getestet werden. Dabei kann die Hierarchie ausgenutzt werden, d.h. zunächst wird **B_j** gestestet, für Ergebnis **PAR** die Kind-Boxen von **B_j** usw.

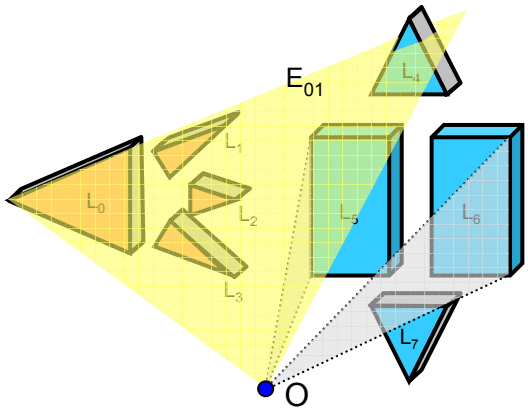


Abb. 4.25
Alle 3D-Objekte im Unterbaum müssen getestet werden

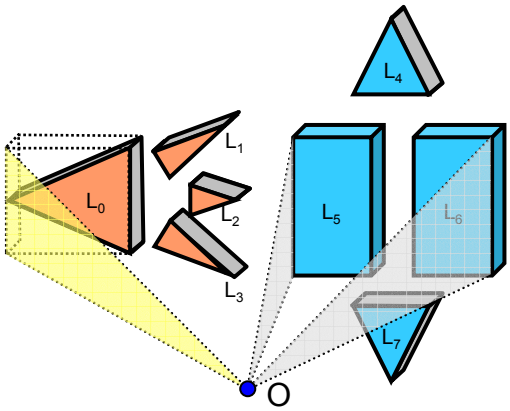


Abb. 4.26
Betrachtung der B-Box-Shafts löst einige Problemfälle

Abschließend müssen nun noch die Problemfälle geeignet behandelt werden: Kann aus einer Kind-Liste $\mathbf{S_i}$ keine Ebene in $\mathbf{S_{ij}}$ verschoben werden, kann man die Ebenen des Shafts $\mathbf{S(B_O, B_i)}$ der B-Box von **O** und der B-Box $\mathbf{B_i}$ betrachten und für diese testen, ob sie in $\mathbf{S_{ij}}$ verschoben werden können. In Abb. 4.26 sieht man, dass dadurch einige Problemfälle gelöst werden können.

In Abb. 4.27 ist jedoch zu erkennen, dass auch so Problemfälle bleiben. In diesem Fall wird die Liste $\mathbf{S_{ij}}$ mit den Ebenen der B-Box von **O** und $\mathbf{B_{ij}}$ belegt. Da diese B-Box immer existiert und für die Ebenen dieser B-Box die 3D-Objekte **O**, $\mathbf{L_i}$ und $\mathbf{L_j}$ im negativen Halbraum liegen, gibt es keine weiteren Problemfälle.

Das folgende Code-Beispiel zeigt den Test für eine Liste von Ebenen bzw. Linien und die zu klassifizierende B-Box. Die Methode **`rekState`** klassifiziert dabei eine B-Box (-Hierarchie) anhand einer Ebene, d.h. sie gibt **`POS`**/**`NEG`** zurück, wenn alle Kind-Boxen der B-Box im positiven bzw. negativen Halbraum der Ebene liegen, sonst **`PAR`** (vgl. Abb. 4.28).

```
List makeParentList(List shaftElemList, Box b)
     List parList = ∅
     for each shaft element e[k] in shaftElemList
       if (e[k] is Plane && rekState(e[k], b) == NEG)
         move(s[i], e[k], s01)
         if (e[k-1] is Line) move(e[k-1], shaftElemList, parList)
         if (e[k+1] is Line) move(e[k+1], shaftElemList, parList)
     return parList
```

Code-Beispiel 4.2

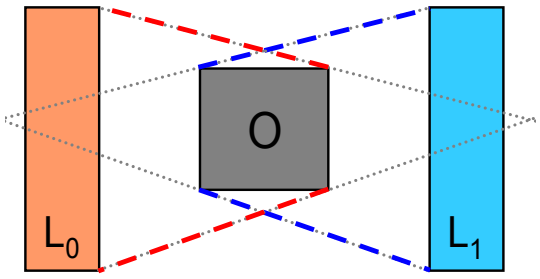


Abb. 4.27
Problemfall: Objekt **O** „zwischen“ $\mathbf{L_1}$ und $\mathbf{L_2}$

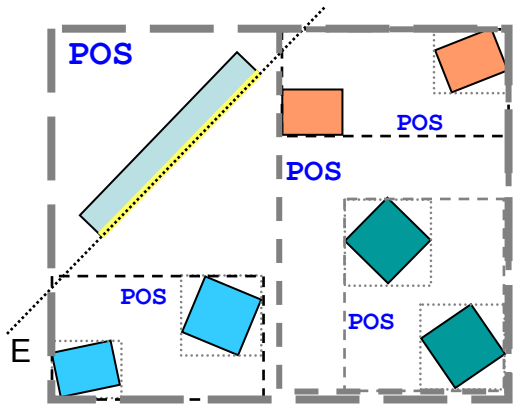


Abb. 4.28
`rekState` liefert den Status der Kind-Boxen

4.3.3.3 Hierarchische Shaft Klassifikation

Stand	Für ein festes 3D-Objekt $\mathbf{O_i}$ und alle $\mathbf{m}$ Lichtquellen $\mathbf{L_j}$ sind die Lichtquellen-Shafts $\mathbf{S(O_i, L_j)}$ zu hierarchischen Shafts zusammengefasst und in den Knoten der Lichtquellen-BB-Hiearchie gespeichert.
Teilziel	Die BB-Hierarchie der 3D-Szene soll anhand der hierarchischen Lichtquellen-Shafts klassifiziert werden, d.h. für jedes 3D-Objekt soll berechnet werden, „in" welchen Lichtquellen-Shafts es liegt.
Ansatz	Die Klassifikation einer BB-Hierarchie anhand eines einzelnen Shafts $\mathbf{S(O_i, L_j)}$ wird zur Klassifikation einer BB-Hierarchie anhand einer Hierarchie von Shaft erweitert.
Ausblick	Klassifikation von 3D-Objekten anhand der hierarchischen Lichtquellen-Shafts liefert Kandidaten für die Konstruktion der α- und β-MFLs.

Liegt für ein 3D-Objekt $\mathbf{O_i}$ die Shaft-Hierarchie mit dem Lichtquellen-Teilbaum komplett vor, müssen die anderen 3D-Objekte der Szene anhand dieser hierarchischen Shafts klassifiziert werden. Den bisher betrachteten Shaft-Klassifikationen war gemeinsam, dass eine BB-Hierarchie anhand eines einzelnen Shafts klassifiziert wurde. War das Ergebnis für eine B-Box $\mathbf{B}$ nicht eindeutig (d.h. **PAR**), wurde die Klassifikation rekursiv für die Kind-Boxen von $\mathbf{B}$ durchgeführt.

Hier muss nun eine Hierarchie von B-Boxen anhand einer Hierarchie von Shafts klassifiziert werden. Die entscheidende Frage ist, wann in welcher der beiden Hierarchien die nächste Hierarchiestufe geöffnet wird, d.h. ob eine Klassifikation für die Kind-Boxen einer B-Box oder für die Kind-Shafts eines Shafts wiederholt wird. Im Folgenden wird diese Notation verwendet:

- $\mathbf{B_{ij} \rightarrow (B_i, B_j)}$: Öffne B-Box $\mathbf{B_{ij}}$, d.h. setze Klassifikation mit Kind-Boxen von $\mathbf{B_{ij}}$ fort
- $\mathbf{S_{kl} \rightarrow (S_k, S_l)}$: Öffne Shaft $\mathbf{S_{kl}}$, d.h. setze Klassifikation mit Kind-Shafts von $\mathbf{S_{kl}}$ fort

Die folgende Tabelle zeigt, für welche Klassifikation welche Hierarchie warum geöffnet werden muss, zu jedem Fall ist noch eine Abbildung zur Veranschaulichung angegeben. Der Fall **PAR** ist dabei zunächst ausgespart, da er genauere Betrachtungen nötig macht:

Klassifikation von B_{ij} anhand S_{kl}	*Aktion*	*Begründung*
OCC (vgl. Abb. 4.29)	$B_{ij} \rightarrow (B_i, B_j)$	$V_{qual}(O, B_{ij}) = \text{OCC}$ $\Rightarrow V_{qual}(O, B_i) = \text{OCC} \wedge V_{qual}(O, B_j) = \text{OCC}$
INS (vgl. Abb. 4.30)	$S_{kl} \rightarrow (S_k, S_l)$	$B_{ij} \subset S_{kl} \Rightarrow B_i \subset S_{kl} \wedge B_j \subset S_{kl}$
OUT (vgl. Abb. 4.31)	*Abbruch Rekursionszweig*	$B_{ij} \cap S_{kl} = \varnothing \Rightarrow B_i \cap S_{kl} = B_j \cap S_{kl} = \varnothing$ $B_{ij} \cap S_{kl} = \varnothing \Rightarrow B_{ij} \cap S_k = B_{ij} \cap S_l = \varnothing$

Tabelle 4.4

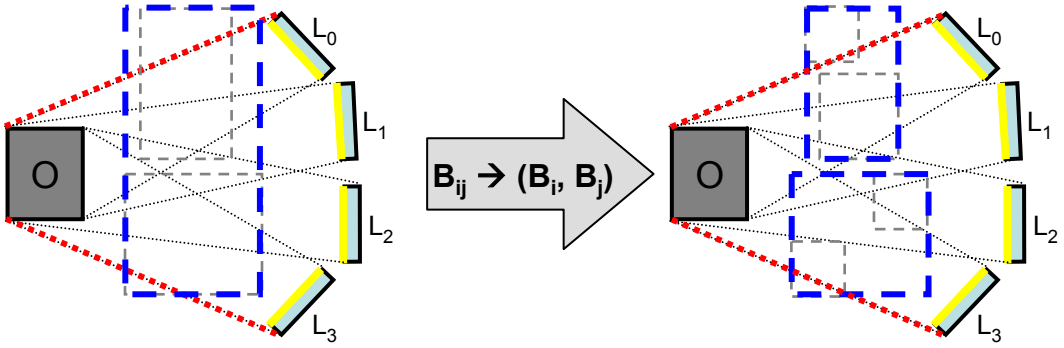


Abb. 4.29
Klassifikation **OCC**: $B_{ij} \rightarrow (B_i, B_j)$

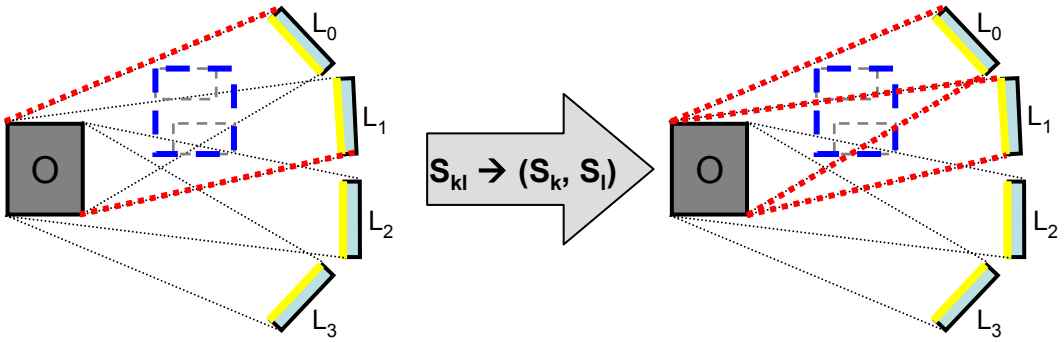


Abb. 4.30
Klassifikation **INS**: $S_{kl} \rightarrow (S_k, S_l)$

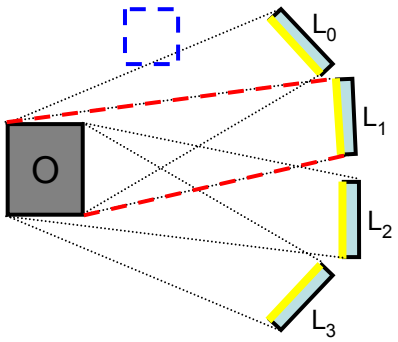


Abb. 4.31
Klassifikation **OUT**: Abbruch Rekursionszweig

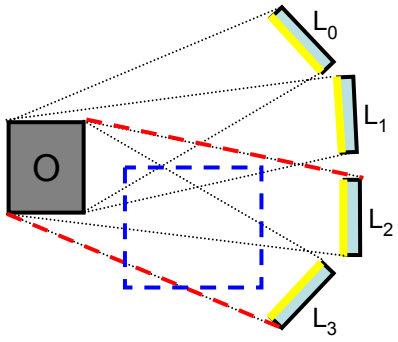


Abb. 4.32
Klassifikation **PAR**

Es bleibt also noch der Fall **PAR** (vgl. Abb. 4.32) zu betrachten. Die hier auftretenden Fälle müssen im 3D betrachtet werden, da im 2D die wesentlichen Eigenschaften verloren gehen. Für die folgenden Ausführungen bezeichnen S_k' bzw. S_l' die Teil-Shafts bzw. –Listen, die von S_k bzw. S_l in S_{kl} übernommen wurden.

Die folgende Tabelle gibt wieder für alle Klassifikationen der B-Box B_{ij} bezüglich S_k' und S_l' die auszuführende Aktion und eine Begründung hierfür an. Dabei bedeutet die Klassifikation PAR_L, dass mindestens eine Shaft-Linie von S_k' bzw. S_l' die B-Box B_{ij} schneidet. Der Fall **OCC** (d.h. alle Shaft-Linien in S_k' oder S_l' schneiden B_{ij}) wird hier nicht betrachtet, der Fall **OUT** für eine der bzw. **INS** für beide Teillisten kann, laut Tabelle 4.4, nicht vorkommen. Da weiterhin die Klassifikationsbedingungen symmetrisch sind, wird jeweils nur einer der Fälle betrachtet, d.h. z.B. in der ersten Zeile nicht **(INS, PAR_L)** und **(PAR_L, INS)**

Klassifikation von B_{ij} bezüglich…		*Aktion*	*Begründung*
S_k'	S_l'		
INS	PAR_L	$S_{kl} \rightarrow (S_k, S_l)$	Status von B_{ij} bzgl. S_k' ändert sich nicht mehr (vgl. Abb. 4.33)
INS	PAR		
PAR	PAR_L	$B_{ij} \rightarrow (B_i, B_j)$	Status von B_{ij} bzgl. S_k' und/oder S_l' ändert sich nicht mehr (vgl. Abb. 4.34)
PAR_L	PAR_L		
PAR	PAR	peek(B_{ij}, S_{kl})	*Begründung siehe unten!*

Tabelle 4.5

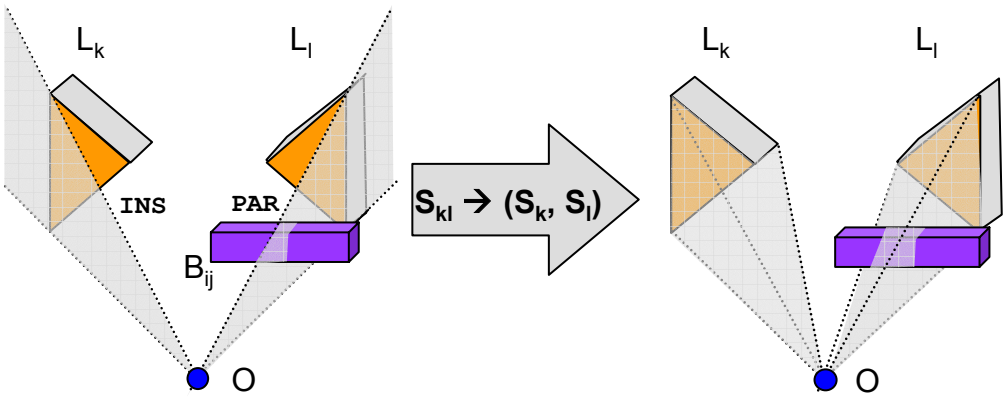


Abb. 4.33
Klassifikation **(INS, $PAR_{(L)}$)**: $S_{kl} \rightarrow (S_k, S_l)$

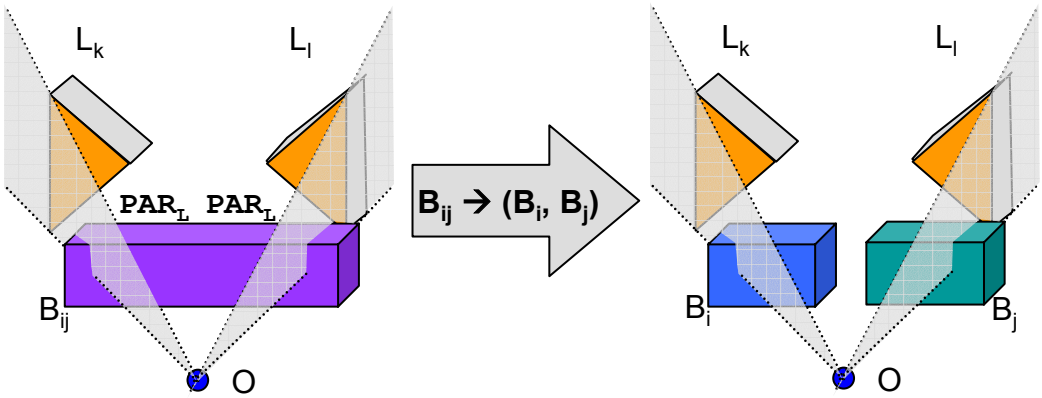


Abb. 4.34
Klassifikation **(PAR_L, PAR_L)**: **$B_{ij} \rightarrow (B_i, B_j)$**

Die Aktionen für die ersten vier Zeilen der Tabelle sind klar, es bleibt also noch der Fall **(PAR, PAR)** zu betrachten: Problematisch ist hier, dass man nicht unterscheiden kann, ob die beiden **PAR**-Klassifikationen richtig sind (in diesem Fall sollte **$B_{ij} \rightarrow (B_i, B_j)$** ausgeführt werden), oder ob es sich um eine konservative Fehl-Klassifikation handelt (dann sollte **$S_{kl} \rightarrow (S_k, S_l)$** ausgeführt werden).

Im Gegensatz zum einfachen Shaft-Culling kann dieses Problem auch nicht durch eine gemeinsame Betrachtung von Ebenen- und Linien-Shaft gelöst werden: die Idee bei dieser Vorgehensweise war, eine 3D-Entität bezüglich einer Shaft-Ebene (korrekt) als **PAR** zu klassifizieren, wenn die Ebene geschnitten wird und die 3D-Entität entweder „zwischen“ den der Ebene adjazenten Shaft-Linien liegt, oder von ihnen geschnitten wird. Die funktioniert jedoch nur bei geschlossenen Shafts, die hierarchischen Shafts bestehen dagegen aus Teillisten von Shaft-Ebenen und –Linien.

Eine Brute-Force-Lösung für dieses Problem wäre, einfach beide Aktionen (d.h. sowohl **$B_{ij} \rightarrow (B_i, B_j)$** als auch **$S_{kl} \rightarrow (S_k, S_l)$**) auszuführen. Dies würde jedoch für die folgenden Schritte des Algorithmus eine Verdopplung des Aufwandes bedeuten.

Eine einfache Lösung für dieses Problem ist, den Kind-Shaft nicht wirklich zu öffnen, sondern nur „einen Blick hinein zu werfen“, dies leistet die schon in Tabelle 4.5 genannte Methode **`peek`**.

Die folgende Tabelle listet die Aktionen bei verschiedenen Ergebnissen von **peek** auf:

peek(B_{ij}, S_{kl})		*Aktion*	*Begründung*
PAR	**PAR**	**B_{ij} → (B_i, B_j)**	**PAR** ist nicht notwendig richtiges, aber „sinnvolles“ Ergebnis
PAR	**PAR_L**		
PAR_L	**PAR_L**		
OUT	**PAR**	**S_{kl} → S_l**	**B_{ij}** kann nicht im Shaft **S_k** liegen
OUT	**PAR_L**		
OUT	**OUT**	*Abbruch des Rekursionszweigs*	**B_{ij}** liegt weder in **S_k** noch in **S_l**

Tabelle 4.6

Abschließend zeigt Abb. 4.35 ein einfaches 2D-Beispiel für die hierarchische Shaft-Klassifikation:

An der Wurzel des Rekursionsbaums wird die B-Box **B_{03}** anhand des Shafts **S_{02}** klassifiziert, das Ergebnis ist **OCC** (❶). Also wird die Aktion **B_{03} → (B_{01}, B_{23})** ausgeführt und der Rekursionsbaum spaltet sich in zwei Teile auf: Klassifikation von **B_{01}** anhand **S_{02}** ergibt **OUT** (❷), also wird der Rekursionszweig abgebrochen.

Damit wird die Rekursion im Zweig **B_{23}** weitergeführt, Klassifikation von **B_{23}** anhand **S_{02}** ergibt **PAR** (❸). Ein „Blick“ (**peek**) in die Kind-Shafts von **S_{02}** ergibt für **S_0 OUT** und für **S_{12}** wiederum **PAR** (❹), es wird also die Aktion **S_{02} → S_{12}** ausgeführt.

Ein erneuter **peek** für **B_{23}** und **S_{12}** liefert zweimal **PAR** (❺), es wird also **B_{23} → (B_2, B_3)** ausgeführt. **B_2** liegt nun komplett in **S_{12}** (❻), also wird **S_{12} → (S_1, S_2)** ausgeführt, erneute Klassifikation liefert für **S_1 INS** und für **S_2 PAR**. Der B-Box **B_2** können also Verweise auf die Lichtquellen **L_1** und **L_2** hinzugefügt werden, da **B_2** (teilweise) in den Shafts **$S(O, L_1)$** bzw. **$S(O, L_1)$** liegt.

Für **B_3** gilt, dass die Box teilweise in S_{12} (❼) und weiter außerhalb von **S_1** und teilweise in **S_2** liegt, also wird **B_3** ein Verweis auf die Lichtquelle **L_1** hinzugefügt. Damit ist die gesamte BB-Hierarchie anhand der gesamten Shaft-Hierarchie klassifiziert.

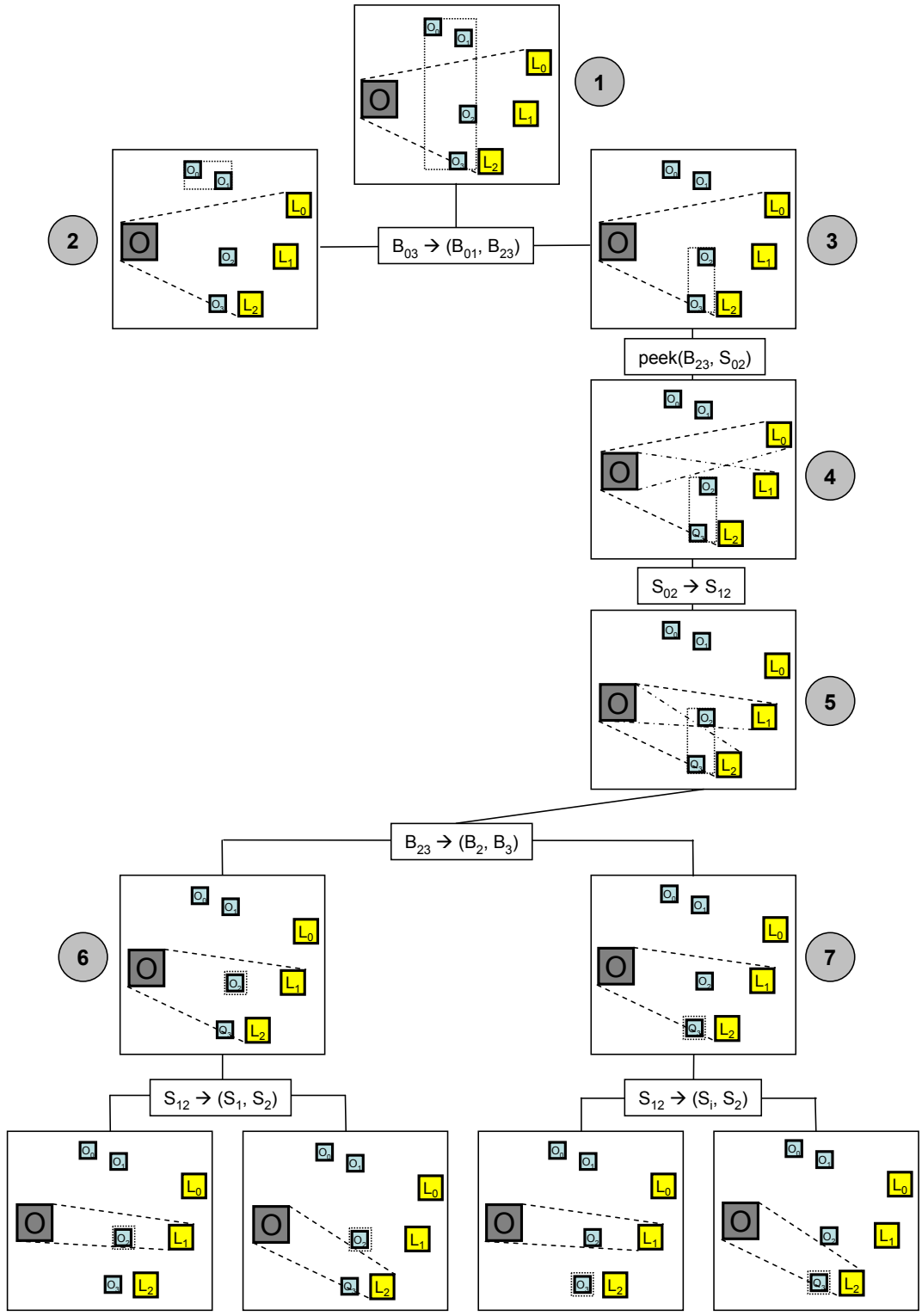


Abb. 4.35
Rekursionsbaum für die Methode **`hierCull`**

4.3.3.4 Zwischenspeicherung von Klassifikationen: Shaft Signaturen

Stand	Für ein festes 3D-Objekt $\mathbf{O_i}$ und alle $\mathbf{m}$ Lichtquellen $\mathbf{L_j}$ ist bekannt, welche 3D-Objekte in welchen Lichtquellen-Shafts $\mathbf{S(O_i, L_j)}$ liegen.
Teilziel	Für jede B-Box „in“ einem oder mehreren Lichtquellen-Shafts soll Information über diese Shafts gespeichert werden. Dadurch liegt bei folgenden Traversierungen der BB-Hierarchie für jede B-Box die Information vor, „in“ welchen Lichtquellen-Shafts sie liegt.
Ansatz	Es werden zwei Ansätze zur Speicherung von Shaft-Klassifikationen diskutiert: Bitfelder und Listen von Teilbäumen der Lichtquellen-BB-Hierarchie.
Ausblick	Mit den so gespeicherten Klassifikationen können bei der Konstruktion der α- und β-MFLs auf einfache Weise die Kandidaten-Objekte bestimmt werden.

Mit der hierarchischen Shaft-Klassifikation kann auf einfache Weise die Menge der 3D-Objekte, die „in“ mindestens einem Lichtquellen-Shaft liegen, bestimmt werden. Für jedes dieser 3D-Objekte muss die Menge der Lichtquellen-Shafts „in“ denen es liegt gespeichert werden. Diese Notwendigkeit kann man sich wie folgt klarmachen:

In Abb. 4.36 ist eine 2D-Szene mit zwei Lichtquellen dargestellt. Die 2D-Objekte $\mathbf{O_0}$,…, $\mathbf{O_4}$ liegen in beiden Lichtquellen-Shafts, die 2D-Objekte $\mathbf{O_5}$,…, $\mathbf{O_7}$ jeweils in einem Lichtquellen-Shaft. Dann gilt, dass jedes Paar $\mathbf{(O_i, O_j)}$ von 2D-Objekten nur dann mit $\mathbf{O}$ ein α- oder β-MFL bilden kann, wenn $\mathbf{O_i}$ und $\mathbf{O_j}$ (teilweise) im selben Lichtquellen-Shaft liegen. So kann es z.B. keine Linie geben, die $\mathbf{O}$, $\mathbf{O_5}$, $\mathbf{O_6}$ und $\mathbf{L_j}$ schneidet, da $\mathbf{O_5}$ und $\mathbf{O_6}$ nicht „im“ selben Lichtquellen-Shaft liegen (vgl. auch Abb. 4.37).

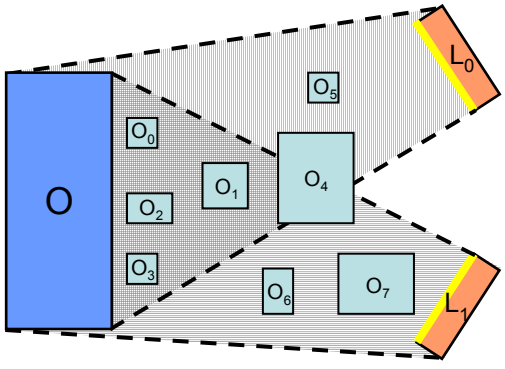


Abb. 4.36
2D-Objekte $\mathbf{O_i}$ liegen in verschiedenen Lichtquellen-Shafts

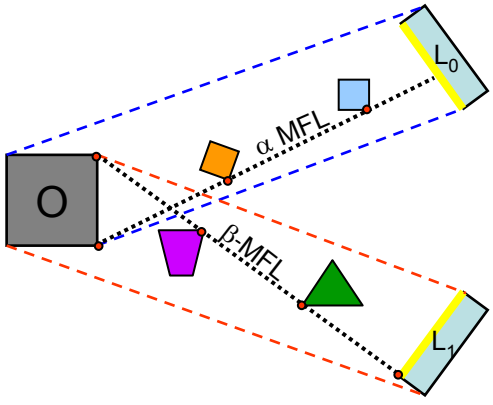


Abb. 4.37
Nur Objekte im selben Shaft können α- oder β-MFLs definieren

Bevor für ein Tupel von 3D-Objekten die Existenz einer α- oder β-MFL durch sie geprüft wird, kann also einfach verglichen werden, ob diese beiden 3D-Objekte (teilweise) im selben Lichtquellen-Shaft liegen.

Dies kann auf die BB-Hierarchie erweitert werden: für ein 3D-Objekt $\mathbf{O_k}$ müssen nur die B-Boxen traversiert werden, die „in" denselben Lichtquellen-Shafts liegen wie $\mathbf{O_k}$, denn nur diese B-Boxen können 3D-Objekte $\mathbf{O_l}$ enthalten, die „in" denselben Lichtquellen-Shafts wie $\mathbf{O_k}$ liegen. Wird noch ein drittes Generator-Objekt $\mathbf{O_m}$ benötigt, muss für dieses nur der Teil der BB-Hierarchie traversiert werden, der „in" denselben Lichtquellen-Shafts wie $\mathbf{O_k}$ und $\mathbf{O_l}$ liegt.

Die Menge der Klassifikationen eines 3D-Objekts bezüglich aller Lichtquellen-Shafts wird in dieser Arbeit als Shaft-Signatur bezeichnet. Bevor auf die Details der Speicherung und Verwaltung dieser Signaturen eingegangen wird, sollen zunächst einige wünschenswerte Eigenschaften diskutiert werden:

- ***Speicherplatz:*** Die Shaft-Signaturen sollten wenig Speicherplatz benötigen, da im Worst Case für jedes der $\mathbf{n}$ 3D-Objekte eine Shaft-Signatur mit Informationen über $\mathbf{m}$ Lichtquellen gespeichert werden muss, d.h. Speicheraufwand $\mathbf{O(m \cdot n)}$.
- ***Zugriff:*** Der Zugriff auf die Klassifikationen muss sowohl für einzelne Klassifiaktionen („Liegt $\mathbf{O_k}$ im Shaft $\mathbf{S(O_i, L_j)}$?"), als auch für alle Klassifikationen („In welchen Shafts $\mathbf{S(O_i, L_j)}$ liegt $\mathbf{O_k}$?") performant sein.
- ***Schnitt:*** Der Schnitt zweier Shaft-Signaturen sollte einfach zu bilden sein, da zwei 3D-Objekte $\mathbf{O_k}$ und $\mathbf{O_l}$ nur dann eine α- oder β-MFL definieren können, wenn es (mindestens) eine Lichtquelle $\mathbf{L_j}$ gibt, die in beiden Shaft-Signaturen vorkommt.
- ***Hierarchie:*** Es sollte einfach möglich sein, die Shaft-Signatur auf die BB-Hierarchie zu erweitern, d.h. für eine B-Box zu speichern, ob (mindestens) eine ihrer Kind-Boxen eine bestimmte Shaft-Signatur enthält.

Hier werden zwei Ansätze für Shaft-Signaturen vorgestellt, wobei nur der erste für eine kleine Zahl von Lichtquellen und der zweite allgemein verwendet werden kann.

1. Ansatz: Shaft-Signaturen als Bitfelder

Zunächst gilt, dass für ein 3D-Objekt O_k nur gespeichert werden muss, ob es in einem Lichtquellen-Shaft liegt oder nicht, eine Unterscheidung zwischen den Klassifikationen **INS** und **PAR** ist nicht nötig. Die Shaft-Klassifikationen können also für jeden Shaft durch ein einzelnes Bit repräsentiert werden, jedes 3D-Objekt muss dazu ein Bitfeld der Länge **m** (= Zahl der Lichtquellen) speichern. Im Folgenden bezeichnet M_k das Bitfeld des 3D-Objekts O_k, das **j**-te Bit $M_k(j)$ ist gesetzt, wenn O_k (teilweise) im Shaft $S(O_i, L_j)$ liegt. Die Eigenschaften dieser Art von Shaft-Signaturen sind:

- ***Speicherplatz:*** Zur Speicherung der Shaft-Signaturen sind $O(m \cdot n)$ Bits nötig. Werden Signaturen nur für die **s** 3D-Objekte gespeichert, die in mindestens einem Lichtquellen-Shaft liegen, reduziert sich der Speicher-Aufwand auf $O(n + s \cdot m)$.
- ***Zugriff:*** Die Abfrage, ob ein Bit in einem Bitfeld gesetzt ist, ist trivial. Eine Liste der gesetzten Bits kann mit Aufwand $O(m)$ extrahiert werden. Mit einigen Überlegungen kann dies auf $O(g \log g)$ (**g** = Anzahl gesetzter Bits) verbessert werden, dies soll aber hier nicht weiter ausgeführt werden.
- ***Schnitt:*** Der Schnitt zweier Bitfelder M_k und M_l entspricht einer logischen **UND**-Verknüpfung von M_k und M_l, die gesetzten Bits von $M_{kl} = M_k$ **AND** M_l können mit Aufwand $O(g \log g)$ extrahiert werden.
- ***Hierarchie:*** Jeder B-Box wird das Ergebnis der logischen **ODER**-Verknüpfung der beiden Kind-Bitfelder zugeordnet.

Damit liegt insgesamt ein einfaches und performantes Verfahren zur Speicherung von Shaft-Signaturen vor. Der größte Nachteil ist, dass das Verfahren ungeeignet für sehr große Zahlen von Lichtquellen ist: alle 3D-Objekte bzw. B-Boxen, die in mindestens einem Lichtquellen-Shaft liegen, müssen $O(m)$ Bits speichern.

2.Ansatz: Shaft-Signaturen als Listen von Teilbäumen

Sei M_k die Liste der Lichtquellen L_j, in deren Shafts $S(O_i, L_j)$ das 3D-Objekt O_k liegt. Da für die Lichtquellen bereits eine Hierarchie vorliegt, kann diese Liste auf einfache Weise in eine Liste von Teilbäumen überführt werden: sind zwei Elemente von M_k direkte Kind-Knoten desselben Vater-Knotens, werden sie aus M_k entfernt und der

Vaterknoten wird $\mathbf{M_k}$ hinzugefügt. Dies wird solange wiederholt, bis keine Elemente von $\mathbf{M_k}$ einen direkten gemeinsamen Vaterknoten haben (vgl. Abb. 4.38). Ist die Liste $\mathbf{M_k}$ mit $\mathbf{s}$ Elementen nach aufsteigenden Lichtquellen-IDs geordnet, so ist diese Zusammenfassung von Lichtquellen mit Aufwand $\mathbf{O(s \cdot lg\ s)}$ möglich: es müssen jeweils nur direkte Nachbarn miteinander verglichen werden und die Liste $\mathbf{M_i}$ kann maximal $\mathbf{O(lg\ s)}$ mal halbiert werden.

Jede B-Box muss also die Liste der Lichtquellen speichern, in deren Shaft sie liegt. So wird jedoch Speicherplatz verschwendet, da viele B-Boxen in denselben (Kombinationen von) Lichtquellen-Shafts liegen, d.h. dieselben Listen mehrfach gespeichert werden. Um dies zu vermeiden, werden die Listen in einem Pool gespeichert, die einzelnen B-Boxen enthalten dann einen Verweis in diesen Pool.

- ***Speicherplatz:*** Jede B-Box speichert einen Verweis in den Pool, d.h. Speicherplatz $\mathbf{O(n)}$. Der Pool kann theoretisch bis zu $\mathbf{2^m}$ (= Anzahl Teilmengen von $\mathbf{m}$ Lichtquellen) Listen enthalten, da aber jeder Eintrag im Pool von einer B-Box referenziert werden muss, können nur $\mathbf{O(n)}$ Einträge im Pool liegen. Eine Liste im Pool kann maximal $\mathbf{^m/_2}$ Einträge enthalten. Dieser Fall kann nur vorkommen, wenn die Blatt-Liste immer abwechselnd **0**en und **1**en enthält und somit keine zwei Kind-Knoten zu einem gemeinsamen Vaterknoten zusammengefasst werden können.
- ***Zugriff:*** Die Liste der Lichtquellen liegt bereits explizit vor.
- ***Schnitt:*** Der Schnitt zweier Listen der Längen $\mathbf{l}$ und $\mathbf{k}$ benötigt Aufwand $\mathbf{O(l + k)}$.
- ***Hierarchie:*** Jede B-Box speichert Verweis auf Vereinigung der Kind-Listen.

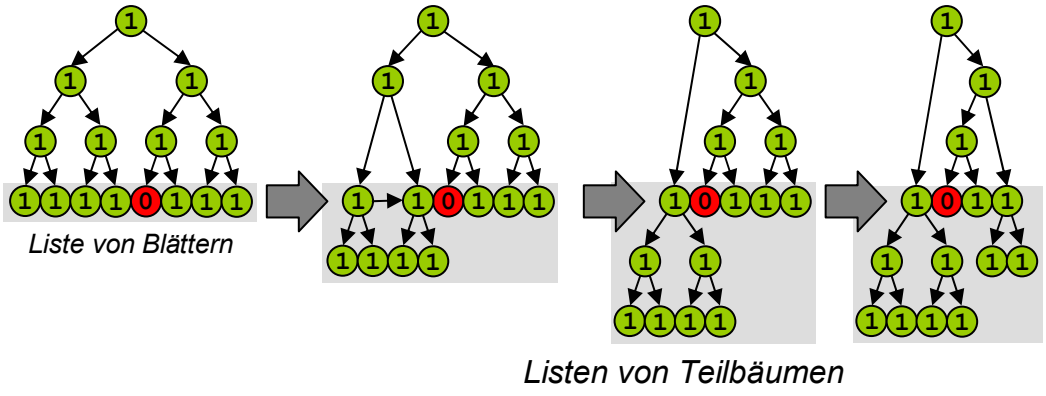


Abb. 4.38
Aus Liste von Blättern wird Liste von Teilbäumen erzeugt

4.3.4 Konstruktion der maximal freien Liniensegmente

Stand	Für ein festes 3D-Objekt $\mathbf{O_i}$ ist in der BB-Hierarchie gespeichert, in welchen Lichtquellen-Shafts die B-Boxen bzw. 3D-Objekte liegen.
Teilziel	Für Tupel von 3D-Objekten, die in denselben Lichtquellen-Shafts liegen, muss getestet werden, ob es ein α- oder β-MFL durch sie gibt.
Ansatz	Keine prinzipiellen Änderungen gegenüber der Konstruktion des einfachen Visibility Skeletons (vgl. Kap. 3.5.6.3 und 3.5.7.2). Zusätzlich müssen hier nur noch die lokalen Sichtbarkeitsinformationen und die Shaft-Signaturen ausgenutzt werden sowie alle betrachteten Generator-Shafts (**EV**, **FE***, **E***) auf die Lichtquellen beschränkt werden.
Ausblick	Mit den so berechneten α- und β-MFLs können die α- und β-Sichtbarkeitswechsel konstruiert werden, die die Sichtbarkeit zwischen den Lichtquellen und den 3D-Objekten der 3D-Szene beschreiben.

In diesem Teilkapitel wird die Konstruktion der α- und β-MFLs beschrieben. Die α-MFLs haben eine Lichtquelle als Extremal-Element und die β-MFLs haben eine Lichtquelle als Generator-Element.

Die Konstruktion der β-MFLs ist wesentlich einfacher als die der α-MFLs: ein β-MFL schneidet das feste 3D-Objekt $\mathbf{O_i}$, eine Lichtquelle $\mathbf{L_j}$ sowie ein oder zwei Generator-Objekte, die im Lichtquellen-Shaft $\mathbf{S(O_i, L_j)}$ liegen. Für ein Paar von Generator-Elementen aus $\mathbf{O_i}$ und $\mathbf{L_j}$ kann also ein verallgemeinerter Shaft (**EV**, **FE*** oder **E***, vgl. Kap. 3.5.7.2) gebildet und die BB-Hierarchie anhand dieses Shafts klassifiziert werden. Dabei müssen nur Teilbäume der BB-Hierarchie betrachtet werden, die (teilweise) „im“ Lichtquellen-Shaft $\mathbf{S(O_i, L_j)}$ liegen, d.h. deren Shaft-Signatur einen Verweis auf $\mathbf{L_j}$ enthält.

Für die Konstruktion der α-MFLs müssen hingegen alle Kombinationen von Elementen aus dem Generator-Objekt $\mathbf{O_i}$ und den 3D-Objekten $\mathbf{O_j}$, die in mindestens einem Lichtquellen-Shaft $\mathbf{S(O_i, L_j)}$ liegen, betrachtet werden. Jede solche Kombination definiert einen verallgemeinerten **EV**-, **FE***- oder **E***-Shaft. Liegt mindestens eine Lichtquelle „in“ diesem Shaft (d.h. der Shaft schneidet die Lichtquelle), so können „im“ Shaft α-MFLs (d.h. MFLs die eine Lichtquelle schneiden) verlaufen. Nur 3D-Objekte, die „in“ diesem Shaft liegen, können α-MFLs definieren.

Beiden MFL-Typen ist also gemeinsam, dass jeweils aus Elementen des 3D-Objekts $\mathbf{O_i}$ und anderen Elementen ein verallgemeinerter **EV**-, **FE***- oder **E***-Shaft gebildet wird und dann die BB-Hierarchie anhand dieses Shafts klassifiziert wird. Der wesentliche Unterschied ist, dass für die β-MFLs der Shaft aus einem Element von $\mathbf{O_i}$ und einem Element einer Lichtquelle $\mathbf{L_j}$ gebildet wird, während für die α-MFLs der Shaft aus einem Element von $\mathbf{O_i}$ und einem Element „in“ einem Lichtquellen-Shaft $\mathbf{S(O_i, L_j)}$ gebildet wird.

Dies bedeutet insbesondere, dass die Shafts für die β-MFLs jeweils eindeutig einer Lichtquelle zugeordnet werden können, während die Shafts für die α-MFLs zu mehreren Lichtquellen gehören können. Dies ist in Abb. 4.39 dargestellt: der **EV**-Shaft schneidet zwei Lichtquellen $\mathbf{L_0}$ und $\mathbf{L_1}$, α-MFLs „in“ diesem Shaft können entweder $\mathbf{L_0}$ oder $\mathbf{L_1}$ schneiden.

In den folgenden Teilkapiteln wird für jeden MFL-Typ die Konstruktion der α- und β-MFLs vorgestellt. Dabei werden im Wesentlichen die Ideen und Vorschläge aus Kap. 3.5.7 aufgegriffen: EV- und FE*-Shafts werden in Kind-Shafts aufgespalten und E*-Shafts werden schrittweise beschränkt. Daher werden in den folgenden Teilkapiteln hauptsächlich die Konstruktionen der verallgemeinerten Shafts diskutiert. Für die α-Shafts (d.h. die Shafts, die α-MFLs enthalten können) wird diese Idee zur Beschränkung von Shafts auf Lichtquellen erweitert.

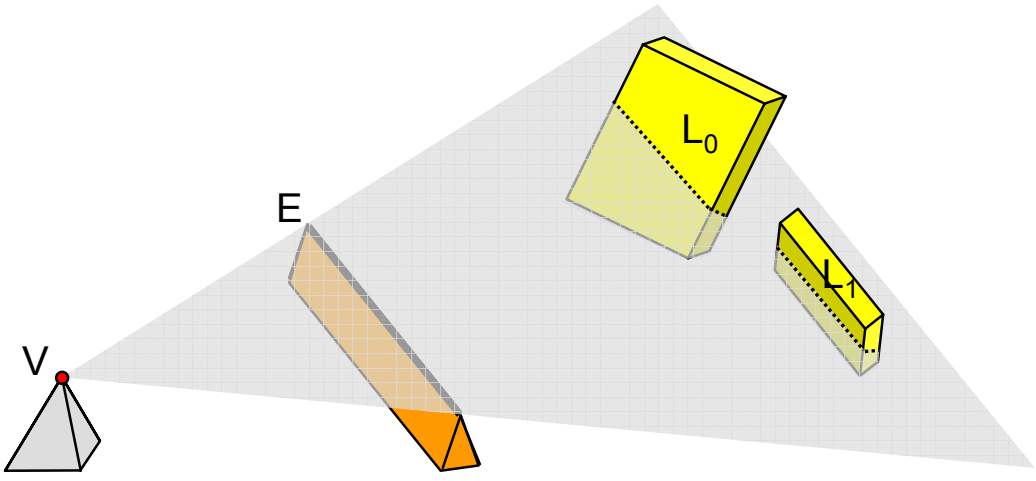


Abb. 4.39
Shaft schneidet zwei Lichtquellen

4.3.4.1 Vertex-MFLs

In diesem Teilkapitel wird die Konstruktion der α- und β-Vertex-MFLs diskutiert. Dabei gilt, dass für die Konstruktion der α- und β-**VV**-MFLs keine weiteren Überlegungen notwendig sind, alle Ergebnisse aus Kap. 3.5.7.2 können ohne Änderungen übertragen werden. Alle weiteren Ausführungen beziehen sich daher nur auf die α- und β-**EV**-Shafts.

Zunächst fällt ein Unterschied zwischen „normalen“ und α-**EV**-Shafts auf. Dazu kann man Abb. 4.40 betrachten: hier ist ein „normaler“ **EV**-Shaft dargestellt, ein 3D-Objekt $\mathbf{O_j}$ schneidet diesen Shaft und unterteilt ihn somit in drei Kind-Shafts $\mathbf{EV_0}$, $\mathbf{EV_1}$ und $\mathbf{EV_2}$.

Abb. 4.41 zeigt dieselbe Situation für einen α-**EV**-Shaft. Hier gilt, dass der Shaft nur in zwei Kind-Shafts unterteilt wird, da der „mittlere“ Kind-Shaft keine MFLs enthalten kann, die die Lichtquelle **L** schneiden, d.h. α-MFLs sind.

Für die α-Shafts gilt, dass sie aus einem Element aus $\mathbf{O_i}$ und einem Element aus einem Lichtquellen-Shaft $\mathbf{S(O_i, L_j)}$ aufgespannt werden. Damit ergibt sich das Problem, dass diese Shafts zunächst nicht notwendig auf eine oder mehrere Lichtquellen beschränkt sind. Dies ist in Abb. 4.42 und Abb. 4.43 dargestellt:

Abb. 4.42 zeigt oben einen α-**EV**-Shaft, der eine Lichtquelle **L** schneidet. Dieser Shaft umfasst auch MFLs, die „links“ oder „rechts“ an der Lichtquelle vorbeigehen. Zunächst muss der Shaft also auf die Lichtquelle beschränkt werden, dies ist im unteren Teil von Abb. 4.42 dargestellt.

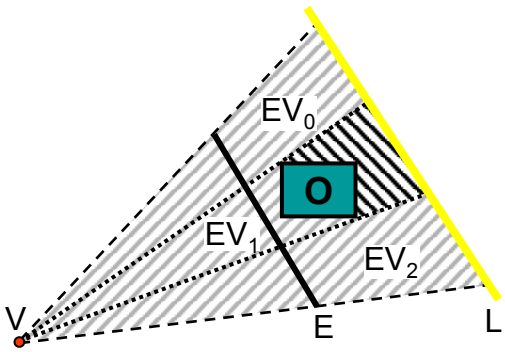


Abb. 4.40
3D-Objekt **O** zerlegt **EV**-Shaft in drei Kind-Shafts

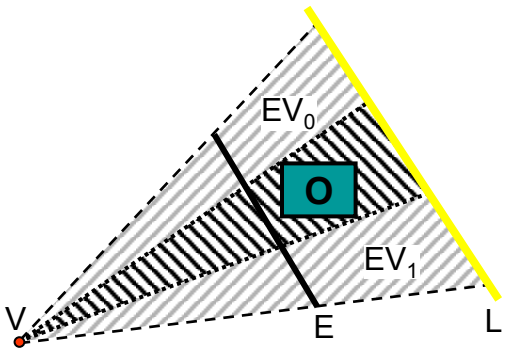
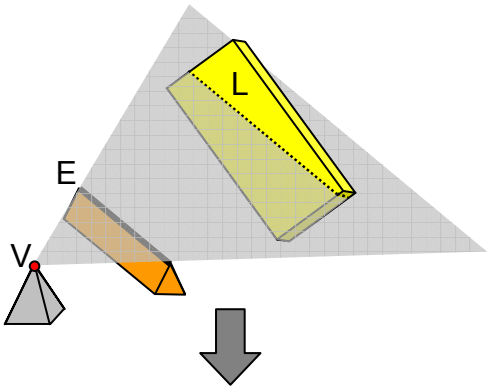


Abb. 4.41
3D-Objekt **O** zerlegt α-**EV**-Shaft in zwei Kind-Shafts



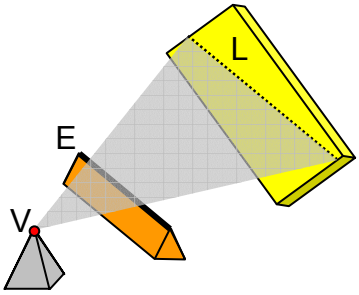
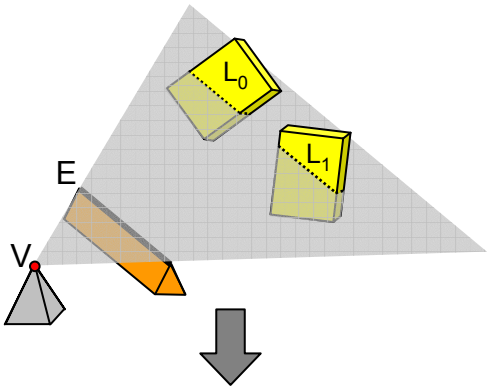


Abb. 4.42
Anpassung der Seitenebenen



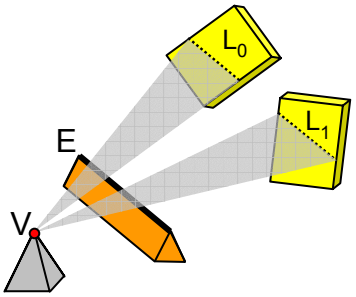


Abb. 4.43
Aufspalten in Kind-Shafts

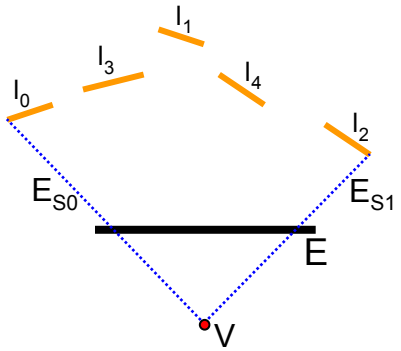


Abb. 4.44
Liste der Liniensegmente im **EV**-Shaft

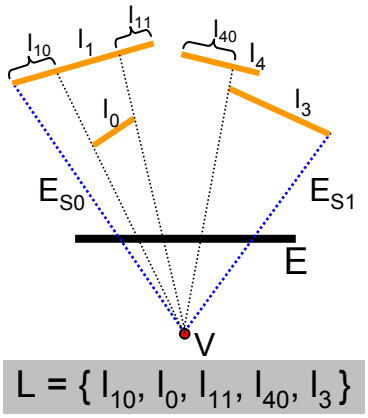


Abb. 4.45
Sonderfallbehandlung bei überlappenden Liniensegmenten

In Abb. 4.43 ist eine ähnliche Situation für zwei Lichtquellen $\mathbf{L_0}$ und $\mathbf{L_1}$ dargestellt. Beschränkung des **α-EV**-Shafts auf die Lichtquellen spaltet den Shaft in zwei Kind-Shafts auf.

Die Beschränkung eines **EV**-Shafts auf eine einzelne Lichtquelle ist trivial, interessanter ist die Aufspaltung in Kind-Shafts: Gegeben sei ein **EV**-Shaft **S** sowie eine Liste **L** von **n** Liniensegmenten $\mathbf{l_i}$, die den Schnitten der Lichtquellen mit dem **EV**-Shaft entsprechen (vgl. Abb. 4.44). Die Liste **L** sei von „links“ nach „rechts“ aufsteigend sortiert.

Im Sonderfall zweier überlappender Liniensegmente $\mathbf{l_i}$ und $\mathbf{l_j}$ (vgl. Abb. 4.45) wird das „vordere“ Liniensegment $\mathbf{l_i}$ unverändert übernommen und das „hintere“ Liniensegment $\mathbf{l_j}$ auf das von **v** sichtbare Teilsegment $\mathbf{l_j}$ zugeschnitten.

Ziel ist nun, die Liste **L** in zwei Kind-Listen $\mathbf{L_0 = (l_0, \ldots, l_{m-1})}$ und $\mathbf{L_1 = (l_m, \ldots, l_{n-1})}$ aufzuspalten und auf diesen Kind-Listen zwei Kind-Shafts $\mathbf{S_0}$ und $\mathbf{S_1}$ zu definieren. Damit ergibt sich die Frage, nach welchem Kriterium die beiden Kind-Listen aufgespalten werden sollten. Dazu werden zwei mögliche Kriterien vorgestellt:

1. ***Halbierung der Liste:*** Liste **L** wird in zwei Hälften aufgespalten, d.h. $\mathbf{m = \lfloor n/2 \rfloor}$. In Abb. 4.46 ist zu sehen, dass bei Lichtquellen(-Schnitten) stark unterschiedlicher Größe dadurch eine sehr ungleichmäßige Aufteilung entstehen kann.
2. ***Halbierung der „Shaft-Breite“:*** Es wird das Liniensegment $\mathbf{l_i}$ gesucht, dass den kleinsten Abstand zur „Mittel-Linie“ $\mathbf{M = \tfrac{1}{2}(l_{n-1}^{\,1} - l_0^{\,0}) - v}$ hat (vgl. Abb. 4.47). Sei $\mathbf{l_i'}$ das Teilsegment von $\mathbf{l_i}$, dass „links“ von **M** liegt und $\mathbf{l_i''}$ das Teilsegment von $\mathbf{l_i}$, dass vollständig rechts von **M** liegt. Dann gilt

$$\mathbf{m} = \begin{cases} \mathbf{i+1} & \|\mathbf{l_i'}\| < \|\mathbf{l_i''}\| \\ \mathbf{i} & \text{sonst} \end{cases}$$

Dieses Verfahren wird rekursiv für die Kind-Listen fortgesetzt, dadurch entsteht eine Hierarchie von **EV**-Shafts. Diese Hierarchie kann völlig analog zu der in Kap. 3.5.7.3 diskutierten behandelt werden: 3D-Entitäten werden mittels der hierarchischen Shaft-Klassifikation anhand der Hierarchie von **EV**-Shafts klassifiziert. Jedes 3D-Objekt „in“ einem Blatt-Shaft der Hierarchie spaltet den Blatt-Shaft in neue Kind-Shafts auf.

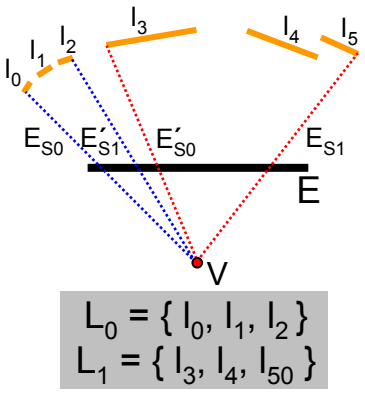


Abb. 4.46
Aufspaltung durch Halbierung der Liste

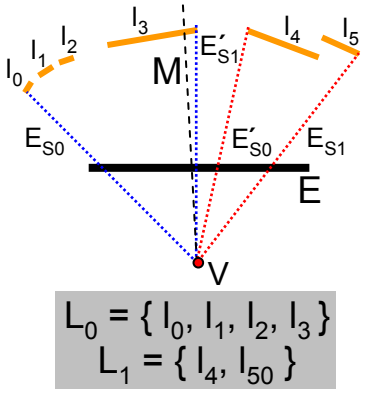


Abb. 4.47
Aufspaltung durch Halbierung der „Shaft-Breite“

4.3.4.2 Facetten-MFLs

Für die Konstruktion der α- und β-Facetten-MFLs müssen einige Fälle unterschieden werden. Die Überlegungen dazu beginnen wieder mit den einfacheren β-Facetten-MFLs: ein β-Facetten-MFL hat eine Lichtquelle als Generator-Element. Dies ergibt zwei Unterfälle:

(1) Eine Kante der Lichtquelle ist Generator-Element. Dieser Fall wird bei der Konstruktion der α-MFLs behandelt.

(2) Eine Lichtquellen-Facette ist Generator-Element.

Im Folgenden wird zunächst Fall (2) behandelt: Wenn eine Lichtquelle Generator-Element eines β-Facetten-MFL ist, so kann man zunächst völlig analog zur Konstruktion der Facetten-MFLs nach Kap. 3.5.7.4 vorgehen. Bei genauerer Betrachtung ergibt sich noch eine einfache Optimierungsmöglichkeit:

Ist die Facette selber eine Lichtquelle, so müssen die **EFE**-MFLs nicht betrachtet werden. Dies kann man sich an Abb. 4.48 veranschaulichen: Die dort abgebildete MFL beschreibt die Sichtbarkeit zwischen den Extremal-Elementen $\mathbf{F_0}$ und $\mathbf{F_1}$, jedoch nicht die Sichtbarkeit zwischen der Lichtquelle und den Extremal- oder Generator-Elementen. Dies entspricht auch genau den Überlegungen aus Tabelle 4.1 - Tabelle 4.3: Die Lichtquellen-Facette ist kein „äußeres“ Generator-Element, kann also kein β-**EFE**-MFL definieren.

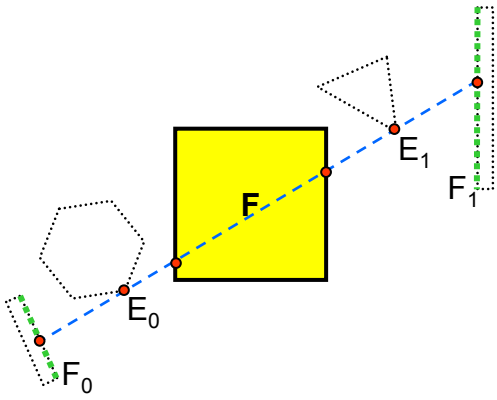


Abb. 4.48
Lichtquellen-Facette ist Generator-Element eines **EFE**-MFL

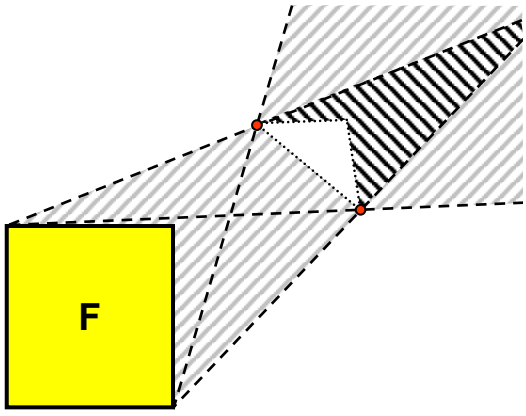


Abb. 4.49
Shaft „auf einer Seite“ der Facette

Also müssen in diesem Fall für die β-Facetten-MFLs nur die MFL-Typen **FvE**, **FEE** und **FF** betrachtet werden, die Konstruktion der β-**Fvv**-MFLs ist ohnehin trivial. Die Idee der Quadtree-Optimierung aus Kap. 3.5.7.4 kann direkt übernommen werden, wobei lediglich die Shaft-Bedingungen so angepasst werden müssen, dass nur Objekte „auf derselben Seite“ der Lichtquellen-Facette betrachtet werden (vgl. Abb. 4.49).

Die Konstruktion der α-Facetten-Shafts verläuft ebenfalls analog zu den bisherigen Überlegungen und wird zunächst wieder für eine einzelne Lichtquelle betrachtet: Gegeben sei eine (nicht-Lichtquellen-)Facette **F**, deren Ebene eine Lichtquelle $\mathbf{L_j}$ schneidet. Es muss ein Shaft definiert werden, der alle Objekte umschließt, die mit **F** und $\mathbf{L_j}$ eine α-Facetten-MFL bilden können.

Diese Bedingung entspricht genau der Facetten-Shaft-Bedingung aus Kap. 3.5.7.4, Abb. 3.129, d.h. alle Aussagen zur Konstruktion und Klassifikation anhand dieses Shafts können für eine einzelne Lichtquelle $\mathbf{L_j}$ unverändert übernommen werden.

Um das Verfahren auf mehrere Lichtquellen zu erweitern, können wiederum Ideen aus Kap. 4.3.3.2 bzw. 4.3.3.3 übernommen werden: mehrere Lichtquellen definieren mehrere Shafts, diese Shafts können mit dem bekannten Verfahren zu hierarchischen Shafts zusammengefasst werden (vgl. Abb. 4.50). Auch die Klassifikation anhand der hierarchischen Shafts entspricht den bisherigen Überlegungen. Es bleibt allerdings ein Unterschied zu den bisherigen Überlegungen zu betrachten: die Facette **F** gehört zum 3D-Objekt $\mathbf{O_i}$, für dieses wurden mittels der hierarchischen Shaft-Klassifikation alle 3D-Objekte $\mathbf{O_k}$ gefunden, die „zwischen“ $\mathbf{O_i}$ und den Lichtquellen $\mathbf{L_j}$ liegen.

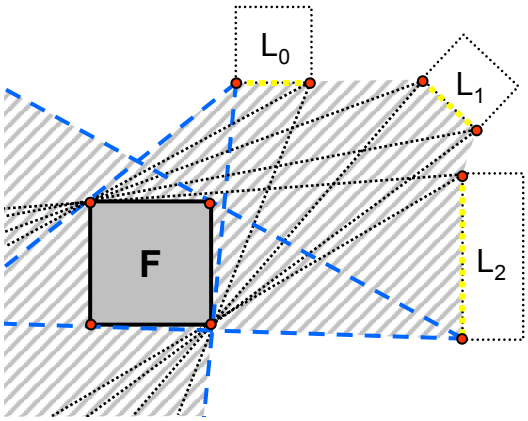


Abb. 4.50
Hierarchische Facetten-Shafts

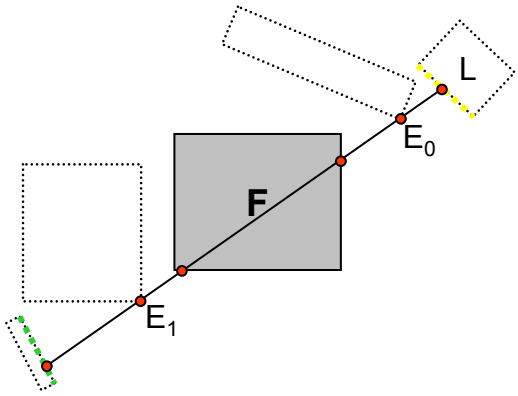


Abb. 4.51
Generator-Kanten für α-**EFE**-MFLs liegen auf verschiedenen Seiten der Facette

Anschaulich bedeutet dies, dass nur 3D-Objekte $\mathbf{O_k}$ betrachtet werden, die „zwischen“ der Facette **F** und den Lichtquellen $\mathbf{L_j}$ liegen. Für die α-**EFE**-MFLs müssen aber auch 3D-Objekte $\mathbf{O_k}$ betrachtet werden, die auf verschiedenen Seiten von **F** liegen (vgl. Abb. 4.51).

Daher muss man für die α-**EFE**-MFLs von der bisherigen Traversierung der BB-Hierarchie abweichen: bei den anderen MFL-Typen wurde nur der Teil der BB-Hierarchie traversiert, der eine nicht-leere Shaft-Signatur hat, d.h. der in mindestens einem Lichtquellen-Shaft $\mathbf{S(O, L_j)}$ liegt. Für die α-**EFE**-MFLs muss zusätzlich auch der Teil der BB-Hierarchie traversiert werden, der in mindestens einem **FE***-Shaft der Facetten von **O** und den Lichtquellen $\mathbf{L_j}$ liegt.

4.3.4.3 Kanten-MFLs

Für die Konstruktion der β-**E4**-MFLs sind keine weiteren Überlegungen notwendig, alle Ergebnisse aus Kap. 3.5.7.5 können direkt übertragen werden. Für die Konstruktion der α-MFLs müssen die **EE**-Shafts auf die geschnittenen Licht-quellen beschränkt werden, d.h. aus dem **EE**-Shaft muss ein **EEO**-Shaft (Kante-Kante-3D-Objekt) konstruiert werden. Dies verläuft ähnlich wie die Konstruktion der **EEE**-Shafts und wird hier nur für eine einzelne Lichtquelle $\mathbf{L_j}$ diskutiert:

Zunächst wird die Lichtquelle $\mathbf{L_j}$ anhand des **EE**-Shaft klassifiziert. Dabei gilt, dass $\mathbf{L_j}$ entweder außerhalb des **EE**-Shafts liegt oder einen der „äußeren“ Keile schneidet (vgl. Abb. 4.52). Würde die Lichtquelle den „mittleren“ Teil des **EE**-Shafts schneiden, so könnte sie weder Extremal- noch „äußeres“ Generator-Element eines **E4**-MFL sein.

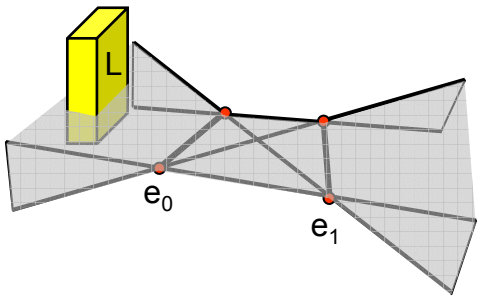


Abb. 4.52
$\mathbf{L_j}$ muss im „äußeren“ Keil des **EE**-Shafts liegen

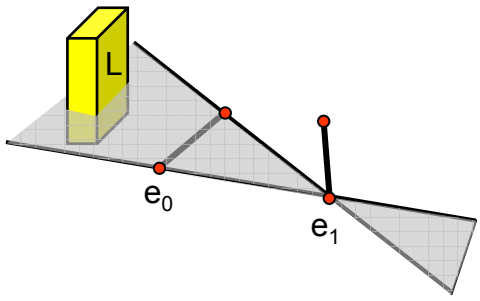


Abb. 4.53
Ebene $\mathbf{v_0e_1}$ kann übernommen werden

Dann müssen die Ebenen des **EE**-Shafts gefunden werden, die in den **EEO**-Shaft übernommen werden können. Dies sind diejenigen Ebenen, die von Kanten von $\mathbf{L_j}$ geschnitten werden (vgl. Abb. 4.53). Dies ist analog zur **EV**-Shaft-Bedingung, bei der die Seiten des **EV**-Shafts übernommen wurden, die die Lichtquelle schneiden.

Um den **EE**-Shaft auf die Lichtquelle zu beschränken, muss man ähnlich wie bei der Anpassung der **EV**-Shafts vorgehen: Lag dort eine Lichtquelle (d.h. ein Liniensegment $\mathbf{l_i}$) vollständig im **EV**-Shaft, wurden die Seitenebene des **EV**-Shafts so angepasst, dass sie durch die Endpunkte des Liniensegment $\mathbf{l_i}$ gehen.

Für die **EE**-Shafts müssen analog zwei Typen von Ebenen betrachtet werden:

(1) Ebenen $\mathbf{E(v_i, e_j)}$, die durch einen Vertex $\mathbf{v_i}$ der Lichtquelle und eine der beiden Kanten $\mathbf{e_j}$ des **EE**-Shafts definiert werden, wobei $\mathbf{v_i}$ in einem „äußeren“ Keil des **EE**-Shafts liegt (vgl. Abb. 4.54).

(2) Ebenen $\mathbf{E(v_i, e_j)}$, die durch eine Kante $\mathbf{e_j}$ der Lichtquelle und einen der vier Vertices $\mathbf{v_i}$ des **EE**-Shafts definiert werden, wobei $\mathbf{e_j}$ in einem „äußeren“ Keil des **EE**-Shafts liegt (vgl. Abb. 4.55).

Jede dieser Ebenen, für die die Lichtquelle vollständig im negativen Halbraum liegt, wird in den neuen **EEO**-Shaft übernommen. Abschließend müssen noch die beiden Kanten $\mathbf{k_0}$ und $\mathbf{k_1}$ des **EE**-Shafts auf den neuen **EEO**-Shaft zugeschnitten werden.

Die Anpassung der **EEEO**-Shafts verläuft analog und wird hier nicht diskutiert.

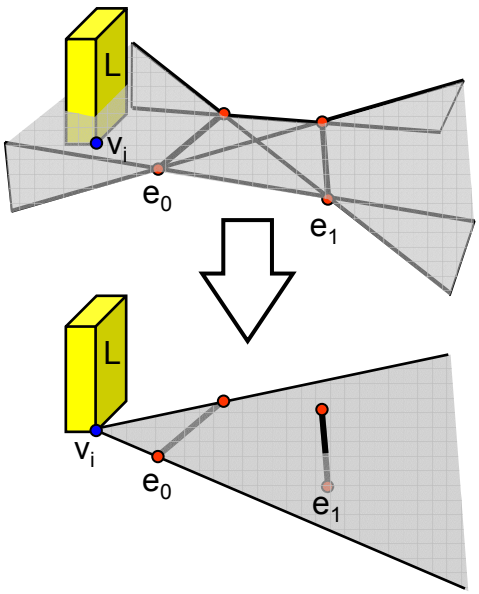


Abb. 4.54
Ebene durch Lichtquellen-Vertex und Shaft-Kante

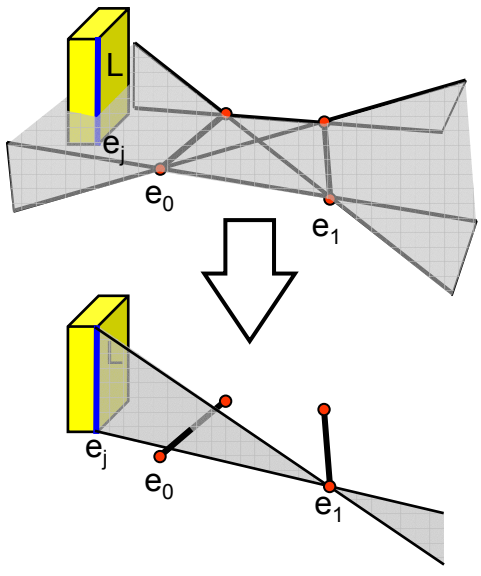


Abb. 4.55
Ebene durch Lichtquellen-Kante und Shaft-Vertex

4.3.5 Konstruktion der Sichtbarkeitswechsel

Für die Konstruktion der α- und β-Sichtbarkeitswechsel sind kaum zusätzliche Überlegungen notwendig: Mit den Ergebnissen des letzten Teilkapitels sind alle α- und β-MFLs berechnet und für jede α- oder β-MFL gilt, dass mindestens eines ihrer Extremal-Elemente eine Lichtquelle ist. Analog zu Kap. 3.5.6.4 werden diese MFLs zunächst in Master-Sichtbarkeitswechsel eingefügt, die nach Abschluss der MFL-Konstruktion in einzelne Sichtbarkeitswechsel aufgeteilt werden.

Für diese Aufteilung wurden aufeinander folgende Paare von MFLs daraufhin getestet, ob sie ein gemeinsames Extremal-Element definieren. Dieser Test kann nun auf α- und β-Sichtbarkeitswechsel erweitert werden: Für ein Paar von α- oder β-MFLs wird ein Sichtbarkeitswechsel nach dem Verfahren aus Kap. 3.5.6.4 berechnet. Ergibt dies einen gültigen Sichtbarkeitswechsel muss nur noch getestet werden, ob eins der beiden Extremal-Elemente des Sichtbarkeitswechsels eine Lichtquelle ist. Ist dies der Fall, liegt ein α- oder β-Sichtbarkeitswechsel vor, sonst kann das Paar von MFLs verworfen werden.

Für den in Abb. 4.56 dargestellten **EV**-Master-Sichtbarkeitswechsel gilt, dass…

- …die MFLs **A** und **B** sowie **C** und **D** gültige α-Sichtbarkeitswechsel definieren.
- …die MFLs **B** und **C** zwar einen gültigen Sichtbarkeitswechsel, aber keinen gültigen α- oder β-Sichtbarkeitswechsel definieren, da das Extremal-Element **O** keine Lichtquelle ist.

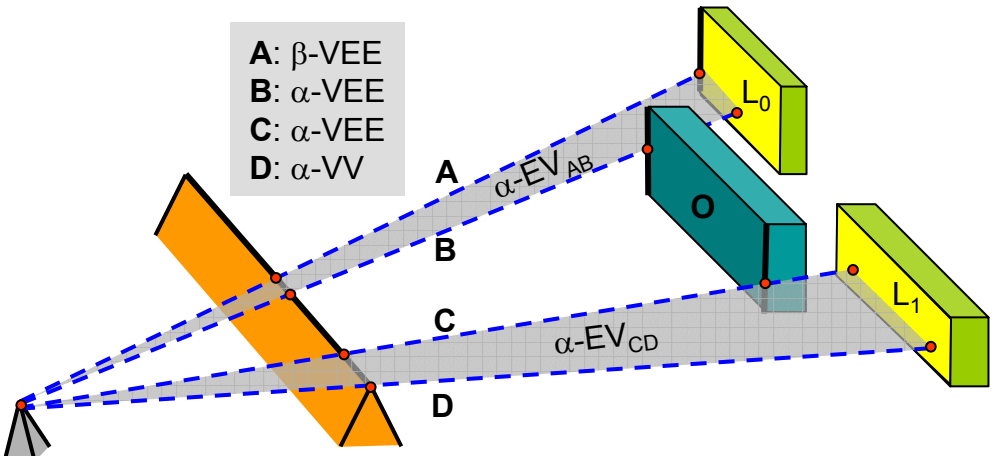


Abb. 4.56
MFLs im Master-Sichtbarkeitswechsel bilden gültige und ungültige α-/β-Sichtbarkeitswechsel

4.4 Experimente und Messungen

In diesem Teilkapitel soll die zuvor erarbeitete Konstruktion des $\mathcal{DIVS}$ anhand von Experimenten und Messungen mit konkreten 3D-Szenen untersucht und bewertet werden. Dazu werden im Folgenden acht 3D-Szenen mit jeweils 25-500 3D-Objekten und 4-16 Lichtquellen betrachtet:

- 3D-Szenen ***S04a*** – ***S16a***: alle Lichtquellen an der Decke des Raums
- 3D-Szenen ***S04b*** – ***S16b***: eine Hälfte der Lichtquellen an der Decke, eine Hälfte zufällig im Raum platziert

Zunächst wird die Zahl der $\mathcal{DIVS}$-MFLs in diesen 3D-Szenen betrachtet. In Abb. 4.57 ist zu erkennen, dass diese Anzahl superlinear mit der Anzahl der 3D-Objekte und ungefähr linear mit der Zahl der Lichtquellen steigt. Eine genauere Analyse ergibt, dass die Zahl der $\mathcal{DIVS}$-MFLs ungefähr mit $\mathbf{O(m \cdot n^{1,2})}$ steigt.

Diese Aussage ist jedoch nur von beschränkter Gültigkeit, da die Anordnung der Lichtquellen in der 3D-Szene hierbei eine große Rolle spielt. So ergibt sich z.B. für die 3D-Szene ***S04b*** lediglich ein Wachstum mit $\mathbf{O(m \cdot n)}$.

Insgesamt kann jedoch festgehalten werden, dass die Zahl der $\mathcal{DIVS}$-MFLs lediglich einen Bruchteil der Zahl aller MFLs (nach Abb. 4.8 in Kap. 4.3.1 ca. 4% aller MFLs) ausmacht und somit ein eigener Ansatz für die gezielte Berechnung dieser MFLs sinnvoll ist.

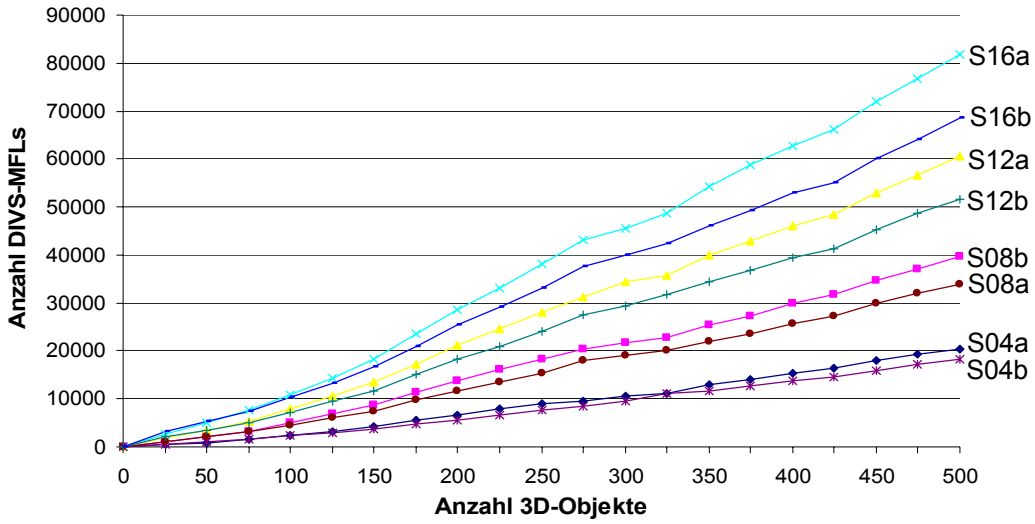


Abb. 4.57
Anzahl $\mathcal{DIVS}$-MFLs in den 3D-Szenen ***S04a-S16***a und ***S04b-S16b***

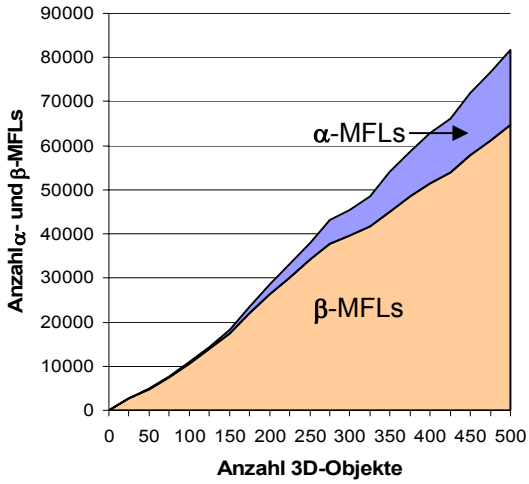


Abb. 4.58
Anzahl α- und β-MFLs in 3D-Szenen ***S04a-S16a***

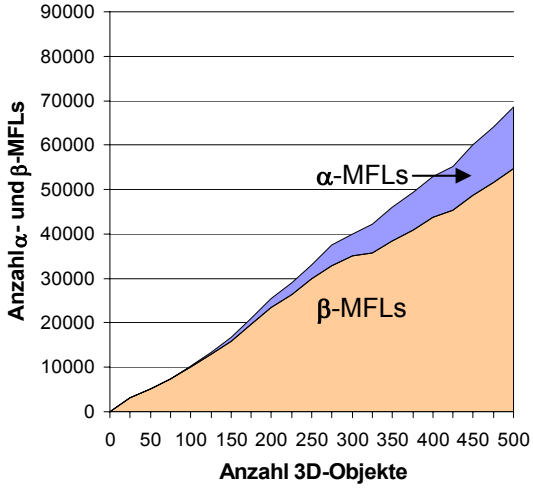


Abb. 4.59
Anzahl α- und β-MFLs in 3D-Szenen ***S04b-S16b***

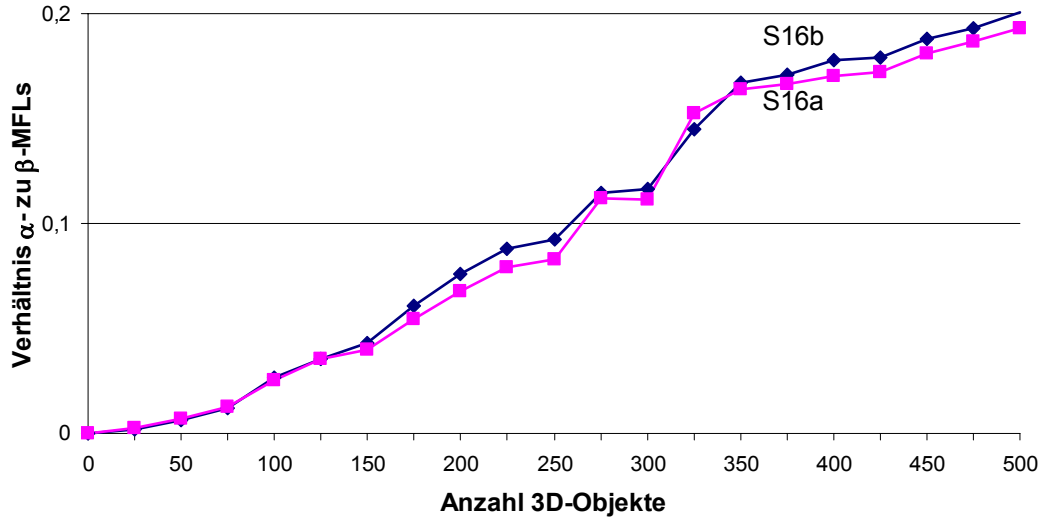


Abb. 4.60
Verhältnis von α- und β-MFLs in den 3D-Szenen ***S16a*** und ***S16b***

Als nächstes soll die Zahl der α- und β-MFLs genauer betrachtet werden, dies ist in Abb. 4.58 und Abb. 4.59 für die 3D-Szenen ***S16a*** und ***S16b*** dargestellt. Während die Anzahl der β-MFLs annährend linear mit der Zahl der 3D-Objekte steigt, wächst die Anzahl der α-MFLs deutlich superlinear. Abb. 4.60 zeigt zur Verdeutlichung das Verhältnis von α- zu β-MFLs. Auch hier ist zu erkennen, dass die Zahl der α-MFLs schneller wächst als die der β-MFLs.

Dies kann dadurch erklärt werden, dass es nach dem Konstruktionsprinzip der α- und β-MFLs mehr Generator-Kandidaten für α- als für β- MFLs gibt: im Falle der **E4**-MFLs gibt es für ein festes 3D-Objekt **O** und eine Lichtquelle **L** mit **k** 3D-Objekten im

Lichtquellen-Shaft **S(O, L)** insgesamt **O(k^3)** Generator-Kandidaten für **α-E4**-MFLs, aber nur **O(k^2)** Generator-Kandidaten für **β-E4**-MFLs, da eine der Generator-Kanten nach Definition der β-MFLs zur Lichtquelle **L** gehören muss.

Mit diesen Überlegungen überrascht es, dass die Zahl α-MFLs zunächst deutlich kleiner ist als die der β-MFLs. Dies liegt daran, dass es eine Menge von β-MFLs gibt, die anschaulich als „Grundstock" immer existieren (z.B. **β-VV**-MFLs) und weiterhin bestimmte α-MFLs erst dann existieren können, wenn es „genügend" Generator-Kandidaten im Lichtquellen-Shaft **S(O, L)** gibt, z.B. mindestens drei für α-E4-MFLs.

Als nächstes soll die Auswahl dieser **k** Generator-Kandidaten betrachtet werden. Nach dem in Kap. 4.3.3 formulierten Verfahren werden für ein 3D-Objekt **O** alle anderen 3D-Objekte $\mathbf{O_i}$ gesucht, die mit **O** und einer Lichtquelle α- oder β-MFLs definieren können.

Abb. 4.61 und Abb. 4.62 zeigen die durchschnittliche Anzahl dieser 3D-Objekte $\mathbf{O_i}$ in den 3D-Szenen ***S04a*** – ***S16a***. Dabei wird unterschieden zwischen der durchschnittlichen Gesamtzahl der 3D-Objekte in allen Lichtquellen-Shafts (Abb. 4.61) und der durchschnittlichen Anzahl der 3D-Objekte in jeweils einem der Lichtquellen-Shafts.

In Abb. 4.62 ist zu erkennen, dass die durchschnittliche Anzahl der 3D-Objekte pro Lichtquellen-Shaft ungefähr mit **O($\sqrt[3]{n}$)** steigt, was der in Kap. 3.5.8 und [20] diskutierten theoretischen Abschätzung entspricht.

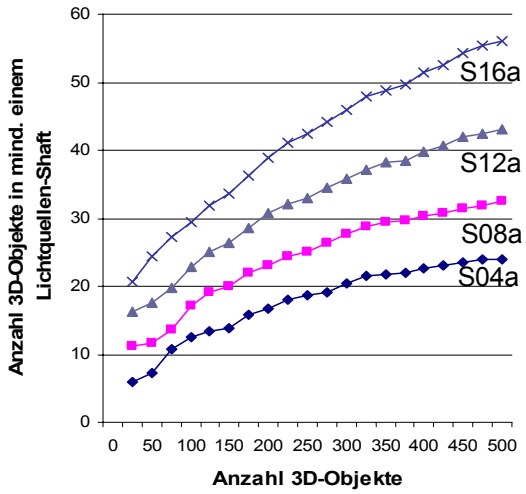


Abb. 4.61
Durchschnittliche Anzahl von 3D-Objekten in allen Lichtquellen-Shafts (3D-Szenen ***S04a-S16a***)

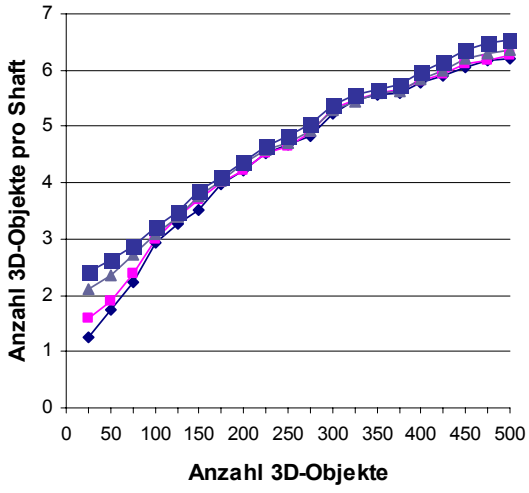


Abb. 4.62
Durchschnittliche Anzahl von 3D-Objekten pro Lichtquellen-Shaft (3D-Szenen ***S04a-S16a***)

Insbesondere wird mit Abb. 4.61 und Abb. 4.62 auch klar, warum die Einteilung der 3D-Objekte in den Lichtquellen-Shafts nach ihren Shaft-Signaturen sinnvoll ist: Ohne die gespeicherten Shaft-Signaturen müssten bei 16 Lichtquellen und 500 3D-Objekten ca. 55 * 55 = 3025 Kombinationen von 3D-Ojekten auf Zugehörigkeit zum $\mathcal{DIVS}$ getestet werden, während mit den Shaft-Signaturen nur ca. 6,5 * 6,5 * 16 ≈ 675 Kombinationen getestet werden müssen.

Um diese 3D-Objekte auszuwählen, wurde in Kap. 4.3.3 die hierarchische Shaft-Klassifikation vorgeschlagen. Deren „Qualität“, d.h. Laufzeitverhalten, hängt in erster Linie von einer gleichmäßigen Verteilung der Shaft-Elemente über die BB-Hierarchie der Lichtquellen ab. Würden sich die Shaft-Elemente auf wenige B-Boxen der BB-Hierarchie konzentrieren, würde die hierarchische Shaft-Klassifikation keinen wesentlichen Vorteil gegenüber der normalen Shaft-Klassifikation bieten.

In Abb. 4.63 und Abb. 4.64 sind die durchschnittlichen Zahlen von Shaft-Elementen pro Ebene der BB-Hierarchie dargestellt. Es ist zu erkennen, dass die Belegung der BB-Hierarchie mit Shaft-Elementen bis auf wenige „Ausreißer“ recht gleichmäßig ist.

Insbesondere ist zu erkennen, dass die Verteilung der Shaft-Ebenen auf die Lichtquellen-BB-Hierarchie mit zunehmender Zahl von Lichtquellen gleichmäßiger wird.

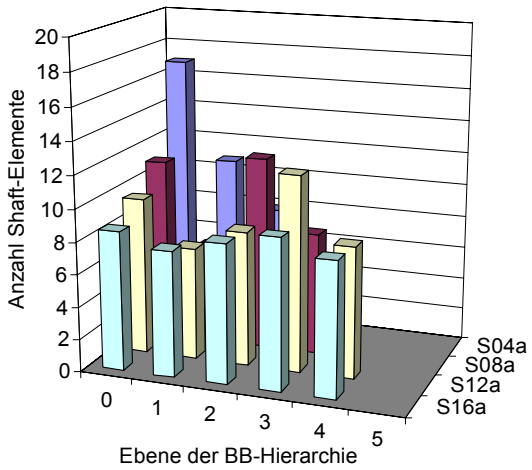


Abb. 4.63
Shaft-Elemente pro Ebene der BB-Hierarchie
(3D-Szenen ***S04a-S16a***)

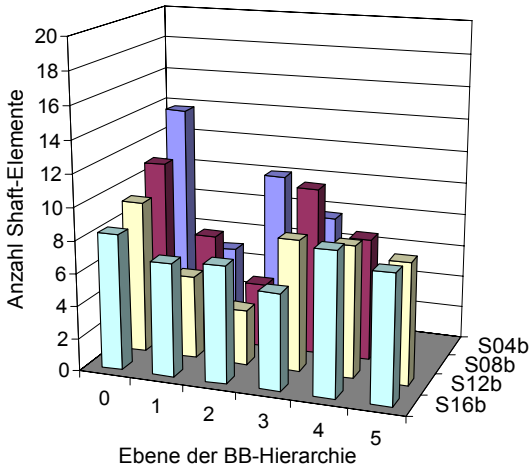


Abb. 4.64
Shaft-Elemente pro Ebene der BB-Hierarchie
(3D-Szenen ***S04b-S16b***)

Abschließend soll nun der eigentliche Aufwand für die Konstruktion des $\mathcal{DIVS}$ einer 3D-Szene betrachtet werden. Abb. 4.65 zeigt die Laufzeiten zur Berechnung des $\mathcal{DIVS}$ für die oben betrachteten 3D-Szenen. Zum Vergleich ist noch der Anfang der Kurve für die Rechenzeit zur Konstruktion des vollständigen Visibility Skeletons für die 3D-Szene ***S16b*** dargestellt. Mit einer maximalen Rechenzeit von ca. 40 Sekunden für die 3D-Szenen ***S04a*** und ***S04b*** gegenüber ca. 30 Minuten bei Konstruktion des vollständigen Visibility Skeletons ist der Vorteil des in diesem Kapitel erarbeiteten Ansatzes zur Konstruktion des $\mathcal{DIVS}$ deutlich zu erkennen.

Eine genauere Analyse der Laufzeiten ergibt eine Komplexität von $\mathbf{O(m \cdot n^{1,7})}$ gegenüber $\approx\mathbf{O(n^{2,45})}$ für die Konstruktion des vollständigen Visibility Skeletons. Analog zu den Überlegungen in Kap. 3.5.8.2 und [20] ergibt sich theoretisch mit jeweils $\mathbf{O(\sqrt[3]{n})}$ 3D-Objekten in $\mathbf{O(m \cdot n)}$ Lichtquellen-Shafts (vgl. auch Abb. Abb. 4.62) ein Gesamtaufwand von $\mathbf{O(m \cdot n^2)}$. Bezieht man jedoch wiederum analog zu Kap. 3.5.8.2 auch die vollständige Verdeckung von Generator-Kandidaten ein (vgl. Abb. 3.165-Abb. 3.170), so ergibt sich ein Aufwand von $\mathbf{O(m \cdot n^{5/3})}$, was den in Abb. 4.65 dargestellten Messwerten entspricht.

Zusammenfassend kann man also festhalten, dass durch das in diesem Kapitel vorgeschlagene Verfahren zur Konstruktion des $\mathcal{DIVS}$ der Aufwand zur Konstruktion der α- und β-MFLs im Average Case von $\mathbf{O(n^{2,45})}$ auf $\mathbf{O(m \cdot n^{1,7})}$ sinkt.

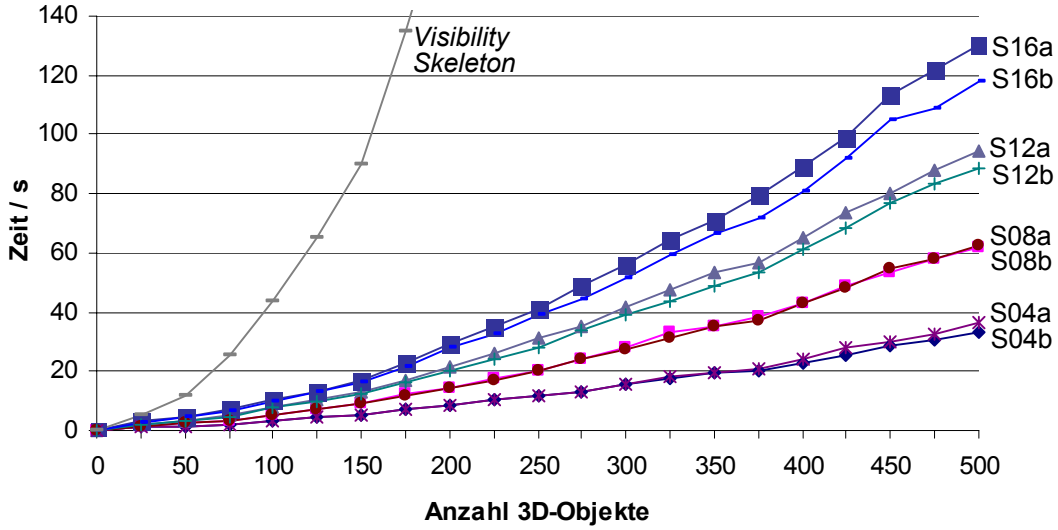


Abb. 4.65
Zeit zur Berechnung des $\mathcal{DIVS}$ für die 3D-Szenen ***S04a-S16a*** und ***S04b-S16b***

4.5 Zusammenfassung

In diesem Kapitel wurde die Konstruktion des Direct Illumination Visibility Skeletons ($\mathcal{DIVS}$) einer 3D-Szene diskutiert. Das $\mathcal{DIVS}$ ist ein Teilgraph des Visibility Skeletons, der nur die MFLs und Sichtbarkeitswechsel enthält, die zur direkten Beleuchtung gehören.

Die grundlegende Idee der Konstruktion ist, für alle 3D-Objekte $\mathbf{O_i}$ diejenigen 3D-Objekte $\mathbf{O_j}$ auszuwählen, die mit $\mathbf{O_i}$ und den Lichtquellen α- oder β-MFLs, d.h. MFLs, die zur direkten Beleuchtung gehören, definieren können.

Diese Konstruktion wurde in drei Schritten durchgeführt:

(1) Auswahl der von mindestens einer Lichtquelle lokal sichtbaren 3D-Objekte $\mathbf{O_i}$

(2) Für jedes dieser 3D-Objekte $\mathbf{O_i}$: Auswahl der 3D-Objekte $\mathbf{O_j}$, die mit dem 3D-Objekte $\mathbf{O_i}$ und den Lichtquellen α- oder β-MFLs definieren können.

(3) Konstruktion der α- und β-MFLs aus den den so gefundenen Generator-Kandidaten.

Für jeden dieser Schritte wurde jeweils ein Schema zur Umsetzung vorgeschlagen. Insbesondere wurden für Punkt (2) die Konzepte der verallgemeinerten Shafts und der Shaft-Klassifikation zu hierarchischen Shafts und der hierarchischen Shaft-Klassifikation erweitert.

Dabei übertragen sich sämtliche in Kap. 3.4 und 3.5 erarbeiteten Verbesserungen für die Shaft-Klassifikation und die Aufspaltung in Kind-Shafts bzw. die Beschränkung von Shafts direkt auf die Konstruktion des $\mathcal{DIVS}$ einer 3D-Szene.

Als Ergebnis dieser Verbesserungen kann festgehalten werden, dass das $\mathcal{DIVS}$ einer 3D-Szene $\mathbf{O(m \cdot n^{1,2})}$ α- und β-MFLs enthält und im Average Case mit Aufwand $\mathbf{O(m \cdot n^{1,7})}$ berechnet werden kann. Damit ist dieser Ansatz wesentlich besser als die vollständige Berechnung des Visibility Skeletons und anschließender Auswahl der α- und β-MFLs mit Gesamtaufwand $\mathbf{O(n^{2,45})}$.

5 GloVE – Global Visibility Environment

Die in den Kapiteln 3 und 4 dargestellten Algorithmen zur Berechnung der globalen Sichtbarkeit in einer 3D-Szene wurden im Rahmen dieser Arbeit in einem gemeinsamen System namens GloVe (**Glo**bal **V**isibility **E**nvironment) implementiert und getestet.

In diesem Kapitel wird das GloVe-System in seiner grundlegenden Struktur und Funktionalität vorgestellt und diskutiert. Die dabei verwendete Darstellungsform ist im Wesentlichen [5] entnommen.

Zunächst werden die Anforderungen an das System beschrieben, insbesondere die für die Software-Entwicklung eher untypischen schrittweisen Reduzierungen und Erweiterungen dieser Anforderungen. Dann wird der grundlegende Aufbau von GloVe anhand eines Paketdiagramms dargestellt. Abschließend werden kurz die Implementierung von GloVe und die dazu notwendigen Überlegungen diskutiert.

5.1 Anforderungen an das GloVe-System

Das GloVe-System ist kein ausgereiftes Programm für den Softwaremarkt, sondern eine Testumgebung für Algorithmen zur Berechnung der globalen Sichtbarkeit in 3D-Szenen. Dies bedeutet insbesondere, dass GloVe nicht als Abschluss der theoretischen Überlegungen in Kap. 3 und 4 implementiert wurde, sondern während dieser Überlegungen schrittweise weiter entwickelt wurde.

Damit ergab sich das Problem, dass zu Beginn der Entwicklung von GloVe keine wohldefinierten Anforderungen, z.B. in Form eines Pflichtenhefts, vorlagen. Zwar gab es gewisse Mindestanforderungen, aber diese waren zunächst nur vage „Absichtserklärungen“.

Eine Mindestanforderung war z.B., dass GloVe 3D-Szenen laden und anzeigen können muss. Was dabei allerdings eine 3D-Szene genau ist, d.h. welche 3D-Objekte mit welchen Eigenschaften sie enthalten darf, war zunächst nicht klar und ergab sich erst aus den genauen Betrachtungen der Näherungs- und exakten Verfahren in Kap. 3.

Damit entsprechen der Entwurf und die Implementierung von GloVe nicht den typischen inkrementellen oder evolutionären Vorgehensmodellen der Software-Entwicklung: Bei diesen Vorgehensmodellen wird ausgehend von einer Liste wohldefinierter Anforderungen ein System implementiert, das diesen Anforderungen entspricht und dann schrittweise durch neue Anforderungen erweitert.

Beim GloVe wurde dagegen eine Folge von Teilsystemen entworfen und implementiert, die sich durch abwechselnde Reduzierung und Erweiterung der Anforderungen unterschieden. So war z.B. die Beschränkung auf polygonale 3D-Szenen mit konvexen Objekten eine Reduzierung der Anforderungen gegenüber allgemeinen 3D-Szenen, während die Berechnung der globalen Sichtbarkeit in solchen 3D-Szenen eine Erweiterung der Anforderungen ist.

Damit entspricht das Vorgehen bei Entwurf und Implementierung von GloVe also einem inkrementellen Prototyping mit abwechselnden Reduzierungen und Erweiterungen der Anforderungen bei gleichzeitig wachsendem System-Umfang.

Dies ist in Abb. 5.1 verdeutlicht: auf der linken Seite sind die schrittweisen Reduzierungen und Erweiterungen der Anforderungen und auf der rechten Seite die schrittweise Erweiterung des Umfangs des GloVe-Systems dargestellt.

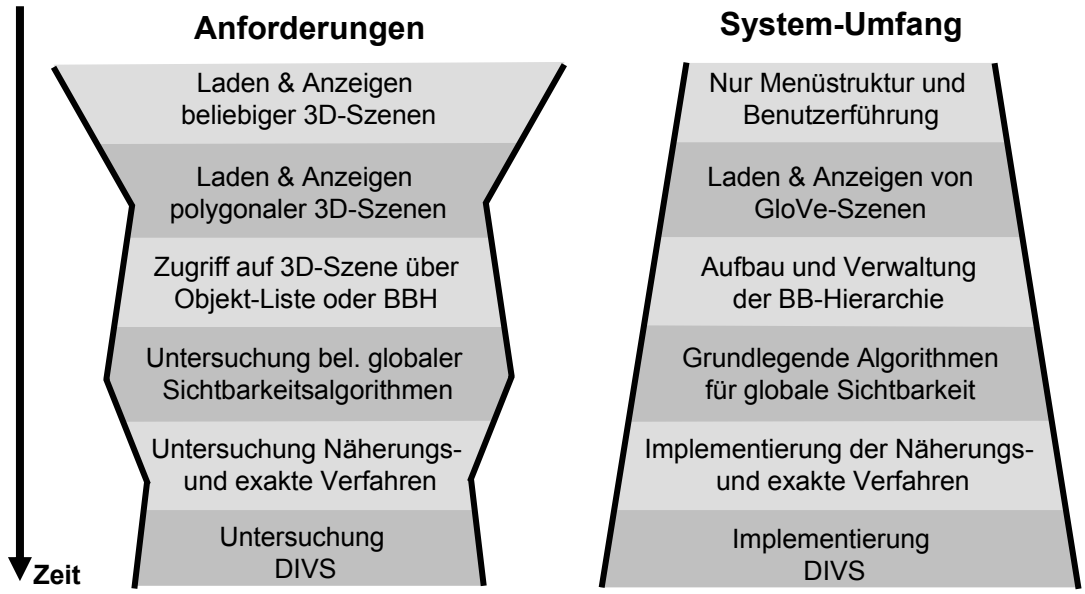


Abb. 5.1
Reduzierungen und Erweiterungen der Anforderungen und Erweiterungen des Systems

5.2 Aufbau des GloVe-Systems

Die im letzten Teilkapitel diskutierte Problematik der schrittweisen Reduzierungen und Erweiterungen der Anforderungen spiegelt sich auch im Aufbau des GloVe-Systems wider: Der grobe Aufbau, d.h. die Paketstruktur, ist klar definiert und einzelne Pakete sind sauber voneinander abgegrenzt. Im Detail, d.h. den Klassen und ihren Beziehungen, macht sich dagegen bemerkbar, dass es sich bei GloVe um eine Testumgebung handelt.

Hier soll nur die Paketstruktur von GloVe diskutiert werden. Abb. 5.2 zeigt eine grobe Sicht auf die Struktur von GloVe in Form eines Paketdiagramms, die eingezeichneten Pfeile symbolisieren „*Benutzt*"-Beziehungen. Der proprietäre Charakter der dargestellten Aufteilung in Pakete zeigt sich z.B. darin, dass alles, „was sonst nirgendwo hineinpasst", in einem Paket „*Hilfsmittel*" zusammengefasst wurde.

Ansonsten entspricht die Aufteilung einer einfachen Drei-Schichten-Architektur: die Benutzeroberfläche bildet die obere Schicht, die Pakete zur Verwaltung von 3D-Szenen und zur Berechnung globaler Sichtbarkeit bilden die mittlere Schicht und das Paket „*Hilfsmittel*" die untere Schicht. Dies entspricht einer vereinfachten *MVC*-Architektur (*Model-View-Controller*, vgl. [5]), wobei *View* und *Controller* zusammengefasst wurden, da es in GloVe nur genau eine mögliche Sicht auf die 3D-Szene gibt und nur für diese Sicht Benutzerinteraktionen verarbeitet werden müssen. .

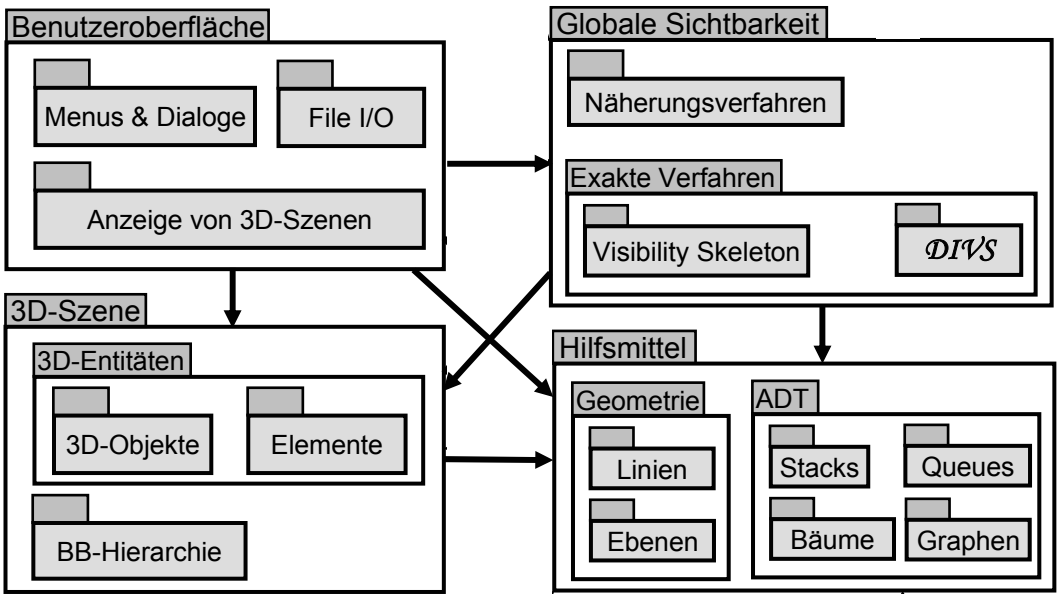


Abb. 5.2
Paketstruktur von GloVe

5.3 Implementierung des GloVe-Systems

Das GloVe-System wurde in Java unter Verwendung des *JDK 1.4.1* und *Java3D 1.3* (vgl. dazu [2] und [49]) implementiert. Die Entscheidung für Java als Implementierungssprache hatte im Wesentlichen zwei Gründe:

(1) Java ist eine objekt-orientierte Sprache mit einer mächtigen Klassenbibliothek. Durch die Plattform-Unabhängigkeit von Java kann GloVe einfach auf andere Betriebssysteme portiert werden. Weiterhin können Java-Programme gegenüber z.B. C++-Programmen schneller entwickelt werden, da dem Programmierer viele Aufgaben wie z.B. die Freigabe nicht mehr benötigten Speicherplatzes von der Laufzeitumgebung *JRE* (*Java Runtime Environment*) abgenommen werden.

(2) Die Klassenbibliothek Java3D bietet eine performante Möglichkeit zur Darstellung und Manipulation von 3D-Szenen. Viele typische Operationen wie z.B. Navigation durch die 3D-Szene oder Interaktion mit den 3D-Objekten (z.B. Auswahl von 3D-Objekten mit der Maus) sind bereits in allgemeinen Klassen implementiert.

Abschließend zeigt Abb. 5.3 die GloVe-Benutzeroberfläche mit einer geladenen 3D-Szenen und einigen MFLs in dieser 3D-Szene.

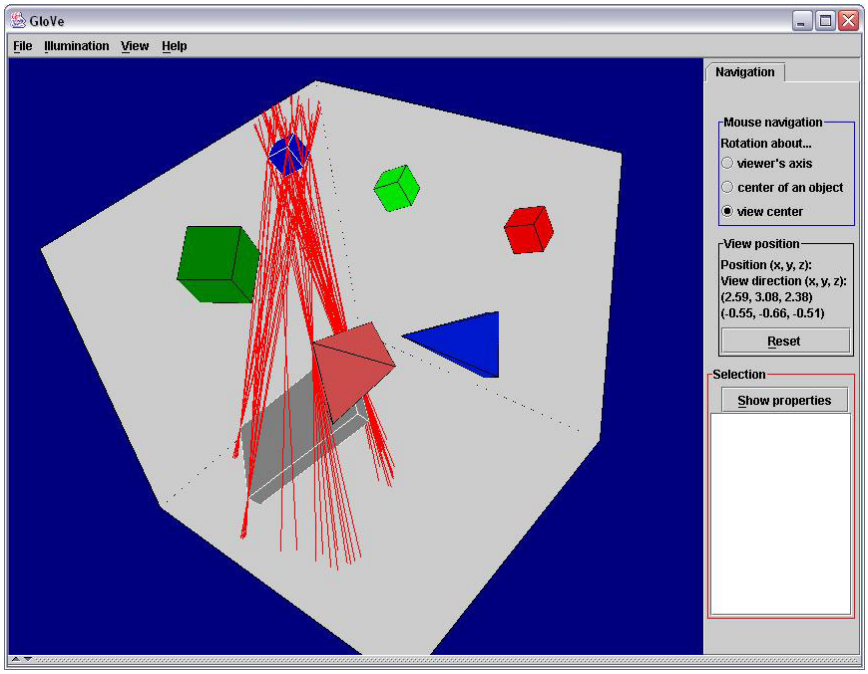


Abb. 5.3
GloVe mit geladener 3D-Szene und einigen MFLs

6 Zusammenfassung

In dieser Arbeit wurden Verfahren zur Berechnung der globalen Sichtbarkeit in einer 3D-Szene untersucht. Dazu wurden zwei Typen von Verfahren betrachtet: Näherungs- und exakte Verfahren. Der Schwerpunkt der Arbeit lag auf den exakten Verfahren, die Näherungsverfahren wurden lediglich der Vollständigkeit halber und als Vorbereitung für die exakten Verfahren betrachtet.

Das wichtigste Konzept der Näherungsverfahren ist das des (verallgemeinerten) Shafts: alle 3D-Objekte, die „im“ Shaft zweier Extremal-Objekte liegen, beeinflussen die Sichtbarkeit zwischen diesen Extremal-Objekten. Die Frage, ob ein 3D-Objekt „in“ einem Shaft liegt, wird als Shaft-Klassifikation bezeichnet.

Als Vertreter der exakten Verfahren wurde das Visibility Skeleton einer 3D-Szene vorgestellt. Das Visibility Skeleton ist ein Graph, der alle globalen Sichtbarkeitsinformationen einer 3D-Szene enthält. Dieser Graph ist aus Sichtbarkeitswechseln und maximal freien Liniensegmenten aufgebaut.

Sichtbarkeitswechsel sind Flächen im 3D, an denen sich die Sichtbarkeit zwischen Extremal-Elementen nicht unendlich oft stetig differenzierbar ändert, d.h. anschaulich „Schattenlinien“ auf den Extremal-Elementen.

Maximal freie Liniensegmente sind Liniensegmente im 3D und entstehen durch Schnitte von Sichtbarkeitswechseln. Die wichtigste Eigenschaft von MFLs ist, dass sie genau vier Generator-Kanten der 3D-Szene schneiden. Für die Konstruktion der MFLs müssen Tupel von Generator-Elementen daraufhin getestet werden, ob es ein MFL durch sie gibt und ob dieses MFL global sichtbar ist.

Als Spezialfall der exakten Verfahren wurde das im Rahmen dieser Arbeit entwickelte Direct Illumination Visibility Skeleton vorgestellt. Das $\mathcal{DIVS}$ ist ein Teilgraph des Visibility Skeletons, der nur die MFLs und Sichtbarkeitswechsel enthält, die zur direkten Beleuchtung gehören. Damit ist es möglich, die Sichtbarkeit in 3D-Szenen getrennt für direkte und indirekte Beleuchtung zu berechnen.

6.1 Ergebnisse

Das zentrale Ergebnis dieser Arbeit ist die Definition des Direct Illumination Visibility Skeletons und der vorgeschlagene Algorithmus zur Konstruktion desselben. Die Überlegungen hierzu gingen von den Verbesserungsvorschlägen für die Näherungs- und exakten Verfahren in Kap 3 aus. Die folgende Übersicht zeigt die Ergebnisse der Arbeit für die betrachteten Verfahren:

- ***Näherungsverfahren*** (Kap. 3.4): In diesem Kapitel wurde hauptsächlich die performante Umsetzung der Klassifikation einer 3D-Szene anhand eines (verallgemeinerten) Shafts betrachtet. Dazu wurde die BB-Hierarchie der 3D-Szene ausgenutzt und die Klassifikation anhand eines vollständigen Shafts zur Klassifikation anhand einzelner Shaft-Elemente verallgemeinert. Mit diesen Verbesserungen kann eine 3D-Szene mit **n** 3D-Objekten mit sublinearem Aufwand anhand eines verallgemeinerten Shafts klassifiziert werden.

- ***Exakte Verfahren*** (Kap. 3.5): Für die Konstruktion des Visibility Skeletons einer 3D-Szene wurden mehrere Verbesserungen vorgeschlagen. Zunächst wurde ein Schema zur Berechnung der Sichtbarkeitswechsel aus den MFLs vorgestellt, das die Sichtbarkeitswechsel alleine aus den Extremal- und Generator-Elementen der adjazenten MFLs konstruiert. Für die Berechnung der MFLs wurden die verallgemeinerten Shafts aus Kap. 3.4 verwendet und an das Problem der MFL-Konstruktion angepasst. Insbesondere wurde für die Berechnung der **E4**-MFLs ein eigenes Optimierungsschema erarbeitet, das den globalen Sichtbarkeitstest so früh wie möglich durchführt und damit die Berechnung ungültiger MFLs vermeidet.

  In einer 3D-Szene mit **n** 3D-Objekten wurden folgende Resultate gemessen:

Fall	***ohne Opt.***	***mit Opt.***
Best Case (3D-Objekte ungefähr gleicher Größe)	$\approx O(n^{2,8})$	$\approx O(n^{2,3})$
Average Case (3D-Objekte gleicher Größenordnung)	$\approx O(n^{2,83})$	$\approx O(n^{2,45})$
Worst Case (alle 3D-Objekte sind „Balken“)	$\approx O(n^{2,85})$	$\approx O(n^{2,75})$

Tabelle 6.1

Als Ergebnis dieser Verbesserungen kann also festgehalten werden, dass der Aufwand zur Konstruktion des Visibility Skeletons einer 3D-Szene mit den vorgeschlagenen Verbesserungen im besten Fall deutlich absinkt, gleichzeitig allerdings im schlechtesten Fall weitgehend gleich bleibt. Der konkrete Aufwand hängt dabei in hohen Maßen von der konkreten 3D-Szene ab.

- ***Direct Illumination Visibility Skeleton*** (Kap. 4): Der Aufwand zur Konstruktion des Direct Illumination Visibility Skeleton entspricht in der einfachsten Variante dem zur Konstruktion des vollen Visiblity Skeletons mit anschließender Auswahl der MFLs bzw. Sichtbarkeitswechsel, die zur direkten Beleuchtung gehören.

  Mit dem vorgeschlagenen Konstruktionsverfahren sinkt dieser Aufwand in einer 3D-Szene mit **n** 3D-Objekten und **m** Lichtquellen, auf ungefähr $\mathbf{O(m \cdot n^{1,7})}$. Hierbei gelten alle Aussagen, die bereits für die Konstruktion des vollen Visibility Skeletons getroffen wurden, d.h. der Aufwand hängt wesentlich von der konkreten 3D-Szene ab und kann im Worst Case deutlich über $\mathbf{O(m \cdot n^{1,7})}$ liegen.

6.2 Ausblick

Das (Direct Illumination) Visibility Skeleton ist eine komplexe Datenstruktur, die in vielen aufwändigen Schritten konstruiert wird. In dieser Arbeit wurden einige Ansatzpunkte zur Verbesserung dieser Konstruktion erarbeitet, die jedoch nicht den Anspruch auf Vollständigkeit erheben. Die folgenden Punkte zeigen, an welchen Stellen diese Arbeit fortgeführt werden könnte:

- Die Forschung im Kontext des Visibility Skeletons konzentriert sich in jüngster Zeit auf die Reduzierung der Größe der erzeugten Datenstruktur. In [17] wird ein viel versprechender Ansatz zur robusten Verschmelzung adjazenter Sichtbarkeitswechsel und MFLs vorgeschlagen.

  Die Grundidee dabei ist, die Menge der Generatoren eines MFLs um die Elemente zu erweitern, die „nahe“ bei diesem MFL liegen und für diese (Kombinationen von) Elemente(n) keine MFLs mehr zu berechnen.

Der Aufwand zur Konstruktion des Visibility Skeletons sinkt hierdurch jedoch nicht, da immer noch alle Kombinationen von Generator-Elementen betrachtet werden müssen und lediglich für einige dieser Kombinationen die explizite Berechnung der MFLs nicht mehr durchgeführt werden muss.

- Das in Kap. 3.5 und 4 diskutierte (Direct Illumination)Visibility Skeleton ist nur für polygonale 3D-Objekte definiert. In [20] wird eine Verallgemeinerung des Visibility Skeletons zum 3D-Visibility Complex vorgeschlagen, womit auch nicht-polygonale 3D-Objekte wie z.B. Kugeln oder Bezier-Flächen behandelt werden können.

  Hierfür existieren nach aktuellem Stand nur Ansätze zur Berechnung der MFLs und Sichtbarkeitswechsel, jedoch keine Implementierung. Die in dieser Arbeit entwickelten Ansätze zur Verbesserung der Konstruktion können nicht auf den 3D-Visibility Complex übertragen werden, da die polygonale Form der 3D-Szenen eine wesentliche Grundvorrausetzung für alle Verfahren zur (hierarchischen) Shaft-Klassifikation und Auswahl der Generator-Kandidaten war.

- In der momentanen Form ist das Visibility Skeleton nicht für dynamische 3D-Szenen, d.h. 3D-Szenen mit sich bewegenden oder verändernden 3D-Objekten, geeignet. In [21] wird eine Erweiterung des Visibility Skeletons für dynamische 3D-Szenen vorgeschlagen, die jedoch großen Einschränkungen unterliegt: die 3D-Objekte müssen in statische und dynamische eingeteilt und getrennt behandelt werden. Die Bewegung eines dynamischen 3D-Objekts entspricht dann dem Entfernen an der alten und anschließenden Einfügen an der neuen Position bei gleichzeitiger Neuberechnung des betroffenen Teils des Visibility Skeletons.

Das Gebiet der globalen Sichtbarkeitsalgorithmen bietet also viel Raum für zukünftige Forschungen. Der Autor hofft, neben dem eigenen Beitrag auch eine Grundlage für weitere Arbeiten auf diesem Gebiet gelegt zu haben.

7 Anhang

7.1 Übersicht der betrachteten 3D-Szenen

Für die Experimente und Messungen in Kap. 3.5.8 und 4.4 wurden verschiedene Zufallsszenen betrachtet. Im Folgenden soll ein kurzer Überblick der wichtigsten Eigenschaften dieser 3D-Szenen gegeben werden:

- ***Enthaltene 3D-Objekte:*** Alle betrachteten 3D-Szenen sind aus regelmäßigen Polyedern aufgebaut, d.h. Tetraeder, Pyramiden, Quader und Oktaeder. Die Wahrscheinlichkeiten verteilen sich dabei wie folgt:

Typ	***Tetraeder***	***Pyramide***	***Quader***	***Oktaeder***
Wkt.	*15%*	*20%*	*40%*	*25%*

  Jede der 3D-Szenen wird von einem quadratischen Raum mit Seitenlänge $\mathbf{r = 1}$ umgeben.

- ***Position und Rotation:*** Die Position und die Rotation der 3D-Objekte wird als gleichverteilt angenommen.

- ***Größe der 3D-Objekte:*** Für die Messungen in Kap. 3.5.8 und 4.4 wurden Folgen von 3D-Szenen (meist mit 25 – 500 3D-Objekten) betrachtet. Um einen ähnlichen Aufbau dieser Folgen von 3D-Szenen zu gewährleisten, wurde folgendes Schema angewendet:
  - Standardgröße: Für ein 3D-Szene mit $\mathbf{n}$ 3D-Objekten wird eine Standardgröße $\mathbf{s}$ der 3D-Objekte nach folgender Formel berechnet: $\mathbf{s = d\sqrt{0,3}\sqrt[3]{n}}$

    Die einzelnen Faktoren kann man sich wie folgt klar machen:
    - $\mathbf{d}$: Faktor, der die „Dichte“ der 3D-Szene beschreibt (meist $\mathbf{0,1 < d < 0,5}$)
    - $\sqrt[3]{\mathbf{n}}$: Bei Gleichverteilung der 3D-Objekte (vgl. oben) „passen“ in einen Würfel mit Seitenlänge $\mathbf{r = 1}$ maximal $\mathbf{n}$ Würfel mit Seitenlänge $\sqrt[3]{\mathbf{n}}$.
    - $\sqrt{\mathbf{3}}^{-1}$: Da die 3D-Objekte in der 3D-Szenen rotiert sein können, kann ein Würfel der Seitenlänge $\sqrt[3]{\mathbf{n}}$ in x-, y- oder z-Richtung maximale Ausdehnung $\sqrt{\mathbf{3}}$ (entspricht maximaler Diagonalen durch Würfel) haben.

- ***Verhältnis der Seitenlängen:*** Für ein 3D-Objekt **O** unterscheidet sich die Skalierung in x-, y- und z-Richtung maximal um Faktor 3. Eine Sonderregelung wird für „große“ 3D-Objekte getroffen: Hier gilt, dass mindestens eine Skalierung in x-, y- oder z-Richtung Faktor 0.1 haben muss, da sonst die 3D-Szene schon bei wenigen „großen“ 3D-Objekten „voll“ ist.
- ***Verteilung der Größen der 3D-Objekte:*** Mit den bisherigen Aussagen sind alle 3D-Objekte bis auf verschiedene Seitenlängen ungefähr gleich groß. Da diese Annahme unrealistisch ist, wird mit dem folgenden Modell versucht, Eigenschaften „realistischer“ 3D-Szenen nachzubilden.

  Dazu wurde die Verteilung der Größen der 3D-Objekte mittels einer Standardnormalverteilung (Varianz $\sigma^2=1$ und Standardabweichung $\mu=0$) modelliert (vgl. Abb. 7.1). Anschaulich ergibt dies 3D-Szenen mit 3D-Objekten, die sich maximal um einen Faktor von ungefähr 16 unterscheiden und im Wesentlichen aus 3D-Objekten gleicher Größenordnung bestehen.

Abb. 7.2 – Abb. 7.5 zeigen beispielhaft einige der betrachteten 3D-Szenen:

Abb.	*3D-Szene*	*Verwendung*
Abb. 7.2	500 3D-Objekte	Messungen für Visibility Skeleton
Abb. 7.3	2000 3D-Objekte	Messungen für Beschränkung der **EE**-Shafts (***S1***)
Abb. 7.4	500 3D-Objekte	Messungen für DIVS (3D-Szene ***S16a***)
Abb. 7.5	500 3D-Objekte	Messungen für DIVS (3D-Szene ***S16b***)

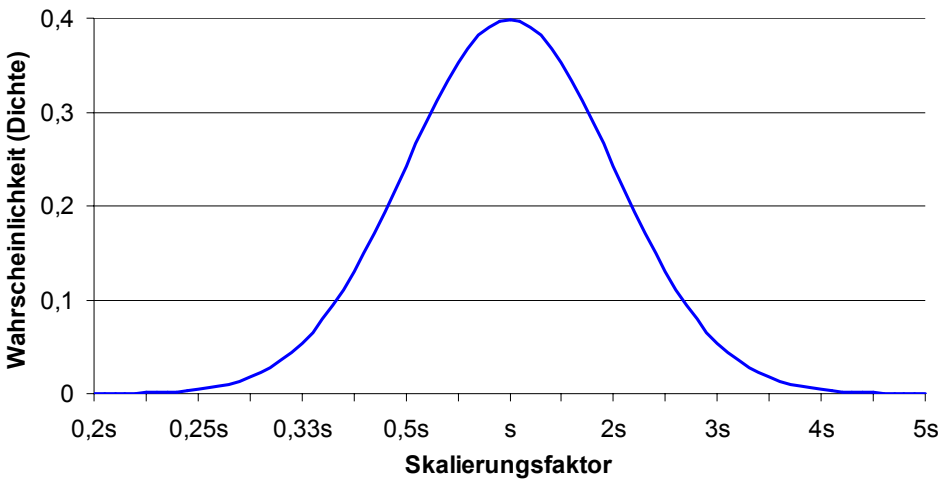


Abb. 7.1
Normalverteilung der Objektgrößen (Varianz $\sigma^2=1$ und Standardabweichung $\mu=0$)

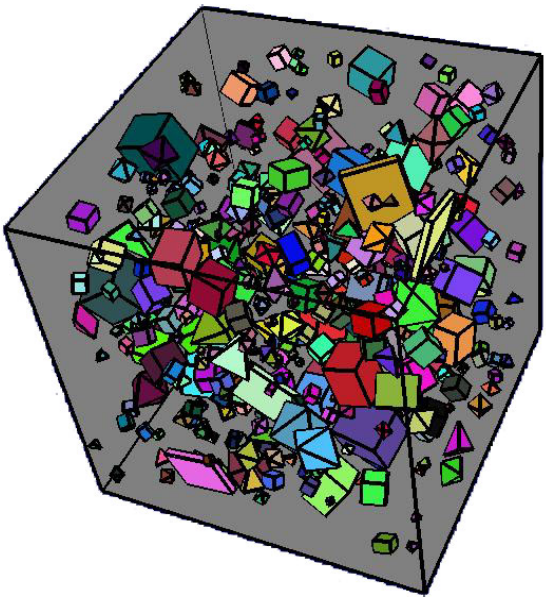


Abb. 7.2
3D-Szene mit 500 3D-Objekten
(Zeitmessung Visibility Skeleton)

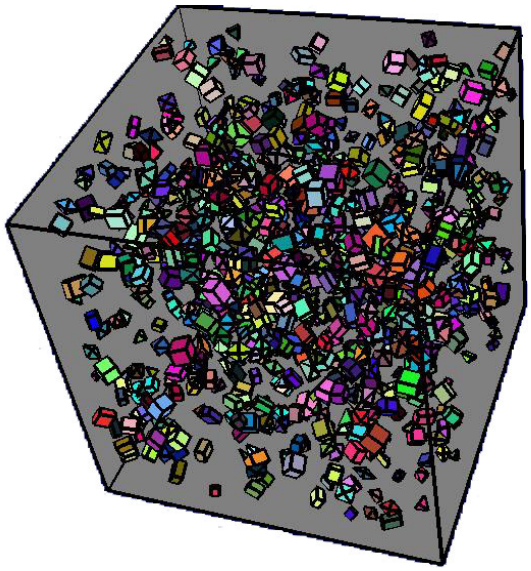


Abb. 7.3
3D-Szene ***S1*** mit 2000 3D-Objekten
(Beschränkung von **EE**-Shafts)

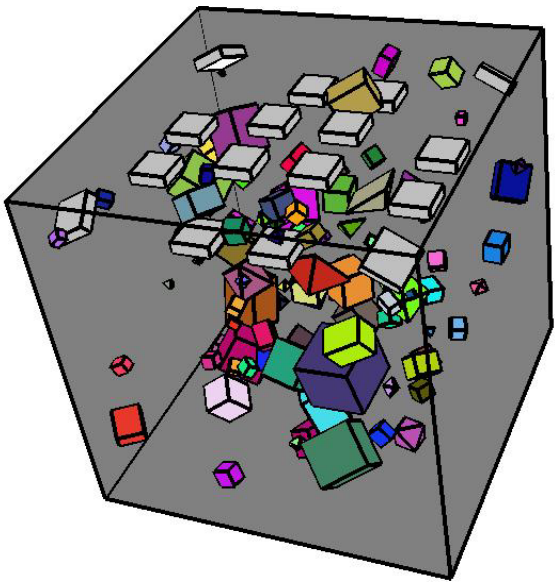


Abb. 7.4
3D-Szene ***S16a*** mit 16 Lichtquellen (hellgrau)
(insgesamt 500 3D-Objekte, hier 400 ausgeblendet)

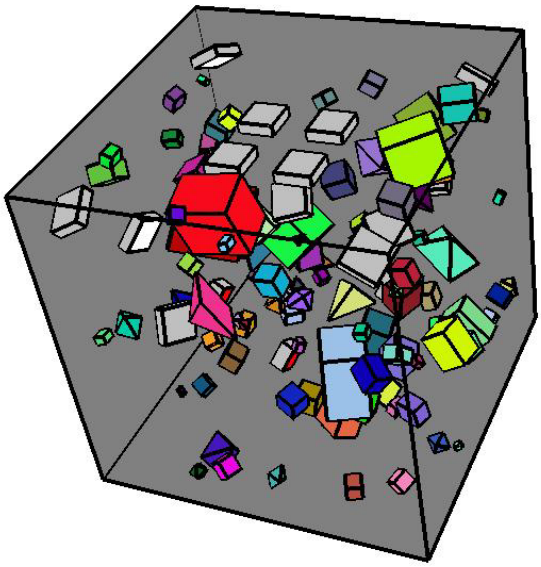


Abb. 7.5
3D-Szene ***S16b*** mit 16 Lichtquellen (hellgrau)
(insgesamt 500 3D-Objekte, hier 400 ausgeblendet)

7.2 Abkürzungsverzeichnis

Abkürzung	Volle Schreibweise	Erklärung
B-Box	***B****ounding Box*	Eine B- Box ist ein achsenparalleler Quader, der 3D-Entitäten (insbesondere auch andere B-Boxen) enthält
BB-Hierarchie	***B****ounding-****B****ox-Hierarchie*	Ein Baum von Bounding Boxen. Die Wurzel der BB-Hierarchie ist eine B-Box, die die gesamte 3D-Szene umfasst, die Blätter sind B-Boxen, die 3D-Objekte enthalten
MFL	***M****aximal* ***f****reies* ***L****iniensegment*	Ein MFL ist eine Linie, die durch vier Generator-Kanten der 3D-Szene geht und von zwei Extremal-Elementen begrenzt wird. Jedes MFL ist adjazent zu mindestens zwei Sichtbarkeitswechseln
E-MFL	***E****dge-MFL*	Ein MFL, das kolinear zu einer Kante der 3D-Szene ist
E4-MFL	***E****dge-****E****dge-****E****dge-****E****dge-MFL*	Ein MFL, das durch vier Kanten der 3D-Szene verläuft
EFE-MFL	***E****dge-****F****acette-****E****dge-MFL*	Ein MFL, das durch zwei Kanten und eine Facette der 3D-Szene verläuft
EVE-MFL	***E****dge-****V****ertex-****E****dge-MFL*	Ein MFL, das durch zwei Kanten und einen Vertex der 3D-Szene verläuft
FEE-MFL	***F****acette-****E****dge- ****E****dge-MFL*	Ein MFL, das durch zwei Kanten und eine Facette der 3D-Szene verläuft
FF-MFL	***F****acette-****F****acette-MFL*	Ein MFL, das durch zwei Facetten verläuft
FvE-MFL	***F****acette-(Facetten-)* ***V****ertex-****E****dge-MFL*	Ein MFL, das durch eine Facette, einen Facetten-Vertex und eine Kante verläuft.
Fvv-MFL	*Facette-(Facetten-) Vertex-(Facetten-) Vertex-MFL*	Ein MFL, das durch zwei nicht adjazente Vertices einer Facette verläuft
VEE-MFL	***V****ertex-****E****dge- ****E****dge-MFL*	Ein MFL, das durch zwei Kanten und einen Vertex der 3D-Szene verläuft
VV-MFL	***V****ertex-****V****ertex-MFL*	Ein MFL, das durch zwei Vertices der 3D-Szene verläuft

7.3 Literaturverzeichnis

Kürzel	*Quelle*
[1]	Angelier, P., M. Pocchiola: „*A sum of squares theorem for visibility*“, Proceedings of the 17th annual symposium on Computational geometry, 2001 , S. 302-311
[2]	Arnold, K., J. Gosling, D. Holmes: „*The Java Programming Language (3rd Edition)*“, Addison Wesley Publishing Corp., 2000 (ISBN: 0-201-704331)
[3]	Arvo, J., D. Kirk: „*A survey of ray tracing acceleration techniques*“ in „*An introduction to ray tracing*“, Academic Press Ltd., 1989, S. 201-262
[4]	Ashdown, I.: „*Radiosity: A Programmers Perspective*“, John Wiley & Sons, 1994 (ISBN: 0-.471-30488-3)
[5]	Balzert, H.: „*Lehrbuch der Objektmodellierung: Analyse und Entwurf*“, Spektrum Akademischer Verlag GmbH, 1999 (ISBN: 3-8274-0285-9)
[6]	Barber , C. B., D.P. Dobkin, H. Huhdanpaa: „*The quickhull algorithm for convex hulls*“, ACM Transactions on Mathematical Software, Volume 22, Issue 4, December 1996, S. 469–483
[7]	Becker, B., P. G. Franciosa, S. Gschwind, T. Ohler, G. Thiemt, P. Widmayer: „*Enclosing many boxes by an optimal pair of boxes*“, Proceedings of STACS 1992
[8]	Bronstein, I. N, K.A. Semendjajew, G. Musiol, H. Mühlig: „*Taschenbuch der Mathematik*“, Verlag Harry Deutsch, 2001 (ISBN: 3-817-120052)
[9]	Cohen, M.F., D. P. Greenberg, D. S. Immel, P. J. Brock: „*An efficient radiosity approach for realistic image synthesis*“, IEEE Computer Graphics and Applications Volume 6, Issue 3, 1986, S. 25-36
[10]	Cohen, M.F., D. P. Greenberg, S. E. Chen, J. R. Wallace: „*A progressive refinement approach for fast radiosity image generation*“, Computer Graphics Volume 22, Issue 4, 1988, S. 75-84
[11]	Cohen, M.F., D. P. Greenberg: „*The Hemi-Cube: A radiosity solution for complex environments*“, Computer Graphics Volume 19, Issue 3, 1985, S. 31-40

[12] Coorg, S., S. Teller: „*Temporally coherent conservative visibility*“, Selected papers from the 12th annual symposium on Computational Geometry, 1999, S. 105-124

[13] Cromwell, P. R: „*Polyhedra*“, Cambridge University Press, 1999 (ISBN: 0-521-664055)

[14] Drettakis, G., E. Fiume: „*A fast shadow algorithm for area light sources using back projection*“ SIGGRAPH 94 Conference Proceedings, 1994, S. 223–230.

[15] Drettakis, G., F. Sillion: „*Accurate visibility and meshing calculations for hierarchical radiosity*“, Proceedings of the eurographics workshop on Rendering techniques, 1996, S. 269-282

[16] Drettakis, G., F. Sillion: „*Interactive Update of global Illumination using a line space hierarchy*“, SIGGRAPH 97 Conference Proceedings, August 1997, S. 57-64

[17] Duguet, F., G. Drettakis: „*Robust epsilon visibility*“, Proceedings of the 29th annual conference on Computer Graphics, 2002, S. 567-575

[18] Durand, F., G. Drettakis, C. Puech: „*3D visibility made visibly simple: An introduction to the visibility skeleton*“, Proceedings of the 13th annual symposium on Computational geometry, 1998, S. 475-476

[19] Durand, F., G. Drettakis, C. Puech: „*Fast and accurate hierarchical radiosity using global visibility*“, ACM Transactions on Graphics, Volume 18, Issue 2, 2000, S. 128-170

[20] Durand, F., G. Drettakis, C. Puech: „*The 3D visibility complex*“, ACM Transactions on Graphics, Volume 21, Issue 1, 2002, S. 176-206

[21] Durand, F., G. Drettakis, C. Puech: „*The visibility skeleton: A powerful and efficient multi-purpose global visibility tool*“, Proceedings of the 24th annual conference on Computer Graphics, 1997, S. 89-100

[22] Durand, F., G. Drettakis, J. Thollot, C. Puech: „*Conservative visibility preprocessing using extended projections*“, Proceedings of the 27th annual conference on Computer Graphics, 2000, S. 239-248

[23] Glaeser, G.: „*Fast Algorithms for 3D-Graphics*“, Springer Verlag, 1994 (ISBN: 0387942882)

[24] Goldman, R. N.: „*Area of Planar Polygons and Volume of Polyhedra*“, Graphics Gems II, Academic Press Inc., 1991 (ISBN: 0-12-064480-0), S. 170-172

[25] Goldmann, R. N.: „*Intersection of two lines in three-space*“, Graphics Gems, Academic Press Inc., 1990(ISBN: 0-12-286166), S.304-305

[26] Goldsmith, J., J. Salmon: „*Automatic creation of object hierarchies for ray tracing*“, IEEE Computer Graphics and Applications, Volume 7, Issue 5, 1987, S. 14-20

[27] Goral, M. C., K. E. Torrance, D. P. Greenberg, B. Battaile: „*Modelling the interaction of light between diffuse surfaces*“, Computer Graphics 18 (3), 1984, S. 213-222

[28] Greiner, G., K. Hormann: „*Efficient clipping of arbitrary polygons*“, ACM Transactions on Graphics, Volume 17, Issue 2, 1998, S. 71-83

[29] Haines, A.E., J. R. Wallace, „*Shaft culling for efficient ray-traced radiosity*“, Proceedings of the 2nd Eurographics Workshop on Rendering, 1994, S. 122-138

[30] Hanrahan, P., D. Salzman, Larry Aupperle: „*A rapid hierarchical radiosity algorithm“,* Computer Graphics, Volume 25, Number 4, 1991, S. 197-206

[31] Holzschuch, N., F. Sillion: „*A efficient progessive refinement strategy for hierarchical radiosity*“, Proceedings of 5th Eurographics Workshop on Rendering, 1994, S. 217-231

[32] Leblanc, L., P. Poulin: „*Guaranteed occlusion and visibility in cluster hierarchical radiosity*“, Proceedings of the 11th Eurographics Workshop on Rendering, 2000, S. 89–100

[33] Liang, Y, B. A. Barsky: „*An analysis and algorithm for polygon clipping*“, Communications of the ACM, Volume 26, Issue 11, 1983, S. 868-877

[34] Lischinski, D. F., Tampieri, D. Greenberg: „*Discontinuity meshing for accurate radiosity*“, IEEE Computer Graphics and Applications, Volume 12 , Issue 6, 1992, S. 25-39

[35] Lischinski, D., F. Tampieri, D. Greenberg: „*Combining hierarchical radiosity and discontinuity meshing*“, Proceedings of the 20th annual conference on Computer Graphics, 1993, S. 199-208

[36] Maillot, P.: „*A fast method for 2D polygon clipping*“, ACM Transactions on Graphics, Volume 11, Issue 3, 1992, S. 276-290

[37] Mariano, A., L. Upson: „*Penumbral shadows*“, Proceedings of the tenth annual symposium on Computational geometry, 1994, S. 390-398

[38] Pellegrini, M., P. Shor: „*Finding stabbing lines in 3-dimensional space*“, Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms, 1991, S. 24-31

[39] Pellegrini, M.: „*Rendering equation revisited: how to avoid explicit visibility computations*“, Proceedings of the 10th annual ACM-SIAM symposium on Discrete algorithms, 1999, S. 725-733

[40] Pellegrini, M.: „*Stabbing and ray shooting in 3-dimensional space*“, Proceedings off the 6th annual ACM Symposium on Computational Geometry, 1990, S. 177–186

[41] Pocchiola, M., G. Vegter: „*The visibility complex*“, Proceedings of the 9th annual symposium on Computational Geometry, 1993, S. 328-337

[42] Samet, H.: „*Applications of Spatial Data Structures*“, Addison Wesley Publishing Corp., 1990 (ISBN: 0-201-5030-5)

[43] Samet, H.: „*The Design and Analysis of Spatial Data Structures*“, Addison Wesley Publishing Corp., 1990 (ISBN: 0-201-50255-5)

[44] Schauffler, G., J. Dorsey, X. Decoret, F. Sillion: „*Conservative volumeric visibility with occluder fusion*“, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000, S.229-238

[45] Sillion, F., C. Puech, „*Radiosity & Global Illumination*“, Morgan Kaufmann Publishers Inc., 1994 (ISBN: 1-55860-277-1)

[46] Sillion, F., G. Drettakis: „*Feature-based control of visibility error: a multi-resolution clustering algorithm for global illumination*“, Proceedings of the 22nd annual conference on Computer Graphics, 1995, S. 145-152

[47] Smits, B., J. Arvo, D. Greenberg: „*A clustering algorithm für radiosity in complex environments*“, SIGGRAPH 94 Conference Proceedings, S. 435-442

[48] Soler, C., F. Sillion: „*Texture-based visibility for efficient lighting simulation*“, ACM Transactions on Graphics, Volume 19, Issue 4, 2000, S. 302-342

[49] Sowizral, H., K. Rushforth, M. Deering: „*The Java3D API Specification*“, Addison Wesley Publishing Corp., 1998 (ISBN: 0-201-32576-4)

[50] Stewart, A. J., S. Ghali: „*Fast computation of shadow boundaries using spatial coherence and backprojections*“, SIGGRAPH 94 Conference Proceedings, 1994, S. 231-238.

[51] Suri, S., P. M. Hubbard, J. F. Hughes: „*Analyzing bounding boxes for object intersection*“, ACM Transactions on Graphics, Volume 18 , Issue 3, 1999, S. 257-277

[52] Tan, T., K. Chong, K. Low: „*Computing bounding volume hierarchies using model simplification*“, Proceedings of the 1999 symposium on Interactive 3D graphics, 1999, S. 63-69

[53] S. Coorg, S. Teller: „*Real-time occlusion culling for models with large occluders*“, Proceedings of the 1997 symposium on Interactive 3D graphics, 1997, S. 83-97

[54] Teller, S. J., M. E. Hohmeyer: „*Computing the lines piercing four lines“,* Technical Report UCB/CSD 91/665, Computer Science Department, U.C. Berkeley, 1991

[55] Teller, S. J.: „*Computing the antipenumbra of an area light source*“, Proceedings of the 19th annual conference on Computer Graphics, 1992, S. 139-148

[56] Teller, S., P. Hanrahan: „*Global visibility algorithms for illumination computations*“, Proceedings of the 20th annual conference on Computer Graphics, 1993, S. 239-246

[57] Teller, S.: „*Visibility computations in densely occluded polyhedral environments*“, PhD. Thesis, University of California at Berkeley, 1992

[58] Watt, A., M. Watt: „*Advanced Animation & Rendering Techniques*“, Addison Wesley Publishing Corp., 1992 (ISBN: 0-201-54412-1)

[59] Watt, Alan: „*3D-Computer Graphics (3rd Edition)*“, Addison Wesley Publishing Corp., 2000 (ISBN: 0-201-398559)

[60] Woo, A.: „*Fast Ray-Box-Intersection*“, Graphics Gems, Academic Press Inc., 1990 (ISBN: 0-12-286166), S. 395-396

[61] Yamaguchi, F., M. Niizeki: „*Some basic geometic test conditions in terms of Plücker coordinates and Plücker coefficients*“, The Visual Computer, Volume 13, Issue 1, 1997, S. 29-41

[62] Zhou, Y., S. Suri: „*Analysis of a bounding box heuristic for object intersection*“, Proceedings of the 10th annual ACM-SIAM symposium on Discrete algorithms, 1999, S. 830-839

7.4 Erklärung

Ich versichere, dass ich meine Dissertation

Globale Sichtbarkeitsalgorithmen

selbständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient habe.

Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen sonstigen Prüfungszwecken gedient.

Marburg, den 25.4.2003 _____

(Axel Schröder)

7.5 Lebenslauf

Axel Schröder
Am Wall 6
35041 Marburg-Michelbach

Geboren am 31.1.1973 in Eschwege, Konfession evangelisch.

Eltern: Ruth Schröder geb. Horn
Ulrich Schröder

Schulbildung:

1979-1983	Gerhart-Hauptmann Grundschule in Wanfried
1983-1989	Leuchtberg Gymnasium in Eschwege
1989-1992	Oberstufengymnasium in Eschwege

Sonstige Tätigkeiten:

1992-1993	Zivildienst bei der Diakonie in Wanfried

Studium:

1993- 1998	Informatik Studium mit NF Physik an der Philipps-Universität Marburg Abschluss als Diplom-Informatiker
1998- 2003	Anstellung als wissenschaftlicher Mitarbeiter am Fachbereich Mathematik und Informatik an der Philipps-Universität Marburg

Marburg, den 25.4.2003

Axel Schröder

7.6 Danksagung

Mein Dank gilt vor allem meinen Eltern und meiner Schwester, die mich während der gesamten Studien- und Promotionszeit immer unterstützt und mir diesen Lebensweg - auch in schwierigen Zeiten - ermöglicht haben.

Weiterhin möchte ich meinem Doktovater Prof. Dr. Manfred Sommer für die Betreuung der Arbeit und die gute Zusammenarbeit während meiner Anstellung am Fachbereich danken. Auch Prof. Dr. Wolfgang Hesse gilt mein Dank für die Zweitkorrektur der Dissertation und viele sinnvolle Anregungen zum Stil und Aufbau der Arbeit.

Schließlich möchte ich mich besonders bei meinen Kollegen am Fachbereich für die angenehme Arbeitsatmosphäre und die vielen fachlichen und privaten Gespräche und Diskussionen bedanken. Ausdrücklich genannt seien hier Alex, die beiden Barbaras, Elko, Jost, Lydia, Markus, Martin, Roman, Steffen und – beruflich wie privat - Yan.