Echtzeitfähige Kollisionserkennung für Virtual Reality Anwendungen

Dissertation

zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Technischen Fakultät
der Universität des Saarlandes

von

Jens Eckstein

Saarbrücken
1998

Tag des Kolloquiums:	24.02.1999
Dekan:	Prof. Dr. Wolfgang Paul
Gutachter:	Prof. Dr. Dr. h.c. mult. Günter Hotz
	Prof. Dr. Kurt Mehlhorn

Für meine Eltern

Inhaltsverzeichnis

Abbildungsverzeichnis

Kurzzusammenfassung

Die vorliegende Arbeit behandelt das Problem der Kollisionserkennung für *Virtual Reality* Anwendungen.

Der Schwerpunkt liegt auf einer durchgängigen *dynamischen* Kollisionserkennung unter Verwendung von Hüllkörperhierarchien, die einen Einsatz der Kollisionserkennung in geometrisch komplexen Szenarien unter Verwendung weitergehender Simulationen, z.B. Kontakt- oder Dynamiksimulation, ermöglichen. Bisher bekannte Verfahren zur Kollisionserkennung, die Hüllkörperhierarchien verwenden, wurden nur für *statische* Kollisionserkennung eingesetzt. Wir modifizieren zum einen bekannte Verfahren zur Berechnung von Hüllkörperhierarchien und entwickeln neue, zum anderen erarbeiten wir Verfahren zur Integration der Hüllkörperhierarchien in einen *dynamischen* Kollisionserkennungsprozeß. Eingebettet werden diese Verfahren in ein System zur *dynamischen* Kollisionserkennung unter Verwendung von Raumpartitionierung, Hüllkörperhierarchien und zeitkritischer Kollisionserkennung.

Die Qualität der Verfahren wird an praktischen Beispielen aus dem Bereich Ein-/Ausbauuntersuchungen in der Automobilentwicklung demonstriert.

Abstract

This thesis deals with the problem of collision detection for *Virtual Reality* applications.

It focuses on *dynamic* collision detection using bounding object hierarchies, which enables the collision detection to be used in geometric complex scenes together with further simulation systems, e.g. contact or dynamics simulation systems. Known techniques for collision detection using bounding object hierarchies are only used with *static* collision detection so far. We modify known algorithms for the computation of bounding object hierarchies und we develop new ones. Furthermore methods are developed to integrate bounding object hierarchies within a process of *dynamic* collision detection. Those methods together with space partitioning und time critical collision detection are combined to a system for *dynamic* collision detection.

The performance of the methods is shown using examples for fitting simulations from automotive industry.

Ausführliche Zusammenfassung

Virtual Reality wird als eine Schlüsseltechnologie angesehen, die kürzere Entwicklungszeiten, eine Fehlerreduktion in der Planungsphase, geringe Produktionskosten und eine verbesserte Produktqualität ermöglichen kann. Ein unmittelbares Anwendungsfeld für *Virtual Reality* Anwendungen besteht im Automobilbau im Bereich des *Packagings* während des Produktentwicklungsprozesses. Wichtige Fragestellungen in diesem Problemkreis beschäftigen sich mit Einbau- und Wartungsuntersuchungen z.B. im Motorraum von Fahrzeugen. *Virtual Reality* bietet die Möglichkeit, im Rahmen des *Virtual Prototypings*, d.h. der Produktentwicklung mit virtuellen Prototypen, diese Untersuchungen realitätsnah durchführen zu können. Voraussetzung dafür ist, daß die *Virtual Reality* Anwendungen das reale Objektverhalten in den virtuellen Szenen echtzeitnah widerspiegeln. Dazu gehören das Erkennen von Kollisionen zwischen Objekten und deren physikalisch korrekten Modellierungen, z.B. das Entlanggleiten von Objekten aneinander unter Berücksichtigung von Reibungseffekten.

Die vorliegende Arbeit beschäftigt sich mit dem Problem der Kollisionserkennung für *Virtual Reality* Anwendungen. Wir betrachten Einbau- und Wartungsuntersuchungen im Produktentwicklungsprozeß unter Verwendung des *Virtual Prototypings* als Anwendungshintergrund. Die Objekte sind *unstrukturierte Flächenmengen*, die bei der Konvertierung von CAD-Daten in Oberflächendaten entstehen. Die Kollisionserkennung muß dabei eine *große Anzahl* von Objekten mit jeweils einer *großen geometrischen Komplexität* handhaben. Dabei werden in jedem Zeitschritt *mehrere Objekte* gleichzeitig mit Hilfe *nicht vorspezifizierter Bewegungen* manipuliert. Die Kollisionserkennungsergebnisse sollen für weitergehende Simulationen geeignet sein, wie z.B. Kontakt- oder Dynamiksimulation, wodurch neben den *statischen* auch *dynamische* Kollisionserkennungsergebnisse (mit Bestimmung von Kollisionszeitpunkt und beteiligten Objektteilen) berechnet werden müssen. Die Kollisionserkennung ist darüber hinaus *echtzeitfähig* in ein *visualisierungsgesteuertes System* einzugliedern.

Zur Beherrschung vieler Objekte wird ein Raumpartitionierungsverfahren vorgestellt, das auf dem *one-dimensional Sort-and-Sweep* von Lin für *statische* Kollisionserkennung basiert und für den Einsatz in einem *dynamischen* Kollisionserkennungsprozeß erweitert wird.

Zur Beherrschung der geometrischen Komplexität eines Objektpaartests werden Hüllkörperhierarchien eingesetzt. Kollisionserkennung unter Verwendung von Hüllkörperhierarchien stellt derzeit die mächtigste Klasse von Kollisionserkennungsverfahren dar, die bisher jedoch nur zur *statischen* Kollisionserkennung verwendet wurde. Wir betrachten bei diesen sowohl die Verfahren zu deren Berechnung als auch die zu ihrer Verwendung, wobei insbesondere die Integration von Hüllkörperhierarchien in einen *dynamischen* Kollisionserkennungsprozeß vorgestellt wird.

Bei der Berechnung modifizieren wir unter anderem bekannte Verfahren von Gottschalk und Zachmann, so daß sie neben dem bereits vorgeschlagenen Verzweigungsgrad 2 auch Hierarchien berechnen können, die Verzweigungsgrad 3 bis 6 oder flexible Verzweigungsgrade haben. Darüber hinaus entwickeln wir Optimierungskriterien für die Berechnung von Hüllkörperhierarchien, auf deren Grundlage wir zwei neue Heuristiken zur top-down Berechnung von Hüllkörperhierarchien vorstellen. Diese können Hierarchien mit Verzweigungsgraden 2 bis 6 oder flexiblen Verzweigungsgraden berechnen, wobei als Hüllkörper wahlweise Kugeln, Iso-Boxen, Boxen oder alle Hüllkörpertypen gemeinsam benutzt werden können.

Bei der Verwendung von Hüllkörperhierarchien betrachten wir sowohl die einzelnen Hüllkörpertests als auch verschiedene Hierarchietraversierungsstrategien.

Bei den Hüllkörpertests stellen wir zum einen die bekannten Verfahren zur *statischen* Kollisionserkennung zwischen den einfachen Hüllkörpertypen Kugel, Iso-Box und Box vor und erweitern diese zu *dynamischen* Hüllkörpertests, wobei mit Hilfe der gegebenen Bewegung und des Hüllkörpers eine Approximation des überstrichenen Volumens in Form eines Hüllkörpers berechnet wird, der selbst wieder in einem *statischen* Hüllkörpertest eingesetzt werden kann. Dadurch wird der Einsatz von Hüllkörpern in einem *dynamischen* Kollisionserkennungsprozeß ermöglicht.

Bei den Hierarchietraversierungsstrategien, die die Anzahl der durchzuführenden Hüllkörpertests in einem Kollisionserkennungsschritt gering halten sollen, stellen wir die bekannten Verfahren der einseitigen und der simultanen Expansion vor und kombinieren diese zu einer hybriden Hierarchietraversierungsstrategie. Für die Basiskollisionserkennung zwischen Flächenpaaren werden zum einen die bekannten Verfahren der *statischen* Kollisionserkennung vorgestellt und darüber hinaus verschiedene Verfahren zur *dynamischen* Basiskollisionserkennung erarbeitet. Diese variieren in der Genauigkeit der Approximation eines Kollisionszeitpunktes und dem dafür verwendeten Berechnungsaufwand.

Die Verfahren zur Beherrschung vieler und geometrisch komplexer Objekte werden in einem Verfahren zur Kontrolle eines Kollisionserkennungsschrittes zusammengefaßt, das mit Hilfe von Methoden der zeitkritischen Kollisionserkennung echtzeitfähig in ein *Virtual Reality* System eingegliedert werden kann.

Die beschriebenen Verfahren zur Kollisionserkennung sind als Modul innerhalb der *Virtual Reality* Softwareplattform **DBView** des Virtual Reality Competence Centers der Daimler-Benz AG realisiert und stellen die Basis für weitergehende Simulationen dar, wie Kontakt- oder Dynamiksimulation. Mit dieser Realisierung wird zum einen das Laufzeitverhalten der verschiedenen Verfahren anhand praktischer Beispiele aus dem Bereich Ein-/Ausbauuntersuchung von Bauteilen im Motorraum während der Fahrzeugentwicklung untersucht und zum anderen werden Beispiele für das erfolgreiche Zusammenspiel von *dynamischer* Kollisionserkennung mit Kontakt- und Dynamiksimulation präsentiert.

Kapitel 1

Einleitung

Virtual Reality wird heute als eine Schlüsseltechnologie angesehen. Der US-Vizepräsident Gore bezeichnete *Virtual Reality* gar als einen der entscheidenden Wachstumsmotoren der US-Wirtschaft (vgl. [BB94]). Der ehemalige Forschungsvorstand der Daimler-Benz AG Weule charakterisierte die Bedeutung von *Virtual Reality* für den Konzern wie folgt:

> Als weltweit agierender Hersteller technisch anspruchsvoller Geräte kann Daimler-Benz im Wettlauf um kürzere Entwicklungszeiten, um Fehlerreduktion in der Planungsphase, um geringere Produktionskosten und verbesserte Produktqualität auf diese Schlüsseltechnologie nicht verzichten.

Folgerichtig eröffnete die Daimler-Benz AG in Ulm ein *Virtual Reality Competence Center*, um darin konsequent die Möglichkeiten von *Virtual Reality* für den Gesamtkonzern zu entwickeln und deren Verbreitung innerhalb der verschiedenen Unternehmensbereiche zu forcieren. Andere Unternehmen, allen voran die Unternehmen der Luft-, Raumfahrt- und Automobilindustrie, entdecken *Virtual Reality* als eine der Technologien, die es ihnen erlauben, im immer härter werdenden internationalen Wettbewerb nicht nur zu bestehen, sondern Wettbewerbsvorteile gegenüber Konkurrenten zu erarbeiten. Unter diesem Gesichtspunkt ist es nicht verwunderlich, daß in diesem Bereich derzeit größere Anstrengungen unternommen werden, um *Virtual Reality* im Hinblick auf industrielle Anwendungen weiterzuentwickeln.

1.1 Was ist *Virtual Reality*?

Was macht nun *Virtual Reality* aus, was sind deren potentielle Anwendungsfelder und die daraus resultierenden wissenschaftlichen Herausforderungen?

Da der Begriff der *Virtual Reality* [1] in inflationärem Maße gebraucht wird, müssen wir zunächst abgrenzen, was wir unter *Virtual Reality* verstehen. Eine gute Beschreibung der Säulen von *Virtual Reality* erscheint uns die von Burdea und Coiffet [BC94]. Danach ist *Virtual Reality* durch das Vorhandensein von drei wichtigen Komponenten bestimmt:

1. 3D-Echtzeitvisualisierung
2. Interaktion
3. Immersion

Die *3D-Echtzeitvisualisierung* grenzt *Virtual Reality* von anderen Gebieten der Computergraphik ab, bei denen z.B. die Hochqualitätsvisualisierung im Vordergrund steht, wie sie im Zusammenhang mit der Produktion von Filmen benötigt wird. *3D-Echtzeitvisualisierung* ist notwendig, um *Interaktionen* des Anwenders zuzulassen. Mögliche *Interaktionen* sind dabei das Navigieren des Benutzers durch die virtuelle Welt, bei der abhängig vom aktuellen Blickwinkel des Betrachters die Szene visualisiert werden muß oder die Manipulation von Objekten in der virtuellen Welt. *Immersion* schließlich bedeutet, daß der Anwender die virtuelle Welt als real erlebt, daß er zu einem gewissen Teil in diese virtuelle Welt versetzt wird.

1.1.1 3D-Echtzeitvisualisierung

Eine leistungsfähige *3D-Echtzeitvisualisierung* ist die Grundvoraussetzung für den sinnvollen Einsatz von *Virtual Reality*. Daher war die Entwicklung von Algorithmen und unterstützender Graphikhardware für die *3D-Echtzeitvisualisierung* eine der Hauptaufgabengebiete der ersten Hälfte der 90er Jahre. Die Entwicklung ist mit dem heutigen Stand nicht abgeschlossen. Ganz im Gegenteil: Die Entwicklung immer mächtigerer Verfahren zur *3D-Echtzeitvisualisierung* wird ein immerwährendes Problem bleiben, da die Anforderungen an die Qualität der Visualisierung ständig steigen. Jedoch ist die Leistungsfähigkeit heutiger Visualisierungssysteme auf einem Stand angelangt, bei dem der praktische Einsatz von *Virtual Reality* im Produktentwicklungsprozeß möglich wird. Die Tendenz im Bereich der *3D-Echtzeitvisualisierung* geht derzeit in zwei unterschiedliche Richtungen. Zum einen wird die Leistungsfähigkeit bestehender Systeme weiter erhöht, zum anderen wird die Verfügbarkeit von geeigneter Graphikhardware für eine große Palette von Computersystemen (PC, Spielekonsolen usw.) forciert. Dies führt dazu, daß sich die Preise für leistungsfähige Visualisierungssysteme drastisch reduzieren und einen großflächigen Einsatz von *Virtual Reality* insbesondere auch im industriellen Bereich ermöglichen.

[1] Geprägt wurde der Begriff 1989 von Jaron Lanier. Alternativ dazu wird heute in der wissenschaftlichen Welt der Begriff *Virtual Environment* verwendet.

1.1.2 Interaktion

Interaktion erlaubt es dem Benutzer, sich durch die virtuelle Welt zu bewegen (*fly-through*) und Objekte zu manipulieren. Dies kann mit Hilfe von Desktop-Geräten erfolgen wie z.B. der Computermouse oder der SPACE MOUSE® , die zusammen mit geringer *Immersion* benutzt werden. Im Zusammenhang mit einem höheren Immersionsgrad wird die *Interaktion* des Benutzers mit der virtuellen Welt komplexer. Bewegungen des Benutzers in virtuellen Welten benötigen *Trackingsysteme*, die die Position des Benutzers z.B. die Kopfposition, die Position einer Hand oder eines ganzen Armes verfolgen. Neben der Positionsbestimmung werden zusätzliche Geräte benötigt, mit Hilfe derer *Interaktionen* durchgeführt werden können. Als Beispiel sei hier der Datenhandschuh erwähnt, der benutzt wird, um die Hand des Benutzers in der virtuellen Welt abzubilden. Dabei bildet die reine Hardware nur das Vehikel, um *Interaktionen* innerhalb von virtuellen Welten zu ermöglichen. Die Realitätsnähe der *Interaktion* ist neben dem Einsatz des für die Anwendung richtigen Interaktionsgerätes wesentlich von der softwaretechnischen Realisierung abhängig.

1.1.3 Immersion

Das bekannteste Beispiel für die *Immersion* ist der Einsatz eines *Head Mounted Displays (HMD)*, bei dem der Anwender visuell völlig von der realen Welt abgeschnitten wird und in die virtuelle Welt eintaucht. Die Nachteile dieser Technik sind ebenfalls bekannt. Die derzeit verfügbaren HMDs sind recht unhandlich, die virtuelle Welt ist nur für einen Anwender erlebbar und die Symptome der *Simulatorkrankheit* tauchen häufig auf. Ein Immersionseffekt kann aber auch auf andere Art und Weise erzielt werden. Beispiele hierfür sind stereoskopische Visualisierungen mit Hilfe von Rundumprojektionen, Hinterwandprojektionen oder CAVE™s (CAVE Automatic Virtual Environment). Die *Immersion* hängt neben der Art der Visualisierung auch von den eingesetzten Interaktionsgeräten ab. Hierbei bietet der Datenhandschuh ein großes Potential für eine natürliche *Interaktion*.

Ein umfassender Gesamteinblick in die Technologie und die Anwendungsgebiete von *Virtual Reality* wird in [DM94] gegeben. Da sich jedoch die Technologie und dadurch auch die Anwendungsfelder von *Virtual Reality* geradezu stürmisch weiterentwickeln, kann jede Studie nur einen Einblick in den jeweiligen Stand der Technik und der Anwendungen geben. Literatur, die einen Überblick über den aktuellen Stand der Technologie und der Anwendungen von *Virtual Reality* gibt, entsteht derzeit in größerem Umfang. In diesem Zusammenhang sei auf [He97] verwiesen.

1.2 Anwendungsgebiete von *Virtual Reality*

Mögliche Anwendungsgebiete für *Virtual Reality* sind:

- Training
- Marketing
- Unterhaltung
- Produktentwicklung

1.2.1 Training

Trainingssimulatoren gibt es schon seit längerer Zeit. Man denke nur an die Flugsimulatoren für die Pilotenaus- und weiterbildung oder die Fahrsimulatoren wie der der Daimler-Benz AG in Berlin. Der Nutzen von Simulatoren zur Aus- und Weiterbildung ist in vielen Bereichen unbestritten, erlauben sie es doch, Ausbildungsszenarien durchzuführen, die in Realität entweder nicht oder nur mit großem Aufwand realisiert werden können. Mit Hilfe von *Virtual Reality* Techniken ist es möglich, die Realitätsnähe der Simulatoren zu verbessern. Darüber hinaus können aufgrund der sinkenden Kosten für *Virtual Reality* Systeme Simulatoren für Bereiche entwickelt werden, in denen bisher keine eingesetzt werden. Als Beispiel sei hier die Fahrschulausbildung für Pkw und Lkw erwähnt.

1.2.2 Marketing

Auch im Marketing gibt es weitverbreitete Anwendungsmöglichkeiten für *Virtual Reality*. Ein mögliches Einsatzgebiet ist das Vermarkten von Produkten, die eigens für einen Kunden konfiguriert werden. Der Verkauf von Flugzeugen ist an dieser Stelle ein gutes Beispiel. Ein Flugzeug wird nach Kundenwünschen im Rahmen des technisch Machbaren konfiguriert. Die Kundenzufriedenheit hängt wesentlich von der Übereinstimmung der Erwartung des Kunden bzgl. des erworbenen Produktes mit den Eigenschaften des tatsächlich erhaltenen Produktes zusammen. Ein Mittel, die Kundenzufriedenheit zu erhöhen, ist die verbesserte Darstellung der Produkte. Mit Hilfe von *Virtual Reality* Techniken ist der Kunde in der Lage, sich einen besseren Einblick in ein Produkt zu verschaffen, so daß seine Erwartungshaltung besser mit dem tatsächlichen Produkt übereinstimmt.

Gilt dies offensichtlich im starken Maße für große und weitreichende Investitionsentscheidungen wie der Kauf eines Flugzeuges, so ist dies auch im privaten Bereich, z.B. beim Kauf eines Pkws nicht zu unterschätzen. Auch für einen Privatmann stellt der Kauf eines Fahrzeuges eine große und weitreichende Entscheidung dar. Die Kundenzufriedenheit hängt auch hierbei neben der Qualität des Produktes und dem Service nach dessen Kauf wesentlich von der Übereinstimmung der Kundenerwartungen mit dem tatsächlich erhaltenen Fahrzeug ab. Dabei kann eine Kundenunzufriedenheit zustande kommen, die scheinbar nicht im Verschulden des Fahrzeugherstellers liegt. Denn wählt sich ein Kunde eine spezielle Variante eines Fahrzeuges aus und liefert der Hersteller dieses Fahrzeug dem Kundenwunsch entsprechend in höchster Qualität, kann dennoch eine Unzufriedenheit beim Kunden auftreten, weil er sich das von ihm erworbene Produkt nicht derart vorgestellt hat. Auch wenn den Hersteller keine Schuld zu treffen scheint, wird der Kunde bei zukünftigen Fahrzeugkäufen eventuell einen anderen Hersteller bevorzugen. Um den Grad der Kundenzufriedenheit zu erhöhen, müssen beim Fahrzeugverkauf Methoden eingesetzt werden, die die Erwartungshaltung des Kunden mit den Eigenschaften des tatsächlichen Produktes in Deckung bringt. *Virtual Reality* Techniken bieten dabei eine breite Palette von Unterstützungsmöglichkeiten. Zum einen kann der Kunde das von ihm konfigurierte Fahrzeug von außen und innen begutachten, so daß das spätere Aussehen mit den Vorstellungen des Kunden übereinstimmt. Zum anderen können darüber hinausgehende Eindrücke vermittelt werden wie z.B. Fahreigenschaften.

Ein weiterer Bereich des Marketings, der in die Richtung der Unterhaltung tendiert, ist das Einbringen neuer Erlebnisse in den Kaufvorgang. Ist die gute Beurteilungsmöglichkeit des Produktes, das real nicht verfügbar ist, ein Kriterium für den Einsatz von *Virtual Reality* im Marketingbereich, so kann die Technik zusätzlich eine Komponente darstellen, die dem Wandel des Kaufverhaltens der Kunden weg vom reinen Nutzenkauf hin zum Erlebniskauf gerecht wird.

1.2.3 Unterhaltung

Derzeit übernimmt auch die Unterhaltungsindustrie eine führende Rolle bei der Entwicklung von *Virtual Reality* Technologien, was aufgrund der massenhaften Verbreitungsmöglichkeiten zu Veränderungen bei der Entwicklung von VR-Komponenten geführt hat. Wurden in den letzten Jahren vorallem die High-End Komponenten für den industriellen Einsatz entwickelt, wendet sich die Aufmerksamkeit derzeit der Entwicklung von VR-Komponenten zu, die für den Einsatz im privaten oder industriellen Umfeld mit geringen Kosten geeignet sind.

Vorstufen von *Virtual Reality* gibt es seit einiger Zeit in den Vergnügungs- und Themenparks zu erleben, bei denen die Zuschauer z.B. einen virtuellen Flug durch das All erleben können, wobei neben einer Großprojektion eine Bewegungsplattform eingesetzt wird, um den Erlebnischarakter für den Zuschauer zu verbessern. Bei diesen Systemen fehlt jedoch gänzlich die Interaktionskomponente. Computerspiele in Spielhallen profitieren derzeit stark von den Entwicklungen im *Virtual Reality* Bereich. Sind die Investitionen für VR-Konsolen derzeit auch nicht unerheblich, so wird doch durch die Möglichkeiten von *Virtual Reality* ein deutlich verbessertes Spielerlebnis erzeugt.

Ein weiterer wichtiger Bereich für den Einsatz von VR-Komponenten stellt der private Spielemarkt dar. Derzeit werden strategische Kooperationen geknüpft, um bei der Erorberung des privaten Marktes durch VR-Technologien erfolgreich sein zu können. Eine weite Verbreitung von VR-Techniken im privaten Sektor kann auch an anderen Stellen hilfreich sein für eine positive Entwicklung im Akzeptanzverhalten der Anwender gegenüber dem Einsatz von *Virtual Reality* Techniken.

1.2.4 Produktentwicklung

Um am Markt erfolgreich zu sein, ist die Industrie gezwungen, in immer kürzerer Zeit immer komplexere Produkte zu entwickeln. Ein Beispiel dafür ist die Entwicklung in der Automobilindustrie. Nach STEGER [St95] hat sich der Produktlebenszyklus eines Autos in den letzten zehn Jahren um 12,5 % verkürzt. Die Unternehmen müssen daher neue Methoden zur Produktentwicklung erarbeiten, die es ihnen ermöglichen, mit dieser Entwicklung Schritt zu halten. Eine Maßnahme war die Einführung des *Concurrent Engineerings*, wodurch der bisher sequentiell ablaufende Produktentwicklungsprozeß in größeren Teilen parallelisiert wurde. Diese Vorgehensweise erfordert ein leistungsfähiges Projektmanagement, das eine flexible Zeitplanung ermöglicht. Ein weiterer Schritt in Richtung auf die Verkürzung der Entwicklungszeiten ist das *Rapid Prototyping*. Bei diesem Verfahren wird von Anfang an ein Prototyp bearbeitet. Dieser entwickelt sich während des gesamten Produktentstehungsprozeß von einem ersten Entwurf bis hin zum serienreifen Produkt. Der Vorteil bei dieser Vorgehensweise ist, daß in jeder Stufe der Entwicklung das Produkt als Ganzes sichtbar ist. Die Methode des *Virtual Prototyping* erweitert den Ansatz des *Rapid Prototyping*, indem der Prototyp, der während des Produktentstehungsprozesses stetig weiterentwickelt wird, kein real existierender Prototyp ist, sondern ein virtueller, d.h. er existiert nur in Form von Daten. Diese Form der Repräsentation erlaubt es, den Prototypen möglichst schnell dem aktuellen Stand der Produktentwicklung anzupassen. Um die Entwicklung eines Produktes anhand eines virtuellen Prototypen durchführen zu können, benötigt man zum einen mächtige Werkzeuge zur Konstruktion auch komplexer Bauteile und zum anderen Werkzeuge, die Untersuchungen an diesem virtuellen Prototypen durchführen können.

Je früher ein Fehler während des Produktentwicklungsprozesses gefunden wird, desto einfacher ist er zu beheben. Fehler, die erst in späten Phasen entdeckt werden, sind entweder nur mit hohem Aufwand oder ohne Redesign großer Teile des Produktes überhaupt nicht mehr zu beheben. In jedem Fall wird die Rentabilität des Produktes nachhaltig negativ beeinflußt. Um die Entwicklungszeiten für immer komplexere Produkte dennoch verkürzen zu können, müssen möglichst viele Untersuchungen in frühen Phasen des Produktentwicklungsprozesses durchgeführt werden, die in der Lage sind, Mängel des Produktes zu offenbaren, die bisher erst in späteren Phasen entdeckt werden können. *Virtual Reality* stellt bei der Durchführung solcher Untersuchungen ein Werkzeug mit großem Entwicklungspotential dar.

Packaging

Eine wichtige Aufgabe im Produktentwicklungsprozeß ist das *Packaging*. Der Begriff des *Packaging* hat ebensowenig wie *Virtual Reality* derzeit eine allgemein anerkannte Definition. Unter *Packaging* verstehen wir den Prozeß, der sich mit der Untersuchung der Anordnung von Komponenten in einem Produkt beschäftigt. Dabei beschränkt sich das Problem nicht auf die reine Gruppierbarkeit von Komponenten, indem betrachtet wird, wie Teile geometrisch oder logisch – z.B. daß sich heiße Teile nicht in der Nähe von hitzeempfindlichen Teilen befinden dürfen – positioniert werden können. Eine derartige Beschränkung auf statische Betrachtungen der Anordnung von Komponenten würde wichtige Teile des Produktentwicklungsprozesses außer acht lassen. Dieser umfaßt darüber hinausgehende Fragestellungen wie die des Zusammenbaus des Produktes und dessen Wartung. *Virtual Reality* bietet dabei den Rahmen, um solche Einbau- oder Wartungsuntersuchungen durchzuführen. Ziel der Einbindung von *Virtual Reality* Techniken in den *Packagingprozeß* ist es somit, ausgehend von den aktuellen Konstruktionsdaten, derartige Untersuchungen möglichst effizient durchführen zu können.

1.3 Kollisionserkennung in *Virtual Reality*

Welche Rolle spielt nun die Kollisionserkennung innerhalb von *Virtual Reality* Anwendungen?

Die *3D-Echtzeitvisualisierung* und die verschiedenen Hardwaregeräte zur Steigerung des *Immersionsgrades* und zur Unterstützung der *Interaktion* stellen das Potential für die Ausgestaltung anwendungsspezifischer Anforderungen in *Virtual Reality* Anwendungen dar. Manipuliert der Benutzer ein Objekt in einer virtuellen Welt z.B. mit Hilfe einer SPACE MOUSE® , werden die jeweils aktuellen Positionen des Objektes aus den Eingabedaten der SPACE MOUSE® berechnet. Ohne weitere Maßnahmen wird das Objekt frei durch die virtuelle Welt fliegen und dabei andere Objekte durchdringen. Ziel aller industriellen *Virtual Reality* Anwendungen ist es jedoch, das Verhalten der realen Welt auf die virtuelle Welt abzubilden. Eine fundamentale Eigenschaft realer Objekte ist es, daß sie sich gegenseitig nicht durchdringen können (Paradigma der *Solidität* der Objekte). Um Anwendungen in virtuellen Welten durchführen zu können, bei denen reales Objektverhalten auf die virtuelle Welt abgebildet werden soll, wie das z.B. bei Einbau- oder Wartungsuntersuchungen der Fall ist, wird als eine erste wichtige Objekteigenschaft die Solidität benötigt, d.h. daß Objekte sich gegenseitig nicht durchdringen können. Um diese fundamentale Eigenschaft zu gewährleisten, benötigen *Virtual Reality* Anwendungen eine leistungsfähige Kollisionserkennungskomponente, die Durchdringungen von Objekten echtzeitfähig erkennt. Echtzeitfähigkeit ist wichtig, da *Virtual Reality* Anwendungen von der *3D-Echtzeitvisualisierung* ausgehend gesteuert werden und alle Aktivitäten, die sich darum gruppieren, wie z.B. das Handling der verschiedenen VR-Hardwaregeräte (Trackingsysteme, Datenhandschuh usw.) oder Simulationen (Kollisionserkennung, Animationen usw.), sich in diese Steuerung einfügen müssen. Das Ergebnis der Kollisionserkennung dient dabei als Eingabe für weitergehende Simulationen, wie z.B. die Simulation dynamischer Eigenschaften von Objekten oder die Simulation des natürlichen Greifens von Objekten mit Hilfe eines Datenhandschuhs.

Eine echtzeitfähige Kollisionserkennung stellt daher die Basis für eine realitätsnahe Simulation von Objektverhalten in *Virtual Reality* Anwendungen dar, deren Ergebnisse als

Eingabe für weitergehende Simulationen zur weiteren Steigerung der Realitätsnähe des Objektverhaltens genutzt werden können.

1.4 Gliederung und Resultate der Arbeit

Gliederung

In Kapitel 2 werden wir zunächst die Anforderungen an eine echtzeitfähige Kollisionserkennung in *Virtual Reality* Anwendungen erarbeiten und einen Überblick über andere Arbeiten in diesem Bereich geben. In Abschnitt 2.3 wird ein Überblick über die Gliederung der Lösungsansätze gegeben, die die Anforderungen von *Virtual Reality* Anwendungen an eine Kollisionserkennung erfüllen. Die Lösungen zum Erfüllen dieser Anforderungen werden in den Kapiteln 3 bis 8 erarbeitet und deren Realisierung in Kapitel 9 beschrieben. Der Einsatz des enstandenen Kollisionserkennungssystems im Vergleich der verschiedenen Hüllkörperhierarchieverfahren und im Zusammenhang mit *Kontaktsimulation* und *Dynamiksimulation* werden in Kapitel 10 dargestellt. In Kapitel 11 werden die vorgestellten Verfahren zusammengefaßt und ein Ausblick für weitere Arbeiten im Bereich der Kollisionserkennung gegeben.

Resultate

In dieser Arbeit werden Verfahren vorgestellt, die in der Lage sind, das Kollisionserkennungsproblem innerhalb von *Virtual Reality* Anwendungen zu lösen. Dabei sind die besonderen Gegebenheiten zu berücksichtigen, daß die betrachteten Objekte Flächenmengen sind, von denen sich in einem Kollisionserkennungsschritt beliebig viele bewegen können, die dabei auftretenden Bewegungen nicht durch maximale Geschwindigkeit oder Beschleunigung der Objekte beschränkt sind und die Berechnung echtzeitfähig sein muß. Darüber hinaus werden die Ergebnisse als Eingabe für weitergehende Simulationen wie *Kontaktsimulation* oder *Dynamiksimulation* verwendet, die *dynamische* Kollisionserkennungsergebnisse benötigen.

Zentraler Gegenstand ist dabei die Klasse der Verfahren, die Hüllkörperhierarchien zum Objektpaartest verwenden. Diese Verfahren zur schnellen Durchführung von Objektpaartests werden abgerundet durch Verfahren zur Beherrschung vieler Objekte in einer Szene mittels Raumpartitionierung (Kapitel 6) und zur zeitkritischen Kollisionserkennung (Kapitel 8), die auf der Steuerung der Kollisionserkennung (Kapitel 7) aufsetzen.

Für Berechnung der Hüllkörperhierarchien werden in Kapitel 3 Optimierungskriterien entwickelt, an denen ausgerichtet zum einen eigene Verfahren zur Berechnung von Hüllkörperhierarchien erstellt und zum anderen bekannte Verfahren weiterentwickelt werden. Die Performance dieser Verfahren werden in Kapitel 10 anhand von Beispielen aus dem Bereich der Ein-/Ausbauuntersuchungen im Automobilbau miteinander verglichen.

Kollisionserkennung mit Hüllkörperhierarchien wird bisher ausschließlich im Zusammenhang mit *statischer* und *pseudo-dynamischer* Kollisionserkennung verwendet. Weitergehende Simulationen wie *Kontaktsimulation* und *Dynamiksimulation* benötigen als Ergebnis der Kollisionserkennung Kollisionszeitpunkte und dabei kollidierende Objektteile, wie sie als Ergebnis der *dynamischen* Kollisionserkennung zur Verfügung stehen. Wir machen die Klasse aller Kollisionserkennungsverfahren, die auf Hüllkörperhierarchien beruhen, für die *dynamische* Kollisionserkennung zugänglich, indem wir zum einen für die gängigen Hüllkörpertypen (Box, Iso-Box, Kugel) basierend auf den bekannten effizienten *statischen* Hüllkörpertests *dynamische* entwickeln (Kapitel 4) und zum anderen *dynamische* Basiskollisionstests zwischen Objektteilen (Ecke, Kante, Fläche) entwerfen (Kapitel 5), die ohne die Beschränkung von Geschwindigkeit und Beschleunigung der beteiligten Objekte auskommen.

In das Konzept der *dynamischen* Kollisionserkennung mit Hüllkörperhierarchien wird auch die Raumpartitionierung eingebunden (Kapitel 6), so daß ein gesamtes Kollisionserkennungssystem zur *statischen*, *pseudo-dynamischen* und *dynamischen* Kollisionserkennung entsteht, das sowohl für Polyeder als auch Flächenmengen quasi ohne Beschränkungen für betrachtete Bewegungen einsetzbar ist.

Danksagungen

Ich bedanke mich herzlichst bei Herrn Prof. Dr. Günter Hotz für die Betreuung dieser Arbeit und insbesondere für seine Bereitschaft, die Anfertigung in der Daimler-Benz Forschung zu unterstützen. Weiterhin danken möchte ich meinen Vorgesetzten bei Daimler-Benz, Herrn Dr. Franz May und Herrn Dr. Klaus Grebner, die jederzeit Vertrauen in meine Vorgehensweisen hatten und mir somit bei der Gestaltung der Arbeit freie Hand gelassen haben, sowie der Firma Daimler-Benz AG, die mir in ihrem Forschungszentrum in Ulm das Equipment zur Erstellung dieser Arbeit zur Verfügung gestellt hat.

Danken möchte ich auch Herrn Dr. Elmar Schömer für die hilfreichen Diskussionen und die daraus resultierenden Anregungen sowie seine Bereitschaft, die gesamte Arbeit sorgfältig Korrektur zu lesen. Herrn Jörg Sauer gebührt ein besonderer Dank. Mit ihm konnte ich jederzeit meine Ideen und Probleme diskutieren und die daraus resultierenden Anregungen haben die Inhalte dieser Arbeit nicht unwesentlich beeinflußt. Über die fast dreijährige partnerschaftliche Zusammenarbeit bei Daimler-Benz hinaus schätze ich ihn als Freund.

Ein Dank gilt darüber hinaus meinen Freunden, Kollegen und allen Leuten, die mir durch die Gestaltung des Umfelds geholfen haben, diese Arbeit zu erstellen.
Besonders hervorzuheben sind schließlich meine Eltern, die mich auch in diesem Lebensabschnitt uneingeschränkt unterstützt haben. Mit diesem Rückhalt haben sie ihren Anteil an der Entstehung dieser Arbeit.

Kapitel 2

Grundlagen

2.1 Anforderungen

Echtzeitfähige Kollisionserkennung stellt ein Schlüsselmodul für viele echtzeitnahe, interaktive Simulationsanwendungen dar. Mit Hilfe von Kollisionserkennung ist es möglich zu entscheiden, ob sich Objekte in einer Szene überlappen oder während der Bewegung von Objekten Kollisionen auftreten; damit ist es möglich, Durchdringungen von Objekten in virtuellen Szenarien zu verhindern. Diese wichtige Fähigkeit stellt die Basis für eine realitätsnahe Simulation von Objektverhalten in *Virtual Reality* Anwendungen dar, insbesondere auch im Zusammenhang mit weitergehenden Simulationen wie z.B. *Dynamiksimulation* [Sa99, SS98, Mi96] oder *interaktiver Objekthandhabung* [Bu98, BS98, Jo97].

Aus diesen Anwendungsfeldern heraus leiten sich die Anforderungen ab, die an die Kollisionserkennung gestellt werden. Für den Einsatz in *Virtual Reality* Anwendungen ist die *Effizienz* der Kollisionserkennung von sehr großer Bedeutung. Betrachten wir die Datenbasen von *Virtual Reality* Anwendungen, so müssen Szenarien mit Objekten *hoher geometrischer Komplexität* behandelt werden, die in *großer Anzahl* auftreten. Eine wichtige Quelle von Daten sind konvertierte CAD-Daten, z.B. aus Konstruktionen für Fahrzeuge oder Flugzeuge. Die dabei zum Einsatz kommenden Konverter erzeugen quasi ausschließlich *unstrukturierte Flächenmengen*, d.h. die erzeugten Objekte sind keine Polyeder (zur Definition siehe [PS85]), sondern eine Ansammlung von planaren Flächen. Diese können z.B. nichtzusammenhängend sein oder sich durchdringen. Die Bewegungen der Objekte resultieren aus Interaktionen mit dem System oder Ergebnissen von Simulationsberechnungen weitergehender Simulationen, wodurch in jedem Kollisionserkennungsschritt *nicht vorspezifizierte Bewegungen* für *mehrere bewegte Objekte* behandelt werden müssen. Darüber hinaus stellen weitergehende Simulationen Anforderungen an die Qualität des Ergebnisses der Kollisionserkennung, indem sie die Bestimmung des *Kollisionszeitpunktes* und der dabei kollidierenden *Objektteile* (Ecke, Kante, Fläche) benötigen. Diese Anforderungen müssen im Einklang mit einem *visualisierungsgesteuerten System*, wie es ein *Virtual Reality System* ist, realisiert werden. Strukturieren wir die Anforderungen an die Kollisionserkennung, ergeben sich die beiden Problemkreise

- **geometrisch komplexe Objekte** und
- **große Anzahl von Objekten**

unter Berücksichtigung der Rahmenbedigungen

- *unstrukturierte Flächenmengen*,
- *dynamische Kollisionserkennungsergebnisse*,
- *nicht vorspezifizierte Bewegungen*,
- *mehrere bewegte Objekte* und
- *Integration in ein* Virtual Reality *System und Echtzeitfähigkeit*

wie in Abbildung 2.1 graphisch veranschaulicht wird.

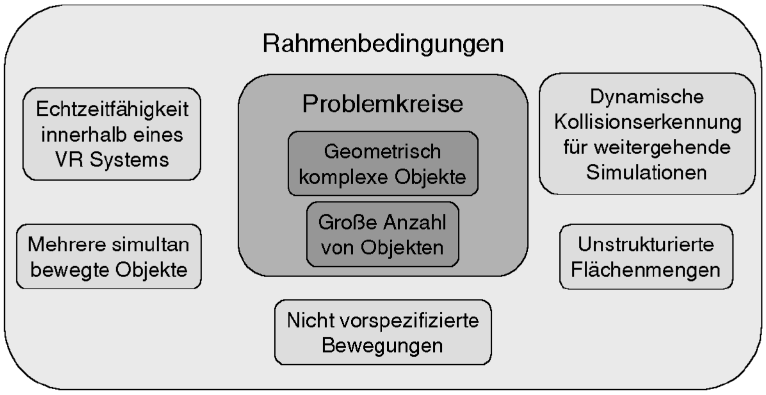

Abbildung 2.1: Anforderungsstruktur der Kollisionserkennung

2.2 Bisherige Arbeiten

2.2.1 Geometrisch komplexe Objekte

Der schwierigste Teil des Kollisionserkennungsprozesses liegt in der Beherrschung der *geometrischen Komplexität* der Objekte in der Szene. Nachfolgend geben wir einen Überblick über die bisher vorgeschlagenen Verfahren zur Behandlung *geometrisch komplexer* Objekte.

Eine Möglichkeit, Kollisionserkennung für konvexe Polyeder durchzuführen, ist die Verwendung des von Gilbert, Johnson und Keerthi in [GJK88] beschriebenen *simplexbasierten GJK*-Algorithmus. Dieser sucht aus dem *Minkowski-Differenz-Polyeder* der beiden betrachteten konvexen Polyeder den Simplex, der den Ursprung enthält oder am nächsten zu ihm liegt. Ist der Ursprung in diesem enthalten, überlappen sich die beiden Polyeder, sonst ist der Abstand dieses Simplex vom Ursprung der Abstand der beiden Polyeder. Rabbitz modifiziert in [He94] diese Vorgehensweise durch die Ausnutzung von Kohärenzeigenschaften. Chung Tat Leung verbindet in [CTL96] die Vorgehensweise von *I-Collide* [Li93] mit dem Verfahren von Rabbitz in der Kollisionserkennungsbibliothek *Q-Collide*. Darüber hinaus verwendet Cameron in [Ca97] den *GJK*-Algorithmus.

Eine weitere Möglichkeit der Kollisionserkennung für konvexe Polyeder besteht in der Verwendung *feature-basierter* Algorithmen. Die *Features* eines Polyeders sind dabei dessen *Ecken*, *Kanten* und *Flächen*. Baraff verwendet diese Methode in [Ba90]. Dabei wird für ein Paar von konvexen Polyedern eine trennende Fläche mitgeführt, die zu einer Fläche eines der Polyeder oder zu einem Paar von Kanten jeweils eines der Polyeder parallel ist. Eine solche Ebene existiert genau dann, wenn zwei konvexe Polyeder sich nicht überlappen. Lin beschreibt in [Li93] den als *closest-feature tracking* bekannten Algorithmus, der Teil der Kollisionserkennungsbibliothek *I-Collide* ist. Dieser berechnet für konvexe Polyeder deren Abstand. Dazu wird für jedes Polyeder das Voronoidiagramm der Umgebung berechnet, das den Raum in die Regionen unterteilt, die zu jeweils einem *Feature* des Polyeders den minimalen Abstand haben. Zwei *Features* definieren genau dann den minimalen Abstand zwischen den Polyedern, wenn diese gegenseitig in den zugehörigen Voronoiregionen liegen. Diese bilden das aktuelle *closest-feature pair*. Das *closest-feature tracking* benutzt zeitliche und räumliche Kohärenz, um das *closest-feature pair* während der Bewegungen der Polyeder jeweils neu zu berechnen. Ponamgi, Manocha und Lin erweitern in [PML95] den Ansatz von [Li93] auf nichtkonvexe Polyeder. Dazu wird zunächst auf die konvexe Hülle der Objekte eine Variante des *closest-feature trackings* angewandt, die auch überlappende Polyeder behandeln kann (der ursprüngliche Algorithmus kam in diesem Fall in eine Endlosschleife). Kollidieren die konvexen Hüllen in Teilen, die nicht zu den ursprünglichen Objekten gehören, so werden für diese konkaven Teile weitergehende Berechnungen durchgeführt. Mirtich modifiziert in [Mi97] mit seiner Vorgehensweise *V-Clip* ebenfalls die Vorgehensweise von [Li93], so daß der Algorithmus Durchdringungen der Objekte behandeln kann, numerisch robuster ist und die Implementierung erleichtert wird.

Eine dritte Klasse von Kollisionserkennungsverfahren benutzt *Hüllkörperhierarchien* zur Behandlung *geometrisch komplexer Objekte*. Die in der Literatur verwendeten

Hüllkörperhierarchien sind Bäume, deren Knoten mit einfachen Hüllkörpern wie z.B. Kugeln, Boxen oder Prismenkörpern assoziiert sind und die Oberfläche des Objektes überdecken. Der mit der Wurzel assoziierte Hüllkörper überdeckt das gesamte Objekt. Die mit den Knoten einer beliebigen Stufe des Baumes assoziierten Hüllkörper des Baumes überdecken die Teile der Oberfläche des Objektes, die nicht von Blattknoten einer höheren Stufe des Baumes überdeckt werden. Die Qualität der Überdeckung nimmt dabei von Stufe zu Stufe zu und die Blattknoten des Baumes bilden die feinste Überdeckung der gesamten Objektoberfläche. Die verschiedenen Verfahren unterscheiden sich in der Art der verwendeten Hüllkörper, im Verzweigungsgrad der Bäume sowie deren Berechnung und der Verwendung während des Kollisionserkennungsprozesses. Beispiele hierfür sind [HL+97, Za97, GLM96, BC+96, KH+, Hu95, PG95, FZ95, Qu94]. Eine genauere Betrachtung der einzelnen Verfahren zur Berechnung von Hüllkörperhierarchien erfolgt in Kapitel 3 und deren Verwendung wird in Kapitel 4 beschrieben.

Darüber hinaus gibt es noch weitere Verfahren wie z.B. Verfahren, die auf *Binary Space Partitioning Trees* (vgl. [NAT90]) oder *Voxeln (Volume Pixel)* (vgl. [MW88, NFA89]) basieren.

2.2.2 Große Anzahl von Objekten

Zur Beherrschung vieler Objekte in einer Szene werden Mechanismen benötigt, die in der Lage sind, die Anzahl der durchzuführenden Objektpaartests zu reduzieren. Dies wird dadurch erreicht, daß in einer ersten Stufe der Kollisionserkennung die Paare von Objekten bestimmt werden, die in so enger Nachbarschaft zueinander sind, daß eine Kollision nicht ausgeschlossen werden kann.

Dazu eignen sich Verfahren der Raumpartitionierung wie *uniforme Raumunterteilung* [ZO+93], *Oktrees* [Ha88, MW88, SH92], *BSP*- oder *MSP-Trees* [TN87, Va91]. Alternativ dazu eignen sich die *Sweep-and-Prune* Methoden [Li93, CL+95, LM+] sowie 4-dimensionale Verfahren [Hu95, Hu95a, GSF94, Ca90].

2.2.3 Unstrukturierte Flächenmengen

Feature-basierte oder *simplex-basierte* Algorithmen haben in ihrer jeweiligen Basisversion das Problem, daß sie nur konvexe Polyeder behandeln können. Durch Erweiterungen sind sie in der Lage unter Inkaufnahme von Performanceverlusten auch nichtkonvexe Polyeder zu behandeln. Eine Erweiterung auf *unstrukturierte Flächenmengen* erscheint schwierig bzw. nicht möglich und wurde bisher nicht in Angriff genommen.

Die Algorithmen hingegen, die auf *Hüllkörperhierarchien* basieren, sind alle in der Lage, *unstrukturierte Flächenmengen* zu behandeln. Insofern ist diese Klasse von Algorithmen bestens geeignet, in *Virtual Reality* Anwendungen zur Kollisionserkennung eingesetzt zu werden, da die oft aus Konvertern für CAD-Daten stammenden Objekte der Simulationen in der Regel *unstrukturierte Flächenmengen* sind.

2.2.4 Dynamische Kollisionserkennungsergebnisse

Held, Klosowski und Mitchell [HKM95] klassifizieren die Kollisionserkennungsverfahren in drei Kategorien:

1. *Statische Kollisionserkennung*
   Bei der *statischen Kollisionserkennung* wird für eine vorgegebene Konfiguration von Objekten getestet, ob in dieser Überlappungen zwischen den Objekten vorliegen oder nicht.

2. *Pseudo dynamische Kollisionserkennung*
   Die *pseudo dynamische Kollisionserkennung* verwendet als Basis die *statische Kollisionserkennung.* Diese wird zu diskreten Zeitpunkten während der Bewegung der Objekte in einer Szene verwendet, um Kollisionen statisch zu erkennen.

3. *Dynamische Kollisionserkennung*
   Bei der *dynamischen Kollisionserkennung* werden nicht nur diskrete Zeitpunkte und die zugehörigen Konfigurationen der Objekte zur Kollisionserkennung benutzt, sondern die Bewegungen der Objekte selbst werden kontinuierlich auf Kollision getestet.

Mirtich benutzt in [Mi96] bei seiner physikalischen Simulation das *closest-feature tracking* von Lin [Li93] bzw. in [Mi98] die Weiterentwicklung *V-Clip* zur Kollisionserkennung. Baraff [Ba92] verwendet ebenfalls einen *feature-basierten* Ansatz zur Kollisionserkennung in seiner physikalischen Simulation. Mit Hilfe dieser Kollisionserkennungsverfahren können für diese Anwendungen qualitativ geeignete Ergebnisse berechnet werden.

Weitergehende Simulationen, die Kollisionen von Objekten modellieren, benötigen im allgemeinen den Kollisionszeitpunkt zwischen den Objekten und die dabei beteiligten Objektteile. Die *feature-basierten* Verfahren wurden in der Vergangenheit zur Berechnung solcher Kollisionsinformationen verwendet. Verfahren, die *Hüllkörperhierarchien* verwenden hingegen, werden bisher lediglich zur *statischen* oder *pseudo dynamischen Kollisionserkennung* eingesetzt. Hubbard [Hu95] benutzt 4-dimensionale *space-time bounds* zum einen zur Behandlung einer großen Anzahl von Objekten in der Szene und zum anderen, um den Einsatz der *statischen* oder *pseudo dynamischen* Kollisionserkennung zu steuern. Damit kann er garantieren, daß er keine Kollisionen übersieht, die mindestens einen vorgebenenen Zeitraum Δ andauern, und den Kollisionszeitpunkt bis auf Δ genau bestimmen. Voraussetzung für dieses Verfahren ist jedoch, daß a priori Informationen über die maximale Geschwindigkeit bzw. Beschleunigung jedes Objektes in der Szene vorliegen. Alle anderen Verfahren mit *Hüllkörperhierarchien* berechnen ausschließlich *statische Kollisionsinformationen.* Nicht zuletzt deshalb ist ihr Einsatz im Zusammenhang mit weitergehenden Simulationen bisher nicht möglich, da sie die von den Simulationen benötigten Informationen zum genauen Kollisionszeitpunkt und der dabei kollidierenden Objektteile nicht bestimmen können.

2.2.5 Nicht vorspezifizierte Bewegungen

Bewegungen von Objekten in *Virtual Reality* Anwendungen sind nicht a priori bekannt. Daher können in der Regel auch keine speziellen Eigenschaften über die Bewegungen der Objekte ausgenutzt werden. Für eine allgemeine Einsetzbarkeit in solchen Systemen sollte daher auf die Nutzung von Informationen wie maximale Beschleunigung oder maximale Geschwindigkeit, wie sie z.B. HUBBARD [Hu95] zur Berechnung seiner *space-time bounds* oder LIN [Li93] in ihrem *scheduling scheme* verwendet, verzichtet werden. Die Informationen werden in diesen Berechnungen dazu benutzt, um die Anzahl der Objektpaartests in einem Kollisionserkennungsschritt zu reduzieren bzw. um die durchzuführenden Kollisionserkennungstests zu steuern.

2.2.6 Mehrere bewegte Objekte

Ein Teil der Arbeiten zur Kollisionserkennung beschränken sich in ihrer Betrachtung auf die *Fly-Through* Situation [KH+, HKM95]. Dabei wird ein bewegtes Objekt betrachtet, das sich durch eine ansonsten statische Szene bewegt. In dieser Situation kann in den Verfahren ausgenutzt werden, daß ein Großteil der Objekte in einer Szene unbewegt oder ein Teil der Objekte nicht bewegbar ist.

2.2.7 Integration in ein visualisierungsgesteuertes System und Echtzeitfähigkeit

Die *Integration in ein visualisierungsgesteuertes System* hängt mit der *Echtzeitfähigkeit* der Kollisionserkennung zusammen. Als *Echtzeitfähigkeit* der Kollisionserkennung bezeichnen wir dabei die Möglichkeit, Zeit gegen Genauigkeit der Berechnung tauschen zu können. D.h. der Kollisionserkennungsprozeß kann nahezu zu jedem Zeitpunkt während der Berechnung abgebrochen werden und es wird dadurch ein approximatives Kollisionserkennungsergebnis berechnet. Ein *visualisierungsgesteuertes System* begrenzt die Zeit für die Durchführung von Simulationen, da z.B. aufgrund von Immersion eine Grenze für die Bildwiederholrate nicht unterschritten werden darf, weil sonst die bekannten Symptome der *Simulatorkrankheit* bei den Benutzern auftreten. Daher muß die Kollisionserkennung in der Lage sein, innerhalb eines vorgegebenen Zeitraumes ein Ergebnis zu berechnen, wobei ein Tradeoff zwischen Zeit und Ergebnisqualität entsteht.

Die *simplex-* und *feature-basierten* Verfahren zur Kollisionserkennung sind nicht in der Lage, mit einem eingeschränkten Zeitbudget ein approximatives Ergebnis zu berechnen. Sie berechnen zu jeder Konfiguration einen internen Zustand, von dem ausgehend sie für vorgegebene Objektbewegungen den neuen internen Zustand berechnen. Vor dem Ende dieser Berechnung können diese Verfahren nicht unterbrochen werden, da sie sonst nicht über den internen Zustand für die aktuelle Objektkonfiguration verfügen, von der ausgehend sie für neue Objektbewegungen die Kollisionserkennung durchführen.

HUBBARD [Hu95] integriert in sein System zur Kollisionserkennung Methoden der zeitkritischen Berechnung, um eine konstante Framerate für *visualisierungsgesteuerte Systeme* zu erreichen. Dabei wird bei jeder Bewegung der Objekte höchstens einmal die *broad-phase* mit der Verwendung der *space-time bounds* und einmal die *narrow-phase* mit *statischer Kollisionserkennung* unter Zuhilfenahme der *sphere-trees* verwendet. In diesem Fall kann das Verfahren in einem vorgegebenen Zeitbudget ein evtl. approximatives Ergebnis berechnen. Dabei kann jedoch nicht garantiert werden, daß Kollisionen erkannt werden, die kürzer als einen Frame andauern. Die Genauigkeit des Verfahrens ist somit die zeitliche Auflösung der Frames. Alle anderen Verfahren mit *Hüllkörperhierarchien* können in dieses Konzept von HUBBARD zur Verwendung in der *narrow-phase* eingebunden werden.

2.2.8 Alternative Schwerpunkte

Kollisionserkennung wird in der Literatur auch mit anderen Schwerpunkten betrachtet. Ein Beispiel dafür ist die Untersuchung der theoretischen Komplexität des Kollisionserkennungsproblems. DOBKIN und KIRKPATRICK beschreiben in [DK85] eine Methode zur hierarchischen Repräsentation *konvexer Polyeder*. Damit kann der Abstand zwischen zwei Polyedern P_1 und P_2 der Komplexität n in Zeit $O(\log^2 n)$ berechnet werden, wenn P_1 und P_2 *konvex* sind und für den Fall, daß P_1 konvex ist und P_2 beliebig, ist dies in Zeit $O(n \cdot \log n)$ möglich (vgl. [Sch94, DH+90, DK90, DK85]). Darüber hinaus kann in derselben Zeit berechnet werden, ob eine Translation kollisionsfrei ist.

SCHÖMER zeigt in [Sch94], daß für nichtkonvexe Polyeder P_1 und P_2 der Komplexität n das Kollisionserkennungsproblem für ein translatorisch bewegtes Polyeder in subquadratischer Laufzeit $O(n^{2-\epsilon})$ mit $\epsilon > 0$ gelöst werden kann. SCHÖMER und THIEL verbessern in [ST94] dieses Resultat auf $O(n^{8/5+\epsilon})$ im Fall eines translatorisch bewegten Polyeders und zeigen, daß für den Fall eines rotatorisch bewegten Polyeders das Problem in Zeit $O(n^{5/3+\epsilon})$ gelöst werden kann, wobei ϵ eine beliebige positive Konstante ist. Darüber hinaus wird in [ST96] gezeigt, daß für zwei bewegte Polyeder, deren Position und Orientierung mittels einer polynomiellen Funktion beschrieben ist, das Kollisionserkennungsproblem in subquadratischer Laufzeit gelöst werden kann. Andere Schwerpunkte sind z.B. Objekte mit gekrümmten Oberflächen [LM, SW+93, HBZ90], deformierbare Objekte [KS+98, HD+96, SK+95], numerisch stabile Kollisionserkennung [SSW95, St94] oder Kollisionserkennung im Bereich der algorithmischen Bewegungsplanung [CES96, ECS96, Ec94, La91, Ca86].

2.3 Gliederung der Lösungsansätze

Orientiert an den oben ausgearbeiteten Anforderungen an die Kollisionserkennung werden wir im weiteren Verlauf die Lösungsansätze strukturieren. Als Objekte in der Kollisionserkennung werden wir in der gesamten Arbeit *unstrukturierte Flächenmengen* benutzen, wobei wir davon ausgehen, daß die Flächen planar und konvex sind. Dies ist sinnvoll, da z.B. *Virtual Reality* Anwendungen zur Visualisierung *Renderer* benutzen, die ausschließlich Dreiecke visualisieren. Folgerichtig erzeugen Konverter, die Daten für *Virtual Reality* Anwendungen aufbereiten, Dreiecksnetze als Objekte, wobei in der Regel keine Polyeder sondern *unstruktierte Flächenmengen* entstehen. Diese Objekte werden mittels *Transformationen* (siehe Anhang A) bewegt. Die Anzahl der in einem Kollisionserkennungsschritt bewegten Objekte ist beliebig.

Ziel der Arbeit ist es, eine ganzheitliche Betrachtung des Kollisionserkennungsprozesses unter Berücksichtigung der Anforderungen durchzuführen. Wir haben uns zur effizienten Behandlung des Objektpaartests für die Verwendung von Hüllkörperhierarchien entschieden, weil sie aus unserer Sicht das größte Potential zur Beherrschung komplexer Objekte bieten. Besonderes Augenmerk haben wir dabei darauf gelegt, daß neben der *statischen* auch eine durchgängige *dynamische* Kollisionserkennung ermöglicht wird.

Kapitel 3 bis 5 beschreiben Verfahren, die es ermöglichen, einen Objektpaartest effizient durchzuführen und dabei auch eine *dynamische* Kollisionserkennung realisieren zu können.

In Kapitel 3 werden wir uns mit der Berechnung von Hüllkörperhierarchien beschäftigen. Diese werden zur Beherrschung *geometrisch komplexer Objekte* eingesetzt. Nach der Beleuchtung der in der Literatur vorgeschlagenen Verfahren werden wir dort Optimierungskriterien für die Berechnung von Hüllkörperhierarchien entwickeln und Algorithmen beschreiben, die entsprechend dieser Optimierungskriterien möglichst optimale Hüllkörperhierarchien unter Verwendung unterschiedlicher Hüllkörpertypen berechnen.

Gegenstand von Kapitel 4 ist die Verwendung von Hüllkörperhierarchien während des Kollisionserkennungsprozesses. Dazu werden wir neben der bekannten *statischen* Überlappungstests für die gängigen Hüllkörpertypen *dynamische* Kollisionserkennungsverfahren entwickeln, mit Hilfe derer Hüllkörperhierarchien in den Prozeß der *dynamischen* Kollisionserkennung integriert werden können, so daß in einem Objektpaartest keine Kollision übersehen wird. Darüber hinaus werden verschiedene Expansionsstrategien für Hüllkörperhierarchien vorgestellt, die die Kosten für die Benutzung der Hüllkörperhierarchien gering halten sollen. Alle vorgestellten Verfahren zur Benutzung der Hüllkörperhierarchien sind dabei nicht nur auf die von uns berechneten Hüllkörperhierarchien anwendbar, sondern können auch mit allen in der Literatur vorgeschlagenen Hüllkörperhierarchien kombiniert werden.

Kapitel 5 beschäftigt sich mit der Basiskollisionserkennung für Objektteile. Dazu werden nach der Vorstellung des bekannten *statischen* Kollisionstests für zwei Flächen Verfahren beschrieben, die es erlauben *dynamische* Kollisionstests durchzuführen. Die Verfahren unterscheiden sich in der Genauigkeit der Approximation eines möglichen Kollisionszeitpunktes, wobei auch eine beliebig genaue Bestimmung des Kollisionszeitpunktes möglich ist.

Nach der Behandlung *geometrisch komplexer Objekte* wenden wir uns in Kapitel 6 der Beherrschung *einer großen Anzahl von Objekten* in einer Szene zu. Neben der Beschreibung der gängigen Verfahren zur Raumpartitionierung werden wir das *one-dimensional*

Sort-and-Sweep Verfahren von Lin [Li93] so modifizieren, daß es in einen *dynamischen* Kollisionserkennungsprozeß integriert werden kann.

Der gesamte Ablauf eines Kollisionserkennungsschrittes innerhalb eines *Virtual Reality* Systems ist Gegenstand von Kapitel 7. Die Problematik der *Echtzeitfähigkeit* der Kollisionserkennung wird in Kapitel 8 betrachtet. Darin werden, ausgehend von der in Kapitel 7 beschriebenen Einbettung der Kollisionserkennung in ein *Virtual Reality System*, Methoden erarbeitet, die eine *zeitkritische Kollisionserkennung* im Zusammenhang auch mit einer *dynamischen* Kollisionserkennung ermöglichen.

Die Kollisionserkennung stellt die Basis für die Simulationsumgebung innerhalb der *Virtual Reality* Plattform **DBView** des Virtual Reality Competence Center der Daimler-Benz AG dar. Die Realisierungsaufgabe besteht somit neben der Implementierung der vorgestellten Verfahren zur Kollisionserkennung innerhalb eines Kollisionserkennungsmoduls auch in dem Aufbau der Basismodule zur *Objektrepräsentation* und zur *Transformationsmathematik*. Aufbauend auf dem Kollisionserkennungsmodul wurden und werden die weitergehenden Simulationen aus dem Bereich der *Dynamiksimulation* [Sa99, SS98] und der *interaktiven Objektmanipulation* [Bu98, BS98] realisiert. Durch die eigenständige Realisierung der Objektrepräsentation und der Transformationsmathematik wird eine weitestgehende Unabhängigkeit von der Visualisierungsplattform erreicht, so daß eine einfache Integration in alternative Visualisierungssysteme ermöglicht wird. Kapitel 9 beschäftigt sich mit der Realisierung des Kollisionserkennungsmoduls sowie der Basismodule.

In Kapitel 10 werden die Ergebnisse der verschiedenen Verfahren anhand von Beispielszenarien dargestellt.

Kapitel 11 faßt die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf weitergehende Arbeiten im Bereich der Kollisionserkennung.

Kapitel 3

Berechnung von Hüllkörperhierarchien

3.1 Einführung

Hierarchische Approximationen von Flächenmengen wurden bereits mehrfach in der Literatur zur Beschleunigung der Kollisionserkennung vorgeschlagen. Allen Ansätzen ist dabei der Gedanke gemeinsam, daß Teile der Flächenmengen mit Hilfe einfacher Hüllkörper approximiert werden. Die berechneten Hüllkörperhierarchien werden dazu benutzt, den *Objektpaartest* zu beschleunigen. Dies geschieht dadurch, daß anstelle der Flächenmenge bzw. Teilen dieser in der Kollisionserkennung die approximierenden Hüllkörper verwendet werden, die einfach gegeneinander auf Kollision zu testen sind (vgl. Kapitel 4). Kollidieren zwei Hüllkörper nicht, so kann gleichzeitig eine Kollision zwischen den beiden umhüllten Flächenmengen ausgeschlossen werden. Kollidieren sie dagegen, so kann eine Kollision zwischen den Flächenmengen nicht ausgeschlossen werden. In diesem Fall müssen genauere Approximationen der Flächenmengen mit einer größeren Anzahl von Hüllkörpern betrachtet werden. Dabei sollen die Stufen dieser Approximation derart organisiert sein, daß sie während des Kollisionserkennungsprozesses das Verfahren mit möglichst wenigen Hüllkörpertests zu der Stelle einer Kollision führen oder entscheiden, daß keine Kollision vorliegt.

Die bei einem *Objektpaartest* unter Verwendung von Hüllkörperhierarchien zu optimierende Kostenfunktion läßt sich wie folgt beschreiben:

$$K_{gesamt} = \sum_{i=0}^{A^{Hkt}} K_i^{Hkt} + \sum_{i=0}^{A^{Bkt}} K_i^{Bkt}$$

wobei

A^{Hkt}	Anzahl Hüllkörpertests
K_i^{Hkt}	Kosten des Hüllkörpertests i
A^{Bkt}	Anzahl Basiskollisionstest
K_i^{Bkt}	Kosten des Basiskollisionstest i

Wird nur ein Typ von Hüllkörpern benutzt und nur eine Art von Basiskollisionstest, vereinfacht sich die Kostenfunktion zu der von GOTTSCHALK [GLM96] beschriebenen Formel:

$$K_{gesamt} = A^{Hkt} \cdot K^{Hkt} + A^{Bkt} \cdot K^{Bkt}$$

wobei

K^{Hkt}	Kosten eines Hüllkörpertests
K^{Bkt}	Kosten eines Basiskollisionstest

Ziel der Benutzung von Hüllkörperhierarchien ist es, diese Kostenfunktion zu minimieren. Sie hängt neben den beiden vorliegenden Hierarchien auch von den Bewegungen der beiden Objekte ab. Die Bewegungen der Objekte sind jedoch zum Vorberechnungszeitpunkt nicht bekannt sondern nur die Geometrie der Objekte. Das Ziel der Vorberechnung ist es daher, eine Hüllkörperhierarchie zu berechnen, die während des Kollisionserkennungsprozesses in möglichst vielen Situationen ein gutes Verhalten zeigt.

Betrachten wir die Struktur der Kostenfunktion, so ergeben sich vier Optimierungsaufgaben:

1. Minimierung der Kosten eines Basiskollisionstests
2. Minimierung der Kosten eines Hüllkörpertests
3. Minimierung der Anzahl der Hüllkörpertests
4. Minimierung der Anzahl der Basiskollisionstests

Die Minimierung der Kosten eines Basiskollisionstests und eines Hüllkörpertests werden dabei im wesentlichen von der Qualität des berechneten Ergebnisses bestimmt. In der Literatur wird im Zusammenhang mit Hüllkörperhierarchien im wesentlichen von einem *statischen* Kollisionstest ausgegangen, dessen Ergebnis sich nicht direkt für weitergehende Simulationen eignet. Hüllkörpertests werden in Kapitel 4 und Basiskollisionstests in Kapitel 5 beschrieben.

Die Minimierung der Kosten eines Basiskollisionstests ist dabei losgelöst von den übrigen Optimierungszielen betrachtbar, während die drei restlichen Ziele konkurrierend sind. Minimale Kosten für einen Hüllkörpertest fallen bei der Verwendung von Kugeln an, jedoch lassen sich nur wenige reale Objekte mit Hilfe von Kugeln optimal approximieren, so daß viele Kugeln nötig sind, um eine geeignete Approximation zu erreichen. Dadurch steigt aber die Anzahl der Hüllkörpertests. Eine minimale Anzahl von Hüllkörpertests ensteht, wenn zur Approximation verschiedener Objekteile unterschiedliche Hüllkörper mit den jeweils besten Approximationseigenschaften für das spezielle Objektteil gewählt werden. Dadurch variieren die Kosten eines Hüllkörpertests je nach verwendetem Hüllkörpertyp. Die Anzahl der Basistests wird minimiert, wenn Hüllkörper bis auf die unterste Stufe der Hierarchie verwendet werden, d.h. wenn einzelne Flächen mit Hüllkörpern approximiert werden. Dies steigert jedoch die Anzahl der Hüllkörpertests. Das Ziel, in möglichst vielen Situationen eine gute Performance der Hüllkörperhierarchien zu erreichen, wird also dadurch erreicht, daß ein Kompromißziel für die konkurrierenden Optimierungsaufgaben realisiert wird.

3.2 Bisherige Arbeiten

Bisherige Ergebnisse im Bereich von Hüllkörperhierarchien für Polyeder bzw. Flächenmengen werden u.a. in [BC+96, GLM96, Hu95, PG95, Qu94, FZ95, Za97, KH+] vorgestellt. Sie unterscheiden sich in der Art der benutzten Hüllkörper und den Verfahren zum Aufbau der Hierarchien.

Als Hüllkörper werden

- Kugeln [Hu95, PG95, Qu94]
- achsenorientierte Boxen [FZ95, Za97]
- beliebig orientierte Boxen [GLM96]
- Zylinder mit dreieckiger Grundfläche und beliebig orientierte Boxen [BC+96]
- konvexe Polyeder beschränkter Richtungen [KH+, Za98]

verwendet. Kugeln haben dabei den Vorteil, daß der Kollisionstest zwischen zwei Kugeln extrem schnell ist, so daß die Kosten für einen Hüllkörpertest minimiert werden. Der Nachteil bei diesen Methoden ist, daß viele reale Objekte lokal quasi-planare oder -orthogonale Strukturen aufweisen, die sich nur schlecht mit Hilfe von Kugeln approximieren lassen, so daß in diesem Fall die Anzahl der Hüllkörper- und Basiskollisionstests stark ansteigt, wenn zwei Objekte einander nahe kommen. Der Test zwischen achsenorientierten Boxen ist kaum schwieriger als der Test zwischen Kugeln, aber quasi-planare oder -orthogonale Strukturen lassen sich mit Hilfe von achsenorientierten Boxen besser approximieren als mit Hilfe von Kugeln. Die Güte der Approximation hängt dabei wegen der festgelegten Orientierung der Boxen auch von der Spezifikation der Objekte ab. Werden quasi-planare Strukturen nicht achsenorientiert spezifiziert, so ergeben sich schlechte Approximationsergebnisse. Dieses Problem kann mit Hilfe von beliebig orientierten Boxen gelöst werden. Beliebig orientierte Boxen approximieren quasi-planare oder -orthogonale Strukturen in jeder Orientierung gut, so daß das Approximationsergebnis in dieser Hinsicht nicht von der speziellen Lage der Flächenmenge abhängt. Der Nachteil von beliebig orientierten Boxen gegenüber achsenorientierten Boxen und Kugeln ist, daß der Kollisionstest zwischen beliebig orientierten Boxen aufwendiger ist. Die konvexen Polyeder beschränkter Richtungen (*k-dops*) versuchen analog zu den achsenorientierten Boxen durch die Beschränkung der Anzahl der Richtungen den Test zwischen 2 *k-dops* möglichst einfach zu gestalten. Dies wird zu dem Preis erzielt, daß bei einer Bewegung des *k-dops* die Modifikation in die Endlage aufwendiger ist als bei den anderen Hüllkörpertypen, da in jeder Lage eines *k-dops* nur die vorgegebenen Richtungen für den Hüllkörper benutzt werden dürfen.

Als Strategien für den Aufbau von Hierarchien werden *bottom-up* und *top-down* Strategien verwendet. Bei *bottom-up* Strategien werden zunächst die einzelnen Flächen der zu approximierenden Flächenmenge mit Hüllkörpern überdeckt und danach sukzessive Teile der Flächenmenge bzw. deren Hüllkörper zusammengefaßt bis die Hierarchie vollständig ist. *Top-down* Strategien beginnen mit der gesamten Flächenmenge und berechnen zunächst dafür einen Hüllkörper und teilen dann rekursiv die Hüllkörper bzw. die dadurch umhüllten Teile der Flächenmenge in kleinere Einheiten auf, bis die dadurch enstandenen Einheiten nicht weiter unterteilt werden können.

3.2.1 Kugelbäume (*sphere-trees*)

Quinlan [Qu94] benutzt Kugeln als Hüllkörper. Er berechnet die Hierarchie mit Hilfe einer *top-down* Strategie. In einem ersten Schritt werden die Flächen mit Kugeln, die auf einem regelmäßigen Gitter liegen, approximiert. Diese bilden die Blätter des *sphere-trees*. Der *sphere-tree* wird dann mit Hilfe einer *divide-and-conquer* Strategie berechnet. Dabei wird die Menge der Blätter rekursiv in zwei möglichst gleichgroße Mengen unterteilt, der Unterbaum für die beiden Mengen und die Kugel des Wurzelknotens berechnet. Um eine Menge von Blattkugeln eines Unterbaumes zu unterteilen, wird die kleinste achsenorientierte Box berechnet, die alle Kugeln enthält und diese entlang der längsten Achse in zwei kleinere unterteilt. Jede Blattkugel gehört zu der Box, in der der Mittelpunkt der Kugel liegt. Die Kugel für den Wurzelknoten wird berechnet, indem heuristisch eine möglichst kleine Kugel berechnet wird, die alle Blattkugeln des Unterbaums enthält.

In dieser Vorgehensweise werden als Hüllkörper Kugeln berechnet, die andere Kugeln umhüllen. Was berechnet werden soll, ist eine Kugel, die den Teil der Flächenmenge überdeckt, die die Blattkugeln des Unterbaumes überdecken. Da auf der untersten Ebene die Kugeln auf einem regelmäßigen Gitter verteilt werden, überdeckt eine Blattkugel in der Regel nicht einen speziellen Teil der Flächenmenge, z.B. eine Fläche. Werden auf der untersten Ebene der Hierarchie Kugeln benutzt, die einzelne Flächen einhüllen, so überdeckt jeder innere Knoten der Hierarchie eine Menge von Flächen der gesamten Flächenmenge. Zu dieser Menge von Flächen kann die kleinste einschliessende Kugel jedoch entweder stabil numerisch, z.B. mit den *Downhill Simplex* Algorithmus (vgl. [PT+92, NM65]), oder kombinatorisch mit dem inkrementellen Verfahren von Welzl [We91] berechnet werden. Dadurch würde sich die Qualität der berechneten Kugeln direkt verbessern.

Die Beschränkung, daß auf unterster Ebene jeweils Flächen eingeschlossen werden, ist dabei keine echte Einschränkung, da bei triangulierten CAD-Modellen die Flächen üblicherweise derart klein sind, daß das Einschliessen der Flächen mit Kugeln einem sehr feinen Gitter entspricht, das dadurch vergröbert werden kann, daß die untersten Schichten der Hierarchie entfernt werden. Sind dagegen die Flächen zu grob modelliert, so können sie durch Triangulierung lokal verfeinert werden, so daß die unterste Ebene der Hierarchie einem geeigneten Raster auf der Flächenmenge entspricht.

Hubbard [Hu95, Hu95a] benutzt als Hüllkörper ebenfalls Kugeln. Dabei schlägt er zwei verschiedene Methoden zur Berechnung des *sphere-trees* vor.

- **Oktree-basiert**
  Berechne zunächst die Oktreerepräsentation des Objektes (vgl. [SW88a, SW88b]) und bilde daraus einen *sphere-tree*, indem jeder Oktant des Oktrees mit einer Kugel eingehüllt wird. Dieser *sphere-tree* kann nun mit Hilfe von *simulated annealing* Techniken (vgl. [PT+92]) optimiert werden, so daß die Kugeln das Objekt fester umschliessen. Als Maß für die Güte einer Kugelapproximation wird dabei der *gerichtete Hausdorff-Abstand* jeder Kugel zu der umhüllten Flächenmenge benutzt. Dabei wird der *gerichtete Hausdorff-Abstand* der Kugel zu der umhüllten Flächenmenge mit Hilfe eines einfachen Verfahrens grob approximiert. Nach Hubbard ist diese Methode jedoch zeitaufwendig und erzeugt mitunter hochgradig ungleiche Verteilungen der Kugeln.

- ***Medial-axis surfaces*-basiert**
  Berechne mit Hilfe einer Modifikation des Algorithmus von Goldak (vgl. [GY+91]) eine Approximation der *medial-axis surfaces*. Die Eckpunkte des dabei berechneten Voronoi-Diagramms innerhalb des Körpers beschreiben Kugeln, die die Oberfläche des Objektes in vier Punkten berühren. Diese bilden die unterste Stufe der Hierarchie. Nun werden *bottom-up* in jedem Schritt jeweils acht Kugeln zu einer neuen zusammengefaßt (in Analogie zu der Oktree-Repräsentation). Der dabei entstehende *sphere-tree* muß in einem Nachbearbeitungsprozeß so modifiziert werden, daß alle Teile der Objektoberfläche überdeckt werden, da während des Aufbauprozesses Teile der Objektoberfläche nicht überdeckt werden.

Palmer und Grimsdale [PG95] benutzen ebenfalls Kugeln als Hüllkörper. Sie benutzen zur Berechnung ihrer Hierarchien eine *top-down* Strategie, die sich an der Oktree-Datenstruktur orientiert. Im ersten Schritt wird das gesamte Objekt durch eine achsenorientierte Box umschlossen, deren Approximation durch eine Kugel berechnet wird. Diese bildet den Wurzelknoten des *sphere-trees*. Nun wird *top-down* jede Box aufgeteilt, die mehr als zwei Flächen enthält. Der Grenzwert von zwei Flächen je Box wurde dabei in den betrachteten Beispielen als empirisch gut ermittelt. Im Aufteilungsschritt für eine Box werden die acht Oktanten ermittelt und alle Boxen, die Flächen enthalten, durch eine Kugel umhüllt. Diese bilden die nächste Stufe der Hierarchie. Diese Vorgehensweise wird solange fortgesetzt, bis keine neuen Kugeln mehr erzeugt werden können. Durch die Benutzung einer *oktree-basierten* Methode ergibt sich das Problem, daß eine Fläche in mehreren Blattkugeln enthalten sein kann. Daher ist bei der Verwendung der Hierarchie darauf zu achten, daß Paare von Flächen, die exakt miteinander auf Kollisionen verglichen werden, nicht mehrfach getestet werden. Dies wird dadurch erreicht, daß zu jeder Fläche eine Liste von Flächen existiert, in die alle Flächen eingetragen werden, mit denen diese Fläche auf Kollision in der aktuellen Bewegung getestet worden ist.

3.2.2 Bäume mit achsenorientierten Boxen (*axis-oriented bounding box trees*)

Felger und Zachmann benutzen in [FZ95, Za97] achsenorientierte Boxen (*Iso-Boxen*) als Hüllkörper. Ihre Hierarchien werden mittels eines *top-down* Verfahrens berechnet. Im ersten Schritt wird die kleinste achsenorientierten Box berechnet, die die gesamte Flächenmenge umhüllt. In einem Aufteilungsschritt wird eine bereits berechnete achsenorientierte Box mittels einer Schnittebene parallel zu einer Koordinatenachse in zwei achsenorientierte Boxen zerlegt, die zusammen die ursprüngliche Box ergeben. Dabei wird die Schnittebene derart gewählt, daß die Menge der enthaltenen Flächen der Box möglichst halbiert wird. Von den drei möglichen Schnitten wird derjenige gewählt, der die Halbierung der Menge der Flächen bestmöglich realisiert. Dabei auftretende Flächen, die durch die Schnittebene geteilt werden, werden beiden entstehenden Boxen zugeteilt. Die Aufteilung der Boxen endet, wenn

- die Tiefe des Baumes eine bestimmte Schranke übersteigt,
- die Anzahl der Polygone in einer Box eine bestimmte Schranke unterschreitet,
- eine der entstehenden Boxen leer ist oder
- das Verhältnis der Anzahl der Flächen in den beiden neuen Boxen zu unausgewogen ist.

3.2.3 Bäume mit beliebig orientierten Boxen (*oriented bounding box trees*)

Gottschalk, Lin und Manocha [GLM96] verwenden beliebig orientierte Boxen als Hüllkörper. Diese berechnen sie mit Hilfe einer einfachen *top-down* Strategie. Zunächst wird eine beliebig orientierte Box für die gesamte Flächenmenge berechnet. Eine Box wir entlang der längsten Achse durch eine orthogonale Ebene unterteilt, wobei die Teilungskoordinate entlang der Achse durch den Mittelwert der Eckpunkte auf der konvexen Hülle der umhüllten Flächenmenge bestimmt wird. Alternativ zu der Vorgehensweise mit dem

Mittelwert kann auch die Halbierung der längsten Achse erfolgen. Die Unterteilung einer Flächenmenge endet, wenn durch den Aufteilungsschritt alle Flächen in eine enstehende Box fallen.

3.2.4 Bäume mit konvexen Polyedern beschränkter Richtungen (*k-dop trees*)

Klosowski et.al. [KH+] benutzen konvexe Polyeder beschränkter Richtungen (*k-dops*) als Hüllkörper. Durch die Beschränkung der Richtung der *k-dops* ergibt sich ein ähnlicher Kollisionstests wie bei Iso-Boxen. Eine Iso-Box ist der Spezialfall eines *6-dops*, dessen Richtungen die x-, y- und z-Achse des Weltkoordinatensystems sind. Ein *k-dop* wird durch $k/2$ Intervalle beschrieben und zwei *k-dops* kollidieren genau dann, wenn sich alle $k/2$ zugehörige Paare von Intervallen überlappen. Dadurch ergibt sich ein recht einfaches Verfahren zur Kollisionserkennung zwischen zwei *k-dops*. Nachteilig bei den *k-dops* ist, daß der Aktualisierungsschritt, d.h. die Berechnung eines *k-dops* in der aktuellen Position der Flächenmenge aus dem zugehörigen *k-dop* in der Ausgangsposition oder der vorherigen Position, i.a. aufwendiger ist als für einfache Hüllkörper wie Kugel, Iso-Box oder beliebig orientierte Box. Dadurch entsteht ein Trade-Off zwischen den Kosten für die Aktualisierung der *k-dops* und den Kosten für den Kollisionstest zweier *k-dops*. Die Idee, Objekte mit Hilfe von Ebenen einer beschränkten Anzahl von Richtungen zu approximieren, wurde Klosowski et.al. zufolge zuerst von Kay und Kajiya [KK86] im Rahmen des *ray tracings* eingeführt und später von Wissenschaftler bei IBM [CR95] als *18-dops* verwendet.

Klosowski et.al. berechnen binäre Bäume mit Hilfe einer *top-down* Strategie. Die von ihnen betrachteten Flächen sind Dreiecke und werden für die Aufteilung einer Flächenmenge durch einen Punkt repräsentiert. In einem ersten Schritt wird der *k-dop* für die gesamte Flächenmenge berechnet. Die Aufteilung einer Flächenmenge erfolgt durch eine Ebene orthogonal zu einer der Weltkoordinatenachsen. Die Lage der Ebene auf einer der Achsen ergibt sich entweder als Mittelwert oder Median der Projektion der repräsentierenden Punkte auf diese Achse. Die Auswahl der Achse erfolgt nach einem der folgenden Kriterien:

- Minimum der Summe der Volumina der beiden enstehenden Hüllkörper
- Minimum des Maximums der Volumina der beiden entstehenden Hüllkörper
- Maximum der Varianz der Projektion der repräsentierenden Punkte auf die Achse
- Maximum der Länge der Projektion des Hüllkörpers auf die Achse

Die Aufteilung einer Flächenmenge endet, wenn die Anzahl der Flächen in einem Knoten unter eine Grenze τ fällt. Diese ist bei den bewegten Objekten auf 40 und bei den festen Objekten auf 1 voreingestellt.

3.2.5 Bäume mit verschiedenen Hüllkörpern

Barequet et.al. [BC+96] berechnen binäre Hierarchien mit verschiedenen Typen von Hüllkörpern in verschiedenen Stufen der Hierarchie. Experimentell wurden als Hüllkörper achsenorientierte Boxen, beliebig orientierte Boxen und Zylinder mit dreieckiger Grundfläche (*pie slices*) benutzt.

Die Bäume werden mittels einer *bottom-up* Strategie aufgebaut. Dabei werden unterschiedliche Regeln zum Auffinden von Hüllkörperpaaren, die in der nächsthöheren Stufe zu einem neuen Knoten zusammengefaßt werden, und zur Bestimmung des neuen Hüllkörpers benutzt:

- **Achsenorientierte Boxen**
  In der ersten Stufe werden die einzelnen Flächen mit achsenorientierten Rechtecken als entartete Boxen umhüllt. In einer Stufe werden Paare von adjazenten oder überlappenden Boxen zu neuen Boxen zusammengefaßt. Die Box des Vaterknotens eines Paares zusammengefaßter Knoten ist die kleinste achsenorientierte Box, die die beiden Boxen umhüllt.

- **beliebig orientierte Boxen**
  In der ersten Stufe werden die einzelnen Flächen mit den kleinsten umhüllenden Rechtecken approximiert. Zur Berechnung der Box des Vaterknotens eines Paares von Boxen werden die Eckpunkte der beiden Boxen benutzt und als 3D-Population interpretiert, d.h. diese Punkte werden als eine stochastische Verteilung von Vektoren im $\mathbb{R}^3$ angesehen. Von dieser Verteilung wird als statistische Größe die Kovarianzmatrix berechnet. Die Eigenvektoren und Eigenwerte werden dann zur Bestimmung der Orientierung der neuen Box in verschiedener Form herangezogen:

  - *All-principal-component box*
    Die Box hat ihr Zentrum im Massezentrum der 16 Eckpunkte. Als Orientierung werden die drei Eigenvektoren der Kovarianzmatrix benutzt.
  - *Max-principal-component box*
    Der Eigenvektor zum größten Eigenwert der Kovarianzmatrix wir als eine Richtung der Boxorientierung benutzt. Die 16 Eckpunkte werden auf eine Ebene orthogonal zu dieser Richtung projiziert und das kleinste einschliessende Rechteck in zwei Dimensionen berechnet. Dieses ergibt die beiden fehlenden Richtungen der neuen Box.
  - *Min-principal-component box*
    Die Vorgehensweise ist wie bei der vorherigen, nur daß der Eigenvektor zum kleinsten Eigenwert der Kovarianzmatrix verwendet wird.

  Es werden jeweils solche Paare von Boxen zusammengefaßt, deren Vaterbox minimales Volumen oder Oberfläche aufweist.

- **Dreieckige Zylinder (*pie slices*)**
  Ähnlich der Berechnung beliebig orientierter Boxen werden auch bei den dreieckigen Zylindern die 12 Eckpunkte als 3D-Population interpretiert und die Kovarianzmatrix zur Berechnung eines neuen Zylinders herangezogen. Dabei werden folgende beiden Methoden verwendet:

  – *Max-principal-component pie slice*
    Der Eigenvektor zum größten Eigenwert wird als eine Richtung des neuen Zylinders benutzt. Die 12 Eckpunkte werden auf eine Ebene orthogonal zu dieser Richtung projiziert und das kleinste einschliessende gleichschenklige Dreieck berechnet. Die *Minkowski-Summe* des Dreiecks mit dem Richtungsvektor ergibt den neuen Zylinder.

  – *Min-principal-component pie slice*
    Die Vorgehensweise ist wie bei der vorherigen, nur daß der Eigenvektor zum kleinsten Eigenwert der Kovarianzmatrix verwendet wird.

Die Performance der verschiedenen Verfahren wurde empirisch durch die Anzahl der Hüllkörpertests im Kollisionserkennungsprozeß miteinander verglichen. Dabei zeigten die dreieckigen Zylinder, die mit Hilfe der *min-principal-component* Regel generiert wurden, die beste Performance vor den beliebig orientierten Boxen, die mit derselben Regel erzeugt wurden. Die Hierarchien der *max-principal-component* Regel zeigten dagegen eine schlechte Performance, weil die durch sie erzeugten Hüllkörper eine maximal lange Kante enthalten. Die achsenorientierten Boxen waren dabei den beiden anderen Hüllkörpertypen unterlegen. Fraglich bei dieser Vorgehensweise ist, ob die verschiedenen Hüllkörpertests mit einem einheitlichen Kostenmaß miteinander verglichen werden dürfen. Bei uniformem Kostenmaß für einen Hüllkörpertest ist z.B. klar, daß beliebig orientierte Boxen den achsenorientierten überlegen sind.

3.3 Berechnungsverfahren für Hierarchien

Eine Hüllkörperhierarchie ist ein Baum, dessen Knoten n eine Flächenmenge F_n und der F_n überdeckende Hüllkörper h_n zugeordnet ist. Dem Wurzelknoten r des Baumes ist die gesamte Flächenmenge F zusammen mit dem überdeckenden Hüllkörper h_r zugeordnet. Die Kinder $n_1, \ldots, n_k$ eines Knotens n haben dabei die Eigenschaft, daß die Vereinigung der Flächenmengen F_{n_i} die Flächenmenge des Vaterknotens F_n ergeben und daß die Vereinigung der Hüllkörper h_{n_i} die gesamte Flächenmenge des Vaterknotens F_n überdecken. Zur Berechnung von Hüllkörperhierarchien bieten sich die beiden Klassen der *top-down*- und *bottom-up* Berechnungsverfahren an.

3.3.1 *Top-down* Berechnung von Hüllkörperhierarchien

Der Vorgang der *top-down* Berechnung von Hierarchien startet an der Wurzel des Baumes. Der Wurzelknoten r erhält die gesamte Flächenmenge F zugeordnet, für die ein Hüllkörper h_r berechnet wird, und bildet die Menge N_0 der Knoten der Tiefe 0. In jedem Schritt der Hierarchieberechnung wird nun aus der Menge N_i der Knoten der Tiefe i die Menge der Knoten N_{i+1} der Tiefe $i+1$ berechnet. Dazu werden für alle Knoten n aus der Menge N_i die Kindknoten berechnet oder festgestellt, daß n nicht weiter unterteilt werden kann. In diesem Fall wird n zu einem Blatt des Baumes. Die in diesem Schritt erzeugten Kindknoten bilden die Menge N_{i+1}. Die Berechnung endet, wenn in einem Berechnungsschritt keine neuen Kindknoten mehr erzeugt werden können. Der Vorgang der *top-down* Berechnung einer Hierarchie unterteilt sich somit in die sukzessive Berechnung von Hierarchiestufen immer größerer Tiefe, solange die neu entstehende Hierarchiestufe Knoten enthält. Die Berechnung einer Hierarchiestufe der Tiefe $i+1$ aus der der Tiefe i besteht dabei aus der sukzessiven Aufteilung der Knoten n in Kindknoten $n_1, \ldots, n_k$ mit den zugeordneten Flächenmengen und Hüllkörpern. Die Berechnung der Kindknoten $n_1, \ldots, n_k$ eines Knotens n wird durchgeführt, indem die Flächenmenge F_n in k Flächenmengen $F_1, \ldots, F_k$ unterteilt wird und Hüllkörper für diese berechnet werden. In der Art dieses Aufteilungsschrittes unterscheiden sich die verschiedenen *top-down*-Verfahren.

3.3.2 *Bottom-up* Berechnung von Hüllkörperhierarchien

Der Vorgang der *bottom-up* Berechnung von Hierarchien startet bei den Blattknoten des Baumes. Die zu approximierende Flächenmenge F wird dabei mit Hilfe einer Menge von Hüllkörpern überdeckt, z.B. indem jede Fläche durch einen Hüllkörper eingeschlossen wird. Daraus ergeben sich die Blattknoten, denen ein Hüllkörper und die überdeckte Teilflächenmenge der gesamten Flächenmenge zugeordnet ist. Sie bilden die initiale Menge N_0 der Knoten, die noch zusammengefaßt werden müssen. In jedem Schritt der Hierarchieberechnung werden die Knoten N_i zu Knoten N_{i+1} zusammengefaßt. Ein einzelner Zusammenfassungsschritt besteht daraus, daß Knoten $n_1, \ldots, n_k$ aus N_i zu einem Knoten n zusammengefaßt werden, der in N_{i+1} aufgenommen wird. n wird die Vereinigung $\cup_{j=1}^{k} F_j$ der Flächenmengen der Kindknoten zugeordnet und ein Hüllkörper dafür berechnet. Zusätzlich zu den auf diese Art und Weise neu erzeugten Knoten werden die Knoten n aus N_i, die nicht zu neuen Knoten zusammengefaßt werden konnten, in N_{i+1} aufgenommen. Die Berechnung endet, wenn die neu entstehende Menge N_r nur noch einen Knoten enthält. Dieser bildet den Wurzelknoten des Baumes.

3.4 Statische Gütemaße für Hüllkörperhierarchien

Die in der Literatur präsentierten Verfahren orientieren sich im wesentlichen an der Praktikabilität der Hierarchiegenerierung. Die *statische Güte* der erzeugten Hierarchie kann erst a posteriori ermittelt werden. Als *statische Güte* bezeichnen wir dabei ein Qualitätsmaß, das sich alleine aus den überdeckten Flächenmengen und den überdeckenden Hüllkörpern berechnen läßt.

Wir wollen an dieser Stelle den umgekehrten Weg gehen, zunächst *statische Gütemaße* für Hierarchieteile ausarbeiten, diese als Optimierungskriterien für die Berechnung von Hüllkörperhierarchien benutzen und danach Verfahren entwickeln, die entsprechend dieser Optimierungskriterien möglichst gute Hierarchien berechnen.

Das Wesen einer Hüllkörperhierarchie besteht darin, daß für immer kleinere Teile einer Flächenmenge F Hüllkörper berechnet sind. Dabei überdecken die Hüllkörper der Knoten der Tiefe i zusammen mit den Hüllkörpern aller Blattknoten der Tiefe $< i$ die gesamte Flächenmenge F. Die statischen Approximationseigenschaften einer Hüllkörperhierarchie ändern sich somit von Stufe zu Stufe. Jeder dieser Hierarchiestufen kann daraus eine *statische Güte* zugeordnet werden.

3.4.1 Statische Gütemaße für Hierarchiestufen

Die statischen Approximationseigenschaften einer Hierarchiestufe beruhen auf den statischen Approximationseigenschaften der in ihr enthaltenen Knoten N. Sind diese bekannt, so kann aus diesen eine *statische Güte* $G : \mathcal{N} \longrightarrow \mathbb{R}$ für die gesamte Hierarchiestufe berechnet werden. Als Gütefunktionen eignen sich dabei

- der Mittelwert der Gütemaße der Knoten

$$G(N) = \frac{1}{|N|} \cdot \sum_{n \in N} G(n)$$

oder

- das Maximum der Gütemaße der Knoten

$$G(N) = \max_{n \in N} G(n).$$

Die Knoten einer Hierarchiestufe ergeben sich durch die Aufteilung eines Knotens n der darüberliegenden Hierarchiestufe in Knoten $n_1, \ldots, n_k$. Eine Hierarchiestufe ist somit gleichzeitig die Vereinigung N der bei der Knotenaufteilung der Knotenmenge der darüberliegenden Hierarchiestufe entstehenden Knotenmengen N_i. Eine alternative Beschreibung der Gütefunktion einer Hierarchiestufe ist über diese Knotenmenge möglich. Die Gütefunktion für die *statische Güte* einer Hierarchiestufe $G : \mathcal{N} \longrightarrow \mathbb{R}$ hat für $N = \cup_{i=1}^{K} N_i$ folgendes Aussehen:

- Mittelwert der Gütemaße der Knotenmengen

$$G(N) = \frac{1}{K} \cdot \sum_{i=1}^{K} G(N_i)$$

- Maximum der Gütemaße der Knotenmengen

$$G(N) = \max_{i=1,\ldots,K} G(N_i)$$

3.4.2 Statische Gütemaße für Knotenmengen

Betrachten wir eine gesamte Hierarchiestufe, so ergibt sich durch das Gütemaß eine Aussage über die Approximationseigenschaften aller darin enthaltenen Knoten. Die Approximationseigenschaften können jedoch von Knoten zu Knoten variieren, da Teile der gesamten Flächenmenge gut und andere weniger gut mit Hilfe der verfügbaren Hüllkörper approximiert werden können. Während des Prozesses der Hierarchiegenerierung muß bei den *top-down* Verfahren die Flächenmenge F eines Knotens n in mehrere Flächenmengen $F_1, \ldots, F_k$ unterteilt werden, für die überdeckende Hüllkörper berechnet werden. Bei den *bottom-up* Verfahren werden mehrere Flächenmengen $F_1, \ldots, F_k$ zu einer Flächenmenge F zusammengefaßt. Um die Güte für diesen Aufteilungs- bzw. Zusammenfassungsschritt beurteilen zu können, muß die *statische Güte* eines Knotens mit der einer Knotenmenge verglichen werden. Zur Beurteilung der *statischen Güte* $G : \mathcal{N} \longrightarrow \mathbb{R}$ einer Knotenmenge N bieten sich analog zu der Hierarchiestufe

- der Mittelwert der Gütemaße der Knoten

$$G(N) = \frac{1}{|N|} \cdot \sum_{n \in N} G(n)$$

oder

- das Maximum der Gütemaße der Knoten

$$G(N) = \max_{n \in N} G(n)$$

an.

3.4.3 Statische Gütemaße für Knoten

Statische Gütemaße eines Knotens n beschreiben die Approximationseigenschaften des Hüllkörpers h_n für die überdeckte Flächenmenge F_n. Diese bestehen bezogen auf die Kollisionserkennung aus zwei Teilen:

1. Statisch geometrische Güte
2. Statisch kollisionstechnische Güte

Die *statisch geometrische Güte* beschreibt das Verhältnis des Hüllkörpers zu der umhüllten Flächenmenge, wohingegen die *statisch kollisionstechnische Güte* nur vom Typ (nicht der speziellen Instanz) des Hüllkörpers abhängt. D.h. werden nur Hüllkörper eines Types betrachtet, z.B. nur Kugeln, so kann bei der Berechnung von *statischen Gütemaßen für Knoten* auf die Berücksichtigung der *statisch kollisionstechnischen Güte* verzichtet werden. Sollen hingegen unterschiedliche Hüllkörpertypen verwendet werden, so müssen neben der *statisch geometrischen Güte* ihre *statisch kollisionstechnische Güte* berücksichtigt werden, um sie vergleichbar zu machen.

Das gesamte *statische Gütemaß* eines Knotens n ergibt sich dann als Produkt der *statisch geometrischen Güte* und der *statisch kollisionstechnischen Güte*, d.h.

$$G(n) = G_{Geometrie}(n) \cdot G_{Kollision}(Typ(h_n)).$$

Statisch geometrische Gütemaße für Knoten

Ein *statisch geometrisches Gütemaß* für einen Knoten n beschreibt die geometrischen Approximationseigenschaften des Hüllkörpers h_n für die überdeckte Flächenmenge F_n. Einfache Gütemaße orientieren sich allein an den geometrischen Eigenschaften des Hüllkörpers. Dafür sind *Volumen*, *Oberfläche* oder *Durchmesser* der Hüllkörper geeignet.

Genauere Gütemaße können durch die Betrachtung sowohl des Hüllkörpers als auch der umhüllten Flächenmenge berechnet werden. Der geometrische Fehler, der durch die Approximation der Flächenmenge F_n durch den Hüllkörper h_n entsteht, kann durch den maximalen Abstand von der Oberfläche des Hüllkörpers zu der umhüllten Flächenmenge beschrieben werden. Dies entspricht dem *gerichteten Hausdorff-Abstand* der Hüllkörperoberfläche zu der Flächenmenge F_n unter Benutzung des euklidischen Abstandsmaßes.

Formal ist der *gerichtete Hausdorff-Abstand* von der Menge A zu der Menge B mit dem Abstandsmaß d definiert durch

$$hd(A, B) := \max_{a \in A} \min_{b \in B} d(a, b)$$

Anschaulich bedeutet der *gerichtete Hausdorff-Abstand* von A zu B, daß es kein Element $a \in A$ gibt, dessen minimaler Abstand zu B größer ist als $hd(A, B)$. Auf den Fall des Hüllkörpers h_n und der Flächenmenge F_n übertragen bedeutet dies, daß es keinen Punkt des Hüllkörpers gibt, dessen minimaler Abstand zur Flächenmenge größer ist als $hd(h_n, F_n)$.

Statisch kollisionstechnische Gütemaße für Knoten

Statisch kollisionstechnische Gütemaße beschreiben die Eigenschaften von Hüllkörpertypen in der Kollisionserkennung. Sie sind nur abhängig vom Typ des Hüllkörpers nicht von der speziellen Instanz und dienen dazu, die geometrischen Eigenschaften der verschiedenen Hüllkörpertypen in Bezug auf den Einsatz in der Kollisionserkennung vergleichbar zu machen. Dazu bietet sich die Komplexität der statischen Kollisionserkennung der einzelnen Hüllkörper untereinander an. Seien $H_1, \ldots, H_k$ die verwendeten Hüllkörpertypen, dann ergibt sich daraus eine symmetrische *Gütematrix* der einzelnen Hüllkörpertypen

	H_1	$\cdots$	H_k
H_1	$c(H_1, H_1)$	$\cdots$	$c(H_1, H_k)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
H_k	$c(H_k, H_1)$	$\cdots$	$c(H_k, H_k)$

Die Komplexitäten eines Hüllkörpertyps H_i in der *Gütematrix* müssen zu einem Wert für die *statisch kollisionstechnische Güte* von H_i zusammengefaßt werden $(i = 1, \ldots, k)$. Dazu bieten sich

- der Mittelwert der Gütewerte

$$G_{Kollision}(H_i) = \frac{1}{k} \cdot \sum_{j=1}^{k} c(H_i, H_j)$$

oder

- das Maximum der Gütewerte

$$G_{Kollision}(H_i) = \max_{j=1,\ldots,k} c(H_i, H_j)$$

an.

3.5 *Bottom-up* Berechnung von Hüllkörperhierarchien

Wir wollen uns im Rahmen dieser Arbeit nur mit der *top-down* Berechnung von Hüllkörperhierarchien beschäftigen, jedoch können die zuvor vorgestellten Gütemaße auch im Rahmen von *bottom-up*-Verfahren zur Berechnung von Hüllkörperhierarchien verwendet werden.

Eine *bottom-up* Berechnung von Hüllkörperhierarchien startet mit einer initialen feinsten Überdeckung der gesamten Flächenmenge F durch Hüllkörper. In jedem Schritt der

Berechnung werden aus der Menge der derzeit verfügbaren Knoten N_i eine Teilmenge $\{n_1, \ldots, n_k\}$ zu einem neuen Knoten zusammengefaßt (*Zusammenfassungsschritt*). Die *statische Güte* dieser Zusammenfassung kann dadurch beurteilt werden, daß die *statische Güte* des neu entstandenen Knotens mit der der zusammengefaßten Knotenmenge verglichen wird. Das Ergebnis dieses Vergleiches kann dazu benutzt werden zu enscheiden, ob ein vorgeschlagener *Zusammenfassungsschritt* tatsächlich durchgeführt werden soll.

In diesem Zusammenhang kann die *statische Güte* von Knoten, Knotenmengen und Hierarchiestufen allgemein dazu benutzt werden, das Vorgehen eines *bottom-up* Berechnungsverfahren für Hierarchien zu steuern.

3.6 *Top-down* Berechnung von Hüllkörperhierarchien

In diesem Abschnitt werden wir nun verschiedene *top-down* Berechnungsverfahren für Hüllkörperhierarchien vorstellen, die versuchen entsprechend der vorgegebenen Optimierungskriterien möglichst gute Hierarchien zu berechnen.

Betrachten wir den Vorgang der *Top-down* Berechnung von Hierarchien allgemein, so besteht ein *Großschritt* des Verfahrens darin, aus der Menge der Knoten N_i der Tiefe i die Menge der Knoten N_{i+1} der Tiefe $i + 1$ zu berechnen. Jeder *Großschritt* besteht dabei aus $|N_i|$ *Kleinschritten*, in denen jeweils ein Knoten n der Knotenmenge N_i in k Knoten $n_1, \ldots, n_k$ aufgeteilt wird, die der Menge N_{i+1} angehören, oder entschieden wird, daß der Knoten nicht weiter aufgeteilt werden kann. Diesen Knoten $n_1, \ldots, n_k$ sind Flächenmengen $F_1, \ldots, F_k$ mit $\cup_i^k F_i = F_n$ sowie Hüllkörper $h_{n_1}, \ldots, h_{n_k}$ zugeordnet. Sind alle Knoten aus N_i bearbeitet, liegt die Menge N_{i+1} vollständig vor und ein neuer *Großschritt* kann durchgeführt werden. Das Verfahren beginnt mit dem Wurzelknoten w, der die gesamte Flächenmenge F enthält, für die ein Hüllkörper h_w berechnet wird. Dieser bildet das einzige Element der Menge N_0. Nun werden solange *Großschritte* durchgeführt, bis die entstehende Knotenmenge N_i leer ist.

Die daraus resultierende Kontrollstruktur zur *Top-down* Berechnung von Hierarchien ist in Algorithmus *TopDownBerechnungHierarchie* in Abbildung 3.1 veranschaulicht.

Die *top-down* Berechnungsverfahren für Hüllkörperhierarchien unterscheiden sich in der Art und Weise der Aufteilung eines Knotens und der dabei verwendeten Typen von Hüllkörpern. Bei der Berechnung der Aufteilung eines Knotens können generell solche mit *festem* und *variablem* Verzweigungsgrad unterschieden werden. Z.B. liefert das Verfahren von GOTTSCHALK [GLM96] eine binäre Hüllkörperhierarchie. Bei den Typen der verwendeten Hüllkörper können *homogene* und *heterogene* Hüllkörperhierarchien unterschieden werden. *Homogene* Hierarchien enthalten nur einen einzigen Hüllkörpertyp, *heterogene* verschiedene. Das Verfahren von GOTTSCHALK [GLM96] liefert *homogene* Hierarchien, wohingegen das von BAREQUET ET.AL. [BC+96] bedingt *heterogene* Hierarchien dadurch berechnen kann, daß unterschiedliche Hüllkörpertypen in verschiedenen Hierarchiestufen verwendet werden können.

Algorithmus *TopDownBerechnungHierarchie*(F)
Eingabe: F Flächenmenge
Ausgabe: H Hüllkörperhierarchie für F

$w \leftarrow (F, Huellkoerper(F))$
$N_0 \leftarrow \{w\}$
$H \leftarrow w$
$i \leftarrow 0$
while $(N_i \neq \emptyset)$
  do $N_{i+1} \leftarrow \emptyset$
    for n **in** N_i
    do $((F_1, h_1), \ldots, (F_k, h_k)) \leftarrow Aufteilung(F_n)$
      for $i \leftarrow 1$ **to** k
      do $N_{i+1} \leftarrow N_{i+1} \cup \{(F_i, h_i)\}$
        $Kind_i(n) \leftarrow (F_i, h_i)$
    $i \leftarrow i + 1$
return H

Abbildung 3.1: ALGORITHMUS: TOP-DOWN BERECHNUNG VON HÜLLKÖRPERHIERARCHIEN

In der Folge werden wir Knotenaufteilungsverfahren für *festen* und *variablen* Verzweigungsgrad vorstellen, die wahlweise *homogene* oder *heterogene* Hierarchien berechnen können und dadurch im Gegensatz zu den bisherigen Verfahren eine größtmögliche Flexibilität bei der Berechnung von Hüllkörperhierarchien aufweisen.

3.7 Aufteilung von Knoten

Ein Kleinschritt der *top-down* Berechnung von Hüllkörperhierarchien besteht darin, einen Knoten $n = (F_n, h_n)$ mit Flächenmenge F_n und Hüllkörper h_n in eine Menge von Knoten $n_1, \ldots, n_k$ mit Flächenmengen $F_1, \ldots, F_k$ und Hüllkörpern $h_1, \ldots, h_k$ aufzuteilen oder zu entscheiden, daß eine solche Aufteilung nicht mehr möglich ist.

Mögliche Gründe dafür, daß eine Aufteilung von n nicht mehr möglich ist, sind:

- Maximale Hierarchietiefe ist erreicht
- F_n enthält nur noch eine minimale Anzahl von Flächen
- Aufteilungsalgorithmus kann F_n nicht weiter unterteilen

Bei den Aufteilungsverfahren für Knoten können – wie oben beschrieben – Verfahren mit *festem* und *variablem* Verzweigungsgrad unterschieden werden. Als Optimierungsziel für eine Knotenaufteilung mit *variablem* Verzweigungsgrad bietet sich die Verbesserung der Güte des aufzuteilenden Knotens $G(n)$ um einen festen Faktor $0 < \alpha \leq 1$ an. Eine Aufteilung sollte nicht durchgeführt werden, wenn die Verbesserung um den Faktor α nicht erreicht wird und die Anzahl der Flächen in der Flächenmenge F_n unter einem Schwellwert s_{min} liegt. Ist dies der Fall, liegt ein Knoten mit wenigen Flächen vor, die nur schlecht zu unterteilen sind. Werden sie dennoch unterteilt, ergibt sich eine lokale Hüllkörperhierarchie, die von Stufe zu Stufe nur eine geringe Verbesserung erzielt, gleichzeitig befinden wir uns nahe den Blättern des Baumes. Im Kollisionserkennungsprozeß würden dadurch eine Menge von Hüllkörpertests erzeugt, die wahrscheinlich nicht zur Lösung des Kollisionserkennungsproblems beitragen, sondern die Lösung wird erst durch Betrachtung der überdeckten Flächen selbst berechnet. Diese Hüllkörpertests können eingespart werden, wenn in einem solchen Fall die Hüllkörperhierarchie endet und direkt auf die Betrachtung der Flächen übergegangen wird. Ist die Anzahl der Flächen jedoch zu groß, würden sehr viele Basiskollisionstests durchgeführt, die die Kollisionserkennung sehr langsam machen würden. In diesem Fall sollte die Aufteilung mit dem bestmöglichen Aufteilungsergebnis vorgenommen werden in der Hoffnung, daß das schlechte Aufteilungsverhalten der Flächenmenge F_n nur lokal ist.

Die im Rahmen dieser Arbeit beschriebenen Verfahren *variablen* Verzweigungsgrades entstehen dadurch, daß die Verfahren mit *festem* Verzweigungsgrad iterativ mit steigendem Verzweigungsgrad angewendet werden, solange das Optimierungsziel oder ein maximaler Verzweigungsgrad v_{max} nicht erreicht ist.

Die Kontrollstruktur für ein Aufteilungsverfahren *variablen* Verzweigungsgrades ergibt sich aus dem Algorithmus *KnotenaufteilungVariablerGrad* in Abbildung 3.2.

Für die Aufteilung eines Knotens mit festem Verzweigungsgrad werden wir in der Folge sechs verschiedene Verfahren vorstellen:

1. *Branch-and-Bound*-Verfahren
2. Modifiziertes Hochbaum-Shmoys-Verfahren
3. Heuristik mit Zwischenoptimierung
4. Heuristik ohne Zwischenoptimierung
5. Modifiziertes Gottschalk-Verfahren
6. Modifiziertes Zachmann-Verfahren

Allen Verfahren ist gemeinsam, daß sie für eine Flächenmenge F und einen Aufteilungsgrad d eine Unterteilung von F in Flächenmengen $F_1, \ldots, F_d$ mit zugehörigen Hüllkörpern $h_1, \ldots, h_d$ berechnen können, wobei sowohl eine *homogene* Aufteilung, d.h. unter Benutzung nur eines Hüllkörpertyps, oder eine *heterogene* Aufteilung, d.h. unter Benutzung verschiedener Hüllkörpertypen, berechnet werden kann.

Algorithmus *KnotenaufteilungVariablerGrad*$(n, v_{max}, s_{min}, \alpha)$

Eingabe:

$n = (F_n, h_n)$	Knoten mit Flächenmenge F_n und Hüllkörper h_n
v_{max}	maximaler Verzweigungsgrad
s_{min}	Schwellwert für Anzahl der in jedem Fall zu unterteilenden Flächen
α	Faktor der Verbesserung

Ausgabe: $\mathcal{F} = ((F_1, h_1), \ldots, (F_k, h_k))$ Aufteilung der Flächenmenge F_n

$\mathcal{F}_{min} \leftarrow \emptyset$
$g_{min} \leftarrow \infty$
for $i \leftarrow 2$ **to** v_{max}
do $((F_1, h_1), \ldots, (F_i, h_i)) \leftarrow KnotenAufteilungFesterGrad(F_n, i)$
  $g \leftarrow G(\{(F_1, h_1), \ldots, (F_i, h_i)\}$
  if $(g \leq \alpha \cdot G(n))$
  then return $((F_1, h_1), \ldots, (F_i, h_i))$
  else **if** $(g < g_{min})$
    then $g_{min} \leftarrow g$
      $\mathcal{F}_{min} \leftarrow ((F_1, h_1), \ldots, (F_i, h_i))$
if $(|F_n| < s_{min})$
then return $\emptyset$
return $\mathcal{F}_{min}$

Abbildung 3.2: ALGORITHMUS: KNOTENAUFTEILUNG VARIABLER VERZWEIGUNGSGRAD

3.7.1 *Branch-and-bound*-Verfahren

Ziel einer optimalen Aufteilung eines Knotens n mit Flächenmenge F_n und Hüllkörper h_n ist es, n so in Knoten $n_1, \ldots, n_d$ mit Flächenmengen $F_1, \ldots, F_d$ und überdeckenden Hüllkörpern $h_1, \ldots, h_d$ zu unterteilen, daß das Gütemaß $G(\{n_1, \ldots, n_d\})$ für die Aufteilung optimiert wird.

Für den Fall einer *homogenen* Aufteilung unter Verwendung von Kugeln als Hüllkörpern, für die Volumen, Oberfläche oder Durchmesser minimiert werden soll, ist das Problem der Flächenaufteilung in der Abwandlung, daß eine gegebene Punktmenge im $\mathbb{R}^d$ mit Hilfe von k minimalen Kugeln überdeckt werden soll, unter dem Namen *euclidean k-center problem* ein in der Literatur wohluntersuchtes Problem. AGARWAL und SHARIR geben in [AS96] einen guten Überblick darüber. Komplexitätstheoretisch ist das *euclidean k-center problem* NP-vollständig für jede Dimension $d \geq 2$, wie in [FPT81, MS81, MS84] gezeigt wird.

GONZALEZ in [Go85], sowie HOCHBAUM und SHMOYS in [HS85, HS86] beschreiben Heuristiken, deren Ergebnis höchstens um den Faktor 2 schlechter ist als die optimale Lösung. FEDER und GREEN zeigen in [FG88], daß es keine Heuristik für dieses Problem geben kann, die ein Ergebnis garantiert, das weniger als der Faktor 1.822 schlechter ist als die optimale Lösung. Wir werden auf das Verfahren von HOCHBAUM und SHMOYS in Zusammenhang mit dem *modifizierten* HOCHBAUM-SHMOYS-*Verfahren* und der *Heuristik ohne Zwischenoptimierung* zurückkommen.

Die Aufteilung von F in d Teilmengen, so daß ein Gütemaß für die Aufteilung minimiert wird, ist ein klassisches Partitionierungsproblem, zu dessen Lösung sich *Branch-and-bound*-Verfahren anbieten. BOMZE und GROSSMANN beschreiben *Branch-and-bound* in [BG93] wie folgt:

> *Branch-and-bound* stellt die wesentliche Methode zur exakten Lösung von diskreten Optimierungsproblemen dar. Die Grundidee dabei ist, eine partielle Enumeration des zulässigen Bereichs. Dabei versuchen wir schrittweise Probleme mit einem kleineren zulässigen Bereich zu lösen, die dann ganze Teile des zulässigen Bereichs als mögliche Lösungen ausschließen.

Ausgangspunkt ist daher die Definition einer *Verzweigungsoperation*. Diese beschreibt, wie der Zulässigkeitsbereich D in Teilmengen $D_1, \ldots, D_k$ zerlegt wird, so daß $\cup_{i=1,\ldots,k} D_i = D$. Durch diese Verzweigungsregel wird für das gesamte Problem ein Entscheidungsbaum aufgebaut. Jeder Knoten des Entscheidungsbaums ist ein Optimierungsproblem, wobei mit zunehmender Tiefe des Knotens der Zulässigkeitsbereich des Optimierungsproblems immer weiter abnimmt und in den Blättern nur noch einelementig ist. Somit entspricht die vollständige Traversierung des Baumes der Enumeration des Definitionsbereichs. Diese vollständige Traversierung soll verhindert werden und gleichzeitig eine optimale Lösung des Problems gefunden werden. Dazu berechnen wir für jeden Knoten eine *untere* und *obere* Schranke für den optimalen Wert der Zielfunktion für alle Probleme innerhalb des darunterliegenden Unterbaumes. Für die *obere* Schranke muß zusätzlich gelten, daß der Wert der Zielfunktion für einen zulässigen Punkt tatsächlich angenommen wird. Mit Hilfe dieser Schranken können nun Regeln hergeleitet werden, die das Traversieren von Teilbäumen des Entscheidungsbaumes überflüssig machen.

1. *Untere Schranken*
   Wenn $us(O_i)$ eine untere Schranke für das partielle Optimierungsproblem O_i ist, und eine zulässige Lösung x_0 mit $f(x_0) \leq us(O_i)$ existiert, so kann der Entscheidungsbaum an dem Knoten (O_i) beschnitten werden.

2. *Obere Schranken*
   Wenn für ein partielles Problem O_i gilt, daß die untere Schranke $us(O_i)$ gleich der oberen $os(O_i)$ ist, d.h. $us(O_i) = os(O_i)$, so ist dies die Lösung des Optimierungsproblems O_i und der Entscheidungsbaum kann an dem Knoten (O_i) beschnitten werden.

3. *Dominanz*
   Ein partielles Problem O_j dominiert das Problem O_i ($O_j \lhd O_i$), falls für die optimalen

Lösungen $f^\star(P_j) \leq f^\star(P_i)$ gilt. Diese Regel kann auch mit Hilfe der oberen und unteren Schranken abgeschwächt beschrieben werden: $os(O_j) \leq us(O_i) \Longrightarrow f^\star(P_j) \leq os(O_j) \leq us(O_i) \leq f^\star(P_i) \Longrightarrow (O_j \lhd O_i)$

Mit diesen Regeln ergibt sich die allgemeine algorithmische Struktur von *Branch-and-bound* Verfahren im Algorithmus *Branch-and-bound*, der in Abbildung 3.3 dargestellt ist.

Algorithmus $Branch\text{-}and\text{-}bound(f, M, V, T, us, os, dom)$

Eingabe:	f	zu minimierende Zielfunktion
	M	Zulässige Lösungen
	V	Verzweigungsregel
	T	Traversierungsregel
	us	Verfahren zur Bestimmung unterer Schranken
	os	Verfahren zur Bestimmung oberer Schranken
	dom	Verfahren zur Bestimmung dominierender Probleme
Ausgabe:	$f^\star$	optimaler Zielfunktionswert
	$x^\star \in M$	optimale Lösungspunkt

(* $O_0 : \min_{x \in M} f(x)$ sei das initiale Optimierungsproblem *)
$O_{akt} \leftarrow \{O_0\}$
$f^\star \leftarrow \infty$
while $O_{akt} \neq \emptyset$
  do $O_i \leftarrow T(O_{akt})$
    if $os(O_i) < f^\star$
    then $f^\star \leftarrow os(O_i)$
      $x^\star \leftarrow arg(os(O_i))$
    if $(us(O_i) = os(O_i))$
    then $O_{akt} \leftarrow O_{akt} \setminus O_i$
    else
      if $us(O_i) > f^\star$
      then $O_{akt} \leftarrow O_{akt} \setminus O_i$
      else
        if $dom(O_{akt}, O_i)$
        then $O_{akt} \leftarrow O_{akt} \setminus O_i$
        else
          $O_{akt} \leftarrow O_{akt} \cup V(O_i) \setminus \{O_i\}$
  return $(f^\star, x^\star)$

Abbildung 3.3: ALGORITHMUS: BRANCH-AND-BOUND

Dieses generelle Verfahren muß nun auf unser spezielles Problem angepaßt werden. Dazu müssen wir nur die Eingabeparameter für unser Partitionierungsproblem spezifizieren.

- *Zulässigkeitsbereich*
  Sei F die zu partitionierende Flächenmenge, dann ist $Pa_d(F)$, die Menge aller d-nären Partitionierungen von F, der *Zulässigkeitsbereich.*

- *Verzweigungsregel*
  Eine Verzweigung innerhalb des Entscheidungsbaumes besteht darin, daß eine weitere Fläche einer der Partitionen fest zugeordnet wird. D.h. bei einer d-nären Partitionierung entstehen bei einer Verzweigung d neue Knoten. Dadurch verringert sich die Menge der variablen Flächen in jeder Stufe des Entscheidungsbaumes um eine Fläche.

  Entscheidend für die *oberen* und *unteren* Schranken und damit die Anzahl der evaluierten Knoten des Entscheidungsbaumes ist dabei die Auswahl der Fläche bzgl. deren die Verzweigung vorgenommen wird. Dabei sind folgende Vorgehensweisen sinnvoll:

  - *zufällige Auswahl der Verzweigungsfläche*
    Der Vorteil dieser Vorgehensweise ist, daß die Verzweigungsfläche schnell gefunden wird und durch die zufällige Auswahl i.d.R. eine gute Wahl getroffen wird.
  - *Auswahl der Verzweigungsfläche, die die minimale untere Schranke maximal erhöht*
    Der Vorteil dabei ist, daß das Minimum der *unteren* Schranken der verzweigten Knoten maximal erhöht wird. Durch die schnelle Erhöhung der *unteren* Schranken kann eine mögliche Beschneidung eines Unterbaumes mit Hilfe des *Dominanzkriteriums* frühestmöglich erkannt werden.
    Die Verzweigungsfläche wird berechnet, indem jede der verbleibenden variablen Flächen in jede der Partitionen eingefügt wird und für jede Fläche die minimale Verschlechterung der Partitionierungsgüte berechnet wird. Die Fläche, die diese minimale Verschlechterung der Partitionierungsgüte maximiert, wird als Verzweigungsfläche benutzt.

- *Traversierungsregel*
  Bei der Auswahl des nächsten zu bearbeitenden Knotens aus der Menge der offenen Knoten können verschiedene Methoden angewendet werden. Folgende Vorgehensweisen sind sinnvoll:

  - *untere Schranke*
    Es wird derjenige Knoten ausgewählt, der die kleinste untere Schranke besitzt, da mit Hilfe der unteren Schranken im *Dominanzkriterium* Unterbäume beschnitten werden und daher die unteren Schranken möglichst schnell erhöht werden sollten.
  - *untere Schranke* $\cdot$ |*verteilte Flächen*| + *obere Schranke* $\cdot$ |*variable Flächen*|
    Mit Hilfe der Gewichtung der *unteren Schranke* mit Hilfe der bereits *verteilten Flächen* und der *oberen Schranke* mit Hilfe der *variablen Flächen* versuchen wir, das wahre Optimum des Unterbaumes zu schätzen.

  - *(obere Schranke - untere Schranke) · |variable Flächen|*
    Die Differenz zwischen *oberer* und *unterer* Schranke kann als Potential des Knotens betrachtet werden. Dieses absolute Potential wird mit der Anzahl der *variablen* Flächen gewichtet.

- *Untere Schranke*
  Durch das Hinzufügen von Flächen zu Partitionen wird die Güte des zugehörigen Hüllkörpers nur vergrößert. Daher ist die Zusammenfassung der Gütewerte der durch die bisher verteilten Flächen bestimmten Hüllkörper eine geeignete *untere* Schranke für den erreichbaren Zielfunktionswert des gesamten Unterbaumes. Diese Vorgehensweise ist nur bei der Verwendung einfacher Gütemaße wie Volumen, Durchmesser oder Oberfläche nicht jedoch bei der Verwendung des gerichteten Hausdorff-Abstands korrekt.

  Um die maximale Tiefe des Unterbaumes zu minimieren, kann für jeden Knoten die Menge der variablen Flächen berechnet werden, die bereits in den Hüllkörpern zur Berechnung der *unteren* Schranke enthalten sind (= *echt enthaltene Flächen*). Diese beeinflussen die optimale Lösung des Unterbaumes nicht, da es eine Partition gibt, in die sie ohne Verschlechterung des Zielfunktionswertes eingefügt werden können. Bzgl. dieser Flächen muß daher in der Folge nicht verzweigt werden und sie können damit aus der Menge der variablen Flächen entfernt werden.

- *Obere Schranke*
  Verteilen wir die weiteren Hüllkörper nach einem heuristischen Verfahren, erhalten wir eine zulässige Partitionierung innerhalb des Unterbaumes, die einer möglichst guten Partitionierung entspricht.

- *Dominanz*
  Das *Dominanzkriterium* wird mit Hilfe der *unteren* und *oberen Schranken* definiert.

Daraus ergibt sich der *Branch-and-Bound* Algorithmus *KnotenaufteilungBranchAndBound* für unser Problem der Flächenmengenpartitionierung in Abbildung 3.4.

3.7.2 Modifiziertes Hochbaum-Shmoys-Verfahren

Hochbaum und Shmoys haben in [HS85, HS86] eine einfache Heuristik für das *euclidean k-center problem* beschrieben. Diese bezieht sich auf den Fall, daß Punkte im $\mathbb{R}^d$ mit Hilfe von k Kugeln mit möglichsten kleinem Durchmesser zu überdecken sind. Das Verfahren *GreedyHeuristikHochbaumShmoysPunkte* ist in Abbildung 3.5 dargestellt.

Für den Fall von Kugeln als Hüllkörper kommt das Problem der Flächenmengenpartitionierung dem *euclidean k-center problem* sehr nahe mit dem Unterschied, daß anstelle von Punkten im $\mathbb{R}^3$ Flächen überdeckt werden müssen. Die Heuristik von Hochbaum

Algorithmus *KnotenaufteilungBranchAndBound*(n, d)

Eingabe: $n = (F_n, h_n)$ aufzuteilender Knoten
d Grad der Aufteilung

Ausgabe: $\mathcal{F}$ optimale Aufteilung der Flächenmenge F

$(*\ O = ((F_1, h_1), \ldots, (F_d, h_d), F_{variabel}, us, os)$ mit $\cup_{i=1}^{d} F_i = F_n\ *)$
$x^\star \leftarrow ((F_1, h_1), \ldots, (F_d, h_d)) \leftarrow Knotenaufteilung(n, d)$
$g^\star \leftarrow g \leftarrow G((F_1, h_1), \ldots, (F_d, h_d))$
$\mathcal{O}_{akt} \leftarrow \{((\emptyset, h_\emptyset)_1, \ldots, (\emptyset, h_\emptyset)_d, F_n, 0, g)\}$
while $\mathcal{O}_{akt} \neq \emptyset$
  do $O \leftarrow WaehleNaechstesOffenesProblem(\mathcal{O}_{akt})$
    if $(us(O) = os(O))$
    then $\mathcal{O}_{akt} \leftarrow \mathcal{O}_{akt} \setminus \{O\}$
    else **if** $(us(O) > g^\star)$
      then $\mathcal{O}_{akt} \leftarrow \mathcal{O}_{akt} \setminus \{O\}$
      else $f \leftarrow WaehleNaechsteVerzweigungsflaeche(O)$
        for $i \leftarrow 1$ **to** d
        do $g^{us} \leftarrow G((F_1, h_1), \ldots, (F_i \cup \{f\}, h_i(F_i \cup \{f\}), \ldots, (F_d, h_d))$
          $x \leftarrow ((F_1^{os}, h_1^{os}), \ldots, (F_d^{os}, h_d^{os})) \leftarrow$
          $Knotenaufteilung((F_1, h_1), \ldots, (F_i \cup \{f\}, h_i(F_i \cup \{f\}),$
          $\ldots, (F_d, h_d), F_{variabel} \setminus \{f\})$
          $g^{os} \leftarrow G((F_1^{os}, h_1^{os}), \ldots, (F_d^{os}, h_d^{os}))$
          if $(g^{os} < g^\star)$
          then $g^\star \leftarrow g^{os}$
            $x^\star \leftarrow x$
          $O_i \leftarrow ((F_1, h_1), \ldots, (F_i \cup \{f\}, h_i(F_i \cup \{f\})),$
          $\ldots, (F_d, h_d), F_{variabel} \setminus \{f\}, g^{us}, g^{os})$
        $\mathcal{O}_{akt} \leftarrow \mathcal{O}_{akt} \cup \left(\cup_{i=1}^{d} O_i\right) \setminus \{O\}$
return $(g^\star, x^\star)$

Abbildung 3.4: ALGORITHMUS: KNOTENAUFTEILUNG MIT *Branch-and-bound*-VERFAHREN

und SHMOYS geht so vor, daß sie für alle zu überdeckenden Punkte den minimalen Abstand zu einem Mittelpunkt einer überdeckenden Kugel zunächst unendlich setzt und in einem ersten Schritt einen beliebigen Punkt als Mittelpunkt für die erste Kugel auswählt. Für diesen Mittelpunkt wird der Abstand zu jedem zu überdeckenden Punkt berechnet. Nun wird sukzessive jeweils eine neue Kugel hinzugefügt. Dabei wird der Mittelpunkt einer neuen Kugel jeweils in den Punkt gelegt, der bisher den maximalen Abstand zu einem bestehenden Kugelmittelpunkt hat. Für alle anderen Punkte wird dann berechnet, ob durch das Einfügen der neuen Kugel sich der Abstand zum nächstgelegenen Kugelmittelpunkt reduziert.

Algorithmus *GreedyHeuristikHochbaumShmoysPunkte*(P, k)
Eingabe: P Punktmenge im $\mathbb{R}^d$
k Anzahl der Hüllkugeln
Ausgabe: S Menge der Mittelpunkte der Kugeln
1. **for** p **in** P
2. **do** maxDistance(p) $\leftarrow \infty$
3. **for** $i \leftarrow 1$ **to** k
4. **do** $\boldsymbol{s_i} \leftarrow \underset{p \in P}{\operatorname{argmax}} \operatorname{maxDistance}(p)$
5. **for** p **in** P
6. **do** maxDistance(p) $\leftarrow \min\{\operatorname{maxDistance}(p), d(\boldsymbol{s_i}, \boldsymbol{p})\}$
7. **return** $\{\boldsymbol{s_1}, \ldots, \boldsymbol{s_k}\}$

Abbildung 3.5: Algorithmus: Heuristik für *euclidean k-center problem* von Hochbaum und Shmoys

Beim Übergang von Punkten zu Flächen im $\mathbb{R}^3$ muß die Berechnung des minimalen Abstands einer Fläche zum Mittelpunkt einer Kugel modifiziert werden. Der Abstand einer Fläche f zum Mittelpunkt $\boldsymbol{c}$ einer Kugel ist gleich dem maximalen Abstand eines Eckpunktes der Fläche zu diesem Mittelpunkt, d.h.

$$d(\boldsymbol{c}, f) = \max_{\boldsymbol{v} \in Ecken(f)} d(\boldsymbol{c}, \boldsymbol{v})$$

Als ersten Mittelpunkt für eine Kugel sollte darüber hinaus der Eckpunkt einer Fläche gewählt werden, der möglichst am Rand der Flächenmenge liegt, da bei kleiner Anzahl von Hüllkugeln k die anfängliche Auswahl eines Punktes im Zentrum der Flächenmenge keine gute Verteilung von Hüllkugeln ergibt. Daher berechnen wir in einem ersten Schritt eine umhüllende Kugel für die gesamte Flächenmenge und benutzen als ersten Mittelpunkt für eine Hüllkugel den Eckpunkt, der den maximalen Abstand vom Mittelpunkt dieser Hüllkugel besitzt. Wird die Entfernungsinformation für eine Fläche modifiziert, so wird gleichzeitig festgelegt, in welcher Hüllkugel sie sich befindet. Dadurch wird neben den Mittelpunkten der Hüllkugeln eine Partitionierung der Flächenmenge berechnet. Mit diesen Modifikationen erhalten wir durch die Heuristik eine Partitionierung der Flächenmenge, die auf Kugeln als Hüllkörper ausgerichtet ist. Jedoch ist das Ergebnis der Partitionierung auch für die Verwendung anderer Hüllkörpertypen geeignet. Mit diesen Modifikationen ergibt sich der Algorithmus *GreedyHeuristikHochbaumShmoysFlächen* in Abbildung 3.6.

Betrachten wir die Heuristik, so sehen wir, daß sie auch auf teilweise vorberechneten Flächenmengenpartitionierungen eingestzt werden kann. Denn sei $((F_1, h_1), \ldots, (F_k, h_k))$ eine vorberechnete Partitionierung für Grad k und es sollen weitere Flächen F_{neu} eingefügt werden und insgesamt eine Partitionierung vom Grad $d \geq k$ berechnet werden, so werden als Mittelpunkte $\boldsymbol{c}_i$ die Mittelpunkte der Hüllkörper h_i benutzt und die Flächen von F_{neu}

Algorithmus *GreedyHeuristikHochbaumShmoysFlächen*(n, d)

Eingabe: $n = (F_n, h_n)$ aufzuteilender Knoten
d Grad der Aufteilung

Ausgabe: $((F_1, h_1), \ldots, (F_d, h_d))$ Aufteilung von n

$K \leftarrow Kugel(F_n)$
$\boldsymbol{c}_1 \leftarrow \text{argmax}_{\boldsymbol{v} \in Ecken(F_n)}\, d(\boldsymbol{c}(K), \boldsymbol{v})$
for f **in** F_n
do $d(f) \leftarrow d(\boldsymbol{c}_1, f)$
  $Partition(f) \leftarrow 1$
$F_1 \leftarrow F_n$
for $i \leftarrow 2$ **to** d
do $F_i \leftarrow \emptyset$
  $\boldsymbol{c}_i \leftarrow \text{argmax}_{\boldsymbol{v} \in Ecken(F_n)} \min_{1 \leq j \leq i-1} d(\boldsymbol{c}_j, \boldsymbol{v})$
  for f **in** F_n
  do if $(d(\boldsymbol{c}_i, f) < d(f))$
    then $d(f) \leftarrow d(\boldsymbol{c}_i, f)$
      $F_{Partition(f)} \leftarrow F_{Partition(f)} \setminus \{f\}$
      $F_i \leftarrow F_i \cup \{f\}$
      $Partition(f) \leftarrow i$
for $i \leftarrow 1$ **to** d
do $h_i \leftarrow Huellkoerper(F_i)$
return $((F_1, h_1), \ldots, (F_d, h_d))$

Abbildung 3.6: ALGORITHMUS: KNOTENAUFTEILUNG MIT MODIFIZIERTEM HOCHBAUM-SHMOYS-VERFAHREN

in die bestehende Partitionierung eingearbeitet. Nun werden entstprechend dem Verfahren die verbleibenden Mittelpunkte generiert, jedoch nur die Flächen aus F_{neu} bzgl. dieser Mittelpunkte klassifiziert und zwischen den erzeugten Flächenmengen ausgetauscht. Damit ist die Vorgehensweise auch dazu geeignet, *obere Schranken* für Probleme innerhalb des *Branch-and-bound* -Verfahrens zu berechnen.

3.7.3 Heuristik ohne Zwischenoptimierung

Die Erweiterung der Heuristik von HOCHBAUM und SHMOYS auf Flächen ist insofern unflexibel, als daß sie ein einmal gewähltes Zentrum für eine Flächenmenge der Partitionierung nicht mehr modifiziert. Jedoch könnte es sinnvoll sein, während der Berechnung der Partitionierung ein einmal gewähltes Zentrum zu verändern, wenn die bisherige Aufteilung der Flächenmenge dies nahelegt. Eine Möglichkeit der dynamischen Modifikation eines solchen Zentrum besteht darin, daß es in den jeweiligen Mittelpunkt der zugeordneten Flächenmenge gelegt wird. Dabei ist der Mittelpunkt einer Flächenmenge der Mittelwert der Mittelpunkte der Flächen in dieser Menge. Der Mittelpunkt einer Fläche ist der Mittelwert ihrer Eckpunkte.

Im ersten Schritt liegt damit das Zentrum im Mittelpunkt aller Flächen. In jedem neuen Schritt, wird dann die Fläche bestimmt, die den maximalen Abstand von einem bisher bestimmten Mittelpunkt definiert. Der Mittelpunkt dieser Fläche wird zum Zentrum der neuen Flächenmenge der Partitionierung. Für alle Flächen wird analog zu der bisherigen Vorgehensweise der minimale Abstand zu einem Zentrum bestimmt und die Fläche der zugehörigen Flächenmenge zugeordnet. Durch die Umordnung der Flächen werden die Zentren der Flächenmengen verändert. Diese werden nun iterativ solange neu bestimmt und die Flächen bzgl. dieser veränderten Zentren klassifiziert, bis keine Veränderungen mehr vorgenommen werden.

Mit dieser Vorgehensweise ergibt sich der Algorithmus *HeuristikOhneZwischenoptimierung* in Abbildung 3.7.

Auch diese Heuristik ist wie die modifizierte Heuristik von Hochbaum und Shmoys dazu geeignet vorberechnete Partitionierungen zu vervollständigen. Denn auch hier kann in einem ersten Schritt eine Flächenmenge F_{neu} in eine bestehende Partitionierung $((F_1, h_1), \ldots, (F_d, h_d))$ des Grades d eingearbeitet werden, wobei im Austauschschritt nur Flächen aus F_{neu} neu zugeordnet werden. Danach werden sukzessive die fehlenden Zentren bis zur Erreichung des Zielgrades $k \geq d$ hinzugefügt, wobei auch dabei nur die Flächen von F_{neu} zwischen den Flächenmengen getauscht werden. Daher ist diese Heuristik geeignet, *obere Schranken* für das *Branch-and-bound*-Verfahren zu berechnen.

3.7.4 Heuristik mit Zwischenoptimierung

Die beiden bisher vorgestellten Heuristiken berechnen die Knotenaufteilung ohne Berücksichtigung der Hüllkörper, die für die partitionierten Flächen berechnet werden. Der Vorteil dabei ist, daß der Zuordnungsschritt für eine Fläche zu einer Partitionen recht einfach ist und dadurch die Verfahren schnell sind, jedoch bringt eine Berücksichtigung der Auswirkung der Zuordnung einer Fläche zu einer Partition auf den zugehörigen Hüllkörper der Flächenmenge Vorteile bei der Steuerung der möglichst optimierten Verteilung von Flächen auf die einzelnen Partitionen.

Daraus läßt sich der Kern einer Heuristik ableiten, der diese Information zur Steuerung der Verteilung von Flächen auf die entstehenden Partitionen nutzt. Sei F die zu partitionierende Flächenmenge und d der Grad der Aufteilung.

> Wähle d Flächen aus F und berechne für diese Hüllkörper. Füge nun sukzessive die verbleibenden Flächen von F in die einzelnen Partitionen ein und zwar derart, daß in jedem Schritt die Güte der bisherigen Partitionierung möglichst wenig negativ beeinflußt wird.

Algorithmus *HeuristikOhneZwischenoptimierung*(n, d)

Eingabe: $n = (F_n, h_n)$ aufzuteilender Knoten
d Grad der Aufteilung

Ausgabe: $((F_1, h_1), \ldots, (F_d, h_d))$ Aufteilung von n

```
F_1 ←F_n  c_1 ←0
for f in F_n
do c_1 ←c_1 + Mittelpunkt(f)/|F_n|
for f in F_n
do d(f) ←d(c_1, f)  Partition(f) ←1
for i ←2 to d
do modifiziert ←FALSE
    F_i ←∅  c_i ←argmax_{f∈F_n} d(f)
    for f in F_n
    do if (d(c_i, f) < d(f))
        then d(f) ←d(c_i, f)
             F_{Partition(f)} ←F_{Partition(f)} \ {f}
             F_i ←F_i ∪ {f}  Partition(f) ←i
             modifiziert ←TRUE
    while (modifiziert)
     do modifiziert ←FALSE
        for j ←1 to i
        do c_j ←0
            for f in F_j
            do c_j ←c_j + Mittelpunkt(f)/|F_j|
        for f in F_n
        do d(f) ←d(c_{Partition(f)}, f)
            for j ←1 to i
            do if (d(c_j, f) < d(f))
                then d(f) ←d(c_j, f)
                     F_{Partition(f)} ←F_{Partition(f)} \ {f}
                     F_j ←F_j ∪ {f}  Partition(f) ←j
                     modifiziert ←TRUE
for i ←1 to d
do h_1 ←Huellkoerper(F_i)
return ((F_1, h_1), ..., (F_d, h_d))
```

Abbildung 3.7: Algorithmus: Knotenaufteilung mit Heuristik ohne Zwischenoptimierung

Die Idee, die hinter dieser Vorgehensweise steht, ist die, daß ausgehend von einer initialen d-nären Partitionierung und deren zugehörigen Hüllkörpern, die Hüllkörper durch sukzessives Hinzufügen von Flächen zu einem möglichst optimalen Hüllkörperkomplex heranwachsen. In jedem Iterationsschritt wird dabei für jede der aktuellen d Flächenmengen die Veränderung der Güte beim Hinzufügen einer neuen Fläche berechnet. Die Fläche wird dann in die Partition eingefügt, bei der die Güte der enstehenden Partitionierung am besten ist. Das Ergebnis der Heuristik hängt dabei von

1. der Wahl der initialen d-nären Partitionierung und
2. der Einfügereihenfolge der verbleibenden Flächen

ab.

Berechnung der initialen Partitionierung

Bei der Berechnung einer initialen d-nären Partitionerung wird eine Menge von d Flächen gesucht, die in der optimalen Partitionerung in unterschiedlichen Flächenmengen der Partitionerung liegen und das Heranwachsen der zugehörigen Hüllkörper zu einem möglichst optimalen Hüllkörperkomplex ermöglichen. Geeignete Verfahren hierzu sind:

1. *Zufällige Auswahl von d initialen Flächen*
2. *Flächen mit minimalem Abstand von der Oberfläche der umhüllenden Kugel*
   Für die Flächenmenge F wird die umhüllende Kugel berechnet und für eine Anzahl von zufälligen Beispielpunkten auf der Oberfläche die Fläche mit minimalem Abstand berechnet. Dadurch ergibt sich eine Menge $F_{min} = \{f_i | i = 1, \ldots, |\text{Punkte}|\}$ von minimalen Flächen. Aus dieser Menge wird die Menge $F_d = \{f_i | i = 1, \ldots, d\} \subseteq F_{min}$ berechnet, die den minimalen Abstand zwischen zwei Flächen aus F_d maximiert. Das Verfahren schlägt fehl, wenn keine d unterschiedlichen minimalen Flächen bestimmt werden können.
3. *Minimaler Abstand von den Eckpunkten des umhüllenden Isokaeders*
   Für die Flächenmenge F wird die kleinste umhüllende Kugel berechnet und die 12 Eckpunkte des in dieser Kugel eingeschriebenen *Isokaeders* betrachtet. Für jeden dieser 12 Punkte $\boldsymbol{p}_i$ wird die Fläche $f_i \in F$ berechnet, die von $\boldsymbol{p}_i$ den minimalen Abstand besitzt. Aus dieser Menge wird wie oben die Menge F_d berechnet.
4. *Raumpartitionierung*
   Für die Flächenmenge F wird die umhüllende Box $B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r})$ mit Orientierung $\boldsymbol{R}$, Mittelpunkt $\boldsymbol{c}$ und Radiusvektor $\boldsymbol{r}$ berechnet. In dieser Box werden Punkte $\boldsymbol{p}_i$, $1 \leq i \leq d$ berechnet und für jeden der Punkte $\boldsymbol{p}_i$ die Fläche $f_i \in F$, die den minimalen Abstand zu $\boldsymbol{p}_i$ hat. Diese bilden die initiale Partitionierung. Die Berechnung der $\boldsymbol{p}_i$ geht aus der Tabelle 3.1 auf Seite 53 hervor.

Für die so ermittelten Flächen werden optimierte Hüllkörper berechnet. Zusammen mit diesen dienen die Flächen als initiale Partitionierung für die weitere Verteilung der Flächen.

Grad	Berechnung von p_i
2	$max = \mathrm{argmax}_{i=1,2,3}\, \boldsymbol{r}_i$ $p_1 = \boldsymbol{c} - \frac{1}{2} \cdot \boldsymbol{R}_{max} \cdot \boldsymbol{r}_{max}$ $p_2 = \boldsymbol{c} + \frac{1}{2} \cdot \boldsymbol{R}_{max} \cdot \boldsymbol{r}_{max}$ Das entspricht den Mittelpunkten der beiden Boxen, die entstehen, wenn B in der Mitte der längsten Achse unterteilt wird.
3	$max = \mathrm{argmax}_{i=1,2,3}\, \boldsymbol{r}_i$ $p_1 = \boldsymbol{c} - \frac{2}{3} \cdot \boldsymbol{R}_{max} \cdot \boldsymbol{r}_{max}$ $p_2 = \boldsymbol{c}$ $p_3 = \boldsymbol{c} + \frac{2}{3} \cdot \boldsymbol{R}_{max} \cdot \boldsymbol{r}_{max}$ Das entspricht den Mittelpunkten der drei Boxen, die entstehen, wenn B entlang der längsten Achse zweimal unterteilt wird.
4	$max_1 = \mathrm{argmax}_{i=1,2,3}\, \boldsymbol{r}_i \qquad max_2 = \mathrm{argmax}_{i\in\{1,2,3\}\setminus\{max_1\}}\, \boldsymbol{r}_i$ $p_1 = \boldsymbol{c} - \frac{1}{2} \cdot (\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1} + \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2})$ $p_2 = \boldsymbol{c} - \frac{1}{2} \cdot (\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1} - \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2})$ $p_3 = \boldsymbol{c} + \frac{1}{2} \cdot (\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1} - \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2})$ $p_4 = \boldsymbol{c} + \frac{1}{2} \cdot (\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1} + \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2})$ Das entspricht den Mittelpunkten der vier Boxen, die entstehen, wenn B in der Mitte der beiden längsten Achsen unterteilt wird.
5	$max_1 = \mathrm{argmax}_{i=1,2,3}\, \boldsymbol{r}_i \qquad max_2 = \mathrm{argmax}_{i\in\{1,2,3\}\setminus\{max_1\}}\, \boldsymbol{r}_i$ $p_1 = \boldsymbol{c} - \frac{1}{2} \cdot (\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1} + \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2})$ $p_2 = \boldsymbol{c} - \frac{1}{2} \cdot (\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1} - \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2})$ $p_3 = \boldsymbol{c}$ $p_4 = \boldsymbol{c} + \frac{1}{2} \cdot (\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1} - \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2})$ $p_5 = \boldsymbol{c} + \frac{1}{2} \cdot (\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1} + \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2})$ Das entspricht den Mittelpunkten der vier Boxen, die entstehen, wenn B in der Mitte der beiden längsten Achsen unterteilt und zusätzlich der Mittelpunkt von B hinzugenommen wird.
6	$max_1 = \mathrm{argmax}_{i=1,2,3}\, \boldsymbol{r}_i \qquad max_2 = \mathrm{argmax}_{i\in\{1,2,3\}\setminus\{max_1\}}\, \boldsymbol{r}_i$ $p_1 = \boldsymbol{c} - \frac{1}{2} \cdot \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2} - \frac{2}{3}\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1}$ $p_2 = \boldsymbol{c} - \frac{1}{2} \cdot \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2}$ $p_3 = \boldsymbol{c} - \frac{1}{2} \cdot \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2} + \frac{2}{3}\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1}$ $p_4 = \boldsymbol{c} + \frac{1}{2} \cdot \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2} - \frac{2}{3}\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1}$ $p_5 = \boldsymbol{c} + \frac{1}{2} \cdot \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2}$ $p_6 = \boldsymbol{c} + \frac{1}{2} \cdot \boldsymbol{R}_{max_2} \cdot \boldsymbol{r}_{max_2} + \frac{2}{3}\boldsymbol{R}_{max_1} \cdot \boldsymbol{r}_{max_1}$ Das entspricht den Mittelpunkten der sechs Boxen, die entstehen, wenn B entlang der längsten Achse zweimal und in der Mitte der zweitlängsten Achse einmal unterteilt wird.

Tabelle 3.1: Berechnung der Mittelpunkte der initialen Partitionierung

Einfügereihenfolge der verbleibenden Flächen

Das Ergebnis der Heuristik hängt neben der initialen Partitionierung wesentlich von der Einfügereihenfolge der verbleibenden Flächen ab. Wird eine Fläche, die das Ergebnis der Heuristik nachhaltig beeinflußt, zum falschen Zeitpunkt und dabei in die falsche Partition eingefügt, wird das Ergebnis der Heuristik nachhaltig negativ beeinflußt. Um dies möglichst zu vermeiden, sind folgende Verfahren geeignet:

- *Zufällige Einfügereihenfolge*
  Am Anfang der Heuristik wir die Menge der Flächen zufällig permutiert und dadurch eine zufällige Einfügereihenfolge erzeugt.
- *Beste Partitionsgüte nach dem Einfügen*
  In jedem Schritt wird für jede der verbleibenden Flächen der Einfluß auf die Güte der Partitionierung berechnet und diejenige Fläche eingefügt, die das beste Ergebnis für die neue Partitionierung erzeugt.

Der resultierende Algorithmus *HeuristikMitZwischenoptimierung* ist in Abbildung 3.8 dargestellt.

3.7.5 Modifiziertes Gottschalk-Verfahren

Das *Top-down*-Verfahren von Gottschalk [GLM96] verwendet beliebig orientierte Boxen als Hüllkörper und berechnet binäre Hierarchien. Ein Aufteilungsschritt besteht darin, den Knoten $n = (F_n, B_n)$ zu unterteilen, wobei $B_n = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r})$ eine beliebig orientierte Box ist, die F_n umhüllt. Der Aufteilungsschritt wird durchgeführt, indem die längste Achse $\boldsymbol{R}_{max}$ in der Mitte halbiert wird. Ist dadurch keine Aufteilung möglich, weil alle Flächen in einer Hälfte liegen, so wird dasselbe für die zweitlängste und kürzeste Achse versucht. Schlagen alle Möglichkeiten fehl, so wird die Flächenmenge als unteilbar angesehen.

Der Aufteilungsschritt basiert auf der raumpartitionierungsgestützten Unterteilung von B in zwei Boxen B_1 und B_2 bzgl. derer die Flächen von F_n klassifiziert und dadurch in zwei Mengen F_1, F_2 unterteilt werden. Diese Methode kann nun mit der Vorgehensweise der raumpartitionierungsgestützten Bestimmung einer initialen Partitionierung bei der *Heuristik mit Zwischenoptimierung* kombiniert und dadurch Hierarchien beliebigen Grades berechnet werden. Dazu benutzen wir die durch die Bestimmung der Punkte $\boldsymbol{p}_i$ implizit gegebene Partitionierung der Box B in Regionen zur Klassifikation der Flächen bzgl. dieser Regionen. Die Regionen sind dabei äquivalent zu den durch die Punkte $\boldsymbol{p}_i$ bestimmten Voronoiregionen. Eine Fläche f liegt in der Region R_i, wenn ihr Mittelpunkt in R_i liegt.

Die sich aus der Tabelle 3.1 auf Seite 53 ergebenden Regionen einer Box für die verschiedenen Aufteilungsgrade sind in Abbildung 3.9 für Dimension 2 graphisch dargestellt.

Daraus ergibt sich der Algorithmus *GottschalkHeuristikModifiziert* in Abbildung 3.10.

Algorithmus *HeuristikMitZwischenoptimierung*(n, d)

Eingabe: $n = (F_n, h_n)$ aufzuteilender Knoten
d Grad der Aufteilung

Ausgabe: $((F_1, h_1), \ldots, (F_d, h_d))$ Aufteilung von n

$((F_1, h_1), \ldots, (F_d, h_d)) \leftarrow InitialePartitionierung(F_n)$
$F_{Rest} \leftarrow F_n \setminus \left(\cup_{i=1}^{d} F_i\right)$
while $(F_{Rest} \neq \emptyset)$
 do $f \leftarrow NaechsteEinfuegeFlaeche(F_{Rest})$
  $g_{min} \leftarrow \infty$
  $p_{min} \leftarrow 0$
  for $i \leftarrow 1$ **to** d
  do $h_i^{neu} \leftarrow Huellkoerper(F_i \cup \{f\}, h_i)$
   $g_i \leftarrow G((F_1, h_1), \ldots, (F_i \cup \{f\}, h_i^{neu}), \ldots, (F_d, h_d))$
   if $(g_i < g_{min})$
   then $g_{min} \leftarrow g_i$
    $p_{min} \leftarrow i$
  $F_{p_{min}} \leftarrow F_{p_{min}} \cup \{f\}$
  $h_{p_{min}} \leftarrow h_{p_{min}}^{neu}$
  $F_{Rest} \leftarrow F_{Rest} \setminus \{f\}$
return $((F_1, h_1), \ldots, (F_d, h_d))$

Abbildung 3.8: ALGORITHMUS: KNOTENAUFTEILUNG DURCH HEURISTIK MIT ZWISCHENOPTIMIERUNG

3.7.6 Modifiziertes ZACHMANN Verfahren

Das *Top-down-Verfahren* von ZACHMANN [FZ95, Za97] verwendet Iso-Boxen als Hüllkörper und berechnet binäre Hierarchien. Ein Aufteilungsschritt besteht darin, den Knoten $n = (F_n, B_n^{Iso})$ zu unterteilen, wobei $B_n^{Iso}(\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$ die Iso-Box ist, die F_n umhüllt. der Aufteilungsschritt wird durchgeführt, indem die Iso-Box entlang der Achse unterteilt wird, die eine möglichst balancierte Unterteilung der Flächenmenge F_n zuläßt.

Der Aufteilungsschritt basiert ähnlich wie bei dem Verfahren von GOTTSCHALK auf der raumpartitionierungsgestützten Unterteilung von B^{Iso} in zwei Iso-Boxen B_1^{Iso} und B_2^{Iso} bzgl. derer die Flächen von F_n klassifiziert und dadurch in zwei Mengen F_1, F_2 unterteilt werden. Die Grundidee dieses Verfahrens, daß die Iso-Box B_n^{Iso} entlang einer Achse unterteilt wird, kann ähnlich wie das Verfahren von GOTTSCHALK mit der Vorgehensweise der raumpartitionierungsgestützten Bestimmung der initialen Partitionierung bei der *Heuristik mit Zwischenoptimierung* kombiniert werden. Dazu faßen wir die Iso-Box B_n^{Iso} als Box $B_n(\boldsymbol{I}_3, \frac{1}{2} \cdot (\boldsymbol{b}^{min} + \boldsymbol{b}^{max}), \frac{1}{2} \cdot (\boldsymbol{b}^{max} - \boldsymbol{b}^{min}))$ auf und benutzen die durch die Bestimmung der Punkte $\boldsymbol{p}_i$ implizit gegebene Partitionierung der Box B_n in Regionen zur Klassifikation der Flächen bzgl. dieser Regionen. Die Regionen sind dabei äquivalent zu den durch die

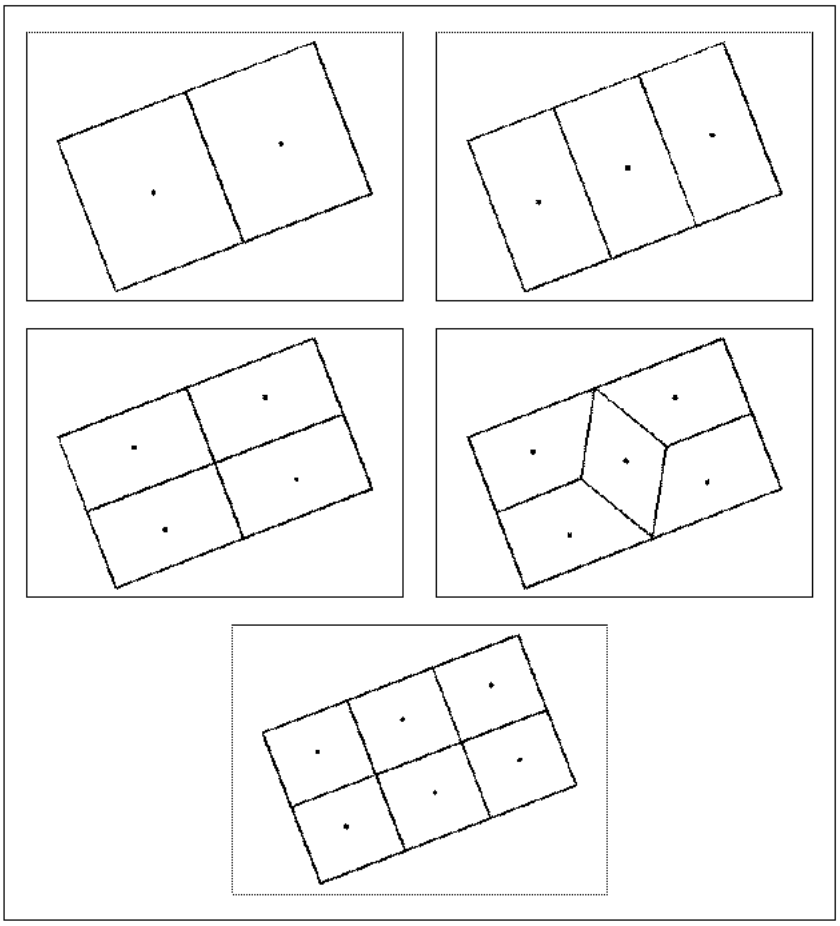

Abbildung 3.9: Partitionierung einer Box

Punkte $\boldsymbol{p}_i$ bestimmten Voronoiregionen. Eine Fläche f liegt in der Region R_i, wenn ihr Mittelpunkt in R_i liegt.

Die sich aus der Tabelle 3.1 auf Seite 53 ergebenden Regionen einer Box für die verschiedenen Aufteilungsgrade sind in Abbildung 3.9 graphisch dargestellt.

Daraus ergibt sich der Algorithmus *ZachmannHeuristikModifiziert* in Abbildung 3.11.

3.8 Berechnung von Hüllkörpern

Während der Berechnung der Aufteilung eines Knotens müssen an verschiedenen Stellen optimierte Hüllkörper für Flächenmengen F berechnet werden. Dabei tritt diese Problemstellung in zwei Varianten auf:

1. Berechne einen optimierten Hüllkörper h für eine Flächenmenge F.

Algorithmus $GottschalkHeuristikModifiziert(n, d)$

Eingabe: $n = (F_n, h_n)$ aufzuteilender Knoten
d Grad der Aufteilung

Ausgabe: $((F_1, h_1), \ldots, (F_d, h_d))$ Aufteilung von n

```
B(R, c, r) ←Box(F_n)
(p_0, ..., p_d) ←BerechneMittelpunkte(B, d)
for f in F_n
do d_min ←∞
    p_min ←0
    for j ←1 to d
    do if (|c_f − p_j| < d_min)
        then d_min ←|c_f − p_j|
             p_min ←j
    F_{p_min} ←F_{p_min} ∪ {f}
for j ←1 to d
do h_j ←Huellkoerper(F_j)
return ((F_1, h_1), ..., (F_d, h_d))
```

Abbildung 3.10: ALGORITHMUS: KNOTENAUFTEILUNG DURCH MODIFIZIERTE GOTTSCHALK-HEURISTIK

2. Berechne einen optimierten Hüllkörper h für eine Flächenmenge $F \cup \{f\}$, wobei ein optimierter Hüllkörper h_F für F bereits vorliegt.

Als Hüllkörper verwenden wir Kugeln, achsenorientierte Boxen (Iso-Boxen) und beliebig orientierte Boxen. Für diese Hüllkörpertypen werden wir in der Folge Verfahren zu deren optimierten Berechnung präsentieren unter Verwendung der in Abschnitt 3.4.3 beschriebenen *statisch geometrischen Gütemaßen*

- Volumen,
- Durchmesser,
- Oberfläche und
- gerichteter Hausdorff-Abstand.

3.8.1 Volumen, Durchmesser, Oberfläche

Zunächst werden wir uns mit den einfachen Gütemaßen *Volumen, Durchmesser* und *Oberfläche* befassen. Diese sind nur von dem Hüllkörper selbst und nicht von der umhüllten Flächenmenge abhängig.

Algorithmus *ZachmannHeuristikModifiziert*(n, d)

Eingabe: $n = (F_n, h_n)$ aufzuteilender Knoten
d Grad der Aufteilung

Ausgabe: $((F_1, h_1), \ldots, (F_d, h_d))$ Aufteilung von n

$B(\boldsymbol{I}, \boldsymbol{c}, \boldsymbol{r}) \leftarrow Box(B^{Iso}(F_n))$
$(\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d) \leftarrow BerechneMittelpunkte(B, d)$
for f **in** F_n
do $d_{min} \leftarrow \infty$
  $p_{min} \leftarrow 0$
  for $j \leftarrow 1$ **to** d
  do if $(|\boldsymbol{c}_f - \boldsymbol{p}_j| < d_{min})$
    then $d_{min} \leftarrow |\boldsymbol{c}_f - \boldsymbol{p}_j|$
      $p_{min} \leftarrow j$
  $F_{p_{min}} \leftarrow F_{p_{min}} \cup \{f\}$
for $j \leftarrow 1$ **to** d
do $h_j \leftarrow Huellkoerper(F_j)$
return $((F_1, h_1), \ldots, (F_d, h_d))$

Abbildung 3.11: ALGORITHMUS: KNOTENAUFTEILUNG DURCH MODIFIZIERTE ZACHMANN-HEURISTIK

Kugel

Für eine Kugel sind die einfachen Gütemaße *Volumen, Durchmesser* und *Oberfläche* äquivalente Optimierungskriterien, denn es gilt für eine Kugel $K = (\boldsymbol{c}, r)$ mit Mittelpunkt $\boldsymbol{c}$ und Radius r:

$$
\begin{aligned}
Volumen(K) &= \frac{4}{3} \cdot \pi \cdot r^3 \\
Durchmesser(K) &= 2 \cdot r \\
\mathit{Oberfläche}(K) &= 4 \cdot \pi \cdot r^2
\end{aligned}
$$

Sollen diese Gütemaße für eine Kugel K minimiert werden, die eine Flächenmenge F umhüllt, so entspricht dies in allen Fällen der Minimierung des Radius der Kugel. Gesucht ist somit ein Verfahren, das für eine Flächenmenge F eine Kugel $K = (\boldsymbol{c}, r)$ berechnet, die F umhüllt und dabei minimalen Radius r hat, wobei wir uns bei der Berechnung von K auf die Betrachtung der Eckpunkte $V(F)$ der Flächenmenge F beschränken können.

Eine Möglichkeit der Lösung des Problems besteht in der Lösung eines nichtlinearen Optimierungsproblemes

$$\min_{\boldsymbol{c} \in \mathbb{R}^3} r(\boldsymbol{c})$$

mit

$$r(\boldsymbol{c}) = \max_{\boldsymbol{v} \in V(F)} |\boldsymbol{v} - \boldsymbol{c}|.$$

Dabei ist die Zielfunktion $r(\boldsymbol{c}) : \mathbb{R}^3 \longrightarrow \mathbb{R}$ konvex, so daß Verfahren der *nichtlinearen konvexen* Optimierung verwendet werden können. Dazu bietet sich z.B. das *Downhill-Simplex* Verfahren von Nelder und Mead (vgl. [NM65, PT+92]) an, weil es ohne die Berechnung von Gradienten auskommt, denn die Funktion $r(\boldsymbol{c})$ ist nicht differenzierbar. Eine Implementierung ist z.B. innerhalb des Softwarepaketes zu *Numerical Recipes in C* [PT+92] verfügbar.

Alternativ dazu kann auch der inkrementelle Algorithmus von Welzl [We91] benutzt werden. Eine Implementierung davon ist von White [Wh] verfügbar.

Möchten wir ausgehend von einer optimierten Kugel $K_1(\boldsymbol{c}_1, r_1)$, die eine Flächenmenge F umhüllt, eine optimierte Kugel $K_2(\boldsymbol{c}_2, r_2)$ berechnen, die die Flächenmenge $F \cup \{f\}$ umhüllt, so muß bei der Verwendung des nichtlinearen Optimierungsverfahren dieses erneut angewandt werden, wobei der Mittelpunkt $\boldsymbol{c}_1$ einen geeigneten Startpunkt für die neue Optimierung darstellt. Bei der Verwendung des Welzl-Algorithmus muß auch dieser erneut gestartet werden. Werden jedoch bei der Berechnung von K_1 neben dem Mittelpunkt $\boldsymbol{c}_1$ und dem Radius r_1 die Eckpunkte V_1 der Flächenmenge F abgespeichert, die K_1 bestimmen, so kann durch die Verwendung der *move-to-front heuristic* die Berechnung von K_2 beschleunigt werden. Denn setzen wir diese Eckpunkte V_1 an den Anfang der zu umhüllenden Eckenmenge, so ergibt die Berechnung der ersten Kugel K_{start} eine Kugel, die alle Eckpunkte $V(F)$ schon optimiert umhüllt und bzgl. der nur evtl. einige Eckpunkte von f außerhalb liegen. Dadurch wird die Performance des Algorithmus entscheidend gesteigert.

Iso-Box

Die Berechnung von Iso-Boxen, die die einfachen Gütemaße *Volumen, Duchmesser* und *Oberfläche* optimieren, ist noch einfacher als die von Kugeln. Eine Iso-Box B^{Iso} ist bestimmt durch drei Intervalle $[\boldsymbol{b}_i^{min}, \boldsymbol{b}_i^{max}]$, wobei $\boldsymbol{b}_i^{min}, \boldsymbol{b}_i^{max} \in \mathbb{R}$ $(i = 1, 2, 3)$. Die einfachen Gütemaße berechnen sich wie folgt:

$$\begin{aligned}
Volumen(B^{Iso}) &= \prod_{i=1}^{3} (\boldsymbol{b}_i^{max} - \boldsymbol{b}_i^{min}) \\
Durchmesser(B^{Iso}) &= \sqrt{\sum_{i=1}^{3} (\boldsymbol{b}_i^{max} - \boldsymbol{b}_i^{min})^2} \\
Oberfläche(B^{Iso}) &= 2 \cdot \sum_{i=1}^{3} (\boldsymbol{b}_i^{max} - \boldsymbol{b}_i^{min}) \cdot (\boldsymbol{b}_{((i \bmod 3)+1)}^{max} - \boldsymbol{b}_{((i \bmod 3)+1)}^{min})
\end{aligned}$$

Wegen der festen Orientierung von B^{Iso} können die Intervalle unabhängig voneinander optimiert werden und die einfachen Gütemaße nehmen in der minimalen Ausdehnung der Intervalle ihr Minimum an. Die Bestimmung der optimalen Intervalle ergibt sich dabei durch die Berechnung der minimalen $\boldsymbol{b}^{min}$ und maximalen Koordinaten $\boldsymbol{b}^{max}$ der Eckpunkte $V(F)$ der Flächenmenge F.

Möchten wir ausgehend von einer optimierten Iso-Box $B_1^{Iso}(\boldsymbol{a}^{min}, \boldsymbol{a}^{max})$, die eine Flächenmenge F umhüllt, eine optimierte Iso-Box $B_2^{Iso}(\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$ berechnen, die die Flächenmenge $F \cup \{f\}$ umhüllt, müssen nur die Werte $\boldsymbol{a}_i^{min}$ und $\boldsymbol{a}_i^{max}$ für die Eckpunkte $V(f)$ von f zu den Werten $\boldsymbol{b}_i^{min}$ und $\boldsymbol{b}_i^{max}$ für $i = 1, 2, 3$ erweitert werden.

Box

Eine Box B ist bestimmt durch deren Lage im Raum bestehend aus Orientierung $\boldsymbol{R}$ und Mittelpunkt $\boldsymbol{c}$ sowie deren Richtungsradiusvektor $\boldsymbol{r}$. Die einfachen Gütemaße berechnen sich analog zur Iso-Box wie folgt:

$$\begin{aligned}
Volumen(B) &= 8 \cdot \prod_{i=1}^{3} \boldsymbol{r}_i \\
Durchmesser(B) &= 2 \cdot \sqrt{\sum_{i=1}^{3} \boldsymbol{r}_i^2} \\
Oberfläche(B) &= 8 \cdot \sum_{i=1}^{3} \boldsymbol{r}_i \cdot \boldsymbol{r}_{((i \mod 3)+1)}
\end{aligned}$$

Im Gegensatz zur Iso-Box können jedoch die einzelnen Komponenten der Box nicht unabhängig voneinander optimiert werden, da die Ausdehnung der Box von deren Orientierung abhängt. Die Berechnung einer optimierten Box kann jedoch analog zur Kugel als nichtlineares Optimierungsproblem parametrisiert in der Orientierung der Box beschrieben werden:

$$\min_{(\alpha,\beta,\gamma)\in\mathbb{R}^3} G(\boldsymbol{s})$$

mit

$$\boldsymbol{r}_i = \frac{1}{2} \cdot |\max_{\boldsymbol{v}\in V(F)} R_{.i} \cdot \boldsymbol{v} - \min_{\boldsymbol{v}\in V(F)} R_{.i} \cdot \boldsymbol{v}| \text{ und } R = R(\alpha, \beta, \gamma) \quad (i = 1, 2, 3)$$

(α, β, γ) sind dabei z.B. *XYZ-Fixwinkel* (zur Umrechung von *XYZ-Fixwinkel* in Rotationmatrizen siehe Anhang A). Im Gegensatz zur Berechnung optimierter Kugeln ergeben sich bei Boxen nichtlineare, nichtkonvexe Optimierungsprobleme. Für diese können spezielle Verfahren der *nichtlinearen nichtkonvexen* Optimierung, wie z.B. *simulated annealing*

(Metropolis et.al. [MR+53], van Laarhoven und Aarts [LA87]), *genetische Algorithmen* (Goldberg [Go89]), *threshold Accepting* (Dueck und Scheuer [DS90]) oder *tabu search* (Glover [Gl89, Gl90]), angewendet werden. Diese haben jedoch alle den Nachteil, daß sie zeitaufwendig sind und eine optimale Lösung nicht immer garantieren können.

Als praktikable Methode zur Approximation der optimalen Box bieten sich Verfahren der *nichtlinearen konvexen* Optimierung, wie z.B. das *Downhill-Simplex* Verfahren von Nelder und Mead [NM65], an. Dieses Verfahren kann von verschiedenen Startpunkten ausgehend eine Reihe von Malen angewendet und die beste Lösung als Approximation für das globale Optimum verwendet werden.

Möchten wir ausgehend von einer optimierten Box $B_1(\boldsymbol{R}_1, \boldsymbol{c}_1, \boldsymbol{r}_1)$, die eine Flächenmenge F umhüllt, eine optimierte Box $B_2(\boldsymbol{R}_2, \boldsymbol{c}_2, \boldsymbol{r}_2)$ berechnen, die die Flächenmenge $F \cup \{f\}$ umhüllt, so muß das eingesetzte Optimierungsverfahren erneut gestartet werden, wobei die Orientierung $\boldsymbol{R}_1$ von B_1 ein geeigneter Startwert für die neue Optimierung darstellt.

Alternativ zur Benutzung numerischer Optimierungsverfahren stehen die Verfahren von O'Rourke, Gottschalk et.al. sowie Barequet et.al. zur Verfügung. Für den Fall des Volumens als Optimierungskriterium beschreibt O'Rourke [Ro85] ein Verfahren, das für n Punkte in Zeit $O(n^3)$ die volumenminimale Box berechnet. Gottschalk, Lin und Manocha richten ihre Boxen bzgl. der Hauptachsen der Flächenmenge F aus (Errata zu [GLM96]). Barequet et.al. benutzen in [BC+96] einzelne Hauptachsen zur Bestimmung möglichst guter Boxen (vgl. dazu Abschnitt 3.2).

3.8.2 Gerichteter Hausdorff-Abstand

Wie schon in Abschnitt 3.4.3 ausgeführt, beschreibt der *gerichtete Hausdorff-Abstand* vom Hüllkörper h zur umhüllten Flächenmenge F den geometrischen Fehler, der durch die Approximation von F durch h entsteht.

Zur Erinnerung an dieser Stelle nocheinmal die Definition des *gerichteten Hausdorff-Abstands*:

> Seien A und B zwei Mengen, auf denen ein Abstandsmaß d definiert ist, dann heißt
>
> $$d_{HA}(A, B) := \max_{a \in A} \min_{b \in B} d(a, b)$$
>
> *gerichteter Hausdorff-Abstand* von A zu B.

Anschaulich bedeutet der *gerichtete Hausdorff-Abstand* zwischen A und B, daß es kein Element $a \in A$ gibt, dessen minimaler Abstand zu B größer ist als $d_{HA}(A, B)$. Auf den

Fall des Hüllkörpers und der Flächenmenge übertragen bedeutet dies, daß es keinen Punkt des Hüllkörpers gibt, dessen minimaler Abstand zur Flächenmenge größer ist als $d_{HA}(h, F)$.

Mit Hilfe des *gerichteten Hausdorff-Abstands* kann nicht nur der geometrische Fehler des Hüllkörpers beschrieben werden, sondern auch im Falle von einem kollidierenden Hüllkörperpaar (h_1, h_2), das die Flächenmengen F_1 und F_2 umhüllt, der maximal noch mögliche Abstand zwischen F_1 und F_2 nach oben abgeschätzt werden.

$$
\begin{aligned}
&\text{Sei } \boldsymbol{x} \in h_1 \cap h_2 : \\
&\qquad d(F_1, F_2) \leq \min_{\boldsymbol{p}_1 \in F_1} d(\boldsymbol{p}_1, \boldsymbol{x}) + \min_{\boldsymbol{p}_2 \in F_2} d(\boldsymbol{p}_2, \boldsymbol{x}) \\
&\qquad\qquad \leq \max_{\boldsymbol{q}_1 \in h_1} \min_{\boldsymbol{p}_1 \in F_1} d(\boldsymbol{q}_1, \boldsymbol{p}_1) + \max_{\boldsymbol{q}_2 \in h_2} \min_{\boldsymbol{p}_2 \in F_2} d(\boldsymbol{q}_2, \boldsymbol{p}_2) \\
&\qquad\qquad = d_{HA}(h_1, F_1) + d_{HA}(h_2, F_2)
\end{aligned}
$$

Die einfachen Gütemaße *Volumen*, *Durchmesser* und *Oberfläche* haben gegenüber dem *gerichteten Hausdorff-Abstand* den Vorteil, daß sie für die gängigen Hüllkörpertypen (Kugel, Iso-Boxen, Boxen) einfach zu berechnen sind, wohingegen der *gerichtete Hausdorff-Abstand* i.d.R. nur approximativ berechnet werden kann. Nur in Ausnahmenfällen kann er exakt bestimmt werden. HUBBARD zeigt dazu in [Hu95] folgenden Zusammenhang:

> Sei $S = (\boldsymbol{c}, r)$ eine Kugel mit Mittelpunkt $\boldsymbol{c}$ und Radius r, die die Oberfläche eines konvexen Polyeders schneidet. Ist $\boldsymbol{c}'$ ein Punkt auf der Oberfläche des Polyeders, der $\boldsymbol{c}$ am nächsten liegt, dann gilt für den Abstand zwischen Kugel und Polyeder:
>
> $$d = \begin{cases} r - d(\boldsymbol{c}, \boldsymbol{c}') & \text{falls } \boldsymbol{c} \text{ innerhalb des Polyeders} \\ r + d(\boldsymbol{c}, \boldsymbol{c}') & \text{falls } \boldsymbol{c} \text{ außerhalb des Polyeders} \end{cases}$$

Mit Hilfe dieser Beziehung kann für ein konvexes Polyeder der *gerichtete Hausdorff-Abstand* zu einer umhüllenden Kugel exakt berechnet werden. FUHRMANN beschreibt in [Fu97] darüber hinaus ein Verfahren, mit dessen Hilfe für ein konvexes Polyeder die umhüllende Kugel berechnet werden kann, die den *gerichteten Hausdorff-Abstand* der Kugel zu dem umhüllten Polyeder minimiert. Im allgemeinen – und i.d.R. bei unseren Problemstellungen vorliegenden – Fall, daß das zu umhüllende Objekt eine Flächenmenge bzw. der Hüllkörper keine Kugel ist, versagen diese Vorgehensweisen jedoch.

Eine Möglichkeit der Berechnung von optimierten Hüllkörpern bzgl. des *gerichteten Hausdorff-Abstands* als Optimierungskriterium besteht nun darin, die Berechung der Hüllkörper als nichtlineares Optimierungsproblem zu beschreiben und es mit Hilfe numerischer Methoden zu lösen. Die enstehenden Probleme sind jedoch nichtkonvex, so daß generell bzgl. der Lösung dieselben Probleme enstehen wie bei der optimierten Bestimmung von beliebig orientierten Boxen bei einfachen Optimierungskriterien. Darüber hinaus ist die Zielfunktionswertbestimmung durch die Approximation des tatsächlichen *gerichteten Hausdorff-Abstands* zeitaufwendig, so daß sich der *gerichtete Hausdorff-Abstand* auch bei der Benutzung numerischer Methoden nicht als Optimierungskriterium eignet. Daher ist es sinnvoll, während der Optimierung ein einfaches Optimierungskriterium (*Volumen*, *Durchmesser* oder *Oberfläche*) als Ausweichoptimierungskriterium zu benutzen und erst für die bzgl. dieses Kriteriums optimierten Hüllkörper den *gerichteten Hausdorff-Abstand* zu bestimmen.

Wir benötigen nun ein Verfahren, das den *gerichteten Hausdorff-Abstand* von einem Hüllkörper zu der umhüllten Flächenmenge approximiert. Einen ersten Ansatz dazu hat HUBBARD in [Hu95] gemacht. Zur Optimierung seiner Kugeln hat er *simulated annealing* zusammen mit dem *gerichteten Hausdorff-Abstand* als Optimierungsgröße benutzt. Zur Approximation des *gerichteten Hausdorff-Abstands* einer Kugel zu der umhüllten Flächenmenge berechnet er für diskrete Punkte auf der Kugeloberfläche den tatsächlichen Abstand zu der Flächenmenge. Zu dem Maximum der so berechneten Werte addiert er als Korrekturterm den maximalen Abstand zwischen zwei Beispielpunkten auf der Kugeloberfläche. Als Beispielpunkte wählt er die Projektion der zwanzig Eckpunkte des Einheitsdodekaeders auf die Kugeloberfläche. Dadurch erhält er eine obere Schranke für den tatsächlichen *gerichteten Hausdorff-Abstand* von der Kugel zu der umhüllten Flächenmenge. Ein Problem des Verfahrens ist, daß es nicht skalierbar ist in der Genauigkeit der Approximation, da der maximale Abstand zwischen zwei projizierten Eckpunkten des Dodekaeders alleine abhängig ist vom Radius der betrachteten Kugel.

Wir wollen in der Folge Verfahren vorstellen, die in der Lage sind, für alle von uns betracheten Hüllkörpertypen (Kugel, Iso-Box, Box) beliebig genaue Approximationen des *gerichteten Hausdorff-Abstands* zu der umhüllten Flächenmenge zu berechnen. Die grundlegende Idee des Verfahrens ist, die Oberfläche des Hüllkörpers in Regionen zu partitionieren. Für jede der Regionen wird eine Approximation des tatsächlichen *gerichteten Hausdorff-Abstands* der Region zu der Flächenmenge berechnet. Diese Approximation besteht zum einem aus dem tatsächlichen Abstand eines Punktes der Region zu der Flächenmenge und einem Korrekturterm, der den möglichen Fehler der Approximation beschreibt. Die Region der Hüllkörperoberfläche mit der maximalen Approximation bestimmt die derzeitige Approximation des gesamten *gerichteten Hausdorff-Abstands*. Solange der Korrekturterm dieser Approximation über einer vorgegebenen Grenze liegt, wird die zugehörige Region in kleinere Regionen unterteilt und für diese eine Approximation des *gerichteten Hausdorff-Abstands* berechnet. Das Verfahren endet, wenn der tatsächliche *gerichtete Hausdorff-Abstand* mit der vorgegebenen Genauigkeit bestimmt ist.

Als Abbruchkriterium sind geeignet:

1. Der Korrekturterm t_k ist kleiner als eine vorgegebene absolute Schranke ε_a.

2. Der Korrekturterm t_k ist relativ zu der berechneten Approximation d_{HA} kleiner als eine vorgegebene relative Schranke ε_r.

Algorithmus *gerichteterHausdorffAbstand*(h, F)
Eingabe: h Hüllkörper
F Flächenmenge
Ausgabe: $d_{HA}(h, F)$ Approximation des *gerichteten Hausdorff-Abstands*
$\mathcal{R} \leftarrow initialePartitionierungOberflaeche(h)$
for R **in** $\mathcal{R}$
do $(d^a_{HA}(R), t_k(R)) \leftarrow ApproxGerichteterHausdorffAbstand(R, F)$
$(d^a_{HA}(R^{max}), t_k(R^{max})) \leftarrow \mathrm{argmax}_{R \in \mathcal{R}}(d^a_{HA}(R), t_k(R))$
while $((t_k^{max}(R) < \varepsilon_a)$ **or** $(\frac{t_k^{max}(R)}{d^a_{HA}(R)} < \varepsilon_r))$
  do $\mathcal{R}_{neu} \leftarrow UnterteileRegion(R^{max})$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_{neu} \setminus \{R^{max}\}$
    for R **in** $\mathcal{R}_{neu}$
    do $(d^a_{HA}(R), t_k(R)) \leftarrow ApproxGerichteterHausdorffAbstand(R, F)$
    $(d^a_{HA}(R^{max}), t_k(R^{max})) \leftarrow \mathrm{argmax}_{R \in \mathcal{R}}(d^a_{HA}(R), t_k(R))$
return $d^a_{HA}(R^{max})$

Abbildung 3.12: Algorithmus: Berechnung gerichteter Hausdorff-Abstand Hüllköper – Flächenmenge

Dadurch ergibt sich der Algorithmus *gerichteterHausdorffAbstand* in Abbildung 3.12.
Zu erläutern sind nun noch die Verfahren zur

1. Bestimmung einer initialen Partitionierung der Hüllkörperoberfläche,
2. Unterteilung einer Region in Teilregionen und
3. Approximation des *gerichteten Hausdorff-Abstands* einer Region.

Bestimmung einer initialen Partitionierung der Hüllkörperoberfläche

Das Aussehen der Regionen unterscheidet sich zwischen der Kugel mit ihrer gekrümmten Oberfläche und den beiden Boxtypen mit ihren planaren Oberflächen. Für diese beiden unterschiedlichen Typen werden wir die initiale Partitionierung der Oberfläche getrennt beschreiben.

- **Kugel**
  Als initiale Partitionierung bietet sich die Projektion des Ikosaeders (vgl. [BS+95]) auf die Kugeloberfläche an. Dadurch wird die Oberfläche der Kugel in zwanzig *Eulersche* Dreiecke unterteilt.

- **Iso-Box, Box**
  Als initiale Partitionierung bietet sich die Betrachtung je einer Seite der Box an, so daß die initiale Partitionierung aus den sechs rechteckigen Flächen der Box besteht.

Unterteilung einer Region in Teilregionen

- **Kugel**
  Eine Region einer Kugel ist ein *Eulersches* Dreieck. Dieses wird durch die Unterteilung in zwei neue *Eulersche Dreiecke* aufgeteilt, die zusammen das alte ergeben. Dazu wird die längste Seite des Dreiecks mit einer Kante von der gegenüberliegenden Ecke zur Mitte der Seite halbiert und es entstehen dadurch zwei *Eulersche* Dreiecke.

  Sei $D(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2)$ ein *Eulersches* Dreieck auf einer Kugel $K(\boldsymbol{c}, r)$, $(\boldsymbol{v}_i, \boldsymbol{v}_{((i+1) \mod 3)})$ der längste Großkreisabschnitt und $\boldsymbol{v}_{((i+2) \mod 3)}$ der gegenüberliegende Eckpunkt. Der Punkt, der den längsten Großkreisabschnitt halbiert ergibt sich zu

  $$\boldsymbol{v}_{neu} = \boldsymbol{c} + r \cdot \frac{\boldsymbol{v}_{((i+1) \mod 3)} + \boldsymbol{v}_i}{|\boldsymbol{v}_{((i+1) \mod 3)} + \boldsymbol{v}_i|}.$$

  Die beiden enstehenden *Eulerschen* Dreiecke sind dann $D_1(\boldsymbol{v}_i, \boldsymbol{v}_{neu}, \boldsymbol{v}_{((i+2) \mod 3)})$ und $D_2(\boldsymbol{v}_{((i+1) \mod 3)}, \boldsymbol{v}_{((i+2) \mod 3)}, \boldsymbol{v}_{neu})$. Durch diese Art der Unterteilung wird die Länge der längsten Seite der Dreiecke kontrolliert und es entstehen keine Dreiecke mit kleinen Innenwinkeln.

- **Iso-Box, Box**
  Eine Region einer Box ist ein Rechteck. Dieses wird durch die Seitenhalbierenden in vier Rechtecke mit halber Seitenlänge unterteilt. Sei $R(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$ ein Rechteck. Der Mittelpunkt und Unterteilungspunkt des Rechtecks ergibt sich zu

  $$\boldsymbol{v}_c = \frac{1}{4} \cdot \sum_{i=0}^{3} \boldsymbol{v}_i$$

  und die Mittelpunkte der Seiten zu

  $$\boldsymbol{v}_{(i,(i+1) \mod 4)} = \frac{1}{2} \cdot (\boldsymbol{v}_i + \boldsymbol{v}_{((i+1) \mod 4)}).$$

  Die vier entstehenden Rechtecke ergeben sich damit zu

  $$R_1(\boldsymbol{v}_0, \boldsymbol{v}_{(0,1)}, \boldsymbol{v}_c, \boldsymbol{v}_{(3,0)}), \qquad R_2(\boldsymbol{v}_1, \boldsymbol{v}_{(1,2)}, \boldsymbol{v}_c, \boldsymbol{v}_{(0,1)}),$$
  $$R_3(\boldsymbol{v}_2, \boldsymbol{v}_{(2,3)}, \boldsymbol{v}_c, \boldsymbol{v}_{(1,2)}), \qquad R_4(\boldsymbol{v}_3, \boldsymbol{v}_{(3,0)}, \boldsymbol{v}_c, \boldsymbol{v}_{(2,3)}).$$

Approximation des *gerichteten Hausdorff-Abstands* einer Region

Gegeben ist nun entweder ein *Eulersches* Dreieck im Fall der Kugel oder ein Rechteck im Fall von Iso-Box oder Box sowie eine Flächenmenge F. Für das *Eulersche* Dreieck oder das Rechteck soll nun eine Approximation des *gerichteten Hausdorff-Abstands* zu der Flächenmenge F berechnet werden. Die Berechnung der Approximation besteht aus zwei Teilen:

1. Berechnung des minimalen Abstands eines Beispielpunktes zu der Flächenmenge F
2. Berechnung des Korrekturterms für einen Beispielpunkt

Die Berechnung des minimalen Abstandes eines Beispielpunktes zu der Flächenmenge F ist unabhängig von dem Aussehen der Region, wohingegen die Berechnung des Korrekturterms sowohl vom verwendeten Berechnungsverfahren als auch vom Aussehen der Region abhängt. Daher werden wir zunächst die Berechnung des minimalen Abstands eines Beispielpunktes $\boldsymbol{p}$ zu der Flächenmenge F betrachten und uns danach der Berechnung der Korrekturterme zuwenden.

Abschließend werden wir uns mit der Auswahl der Beispielpunkte für eine Region beschäftigen. Diese wird von der Art des zugehörigen Korrekturterms abhängen.

Berechnung des minimalen Abstandes eines Punktes zu einer Flächenmenge

Sei $\boldsymbol{x}$ ein Punkt und $F = \{f_1, \ldots, f_n\}$ eine Menge von n konvexen Flächen, dann ist der minimale Abstand von $\boldsymbol{x}$ zu F der minimale Abstand von $\boldsymbol{x}$ zu einer Fläche von F, d.h. $d(\boldsymbol{x}, F) = \min_{f \in F} d(\boldsymbol{x}, f)$. Zur Berechnung von $d(\boldsymbol{x}, F)$ muß also sukzessive der minimale Abstand von $\boldsymbol{x}$ zu jeder Fläche in F berechnet werden und das Minimum der Werte bestimmt werden.

Der minimale Abstand $d(\boldsymbol{x}, f)$ von $\boldsymbol{x}$ zu einer konvexen Fläche $f = \boldsymbol{v}_0, \ldots, \boldsymbol{v}_{k-1}, \boldsymbol{v}_k \equiv \boldsymbol{v}_0$ wird bestimmt durch den Abstand von $\boldsymbol{x}$ zu

1. *der zu f gehörigen Ebene e_f*
   Sei $\boldsymbol{n}_f^T \cdot \boldsymbol{y} - d_f = 0$ die Ebenengleichung von e_f, dann ergibt sich die Projektion $\boldsymbol{x}^{proj}$ von $\boldsymbol{x}$ in e_f durch

   $$\boldsymbol{x}^{proj} = \boldsymbol{x} + (d_f - \boldsymbol{n}_f^T \cdot \boldsymbol{x}) \cdot \boldsymbol{n}_f.$$

   Der projizierte Punkt $\boldsymbol{x}^{proj}$ liegt innerhalb von f, falls

   $$\begin{aligned} \forall\ i: \quad & \det[\boldsymbol{n}_f, \boldsymbol{x}^{proj} - \boldsymbol{v}_i, \boldsymbol{v}_{i+1} - \boldsymbol{v}_i] \leq 0 \\ \iff \quad & ((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f)^T \cdot (\boldsymbol{x}^{proj} - \boldsymbol{v}_i) \leq 0. \end{aligned}$$

   Ist dies der Fall, so wird der minimale Abstand zwischen $\boldsymbol{x}$ und f durch den Abstand von $\boldsymbol{x}$ zu e_f bestimmt und es gilt:

   $$d(\boldsymbol{x}, f) = d(\boldsymbol{x}, e_f) = |\boldsymbol{n}_f^T \cdot \boldsymbol{x} - d_f|$$

2. *einer Kante* $e_i = (\boldsymbol{v}_i, \boldsymbol{v}_{i+1})$ $i = 0, \ldots, k-1$
   Sei $\boldsymbol{v}_i + \lambda \cdot (\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)$ die e_i unterstützende Gerade l_i. Die Projektion von $\boldsymbol{x}$ auf die Gerade l_i ergibt sich aus

$$\boldsymbol{x}^{proj} = \boldsymbol{v}_i + \frac{(\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)^T \cdot (\boldsymbol{x} - \boldsymbol{v}_i)}{|\boldsymbol{v}_{i+1} - \boldsymbol{v}_i|^2} \cdot (\boldsymbol{v}_{i+1} - \boldsymbol{v}_i).$$

   Der projizierte Punkt $\boldsymbol{x}^{proj}$ liegt innerhalb von e_i, falls

$$0 \leq \lambda^{proj} = \frac{(\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)^T \cdot (\boldsymbol{x} - \boldsymbol{v}_i)}{|\boldsymbol{v}_{i+1} - \boldsymbol{v}_i|^2} \leq 1.$$

   Für den Abstand von $\boldsymbol{x}$ zu l_i gilt:

$$d(\boldsymbol{x}, l_i) = \frac{|(\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times (\boldsymbol{x} - \boldsymbol{v}_i)|}{|\boldsymbol{v}_{i+1} - \boldsymbol{v}_i|}$$

   Gibt es eine Projektion von $\boldsymbol{x}$ auf eine Gerade, die innerhalb einer Kante liegt, so wird der minimale Abstand zwischen $\boldsymbol{x}$ und f durch diese Art des Abstands bestimmt und es gilt:

$$d(\boldsymbol{x}, f) = \min_{\{e_i | \boldsymbol{x}^{proj} \in e_i\}} d(\boldsymbol{x}, l_i)$$

3. *einem Eckpunkt* $\boldsymbol{v}_i$ $i = 0, \ldots, k-1$
   Für den Eckpunkt $\boldsymbol{v}_i$ wird der Abstand zu dem Punkt $\boldsymbol{x}$ bestimmt durch

$$d(\boldsymbol{x}, \boldsymbol{v}_i) = |\boldsymbol{x} - \boldsymbol{v}_i|$$

   und für den Abstand von $\boldsymbol{x}$ zu f gilt:

$$d(\boldsymbol{x}, f) = \min_{i=0,\ldots,k-1} d(\boldsymbol{x}, \boldsymbol{v}_i)$$

Berechnung des Korrekturterms für eine Region

Sei nun in der Folge $\boldsymbol{s}$ der Beispielpunkt, $R(\boldsymbol{v}_0, \ldots, \boldsymbol{v}_3)$ eine rechteckige Region im Fall der Iso-Box und der Box, $D(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2)$ ein *Eulersches* Dreieck als Region im Fall der Kugel $K(\boldsymbol{c}, r)$ und F die Flächenmenge. Bei der Berechnung des Korrekturterms für einen Beispielpunkt sind zwei Möglichkeiten sinnvoll:

1. *Maximaler Abstand eines Punktes in der Region zu dem Beispielpunkt*
   Im Fall der rechteckigen Region bei der Iso-Box und der Box ist offensichtlich, daß der maximale Abstand eines Punktes der Region zu einem beliebigen Punkt in der

Region in einem der Eckpunkte angenommen wird, d.h. der Korrekturterm $t(\boldsymbol{s}, R, F)$ ergibt sich zu

$$t(\boldsymbol{s}, R, F) = \max_{\boldsymbol{r} \in R} d(\boldsymbol{s}, \boldsymbol{r}) = \max_{i=0,\dots,3} d(\boldsymbol{s}, \boldsymbol{v}_i).$$

Für die *Eulerschen* Dreiecke gilt die analoge Aussage zu den rechteckigen Regionen nicht uneingeschränkt. Betrachten wir jedoch anstelle des euklidischen Abstandes zwischen dem Beispielpunkt $\boldsymbol{s}$ und einem anderen Punkt $\boldsymbol{x}$ in der Region die Länge des Großkreisabschnittes $d(\widehat{\boldsymbol{s}, \boldsymbol{x}})$, so gilt auch im Fall der *Eulerschen* Dreiecke, daß der maximale Abstand von $\boldsymbol{s}$ zu einem anderen Punkt in der Region in der Länge des Großkreisabschnittes zu einem der Eckpunkte angenommen wird, d.h. der Korrekturterm $t(\boldsymbol{s}, R, F)$ ergibt sich zu

$$t(\boldsymbol{s}, R, F) = \max_{\boldsymbol{r} \in R} d(\boldsymbol{s}, \boldsymbol{r}) = \max_{i=0,\dots,2} d(\widehat{\boldsymbol{s}, \boldsymbol{v}_i}) = \max_{i=0,\dots,2} r \cdot \operatorname{arc}(\widehat{\boldsymbol{s}, \boldsymbol{v}_i}),$$

wobei $\operatorname{arc}(\widehat{\boldsymbol{s}, \boldsymbol{x}})$ der Winkel des Großkreisbogens zwischen $\boldsymbol{s}$ und $\boldsymbol{x}$ in Radiant darstellt.

Der Vorteil dieser Methode ist, daß sie einfach zu berechnen ist. Nachteilig ist, daß der Korrekturterm nur von der Region und dem Beispielpunkt $\boldsymbol{s}$ abhängt, nicht jedoch von dem konkreten minimalen Abstand des Beispielpunktes zu der Flächenmenge.

2. *Differenz zwischen dem minimalen Abstand des Beispielpunktes und der Flächenmenge und dem maximalen Abstand des Punktes der Flächenmenge, der den minimalen Abstand zu dem Beispielpunkt definiert, zu der Region des Beispielpunktes*
Die Idee dieser Vorgehensweise ist, daß für kleine Regionen der Punkt $\boldsymbol{p}_{min}$ in der Flächenmenge, der den minimalen Abstand von den Punkten der Region zu der Flächenmenge bestimmt, entweder eindeutig ist oder wenig variiert, so daß der maximale Abstand von $\boldsymbol{p}_{min}$ zu der Region entweder dem wahren *gerichteten Hausdorff-Abstand* entspricht oder ihm sehr nahe kommt. Die Differenz des zuvor berechneten minimalen Abstandes des Beispielpunktes zu der Flächenmenge und diesem maximalen Abstand führt daher i.d.R. zu einer besseren Approximation des tatsächlichen *gerichteten Hausdorff-Abstands* als die vorige Methode, ist aber aufwendiger zu berechnen. D.h. sei $\boldsymbol{p}_{min}$ der Punkt der Flächenmenge F, der den minimalen Abstand zwischen $\boldsymbol{s}$ und F bestimmt, dann ergibt sich der Korrekturterm $t(\boldsymbol{s}, R, F)$ zu

$$t(\boldsymbol{s}, R, F) = \max_{\boldsymbol{r} \in R} d(\boldsymbol{p}_{min}, \boldsymbol{r}) - \underbrace{d(\boldsymbol{s}, F)}_{= \, d(\boldsymbol{s}, \boldsymbol{p}_{min})} .$$

Mit diesem Korrekturterm ergibt sich die Approximation des *gerichteten Hausdorff-Abstands* mit Hilfe des Beispielpunktes $\boldsymbol{s}$ zu

$$d(\boldsymbol{s}, F) + \left(\max_{\boldsymbol{r} \in R} d(\boldsymbol{p}_{min}, \boldsymbol{r}) - d(\boldsymbol{s}, F) \right) = \max_{\boldsymbol{r} \in R} d(\boldsymbol{p}_{min}, \boldsymbol{r}).$$

D.h. der Beispielpunkt $\boldsymbol{s}$ wird nur dazu benötigt, um den Punkt $\boldsymbol{p}_{min}$ der Flächenmenge F zu bestimmen, der den minimalen Abstand zu $\boldsymbol{s}$ besitzt. Betrachten wir die Definition des *gerichteten Hausdorff-Abstands* $d_{HA}(R, F)$, so ergibt sich der Zusammenhang

$$\max_{\boldsymbol{r}\in R} \min_{\boldsymbol{p}\in F} d(\boldsymbol{r}, \boldsymbol{p}) \geq \max_{\boldsymbol{r}\in R} d(\boldsymbol{r}, \boldsymbol{p}) \qquad \text{für } \boldsymbol{p} \in F \text{ beliebig.}$$

Die Verwendung von $\boldsymbol{p}_{min}$ ist somit eine heuristisch gute Wahl für den betrachteten Punkt $\boldsymbol{p}$ aus F, wodurch die Approximation von $d_{HA}(R, F)$ möglichst genau ist.

Betrachten wir das zuvor vorgestellte Verfahren zur Bestimmung des minimalen Abstandes eines Punktes zu einer Flächenmenge, so sehen wir, daß neben dem Abstand auch der Punkt der Flächenmenge bestimmt werden kann, der diesen Abstand bestimmt. Daher können wir an dieser Stelle davon ausgehen, daß dieser Punkt $\boldsymbol{p}_{min}$ bereits vorliegt und uns damit beschäftigen, wie wir für $\boldsymbol{p}_{min}$ den maximalen Abstand zu der Region R bestimmen. Dies ist unterschiedlich bei der rechteckigen, planaren Region der Iso-Box, Box und dem *Eulerschen* Dreieck der Kugel, so daß wir diese beiden Fälle in der Folge gesondert betrachten werden.

Iso-Box, Box

Im Fall des Rechtecks $R(\boldsymbol{v}_0, \ldots, \boldsymbol{v}_3)$ wird der maximale Abstand zwischen einem Punkt $\boldsymbol{p}$ und einem Punkt des Rechtecks in einem der Eckpunkte des Rechtecks angenommen, d.h.

$$\max_{\boldsymbol{r}\in R} d(\boldsymbol{p}, \boldsymbol{r}) = \max_{i=0,\ldots,3} d(\boldsymbol{p}, \boldsymbol{v}_i).$$

Kugel

Der Fall des *Eulerschen* Dreiecks $D(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2)$ mit $\boldsymbol{v}_3 \equiv \boldsymbol{v}_0$ (wobei die Eckpunkte von außerhalb der Kugel im Gegenuhrzeigersinn aufgezählt sind) auf der Kugel $K(\boldsymbol{c}, r)$ ist aufwendiger. Der Punkt $\boldsymbol{p}$ der Flächenmenge liegt dabei innerhalb der F umhüllenden Kugel K. Der maximale Abstand von $\boldsymbol{p}$ zu D wird bestimmt durch den maximalen euklidischen Abstand von $\boldsymbol{p}$ zu

- der Kugel K, auf der D liegt,
- einer Kante $(\boldsymbol{v}_i, \boldsymbol{v}_{i+1})$ von $D \quad i = 0, 1, 2$ und
- einem Eckpunkt $\boldsymbol{v}_i$ von $D \quad i = 0, 1, 2$.

(a) *Abstand* $\boldsymbol{p}$ *zur Kugel* K*, auf der* D *liegt*
Der maximale Abstand eines Punktes $\boldsymbol{p}$ innerhalb der Kugel zur Kugel wird im Schnittpunkt des Strahls ausgehend von $\boldsymbol{p}$ durch den Mittelpunkt $\boldsymbol{c}$ der Kugel

angenommen. Ist $\boldsymbol{p}$ gleich dem Mittelpunkt der Kugel, so ist der Abstand in alle Richtungen gleich und der maximale Abstand von $\boldsymbol{p}$ zu D ist gleich r. Ansonsten liegt der Schnittpunkt des Strahles ausgehend von $\boldsymbol{p}$ mit der Kugeloberfläche genau dann innerhalb von D, falls

$$\begin{aligned}
((\boldsymbol{v}_0 - \boldsymbol{c}) \times (\boldsymbol{v}_1 - \boldsymbol{c}))^T \cdot (\boldsymbol{p} - \boldsymbol{c}) &\leq 0 \\
((\boldsymbol{v}_1 - \boldsymbol{c}) \times (\boldsymbol{v}_2 - \boldsymbol{c}))^T \cdot (\boldsymbol{p} - \boldsymbol{c}) &\leq 0 \\
((\boldsymbol{v}_2 - \boldsymbol{c}) \times (\boldsymbol{v}_0 - \boldsymbol{c}))^T \cdot (\boldsymbol{p} - \boldsymbol{c}) &\leq 0
\end{aligned}$$

und der maximale Abstand ergibt sich dann zu

$$\max_{\boldsymbol{d} \in D} d(\boldsymbol{p}, \boldsymbol{d}) = |\boldsymbol{p} - \boldsymbol{c}| + r.$$

Die Situation ist in Abbildung 3.13 graphisch veranschaulicht.

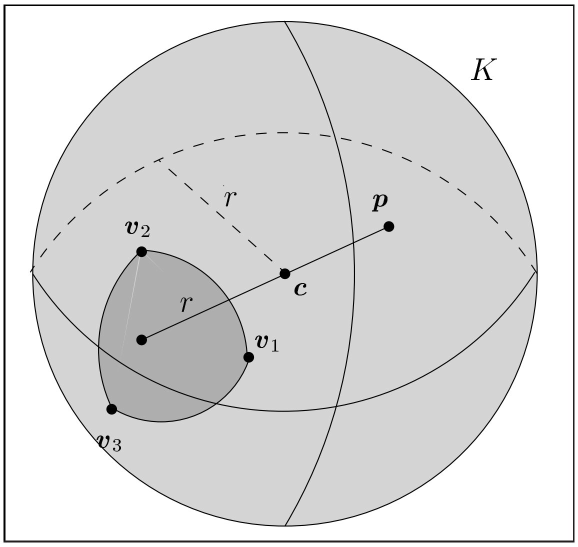

Abbildung 3.13: Maximaler Abstand Punkt – Kugel

(b) *Abstand* $\boldsymbol{p}$ *zu einer Kante* $(\boldsymbol{v}_i, \boldsymbol{v}_{i+1})$ *von* D $(i = 0, 1, 2)$
Zur Bestimmung des maximalen Abstands von $\boldsymbol{p}$ zu der Kante $(\boldsymbol{v}_i, \boldsymbol{v}_{i+1})$ projizieren wir den Punkt $\boldsymbol{p}$ auf die Ebene, in der der Großkreis zu $(\boldsymbol{v}_i, \boldsymbol{v}_{i+1})$ liegt und die durch die Punkte $\boldsymbol{v}_i$, $\boldsymbol{v}_{i+1}$ und $\boldsymbol{c}$ geht. Diese Ebene hat die Ebenengleichung $\left(\frac{(\boldsymbol{v}_i - \boldsymbol{c}) \times (\boldsymbol{v}_{i+1} - \boldsymbol{c})}{|(\boldsymbol{v}_i - \boldsymbol{c}) \times (\boldsymbol{v}_{i+1} - \boldsymbol{c})|}\right)^T \cdot (\boldsymbol{x} - \boldsymbol{c}) = 0$. Der projizierte Punkt $\boldsymbol{p}^{proj}$ ergibt sich somit zu

$$\boldsymbol{p}^{proj} = \boldsymbol{p} - \frac{((\boldsymbol{v}_i - \boldsymbol{c}) \times (\boldsymbol{v}_{i+1} - \boldsymbol{c}))^T \cdot (\boldsymbol{p} - \boldsymbol{c})}{|(\boldsymbol{v}_i - \boldsymbol{c}) \times (\boldsymbol{v}_{i+1} - \boldsymbol{c})|^2} \cdot ((\boldsymbol{v}_i - \boldsymbol{c}) \times (\boldsymbol{v}_{i+1} - \boldsymbol{c})).$$

Der Punkt $\boldsymbol{p}_{max}$ auf dem Großkreis durch $\boldsymbol{v}_i$ und $\boldsymbol{v}_{i+1}$, der den Abstand zu $\boldsymbol{p}$ maximiert, ist der Schnittpunkt des Strahles ausgehend von $\boldsymbol{p}^{proj}$ durch den Mittelpunkt der Kugel mit dem Großkreis. Der Punkt $\boldsymbol{p}_{max}$ liegt innerhalb der Kante $(\boldsymbol{v}_i, \boldsymbol{v}_{i+1})$, falls

$$\begin{aligned}((\boldsymbol{v}_i - \boldsymbol{c}) \times ((\boldsymbol{v}_i - \boldsymbol{c}) \times (\boldsymbol{v}_{i+1} - \boldsymbol{c})))^T \cdot (\boldsymbol{p} - \boldsymbol{c}) &\geq 0 \\ ((\boldsymbol{v}_{i+1} - \boldsymbol{c}) \times ((\boldsymbol{v}_i - \boldsymbol{c}) \times (\boldsymbol{v}_{i+1} - \boldsymbol{c})))^T \cdot (\boldsymbol{p} - \boldsymbol{c}) &\leq 0.\end{aligned}$$

Dies erkennen wir, indem wir die Lage des Punktes $\boldsymbol{p}^{proj}$ in der Projektionsebene bzgl. der beiden Geraden durch $\boldsymbol{c}$ und $\boldsymbol{v}_i$ bzw. durch $\boldsymbol{c}$ und $\boldsymbol{v}_{i+1}$ betrachten. Anstelle der Betrachtung des zweidimensionalen Problems dehnen wir die Geraden senkrecht zur Projektionsebene zu Ebenen aus und argumentieren direkt über den Punkt $\boldsymbol{p}$. Für den maximalen Abstand $\max\limits_{\boldsymbol{v} \in (\boldsymbol{v}_i, \boldsymbol{v}_{i+1})} d(\boldsymbol{p}, \boldsymbol{v})$ gilt dann:

$$\begin{aligned}\max_{\boldsymbol{v} \in (\boldsymbol{v}_i, \boldsymbol{v}_{i+1})} \delta(\boldsymbol{p}, \boldsymbol{v}) &= \sqrt{(\boldsymbol{p} - \boldsymbol{p}^{proj})^2 + (|\boldsymbol{p}^{proj} - \boldsymbol{c}| + r)^2} \\ &= \sqrt{|\boldsymbol{p} - \boldsymbol{c}|^2 + r^2 + 2 \cdot r \cdot |\boldsymbol{p}^{proj} - \boldsymbol{c}|}\end{aligned}$$

Die graphische Veranschaulichung dieser Berechnung ist aus Abbildung 3.14 ersichtlich.

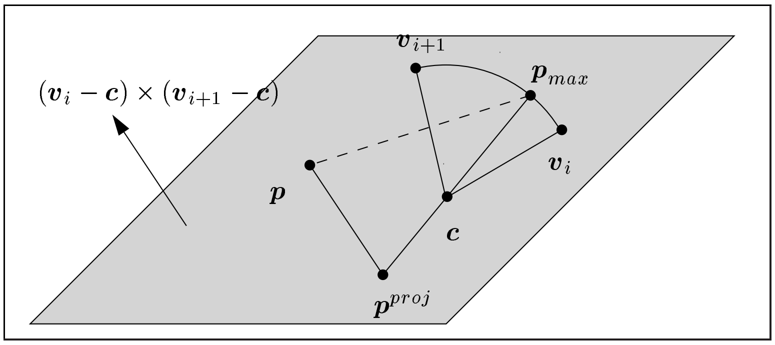

Abbildung 3.14: MAXIMALER ABSTAND PUNKT – GROSSKREISABSCHNITT

(c) *Abstand* $\boldsymbol{p}$ *zu einem Eckpunkt* $\boldsymbol{v}_i$ *von* D $(i = 0, 1, 2)$
Der maximale Abstand zwischen $\boldsymbol{p}$ und $\boldsymbol{v}_i$ ergibt sich zu

$$d(\boldsymbol{p}, \boldsymbol{v}_i) = |\boldsymbol{p} - \boldsymbol{v}_i|.$$

Auswahl der Beispielpunkte einer Region

Die Auswahl der Beispielpunkte für eine Region hängt von der Art des benutzten Korrekturterms ab.

- *Maximaler Abstand des Beispielpunktes zu einem Punkt in der Region*
  Wählen wir den maximalen Abstand des Beispielpunktes zu einem Punkt in der Region als Korrekturterm, so sollte der Beispielpunkt so gelegt werden, daß dieser maximale Abstand möglichst minimiert wird. Im Fall der rechteckigen Region $R(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$ ist der Mittelpunkt $\boldsymbol{c}$ mit

  $$\boldsymbol{c} = \sum_{i=0}^{3} \boldsymbol{v}_i$$

  ein geeigneter Beispielpunkt.

  Für das *Eulersche* Dreieck $D(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2)$ ist dies die Projektion des Schnittpunktes $\boldsymbol{z}$ der Mittelsenkrechten des zugehörigen planaren Dreiecks $D_p(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2)$ auf die Kugel, falls $\boldsymbol{z}$ innerhalb von D_p liegt oder die Projektion des Mittelpunktes der längsten Kante auf die Kugel, falls $\boldsymbol{z}$ außerhalb von D_p liegt. Zur Berechnung des Schnittpunkts der Mittelsenkrechten im $\mathbb{R}^3$ erweitern wir diese zu Ebenen senkrecht zu der Ebene, in der D_p liegt. Daraus ergeben sich die drei Ebenen $e_i : (\boldsymbol{v}_{((i+1) \mod 3)} - \boldsymbol{v}_i)^T \cdot (\boldsymbol{x} - \frac{1}{2} \cdot (\boldsymbol{v}_i + \boldsymbol{v}_{((i+1) \mod 3)}) = 0$ und die Ebene, in der das Dreieck liegt $e : ((\boldsymbol{v}_1 - \boldsymbol{v}_0) \times (\boldsymbol{v}_2 - \boldsymbol{v}_0))^T \cdot (\boldsymbol{x} - \boldsymbol{v}_0) = 0$. Der Schnittpunkt $\boldsymbol{z}$ zweier der drei Ebenen e_i und der Ebene e ergibt den Schnittpunkt der Mittelsenkrechten und dieser berechnet sich durch

  $$\boldsymbol{z} = \begin{pmatrix} (\boldsymbol{v}_1 - \boldsymbol{v}_0)^T \\ (\boldsymbol{v}_2 - \boldsymbol{v}_1)^T \\ ((\boldsymbol{v}_1 - \boldsymbol{v}_0) \times (\boldsymbol{v}_2 - \boldsymbol{v}_0))^T \end{pmatrix}^{-1} \cdot \begin{pmatrix} \frac{1}{2} \cdot (\boldsymbol{v}_1 - \boldsymbol{v}_0)^T \cdot (\boldsymbol{v}_0 + \boldsymbol{v}_1) \\ \frac{1}{2} \cdot (\boldsymbol{v}_2 - \boldsymbol{v}_1)^T \cdot (\boldsymbol{v}_1 + \boldsymbol{v}_2) \\ ((\boldsymbol{v}_1 - \boldsymbol{v}_0) \times (\boldsymbol{v}_2 - \boldsymbol{v}_0))^T \cdot \boldsymbol{v}_0 \end{pmatrix}.$$

  Der zu projizierende Punkt $\boldsymbol{p}$ von D_p ergibt sich somit zu

  $$\boldsymbol{p} = \begin{cases} \boldsymbol{z} & \text{falls } \boldsymbol{z} \in D_p(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2), \\ \frac{1}{2} \cdot (\boldsymbol{v}_i + \boldsymbol{v}_{i+1}) & \text{sonst, für } (\boldsymbol{v}_i, \boldsymbol{v}_{i+1}) = \max\limits_{e \in E(D_p)} |e| \end{cases}$$

  Die Projektion des Punktes $\boldsymbol{p}$ auf die Kugeloberfläche und damit der Beispielpunkt $\boldsymbol{s}$ ergibt sich sodann als

  $$\boldsymbol{s} = \boldsymbol{c} + r \cdot \frac{\boldsymbol{p} - \boldsymbol{c}}{|\boldsymbol{p} - \boldsymbol{c}|}.$$

- *Differenz zwischen minimalem und maximalem Abstand*
  In diesem Fall gibt es keine a priori gute Wahl der Beispielpunkte. Daher ist es sinnvoll, Beispielpunkte zu wählen, die in der Approximation des tatsächlichen *gerichteten Hausdorff-Abstands* von möglichst vielen Regionen benutzt werden können, um den Aufwand für die Berechnung des minimalen Abstandes des Beispielpunktes zu der Flächenmenge zu reduzieren. Daher haben wir die jeweiligen Eckpunkte der Regionen als Beispielpunkte benutzt.

Kapitel 4

Verwendung von Hüllkörperhierarchien

In Kapitel 3 wird beschrieben, wie für Objekte *Hüllkörperhierachien* mit einfachen Hüllkörpern berechnet werden, die aufgrund der Information zum Vorberechnungszeitpunkt hinsichtlich verschiedener Optimierungskriterien möglichst geeignet sind. Diese vorberechneten Hierarchien werden dazu benutzt, um den Kollisionserkennungsprozeß effizient durchführen zu können. Mit Hilfe von *Raumpartitionierungsverfahren* (vgl. Kapitel 6) kann im Vorfeld der Kollisionserkennung für Objektpaare die Anzahl der zu testenden Objektpaare reduziert werden. Vergleichen wir nun ein Objektpaar auf Kollision hin, so liegen zu beiden Objekten je eine Hüllkörperhierarchie vor. Diese werden eingesetzt, um möglichst schnell feststellen zu können, ob eine Kollision zwischen den beiden Objekten während einer Bewegung ausgeschlossen werden kann, oder falls eine vorliegt, diese räumlich möglichst schnell einzugrenzen.

Dazu müssen wir zum einen beschreiben, wie die einzelnen in der Hierarchie vorkommenden Hüllkörpertypen effizient gegeneinander getestet werden und zum anderen wie die beiden Hüllkörperhierarchien während des Kollisionserkennungsprozesses traversiert werden.

Bei der Berechnung von Hüllkörperhierarchien in Kapitel 3 haben wir drei Typen von Hüllkörpern zugelassen:

1. Kugel
2. achsenorientierte Box (Iso-Box)
3. beliebig orientierte Box (Box)

In der Folge werden wir nun in Abschnitt 4.1 für jedes Paar von Hüllkörpertypen ein *statisches* Kollisionserkennungsverfahren vorstellen, daß einen effizienten Test der Hüllkörper ermöglicht. Diese *statischen* Verfahren werden dann in Abschnitt 4.2 auf *dynamische*

Verfahren erweitert, so daß Hüllkörperhierarchien innerhalb eines *dynamischen* Kollisionserkennungsverfahrens eingesetzt werden können. Schließlich werden in Abschnitt 4.3 Verfahren dargelegt, wie Hüllkörperhierarchien während des Kollisionserkennungsprozesses traversiert werden können.

4.1 Statischer Kollisionstest für einfache Hüllkörper

4.1.1 Kugel–Kugel (statisch)

Der einfachste und schnellste Kollisionstest ist der zweier Kugeln. Zwei Kugeln $K_1 = (\boldsymbol{c}_1, r_1)$ und $K_2 = (\boldsymbol{c}_2, r_2)$ kollidieren genau dann zu einem Zeitpunkt t, wenn der Abstand der Mittelpunkte der beiden Kugeln zum Zeitpunkt t geringer ist als die Summe der Radien der beiden Kugeln. Zur effizienteren Berechnung kann man anstatt des Abstandes und der Summe der Radien auch deren Quadrate betrachten. Daraus ergibt sich folgender einfacher Algorithmus *KugelKugelStatisch* in Abbildung 4.1.

Algorithmus *KugelKugelStatisch*$(K_1, \boldsymbol{T}_{K_1}, K_2, \boldsymbol{T}_{K_2})$

Eingabe:	$K_1 = (\boldsymbol{c}_1, r_1)$	Kugel 1
	$\boldsymbol{T}_{K_1}$	Position der Kugel 1 im Weltkoordinatensystem
	$K_2 = (\boldsymbol{c}_2, r_2)$	Kugel 2
	$\boldsymbol{T}_{K_2}$	Position der Kugel 2 im Weltkoordinatensystem
Ausgabe:	c	Kollisionsflag

1. **if** $(|\boldsymbol{T}_{K_2} \cdot \boldsymbol{c}_2 - \boldsymbol{T}_{K_1} \cdot \boldsymbol{c}_1|^2 < (r_1 + r_2)^2)$
2. **then return** COLLISION
3. **else return** NO COLLISION

Abbildung 4.1: ALGORITHMUS: TEST KUGEL–KUGEL (STATISCH)

4.1.2 Kugel–Iso-Box und Kugel–Box (statisch)

Der Kollisionstest zwischen Kugel und Iso-Box/Box ist am einfachsten dann durchzuführen, wenn die Kugel in das Koordinatensystem der Iso-Box/Box transformiert wird. In diesem Fall läßt sich die Box durch zwei Vektoren $\boldsymbol{b}^{min}$ und $\boldsymbol{b}^{max}$ beschreiben. Eine Überlappung zwischen Box und Kugel liegt genau dann vor, wenn der Abstand zwischen der Box und dem Mittelpunkt der Kugel kleiner als der Radius der Kugel ist. Dieser minimale Abstand ist das Ergebnis der Optimierung

$$\begin{array}{ll} \min\limits_{\boldsymbol{p} \in \mathbb{R}^3} dist(\boldsymbol{p}) = \sqrt{(\boldsymbol{c}_1 - \boldsymbol{p}_1)^2 + (\boldsymbol{c}_2 - \boldsymbol{p}_2)^2 + (\boldsymbol{c}_3 - \boldsymbol{p}_3)^2} & \\ \text{u.d.N.} \quad \boldsymbol{b}_i^{min} \leq \boldsymbol{p}_i \leq b_i^{max} \qquad (i = 1, 2, 3). & \end{array}$$

Anstatt des Abstands kann auch das Quadrat des Abstands minimiert und mit dem Quadrat des Radius verglichen werden.

Jeder Summand der Abstandsformel ist nicht negativ, daher kann jeder Summand individuell minimiert werden. Dies bedeutet für die Komponente i der Lösung, daß $\boldsymbol{p}_i = \boldsymbol{c}_i$, falls $\boldsymbol{b}_i^{min} \leq \boldsymbol{c}_i \leq \boldsymbol{b}_i^{max}$. Im anderen Fall wird $\boldsymbol{p}_i$ auf die Grenze $\boldsymbol{b}_i^{min}$ oder $\boldsymbol{b}_i^{max}$ gesetzt, die näher an $\boldsymbol{c}_i$ liegt. Durch diese einfache Vorgehensweise ergibt sich sowohl der Punkt mit minimalem Abstands in der Box als auch dessen Abstand zum Mittelpunkt der Kugel, der dann mit dem Radius der Kugel verglichen werden kann. Diese Vorgehensweise wird von Arvo in [Gla90] beschrieben. Daraus resultiert der Algorithmus *KugelBoxStatisch* in Abbildung 4.2.

Algorithmus *KugelBoxStatisch*$(K, \boldsymbol{T}_K, B, \boldsymbol{T}_B)$

Eingabe:	$K = (\boldsymbol{c}, r)$	Kugel
	$\boldsymbol{T}_K$	Position der Kugel im Weltkoordinatensystem
	$B = (\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$	Iso-Box/Box
	$\boldsymbol{T}_B$	Position der Box im Weltkoordinatensystem

Ausgabe: c Kollisionsflag

$\boldsymbol{c}_{new} \leftarrow \boldsymbol{T}_B^{-1} \cdot \boldsymbol{T}_K \cdot \boldsymbol{c}$
$d_{sqr}^{min} \leftarrow 0$
for $i \leftarrow 1$ **to** 3
do if $(\boldsymbol{c}_i < \boldsymbol{b}_i^{min})$
    then $d_{sqr}^{min} \leftarrow d_{sqr}^{min} + (\boldsymbol{c}_i - \boldsymbol{b}_i^{min})^2$
    else **if** $(\boldsymbol{c}_i > \boldsymbol{b}_i^{max})$
        then $d_{sqr}^{min} \leftarrow d_{sqr}^{min} + (\boldsymbol{c}_i - \boldsymbol{b}_i^{max})^2$
if $(d_{sqr}^{min} < r^2)$
then return COLLISION
else **return** NO COLLISION

Abbildung 4.2: Algorithmus: Test Kugel–Iso-Box und Kugel–Box (statisch)

4.1.3 Iso-Box–Iso-Box (statisch)

Zwei Iso-Boxen überlappen sich genau dann, wenn sich ihre Projektionen auf die drei Koordinatenachsen überlappen, d.h. der Test beschränkt sich auf den Überlappungstest von drei Intervallen. Problematisch bei dem Test ist nur, daß zum Testzeitpunkt t die beiden Boxen durch die wirkenden Transformationen aus der Ausgangslage nicht mehr achsenorientiert sind. Um den Effekt, daß durch die Transformationen die Iso-Boxen nicht mehr achsenorientiert sind, auf eine Iso-Box zu beschränken, transformieren wir eine Iso-Box in das Koordinatensystem der anderen, die dadurch im Gegensatz zu der anderen Iso-Box i.a. nicht mehr achsenorientiert ist. Für diese wird mit Hilfe der acht aktuellen Eckpunkte der

Box die achsenorientierte Box berechnet, die diese Box einhüllt. Dadurch ergeben sich wieder zwei Iso-Boxen, die mittels des Intervalltests gegeneinander auf Überlappung getestet werden können. Dadurch ergibt sich der Algorithmus *IsoBoxIsoBoxStatisch* in Abbildung 4.3.

Algorithmus *IsoBoxIsoBoxStatisch*$(B_1, \boldsymbol{T}_{B_1}, B_2, \boldsymbol{T}_{B_2})$

Eingabe:	$B_1 = (\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$	Iso-Box 1
	$\boldsymbol{T}_{B_1}$	Position Iso-Box 1 im Weltkoordinatensystem
	$B_2 = \boldsymbol{v}_1, \ldots, \boldsymbol{v}_8$	Iso-Box 2
	$\boldsymbol{T}_{B_2}$	Position Iso-Box 2 im Weltkoordinatensystem

Ausgabe: c Kollisionsflag

$\boldsymbol{a}^{min} \leftarrow (\infty, \infty, \infty)$
$\boldsymbol{a}^{max} \leftarrow (-\infty, -\infty, -\infty)$
for $i \leftarrow 1$ **to** 8
do $\boldsymbol{v} \leftarrow \boldsymbol{T}_{B_1}^{-1} \cdot \boldsymbol{T}_{B_2} \cdot \boldsymbol{v}_i$
  for $j \leftarrow 1$ **to** 3
  do if $(\boldsymbol{v}_j < \boldsymbol{a}_j^{min})$
    then $\boldsymbol{a}_j^{min} \leftarrow \boldsymbol{v}_j$
    if $(\boldsymbol{v}_j > \boldsymbol{a}_j^{max})$
    then $\boldsymbol{a}_j^{max} \leftarrow \boldsymbol{v}_j$
for $i \leftarrow 1$ **to** 3
do if $(\boldsymbol{a}_i^{max} < \boldsymbol{b}_i^{min})$
  then return NO COLLISION
  if $(\boldsymbol{b}_i^{max} < \boldsymbol{a}_i^{min})$
  then return NO COLLISION
return COLLISION

Abbildung 4.3: ALGORITHMUS: TEST ISO-BOX–ISO-BOX (STATISCH)

4.1.4 Iso-Box–Box und Box–Box (statisch)

Für den Test einer Iso-Box mit einer Box sind zwei Varianten vorstellbar. Entweder wird der Test Iso-Box–Iso-Box angewendet, indem für die Box die Iso-Box berechnet wird nach der Transformation der Box in das Koordinatensystem der Iso-Box oder die Iso-Box wird wie eine Box behandelt und ein Box–Box Test wird durchgeführt.

Ein effizienter Box-Box Test wurde von GOTTSCHALK in [GLM96] beschrieben. Er ist deutlich schneller als der kanonische Test, bei dem die beiden zu testenden Boxen wie Polyeder behandelt werden, für die ein statischer Kollisionstest durchgeführt wird, wobei bei der Berechnung die parallelen Strukturen der Boxen ausgenutzt werden. Der Test basiert auf der Tatsache, daß zwei nichtüberlappende konvexe Polyeder durch eine Ebene getrennt werden können, die parallel zu einer Fläche eines Polyeders oder zu je einer Kante der beiden Polyeder ist (zum Beweis siehe [Go96]). Daraus resultiert, daß zwei konvexe Polyeder genau dann nicht überlappen, wenn es eine trennende Achse orthogonal zu einer Fläche oder zu je einer Kante der beiden Polyeder gibt. Im Fall der Iso-Boxen wird während des Tests genau diese Tatsache angewendet, wobei die trennenden Achsen, die getestet werden, die drei Koordinatenachsen sind. Im Fall beliebig orientierter Boxen kann die Anzahl der Achsen, die zu testen sind, auf 15 eingeschränkt werden, denn jede Box hat 3 Flächenrichtungen und 3 Kantenrichtungen, so daß sich 6 Achsenrichtungen aus den Flächen und 9 Achsenrichtungen aus der Kombination der Kantenrichtungen ergeben.

Um nun den Test, ob eine der Achsen eine trennende Achse darstellt, möglichst einfach zu gestalten, wird der Test im Koordinatensystem einer der beiden Boxen durchgeführt, wobei der Mittelpunkt des Koordinatensystems der Box in ihrem Mittelpunkt liegt. Dadurch vereinfacht sich die Projektion dieser Box auf eine der Koordinatenachsen und nur für die zweite muß ein etwas größerer Rechenaufwand durchgeführt werden.

Seien nun $B_i = (\boldsymbol{c}_i, \boldsymbol{r}_i) \quad (i = 1, 2)$ die beiden Boxen mit Mittelpunkt $\boldsymbol{c}_i$ und Radiusvektor $\boldsymbol{r}_i$ mit den aktuellen Transformationen $\boldsymbol{T}_{B_i}$, so ergibt sich die relative Positionierung von B_2 im Koordinatensystem von B_1 durch $\boldsymbol{T}_{rel} = \boldsymbol{T}_{B_1}^{-1} \cdot \boldsymbol{T}_{B_2}$. $\boldsymbol{T}_{rel}$ besteht aus einer Rotation $\boldsymbol{R}$ und einer Translation $\boldsymbol{t}$. Durch die Transformation sind die Achsen der Box B_1 die Einheitsvektoren $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, $\boldsymbol{e}_3$ und die von B_2 die Spalten $\boldsymbol{R}_{\cdot i} \quad (i = 1, 2, 3)$ der Rotationsmatrix $\boldsymbol{R}$.

Die Projektion des Mittelpunktes einer Box auf eine Achse ist der Mittelpunkt des entstehenden Intervalls der gesamten Projektion. Der Radius des Intervalls ergibt sich als Summe der Längen der Projektionen der Radien der Box auf die Achse. Für eine Box $B = (\boldsymbol{c}, \boldsymbol{r})$ mit einer Transformation $T = (\boldsymbol{R}, \boldsymbol{t})$ ergibt sich somit folgendes Intervall $I = (c, r)$ bei der Projektion auf eine Achse mit dem Richtungsvektor $\boldsymbol{l}$ mit Einheitslänge, die durch den Nullpunkt des Koordinatensystems geht:

$$c = (T \cdot \boldsymbol{c}) \cdot \boldsymbol{l} \qquad r = \sum_{i=1}^{3} |\boldsymbol{r}_i \cdot \boldsymbol{R}_{\cdot i} \cdot \boldsymbol{l}|$$

Zwei Intervalle $I_i = (c_i, r_i) \quad (i = 1, 2)$ überlappen sich genau dann, wenn ihre Mittelpunkte einen kleineren Abstand haben als die Summe der beiden Radien, d.h. falls $|c_2 - c_1| < r_1 + r_2$.

Überträgt man diesen Test auf die beiden Boxen $B_i \quad (i = 1, 2)$, trennt die Achse mit dem Richtungsvektor $\boldsymbol{l}$ mit Einheitslänge, die durch den Mittelpunkt der Box B_1 geht, genau dann die beiden Boxen, falls

$$|\boldsymbol{t} \cdot \boldsymbol{l}| \geq \sum_{i=1}^{3} |\boldsymbol{r}_{1i} \cdot \boldsymbol{e}_i \cdot \boldsymbol{l}| + \sum_{i=1}^{3} |\boldsymbol{r}_{2i} \cdot \boldsymbol{R}_{\cdot i} \cdot \boldsymbol{l}|$$

ist. Werden diese Berechnungen für die 15 möglichen Achsenorientierungen durchgeführt und entsprechend den Voraussetzungen vereinfacht, so ergeben sich folgende Tests:

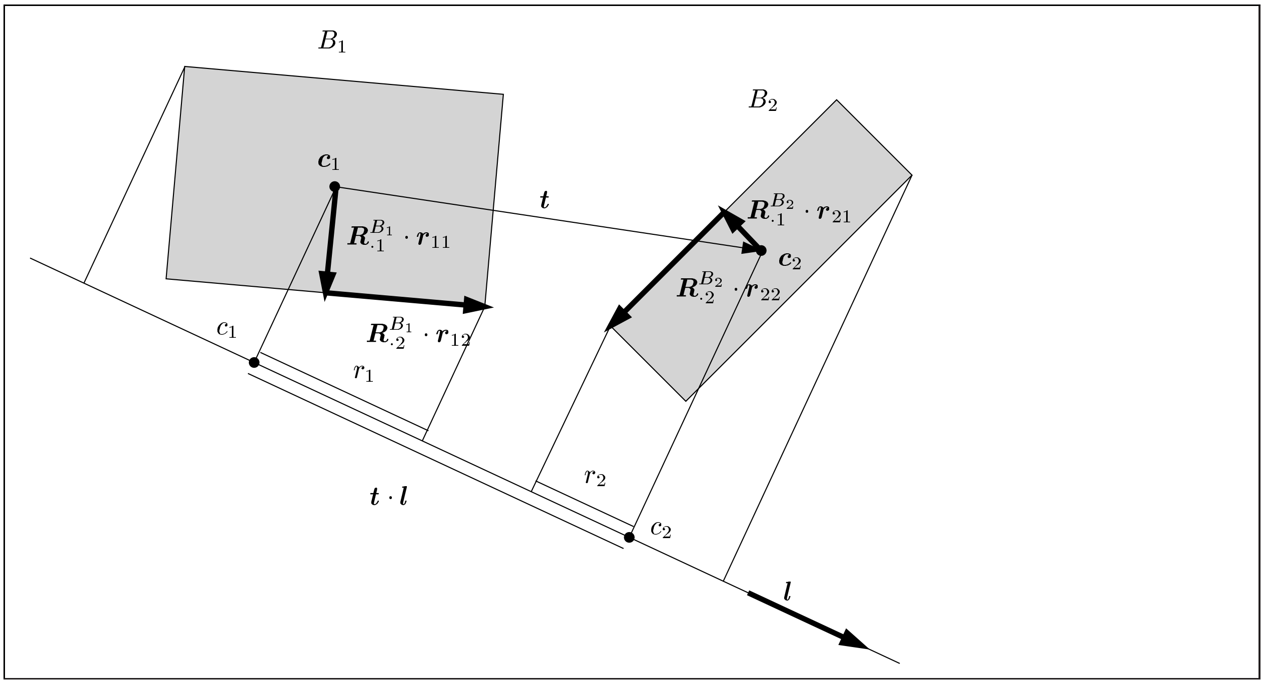

Abbildung 4.4: Kollisionstest Box – Box

Flächenrichtungen von B_1	
$\boldsymbol{e}_1$	$\lvert\boldsymbol{t}_1\rvert \geq \boldsymbol{r}_{11} + \boldsymbol{r}_{21} \cdot \lvert\boldsymbol{R}_{11}\rvert + \boldsymbol{r}_{22} \cdot \lvert\boldsymbol{R}_{12}\rvert + \boldsymbol{r}_{23} \cdot \lvert\boldsymbol{R}_{13}\rvert$
$\boldsymbol{e}_2$	$\lvert\boldsymbol{t}_2\rvert \geq \boldsymbol{r}_{12} + \boldsymbol{r}_{21} \cdot \lvert\boldsymbol{R}_{21}\rvert + \boldsymbol{r}_{22} \cdot \lvert\boldsymbol{R}_{22}\rvert + \boldsymbol{r}_{23} \cdot \lvert\boldsymbol{R}_{23}\rvert$
$\boldsymbol{e}_3$	$\lvert\boldsymbol{t}_3\rvert \geq \boldsymbol{r}_{13} + \boldsymbol{r}_{21} \cdot \lvert\boldsymbol{R}_{31}\rvert + \boldsymbol{r}_{22} \cdot \lvert\boldsymbol{R}_{32}\rvert + \boldsymbol{r}_{23} \cdot \lvert\boldsymbol{R}_{33}\rvert$
Flächenrichtungen von B_2	
$\boldsymbol{R}_{\cdot 1}$	$\lvert\boldsymbol{t} \cdot \boldsymbol{R}_{\cdot 1}\rvert \geq \boldsymbol{r}_{11} \cdot \lvert\boldsymbol{R}_{11}\rvert + \boldsymbol{r}_{12} \cdot \lvert\boldsymbol{R}_{21}\rvert + \boldsymbol{r}_{13} \cdot \lvert\boldsymbol{R}_{31}\rvert + \boldsymbol{r}_{21}$
$\boldsymbol{R}_{\cdot 2}$	$\lvert\boldsymbol{t} \cdot \boldsymbol{R}_{\cdot 2}\rvert \geq \boldsymbol{r}_{11} \cdot \lvert\boldsymbol{R}_{12}\rvert + \boldsymbol{r}_{12} \cdot \lvert\boldsymbol{R}_{22}\rvert + \boldsymbol{r}_{13} \cdot \lvert\boldsymbol{R}_{32}\rvert + \boldsymbol{r}_{22}$
$\boldsymbol{R}_{\cdot 3}$	$\lvert\boldsymbol{t} \cdot \boldsymbol{R}_{\cdot 3}\rvert \geq \boldsymbol{r}_{11} \cdot \lvert\boldsymbol{R}_{13}\rvert + \boldsymbol{r}_{12} \cdot \lvert\boldsymbol{R}_{23}\rvert + \boldsymbol{r}_{13} \cdot \lvert\boldsymbol{R}_{33}\rvert + \boldsymbol{r}_{23}$
Kombinierte Kantenrichtungen von B_1 und B_2	
$\boldsymbol{e}_1 \times \boldsymbol{R}_{\cdot 1}$	$\lvert\boldsymbol{t}_3 \cdot \boldsymbol{R}_{21} - \boldsymbol{t}_2 \cdot \boldsymbol{R}_{31}\rvert \geq \boldsymbol{r}_{12} \cdot \lvert\boldsymbol{R}_{31}\rvert + \boldsymbol{r}_{13} \cdot \lvert\boldsymbol{R}_{21}\rvert + \boldsymbol{r}_{22} \cdot \lvert\boldsymbol{R}_{13}\rvert + \boldsymbol{r}_{23} \cdot \lvert\boldsymbol{R}_{12}\rvert$
$\boldsymbol{e}_1 \times \boldsymbol{R}_{\cdot 2}$	$\lvert\boldsymbol{t}_3 \cdot \boldsymbol{R}_{22} - \boldsymbol{t}_2 \cdot \boldsymbol{R}_{32}\rvert \geq \boldsymbol{r}_{12} \cdot \lvert\boldsymbol{R}_{32}\rvert + \boldsymbol{r}_{13} \cdot \lvert\boldsymbol{R}_{22}\rvert + \boldsymbol{r}_{21} \cdot \lvert\boldsymbol{R}_{13}\rvert + \boldsymbol{r}_{23} \cdot \lvert\boldsymbol{R}_{11}\rvert$
$\boldsymbol{e}_1 \times \boldsymbol{R}_{\cdot 3}$	$\lvert\boldsymbol{t}_3 \cdot \boldsymbol{R}_{23} - \boldsymbol{t}_2 \cdot \boldsymbol{R}_{33}\rvert \geq \boldsymbol{r}_{12} \cdot \lvert\boldsymbol{R}_{33}\rvert + \boldsymbol{r}_{13} \cdot \lvert\boldsymbol{R}_{23}\rvert + \boldsymbol{r}_{21} \cdot \lvert\boldsymbol{R}_{12}\rvert + \boldsymbol{r}_{22} \cdot \lvert\boldsymbol{R}_{11}\rvert$
$\boldsymbol{e}_2 \times \boldsymbol{R}_{\cdot 1}$	$\lvert\boldsymbol{t}_1 \cdot \boldsymbol{R}_{31} - \boldsymbol{t}_3 \cdot \boldsymbol{R}_{11}\rvert \geq \boldsymbol{r}_{11} \cdot \lvert\boldsymbol{R}_{31}\rvert + \boldsymbol{r}_{13} \cdot \lvert\boldsymbol{R}_{11}\rvert + \boldsymbol{r}_{22} \cdot \lvert\boldsymbol{R}_{23}\rvert + \boldsymbol{r}_{23} \cdot \lvert\boldsymbol{R}_{22}\rvert$
$\boldsymbol{e}_2 \times \boldsymbol{R}_{\cdot 2}$	$\lvert\boldsymbol{t}_1 \cdot \boldsymbol{R}_{32} - \boldsymbol{t}_3 \cdot \boldsymbol{R}_{12}\rvert \geq \boldsymbol{r}_{11} \cdot \lvert\boldsymbol{R}_{32}\rvert + \boldsymbol{r}_{13} \cdot \lvert\boldsymbol{R}_{12}\rvert + \boldsymbol{r}_{21} \cdot \lvert\boldsymbol{R}_{23}\rvert + \boldsymbol{r}_{23} \cdot \lvert\boldsymbol{R}_{21}\rvert$
$\boldsymbol{e}_2 \times \boldsymbol{R}_{\cdot 3}$	$\lvert\boldsymbol{t}_1 \cdot \boldsymbol{R}_{33} - \boldsymbol{t}_3 \cdot \boldsymbol{R}_{13}\rvert \geq \boldsymbol{r}_{11} \cdot \lvert\boldsymbol{R}_{33}\rvert + \boldsymbol{r}_{13} \cdot \lvert\boldsymbol{R}_{13}\rvert + \boldsymbol{r}_{21} \cdot \lvert\boldsymbol{R}_{22}\rvert + \boldsymbol{r}_{22} \cdot \lvert\boldsymbol{R}_{21}\rvert$
$\boldsymbol{e}_3 \times \boldsymbol{R}_{\cdot 1}$	$\lvert\boldsymbol{t}_2 \cdot \boldsymbol{R}_{11} - \boldsymbol{t}_1 \cdot \boldsymbol{R}_{21}\rvert \geq \boldsymbol{r}_{11} \cdot \lvert\boldsymbol{R}_{21}\rvert + \boldsymbol{r}_{12} \cdot \lvert\boldsymbol{R}_{11}\rvert + \boldsymbol{r}_{22} \cdot \lvert\boldsymbol{R}_{33}\rvert + \boldsymbol{r}_{23} \cdot \lvert\boldsymbol{R}_{32}\rvert$
$\boldsymbol{e}_3 \times \boldsymbol{R}_{\cdot 2}$	$\lvert\boldsymbol{t}_2 \cdot \boldsymbol{R}_{12} - \boldsymbol{t}_1 \cdot \boldsymbol{R}_{22}\rvert \geq \boldsymbol{r}_{11} \cdot \lvert\boldsymbol{R}_{22}\rvert + \boldsymbol{r}_{12} \cdot \lvert\boldsymbol{R}_{12}\rvert + \boldsymbol{r}_{21} \cdot \lvert\boldsymbol{R}_{33}\rvert + \boldsymbol{r}_{23} \cdot \lvert\boldsymbol{R}_{31}\rvert$
$\boldsymbol{e}_3 \times \boldsymbol{R}_{\cdot 3}$	$\lvert\boldsymbol{t}_2 \cdot \boldsymbol{R}_{13} - \boldsymbol{t}_1 \cdot \boldsymbol{R}_{23}\rvert \geq \boldsymbol{r}_{11} \cdot \lvert\boldsymbol{R}_{23}\rvert + \boldsymbol{r}_{12} \cdot \lvert\boldsymbol{R}_{13}\rvert + \boldsymbol{r}_{21} \cdot \lvert\boldsymbol{R}_{32}\rvert + \boldsymbol{r}_{22} \cdot \lvert\boldsymbol{R}_{31}\rvert$

Insgesamt ergibt sich somit der Algorithmus *BoxBoxStatisch* in Abbildung 4.5.

Algorithmus *BoxBoxStatisch*$(B_1, \boldsymbol{T}_{B_1}, B_2, \boldsymbol{T}_{B_2})$

Eingabe:	$B_1 = (\boldsymbol{c}_1, \boldsymbol{r}_1)$	Box 1
	$\boldsymbol{T}_{B_1}$	Position Box 1 im Weltkoordinatensystem
	$B_2 = (\boldsymbol{c}_2, \boldsymbol{r}_2)$	Box 2
	$\boldsymbol{T}_{B_2}$	Position Box 2 im Weltkoordinatensystem

Ausgabe: c Kollisionsflag

$\boldsymbol{T}_{rel} \leftarrow \boldsymbol{T}_{B_1}^{-1} \cdot \boldsymbol{T}_{B_2}$; $\boldsymbol{R} \leftarrow Rotation(\boldsymbol{T}_{rel})$; $\boldsymbol{t} \leftarrow Translation(\boldsymbol{T}_{rel})$
if $(|\boldsymbol{t}_1| \geq \boldsymbol{r}_{11} + \boldsymbol{r}_{21} \cdot |\boldsymbol{R}_{11}| + \boldsymbol{r}_{22} \cdot |\boldsymbol{R}_{12}| + \boldsymbol{r}_{23} \cdot |\boldsymbol{R}_{13}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_2| \geq \boldsymbol{r}_{12} + \boldsymbol{r}_{21} \cdot |\boldsymbol{R}_{21}| + \boldsymbol{r}_{22} \cdot |\boldsymbol{R}_{22}| + \boldsymbol{r}_{23} \cdot |\boldsymbol{R}_{23}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_3| \geq \boldsymbol{r}_{13} + \boldsymbol{r}_{21} \cdot |\boldsymbol{R}_{31}| + \boldsymbol{r}_{22} \cdot |\boldsymbol{R}_{32}| + \boldsymbol{r}_{23} \cdot |\boldsymbol{R}_{33}|)$
then return NO COLLISION
if $(|\boldsymbol{t} \cdot \boldsymbol{R}_{\cdot 1}| \geq \boldsymbol{r}_{11} \cdot |\boldsymbol{R}_{11}| + \boldsymbol{r}_{12} \cdot |\boldsymbol{R}_{21}| + \boldsymbol{r}_{13} \cdot |\boldsymbol{R}_{31}| + \boldsymbol{r}_{21})$
then return NO COLLISION
if $(|\boldsymbol{t} \cdot \boldsymbol{R}_{\cdot 2}| \geq \boldsymbol{r}_{11} \cdot |\boldsymbol{R}_{12}| + \boldsymbol{r}_{12} \cdot |\boldsymbol{R}_{22}| + \boldsymbol{r}_{13} \cdot |\boldsymbol{R}_{32}| + \boldsymbol{r}_{22})$
then return NO COLLISION
if $(|\boldsymbol{t} \cdot \boldsymbol{R}_{\cdot 3}| \geq \boldsymbol{r}_{11} \cdot |\boldsymbol{R}_{13}| + \boldsymbol{r}_{12} \cdot |\boldsymbol{R}_{23}| + \boldsymbol{r}_{13} \cdot |\boldsymbol{R}_{33}| + \boldsymbol{r}_{23})$
then return NO COLLISION
if $(|\boldsymbol{t}_3 \cdot \boldsymbol{R}_{21} - \boldsymbol{t}_2 \cdot \boldsymbol{R}_{31}| \geq \boldsymbol{r}_{12} \cdot |\boldsymbol{R}_{31}| + \boldsymbol{r}_{13} \cdot |\boldsymbol{R}_{21}| + \boldsymbol{r}_{22} \cdot |\boldsymbol{R}_{13}| + \boldsymbol{r}_{23} \cdot |\boldsymbol{R}_{12}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_3 \cdot \boldsymbol{R}_{22} - \boldsymbol{t}_2 \cdot \boldsymbol{R}_{32}| \geq \boldsymbol{r}_{12} \cdot |\boldsymbol{R}_{32}| + \boldsymbol{r}_{13} \cdot |\boldsymbol{R}_{22}| + \boldsymbol{r}_{21} \cdot |\boldsymbol{R}_{13}| + \boldsymbol{r}_{23} \cdot |\boldsymbol{R}_{11}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_3 \cdot \boldsymbol{R}_{23} - \boldsymbol{t}_2 \cdot \boldsymbol{R}_{33}| \geq \boldsymbol{r}_{12} \cdot |\boldsymbol{R}_{33}| + \boldsymbol{r}_{13} \cdot |\boldsymbol{R}_{23}| + \boldsymbol{r}_{21} \cdot |\boldsymbol{R}_{12}| + \boldsymbol{r}_{22} \cdot |\boldsymbol{R}_{11}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_1 \cdot \boldsymbol{R}_{31} - \boldsymbol{t}_3 \cdot \boldsymbol{R}_{11}| \geq \boldsymbol{r}_{11} \cdot |\boldsymbol{R}_{31}| + \boldsymbol{r}_{13} \cdot |\boldsymbol{R}_{11}| + \boldsymbol{r}_{22} \cdot |\boldsymbol{R}_{23}| + \boldsymbol{r}_{23} \cdot |\boldsymbol{R}_{22}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_1 \cdot \boldsymbol{R}_{32} - \boldsymbol{t}_3 \cdot \boldsymbol{R}_{12}| \geq \boldsymbol{r}_{11} \cdot |\boldsymbol{R}_{32}| + \boldsymbol{r}_{13} \cdot |\boldsymbol{R}_{12}| + \boldsymbol{r}_{21} \cdot |\boldsymbol{R}_{23}| + \boldsymbol{r}_{23} \cdot |\boldsymbol{R}_{21}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_1 \cdot \boldsymbol{R}_{33} - \boldsymbol{t}_3 \cdot \boldsymbol{R}_{13}| \geq \boldsymbol{r}_{11} \cdot |\boldsymbol{R}_{33}| + \boldsymbol{r}_{13} \cdot |\boldsymbol{R}_{13}| + \boldsymbol{r}_{21} \cdot |\boldsymbol{R}_{22}| + \boldsymbol{r}_{22} \cdot |\boldsymbol{R}_{21}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_2 \cdot \boldsymbol{R}_{11} - \boldsymbol{t}_1 \cdot \boldsymbol{R}_{21}| \geq \boldsymbol{r}_{11} \cdot |\boldsymbol{R}_{21}| + \boldsymbol{r}_{12} \cdot |\boldsymbol{R}_{11}| + \boldsymbol{r}_{22} \cdot |\boldsymbol{R}_{33}| + \boldsymbol{r}_{23} \cdot |\boldsymbol{R}_{32}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_2 \cdot \boldsymbol{R}_{12} - \boldsymbol{t}_1 \cdot \boldsymbol{R}_{22}| \geq \boldsymbol{r}_{11} \cdot |\boldsymbol{R}_{22}| + \boldsymbol{r}_{12} \cdot |\boldsymbol{R}_{12}| + \boldsymbol{r}_{21} \cdot |\boldsymbol{R}_{33}| + \boldsymbol{r}_{23} \cdot |\boldsymbol{R}_{31}|)$
then return NO COLLISION
if $(|\boldsymbol{t}_2 \cdot \boldsymbol{R}_{13} - \boldsymbol{t}_1 \cdot \boldsymbol{R}_{23}| \geq \boldsymbol{r}_{11} \cdot |\boldsymbol{R}_{23}| + \boldsymbol{r}_{12} \cdot |\boldsymbol{R}_{13}| + \boldsymbol{r}_{21} \cdot |\boldsymbol{R}_{32}| + \boldsymbol{r}_{22} \cdot |\boldsymbol{R}_{31}|)$
then return NO COLLISION
return COLLISION

Abbildung 4.5: ALGORITHMUS: TEST BOX–BOX (STATISCH)

4.2 Dynamischer Kollisionstest für einfache Hüllkörper

Sollen *dynamische* Kollisionserkennungsergebnisse berechnet werden, die für weitergehende Simulationen, wie z.B. *Dynamiksimulation*, benutzt werden, muß auch bei der Benutzung von Hüllkörperhierarchien diesem Umstand Rechnung getragen werden.

Die berechneten Hüllkörper schließen die Flächenmenge *statisch*, d.h. in jeder Positionierung ein. Gehen wir aber zur *dynamischen* Kollisionserkennung über, so müssen entweder die Hüllkörper im Test selbst *dynamisch* behandelt werden oder das *überstrichene Volumen* der umhüllten Flächen (*swept volume*) konstruiert bzw. approximiert werden, für das dann ein *statischer* Kollisionstest durchgeführt wird.

Werden die Hüllkörper in der Kollisionserkennung *dynamisch* behandelt, so können die Boxtypen Box und Iso-Box direkt als Polyeder behandelt werden und die *dynamischen* Verfahren der Basiskollisionserkennung (vgl. Kapitel 5) im Test angewendet werden. Für die Tests, bei denen Kugeln beteiligt sind, müßten zudem noch weitere *dynamische* Verfahren entwickelt werden. Dadurch geht jedoch die besondere Effizienz der zuvor vorgestellten *statischen* Kollisionstests für einfache Hüllkörper verloren, denn es ist nicht offensichtlich, wie diese auf die effiziente *dynamische* Behandlung von Hüllkörpern erweitert werden können.

Wir gehen an dieser Stelle den zweiten Weg und stellen Verfahren vor, die das *überstrichene Volumen* der Hüllkörper approximieren.

4.2.1 Approximation des überstrichenen Volumens einfacher Hüllkörper

Approximation mit Start- und Endposition

Eine erste Möglichkeit, das *überstrichene Volumen* zu approximieren, besteht darin, die Hüllkörper in ihrer Start- und Endposition zu betrachten und die beiden entstehenden Hüllkörper durch einen neuen zu approximieren, der die beiden enthält. Dieses Verfahren wollen wir nun für alle verwendeten Hüllkörpertypen vorstellen.

Kugel

1. *Kugel*
   Seien $K_1 = (\boldsymbol{c}_1, r_1)$ und $K_2 = (\boldsymbol{c}_2, r_2)$ zwei Kugeln, so hat die kleinste einhüllende Kugel $K_{min} = (\boldsymbol{c}_{min}, r_{min})$ ihren Mittelpunkt in der Mitte zwischen den Mittelpunkten der beiden Kugeln und einen Durchmesser, der der Summe der beiden Radien und des Abstandes der beiden Mittelpunkte entspricht, d.h.

$$\boldsymbol{c}_{min} = \frac{\boldsymbol{c}_1 + \boldsymbol{c}_2}{2} \quad \text{und} \quad r_{min} = \frac{|\boldsymbol{c}_2 - \boldsymbol{c}_1| + r_1 + r_2}{2}.$$

Algorithmus *ÜberstrichenesVolumenKugelKugelStatisch*$(K, \boldsymbol{T}_{start}, \boldsymbol{T}_{end})$

	$K_1 = (\boldsymbol{c}, r)$	Kugel
Eingabe:	$\boldsymbol{T}_{start}$	Startposition der Kugel im Weltkoordinatensystem
	$\boldsymbol{T}_{end}$	Endposition der Kugel im Weltkoordinatensystem
Ausgabe:	$K_{min} = (\boldsymbol{c}_{min}, r_{min})$	einhüllende Kugel

1. $\boldsymbol{c}_{start} \leftarrow \boldsymbol{T}_{start} \cdot \boldsymbol{c}$
2. $\boldsymbol{c}_{end} \leftarrow \boldsymbol{T}_{end} \cdot \boldsymbol{c}$
3. $\boldsymbol{c}_{min} \leftarrow \frac{\boldsymbol{c}_{start} + \boldsymbol{c}_{end}}{2}$
4. $r_{min} \leftarrow \frac{|\boldsymbol{c}_{end} - \boldsymbol{c}_{start}|}{2} + r$
5. **return** $(\boldsymbol{c}_{min}, r_{min})$

Abbildung 4.6: Algorithmus: Berechnung überstrichenes Volumen Kugel durch Kugel (statisch)

Daraus ergibt sich der Algorithmus *ÜberstrichenesVolumenKugelKugelStatisch* in Abbildung 4.6.

2. *Iso-Box*
   Approximiert man zwei Kugeln $K_1 = (\boldsymbol{c}_1, r_1)$ und $K_2 = (\boldsymbol{c}_2, r_2)$ mittels einer Iso-Box $B^{Iso} = (\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$, so bilden die Mittelpunkte der beiden Kugeln zwei diagonal gegenüberliegende Eckpunkte einer Iso-Box, deren Ausdehnung in jeder Koordinatenrichtung um die Summe der beiden Radien erweitert werden muß und zwar an jedem Ende des Intervalls um den Radius der Kugel, dessen Mittelpunkt das Intervallende definiert hat. Daraus ergibt sich der Algorithmus *ÜberstrichenesVolumenKugelIsoBoxStatisch* in Abbildung 4.7.

3. *Box*
   Approximiert man zwei Kugeln $K_1 = (\boldsymbol{c}_1, r_1)$ und $K_2 = (\boldsymbol{c}_2, r_2)$ mittels einer Box $B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r})$ mit Orientierung $\boldsymbol{R}$, Mittelpunkt $\boldsymbol{c}$ und Richtungsradiusvektor $\boldsymbol{r}$, so sollte eine der Boxrichtungen dem Differenzvektor zwischen $\boldsymbol{c}_1$ und $\boldsymbol{c}_2$ entsprechen. Die Richtungen der beiden anderen Boxrichtungen haben keinen Einfluß auf das Ergebnis. Daraus kann eine Orientierung $\boldsymbol{R}$ berechnet werden. Bzgl. dieser Orientierung werden die beiden Mittelpunkte der Kugeln transformiert und eine Iso-Box für die durch die Mittelpunkte transformierten Kugeln berechnet. Der Mittelpunkt $\boldsymbol{c}$ ist gleich dem Mittelwert der beiden Mittelpunkte und der Richtungsradiusvektor $\boldsymbol{r}$ gleich dem halben Differenzvektor zwischen $\boldsymbol{b}_{max}$ und $\boldsymbol{b}_{min}$. Daraus ergibt sich der Algorithmus *ÜberstrichenesVolumenKugelBoxStatisch* in Abbildung 4.8.

Iso-Box und Box

Transformiert man eine Iso-Box $B^{Iso} = (\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$ mittels einer Transformation $\boldsymbol{T}$, so ergibt sich die Box $B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r}) = (Rotation(\boldsymbol{T}), \boldsymbol{T} \cdot (\boldsymbol{b}^{min} + \frac{1}{2} \cdot (\boldsymbol{b}^{max} - \boldsymbol{b}^{min})), \frac{1}{2} \cdot (\boldsymbol{b}^{max} -$

Algorithmus *ÜberstrichenesVolumenKugelIsoBoxStatisch*$(K, \boldsymbol{T}_{start}, \boldsymbol{T}_{end})$

Eingabe:	$K_1 = (\boldsymbol{c}, r)$	Kugel
	$\boldsymbol{T}_{start}$	Startposition der Kugel im Weltkoordinatensystem
	$\boldsymbol{T}_{end}$	Endposition der Kugel im Weltkoordinatensystem
Ausgabe:	$B^{Iso} = (\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$	einhüllende Iso-Box

$\boldsymbol{c}^{start} \leftarrow \boldsymbol{T}_{start} \cdot \boldsymbol{c}$
$\boldsymbol{c}^{end} \leftarrow \boldsymbol{T}_{end} \cdot \boldsymbol{c}$
for i **in** 1 **to** 3
do if $(\boldsymbol{c}_i^{start} < \boldsymbol{c}_i^{end})$
  then $\boldsymbol{b}_i^{min} \leftarrow \boldsymbol{c}_i^{start} - r$
    $\boldsymbol{b}_i^{max} \leftarrow \boldsymbol{c}_i^{end} + r$
  else $\boldsymbol{b}_i^{min} \leftarrow \boldsymbol{c}_i^{end} - r$
    $\boldsymbol{b}_i^{max} \leftarrow \boldsymbol{c}_i^{start} + r$
return $(\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$

Abbildung 4.7: Algorithmus: Berechnung überstrichenes Volumen Kugel durch Iso-Box (statisch)

Algorithmus *ÜberstrichenesVolumenKugelBoxStatisch*$(K, \boldsymbol{T}_{start}, \boldsymbol{T}_{end})$

Eingabe:	$K_1 = (\boldsymbol{c}, r)$	Kugel
	$\boldsymbol{T}_{start}$	Startposition der Kugel im Weltkoordinatensystem
	$\boldsymbol{T}_{end}$	Endposition der Kugel im Weltkoordinatensystem
Ausgabe:	$B = (\boldsymbol{R}_{min}, \boldsymbol{c}_{min}, \boldsymbol{r}_{min})$	einhüllende Box

$\boldsymbol{c}^{start} \leftarrow \boldsymbol{T}_{start} \cdot \boldsymbol{c}$
$\boldsymbol{c}^{end} \leftarrow \boldsymbol{T}_{end} \cdot \boldsymbol{c}$
$\boldsymbol{d} \leftarrow \frac{\boldsymbol{c}^{end} - \boldsymbol{c}^{start}}{|\boldsymbol{c}^{end} - \boldsymbol{c}^{start}|}$
$\boldsymbol{R} \leftarrow \boldsymbol{R}(\frac{\boldsymbol{e_1} \times \boldsymbol{d}}{|\boldsymbol{e_1} \times \boldsymbol{d}|}, \arccos(\boldsymbol{e_1} \cdot \boldsymbol{d}))$
$\boldsymbol{c}^{start'} \leftarrow \boldsymbol{R}^{-1} \cdot \boldsymbol{c}^{start}$
$\boldsymbol{c}^{end'} \leftarrow \boldsymbol{R}^{-1} \cdot \boldsymbol{c}^{end}$
for i **in** 1 **to** 3
do if $(\boldsymbol{c}_i^{start'} < \boldsymbol{c}_i^{end'})$
  then $\boldsymbol{b}_i^{min} \leftarrow \boldsymbol{c}_i^{start'} - r$
    $\boldsymbol{b}_i^{max} \leftarrow \boldsymbol{c}_i^{end'} + r$
  else $\boldsymbol{b}_i^{min} \leftarrow \boldsymbol{c}_i^{end'} - r$
    $\boldsymbol{b}_i^{max} \leftarrow \boldsymbol{c}_i^{start'} + r$
return $(\boldsymbol{R}, \frac{\boldsymbol{c}^{start} + \boldsymbol{c}^{end}}{2}, \frac{\boldsymbol{b}^{max} - \boldsymbol{b}^{min}}{2})$

Abbildung 4.8: Algorithmus: Berechnung überstrichenes Volumen Kugel durch Box (statisch)

$\boldsymbol{b}^{min}$)), eine Box mit Orientierung $\boldsymbol{R}$, Mittelpunkt $\boldsymbol{c}$ und Richtungsradiusvektor $\boldsymbol{r}$, so daß in der Folge Iso-Boxen und Boxen gemeinsam als Boxen betrachtet werden.

1. *Kugel*
   Sollen zwei Boxen $B_1 = (\boldsymbol{R}_1, \boldsymbol{c}_1, \boldsymbol{r}_1)$ und $B_2 = (\boldsymbol{R}_2, \boldsymbol{c}_2, \boldsymbol{r}_2)$ mit einer Kugel $K = (\boldsymbol{c}, r)$ umhüllt werden, so legen wir den Mittelpunkt auf den Mittelpunkt zwischen den Mittelpunkten der beiden Boxen und als Durchmesser wählen wir die Summe aus den Längen der Richtungsradien und dem Abstand zwischen den Mittelpunkten der Boxen, d.h.

   $$\boldsymbol{c} = \frac{\boldsymbol{c}_1 + \boldsymbol{c}_2}{2} \quad \text{und} \quad r = \frac{|\boldsymbol{c}_2 - \boldsymbol{c}_1| + |\boldsymbol{r}_1| + |\boldsymbol{r}_2|}{2}.$$

   Daraus ergibt sich der Algorithmus *ÜberstrichenesVolumenBoxKugelStatisch* in der Abbildung 4.9.

Algorithmus *ÜberstrichenesVolumenBoxKugelStatisch*$(B, \boldsymbol{T}_{start}, \boldsymbol{T}_{end})$

	$B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r})$	Box
Eingabe:	$\boldsymbol{T}_{start}$	Startposition der Box im Weltkoordinatensystem
	$\boldsymbol{T}_{end}$	Endposition der Box im Weltkoordinatensystem
Ausgabe:	$K = (\boldsymbol{c}, r)$	einhüllende Kugel

1. $\boldsymbol{c}^{start} \leftarrow \boldsymbol{T}_{start} \cdot \boldsymbol{c}$
2. $\boldsymbol{c}^{end} \leftarrow \boldsymbol{T}_{end} \cdot \boldsymbol{c}$
3. $\boldsymbol{c} \leftarrow \frac{\boldsymbol{c}^{start} + \boldsymbol{c}^{end}}{2}$
4. $r \leftarrow \frac{|\boldsymbol{c}^{end} - \boldsymbol{c}^{start}|}{2} + |\boldsymbol{r}|$
5. **return** $(\boldsymbol{c}, r)$

Abbildung 4.9: ALGORITHMUS: BERECHNUNG ÜBERSTRICHENES VOLUMEN BOX DURCH KUGEL (STATISCH)

2. *Iso-Box*
   Sollen zwei Boxen $B_1 = (\boldsymbol{R}_1, \boldsymbol{c}_1, \boldsymbol{r}_1)$ und $B_2 = (\boldsymbol{R}_2, \boldsymbol{c}_2, \boldsymbol{r}_2)$ mit einer Iso-Box $B^{Iso} = (\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$ eingeschlossen werden, so kann man entweder für die beiden Boxen die einhüllenden Kugeln $K_i = (\boldsymbol{c}_i, |\boldsymbol{r}_i|)$ $(i = 1, 2)$ betrachten und die einhüllenden Iso-Boxen für die beiden Kugeln berechnen oder man projiziert die beiden Boxen auf die Koordinatenachsen und berechnet aufgrund der Projektion die einhüllende Iso-Box. Daraus ergibt sich der Algorithmus *ÜberstrichenesVolumenBoxIsoBoxStatisch* in Abbildung 4.10.

3. *Box*
   Sollen zwei Boxen $B_1 = (\boldsymbol{R}_1, \boldsymbol{c}_1, \boldsymbol{r}_1)$ und $B_2 = (\boldsymbol{R}_2, \boldsymbol{c}_2, \boldsymbol{r}_2)$ mit Orientierungen $\boldsymbol{R}_i$, Mittelpunkten $\boldsymbol{c}_i$ und Richtungsradiusvektoren $\boldsymbol{r}_i$, $(i = 1, 2)$, durch eine Box $B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r})$ eingeschlossen werden, so stellt sich das Problem der Bestimmung

Algorithmus *ÜberstrichenesVolumenBoxIsoBoxStatisch*$(B, \boldsymbol{T}_{start}, \boldsymbol{T}_{end})$

	$B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r})$	Box
Eingabe:	$\boldsymbol{T}_{start}$	Startposition der Box im Weltkoordinatensystem
	$\boldsymbol{T}_{end}$	Endposition der Box im Weltkoordinatensystem
Ausgabe:	$B = (\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$	einhüllende Iso-Box

1. $\boldsymbol{c}^{start} \leftarrow \boldsymbol{T}_{start} \cdot \boldsymbol{c}$
2. $\boldsymbol{c}^{end} \leftarrow \boldsymbol{T}_{end} \cdot \boldsymbol{c}$
3. $\boldsymbol{R}^{start} \leftarrow Rotation(\boldsymbol{T}_{start}) \cdot \boldsymbol{R}$
4. $\boldsymbol{R}^{end} \leftarrow Rotation(\boldsymbol{T}_{end}) \cdot \boldsymbol{R}$
5. **for** i **in** 1 **to** 3
6. **do** $\boldsymbol{r}_i^{start} \leftarrow \sum_{j=1}^{3} \boldsymbol{r}_j \cdot |\boldsymbol{R}_{ij}^{start}|$
7. $\quad\boldsymbol{r}_i^{end} \leftarrow \sum_{j=1}^{3} \boldsymbol{r}_j \cdot |\boldsymbol{R}_{ij}^{end}|$
8. $\quad\boldsymbol{b}_i^{min} \leftarrow \min(\boldsymbol{c}_i^{start} - \boldsymbol{r}_i^{start}, \boldsymbol{c}_i^{end} - \boldsymbol{r}_i^{end})$
9. $\quad\boldsymbol{b}_i^{max} \leftarrow \max(\boldsymbol{c}_i^{start} + \boldsymbol{r}_i^{start}, \boldsymbol{c}_i^{end} + \boldsymbol{r}_i^{end})$
10. **return** $(\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$

Abbildung 4.10: ALGORITHMUS: BERECHNUNG ÜBERSTRICHENES VOLUMEN BOX DURCH ISO-BOX (STATISCH)

der richtigen Orientierung von B. Dazu sind zur Berechnung aus den bisherigen Orientierungen drei verschiedene Varianten geeignet. Zum ersten die Orientierung der ersten Box, zum zweiten die Orientierung der zweiten Box und zum dritten eine interpolierte Orientierung zwischen den beiden Orientierungen. Beachtet man nun, daß die betrachteten Bewegungen klein sein sollten, d.h. die Orientierungen der Boxen sich wenig unterscheiden und daß die Berechnung der umhüllenden Box während des Kollisionserkennungsprozesses bei jeder Bewegung erneut durchgeführt werden muß, so erscheint die Wahl einer der beiden Boxenorientierungen als sinnvoll. Um die einhüllende Box zu berechnen, projiziert man dann die beiden Boxen auf die zu der Orientierung gehörenden Koordinatenachsen und berechnet daraus den neuen Mittelpunkt und Richtungsradiusvektor. Daraus ergibt sich der Algorithmus *ÜberstrichenesVolumenBoxBoxStatisch* in Abbildung 4.11.

Approximation mit maximaler Bewegungslänge

Eine weitere Möglichkeit der Approximation des *überstrichenen Volumens* besteht darin, die maximale Bewegungslänge, d.h. die maximale Distanz eines bewegten Punktes von seiner Startposition während der Bewegung, nach oben abzuschätzen und mit dieser Größe die Dimension des *statischen* Hüllkörpers derart zu vergrößern, daß alle Punkte während der Bewegung innerhalb des vergrößerten Hüllkörpers enthalten sind.

Ein sinnvolle Möglichkeit der Abschätzung besteht dabei darin, die Bewegungslänge der Rotation, d.h. den Abstand des Punktes von der aktuellen Position nach Durchführung der

Algorithmus $\textit{ÜberstrichenesVolumenBoxBoxStatisch}(B, \boldsymbol{T}_{start}, \boldsymbol{T}_{end})$

	$B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r})$	Box
Eingabe:	$\boldsymbol{T}_{start}$	Startposition der Box im Weltkoordinatensystem
	$\boldsymbol{T}_{end}$	Endposition der Box im Weltkoordinatensystem

Ausgabe: $B = (\boldsymbol{R}_{min}, \boldsymbol{c}_{min}, \boldsymbol{r}_{min})$ einhüllende Box

$\boldsymbol{R}^{start} \leftarrow Rotation(\boldsymbol{T}_{start}) \cdot \boldsymbol{R}$
$\boldsymbol{R}^{end} \leftarrow \boldsymbol{R}^{min} \leftarrow Rotation(\boldsymbol{T}_{end}) \cdot \boldsymbol{R}$
$\boldsymbol{c}^{start} \leftarrow \boldsymbol{T}_{start} \cdot \boldsymbol{c}$
$\boldsymbol{c}^{end} \leftarrow \boldsymbol{T}_{end} \cdot \boldsymbol{c}$
for i **in** 1 **to** 3
do $c_i^{start} \leftarrow \boldsymbol{c}^{start^T} \cdot \boldsymbol{R}_{\cdot i}^{min}$
$\quad c_i^{end} \leftarrow \boldsymbol{c}^{end^T} \cdot \boldsymbol{R}_{\cdot i}^{min}$
$\quad r_i^{start} \leftarrow \sum_{j=1}^{3} \boldsymbol{r}_j \cdot |\boldsymbol{R}_{\cdot j}^{start^T} \cdot \boldsymbol{R}_{\cdot i}^{min}|$
$\quad r_i^{end} \leftarrow \sum_{j=1}^{3} \boldsymbol{r}_j \cdot |\boldsymbol{R}_{\cdot j}^{end^T} \cdot \boldsymbol{R}_{\cdot i}^{min}|$
$\quad \boldsymbol{b}_i^{min} \leftarrow \min(c_i^{start} - r_i^{start}, c_i^{end} - r_i^{end})$
$\quad \boldsymbol{b}_i^{max} \leftarrow \max(c_i^{start} + r_i^{start}, c_i^{end} + r_i^{end})$
$\boldsymbol{c}_{min} \leftarrow \frac{\boldsymbol{b}^{min} + \boldsymbol{b}^{max}}{2}$
$\boldsymbol{r}_{min} \leftarrow \frac{\boldsymbol{b}^{max} - \boldsymbol{b}^{min}}{2}$
return $(\boldsymbol{R}_{min}, \boldsymbol{c}_{min}, \boldsymbol{r}_{min})$

Abbildung 4.11: ALGORITHMUS: BERECHNUNG ÜBERSTRICHENES VOLUMEN BOX DURCH BOX (STATISCH)

Rotation, und die Bewegungslänge der Translation, d.h. die Länge des Translationsvektors, zu addieren und als Abschätzung für die gesamte Bewegungslänge zu benutzen. Diese ist für jeden Punkt unterschiedlich, denn die Bewegungslänge der Rotation ist abhängig von der Lage des Punktes bzgl. der Rotationsachse. Je weiter der Punkt davon entfernt ist, desto länger ist die zugehörige Bewegungslänge. Durch diese Art der Abschätzung der Bewegungslänge ist zudem sichergestellt, daß ausgehend von der Startposition des Hüllkörpers der Hüllkörper in seiner Endposition innerhalb des vergrößerten Hüllkörpers enthalten ist.

Die Abschätzung l der Bewegungslänge für einen Punkt $\boldsymbol{x}$ bei der Durchführung einer Bewegung $\boldsymbol{T} = (\boldsymbol{R}, \boldsymbol{t})$ ist:

$$l = |\boldsymbol{R} \cdot \boldsymbol{x} - \boldsymbol{x}| + |\boldsymbol{t}| \geq |\boldsymbol{R} \cdot \boldsymbol{x} + \boldsymbol{t} - \boldsymbol{x}| = |\boldsymbol{T} \cdot \boldsymbol{x} - \boldsymbol{x}|$$

Die Berechnung geht davon aus, daß während des rotativen Bewegungsanteils ein Punkt am Ende der Bewegung den größten Abstand von der Startposition hat. Bei den von uns betrachteten eher kleinen Bewegungen ist diese Voraussetzung in der Regel gegeben. Die Vorgehensweise bei der Vergrößerung der Hüllkörper wird nun für alle verwendeten Hüllkörpertypen vorgestellt.

1. *Kugel*
   Die maximale Bewegungslänge einer Kugel $K = (\boldsymbol{c}, r)$ hängt nur von der Bewegungslänge des Punktes der Kugel ab, der den maximalen Abstands von der Rotationachse $\boldsymbol{r}$ hat. Dieser maximale Abstand wird in dem Punkt

$$\boldsymbol{x}_{max} = \boldsymbol{c} + r \cdot \frac{\boldsymbol{c} - \boldsymbol{r} \cdot (\boldsymbol{r} \cdot \boldsymbol{c})}{|\boldsymbol{c} - \boldsymbol{r} \cdot (\boldsymbol{r} \cdot \boldsymbol{c})|}$$

   angenommen. Der Radius der Kugel wird um die Bewegungslänge dieses Punktes vergrößert. Daraus ergibt sich der Algorithmus *ÜberstrichenesVolumenKugelDynamisch* in Abbildung 4.12.

Algorithmus *ÜberstrichenesVolumenKugelDynamisch*$(K, \boldsymbol{T}_{start}, \boldsymbol{T}_{rel})$

	$K = (\boldsymbol{c}, r)$	Kugel
Eingabe:	$\boldsymbol{T}_{start}$	Startposition der Kugel im Weltkoordinatensystem
	$\boldsymbol{T}_{rel}$	Bewegung der Kugel im Weltkoordinatensystem
Ausgabe:	$K = (\boldsymbol{c}_{min}, r_{min})$	einhüllende Kugel

1. $\boldsymbol{c}_{start} \leftarrow \boldsymbol{T}_{start} \cdot \boldsymbol{c}$
2. $(\boldsymbol{r}, \alpha) \leftarrow Rotation(\boldsymbol{T}_{rel})$
3. $\boldsymbol{x}_{max} \leftarrow \boldsymbol{c}_{start} + r \cdot \frac{\boldsymbol{c}_{start} - \boldsymbol{r} \cdot (\boldsymbol{r} \cdot \boldsymbol{c}_{start})}{|\boldsymbol{c}_{start} - \boldsymbol{r} \cdot (\boldsymbol{r} \cdot \boldsymbol{c}_{start})|}$
4. $l \leftarrow |Rotation(\boldsymbol{T}_{rel}) \cdot \boldsymbol{x}_{max} - \boldsymbol{x}_{max}| + |Translation(\boldsymbol{T}_{rel})|$
5. **return** $((\boldsymbol{c}_{start}, r + l)$

Abbildung 4.12: ALGORITHMUS: BERECHNUNG ÜBERSTRICHENES VOLUMEN KUGEL (DYNAMISCH)

2. *Iso-Box und Box*
   Transformiert man eine Iso-Box $B^{Iso} = (\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$ mittels einer Transformation $\boldsymbol{T}$, ergibt sich die Box $B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r}) = (Rotation(\boldsymbol{T}), \boldsymbol{T} \cdot (\frac{1}{2} \cdot (\boldsymbol{b}^{max} + \boldsymbol{b}^{min})), \frac{1}{2} \cdot (\boldsymbol{b}^{max} - \boldsymbol{b}^{min}))$, eine Box mit Orientierung $\boldsymbol{R}$, Mittelpunkt $\boldsymbol{c}$ und Richtungsradiusvektor $\boldsymbol{r}$, so daß in der Folge Iso-Boxen und Boxen gemeinsam als Boxen betrachtet werden.

   Die Bewegungslänge einer Box hängt ebenfalls wie bei der Kugel nur von der Bewegungslänge des Punktes der Box ab, der den maximalen Abstand von der Rotationsachse $\boldsymbol{r}$ der Bewegung hat. Die Berechnung dieses Punktes ist bei einer Box wegen des in verschiedenen Richtungen der Box unterschiedlichen Radius schwerer durchführbar als bei der Kugel. Um die Berechnung zu beschleunigen, kann man die Box $B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r})$ als eine Kugel $K = (\boldsymbol{c}, |\boldsymbol{r}|)$ betrachten, die die Box enthält und die Berechnung der Bewegungslänge für diese Kugel durchführen. Der Radiusvektor der Box wird dann um die errechnete Bewegungslänge vergrößert. Daraus ergibt sich der Algorithmus *ÜberstrichenesVolumenBoxDynamisch* in Abbildung 4.13.

Algorithmus *ÜberstrichenesVolumenBoxDynamisch*$(B, \boldsymbol{T}_{start}, \boldsymbol{T}_{rel})$		
	$B = (\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{r})$	Box
Eingabe:	$\boldsymbol{T}_{start}$	Startposition der Box im Weltkoordinatensystem
	$\boldsymbol{T}_{rel}$	Bewegung der Box im Weltkoordinatensystem
Ausgabe:	$K = (\boldsymbol{R}_{min}, \boldsymbol{c}_{min}, \boldsymbol{r}_{min})$	einhüllende Box

1. $\boldsymbol{c}_{start} \leftarrow \boldsymbol{T}_{start} \cdot \boldsymbol{c}$
2. $(\boldsymbol{w}, \alpha) \leftarrow Rotation(\boldsymbol{T}_{rel})$
3. $\boldsymbol{x}_{max} \leftarrow \boldsymbol{c}_{start} + |\boldsymbol{r}| \cdot \frac{\boldsymbol{c}_{start} - \boldsymbol{w} \cdot (\boldsymbol{w} \cdot \boldsymbol{c}_{start})}{|\boldsymbol{c}_{start} - \boldsymbol{w} \cdot (\boldsymbol{w} \cdot \boldsymbol{c}_{start})|}$
4. $l \leftarrow |Rotation(\boldsymbol{T}_{rel}) \cdot \boldsymbol{x}_{max} - \boldsymbol{x}_{max}| + |Translation(\boldsymbol{T}_{rel})|$
5. **return** $(Rotation(\boldsymbol{T}_{start}), \boldsymbol{c}_{start}, \boldsymbol{r} + (l, l, l)^T)$

Abbildung 4.13: ALGORITHMUS: BERECHNUNG ÜBERSTRICHENES VOLUMEN BOX (DYNAMISCH)

4.3 Hierarchietraversierungsverfahren

In den Abschnitten 4.1 und 4.2 wird beschrieben, wie einfache Hüllkörper *statisch* oder *dynamisch* auf Kollisionen getestet werden können. Damit haben wir das Werkzeug, um die während der Benutzung der vorberechneten Hüllkörperhierarchien vorkommenden Hüllkörpertests effizient durchführen zu können. In diesem Abschnitt wollen wir uns mit Verfahren zur Traversierung der Hüllkörperhierarchien während des Kollisionserkennungsprozesses beschäftigen.

Als Voraussetzung haben wir nun für jedes Objekt einen Baum von Hüllkörpern als Hüllkörperhierarchie. An der Wurzel dieses Baumes befindet sich ein Hüllkörper, der das gesamte Objekt überdeckt. In der nächsten Stufe befinden sich dann mehrere Hüllkörper, die gemeinsam das gesamte Objekt überdecken. Dadurch ergibt sich für jeden Hüllkörper der nächsten Stufe eine höhere Approximationsgenauigkeit bzgl. der überdeckten Objektteile als die des einzelnen Hüllkörpers eine Stufe darüber. Jede Ebene des Baumes überdeckt dabei entweder das gesamte Objekt, falls keine höherliegenden Blätter in der Hierarchie vorhanden sind, oder den Teil des Objektes, der nicht durch höherliegende Blätter der Hierarchie überdeckt ist.

Werden nun zwei Objekte mit zugehörigen Hüllkörperhierachien auf Kollision getestet, so werden immer Paare von Knoten der beiden Hierarchiebäume gegeneinander getestet. Ergibt der Test keine Kollision, so ergibt sich keine Kollision für alle überdeckten Objektteile der beiden Objekte gegeneinander und die Untersuchung der beiden Unterbäume der Hierarchien kann an dieser Stelle beendet werden. Tritt dagegen eine Kollision auf, müssen die Unterbäume der beiden betrachteten Knoten weiter untersucht werden.

Der Vergleich zweier Hierarchien beginnt jeweils an der Wurzel. Die Hüllkörper der zugehörigen Knoten werden auf Kollision getestet. Tritt dabei eine Kollision auf, werden entweder beide Knoten expandiert und alle Paare von Kindern auf Kollision getestet, oder nur einer der beiden Knoten wird expandiert und dadurch ergeben sich die weiteren zu testenden Knotenpaare. Konzeptionell kann der Vergleich zweier Hierarchien als Aufbau eines *Vergleichsbaums* interpretiert werden. Jeder Knoten des *Vergleichsbaums* ist dabei ein Paar von Knoten der verwendeten Hierarchien, die gegeneinander getestet werden. Die Struktur des *Vergleichbaums* hängt dabei neben den beiden Hierarchien von der angewendeten *Traversierungsstrategie* für die beiden Hierarchien ab. Ein Knoten des Vergleichsbaumes *kollidiert*, wenn das zu dem Knotenpaar gehörige Hüllkörperpaar kollidiert und mindestens einer der beiden Knoten kein Blattknoten des Hierarchiebaumes ist oder, falls beide Knoten Blattknoten der Hierarchien sind, neben dem Hüllkörperpaar tatsächlich ein Flächenpaar der umhüllten Flächen kollidiert. Sonst heißt der Knoten *kollisionsfrei*. Ein Paar von Blattknoten der Hierarchien bildet dabei einen Blattknoten des *Vergleichsbaums*. Kollidieren zwei Objekte miteinander, gibt es mindestens einen Blattknoten des *Vergleichsbaums*, der kollidiert. Sind zwei Objekte dagegen kollisionsfrei, gibt es eine *Wellenfront* von *kollisionsfreien* Knoten im *Vergleichsbaum*.

Ziel eines Hierarchietraversierungsverfahrens ist es, ausgehend von den beiden vorliegenden Hierarchien einen *Vergleichsbaum* zu generieren, dessen *kollisionsfreie Wellenfront* mit einer minimalen Anzahl von generierten Knoten gefunden wird, falls die Objekte kollisionsfrei sind, oder die *kollidierenden Blattknoten* mit einer minimalen Anzahl von generierten Knoten zu finden. Dabei bedeutet die Generierung eines Knotens, daß der zugehörige Hüllkörpertest durchgeführt wird.

Für die Hierarchietraversierungsverfahren gilt dabei generell, daß sie aus einfachen Mechanismen bestehen sollten, da sie während des Kollisionserkennungsprozesses eingesetzt werden und nur strukturierenden Charakter haben.

4.3.1 Bisherige Arbeiten

In der Literatur wurden bisher die Verfahren zum Vergleich zweier gegebener Hüllkörperhierarchien zur schnellen Kollisionserkennung nur wenig beachtet. Das Hauptaugenmerk im Zusammenhang mit Hüllkörperhierarchien lag bisher in der Berechnung möglichst geeigneter Hierarchien zur Kollisionserkennung (vgl. Kapitel 3). Der Vergleich gegebener Hierarchien während des Kollisionserkennungsprozesses wurde dagegen nur am Rande erwähnt. Im wesentlichen werden zwei verschiedene Traversierungsstrategien vorgeschlagen:

1. *Simultane Expansion*
   Die am häufigsten vorgeschlagene Variante bei der Hierarchietraversierung ist die *simultane Expansion*. Kollidieren die Hüllkörper eines Knotenpaares der Hierarchien, geht man in beiden Bäumen gleichzeitig zu der Betrachtung der Kinder der beiden Knoten über. Diese Vorgehensweise wird von Hubbard in [Hu95], Felger und Zachmann in [FZ95, Za97] sowie Barequet et.al. in [BC+96] vorgeschlagen.

2. *Einseitige Expansion*
   Bei der *einseitigen Expansion* geht man nicht in beiden Hierarchien gleichzeitig auf die Kinder beider Knoten über, sondern nur in der Hierarchie, dessen Hüllkörper die schlechtere Qualität hat. Quinlain schlägt diese Vorgehensweise in [Qu94] vor.

4.3.2 Hybride Expansion

Den beiden starren Konzepten der *simultanen* und der *einseitigen Expansion* fügen wir das flexiblere Konzept der *hybriden Expansion* hinzu.

Einseitige Expansion ist dann sinnvoll, wenn die zu testenden Hüllkörper stark unterschiedliche Qualitäten haben. Wird beispielsweise ein sehr kleines Objekt gegen ein sehr großes Objekt getestet, ist es ungünstig, sofort mit der Expansion der Hierarchie des kleinen Objektes zu beginnen. Es ist erst dann sinnvoll, wenn die bei einer möglichen Kollision beteiligten Objektteile bei dem größeren Objekt auf einen Bereich eingeschränkt wurden, der der Größenordnung des kleineren Objektes entspricht. Eine direkte *simultane Expansion* würde dagegen durch die sofortige Expansion der Hierarchie des kleineren Objektes viele Hüllkörpertests für Teile des kleineren Objektes generieren, die erst in einer späteren Phase des Kollisionserkennungsprozesses sinnvoll sind und dadurch insgesamt eine wesentlich höhere Anzahl von Hüllkörpertests durchführen.

Simultane Expansion ist günstig, wenn die durch die Expansion der Hierarchien entstehende Verbesserung der Objektapproximation bei beiden Objekten in einer ähnlichen Größenordnung liegt. Denn wird durch die Expansion eines Knotens eine große Verbesserung der Approximationseigenschaften der Hüllkörper erreicht, bei dem anderen jedoch nur eine kleine, so wird durch die *simultane Expansion* eine größere Anzahl durchzuführender Hüllkörpertests erzeugt, ohne daß das Ergebnis qualitativ wesentlich besser ist als das bei einer *einseitigen Expansion*. Sind jedoch die Verbesserungen der Approximationseigenschaften der Hüllkörper von gleicher Größenordnung, wird durch die *simultane Expansion* in der nächsten Stufe ein wesentlich besseres Ergebnis berechnet als bei der *einseitigen Expansion*, was zu einer geringeren Gesamtanzahl durchzuführender Hüllkörpertests führen kann.

Wir wollen nun beide Verfahren zu einem neuen Verfahren, der *hybriden Expansion*, zusammenfassen, wobei in jeder Situation das mutmaßlich vorteilhaftere der beiden Verfahren angewendet werden soll. Dazu wird bei jedem Knoten einer Hierarchie die *worst-case* Abschätzung für die statische Güte seiner Kinder und die Kosten der Expansion gespeichert. Dabei bestehen die Kosten der Expansion in der Summe der Kosten für den Test der Hüllkörper der Kindknoten. Diese können als Vorberechnung bereitgestellt werden. Werden bei der Benutzung der Hierarchien im Kollisionserkennungsprozeß die statischen Hüllkörper getestet, (mittels *statischer* oder *dynamischer* Kollisionserkennung), entspricht die Güte der der verwendeten Hüllkörper. Wird dagegen das überstrichene Volumen konstruiert und *statisch* auf Kollision getestet, so ist die Güte der dabei entstehenden

Algorithmus *HybrideHierarchieExpansion*(K_1, K_2)
Eingabe: $K_1 = (q_1, q_1^e, c_1^e)$ Knoten 1
$K_2 = (q_2, q_2^e, c_2^e)$ Knoten 2
Ausgabe: l Menge neuer zu testender Knotenpaare

$l \leftarrow \emptyset$
$one \leftarrow c_1^e \cdot (q_1^e + q_2)$
$two \leftarrow c_2^e \cdot (q_2^e + q_1)$
$both \leftarrow c_1^e \cdot c_2^e \cdot (q_1^e + q_2^e)$
if $((one < two)$ **and** $(one < both))$
then (∗ Expandiere K_1 ∗)
  for K **in** $Kinder(K_1)$
  do $l \leftarrow l \cup \{(K, K_2)\}$
if $((two < one)$ **and** $(two < both))$
then (∗ Expandiere K_2 ∗)
  for K **in** $Kinder(K_2)$
  do $l \leftarrow l \cup \{(K_1, K)\}$
if $((both \leq one)$ **and** $(both \leq two))$
then (∗ Expandiere K_1 und K_2 ∗)
  for K **in** $Kinder(K_1)$
  do for M **in** $Kinder(K_2)$
    do $l \leftarrow l \cup \{(K, M)\}$
return l

Abbildung 4.14: ALGORITHMUS: HYBRIDE EXPANSION VON HIERARCHIEKNOTEN

Hüllkörper a priori nicht bekannt. Da jedoch die betrachteten Bewegungen in der Regel klein sind, weichen die konstruierten *überstrichenen Volumina* nur wenig von den statischen Hüllkörpern ab, so daß die Verwendung der statischen Güten der Hüllkörper eine geeignete Näherung für die tatsächliche Güte der getesteten Hüllkörper darstellt.

Das Vergleichsmaß zur Entscheidung, welche Knoten in einer gegebenen Situation expandiert werden sollen, berechnet sich damit wie folgt. Sei $K_1 = (q_1, q_1^e, c_1^e)$ und $K_2 = (q_2, q_2^e, c_2^e)$ ein kollidierendes Knotenpaar, so daß eine Expansion notwendig ist.

Expansion	erlangte Güte	Kosten	Vergleichsmaß
Expansion K_1	$q_1^e + q_2$	c_1^e	$c_1^e \cdot (q_1^e + q_2)$
Expansion K_2	$q_2^e + q_1$	c_2^e	$c_2^e \cdot (q_2^e + q_1)$
Expansion K_1 und K_2	$q_1^e + q_2^e$	$c_1^e \cdot c_2^e$	$c_1^e \cdot c_2^e \cdot (q_1^e + q_2^e)$

Wird nur K_1 expandiert erhält man als Gütestand des Kollisionserkennungsprozesses die Güte des schlechtesten Kindes von K_1 plus die Güte des nicht expandierten Knotens K_2. Die Kosten für die erreichte Güte sind die Kosten für die durchzuführenden Hüllkörpertests der Kinder von K_1 mit dem Hüllkörper von K_2. Das Produkt der beiden Faktoren ergibt das Vergleichsmaß der Expansion. Analog kann das Vergleichsmaß für die *einseitige Expansion* von K_2 und die *simultane Expansion* erklärt werden. Die Expansion mit dem geringsten Vergleichsmaß wird durchgeführt. Daraus resultiert der Algorithmus *Hybride-HierarchieExpansion* in Abbildung 4.14.

Kapitel 5

Basiskollisionserkennung

Mit Hilfe von *Raumpartitionierungsverfahren* (vgl. Kapitel 6) kann die Anzahl der zu testenden Objektpaare reduziert werden und durch die Benutzung von *Hüllkörperhierarchien* (vgl. Kapitel 3 und 4) wird die Anzahl der zu testenden Paare von Objektteilen während eines Objektpaartests reduziert. Kommt es jedoch tatsächlich zu einer Kollision zwischen einem Objektpaar oder aber kommt ein Objektpaar so dicht zusammen, daß die Approximation der Objektteile mit Hilfe von Hüllkörpern nicht ausreicht, müssen einzelne Paare von Objektteilen mit Hilfe der *Basiskollisionserkennung* getestet werden.

Aufgabe der *Basiskollisionserkennung* ist es dabei, entweder eine Kollision zwischen Objektteilen auszuschließen oder den Kollisionszeitpunkt und die dabei beteiligten Objektteile zu bestimmen. Dabei lassen sich grundsätzlich *statische* und *dynamische* Verfahren unterscheiden. Bei einem *statischen* Verfahren wird getestet, ob sich ein Paar von Objektteilen in einer vorgegebenen Konfiguration überlappt. *Dynamische* Verfahren dagegen bestimmen, ob eine Kollision während einer vorgegebenen Bewegung vorliegt, und, falls eine Kollision vorliegt, den Zeitpunkt der Kollision zwischen einem Paar von Objektteilen bzw. dessen Approximation.

Wird die Kollisionserkennung unabhängig von weitergehenden Simulationen eingesetzt, so genügt in der Regel ein *statisches* Basiskollisionserkennungsverfahren, das entscheidet, ob die Endposition einer vorgegebenen Bewegung kollisionsfrei ist. Werden die Ergebnisse der Kollisionserkennung jedoch als Eingabe für weitergehende Simulationen benutzt, wie z.B. für *interaktive Objektmanipulationen* (vgl. [Bu98, BS98, Jo97]) oder die Simulation *physikalischen* und *dynamischen* Objektverhaltens (vgl. [Sa99, SS98, Mi96]), wird die Bestimmung eines Kollisionszeitpunktes und der zu diesem Zeitpunkt kollidierenden Objektteile benötigt.

Nach der Betrachtung der Literatur im Bereich der Basiskollisionserkennung in Abschnitt 5.1 werden wir uns in Abschnitt 5.2 zunächst die Vorgehensweise der bekannten *statischen* Kollisionserkennung betrachten, die nahezu im Zusammenhang mit allen gängigen Verfahren zur schnellen Kollisionserkennung eingesetzt wird (vgl. [Za97, GLM96, FZ95, Qu94]).

Diese Verfahren sind nicht in der Lage, Kollisionszeitpunkte, die genauer sind als das Zeitraster der *statischen* Kollisionserkennung, oder die zum Kollisionszeitpunkt kollidierenden Objektteile zu bestimmen. Danach werden wir uns in Abschnitt 5.3 mit *dynamischen* Verfahren zur Bestimmung von Kollisionszeitpunkten und kollidierenden Objektteilen beschäftigen. In Abschnitt 5.4 werden wir die *statische* und die *dynamische* Kollisionserkennung zu einem *gemischt statisch-dynamischen* Verfahren kombinieren und abschließend in Abschnitt 5.5 die vorgestellten Verfahren miteinander vergleichen.

Voraussetzung für die Basiskollisionserkennung in diesem Kapitel ist, daß die Objekte in der Kollisionserkennung entsprechend der Beschreibung in Kapitel 2 Flächenmengen konvexer Flächen sind, die sich beide mittels einer rigiden Transformation bewegen. Ist ein Objekt während eines Kollisionserkennungsschrittes unbewegt, so wird dies dadurch erzielt, daß Start- und Endposition der Bewegung identisch sind.

5.1 Bisherige Arbeiten

Die Grundlagen der *statischen* Basiskollisionserkennung wurden von Cyrus und Beck [CB78] beschrieben. Zwei Flächen durchdringen sich genau dann, wenn es eine Kante der einen Fläche gibt, die die andere Fläche durchdringt. Zwei Flächenmengen durchdringen sich somit genau dann, wenn es ein Paar von Flächen gibt, das sich durchdringt. Als Ergebnis einer statischen Kollisionserkennung kann somit entschieden werden, ob sich zwei Flächenmengen durchdringen und falls dies der Fall ist, mindestens ein Paar von Flächen bzw. ein Kante – Fläche Paar spezifiziert werden, das sich durchdringt.

Die Bestimmung von Kollisionszeitpunkten bei der *dynamischen* Basiskollisionserkennung gestaltet sich dagegen weitaus schwieriger. Zwei Flächen kollidieren genau dann während einer Bewegung, wenn es ein Ecke – Fläche oder ein Kante – Kante Paar gibt, das während der Bewegung kollidiert. Zwei Flächenmengen kollidieren somit genau dann während einer Bewegung, wenn es ein Flächenpaar gibt, das während der Bewegung miteinander kollidiert. Als Ergebnis einer *dynamischen* Basiskollisionserkennung kann somit entschieden werden, ob zwei Flächenmengen während einer Bewegung miteinander kollidieren und falls dies geschieht, das Ecke – Fläche oder Kante – Kante Paar berechnet werden, das den frühesten Kollisionszeitpunkt bestimmt. Diese Art von Ergebnis wird von den oben beschriebenen weitergehenden Simulationssystemen als Eingabe für deren Berechnungen benötigt.

Schömer beschreibt in [Sch94] die exakte Bestimmung von Kollisionszeitpunkten für zwei Polyeder, von denen eines entweder rein rotatorisch oder rein translatorisch bewegt wird. Im translatorischen Fall können sich dabei auch beide Polyeder rein translatorisch bewegen, da die beiden translatorischen Bewegungen auf eine translatorische Relativbewegung für ein Polyeder zurückgeführt werden kann. Das grundlegende Verfahren ohne die Berücksichtigung polyederspezifischer Beschleunigungsverfahren wie z.B. *Elimination von Rückseiten* kann direkt auf Flächenmengen angewendet werden.

Betrachten wir die Literatur, die sich mit schnellen Kollisionserkennungsalgorithmen beschäftigt, so sehen wir, daß ein Großteil des Augenmerks auf der Entwicklung effizienter Verfahren und Datenstrukturen zur hierarchischen Approximation von Objekten mit Hüllkörpern gelegt wurde (vgl. [Za97, BC+96, GLM96, FZ95, Qu94]). Diese werden im Zusammenhang mit *statischer* Basiskollisionserkennung eingesetzt.

Hubbard verknüpft in [Hu95] die *statische* Kollisionserkennung unter Verwendung von Hüllkörperhierarchien mit 4-dimensionalen *space-time-bounds* (vgl. dazu auch die Weiterführung von Sauer in [Sa95]), um eine exaktere Kollisionserkennung zu realisieren. Voraussetzung dafür ist, daß für jeden beteiligten Körper die aktuelle Geschwindigkeit und die maximal mögliche Beschleunigung bekannt ist. Ziel ist es dabei, alle Kollisionen zu erkennen, die länger als eine vorgegebene Zeitdauer δ_{col} andauern. Der Kollisionserkennungsprozeß wird dazu in zwei Phasen unterteilt:

- **Allgemeine Phase (*broad phase*)**
  Mit Hilfe der aktuellen Bewegungsgeschwindigkeit und der maximalen Beschleunigung für einen Körper wird ausgehend von der Position des Körpers zum Zeitpunkt t_i eine 4-dimensionale Raum-Zeit-Region (*space-time-bound*) berechnet, innerhalb der sich der Körper befinden wird. Schneiden sich die Raum-Zeit-Regionen zweier Körper zu einem Zeitpunkt $s > t_i$ das erste Mal, so kann bis zu diesem Zeitpunkt eine Kollision zwischen diesen beiden Körpern ausgeschlossen werden. Nach dem Zeitpunkt s kann über die Kollisionsfreiheit in diesem Stadium keine Aussage gemacht werden. Liegt dieser Zeitpunkt t jenseits des betrachteten Zeitintervals $[t_i, t_{i+1}]$, so kann es keine Kollision zwischen den Körpern geben. Ist jedoch $t_i \leq s \leq t_{i+1}$, so ist eine Kollision innerhalb des betrachten Zeitraumes möglich. In diesem Fall wird die *allgemeine Phase* iteriert, so lange die Differenz $s_{j+1} - s_j$ zwischen zwei frühest möglichen Kollisionszeitpunkten größer ist als eine festgelegte Schranke δ_{col}. Wird diese Schranke unterschritten, so wird für den zuletzt berechneten frühestmöglichen Kollisionszeitpunkt s_j die *spezielle Phase* eingeleitet. Ergibt sich dabei keine Kollision, wird die *allgemeine Phase* zum Zeitpunkt $s_j + \delta_{col}$ erneut gestartet.

- **Spezielle Phase (*narrow phase*)**
  In dieser Phase werden zwei Körper mit Hilfe eines *sphere-trees* zu einem diskreten Zeitpunkt *statisch* auf Kollision getestet.

Ein Problem bei dieser Vorgehensweise ist, daß die Simulationsschritte während des Kollisionserkennungsprozesses unterschiedlich groß sind, je nach Kollisionsfreiheit der berechneten *space-time-bounds*. D.h. sind die betrachteten Objekte weit voneinander entfernt, ergeben sich große Simulationsschritte, sind dagegen die Körper dicht zusammen, nimmt die Anzahl der durchgeführten *statischen* Kollisionstests zu und die simulierte Zeit vergeht nur in kleinen Schritten. Darüber hinaus benutzen die *space-time-bounds* die Geschwindigkeit und die maximale Beschleunigung von Objekten in der Szene, die nicht immer gegeben sind.

Betrachten wir zusätzlich die Anforderungen von *Virtual Reality* Anwendungen, so sehen wir, daß in jedem Kollisionserkennungsschritt ein gewisses Zeitintervall simuliert werden muß. Solange die *space-time-bounds* ausreichende Schranken liefern, ergibt sich mit Hilfe der Methode von HUBBARD eine gute Performance bei der Kollisionserkennung. In diesem Fall zeigen jedoch auch *Raumpartitionierungsverfahren* (vgl. Kapitel 6) eine gute Performance. Kommen sich jedoch die betrachteten Objekte während eines Kollisionserkennungsschrittes nahe, kann durch die Methode von HUBBARD eine Vielzahl von *statischen* Kollisionstests notwendig sein, mit deren Hilfe jedoch keine Kollisionszeitpunkte bzw. die dabei kollidierenden Objektteile bestimmt werden können.

Das Verfahren von LIN [Li93] bzw. die Weiterentwicklung von CHUNG TAT LEUNG [CTL96] gehen beim Kollisionserkennungsprozeß einen anderen Weg. Das ursprünglich von LIN und CANNY [LC91] entwickelte *closest feature tracking* basiert nicht auf der Berechnung von Hüllkörperhierarchien für Objekte, sondern berechnet und verfolgt während der Bewegung der Objekte die Teile zweier Objekte, die den kleinsten Abstand zwischen den Objekten definieren. Durch das Ausnutzen der *geometrischen Stetigkeit* der Bewegungen kann der kleinste Abstand zwischen einem Objektpaar effizient (die erwartete Laufzeit hierfür ist $O(1)$) verfolgt werden. Dadurch ist das Verfahren auch in der Lage, Kollisionszeitpunkte und kollidierende Objektteile zu berechnen. MIRTICH [Mi96] benutzt diese Kollisionserkennung als Basis für seine Dynamiksimulation.

5.2 Statische Basiskollisionserkennung

Statische Basiskollisionserkennung ist in der Lage, zu einem vorgegebenen Zeitpunkt t zu erkennen, ob sich zwei Flächenmengen überlappen. Dadurch kann entschieden werden, ob die Positionierung zweier Flächenmengen kollisionsfrei ist oder nicht.

Ist eine Positionierung zweier Flächenmengen vor einer Bewegung kollisionsfrei und danach nicht, so ist während der Bewegung eine Kollision zwischen den beiden Flächenmengen aufgetreten, deren genauer Zeitpunkt und die daran beteiligte Objektteile nicht spezifiziert werden können. Anstatt dessen kann spezifiziert werden, welche Objektteile sich am Ende der Bewegung durchdringen.

Sind die betrachteten Bewegungen klein, d.h. sind die translatorischen und rotatorischen Positionsänderungen der Objekte gering, und werden die Ergebnisse der Kollisionserkennung nicht für weitergehende Simulationen, z.B. *Objektmanipulation* oder *Dynamiksimulation* verwendet, so ist es ohne großen Genauigkeitsverlust möglich, den Startzeitpunkt der Bewegung als Kollisionszeitpunkt anzunehmen und die gesamte Bewegung für nicht kollisionsfrei zu erklären. Das Ergebnis dieser Kollisionserkennung besteht zum einen aus der Entscheidung, ob eine Bewegung kollisionsfrei ist und zum anderen, falls eine Kollision vorliegt, aus der Liste der überlappenden Objektteile. Wird als Ergebnis der Kollisionserkennung nur die Entscheidung und ein Paar von überlappenden Objektteilen benötigt, so kann die Kollisionserkennung beendet werden, sobald das erste überlappende Paar von Objektteilen gefunden worden ist.

Eine Positionierung zweier Flächenmengen ist genau dann nicht kollisionsfrei, wenn es ein Paar von Flächen gibt, das sich überlappt. Zwei Flächen überlappen sich genau dann, wenn eine Kante der einen Fläche die andere Fläche durchdringt. Daher läßt sich der statische Test zweier Flächenmengen auf den Test

- Kante–Fäche

zurückführen.

Kante–Fläche

Sei f eine Fläche, $p_f : \boldsymbol{n}_f \cdot \boldsymbol{x} - d_f = 0$ die zugehörige Ebene, und $e = (\boldsymbol{w}_1, \boldsymbol{w}_2)$ eine Kante mit zugehöriger Gerade $l_e : \boldsymbol{x} = \boldsymbol{w}_1 + \lambda \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1)$. e durchdringt f genau dann, wenn die Gerade l_e die Ebene p_f durchdringt und der Schnittpunkt $\boldsymbol{s}$ innerhalb von e und f liegt.

1. $\boldsymbol{n}_f \perp (\boldsymbol{w}_2 - \boldsymbol{w}_1) \Longleftrightarrow \boldsymbol{n}_f \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1) = 0$
   Steht der Richtungsvektor der Geraden senkrecht zu dem Normalenvektor der Ebene, d.h. sind Fläche und Gerade parallel, so gibt es keinen Schnittpunkt, falls der Aufpunkt der Geraden nicht in der Ebene liegt, d.h. $\boldsymbol{n}_f \cdot \boldsymbol{w}_2 - danderen \neq 0$ und damit auch keine Kollision. Sonst liegt die Gerade in der Ebene und wir betrachten den Prismenkörper zur Fläche f in Normalenrichtung (vgl. Abbildung 5.1).

   Eine Überlappung zwischen Kante und Fläche liegt genau dann vor, wenn die Kante den Prismenkörper der Fläche schneidet oder die Kante vollständig in der Fläche liegt. Das ist genau dann der Fall, wenn der Schnittpunkt $\boldsymbol{s}$ der Geraden l_e mit der Ebene $p_{e_i} : ((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f) \cdot (\boldsymbol{x} - \boldsymbol{v}_i) = 0$, die zu der Prismenfläche f_{e_i} der Kante e_i gehört, innerhalb der beiden Kante e und e_i liegt oder einer der beiden Endpunkte der Kante e in der Fläche f liegt. Die Parametrisierung des Schnittpunktes $\boldsymbol{s}$ bzgl. l_e ergibt sich durch

   $$\lambda = \frac{((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f) \cdot (\boldsymbol{v}_i - \boldsymbol{w}_1)}{((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f) \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1)}.$$

   Sei $l_{e_i} : \boldsymbol{x} = \boldsymbol{v}_i + \delta \cdot (\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)$ die zu e_i gehörige Gerade. Dann ist die Parametrisierung des Schnittpunktes von l_e mit p_{e_i} bzgl. l_{e_i} gegeben durch

   $$\delta = \frac{(\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \cdot ((1 - \lambda) \cdot \boldsymbol{w}_1 + \lambda \cdot \boldsymbol{w}_2 - \boldsymbol{v}_i)}{|\boldsymbol{v}_{i+1} - \boldsymbol{v}_i|^2}.$$

   Für $0 \leq \lambda \leq 1$ und $0 \leq \delta \leq 1$ liegt der Schnittpunkt innerhalb der beiden Kanten und eine Überlappung liegt vor.

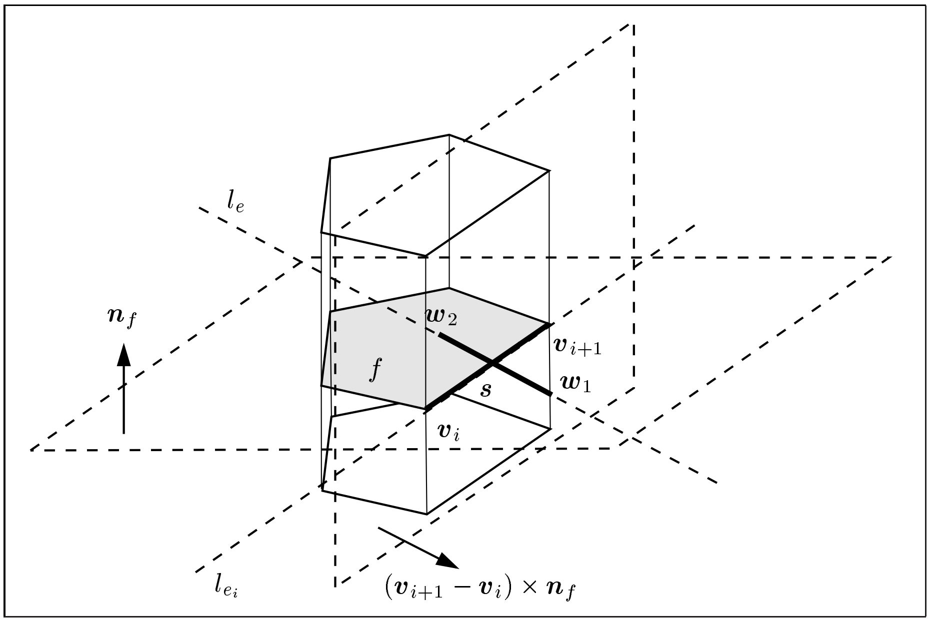

Abbildung 5.1: Statische Kollisionssituation Kante–Fläche mit Kante parallel zur Fläche

Ein Endpunkt $\boldsymbol{w}_j$ $(j = 1, 2)$ liegt genau dann in der Fläche f, wenn

$$\begin{aligned} \forall\ i = 0, \ldots, k-1 \ : \quad & \det[\boldsymbol{n}_f, \boldsymbol{w}_j - \boldsymbol{v}_i, \boldsymbol{v}_{i+1} - \boldsymbol{v}_i] \\ = \quad & ((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f) \cdot (\boldsymbol{w}_j - \boldsymbol{v}_i) \leq 0. \end{aligned}$$

2. $\boldsymbol{n}_f \not\perp (\boldsymbol{w}_2 - \boldsymbol{w}_1) \Longleftrightarrow \boldsymbol{n}_f \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1) \neq 0$
   In diesem Fall ergibt sich die Parametrisierung des Schnittpunktes der Geraden mit der Ebene durch

$$\lambda = \frac{d_f - \boldsymbol{n}_f \cdot \boldsymbol{w}_1}{\boldsymbol{n}_f \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1)}.$$

   Der Schnittpunkt ist genau dann ein Durchdringungspunkt der Kante durch die Ebene, wenn $0 \leq \lambda \leq 1$. In diesem Fall muß noch getestet werden, ob der Schnittpunkt der Kante mit der Ebene innerhalb der Fläche liegt.

   Der Schnittpunkt $\boldsymbol{s} = (1 - \lambda) \cdot \boldsymbol{w}_1 + \lambda \cdot \boldsymbol{w}_2$ liegt genau dann innerhalb von f, wenn

$$\begin{aligned} \forall\ i = 0, \ldots, k-1 \ : \quad & \det[\boldsymbol{n}_f, \boldsymbol{s} - \boldsymbol{v}_i, \boldsymbol{v}_{i+1} - \boldsymbol{v}_i] \\ = \quad & ((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f) \cdot (\boldsymbol{s} - \boldsymbol{v}_i) \leq 0. \end{aligned}$$

   Diese Vorgehensweise testet, ob der Punkt $\boldsymbol{s}$ in der Ebene auch innerhalb des von der Fläche definierten Prismenkörper in Normalenrichtung $\boldsymbol{n}_f$ liegt (vgl. Abbildung 5.2).

Daraus ergibt sich der Algorithmus *KanteFlächeStatisch* in Abbildung 5.3.

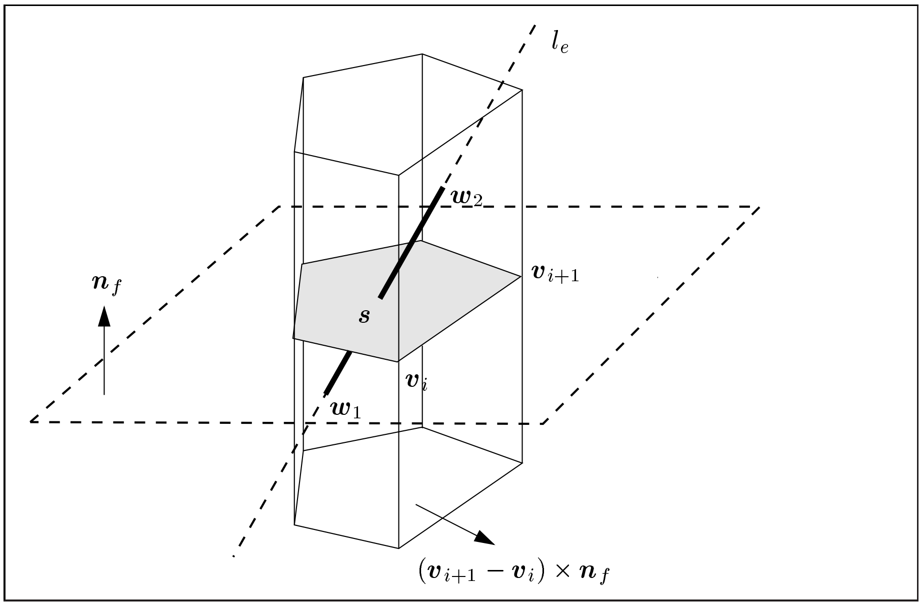

Abbildung 5.2: Statische Kollisionssituation Kante–Fläche

5.3 Dynamische Basiskollisionserkennung

Eine Kollision zwischen zwei Objekten liegt vor, wenn der minimale Abstand zwischen ihnen null ist. Dabei wird der minimale Abstand zwischen zwei Flächenmengen durch ein Paar

- Ecke–Fläche
- Kante–Kante

bestimmt. Dadurch läßt sich die dynamische Basiskollisionserkennung zwischen zwei Flächenmengen auf die Kollisionserkennung der beiden obigen Typen zurückführen.

Bewegungsannahmen

Der Kollisionszeitpunkt hängt dabei neben der Geometrie der Objekte, deren aktuellen Position und der vorgegebenen Bewegungen auch von einer Bewegungsannahme ab, die beschreibt, wie die Rotation und die Translation während der Bewegung ablaufen.

Dazu sind verschiedene Ablaufschemata möglich:

1. Translation und Rotation laufen gleichverteilt über den Bewegungszeitraum ab.

Algorithmus *KanteFlächeStatisch*(e, f)

	$e = (\boldsymbol{w}_1, \boldsymbol{w}_2)$	Kante
Eingabe:	$f = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{k-1}, \boldsymbol{v}_k \equiv \boldsymbol{v}_0)$	Fläche mit Ebene $\boldsymbol{n}_f \cdot \boldsymbol{x} - d_f = 0$
Ausgabe:	c	Kollisionsflag

if $((\boldsymbol{n}_f \cdot \boldsymbol{w}_1 - d_f = 0)$ **and** $(\boldsymbol{n}_f \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1) = 0))$
then (∗ Kante und Fläche sind parallel und liegen in einer Ebene ∗)
    inside ←TRUE
    for i **in** 0 **to** $k-1$
    do $\lambda \leftarrow \frac{((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f) \cdot (\boldsymbol{v}_i - \boldsymbol{w}_1)}{((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f) \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1)}$
        if $(0 \leq \lambda \leq 1)$
        then $\delta \leftarrow \frac{(\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \cdot ((1-\lambda) \cdot \boldsymbol{w}_1 + \lambda \cdot \boldsymbol{w}_2 - \boldsymbol{v}_i)}{|\boldsymbol{v}_{i+1} - \boldsymbol{v}_i|^2}$
            if $(0 \leq \delta \leq 1)$
            then return COLLISION
        if (*inside* = TRUE)
        then *inside* $\leftarrow ((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f) \cdot (\boldsymbol{w}_1 - \boldsymbol{v}_i) \leq 0$
    if (*inside* = TRUE)
    then return COLLISION
else (∗ Kante und Fläche nicht parallel ∗)
    $\lambda \leftarrow \frac{d_f - \boldsymbol{n}_f \cdot \boldsymbol{w}_1}{\boldsymbol{n}_f \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1)}$
    if $(0 \leq \lambda \leq 1)$
    then $\boldsymbol{s} \leftarrow (1 - \lambda) \cdot \boldsymbol{w}_1 + \lambda \cdot \boldsymbol{w}_2$
        inside ←TRUE
        $i \leftarrow 0$
        while $((inside = TRUE)$ **and** $(i < k))$
          do *inside* $\leftarrow ((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n}_f) \cdot (\boldsymbol{s} - \boldsymbol{v}_i) \leq 0$
        if (*inside* = TRUE)
        then return COLLISION
return NO COLLISION

Abbildung 5.3: ALGORITHMUS: TEST KANTE GEGEN FLÄCHE (STATISCH)

2. Verteilung von Rotation und Translation über den Bewegungszeitraum werden durch externe Simulationen bestimmt, z.B. Dynamiksimulation [Sa99, SS98].

Die Position eines Körpers wird beschrieben durch eine Rotation $\boldsymbol{R}$ und eine Translation $\boldsymbol{s}$, wobei sich die aktuelle Position $\boldsymbol{x}^{akt}$ eines Punktes $\boldsymbol{x}$ ergibt durch die Formel

$$\boldsymbol{x}^{akt} = \boldsymbol{R} \cdot \boldsymbol{x} + \boldsymbol{s}.$$

Eine gleichverteilte Bewegung über dem Bewegungszeitraum $[0, 1]$ bedeutet nun, daß sowohl die Rotation als auch die Translation linear interpoliert werden. Für die Translation

$\boldsymbol{s}(t)$ ergibt sich dabei die einfache Beziehung

$$\boldsymbol{s}(t) = \boldsymbol{s}(0) + t \cdot (\boldsymbol{s}(1) - \boldsymbol{s}(0)).$$

Die gleichförmige Interpolation von Orientierungen $\boldsymbol{R}(t)$ gestaltet sich dagegen schwieriger. Betrachten wir die verschiedenen gängigen Repräsentationen für Rotationen (vgl. Anhang A), so erkennen wir, daß eine einfache Interpolation von *Rotationsmatrizen*, *Winkeldarstellungen* sowie *Achse-Winkel Darstellungen* nicht möglich ist.

Shoemake beschreibt in [Sh85] eine von Davis vorgeschlagene Formel zur Interpolation von Orientierungen, die als *Einheitsquaternionen* beschrieben sind, so daß die Winkelgeschwindigkeit während der Bewegung konstant bleibt. Dies entspricht der Interpolation der beiden Einheitsquaternionen $\mathsf{q}(0)$ und $\mathsf{q}(1)$ auf der 4-dimensionalen Einheitskugel über die Geodätische.

$$\mathsf{q}(t) = \frac{\sin\ (1-t)\cdot\theta}{\sin\theta}\mathsf{q}(0) + \frac{\sin\ t\cdot\theta}{\sin\theta}\mathsf{q}(1),$$

wobei $\theta = \arccos(\mathsf{q}(0)\cdot\mathsf{q}(1))$ ist. Mit dieser Vorgehensweise ergibt sich der Punkt $\boldsymbol{x}$ zum Zeitpunkt t durch

$$\boldsymbol{x}(t) = \boldsymbol{R}(\mathsf{q}(t))\cdot\boldsymbol{x} + \boldsymbol{s}(t).$$

Alternativ dazu können wir auch die einfacherere lineare Interpolation der beiden Einheitsquaternionen $\mathsf{q}(0)$ und $\mathsf{q}(1)$ durch die 4-dimensionale Einheitskugel benutzen und die resultierenden Quaternionen auf die Einheitskugel projizieren:

$$\mathsf{q}(t) = \frac{\mathsf{q}(0) + t\cdot(\mathsf{q}(1) - \mathsf{q}(0))}{|\mathsf{q}(0) + t\cdot(\mathsf{q}(1) - \mathsf{q}(0))|}$$

Dadurch ergibt sich eine Rotationsinterpolation mit nichtkonstanter Winkelgeschwindigkeit, da zur Mitte des Intervalls die Winkelgeschwindigkeit ansteigt. Werden jedoch nur Bewegungen mit kleinen Winkeländerungen betrachtet, so ist diese Beschleunigung zur Bewegungsmitte hin nahezu vernachlässigbar.

Die in der Folge vorgestellten Verfahren zur approximativen Bestimmung des frühesten Kollisionszeitpunktes sind unabhängig von der gewählten Bewegungsannahme und arbeiten daher insbesondere auch mit von externen Simulationen berechneten Bewegungsüberlagerungen zusammen. Voraussetzung ist nur, daß es eine Möglichkeit gibt, für einen beliebigen Zeitpunkt $t \in [0,1]$ die aktuelle Position des Objektes zu berechnen. Weitere Einzelheiten dazu werden in den Arbeiten von Sauer im Bereich der Dynamiksimulation (vgl. [Sa99, SS98]) beschrieben. Die hier vorgestellte Kollisionserkennung stellt hierfür einen möglichst flexiblen Rahmen zur Verfügung.

Die beiden möglichen Kollisionssituationen wollen wir nun genauer betrachten und daraus unsere Vorgehensweise bei deren Erkennung ableiten.

Ecke–Fläche

Eine Kollision zwischen einer Ecke $\boldsymbol{v}$ und einer konvexen Fläche f zu einem Zeitpunkt t liegt genau dann vor, wenn die Ecke $\boldsymbol{v}(t)$ in der zu f gehörigen Ebene $p_f(t) : \boldsymbol{n}_f(t) \cdot \boldsymbol{x} - d_f(t) = 0$ liegt und sich dabei gleichzeitig innerhalb von f befindet. Aus dieser Beschreibung läßt sich direkt eine Vorgehensweise zur Kollisionserkennung für den *Ecke–Fläche* Fall ableiten.

Zum einen benötigen wir ein Verfahren, das uns erlaubt festzustellen, wann ein Eckpunkt $\boldsymbol{v}(t)$ in der zu der Fläche $f(t)$ gehörigen Ebene $p_f(t)$ liegt, und somit in der Lage ist, einen möglichen Kollisionszeitpunkt zu bestimmen. Damit werden wir uns in Abschnitt 5.3.1 beschäftigen. Zum anderen ist ein Verfahren nötig, das für einen möglichen Kollisionszeitpunkt testet, ob tatsächlich eine Kollision vorliegt. Dabei soll der Test möglicher Kollisionszeitpunkte robust in der Hinsicht sein, daß er nicht notwendigerweise verlangt, daß die Ecke $\boldsymbol{v}(t)$ exakt in der Ebene $p_f(t)$ liegt. Dazu stellen wir nun einen *exakten* und einen *approximativen* Test vor:

1. *Exakter* Test
   Liegt $\boldsymbol{v}(t)$ ungefähr in der Ebene $p_f(t)$, so befindet sich $\boldsymbol{v}(t)$ genau dann innerhalb von $f(t)$, wenn

   $$\begin{aligned} \forall\ i = 0, \ldots, k-1\ : s_i(t) &= \det[\boldsymbol{n}_f(t), \boldsymbol{v}(t) - \boldsymbol{v}_i(t), \boldsymbol{v}_{i+1}(t) - \boldsymbol{v}_i(t)] \\ &= ((\boldsymbol{v}_{i+1}(t) - \boldsymbol{v}_i(t)) \times \boldsymbol{n}_f(t)) \cdot (\boldsymbol{v}(t) - \boldsymbol{v}_i(t)) \leq 0. \end{aligned}$$

   Diese Vorgehensweise testet, ob der Eckpunkt $\boldsymbol{v}(t)$ innerhalb des von der Fläche $f(t)$ definierten Prismenkörpers in Normalenrichtung $\boldsymbol{n}_f(t)$ liegt (vgl. Abbildung 5.4).

   Daraus ergibt sich der Algorithmus *EckeInnerhalbFlächeExakt* in Abbildung 5.5.

2. *Approximativer* Test
   Problem des *exakten* Kollisionstests ist es, daß für jeden möglichen Kollisionszeitpunkt t die Position des Eckpunkts $\boldsymbol{v}(t)$ und der Fläche $f(t)$ neu bestimmt werden muß, um die *Innen-Außen-Relation* zu berechnen. Dies kann dadurch beschleunigt werden, daß für die Zeitpunkte 0 und 1 die *Innen-Außen Relationen* berechnet wird, wobei wiederverwertbare, vorberechenbare Werte verwendet werden können, und diese linear interpoliert werden. Dadurch ergibt sich folgende Vorgehensweise:

   $$\begin{aligned} \forall\ i = 0, \ldots, k-1\ : s_i(0) &= ((\boldsymbol{v}_{i+1}(0) - \boldsymbol{v}_i(0)) \times \boldsymbol{n}_f(0)) \cdot (\boldsymbol{v}(0) - \boldsymbol{v}_i(0)) \\ s_i(1) &= ((\boldsymbol{v}_{i+1}(1) - \boldsymbol{v}_i(1)) \times \boldsymbol{n}_f(1)) \cdot (\boldsymbol{v}(1) - \boldsymbol{v}_i(1)) \\ s_i(t) &= s_i(0) + t \cdot (s_i(1) - s_i(0)) \end{aligned}$$

   $\boldsymbol{v}(t)$ liegt approximativ innerhalb von $f(t)$, falls

   $$\forall\ i = 0, \ldots, k-1\ : s_i(t) \leq 0.$$

   Daraus ergibt sich der Algorithmus *EckeInnerhalbFlächeApproximativ* in Abbildung 5.6.

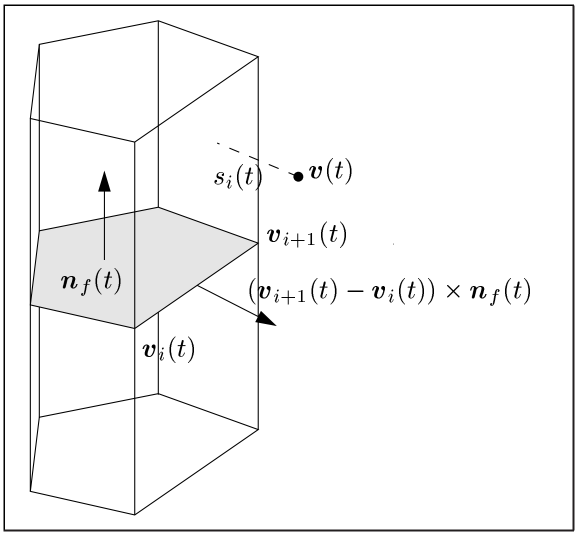

Abbildung 5.4: Prismenkörper zu einer konvexen Fläche in Normalenrichtung

Algorithmus *EckeInnerhalbFlächeExakt*$(\boldsymbol{v}(t), f(t))$

Eingabe:	$\boldsymbol{v}(t)$	Ecke
	$f(t) = (\boldsymbol{v}_0(t), \ldots, \boldsymbol{v}_k(t) \equiv \boldsymbol{v}_0(t))$	Fläche mit Ebene $\boldsymbol{n}_f(t) \cdot \boldsymbol{x} - d_f(t) = 0$

Ausgabe: c Anzeigeflag, ob Ecke innerhalb Fläche

1. *inside* ←TRUE
2. i ←0
3. **while** $((inside = TRUE)$ **and** $(i < k))$
4. **do** *inside* ←$((\boldsymbol{v}_{i+1}(t) - \boldsymbol{v}_i(t)) \times \boldsymbol{n}_f(t)) \cdot (\boldsymbol{v}(t) - \boldsymbol{v}_i(t)) \leq 0$
5. **if** (*inside* =TRUE)
6. **then return** INSIDE
7. **else return** OUTSIDE

Abbildung 5.5: Algorithmus: Test Ecke innerhalb Fläche (exakt)

Kante–Kante

Seien $e_1 = (\boldsymbol{v}_1, \boldsymbol{v}_2)$ und $e_2 = (\boldsymbol{w}_1, \boldsymbol{w}_2)$ zwei Kanten sowie $l_1 = \boldsymbol{v}_1 + \lambda_1 \cdot (\boldsymbol{v}_2 - \boldsymbol{v}_1)$ und $l_2 = \boldsymbol{w}_1 + \lambda_2 \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1)$ die zugehörigen Geraden. $e_1(t)$ und $e_2(t)$ kollidieren genau dann miteinander, wenn die beiden Geraden $l_1(t)$ und $l_2(t)$ miteinander kollidieren und der Schnittpunkt $\boldsymbol{z}$ innerhalb von e_1 und e_2 liegt, d.h. $\boldsymbol{z} = \boldsymbol{v}_1(t) + \lambda_1(\boldsymbol{z}) \cdot (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) = \boldsymbol{w}_1(t) + \lambda_2(\boldsymbol{z}) \cdot (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))$ mit $0 \leq \lambda_1(\boldsymbol{z}) \leq 1$ und $0 \leq \lambda_2(\boldsymbol{z}) \leq 1$. Auch hier ergibt sich analog zum *Ecke–Fläche* Test eine zweigeteilte Vorgehensweise beim *Kante–Kante* Kollisionstest.

Algorithmus *EckeInnerhalbFlächeApproximativ*(v, f, t)

Eingabe:	$\boldsymbol{v}$	Ecke
	$f = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_k \equiv \boldsymbol{v}_0)$	Fläche mit Ebene $\boldsymbol{n}_f \cdot \boldsymbol{x} - d_f = 0$
	t	Zeitpunkt

Ausgabe: c Anzeigeflag, ob Ecke innerhalb Fläche

1. $inside \leftarrow$TRUE
2. $i \leftarrow 0$
3. **while** $((inside = TRUE)$ **and** $(i < k))$
4. **do** $s(0) \leftarrow ((\boldsymbol{v}_{i+1}(0) - \boldsymbol{v}_i(0)) \times \boldsymbol{n}_f(0)) \cdot (\boldsymbol{v}(0) - \boldsymbol{v}_i(0))$
5. $s(1) \leftarrow ((\boldsymbol{v}_{i+1}(1) - \boldsymbol{v}_i(1)) \times \boldsymbol{n}_f(1)) \cdot (\boldsymbol{v}(1) - \boldsymbol{v}_i(1))$
6. $inside \leftarrow s(0) + t \cdot (s(1) - s(0)) \leq 0$
7. **if** $(inside =$TRUE)
8. **then return** INSIDE
9. **else return** OUTSIDE

Abbildung 5.6: ALGORITHMUS: TEST ECKE INNERHALB FLÄCHE (APPROXIMATIV)

Im ersten Schritt werden die Zeitpunkte bestimmt, zu denen die beiden Geraden $l_1(t)$ und $l_2(t)$ kollidieren. Damit beschäftigen wir uns in Abschnitt 5.3.1. Im zweiten Schritt wird dann getestet, ob die Kollision der beiden Geraden $l_1(t)$ und $l_2(t)$ innerhalb der zugehörigen Kanten $e_1(t)$ und $e_2(t)$ stattfindet. Dabei soll der Test analog zum *Ecke–Fläche* Test robust sein in der Hinsicht, daß er nicht notwendigerweise verlangt, daß die beiden Geraden $l_1(t)$ und $l_2(t)$ exakt kollidieren. Dazu stellen wir wieder einen *exakten* und einen *approximativen* Test vor:

1. *Exakter* Test
   Um zu berechnen, ob sich die beiden Kanten $e_1(t)$ und $e_2(t)$ schneiden, berechnen wir den Abstand zwischen den beiden Geraden $l_1(t)$ und $l_2(t)$ sowie die Parametrisierung der Lotfußpunkte der kürzesten Verbindungsstrecke zwischen $l_1(t)$ und $l_2(t)$ bzgl. $l_1(t)$ und $l_2(t)$ und betrachten, ob diese innerhalb von $e_1(t)$ und $e_2(t)$ liegen. Dabei muß unterschieden werden, ob die beiden Geraden $l_1(t)$ und $l_2(t)$ parallel sind oder nicht.

   (a) $l_1(t) \parallel l_2(t) \Longleftrightarrow (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) \times (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t)) = \boldsymbol{0}$

   Sind die Geraden parallel, so läßt sich der Abstand anhand eines Punktes berechnen:

$$\delta = \frac{|(\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t)) \times (\boldsymbol{v}_1(t) - \boldsymbol{w}_1(t))|}{|(\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))|}$$

   Haben die Geraden den Abstand 0, so muß getestet werden, ob sich die beiden Kanten überlappen oder nicht, was mit Hilfe des Tests dreier der vier Endpunkte der beiden Kanten berechnet werden kann. Liegt einer innerhalb der anderen Kante, so liegt eine Kollision vor. Dazu wird zur numerisch robusten Berechnung die Parametrisierung des Lotfußpunktes des betrachteten Punktes bzgl. der Gerade berechnet:

$$\begin{aligned}\lambda_1 &= \frac{(\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))^T \cdot (\boldsymbol{v}_1(t) - \boldsymbol{w}_1(t))}{|(\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))|^2} \\ \lambda_2 &= \frac{(\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))^T \cdot (\boldsymbol{v}_2(t) - \boldsymbol{w}_1(t))}{|(\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))|^2} \\ \lambda_3 &= \frac{(\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t))^T \cdot (\boldsymbol{w}_1(t) - \boldsymbol{v}_1(t))}{|(\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t))|^2}\end{aligned}$$

Gilt für mindestens ein λ_i $(i = 1,2,3): \; 0 \leq \lambda_i \leq 1$, so liegt eine Kollision zwischen den beiden Kanten vor, sonst nicht.

(b) $l_1(t) \not\parallel l_2(t) \Longleftrightarrow (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) \times (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t)) \neq \boldsymbol{0}$

Der Vektor $\boldsymbol{n}(t) = (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) \times (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))$ ist die Richtung der kürzesten Verbindungsstrecke zwischen $l_1(t)$ und $l_2(t)$ und der Abstand sowie die Parametrisierungen der Lotfußpunkte ergeben sich als Lösung des Gleichungssystems

$$\boldsymbol{v}_1(t) + \lambda_1 \cdot (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) + \delta \cdot \boldsymbol{n}(t) = \boldsymbol{w}_1(t) + \lambda_2 \cdot (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t)),$$

wobei

$$\begin{aligned}\delta &= \frac{\boldsymbol{n}(t)^T \cdot (\boldsymbol{w}_1(t) - \boldsymbol{v}_1(t))}{|\boldsymbol{n}(t)|}, \\ \lambda_1 &= \frac{(\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))^T \cdot (\boldsymbol{n}(t) \times (\boldsymbol{w}_1(t) - \boldsymbol{v}_1(t)))}{|\boldsymbol{n}(t)|^2}, \\ \lambda_2 &= \frac{(\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t))^T \cdot (\boldsymbol{n}(t) \times (\boldsymbol{w}_1(t) - \boldsymbol{v}_1(t)))}{|\boldsymbol{n}(t)|^2}\end{aligned}$$

ist. Liegen die beiden Lotfußpunkte innerhalb der Kanten, d.h. $0 \leq \lambda_i \leq 1$ $(i = 1,2)$, so kollidieren die beiden Kanten. Die zugehörige geometrische Anschauung kann Abbildung 5.7 entnommen werden.

Dadurch ergibt sich der Algorithmus *KanteInnerhalbKanteExakt* in Abbildung 5.8.

2. *Approximativer* Test
Beim *approximativen* Test wird analog zum Vorgehen beim *Ecke–Fläche* Test zwischen der Start- und der Endkonfiguration interpoliert. Für die Start- und Endkonfiguration gilt wieder, daß sie aus während eines gesamten Kollisionserkennungsschrittes wiederverwertbaren, vorberechenbaren Werten bestimmt werden können. Eine Interpolation der Werte ist jedoch nur dann sinnvoll, wenn weder zum Start- noch zum Endzeitpunkt die beiden Geraden $l_1(t)$ und $l_2(t)$ parallel sind. Daraus ergibt sich folgende Vorgehensweise:

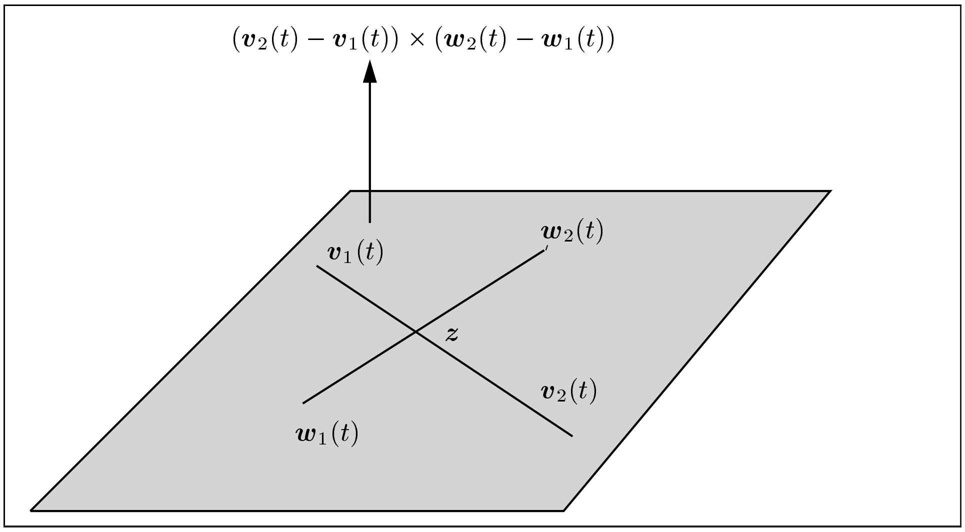

Abbildung 5.7: Kante–Kante Kollisionssituation

(a) $l_1(0) \parallel l_2(0) \quad \vee \quad l_1(1) \parallel l_2(1)$
$\Longleftrightarrow$
$(\boldsymbol{v}_2(0)-\boldsymbol{v}_1(0))\times(\boldsymbol{w}_2(0)-\boldsymbol{w}_1(0)) = \boldsymbol{0} \quad \vee \quad (\boldsymbol{v}_2(1)-\boldsymbol{v}_1(1))\times(\boldsymbol{w}_2(1)-\boldsymbol{w}_1(1)) = \boldsymbol{0}$

Führe den *exakten* Test durch.

(b) $l_1(0) \nparallel l_2(0) \quad \wedge \quad l_1(1) \nparallel l_2(1)$
$\Longleftrightarrow$
$(\boldsymbol{v}_2(0)-\boldsymbol{v}_1(0))\times(\boldsymbol{w}_2(0)-\boldsymbol{w}_1(0)) \neq \boldsymbol{0} \quad \wedge \quad (\boldsymbol{v}_2(1)-\boldsymbol{v}_1(1))\times(\boldsymbol{w}_2(1)-\boldsymbol{w}_1(1)) \neq \boldsymbol{0}$

Die Vektoren $\boldsymbol{n}(0) = (\boldsymbol{v}_2(0) - \boldsymbol{v}_1(0)) \times (\boldsymbol{w}_2(0) - \boldsymbol{w}_1(0))$ sowie $\boldsymbol{n}(1) = (\boldsymbol{v}_2(1) - \boldsymbol{v}_1(1))\times(\boldsymbol{w}_2(1)-\boldsymbol{w}_1(1))$ sind die Richtungen der kürzesten Verbindungsstrecken zwischen l_1 und l_2 zu den Zeitpunkten 0 und 1. Der Abstand sowie die Parametrisierungen der Lotfußpunkte ergeben sich als Lösung des Gleichungssystems

$$\boldsymbol{v}_1(x) + \lambda_1(x) \cdot (\boldsymbol{v}_2(x) - \boldsymbol{v}_1(x)) + \delta(x) \cdot \boldsymbol{n}(x) = \boldsymbol{w}_1 + \lambda_2(x) \cdot (\boldsymbol{w}_2(x) - \boldsymbol{w}_1(x))$$

für $x = 0$ und $x = 1$, wobei

$$\begin{aligned}
\delta(x) &= \frac{\boldsymbol{n}(x)^T \cdot (\boldsymbol{w}_1(x) - \boldsymbol{v}_1(x))}{|\boldsymbol{n}(x)|}, \\
\lambda_1(x) &= \frac{(\boldsymbol{w}_2(x) - \boldsymbol{w}_1(x))^T \cdot (\boldsymbol{n}(x) \times (\boldsymbol{w}_1(x) - \boldsymbol{v}_1(x)))}{|\boldsymbol{n}(x)|^2}, \\
\lambda_2(x) &= \frac{(\boldsymbol{v}_2(x) - \boldsymbol{v}_1(x))^T \cdot (\boldsymbol{n}(x) \times (\boldsymbol{w}_1(x) - \boldsymbol{v}_1(x)))}{|\boldsymbol{n}(x)|^2}
\end{aligned}$$

Algorithmus *KanteInnerhalbKanteExakt*$(e_1(t), e_2(t))$

Eingabe: $e_1(t) = (\boldsymbol{v}_1(t), \boldsymbol{v}_2(t))$ Kante 1
$e_2(t) = (\boldsymbol{w}_1(t), \boldsymbol{w}_2(t))$ Kante 2

Ausgabe: c Anzeigeflag, ob Kante innerhalb Kante

$\boldsymbol{n}(t) \leftarrow (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) \times (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))$
if $(|\boldsymbol{n}(t)| = 0)$
then $(*$ Kanten sind parallel $*)$
  $\delta(t) \leftarrow \frac{|(\boldsymbol{w}_2(t)-\boldsymbol{w}_1(t))\times(\boldsymbol{v}_1(t)-\boldsymbol{w}_1(t))|}{|\boldsymbol{w}_2(t)-\boldsymbol{w}_1(t)|}$
  if $(\delta(t) = 0)$
  then $\lambda(t) \leftarrow \frac{(\boldsymbol{w}_2(t)-\boldsymbol{w}_1(t))^T\cdot(\boldsymbol{v}_1(t)-\boldsymbol{w}_1(t))}{|\boldsymbol{w}_2(t)-\boldsymbol{w}_1(t)|^2}$
    if $(0 \leq \lambda(t) \leq 1)$
    then return INSIDE
    else $\lambda(t) \leftarrow \frac{(\boldsymbol{w}_2(t)-\boldsymbol{w}_1(t))^T\cdot(\boldsymbol{v}_2(t)-\boldsymbol{w}_1(t))}{|\boldsymbol{w}_2(t)-\boldsymbol{w}_1(t)|^2}$
      if $(0 \leq \lambda(t) \leq 1)$
      then return INSIDE
      else $\lambda(t) \leftarrow \frac{(\boldsymbol{v}_2(t)-\boldsymbol{v}_1(t))^T\cdot(\boldsymbol{w}_1(t)-\boldsymbol{v}_1(t))}{|\boldsymbol{v}_2(t)-\boldsymbol{v}_1(t)|^2}$
        if $(0 \leq \lambda(t) \leq 1)$
        then return INSIDE
        else return OUTSIDE
  else return OUTSIDE
else $(*$ Kanten sind nicht parallel $*)$
  $\delta(t) \leftarrow \frac{\boldsymbol{n}(t)^T\cdot(\boldsymbol{w}_1(t)-\boldsymbol{v}_1(t))}{|\boldsymbol{n}(t)|}$
  if $(\delta(t) = 0)$
  then $\lambda_1(t) \leftarrow \frac{(\boldsymbol{w}_2(t)-\boldsymbol{w}_1(t))^T\cdot(\boldsymbol{n}(t)\times(\boldsymbol{w}_1(t)-\boldsymbol{v}_1(t)))}{|\boldsymbol{n}(t)|^2}$
    $\lambda_2(t) \leftarrow \frac{(\boldsymbol{v}_2(t)-\boldsymbol{v}_1(t))^T\cdot(\boldsymbol{n}(t)\times(\boldsymbol{w}_1(t)-\boldsymbol{v}_1(t)))}{|\boldsymbol{n}(t)|^2}$
    if $(0 \leq \lambda_1(t) \leq 1)$ **and** $(0 \leq \lambda_2(t) \leq 1)$
    then return INSIDE
    else return OUTSIDE
  else return OUTSIDE

Abbildung 5.8: ALGORITHMUS: TEST KANTE INNERHALB KANTE (EXAKT)

ist. $\lambda_1(t)$ und $\lambda_2(t)$ ergeben sich durch lineare Interpolation zu

$$\begin{aligned}\lambda_1(t) &= \lambda_1(0) + t\cdot(\lambda_1(1) - \lambda_1(0)),\\ \lambda_2(t) &= \lambda_2(0) + t\cdot(\lambda_2(1) - \lambda_2(0)).\end{aligned}$$

Daraus resultiert der Algorithmus *KanteInnerhalbKanteApproximativ* in Abbildung 5.9.

Algorithmus *KanteInnerhalbKanteApproximativ*$(e_1, e_2, t))$

Eingabe:
$e_1 = (\boldsymbol{v}_1, \boldsymbol{v}_2)$ Kante 1
$e_2 = (\boldsymbol{w}_1, \boldsymbol{w}_2)$ Kante 2
t Zeitpunkt

Ausgabe: c Anzeigeflag, ob Kante innerhalb Kante

1. $\boldsymbol{n}(0) \leftarrow (\boldsymbol{v}_2(0) - \boldsymbol{v}_1(0)) \times (\boldsymbol{w}_2(0) - \boldsymbol{w}_1(0))$
2. **if** $(|\boldsymbol{n}(0)| = 0)$
3. **then return** *KanteInnerhalbKanteExakt*$(e_1(t), e_2(t))$
4. **else** $\boldsymbol{n}(1) \leftarrow (\boldsymbol{v}_2(1) - \boldsymbol{v}_1(1)) \times (\boldsymbol{w}_2(1) - \boldsymbol{w}_1(1))$
5. **if** $(|\boldsymbol{n}(1)| = 0)$
6. **then return** *KanteInnerhalbKanteExakt*$(e_1(t), e_2(t))$
7. $\lambda_1(0) \leftarrow \frac{(\boldsymbol{w}_2(0)-\boldsymbol{w}_1(0))^T \cdot (\boldsymbol{n}(0) \times (\boldsymbol{w}_1(0)-\boldsymbol{v}_1(0)))}{|\boldsymbol{n}(0)|^2}$
8. $\lambda_2(0) \leftarrow \frac{(\boldsymbol{v}_2(0)-\boldsymbol{v}_1(0))^T \cdot (\boldsymbol{n}(0) \times (\boldsymbol{w}_1(0)-\boldsymbol{v}_1(0)))}{|\boldsymbol{n}(0)|^2}$
9. $\lambda_1(1) \leftarrow \frac{(\boldsymbol{w}_2(1)-\boldsymbol{w}_1(1))^T \cdot (\boldsymbol{n}(1) \times (\boldsymbol{w}_1(1)-\boldsymbol{v}_1(1)))}{|\boldsymbol{n}(1)|^2}$
10. $\lambda_2(1) \leftarrow \frac{(\boldsymbol{v}_2(1)-\boldsymbol{v}_1(1))^T \cdot (\boldsymbol{n}(1) \times (\boldsymbol{w}_1(1)-\boldsymbol{v}_1(1)))}{|\boldsymbol{n}(1)|^2}$
11. $\lambda_1(t) \leftarrow \lambda_1(0) + t \cdot (\lambda_1(1) - \lambda_1(0))$
12. $\lambda_2(t) \leftarrow \lambda_2(0) + t \cdot (\lambda_2(1) - \lambda_2(0))$
13. **if** $(0 \leq \lambda_1(t) \leq 1)$ **and** $(0 \leq \lambda_2(t) \leq 1)$
14. **then return** INSIDE
15. **else return** OUTSIDE

Abbildung 5.9: ALGORITHMUS: TEST KANTE INNERHALB KANTE (APPROXIMATIV)

Wir haben nun Verfahren zur Verfügung, die in der Lage sind zu testen, ob zu einem möglichen Kollisionszeitpunkt t tatsächlich eine Kollision *Ecke–Fläche* oder *Kante–Kante* vorliegt. Im nächsten Abschnitt wollen wir uns damit beschäftigen, wie mögliche Kollisionszeitpunkte bestimmt werden können.

5.3.1 Bestimmung möglicher Kollisionszeitpunkte

Ecke–Fläche

Im Fall der *Ecke–Fläche* Kollision ist eine notwendige Bedingung für eine Kollision, daß der Eckpunkt $\boldsymbol{v}(t)$ in der zu der Fläche $f(t)$ gehörigen Ebene $p_f(t)$ liegt. In diesem Fall muß dann getestet werden, ob $\boldsymbol{v}(t)$ auch innerhalb der Fläche $f(t)$ liegt.

Die Ecke $\boldsymbol{v}(t)$ liegt genau dann in der Ebene $p_f(t)$, wenn der vorzeichenbehaftete Abstand $d(t) = d(\boldsymbol{v}(t), p_f(t)) = \boldsymbol{n}_f(t) \cdot \boldsymbol{v}(t) - d_f(t) = 0$ ist. Daher betrachten wir zur Bestimmung möglicher Kollisionszeiptunkte $d(t)$. Leider hat $d(t)$ bei den von uns betrachteten Bewegungen keine einfache geschlossene Funktionsbeschreibung, so daß die Nullstellen dieser

Funktion nicht effizient bestimmt werden können. Eine Ausnahme besteht, falls sich die Ecke und die Fläche nur translatorisch bewegen. In diesem Fall ensteht eine lineare Abstandsfunktion, deren Nullstellen einfach zu bestimmen sind. Im allgemeinen werden wir jedoch nur Approximationen der Funktion berechnen können und daher auch nur approximative Nullstellen.

Kante–Kante

Auch im Fall der *Kante–Kante* Kollision wollen wir die notwendige Bedingung, daß zum Kollisionszeitpunkt der Abstand der beiden Geraden $l_1(t)$ und $l_2(t)$ gleich 0 ist, benutzen, um mögliche Kollisionszeitpunkte zu bestimmen. Sei $\boldsymbol{n}(t) = (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) \times (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))$ der Normalenvektor, der durch $l_1(t)$ und $l_2(t)$ aufgespannten Ebene. Sind $l_1(t)$ und $l_2(t)$ parallel, d.h. $|\boldsymbol{n}(t)| = 0$, so betrachten wir anstatt des Zeitpunktes t einen leicht verschobenen Zeitpunkt t', für den $\boldsymbol{n}(t')$ berechnet wird. Ist auch zu diesem $\boldsymbol{n}(t') = 0$, so bewegen sich die beiden Kanten rein translatorisch und eine mögliche Kollision zwischen den beiden Kanten ergibt auch eine *Ecke-Fläche* Kollision, so daß auf diesen *Kante-Kante* Kollisionstest verzichtet werden kann. Der vorzeichenbehaftete Abstand

$$d(t) = d(l_1(t), l_2(t)) = \frac{\boldsymbol{n}(t)^T \cdot (\boldsymbol{v}_1(t) - \boldsymbol{w}_1(t))}{|\boldsymbol{n}(t)|}$$

hat dabei im Vergleich zur Abstandsfunktion beim *Ecke-Fläche* Test neben der fehlenden geschlossenen Funktionsbeschreibung zur effizienten Nullstellenbestimmung den weiteren Nachteil, daß er nicht notwendigerweise stetig ist. Dies ergibt sich dadurch, daß sich der Normalenvektor $\boldsymbol{n}(t)$ diskontinuierlich verändern kann. Diesem Phänomen muß bei der Approximation der Nullstellen der Abstandsfunktion Rechnung getragen werden. Eine Ausnahme bildet auch hier wieder der Fall, daß sich die beiden Kanten nur translatorisch bewegen. In diesem Fall ergibt sich analog zum *Ecke–Fläche* Test eine stetige, lineare Funktion.

Fehlerquellen bei der Approximation von möglichen Kollisionszeitpunkten

Durch die Art der Approximation von möglichen Kollisionszeitpunkten können drei Arten von Fehler auftreten:

1. *Überlappung zum approximierten Kollisionszeitpunkt*
   Durch die approximative Berechnung der möglichen Kollisionszeitpunkte liegt zu diesen schon eine echte Überlappung der Flächenmengen vor.

2. *Nichterkennung einer Kollision trotz erkannter Nullstelle der Abstandsfunktion*
   Es wird zwar eine Nullstelle der Abstandsfunktion bestimmt, jedoch dadurch, daß sie zu ungenau ist, wird zu diesem Zeitpunkt keine tatsächliche Kollision festgestellt.

3. *Nichterkennung einer Kollision wegen nichterkannter Nullstellen der Abstandsfunktion*
   Dies kann dann der Fall sein, wenn während der Bewegung eine Kollision auftritt, diese jedoch wieder verschwindet.

In den nächsten Abschnitten werden wir Verfahren zur Bestimmung möglicher Kollisionszeitpunkte vorstellen. Bei diesen ist zu beachten, daß sie in *Virtual Reality* Anwendungen verwendet werden, so daß nur einfache Approximationsverfahren in Betracht kommen.

5.3.2 Mögliche Kollisionszeitpunkte bei linearer Abstandsinterpolation

Eine erste Methode, die Nullstellen der unbekannten Abstandsfunktion zu bestimmen, ist eine lineare Interpolation der Abstandsfunktion zwischen den Zeitpunkten 0 und 1, d.h. mittels der Funktion

$$d_{lin}(t) = d(0) + t \cdot (d(1) - d(0)).$$

Wechselt das Vorzeichen der Interpolationsfunktion d_{lin} im Intervall $[0, 1]$, so besitzt diese eine Nullstelle zum Zeitpunkt $t = \frac{d(0)}{d(0)-d(1)}$. Die Berechnung der Nullstelle ist numerisch instabil, wenn die Werte $d(0)$ und $d(1)$ dicht zusammenliegen, d.h. näher als eine Schranke ε mit $|d(0) - d(1)| < \varepsilon$. In diesem Fall nehmen wir eine Nullstelle zum Zeitpunkt $t = 0$ an. Die Vorgehensweise des Verfahrens ist in Abbildung 5.10 veranschaulicht.

Für die so ermittelten möglichen Kollisionszeitpunkte t muß dann noch der Test durchgeführt werden, ob tatsächlich eine Kollision *Ecke-Fläche* oder *Kante–Kante* stattgefunden hat. Daraus ergeben sich für den *Ecke–Fläche* Test der Algorithmus *EckeFlächeLineareInterpolation* in Abbildung 5.11 und für den *Kante–Kante* Test der Algorithmus *KanteKanteLineareInterpolation* in Abbildung 5.12.

Für den Fall rein translatorischer Bewegungen entspricht die lineare Interpolation der Abstandsfunktionen dem tatsächlichen Verlauf der Abstandsfunktionen und der approximierte mögliche Kollisionszeitpunkt ist beim Vorliegen einer tatsächlichen Kollision exakt. Ist jedoch mindestens ein rotatorischer Bewegungsanteil vorhanden, dann verläuft die Abstandsfunktion in der Regel nicht mehr linear und die Approximation des möglichen Kollisionszeitpunktes kann fehlerhaft sein. Um den möglichen Fehler in dem Fall zu verkleinern, daß die Abstandsfunktion während der Bewegung ihr Vorzeichen ändert, kann das Verfahren der Approximation der Nullstelle der Abstandsfunktion durch eine Iteration verbessert und dabei zur Bestimmung möglicher Kollisionszeitpunkte bei *iterativer Abstandsinterpolation mit Regula Falsi* übergegangen werden.

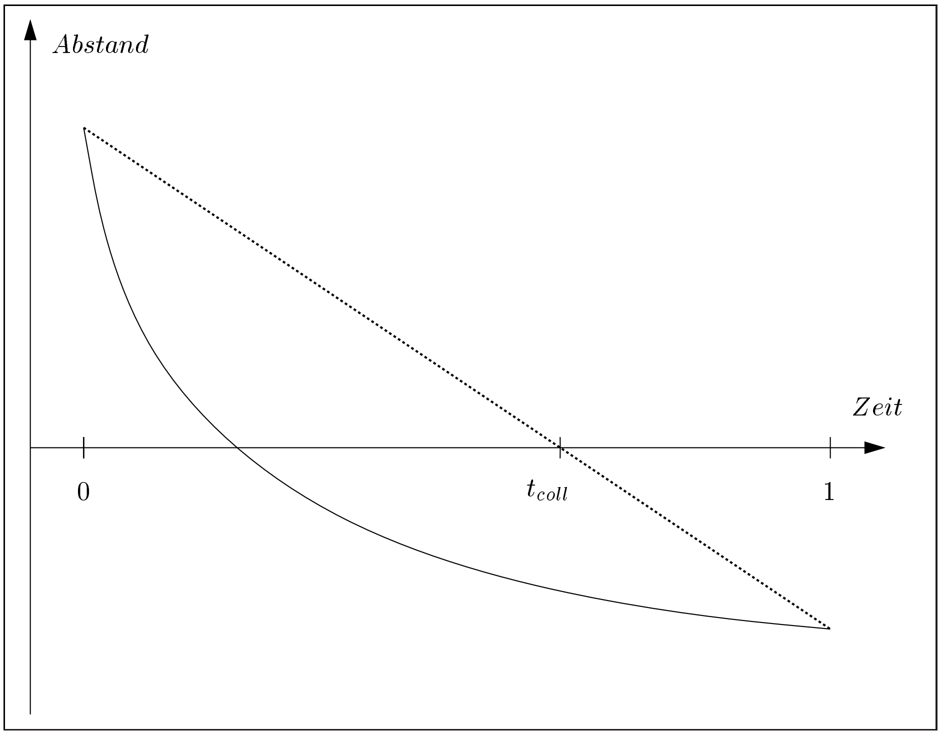

Abbildung 5.10: NULLSTELLENBESTIMMUNG MITTELS LINEARER INTERPOLATION

5.3.3 Mögliche Kollisionszeitpunkte bei iterativer Abstandsinterpolation mit *Regula falsi*

Eine vorhandene Nullstelle der Abstandsfunktion wird durch die *lineare Abstandsinterpolation* evtl. nur unzureichend approximiert. Um die Approximation zu verbessern, kann die Nullstelle durch Anwendung eines iterativen Approximationsverfahrens verbessert werden. Da keine Ableitung der Abstandsfunktion bekannt ist, bietet sich das *Regula falsi* Verfahren als gemischtes Intervallhalbierungs-Sekanten Verfahren an. Die Voraussetzung für die Berechnung einer Nullstelle ist dabei wie bei der *linearen Abstandsinterpolation*, daß die Abstandsfunktion im Intervall $[0, 1]$ einen Vorzeichenwechsel durchführt. Als Startwerte für das Verfahren dienen

$$t_0 = 0 \quad , \quad t_1 = 1$$

sowie die zugehörigen Abstandswerte $d(0)$ und $d(1)$. Ein Schritt des *Regula falsi* Verfahren ergibt sich dann durch

$$\begin{array}{l} t_{neu} = t_0 - \frac{d(t_0)\cdot(t_1-t_0)}{d(t_1)-d(t_0)} \\ t_0 := t_{neu},\ \text{falls } d(t_0)\cdot d(t_{neu}) > 0 \\ t_1 := t_{neu},\ \text{falls } d(t_0)\cdot d(t_{neu}) < 0. \end{array}$$

Algorithmus *EckeFlächeLineareInterpolation*$(\boldsymbol{v}, f, typ)$

Eingabe:	$\boldsymbol{v}$	Ecke
	$f = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{k-1}, \boldsymbol{v}_k \equiv \boldsymbol{v}_0)$	Fläche mit Ebene $\boldsymbol{n}_f \cdot \boldsymbol{x} - d_f = 0$
	typ	Typ der Ecke-in-Fläche Berechnung EXAKT = exakte Berechnung INTER = lineare Interpolation
Ausgabe:	c	Kollisionsflag
	t	Kollisionszeitpunkt

$d(0) \leftarrow \boldsymbol{n}_f(0) \cdot \boldsymbol{v}(0) - d_f(0)$
$d(1) \leftarrow \boldsymbol{n}_f(1) \cdot \boldsymbol{v}(1) - d_f(1)$
if $(d(0) \cdot d(1) < 0)$
then $(*$ Abstandsfunktion wechselt Vorzeichen $*)$
  if $(|d(0) - d(1)| > \varepsilon)$
  then $t \leftarrow \frac{d(0)}{d(0)-d(1)}$
  else $t \leftarrow 0$
else $(*$ Abstandsfunktion wechselt das Vorzeichen nicht $*)$
  return (NO COLLISION,-1)
if (typ =EXAKT)
then *inside* $\leftarrow$ *EckeInnerhalbFlächeExakt*$(\boldsymbol{v}(t), f(t))$
else *inside* $\leftarrow$ *EckeInnerhalbFlächeApproximativ*$(\boldsymbol{v}, f, t)$
if (*inside* = TRUE)
then return (COLLISION,t)
else return (NO COLLISION,-1)

Abbildung 5.11: ALGORITHMUS: TEST ECKE GEGEN FLÄCHE (LINEARE INTERPOLATION)

Für den Fall, daß $d(t_{neu}) = 0$ ist, wird das Verfahren beendet, da t_{neu} die gesuchte Nullstelle ist. Das Vorgehen des Verfahrens ist in Abbildung 5.13 graphisch veranschaulicht.

Eine Eigenschaft des Verfahrens ist, daß $t_0 \leq t_{neu} \leq t_1$ gilt, d.h. eine intelligente Form der Intervallschachtelung für die Nullstelle durchgeführt wird (vgl. [St93]). Dieses Verfahren kann nun solange durchgeführt werden, bis die Größe des Intervalls $[t_0, t_1]$, auf das die Nullstelle eingegrenzt wurde, eine Schranke unterschreitet, d.h. $|t_0 - t_1| < \varepsilon_1$, oder eine maximale Anzahl von Iterationen durchgeführt worden ist. Dabei ist gleichzeitig darauf zu achten, daß das Verfahren numerisch instabil wird, wenn die Werte $d(t_0)$ und $d(t_1)$ zu dicht zusammenliegen, d.h. $|d(t_0) - d(t_1)| < \varepsilon_2$, so daß auch dann das Verfahren abgebrochen werden sollte. Zu diesem Zeitpunkt stellt t_0 eine konservative Approximation der Nullstelle der Abstandsfunktion dar.

Für die so ermittelten möglichen Kollisionszeitpunkte t muß dann noch der Test durchgeführt werden, ob tatsächlich eine Kollision *Ecke-Fläche* oder *Kante–Kante* stattgefunden hat. Daraus ergeben sich für den *Ecke–Fläche* Test der Algorithmus *Ecke-*

Algorithmus *KanteKanteLineareInterpolation*(e_1, e_2, typ)

Eingabe:
e_1, e_2 Kanten
typ Typ der Schnittpunkt-in-Kante Berechnung
EXAKT = exakte Berechnung
INTER = lineare Interpolation

Ausgabe:
c Kollisionsflag
t Kollisionszeitpunkt

for t **in** $\{0,1\}$
do $\boldsymbol{n}(t) \leftarrow (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) \times (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))$
  if $(|\boldsymbol{n}(t)| > 0)$
  then $d(t) \leftarrow \frac{\boldsymbol{n}(t)^T \cdot (\boldsymbol{v}_1(t) - \boldsymbol{w}_1(t))}{|\boldsymbol{n}(t)|}$
  else **return** (NO COLLISION,-1)
if $(d(0) \cdot d(1) < 0)$
then (∗ Abstandsfunktion wechselt Vorzeichen ∗)
  if $(|d(0) - d(1)| > \varepsilon)$
  then $t \leftarrow \frac{d(0)}{d(0)-d(1)}$
  else $t \leftarrow 0$
else (∗ Abstandsfunktion wechselt das Vorzeichen nicht ∗)
  return (NO COLLISION,-1)
if $(typ = \text{EXAKT})$
then $inside \leftarrow$ *KanteInnerhalbKanteExakt*$(e_1(t), e_2(t))$
else $inside \leftarrow$ *KanteInnerhalbKanteApproximativ*(e_1, e_2, t)
if $(inside = \text{TRUE})$
then **return** (COLLISION,t)
else **return** (NO COLLISION,-1)

Abbildung 5.12: ALGORITHMUS: TEST KANTE GEGEN KANTE (LINEARE INTERPOLATION)

FlächeIterativeInterpolationRegulaFalsi in Abbildung 5.14 und für den *Kante–Kante* Test der Algorithmus *KanteKanteIterativeInterpolationRegulaFalsi* in Abbildung 5.15.

Die *iterative Abstandsinterpolation mit Regula Falsi* ist in der Lage Kollisionszeitpunkte genauer zu approximieren, für die die *lineare Abstandsinterpolation* mitunter nur unzureichende Näherungen zu berechnen vermag. Jedoch versagt auch sie, wenn während der Bewegung die Abstandsfunktion Nullstellen hat, jedoch zwischen dem Abstand zum Zeitpunkt 0 und 1 kein Vorzeichenwechsel vorliegt. Dies kann z.B. schon dann der Fall sein, wenn sich nur eines der beiden Objekte rein rotatorisch bewegt. Um in diesem Fall eine Möglichkeit zu haben, einen Kollisionszeitpunkt zu bestimmen, kann von der *linearen* zur *quadratischen Abstandsinterpolation* übergegangen werden.

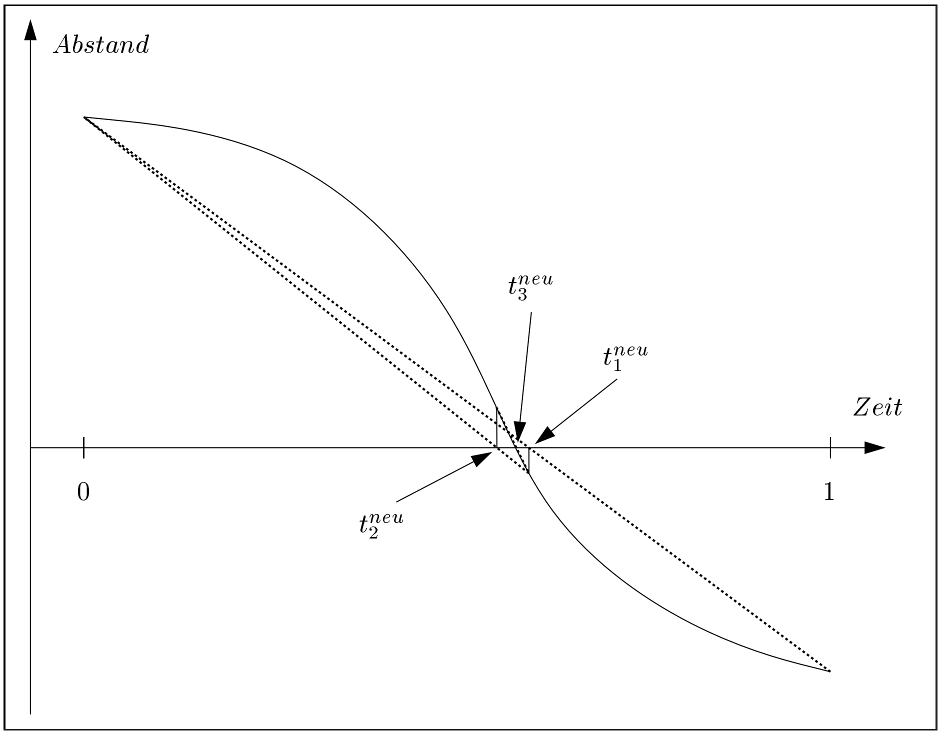

Abbildung 5.13: NULLSTELLENBESTIMMUNG MIT *Regula falsi*

5.3.4 Mögliche Kollisionszeitpunkte bei quadratischer Abstandsinterpolation

Ein Problem der Nullstellenbestimmung mit *linearer Interpolation* und *iterativer Interpolation mit Regula falsi* ist, daß sie nur in den Fällen Erfolg hat, in denen sich das Vorzeichen der Abstandsfunktion im Intervall $[0,1]$ ändert. Es kann jedoch sein, daß das Vorzeichen am Anfang und am Ende des Intervalls $[0,1]$ gleich ist, jedoch trotzdem eine Kollision während der Bewegung aufgetreten ist. Diesem Problem kann dadurch besser begegnet werden, indem der Grad des Interpolationspolynoms der Abstandsfunktion auf den Grad 2 erhöht wird.

Zur Interpolation der Abstandsfunktion mittels eines Polynoms zweiten Grades benötigen wir neben den Zeitpunkten 0 und 1 eine weitere Stützstelle. Die Vorgehensweise der *quadratischen Interpolation* ist in Abbildung 5.16 graphisch veranschaulicht.

Wechselt die Distanzfunktion im Intervall $[0,1]$ das Vorzeichen, so ist es sinnvoll, als dritte Stützstelle das Ergebnis der *linearen Interpolation* zu benutzen in der Hoffnung, daß der derart berechnete Wert einer wahren Nullstelle der Abstandsfunktion sehr nahe kommt und die Qualität des Interpolationspolynoms in der Nähe der Stützstellen tendenziell besser ist als in den restlichen Bereichen. Sei $\alpha{\cdot}t^2{+}\beta{\cdot}t{+}\gamma$ das zu berechnende Interpolationspolynom, dann ergeben sich die Koeffizienten α, β und γ wie folgt:

Algorithmus $\mathit{EckeFl\ddot{a}cheIterativeInterpolationRegulaFalsi}(\boldsymbol{v}, f, typ)$

	$\boldsymbol{v}$	Ecke
	$f = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{k-1}, \boldsymbol{v}_k \equiv \boldsymbol{v}_0)$	Fläche mit Ebene $\boldsymbol{n}_f \cdot \boldsymbol{x} - d_f = 0$
Eingabe:	typ	Typ der Ecke-in-Fäche Berechnung
		EXAKT = exakte Berechnung
		INTER = lineare Interpolation
Ausgabe:	c	Kollisionsflag
	t	Kollisionszeitpunkt

$d(0) \leftarrow \boldsymbol{n}_f(0) \cdot \boldsymbol{v}(0) - d_f(0)$

$d(1) \leftarrow \boldsymbol{n}_f(1) \cdot \boldsymbol{v}(1) - d_f(1)$

if $(d(0) \cdot d(1) < 0)$

then $t_{neu} \leftarrow t_0 \leftarrow 0$

  $d(t_{neu}) \leftarrow d(t_0) \leftarrow d(0)$

  $t_1 \leftarrow 1$

  $d(t_1) \leftarrow d(1)$

  while $((d(t_{neu}) \neq 0)$ **and** $(|t_0 - t_1| > \varepsilon_1)$ **and** $(|d(t_0) - d(t_1)| > \varepsilon_2))$

   do $t_{neu} \leftarrow t_0 - \frac{d(t_0)\cdot(t_1 - t_0)}{d(t_1) - d(t_0)}$

    $d(t_{neu}) \leftarrow \boldsymbol{n}_f(t_{neu}) \cdot \boldsymbol{v}(t_{neu}) - d_f(t_{neu})$

    if $(d(t_0) \cdot d(t_{neu}) \geq 0)$

    then $t_0 \leftarrow t_{neu}$

     $d(t_0) \leftarrow d(t_{neu})$

    else $t_1 \leftarrow t_{neu}$

     $d(t_1) \leftarrow d(t_{neu})$

  $t \leftarrow t_0$

else **return** (NO COLLISION,-1)

if (typ =EXAKT)

then $inside \leftarrow \mathit{EckeInnerhalbFl\ddot{a}cheExakt}(\boldsymbol{v}(t), f(t))$

else $inside \leftarrow \mathit{EckeInnerhalbFl\ddot{a}cheApproximativ}(\boldsymbol{v}, f, t)$

if ($inside$ =TRUE)

then **return** (COLLISION,t)

else **return** (NO COLLISION,-1)

Abbildung 5.14: ALGORITHMUS: TEST ECKE GEGEN FLÄCHE (ITERATIVE INTERPOLATION REGULA FALSI)

$d(0) \cdot d(1) \geq 0$	$d(0) \cdot d(1) < 0$
$t_0 = 0$ $t_1 = 0.5$ $t_2 = 1$ $y_0 = d(0)$ $y_1 = d(0.5)$ $y_2 = d(1)$	$t_0 = 0$ $t_1 = \frac{d(0)}{d(0)-d(1)}$ $t_2 = 1$ $y_0 = d(0)$ $y_1 = d(t_1)$ $y_2 = d(1)$
$\alpha = 2 \cdot (d(0) - 2 \cdot d(0.5) + d(1))$ $\beta = 4 \cdot d(0.5) - 3 \cdot d(0) - d(1)$ $\gamma = d(0)$	$\alpha = \frac{d(0)-y_1}{t_1} + \frac{y_1 - d(1)}{t_1 - 1}$ $\beta = \frac{y_1 - d(0)}{t_1} + \frac{t_1 \cdot d(1) - y_1}{t_1 - 1} - d(0)$ $\gamma = d(0)$

Algorithmus *KanteKanteIterativeInterpolationRegulaFalsi*(e_1, e_2, typ)

Eingabe:	e_1, e_2	Kanten
	typ	Typ der Schnittpunkt-in-Kante Berechnung
		EXAKT = exakte Berechnung
		INTER = lineare Interpolation
Ausgabe:	c	Kollisionsflag
	t	Kollisionszeitpunkt

for t **in** $\{0, 1\}$
do $\boldsymbol{n}(t) \leftarrow (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) \times (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))$
    if $(|\boldsymbol{n}(t)| > 0)$
    then $d(t) \leftarrow \frac{\boldsymbol{n}(t)^T \cdot (\boldsymbol{v}_1(t) - \boldsymbol{w}_1(t))}{|\boldsymbol{n}(t)|}$
    else return (NO COLLISION,-1)
if $(d(0) \cdot d(1) < 0)$
then $t_{neu} \leftarrow t_0 \leftarrow 0; \quad d(t_{neu}) \leftarrow d(t_0) \leftarrow d(0)$
        $t_1 \leftarrow 1; \quad d(t_1) \leftarrow d(1)$
        while $((d(t_{neu}) \neq 0)$ **and** $(|t_0 - t_1| > \varepsilon_1)$ **and** $(|d(t_0) - d(t_1)| > \varepsilon_2))$
          do $t_{neu} \leftarrow t_0 - \frac{d(t_0) \cdot (t_1 - t_0)}{d(t_1) - d(t_0)}$
              $\boldsymbol{n}(t_{neu}) \leftarrow (\boldsymbol{v}_2(t_{neu}) - \boldsymbol{v}_1(t_{neu})) \times (\boldsymbol{w}_2(t_{neu}) - \boldsymbol{w}_1(t_{neu}))$
              if $(|\boldsymbol{n}(t_{neu})| > 0)$
              then $d(t_{neu}) \leftarrow \frac{\boldsymbol{n}(t_{neu})^T \cdot (\boldsymbol{v}_1(t_{neu}) - \boldsymbol{w}_1(t_{neu}))}{|\boldsymbol{n}(t_{neu})|}$
              else return (NO COLLISION,-1)
              if $(d(t_0) \cdot d(t_{neu}) \geq 0)$
              then $t_0 \leftarrow t_{neu}; \quad d(t_0) \leftarrow d(t_{neu})$
              else $t_1 \leftarrow t_{neu}; \quad d(t_1) \leftarrow d(t_{neu})$
else return (NO COLLISION,-1) $t \leftarrow t_0$
if $(typ =$ EXAKT$)$
then *inside* $\leftarrow$ *KanteInnerhalbKanteExakt*$(e_1(t), e_2(t))$
else *inside* $\leftarrow$ *KanteInnerhalbKanteApproximativ*(e_1, e_2, t)
if (*inside* =TRUE)
then return (COLLISION,t)
else return (NO COLLISION,-1)

Abbildung 5.15: Algorithmus: Test Kante gegen Kante (iterative Interpolation Regula falsi)

Die Nullstellen dieses Interpolationspolynoms ergeben sich bekanntermaßen zu

$$t_{1,2} = \frac{-\beta \pm \sqrt{\beta^2 - 4 \cdot \alpha \cdot \gamma}}{2 \cdot \alpha}.$$

Zur Beschleunigung der Berechnung der t_i im Intervall $[0, 1]$ werden Verfahren in [ST94] Anhang A beschrieben. Für die so ermittelten möglichen Kollisionszeitpunkte $t \in [0, 1]$

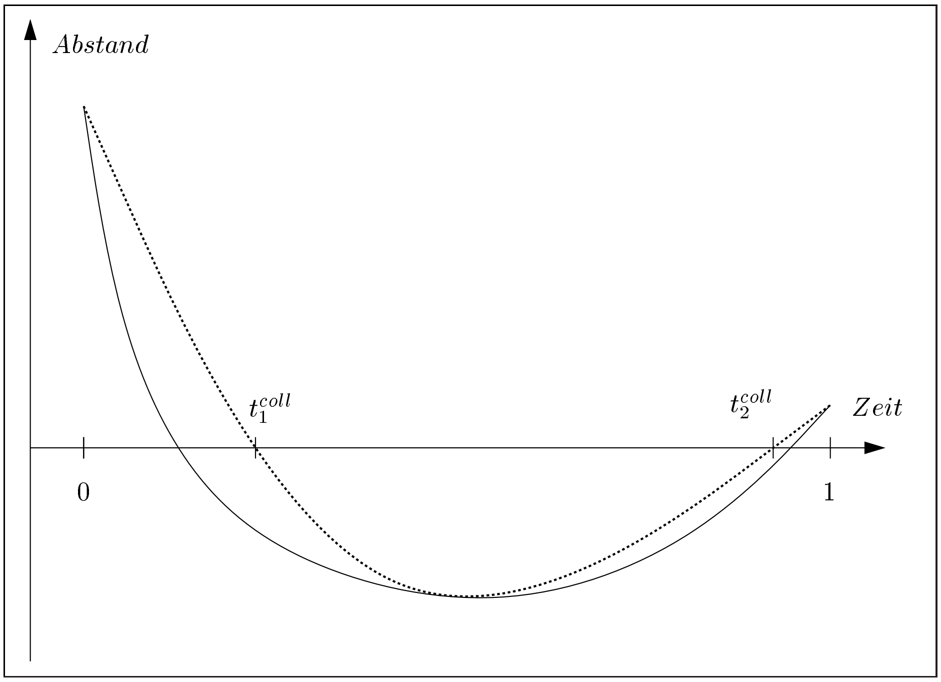

Abbildung 5.16: Nullstellenbestimmung mit quadratischem Interpolationspolynom

muß dann noch der Test durchgeführt werden, ob tatsächlich eine Kollision *Ecke-Fläche* oder *Kante–Kante* stattgefunden hat. Daraus ergeben sich für den *Ecke–Fläche* Test der Algorithmus *EckeFlächeQuadratischeInterpolation* in Abbildung 5.17 und für den *Kante–Kante* Test der Algorithmus *KanteKanteQuadratischeInterpolation* in Abbildung 5.18.

Die *quadratische Abstandsinterpolation* ist in der Lage, Kollisionszeitpunkte zu approximieren, bei denen während der Bewegung eine Kollision auftritt, jedoch die Abstandsfunktion zum Zeitpunkt 0 und 1 betrachtet keinen Vorzeichenwechsel durchführt, so daß jede Form der *linearen Interpolation* keinen Ansatzpunkt zur Bestimmung eines möglichen Kollisionszeitpunktes vorfindet. Dies kann z.B. dann der Fall sein, wenn sich mindestens eines der beiden Objekte rotatorisch bewegt. Jedoch können auch im Falle der *quadratischen Interpolation* die approxmierten Kollisionszeitpunkte analog zur *linearen Interpolation* zu ungenau sein. Um eine beliebig genaue Lokalisierung von möglichen Kollisionszeitpunkten zu erhalten, können wir die Abstandsfunktion mittels eines Rasters abtasten und auf diese Weise mögliche Kollisionszeitpunkte ermitteln.

5.3.5 Mögliche Kollisionszeitpunkte bei linearer Abstandsabtastung

Die *quadratische Abstandsinterpolation* ist mitunter in der Lage mögliche Kollisionszeitpunkte zu bestimmen, die während einer Bewegung auftreten, bei denen die Abstands-

Algorithmus *EckeFlächeQuadratischeInterpolation*$(\boldsymbol{v}, f, typ)$

Eingabe:	$\boldsymbol{v}$	Ecke
	$f = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{k-1}, \boldsymbol{v}_k \equiv \boldsymbol{v}_0)$	Fläche mit Ebene $\boldsymbol{n}_f \cdot \boldsymbol{x} - d_f = 0$
	typ	Typ der Ecke-in-Fäche Berechnung EXAKT = exakte Berechnung INTER = lineare Interpolation
Ausgabe:	c	Kollisionsflag
	t	Kollisionszeitpunkt

$d(0) \leftarrow \boldsymbol{n}_f(0) \cdot \boldsymbol{v}(0) - d_f(0)$
$d(1) \leftarrow \boldsymbol{n}_f(1) \cdot \boldsymbol{v}(1) - d_f(1)$
if $(d(0) \cdot d(1) < 0)$
then $t_1 \leftarrow \frac{d(0)}{d(0)-d(1)}$
else $t_1 \leftarrow 0.5$
$y_1 \leftarrow \boldsymbol{n}_f(t_1) \cdot \boldsymbol{v}(t_1) - d_f(t_1)$
$\alpha \leftarrow \frac{d(0)-y_1}{t_1} - \frac{y_1-d(1)}{t_1-1}; \quad \beta \leftarrow \frac{y_1-d(0)}{t_1} + \frac{t_1 \cdot d(1)-y_1}{t_1-1} - d(0); \quad \gamma \leftarrow d(0)$
$t_{1,2} \leftarrow \frac{-\beta \pm \sqrt{\beta^2 - 4 \cdot \alpha \cdot \gamma}}{2 \cdot \alpha}$
$t_{min} \leftarrow \min(t_1, t_2); \quad t_{max} \leftarrow \max(t_1, t_2)$
for t **in** $\{t_{min}, t_{max}\}$
do if $(0 \leq t \leq 1)$
    then if $(typ = \text{EXAKT})$
        then $inside \leftarrow$ *EckeInnerhalbFlächeExakt*$(\boldsymbol{v}(t), f(t))$
        else $inside \leftarrow$ *EckeInnerhalbFlächeApproximativ*$(\boldsymbol{v}, f, t)$
        if $(inside = \text{TRUE})$
        then return (COLLISION,t)
return (NO COLLISION,-1)

Abbildung 5.17: ALGORITHMUS: TEST ECKE GEGEN FLÄCHE (QUADRATISCHE INTERPOLATION)

funktion zum Zeitpunkt 0 und 1 betrachtet, keinen Vorzeichenwechsel aufzeigt. Jedoch ist dies nicht in jedem Fall möglich, so daß nicht garantiert werden kann, daß die *quadratische Abstandsinterpolation* alle möglichen Kollisionen erkennt, die länger andauern als ein vorgegebener Bruchteil Δ der Gesamtlänge der Bewegung.

Dies kann man dadurch erreichen, daß man die Abstandsfunktion mit einem Raster der Größe Δ abtastet. Wird dabei vom Zeitpunkt t_i zu t_{i+1} ein Vorzeichenwechsel in der Abstandsfunktion registriert, so ergibt sich die Approximation eines möglichen Kollisionszeitpunktes mittels linearer Interpolation zwischen $(t_i, d(t_1))$ und $(t_{i+1}, d(t_{i+1}))$. Die Vorgehensweise der *linearen Abstandsabtastung* ist in Abbildung 5.19 graphisch veranschaulicht.

Für die so ermittelten möglichen Kollisionszeitpunkte $t \in [0, 1]$ wird sukzessive getestet, ob tatsächlich eine Kollision *Ecke–Fläche* oder *Kante–Kante* vorliegt. Daraus ergeben sich für den *Ecke–Fläche* Test der Algorithmus *EckeFlächeLineareAbtastung* in Abbildung 5.20 und

Algorithmus *KanteKanteQuadratischeInterpolation*(e_1, e_2, typ)

Eingabe:
e_1, e_2 Kanten
typ Typ der Schnittpunkt-in-Kante Berechnung
EXAKT = exakte Berechnung
INTER = lineare Interpolation

Ausgabe:
c Kollisionsflag
t Kollisionszeitpunkt

for t **in** $\{0, 1\}$
do $\boldsymbol{n}(t) \leftarrow (\boldsymbol{v}_2(t) - \boldsymbol{v}_1(t)) \times (\boldsymbol{w}_2(t) - \boldsymbol{w}_1(t))$
  if $(|\boldsymbol{n}(t)| > 0)$
  then $d(t) \leftarrow \frac{\boldsymbol{n}(t)^T \cdot (\boldsymbol{v}_1(t) - \boldsymbol{w}_1(t))}{|\boldsymbol{n}(t)|}$
  else **return** (NO COLLISION,-1)
if $(d(0) \cdot d(1) < 0)$
then $t_1 \leftarrow \frac{d(0)}{d(0) - d(1)}$
else $t_1 \leftarrow 0.5$
$\boldsymbol{n}(t_1) \leftarrow (\boldsymbol{v}_2(t_1) - \boldsymbol{v}_1(t_1)) \times (\boldsymbol{w}_2(t_1) - \boldsymbol{w}_1(t_1))$
if $(|\boldsymbol{n}(t_1)| > 0)$
then $y_1 \leftarrow \frac{\boldsymbol{n}(t_1)^T \cdot (\boldsymbol{v}_1(t_1) - \boldsymbol{w}_1(t_1))}{|\boldsymbol{n}(t_1)|}$
else **return** (NO COLLISION,-1)
$\alpha \leftarrow \frac{d(0) - y_1}{t_1} - \frac{y_1 - d(1)}{t_1 - 1}; \quad \beta \leftarrow \frac{y_1 - d(0)}{t_1} + \frac{t_1 \cdot d(1) - y_1}{t_1 - 1} - d(0); \quad \gamma \leftarrow d(0)$
$t_{1,2} \leftarrow \frac{-\beta \pm \sqrt{\beta^2 - 4 \cdot \alpha \cdot \gamma}}{2 \cdot \alpha}$
$t_{min} \leftarrow \min(t_1, t_2); \quad t_{max} \leftarrow \max(t_1, t_2)$
for t **in** $\{t_{min}, t_{max}\}$
do if $(0 \leq t \leq 1)$
  then if $(typ =$ EXAKT$)$
    then $inside \leftarrow$ *KanteInnerhalbKanteExakt*$(e_1(t), e_2(t))$
    else $inside \leftarrow$ *KanteInnerhalbKanteApproximativ*(e_1, e_2, t)
    if $(inside =$TRUE$)$
    then return (COLLISION,t)
return (NO COLLISION,-1)

Abbildung 5.18: ALGORITHMUS: TEST KANTE GEGEN KANTE (QUADRATISCHE INTERPOLATION)

für den *Kante–Kante* Test der Algorithmus *KanteKanteLineareAbtastung* in Abbildung 5.21.

Diese Vorgehensweise ist in der Lage durch Verkleinerung des Rasters eine immer genauere Lokalisierung von möglichen Kollisionszeitpunkten durchzuführen. Durch die Wahl von $\Delta = 1$ ergibt sich die *lineare Abstandsinterpolation.* Die *lineare Abtastung* kann noch dadurch verbessert werden, daß sie mit der iterativen Nullpunktbestimmung verbunden

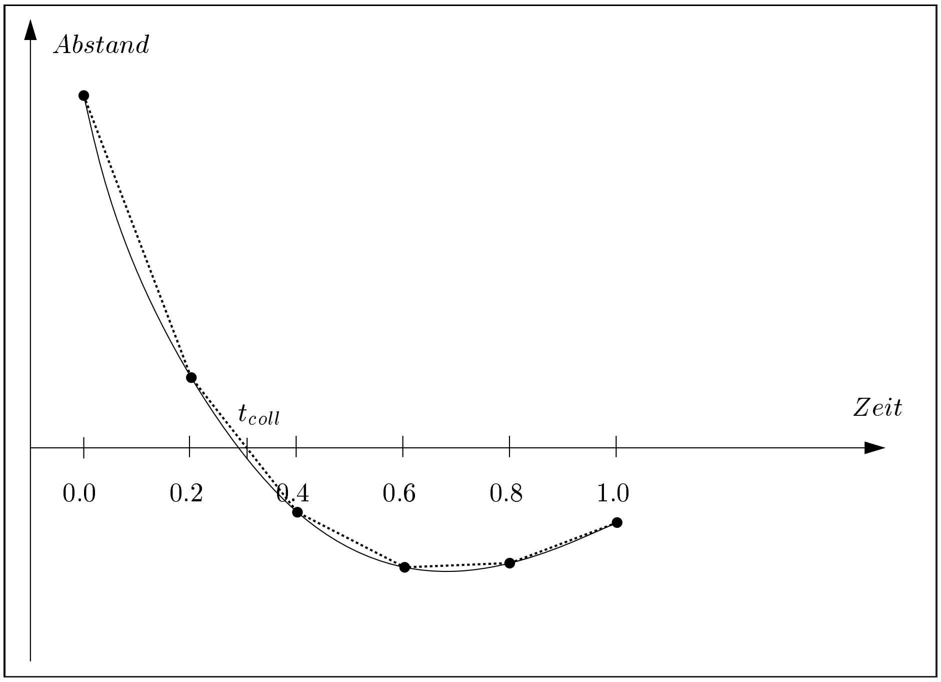

Abbildung 5.19: Nullstellenbestimmung mit linearer Abtastung

wird. Dadurch kann bei gleicher Rastergröße eine genauere Bestimmung von möglichen Kollisionszeitpunkten durchgeführt werden.

5.3.6 Mögliche Kollisionszeitpunkte bei Abstandsabtastung und *Regula falsi*

Die *Abstandsabtastung zusammen mit Regula falsi* verbessert bei gleicher Rastergröße die Approximation der möglichen Kollisionszeitpunkte durch die Anwendung des iterativen Verfahrens zur Nullstellenbestimmung der *linearen Abstandsinterpolation mit Regula falsi*, falls bei der Abtastung der Abstandsfunktion ein Vorzeichenwechsel aufgetreten ist.

Für die so ermittelten möglichen Kollisionszeitpunkte $t \in [0, 1]$ wird sukzessive getestet, ob tatsächlich eine Kollision *Ecke–Fläche* oder *Kante–Kante* vorliegt. Daraus ergeben sich für den *Ecke–Fläche* Test der Algorithmus *EckeFlächeAbtastungRegulaFalsi* in Abbildung 5.22 und für den *Kante–Kante* Test der Algorithmus *KanteKanteAbtastungRegulaFalsi* in Abbildung 5.23.

5.3.7 Zusammenfassung und Vergleich der Approximationsverfahren

1. *Lineare Abstandsinterpolation*

Algorithmus *EckeFlächeLineareAbtastung*$(\boldsymbol{v}, f, \Delta, typ)$

Eingabe:

$\boldsymbol{v}$	Ecke
$f = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{k-1}, \boldsymbol{v}_k \equiv \boldsymbol{v}_0)$	Fläche mit Ebene $\boldsymbol{n}_f \cdot \boldsymbol{x} - d_f = 0$
Δ	Abtastraster
typ	Typ der Ecke-in-Fäche Berechnung EXAKT = exakte Berechnung INTER = lineare Interpolation

Ausgabe:

c	Kollisionsflag
t	Kollisionszeitpunkt

$t_0 \leftarrow t_1 \leftarrow 0 \;\; d_0 \leftarrow d_1 \leftarrow \boldsymbol{n}_f(0) \cdot \boldsymbol{v}(0) - d_f(0)$
while $(t_1 \leq 1)$
  do $t_0 \leftarrow t_1$
    $d_0 \leftarrow d_1$
    $t_1 \leftarrow t_0 + \Delta$
    $d_1 \leftarrow \boldsymbol{n}_f(t_1) \cdot \boldsymbol{v}(t_1) - d_f(t_1)$
    if $(d_0 \cdot d_1 \leq 0)$
    then if $(|d_1 - d_0| > \varepsilon)$
      then $t \leftarrow t_0 - d_0 \cdot \frac{t_1 - t_0}{d_1 - d_0}$
      else $t \leftarrow t_0$
      if $(typ = \text{EXAKT})$
      then $inside \leftarrow$ *EckeInnerhalbFlächeExakt*$(\boldsymbol{v}(t), f(t))$
      else $inside \leftarrow$ *EckeInnerhalbFlächeApproximativ*$(\boldsymbol{v}, f, t)$
      if $(inside = \text{TRUE})$
      then return (COLLISION,t)
return (NO COLLISION,-1)

Abbildung 5.20: Algorithmus: Test Ecke gegen Fläche (Lineare Abtastung)

Die lineare Abstandsinterpolation ist in der Lage mögliche Kollisionszeitpunkte zu approximieren für den Fall, daß sich das Vorzeichen der gerichteten Abstandsfunktion während der Bewegung ändert. Für den *Ecke–Fläche* Test bedeutet dies, daß der Eckpunkt $\boldsymbol{v}$ während der Bewegung durch die zu der Fläche f gehörenden Ebene p_f hindurchtaucht und sich am Ende der Bewegung auf der anderen Seite der Ebene befindet. Für den *Kante–Kante* Test bedeutet dies, daß die zu den beiden Kanten e_1 und e_2 gehörenden Geraden l_1 und l_2 während der Bewegung miteinander kollidiert sind, und sich eine Kante bzgl. der durch die beiden Richtungsvektoren $\boldsymbol{u}_{l_1}$ und $\boldsymbol{u}_{l_2}$ und einem Punkt der anderen Kante definierten Ebene von der einen auf die andere Seite bewegt hat und sich am Ende der Bewegung auf der anderen Seite der Ebene befindet. In diesen Situationen wird mittels einer linearen Interpolation zwischen den Abständen zum Start- und zum Endzeitpunkt der Bewegung interpoliert und damit ein möglicher Kollisionszeitpunkt approximiert.

Für den Fall rein translatorischer Bewegungen ergibt sich eine lineare Abstandsfunk-

Algorithmus *KanteKanteLineareAbtastung*(e_1, e_2, Δ, typ)

	e_1, e_2	Kanten
	Δ	Abtastraster
Eingabe:	typ	Typ der Schnittpunkt-in-Kante Berechnung
		EXAKT = exakte Berechnung
		INTER = lineare Interpolation
Ausgabe:	c	Kollisionsflag
	t	Kollisionszeitpunkt

$t_0 \leftarrow t_1 \leftarrow 0$
$\boldsymbol{n}_0 \leftarrow \boldsymbol{n}_1 \leftarrow (\boldsymbol{v}_2(0) - \boldsymbol{v}_1(0)) \times (\boldsymbol{w}_2(0) - \boldsymbol{w}_1(0))$
if $(|\boldsymbol{n}_0| > 0)$
then $d_0 \leftarrow d_1 \leftarrow \frac{\boldsymbol{n}_0^T \cdot (\boldsymbol{v}_1(0) - \boldsymbol{w}_1(0))}{|\boldsymbol{n}_0|}$
else **return** (NO COLLISION,-1)
while $(t_1 \leq 1)$
  do $t_0 \leftarrow t_1$; $d_0 \leftarrow d_1$
    $t_1 \leftarrow t_0 + \Delta$; $\boldsymbol{n}_1 \leftarrow (\boldsymbol{v}_2(t_1) - \boldsymbol{v}_1(t_1)) \times (\boldsymbol{w}_2(t_1) - \boldsymbol{w}_1(t_1))$
    if $(|\boldsymbol{n}_1| > 0)$
    then $d_1 \leftarrow \frac{\boldsymbol{n}_1^T \cdot (\boldsymbol{v}_1(t_1) - \boldsymbol{w}_1(t_1))}{|\boldsymbol{n}_1|}$
    else **return** (NO COLLISION,-1)
    if $(d_0 \cdot d_1 \leq 0)$
    then if $(|d_1 - d_0| > \varepsilon)$
        then $t \leftarrow t_0 - d_0 \cdot \frac{t_1 - t_0}{d_1 - d_0}$
        else $t \leftarrow t_0$
        if $(typ =$EXAKT$)$
        then *inside* $\leftarrow$ *KanteInnerhalbKanteExakt*$(e_1(t), e_2(t))$
        else *inside* $\leftarrow$ *KanteInnerhalbKanteApproximativ*(e_1, e_2, t)
        if $($*inside* $=$TRUE$)$
        then return (COLLISION,t)
return (NO COLLISION,-1)

Abbildung 5.21: ALGORITHMUS: TEST KANTE GEGEN KANTE (LINEARE ABTASTUNG)

tion und die lineare Abstandsinterpolation berechnet den exakten möglichen Kollisionszeitpunkt.

Ungenauigkeiten bei der Bestimmung des möglichen Kollisionszeitpunktes ergeben sich dadurch, daß durch die lineare Interpolation der Kollisionszeitpunkt nur unzureichend angenähert werden kann, daß während der Bewegung mehr als ein möglicher Kollisionszeitpunkt vorhanden sein kann, von denen nur einer erkannt wird, und daß für den Fall, daß die Abstandsfunktion keinen Vorzeichenwechsel bei der Betrachtung des Start- und des Endzeitpunktes der Bewegung hat, keine möglichen Kollisionszeitpunkte bestimmt werden können.

Algorithmus $\mathit{EckeFl\ddot{a}cheAbtastungRegulaFalsi}(\boldsymbol{v}, f, \Delta, \mathit{typ})$

Eingabe:

$\boldsymbol{v}$	Ecke
$f = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{k-1}, \boldsymbol{v}_k \equiv \boldsymbol{v}_0)$	Fläche mit Ebene $\boldsymbol{n}_f \cdot \boldsymbol{x} - d_f = 0$
Δ	Abtastraster
typ	Typ der Ecke-in-Fäche Berechnung EXAKT = exakte Berechnung INTER = lineare Interpolation

Ausgabe:

c	Kollisionsflag
t	Kollisionszeitpunkt

$t_0 \leftarrow t_1 \leftarrow 0;\quad d_0 \leftarrow d_1 \leftarrow \boldsymbol{n}_f(0) \cdot \boldsymbol{v}(0) - d_f(0)$
while $(t_1 \leq 1)$
  do $t_0 \leftarrow t_1;\quad d_0 \leftarrow d_1$
    $t_1 \leftarrow t_0 + \Delta;\quad d_1 \leftarrow \boldsymbol{n}_f(t_1) \cdot \boldsymbol{v}(t_1) - d_f(t_1)$
    if $(d_0 \cdot d_1 \leq 0)$
    then $t_{neu} \leftarrow t_0 \quad d_{neu} \leftarrow d_0$
      while $((d_{neu} \neq 0)$ **and** $(|t_0 - t_1| > \varepsilon_1)$ **and** $(|d_0 - d_1| > \varepsilon_2))$
        do $t_{neu} \leftarrow t_0 - \frac{d_0 \cdot (t_1 - t_0)}{d_1 - d_0}$
          $d(t_{neu}) \leftarrow \boldsymbol{n}_f(t_{neu}) \cdot \boldsymbol{v}(t_{neu}) - d_f(t_{neu})$
          if $(d_0 \cdot d_{neu} \geq 0)$
          then $t_0 \leftarrow t_{neu};\quad d_0 \leftarrow d_{neu}$
          else $t_1 \leftarrow t_{neu};\quad d_1 \leftarrow d_{neu}$
      $t \leftarrow t_0$
      if $(\mathit{typ} =$EXAKT$)$
      then $\mathit{inside} \leftarrow \mathit{EckeInnerhalbFl\ddot{a}cheExakt}(\boldsymbol{v}(t), f(t))$
      else $\mathit{inside} \leftarrow \mathit{EckeInnerhalbFl\ddot{a}cheApproximativ}(\boldsymbol{v}, f, t)$
      if $(\mathit{inside} =$TRUE$)$
      then return (COLLISION,t)
return (NO COLLISION,-1)

Abbildung 5.22: ALGORITHMUS: TEST ECKE GEGEN FLÄCHE (ABSTANDSABTASTUNG UND REGULA FALSI)

2. *Iterative Abstandsinterpolation mit Regula falsi*
   Die iterative Abstandsinterpolation mit Regula falsi verbessert die Ungenauigkeit der *linearen Interpolation* bei der Approximation eines möglichen Kollisionszeitpunktes. Die Voraussetzungen zum Auffinden eines möglichen Kollisionzeitpunktes sind analog zu denen bei der *linearen Interpolation*. Durch den Einsatz eines iterativen Verfahrens zur Nullstellenbestimmung der Abstandsfunktion wird eine Nullstelle beliebig genau approximiert. Hat die Abstandsfunktion jedoch während der Bewegung mehrere Nullstellen, so wird durch das iterative Verfahren eine davon gefunden, ohne garantieren zu können, daß es die früheste ist.

Algorithmus *KanteKanteAbtastungRegulaFalsi*(e_1, e_2, Δ, typ)

Eingabe:
- e_1, e_2 Kanten
- Δ Abtastraster
- *typ* Typ der Schnittpunkt-in-Kante Berechnung
  - EXAKT = exakte Berechnung
  - INTER = lineare Interpolation

Ausgabe:
- c Kollisionsflag
- t Kollisionszeitpunkt

$t_0 \leftarrow t_1 \leftarrow 0$
$\boldsymbol{n}_0 \leftarrow \boldsymbol{n}_1 \leftarrow (\boldsymbol{v}_2(0) - \boldsymbol{v}_1(0)) \times (\boldsymbol{w}_2(0) - \boldsymbol{w}_1(0))$
if $(|\boldsymbol{n}_0| > 0)$
then $d_0 \leftarrow d_1 \leftarrow \frac{\boldsymbol{n}_0^T \cdot (\boldsymbol{v}_1(0) - \boldsymbol{w}_1(0))}{|\boldsymbol{n}_0|}$
else **return** (NO COLLISION,-1)
while $(t_1 \leq 1)$
  do $t_0 \leftarrow t_1$; $d_0 \leftarrow d_1$
    $t_1 \leftarrow t_0 + \Delta$; $\boldsymbol{n}_1 \leftarrow (\boldsymbol{v}_2(t_1) - \boldsymbol{v}_1(t_1)) \times (\boldsymbol{w}_2(t_1) - \boldsymbol{w}_1(t_1))$
    if $(|\boldsymbol{n}_1| > 0)$
    then $d_1 \leftarrow \frac{\boldsymbol{n}_1^T \cdot (\boldsymbol{v}_1(t_1) - \boldsymbol{w}_1(t_1))}{|\boldsymbol{n}_1|}$
    else **return** (NO COLLISION,-1)
    if $(d_0 \cdot d_1 \leq 0)$
    then if $(|d_1 - d_0| > \varepsilon)$
        then $t \leftarrow t_0 - d_0 \cdot \frac{t_1 - t_0}{d_1 - d_0}$
        else $t \leftarrow t_0$
        if $(typ =$EXAKT$)$
        then *inside* $\leftarrow$ *KanteInnerhalbKanteExakt*$(e_1(t), e_2(t))$
        else *inside* $\leftarrow$ *KanteInnerhalbKanteApproximativ*(e_1, e_2, t)
        if (*inside* =TRUE)
        then return (COLLISION,t)
return (NO COLLISION,-1)

Abbildung 5.23: Algorithmus: Test Kante gegen Kante (Abstandsabtastung und Regula Falsi)

Ebenso wie die *lineare Abstandsinterpolation* ist die *iterative Abstandsinterpolation mit Regula Falsi* bei rein translatorischen Bewegungen exakt und endet nach der ersten Iteration.

Ungenauigkeiten bei der Bestimmung des möglichen Kollisionszeitpunktes ergeben sich dadurch, daß während der Bewegung mehr als ein möglicher Kollisionszeitpunkt vorhanden sein kann, von denen nur einer – beliebig genau – erkannt wird und daß für den Fall, daß die Abstandsfunktion keinen Vorzeichenwechsel bei der Betrachtung des Start- und Endzeitpunktes der Bewegung hat, keine möglichen Kollisionszeitpunkte erkannt werden.

3. *Quadratische Abstandsinterpolation*
   Die quadratische Abstandsinterpolation versucht die Ungenauigkeiten der beiden linearen Abstandsinterpolationsverfahren durch die Erhöhung des Grades der Interpolationsfunktion zu verbessern. Dadurch ist sie mitunter in der Lage, mögliche Kollisionszeitpunkte zu berechnen, obwohl bei der Betrachtung des Start- und des Endzeitpunktes der Bewegung kein Vorzeichenwechsel in der Abstandsfunktion stattgefunden hat.

   Bei rein translatorischen Bewegungen ergibt sich dabei eine ebenfalls lineare Interpolationsfunktion, die den exakten möglichen Kollisionszeitpunkt berechnen kann. Bewegt sich ein Objekt rein rotatorisch, wobei während der Bewegung eine Kollision auftritt, diese am Ende der Bewegung nicht mehr vorliegt, z.B. daß ein Eckpunkt in eine Fläche eintaucht, aus ihr aber auch wieder hervorkommt, so hat dieses Verfahren die Chance, die beiden möglichen Kollisionszeitpunkte zu bestimmen.

   Ungenauigkeiten bei der Bestimmung möglicher Kollisionszeitpunkte ergeben sich dadurch, daß durch die Interpolation mögliche Kollisionszeitpunkte nur unzureichend approximiert werden und daß mögliche Kollisionszeitpunkte durch eine komplexer verlaufende Abstandsfunktion übersehen werden.

4. *Lineare Abstandsabtastung*
   Bei der linearen Abstandsabtastung wird die Abstandsfunktion in einem festen Abtastungsgitter Δ mit Stützstellen versehen, zwischen denen linear interpoliert wird. Eine beliebig kleine Wahl von Δ ergibt somit eine beliebig genaue Approximation der tatsächlichen Abstandsfunktion.

   Bei rein translatorischen Bewegungen ergibt sich durch die Linearität der tatsächlichen Abstandsfunktion für jede Wahl von Δ eine exakte Bestimmung des möglichen Kollisionszeitpunktes. Darüber hinaus werden bei einem Abtastgitter Δ keine Vorzeichenwechsel der Abstandsfunktion übersehen, die länger andauern als Δ.

   Ungenauigkeiten bei der Bestimmung möglicher Kollisionszeitpunkte ergeben sich dadurch, daß Vorzeichenwechsel der Abstandsfunktion übersehen werden können, die kürzer als Δ andauern und daß durch die lineare Interpolation zwischen zwei Stützstellen ein möglicher Kollisionszeitpunkt nur unzureichend approximiert wird.

5. *Abstandsabtastung und Regula falsi*
   Bei der iterativen Abstandsabtastung wird die *lineare Abstandsabtastung* mit der iterativen Bestimmung von Nullstellen verknüpft. Analog zur *linearen Abstandsabtastung* ergibt sich eine beliebig genaue Approximation der tatsächlichen Abstandsfunktion zusammen mit den verbesserten Approximationseigenschaften der iterativen Nullstellenbestimmung.

   Bei rein translatorischen Bewegungen ergibt sich somit eine exakte Bestimmung des möglichen Kollisionszeitpunktes für jede Wahl von Δ. Darüber hinaus werden bei einem Abtastgitter Δ keine Vorzeichenwechsel der Abstandsfunktion übersehen, die länger andauern als Δ.

Ungenauigkeiten bei der Bestimmung möglicher Kollisionszeipunkte ergeben sich dadurch, daß Vorzeichenwechsel der Abstandsfunktion übersehen werden können, die kürzer als Δ andauern und daß durch die iterative Interpolation zwischen zwei Stützstellen beim Vorliegen mehrerer Nullstellen im betrachteten Intervall der Länge Δ nur eine der Nullstellen beliebig genau approximiert wird.

5.4 Gemischt statisch-dynamische Kollisionserkennung

Um über ein Verfahren zu verfügen, das in seiner Aussagekraft und Performance zwischen der *statischen* und der *dynamischen* Kollisionserkennung liegt, können beide Verfahren miteinander kombiniert werden. Dies macht insbesondere dann Sinn, wenn die Basiskollisionserkennung im Zusammenhang mit der *hierarchischen Kollisionserkennung* (vgl. Kapitel 3, 4) verwendet wird und für die Hüllkörper in der Hüllkörperhierarchie der *statische* Kollisionstest zur Anwendung kommt. Werden dabei an den Blättern der Hierarchie die einzelnen Objektteile mittels der *Basiskollisionserkennung* auf Kollision getestet, so kann in einem ersten Schritt die *statische* Kollisionserkennung dazu benutzt werden zu testen, ob zum Endpunkt der Bewegung eine statische Überlappung der Objektteile vorliegt. Nur in diesem Fall wird in einem zweiten Schritt eines der Verfahren der *dynamischen* Kollisionserkennung benutzt, um den Kollisionszeitpunkt und die dabei beteiligten Objektteile näher zu spezifizieren.

Daraus ergibt sich der Algorithmus *StatischDynamischerKollisionstest* in Abbildung 5.24.

5.5 Zusammenfassung und Vergleich der Basiskollisionserkennungsverfahren

Statische Basiskollisionserkennungsverfahren sind in der Lage zu entscheiden, ob eine Konfiguration von Objekten kollisionsfrei ist oder nicht und, falls sie nicht kollisionsfrei ist, die Menge der Objektteile zu spezifizieren, die sich durchdringen. Angewendet auf Bewegungen kann somit getestet werden, ob die Konfiguration zweier Objekte am Ende einer Bewegung kollisionsfrei ist oder nicht. Ist sie kollisionsfrei, so wird die gesamte Bewegung als kollisionsfrei klassifiziert, unabhängig davon, ob während der Bewegung eine Kollision auftritt oder nicht. Auf diese Art und Weise können keine Kollisionen erkannt werden, die während einer Bewegung auftauchen, jedoch die Zielpositionierung kollisionsfrei ist. Dies ist beispielsweise der Fall, wenn ein Objekt in einer Bewegung vollkommen durch ein anderes hindurchbewegt wird.

Diese Art der Basiskollisionserkennung ist der Standard in fast allen Kollisionserkennungsverfahren mit Hüllkörperhierarchien. Die dadurch berechneten Resultate sind jedoch nicht dazu geeignet, in weitergehenden Simulationen, wie z.B. *Dynamiksimulationen*, eingesetzt zu werden. Nicht zuletzt auch deshalb existiert bisher kaum ein Simulationsmodul, das mit einer schnellen hierarchischen Kollisionserkennung zusammenarbeitet und damit für die Behandlung auch von Szenarien mit komplexen Objekten geeignet ist.

Dynamische Kollisionserkennungsverfahren versuchen nun, früheste Kollisionszeitpunkte zusammen mit den dabei beteiligten Objektteilen zu bestimmen, damit das Ergebnis der Kollisionserkennung zu verfeinern und für weitergehende Simulationen nutzbar zu machen. Insbesondere muß der Einsatz *dynamischer* Kollisionserkennungsverfahren auch bei der Benutzung weiterer Kompenenten des gesamten Kollisionserkennungsprozesses, wie *Raumpartitionierung* (vgl. Kapitel 6) und *Verwendung von Hüllkörperhierarchien* (vgl. Kapitel 4) berücksichtigt werden.

Ein Kompromiß zwischen *statischen* und *dynamischen* Kollisionserkennungsverfahren stellen die *gemischt statisch-dynamischen* Verfahren dar. Sie erkennen keine Kollision, falls in der Zielposition der Objekte keine Kollision vorliegt, d.h. auch sie übersehen die Kollision, die auftritt, wenn sich ein Objekt in einem Schritt durch ein anderes hindurchbewegt. Ist jedoch die Zielposition nicht kollisionsfrei, versuchen sie, mit Hilfe *dynamischer* Verfahren den frühesten Kollisionszeitpunkt und die dabei beteiligten Objektteile zu approximieren. Eine zusätzliche Ungenauigkeit zu denen der verwendeten Verfahren der *dynamischen* Kollisionserkennung besteht jedoch darin, daß die in der Zielposition statisch kollidierenden Objektteile nicht notwendigerweise diejenigen sind, die den frühesten Kollisionszeitpunkt bestimmen. Da jedoch nicht lokal entschieden werden kann, welche Objektteile beim Vorliegen einer statischen Kollision in der Zielpositionierung den frühesten Kollisionszeitpunkt bestimmen, werden heuristisch die statisch kollidierenden Objektteile zur Bestimmung des frühesten Kollisionszeitpunkt herangezogen. Da *statische* Kollisionserkennungsverfahren benutzt werden, um die *dynamischen* anzustoßen, können in den weiteren Kompenenten des gesamten Kollisionserkennungsprozesses, wie *Raumpartitionierung* und *Benutzung von Hüllkörperhierarchien* die bekannten *statischen* Verfahren benutzt werden.

Algorithmus $StatischDynamischerKollisionstest(F_1, F_2, typ_1, typ_2)$

Eingabe:

$F_1 = \{f_0, \ldots, f_{n_1}\}$	Flächenmenge 1
$F_2 = \{g_0, \ldots, g_{n_2}\}$	Flächenmenge 1
typ_1	Typ der dynamischen Kollisionserkennung
typ_2	Typ der Innerhalb-außerhalb Berechnung

Ausgabe:

c	Kollisionsflag
t	Kollisionszeitpunkt
o_1	Objektteil 1 bei der Kollision
o_2	Objektteil 2 bei der Kollision

$eFM \leftarrow \emptyset$; $kKM \leftarrow \emptyset$
for e **in** $E(F_1)$
do for f **in** F_2
  do if $(KanteFl\ddot{a}cheStatisch(e, f) = \text{COLLISION})$
    then $eFM \leftarrow eFM \cup \{(start(e), f), (end(e), f)\}$
      for d **in** $E(f)$
      do $kKM \leftarrow kKM \cup \{(e, d)\}$
for e **in** $E(F_2)$
do for f **in** F_1
  do if $(KanteFl\ddot{a}cheStatisch(e, f) = \text{COLLISION})$
    then $eFM \leftarrow eFM \cup \{(start(e), f), (end(e), f)\}$
      for d **in** $E(f)$
      do $kKM \leftarrow kKM \cup \{(e, d)\}$
$t \leftarrow \infty$
for $(\boldsymbol{v}, f)$ **in** eFM
do $(c_{akt}, t_{akt}) \leftarrow EckeFl\ddot{a}cheDynamisch(\boldsymbol{v}, f, typ_1, typ_2)$
  if $((c_{akt} = \text{COLLISION})$ **and** $(t_{akt} < t))$
  then $t \leftarrow t_{akt}$
    $o_1 \leftarrow \boldsymbol{v}$
    $o_2 \leftarrow \boldsymbol{f}$
for (e_1, e_2) **in** kKM
do $(c_{akt}, t_{akt}) \leftarrow KanteKanteDynamisch(e_1, e_2, typ_1, typ_2)$
  if $((c_{akt} = \text{COLLISION})$ **and** $(t_{akt} < t))$
  then $t \leftarrow t_{akt}$
    $o_1 \leftarrow e_1$
    $o_2 \leftarrow e_2$
if $(t < \infty)$
then return (COLLISION,t, o_1, o_2)
else return (NO COLLISION,-1)

Abbildung 5.24: ALGORITHMUS: TEST FLÄCHENMENGE GEGEN FLÄCHENMENGE (STATISCH-DYNAMISCH)

Kapitel 6

Raumpartitionierung

In den Kapiteln 3, 4 und 5 haben wir uns damit beschäftigt, Verfahren zu entwickeln, die einen Objektpaartest im Kollisionserkennungsprozeß sowohl mit *statischer* als auch mit *dynamischer* Kollisionserkennung schnell durchführen können. Diese Verfahren beschäftigen sich damit, die geometrische Komplexität einzelner Objekte in der Kollisionserkennung zu beherrschen.

Eine zweite Art der Komplexität, die die Kollisionserkennung beherrschen muß, ist die Behandlung einer großen Anzahl von Objekten in der Kollisionserkennung. Besteht eine Szene aus n bewegten und m festen Objekten, so müssen $\binom{n}{2} + n \cdot m$ Objektpaartests durchgeführt werden. Ist die Anzahl der bewegten und festen Objekte groß, müssen Verfahren eingesetzt werden, die die Anzahl der tatsächlich durchzuführenden Objektpaartests reduzieren.

Dazu geeignet sind klassische Raumpartitionierungsverfahren wie *Oktrees*, *BSP-Trees* und *uniforme Raumunterteilungen*. Daneben gibt es Verfahren wie *Space-Time-Bounds*, die aufgrund von aktueller Geschwindigkeit und maximaler Beschleunigung der Objekte den frühestmöglichen zukünftigen Kollisionszeitpunkt abschätzen, und *Sweep-And-Prune*-Techniken, die durch Sortieren die Anzahl der Objektpaartests reduzieren. Einige dieser Verfahren werden wir in der Folge näher erläutern.

6.1 Bisherige Arbeiten

6.1.1 Uniforme Raumunterteilung

Eine erste Möglichkeit die räumliche Anordnung der Objekte in einer Szene zu repräsentieren ist die *uniforme Raumunterteilung.* Dabei wird die Szene durch ein Gitter fester Größe in Zellen oder *Voxel (Volume Pixel)* unterteilt. Jedes Objekt der Szene wird jeder Zelle zugeordnet, die es schneidet. Zyda et. al. beschreiben eine solche Raumpartitionierung in [ZO+93].

Sind die meisten in einer Szene befindlichen Objekte von einheitlicher Größe und *fat*, so ist es möglich, eine geeignete Gitterweite zu berechnen mit Hilfe derer der Szenenraum unterteilt werden sollte. Diese Voraussetzung ist z.B. bei der Simulation des Vorratsbehälters eines Part Feeders oder von Molekülen (vgl. [Tu90]) gegeben. Overmars zeigt in [Ov92], daß mit Hilfe einer Hash-Tabelle eine Datenstruktur der Größe $O(n)$ benutzt werden kann, um Lokalisierungsanfragen in konstanter Zeit zu beantworten. Problematisch wird die *uniforme Raumunterteilung* dann, wenn die Objekte in der Szene unterschiedliche Größe haben. Dann ist es schwierig, eine geeignete Gitterweite zu bestimmen.

6.1.2 Oktree

Eine flexiblere Raumunterteilung als die *uniforme* stellt der *Oktree* dar. Er besteht aus einer Hierarchie von achsenorientierten Boxen, wobei jede Stufe der Hierarchie den gesamten zu unterteilenden Raum partitioniert, der Objekte enthält. An der Wurzel des Oktrees befindet sich eine Box, die alle Objekte in der Szene enthält. Diese Box wird in der nächsten Stufe aufgeteilt in acht Boxen, indem jede Kante der Box unterteilt wird. In den verschiedenen Varianten wird diese Unterteilung unterschiedlich gehandhabt. Denkbar dabei sind z.B.

- Halbierung der Kantenlänge der Box oder
- Halbierung der Anzahl der Objekte in jeder Halbbox.

Von den entstehenden acht Unterboxen müssen nur diejenigen behandelt werden, in denen tatsächlich Objekte enthalten sind. Die Unterteilung einer Box endet, wenn nur noch eine kleine Anzahl von Objekten in einer Box enthalten ist.

Der Vorteil gegenüber der *uniformen Raumunterteilung* liegt darin, daß die Zellengröße adaptiv an die Größe der Objekte in der Szene angepaßt wird. Durch die komplexere Struktur erhöht sich jedoch auch der Aufwand der dynamischen Aktualisierung, wenn Objekte in der Szene bewegt werden. Dieser Raumunterteilungsansatz wird z.B. in [Ha88, MW88, SH92] im Zusammenhang mit Kollisionserkennung benutzt.

Moore und Wilhelms gehen in [MW88] bei der Benutzung von *Oktrees* noch ein Stück weiter, indem sie sie nicht nur zur Raumpartitionierung benutzen, sondern auch zur hierarchischen Approximation von Objekten in der Szene selbst. Dazu werden in jeder Zelle des *Oktrees* nur die Objektteile gespeichert, die tatsächlich darin enthalten sind, anstatt des gesamten Objektes.

6.1.3 BSP- und MSP-Tree

Eine Methode zur Raumpartitionierung, die von Fuchs et.al. [FKN80] für das *hidden surface removal* während des Polygonrenderings entwickelt worden ist, stellt der *Binary Space Partitioning Tree (BSP-Tree)* dar. Dies ist ein binärer Baum, in dem bei jedem Knoten eine Ebene gespeichert wird, die den Raum unterteilt, sowie eine Menge von Flächen. Im Wurzelknoten sind alle Flächen gespeichert. Als unterteilende Ebene wird die Ebene einer Fläche gewählt und die Flächen werden den Halbräumen zugeordnet, in denen sie liegen. Flächen, die in beiden Halbräumen liegen, werden in zwei Flächen unterteilt, die jeweils nur noch in einem Halbraum liegen. Diese Aufteilung wird rekursiv fortgesetzt, bis in jeder Flächenmenge nur noch eine Fläche enthalten ist.

Thibault und Naylor [TN87] benutzen eine Variante des *BSP-Trees* um Boolsche Mengenoperationen für zwei Polyeder durchzuführen. Dabei werden die Flächen, die in beiden Halbräumen liegen, nicht unterteilt, sondern beiden Halbräumen zugeordnet.

Benutzt man als unterteilende Ebene eine Ebene parallel zu einer der Koordinatenachsen, so ergibt sich aus einem *BSP-Tree* eine Art *Oktree*, bei dem in jedem Schritt die Box nur durch eine Schnittebene unterteilt wird. Diese Vorgehensweise benutzen Felger und Zachmann in [FZ95, Za97] zur Approximation von Objekten.

Vaněček [Va91] erweitert *BSP-Trees* zu *Multidimensional Space Partitioning Trees (MSP-Tree)*, die er auch *BRrep-Index* nennt. Bouma und Vaněček [BV91] benutzen diese Datenstruktur zur Kollisionserkennung.

6.1.4 1D- und 2D-Sweep-and-Prune

Lin [Li93] und Baraff et.al. [BWK95] beschreiben andere Methoden zur Behandlung von Szenen mit vielen Objekten. Die dahinterliegende Idee ist, daß bei zwei Objekten, die sich überlappen, dies auch für ihre Projektionen in niedrigere Dimensionen der Fall ist.

Dies gilt für Projektionen auf Koordinatenachsen (eindimensionaler Fall) ebenso wie für die Projektion in Koordinatenebenen (x-y, x-z und y-z, zweidimensionaler Fall). Dies kann genutzt werden, um die Anzahl der Objektpaartests in der Kollisionserkennung zu reduzieren.

Überlappen sich zwei Objekte, so gilt dies auch für jedes einhüllende Objekt, wie z.B. eine achsenorientierte Box. Achsenorientierte Boxen sind deshalb an dieser Stelle besonders geeignet, weil ihre Projektion in die Koordinatenebenen bzw. die Koordinatenachsen direkt aus der Beschreibung abgelesen werden kann. Dies wird genutzt, um mit Hilfe der Projektionen achsenorientierter Boxen in niedrigere Dimensionen aus einer Menge von n Objekten, diejenigen schnell zu identifizieren, die ein gegebenes Objekt überlappen können.

Berechnen wir für ein Objekt eine einhüllende achsenorientierte Box $B^{Iso} = (\boldsymbol{b}^{min}, \boldsymbol{b}^{max})$ und bewegen das Objekt mittels einer Transformation $\boldsymbol{T} = (\boldsymbol{R}, \boldsymbol{t})$, ergibt sich aus der

Iso-Box eine beliebig orientierte Box $B = (\boldsymbol{R}, \boldsymbol{T} \cdot (\frac{1}{2} \cdot (\boldsymbol{b}^{max} + \boldsymbol{b}^{min})), \frac{1}{2} \cdot (\boldsymbol{b}^{max} - \boldsymbol{b}^{min}))$, die auf die Koordinatenachsen projiziert wird, um die umhüllende Iso-Box für diese Box B zu berechnen.

Das *1D-* und *2D-Sweep-and-Prune* stellen keine klassischen *Raumpartitionierungsverfahren* dar, da sie nicht eine Repräsentation des Raumes beinhalten, sondern eine simultane Repräsentation der Objekte in einer gemeinsamen Datenstruktur. Dadurch haben sie gegenüber der *uniformen Raumunterteilung* den Vorteil, daß keine optimale Zellengröße berechnet werden muß bzw. die Wahl der Zellengröße die Performance der Datenstruktur bestimmt. *Oktrees* haben ein ähnliches Problem, weil sie die minimale Oktantengröße entweder a priori bestimmen oder dynamisch verändern müssen. Diese Problematik entsteht hier nicht, da für die Datenstruktur nur die Hüllkörper der Objekte und deren Projektionen bestimmt werden. Die Qualität der Ergebnisse des *1D-* und *2D-Sweep-and-Prunes* für die Überlappung von Objekten unterscheiden sich nicht. Das *1D-Sweep-and-Prune* ist einfacher in der Handhabung, wohingegen das *2D-Sweep-and-Prune* aufwendigere Datenstrukturen benötigt. Dafür sind beim *2D-Sweep-and-Prune* nur die Projektionen entlang zwei der drei Koordinatenachsen zu behandeln, wohingegen beim *1D-Sweep-and-Prune* alle drei Koordinatenachsen behandelt werden müssen.

2D-Sweep-and-Prune

Die Projektion einer achsenorientierten Box entlang einer der drei Koordinatenachsen ergibt ein achsenorientiertes Rechteck. Um effizient die 2D-Schnitte zwischen den achsenorientierten rechteckigen Projektionen zu berechnen, schlägt LIN als Datenstruktur einen *Intervallbaum* vor. Alternativ dazu kann auch ein *Segmentbaum* oder ein *Priority Suchbaum* verwendet werden (vgl. [Sa90]). Die Vorgehensweise ist in Abbildung 6.1 graphisch veranschaulicht.

Die Menge aller Objekte, deren zweidimensionale Projektion der zugehörigen achsenorientierten Box in zwei Koordinatenebenen mit der des Anfrageobjektes überlappen, können das Anfrageobjekt selbst überlappen.

1D-Sweep-and-Prune

Die Projektion einer achsenorientierten Box auf eine Koordinatenachse ist ein Intervall. Die Menge aller Objekte, deren eindimensionale Projektion der zugehörigen achsenorientierten Box in allen drei Koordinatenachsen mit der des Anfrageobjektes überlappen, können das Anfrageobjekt selbst überlappen. Die Vorgehensweise ist in Abbildung 6.2 graphisch veranschaulicht.

Als Datenstruktur für jede Koordinatenachse des *1D-Sweep-and-Prune* benutzen wir eine sortierte doppelt verkettete Liste von Intervallgrenzen, wobei die Intervallgrenzen in der Liste mit den zugehörigen Intervallen assoziiert sind, zu denen in jeder Situtation die Menge der überlappenden Intervalle gespeichert wird.

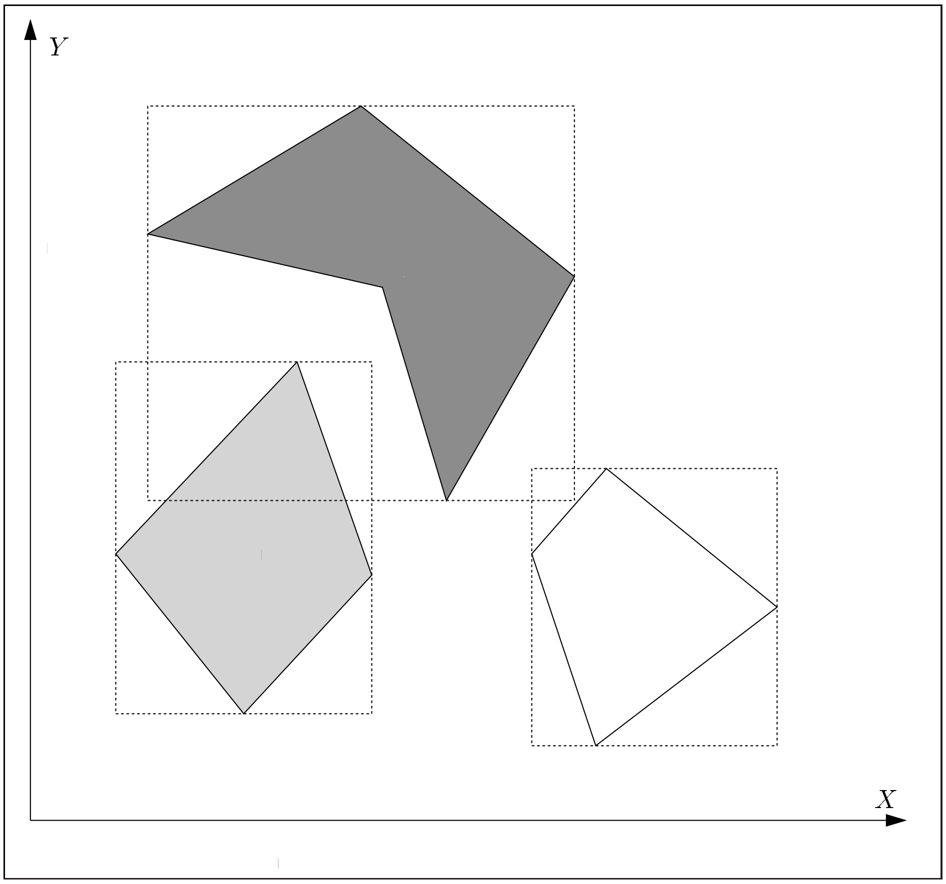

Abbildung 6.1: 2D-SWEEP-AND-PRUNE

6.2 One-dimensional Sweep-and-Prune

Wir haben uns zur Verwendung des von LIN in [Li93] vorgeschlagenen *one-dimensional Sweep-and-Sort* entschieden. Generell haben die *Sweep-and-Prune* Techniken gegenüber den klassischen Raumpartitionierungsverfahren den Vorteil, daß sie keine a priori Größe des Raumgitters wählen oder diese dynamisch während der Simulation adaptieren müssen. Darüber hinaus eignen sie sich sowohl zum Einsatz mit der *statischen* Kollisionserkennung als auch zur Erweiterung auf die *dynamische* Kollisionserkennung. Das Grundprinzip des *Sweep-and-Prune* von LIN ist, daß für jedes Objekt in der Szene dessen umhüllende Iso-Box berechnet wird und die Iso-Boxen aller Objekte in einer gemeinsamen Datenstruktur verwaltet werden. Die benutzte Iso-Box kann dabei die in einer festen Position berechnete sein (*statisch*) oder eine, die das überstrichene Volumen des Objektes während einer Bewegung approximiert (*dynamisch*). Die Abläufe zur Bestimmung von Objekten, die mit dem Anfrageobjekt möglicherweise kollidieren, unterscheiden sich im *dynamischen* Fall nicht von der *statischen* Vorgehensweise.

Zur Approximation des überstrichenen Volumens eines Objektes benutzen wir nicht das

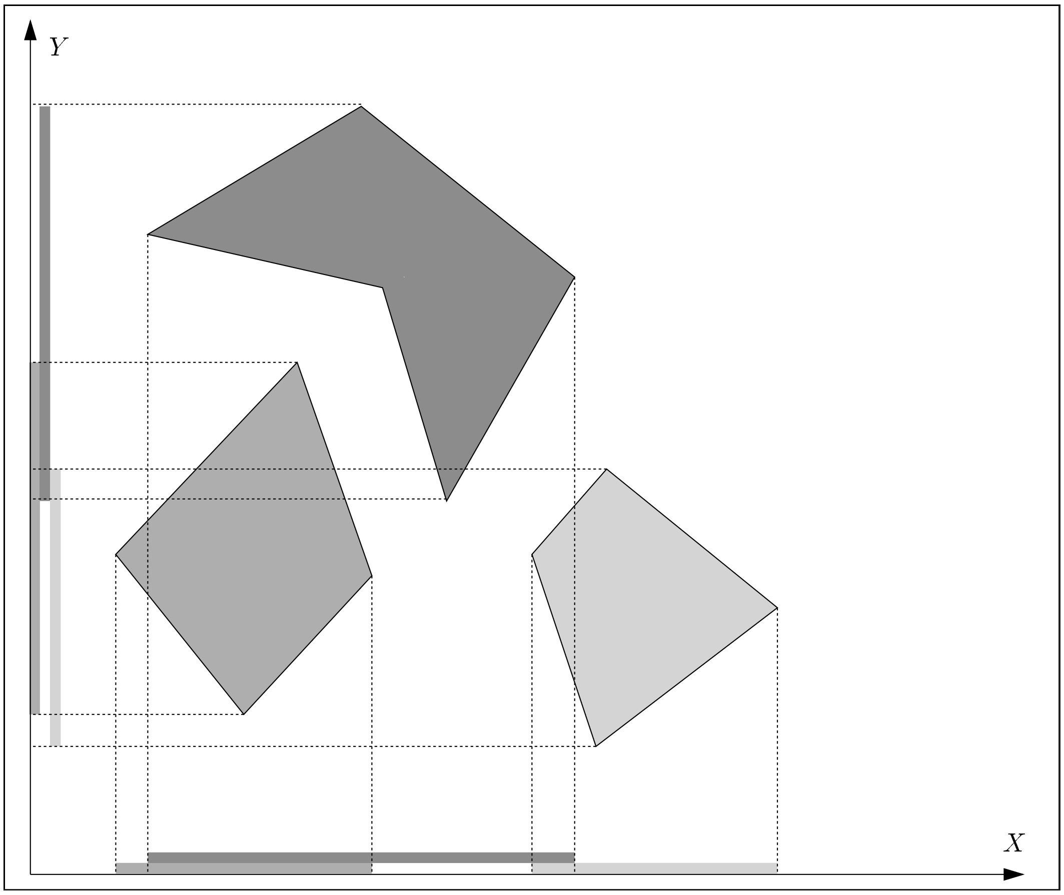

Abbildung 6.2: 1D-Sweep-and-Prune

Objekt selbst, sondern einen Hüllkörper für das gesamte Objekt, z.B. eine Iso-Box. Die Approximation des überstrichenen Volumens erfolgt mit den Methoden, die in Abschnitt 4.2.1 für die *dynamische* Kollisionserkennung für Hüllkörper entwickelt wurden.

Bei uns kommt das *one-dimensional Sweep-and-Prune* zur Anwendung, da in unseren Szenarien, z.B. Einbauuntersuchungen, der größte Teil der Objekte in der Szene fest sind und nur ein kleiner Teil bewegt. In diesem Zusammenhang zeigt das *one-dimensional Sweep-and-Prune* eine gute Performance; das *two-dimensional Sweep-and-Sort* hat größere Stärken in Szenarien, bei denen viele Objekte gleichzeitig bewegt werden, z.B. bei der Simulation von Part Feedern.

In der Folge werden wir nun die drei Operationen im Zusammenhang mit dem *one-dimensional Sweep-and-Prune* näher erläutern:

1. Aufbau der Datenstruktur
2. Modifikation eines Objektes

3. Überlappungsanfrage für ein Objekt

Dabei gehen wir davon aus, daß für jedes Objekt O eine Iso-Box $B^{Iso}(O)$ berechnet ist, aus der sich die drei Intervalle $I_X(O) = [\boldsymbol{b}_X^{min}, \boldsymbol{b}_X^{max}]$, $I_Y(O) = [\boldsymbol{b}_Y^{min}, \boldsymbol{b}_Y^{max}]$ und $I_Z(O) = [\boldsymbol{b}_Z^{min}, \boldsymbol{b}_Z^{max}]$ ergeben.

6.2.1 Aufbau der Datenstruktur

Algorithmus *Initialisierung1DSweepAndPrune*(M_O)
Eingabe: M_O Menge der Objekte
Ausgabe: S 1D-Raumpartionierungsdatenstruktur
1. $M_{I_X} \leftarrow \emptyset$
2. $M_{I_Y} \leftarrow \emptyset$
3. $M_{I_Z} \leftarrow \emptyset$
4. **for** O **in** M_O
5. **do** $M_{I_X} \leftarrow M_{I_X} \cup \{I_X(O)\}$
6. $\quad M_{I_Y} \leftarrow M_{I_Y} \cup \{I_Y(O)\}$
7. $\quad M_{I_Z} \leftarrow M_{I_Z} \cup \{I_Z(O)\}$
8. $Liste_X(S) \leftarrow$ *InitialisierungIntervalliste*(M_{I_X})
9. $Liste_Y(S) \leftarrow$ *InitialisierungIntervalliste*(M_{I_Y})
10. $Liste_Z(S) \leftarrow$ *InitialisierungIntervalliste*(M_{I_Z})
11. **return** S

Abbildung 6.3: ALGORITHMUS: INITIALISIERUNG DER 1D-SWEEP-AND-PRUNE DATENSTRUKTUR

Zunächst fügen wir die Menge der Anfangs- und Endpunkte der Intervalle in die doppelt verkettete Liste und sortieren diese in aufsteigender Reihenfolge der Intervallgrenzen. Nach der Sortierung traversieren wir die Liste und speichern zu jeder Position in der Liste die Menge der gerade offenen, d.h. überlappenden Intervalle. Diese Menge kann einfach modifiziert werden, denn treffen wir in der Liste auf eine untere Intervallgrenze, wird das zugehörige Intervall in die Menge der offenen Intervalle aufgenommen; treffen wir auf eine obere Intervallgrenze, wird das Intervall aus der Menge der aktuell offenen Intervalle gestrichen. Öffnet sich ein neues Intervall, wird für dieses die Menge der aktuell offenen Intervalle als initiale Menge von überlappenden Intervallen übernommen und gleichzeitig das neue Intervall den überlappenden Intervallen aller derzeit offenen Intervallen hinzugefügt. Die Vorgehensweise startet am Anfang der Liste mit einer leeren Menge von offenen Intervallen. Nach einem Lauf über die sortierte Liste ist für jedes Intervall die Menge der aktuell überlappenden Intervalle bekannt. Daraus ergibt sich der Algorithmus *Initialisierung1DSweepAndPrune* in Abbildung 6.3, der die Prozedur *InitialisierungIntervalliste* in Abbildung 6.4 benutzt.

Algorithmus *InitialisierungIntervalliste*(S_I)
Eingabe: S_I Menge der Intervalle
Ausgabe: L_I Aufbereitete Intervalliste
for I **in** S_I
do $L_I \leftarrow L_I \cup I_{unten}$
    $L_I \leftarrow L_I \cup I_{oben}$
$Sortiere(L_I)$
$M_{offen} \leftarrow \emptyset$ (∗ Menge der offenen Intervalle ∗)
for l **in** L_I
do if $(Type(l) ==$UNTEN$)$
    then $M_{\text{ü}}(I_l) \leftarrow M_{offen}$
        for I **in** M_{offen}
        do $M_{\text{ü}}(I) \leftarrow M_{\text{ü}}(I) \cup \{I_l\}$
        $M_{offen} \leftarrow M_{offen} \cup \{I_l\}$
    else $M_{offen} \leftarrow M_{offen} \setminus \{I_l\}$
return L_I

Abbildung 6.4: Prozedur: Initialisierung einer Intervalliste (1D-Sweep-and-Prune)

6.2.2 Modifikation eines Objektes

Algorithmus *ModifikationObjektPosition1DSweepAndPrune*$(S, O, \boldsymbol{T}_{start}, \boldsymbol{T}_{Ziel})$
Eingabe:
S 1D-Sweep-and-Prune Datenstruktur
O zu modifizierendes Objekt
$\boldsymbol{T}_{Start}$ Startposition von O
$\boldsymbol{T}_{Ziel}$ Zielposition von O
Ausgabe: S Modifizierte 1D-Sweep-and-Prune Datenstruktur
$b \leftarrow B^{Iso}(O, \boldsymbol{T}_{Start}, \boldsymbol{T}_{Ziel})$
$L_X(S) \leftarrow ModifikationIntervalliste(L_X(S), I(O), b_X^{min}, b_X^{max})$
$L_Y(S) \leftarrow ModifikationIntervalliste(L_Y(S), I(O), b_Y^{min}, b_Y^{max})$
$L_Z(S) \leftarrow ModifikationIntervalliste(L_Z(S), I(O), b_Z^{min}, b_Z^{max})$
return S

Abbildung 6.5: Prozedur: Modifikation Objektposition (1D-Sweep-and-Prune)

Wird ein Objekt bewegt, so bewegt sich das zugehörige Projektionsintervall in der Liste. Gehen wir davon aus, daß die Bewegungen der Objekte klein sind, unterscheiden sich die aktuelle und die modifizierte Liste ebenfalls wenig. Daher ist es sinnvoll, *Sortieren durch Einfügen* zu verwenden, um die beiden modifizierten Intervallgrenzen ausgehend von der aktuellen Position an die neue zu verschieben und gleichzeitig die Inter-

Algorithmus *ModifikationIntervalliste*$(L_I, I, u_{neu}, o_{neu})$

Eingabe:	L_I	zu modifizierende Intervalliste
	I	zu modifizierendes Intervall
	u_{neu}	neue untere Intervallgrenze
	o_{neu}	neue obere Intervallgrenze
Ausgabe:	L_I	Modifizierte Intervalliste

if $(o_{neu} < I_u)$
then $ModifikationUntereIntervallgrenze(L_I, I, u_{neu})$
    $ModifikationObereIntervallgrenze(L_I, I, o_{neu})$
else $ModifikationObereIntervallgrenze(L_I, I, o_{neu})$
    $ModifikationUntereIntervallgrenze(L_I, I, u_{neu})$
return L_I

Abbildung 6.6: Prozedur: Modifikation Intervalliste (1D-Sweep-and-Prune)

Algorithmus *ModifikationUntereIntervallgrenze*(L_I, I, g_{neu})

Eingabe:	L_I	zu modifizierende Intervalliste
	I	zu modifizierendes Intervall
	g_{neu}	neue untere Intervallgrenze
Ausgabe:	L_I	Modifizierte Intervalliste

if $(g_{neu} < I_u nten)$
then $g_{akt} \leftarrow L_I.Vorgänger(I_u nten)$
    while $(g_{neu} < g_{akt})$
     do if $(Typ(g_{akt}) = \text{OBEN})$
       then $M_{\text{ü}}(I) \leftarrow M_{\text{ü}}(I) \cup I_{g_{akt}}$
         $M_{\text{ü}}(I_{g_{akt}}) \leftarrow M_{\text{ü}}(I_{g_{akt}}) \cup \{I\}$
         $L_I.tausche(I_u nten, g_{akt})$
         $g_{akt} \leftarrow L_I.Vorgänger(I_u nten)$
else $g_{akt} \leftarrow L_I.Nachfolger(I_u nten)$
    while $(g_{neu} > g_{akt})$
     do if $(Typ(g_{akt}) = \text{OBEN})$
       then $M_{\text{ü}}(I) \leftarrow M_{\text{ü}}(I) \setminus I_{g_{akt}}$
         $M_{\text{ü}}(I_{g_{akt}}) \leftarrow M_{\text{ü}}(I_{g_{akt}}) \setminus \{I\}$
         $L_I.tausche(I_u nten, g_{akt})$
         $g_{akt} \leftarrow L_I.Nachfolger(I_u nten)$
$I_u nten \leftarrow g_{neu}$
return L_I

Abbildung 6.7: Prozedur: Modifikation untere Intervallgrenze (1D-Sweep-and-Prune)

Algorithmus *ModifikationObereIntervallgrenze*(L_I, I, g_{neu})

	L_I	zu modifizierende Intervalliste
Eingabe:	I	zu modifizierendes Intervall
	g_{neu}	neue obere Intervallgrenze
Ausgabe:	L_I	Modifizierte Intervalliste

if $(g_{neu} < I_oben)$
then $g_{akt} \leftarrow L_I.Vorgänger(I_oben)$
  while $(g_{neu} < g_{akt})$
   do if $(Typ(g_{akt}) = \text{UNTEN})$
    then $M_{\text{ü}}(I) \leftarrow M_{\text{ü}}(I) \setminus I_{g_{akt}}$
     $M_{\text{ü}}(I_{g_{akt}}) \leftarrow M_{\text{ü}}(I_{g_{akt}}) \setminus \{I\}$
     $L_I.tausche(I_oben, g_{akt})$
     $g_{akt} \leftarrow L_I.Vorgänger(I_oben)$
else $g_{akt} \leftarrow L_I.Nachfolger(I_oben)$
  while $(g_{neu} > g_{akt})$
   do if $(Typ(g_{akt}) = \text{UNTEN})$
    then $M_{\text{ü}}(I) \leftarrow M_{\text{ü}}(I) \cup I_{g_{akt}}$
     $M_{\text{ü}}(I_{g_{akt}}) \leftarrow M_{\text{ü}}(I_{g_{akt}}) \cup \{I\}$
     $L_I.tausche(I_oben, g_{akt})$
     $g_{akt} \leftarrow L_I.Nachfolger(I_oben)$
$I_oben \leftarrow g_{neu}$
return L_I

Abbildung 6.8: Prozedur: Modifikation obere Intervallgrenze (1D-Sweep-and-Prune)

vallüberlappungsstruktur zu modifizieren.

Dazu werden nacheinander die obere und die untere Intervallgrenze in der Liste verschoben und zwar derart, daß in jeder Situation während der Verschiebung ein Intervall durch die beiden Intervallgrenzen beschrieben wird, d.h. die obere Intervallgrenze darf die untere nicht nach unten überschreiten. Folgende Modifikationen der Intervallüberlappungsstruktur sind beim Austausch zweier Intervallgrenzen in der Liste vorzunehmen:

1. Untere Schranke

   (a) Überschreitung einer oberen Intervallgrenze nach oben
   Die zugehörigen Intervalle überlappen sich nun nicht mehr, d.h. die Intervalle sind gegenseitig aus der Menge der überlappenden Intervalle zu löschen.

   (b) Überschreitung einer oberen Intervallgrenze nach unten
   Die zugehörigen Intervalle überlappen sich, d.h. die Intervalle sind gegenseitig in die Menge der überlappenden Intervalle einzufügen.

2. Obere Intervallgrenze

   (a) Überschreitung einer unteren Intervallgrenze nach oben
   Die zugehörigen Intervalle überlappen sich, d.h. die Intervalle sind gegenseitig in die Menge der überlappenden Intervalle einzufügen.

   (b) Überschreitung einer unteren Intervallgrenze nach unten
   Die zugehörigen Intervalle überlappen sich nun nicht mehr, d.h. die Intervalle sind gegenseitig aus der Menge der überlappenden Intervalle zu löschen.

Daraus ergibt sich der Algorithmus *ModifikationObjektPosition1DSweepAndPrune* in Abbildung 6.5, der die Prozeduren *ModifikationIntervalliste* in Abbildung 6.6, *ModifikationUntereIntervallgrenze* in Abbildung 6.7 und *ModifikationObereIntervallgrenze* in Abbildung 6.8 benutzt.

6.2.3 Überlappungsanfrage für ein Objekt

Algorithmus *ÜberlappungsanfrageObjekt*(O)
Eingabe: O Anfrageobjekt
Ausgabe: $O_{\text{ü}}$ Menge der überlappenden Objekte
1. **return** $Objekte(I_X(O)) \cap Objekte(I_Y(O)) \cap Objekte(I_Z(O))$

Abbildung 6.9: ALGORITHMUS: ANFRAGE ÜBERLAPPENDER OBJEKTE (1D-SWEEP-AND-PRUNE)

Will man für ein in der Datenstruktur befindliches Objekt die Menge der überlappenden Objekte bestimmen, muß nur die Menge der zu den überlappenden Intervallen gehörenden Objekte zurückgegeben werden. Daraus ergibt sich der einfache Algorithmus *ÜberlappungsanfrageObjekt* in Abbildung 6.9.

Kapitel 7

Steuerung der Kollisionserkennung

In den Kapiteln 4 und 5 werden Verfahren beschrieben, die in der Lage sind, Objektpaartests in der Kollisionserkennung schnell durchzuführen und dabei *statische* oder *dynamische* Kollisionserkennungsergebnisse berechnen. Angepaßt dazu werden in Kapitel 6 Methoden vorgestellt, die es erlauben, für ein Objekt die Anzahl der Objektpaartests zu reduzieren und damit den Einsatz der Kollisionserkennung auch in Szenarien mit vielen Objekten zu ermöglichen. Was an dieser Stelle noch fehlt ist der Kontrollmechanismus, der die verschiedenen Verfahren zur Kollisionserkennung in *Virtual Reality* Anwendungen einbettet.

Der grobe Ablauf von *Virtual Reality* Anwendungen ist in Abbildung 7.1 verdeutlicht.

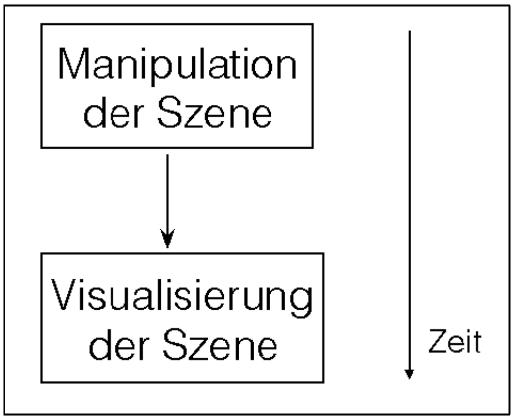

Abbildung 7.1: Grober Zeitablauf in Virtual Reality Anwendungen

Die Anwendungen bestehen aus einem Wechselspiel von szenenmanipulierenden Aktionen und Visualisierung der aktuellen Szene. Jedem visualisierten Bild kann dabei ein diskretes Zeitintervall zugeordnet werden, das sich in die beiden Teile

- Manipulation der virtuellen Szene und

- Visualisierung der virtuellen Szene

unterteilt.

Ein Teil der Manipulation der virtuellen Szene beschäftigt sich damit, objektunabhängige Manipulationen vorzunehmen, z.B. Modifikation des Blickwinkels des Betrachters, ein anderer erzeugt die Eingaben für die Kollisionserkennung, z.B. Bewegungsdaten von Eingabegeräten (SPACE MOUSE® , FlyingMouse usw.) oder Animationen von Objekten. Dazu kommen Bewegungsdaten von weitergehenden Simulationen, z.B. *Dynamiksimulationen* (vgl. [Sa99, SS98]). Die daraus resultierenden Objektbewegungen werden auf Kollision hin getestet und das Ergebnis entweder weitergehenden Simulationen zur Verfügung gestellt oder die Position der Objekte in der Szene direkt beeinflußt. Diese Berechnungen werden durchgeführt, bevor ein neues Bild der virtuellen Szene generiert wird. Daraus ergibt sich der verfeinerte Ablauf einer *Virtual Reality* Anwendung in Abbildung 7.2.

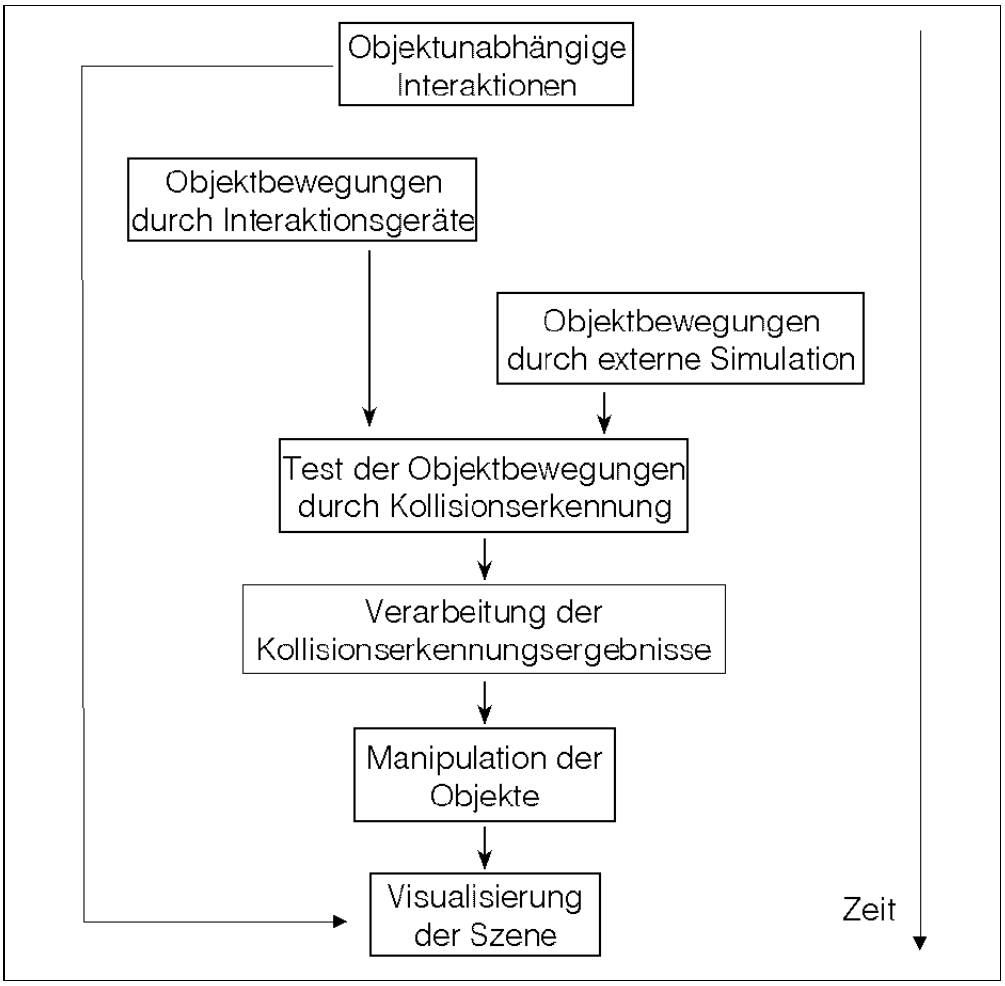

Abbildung 7.2: FEINERER ZEITABLAUF IN VIRTUAL REALITY ANWENDUNGEN

Aus Sicht der Kollisionserkennung ergibt sich dadurch eine dreiteilige Aufgabenstellung:

1. Registrierung der Objektbewegungen
2. Durchführung der Kollisionserkennung
3. Modifikation der Objektpositionen

7.1 Registrierung der Objektbewegungen

Bevor die Kollisionserkennung ihre Berechnungen bzgl. der bewegten Objekte durchführt, müssen in einem ersten Schritt alle Objektbewegungen innerhalb des durchzuführenden Kollisionserkennungsschrittes registriert werden. Darunter fallen die Objektbewegungen durch Interaktionen und durch andere Simulationen. Dabei bedeutet eine Objektbewegung, daß sich eine Gruppe von einem oder mehreren Objekten simultan bewegen, d.h. sie werden als starr miteinander verbunden betrachtet.

Als Ergebnis ergibt sich eine Menge

$$M_{bewegteObjektgruppen} = \{(O, \boldsymbol{T}_{relativ}) | O \text{ bewegte Objektgruppe}\}$$

von bewegten Objektgruppen und deren Bewegungen.

7.2 Durchführung der Kollisionserkennung

Die Durchführung der Kollisionserkennung selbst teilt sich in folgende Phasen auf:

1. Ermittlung aller bewegten Einzelobjekte
2. Ermittlung aller durchzuführenden Objektpaartests
3. Durchführung der Objektpaartests

7.2.1 Ermittlung aller bewegten Einzelobjekte

Liegt eine hierarchische Szenenrepräsentation vor, z.B. in OpenInventor^{TM} , kann durch die Spezifikation einer Bewegung für ein Objekt, eine ganze Reihe von Objekten bewegt werden (vgl. Anhang B). Diese Abhängigkeiten können in jedem Objekt einfach gespeichert und modifiziert werden, indem jedes Objekt O die Menge der Objekte speichert, die sich bewegen, wenn es selbst bewegt wird (M^O_{bewegt}), und die Menge der Objekte, durch die O mittelbar bewegt wird ($M^O_{wird\ bewegt}$), wobei O selbst nur zur Menge M^O_{bewegt} gehört nicht aber zu $M^O_{wird\ bewegt}$.

Zur Kollisionserkennungszeit liegt dann die Menge der bewegten Objekte bei der Bewegung des Objektes O als Menge M^O_{bewegt} vor.

Daraus ergibt sich der Algorithmus *BestimmungAllerBewegtenObjekte* in Abbildung 7.3.

Algorithmus $BestimmungAllerBewegtenObjekte(M_{bewegteObjektgruppen})$
Eingabe: $M_{bewegteObjektgruppen}$ Menge der bewegten Objektgruppen
Ausgabe: $M_{bewegteObjekte}$ Menge aller bewegten Einzelobjekte
$M_{bewegteObjekte} \leftarrow \emptyset$
for $(G, \boldsymbol{T}_{relativ})$ **in** $M_{bewegteObjektgruppen}$
do for O **in** M^{O}_{bewegt}
    do $M_{bewegteObjekte} \leftarrow M_{bewegteObjekte} \cup \{(O, \boldsymbol{T}_{Start}, \underbrace{\boldsymbol{T}_{relativ} \cdot \boldsymbol{T}_{Start}}_{\boldsymbol{T}_{Ziel}})\}$
return $M_{bewegteObjekte}$

Abbildung 7.3: ALGORITHMUS: BESTIMMUNG ALLER BEWEGTEN OBJEKTE AUS DEN BEWEGTEN OBJEKTGRUPPEN

7.2.2 Ermittlung aller durchzuführenden Objektpaartests

Liegt die Menge der tatsächlich bewegten Objekte $M_{bewegteObjekte}$ vor, können für diese Objekte die durchzuführenden Objektpaartests ermittelt werden. Dazu wird für diese Objekte die Raumpartitionierungsdatenstruktur aufgrund der vorgegebenen Bewegungen modifiziert und daraus die Menge der durchzuführenden Objektpaartests ermittelt (vgl. Kapitel. 6). Dabei ist darauf zu achten, daß für ein Paar von Einzelobjekten, die sich beide bewegen, nur ein Objektpaartest generiert wird. Als Ergebnis ergibt sich eine Menge von Objektpaartests. Diese Vorgehensweise ist in dem Algorithmus *BestimmungAllerObjektpaartests* in Abbildung 7.4 dargestellt.

Algorithmus $BestimmungAllerObjektpaartests(M_{bewegteObjekte}, S)$
Eingabe: $M_{bewegteObjekte}$ Menge der bewegten Objekte
S Raumpartitionierungsdatenstruktur
Ausgabe: $M_{Objektpaartests}$ Menge aller durchzuführenden Objektpaartests
$M_{Objektpaartests} \leftarrow \emptyset$
for $(O, \boldsymbol{T}^{O}_{Start}, \boldsymbol{T}^{O}_{Ziel})$ **in** $M_{bewegteObjekte}$
do $ModifikationRaumposition(S, O, \boldsymbol{T}^{O}_{Start}, \boldsymbol{T}^{O}_{Ziel})$
for $(O, \boldsymbol{T}^{O}_{Start}, \boldsymbol{T}^{O}_{Ziel})$ **in** $M_{bewegteObjekte}$
do $M^{O}_{ueberlappend} \leftarrow \ddot{U}berlappungsanfrageObjekt(O)$
    $M_{Objektpaartests} \leftarrow M_{Objektpaartests} \cup M^{O}_{ueberlappend}$
for $(O, \boldsymbol{T}^{O}_{Start}, \boldsymbol{T}^{O}_{Ziel})$ **in** $M_{bewegteObjekte}$
do $ModifikationRaumposition(S, O, \boldsymbol{T}^{O}_{Start})$
return $M_{Objektpaartests}$

Abbildung 7.4: ALGORITHMUS: BESTIMMUNG ALLER OBJEKTPAARTESTS AUS DEN BEWEGTEN EINZELOBJEKTEN

Algorithmus *BestimmungKollisionszeitpunkte*$(M_{Objektpaartests}, M_{bewegteObjektgruppen})$

Eingabe: $M_{Objektpaartests}$ Menge aller durchzuführenden Objektpaartests
$M_{bewegteObjektgruppen}$ Menge der bewegten Objektgruppen

Ausgabe: $M_{bewegteObjektgruppen}$ Menge der bewegten Objektgruppen mit frühesten Kollisionszeitpunkten

1. **for** G **in** $M_{bewegteObjektgruppen}$
2. **do** $Resultat(G) \leftarrow$NO COLLISION
3. **for** $(O_1, T^{O_1}_{Start}, T^{O_1}_{Ziel}, O_2, T^{O_2}_{Start}, T^{O_2}_{Ziel})$ **in** $M_{Objektpaartests}$
4. **do** $Resultat \leftarrow Objektpaartest(O_1, T^{O_1}_{Start}, T^{O_1}_{Ziel}, O_2, T^{O_2}_{Start}, T^{O_2}_{Ziel})$
5.   **if** $(O_1$ *ist bewegt*$)$
6.   **then if** $(t_{coll}(Resultat) < t_{coll}(Objektgruppe(O_1))$
7.   **then** $Resultat(Objektgruppe(O_1)) \leftarrow Resultat$
8.   **if** $(O_2$ *ist bewegt*$)$
9.   **then if** $(t_{coll}(Resultat) < t_{coll}(Objektgruppe(O_2))$
10.   **then** $Resultat(Objektgruppe(O_2)) \leftarrow Resultat$
11. **return** $M_{bewegteObjektgruppen}$

Abbildung 7.5: Algorithmus: Bestimmung frühester Kollisionszeitpunkte für alle bewegten Objektgruppen

Algorithmus *ModifikationObjektposition*$(M_{bewegteObjektgruppen})$
Eingabe: $M_{bewegteObjektgruppen}$ Menge der bewegten Objektgruppen

1. **for** G **in** $M_{bewegteObjektgruppen}$
2. **do** $\boldsymbol{T}_{relativ} \leftarrow KollisionsfreieBewegung(G)$
3.   **for** O **in** G
4.   **do** $\boldsymbol{T}^{O}_{Start} \leftarrow \boldsymbol{T}_{relativ} \cdot \boldsymbol{T}^{O}_{Start}$
5. $ModifikationRaumposition(S, O, \boldsymbol{T}^{O}_{Start})$
6.

Abbildung 7.6: Algorithmus: Modifikation der Objektpositionen der bewegten Einzelobjekte

7.2.3 Durchführung der Objektpaartests

Für alle zu testenden Objektpaare wird mit Hilfe der Hüllkörperhierarchien und der Basiskollisionserkennung (vgl. Kapitel 4,5) ein frühester Kollisionszeitpunkt ermittelt. Wird eine *statische* Kollisionserkennung durchgeführt, haben alle erkannten Kollisionen Zeitpunkt 0. Aus den frühesten Kollisionszeitpunkten der einzelnen Objektpaare wird danach der früheste Kollisionszeitpunkt für die zugehörige bewegte Objektgruppe ermittelt. Dieser stellt den frühesten Kollisionszeitpunkt für die spezifizierte Objektbewegung dar.

Daraus ergibt sich der Algorithmus *BestimmungKollisionszeitpunkte* in Abbildung 7.5.

7.3 Modifikation der Objektpositionen

Nachdem das Kollisionserkennungsergebnis für jede bewegte Objektgruppe vorliegt, werden diese entweder zur direkten Objektmanipulation benutzt, d.h. die Objekte werden bis zum frühesten Kollisionszeitpunkt bewegt, oder weitergehende Simulationen modifizieren die Objektposition. Das Ergebnis dieser Nachbearbeitung wird dazu benutzt, um die aktuellen Positionen aller bewegten Objekte zu bestimmen.

Dazu werden die aktuellen Transformationen aller bewegten Einzelobjekte und die Raumpartitionierungsdatenstruktur (vgl. Kapitel 6) modifiziert.

Daraus ergibt sich der Algorithmus *ModifikationObjektposition* in Abbildung 7.6.

Kapitel 8

Kollisionserkennung bei zeitkritischer Berechnung

Wird Kollisionserkennung im Zusammenhang mit *Virtual Reality* Anwendungen eingesetzt, dann ergibt sich für die Kollisionserkennung das Problem, daß sie sich der allgemeinen Zeitstruktur unterordnen muß, die durch die Visualisierung und die Interaktion gesteuert wird. Wie der Kollisionserkennungsprozeß in diese Zeitstruktur eingebettet wird, wurde in Kapitel 7 beschrieben.

Dabei zeigt sich, daß der Kollisionserkennungsprozeß innerhalb eines Manipulationsschrittes der Anwendung in drei Teile unterteilt werden kann, wobei der zweite Teil derjenige ist, bei dem die Kollisionserkennung den größten Teil ihrer Zeit verbraucht. Der Zeitverbrauch der Kollisionserkennung ist im wesentlichen von der Anzahl und der Komplexität der durchzuführenden Objektpaartests abhängig.

Um eine feste Visualisierungs- und Interaktionsrate zu erhalten, müssen die Vorgänge der Szenenmanipulation in einem vorgegebenen Zeitintervall ablaufen. Um dies zu ermöglichen, müssen die zeitintensiven Teile der Szenenmanipulation mit Hilfe von vorgegebenen Zeitintervallen, innerhalb derer die Berechnung beendet werden muß, gesteuert werden, wobei diese in der Lage sein müssen, Zeit gegen Genauigkeit zu tauschen (*graceful degradation*). Betrachten wir den gesamten Kollisionserkennungsprozeß, muß die zweite Phase der *Durchführung der Kollisionserkennung* (vgl. Kapitel 7, S. 142) mit Hilfe solcher Zeitintervalle zeitlich begrenzt werden.

D.h. ausgehend von der Szenenmanipulationssteuerung wird der Kollisionserkennung ein Zeitintervall Δ zugeordnet, innerhalb dessen die nächste Kollisionserkennungsaufgabe bearbeitet werden muß. Dabei besteht eine Kollisionserkennungsaufgabe aus einer Menge von bewegten Objektgruppen, deren Bewegungen auf Kollision getestet werden müssen.

Die Phase der *Durchführung der Kollisionserkennung* selbst teilt sich wieder in die drei Teilphasen

1. Ermittlung aller bewegten Einzelobjekte,
2. Ermittlung aller durchzuführenden Objektpaartests und
3. Durchführung der Objektpaartests

auf (vgl. Abschnitt 7.2). Dabei ergeben sich für die beiden ersten Phasen *Ermittlung aller bewegten Einzelobjekte* und *Ermittlung aller durchzuführenden Objektpaartests* nur schwerlich Möglichkeiten, diese durch zeitkritische Berechnungen zu steuern, da durch Eingriffe darin zu testende Objektpaare übersehen würden, auch wenn Zyda et. al. [ZO+93] dies für große Fahrsimulationen als tolerabel erachten. Deren zeitlicher Anteil an der gesamten *Durchführung der Kollisionserkennung* sind jedoch gering, so daß dies keine Einschränkung für eine gesamte zeitkritische Berechnung darstellt.

8.1 Bisherige Arbeiten

Eines der Hauptanwendungsgebiete der zeitkritischen Berechnung war bisher der Bereich der *Computergraphik*. Hubbard gibt in [Hu95] einen Überblick darüber.

In den bisherigen Arbeiten zur schnellen Kollisionserkennung stand im wesentlichen die Beherrschung der geometrischen Komplexität der beteiligten Objekte im Vordergrund, wobei das Ergebnis der Kollisionserkennung vorwiegend zur direkten Manipulation der Bewegungen der Objekte benutzt wurde. Dies führte dazu, daß zur Kollisionserkennung im wesentlichen *statische* Kollisionstests verwendet wurden (vgl. Kapitel 4). Beispiele dafür sind [Qu94, FZ95, GLM96, BC+96, Za97].

Hubbard beschäftigt sich in [Hu95] mit dem Problem der zeitkritischen Berechnung (*time critical computing*) im Zusammenhang mit der Kollisionserkennung. Die Kollisionserkennung von Hubbard teilt sich iterativ in die beiden Teile *broad phase* und *narrow phase* auf, wobei in der *broad phase* jeweils immer berechnet wird, wann aufgrund der derzeitigen Geschwindigkeit und der maximal möglichen Beschleunigung der Objekte eine Kollision frühest möglich ist. Folgen diese Zeitpunkte zu dicht aufeinander, wird mit Hilfe der *narrow phase* zu einem festen Zeitpunkt getestet, ob eine *statische* Kollision eines Objektpaares vorliegt. Die *broad phase* kann dabei nicht unterbrochen werden, da jedoch die dabei durchgeführte Berechnung recht einfach ist, ergibt sich trotzdem eine hinreichende Granularität. Die *narrow phase* kann nach jedem Hüllkörpertest unterbrochen werden, so daß dabei eine sehr hohe Granularität gegeben ist. Kann die Vorgehensweise innerhalb einer vorgegebenen Zeitspanne eine Kollision nicht ausschliessen, wird dies als mögliche Kollision zurückgeliefert.

Für alle hierarchischen Verfahren wie [Qu94, FZ95, GLM96, BC+96, Za97] besteht die Möglichkeit, analog zu der Methode von Hubbard die Kollisionserkennung mit einer Menge von verbleibenden Hüllkörpertests abzubrechen und eine nicht näher spezifizierbare Kollision als Ergebnis zurückzuliefern.

Gleichzeitig ergibt sich das Problem, daß durch die Verwendung von *statischer* Kollisionserkennung auf der Basis des *Clipping-Algorithmus* von Cyrus und Beck [CB78] kein Ergebnis berechnet werden kann, das als Eingabe für weitergehende Simulationen wie z.B. von Mirtich [Mi96] oder Sauer [Sa99, SS98] dienen kann.

8.2 Verteilung des Zeitintervalls auf die Objektpaartests

Eine Möglichkeit ist, das vorgegebene Zeitintervall Δ auf die durchzuführenden n Objektpaartests zu verteilen. Dabei kann das Zeitintervall gleichmäßig auf die Objektpaartests verteilt werden, so daß jeder Objektpaartest Zeit $\frac{\Delta}{n}$ erhält. Alternativ dazu ist denkbar, daß Objektpaartests, die sich in der Nähe des Betrachters abspielen, da sie von größerem Interesse für das intuitiv korrekte Objektverhalten aus Sicht des Benutzers sind, mehr Zeit erhalten als andere, die sich in weiter Ferne abspielen.

Aufgrund dieser Vorgehensweise ist nun jeder Objektpaartest P mit einem Zeitintervall Δ_P ausgestattet. Es verbleibt das Problem, innerhalb dieses Zeitintervalls ein Kollisionserkennungsergebnis für das Objektpaar zu berechnen.

8.2.1 Approximativer Objektpaartest innerhalb eines festen Zeitintervalls

Gegeben ist nun das Problem, daß für ein Objektpaar die spezifizierten Bewegungen innerhalb eines festgelegten Zeitintervalls auf Kollisionen überprüft werden müssen. Dabei ist es das Ziel, ein Ergebnis zu berechnen, das entweder für die direkte Objektmanipulation oder für weitergehende Simulationen geeignet ist und dabei eine möglichst gute Übereinstimmung mit dem vollständigen Ergebnis der Kollisionserkennung aufweist.

Um die Kollisionserkennung für ein Objektpaar möglichst effizient durchführen zu können, benutzen wir zum einen Hüllkörperhierarchien (vgl. Kapitel 3) und zum anderen Verfahren zur Benutzung dieser Hierarchien (vgl. Kapitel 4).

Gehen wir davon aus, daß die beiden Hierarchien und das Verfahren zur Benutzung der Hierarchien festgelegt ist, besteht das Problem, daß die Kollisionserkennung zu einem Zeitpunkt abgebrochen werden muß, zu dem noch eine Menge von zu testenden Hüllkörperpaaren vorhanden ist. Mit Hilfe dieser Hüllkörperpaare muß eine Entscheidung über das Vorliegen einer Kollision und evtl. den Kollisionszeitpunkt und die dabei kollidierenden Objektteile getroffen werden. Dazu können zwei Methoden benutzt werden:

1. Kollisionsentscheidung mit verbleibenden Hüllkörperpaaren
2. Kollisionsentscheidung mit approximierenden Flächen

Kollisionsentscheidung mit verbleibenden Hüllkörperpaaren

Eine erste Möglichkeit der Kollisionsentscheidung zum Abbruchzeitpunkt besteht darin, die verbleibenden Hüllkörperpaare selbst zur Kollisionsentscheidung zu benutzen.

Wird das Ergebnis der Kollisionserkennung nicht für weitergehende Simulationen benutzt, sondern manipuliert die Objekte direkt, können bei verbleibenden Hüllkörperpaaren die Bewegungen der Objekte als kollidierend angenommen und der Kollisionszeitpunkt auf $t = 0$ gesetzt werden. Diese Vorgehensweise garantiert, daß keine Kollision übersehen wird, d.h. keine Durchdringung der beteiligten Objekte erfolgt. Jedoch kann dadurch eine Kollision erkannt werden, die tatsächlich nicht vorliegt.

Wird dabei zur Kollisionserkennung die *statische* Kollisionserkennung benutzt, kann mit Hilfe des *gerichteten Hausdorff-Abstands* des Hüllkörpers zu der von ihm umhüllten Flächenmenge eine Abschätzung für den tatsächlich noch möglichen Abstand zwischen den beiden Objekten berechnet werden, denn der Abstand zweier Punkte von jeweils einem der beiden Objekte ist kleiner oder gleich der Summe der beiden *gerichteten Hausdorff-Abstände* der überdeckenden Hüllkörper, d.h. seien F_1 und F_2 die beiden Flächenmengen sowie h_1 und h_2 die betrachteten Hüllkörper, dann gilt:

$$d(F_1, F_2) \leq d_{HA}(h_1, F_1) + d_{HA}(h_2, F_2)$$

Dadurch kann dem Ergebnis, daß eine Kollision nicht ausgeschlossen werden kann, noch die Information hinzugefügt werden, daß, falls tatsächlich keine Kollision vorliegen sollte, der maximale Abstand zwischen den beiden Objekten kleiner oder gleich dem Maximum über alle Summen von *gerichteten Hausdorff-Abständen* der verbleibenden zu testenden Hüllkörperpaare ist.

Sollen mit Hilfe der verbleibenden Hüllkörperpaaren früheste Kollisionszeitpunkte bestimmt werden, können die Hüllkörper selbst als einfache Objekte angesehen und auf Kollision getestet werden. Werden alle verbleibenden Hüllkörperpaare mittels *dynamischer* Kollisionserkennung getestet, erhält man eine konservative Abschätzung für den tatsächlichen Kollisionszeitpunkt, denn bevor die Flächenmengen kollidieren, kollidieren zunächst die zugehörigen Hüllkörper.

Soll dieses Konzept zum Abbruchzeitpunkt eingesetzt werden, dann ist klar, daß nicht alle verbleibenden Hüllkörperpaare auf Kollision getestet werden können, sondern eines als Indikator ausgewählt werden muß. Durch diese Vorgehensweise ist kein konservatives Ergebnis in der Hinsicht möglich, daß keine Kollision bzw. kein früherer Kollisionszeitpunkt übersehen wird. Bleibt die Frage, welches Hüllkörperpaar als Indikator ausgewählt werden sollte. Dazu eignet sich das Paar von Hüllkörpern, das tendenziell die Stelle mit dem geringsten Abstand zwischen den beiden Objekten definiert. Das ist das verbleibende Hüllkörperpaar, dessen Summe der *gerichteten Hausdorff-Abstände* minimal ist. Das zu jedem Zeitpunkt vorhandene minimale Paar kann während des Kollisionstests mit Hilfe der beiden Hierarchien einfach gespeichert werden, so daß es zum Abbruchzeitpunkt direkt zur Kollisionsentscheidung vorliegt.

Gleichzeitig kann der Hüllkörper als lokal gute Approximation der umhüllten Flächenmenge angesehen werden (was zumindest in der Nähe der Blattknoten der Hierarchie gegeben ist) und somit das Ergebnis der Kollisionserkennung auch für weitergehende Simulationen benutzt werden.

Kollisionszeitpunkt mit approximierenden Flächen

Werden die Hüllkörper als lokal gute Approximation der jeweils umhüllten Flächenmenge benutzt, ist dies tendentiell richtig. Gleichzeitig kann man sich aber überlegen, ob man eine Flächenmenge mittels einiger weniger Flächen so repräsentieren kann, daß das Kollisionsverhalten der approximierenden Flächen ähnlich dem der gesamten Flächenmenge ist. Liegt eine große, stark gekrümmte Flächenmenge vor, ist dies schwierig. Betrachtet man dagegen einzelne Teile der Flächenmenge lokal, so sind diese wenig gekrümmt und können durch wenige Flächen gut approximiert werden.

Daher fügen wir jedem Knoten der Hierarchie neben der umhüllten Flächenmenge und dem Hüllkörper *approximierende Flächen* hinzu. Mit Hilfe derer wird dann zum Abbruchzeitpunkt der Kollisionserkennung eine approximative Kollisionsberechnung durchgeführt.

Dazu wird analog zu der Benutzung der Hüllkörperpaare bei der approximativen Bestimmung des frühesten Kollisionszeitpunktes auch hier nur das Hüllkörperpaar betrachtet, das tendenziell die Stelle mit dem geringsten Abstand zwischen den beiden Objekten definiert. Für dieses Hüllkörperpaar wird der *Basiskollisionstest* (vgl. Kapitel 5) auf die approximierenden Flächen angewendet und als Ergebnis des gesamten verbleibenden Objektpaartests benutzt.

Auch hier ist klar, daß durch die Beschränkung auf ein exemplarisches Hüllkörperpaar Fehler bei der Kollisionserkennung auftreten können. Zum einen kann durch falsche Auswahl des Hüllkörperpaares eine Kollision übersehen werden; aber auch durch die Verwendung von approximierenden Flächen können Kollisionen übersehen werden, wenn diese die Flächenmenge im Gegensatz zu einem Hüllkörper nicht vollständig umhüllen. Darüber hinaus können Kollisionen erkannt werden, die zwischen den tatsächlichen Flächenmengen nicht vorliegen.

Ein Beispiel zur Berechnung solcher Fläche wollen wir in der Folge geben. Diese können zum Vorberechnungszeitpunkt zusammen mit der gesamten Hüllkörperhierarchie berechnet werden und liegen daher zum Zeitpunkt des Kollisionserkennungsprozesses vor.

Berechnung approximierender Flächen

Gegeben ist nun eine Flächenmenge F, für die approximierende Flächen berechnet werden sollen. Die approximierenden Flächen sollen F in deren Hauptrichtung abdecken. Dazu werden zwei Ebenen berechnet, deren Normalenrichtung dem Mittel der Flächenrichtungen

Algorithmus *BerechnungApproximierenderFlächen*(F)
Eingabe: F Flächenmenge
Ausgabe: F_{approx} Menge approximierender Flächen
$\boldsymbol{n}_{inter} \leftarrow \mathbf{0}$
for f **in** F
do $\boldsymbol{n}_{inter} \leftarrow \boldsymbol{n}_{inter} + \frac{\boldsymbol{n}_f}{|F|}$
$\boldsymbol{n}_{inter} \leftarrow \frac{\boldsymbol{n}_{inter}}{|\boldsymbol{n}_{inter}|}$
$d_{min} \leftarrow \infty$
$d_{max} \leftarrow -\infty$
for $\boldsymbol{v}$ **in** $V(F)$
do $d_{akt} \leftarrow \boldsymbol{n}_{inter} \cdot \boldsymbol{v}$
    if $(d_{akt} < d_{min})$
    then $d_{min} \leftarrow d_{akt}$
    if $(d_{akt} > d_{max})$
    then $d_{max} \leftarrow d_{akt}$
(∗ Projiziere Punkte aus $V(F)$ in eine der beiden Ebenen ∗)
$V_{proj}(F) \leftarrow \emptyset$
for $\boldsymbol{v}$ **in** $V(F)$
do $\boldsymbol{v}_{proj} \leftarrow \boldsymbol{v} - (\boldsymbol{n}_{inter} \cdot \boldsymbol{v} - d_{min}) \cdot \boldsymbol{n}_{inter}$
    $V_{proj}(F) \leftarrow V_{proj}(F) \cup \{\boldsymbol{v}_{proj}\}$
$\omega \leftarrow \mathit{OptimiereRechteck}(V_{proj}(F))$
if $(d_{max} - d_{min} < \varepsilon)$
then $f_{min} \leftarrow \mathit{Rechteck}(\omega, d_{min})$
    $f_{max} \leftarrow \mathit{Rechteck}(\omega, d_{max})$
    $F_{approx} \leftarrow \{f_{min}, f_{max}\}$
else $F_{approx} \leftarrow \mathit{Box}(\boldsymbol{n}_{inter}, \omega)$
return F_{approx}

Abbildung 8.1: ALGORITHMUS: BERECHNUNG APPROXIMIERENDER FLÄCHEN

der gesamten Flächenmenge entsprechen. Ihre Abstände zum Nullpunkt des Koordinatensystems werden so gewählt, daß sie komplett oberhalb bzw. unterhalb der Flächenmenge liegen. In diese Ebenen werden die Eckpunkte der Flächenmenge projiziert und in den Ebenen das kleinste einschliessende Rechteck für die projizierten Eckpunkte berechnet. Dadurch ergeben sich die beiden approximierenden Flächen. Sind die Abstände zwischen den beiden Flächen zu groß, deutet dies auf eine starke Krümmung der Flächenmenge hin. In diesem Fall approximieren die beiden Flächen die Flächenmenge nur unzureichend und sollten zu einer Box erweitert werden.

Daraus ergibt sich der Algorithmus *BerechnungApproximierenderFlächen* in Abbildung 8.1.

8.3 Simultane Behandlung aller Objektpaartests

Der Vorteil der *Verteilung des Zeitintervalls auf die Objektpaartests* ist, daß jeder Objektpaartest betrachtet wird und dabei eine gewisse Zeit zur Verfügung steht. Ein Balancierung der Zeit zwischen den verschiedenen Objektpaartests ist dabei allerdings nicht möglich. Will man erreichen, daß für schwierige Objektpaartests mehr Zeit verwendet wird als für einfache, kann man alle Objektpaartests simultan betrachten. Dabei wird dann nicht ein Paar von Hierarchien gegeneinander getestet, sondern alle Paare gleichzeitig. Dazu wird von allen durchzuführenden Objektpaartests das Paar von Hüllkörpern an der Wurzel der Hierarchien als initiale Menge von zu testenden Hüllkörperpaaren benutzt. Auf diese Menge von Hüllkörperpaaren können nun dieselben Verfahren zur Benutzung von Hierarchien angewendet werden wie bei einem Vergleich eines einzelnen Paares von Hierarchien (vgl. Kapitel 4).

Zum Abbruchzeitpunkt können die Verfahren des *approximativen Objektpaartests innerhalb eines festen Zeitintervalls* angewendet werden. Dazu muß während des Verfahrens das die Kollisionsentscheidung bestimmende Hüllkörperpaar für jeden Objektpaartest aktualisiert werden und für die Objektpaartests, die am Ende des Zeitintervalls nicht abgeschlossen sind, die approximative Kollisionsberechnung durchgeführt werden.

Kapitel 9

Implementierung

In den Kapiteln 3 bis 8 sind Verfahren vorgestellt worden, die eine echtzeitfähige Kollisionserkennung zum Einsatz in *Virtual Reality* Anwendungen ermöglichen. Diese umfassen alle Bereiche des Kollisionserkennungsprozesses:

- Behandlung vieler Objekte (Kapitel 6)
- Berechnung von *Hüllkörperhierarchien* und deren Verwendung mit *statischer* und *dynamischer* Kollisionserkennung (Kapitel 3 und 4)
- *Statische* und *dynamische* Basiskollisionserkennung für Objektteile (Kapitel 5)
- Integration in ein *Virtual Reality* System unter Berücksichtigung der Echtzeitfähigkeit (Kapitel 7 und 8)

Die praktische Performance der Verfahren hängt zusätzlich zu ihrer Konzeption wesentlich von ihrer Implementierung ab. Bei der Realisierung der Datenstrukturen und Verfahren zur Kollisionserkennung treten Fragestellungen auf, die im Zusammenhang mit der reinen Beschreibung der Verfahren keine Rolle spielen. Da sie jedoch wesentlich sind für die Performance des Kollisionserkennungsmoduls, werden wir diese in der Folge in allgemeiner Weise darstellen.

Zunächst werden wir in Abschnitt 9.1 die softwaretechnischen Rahmenbedingungen vorstellen, unter denen die Realisierung der Verfahren erfolgt ist. Danach wollen wir die realisierten Datenstrukturen und Besonderheiten der Verfahren erläutern, die wesentlich sind für das Kollisionserkennungsmodul, ohne dabei auf die verwendete Programmiersprache oder eingesetzten Softwarebibliotheken einzugehen, so daß die Ausführungen allgemein sind für die Realisierung der von uns vorgestellten Verfahren zur Kollisionserkennung. Dazu werden wir in Abschnitt 9.2 kurz auf die Realisierung der Transformationsmathematik eingehen, in Abschnitt 9.3 die Realisierung der Objektrepräsentation und in Abschnitt 9.4 die der Kollisionserkennung beschreiben.

9.1 Softwaretechnische Rahmenbedingungen

Die Rahmenbedingungen für die Realisierung der Verfahren zur Kollisionserkennung werden bestimmt durch die *Virtual Reality* Plattform **DBView** der Daimler-Benz AG, die im Virtual Reality Competence Center (für nähere Informationen siehe Anhang C) entwickelt wurde. Als Computerhardware werden vorrangig Maschinen der Firma Silicon Graphics (Onyx Reality Engine 3, Octane, Maximum Impact, O2, Indy) verwendet. Der Einsatz von graphikstarken PCs wird zukünftig verstärkt werden, da deren Leistungsfähigkeit zunehmend der der Graphikworkstations entspricht. Die Software der *Virtual Reality* Plattform basiert derzeit zur Visualisierung und Szenenverwaltung auf der in C++ als Klassenbibliothek realisierten Software-Bibliothek OpenInventor^{TM} der Firma Silicon Graphics (vgl. [We94]). Auf dieser aufbauend sind verschiedene Module zur Realisierung einer *Virtual Reality* Umgebung in C++ objektorientiert implementiert.

Folgerichtig ist auch die Simulationsumgebung, für die die Kollisionserkennung die Basis darstellt, innerhalb von **DBView** objektorientiert in C++ realisiert. Die Analyse der Datenstrukturen zur *Objektrepräsentation* und *Transformationsmathematik* in OpenInventor^{TM} zeigt, daß beide visualisierungsorientiert ausgelegt sind. Dazu kommt, daß der Markt für 3D-Visualisierungssysteme seit Anfang der 90er Jahre eine schnelle Entwicklung genommen hat. Ausgehend vom OpenInventor^{TM} Dateiformat entwickelte sich der VRML 1-Standard mit den zugehörigen Softwaresystemen. Dieser ermöglichte es zum ersten Mal, die Visualisierung und Navigation innerhalb von 3D-Szenarien über das Internet durchzuführen. Aufbauend auf VRML 1 entwickelte sich der Standard VRML 2, der die Funktionalität von VRML 1 erweitert und Teile von VRML 1 verändert. Daraus kann man ableiten, daß sich in näherer Zukunft die Visualisierungssysteme weiter verändern werden. Gleichzeitig sollten periphäre Module wie z.B. Simulationsumgebung oder Ansteuerung von externen Hardware Devices (Head Mounted Display, Datenhandschuh, Positionstracker usw.) möglichst wenig auf das derzeit bestehende Visualisierungssystem angewiesen sein. Daher sollte die Simulationsumgebung soweit visualisierungsunabhängig realisiert werden, daß eine spätere Integration in eine geänderte Visualisierungsumgebung möglichst einfach durchzuführen ist. Ausgehend von diesen Überlegungen haben wir uns entschlossen, für die Simulationsumgebung eine eigene *simulationsorientierte Objektrepräsentation* und *Transformationsmathematik* zu realisieren, die die Verpflechtung der Simulationsumgebung mit dem Visualisierungssystem auf einige wohldefinierte Schnittstellen reduziert. Dadurch kann eine größtmögliche Unabhängigkeit von dem gewählten Visualisierungssystem erreicht werden. Die daraus sich ergebende Freiheit bei der Wahl des Visualisierungssystems wurde bei der Realisierung der Basismodule und des Kollisionserkennungsmoduls dazu benutzt, die Entwicklung innerhalb eines einfachen Visualisierungssystems **UniSBView** durchzuführen. Dieses ist ebenfalls OpenInventor^{TM} basiert, verfügt jedoch über wesentlich weniger Funktionalität als die gesamte *Virtual Reality* Softwareplattform **DBView**. Während des Entwicklungsprozesses ergibt sich dadurch eine weit geringere Anfälligkeit für externe Fehlerquellen als dies bei einem umfassenden System wie **DBView** fast zwangsläufig der Fall ist. Aufgrund der wohldefinierten Schnittstellen zum Visualisierungssystem ergeben sich bei der Integration der Simulationsumgebung in **DBView** keine größeren Probleme.

Der Aufbau der Simulationsumgebung ist in Abbildung 9.1 graphisch veranschaulicht.

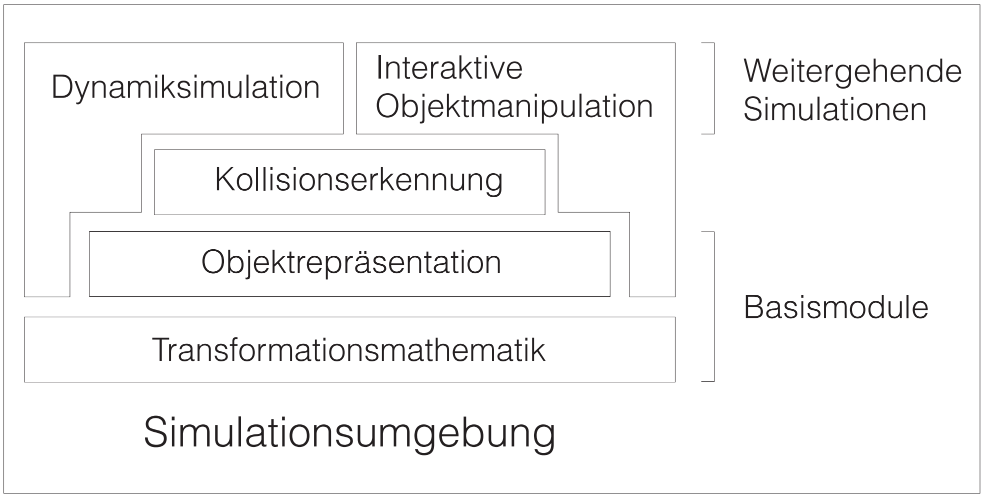

Abbildung 9.1: Aufbau der Simulationsumgebung

Als unterstützende Software wurden Teile der *Numerical Recipes* [PT+92], die Implementierung des Algorithmus von Welzl zur Berechnung kleinster einschliessender Kugeln von White [Wh], die Softwarebibliothek *qhull* des Geometry Centers [BDH93] und die Softwarebibliothek *LEDA* des Max-Planck-Institutes für Informatik in Saarbrücken [MN, MN95] verwendet. Zur Größenordnung des Umfangs der Implementierung sei gesagt, daß die Realisierung des Kollisionserkennungsmoduls zusammen mit den beiden Basismodulen insgesamt etwas über 50000 Zeilen C++-Sourcecode umfaßt. Die Abbildungen D.2 und D.1 zeigen die graphische Benutzeroberfläche zur Parametrisierung der Kollisionserkennung zur Vorberechnung der Hüllkörperhierarchien und zur Steuerung des Ablaufes der Kollisionserkennung.

9.2 Transformationsmathematik

Die mathematische Betrachtung rigider Transformationen ist in Anhang A erläutert. Zur Berechnung der aktuellen Transformationen von Objekten eignet sich die Darstellung der Transformation als 4×4-*Transformationsmatrix*. Diese besteht aus einer 3×3-Matrix $\boldsymbol{R}$ zur Repräsentation des Rotationsanteils und einem 3-dimensionalen Vektor $\boldsymbol{t}$ für den Translationsanteil. Zusammen mit einer konstanten vierten Zeile ergibt sie eine 4×4-*Transformationsmatrix* $\boldsymbol{T}$, die einfach zu invertieren ist und die sich für Berechnungen mit

3-dimensionalen Koordinaten eignet.

$$\boldsymbol{T} = \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \\ \hline \boldsymbol{0} & 1 \end{array}\right] \qquad \boldsymbol{T}^{-1} = \left[\begin{array}{c|c} \boldsymbol{R}^T & -\boldsymbol{R}^T \cdot \boldsymbol{t} \\ \hline \boldsymbol{0} & 1 \end{array}\right]$$

Um die Berechnungen effizienter durchführen zu können, wenn die Transformation keinen Rotations- oder keinen Translationsanteil aufweist, werden zusätzlich zu den Werten der Matrix Indikatoren gespeichert, die anzeigen, ob ein Rotations- bzw. ein Translationsanteil vorliegt.

Des weiteren sind neben dieser gesamten Darstellung von rigiden Transformationen verschiedene Repräsentationen für die Darstellung von Rotationen implementiert:

- *Rotationsmatrix*
- *Quaternion*
- *Achse und Winkel*

Die Darstellung als 3×3-*Rotationsmatrix* wird vor allem bei der Transformation von relativen Vektoren verwendet, *Quaternionen* im wesentlichen bei der Interpolation von Bewegungen der Objekte. *Achse und Winkel* dient fast ausschließlich zu Umrechnungen. Translationen werden als 3-dimensionale Vektoren repräsentiert.

9.3 Objektrepräsentation

Die Aufgaben der Objektrepräsentation für die Kollisionserkennung und weitergehende Simulationen lassen sich in zwei Kategorien einteilen:

1. Bereitstellung einer simulationsgeeigneten Objektdatenstruktur
2. Unterstützung der Berechnungseffizienz während der Simulation

Die Objektrepräsentation heutiger 3D-Visualisierungssysteme wie z.B. OpenInventor™ ist oberflächenbasiert. Sie gehören damit zur Klasse der *Surface Rendering* Systeme und verarbeiten Dreiecke als einfachste zu visualisierende Objekte (*Volume Rendering* ist ein alternatives Visualisierungsverfahren, das derzeit im Forschungsstadium ist, Volumendaten verarbeitet und das Potential hat, zukünftig das *Surface Rendering* zu verdrängen. Ein guter Überblick über den State-of-the-Art im *Volume Rendering* wird in [Ka97a, Ka97b] gegeben). Für die Visualisierung von Oberflächen sind topologische Informationen, die über die Beschreibung der einzelnen Flächen hinausgehen, nicht von Interesse. Folgerichtig sind diese auch in der Objektrepräsentation z.B. von OpenInventor™ nicht oder nur implizit repräsentiert. Betrachten wird die Herkunft der Daten für *Virtual Reality* Anwendungen z.B. als konvertierte CAD-Daten oder Modelle aus Modellierungstools, sehen wir, daß diese Daten in der Regel keine Polyeder darstellen, sondern Flächenmengen aus zusammenhängenden Flächen, die kein inneres Volumen umschließen. Für die Flächen, die umgeben sind von anderen Flächen, können weitergehende topologische Informationen aufgebaut werden, die am Rand der Flächenmenge unvollständig sind. Gleichzeitig können auch inmitten einer Flächenmenge topologische Informationen fehlen, da bei der Konvertierung parametrisierter Oberflächen Löcher in der Triangulierung beim Übergang zwischen unterschiedlichen Parametrisierungen entstehen können.

Für die Vorgehensweise unserer Kollisionserkennung spielen topologische Informationen innerhalb von Flächenmengen, die über die Spezifikation der einzelnen Flächen hinausgehen, keine Rolle, jedoch im Hinblick auf die weitergehenden Simulationen sollten diese nicht unberücksichtigt bleiben. Daher wurde die Datenstruktur zur Objektrepräsentation derart ausgelegt, daß sie auch die Anforderungen der weitergehenden Simulationen erfüllt. Wir haben uns für eine modifizierte *half-edge* Datenstruktur (vgl. [Ke97, Ho89, Ma88]) entschieden, die in Abbildung 9.2 graphisch veranschaulicht ist. Die *half-edge* Datenstruktur als kantenorientierte Objektrepräsentation erlaubt es, entlang der Kanten auf die topologischen Nachbarschaftsbeziehungen zuzugreifen. Die Kollisionserkennung mit Hilfe von *Hüllkörperhierarchien* ist jedoch flächenorientiert realisiert. D.h. es werden einzelne Flächen mit Hüllkörpern überdeckt und von diesen ausgehend die Basiskollisionserkennung angesteuert. Daher ist eine flächenorientierte Repräsentation der Objektgeometrie nötig. Für diese werden bei den Flächen zusätzlich die begrenzenden Kanten und die Eckpunkte gespeichert.

Der Aufbau der topologischen Information einer Flächenmenge läuft in drei Stufen ab:

1. *Erzeugung der Eckpunkte*
   Aus den Visualisierungsdaten werden zunächst die Eckpunkte der Flächenmenge extrahiert. Dabei mußte darauf geachtet werden, daß keine Eckpunkte mit denselben Koordinaten im Raum mehrfach erzeugt werden. Die so erzeugten Eckpunkte haben noch keinen topologischen Zusammenhalt.

2. *Erzeugung der Flächen und der begrenzenden Kanten*
   Dazu wird die Liste der Flächen der Flächenmenge durchlaufen und jede Fläche unabhängig von anderen generiert. Aus der ersten Stufe liegen bereits die Eckpunkte aller Flächen vor. Beim Erzeugen einer Fläche werden die berandenden Kanten erzeugt sowie die Verweise der Kanten auf die zugehörigen Eckpunkte, die vorhergehende Kante, nachfolgende Kante und die Fläche selbst sowie der Verweis der Fläche auf eine berandende Kante.

3. *Auffinden der Rückwärtskanten*
   Um die topologische Information zu komplettieren, muß zu jeder Kante die jeweils entgegengerichtete Kante der adjazenten Fläche gefunden werden. Dazu werden die

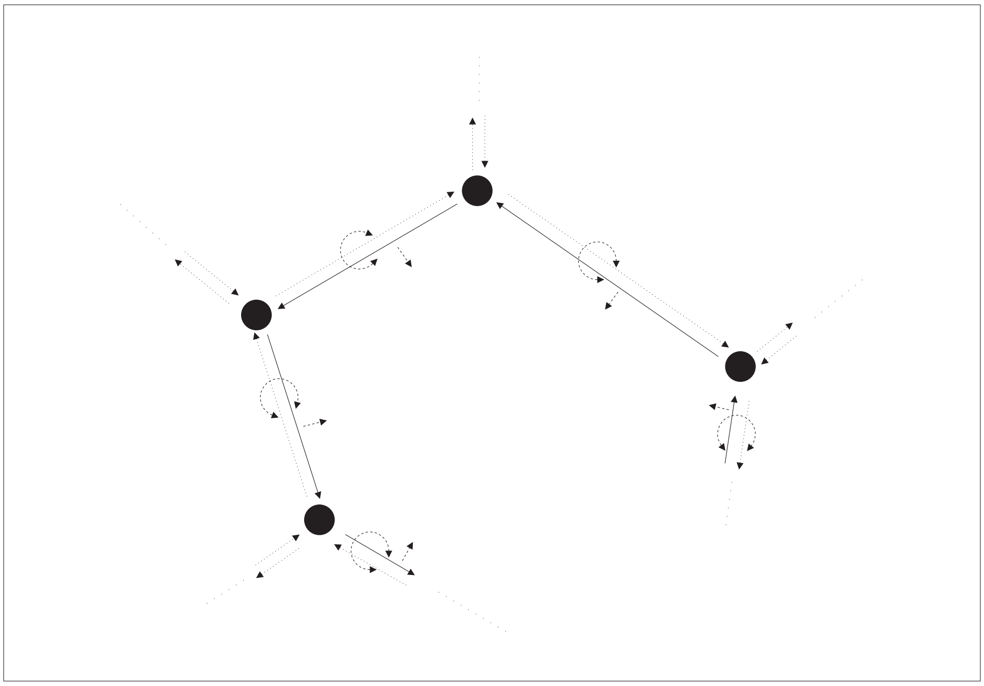

Abbildung 9.2: Graphische Darstellung *half-edge* Datenstruktur

Kanten entsprechend ihrer Orientierung und der Lage ihrer Anfangs- und Endpunkte aufsteigend sortiert. Dadurch gelangen zueinandergehörige Kantenpaare nebeneinander.

Durch die Unvollständigkeit der topologischen Information kann es vorkommen, daß zu einer Kante keine oder mehr als eine Rückwärtskante existiert. In beiden Fällen werden für die betreffenden Kanten keine topologischen Nachbarschaftsbeziehungen aufgebaut.

Durch diese Vorgehensweise ist gewährleistet, daß beim Vorliegen eines Polyeders als Objekt dessen vollständige *half-edge* Repräsentation aufgebaut wird.

Neben der rein geometrischen Darstellung der Objekte kann die gesamte Objektrepräsentation einen Beitrag zur Steigerung der Effizienz der Verfahren während der Kollisionserkennung leisten. Während des Kollisionserkennungsprozesses werden nur für die Objektteile aktuelle Daten benötigt, die in potentiellen Kollisionsregionen liegen. Daher sollten nur für diese die in einem Kollisionserkennungsschritt aktuellen Daten berechnet werden. Dies wird schon dadurch gesteuert, daß aufgrund der Verwendung der *Hüllkörperhierarchien* nur solche Objektteile in die Basiskollisionserkennung gelangen, die in solchen potentiellen Kollisionsregionen liegen. Gleichzeitig ist es wahrscheinlich, daß für solche Objektteile mehr als ein Basiskollisionstest duchzuführen ist. Deshalb sollte die Objektrepräsentation so gestaltet sein, daß in jedem Kollisionserkennungsschritt die Daten eines Objektteiles höchstens einmal aktualisiert werden und dabei Werte, die mehrfach in den Basiskollisionserkennungsverfahren benutzt werden, vorberechnet werden.

Die Verwirklichung der beiden Aufgaben der Objektrepräsentation *geeignete geometrische Objekdarstellung* und *Unterstützung der Effizienz der Kollisionserkennungsverfahren* werden wir in der Folge näher erläutern.

Die gesamte Repräsentation eines Objektes besteht aus zwei großen Komponenten:

1. *Positionierung in der Szene*
2. *Geometrie*

Die *Positionierung in der Szene* hat während der Kollisionserkennung nicht nur Auswirkungen auf das Objekt als solches, sondern wird in den einzelnen Objektteilen (Ecke, Kante, Fläche) zu Berechnungen benötigt. Daher wird von den einzelnen Objektteilen aus ein Verweis auf die Datenstruktur mit der Objektposition gespeichert. Die Datenstruktur des Objektes bildet somit den Container für die beiden Teile *Positionierung* und *Geometrie* des Objektes. Dies ist in Abbildung 9.3 graphisch veranschaulicht.

9.3.1 Position

Durch die Speicherung der Position des Objektes an einer zentralen Stelle innerhalb des Objektes kann die Spezifikation der Bewegung des Objektes dort erfolgen und diese Information ist unmittelbar von allen Objektteilen aus verfügbar. Insbesondere können bei der Festlegung der Position mehrere Darstellungen der Position berechnet werden, die dazu führen, daß die Berechnungen der Kollisionserkennung und der weitergehenden Simulationen auf den Objektteilen beschleunigt durchgeführt werden können.

Die Objektteile ihrerseits verfügen über andere vorberechnete Daten, um die Effizienz der Berechnungen während der Kollisionserkennung und der weitergehenden Simulationen zu steigern. Diese hängen auf der einen Seite von der Spezifikation der Geometrie der Objekte und zum anderen von der Position des Objektes ab. Da die Spezifikation der Geometrie der Objekte während der Simulation unverändert bleibt, hängt deren Aktualisierung allein von der Position des Objektes ab. Diese sollte möglichst selten durchgeführt werden. Daher wird zu jedem Teil der Position des Objektes ein *Zeitstempel* (*time stamp*) gespeichert, der immer dann verändert wird, wenn die zugehörige Position geändert wird.

Die gesamte Position des Objektes besteht aus drei Teilen:

1. *aktuelle Position* $T_{aktuell}$
2. *Zielposition* T_{Ziel}

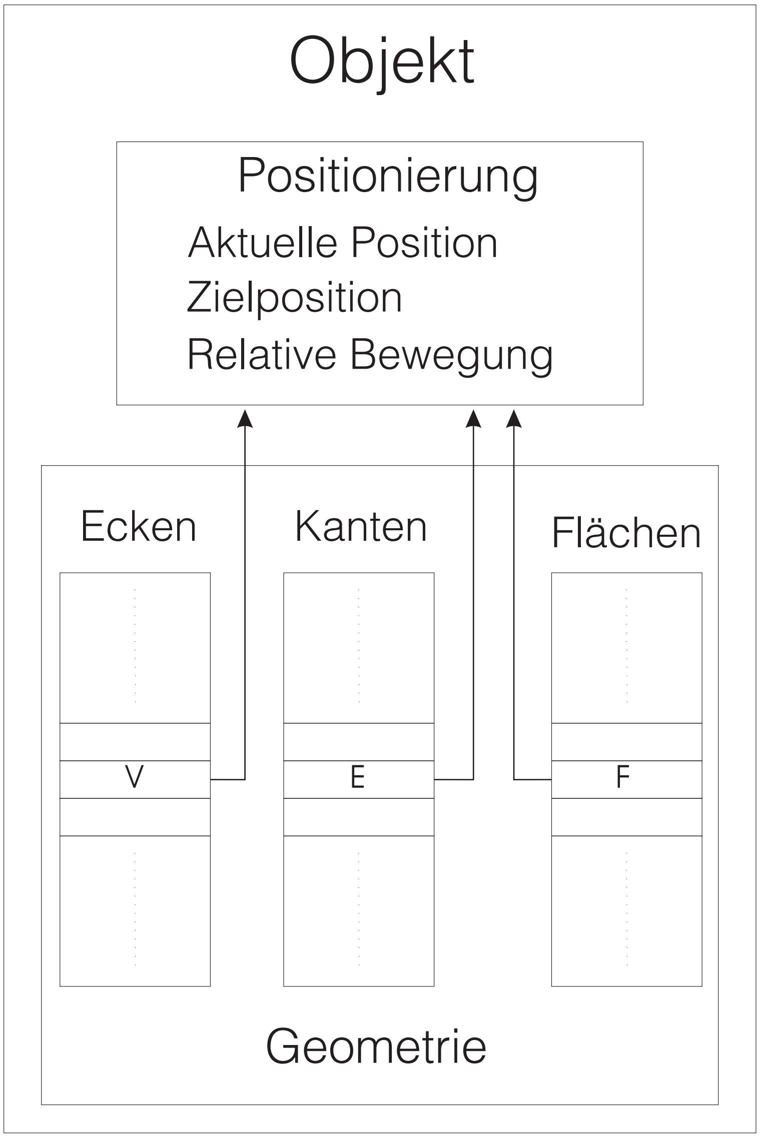

Abbildung 9.3: Teile der Objektdatenstruktur

3. *relative Bewegung* $T_{relativ}$

Jeder Teil der Position wird in verschiedenen Repräsentationen gespeichert. Die Transformationen werden als Transformationsmatrizen gespeichert. Diese sind geeignet zur Transformation von Koordinaten zwischen Koordinatensystem. Die Rotationsanteile der Transformationen werden zum einen als Rotationsmatrizen abgespeichert, die zur Transformation von Relativvektoren wie z.B. die Richtungsvektoren von Kanten oder die Normalen von Flächen geeignet sind, zum anderen als Quaternionen, die bei der Interpolation von Bewegungen benötigt werden. Zusätzlich wird die Translation als 3-dimensionaler Vektor gespeichert. Dieser dient z.B. zusammen mit der Rotationsmatrix zur Berechnung des überstrichenen Volumens von einfachen Hüllkörpern.

Für das *time stamping* werden zwei Zeitstempel benötigt. Einer für die *aktuelle Position* und ein weiterer für die *Zielposition*. Diese verändern sich, wenn die zugehörigen Positionsdaten verändert werden. Bei jeder Veränderung müssen dabei die Daten der *relativen Bewegung* aktualisiert werden.

9.3.2 Geometrie

Die Geometrie eines Objektes besteht aus *Ecken*, *Kanten* und *Flächen* zusammen mit deren topologischen Verknüpfungen. Die von ihnen gespeicherten Daten sind teilweise zeitinvariant während der Simulation wie z.B. die 3-dimensionalen Positionen der Eckpunkte, die Richtungsvektoren der Kanten, die Normalen der Flächen jeweils im Spezifikationskoordinatensystem des Objektes sowie die topologischen Informationen der Objektteile. Andere hingegen sind zeitvariant wie z.B. die aktuellen 3-dimensionalen Positionen der Ecken oder die Normalen der Flächen in der Zielposition des Objektes.

Die raumbezogenen Daten der einzelnen Objektteile werden in drei verschiedenen Varianten benötigt.

1. *Objektkoordinatensystem* $\boldsymbol{T}^{BS}$
2. *aktuelle Position im Weltkoordinatensystem* $\boldsymbol{T}^{WS}_{aktuell}$
3. *Zielposition im Weltkoordinatensystem* $\boldsymbol{T}^{WS}_{Ziel}$

Die Daten im *Objektkoordinatensystem* werden zum einen während der Berechnung der *Hüllkörperhierarchien* verwendet, zum anderen während des Kollisionserkennungsprozesses zur Aktualisierung der aktuellen Daten und der Zieldaten im Weltkoordinatensystem. Die Daten im *Objektkoordinatensystem* selbst sind während der Simulation zeitinvariant.

Die aktuellen und die Zieldaten im Weltkoordinatensystem werden während der Basiskollisionserkennung für die einzelnen Objektteile verwendet. Sie müssen aktualisiert werden, wenn sich die zugehörige Objektposition verändert. Damit die zugehörigen Daten nur dann aktualisiert werden, wenn sich die zugehörige Objektposition verändert hat, werden für jedes Objektteil zwei Zeitstempel eingeführt, die steuern, wann die zugehörigen Daten aktualisiert werden müssen. Dadurch kann für die zeitvarianten Daten der Objektteile die *lazy evaluation* realisiert werden, die zum einen die Daten nur dann aktualisiert, wenn diese tatsächlich im Kollisionserkennungsprozeß für Berechnungen benötigt werden, und zum anderen sicherstellt, daß die Aktualisierung in jedem Gültigkeitsintervall der zugehörigen Transformation des Objektes höchstens einmal durchgeführt wird.

Für Objekte, die innerhalb eines Kollisionserkennungsschrittes nicht bewegt werden, ist ihre aktuelle Position im Weltkoordinatensystem gleich ihrer Zielposition im Weltkoordinatensystem. Die Algorithmen zur Kollisionserkennung behandeln die Nichtbewegung eines Objektes in einem Schritt als einen Spezialfall eines bewegten Objektes. Zu Beginn der Simulation werden die Daten aller in der Szene befindlichen Objekte aktualisiert, wobei aktuelle Position gleich Zielposition der Objekte ist. Dies führt dazu, daß während des Kollisionserkennungsprozesses die Daten aller festen Objekte in der Szene niemals aktualisiert werden müssen.

Ecke

Eine *Ecke* v enthält als numerische Daten

- *3D-Objektkoordinaten* $\boldsymbol{p}_v^{BS}$,
- *3D-Weltkoordinaten in aktueller Position* $\boldsymbol{p}_{v(aktuell)}^{WS}$,
- *3D-Weltkoordinaten in Zielposition* $\boldsymbol{p}_{v(Ziel)}^{WS}$

und als topologische Daten Verweise auf

- *die Kante, dessen Startecke sie ist* e_v.

Kante

Gegenüber einer *Ecke* werden für eine Kante deutlich mehr Daten abgespeichert. Eine *Kante* e enthält als numerische Daten

- *Richtungsvektor im Objektkoordinatensystem* $\boldsymbol{d}_e^{BS}$,
- *Richtungsvektor im Weltkoordinatensystem in aktueller Position* $\boldsymbol{d}_{e(aktuell)}^{WS}$,
- *Richtungsvektor im Weltkoordinatensystem in Zielposition* $\boldsymbol{d}_{e(Ziel)}^{WS}$

und als topologische Daten Verweise auf

- *die Start- und Endecke* v_e^{Start} und v_e^{Ende},
- *die Fläche, die sie begrenzt* f_e,

- *die vorhergehende und die nachfolgende Kante* $e_{Vorgaenger}$ und $e_{Nachfolger}$,
- *die Rückwärtskante (falls existent und eindeutig)* $e_{rueckwaerts}$

und als Hilfsdaten zur Beschleunigung der Berechnungen während des Kollisionserkennungsprozesses

- $\boldsymbol{d}_e^{BS} \times \boldsymbol{n}_{f_e}^{BS}$, $\boldsymbol{d}_{e(aktuell)}^{WS} \times \boldsymbol{n}_{f_e(aktuell)}^{WS}$, $\boldsymbol{d}_{e(Ziel)}^{WS} \times \boldsymbol{n}_{f_e(Ziel)}^{WS}$,
- $\boldsymbol{p}_{v_e^{Start}}^{BS} \times \boldsymbol{p}_{v_e^{Ende}}^{BS}$, $\boldsymbol{p}_{v_e^{Start}(aktuell)}^{WS} \times \boldsymbol{p}_{v_e^{Ende}(aktuell)}^{WS}$, $\boldsymbol{p}_{v_e^{Start}(Ziel)}^{WS} \times \boldsymbol{p}_{v_e^{Ende}(Ziel)}^{WS}$.

Fläche

Eine *Fläche* f speichert als numerische Daten

- *Flächennormale im Objektkoordinatensystem* $\boldsymbol{n}_f^{BS}$,
- *Flächennormale im Weltkoordinatensystem in aktueller Position* $\boldsymbol{n}_{f(aktuell)}^{WS}$,
- *Flächennormale im Weltkoordinatensystem in Zielposition* $\boldsymbol{n}_{f(Ziel)}^{WS}$,
- *Abstand im Objektkoordinatensystem* d_f^{BS},
- *Abstand im Weltkoordinatensystem in aktueller Position* $d_{f(aktuell)}^{WS}$,
- *Abstand im Weltkoordinatensystem in Zielposition* $d_{f(Ziel)}^{WS}$

und als topologische Daten

- *Liste der Eckpunkte* V_f,
- *Liste der begrenzenden Kanten* E_f.

Will man die Daten einer Fläche im Prozeß der *lazy evaluation* aktualisieren, ist die Reihenfolge *Ecken* $\longrightarrow$ *Fläche* $\longrightarrow$ *Kanten* einzuhalten, da nur in dieser Reihenfolge die jeweils benötigten Daten aktualisiert zur Verfügung stehen.

Der geometrische Teil der *Objektrepräsentation* wird in Hinblick auf die Kollisionserkennung (siehe auch Abschnitt 9.4) in einem Vorberechnungsschritt erstellt und in einer Datei abgespeichert. Als Datenformat haben wir das OpenInventor$^{\text{TM}}$ Datenformat gewählt und dazu die OpenInventor$^{\text{TM}}$ Datenstrukturen erweitert (siehe [We94a]).

9.4 Kollisionserkennung

Die Realisierung der *Transformationsmathematik* und der *Objektrepräsentation* bilden die Basis für die Realisierung der Kollisionserkennung. Diese besteht selbst wieder aus drei großen Komponenten:

1. *Vorberechnung von Datenstrukturen*
2. *Berechnungen zum Start eines Simulationslaufes*
3. *Berechnungen während eines Kollisionserkennungsschrittes*

In allgemeiner Hinsicht könnten die ersten beiden Komponenten zusammengefaßt behandelt werden, denn sie gehören beide zur Vorberechnung. Für den praktischen Einsatz eines Kollisionserkennungsmoduls ist es jedoch wichtig, daß aufwendige Berechnungen nicht zu Beginn eines Simulationslaufes durchgeführt werden, sondern in einem separaten Vorverarbeitungsschritt, dessen resultierenden Datenstrukturen in einem geeigneten Format gespeichert werden und zum Start eines Simulationslaufes daraus wieder rekonstruiert werden. Dieser Tatsache wollen wir in der Realisierung gerecht werden.

Die vier Arten von Datenstrukturen, die während des Kollisionserkennungsschrittes benötigt werden, sind:

1. *Simulationskontrolle*
2. *Raumpartitionierung*
3. *Hüllkörperhierarchien*
4. *Objektrepräsentation*

Die *Simulationskontrolle* und die *Raumpartitionierung* werden beim Start eines Simulationslaufes aufgebaut, während die *Hüllkörperhierarchie* und der geometrische Teil der *Objektrepräsentation* zu einem früheren Zeitpunkt berechnet, als Dateien gespeichert und beim Start eines Simulationslaufes aus den Dateien aufgebaut werden.

Die Verknüpfung der Objekte mit ihrer *Hüllkörperhierarchie* und der *Raumpartitionierung* erfolgt durch eine Erweiterung der *Objektrepräsentation* aus Abbildung 9.3. Zu der Position und der Geometrie des Objektes werden nun zusätzlich die *Hüllkörperhierarchie* und die Daten für die *Raumpartitionierung* gespeichert, so daß sich für das Objekt in der Kollisionserkennung die erweiterte Struktur in Abbildung 9.4 ergibt.

Abbildung 9.4: Teile der Objektdatenstruktur in der Kollisionserkennung

Der Ablauf der Kollisionserkennung wird durch die *Simulationskontrolle* gesteuert. Sie stellt zum einen den Container für die benötigten anderen Datenstrukturen (*Raumpartitionierung*, *Objektrepräsentation mit Hüllkörperhierarchie*) und zum anderen die Datenstrukturen zur Steuerung der Kollisionserkennung zur Verfügung. Dies sind im wesentlichen die Menge der bewegten Objekte mit den zugehörigen Resultaten der Kollisionserkennung. Dabei stellen die *Raumpartitionierung* und die *Objektrepräsentationen mit Hüllkörperhierarchien* die *Bestandsdaten* dar und die Menge der bewegten Objekte die *Vorgangsdaten*. Die Komponenten der Simulationskontrolle sind in Abbildung 9.5 graphisch veranschaulicht.

Die *Vorgangsdaten* sind Teil der Schnittstelle zum Kollisionserkennungsmodul. Vor der Durchführung des Kollisionserkennungsschrittes werden die bewegten Objekte mit ihren Bewegungen registriert und bilden die Basis für den durchzuführenden Kollisionserkennungsschritt. Während des Kollisionserkennungsschrittes wird mit Hilfe der *Bestandsdaten* das Resultat der Kollisionserkennung für jedes bewegte Objekte berechnet. In einem

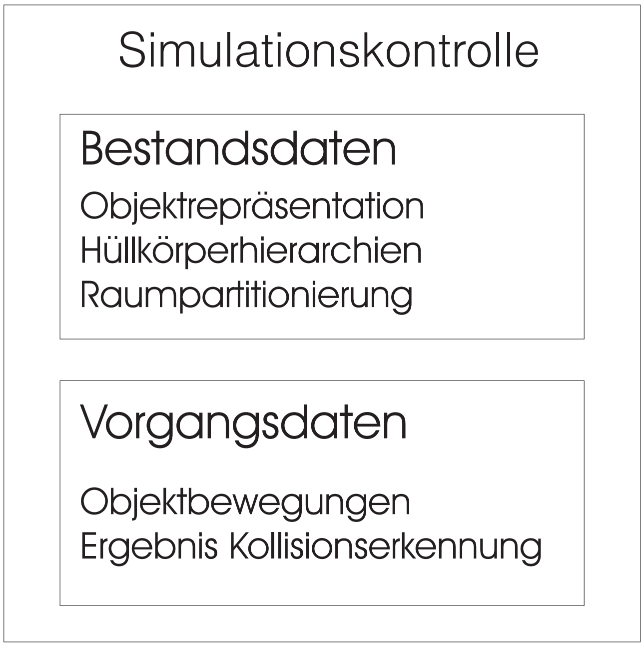

Abbildung 9.5: Teile der Simulationskontrolldatenstruktur

Nachbearbeitungsschritt werden für die bewegten Objekte basierend auf dem Ergebnis der Kollisionserkennung die resultierenden Bewegungen insbesondere auch mit Hilfe weitergehender Simulationen berechnet und die Positionen der Objekte modifiziert. Daraus ergibt sich der grobe Ablauf der Kollisionserkennung in Abbildung 9.6.

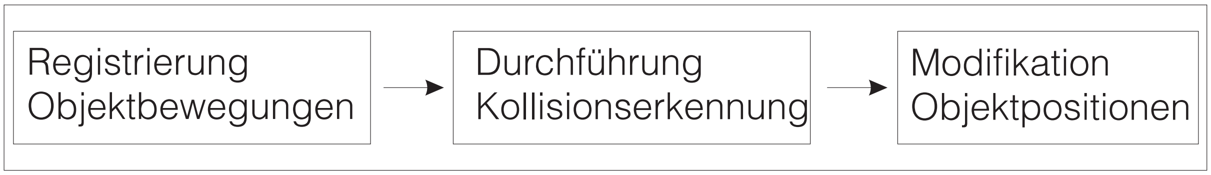

Abbildung 9.6: Grobablauf Kollisionserkennung

9.4.1 Vorberechnung von Datenstrukturen

Der geometrische Teil der *Objektrepräsentation* und die *Hüllkörperhierarchien* werden in einem Vorberechnungsschritt berechnet. Beide Datenstrukturen müssen aufeinander abgestimmt sein, da in der *Hüllkörperhierarchie* Verweise auf Objektteile (Ecken, Kanten, Flächen) der *Objektrepräsentation* enthalten sind. Der Aufbau des geometrischen Teils der *Objektrepräsentation* wurde bereits in Abschnitt 9.3 beschrieben. An dieser Stelle werden wir uns mit dem Aufbau der *Hüllkörperhierarchien* beschäftigen.

Hüllkörperhierarchie

Eine *Hüllkörperhierarchie* ist ein Baum. In dessen Knoten werden neben den Baumverweisen ein Hüllkörper, interpolierende Flächen, Daten zur Hierarchietraversierung bzw. zeitkritischen Berechnung und in Blattknoten die überdeckte Flächenmenge gespeichert. Betrachten wir uns die Verwendung der *Hüllkörperhierarchie* während des Kollisionserkennungsprozesses (siehe Kapitel 4), sehen wir, daß die Verweise auf die überdeckte Flächenmenge nur in den Blattknoten der Hierarchie benutzt werden. Erfolgt eine Beendigung der Kollisionserkennung innerhalb der Hierarchie, z.B. aufgrund eines unzureichenden Zeitbudgets, werden zur Berechnung des Ergebnisses der Kollisionserkennung entweder die Hüllkörper oder die interpolierenden Flächen verwendet, nicht aber die gesamte überdeckte Flächenmenge. Daher sind die Verweise auf die überdeckten Flächenmengen für innere Knoten des Baumes nicht nötig und nur bei den Blattknoten vorhanden.

Das *einseitige* Hierarchietraversierungsverfahren benötigt zur Auswahl des zu expandierenden Knotens die Qualität der Hüllkörper. Die *hybride* Hierarchietraversierungsstrategie benötigt zusätzlich noch den Kosten-Nutzen-Wert der Kinder für jeden Knoten. Diese Werte können zu diesem Vorberechnungszeitpunkt bestimmt und mit den einzelnen Knoten gespeichert werden. Der daraus resultierende Aufbau der Datenstruktur ist in Abbildung 9.7 graphisch veranschaulicht.

Die *Hüllkörperhierarchien* werden entsprechend der Verfahren in Kapitel 3 berechnet. Zum Speichern der *Hüllkörperhierarchien* haben wir wie bei dem geometrischen Teil der *Objektrepräsentation* erweiterte OpenInventor^{TM} Datenstrukturen verwendet (siehe [We94a]).

9.4.2 Berechnungen zum Start eines Simulationslaufes

Zum Start eines Simulationslaufes muß die Simulationsumgebung aufgebaut werden. Dazu werden für die Objekte der Szene die *Objektrepräsentationen*, die *Hüllkörperhierarchien* und die *Raumpartitionierungsdatenstruktur* aufgebaut.

Die *Objektrepräsentationen* und die *Hüllkörperhierarchien* werden aus den entsprechenden Dateien rekonstruiert und die *Raumpartitionierungsdatenstruktur* aufgebaut.

Als *Raumpartitionierungsdatenstruktur* ist das *1-dimensional sweep-and-prune* realisiert. Die Datenstruktur besteht aus drei Listen, eine für jede der Koordinatenachsen des Weltkoordinatensystems. Für jedes Objekt wird die aktuelle kleinste achsenorientierte Box berechnet, die als Beschreibung die Intervalle auf den Koordinatenachsen des Weltkoordinatensystems hat. Die Intervalle der einzelnen Objekte werden in die zugehörigen Listen einsortiert, woraus sich in jeder Dimension eine Menge von überlappenden Intervallen für jedes der Intervalle ergibt. Nachdem die Liste der Intervalle aufgebaut ist, wird durch einen Durchlauf über die drei Listen für jedes Objekt die Menge der Objekte berechnet, mit deren achsenorientierten Boxen sich die eigene achsenorientierte Box überlappt. Dies ist dann der Fall, wenn sich die Intervalle der achsenorientierten Boxen in allen drei Dimensionen überlappen.

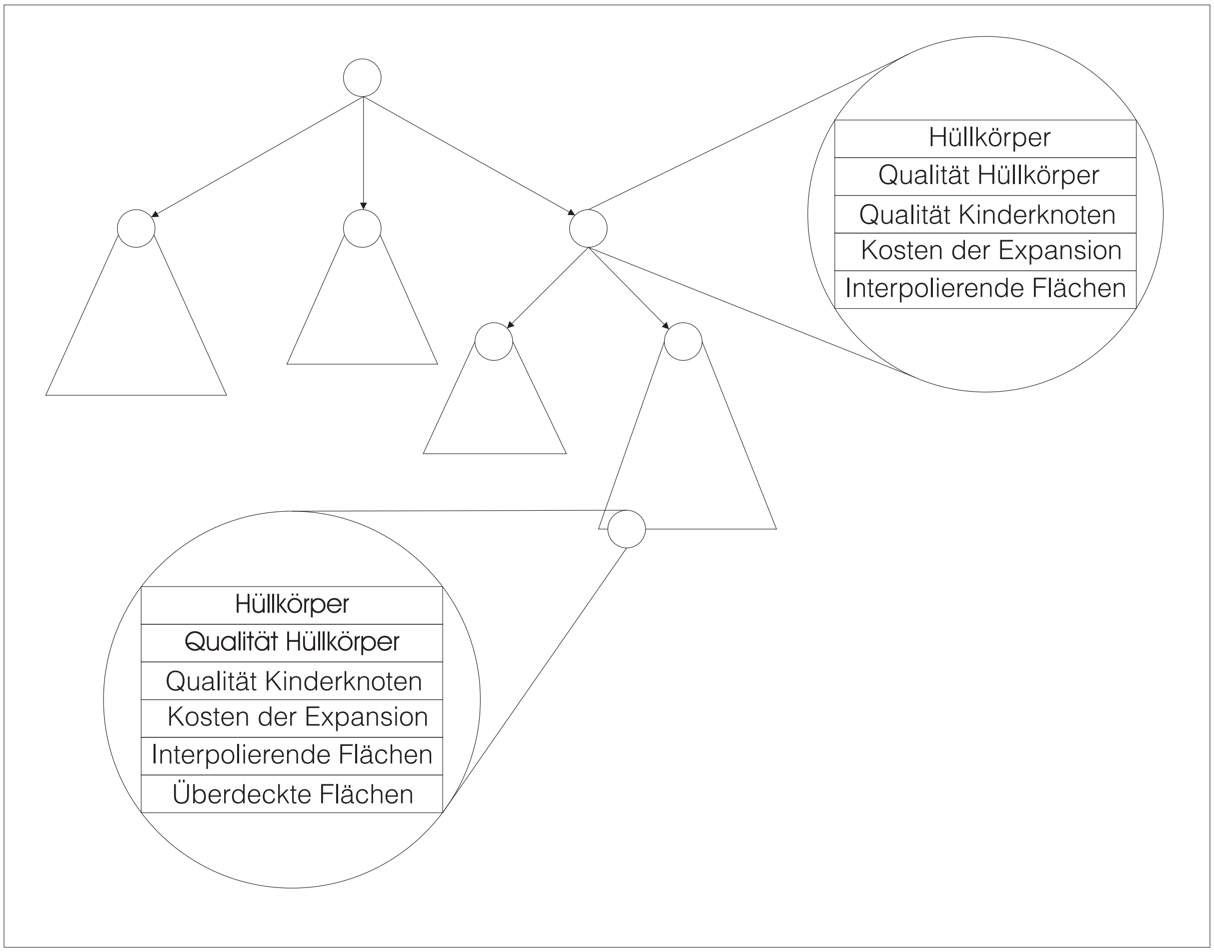

Abbildung 9.7: Datenstruktur für Hüllkörperhierarchien

9.4.3 Berechnungen während eines Kollisionserkennungsschrittes

In einem Kollisionserkennungsschritt wird zunächst die *Raumpartitionierungsdatenstruktur* für die bewegten Objekte aktualisiert und die Menge der durchzuführenden Objektpaartests ermittelt. Die Aktualisierung der *Raumpartitionierungsdatenstruktur* erfolgt in zwei Stufen:

1. *Berechnung der neuen achsenorientierten Boxen*
   Für jedes bewegte Objekt wird entsprechend der Bewegungsvorgabe und dem gewählten Berechnungsverfahren (*statisch* oder *dynamisch*) die neue achsenorientierte Box berechnet.

2. *Aktualisierung der Intervallisten*
   Die einzelnen Intervalle werden aktualisiert, indem die Grenzen der bestehenden In-

tervalle mittels *insertion sort* in den Listen verschoben werden. Dabei wird jeweils die Liste der überlappenden Intervalle und der daraus resultierenden Menge von überlappenden achsenorientierten Boxen bzw. der zugehörigen Objekte aktualisiert. Da die Bewegungen der Objekte in der Regel klein sind, ändert sich die relative Anordnung der Intervalle in den Listen nur geringfügig, so daß nur wenige Operationen während der Aktualisierung der *Raumpartitionierungsdatenstruktur* notwendig sind.

Nach diesem Schritt liegen für alle bewegten Objekte in der Szene die Menge der potientiell kollidierenden Objekte ohne weitere Berechungen sofort vor, so daß die Menge der durchzuführenden Objektpaartests schnell ermittelt werden kann.

In jedem Objektpaartest werden die *Hüllkörperhierarchien* gegeneinander auf Kollision getestet. Die Hierarchietraversierungsstrategie bestimmt dabei die durchzuführenden Tests von Hüllkörperpaaren. Einzelne Hüllkörper werden dadurch mehrfach in Tests verwendet. Um die Tests zu beschleunigen, sind analog zu den Objektteilen auch bei den Hüllkörpern vorberechenbare Werte identifiziert und werden mit Hilfe des *time stamping*-Verfahrens analog zu den Objektteilen nur höchstens einmal je Bewegungsschritt aktualisiert. Die Verwendung der Hüllkörper in der Kollisionserkennung hängt von der Art des verwendeten Kollisionserkennungsverfahren ab. Generell gilt jedoch, daß ein *statischer* Kollisionstest zwischen zwei Hüllkörpern durchgeführt wird, deren Berechnung sich jedoch unterscheiden kann (siehe Abschnitte 4.1 und 4.2). Daher gibt es für jeden Hüllkörper zwei Repräsentationen. Zum einen die in der Spezifikationslage des Objektes und eine zweite in der aktuellen Bewegungssituation. Dabei muß die in der aktuellen Bewegungssituation pro Kollisionserkennungsschritt höchstens einmal berechnet werden. Dies wird gesteuert durch das *time-stamping*-Verfahren.

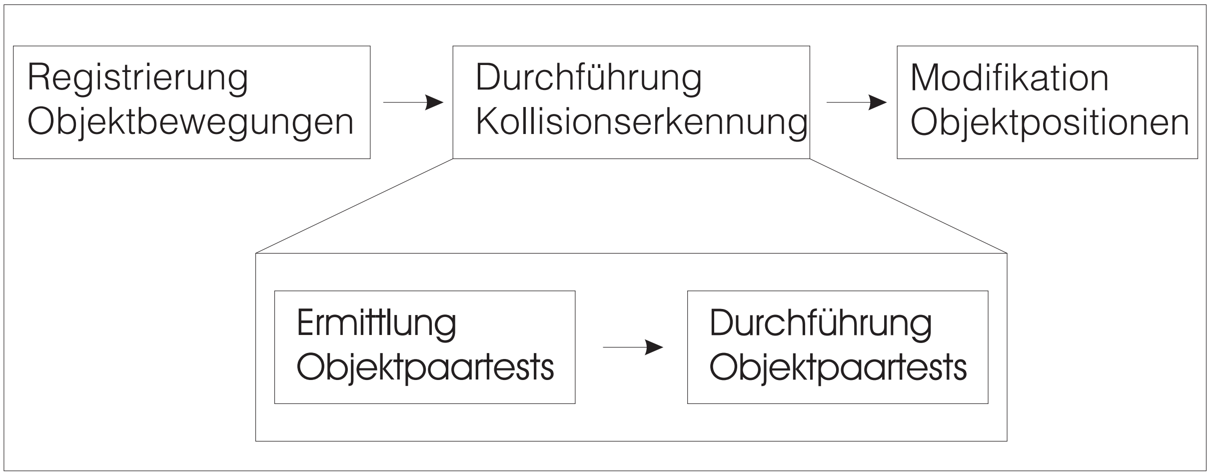

Abbildung 9.8: Durchführung der Kollisionserkennung

Einfache Hierarchietraversierungsstrategien benötigen keine elaborierten Datenstrukturen zur Auswahl des nächsten durchzuführenden Hüllkörperpaartest. Für diese genügen einfache Datenstrukturen wie ein *Stack* oder eine *Schlange*. Werden jedoch die Tests mit Hilfe der Güte der beteiligten Hüllkörper bewertet, kommen sortierte Datenstrukturen wie *Priority Queues* oder *Sorted sequences* zum Einsatz. Effiziente Implementierungen dieser Datenstrukturen basierend auf *Fibonacci-Heaps* bzw. *2-4-Bäumen* (vgl. [CLR90]) sind in LEDA realisiert.

Die Durchführung der Kollisionserkennung, deren grober Gesamtablauf in Abbildung 9.6 skizziert ist, unterteilt sich dann selbst wieder in die beiden Schritte

1. *Bestimmung der Objektpaartests* und
2. *Durchführung der Objektpaartests*,

deren Eingliederung in den Gesamtablauf in Abbildung 9.8 graphisch veranschaulicht ist.

Kapitel 10

Beispiele und Resultate

In den Kapiteln 3 bis 8 werden Verfahren zur echtzeitfähigen Kollisionserkennung unter Berücksichtigung der in Kapitel 2 erarbeiteten Anforderungen beschrieben. Die Ergebnisse der Realisierung innerhalb der *Virtual Reality* Softwareplattform **DBView** werden wir in der Folge anhand von Beispielen ausarbeiten.

Dabei werden wir in Abschnitt 10.1 das Laufzeitverhalten der verschiedenen in Kapitel 3 und 4 beschriebenen Verfahren zur Berechnung und Verwendungen von Hüllkörperhierarchien an praktisch relevanten Beispielen im Bereich Ein-/Ausbauuntersuchungen miteinander vergleichen. In Abschnitt 10.2 werden wir die Verfahren der *dynamischen* Kollisionserkennung betrachten. Das Zusammenspiel von Kollisionserkennung und Kontaktsimulation (zur Beschreibung der eingesetzten Kontaktsimulation vgl. [Bu98, BS98]) wird in Abschnitt 10.3 beschrieben. Schließlich werden wir in Abschnitt 10.4 ein Beispiel für das Zusammenspiel von Kollisionserkennung und Dynamiksimulation beschreiben. Eine vollständige Beschreibung der Dynamiksimulation und der Entwicklung des Zusammenspiels von Kollisionserkennung und Dynamiksimulation kann den Arbeiten von Sauer [Sa99, SS98] entnommen werden.

Ein großes Anwendungsfeld für *Virtual Reality* Anwendungen im Bereich der Automobilentwicklung entsteht derzeit im Bereich des *Packagings* (vgl. auch Abschnitt 1.2.4 auf Seite 10). Bei Ein-/Ausbau- bzw. Wartbarkeitsuntersuchungen wird analysiert, ob die Montage oder Demontage von Bauteilen einfach möglich ist oder nicht. Um die Entwicklungszeiten für Fahrzeuge zu verkürzen, dürfen derartige Untersuchungen nicht wie derzeit üblich am realen Prototypen durchgeführt werden, sondern verstärkt in früheren Entwicklungsstadien an virtuellen Prototypen, um Konstruktionsmängel in möglichst frühen Phasen der Entwicklung zu erkennen und zu beheben. Ein wichtiger Untersuchungsgegenstand sind dabei die Montage- bzw. Demontagemöglichkeiten von Aggregaten im Motorraum von Fahrzeugen. Die Daten für diese Untersuchungen stammen aus CAD-Programmen, mit deren Hilfe z.B. die Motoren von Fahrzeugen entwickelt werden. Für die Ein-/Ausbauuntersuchungen in *Virtual Reality* werden diese CAD-Daten in Oberflächenmodelle konvertiert. Wegen der geforderten hohen Genauigkeit bei diesen Untersuchungen entstehen bei der Konvertierung hochkomplexe Szenarien, wobei die Objekte dichtgedrängt sind. Diese Szenarien stellen eine Herausforderung für jede Kollisionserkennung dar und eignen sich daher hervorragend zum Test der Leistungsfähigkeit der verschiedenen Verfahren. Aus diesem Bereich haben wir uns ein Beispiel ausgesucht, das in den Abbildungen D.3 – D.7 veranschaulicht ist.

Die Aufgabenstellung ist dabei folgende:

- Schritt 1: Ausbau der Abdeckung des Fahrlichts (Abbildung D.4)
- Schritt 2: Ausbau der Glühbirne des Fahrlichts (Abbildung D.5)
- Schritt 3: Ausbau der Abdeckung des Zusatzscheinwerfers (Abbildung D.6)
- Schritt 4: Ausbau der Glühbirne des Zusatzscheinwerfers (Abbildung D.7)

Die getesteten Bewegungen wurden von einem Benutzer aufgezeichnet, der interaktiv mit Hilfe einer SPACE MOUSE® die Ausbauuntersuchung durchgeführt hat. Die Bewegungen enthalten alle Bewegungseingaben und damit sowohl kollidierende als auch kollisionsfreie Bewegungsabschnitte. Um in einem Vergleich äußere Einflüsse möglichst gering zu halten (z.B. durch das Betriebssystem) haben wir in einem ersten Schritt die durchschnittlichen Laufzeiten der Hüllkörper- und Basistests ermittelt. Daraus ergaben sich folgende Werte:

Test	Laufzeit (msecs.)
Kugel – Kugel	0.0090
Kugel – Iso-Box	0.0098
Kugel – Box	0.0110
Iso-Box – Iso-Box	0.0089
Iso-Box – Box	0.0136
Box – Box	0.0137

Basistest	Laufzeit (msecs.)
Kante – Fläche	0.0081

Während der durchgeführten Tests wurde jeweils die Anzahl der verschiedenen durchgeführten Tests ermittelt und mit Hilfe der oben aufgeführten mittleren Laufzeiten eine Vergleichszahl berechnet. Darüber hinaus wurden auch Tests zur Ermittlung der realen Laufzeiten der Verfahren durchgeführt. Alle Werte beziehen sich dabei auf Tests, die auf einer Silicon Graphics Onyx mit MIPS R10000 195 Mhz Prozessoren durchgeführt wurden.

10.1 Vergleich der Hüllkörperhierarchien

Den Vergleich der Hüllkörperhierarchien haben wir in verschiedene Abschnitte aufgeteilt. In einem ersten Schritt betrachten wir die vorgestellten Verfahren zur Berechnung der Hüllkörperhierarchien separat. Für jedes dieser Verfahren werden die Einstellungen ermittelt, bei denen es die beste Performance erzielt. In einem zweiten Schritt werden die derart ermittelten parametrisierten Verfahren miteinander verglichen und ein in diesem Anwendungsumfeld bestes Verfahren identifiziert.

10.1.1 Modifiziertes Zachmann-Verfahren

Um die besten Einstellungen für das *modifizierte* Zachmann-*Verfahren* zu ermitteln, haben wir zunächst unabhängig voneinander getestet, welche Einstellungen optimal sind bei festem Verzweigungsgrad und bei fester Verbesserungsrate und die besten Resultate der beiden Typen miteinander verglichen, um die insgesamt beste Einstellung zu ermitteln.

Grad

Die Ergebnisse der beiden Bewegungen für den Ausbau der Glühbirne des Fahrlichts und der Abdeckung des Fahrlichts in den Abbildungen D.8 und D.9 zeigen, daß die Grade 3 und 6 ein gutes Laufzeitverhalten zeigen und in jedem Fall eine Verbesserung gegenüber dem ursprünglich vorgeschlagenen Grad 2 darstellen.

Verbesserungsraten

Aus den Ergebnissen der beiden Bewegungen für den Ausbau der Glühbirne des Fahrlichts und der Abdeckung des Fahrlichts in den Abbildungen D.10 und D.11 geht hervor, daß Verbesserungsrate 0.5 in beiden Fällen eine gute Performance zeigt.

Gesamt

Vergleichen wir die besten festen Aufteilungsgrade und die beste Verbesserungsrate, so zeigt sich, daß in diesem Fall der Grad 3 insgesamt das beste Laufzeitverhalten aufweist, was in den Abbildungen D.12 und D.13 an den Beispielen des Ausbaus der Glühbirne des Fahrlichts und der Abdeckung des Fahrlichts deutlich wird.

10.1.2 Modifiziertes Gottschalk-Verfahren

Um die besten Einstellungen für das *modifizierte* Gottschalk-*Verfahren* zu ermitteln, haben wir analog zum *modifizierten* Zachmann-*Verfahren* zunächst unabhängig voneinander getestet, welche Einstellungen optimal sind bei festem Verzweigungsgrad und bei fester Verbesserungsrate und die besten Resultate der beiden Typen miteinander verglichen, um die insgesamt beste Einstellung zu ermitteln.

Grad

Die Ergebnisse der beiden Bewegungen für den Ausbau der Glühbirne des Fahrlichts und der Abdeckung des Zusatzscheinwerfers in den Abbildungen D.14 und D.15 zeigen, daß der Grad 6 in beiden Fällen eine gute Performance zeigt, wohingegen andere Grade sowohl gutes als auch schlechtes Laufzeitverhalten zeigen.

Verbesserungsrate

Aus den Ergebnissen der beiden Bewegungen für den Ausbau der Abdeckung des Fahrlichts und der Abdeckung des Zusatzscheinwerfers in den Abbildungen D.16 und D.17 geht hervor, daß Verbesserungsrate 0.75 in beiden Fällen eine gute Performance zeigt.

Gesamt

Vergleichen wir die besten festen Aufteilungsgrade und die beste Verbesserungsrate, sehen wir, daß die Verbesserungsrate 0.75 insgesamt das beste Laufzeitverhalten zeigt, was in den Abbildungen D.18 und D.19 an den Beispielen des Ausbaus der Abdeckung des Fahrlichts und der Abdeckung des Zusatzscheinwerfers deutlich wird. Aber auch die Erhöhung des Verzweigungsgrades auf 6 gegenüber dem von Gottschalk vorgeschlagenen Grad 2 verbessert die Performance des Verfahrens.

10.1.3 Heuristik ohne Zwischenoptimierung

Bei der *Heuristik ohne Zwischenoptimierung* haben wir in einem ersten Schritt getestet, welche Hüllkörpertypen am besten geeignet sind. Kugeln und Iso-Boxen ausschließlich als Hüllkörper verwendet zeigen dabei eine schlechte Performance, so daß zu entscheiden ist, ob Boxen oder alle Hüllkörper gemischt die beste Alternative darstellen. In den Tests zeigte sich durchgängig, daß Boxen als Hüllkörper am besten geeignet sind. Den Vergleich der beiden Alternativen haben wir in den Abbildungen D.20 und D.21 exemplarisch für die berechneten Hüllkörperhierarchien der Grade 5 und 6 dargestellt. In einem zweiten Schritt haben wir wieder unabhängig voneinander bestimmt, welcher Aufteilungsgrad und welche Verbesserungsrate zu einer möglichst guten Performance führt.

Grad

Bei der Untersuchung der Aufteilungsgrade zeigt der Ausbau der Abdeckung des Zusatzscheinwerfers auf, welche Aufteilungsgrade für diese Heuristik am besten geeignet sind. Die Ergebnisse in der Abbildung D.22 veranschaulichen, daß die Aufteilungsgrade 3, 4 und 5 die beste Performance erzeugen. Die beiden extremen Aufteilungen vom Grad 2 und 6 sind dagegen weniger geeignet.

Verbesserungsrate

Bei der Untersuchung der Verbesserungsraten zeigt ebenfalls der Ausbau der Abdeckung des Zusatzscheinwerfers auf, welche Verbesserungsraten für diese Heuristik am besten geeignet sind. Die Ergebnisse in der Abbildung D.23 verdeutlichen, daß die Verbesserungsrate 0.75 die beste Performance zeigt.

Gesamt

Vergleichen wir die besten festen Aufteilungsgrade und die beste Verbesserungsrate, sehen wir, daß die feste Verbesserungsrate 0.75 insgesamt das beste Laufzeitverhalten zeigt, was in der Abbildung D.24 am Beispiel des Ausbaus der Abdeckung des Zusatzscheinwerfers deutlich wird. Bei der Verbesserungsrate 0.75 sind die größten Laufzeitspitzen minimal, so daß in den Extremsituationen, bei denen sich die Objekte sehr nahe kommen und daher ein großer Berechnungsaufwand nötig ist, das bestmögliche Laufzeitverhalten erzeugt wird.

10.1.4 Heuristik

Bei der *Heuristik* haben wir wie bei der *Heuristik ohne Zwischenoptimierung* in einem ersten Schritt getestet, welche Hüllkörpertypen am besten geeignet sind. Analog zu der *Heuristik ohne Zwischenoptimierung* zeigen Kugeln und Iso-Boxen ausschließlich als Hüllkörper verwendet eine schlechte Performance, so daß zu entscheiden ist, ob Boxen oder alle Hüllkörper gemischt die beste Alternative darstellen. Den Vergleich der beiden Alternativen haben wir in den Abbildungen D.25 und D.26 exemplarisch für die berechneten Hüllkörperhierarchien der Grade 5 und 6 dargestellt. Wie bei der *Heuristik ohne Zwischenoptimierung* ergibt sich auch hier, daß Boxen als Hüllkörper am besten geeignet sind. In einem zweiten Schritt haben wir unabhängig voneinander bestimmt, welcher Aufteilungsgrad und welche Verbesserung zu einer möglichst guten Performance führt.

Grad

Bei der Untersuchung der Aufteilungsgrade zeigt der Ausbau der Glühbirne des Fahrlichts und der Abdeckung des Zusatzscheinwerfers auf, welche Aufteilungsgrade für diese Heuristik am besten geeignet sind. Die Ergebnisse in den Abbildungen D.27 und D.28 veranschaulichen, daß die Aufteilungsgrade 3 und 5 das beste Laufzeitverhalten erzeugen.

Verbesserungsrate

Bei der Untersuchung der Verbesserungsraten zeigt ebenfalls der Ausbau der Glühbirne des Fahrlichts und der Abdeckung des Zusatzscheinwerfers auf, welche Verbesserungsraten für diese Heuristik am besten geeignet sind. Die Ergebnisse in den Abbildungen D.29 und D.30 verdeutlichen, daß die Verbesserungsrate 0.75 die beste Performance zeigt.

Gesamt

Vergleichen wir die besten festen Aufteilungsgrade und die beste Verbesserungsrate, sehen wir, daß die feste Verbesserungsrate 0.75 insgesamt das beste Laufzeitverhalten zeigt, was in den Abbildungen D.31 und D.32 an den Beispielen des Ausbaus der Glühbirne des Fahrlichts und der Abdeckung des Fahrlichts deutlich wird.

10.1.5 *Branch-and-bound*-Verfahren

Das *Branch-and-bound*-Verfahren haben wir in der praktischen Anwendung nicht betrachtet, da die Berechnungszeit selbst für kleinere Beispiele enorm ist und die Vorberechnung für das gewählte Anwendungsbeispiel nicht durchführbar ist.

10.1.6 Modifiziertes Hochbaum-Shmoys-Verfahren

Das Verfahren von Hochbaum-Shmoys ist dazu ausgelegt, Hüllkörperhierarchien mit Kugeln zu berechnen. Wurden ausschließlich Kugeln als Hüllkörper verwendet, zeigten die Hüllkörperhierarchien unabhängig von der Art der Berechnung ein extrem schlechtes Laufzeitverhalten, bei dem die Zeiten pro Kollisionserkennungsschritt bis zu mehreren Sekunden betragen haben. Hüllkörperhierarchien mit Kugeln als Hüllkörper allgemein und die durch das modifizierte Verfahren von Hochbaum und Shmoys im besonderen sind für dichtgepackte, hochkomplexe Szenarien wie sie bei Ein-/Ausbauuntersuchungen auftreten nicht geeignet.

10.1.7 Vergleich der Verfahren

Wir haben für jedes einzelne Verfahren die besten Einstellungen ermittelt. Das Laufzeitverhalten der Verfahren mit diesen Einstellungen werden wir nun miteinander vergleichen, um das bestmögliche Verfahren zu ermitteln. Dazu verwenden wir die Verfahren folgendermaßen:

Verfahren	Einstellung
Modifiziertes ZACHMANN-Verfahren	Grad 3
Modifiziertes GOTTSCHALK-Verfahren	Verbesserungsrate 0.75
Heuristik ohne Zwischenoptimierung	Verbesserungsrate 0.75
Heuristik	Verbesserungsrate 0.75

Das Ergebnis des Vergleichs ist in den Abbildungen D.33 und D.34 graphisch veranschaulicht. Dort sehen wir, daß das *modifizierte* ZACHMANN-*Verfahren* mit seinen Iso-Boxen gegenüber den anderen Verfahren deutlich abfällt. Eliminieren wir dieses Verfahren aus dem Vergleich, sehen wir in den Abbildungen D.35 und D.36 an den Beispielen des Ausbaus der Glühbirne des Fahrlichts und der Abdeckung des Zusatzscheinwerfers, daß die *Heuristik* und die *Heuristik ohne Zwischenoptimierung* insgesamt das beste Laufzeitverhalten haben, jedoch auch das *modifizierte* GOTTSCHALK-*Verfahren* gute Werte erzielt.

Nach dem Vergleich auf der Basis von Vergleichszahlen, die mit Hilfe der Anzahl der durchgeführten Tests und einer mittleren Laufzeit für die Tests berechnet wurden, wollen wir den Vergleich auch auf der Basis realer Laufzeiten durchführen. Dazu haben wir die drei besten Verfahren

- modifiziertes GOTTSCHALK-Verfahren (Verbesserungsrate 0.75),
- Heuristik ohne Zwischenoptimierung (Verbesserungsrate 0.75) und
- Heuristik (Verbesserungsrate 0.75)

verwendet. Das Ergebnis ist in den Abbildungen D.37 – D.40 graphisch veranschaulicht. Das Testergebnis auf der Basis von Vergleichszahlen wird darin auch in realen Laufzeiten voll bestätigt.

10.1.8 Hierarchietraversierung

Im vorhergehenden Abschnitt haben wir die besten Berechnungsverfahren für Hüllkörperhierarchien anhand von Beispielen zusammen mit der jeweils bestmöglichen Parametrisierung ermittelt. In der Folge werden wir uns mit dem Einfluß des Hierarchietraversierungsverfahrens auf das Laufzeitverhalten der Kollisionserkennung beschäftigen. In Kapitel 4 haben wir drei verschiedene Hierarchietraversierungsstrategien vorgestellt:

- Einseitige Expansion
- Simultane Expansion
- Hybride Expansion

Diese drei Verfahren haben wir im Zusammenhang mit den Hüllkörperhierarchien getestet, die mittels der *Heuristik* und der *Heuristik ohne Zwischenoptimierung* berechnet wurden. Analog zum Vergleich der Hüllkörperhierarchien selbst haben wir auch hier wieder in einem ersten Schritt die Anzahl der jeweils durchgeführten Tests (Hüllkörper- und Basistests) ermittelt und mit den mittleren Laufzeitkosten gewichtet. Das Ergebnis des Vergleichstests kann den Abbildungen D.41 – D.44 für die Bewegungen der Abdeckung des Fahrlichts und des Zusatzscheinwerfers entnommen werden. Daraus ist ersichtlich, daß die *einseitige* Expansion stets das beste Laufzeitverhalten zeigt, wohingegen die *hybride* Expansion enttäuschend ist.

Das Ergebnis des idealisierten Vergleichstests spiegelt sich auch in den tatsächlichen Laufzeiten der einzelnen Verfahren wieder, was den Abbildungen D.45 – D.48 für die Bewegungen der Abdeckung des Fahrlichts und des Zusatzscheinwerfers entnommen werden kann.

10.1.9 Zusammenfassung

Wir haben die in Kapitel 3 und 4 beschriebenen Verfahren zur Berechnung und zur Traversierung von Hüllkörperhierarchien anhand eines praktischen Anwendungsszenarios aus dem Bereich des Packagings im Automobilbau miteinander verglichen. Dabei haben sich die *Heuristik*, die *Heuristik ohne Zwischenoptimierung* sowie das *modifizierte* GOTTSCHALK-*Verfahren* als beste Verfahren herauskristallisiert, wobei in allen Fällen die Parametrisierung identisch ist:

1. Als Hüllkörper werden ausschließlich *Boxen* verwendet, die auf minimale *Oberfläche* optimiert werden.

2. Als Aufteilungskriterium eines Knotens wird die *Verbesserungsrate 0.75* benutzt, was zu Hüllkörperhierarchien mit unterschiedlichem Verzweigungsgrad führt.

3. Als Hierarchietraversierungsstrategie wird die *einseitige Expansion* der Hüllkörperhierarchie gewählt.

Mit Hilfe dieser Einstellungen können in dem betrachteten Beispiel die tatsächlichen Laufzeiten erzielt werden, die in den Abbildungen D.37 – D.40 ersichtlich sind. Dabei sieht man, daß selbst im schlechtesten Fall am Beispiel der Bewegung für die Abdeckung des Fahrlichts und die Verwendung der Hüllkörperhierarchien, die mit dem *modifizierten* GOTTSCHALK-*Verfahren* berechnet wurden, eine Kollisionserkennungszeit von 40 msecs. pro Erkennungsschritt nicht überschritten wird, so daß eine interaktive Handhabung der Objekte in diesem Szenario jederzeit gewährleistet ist.

10.2 Dynamische Kollisionserkennung

Im vorhergehenden Abschnitt haben wir die in Kapitel 3 beschriebenen Verfahren zur Berechnung von Hüllkörperhierarchien und die in Kapitel 4 dargestellten Hierarchietraversierungsverfahren betrachtet. Für den Bereich der Ein-/Ausbauuntersuchungen haben wir unter Verwendung von *statischer* Kollisionserkennung das Berechnungsverfahren und die Traversierungsstrategie identifiziert, die das beste Laufzeitverhalten erzeugt. In diesem Abschnitt werden wir für diese Verfahren untersuchen, wie sie sich unter Verwendung von *dynamischen* Kollisionserkennungsverfahren bzgl. ihrer Performance verhalten. Die Untersuchungen haben wir ebenfalls am Beispiel der Ein-/Ausbauuntersuchung im Motorraum eines Fahrzeuges durchgeführt (vgl. Abbildungen D.3 – D.7). Als Hüllkörperhierarchien haben wir die mit der *Heuristik* und der *Heuristik ohne Zwischenoptimierung* berechneten verwendet.

Während der Untersuchung tauchte dann das Problem auf, das der *statischen* Kollisionserkennung anhaftet. Die Bewegung, die zum Vergleich der verschiedenen Hüllkörperhierarchien mit *statischer* Kollisionserkennung generiert worden ist, stellt keine letztendlich kollisionsfreie Bewegung dar, obwohl die Untersuchung der Bewegung mit *statischer* Kollisionserkennung dies ergibt. Dies zeigt praktisch auf, daß die Aussagen über die Kollisionsfreiheit von Ausbaupfaden, die mit Hilfe bisher bekannter *statischer* Kollisionserkennungsverfahren getroffen wurden, tatsächlich nur von eingeschränkter Aussagekraft sind.

Für den Vergleich des Laufzeitverhaltens der *statischen* mit den verschiedenen Verfahren der *dynamischen* Kollisionserkennung haben wir daher mit der SPACE MOUSE® interaktiv neue Bewegungen spezifiziert, die auch unter Verwendung der *dynamischen* Kollisionserkennung letztendlich kollisionsfrei sind. Für die *statische* Kollisionserkennung haben wir sowohl bei den Hüllkörpertests als auch bei den Basistests die *statischen* Kollisionserkennungsverfahren verwendet. Bei der *dynamischen* Kollisionserkennung haben wir für die Hüllkörpertests das Verfahren verwendet, das das überstrichene Volumen mit Hilfe des Hüllkörpers in dessen Start- und Endposition approximiert, und dieses mit den verschiedenen Verfahren der Basiskollisionserkennung gemischt. Für die Basiskollisionserkennung mit Hilfe der *linearen Abstandsabtastung* haben wir 5 Stützstellen zur Polygonzugapproximation verwendet. Der Laufzeitvergleich kann den Abbildungen D.49 – D.56 entnommen werden. Es ist klar, daß die Verwendung der *dynamischen* Kollisionserkennung zu einer erhöhten Laufzeit der Kollisionserkennung führt. Gleichzeitig ist ersichtlich, daß bei der Verwendung der *linearen Interpolation* und der *regula falsi* die Laufzeit der Kollisionserkennung in einem für *Virtual Reality* Anwendungen geeigneten Rahmen bleibt. Diese Verfahren wurden erfolgreich im Zusammenspiel mit den weitergehenden Simulationen benutzt, die in den nächsten beiden Abschnitten dargestellt werden.

10.3 Kollisionserkennung und Kontaktsimulation

Ein erstes Einsatzgebiet für die *dynamische* Kollisionserkennung im Zusammenhang mit weitergehenden Simulationen ist der Bereich der Kontaktsimulation. Diese ist insbesondere bei der Steigerung der Realitätsnähe von Ein-/Ausbauuntersuchungen im Rahmen von Packagingfragestellungen hilfreich. Die Lösungsansätze der verwendeten Kontaktsimulation werden in der Arbeit von Buck [Bu98] detailliert dargestellt. Ebenso werden dort die beiden Anwendungsbeispiele, die in den Abbildungen D.57 und D.58 graphisch veranschaulicht sind, näher erläutert. Allgemein setzt die Kontaktsimulation auf den Ergebnissen der *dynamischen* Kollisionserkennung auf und erzeugt mit Hilfe *constraint-basierter* Berechnungsverfahren Kontaktkräfte, die eine Durchdringung der Objekte verhindern und gleichzeitig ein intuitiv plausibles Verhalten der Objekte realisieren, wie z.B. das Gleiten eines Objektes über die Oberfläche eines anderen. Diese berechneten Kontaktkräfte sind in den beiden Abbildungen D.57 und D.58 als rote Pfeile dargestellt.

In beiden Anwendungsbeispielen sind die Bewegungen der Objekte interaktiv steuerbar und eine Latenzzeit für den Benutzer aufgrund der laufenden Simulation kaum wahrnehmbar.

In einem weiteren Anwendungsbeispiel wurde die *dynamische* Kollisionserkennung zusammen mit der Kontaktsimulation beim Ausbau der Glühbirnen aus dem Beispiel D.3 verwendet. Auch wenn der Ausbau ohne Kontaktsimulation möglich ist, so steigert die Verwendung der Kontaktsimulation die Realitätsnähe der Untersuchung und damit die Aussagekraft der Untersuchung für die Packagingfragestellung. Wie in den beiden anderen Beispielen sind die Bewegungen der Objekte interaktiv steuerbar und eine Latenzzeit ist für den Benutzer trotz weitaus größerer Komplexität des Kollisionserkennungsproblems kaum wahrnehmbar.

10.4 Kollisionserkennung und Dynamiksimulation

Ein zweites wichtiges Einsatzgebiet der *dynamischen* Kollisionserkennung ist der Bereich der Simulation physikalischer Objekteigenschaften. Die Lösungsansätze für die Simulation physikalischer Objekteigenschaften starrer Körper werden in der Arbeit von Sauer [Sa99, SS98] detailliert dargestellt. Das dargestellte Beispiel entstammt der Integration der *dynamischen* Kollisionserkennung mit Hüllkörperhierarchien in ein *impulsbasiertes* Simulationsverfahren. Abbildung D.59 stellt das mit diesem Verfahren berechnete physikalisch korrekte Fallen eines Stabes dar.

Es demonstriert eine erste erfolgreiche Verknüpfung von *dynamischer Kollisionserkennung* mit Hüllkörperhierarchien und der Simulation physikalischer Objekteigenschaften starrer Körper. Die Bewegung des Stabes mit *dynamischer* Kollisionserkennung und *impulsbasierte* Dynamiksimulation läuft in der *Virtual Reality* Plattform **DBView** echtzeitnah ab. Zur Beschreibung weiterer Verfahren aus dem Bereich der Simulation physikalischer Eigenschaften starrer Körper und deren Verknüpfung mit der *dynamischen* Kollisionserkennung sei an dieser Stelle auf die Arbeit von Sauer [Sa99, SS98] verwiesen.

Kapitel 11

Zusammenfassung und Ausblick

11.1 Zusammenfassung

Wir haben in dieser Arbeit ausgehend von den in Kapitel 2 aufgezeigten Problemkreisen der Kollisionserkennung

- Behandlung geometrisch komplexer Objekte und
- Behandlung einer großen Anzahl von Objekten

unter Berücksichtigung der Rahmenbedingungen

- Behandlung unstrukturierter Flächenmengen,
- Bestimmung *statischer* und *dynamischer* Kollisionserkennungsergebnisse,
- Behandlung nicht vorspezifizierter Bewegungen,
- mehrere gleichzeitig bewegte Objekte und
- Integration in ein *Virtual Reality* System und Echtzeitfähigkeit

systematisch ein System zur Kollisionserkennung ausgearbeitet, das allen Anforderungen gerecht wird.

Das System gliedert sich folgerichtig bezüglich der Problemkreise in zwei große Komponenten auf:

- *Behandlung einer großen Anzahl von Objekten* (siehe Kapitel 6)
- *Kollisionserkennung für ein Objektpaar* (siehe Kapitel 3, 4, 5)

die von einer *gemeinsamen Kontrolle* (siehe Kapitel 7, 8) gesteuert werden. Deren Implementierung wird in Kapitel 9 und die Resultate unserer Untersuchungen in Kapitel 10 beschrieben.

11.1.1 Steuerung der Kollisionserkennung

Die Bewegungen für die Objekte in der Szene bzw. Anfragen für feste Objekte stammen innerhalb von *Virtual Reality* Systemen aus verschiedenen Quellen. Zum einen aus Benutzerinteraktionen mittels Eingabegeräten wie z.B. Mouse, SPACE MOUSE® , Datenhandschuh oder aus Bewegungsvorgaben von Simulationen oder Animationen. Diese werden in der Kontrolle der Kollisionserkennung registriert. Nach Beendigung der Eingaben von Objektbewegungen werden die Bewegungen aller Objekte innerhalb eines Kollisionserkennungsschrittes simultan auf Kollision getestet.

Die Kollisionserkennung besteht dabei aus den beiden Phasen *Ermittlung aller durchzuführenden Objektpaartests* und *Durchführung der Objektpaartests*. Nach der Durchführung der Objektpaartests liegt für alle bewegten Objekte das Ergebnis der Kollisionserkennung vor. Diese Ergebnisse können dann in einem Nachbearbeitungsschritt von den anfragenden Systemen verarbeitet, die resultierenden Bewegungen berechnet und dem Kollisionserkennungssystem die daraus resultierenden neuen Objektpositionen mitgeteilt werden. Die Vorgehensweise der *Steuerung der Kollisionserkennung* ist in Kapitel 7 beschrieben.

11.1.2 Behandlung einer großen Anzahl von Objekten

Um Systeme mit einer großen Anzahl von Objekten handhaben zu können, benötigen wir Verfahren, die für die Anfrageobjekte – seien es nun bewegte oder feste Objekte – eine möglichst kleine Menge von Objekten bestimmt, die mit dem Anfrageobjekt kollidieren können und mit denen es in einem Objektpaartest auf Kollision hin getestet werden muß.

Das von uns beschriebene Verfahren basiert auf dem *1-dimensional sweep-and-prune*. In diesem Verfahren wird jedes Objekt durch eine achsenorientierte Box repräsentiert, die sich durch drei Intervalle auf den Koordinatenachsen des Weltkoordinatensystems beschreiben läßt. Zwei achsenorientierte Boxen überlappen sich genau dann, wenn sich alle zugehörigen Paare von Intervallen überlappen.

Die Rahmenbedingungen, die die Erweiterung des *1-dimensional sweep-and-prune* bestimmen, sind zum einen die *Berechnung statischer und dynamischer Kollisionserkennungsergebnisse* und zum anderen die *Behandlung mehrerer gleichzeitig bewegter Objekte*, wobei die *Behandlung nicht vorspezifizierter Bewegungen* und die *Integration in ein Virtual Reality System* durch die Vorauswahl des Verfahrens berücksichtigt wird.

Die Berechnung qualitativ geeigneter Kollisionserkennungsergebnisse reicht von der *statischen* bis zu verschiedenen Verfahren der *dynamischen* Behandlung von Objekten. Bei der Berechnung *statischer* Ergebnisse werden zur Ermittlung der durchzuführenden Objektpaartests die achsenorientierten Boxen der Objekte in der jeweiligen Endposition der Bewegung betrachtet, d.h. für alle in einem Kollisionserkennungsschritt bewegten Objekte werden die achsenorientierten Boxen in der jeweiligen Endposition berechnet, und diese fließen so in die Berechnungen des *1-dimensional sweep-and-prune* ein.

Die Berechnung *dynamischer* Ergebnisse ist so ausgelegt, daß das prinzipielle Vorgehen des *1-dimensional sweep-and-prune* nicht verändert wird. Jedes Objekt ist durch eine achsenorientierte Box repräsentiert. Im Unterschied zur rein *statischen* Behandlung von Objekten gehen jedoch die Bewegungen der Objekte in die Berechnung der achsenorientierten Boxen ein. Die zwei von uns beschriebenen *dynamischen* Verfahren berechnen jeweils eine achsenorientierte Box, die das überstrichene Volumen des Objektes während seiner Bewegung approximiert. Ein Verfahren approximiert dazu die achsenorientierten Boxen in der Start- und Endposition des Objektes mit einer neuen achsenorientierten Box und ein zweites berechnet für jedes Objekt die Länge der Bewegung, d.h. den maximalen Abstand, den ein Punkt des Objektes nach der Bewegung von seinem Ausgangspunkt aus haben kann, und vergrößert den Radius der aktuellen Box um diesen Wert.

Einzelheiten des *1-dimensional sweep-and-prune* wurden in Kapitel 6 und die Approximation überstrichener Volumina in Abschnitt 4.2 beschrieben.

11.1.3 Kollisionserkennung für ein Objektpaar

Aus den Berechnungen zur Behandlung einer *großen Anzahl von Objekten* geht eine Menge von durchzuführenden Objektpaartests hervor. Die Behandlung der *geometrischen Komplexität* stellt bei der Durchführung der Objektpaartests das Hauptproblem dar. Um dieses unter Berücksichtigung der restlichen Rahmenbedigungen bestmöglich behandeln zu können, haben wir uns für den Einsatz von Hüllkörperhierarchien entschieden, da sie geeignet sind, *geometrisch komplexe Objekte*, die als *unstrukturierte Flächenmengen* spezifiziert sind, behandeln zu können und gute Ansatzpunkte für die *Integration in Virtual Reality Systeme* unter Berücksichtigung der *Echtzeitfähigkeit* durch Verwendung von Methoden zur *zeitkritischen Berechnung* bieten.

Berechnung von Hüllkörperhierarchien

Bisherige Vorgehensweisen im Zusammenhang mit Hüllkörperhierarchien beschreiben im wesentlichen Verfahren und erklären deren Vorzüge meist anhand von Beispielen, wobei als Hüllkörper in der Regel ein fester Typ (Box, Iso-Box, Kugel) verwendet wird und Hierarchien festen Verzweigungsgrades entstehen.

Wir haben zur Berechnung von Hüllkörperhierarchien zunächst allgemeine Optimierungskriterien entwickelt, auf die unsere *top-down* Verfahren zur Berechnung von Hierarchien aufbauen. Die Optimierungskriterien basieren auf Gütemaßen von Hüllkörpern, die damit die Güte der Knoten bestimmen, denen sie zugeordnet sind. Als Gütemaße für Hüllkörper haben wir Volumen, Oberfläche und gerichteter Hausdorff-Abstand vom Hüllkörper zu der umhüllten Flächenmenge betrachtet. Ausgehend von der Güte der Knoten kann die Aufteilung eines Knotens bzw. der mit ihm assoziierten Flächenmenge in mehrere Knoten durch die Kombination der Güten der entstehenden Knoten bewertet werden. Der Aufteilungsschritt eines Knotens ist das Herzstück eines jeden *top-down* Verfahren zur Berechnung von

Hüllkörperhierarchien. Ziel der Aufteilung eines Knotens ist es dabei, entweder bei einem festgelegten Aufteilungsgrad die Güte der Aufteilung zu optimieren oder für eine vorgegebene Güte der Aufteilung den miminalen Aufteilungsgrad zu ermitteln, für den diese Güte erreicht werden kann, wobei in beiden Fällen für jeden neuen Knoten ein beliebiger der zur Verfügung stehenden Hüllkörpertypen verwendet werden kann. Im ersten Fall entstehen Hierarchien festen Verzweigungsgrades wohingegen im zweiten Fall Hierarchien mit unterschiedlichen Verzweigungsgraden entstehen.

Zum Erreichen der Ziele haben wir mehrere Verfahren vorgestellt:

- Branch-and-Bound Verfahren
- Heuristik mit Zwischenoptimierung
- Heuristik ohne Zwischenoptimierung

Zusätzlich haben wir bekannte Verfahren hinsichtlich ihrer Flexibilität modifiziert:

- Gottschalk-Verfahren
- Zachmann-Verfahren
- Hochbaum-Shmoys-Verfahren

Alle Verfahren sind für die Aufteilung eines Knotens mit festem Grad ausgelegt. Soll ein minimaler Grad bei der Aufteilung eines Knotens berechnet werden, wird das Verfahren angefangen von Grad zwei solange mit aufsteigendem Grad durchgeführt bis entweder das Aufteilungsziel oder ein maximaler Aufteilungsgrad erreicht ist.

Das *Branch-and-Bound* Verfahren ist in der Lage, lokal optimale Knotenaufteilungen zu berechnen, ist jedoch viel zu aufwendig in der Berechnung.

Die *Heuristik mit Zwischenoptimierung* ist ein inkrementelles Greedy-Verfahren. Es berechnet zunächst für Grad d eine Menge von d Flächen, die in unterschiedlichen Teilen der Aufteilung liegen werden und die initiale Aufteilung der ersten Flächen darstellt. Für diese werden die Hüllkörper und die Güte der bisherigen Aufteilung berechnet. Danach werden sukzessive die restlichen Flächen in die bisherige Partitionierung einsortiert, wobei jede Fläche so in einen Teil eingefügt wird, daß die Gesamtgüte der Aufteilung möglichst wenig verschlechtert wird. Dazu wird jede Fläche probehalber in jede der Unterteilungsmengen eingefügt und die daraus resultierende Güte der gesamten bisherigen Aufteilung berechnet, wobei für jedes versuchsweise Einfügen ein Hüllkörper optimiert werden muß.

Die *Heuristik ohne Zwischenoptimierung* ist ebenfalls ein inkrementelles Greedy-Verfahren. Analog zur *Heuristik mit Zwischenoptimierung* berechnet es zunächst eine initiale Aufteilung der ersten Flächen, jedoch werden für die Flächen keine Hüllkörper und deren Güten berechnet, sondern die Mittelpunkte der Flächen. Jetzt werden wieder sukzessive die verbleibenden Flächen in die bisherige Aufteilung einsortiert, wobei jede Fläche in die Unterteilungsmenge eingefügt wird, zu dessen Mittelpunkt sie den kleinsten Abstand hat. Dadurch verändert sich der Mittelpunkt der veränderten Unterteilungsmenge und alle bisher verteilten Flächen werden neu klassifiziert. Solange sich die Zugehörigkeit einzelner Flächen in den erreichten Unterteilungsmengen verändert, wird dieses Balancierungsverfahren bis zum Erreichen eines stabilen Zustands iterativ durchgeführt. Sind alle Flächen einsortiert, werden für die entstandenen Unterteilungsmengen einmalig Hüllkörper berechnet.

Die Berechnung von Hüllkörperhierarchien, die ein Teil der Vorberechnung zur Kollisionserkennung darstellt, wird in Kapitel 3 beschrieben.

Verwendung von Hüllkörperhierarchien

Während des Kollisionserkennungsprozesses werden Hüllkörperhierarchien verwendet, um in einem Objektpaartest eine Kollision möglichst schnell auszuschließen oder die Objektteile der beteiligten Objekte zu bestimmen, zwischen denen möglicherweise eine Kollision vorliegt. Dazu werden gesteuert von der Hierarchietraversierungsstrategie Paare von Hüllkörpern gegeneinander auf Kollision getestet.

Hierarchietraversierungsstrategien

Die Hierarchietraversierungsstrategie legt zum einen fest, welche Knoten beim Vorliegen einer Kollision zwischen einem Hüllkörperpaar expandiert werden; zum anderen bestimmt sie die Reihenfolge der Durchführung der Hüllkörperpaartests. Letzteres ist interessant im Zusammenhang mit *zeitkritischer Berechnung*. Neben den bisher vorgeschlagenen Verfahren zur *einseitigen* und zur *simultanen Expansion*, stellten wir ein Verfahren zur *hybriden Expansion* von Knoten vor. Bei der *einseitigen Expansion* wird nur ein Knoten einer der beiden Hierarchien expandiert, wohingegen bei der *simultanen Expansion* die Knoten beider Hierarchien expandiert werden. Die *einseitige Expansion* erzeugt weniger durchzuführende Hüllkörpertests, jedoch sind die dabei benutzten Hüllkörper qualitativ schlechter als die, die in den durchzuführenden Tests bei der *simultanen Expansion* benutzt werden. Die *hybride Expansion* wägt die Vorteile der *einseitigen Expansion* – geringer Anzahl von Hüllkörpertests – gegen die Vorteile der *simultanen Expansion* – bessere Qualität der Hüllkörper – ab und entscheidet sich je nach vorliegender Situation für die eine oder andere Vorgehensweise.

Für die Reihenfolge der durchzuführenden Hüllkörpertests gibt es ebenfalls verschiedene Vorgehensweisen. Wird die Kollisionserkennung bis zum Ende durchgeführt, d.h. werden keine Methoden der *zeitkritischen Kollisionserkennung* angewendet, um die Echtzeitfähigkeit der Kollisionserkennung zu gewährleisten, oder soll mit Hilfe der *statischen*

Kollisionserkennung nur die Existenz einer Kollision ermittelt werden, hat die Reihenfolge der Abarbeitung der Hüllkörpertests keinen Einfluß auf das Ergebnis der Kollisionserkennung. Einfache Verfahren führen die Hüllkörpertests entweder *first-in-first-out* oder *last-in-first-out* durch. Sortierte Verfahren führen die Hüllkörpertests in aufsteigender oder absteigender Reihenfolge der Güte der beteiligten Hüllkörper durch.

Die Hierarchietraversierungsstrategien wurden in Abschnitt 4.3 beschrieben.

Hüllkörpertest

In den bisherigen Verfahren werden dazu rein *statische* Verfahren vorgeschlagen, d.h. es werden beim Test zweier Hüllkörper diese in der Endposition der aktuellen Bewegung betrachtet und ein *statischer* Kollisionstest durchgeführt. Dieser ist für die verschiedenen Hüllkörpertypen in Abschnitt 4.1 beschrieben. Rein *statische* Verfahren können dazu führen, daß Kollisionen, die während einer Bewegung auftreten jedoch nicht bis zum Ende der Bewegung andauern (z.B. das Durchfahren einer Wand) übersehen werden. Um die Erkennung solcher Kollisionssituationen zu ermöglichen, müssen die Hüllkörpertests *dynamisch* durchgeführt werden. Die von uns vorgestellten Verfahren basieren darauf, das von einem Hüllkörper während einer Bewegung überstrichene Volumen durch einen anderen Hüllkörper zu approximieren und die resultierenden Hüllkörper *statisch* auf Kollision zu testen.

Dazu werden in Abschnitt 4.2 zwei Verfahren vorgestellt. Ein Verfahren umhüllt dazu die Hüllkörper in der Start- und der Endposition mit einem neuen Hüllkörper und ein zweites berechnet die maximale Länge der Bewegung und vergrößert den Hüllkörper in seiner aktuellen Lage um diesen Wert.

Basiskollisionserkennung

Die Verwendung von Hüllkörperhierarchien während des Kollisionserkennungsprozesses ermittelt potientielle Kollisionsregionen zwischen Objektpaaren. Reduziert sich der Abstand zwischen zwei Objekten derart, daß eine Trennung mit Hilfe der Hüllkörperhierarchie alleine nicht mehr möglich ist, müssen in der möglichen Kollisionsregion die Objektteile gegeneinander auf Kollision getestet werden. Dazu wird in der Literatur analog zu der *statischen* Kollisionserkennung für Hüllkörper ebenfalls ein *statisches* Verfahren eingesetzt, das in Abschnitt 5.2 beschrieben wird. Dieses besteht ausschließlich aus *Kante–Fläche*-Tests, denn zwei Flächen überschneiden sich genau dann, wenn eine Kante der einen Fläche die andere Fläche durchdringt. Mit Hilfe der *statischen* Kollisionserkennung kann nur das Entscheidungsproblem gelöst werden, ob in einer Konfiguration zweier Objekte eine Überlappung vorliegt.

Für den Fall der *dynamischen* Kollisionserkennung werden verschiedene Verfahren zur Bestimmung von Kollisionszeitpunkten benötigt. Berühren sich zwei Flächen, liegt dabei entweder ein *Kante–Kante*- oder ein *Ecke–Fläche*-Kontakt vor. Daher werden bei der *dynamischen* Kollisionserkennung alle *Kante–Kante*- und *Ecke–Fläche*-Paare der beiden Flächen gegeneinander auf Kollision getestet. Die von uns vorgestellten Verfahren zur Bestimmung von Kollisionszeitpunkten beruhen auf der Approximation der Abstandsfunktion zwischen den Objektteilen. Ein *Ecke–Fläche*- bzw. ein *Kante–Kante*-Paar definiert eine Abstandsfunktion zwischen ihren übergeordneten Geometrien, d.h. es ergeben sich Abstandsfunktionen *Punkt–Ebene* und *Gerade–Gerade*. Orientieren wird die Abstandsfunktion bezüglich der Normale der Ebene bzw. der Normale, die durch das Vektorprodukt der Richtungsvektoren der beiden Geraden bestimmt wird, ergeben sich in beiden Fällen vorzeichenbehaftete Abstandsfunktionen. Diese sind von den beteiligten Objektteilen und ihren durchgeführten Bewegungen abhängig. Da diese Abstandsfunktionen in der Regel keine einfachen Beschreibungen haben, werden ihre Nullstellen, die mögliche Kollisionszeitpunkte darstellen, mittels approximativer Verfahren bestimmt. Zur Approximation der Nullstellen haben wir verschiedene Verfahren verwendet:

- lineare Interpolation
- iterative Interpolation mit *Regula Falsi*
- quadratische Interpolation
- lineare Abtastung
- lineare Abtastung und *Regula Falsi*

Für diese approximativ bestimmten Nullstellen wird dann bestimmt, ob die Kollision innerhalb der betrachteten Objektteile aufgetreten ist. Die *dynamische* Basiskollisionserkennung ist in Abschnitt 5.3 beschrieben.

11.1.4 Zeitkritische Kollisionserkennung

Die *zeitkrische Kollisionserkennung* sichert die Echtzeitfähigkeit der Kollisionserkennung z.B. innerhalb von *Virtual Reality* Systemen. Sie ermöglicht die Berechnung approximativer Kollisionserkennungsergebnisse für den Fall, daß ein vorgegebenes Zeitbudget nicht ausreicht, um das Kollisionserkennungsproblem vollständig zu lösen.

Um für ein *Virtual Reality* System eine feste Bildgenerierungsrate zu erhalten, müssen die visualisierungsunabhängigen Aktivitäten, wie Ansteuerung der Interaktionsgeräte oder Simulationen, in der Lage sein, ihre Berechnungen innerhalb eines festen Zeitbudgets durchführen zu können. Dies bedeutet für die Kollisionserkennung, daß sie Verfahren bereitstellen muß, die sie in die Lage versetzen, innerhalb eines vorgegebenen Zeitbudgets ein möglichst gutes Kollisionserkennungsergebnis zu berechnen, das dann approximativ ist, wenn die Zeit nicht ausreicht, um das gesamte Kollisionserkennungsproblem exakt zu lösen.

Betrachten wir die beiden Phasen der Kollisionserkennung *Ermittlung der Objektpaartests* und *Durchführung der Objektpaartests* (siehe Abschnitt 7.2), sehen wir, daß die *Durchführung der Objektpaartests* den zeitaufwendigen Teil im Kollisionserkennungsprozeß darstellt. Es genügt daher, die Methoden der *zeitkritischen Berechnung* auf diesen Teil der Berechnung anzuwenden. Diese teilen sich in zwei Problemstellungen auf:

- Aufteilung des Zeitbudgets
- Verfahrensweise nach Ablauf des Zeitbudgets

Aufteilung des Zeitbudgets

Für die Aufteilung des Zeitbudgets auf die verschiedenen Objektpaartests und die dazugehörige Behandlung der Objektpaartests gibt es zwei prinzipielle Möglichkeiten:

1. *Separate Behandlung der Objektpaartests und Verteilung des Zeitbudgets*
   In diesem Fall wird die vorhandene Zeit auf die verschiedenen Objektpaartests verteilt. Eine Möglichkeit ist die Gleichverteilung der Zeit auf die durchzuführenden Tests, eine zweite die Verteilung abhängig zu machen vom Abstand der Kamera, denn Kollisionen in der Nähe des Betrachters sind wichtiger als die weit entfernt von ihm. Durch diese a priori Verteilung des Zeitbudgets kann es jedoch vorkommen, daß einzelne Objektpaartests mehr Zeit erhalten als sie benötigen, andere hingegen weniger. Dadurch können mitunter einige Objektpaartests nicht bis zum Ende durchgeführt werden, obwohl insgesamt genügend Zeit für alle Tests zur Verfügung gestanden hätte.
2. *Simultane Behandlung der Objektpaartests ohne Verteilung des Zeitbudgets*
   Das Problem der a priori Verteilung des Zeitbudgets auf die verschiedenen durchzuführenden Objektpaartests kann dadurch umgangen werden, daß alle Objektpaartests simultan behandelt werden. Dazu werden die ersten Hüllkörpertests aller Objektpaartests in eine gemeinsame Datenstruktur der durchzuführenden Hüllkörpertests eingefügt und im weiteren Verlauf somit alle Objektpaartests simultan durchgeführt solange die Zeit noch nicht vergangen ist. Sind am Ende der Zeit noch durchzuführende Hüllkörpertests vorhanden, werden für die zugehörigen Objektpaartests die *Verfahrensweisen nach Ablauf des Zeitbudgets* durchgeführt.

Verfahrensweisen nach Ablauf des Zeitbudgets

Liegt für einen Objektpaartest nach Ablauf des Zeitbudgets noch eine Menge durchzuführender Hüllkörpertests vor, muß aufgrund der vorliegenden Situation ein approximatives Kollisionserkennungsergebnis berechnet werden. Dazu kann man die Hüllkörper alleine oder *interpolierende Flächen* betrachen:

1. *Approximative Kollisionserkennung mit Hüllkörpern*
   Von den verbleibenden Hüllkörperpaaren wird dasjenige ausgewählt, das die beste Güte hat und als Indikator für das gesamte Kollisionserkennungsergebnis benutzt. Für dieses wird die Kollisionserkennung durchgeführt und das Ergebnis als Gesamtergebnis des verbleibenden Objektpaartests benutzt. Dieses Ergebnis ist nicht konservativ in der Hinsicht, daß mit dieser Vorgehensweise keine Kollision übersehen werden kann. Ist dies gewünscht, so kann beim Vorliegen von noch zu testenden Hüllkörperpaaren nicht ausgeschlossen werden, daß eine Kollision vorliegt ohne diese näher bestimmen zu können.

2. *Approximative Kollisionserkennung mit interpolierenden Flächen*
   Eine zweite Möglichkeit besteht darin, daß für jede Flächenmenge in der Hüllkörperhierarchie *interpolierende Flächen* berechnet werden, die die gesamte Flächenmenge möglichst gut approximieren. Von den verbleibenden Hüllkörperpaaren wird dasjenige ausgewählt, das die beste Güte hat. Für die beteiligten *interpolierenden Flächen* wird dann eine Basiskollisionserkennung durchgeführt und als Gesamtergebnis des verbleibenden Objektpaartests verwendet.

Die Verfahren der *zeitkritischen Berechnung* in der Kollisionserkennung werden in Kapitel 8 beschrieben.

11.2 Ausblick

Die von uns vorgestellten Verfahren sind in ihrer Konzeption dazu ausgelegt, Kollisionserkennung innerhalb eines umfassenderen Systems zur Echtzeitsimulation von Objektverhalten starrer Objekte zu ermöglichen. Erste erfolgreiche Beispiele für eine Einbindung in weitergehende Simulationen (*Dynamiksimulation* und *interaktive Objektmanipulation*) werden in dieser Arbeit bereits demonstriert. Gleichzeitig wurde bei der Bearbeitung der Integration der verschiedenen Komponenten eines solchen Simulationssystems klar, daß hinsichtlich des Zusammenspiels der Komponenten, insbesondere zwischen *Dynamiksimulation* und *Kollisionserkennung*, weitere Entwicklungen notwendig sind, die sich durch die Entwicklung der weitergehenden Simulationskomponenten ergeben und die Auswirkungen auf die Vorgehensweisen der Kollisionserkennung haben.

Ein Problem, dem sich im Simulationsbereich insbesondere die Kollisionserkennung stellen muß, ist, daß immer größere Szenarien hinsichtlich der Anzahl der Objekte und der geometrischen Komplexität der Einzelobjekte betrachtet werden. Die Auslöser für diese Entwicklung liegen in dem Wunsch nach Steigerung der Realitätsnähe von Echtzeitsimulationen als Teil von *Virtual Reality* Systemen, um die Aussagekraft von durch derartige Simulationen getroffenen Aussagen zu erhöhen. Derartige Bestrebungen dienen dazu, den Einsatz von Simulationen (z.B. im Bereich der Produktentwicklung) zu verstärken und dabei immer komplexere Produkte in kürzerer Zeit und mit geringeren Kosten entwickeln zu können. Die geometrische Komplexität der Objekte kann dabei in verschiedenen Formen erhöht werden:

- Größere Anzahl von Flächen
  Möglichkeiten, dieser Art der Komplexitätserhöhung zu begegnen, bestehen darin, die bekannten Verfahren weiterzuentwickeln und dabei insbesondere auf deren Parallelisierbarkeit zu achten bzw. die Parallelisierbarkeit der bestehenden Verfahren konsequent auszunutzen.

- Gekrümmte Oberflächen
  Man denke in diesem Bereich nur an die parametrisierten Oberflächen, die im Konstruktionsbereich mit Hilfe von CAD-Werkzeugen generiert werden. Ist man in der Lage solche Oberflächen zu behandeln, können größere Bereiche von Objekten in der Kollisionserkennung einheitlich bearbeitet werden anstatt – wie bisher – die Triangulierung solcher Flächen mit den herkömmlichen Methoden zu bearbeiten. Es besteht dann die Möglichkeit, diese Vorgehensweise mit Verfahren der hierarchischen Kollisionserkennung zu verbinden.

- Flexible Objekte anstatt starre Körper
  Durch die Erhöhung der Realitätstreue der Simulationen werden sich in Zukunft auch die Modelle der Objekte weg von der Simulation starrer Körper hin zur Simulation flexibler Körper entwickeln. Die Kollisionserkennung muß dem in ihren Verfahren gerecht werden.

Da die *Kollisionserkennung* ein zentrales Problem der Simulation von Objektverhalten in virtuellen Szenarien ist, stellt sich weiterhin die Frage, in wieweit die Entwicklung von Spezialhardware – analog zur Graphikhardware – das Problem der Kollisionserkennung in *Virtual Reality* Anwendungen entschärfen kann.

Anhang A

Transformationsmathematik

Bei der Kollisionserkennung werden als Transformationen *rigide Transformationen* betrachtet. Eine Transformation heißt *rigide*, wenn sie Abstände erhält, d.h. seien $\boldsymbol{x}$ und $\boldsymbol{y}$ zwei Punkte, auf die Transformation T wirkt, dann gilt für den Abstand zwischen $\boldsymbol{x}$ und $\boldsymbol{y}$

$$|\boldsymbol{x} - \boldsymbol{y}| = |T \cdot \boldsymbol{x} - T \cdot \boldsymbol{y}|$$

A.1 Repräsentation von Transformationen in der Objektinteraktion

Transformationen in der Objektinteraktion bestehen aus zwei elementaren Bestandteilen:

- Rotation
- Translation

Ein Punkt $\boldsymbol{x}$ wird mit Hilfe einer *rigiden Transformation* zu einem Punkt $\boldsymbol{x}'$ manipuliert, indem auf ihn zunächst die *Rotation* und danach die *Translation* wirkt.

$$\boldsymbol{x}' = Translation \circ Rotation(\boldsymbol{x})$$

Für die Repräsentation von *Translationen* wählen wir die Darstellung als Vektor.

$$t = \boldsymbol{t}$$

Der translatierte Punkt $\boldsymbol{x}^{trans}$ ergibt sich aus dem ursprünglichen Punkt $\boldsymbol{x}$ mittels $\boldsymbol{t}$ durch einfache Vektoraddition.

$$\boldsymbol{x}^{trans} = \boldsymbol{x} + \boldsymbol{t}$$

Die Berechnung der inversen Translation ergibt sich durch einfache Multiplikation des Translationsvektors $\boldsymbol{t}$ mit -1, d.h.

$$\boldsymbol{t}^{-1} = -\boldsymbol{t}.$$

Die Hintereinanderausführung von Translationen entspricht der Vektoraddition, d.h.

$$\boldsymbol{u} \circ \boldsymbol{t} = \boldsymbol{u} + \boldsymbol{t}.$$

Mit dieser Darstellung der Translation ergibt sich der manipulierte Punkt $\boldsymbol{x}'$ aus dem Punkt $\boldsymbol{x}$ mittels einer *rigiden Transformation* $T = (R, \boldsymbol{t})$ mittels

$$\boldsymbol{x}' = \mathit{Rotation}(\boldsymbol{x}) + \boldsymbol{t}.$$

A.1.1 Repräsentationen von Rotationen

Für *Rotationen* werden in der Literatur verschiedene Repräsentationsmöglichkeiten beschrieben (vgl. [Cr89]). Dabei sind folgende für die Szenenrepräsentation und Objektmanipulation in *Virtual Reality* Anwendungen im Hinblick auf Kollisionserkennung und Dynamiksimulation (vgl. [Sa99, SS98]) von Bedeutung:

- Achse und Winkel
- XYZ-Fixwinkel
- Quaternion
- Rotationsmatrix

Wir wollen nun in der Folge diese relevanten Darstellungen beschreiben und den Zusammenhang zwischen diesen aufzeigen.

Achse und Winkel

Eine erste anschauliche Möglichkeit eine Rotation R zu beschreiben ist *Achse und Winkel*. Die Rotation wird mit Hilfe einer Achse $\boldsymbol{r}$, die durch den Ursprung des Koordinatensystems geht, und einem Winkel φ, der die Drehung im Gegenuhrzeigersinn um die Achse angibt, beschrieben.

$$R = (\boldsymbol{r}, \varphi)$$

Nimmt man dabei an, daß die Rotationsachse normiert ist, dann ergibt sich die Möglichkeit, diese Rotationsrepräsentation auf einen einzigen Vektor im $\mathbb{R}^3$ zu reduzieren, indem die Länge des Richtungsvektors den Winkel angibt, d.h.

$$R = (\boldsymbol{r}, \varphi) \text{ mit } |\boldsymbol{r}| = 1 \quad \Longrightarrow \quad \varphi \cdot \boldsymbol{r}.$$

Mit Hilfe dieser Darstellung ergibt sich der rotierte Punkt $\boldsymbol{x}^{rot}$ aus $\boldsymbol{x}$ mittels

$$\begin{aligned}\boldsymbol{x}^{rot} &= (\boldsymbol{r}^T \cdot \boldsymbol{x}) \cdot \boldsymbol{r} + \cos\varphi \cdot \boldsymbol{r} \times (\boldsymbol{x} \times \boldsymbol{r}) + \sin\varphi \cdot \boldsymbol{r} \times \boldsymbol{x} \\ &= (1 - \cos\varphi) \cdot (\boldsymbol{r}^T \cdot \boldsymbol{x}) \cdot \boldsymbol{r} + \cos\varphi \cdot \boldsymbol{x} + \sin\varphi \cdot \boldsymbol{r} \times \boldsymbol{x}.\end{aligned}$$

Sind die betrachteten Winkel φ klein, so gilt, daß $\cos\varphi \approx 1$ und $\sin\varphi \approx \varphi$ ist. Unter diesen Annahmen ergibt sich der rotierte Punkt $\boldsymbol{x}^{rot}$ aus dem Punkt $\boldsymbol{x}$ durch

$$\boldsymbol{x}^{rot} = \boldsymbol{x} + (\varphi \cdot \boldsymbol{r}) \times \boldsymbol{x}.$$

Die Idee, daß ein Punkt $\boldsymbol{x}$ um eine Achse $\boldsymbol{r}$ im Gegenuhrzeigersinn dreht, ist an das räumliche Vorstellungsvermögen des Menschens angepaßt. Die mathematische Berechnung des rotierten Punktes ist dagegen eher kompliziert und die Hintereinanderausführung von Rotationen mittels der *Achse und Winkel*-Beschreibung ist unhandlich, denn für $(\boldsymbol{t}, \psi) = (\boldsymbol{r}, \varphi) \circ (\boldsymbol{s}, \theta)$ gilt:

$$\begin{aligned}\psi &= 2 \cdot \arccos\left(\cos\frac{\varphi}{2} \cdot \cos\frac{\theta}{2} - \left(\sin\frac{\varphi}{2} \cdot \boldsymbol{r}\right) \cdot \left(\sin\frac{\theta}{2} \cdot \boldsymbol{s}\right)\right) \\ \boldsymbol{t} &= \frac{1}{\sin\frac{\psi}{2}} \cdot \left(\cos\frac{\varphi}{2} \cdot \sin\frac{\theta}{2} \cdot \boldsymbol{s} + \cos\frac{\theta}{2} \cdot \sin\frac{\varphi}{2} \cdot \boldsymbol{r} + \left(\sin\frac{\varphi}{2} \cdot \boldsymbol{r}\right) \times \left(\sin\frac{\theta}{2} \cdot \boldsymbol{s}\right)\right)\end{aligned}$$

Die Berechnung der inversen Rotation ist dagegen einfach durch Multiplikation des Winkels oder der Achse mit -1 möglich:

$$(\boldsymbol{r}, \varphi)^{-1} = (\boldsymbol{r}, -\varphi) = (-\boldsymbol{r}, \varphi)$$

Die Umrechnungen in die anderen Darstellungen werden wie folgt vorgenommen:

1. *Achse und Winkel* $(\boldsymbol{r}, \varphi) \Longrightarrow$ *Rotationsmatrix* $\boldsymbol{R}$

$$\boldsymbol{R}(\boldsymbol{r}, \varphi) = (1 - \cos\varphi) \cdot \boldsymbol{r} \cdot \boldsymbol{r}^T + \cos\varphi \cdot \boldsymbol{I_3} + \sin\varphi \cdot \begin{pmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{pmatrix}$$

   Für kleine Winkel φ unter Verwendung von $\cos\varphi \approx 1$ und $\sin\varphi \approx \varphi$ vereinfacht sich die Berechnung zu

$$\boldsymbol{R}(\boldsymbol{r}, \varphi) = \boldsymbol{I_3} + \varphi \cdot \begin{pmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{pmatrix}.$$

2. *Achse und Winkel* $(\boldsymbol{r}, \varphi) \Longrightarrow$ *Quaternion* $(q_0, \boldsymbol{q})$

$$q_0(\boldsymbol{r}, \varphi) = \cos\frac{\varphi}{2} \qquad \boldsymbol{q} = \boldsymbol{r} \cdot \sin\frac{\varphi}{2}$$

3. *Achse und Winkel* $(\boldsymbol{r}, \varphi) \Longrightarrow$ *XYZ-Fixwinkel* (α, β, γ)

$$\begin{aligned}
\beta(\boldsymbol{r}, \varphi) &= \operatorname{arctan2}\left(\begin{array}{c} -\boldsymbol{r}_3 \cdot \boldsymbol{r}_1 \cdot (1-\cos\varphi) + \boldsymbol{r}_2 \cdot \sin\varphi, \\ \sqrt{(\boldsymbol{r}_1^2 \cdot (1-\cos\varphi) + \cos\varphi)^2 + (\boldsymbol{r}_2 \cdot \boldsymbol{r}_1 \cdot (1-\cos\varphi) - \boldsymbol{r}_3 \cdot \sin\varphi)^2} \end{array} \right) \\
\alpha(\boldsymbol{r}, \varphi) &= \operatorname{arctan2}\left(\tfrac{\boldsymbol{r}_2 \cdot \boldsymbol{r}_1 \cdot (1-\cos\varphi) + \boldsymbol{r}_3 \cdot \sin\varphi}{\cos\beta}, \tfrac{\boldsymbol{r}_1^2 \cdot (1-\cos\varphi) + \cos\varphi}{\cos\beta} \right) \\
\gamma(\boldsymbol{r}, \varphi) &= \operatorname{arctan2}\left(\tfrac{\boldsymbol{r}_3 \cdot \boldsymbol{r}_2 \cdot (1-\cos\varphi) + \boldsymbol{r}_1 \cdot \sin\varphi}{\cos\beta}, \tfrac{\boldsymbol{r}_3^2 \cdot (1-\cos\varphi) + \cos\varphi}{\cos\beta} \right)
\end{aligned}$$

Quaternion

Eine mathematisch verwandte Möglichkeit der Rotationsbeschreibung stellt das *Quaternion* dar. Es besteht ebenfalls aus vier Komponenten, die sich in einen dreidimensionalen Vektoranteil $\boldsymbol{q}$ und einen Skalaranteil q_0 aufspalten, d.h.

$$\mathfrak{q} = (q_0, \boldsymbol{q}).$$

Mit Hilfe dieser Darstellung ergibt sich der rotierte Punkt $\boldsymbol{x}^{rot}$ aus dem Punkt $\boldsymbol{x}$ durch

$$\boldsymbol{x}^{rot} = \frac{1}{|\mathfrak{q}|} \cdot \left[(q_0^2 - |\boldsymbol{q}|^2|) \cdot \boldsymbol{I_3} + 2 \cdot \boldsymbol{q} \cdot \boldsymbol{q}^T + 2 \cdot q_0 \cdot \begin{pmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{pmatrix} \right] \cdot \boldsymbol{x}.$$

Die Hintereinanderausführung zweier Rotationen $\mathfrak{p}$ und $\mathfrak{q}$ entspricht dabei einer Multiplikation zweier Quaternionen

$$\mathfrak{p} \cdot \mathfrak{q} = (p_0, \boldsymbol{p}) \cdot (q_0, \boldsymbol{q}) := (p_0 \cdot q_0 - \boldsymbol{p}^T \cdot \boldsymbol{q}, p_0 \cdot \boldsymbol{q} + q_0 \cdot \boldsymbol{p} + \boldsymbol{p} \times \boldsymbol{q})$$

und die inverse Rotation $\mathfrak{q}^{-1}$ ergibt sich zu

$$\mathfrak{q}^{-1} = \frac{\mathfrak{q}^*}{|\mathfrak{q}|^2},$$

wobei $\mathfrak{q}^*$ das zu $\mathfrak{q}$ konjugierte Quaternion bezeichnet mit

$$\mathfrak{q}^* = (q_0, -\boldsymbol{q}).$$

Die Umrechnungen in die anderen Darstellungen werden wie folgt vorgenommen:

1. *Quaternion* $(q_0, \boldsymbol{q}) \Longrightarrow$ *Achse und Winkel* $(\boldsymbol{r}, \varphi)$

$$\varphi(q_0, \boldsymbol{q}) = 2 \cdot \arctan\left(\frac{|\boldsymbol{q}|}{q_0}\right) \qquad \boldsymbol{r}(q_0, \boldsymbol{q}) = \frac{\boldsymbol{q}}{|\boldsymbol{q}|}$$

2. *Quaternion* $(q_0, \boldsymbol{q}) \Longrightarrow$ *XYZ-Fixwinkel* (α, β, γ)

$$\begin{aligned}
\beta &= \arctan 2\left(2 \cdot (\boldsymbol{q}_2 \cdot q_0 - \boldsymbol{q}_1 \cdot \boldsymbol{q}_3), \sqrt{(1 - 2 \cdot \boldsymbol{q}_2^2 - 2 \cdot \boldsymbol{q}_3^2)^2 + (2 \cdot (\boldsymbol{q}_1 \cdot \boldsymbol{q}_2 + \boldsymbol{q}_3 \cdot q_0))^2}\right) \\
\alpha &= \arctan 2\left(\frac{2 \cdot (\boldsymbol{q}_1 \cdot \boldsymbol{q}_2 + \boldsymbol{q}_3 \cdot q_0)}{\cos\beta}, \frac{1 - 2 \cdot \boldsymbol{q}_2^2 - 2 \cdot \boldsymbol{q}_3^2}{\cos\beta}\right) \\
\gamma &= \arctan 2\left(\frac{2 \cdot (\boldsymbol{q}_2 \cdot \boldsymbol{q}_3 + \boldsymbol{q}_1 \cdot q_0)}{\cos\beta}, \frac{1 - 2 \cdot \boldsymbol{q}_1^2 - 2 \cdot \boldsymbol{q}_2^2}{\cos\beta}\right)
\end{aligned}$$

3. *Quaternion* $(q_0, \boldsymbol{q}) \Longrightarrow$ *Rotationsmatrix* $\boldsymbol{R}$

$$\boldsymbol{R}(q_0, \boldsymbol{q}) = \frac{1}{q_0^2 + \boldsymbol{q}^2} \cdot \begin{pmatrix} 2 \cdot (q_0^2 + \boldsymbol{q}_1^2) - 1 & 2 \cdot (\boldsymbol{q}_1 \cdot \boldsymbol{q}_2 - \boldsymbol{q}_3 \cdot q_0) & 2 \cdot (\boldsymbol{q}_1 \cdot \boldsymbol{q}_3 + \boldsymbol{q}_2 \cdot q_0) \\ 2 \cdot (\boldsymbol{q}_1 \cdot \boldsymbol{q}_2 + \boldsymbol{q}_3 \cdot q_0) & 2 \cdot (q_0^2 + \boldsymbol{q}_2^2) - 1 & 2 \cdot (\boldsymbol{q}_2 \cdot \boldsymbol{q}_3 - \boldsymbol{q}_1 \cdot q_0) \\ 2 \cdot (\boldsymbol{q}_1 \cdot \boldsymbol{q}_3 - \boldsymbol{q}_2 \cdot q_0) & 2 \cdot (\boldsymbol{q}_2 \cdot \boldsymbol{q}_3 + \boldsymbol{q}_1 \cdot q_0) & 2 \cdot (q_0^2 + \boldsymbol{q}_3^2) - 1 \end{pmatrix}$$

XYZ-Fixwinkel

Eine weitere Möglichkeit, Rotationen zu beschreiben, besteht in der Angabe von drei Winkeln.

$$(\alpha, \beta, \gamma)$$

Dabei bestehen insgesamt 24 Möglichkeiten die drei Winkel zu interpretieren. Diese Interpretationsmöglichkeiten lassen sich in die zwei Gruppen der *Fixwinkel* und der *Eulerwinkel* einteilen, wobei jede der beiden Gruppen jeweils 12 Möglichkeiten enthält. *Fixwinkel* unterscheiden sich von *Eulerwinkel* dadurch, daß das Koordinatensystem, um dessen Achsen sukzessive gedreht wird, während der Rotation konstant bleibt, wohingegen sich dieses bei den *Eulerwinkeln* enstprechend den spezifizierten Drehungen bewegt. Aus diesen Spezifikationsmöglichkeiten haben wir die *XYZ-Fixwinkel* gewählt, weil sie für die Interpretation der Eingabewerte der SPACE MOUSE® bei der Interaktion mit Objekten in virtuellen Szenen am besten geeignet sind. Für eine komplette Übersicht sei auf [Cr89] verwiesen.

Der rotierte Punkt $\boldsymbol{x}^{rot}$ ergibt sich dabei aus dem Punkt $\boldsymbol{x}$ durch

$$\boldsymbol{x}^{rot} = \begin{pmatrix} \mathrm{c}\alpha \cdot \mathrm{c}\beta & \mathrm{c}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{s}\gamma - \mathrm{s}\alpha \cdot \mathrm{c}\gamma & \mathrm{c}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{c}\gamma + \mathrm{s}a \cdot \mathrm{s}\gamma \\ \mathrm{s}\alpha \cdot \mathrm{c}\beta & \mathrm{s}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{s}\gamma + \mathrm{c}\alpha \cdot \mathrm{c}\gamma & \mathrm{s}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{c}\gamma - \mathrm{c}\alpha \cdot \mathrm{s}\gamma \\ -\mathrm{s}\beta & \mathrm{c}\beta \cdot \mathrm{s}\gamma & \mathrm{c}\beta \cdot \mathrm{c}\gamma \end{pmatrix} \cdot \boldsymbol{x}.$$

Die inverse Rotation interpretiert als *ZYX-Fixwinkel* ergibt sich einfach als $(-\alpha, -\beta, -\gamma)$. Soll diese Rotation jedoch als *XYZ-Fixwinkel* beschrieben werden, so ergibt sich $(\sigma, \rho, \tau) =$

$(\alpha, \beta, \gamma)^{-1}$ zu

$$\rho = \arctan 2 \left(-\mathrm{c}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{c}\gamma - \mathrm{s}\alpha \cdot \mathrm{s}\gamma \ , \ \sqrt{(\mathrm{c}\alpha \cdot \mathrm{c}\beta)^2 + (\mathrm{c}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{s}\gamma - \mathrm{s}\alpha \cdot \mathrm{c}\gamma)^2} \right)$$
$$\sigma = \arctan 2 \left(\frac{\cos\alpha \cdot \sin\beta \cdot \sin\gamma - \sin\alpha \cdot \cos\gamma}{\cos\rho} \ , \ \frac{\cos\alpha \cdot \cos\beta}{\cos\rho} \right)$$
$$\tau = \arctan 2 \left(\frac{\sin\alpha \cdot \sin\beta \cdot \cos\gamma - \cos\alpha \cdot \sin\gamma}{\cos\rho} \ , \ \frac{\cos\beta \cdot \cos\gamma}{\cos\rho} \right)$$

mit $\mathrm{s} := \sin$ und $\mathrm{c} := \cos$.

Die Berechnung des rotierten Punktes mit Hilfe von *XYZ-Fixwinkeln* und deren Inversion, die quasi über die zugehörige *Rotationsmatrix* erfolgt, zeigt, daß *XYZ-Fixwinkel* zur Interpretation von Eingabewerten von Interaktionsgeräten geeignet, jedoch für weitergehende Rechnungen ungeeignet sind. Die Beschreibung einer Hintereinanderausführung von *XYZ-Fixwinkeln* würde an dieser Stelle einen erheblichen Aufwand darstellen ohne praktisch relevant zu sein. Daher verzichten wir an dieser Stelle darauf.

Die Umrechnungen in die anderen Darstellungen werden wie folgt vorgenommen:

1. *XYZ-Fixwinkel* $(\alpha, \beta, \gamma) \Longrightarrow$ *Achse und Winkel* $(\boldsymbol{r}, \varphi)$

$$\varphi(\alpha, \beta, \gamma) = \arccos \left(\frac{1}{2} \cdot (\mathrm{c}\alpha \cdot \mathrm{c}\beta + \mathrm{s}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{s}\gamma + \mathrm{c}\alpha \cdot \mathrm{c}\gamma + \mathrm{c}\beta \cdot \mathrm{c}\gamma - 1) \right)$$
$$\boldsymbol{r}(\alpha, \beta, \gamma) = \frac{1}{2 \cdot \sin(\varphi(\alpha, \beta, \gamma))} \cdot \begin{pmatrix} \cos\beta \cdot \sin\gamma - \sin\alpha \cdot \sin\beta \cdot \cos\gamma + \cos\alpha \cdot \sin\gamma \\ \cos\alpha \cdot \sin\beta \cdot \cos\gamma + \sin\alpha \cdot \sin\gamma + \sin\beta \\ \sin\alpha \cdot \cos\beta - \cos\alpha \cdot \sin\beta \cdot \sin\gamma + \sin\alpha \cdot \cos\gamma \end{pmatrix}$$

mit $\mathrm{s}\theta := \sin\theta$, $\mathrm{c}\theta := \cos\theta$ und $\theta = \alpha, \beta, \gamma$.

2. *XYZ-Fixwinkel* $(\alpha, \beta, \gamma) \Longrightarrow$ *Quaternion* $(q_0, \boldsymbol{q})$

$$q_0 = \frac{1}{2} \cdot \sqrt{1 + \cos\alpha \cdot \cos\beta + \sin\alpha \cdot \sin\beta \cdot \sin\gamma + \cos\alpha \cdot \cos\gamma + \cos\beta \cdot \cos\gamma}$$
$$\boldsymbol{q} = \frac{1}{4 \cdot q_0} \cdot \begin{pmatrix} \cos\beta \cdot \sin\gamma - \sin\alpha \cdot \sin\beta \cdot \cos\gamma + \cos\alpha \cdot \sin\gamma \\ \cos\alpha \cdot \sin\beta \cdot \cos\gamma + \sin\alpha \cdot \sin\gamma + \sin\beta \\ \sin\alpha \cdot \cos\beta - \cos\alpha \cdot \sin\beta \cdot \sin\gamma + \sin\alpha \cdot \cos\gamma \end{pmatrix}$$

3. *XYZ-Fixwinkel* $(\alpha, \beta, \gamma) \Longrightarrow$ *Rotationsmatrix* $\boldsymbol{R}$

$$\boldsymbol{R}(\alpha, \beta, \gamma) = \begin{pmatrix} \mathrm{c}\alpha \cdot \mathrm{c}\beta & \mathrm{c}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{s}\gamma - \mathrm{s}\alpha \cdot \mathrm{c}\gamma & \mathrm{c}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{c}\gamma + \mathrm{sa} \cdot \mathrm{s}\gamma \\ \mathrm{s}\alpha \cdot \mathrm{c}\beta & \mathrm{s}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{s}\gamma + \mathrm{c}\alpha \cdot \mathrm{c}\gamma & \mathrm{s}\alpha \cdot \mathrm{s}\beta \cdot \mathrm{c}\gamma - \mathrm{c}\alpha \cdot \mathrm{s}\gamma \\ -\mathrm{s}\beta & \mathrm{c}\beta \cdot \mathrm{s}\gamma & \mathrm{c}\beta \cdot \mathrm{c}\gamma \end{pmatrix}$$

mit $\mathrm{s}\varphi := \sin\varphi$, $\mathrm{c}\varphi := \cos\varphi$ und $\varphi = \alpha, \beta, \gamma$.

Rotationsmatrix

Die für Rechnungen mit Rotationen am besten geeignete Darstellung ist die der *Rotationsmatrix*. *Rotationsmatrizen* sind orthonormale 3x3-Matrizen über $\mathbb{R}^{3\times 3}$. Der rotierte Punkt $\boldsymbol{x}^{rot}$ ergibt sich aus dem Punkt $\boldsymbol{x}$ mittels $\boldsymbol{R}$ durch

$$\boldsymbol{x}^{rot} = \boldsymbol{R} \cdot \boldsymbol{x}.$$

Die Berechnung der inversen Rotation $\boldsymbol{R}^{-1}$ ist durch die Eigenschaft der Orthogonalität der *Rotationsmatrizen* einfach und es gilt:

$$\boldsymbol{R}^{-1} = \boldsymbol{R}^{T}$$

Die Hintereinanderausführung von Rotationen $\boldsymbol{R}$ und $\boldsymbol{P}$ entspricht der Matrixmultiplikation und es gilt:

$$\boldsymbol{P} \circ \boldsymbol{R} = \boldsymbol{P} \cdot \boldsymbol{R}$$

Die Umrechnungen in die anderen Darstellungen ergeben sich wie folgt:

1. *Rotationsmatrix* $\boldsymbol{R} \Longrightarrow$ *Achse und Winkel* $(\boldsymbol{r}, \varphi)$

$$\begin{aligned}
\varphi(\boldsymbol{R}) &= \arccos\left(\frac{1}{2} \cdot (\boldsymbol{R}_{11} + \boldsymbol{R}_{22} + \boldsymbol{R}_{33} - 1)\right) \\
\boldsymbol{r}(\boldsymbol{R}) &= \frac{1}{2 \cdot \sin\varphi(\boldsymbol{R})} \cdot \begin{pmatrix} \boldsymbol{R}_{32} - \boldsymbol{R}_{23} \\ \boldsymbol{R}_{13} - \boldsymbol{R}_{31} \\ \boldsymbol{R}_{21} - \boldsymbol{R}_{12} \end{pmatrix}
\end{aligned}$$

2. *Rotationsmatrix* $\boldsymbol{R} \Longrightarrow$ *Quaternion* $(q_0, \boldsymbol{q})$

$$\begin{aligned}
q_0(\boldsymbol{R}) &= \frac{1}{2} \cdot \sqrt{1 + \boldsymbol{R}_{11} + \boldsymbol{R}_{22} + \boldsymbol{R}_{33}} \\
\boldsymbol{q}(\boldsymbol{R}) &= \frac{1}{4 \cdot q_0(\boldsymbol{R})} \cdot \begin{pmatrix} \boldsymbol{R}_{32} - \boldsymbol{R}_{23} \\ \boldsymbol{R}_{13} - \boldsymbol{R}_{31} \\ \boldsymbol{R}_{21} - \boldsymbol{R}_{12} \end{pmatrix}
\end{aligned}$$

3. *Rotationsmatrix* $\boldsymbol{R} \Longrightarrow$ *XYZ-Fixwinkel* (α, β, γ)

$$\begin{aligned}
\beta &= \arctan 2\left(-\boldsymbol{R}_{31} \;,\; \sqrt{\boldsymbol{R}_{11}^2 + \boldsymbol{R}_{21}^2}\right) \\
\alpha &= \arctan 2\left(\frac{\boldsymbol{R}_{21}}{\cos\beta} \;,\; \frac{\boldsymbol{R}_{11}}{\cos\beta}\right) \\
\gamma &= \arctan 2\left(\frac{\boldsymbol{R}_{32}}{\cos\beta} \;,\; \frac{\boldsymbol{R}_{33}}{\cos\beta}\right)
\end{aligned}$$

A.1.2 Rigide Transformationsmatrizen

Rigide Transformationsmatrizen $\boldsymbol{T}$ sind 4×4-Matrizen über $\mathbb{R}^{4\times 4}$ mit

$$\boldsymbol{T} = \left(\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \\ \hline \mathbf{0} & 1 \end{array}\right),$$

wobei $\boldsymbol{R}$ eine *Rotationsmatrix* und $\boldsymbol{t}$ ein *Translationsvektor* ist. Ein transformierter Punkt $\boldsymbol{x}'$ ergibt sich aus dem Punkt $\boldsymbol{x}$ mittels einer *rigiden Transformationsmatrix* $\boldsymbol{T}$ zu

$$\boldsymbol{x}' = \boldsymbol{T} \cdot \boldsymbol{x} = \boldsymbol{R} \cdot \boldsymbol{x} + \boldsymbol{t}.$$

Die Inverse $\boldsymbol{T}^{-1}$ zu $\boldsymbol{T}$ ergibt sich zu

$$\boldsymbol{T}^{-1} = \left(\begin{array}{c|c} \boldsymbol{R}^{-1} & -\boldsymbol{R}^{-1} \cdot \boldsymbol{t} \\ \hline \mathbf{0} & 1 \end{array}\right) = \left(\begin{array}{c|c} \boldsymbol{R}^{T} & -\boldsymbol{R}^{T} \cdot \boldsymbol{t} \\ \hline \mathbf{0} & 1 \end{array}\right).$$

Die Hintereinanderausführung zweier Transformationen $\boldsymbol{T}_1$ und $\boldsymbol{T}_2$ entspricht der Matrixmultiplikation von 4×4-Matrizen und es gilt:

$$\boldsymbol{T}_2 \circ \boldsymbol{T}_1 = \boldsymbol{T}_2 \cdot \boldsymbol{T}_1 = \left(\begin{array}{c|c} \boldsymbol{R}_2 \cdot \boldsymbol{R}_1 & \boldsymbol{R}_2 \cdot \boldsymbol{t}_1 + \boldsymbol{t}_2 \\ \hline \mathbf{0} & 1 \end{array}\right)$$

Anhang B

Szenenrepräsentation

B.1 Einführung

Die Objekte in einer Szene müssen derart repräsentiert werden, daß eine geeignete funktionale Manipulation möglich ist. Als Quellen der Manipulation kommen dabei zwei generelle Techniken in betracht:

- Objektmanipulation durch Benutzerinteraktion
- Objektmanipulation durch Simulation

In beiden Fällen spielt die Szenenrepräsentation eine entscheidende Rolle. Durch sie wird festgelegt, welche Effekte Objektmanipulationen innerhalb der Szene haben.
Als Beispiel denke man dabei an die verschiedenen Gelenke eines Roboters:
Wird das Basisgelenk des Roboters manipuliert, verändern sich die Positionen aller anderen Gelenke ebenfalls. Gleichzeitig bleiben jedoch die relativen Positionen der Gelenke untereinander unverändert.
Wird der Endeffektor des Roboters manipuliert, verändert sich nur die Position dieses Gelenkes entsprechend seiner kinematischen Beschränkungen.
Eine etwas veränderte Situation ergibt sich, wenn Objekte nicht fest miteinander verbunden sind, sondern ihre Positionen aus einer freien relativen Positionierung herrühren.
Als Beispiel sei eine Tasse, die auf einem Tisch steht, angeführt:
Wird der Tisch bewegt, bewegt sich die Tasse mit dem Tisch – von dem Fall, daß die Tasse dabei vom Tisch fallen kann, sei an dieser Stelle einmal abgesehen.
Wird die Tasse bewegt, hat dies keinen Einfluß auf den Tisch. Bei der Tasse kommt jedoch noch hinzu, daßsich die Bewegungsabhängigkeiten für die Tasse ändern, wenn sie vom Tisch auf einen benachbarten Schrank befördert wird, da in diesem Fall die Position der Tasse nicht mehr von der des Tischs sondern von der des Schranks abhängt.

Im Falle des Roboters können die Abhängigkeiten der Gelenke untereinander durch Manipulation einzelner Objekte nicht verändert werden, wohingegen im Falle der Tasse und der Möbel eine Veränderung der Abhängigkeiten sinnvoll möglich ist. Eine Szenenrepräsentation mit geeigneten Objektmanipulationsmöglichkeiten muß dieser Tatsache Rechnung tragen und gleichzeitig effiziente funktionale Manipulationsmöglichkeiten für die darin enthaltenen Objekte ermöglichen.

B.2 Szenenrepräsentation

Zur Visualisierung von virtuellen Szenen müssen die Positionen aller in der Szene befindlichen Objekte in einem einheitlichen Koordinatensystem beschrieben werden. Dieses nennt man *Weltkoordinatensystem* WS.
Es ist zeitinvariant und das Koordinatensystem, in dem alle anderen Koordinatensystem beschrieben werden, d.h. ein Koordiantensystem CS beschreibt eine Transformation von Koordinaten in CS in Koordinaten in WS. Jedes Objekt in der Szene besitzt ein *körpereigenes Koordinatensystem* BS, in dem die Koordinaten des Objektes spezifiziert sind. Die Umrechung der Objektkoordinaten in Weltkoordinaten erfolgt mit Hilfe einer *Transformation* $T_{BS}^{WS}(t_i)$. Diese beschreibt die Position des Objektkoordinatensystems BS im Weltkoordinatensystem WS zum Zeitpunkt t_i. In der Folge wird folgende Notation benutzt:
${}^{O_j}T_{CS_1(t_i)}^{CS_2(t_k)}$ beschreibt die Transformation für Objekt O_j vom Koordinatensystem CS_1 zum Zeitpunkt t_i in das Koordinatensystem CS_2 zum Zeitpunkt t_k. Dabei bedeutet die Vereinfachung ${}^{O_j}T_{CS_1}^{CS_2}(t_i)$ dasselbe wie ${}^{O_j}T_{CS_1(t_i)}^{CS_2(t_i)}$.
Für ein Objekt O_j gelten folgende Eigenschaften. Sei

$x_{BS}(t_0)$	ein Punkt im Objektkoordinatensystem von O_j zum Zeitpunkt t_0,
$x_{BS}(t)$	ein Punkt im Objektkoordinatensystem von O_j zum Zeitpunkt t,
$x_{WS}(t_0)$	ein Punkt im Weltkoordinatensystem zum Zeitpunkt t_0,
$x_{WS}(t)$	ein Punkt im Weltkoordinatensystem zum Zeitpunkt t,
${}^{O_j}T_{BS}^{WS}(t_0)$	die Position des Objektkoordinatensystems von O_j im Weltkoordinatensystem zum Zeitpunkt t_0 und
${}^{O_j}T_{BS}^{WS}(t)$	die Position des Objektkoordinatensystems von O_j im Weltkoordinatensystem zum Zeitpunkt t.

Dann gilt:

1. $x_{BS}(t) = x_{BS}(t_0) \quad \forall t \geq t_0$
   Die Koordinaten eines Objektes im Objektkoordinatensystem BS sind zeitinvariant.

2. $x_{WS}(t) = T_{BS}^{WS}(t) \cdot x_{BS}(t) = T_{BS}^{WS}(t) \cdot x_{BS}(t_0)$
   Die Position eines Punktes eines Objektes im Weltkoordinatensystem ergibt sich durch die Multiplikation der Position des Objektkoordinatensystems $T_{BS}^{WS}(t)$ mit den Koordinaten $\boldsymbol{x}_{BS}(t)$ des Punktes im Objektkoordinatensystem BS.

In der Folge gehen wir davon aus, daß

$$T_{BS}^{WS}(t_0) = \text{ identische Transformation,}$$

d.h. die Beschreibung des Objektkoordinatensystems zum Zeitpunkt t_0 ist mit der des Weltkoordinatensystems identisch. Dadurch gilt zusätzlich folgende Beziehung:

$$x_{WS}(t_0) = x_{BS}(t_0) = x_{BS}(t) \quad \forall t \geq t_0$$

Der Gedanke, daß $T_{BS}^{WS}(t_0)$ die identische Transformation darstellt, ist konzeptioneller Natur. Liegt die Beschreibung einer Szene vor, in der bei der Beschreibung Transformationen auf die Objekte wirken, kann diese auf zwei Arten mit dieser konzeptionellen Sicht vereinbart werden:

- Die Szene mit den Transformationen, die auf die Objekte wirken, beschreibt bereits die Szene zum Zeitpunkt t_1, in der jedes Objekt, auf das eine Transformation wirkt, neu positioniert wurde.
- Die Beschreibung der Szene wird dahingehend modifiziert, daß auf die Objekte keine Transformationen wirken und die Transformationen in die Objektkoordinaten der Objekte hineingezogen wurden.

In den nächsten Abschnitten wollen wir nun zwei grundsätzliche Möglichkeiten der Szenenrepräsentation vorstellen:

- Hierarchische Szenenrepräsentation mit globalen Transformationen
- Hierarchische Szenenrepräsentation mit lokalen Transformationen

B.2.1 Nicht-hierarchische Szenenrepräsentation mit globalen Transformationen

Eine erste Möglichkeit der Szenenrepräsentation besteht darin, daß für jedes Objekt O_j genau eine Transformation ${}^{O_j}T_{BS}^{WS}(t_i)$ existiert, die die Transformation der Objektkoordinaten in Weltkoordinaten beschreibt.

Zusätzlich zu der Transformation werden zu jedem Objekt O_j die beiden Mengen

$M_{\rightarrow}$ Menge der Objekte O_k, deren Position durch O_j direkt beeinflußt wird, und
$M_{\leftarrow}$ Menge der Objekte O_k, deren Position die von O_j beeinflussen

verwaltet. Dabei ist darauf zu achten, daß durch die Repräsentation keine zyklischen Abhängigkeiten zwischen den Objekten entstehen. Dies kann dadurch sichergestellt werden, daß nur Baumoperationen zugelassen werden. Dadurch ergibt sich als hierarchische Darstellung ebenfalls ein Baum, der keine zyklischen Abhängigkeiten enthalten kann.

Auf ein Objekt O_j können nun *relative* und *absolute* Transformationen wirken, die unterschiedliche Positionsänderungen hervorrufen:

- **Relative Transformation**
  Wird das Objekt O_j mittels einer relativen Transformation $T_{WS(t_i)}^{WS(t_{i+1})}$ manipuliert, ergibt sich die neue Transformation der Objektkoordinaten in Weltkoordinaten von O_j zu
  $$^{O_j}T_{BS}^{WS}(t_{i+1}) \;=\; T_{WS(t_i)}^{WS(t_{i+1})} \;\cdot\; {}^{O_j}T_{BS}^{WS}(t_i).$$
  Gleichzeitig setzt sich diese Manipulation transitiv auf die von O_j beeinflußten Objekte $M_{\rightarrow}$ fort. Für alle Objekte $O_k \in M_{\rightarrow}$ ergibt sich die neue Transformation zu
  $$^{O_k}T_{BS}^{WS}(t_{i+1}) \;=\; T_{WS(t_i)}^{WS(t_{i+1})} \;\cdot\; {}^{O_k}T_{BS}^{WS}(t_i).$$

- **Absolute Transformation**
  Wird das Objekt O_j mittels einer absoluten Transformation $T_{BS}^{WS}(t_{i+1})$ positioniert, ergibt sich die neue Transformation der Objektkoordinaten in Weltkoordinaten von O_j zu
  $$^{O_j}T_{BS}^{WS}(t_{i+1}) \;=\; T_{BS}^{WS}(t_{i+1}).$$
  Diese absolute Positionierung von O_j überträgt sich nun transitiv auf alle von O_j beeinflußten Objekte $M_{\rightarrow}$, indem aus der alten und der neuen Transformation von O_j eine *relative* Transformation berechnet wird mittels
  $$T_{WS(t_i)}^{WS(t_{i+1})} = {}^{O_j}T_{BS}^{WS}(t_{i+1}) \;\cdot\; ({}^{O_j}T_{BS}^{WS}(t_i))^{-1} = {}^{O_j}T_{BS}^{WS}(t_{i+1}) \;\cdot\; {}^{O_j}T_{WS}^{BS}(t_i).$$
  Diese wirkt nun analog zu oben auf alle Objekte $O_k \in M_{\rightarrow}$ und deren neue Transformationen ergeben sich zu
  $$^{O_k}T_{BS}^{WS}(t_{i+1}) \;=\; T_{WS(t_i)}^{WS(t_{i+1})} \;\cdot\; {}^{O_k}T_{BS}^{WS}(t_i).$$

Die Menge $M_{\rightarrow}$ wird dazu genutzt, um effizient die Menge aller Objekte O_k zu verwalten, die durch die Manipulation von O_j ebenfalls beeinflußt werden.
$M_{\leftarrow}$ dient dazu, bei einer Änderung der Abhängigkeiten die neuen Abhängigkeiten effizient repräsentieren zu können.

Wird die Abhängigkeit des Objektes O_j von den Objekten $O_k \in M_{\leftarrow}$ gelöscht, muß für alle O_k die Menge ${}^{O_j}M_{\rightarrow} \cup \{O_j\}$ aus der Menge ${}^{O_k}M_{\rightarrow}$ gelöscht werden. Gleichzeitig muß für alle Objekte $O_l \in {}^{O_j}M_{\rightarrow} \cup \{O_j\}$ die Menge ${}^{O_j}M_{\leftarrow}$ aus der Menge ${}^{O_l}M_{\leftarrow}$ gelöscht werden. Für O_j wird diese dadurch zu $\emptyset$, d.h. O_j wird durch kein anderes Objekt beeinflußt.

Wird das Objekt O_j von einem Objekt O_k abhängig gemacht, muß für alle Objekte $O_l \in {}^{O_j}M_{\rightarrow} \cup \{O_j\}$ zu der Menge ${}^{O_j}M_{\leftarrow}$ die Menge ${}^{O_k}M_{\leftarrow}\{O_k\}$ hinzugefügt werden und für alle Objekte $O_m \in {}^{O_k} M_{\leftarrow} \cup \{O_k\}$ muß die Menge ${}^{O_j}M_{\rightarrow} \cup \{O_j\}$ zu der Menge ${}^{O_m}M_{\rightarrow}$ hinzugefügt werden.

Durch diese beiden Operationen können beliebige Unterbäume der *hierarchischen Szenenrepräsentation* umgehängt und die daraus resultierenden Abhängigkeiten der Objekte untereinander modifiziert werden.

Die *hierarchische Szenenrepräsentation mit globalen Transformationen* hat ihre Vorteile darin, daß lokal bei jedem Objekt O_j die aktuelle Transformation ${}^{O_j}T_{BS}^{WS}(t_i)$ gespeichert ist, so daß ein schneller Zugriff auf die aktuelle Position im Weltkoordinatensystem möglich ist. Nachteilig ist, daß bei der Manipulation eines Objektes O_j rekursiv die Transformationen aller von O_j abhängigen Objekte modifiziert werden müssen.

B.2.2 Hierarchische Szenenrepräsentation mit lokalen Transformationen

Bei dieser hierarchischen Szenenrepräsentation werden ähnlich wie bei der vorherigen zu jedem Objekt O_j die Zusatzinformationen

$M_{\rightarrow}$	Menge der Objekte O_k, deren Position durch O_j direkt beeinflußt wird, und
$P_{\leftarrow}$	Menge der Objekte, deren Position die von O_j beeinflussen, angeordnet im Pfad von der Wurzel des Baum bis zu O_j

verwaltet. Dabei ist darauf zu achten, daß durch die Repräsentation keine zyklischen Abhängigkeiten zwischen den Objekten entstehen. Dies kann dadurch sichergestellt werden, daß nur Baumoperationen zugelassen werden. Dadurch ergibt sich als hierarchische Darstellung ebenfalls ein Baum, der keine zyklischen Abhängigkeiten enthalten kann.

Zusätzlich zu diesen Objektabhängigkeiten werden im Gegensatz zur *hierarchischen Szenenrepräsentation mit globalen Transformationen* zu jedem Objekt nur die Transformationen ${}^{O_j}_{local}T_{BS}^{WS}(t_i)$ gespeichert, die auf O_j direkt gewirkt haben.
Um die globale Transformation ${}^{O_j}T_{BS}^{WS}(t_i)$ zu erhalten, müssen die lokalen Transformationen der Objekte berücksichtigt werden, die die Position von O_j beeinflussen. Das sind alle Objekte, die auf dem Pfad von der Wurzel des Szenenbaumes zu O_j liegen. Dabei werden die lokalen Transformationen ${}_{local}T_{BS}^{WS}(t_i)$ als relative Transformation $T_{WS}^{WS}(t_i)$ für O_j interpretiert.
Sei $P(O_j) = O_j^{(n)} \rightarrow \cdots \rightarrow O_j^{(1)} \equiv O_j$ der Pfad von der Wurzel des Szenenbaumes zu O_j, dann gilt:

$$\begin{aligned} {}^{O_j}T_{BS}^{WS}(t_i) &= {}^{O_j^{(n)}}T_{WS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}T_{WS}^{WS}(t_i) \cdot {}^{O_j^{1}}_{local}T_{BS}^{WS}(t_i) \\ &= {}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i) \cdot {}^{O_j}_{local}T_{BS}^{WS}(t_i) \end{aligned}$$

Um die Positionen von Objekten zu manipulieren, muß in beiden Fällen – *relative* und *absolute* Transformation – nur eine Transformation manipuliert werden. Dies wird genau dadurch erreicht, daß nur die lokale Transformation ${}^{O_j}_{local}T_{BS}^{WS}(t_i)$ zu einer Transformation ${}^{O_j}_{local}T_{BS}^{WS}(t_{i+1})$ modifiziert wird.

- **Relative Transformation**
  Es gilt:

$$ {}^{O_j}T_{BS}^{WS}(t_i) \; = \; {}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i) \cdot {}^{O_j}_{local}T_{BS}^{WS}(t_i) $$

  und

$$ {}^{O_j}T_{BS}^{WS}(t_{i+1}) \; = \; T_{WS(t_i)}^{WS(t_{i+1})} \cdot {}^{O_j}T_{BS}^{WS}(t_i) $$

  Gleichzeitig muß gelten:

$$ {}^{O_j}T_{BS}^{WS}(t_{i+1}) \; = \; {}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i) \cdot {}^{O_j}_{local}T_{BS}^{WS}(t_{i+1}) $$

  Daraus folgt unmittelbar:

$$ \begin{aligned} & T_{WS(t_i)}^{WS(t_{i+1})} \cdot {}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i) \cdot {}^{O_j}_{local}T_{BS}^{WS}(t_i) \\ = \; & {}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i) \cdot {}^{O_j}_{local}T_{BS}^{WS}(t_{i+1}) \end{aligned} $$

  Durch die Multiplikation von $({}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i))^{-1}$ von links an die Gleichung ergibt sich direkt:

$$ \begin{aligned} {}^{O_j}_{local}T_{BS}^{WS}(t_{i+1}) = \; & ({}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i))^{-1} \cdot T_{WS(t_i)}^{WS(t_{i+1})} \\ & \cdot {}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i) \cdot {}^{O_j}_{local}T_{BS}^{WS}(t_i) \end{aligned} $$

- **Absolute Transformation**
  Es gilt:

$$ {}^{O_j}T_{BS}^{WS}(t_i) \; = \; {}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i) \cdot {}^{O_j}_{local}T_{BS}^{WS}(t_i) $$

  und

$$ {}^{O_j}T_{BS}^{WS}(t_{i+1}) \; = \; T_{BS}^{WS}(t_{i+1}) $$

  Gleichzeitig muß wie im Falle der *relativen Transformation* gelten:

$$ {}^{O_j}T_{BS}^{WS}(t_{i+1}) \; = \; {}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i) \cdot {}^{O_j}_{local}T_{BS}^{WS}(t_{i+1}) $$

  Daraus folgt unmittelbar:

$$ T_{BS}^{WS}(t_{i+1}) \; = \; {}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i) \cdot {}^{O_j}_{local}T_{BS}^{WS}(t_{i+1}) $$

  Durch die Multiplikation von $({}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i))^{-1}$ von links an die Gleichung ergibt sich direkt:

$$ {}^{O_j}_{local}T_{BS}^{WS}(t_{i+1}) \; = \; ({}^{O_j^{(n)}}_{local}T_{BS}^{WS}(t_i) \cdot \ldots \cdot {}^{O_j^{(2)}}_{local}T_{BS}^{WS}(t_i))^{-1} \cdot T_{BS}^{WS}(t_{i+1}) $$

Die *hierarchische Szenenrepräsentation mit lokalen Transformationen* hat ihre Vorteile bei der Manipulation der Objekte. Bei der Manipulation eines Objektes O_j muß nur eine Transformation ${}^{O_j}_{local}T^{WS}_{BS}(t_i)$ zu einer Transformation ${}^{O_j}_{local}T^{WS}_{BS}(t_{i+1})$ modifiziert werden. Um jedoch die aktuelle Position eines Objektes zu erlangen, muß eine Folge von Transformationen aufmultipliziert werden.

Anhang C

VIRTUAL REALITY COMPETENCE CENTER der Daimler-Benz Forschung

Der Daimler-Benz-Konzern hat im Februar 1996 beschlossen, die Möglichkeiten von *Virtual Reality* für den Gesamtkonzern systematisch zu erschließen, nachdem zuvor bereits erste Versuche zur Anwendung von *Virtual Reality*-Techniken durchgeführt wurden. Die Notwendigkeit der Ausnutzung der Möglichkeiten, die *Virtual Reality* für den Daimler-Benz-Konzern bietet, hat der damalige Forschungsvorstand Prof. WEULE zusammengefaßt in der Aussage:

> Als weltweit agierender Hersteller technisch anspruchsvoller Geräte kann Daimler-Benz im Wettlauf um kürzere Entwicklungszeiten, um Fehlerreduktion in der Planungsphase, um geringere Produktionskosten und verbesserte Produktqualität auf diese Schlüsseltechnologie nicht verzichten.

Folgerichtig wurde mit dem Aufbau des VIRTUAL REALITY COMPETENCE CENTER im Forschungszentrum Ulm des Daimler-Benz-Konzerns begonnen und im September 1996 nach nur sechsmonatiger Aufbauphase der Betrieb aufgenommen.

Das VIRTUAL REALITY COMPETENCE CENTER ist die zentrale Anlaufstelle für alle Fragen zur VR-Technologie und deren Anwendung im Daimler-Benz-Konzern (siehe auch Abbildung D.60).

Es hat die Aufgabe

- innovative Problemlösungen zu erarbeiten und in die Produktbereiche zu transferieren,
- VR-Komponenten zu VR-Systemen zu integrieren,
- die Forschungs- und Produktbereiche als VR-Service-Center zu unterstützen und

- vermarktbare VR-Softwarewerkzeuge zu identifizieren und auszugliedern.

Um diese Aufgaben erfüllen zu können, verfügt das VIRTUAL REALITY COMPETENCE CENTER mit seinen derzeit 13 wissenschaftlichen Mitarbeitern und 3 Doktoranden über verschiedene Visualisierungseinrichtungen, z.B.

- dreikanalige 200°-Projektion,
- Hinterwandprojektion,
- vierseitige CAVE^{TM} (CAVE Automatic Virtual Environment),
- BOOM (Binocular Omni Orientation Monitor) und
- Head-Mounted-Displays.

Als Computerhardware stehen unterschiedliche Maschinen der Firma Silicon Graphics darunter eine SGI ONYX mit 3 Infinite Reality Graphikpipes sowie verstärkt leistungsfähige PCs zur Verfügung. Darüber hinaus verfügt das VIRTUAL REALITY COMPETENCE CENTER über eine Reihe gängiger Interaktions- und Trackingsysteme wie

- Datenhandschuh,
- Spacemouse,
- Flying mouse,
- optisches Trackingsystem,
- Magnetfeldtrackingsystem und
- mechanisches Trackingsystem.

Die Einsatzfelder von *Virtual Reality* im VIRTUAL REALITY COMPETENCE CENTER erstrecken sich heute über eine breite Palette von Anwendungen wie

- Ein-/Ausbau- und Wartungsuntersuchungen,
- Ergonomieuntersuchungen und Bedienplatzdesign,
- Hochqualitätsvisualisierung im Designbereich,
- virtuelle Verkaufsunterstützung,
- Fahr- und Flugsimulation,
- Telemanipulation von Weltraum-Robotern,

- Visualisierung von Simulationsdaten (Strömungen, Crash),
- Planung und Optimierung von Produktionsanlagen und
- Trainingssimulatoren.

Die Basis für die Integration der verschiedenen Visualisierungs-, Interaktions- und Trackingsysteme bildet die von Daimler-Benz eigenentwickelte, derzeit auf der Visualisierungsbibliothek OpenInventor™ der Firma Silicon Graphics basierende *Virtual Reality*-Softwareplattform **DBView** (siehe Abbildung D.61), die objektorientiert in C++ realisiert ist.

Im Rahmen dieser Arbeit wurde **DBView** um eine Simulationsumgebung mit echtzeitfähiger Kollisionserkennung erweitert. Diese Simulationsumgebung bildet die Basis für realitätsnahe Objektinteraktionen innerhalb von **DBView**. Aufbauend auf diesem Basissimulationssystem werden derzeit weitergehende echtzeitfähige Simulationen aus dem Bereich der Objektmanipulation und der Simulation dynamischen Objektverhaltens realisiert.

Literaturverzeichnis

[AS96] P.K. Agarwal, M. Sharir
Efficient Algorithms for Geometric Optimization
New York University, Dezember 1996

[Ba90] D. Baraff
Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation
Proceedings ACM SIGGRAPH, Vol. 24, No 4., S. 19–28, 1990

[Ba92] D. Baraff
Dynamic Simulation of Non-Penetrating Rigid Bodies
PhD thesis, Department of Computer Science, Cornell University, 1992

[BWK95] D. Baraff, A. Witkin, M. Kass
An Introduction to Physically Based Modelling
Course Notes ACM SIGGRAPH, 1995

[BDH93] B. Barber, D. Dobkin, H. Huhdanpaa
The quickhull algorithm for convex hull
Technical Report GCG53, The Geometry Center, 1993

[BC+96] G. Barequet, B. Chazelle, L. J. Guibas, J. S. B. Mitchell, A. Tal
BOXTREE: A Hierarchical Representation for Surfaces in 3D
Proceedings of Eurographics, Vol. 15, No. 3, S. 26–30, 1996

[BG93] I. M. Bomze, W. Grossmann
Optimierung – Theorie und Algorithmen
BI Wissenschaftsverlag, 1993

[BS+95] I. N. Bronstein, K. A. Semendjajew, G. Musiol, H. Mühlig
Taschenbuch der Mathematik
Verlag Harri Deutsch, 1995

[BV91] W. Bouma, G. Vaněček Jr.
Collision Detection and Analysis in Physically Based Simulation
Proceedings Second Eurographics Workshop on Animation and Simulation, S. 191–203, 1991

[Bu98] M. Buck
Simulation interaktiv bewegter Objekte mit Hinderniskontakten
erscheint als Dissertation, Universität des Saarlandes, Fachbereich Informatik, 1998

[BS98] M. Buck, E. Schömer
Interactive Rigid Body Manipulation with Obstacle Contacts
Proceedings Sixth International Conference in Central Europe on Computer Graphics and Visualization, S. 49–56, 1998

[BB94] H.-J. Bullinger, W. Bauer
Strategische Dimension der Virtual Reality
IPA/IAO-Forum Virtual Reality, Anwendungen & Trends, Springer Verlag, 1994

[BC94] G. Burdea, P. Coiffet
Virtual Reality Technology
John Wiley & Sons, 1994

[Ca97] S. Cameron
Enhanced GJK: Computing minimum penetration distances between convex polyhedra
Proceedings IEEE International Conference on Robotics and Automation, 1997

[Ca90] S. Cameron
Collision Detection by Four-dimensional Intersection Testing
Proceedings IEEE International Conference on Robotics and Automation, S. 291–302, 1990

[Ca86] J. F. Canny
Collision Detection for Moving Polyhedra
IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, S. 200–209, 1986

[CES96] T. Chadzelek, J. Eckstein, E. Schömer
Heuristic Motion Planning with Many Degrees of Freedom
Proceedings 8th Canadian Conference on Computational Geometry, S. 167–172, 1996

[CTL96] K. Chung Tat Leung
An Efficient Collision Detection Algorithm for Polytopes in Virtual Environments
M. Phil thesis, Department of Computer Science, University of Hong Kong, 1996

[CL+95] J. Cohen, M. Lin, D. Manocha, M. Ponamgi
I-Collide: An Interactive and Exact Collision Detection System for Large-scale

Environments
Proceedings ACM Interactive 3D Graphics Conference, S. 189–196, 1995

[CLR90] T. H. Cormen, C. E. Leiserson, R. L. Rivest
Introduction to Algorithms
MIT Press, 1990

[Cr89] J. J. Craig
Introduction to Robotics: Mechanics and Control
Addison-Wesley, 1989

[CR95] A. Crosnier, J. Rossignac
T-BOX: The Intersection of Three Mini-Max Boxes
Internal Report, IBM T. J. Watson Research Center, Yorktown Heights, 1995

[CB78] M. Cyrus, J. Beck
Generalized Two- and Three-Dimensional Clipping
Computers and Graphics, Vol. 3, No. 1, S. 23–28, 1978

[DK90] D. P. Dobkin, D. G. Kirkpatrick
Determining the Separation of Preprocessed Polyhedra - A Unified Approach
Lecture Notes in Computer Science 443, S. 400–413, 1990

[DH+90] D. P. Dobkin, J. Hershberger, D. G. Kirkpatrick, S. Suri
Implicitly Searching Convolutions and Computing Depth of Collision
Lecture Notes in Computer Science 450, S. 165–180, 1990

[DK85] D. P. Dobkin, D. G. Kirkpatrick
A Linear Algorithm for Determining the Separation of Convex Polyhedra
Journal of Algorithms, Vol. 6, S. 381–392, 1985

[DS90] G. Dueck, T. Scheuer
Threshold Accepting: A General Purpose Optimization Algorithm Superior to Simulated Annealing
Journal of Computational Physics, Vol. 90, S. 161–175, 1990

[DM94] N. I. Durlach, A. S. Mavor, Editors
Virtual Reality: Scientific and Technological Challenges
National Academy Press, 1994

[Ec94] J. Eckstein
Heuristische Bewegungsplanungsstrategien im $\mathbb{R}^3$
Diplomarbeit, Universität des Saarlandes, Dezember 1994

[ECS96] J. Eckstein, T. Chadzelek, E. Schömer
Heuristic Motion Planning with Movable Obstacles

Proceedings 8th Canadian Conference on Computational Geometry, S. 131–136, 1996

[FG88] T. Feder, D.H. Green
Optimal algorithms for approximate clustering
Proc. 20th Ann. ACM Sympos. Theory Comput., S. 434–444, 1988

[FZ95] W. Felger, G. Zachmann
The Boxtree: Enabling Real-Time and Exact Collision Detection
Proceedings of 1st Workshop on Simulation and Interaction in Virtual Environments, July 13–15, 1995, Iowa City

[FPT81] R.J. Fowler, M.S. Paterson, S.L. Tanimoto
Optimal packing and covering in the plane are NP-complete
Inform. Process. Lett., No. 12, S. 133–137, 1981

[FKN80] H. Fuchs, Z. Kedem, B. Naylor
On Visible Surface Generation by A Priori Tree Structures
Proceedings ACM SIGGRAPH, Vol. 14, No. 3, S. 124–133, 1980

[Fu97] A. Fuhrmann
Testing roundness of a polytope and related problems
Proceedings of the 9th Canadian Conference on Computational Geometry, S. 169–174, 1997

[GSF94] A. Garcia-Alonso, N. Serrano, J. Flaquer
Solving the Collision Detection Problem
IEEE Computer Graphics and Applications, Vol. 13, No. 3, S. 36–43, 1994

[GJK88] E.G. Gilbert, D.W. Johnson, S.S. Keerthi
A Fast Procedure for Computing the Distance Between Complex Objects in Three-dimensional Space
IEEE Journal of Robotics and Automation, Vol. 4, No. 2, S. 193–203, 1988

[Gla90] A. Glassner ed.
Graphics Gems
Academic Press, 1990

[Gl89] F. Glover
Tabu Search - Part I
ORSA Journal of Computing, Vol. 1, No. 3, S. 190–206, 1989

[Gl90] F. Glover
Tabu Search - Part II
ORSA Journal of Computing, Vol. 2, No. 1, S. 4–32, 1990

[GY+91] J. A. Goldak, X. Yu, A. Knight, L. Dong
Constructing discrete medial axis of 3-D objects
International Journal on Computational Geometry and Applications, Vol. 1, No. 3, S. 327–339, 1991

[Go89] D. E. Goldberg
Genetic Algorithms in Search, Optimization and Machine Learning
Addison-Wesley, 1989

[Go85] T. Gonzalez
Clustering to minimize the maximimum intercluster distance
Theoret. Comput. Science, No. 38, S. 293–306, 1985

[Go96] S. Gottschalk
Separating axis theorem
Technical Report, TR96-024, Department of Computer Science, University of North Carolina, Chapel Hill, 1996

[GLM96] S. Gottschalk, M. C. Lin, D. Manocha
OBBTree: A Hierarchical Structure for Rapid Interference Detection
Proceedings ACM SIGGRAPH, S. 171–180, 1996

[Ha88] J. Hahn
Realistic Animation of Rigid Bodies
Proceedings ACM SIGGRAPH, Vol. 22, No. 4, S. 299–308, 1988

[He94] P.S. Heckbert ed.
Graphics Gems IV
Academic Press, 1994

[HKM95] M. Held, J. T. Klosowski, J. S. B. Mitchell
Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs
Proceedings 7th Canadian Conference on Computational Geometry, S. 205–210, 1995

[He97] A. Hennig
Die andere Wirklichkeit: Virtual Reality - Konzepte, Standards, Lösungen
Addison-Wesley, 1997

[HBZ90] B. V. Herzen, A. H. Barr, H. R. Zatz
Geometric Collisions for Time-Dependent Parametric Surfaces
Proceedings ACM SIGGRAPH, Vol. 24, No. 4, S. 39–48, 1990

[HS85] D. S. Hochbaum, D. Shmoys
A best possible heuristic for the k-center problem
Math. Oper. Res., No. 10, S. 180–184, 1985

[HS86] D. S. Hochbaum, D. Shmoys
A unified approach to approximation algorithms for bottleneck problems
Journal ACM, No. 33, S. 533–550, 1986

[Ho89] C. M. Hoffmann
Geometric and Solid Modeling - An Introduction
Morgan Kaufmann, San Mateo, California, 1989

[Hu95] P. M. Hubbard
Collision Detection for Interactive Graphics Applications
PhD thesis, Brown University, Department of Computer Science, 1995

[Hu95a] P. M. Hubbard
Collision Detection for Interactive Graphics Applications
IEEE Transactions on Visualization and Computer Graphics, 1(3), Sept. 1995, S. 218–230

[HL+97] T. C. Hudson, M. C. Lin, J. Cohen, S. Gottschalk, D. Manocha
V-Collide: Accelerated Collision Detection for VRML
Proceedings of VRML 1997

[HD+96] M. Hughes C. DiMattia, M. C. Lin, D. Manocha
Efficient and Accurate Interference Detection for Polyhedral Deformation
Proceedings Computer Animation Conference, 1996

[Ka97a] A. E. Kaufman
Principles of Volume Visualization
Course Notes ACM SIGGRAPH, 1997

[Ka97b] A. E. Kaufman
Advances in Volume Visualization
Course Notes ACM SIGGRAPH, 1997

[KK86] T. L. Kay, J. T. Kajiya
Ray Tracing Complex Scenes
Proceedings ACM SIGGRAPH, Vol. 20, S. 269–278, 1986

[Ke97] L. Kettner
Designing a Data Structure for Polyhedral Surfaces
Technischer Bericht No. 278, Fachbereich Informatik, ETH Zürich, 1997

[KS+98] Y. Kitamura, A. Smith, H. Takemura, F. Kishino
A Real-Time Algorithm for Accurate Collision Detection for Deformable Polyhedral Objects
Erscheint in PRESENCE, Vol. 7, No. 1, MIT Press, 1998

[KH+] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, K. Zikan
Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs
Erscheint in IEEE Transactions on Visualization and Computer Graphics

[LA87] P. J. M. van Laarhoven, E. H. L. Aarts
Simulated Annealing: Theory and Applications
D. Reidel Publishing Company, Dordrecht, Holland, 1987

[La91] J. C. Latombe
Robot Motion Planning
Kluwer Academic Publishers, 1991

[LC91] M. C. Lin, J. F. Canny
A Fast Algorithm for Incremental Distance Calculation
Proc. IEEE International Conference on Robotics and Automation, S. 1008–1014, 1991

[Li93] M. C. Lin
Efficient Collision Detection for Animation and Robotics
PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993

[LM] M. C. Lin, D. Manocha
Efficient Contact Determination Between Geometric Models
Erscheint in International Journal of Computational Geometry and Applications

[LM+] M. C. Lin, D. Manocha, J. Cohen, S. Gottschalk
Collision Detection: Algorithms and Applications
in Proc. of Algorithms for Robotics Motion and Manipulation, S. 129–142
eds.: J.-P. Laumond, M. Overmars, A. K. Peters

[Ma88] M. Mäntylä
An Introduction To Solid Modeling
Comuter Science Press, Rockville, Maryland, 1988

[MS81] A. Marchetti - Spaccamelz
The p-Center Problem in the plane is NP-complete
Proc. of the 19th Allerton Commun. Control Comput., 1981

[MS84] N. Megiddo, K. J. Supowit
On the complexity of some common geometric location problems
SIAM Journal Comput., No. 13, S. 182–196, 1984

[MN] K. Mehlhorn, S. Näher
LEDA, a Platform for Combinatorial and Geometric Computing
Buch in Vorbereitung

[MN95] K. Mehlhorn, S. Näher
LEDA, a Platform for Combinatorial and Geometric Computing
Communications of the ACM, Vol. 38, No. 1, S. 96–102, 1995

[MR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller
Journal of Chemical Physics, Vol. 21, S. 1087–1092, 1953

[Mi96] B. Mirtich
Impulsed-based Dynamic Simulation of Rigid Body Systems
PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1996

[Mi97] B. Mirtich
V-Clip: Fast and Robust Polyhedral Collision Detection
TR-97-05, MERL (A Mitsubishi Electric Research Laboratory), 1997

[Mi98] B. Mirtich
Rigid Body Contact: Collision Detection to Force Computation
TR-98-01, MERL (A Mitsubishi Electric Research Laboratory), 1998

[MW88] M. Moore, J. Wilhelms
Collision Detection and Response for Computer Animation
Proceedings ACM SIGGRAPH, Vol. 22, No. 4, S. 289–298, 1988

[NAT90] B. Naylor, J. A. Amatodes, W. Thibault
Merging BSP Trees Yields Polyhedral Set Operations
ACM Computer Graphics, Vol. 24, No. 4, S. 115–124, 1990

[NM65] J. A. Nelder, R. Mead
Computer Journal, Vol. 7, S. 305–309, 1965

[NFA89] H. Noborio, S. Fukuda, S. Arimoto
Fast Interference Check Method Using Octree Representation
Advanced Robotics, No. 3, Vol. 3, S. 193–212, 1989

[Ro85] J. O'Rourke
Finding Minimal Enclosing Boxes
Int. Journal of Computer and Information Sciences, Vol. 14, No. 3, S. 183–199, 1985

[Ov92] M. H. Overmars
Point location in fat subdivisions
Inform. Proc. Lett., No. 44, S. 261–265, 1992

[PG95] I. J. Palmer, R. L. Grimsdale
Collision Detection for Animation using Sphere-Trees
Proceedings of Eurographics, Vol. 14, No. 2, S. 105–116, 1995

[PML95] K. Ponamgi, D. Manocha, M. Lin
Incremental algorithms for collision detection between solid models
Proceedings ACM SIGGRAPH Symposium on Solid Modelling, S. 293–304, 1995

[PS85] F. P. Preparata, M. I. Shamos
Computational Geometry: An Introduction
Springer Verlag, New York, 1985

[PT+92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery
Numerical Recipes in C
Cambridge University Press, Cambridge, England, 1992

[Jo97] S. Jost
Impulsbasierte Dynamik starrer Körper
Diplomarbeit, Fachbereich Informatik, Universität des Saarlandes, 1997

[Qu94] S. Quinlan
Efficient distance computation between non-convex objects
Proceedings of International Conference on Robotics and Automation,
S. 3324–3329, 1994

[Sa90] H. Samet
The Design and Analysis of Spatial Data Structures
Addison Wesley, 1990

[SW88a] H. Samet, R. E. Webber
Hierarchical Data Structures and Algorithms for Computer Graphics, Part I: Fundamentals
IEEE Computer Graphics and Applications, Vol. 8, No. 3, S. 48–68, 1988

[SW88b] H. Samet, R. E. Webber
Hierarchical Data Structures and Algorithms for Computer Graphics, Part II: Applications
IEEE Computer Graphics and Applications, Vol. 8, No. 4, S. 59–75, 1988

[Sa95] J. Sauer
Allgemeine Kollisionserkennung und Formrekonstruktion basierend auf Zellkomplexen
Diplomarbeit, Fachbereich Informatik, Universität des Saarlandes, 1995

[SS98] J. Sauer, E. Schömer
Dynamiksimulation starrer Körper für Virtual Reality Anwendungen erscheint in Proceedings of ASIM 98

[Sa99] J. Sauer
Hybride Echtzeitsimulation der mechanischen Dynamik starrer Körper für Virtual Reality Anwendungen
erscheint als Dissertation, Fachbereich Informatik, Universität des Saarlandes, 1999

[Sch94] E. Schömer
Interaktive Montageplanung mit Kollisionserkennung
Dissertation, Fachbereich Informatik, Universität des Saarlandes, 1994

[SSW95] E. Schömer, J. Sellen, M. Welsch
Exact Geometric Collision Detection
Proceedings 7th Canadian Conference on Computational Geometry, 1995

[ST96] E. Schömer, A. Thiel
Subquadratic Algorithms for the General Collision Detection Problem
12th European Workshop on Computational Geometry, Münster, 1996

[ST94] E. Schömer, A. Thiel
Efficient Collision Detection for Moving Polyhedra
Proceedings of the 11th ACM Symposium on Computational Geometry, 1995, S. 51–60

[SH92] C. Shaffer, G. Herb
A Real-Time Robot Arm Collision Avoidance System
IEEE Transactions on Robotics and Automation, Volume 8, No. 2, S. 149–160, 1992

[Sh85] K. Shoemake
Animating Rotation with Quaternion Curves
Proceedings ACM SIGGRAPH, 22–26 Juli, San Francisco, S. 245–254, 1985

[SK+95] A. Smith, Y. Kitamura, H. Takemura, F. Kishino
A Simple and Efficient Method for Accurate Collision Detection Among Deformable Polyhedral Objects in Arbitrary Motion
Proceedings IEEE Virtual Annual International Symposium, S. 136–145, 1995

[SW+93] J. M. Snyder, A. R. Woodbury, K. Fleischer, B. Currin, A. H. Barr
Interval Methods for Multi-Point Collisions between Time-Dependent Curved Surfaces
Proceedings ACM SIGGRAPH, S. 321–334, 1993

[St95] W. Steger
Neue Verfahren der Prototypherstellung und ihre Integration in der Unternehmensstrategie
Spektrum der Wissenschaft, S. 90, April 1995

[St94] A. J. STEWART
Local Robustness and its Application to Polyhedral Intersection
International Journal of Computational Geometry and Applications, Vol. 4, No. 1, S. 87–118, 1994

[St93] J. STOER
Numerische Mathematik 1
Springer Verlag, 6. korrigierte Neuauflage, 1993

[TN87] W. THIBAULT, B. NAYLOR
Set Operations on Polyhedra Using Binary Space Partitioning Trees
Proceedings ACM SIGGRAPH, Vol. 21, No. 4, S. 153–162, 1987

[Tu90] G. TURK
Interactive Collision Detection for Molecular Graphics
Technical Report TR90-014, Department of Computer Science, University of North Carolina at Chapel Hill, 1990

[Va91] G. VANĚČEK JR.
Brep-Index: A Multidimensional Space Partitioning Tree
Int. Journal of Computational Geometry and Applications, Vol. 1, No. 3, S. 242–261, 1991

[We91] E. WELZL
Smallest enclosing disks (balls and ellipsoids)
Lecture Notes in Computer Science, S. 359–370, 1991

[We94] J. WERNECKE
The Inventor Mentor
Addison-Wesley, 1994

[We94a] J. WERNECKE
The Inventor Toolmaker
Addison-Wesley, 1994

[Wh] D. A. WHITE
Visual Computing Laboratory, University of California, San Diego
http://vision.ucsd.edu/w̃hite/ball.html

[Za98] G. ZACHMANN
Rapid Collision Detection By Dynamically Aligned DOP-Trees
Proceedings IEEE Virtual Reality Annual International Symposium, 1998

[Za97] G. ZACHMANN
Real-Time and Exact Collision Detection for Interactive Virtual Prototyping
Proceedings ASME Design Engineering Technical Conferences, 1997

[ZO+93] J. Zyda, W. Osborne, J. Monahan, D. Pratt
NPSNET: Real-Time Vehicle Collisions, Explosions and Terrain Modifications
Journal of Visualization and Computer Animation, Vol. 4, No. 1, S. 13–24, 1993

Anhang D

Abbildungen

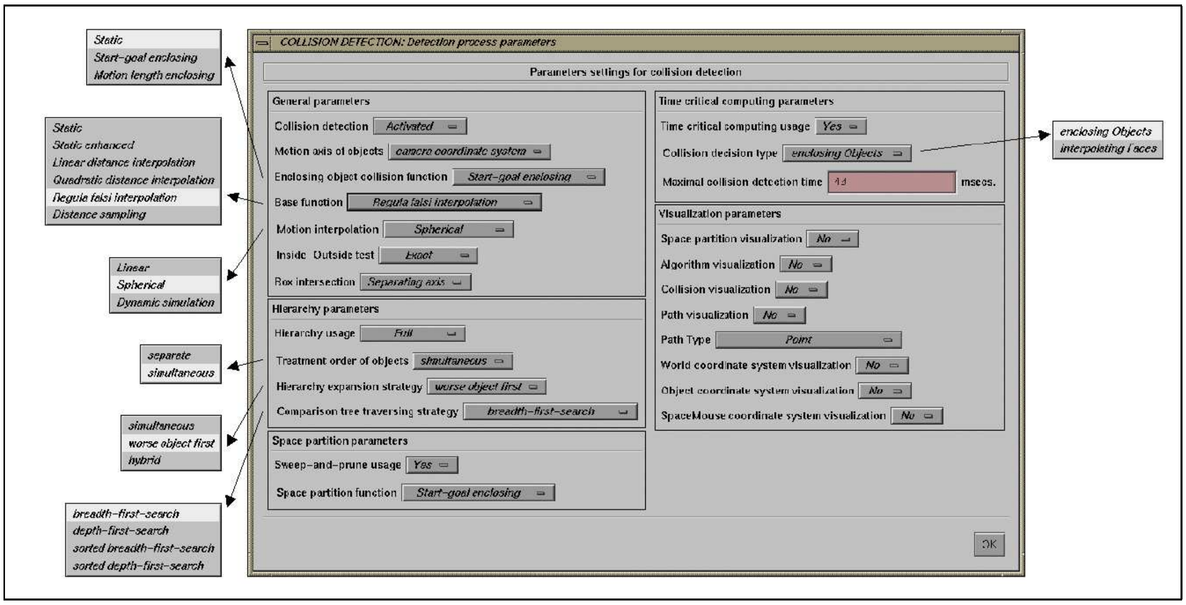

Abbildung D.1: Benutzeroberfläche zur Parametrisierung des Ablaufes der Kollisionserkennung

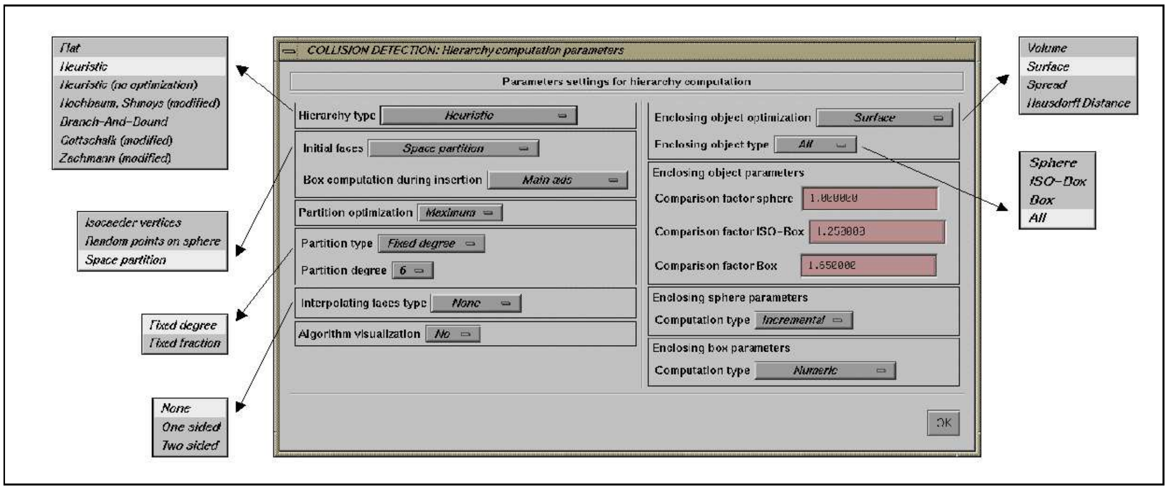

Abbildung D.2: Benutzeroberfläche zur Parametrisierung der Vorberechnung der Hüllkörperhierarchien

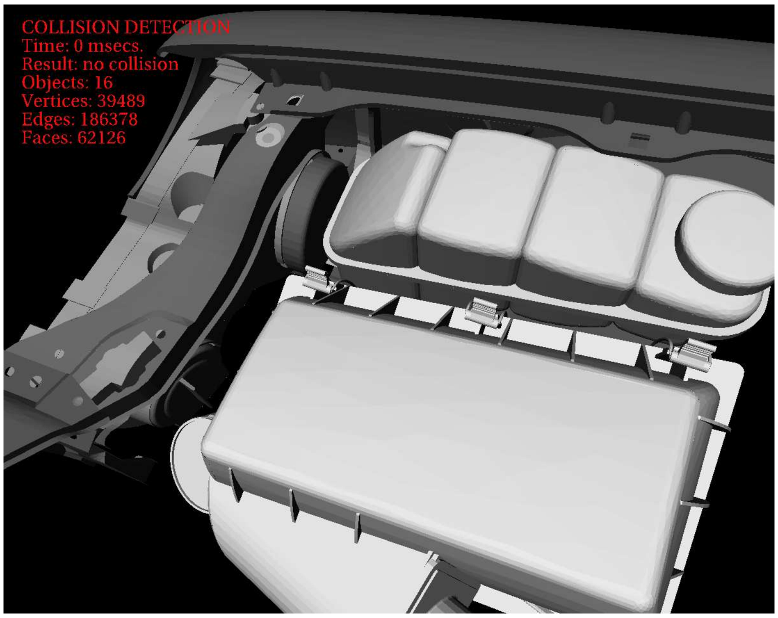

Abbildung D.3: Ausbauuntersuchung der Glühbirnen aus einen Motorraum

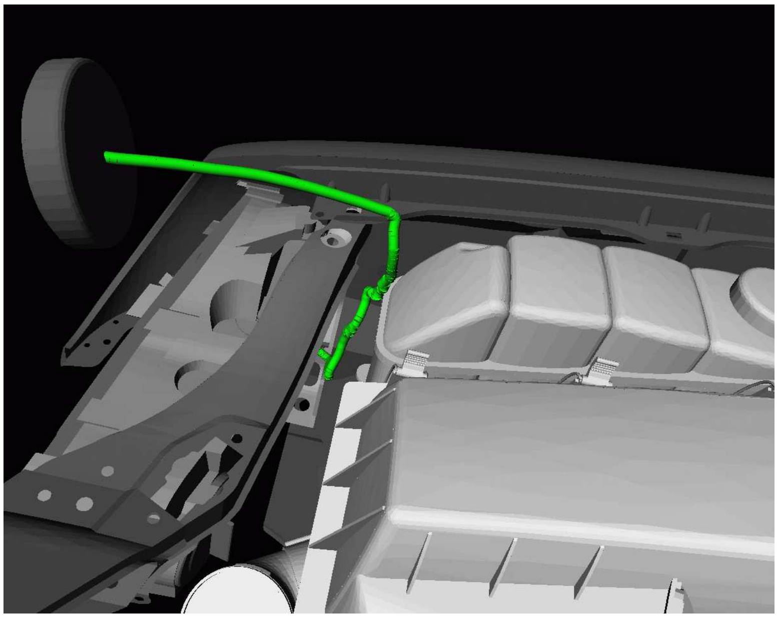

Abbildung D.4: Schritt 1: Ausbau der Abdeckung des Fahrlichts

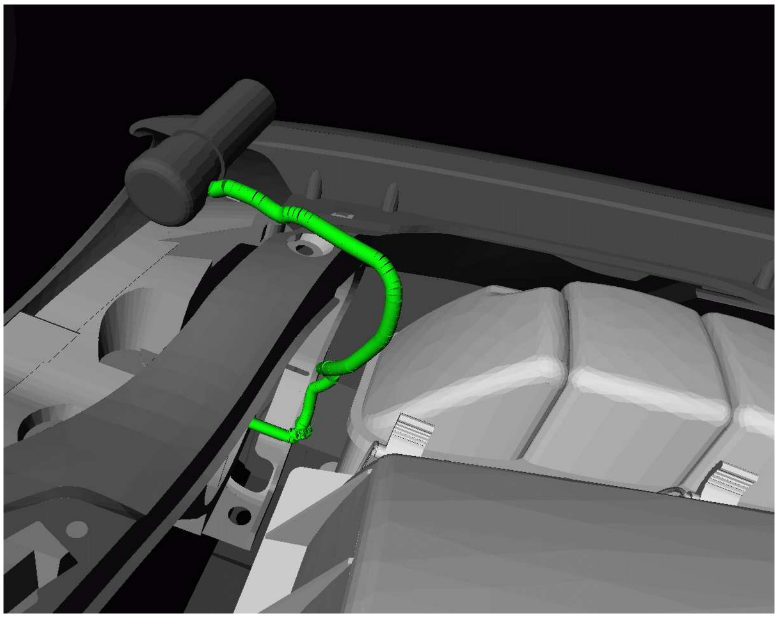

Abbildung D.5: Schritt 2: Ausbau der Glühbirne des Fahrlichts

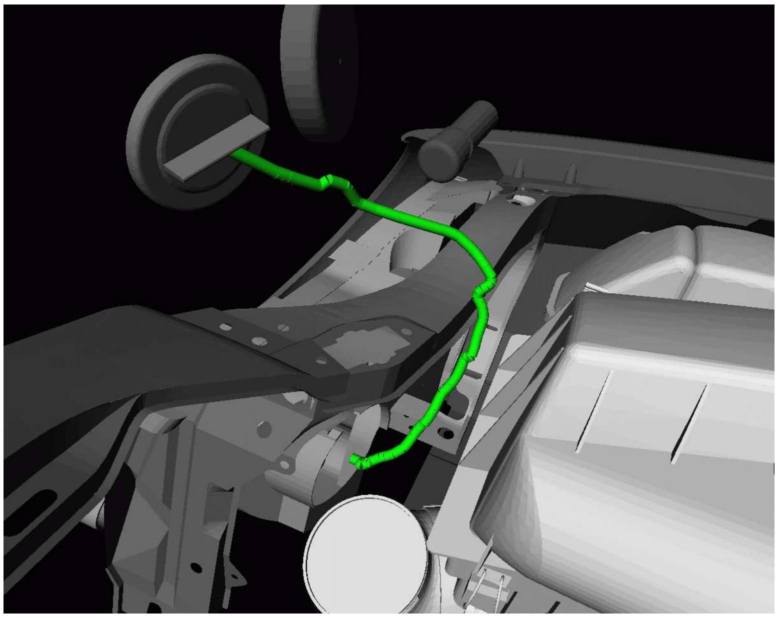

Abbildung D.6: Schritt 3: Ausbau der Abdeckung des Zusatzscheinwerfers

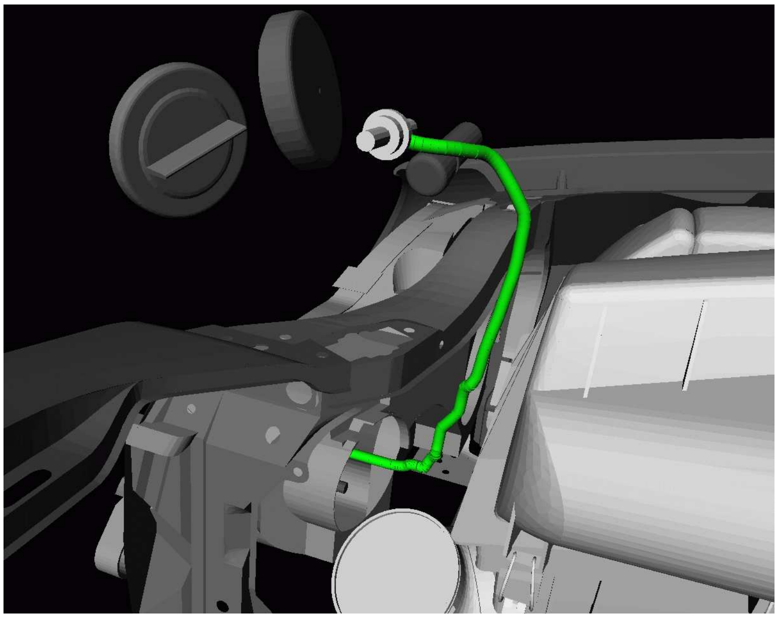

Abbildung D.7: Schritt 4: Ausbau der Glühbirne des Zusatzscheinwerfers

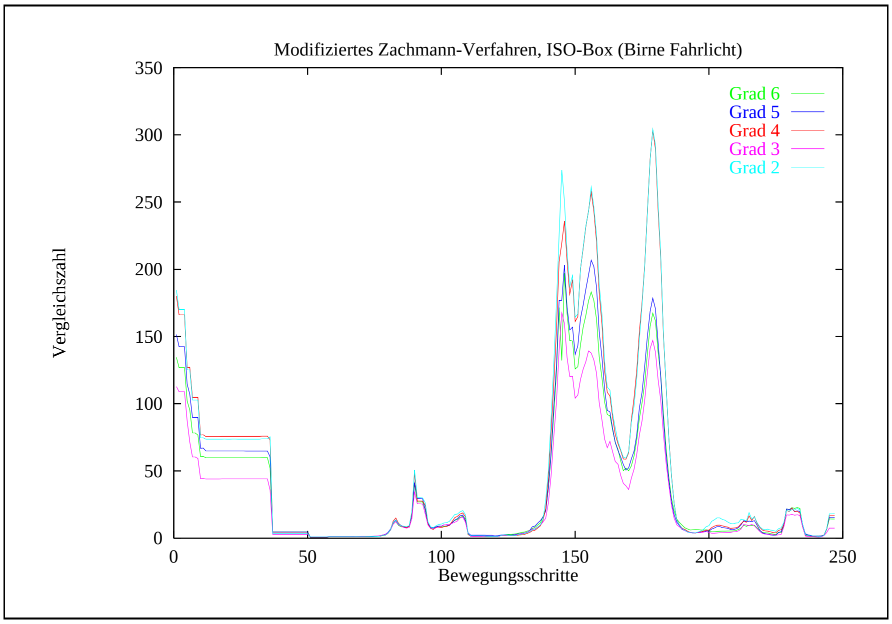

Abbildung D.8: Vergleich der festen Aufteilungsgrade (modifiziertes Zachmann-Verfahren)

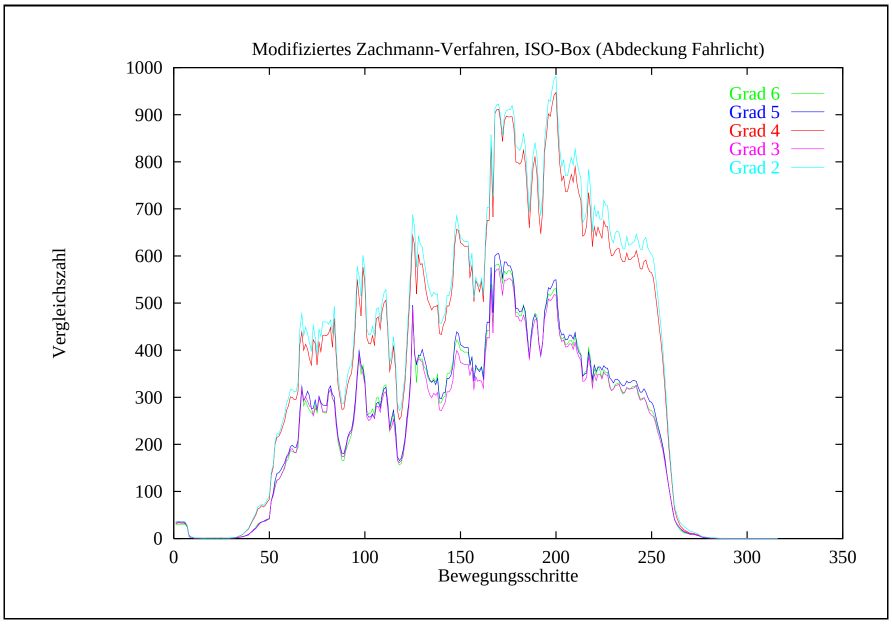

Abbildung D.9: Vergleich der festen Aufteilungsgrade (modifiziertes Zachmann-Verfahren)

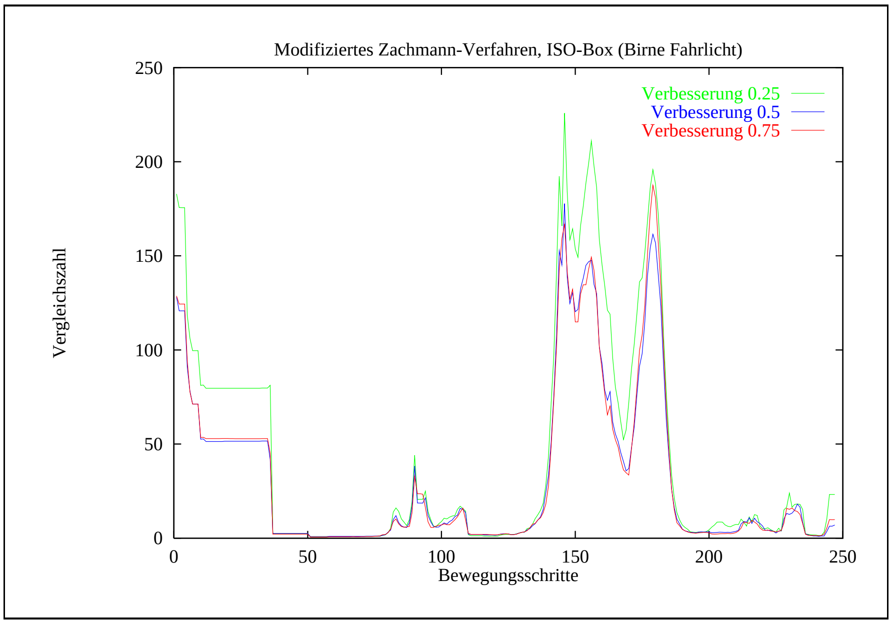

Abbildung D.10: Vergleich der Verbesserungsraten (modifiziertes Zachmann-Verfahren)

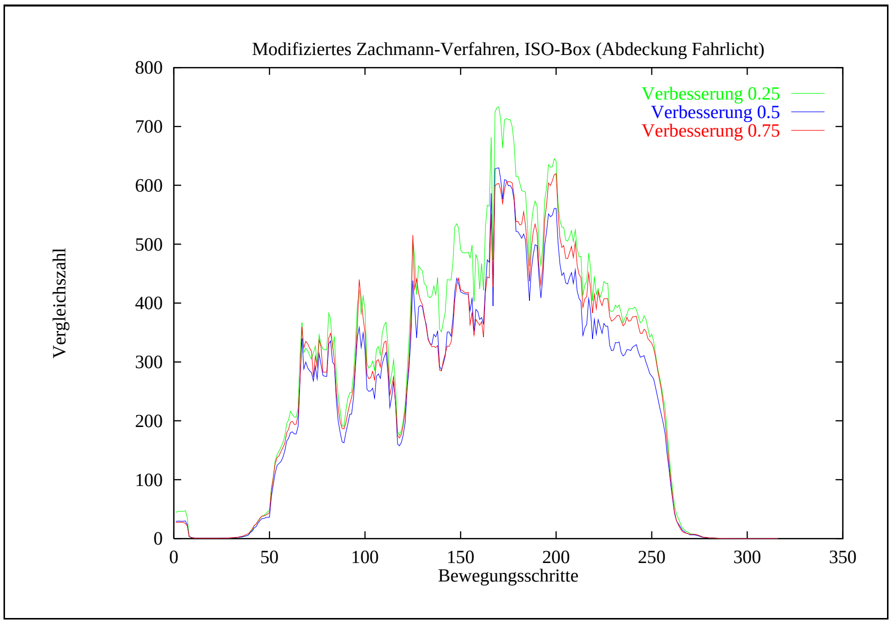

Abbildung D.11: Vergleich der Verbesserungsraten (modifiziertes Zachmann-Verfahren)

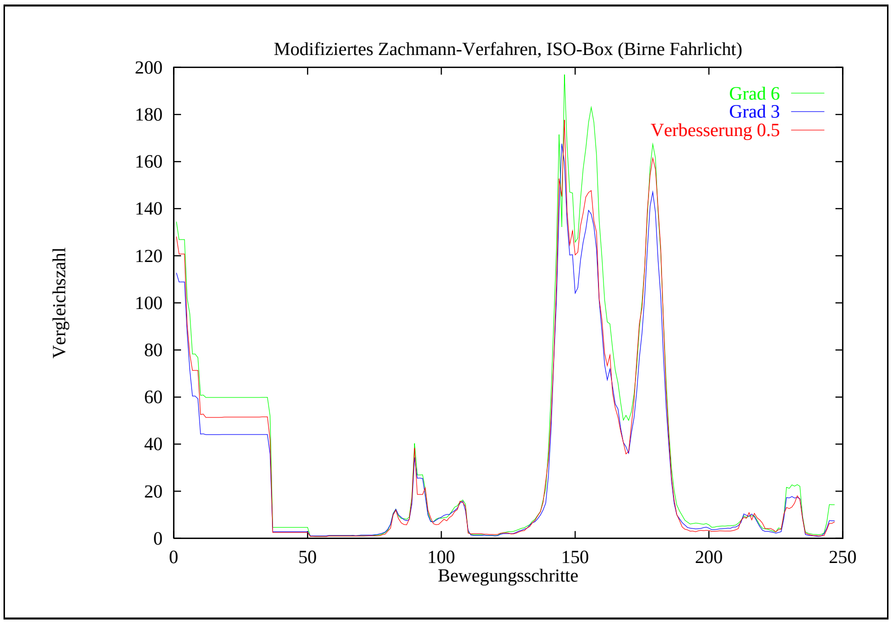

Abbildung D.12: Vergleich der besten Aufteilungsgrade und der besten Verbesserungsrate (modifiziertes Zachmann-Verfahren)

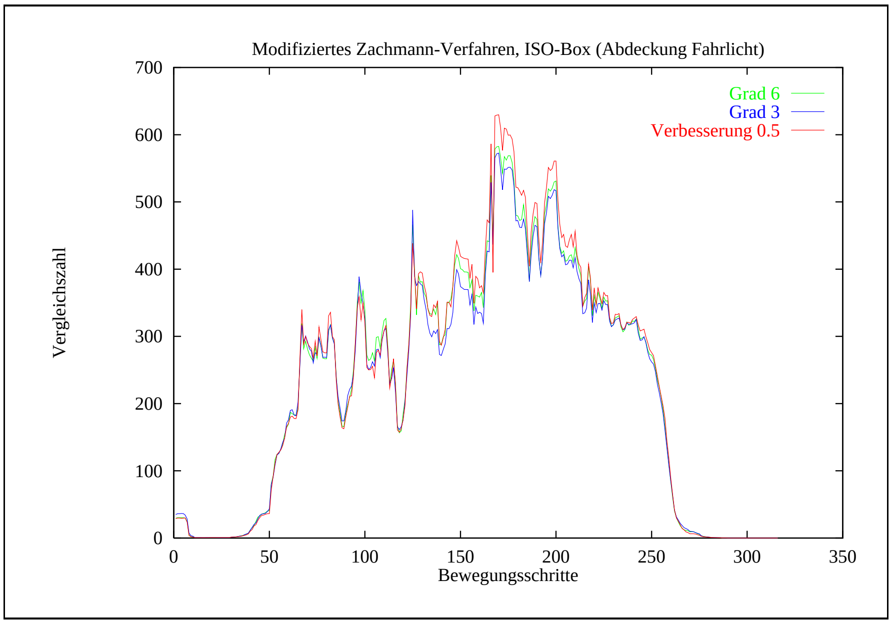

Abbildung D.13: Vergleich der besten Aufteilungsgrade und der besten Verbesserungsrate (modifiziertes Zachmann-Verfahren)

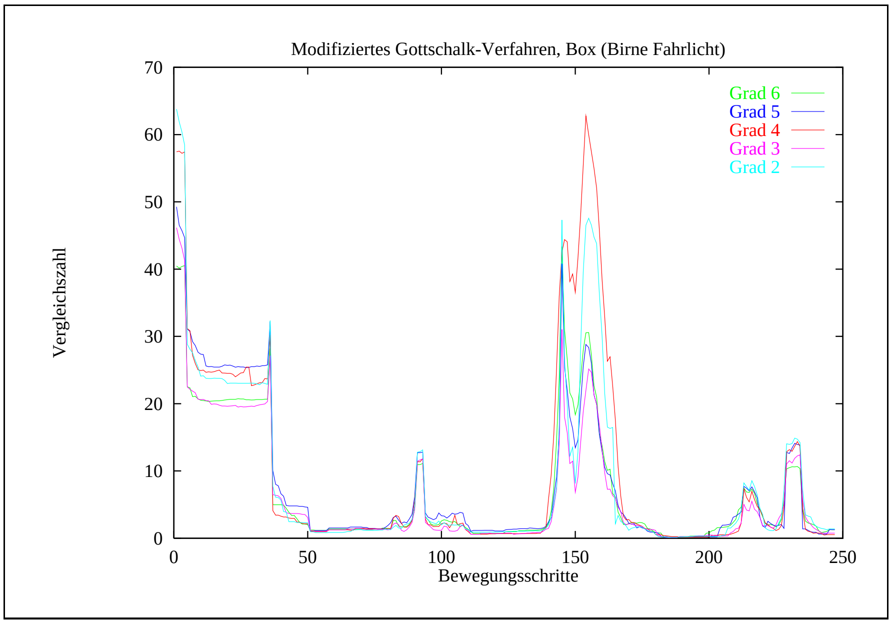

Abbildung D.14: Vergleich der festen Aufteilungsgrade (modifiziertes Gottschalk-Verfahren)

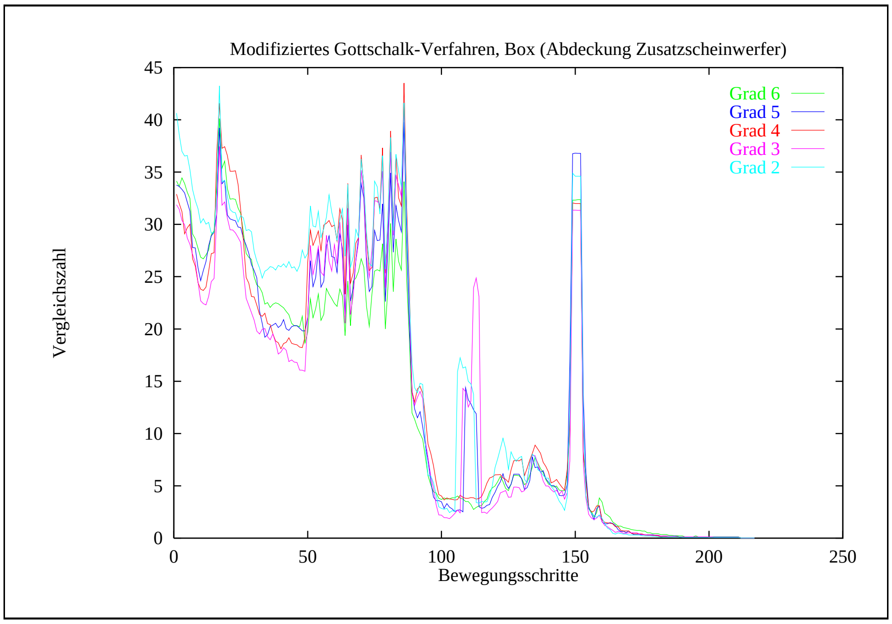

Abbildung D.15: Vergleich der festen Aufteilungsgrade (modifiziertes Gottschalk-Verfahren)

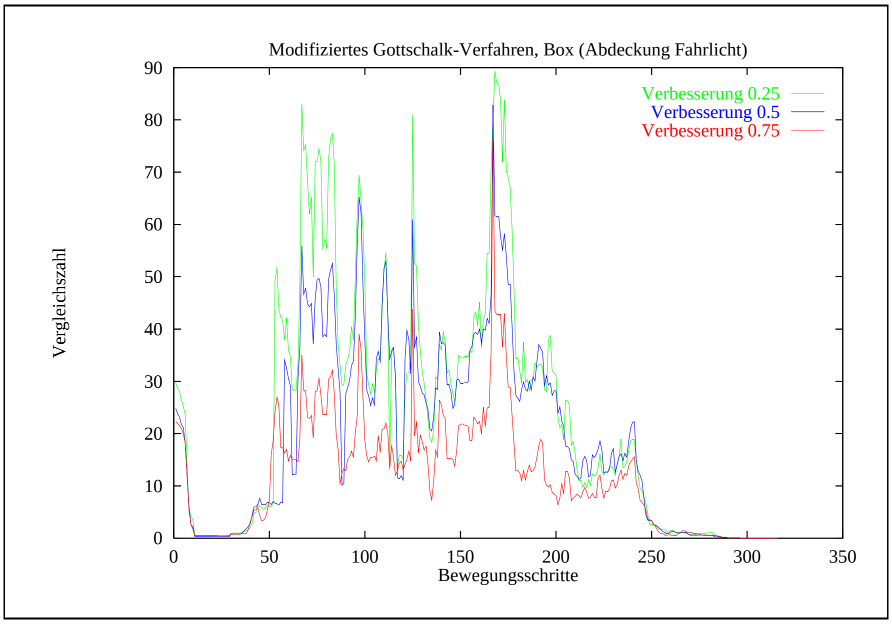

Abbildung D.16: Vergleich der Verbesserungsraten (modifiziertes Gottschalk-Verfahren)

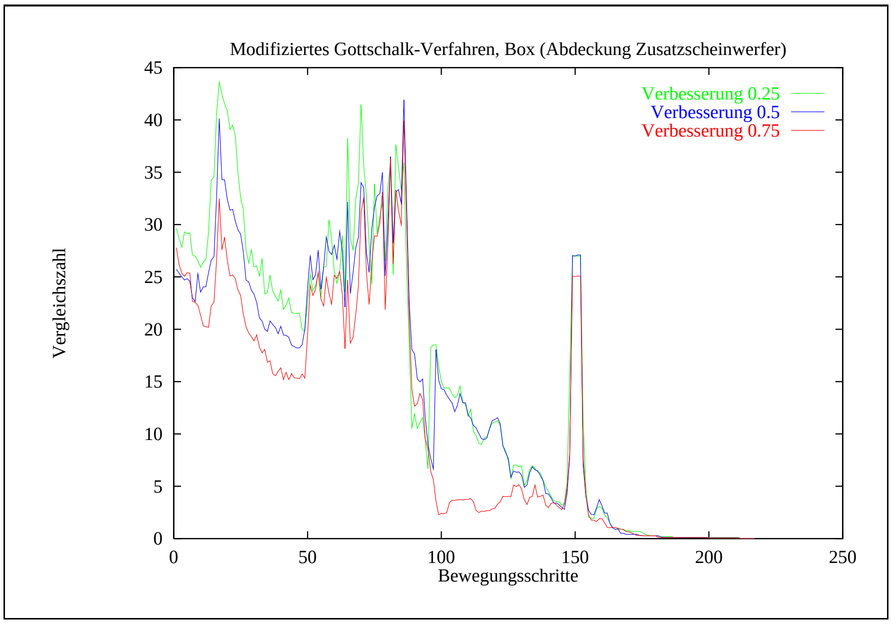

Abbildung D.17: Vergleich der Verbesserungsraten (modifiziertes Gottschalk-Verfahren)

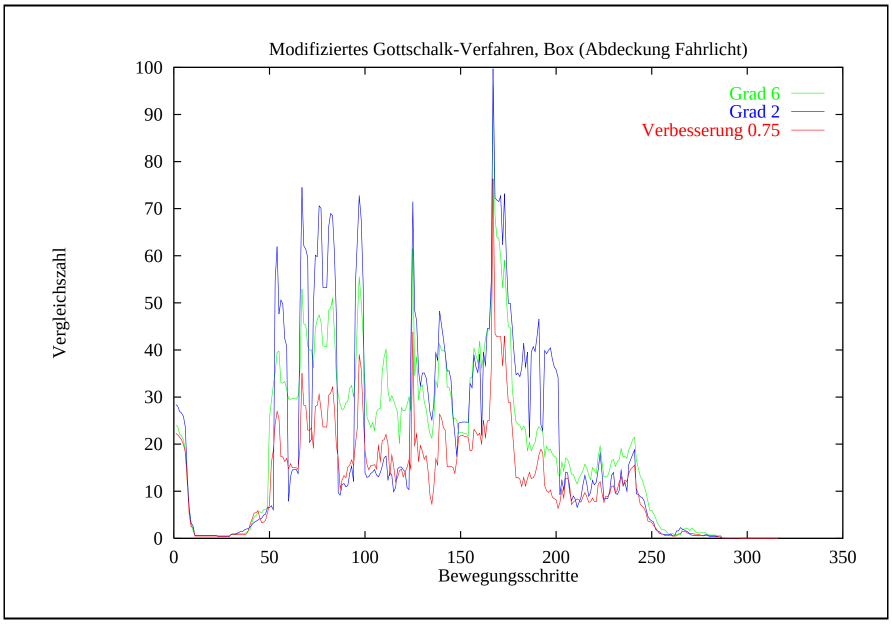

Abbildung D.18: Vergleich der besten Aufteilungsgrade und der besten Verbesserungsrate (modifiziertes Gottschalk-Verfahren)

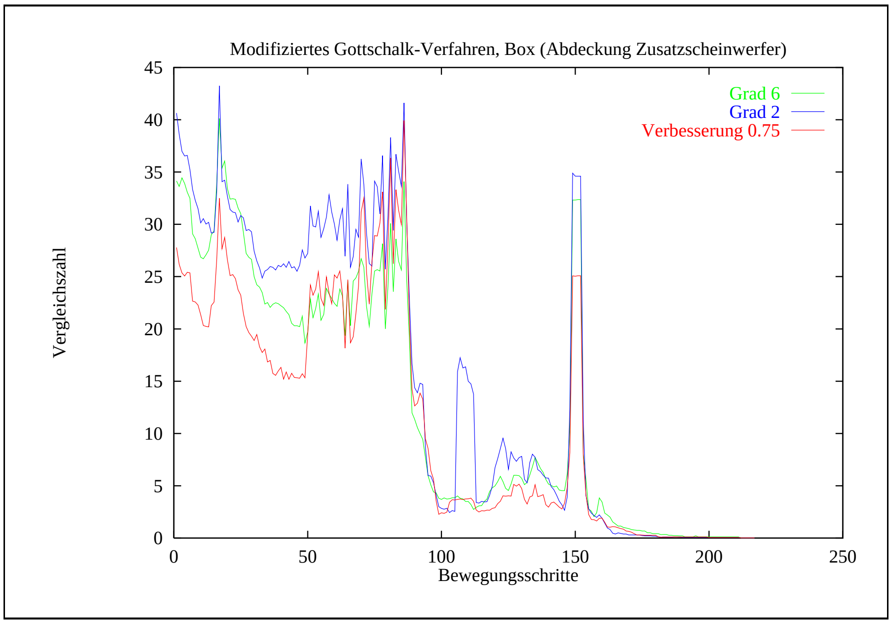

Abbildung D.19: Vergleich der besten Aufteilungsgrade und der besten Verbesserungsrate (modifiziertes Gottschalk-Verfahren)

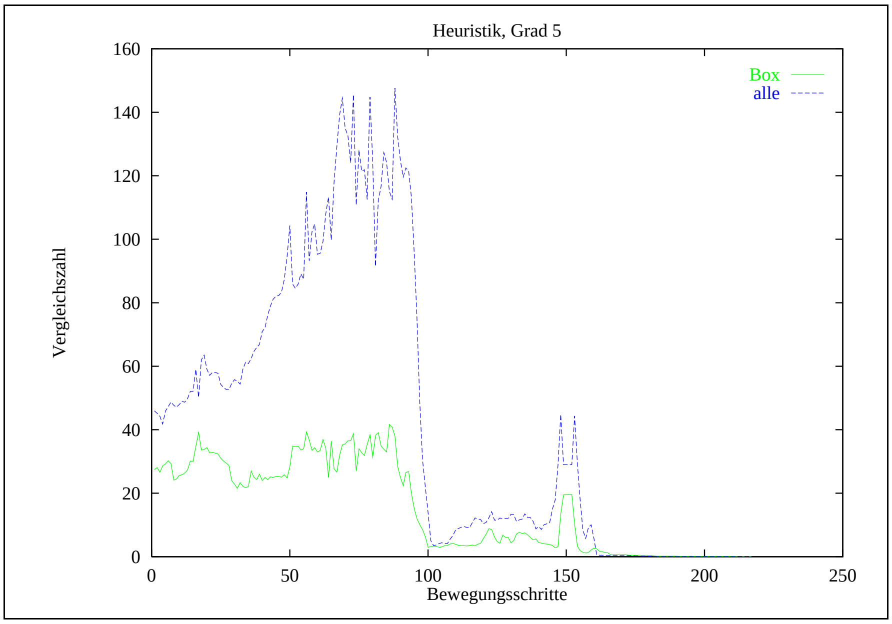

Abbildung D.20: Vergleich Boxen versus alle Hüllkörpertypen bei Grad 5 (Heuristik ohne Zwischenoptimierung)

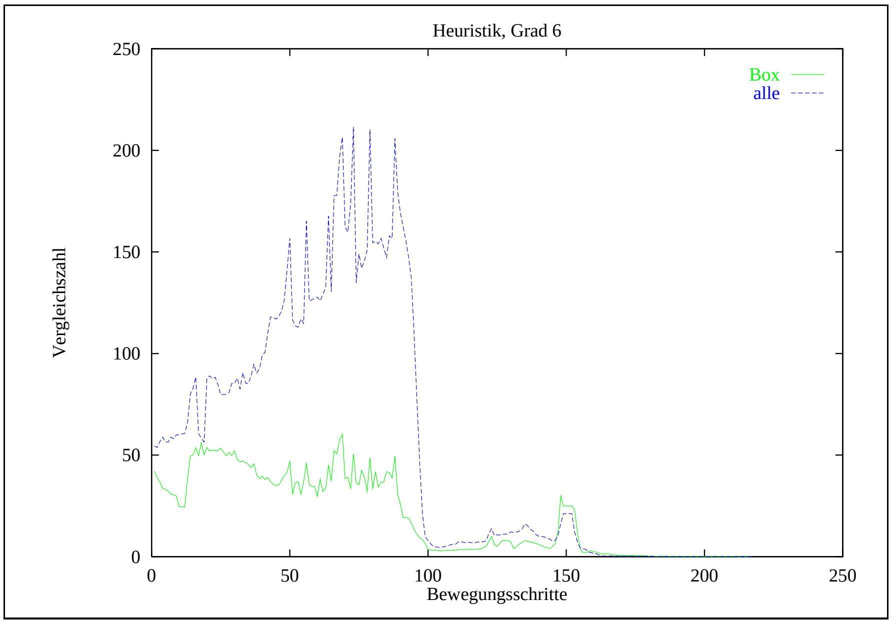

Abbildung D.21: Vergleich Boxen versus alle Hüllkörpertypen bei Grad 6 (Heuristik ohne Zwischenoptimierung)

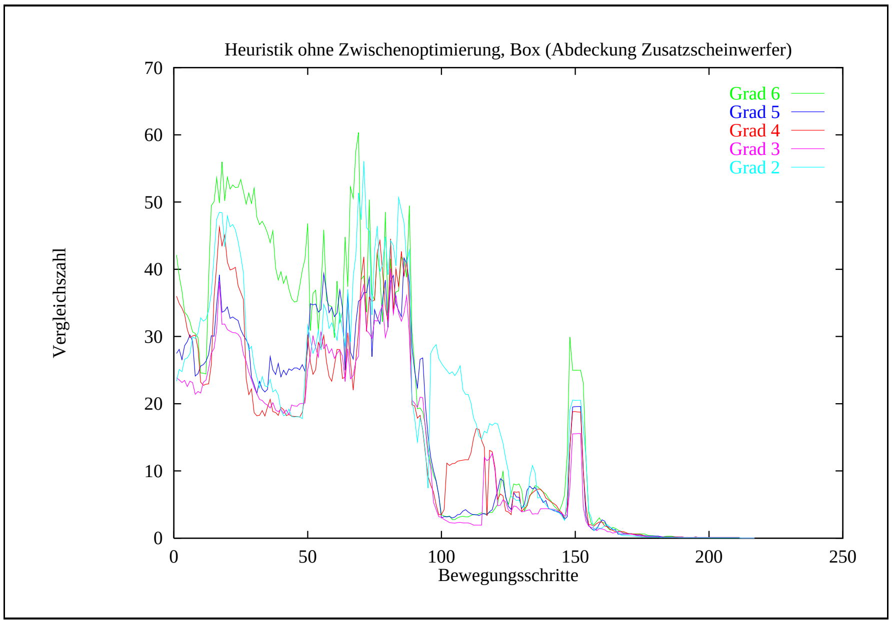

Abbildung D.22: Vergleich der festen Aufteilungsgrade (Heuristik ohne Zwischenoptimierung)

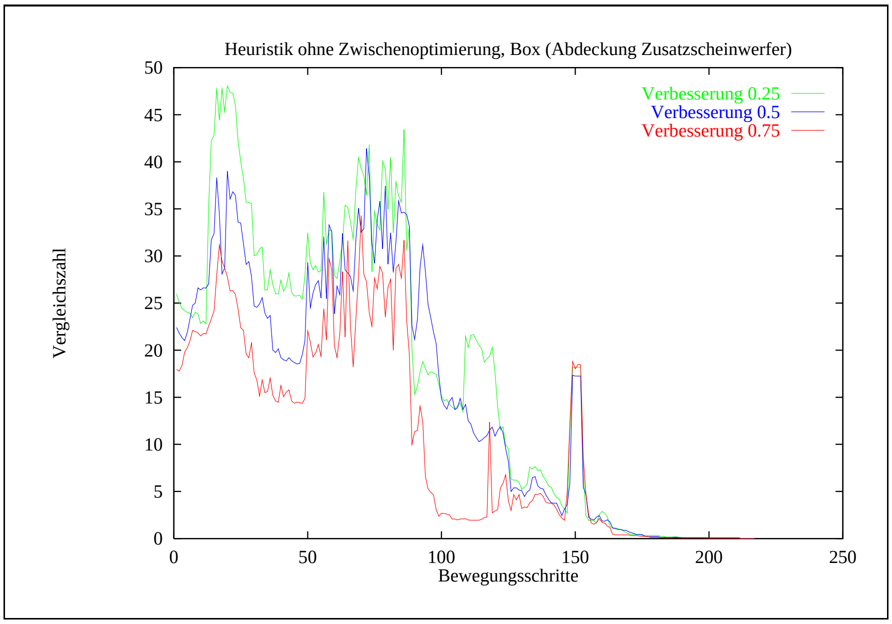

Abbildung D.23: Vergleich der Verbesserungsraten (Heuristik ohne Zwischenoptimierung)

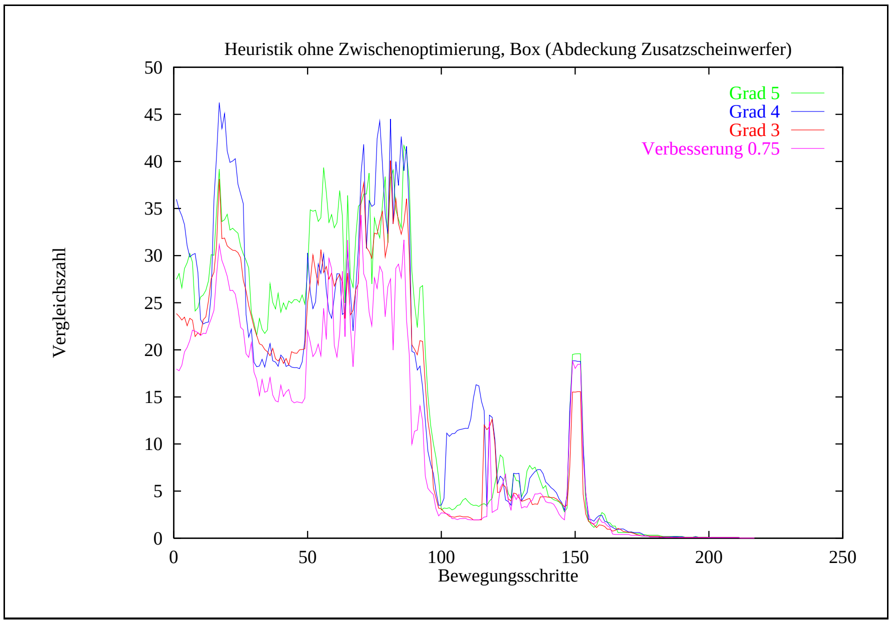

Abbildung D.24: Vergleich der besten Aufteilungsgrade und der besten Verbesserungsrate (Heuristik ohne Zwischenoptimierung)

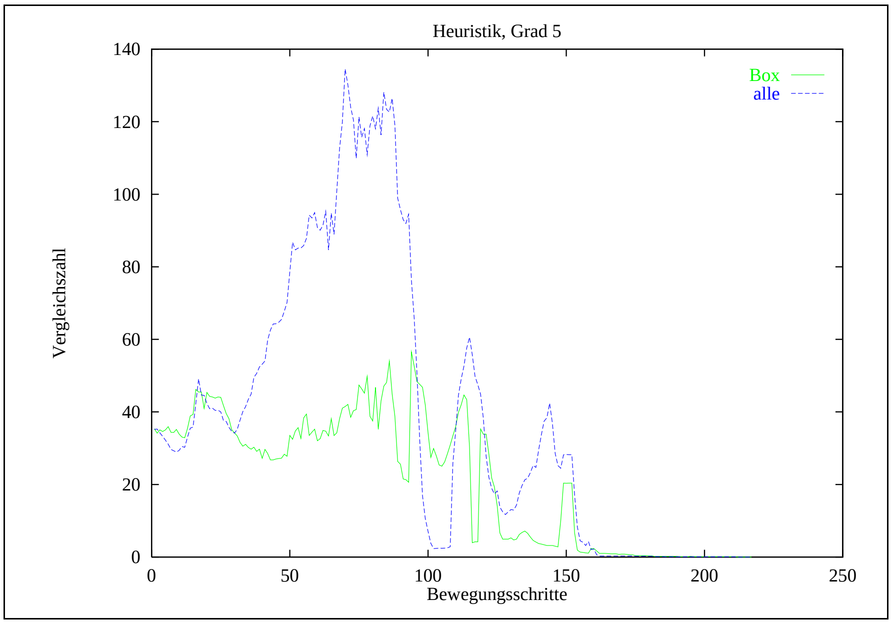

Abbildung D.25: Vergleich Boxen versus alle Hüllkörpertypen bei Grad 5 (Heuristik)

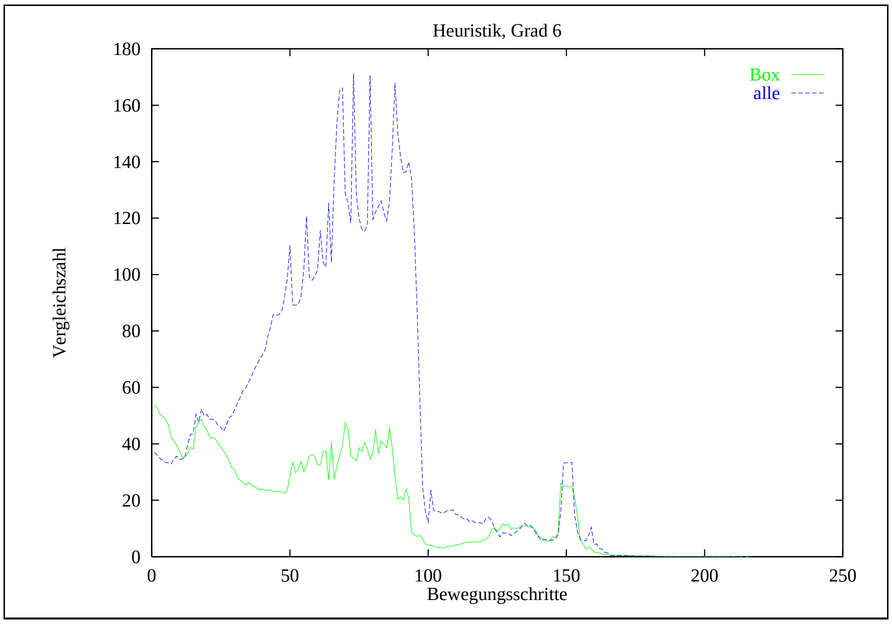

Abbildung D.26: Vergleich Boxen versus alle Hüllkörpertypen bei Grad 6 (Heuristik)

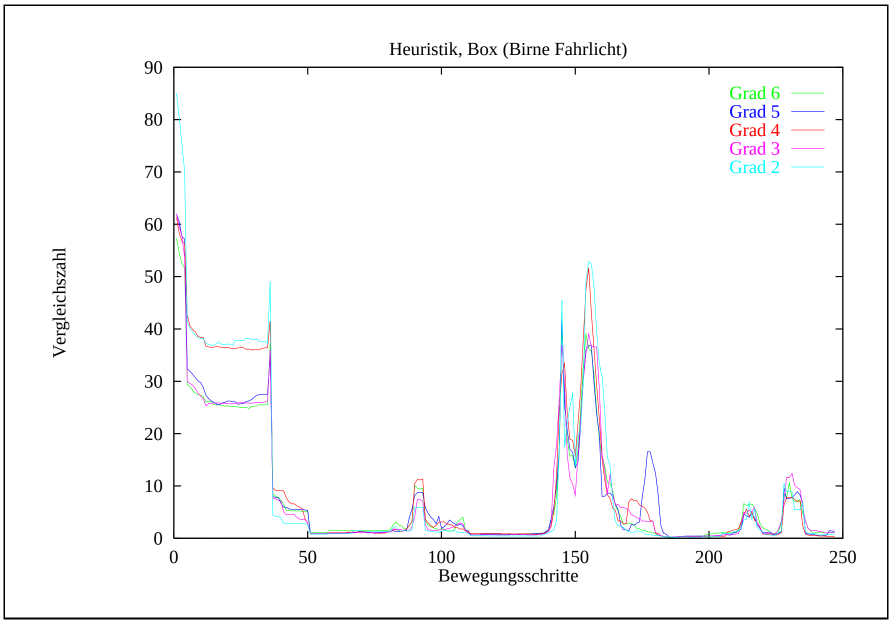

Abbildung D.27: Vergleich der festen Aufteilungsgrade (Heuristik)

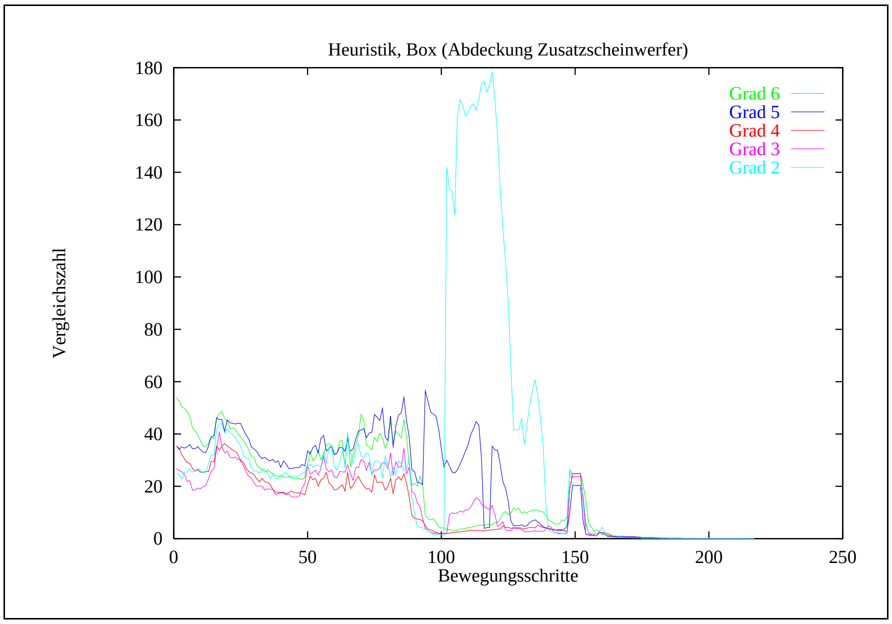

Abbildung D.28: Vergleich der festen Aufteilungsgrade (Heuristik)

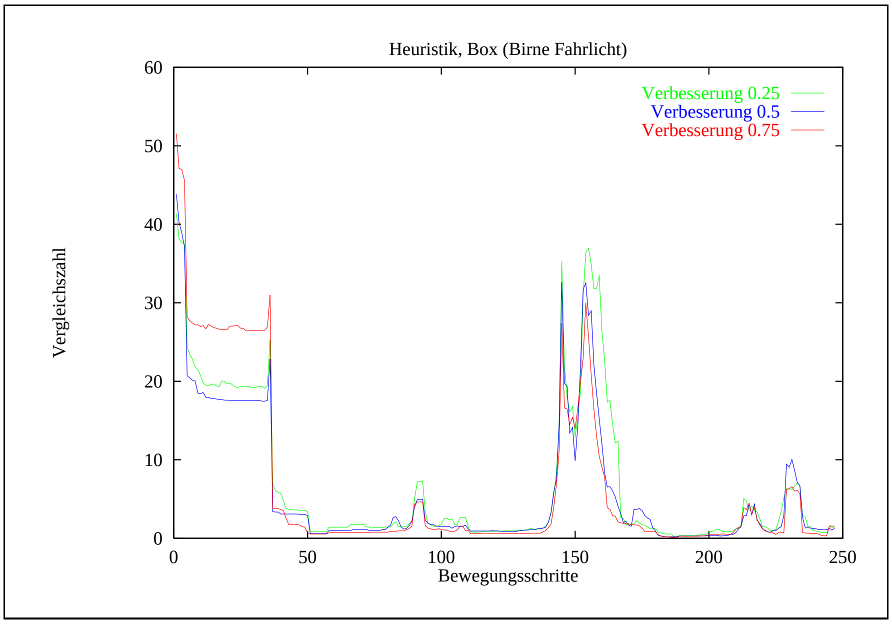

Abbildung D.29: Vergleich der Verbesserungsraten (Heuristik)

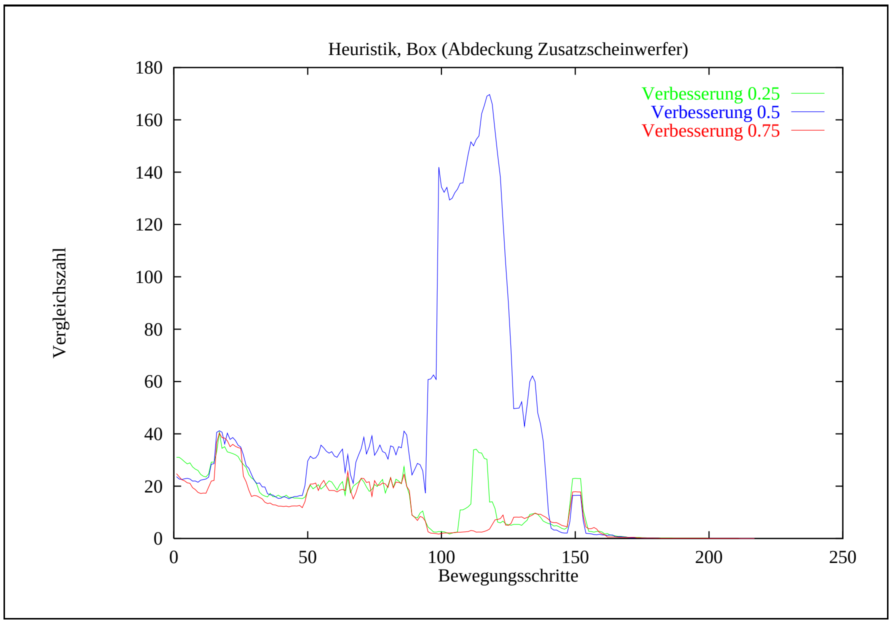

Abbildung D.30: Vergleich der Verbesserungsraten (Heuristik)

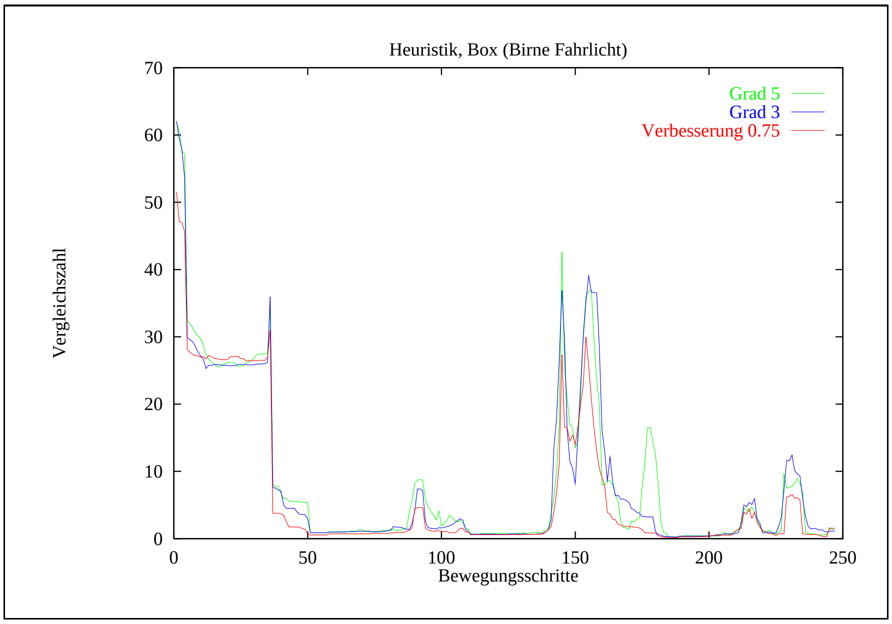

Abbildung D.31: Vergleich der besten Aufteilungsgrade und der besten Verbesserungsrate (Heuristik)

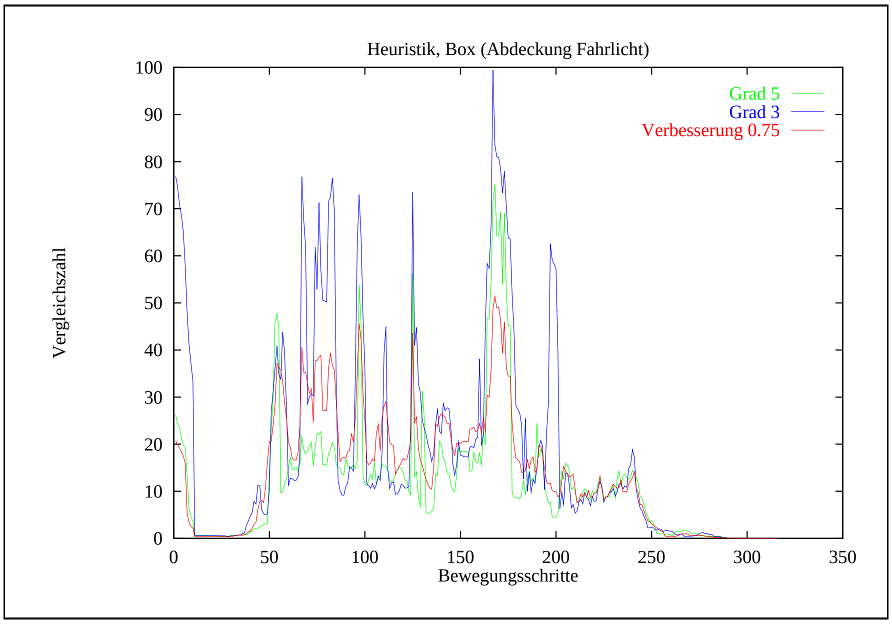

Abbildung D.32: Vergleich der besten Aufteilungsgrade und der besten Verbesserungsrate (Heuristik)

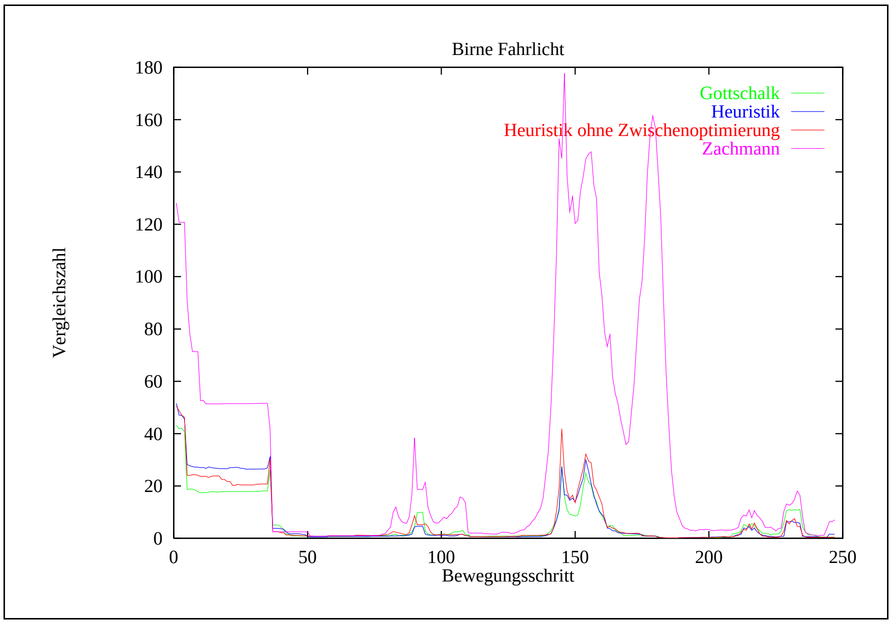

Abbildung D.33: Vergleich der Hüllkörperhierarchieverfahren (mit Zachmann-Verfahren)

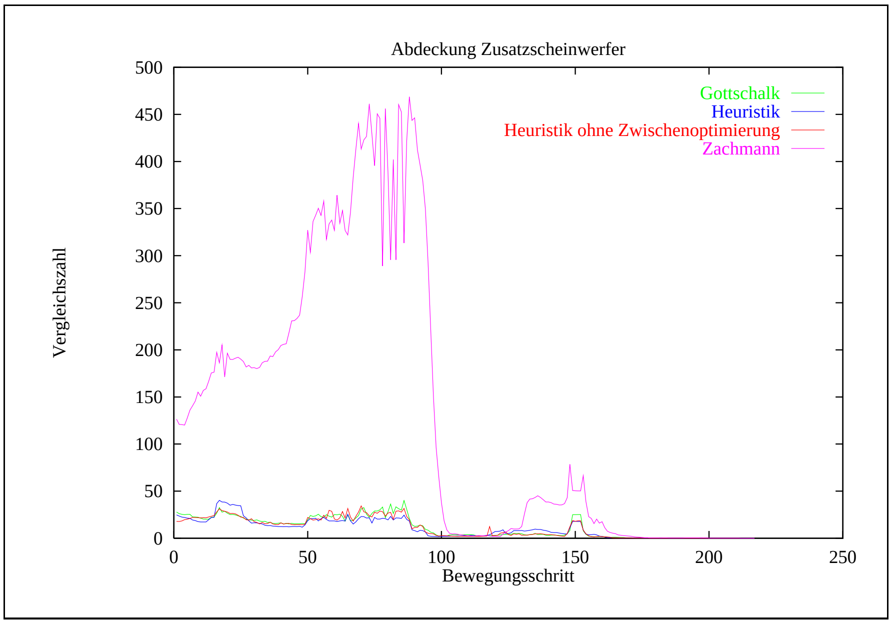

Abbildung D.34: Vergleich der Hüllkörperhierarchieverfahren (mit Zachmann-Verfahren)

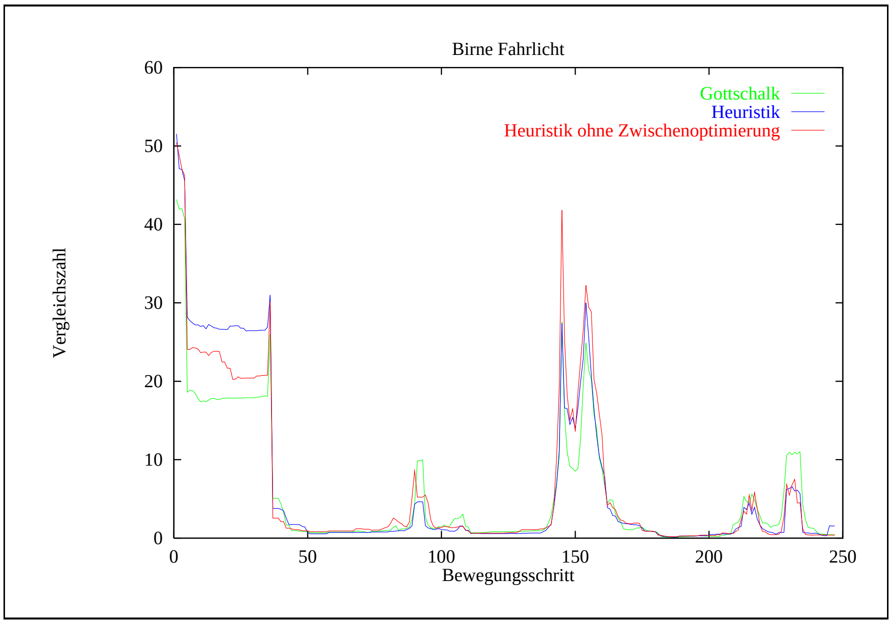

Abbildung D.35: Vergleich der Hüllkörperhierarchieverfahren (ohne Zachmann-Verfahren)

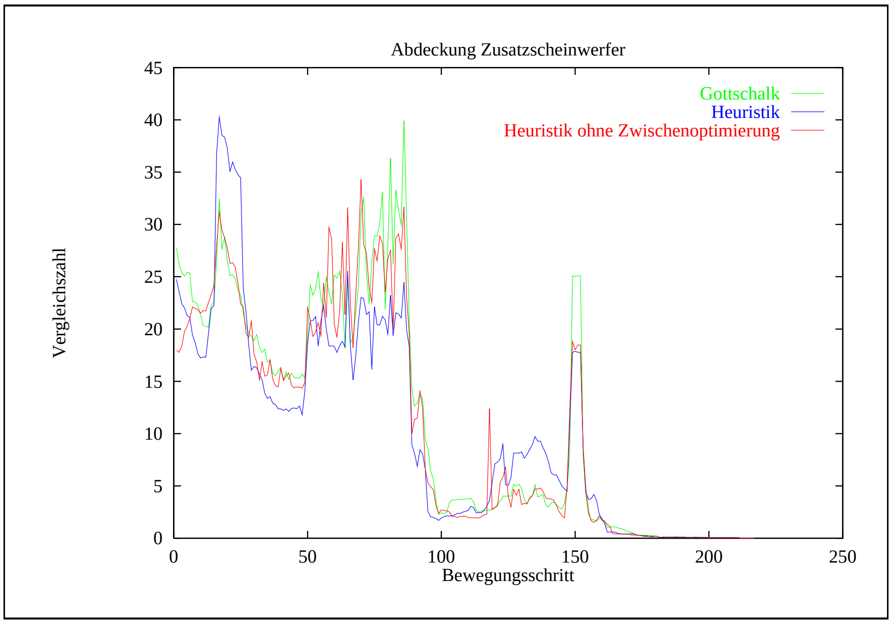

Abbildung D.36: Vergleich der Hüllkörperhierarchieverfahren (ohne Zachmann-Verfahren)

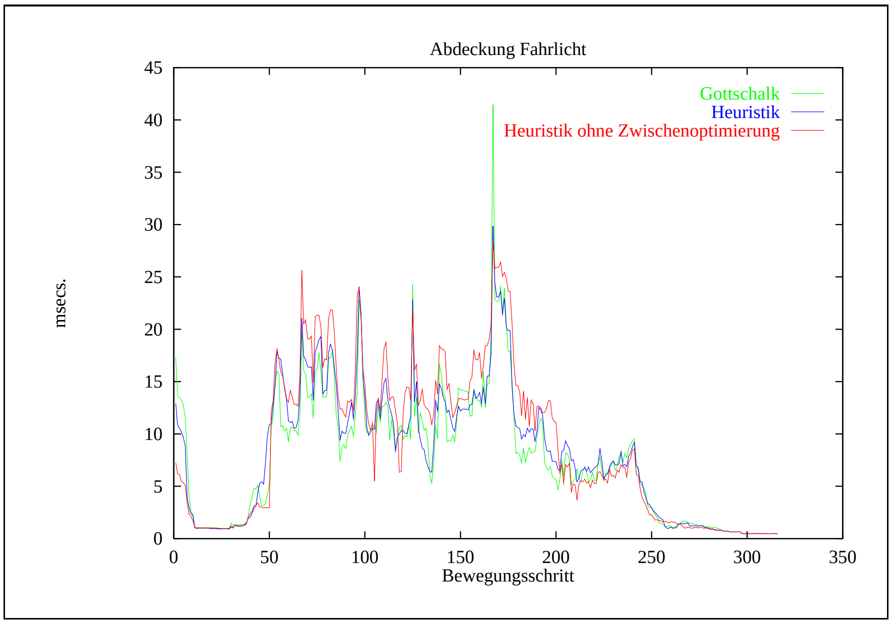

Abbildung D.37: Laufzeitvergleich der Hüllkörperhierarchieverfahren (ohne Zachmann-Verfahren)

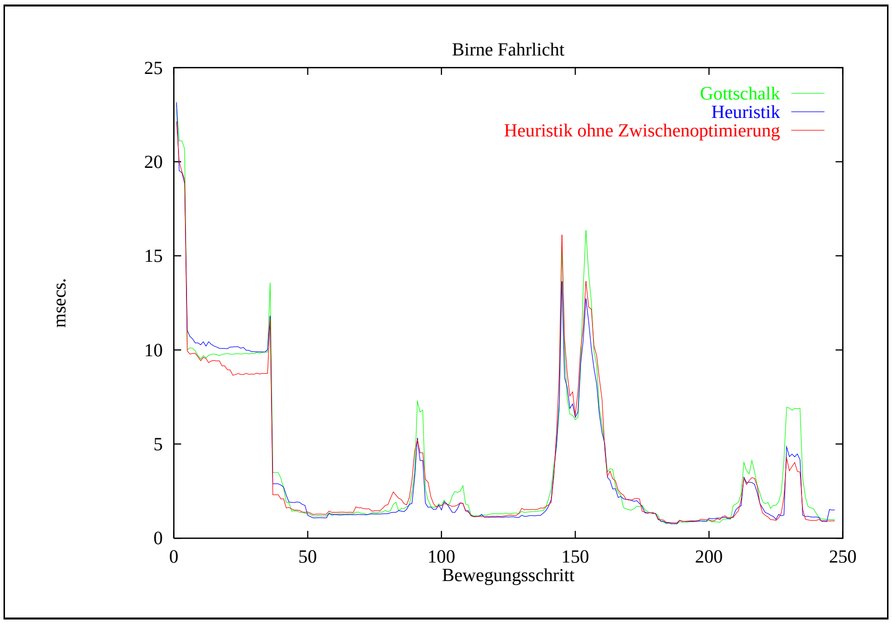

Abbildung D.38: Laufzeitvergleich der Hüllkörperhierarchieverfahren (ohne Zachmann-Verfahren)

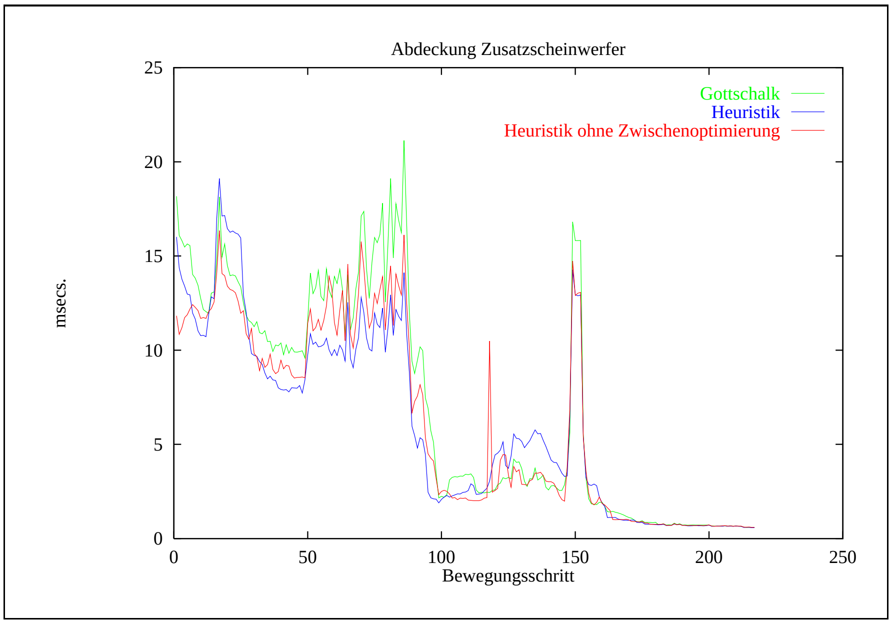

Abbildung D.39: Laufzeitvergleich der Hüllkörperhierarchieverfahren (ohne Zachmann-Verfahren)

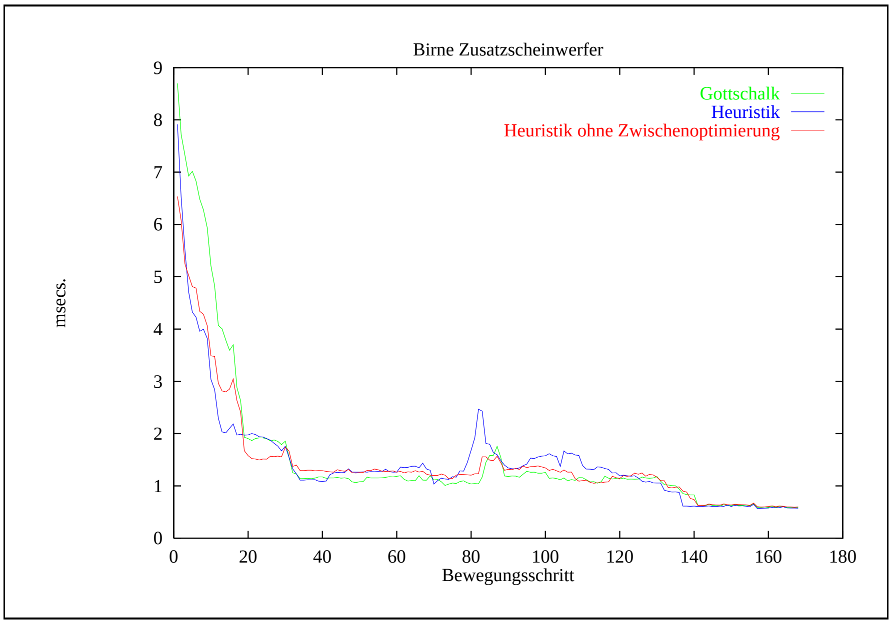

Abbildung D.40: Laufzeitvergleich der Hüllkörperhierarchieverfahren (ohne Zachmann-Verfahren)

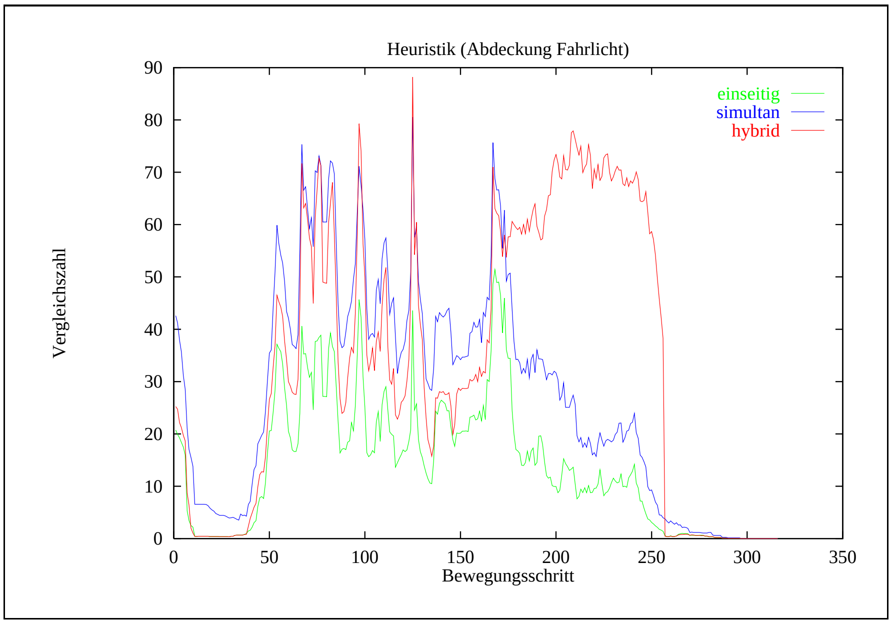

Abbildung D.41: Vergleich der Hierarchietraversierungsverfahren (Heuristik)

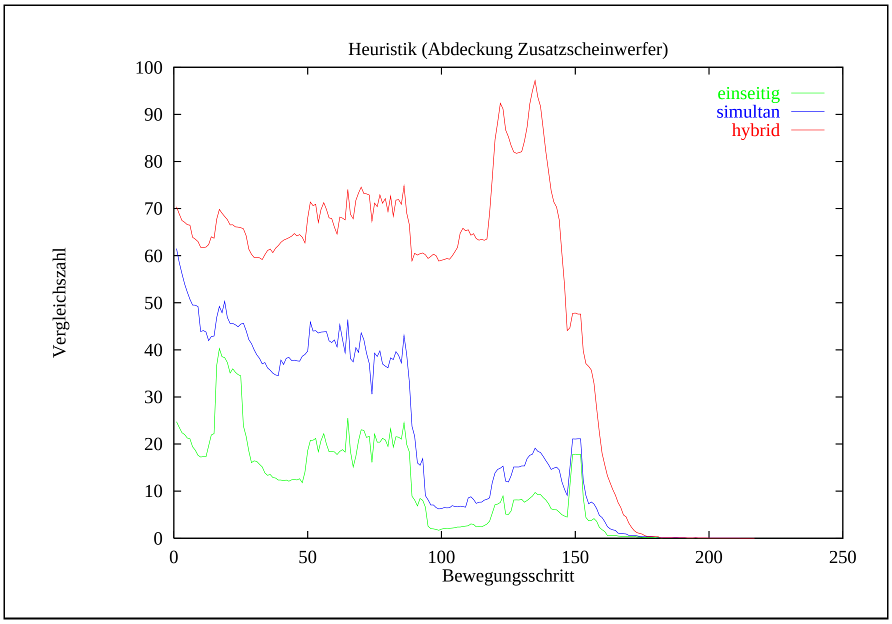

Abbildung D.42: Vergleich der Hierarchietraversierungsverfahren (Heuristik)

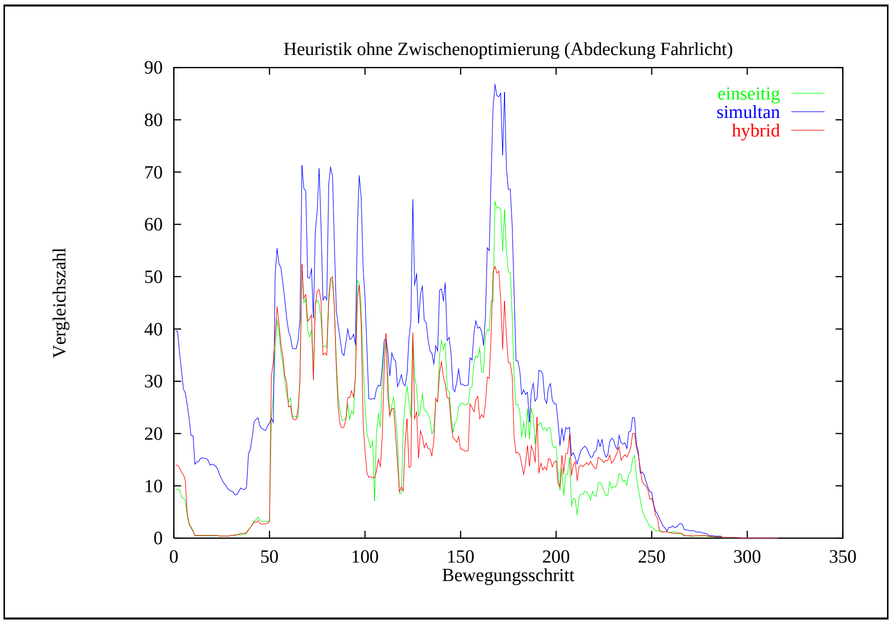

Abbildung D.43: Vergleich der Hierarchietraversierungsverfahren (Heuristik ohne Zwischenoptimierung)

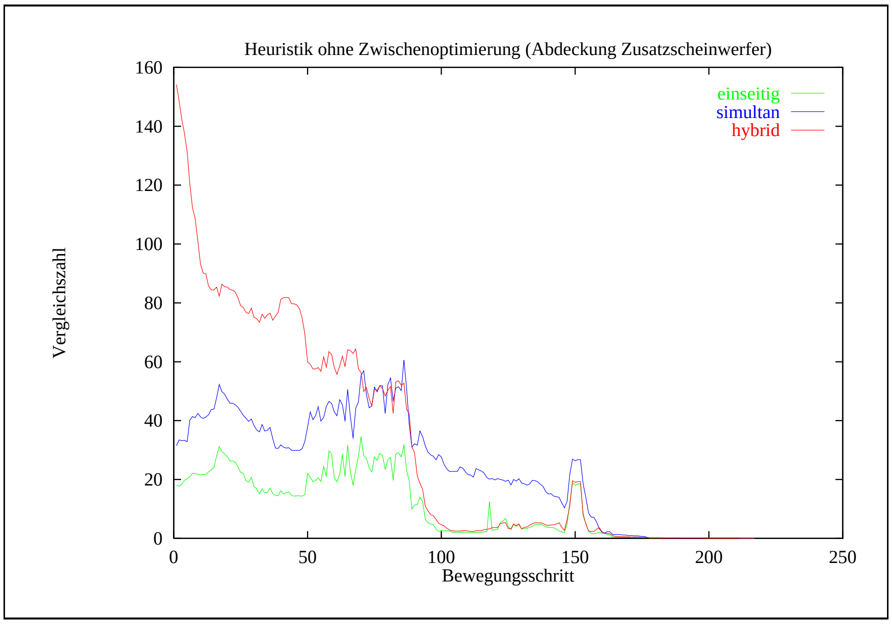

Abbildung D.44: Vergleich der Hierarchietraversierungsverfahren (Heuristik ohne Zwischenoptimierung)

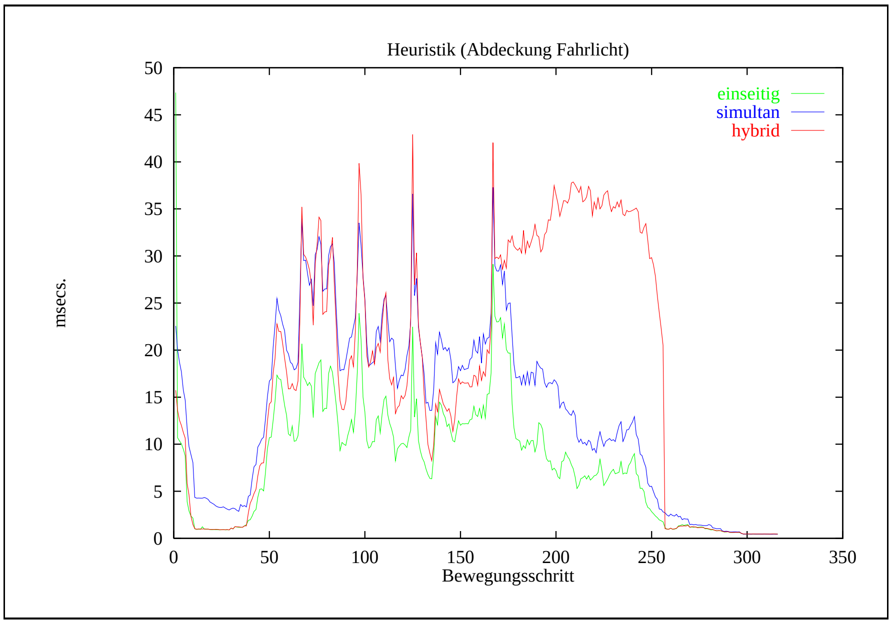

Abbildung D.45: Laufzeitvergleich der Hierarchietraversierungsverfahren (Heuristik)

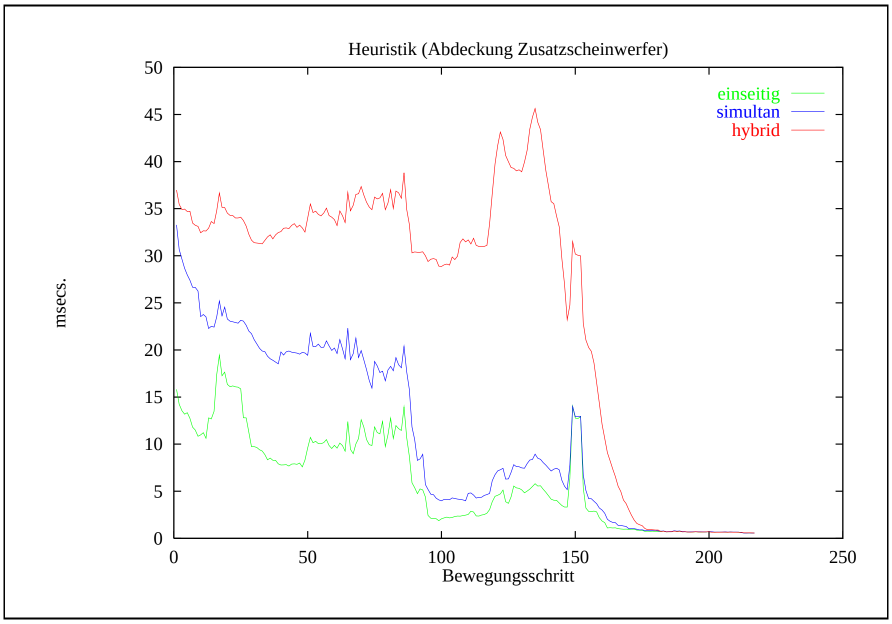

Abbildung D.46: Laufzeitvergleich der Hierarchietraversierungsverfahren (Heuristik)

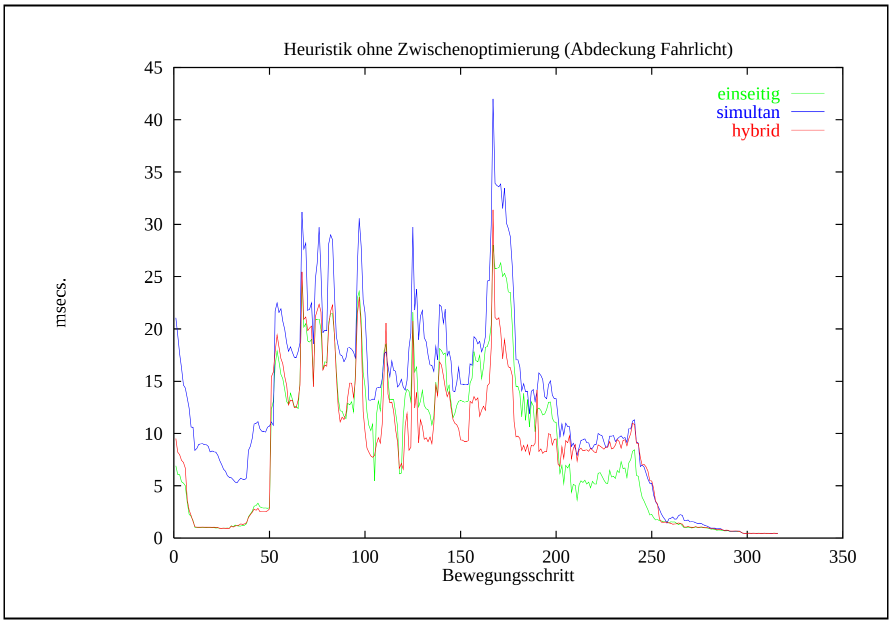

Abbildung D.47: Laufzeitvergleich der Hierarchietraversierungsverfahren (Heuristik ohne Zwischenoptimierung)

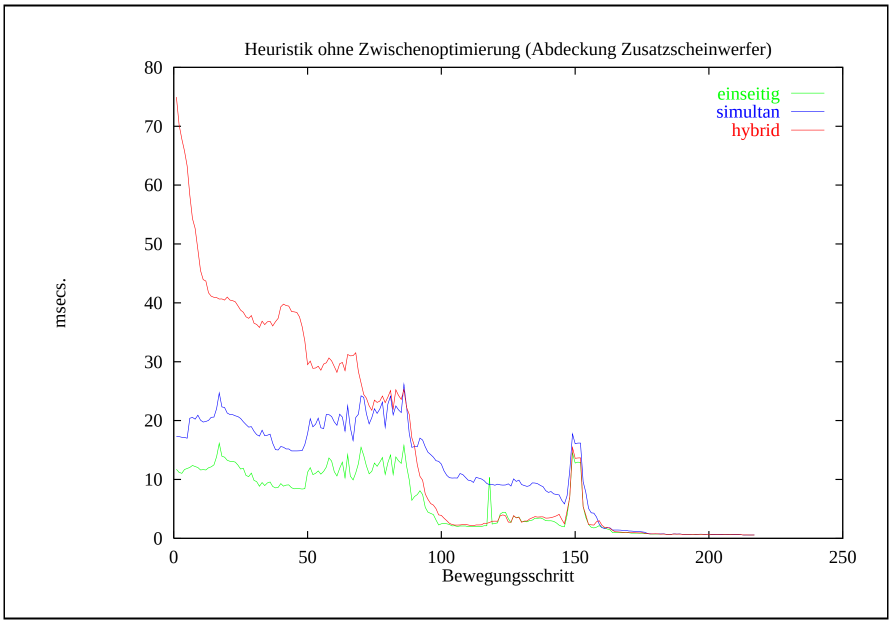

Abbildung D.48: Laufzeitvergleich der Hierarchietraversierungsverfahren (Heuristik ohne Zwischenoptimierung)

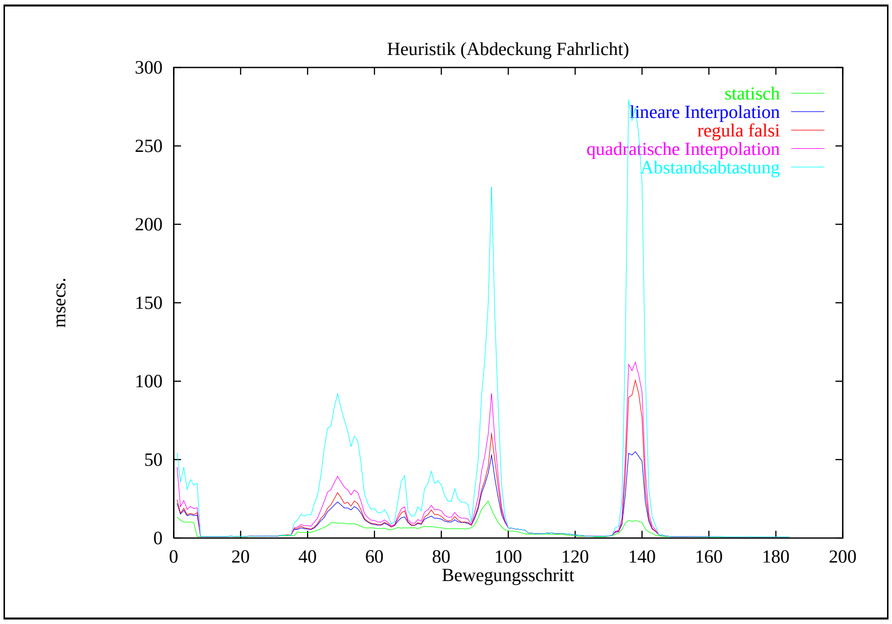

Abbildung D.49: Laufzeitvergleich statische und dynamische Kollisionserkennung (Heuristik ohne ZwischenOptimierung)

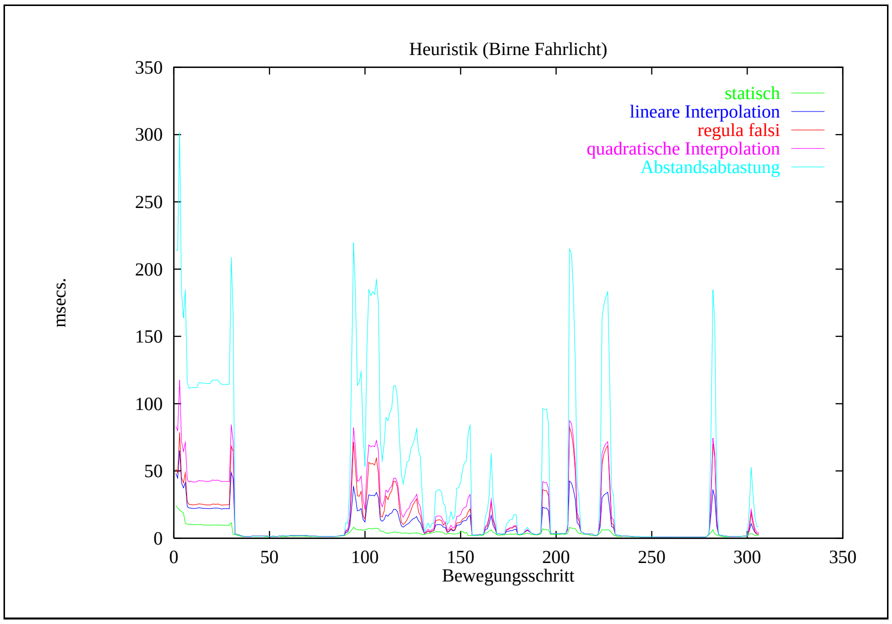

Abbildung D.50: Laufzeitvergleich statische und dynamische Kollisionserkennung (Heuristik ohne ZwischenOptimierung)

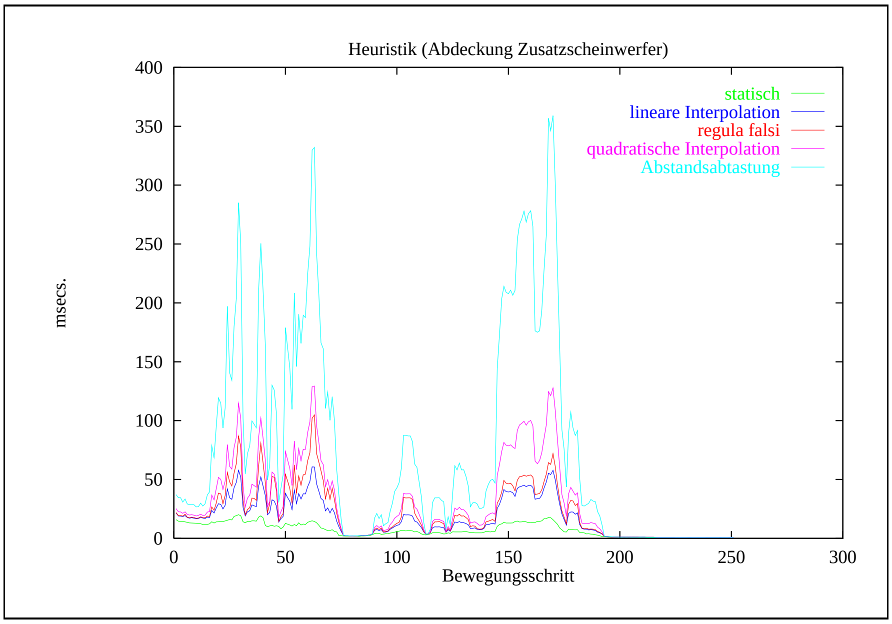

Abbildung D.51: Laufzeitvergleich statische und dynamische Kollisionserkennung (Heuristik ohne ZwischenOptimierung)

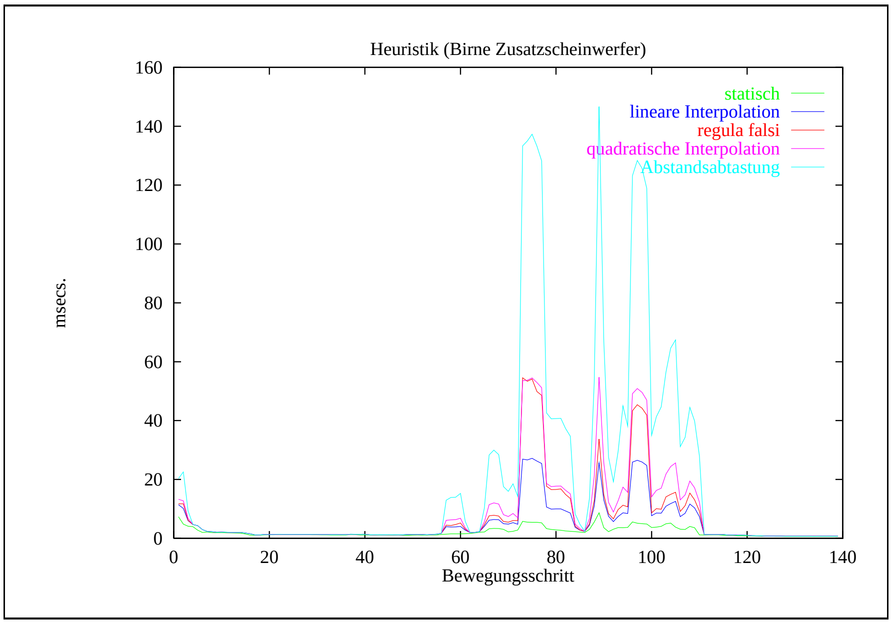

Abbildung D.52: Laufzeitvergleich statische und dynamische Kollisionserkennung (Heuristik ohne ZwischenOptimierung)

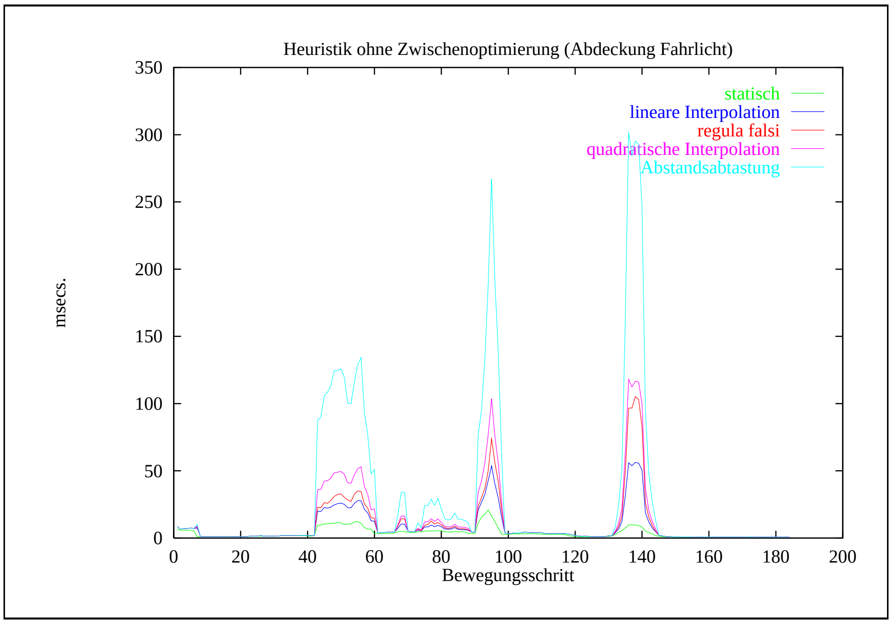

Abbildung D.53: Laufzeitvergleich statische und dynamische Kollisionserkennung (Heuristik ohne ZwischenOptimierung)

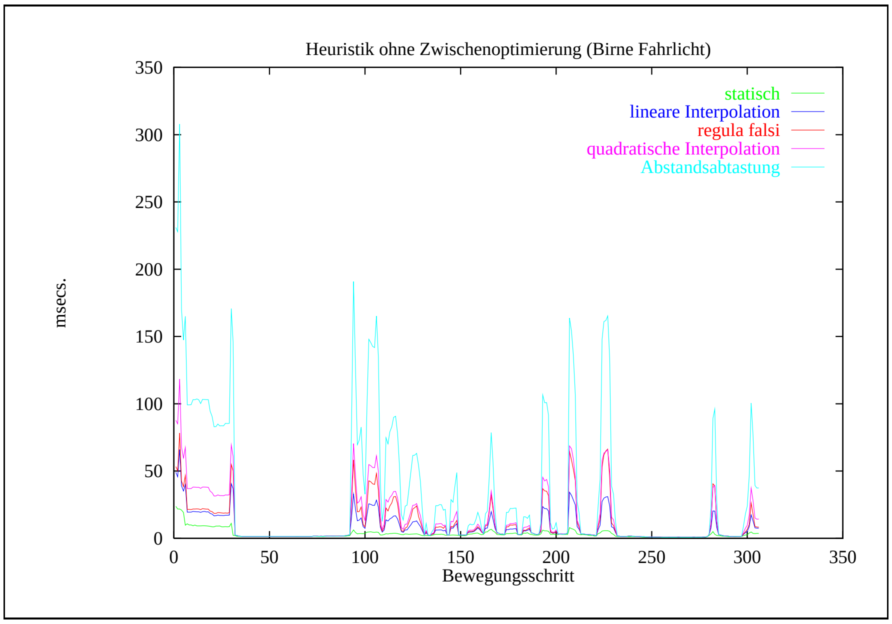

Abbildung D.54: Laufzeitvergleich statische und dynamische Kollisionserkennung (Heuristik ohne ZwischenOptimierung)

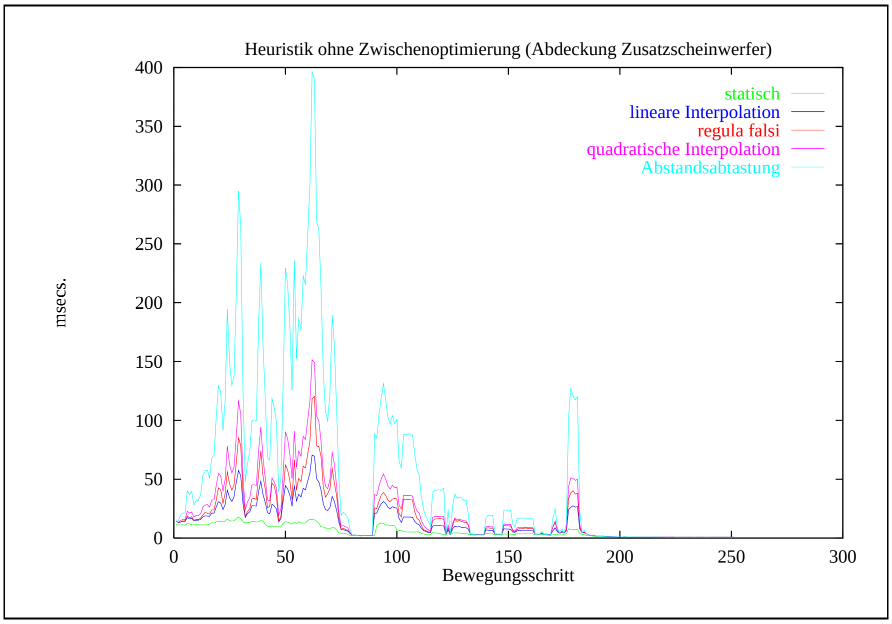

Abbildung D.55: Laufzeitvergleich statische und dynamische Kollisionserkennung (Heuristik ohne ZwischenOptimierung)

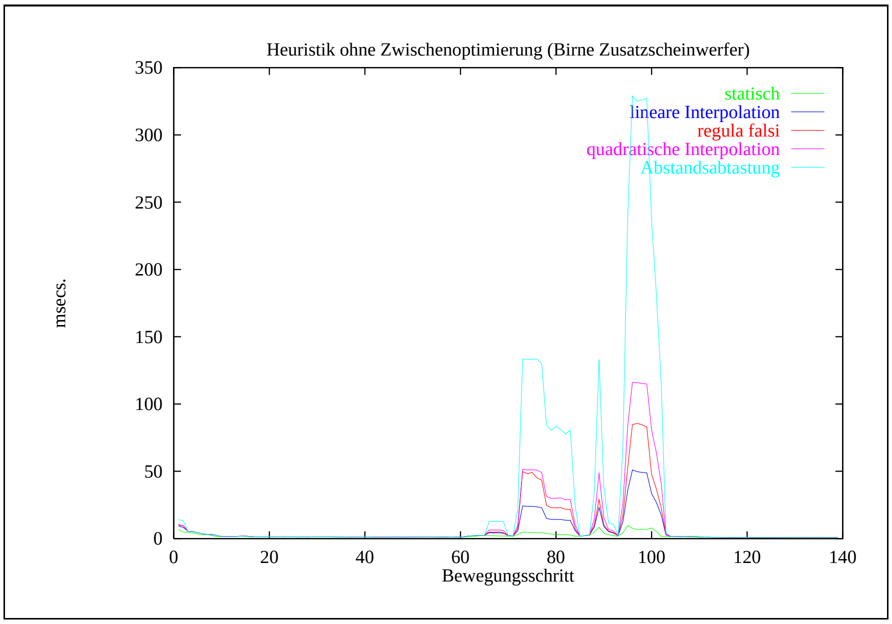

Abbildung D.56: Laufzeitvergleich statische und dynamische Kollisionserkennung (Heuristik ohne ZwischenOptimierung)

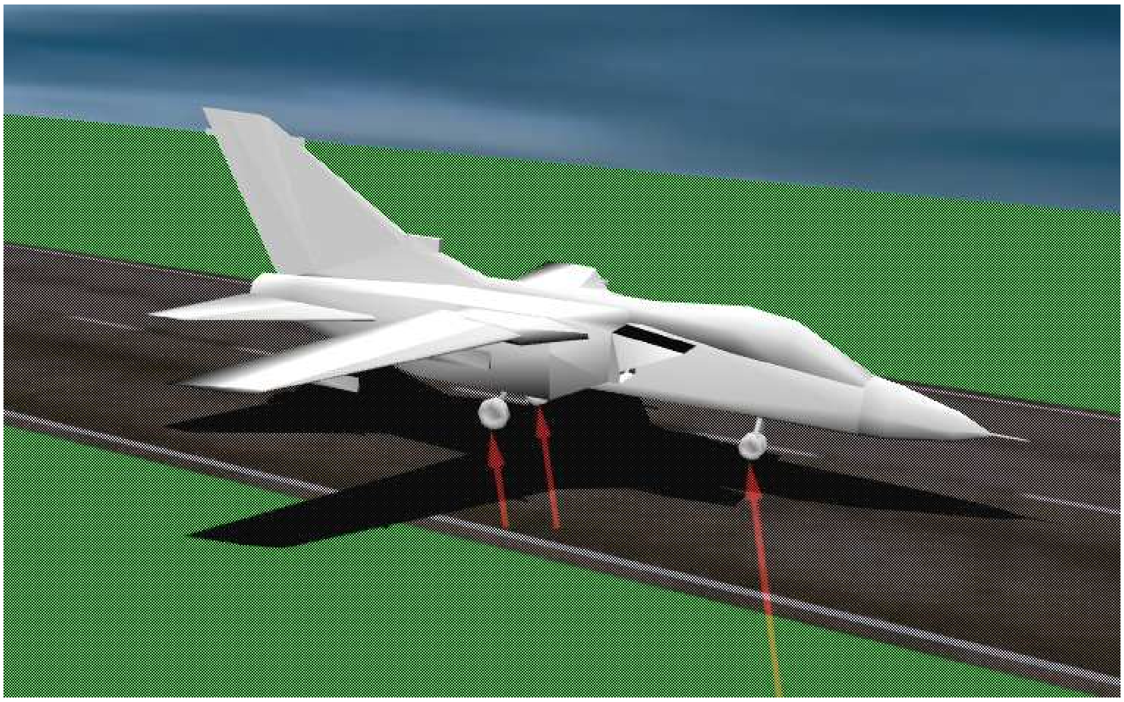

Abbildung D.57: Gleiten eines Flugzeuges über eine Landebahn mit Hilfe dynamischer Kollisionserkennung und Kontaktsimulation

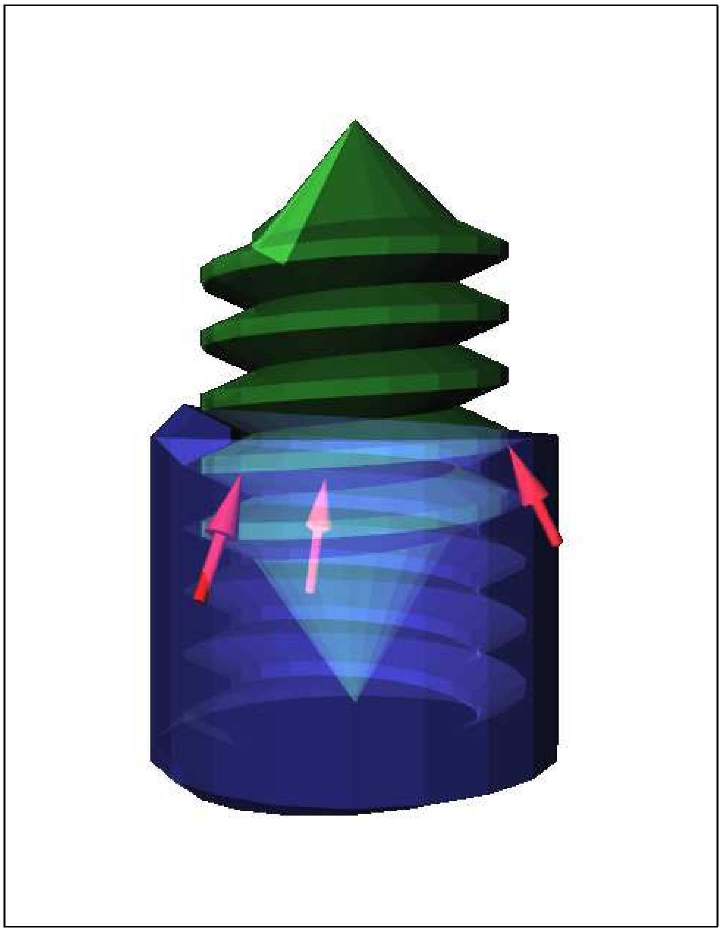

Abbildung D.58: Herausdrehen einer Schraube aus einer Mutter mit Hilfe dynamischer Kollisionserkennung und Kontaktsimulation

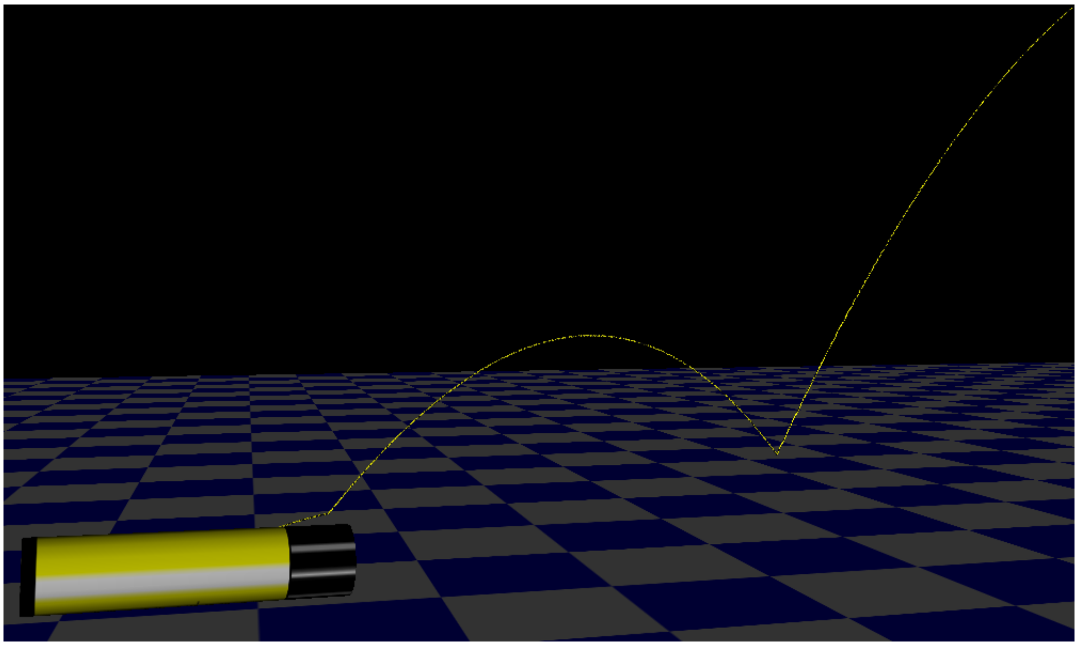

Abbildung D.59: Physikalisch korrektes Fallen eines Stabes mittels dynamischer Kollisionserkennung und impulsbasierter Dynamiksimulation

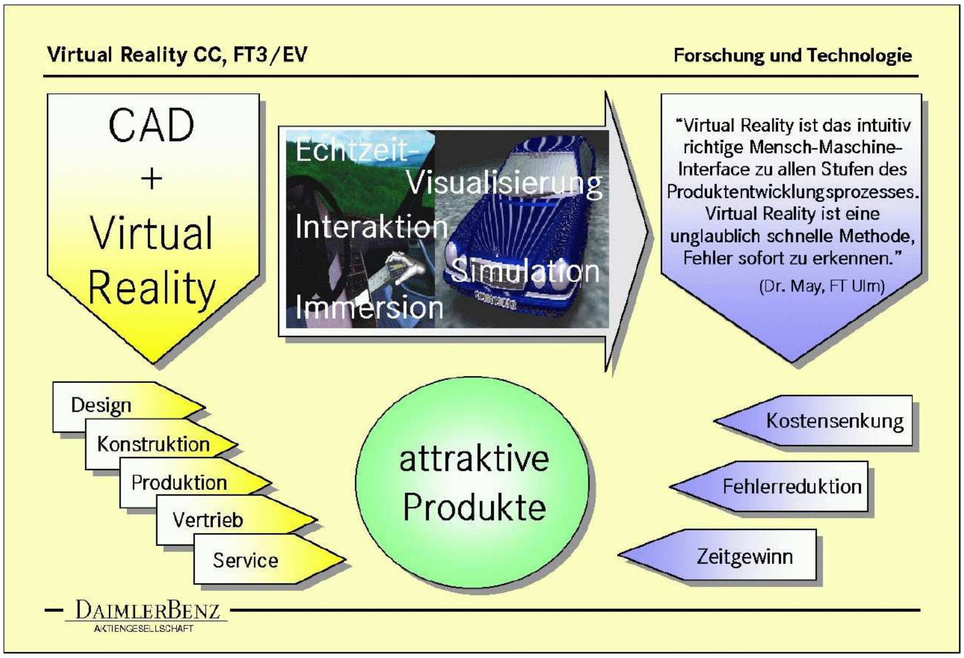

Abbildung D.60: Einsatzgebiete und Wirkungen von Virtual Reality

Abbildung D.61: Virtual Reality-Softwareplattform **DBView**