Kollisionserkennung für echtzeitfähige Starrkörpersimulationen in der Industrie- und Servicerobotik

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von **Dipl.-Inf. Fabian Aichele** aus München

Hauptberichter: *Prof. Dr. rer. nat. habil. Paul Levi*
Mitberichter: *Prof. Dr.-Ing. Michael Weyrich*
Tag der mündlichen Prüfung: *23.04.2015*

Institut für Parallele und Verteilte Systeme
der Universität Stuttgart

2015

Vorwort

Diese Dissertation entstand während meiner Zeit als wissenschaftlicher Mitarbeiter in der Abteilung Bildverstehen am Institut für Parallele und Verteilte Systeme (IPVS) der Universität Stuttgart.
Besonders bedanken möchte ich mich bei Herrn Prof. Levi für die Betreuung dieser Arbeit, das sehr gute Arbeitsklima in der Abteilung und die Gelegenheit bedanken, als Teil dieser Dissertation ein Industrieprojekt in Kooperation mit der Daimler AG durchführen zu können.

Des weiteren möchte ich mich bei meinem ehemaligen Kollegen Dr. Mario Rossdeutscher und bei Dipl.-Ing. Willi Klumpp für die hervorragende Zusammenarbeit im Lauf dieses Industrieprojekts bedanken.

Ebenso gilt mein Dank meinen jetzigen und ehemaligen Kollegen an der Abteilung Bildverstehen für die stets exzellente Zusammenarbeit und die konstruktiven Diskussionen im Zusammenhang mit der vorgelegten Dissertation. Besonders zu erwähnen sind Dr. Oliver Zweigle, Kai Häussermann, Daniel Di Marco, Bernd Eckstein und Andreas Koch.

Abschließend gilt mein besonderer Dank meiner Familie, und besonders meinen Eltern, für die stets gewährte tatkräftige Unterstützung und Motivation während meines Studiums und meiner Promotion.

Stuttgart, 2014 Fabian Aichele

INHALTSVERZEICHNIS

ZUSAMMENFASSUNG

Die mechanisch plausible Simulation von Robotern und deren Arbeitsumgebungen ist in der Industrie- und Service-Robotik ein zunehmend wichtiges Werkzeug bei der Entwicklung und Erprobung neuer Hardware und Algorithmen. Ebenso sind Simulationsanwendungen oftmals eine kostengünstige und vielseitig einsetzbare Alternative, sofern die Beschaffung echter Roboter unrentabel ist, oder Hardware und Arbeitsumgebung nur mit großer zeitlicher Verzögerung zur Verfügung stehen würden.

Besonders wichtig sind Mechanik-Simulationen für Anwendungsfälle, in denen die direkte mechanische Interaktion von Objekten miteinander beziehungsweise der Arbeitsumgebung selbst im Vordergrund stehen, wie etwa in der Greifplanung oder der Ermittlung kollisionsfreier Bewegungsabläufe.

Bei welcher Art von Szenarien der Einsatz von Mechanik-Simulationen sinnvoll ist und inwieweit die Möglichkeiten solcher Simulations-Werkzeuge ein geeigneter Ersatz für eine reale Arbeitsumgebung sein können, hängt sowohl von den technischen Besonderheiten dieser Werkzeuge, als auch von den Anforderungen des jeweiligen Anwendungsgebiets ab.

Die wichtigsten Kriterien sind dabei:

- Die zur Umsetzung der jeweiligen Aufgabe nötige oder gewünschte geometrische Präzision bei der Modellierung von Objekten in einer Simulation.
- Die bei der Simulation mechanischem Verhaltens berücksichtigten Eigenschaften und Phänomene, etwa durch die Berücksichtigung von Verformungsarbeit oder tribologischer Eigenschaften.
- Die Fähigkeit, eine Simulation in oder nahe Echtzeit betreiben zu können: Innerhalb von Laufzeitgrenzen, wie sie auch durch die reale Entsprechung eines simulierten Systems gegeben sind.

Die Fähigkeit zum Echtzeit-Betrieb steht dabei in Konflikt mit der geometrischen und mechanischen Präzision einer Simulation. Jedoch ist es gerade die Kombination aus diesen drei Kriterien, die für Szenarien mit einem hohen Anteil mechanischer Interaktion zwischen aktiv durch einen Benutzer gesteuerten Aktorik und einer simulierten Arbeitsumgebung besonders wichtig sind: Im Speziellen gilt das für Simulationssysteme, die zur Steuerung simulierter

Roboter-Hardware dieselben Hardware- oder Software-Steuerungen verwenden, die auch für die realen Entsprechungen der betrachteten Systeme verwendet werden.

Um einen Betrieb innerhalb sehr kurzer Iterationszeiten gewährleisten zu können, muss eine Mechanik-Simulation zwei Teilaufgaben effizient bewältigen können:

- Die Überprüfung auf Berührung und Überschneidung zwischen simulierten Objekten in der *Kollisionserkennung* in komplex strukturierten dreidimensionalen Szenen.
- Die Gewährleistung einer numerisch stabilen Lösung des zugrundeliegenden Gleichungssystems aus der klassischen Mechanik in der *Kollisionsbehandlung*.

Die Kollisionserkennung erfordert dabei gegenüber der Kollisionsbehandlung ein Vielfaches an Laufzeit-Aufwand, und ist dementsprechend die Komponente einer jeden echtzeitfähigen Mechanik-Simulation mit dem größten Optimierungspotential und -bedarf: Ein Schwerpunkt der vorliegenden Arbeit ist daher die Kombination existierender Ansätze zur Kollisionserkennung unter weitgehender Vermeidung von deren Nachteilen.

Dazu sollen ausgehend von Erfahrungen einer Projektstudie aus der Industrie-Robotik die speziellen Anforderungen an echtzeitfähige Mechanik-Simulationen beim Einsatz in dieser und verwandten Disziplinen hergeleitet und den Möglichkeiten und Einschränkungen existierender Simulations-Lösungen gegenüber gestellt werden.

Basierend auf der Analyse existierender Kollisionserkennnungs-Verfahren soll im weiteren Verlauf der Arbeit eine alternative Möglichkeit zur Bewältigung dieser laufzeitaufwendigen Aufgabe auf Basis der Verwendung massiv paralleler Prozessor-Architekturen, wie sie in Form programmierbarer Grafik-Prozessoren (GPUs) kostengünstig zur Verfügung stehen, erarbeitet und umgesetzt werden.

SUMMARY

Plausible mechanical simulations of robots and their working environments is an increasingly important tool for the development and testing of new hardware and algorithms in robotics. Furthermore, simulation tools often are a cheap and versatile alternative, if the purchase of real robots is nonviable, or the hardware itself or the working environment is not accessible, or will only be available with long delay.

Rigid body simulations are especially important for applications that heavily rely on direct mechanical interactions with objects in the working environment, or the working environment itself, for example in grasp planning or the computation of collision-free motion sequences.

In which cases the use of rigid body simulations is sensible, and how well the abilities of such simulation tools can be a suitable replacement for real working environments, depends on their specific characteristics as well as on the requirements of the intended application domain. The most important criteria to consider are:

- The required or desired precision of geometric models used in a simulation.
- The properties and phenomena that are incorporated when simulating the mechanical behaviour of objects, for example tribological properties or the inclusion of deformation energy.
- The ability to run a simulation within the same run-time limits that exist in the real counterpart of a simulated system.

The ability to run close to real-time conflicts with the geometric and mechanical precision of a simulation. However it is exactly the combination of the three criteria above that can be very beneficial for applications that employ a high degree of mechanical interactions: This is especially important for simulation systems that use the same hardware or software components to control simulated robots that are also used for controlling the real counterparts of simulated systems.

To be able to operate within very short iteration intervals, a rigid body simulation has to solve two tasks as efficiently as possible:

- Checking simulated objects for contacts and intersections as part of the *collision detection* in three-dimensional environments with complex structure.
- Guaranteeing a numerically stable solution of the underlying equations from classical mechanics as part of *collision handling*.

The collision detection phase typically has much higher run-time requirements than the collision response, and so is the component of every rigid body simulation system with the highest need and potential for run-time optimization: One major topic of this thesis therefore is the combination of existing approaches for collision detection, specifically regarding the avoidance of the drawbacks of these approaches.

Based on the hands-on experience made as part of a pre-investment study in industrial robotics, the specific requirements to real-time capable rigid body simulations in this and similar fields shall be derived and compared with the abilities and restrictions of existing software solutions.

Based on the analysis of existing collision detection methods, an alternative possibility for handling this run-time intensive task based on using massively parallel processor platforms (Graphics Processing Units, GPUs) shall be investigated and implemented.

KAPITEL 1

EINLEITUNG

In der Industrie- und Service-Robotik gewinnt die Nutzung virtueller Modelle als Werkzeug für die Entwicklung und Erprobung von Roboter-Systemen zunehmend an Bedeutung.

Für die Industrie-Robotik machen kurze Innovationszyklen, wachsender Kostendruck durch weltweiten Wettbewerb und bei gleichzeitig steigenden Ansprüchen an Qualität und Zuverlässigkeit von Produktionsanlagen und -prozessen eine effektive Planung und Durchführung der Entwicklung neuer roboter-gestützter Fertigungsanlagen und -prozesse in der automatisierten industriellen Produktion zu einem immer wichtiger werdenden Wettbewerbsfaktor.

Eine besondere Rolle nimmt hier die Entwicklung und Erprobung von Steuerungssoftware ein: Steigende Anforderungen an die Flexibilität von Abläufen für den Umgang mit variantenreichen Produktpaletten einerseits, und höhere Qualitätsansprüche bei der Zuverlässigkeit von Steuerungssoftware zur Vermeidung von Produktionsausfällen andererseits machen die Software-Entwicklung in der industriellen Fertigung zu einem kritischen Faktor bei der Entwicklung von Fertigungsanlagen, in denen roboter-gestützte Produktionsprozesse im Vordergrund stehen.

Nicht alle Produktionsschritte können bereits während der Konzeption einer Anlage entworfen und getestet werden, da deren Ablauf vom Zustand bestimmter Anlagenteile oder Werkstücke abhängt, der erst zur Laufzeit ermittelt werden kann. Die Steuerung solcher Produktionsschritte kann bisher erst dann entworfen und erprobt werden, wenn der Aufbau einer neuen Fertigungsanlage abgeschlossen ist. Dadurch bleibt oft nur ein kurzer Zeitraum bis zum Beginn des Produktionsbetriebs, mit entsprechend engem Zeitplan für die Programmierung und Erprobung der Steuerungssoftware.

Eine Möglichkeit, Entwurf und Erprobung solcher Fertigungsprozesse besser zu unterstützen, ist durch die *virtuelle Inbetriebnahme* gegeben: Anstatt erst an einer fertig aufgebauten realen Anlage mit der Programmierung zu beginnen, kann diese bereits anhand eines virtuellen Modells der späteren Anlage erfolgen, und die Ergebnisse dieser Arbeiten können mit wenig Aufwand für den Einsatz in einer realen Anlage angepasst werden.

Betrachtet man handhabungsintensive Fertigungsschritte (z. B. Greifen und Bewegen oder Zusammenfügen), so gibt es bei der Modellierung solcher Fertigungsschritte an virtuellen Modellen ein wesentliches Problem zu lösen: Das mechanische Verhalten der verwendeten Werkzeuge und Werkstücke so plausibel zu simulieren, dass es dem Verhalten der späteren realen Gegenstücke nahe genug kommt, damit ein virtuelles Modell überhaupt als nützlicher Ersatz für eine reale Anlage dienen kann. Das bedeutet, dass sich Struktur und Form der virtuellen Modelle nicht von ihren realen Gegenstücken unterscheiden dürfen: Für eine Simulations-Software, die in diesem Anwendungsfeld zum Einsatz kommen soll, bedeutet das enorme Leistungsanforderungen bei der Berechnung von mechanischen Kontakten zwischen simulierten Objekten.

In der Service-Robotik ist die Nützlichkeit virtueller Werkzeuge nicht nur durch ökonomische Notwendigkeiten bestimmt. Hier gibt es zwei Probleme, zu deren Lösung Simulations-Software generell beitragen kann:

- Bei der parallelen Entwicklung von Roboter-Hardware und der für die Steuerung der Hardware benötigten Software-Komponenten: Ist die reale Hardware erst spät im Verlauf eines Projekts verfügbar oder die Beschaffung aus Kostengründen nicht möglich, so kann wie bei der virtuellen Inbetriebnahme eine geeignete Simulationsumgebung für die Bereitstellung virtueller Modelle der Roboter-Hardware und deren Arbeitsumgebung sorgen.
- Bei der Erbringung von Funktionsnachweisen vor und während der Entwicklung von Roboter-Hardware und neuer Algorithmen, etwa in der autonomen Robotik oder bei direkter Interaktion mit menschlichen Anwendern: Im ersten Fall kann eine Simulationsumgebung als vereinfachte Repräsentation der späteren Hardware dienen, im zweiten Fall kann eine Simulationsumgebung als „virtueller Prototyp“ eingesetzt werden, mit der gleichen Motivation wie zuvor für die Industrie-Robotik beschrieben.

1.1. Motivation

Eine solche Simulationsumgebung hat eine Reihe von Funktionen zu erfüllen, um adäquate Ersatzmodelle für reale Roboter und deren Arbeitsumgebungen bereitstellen zu können:

1. Die Simulation der Aktorik und Sensorik eines Roboters selbst
2. Die Bereitstellung ausreichend detaillierter geometrischer Modelle von Robotern und Arbeitsumgebung
3. Die Simulation der Auswirkungen der Interaktion eines Roboters mit seiner Arbeitsumgebung

Die Kombination detaillierter geometrischer Modelle und der plausiblen Simulation mechanischen Verhaltens bei der Interaktion mit der Arbeitsumgebung ist der Gegenstand der vorliegenden Arbeit. Dabei steht die Herausforderung im Vordergrund, einen möglichst guten Kompromiss zwischen Detailgrad verwendeter Geometrien und der benötigten Rechenzeit pro Iteration der Simulation zu finden.

Mechanisch plausibles Verhalten ist eine grundlegende Eigenschaft, die bei der Nachbildung realer Arbeitsumgebungen in dreidimensionalen Computersimulationen vorausgesetzt wer-

den muss, sofern für die gegebene Problemstellung eine virtuelle Umgebung mehr als nur ein rein visuelles Werkzeug sein soll: Die elementare Eigenschaft, sich nicht gegenseitig durchdringen zu können.

Abgesehen von dieser grundlegenden physikalischen Eigenschaft spielt auch das Verhalten manipulierter Objekte unter dem Einfluss von Kräften und Drehmomente eine Rolle dabei, geometrische Modelle mit einer mechanischen Repräsentation zu versehen.

1.2. Problemstellung

Echtzeitfähige Dynamiksimulationen sind eine bereits seit beinahe drei Jahrzehnten intensiv erforschte Disziplin. Die in diesem Zeitraum unternommenen umfangreichen Entwicklungsanstrengungen haben eine große Anzahl unterschiedlicher Verfahren zur Kollisionserkennung und -behandlung hervorgebracht.

Unabhängig von speziellen Ansätzen sollen zuerst die grundlegenden Probleme bei mechanisch plausibel konzipierten Simulationsumgebungen erläutert werden. Die allgemeine Struktur der zu lösenden Teilaufgaben ergibt sich aus der zwangsläufigen Unterteilung der Gesamtaufgabe, mechanische Körper zu simulieren, in zwei Teilaufgaben.

In der klassischen Mechanik selbst spielt die tatsächliche geometrische beziehungsweise volumetrische Struktur eines Objektes für die Formulierung der Bewegungsgleichungen keine Rolle. Ausschlaggebend sind hier ausschließlich physikalische Größen und deren Ableitungen: Entfernung, Zeit, und Masse, respektive Geschwindigkeiten und Beschleunigungen und die daraus zusammengesetzten Größen wie Kräfte und Momente. Die einzige von der tatsächlichen Struktur eines räumlich ausgedehnten Objektes abhängige Größe ist der Schwerpunkt, der allerdings nur im Rahmen des Superpositionsprinzips als Ansatzpunkt für die Gesamtheit aller auf ein Objekt einwirkenden Kräfte beziehungsweise Momente dient. Auch das gegenseitige Nicht-Durchdringen von Objekten ist intuitiv erklärt als eine inhärente Eigenschaft fester Materie, die nicht separat bei der Formulierung von Modellen für die Bewegung und wechselseitiger Beeinflussung räumlich ausgedehnter Objekte bei gegenseitigem Kontakt berücksichtigt werden muss.

In der Computergrafik allerdings besteht dieses Problem durchaus: Die pure geometrische Modellierung eines dreidimensionalen Objektes in einer virtuellen Umgebung erfasst keinesfalls zwangsläufig die mechanischen Eigenschaften des betreffenden Objekts, auch garantiert sie in keiner Weise die Vermeidung gegenseitigen Durchdringens. Dreidimensionale computergraphische Modelle sind technisch bedingt mehrheitlich als polyhedrale Oberflächen (üblicherweise als Dreiecksnetze) spezifiziert und stellen daher keine volumetrischen Repräsentationen dar. Die Bestimmung von Berührungen und Überschneidungen zwischen Objektmodellen beruht daher meist auf der Schnittberechnung zwischen Paaren von Seitenflächen zweier Objekte.

Um den bei komplexen polygonalen Modellen auftretenden enormen Laufzeitaufwand begrenzen zu können, erfolgt vor der Prüfung der eigentlichen Objekt-Oberflächen üblicherweise eine vorgelagerte Schnittberechnung mit einfacher strukturierten, konvexen Hüllgeometrien, die als Approximation der eigentlichen Objekt-Oberflächen verwendet werden; da sich zwei (unter Umständen konkave) Objekte nicht berühren oder überschneiden können,

wenn dies ein Paar aus konvexen Hüllgeometrien nicht tut, lassen sich so nicht kollidierende Objektpaare aus der später folgenden detaillierten Schnittberechnung vorab ausschließen.

Betrachtet man nun den potentiell zu erwartenden Laufzeitaufwand für die beiden Teilaufgaben einer Mechaniksimulation, so ergibt sich ein enorm höherer Anteil der Kollisionserkennung an der gesamten Laufzeit eines Simulationsschrittes. Dies ist damit zu begründen, dass die nötige Laufzeit der in der Kollisionsbehandlung zum Einsatz kommenden numerischen Lösungsverfahren wesentlich präziser abgeschätzt werden kann: Die Größe des zugrunde liegenden Differentialgleichungssystems hängt von der Anzahl der simulierten Objekte und von der Anzahl der kinematischen Zwangsbedingungen ab, denen die Objekte zum Zeitpunkt der Berechnung unterliegen. Ebenso ist bei der Wahl eines geeigneten numerischen Verfahrens die Anzahl der benötigten Iterationen bis zu einer numerisch akzeptablen Konvergenz des zu lösenden Gleichungssystems eher gering.

Die in der Kollisionserkennung durchgeführten Berechnungen sind dagegen einerseits abhängig von der geometrischen Struktur der simulierten Objekte, vorrangig von der Anzahl der in der Oberflächen-Modellierung verwendeten Seitenflächen und der davon abhängenden Struktur verwendeter Hüllkörperhierarchien. Des weiteren hängt die Anzahl potentiell durchzuführender detaillierter Schnittberechnungen von der Position simulierter Objekte zueinander ab: Je dichter Objekte in einer Raumregion liegen, umso wahrscheinlicher sind Überlappungen zwischen Hüllkörpern, und umso höher fällt die Anzahl durchzuführender Schnittberechnungen zwischen Objektpaaren aus, von denen unter Umständen ein nicht unwesentlicher Anteil falsche Positive darstellt.

Das größte Potential zur Verringerung der benötigten Laufzeit je Simulationsschritt lässt sich also durch die Optimierung der Kollisionserkennung erreichen. Diese Tatsache spiegelt sich auch im Verhältnis der zu echtzeitfähiger Dynamiksimulation verfügbaren Veröffentlichungen und Verfahren wider: Numerische Methoden zur Kollisionsbehandlung machen im Vergleich zu Verfahren zur Kollisionserkennung nur den kleineren Teil der Gesamtheit der durchgeführten Entwicklungsvorhaben aus.

Nicht für alle Anwendungsfälle ist ein strenge Echtzeitfähigkeit nötig; oft ist es ausreichend, die zur Kollisionserkennung benötigte Laufzeit zur Verfügung zu stellen, ohne die Aussagekraft des simulierten Problems in schwerwiegender Weise zu verringern, oder es werden Zwischenergebnisse aus frühen Phasen der Kollisionserkennung verwendet, die nicht auf den tatsächlichen Geometrien, sondern auf einfacheren Hüllkörpern basieren: Ein Beispiel hierfür wäre etwa die Simulation eines Kopplungsvorgangs zwischen individuellen Robotern des Symbrion-Replicator-Projekts: Um in einer Simulation einen Andockvorgang plausibel darstellen zu können, ist es für konzeptionelle Versuche ausreichend, zwei oder mehr Roboter zu einer Verbund-Geometrie zu vereinen, wenn sich die einzelnen Roboter ausreichend nahe gekommen sind, als dass in einer realen Situation ein erfolgreiches Andocken geschehen könnte.

Für Simulations-Szenarien, in denen jedoch die Verwendung präziser Modelle realer Objekte zwingend notwendig ist, um in Realzeit ablaufende Prozesse mit ausreichender Realitätsnähe nachstellen zu können, sind Vereinfachungen in der geometrischen Struktur von Objekten oder die Vernachlässigung von Laufzeitanforderungen keine Alternativen. Man denke etwa an das in Abschnitt 2.1 vorgestellte Montage-Szenario: Hier ist es nicht gangbar, eine ausreichende Annäherung zwischen Werkstück und Schraube als Erfolgskriterium zur Beurteilung

des simulierten Prozesses heranzuziehen. Die zugrundeliegende Kontaktsimulation muss idealerweise robust und performant genug sein, um den umschließenden Kontakt zwischen einer Schraube in ihrer Zielposition und den verschraubten Werkstücken darstellen zu können, ohne Einschränkungen bei der Plausibilität oder der Laufzeit der Simulation in Kauf nehmen zu müssen.

1.3. Ziele der Arbeit

Die in Abschnitt 2.1 bis Abschnitt 2.3 genannten Anwendungsbeispiele verdeutlichen Anwendungen für Starrkörpersimulationen, die Gegenstand dieser Arbeit sein sollen. Ein wesentliches Ziel dieser Arbeit ist es, eine möglichst guten Kompromiss für die schwierigsten Herausforderungen in echtzeitfähigen Mechaniksimulationen zu finden (Abbildung 1.1):

1. Dem Laufzeitbedarf
2. Der präzisen Erkennung der zwischen dreidimensionalen Geometrien auftretenden Kontakte
3. Der geometrischen Komplexität der zur Kollisionserkennung verwendeten geometrischen Modelle

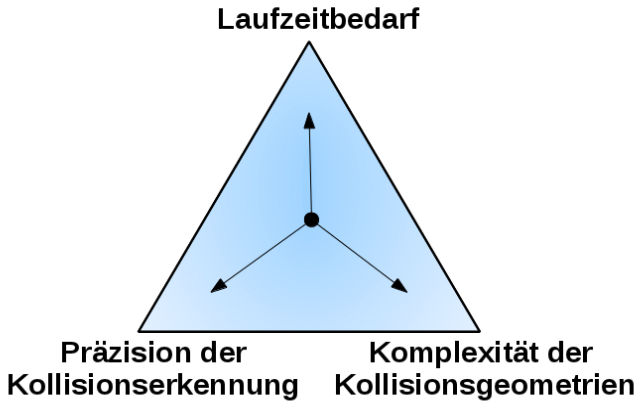
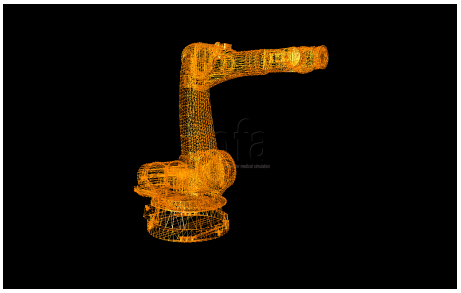


Abbildung 1.1.: Konkurrierende Größen bei der Kollisionserkennung für dreidimensionale Geometrien

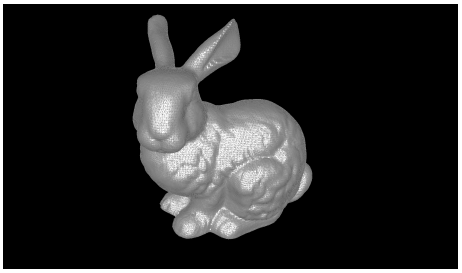
Spezifischer formuliert soll versucht werden, durch die Adaption existierender Verfahren zur Kollisionserkennung die auf modernen Grafik-Prozessoren (GPUs) zur Verfügung stehende Rechenleistung ausnutzen zu können, ohne die oftmals vorhandenen Nachteile der angesprochenen Verfahren (auf die in Kapitel 4 näher eingegangen wird) in Kauf nehmen zu müssen.

Konkret bedeutet dies, unter Voraussetzung der Fähigkeit zum Betrieb in einem interaktiv bedienbaren Bereich (25 Iterationen pro Sekunde oder mehr) eine möglichst komplex strukturierte Umgebung auf Kollisionen berechnen zu können. Als Maß für die Komplexität einer virtuellen Umgebung wird üblicherweise die Anzahl der elementaren Geometrie-Elemente der simulierten Objekte verwendet.

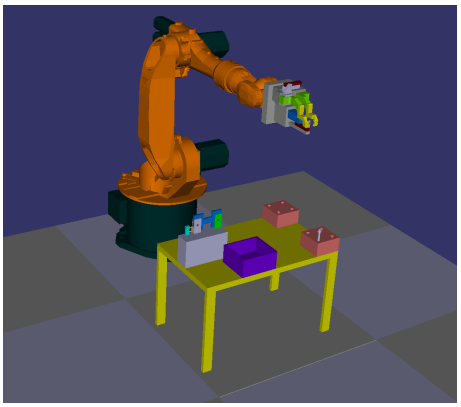
Konkrete Beispiele für die Anzahl verwendeter Geometrie-Elemente für den Aufbau von Geometrien in Starrkörpersimulationen sind in Abbildung 1.2 zu sehen:



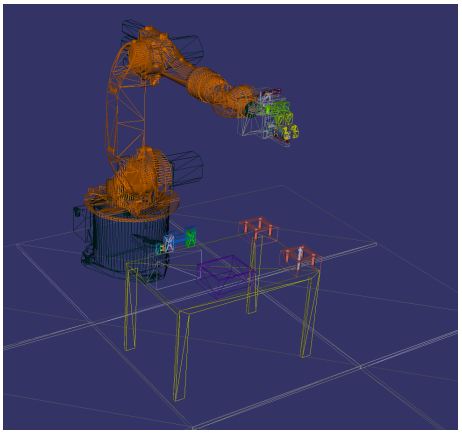
(a) Polygon-Modell eines Kuka KR-180 Roboterarmes



(b) Polygon-Modell des Stanford-Bunny



(c) Polygon-Modell eines Versuchsaufbaus der Projektstudie aus Abschnitt 2.1



(d) Polygon-Modell des Versuchsaufbaus aus Abbildung 1.2c

Abbildung 1.2.: Beispiele für Polygon-Modelle unterschiedlicher Komplexität. Die in Abbildung 1.2a, Abbildung 1.2c und Abbildung 1.2d dargestellten Szenen sind während der in Abschnitt 2.1 durchgeführten Projektstudie verwendet worden; Abbildung 1.2b ist ein häufig verwendetes Referenzmodell für den Test von Kollisionserkennungsalgorithmen.

- Das Modell eines Sechs-Achs-Industrieroboters (Abbildung 1.2a): Es umfasst 10600 Punkte und 21448 Seitenflächen.
- Ein Versuchsaufbau aus der in Abschnitt 2.1 beschriebenen Projektstudie (Abbildung 1.2c und Abbildung 1.2d): Es umfasst 22528 Punkte und 17352 Seitenflächen.
- Das Stanford-Bunny (Abbildung 1.2b): Es umfasst 34835 Punkte und 69666 Seitenflächen.

Die angestrebte Leistungsfähigkeit des im späteren Verlauf der Arbeit entworfenen eigenen Kollisionserkennungs-Algorithmus erstreckt sich auf die Behandlung von Objekten, deren geometrische Komplexität sich in der Dimension der in Abbildung 1.2a und Abbildung 1.2c gezeigten Objekte: Einem Bereich zwischen 10000 und 30000 Seitenflächen.

Ein weiteres Ziel soll die explizite Unterstützung von Szenen mit mehreren sich bewegenden Objekten sein, im Gegensatz zur Funktionsweise einiger existierender Algorithmen, die sich auf die Simulation eines einzelnen oder einer geringen Anzahl aktiv durch Anwender kontrollierter Objekte in einer ansonsten statischen Umgebung konzentrieren.

Eine andere, die Präzision der Kollisionserkennung betreffende Vorgabe soll sein, daß die entwickelten Algorithmen möglichst ohne Einschränkungen der geometrischen Präzision gegenüber den ursprünglichen verwendeten Geometrien funktionieren. Einige existierende Verfahren arbeiten nicht unmittelbar mit den polygonalen Strukturen der Original-Objekte, sondern verwenden zur Optimierung vereinfachte Approximationen, die zwar keine wesentlichen Abweichungen von der Struktur der Originale aufweisen, in bestimmten Situationen aber dennoch zu sichtbaren Unterschieden bei der Erkennung von Berührungen zwischen Objekten führen können. Im Gegensatz dazu soll der eigene Ansatz so konzipiert sein, dass keine derartigen Fehler auftreten.

Im Vordergrund soll dabei der Umgang mit starren Körpern stehen. Obwohl verformbare Geometrien eine wichtige Rolle in vielen Anwendungsgebiete spielen (etwa im medizinischen Bereich), ist die Unterstützung elastischer Körper in der Robotik und im ingenieurstechnischen Bereich oft nicht notwendig, um vorkommende Problemstellungen unter Verwendung von echtzeitfähigen Starrkörpersimulationen zu untersuchen.

1.4. Struktur der Arbeit

Der weitere Verlauf der Arbeit gliedert sich wie folgt:

- Kapitel 2 stellt zwei Anwendungsfälle für Starrkörpersimulationen aus der Industrie-Robotik und der Robotik-Forschung vor und formuliert anhand der in diesen Anwendungen gemachten Beobachtungen und Erfahrungen Kriterien, die Starrkörpersimulationen für einen sinnvollen und nutzenbringenden Einsatz erfüllen müssen.
- Kapitel 3 beschäftigt sich in allgemeiner Weise mit der Funktionsweise von Starrkörpersimulationen und versucht, einen Überblick über die theoretischen und technischen Gegebenheiten der beiden Hauptaufgabenbereiche von Starrkörpersimulationen, der Kollisionserkennung und der Kollisionsbehandlung, zu geben.
- Kapitel 4 analysiert ausführlich unterschiedliche Varianten von Kollisionserkennungs-Verfahren für die Objektpaar-Phase und versucht, deren jeweilige Stärken und Schwächen aufzuzeigen. Ein besonderer Aspekt bei der Analyse dieser Verfahren ist deren potentielle Eignung für den Einsatz auf Grafik-Prozessoren.
- Kapitel 5 stellt ausgehend von den Erkenntnissen der Analysen aus Kapitel 4 einen eigenen Ansatz zur Kollisionserkennung in der Objektpaar-Phase vor. Dabei soll durch eine Segmentierung von Geometrien und den Einsatz von Punktwolken in Kombination mit Raumpartitionierungen ein Leistungsvorteil gegenüber existierenden Verfahren erreicht werden, die primär auf der Verwendung von Hüllkörper-Hierarchien operieren.
- Kapitel 6 validiert den in Kapitel 5 vorgestellten Ansatz anhand von praktischen Experimenten.
- Kapitel 7 skizziert mögliche Erweiterungen des eigenen Ansatzes, etwa hinsichtlich der Unterstützung elastischer Geometrien oder der Berücksichtigung von Objektbewegungen bei der Kollisionserkennung.

Des weiteren beinhaltet die Arbeit drei Anhänge:

- Anhang A stellt als Ergänzung zu den in Kapitel Kapitel 3 nur kurz angesprochenen Verfahren zur Kollisionsbehandlung diese in ausführlicher Form vor.
- Anhang B beschäftigt sich mit verschiedenen Möglichkeiten zur Repräsentation von Geometrien in Kollisionserkennungs-Verfahren.
- Anhang C erläutert anhand eines Beispiels die Funktionsweise und Besonderheiten von GPU-Prozessoren und deren Programmierung.

KAPITEL 2

STAND DER TECHNIK: STARRKÖRPER-SIMULATIONEN IN DER ROBOTIK

Wie Starrkörper-Simulationen im Kontext der Robotik eingesetzt werden, und welche konzeptionellen sowie technischen Herausforderungen und Besonderheiten dabei zu berücksichtigen und zu meistern sind, soll in den anschließenden Abschnitten Abschnitt 2.1 und Abschnitt 2.2 anhand von Beispielen erläutert werden, die der vorliegenden Arbeit vorangegangen sind, oder mit denen der Autor durch Mitarbeit an Projekten indirekt zu tun hatte:

1. Als Beispiel aus der Industrie-Robotik dient eine Projektstudie aus dem Bereich der virtuellen Inbetriebnahme, die einen Machbarkeitsnachweis für die Unterstützung der Entwicklung und der Erprobung von Steuerungsprogrammen für Industrie-Roboter in der Automobilindustrie unter Verwendung virtueller Modelle von Werkstücken, Werkzeugen und Arbeitsumgebung gekoppelt mit einer realen Robotersteuerung erbringen sollte.
2. Als Beispiel aus der Schwarm-Robotik wird unter anderem die Simulationsumgebung Robot3D, die im Rahmen der EU-Projekte Symbrion und Replicator entwickelt worden ist, näher beschrieben. Obwohl im Gegensatz zur eben genannten Projektstudie die mechanisch plausible Simulation einer Arbeitsumgebung nicht die primäre Motivation bei der Entwicklung dieses Simulationssystems war, so ist Robot3D trotzdem ein anschauliches Beispiel für den Einsatz einer echtzeitfähigen Starrkörpersimulation im Bereich der Robotik-Forschung.

Anhand dieser Anwendungen sollen auch exemplarisch zwei Aspekte diskutiert werden, die zwar im weiteren Verlauf der Arbeit keine zentrale Stellung mehr einnehmen werden, jedoch im Zusammenhang mit Simulationen in der Robotik essentielle Bestandteile sind: Die Simulation von Sensorik und Aktorik-Komponenten.

2.1. Anwendungsbeispiel in der Industrie-Robotik: Projektstudie zur virtuell unterstützten Steuerungsprogramm-Entwicklung

Diese im Auftrag der Daimler AG durchgeführte Projektstudie ([Ros11]) hatte das Ziel, das Entwickeln und Testen von Steuerungsprogrammen für Industrieroboter im Rahmen der Entwicklung neuer Fertigungslinien zu vereinfachen. Ein wesentlicher Vorteil des Einsatzes einer simulierten Arbeitsumgebung besteht darin, dass mit der Entwicklung der benötigten Steuerungsprogramme bereits begonnen werden kann, bevor mit dem Aufbau der realen Hardware begonnen wird. Beim derzeitigen Stand der Technik ist zur Erstellung der Steuerungsprogramme für eingesetzte Roboter und deren Peripherie der vollständige Aufbau der realen Fertigungsanlage eine notwendige Voraussetzung, so dass mit der eigentlichen Programmierung erst kurz vor der tatsächlichen Inbetriebnahme begonnen werden kann, mit allen negativen Konsequenzen für die Qualität und Reife der entstehenden Software. Eine Verlagerung wesentlicher Teilen der Entwicklung der benötigten Steuerungsprogramme in frühere Entwurfsphasen würde nicht nur diese soeben angesprochenen Probleme entschärfen. Des weiteren wäre es beteiligten Programmierern zusätzlich möglich, noch auf den Entwurf der Hardware Einfluss nehmen zu können, anstatt nach beendeter Herstellung der eingesetzten Gerätschaften in durch den Entwicklungsprozess bedingten engen Projektzeitplänen etwaige Entwurfsfehler oder -mängel durch aufwendiger gestaltete Steuerungsprogramme ausgleichen zu müssen.

2.1.1. Die Montageaufgabe

Besondere Priorität hatte die Entwicklung und Absicherung von Montageabläufen, deren tatsächlicher Ablauf anhand der angegebenen Bahnbefehle beziehungsweise Aktor-Zustände eines Roboters nicht zum Zeitpunkt der Erstellung des Programmcodes bestimmbar ist. Ein Beispiel für einen solchen Montageablauf das Verschrauben zweier Bauteile, wobei die Positionen der Bauteile zueinander bestimmten Toleranzen unterworfen sind: Um diesen Montagevorgang erfolgreich durchführen zu können, ist es unter Umständen nötig, mehrere Versuche zu unternehmen, um Schrauben korrekt in Gewindeöffnungen zu verbringen. Dazu benötigt das steuernde Programm eine Rückmeldung darüber, ob ein Steck- beziehungsweise Schraubvorgang erfolgreich war; dies kann beispielsweise über eine taktile Rückkopplung erfolgen. Dieser exemplarische Versuchsaufbau ist in Abbildung 2.1 illustriert.

Hierzu ist das die Schrauben manipulierende Werkzeug mit einem Wegmess-System versehen, und die gerade gegriffene Schraube ist nicht unbeweglich im Werkzeug fixiert, sondern hat die Möglichkeit, entgegen der Richtung des Steckvorgangs zurückzuweichen. Verfehlt ein Steckversuch nun die Gewindeöffnung, so führt das mechanisch bedingte Zurückweichen der Schraube dazu, dass das verwendete Wegmesssystem einen von Null abweichenden Wert misst, der einen Blockadezustand anzeigt. Bevor nun ein manueller Eingriff notwendig ist, besteht die Möglichkeit, ein Suchstrategie zu verwenden, die in der Umgebung des ersten Montageversuches weitere Montageversuche unternimmt: Hierzu bieten sich unterschiedliche Suchmuster an, die das Werkzeug mitsamt Schraube in kleinen Inkrementen relativ zu den

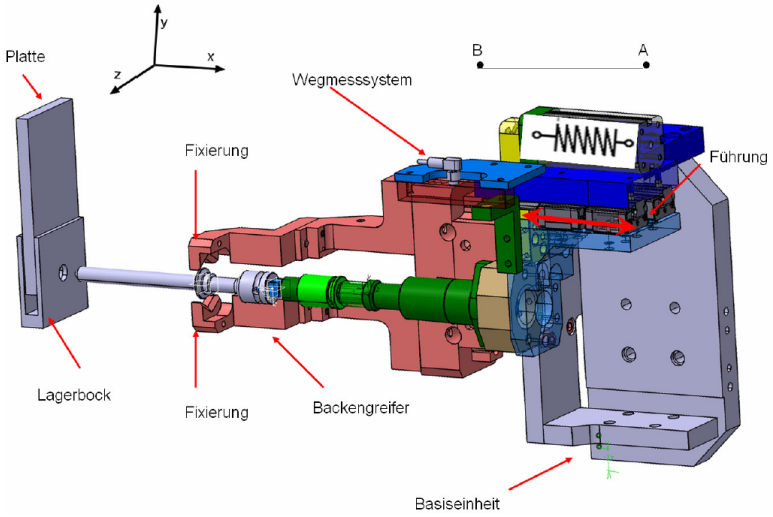


Abbildung 2.1.: Exemplarischer Versuchsaufbau für das Montieren einer Schraube (nach [Ros11]): Das Greifer-Werkzeug ist so konstruiert, dass die gegriffene Schraube nicht fest fixiert ist. Bei Auftreten einer Kollision mit Lagerbock kann die Schraube zurückweichen; eine Ausweichbewegung wird von einem Wegmesssystem erfasst, was dem Steuerungsprogramm einen fehlgeschlagenen Steckversuch signalisiert. In diesem Fall kann mittels verschiedener Suchstrategien eine leicht variierte neue Montageposition bestimmt und der Montageversuch wiederholt werden.

Werkstücken neu positionieren (Abbildung 2.2). Erst wenn die Wiederholung des Montageversuches mehrfach fehlschlägt, bricht der Montagevorgang als Ganzes ab.

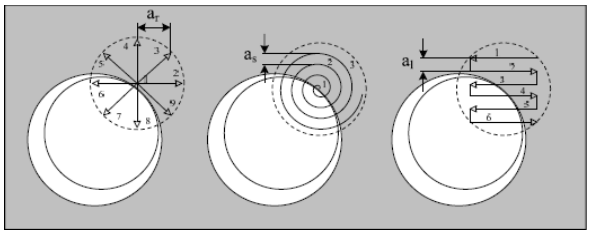


Abbildung 2.2.: Unterschiedliche Suchmuster zum Ausgleich von Toleranzen bei der Montage einer Schraube (nach [Ros11])

2.1.2. Versuchsaufbau für den automatisierten Test von Steuerungsprogrammen

Die Mechaniksimulation liefert je nach Verlauf des gewählten Suchmusters durch einen erneuten Kontakt mit dem Lagerbock eine Veränderung der gemessenen Distanz im Wegmess-System als Ergebnis eines erneut erfolglosen Steckversuchs, oder indiziert den Erfolgsfall durch Ausbleiben eines Kontakts mit der die Gewindeöffnung umgebenden Oberfläche.

Als Teil der Testumgebung fungieren *virtuelle Sensoren* (Abbildung 2.3): Bei Berührung oder Überlappung mit anderen Objekten aus der Simulation wird ein entsprechendes Kollisionsereignis ausgelöst, aber es kommt nicht zu Änderungen im Bewegungsablauf anderer Objekte.

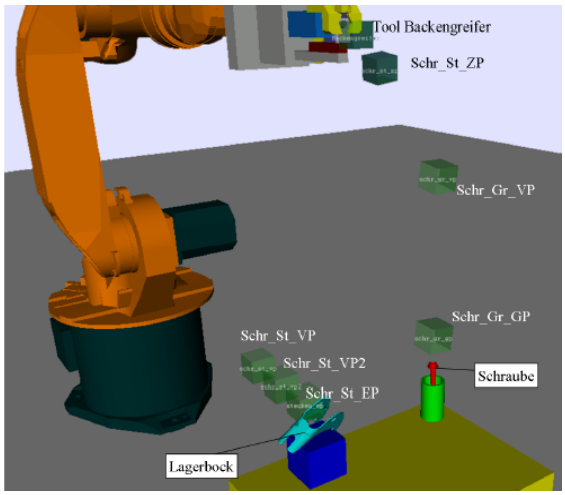


Abbildung 2.3.: Virtuelle Sensoren (grün gefärbte, transparente Quader) in der Simulationsumgebung (nach [Ros11]). Diese dienen zur Überwachung von Teilregionen der simulierten Umgebung zur Registrierung von zuvor spezifizierten Ereignissen.

Die durch diese virtuellen Sensoren ausgelösten Kollisionsereignisse sowie anderweitig auftretende, unerwünschte Kollisionsereignisse (etwa des Roboterarms mit Werkstücken) werden nun durch ein parallel zur eigentlichen Simulation betriebenes Software-Modul zum teilautomatisierten Test ausgewertet, das das Eintreten beziehungsweise Ausbleiben von Kollisionen zwischen Gruppen von Objekten in der laufenden Simulation als Kriterien für Erfolg oder Misserfolg eines simulierten Prozessablaufs enthält und je nach eintretenden Kollisionsereignissen einen Testlauf als korrekt oder fehlerhaft bewertet.

Neben der Überwachung von Kollisionsereignissen sind weitere Ereignisse, wie etwa das Erzeugen eines bestimmten Statussignals eines an die Simulation angebundenen Steuergeräts, das Eintreten eines bestimmter Messwerte bei simulierten Sensoren, oder das Erreichen einer bestimmten Position durch überwachte Objekte als Kriterien zur Beurteilung von Erfolg oder Misserfolg eines Testdurchlaufes denkbar. Durch die Möglichkeit, solche Überwachungs-Bausteine flexibel zu kombinieren, ist es möglich, auch komplexe Testbedingungen zu formulieren und damit eine große Bandbreite von Montagevorgängen zu behandeln.

2.1.3. Die Simulation von Aktorik-Komponenten

Für die Realisierung der zuvor beschriebenen Montageaufgabe war die Simulation zweier verschiedener Formen von kinematischen Strukturen erforderlich:

- Die Übertragung der durch eine reale Roboter-Steuerung bestimmten Vorwärts-Kinematik auf das Modell eines Sechs-Achs-Roboters in der Simulation
- Das Nachvollziehen des Bewegungsverhaltens der Komponenten eines am Sechs-Achs-Roboter angebrachten Greifer-Werkzeugs: Zwei Greiferbacken (Abbildung 2.4) und ein pneumatisch getriebener Stößel (dunkelgrün gefärbte Komponente in Abbildung 2.1)

Aufgrund technischer Restriktionen in der Umsetzung kinematischer Zwangsbedingungen sind einige Besonderheiten zu beachten: Beispielsweise können gängige Starrkörpersimulations-Lösungen Zwangsbedingungen nur direkt zwischen Paaren von Kör-

pern unterstützen (siehe Unterabschnitt A.2.5), nicht aber über eine komplexere kinematische Struktur hinweg. Jedoch beschränken sich die Anforderungen an eine Starrkörpersimulation bei solchen kinematischen Strukturen auf das Nachvollziehen einer extern (durch eine reale Steuerung oder einen software-basierten Controller) spezifizierten Vorwärts-Kinematik. Eine Starrkörpersimulation muss in solchen Fällen gewährleisten können, dass einzelne Segmente einer kinematischen Struktur eine per Vorwärts-Kinematik festgelegte Zielposition zuverlässig erreichen oder einhalten. Für diese Aufgabe existieren Algorithmen ([Fea08]), die für kinematische Strukturen unterschiedlicher Komplexität (zyklisch, verzweigt) unter expliziter Berücksichtigung der Dynamik aller Elemente einer Struktur in der Lage sind, eine stabile Lagekontrolle für kinematische Strukturen sicherzustellen.

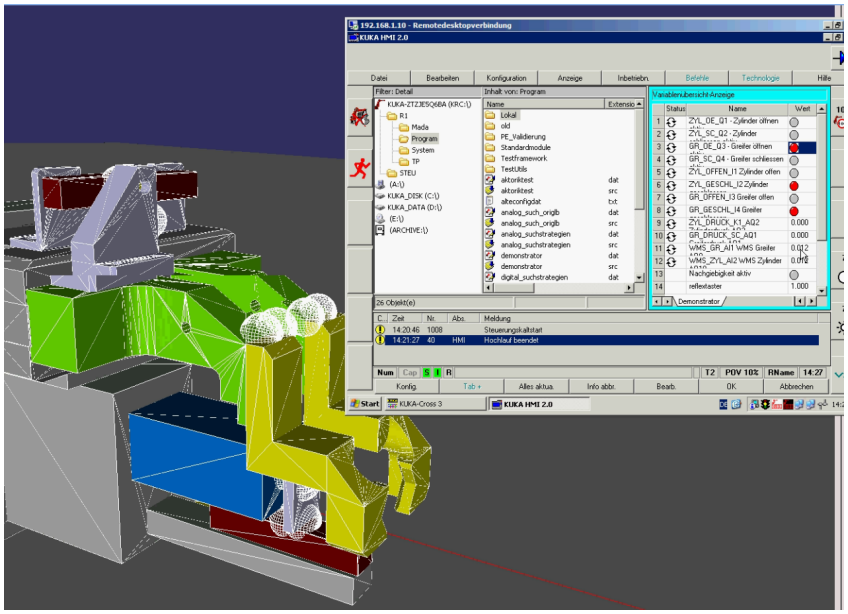


Abbildung 2.4.: Modell eines Greifer-Werkzeugs mit insgesamt drei Freiheitsgraden (zwei Greiferbacken und ein pneumatischer Stößel). Die Steuerung der Bewegung erfolgt über eine externe Robotersteuerung; die Kontrolle der kinematischen Zwangsbedingungen obliegt der Starrkörpersimulationsumgebung.

2.1.4. Die Simulation von Sensorik-Komponenten

Die zu simulierende Sensorik bei der beschriebenen Montageaufgabe bestand aus zwei Typen von Komponenten:

- Aus Objekteigenschaften ableitbare Sensorwerte: Im verwendeten Werkzeug verbaute Wegmesssysteme wurden durch die Bestimmung der relativen Position zweier Kollisionsgeometrien zueinander simuliert.
- Unter Verwendung der Kollisionserkennungs-Komponente ermittelte Sensorwerte: Ein Reflextaster für die Entfernungsmessung wurde durch die Schnittberechnung eines Scan-Strahls mit Kollisionsgeometrien in der Szene simuliert.

Bei der Nutzung von Eigenschaften simulierter Objekte zur Bestimmung von Sensorwerten ist innerhalb einer Starrkörpersimulation selbst kein zusätzlicher Berechnungsaufwand nötig.

Für die Simulation von Sensoren, die auf Basis optischer oder akustischer Signale funktionieren, ist eine vereinfachte Modellierung als Geradenabschnitt (Abbildung 2.5) oder einer entsprechend der Reichweite und Form des Emissionsbereichs strukturierten Kollisionsgeometrie möglich. Diese geometrischen Primitive können dann in gleicher Weise wie die geometrischen Repräsentationen simulierter Objekte im Rahmen der Kollisionserkennung auf mögliche Kontakte überprüft werden. Dadurch erhöht sich wiederum der Berechnungsaufwand für die Kollisionserkennung; vor allem dann, wenn Schnitt-Tests auf Basis einzelner Seitenflächen erforderlich sind (zum Beispiel für die Bestimmung der exakten Distanz zwischen Sensor und detektiertem Objekt), benötigt die Simulation eines Sensors ähnlichen Laufzeitaufwand wie die vollständige Kollisions-Prüfung zweier simulierter Objekte.

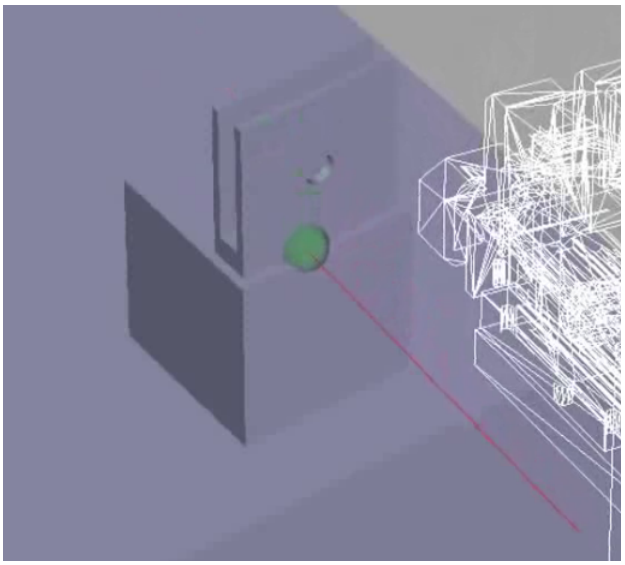


Abbildung 2.5.: Modell eines Reflextasters: Der Abtaststrahl des Sensors ist als Geradenabschnitt realisiert, der genau wie andere Kollisionsgeometrien als Teil der Kollisionserkennung auf Kontakte mit der simulierten Umgebung berechnet wird.

2.2. Anwendungsbeispiel in der Schwarm-Robotik

2.2.1. Robot3D

Die Projekte "Symbrion" und "Replicator" beschäftigten im Rahmen des EU-finanzierten "Seventh Framework Programm for Research and Technological development" (FP7, [Eur07]) mit Anwendungen in der Schwarmrobotik. Das Symbrion-Projekt setzte seine Schwerpunkte hierbei auf evolutionäre und biologisch inspirierte Konzepte zur Steigerung der Anpassungs- und Überlebensfähigkeit beteiligter Roboter respektive höher organisierter Roboter-Organismen geprägt durch Anpassungen an die jeweilige Arbeitsumgebung. Das Replicator-Projekt hingegen beschäftigte sich mit der der Entwicklung von Roboterplattformen, die sowohl als individuelle Bausteine agieren als auch zur Selbst-Assemblierung zu komplexer organisierten Organismen in der Lage sein sollten; im Gegensatz zu Symbrion standen hierbei nicht die theoretischen Möglichkeiten der entwickelten Plattform, sondern

die tatsächliche autonome Langzeit-Überlebensfähigkeit in einer unbekannten Arbeitsumgebung im Vordergrund. Beide Projekte hatten das Problem, dass die verschiedenen Varianten der eingesetzten Roboter-Plattformen erst im Lauf des Projektes konstruiert und in größerer Stückzahl verfügbar waren. Der Einsatz von simulierten Modellen zur Entwicklung von Controllern und zur Validierung evolvierter Organismen war also eine wesentliche Voraussetzung für die erfolgreiche Umsetzung der beiden Projekte. Während beispielsweise Fragestellungen wie die biologisch inspirierte Entwicklung von Organismen sicherlich keine vollwertige Starrkörpersimulation erfordern (hierbei stehen keine mechanischen Interaktionen im Fokus der Fragestellung), so erforderten jedoch andere Teilaspekte der zu leistenden Entwicklung, beispielsweise die Entwicklung von Controllern für makroskopische Fortbewegung im "Organismus-Modus" oder die Wahrnehmung der Arbeitsumgebung durch die auf den individuellen Robotern verbauten Sensoren ein mechanisch plausibel simuliertes Modell, um ohne verfügbare Hardware die konzeptionelle Funktionsfähigkeit der implementierten Verfahren absichern zu können.

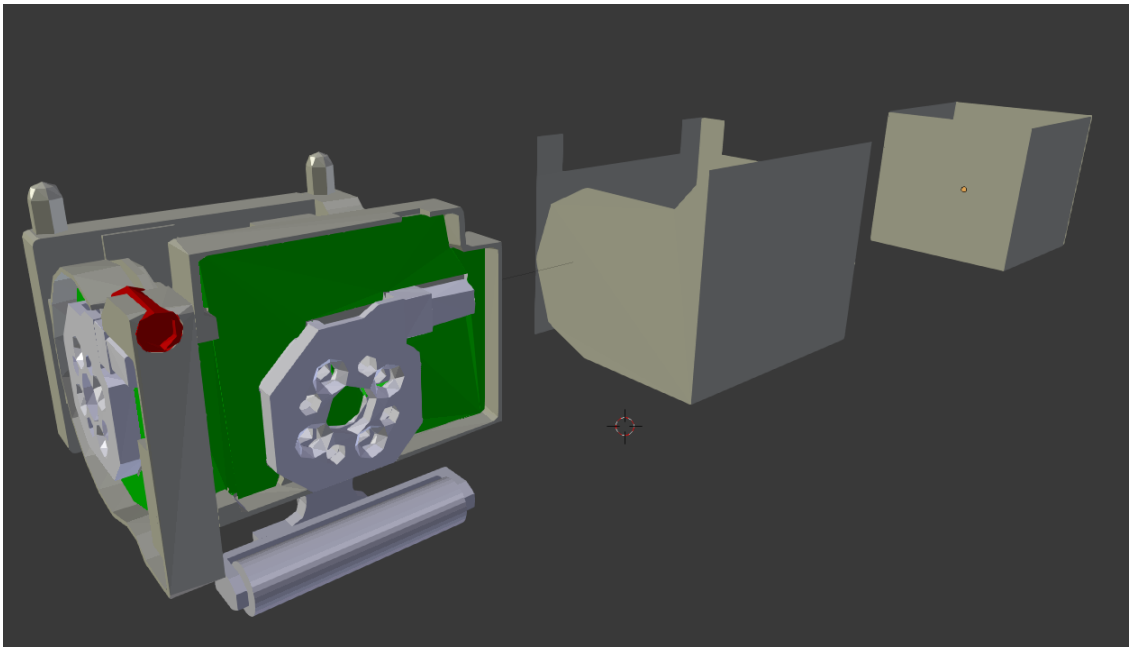


Abbildung 2.6.: Unterschied zwischen visuellem Modell und Kollisions-Geometrien: Die gezeigten Geometrien veranschaulichen den häufig bestehenden Unterschied zwischen für die Visualisierung und die Kollisionserkennung verwendeten Geometrien sehr gut. Die Geometrie links zeigt ein CAD-konstruiertes Modell des K-Bot, die mittlere Geometrie erfasst noch einige strukturelle Details der Andock-Elemente; die rechte Geometrie ist die tatsächlich von Robot3D verwendete Kollisionsgeometrie.

Hierzu existierte eine speziell für die beiden Projekte entwickelte Simulationsumgebung, das Robot3D-Projekt ([Win11], [WW10]). Ein wesentliches, wenn auch für Robot3D nicht essentielles Element dieser Simulationsumgebung ist das Open-Source-Projekt ODE ([Smi07]), welches eine echtzeitfähige Starrkörpersimulationsumgebung realisiert. Während ODE nach Aussage der Entwickler für die hochperformante Simulation von Starrkörperdynamik ausgelegt ist und unter anderem in echtzeitnahen Anwendungen zum Einsatz kommt, bestehen andererseits enorme Nachteile bei der Behandlung komplexer Geometrien und Simulationen mit zahlreichen Objekten, was die Plausibilität des mechanischen Verhaltens von Objekten und das Laufzeitverhalten von ODE betrifft.

Diese Defizite können in gewissem Maß durch Vereinfachungen der Modellierung der simulierten Umgebung und Roboter, etwa durch die Verwendung approximierender Geometrien als Modelle für Objekte und Roboter in der Simulation, ausgeglichen werden: Abbildung 2.6 zeigt diese Möglichkeit anhand des Modells eines K-Bot, einer im Rahmen des Replicator-Projekts entwickelten Roboter-Plattformen. Links in der Abbildung ist das zur Visualisierung in Robot3D verwendete Modell dargestellt, das mittlere und rechte Modell sind zwei vereinfachte Kollisions-Geometrien, die separat für die Verwendung mit ODE konstruiert worden sind. Bei der Entwicklung von Robot3D wurde versucht, diesem Problem durch die Entwicklung einer eigenen Kollisionserkennungs-Bibliothek (libccd, [Fis10]) zu begegnen, die sich allerdings explizit auf die Unterstützung konvexer Kollisionsgeometrien beschränkt.

2.2.2. Die Simulation von Aktorik-Komponenten

Im Unterschied zum in Abschnitt 2.1 beschriebenen Szenario war bei Robot3D die Unterstützung mobiler Roboter erforderlich: So musste eine geeignete Abstraktion für die verschiedenen Antriebs-Varianten der im Rahmen von Replicator entwickelten Roboter-Plattformen (Schrauben-, Ketten- und omnidirektionaler Rad-Antrieb) gefunden werden.

Für die Simulation von aus einzelnen Modulen zusammengesetzten Organismen wurden die integrierten Gelenktypen der ODE-Engine verwendet: Aufgrund der bereits zuvor erwähnten Beschränkung der Funktionsweise dieser Gelenktypen auf einzelne Paare verbundener Starrkörper wurden nur solche Verbindungen in der kinematischen Struktur eines Organismus aktiv simuliert, die keine redundanten Freiheitsgrade darstellten; alle anderen Kopplungen wurden vorgenommen, indem die verbundenen Module in der Simulation zu einem neuen Komposit-Objekt zusammengefasst wurden, das sowohl in der Kollisionserkennung als auch der -behandlung als ein einziges Objekt verwaltet wurde.

2.2.2.1. Skelett-Muskel-Systeme

Eine komplexere Art kinematischer Systeme stellen Skelett-Muskel-Systeme dar: Hier wird die Konfiguration einer kinematischen Struktur nicht durch eine vorgegebene Vorwärts-Kinematik spezifiziert, sondern durch mit einzelnen Segmenten (als Analogie für die Knochen eines Skeletts) verbundene Muskel-/Sehnen-Systeme, die durch Kontraktion und Expansion entsprechend Zug- oder Druck-Kräfte auf die zugeordneten Segmente ausüben und damit die Konfiguration einer kinematischen Struktur beeinflussen.

Eine Umsetzung eines solchen Skelett-Muskel-Systems ist in der Simulationsumgebung CALIPER ([WJD+11], [WAB+13]) basierend auf der Physik-Engine Bullet zu finden, die als Teil des ECCERobot-Projekts ([Ecc11]) entwickelt worden ist.

2.2.3. Die Simulation von Sensorik-Komponenten

Neben Sensorik, die über die in Unterabschnitt 2.1.4 skizzierten Modelle unterstützt werden können, war für Robot3D die Integration weiterer Sensoren vorgesehen, die sich nicht mit Hilfe der durch eine Starrkörpersimulation zur Verfügung gestellten Daten und Funktionen umsetzen ließen:

- Die Erzeugung von Kamera-Bildern konnte durch das Rendern der Darstellung der simulierten Szene aus der Sicht der auf Robotern angebrachten Kameras umgesetzt werden.
- Akustische Wahrnehmungen könnten ähnlich dem Konzept virtueller Hilfs-Sensoren über Kollisions-Geometrien in zwei verschiedenen Rollen abstrahiert werden: Einerseits Objekte, die als Geräuschquellen fungieren, und andererseits Objekte, die den Wahrnehmungsbereich eines Mikrofons erfassen. Beim Auftreten eines Kontakts zwischen Geräuschquelle und Mikrofon kann ausgehend von der Distanz der Objekte ein wahrgenommenes „Geräusch“ als erkannt angenommen werden.
- Ähnlich dieser Modellierung können für Infrarot-Sensoren Wärmequellen mittels Kollisions-Geometrien abstrahiert werden, und der Messwert proportional zum Abstand beziehungsweise zum Schnittvolumen zwischen der Geometrie der Wärmequelle und dem Meßbereich des Sensors bestimmt werden.

2.3. Weitere Anwendungsgebiete für Starrkörpersimulationen

Neben den in Abschnitt 2.1 und Unterabschnitt 2.2.1 angesprochenen Projekten existieren noch weitere Anwendungsgebiete, in denen Starrkörpersimulationen Anwendung finden, und die hier nur kurz angesprochen werden sollen.

2.3.1. Zugänglichkeits-/Assemblierungs-Simulationen

Dieses produktionstechnische Anwendungsfeld, zu dem auch das in Abschnitt 2.1 vorgestellte Industrieprojekt gezählt werden kann, nutzt Starrkörpersimulationen im weitesten Sinn dazu, Montageprozesse in der Produktion oder den Ein- und Ausbau von Einzelteilen in Wartungsvorgängen an CAD-Modellen zu validieren. Ziel hierbei ist es, bereits in der Entwurfsphase die Prozessplanung für die spätere Umsetzung zu unterstützen, um mit weniger Tests mit realen Prototypen auskommen zu können.

Anwendungsbeispiele finden sich etwa in [MPT99], wobei ein Wartungsvorgang am Fahrwerk eines Passagierflugzeugs als Anwendungsbeispiel genannt ist, oder in [Buc99] und [Eck98]; hier werden der Einbau eines Autoradios und das Auswechseln einer Scheinwerferlampe im Frontlicht eines Fahrzeugs als Beispiele angeführt. Auch die Anwendungsbeispiele aus [EMSW13], die Gegenstand von Unterabschnitt 4.3.3 sein werden, sind diesem Anwendungsfeld zuzuordnen.

2.3.2. Robotik und virtuelles Training

Eine weitere Anwendung für mechanisch plausible Simulationen ist das sogenannte virtuelle Training: Hier werden virtuelle Szenarien zur Unterstützung der Aus- und Weiterbildung von menschlichen Anwendern genutzt. Typische Anwendungen sind etwa Fahrzeug-Simulatoren,

wie etwa mit Hilfe der Vortex Engine ([CML13b]) umgesetzte Simulatoren für Kräne und schwere Fahrzeuge ([CML13a]).

Daneben existieren Simulationsumgebungen für Industrie- und Service-Robotik, die sowohl im kommerziellen Bereich als auch im Forschungs- und Bildungssektor zum Einsatz kommen: Beispielsweise das Robotics Developer Studio von Microsoft [Mic12b], Gazebo ([Koe12]) als Teil des ROS-Projektes (Robot Operating System, [Ger12b]) und des Player/Stage-Frameworks ([Ger12a]), WeBots ([Mic12a]), Kuka SimPro ([Kuk12]), die Produktreihe rund um 3DRealize der Firma Visual Components ([Vis12]), oder industrialPhysics ([Wün13]).

Ein weiteres Anwendungsfeld betrifft virtuelles Training im industriellen Bereich, also die Nutzung virtuell nachgebildeter Arbeitsplätze zur Schulung von Mitarbeitern in der Industrie, wie es etwa im Rahmen des VISTRA-Projektes ([KSG11]) angedacht ist.

2.3.3. Medizinische Simulationen

Eine andere Ausprägung des virtuellen Trainings findet sich in medizinischen Anwendungen, hauptsächlich in Simulationen für chirurgische Eingriffe oder invasive Untersuchungen. Die Anforderungen an Simulationssysteme sind hier jedoch noch anspruchsvoller als in den zuvor genannten Szenarien, da zur Simulation des Verhaltens von Zellgewebe oder Muskulatur elastische Objekte unterstützt werden müssen, wobei nicht nur der Kontakt zwischen Objekten, sondern zusätzlich auch die Verformung unter mechanischem Einfluss, oder auch die Zerteilung von Objekten zur Laufzeit mit simuliert werden müssen. Da oft nicht nur visuelle, sondern auch haptische Rückkopplung gewährleistet werden muss, bedingt die dadurch erforderliche Echtzeitfähigkeit solcher Simulationen eine noch höhere Notwendigkeit für effiziente Kollisionserkennungsalgorithmen, die komplexe verformbare Geometrien behandeln können.

Ein Projekt aus diesem Bereich ist das SOFA-Framework (Simulation Open Framework Architecture, [INR13]), das auch im Rahmen der vorliegenden Arbeit als Software-Umgebung für die Entwicklung von Kollisionserkennungsalgorithmen genutzt wird.

2.3.4. Interaktive Unterhaltungs-Anwendungen

Das im alltäglichen Leben wohl am meisten wahrgenommene Anwendungsfeld für mechanisch plausible Simulationen sind Unterhaltungsanwendungen: Zuerst zu nennen sind Spieleanwendungen, für die eine Anzahl meist kommerzieller Softwaresysteme speziell für die Spielewelten zugrundeliegende Physik existiert. Häufig verwendete Softwarelösungen für interaktive Spielenanwendungen sind etwa Havok der Firma Intel [Hav11], PhysX der Firma NVidia [NVi11], oder die CryEngine der Crytek GmbH [Cry11], sowie die quelloffenen Physik-Engines ODE und Bullet ([Cou12]).

Daneben sind Animations- und CGI-Softwarepakete, deren Bedeutung in der Filmindustrie in den vergangenen Jahren immer weiter zugenommen hat, weitere Anwendungsgebiete für mechanische Simulationen; computergenerierte Spezialeffekte und Animationssequenzen stellen zwar völlig andere Anforderungen an die zur Erzeugung verwendete Software als Spiele-Anwendungen, jedoch ist die zugrundeliegende Fragestellung dieselbe: Computergenerierte Modelle und Umgebungen mit einer mechanischen Repräsentation zu versehen, um

dem Benutzer oder Zuschauer eine auf mechanisch “vertraute“ Weise reagierende Umgebung zu vermitteln.

KAPITEL 3

GRUNDLAGEN ZU STARRKÖRPERSIMULATIONEN

Abschnitt 3.1 fasst unterschiedliche Einflussgrößen auf Starrkörpersimulationen in Verbindung mit Faktoren, die deren Laufzeitverhalten bestimmen, zusammen.

In Abschnitt 3.2 wird eine Abgrenzung echtzeitfähiger Starrkörpersimulationen zu anderen Simulations-Technologien, deren Gegenstand ebenfalls die Simulation mechanischen Verhaltens ist, vorgenommen.

Abschnitt 3.3 beschreibt die Grundlagen der allgemeinen Funktionsweise von Starrkörpersimulationen in kurzer Form.

Abschnitt 3.4 und Abschnitt 3.5 erläutern die wichtigsten Grundlagen zur Funktionsweise der einzelnen Teilschritte in der Kollisionserkennung und damit verbundenen Problemen und Herausforderungen.

Abschnitt 3.6 und Abschnitt 3.7 behandeln die Abstraktion mechanischer Kontakte zwischen Objekten in einer Starrkörpersimulation durch die Verwendung punktförmiger Kontakte, sowie deren Relevanz bei der Sicherstellung der Plausibilität und der Stabilität des mechanischen Verhaltens von Objekten in einer Simulation.

Eine Übersicht über die Kollisionsbehandlung und deren Aufgaben ist in Abschnitt A.2 als Teil von Anhang A enthalten.

3.1. Anforderungen an Starrkörpersimulationen beim Einsatz in der Robotik

Nach der Betrachtung zweier Beispielanwendungen aus dem Bereich der Industrie- und Service-Robotik lassen sich nun einige Anforderungen an Starrkörpersimulationen herleiten, deren Erfüllung eine notwendige Voraussetzung für den nutzbringenden Einsatz eines solchen Software-Werkzeugs im Rahmen von Robotik-Anwendungen sind. Die Benennung konkreter Kennzahlen für die im Folgenden genannten Anforderungen ist aufgrund der Bandbreite möglicher Einflussfaktoren und der unterschiedlichen Zielsetzungen der zuvor diskutierten

Projektbeispiele schwierig. Daher führt die folgende Aufstellung als alternative Kriterien solche Einflussfaktoren auf, die Auswirkungen auf die Leistungsfähigkeit und die Plausibilität einer Starrkörpersimulation haben können.

Anforderung	Ausprägung	Kombiniert mit	Ausprägung	Laufzeitbedarf	Plausibilität
Geometrische Komplexität	gering	Anzahl Objekte	wenige	gering	gering
			viele	gering	gering
	groß		gering	groß	gering
			viele	groß	groß
Funktionalität Sensorik	gering	Anzahl Objekte	wenige	gering	gering
			viele	gering	gering
	groß		gering	groß	gering
			viele	groß	groß
Funktionalität Kinematik	gering	Anzahl kinematischer Strukturen	wenige	gering	gering
			viele	gering	gering
	groß		gering	gering	gering
			viele	groß	groß

Tabelle 3.1.: Die wichtigsten Anforderungen an Starrkörpersimulationen in der Robotik, korreliert mit deren Einfluss auf Laufzeitbedarf und Plausibilität einer Simulation: Die Kollisionserkennung wird von der Struktur und Anzahl simulierter Objekte ebenso wie von Art und Anzahl simulierter Sensoren am stärksten beeinflusst, während die Simulation kinematischer Strukturen als Teil der Simulation mechanischen Verhaltens nur einen geringen Einfluß auf diese Größen hat.

Tabelle 3.1 ordnet die Funktionsbereiche einer Starrkörpersimulation, die im Bezug auf die Robotik am wichtigsten sind: Die Simulation von Aktorik und Sensorik. Des weiteren erfasst Tabelle 3.1 die beiden Faktoren, die den größten Einfluss sowohl auf den nötigen Laufzeitaufwand als auch auf die Plausibilität einer Simulation haben: Der Detailgrad bei der Modellierung verwendeter Kollisionsgeometrien, und die Anzahl von Objekten in einer simulierten Arbeitsumgebung.

Die schon in Kapitel 1 (Abschnitt 1.1 und Abschnitt 1.2) erläuterten konkurrierenden Faktoren geometrische Komplexität und Objektanzahl gelten naturgemäß auch in Zusammenhang mit der Robotik. Hier stellt sich die Frage, ob die Verwendung präzise modellierter Kollisionsgeometrien für ein Simulations-Szenario sinnvoll ist: Die in Abschnitt 2.1 und Unterabschnitt 2.2.1 vorgestellten Projektbeispiele aus der Industrie- und Service-Robotik repräsentieren beide Alternativen:

- Durch die Anforderungen des Szenarios in der virtuellen Inbetriebnahme war die Verwendung direkter Entsprechungen CAD-konstruierter Werkstücke und Werkzeuge in der Kollisionserkennung zwingend notwendig: Die Arbeitsumgebung musste mit einer realen Entsprechung übereinstimmen, da ein mit Hilfe der Simulation erstelltes Steuerungsprogramm ansonsten nicht für die Ausführung in einer realen Umgebung geeignet gewesen wäre.
- Bei Robot3D dagegen war es aufgrund der Problemstellung möglich, auf detailgetreu modellierte Kollisionsgeometrien zu verzichten und mit vereinfachten geometrischen Repräsentationen zu arbeiten (beispielsweise konnte auf die detaillierte Simulation von Andock-Vorgängen zwischen zwei Modulen verzichtet werden).

Hinsichtlich der Simulation von Sensorik-Komponenten (sofern geeignete Abstraktionen unter Verwendung der Kollisionserkennungs-Komponente einer Starrkörpersimulation verwendet werden können) gelten dieselben Aussagen wie für die konkurrierenden Größen Objektanzahl und geometrische Komplexität: Mit Hilfe geometrischer Abstraktionen simulierte Sensoren müssen genauso wie andere Objekte in einer Simulation durch die Kollisionserkennung auf Berührung oder Überschneidung überprüft werden, und haben damit in gleichem Maß Einfluss auf den Laufzeitbedarf der Kollisionserkennung.

Die Simulation kinematischer Strukturen hingegen ist neben der Kollisionsbehandlung exklusiv Bestandteil der Simulation mechanischen Verhaltens. Obwohl die numerisch stabile Lösung eines unter Umständen sehr umfangreichen Gleichungssystems an sich eine anspruchsvolle Aufgabe ist, so ist der dafür nötige Laufzeitaufwand gegenüber der Kollisionserkennung jedoch um ein Vielfaches geringer. Diese Beobachtung gilt auch noch für den Fall, dass der Funktionsumfang einer Starrkörpersimulation über die Behandlung einzelner Paare durch Gelenke verbundener Starrkörper hinaus erweitert wird ([Fea08]).

Die zwei zuvor diskutierten Projektbeispiele und die daraus ableitbaren Beobachtungen verdeutlichen, dass die Leistungsfähigkeit der Kollisionserkennung auch im Bereich der Robotik das wesentliche Kriterium für den sinnvollen Einsatz von Starrkörpersimulationen ist. Unter dieser Voraussetzung ist die Entwicklung eines Verfahrens zur Kollisionserkennung, das für die effiziente Behandlung komplex aufgebauter Geometrien konzipiert ist, auch in Zusammenhang mit Simulationsanwendungen in der Robotik eine sinnvolle Wahl.

3.2. Begriffsdefinitionen

Eine kurze Diskussion der Begriffe *echtzeitfähig* und *mechanisch plausible Starkörpersimulation* ist für deren Einordnung und Verwendung im weiteren Verlauf der Arbeit nützlich.

3.2.1. Echtzeitfähigkeit

Echtzeitfähig definiert sich im Rahmen dieser Arbeit als

> die Ausführung einer Iteration einer Simulation in oder unterhalb eines vorgegebenen Zeitraumes.

Der Begriff *Echtzeitfähigkeit* bedarf noch einer genaueren Erläuterung: Innerhalb einer Simulation existiert eine eigene Zeitskala, die mit der vergangenen realen Zeit nicht in allen Fällen in einer direkten 1:1-Relation steht. Etwa in Strömungssimulationen benötigt die Berechnung des Verhaltens eines Mediums in einem Intervall einiger Sekunden oft mehrere Stunden Rechenzeit; andererseits gilt das Gegenteil in Simulationen, denen vergleichsweise einfache Annahmen zugrunde liegen; hier ist es möglich, pro Intervall vergehender realer Zeit ein Vielfaches der simulationsinhärenten Zeit zu behandeln. Für einen Benutzer der Simulation ist die simulationsinhärente Zeitskala jedoch normalerweise nicht von Interesse; in der Interaktion mit einer Simulation ist vielmehr die reale Zeit relevant, die die Simulation benötigt, um einen kompletten Zeitschritt des simulierten Modells zu absolvieren, sprich das Verhältnis zwischen der real vergehenden Zeit und der Zeit, die eine Simulation für ihre Berechnungen benötigt.

Echtzeitfähig bezeichnet nun die Eigenschaft einer Simulation, innerhalb eines durch die spezifischen Anforderungen an die Simulationsanwendung gegebenen Zeitintervalls ihre Berechnungen abschließen zu können. Diese spezifischen Anforderungen betreffen die Notwendigkeit, den Nutzern der Simulation (ob dies menschliche Anwender, oder wie im Fall von "Hardware in the Loop"-Simulationen, elektronische oder mechatronische Steuergeräte oder -systeme sind) ein für deren Bedürfnisse hinreichend exaktes Ergebnis innerhalb der Rahmenbedingungen des simulierten Szenarios oder der simulierten Umgebung bereitzustellen.

Im Kontext interaktiver Computergrafik-Anwendungen ergibt sich dieser Zeitraum durch die Eigenschaft des Sehsinns, bewegte Bilder erst ab einer Frequenz von etwa 25 Einzelbildern als flüssig animiert wahrzunehmen, zu circa 40 Millisekunden.

Ein anderes Echtzeitkriterium ist durch die Notwendigkeit bedingt, eine maximale Ausführungszeit aufgrund einer durch externe Teilnehmer (sei dies ein menschlicher Anwender, oder beispielsweise ein Steuergerät in einer Hardware-in-the-Loop-Simulation) vorgegebene Zeittaktung einhalten zu können.

Diesem Aspekt trägt die Definition des Begriffs *virtuelle Inbetriebnahme* nach Wünsch ([Wün07]) Rechnung:

> "Der Begriff virtuelle Inbetriebnahme beschreibt den abschließenden Steuerungstest anhand eines Simulationsmodells, das in der Kopplung von realer oder virtueller Steuerung mit dem Simulationsmodell das Shannon'sche Abtasttheorem für alle Steuerungssignale einhält."

Das Nyquist-Shannon-Abtasttheorem beschreibt den Zusammenhang zwischen der Abtastrate eines Signals und der maximal daraus ableitbaren Frequenz des Signals. Eine wichtige Aussage des Theorems ist, dass ein Signal mindestens mit dem Zweifachen der maximal nachzuweisenden Frequenz abgetastet werden muss, um diese Frequenz in einem Signal sicher detektieren zu können.

Für binäre Steuerungssignale liegt nach diesem Theorem die maximal nachzuweisende Frequenz bei der halben Steuerungsfrequenz; diese reicht aus, um die Reaktion einer Steuerung anhand eines Signalübergangs sicher detektieren zu können. Abbildung 3.1 illustriert diesen Zusammenhang zwischen Signalübergang und Abtastfrequenz.

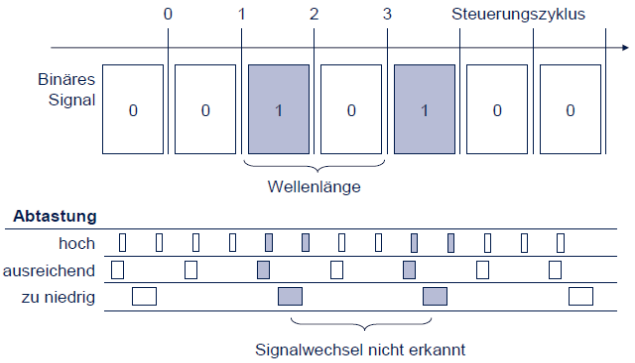


Abbildung 3.1.: Zusammenhang zwischen Abtastrate und Detektierung von Signalübergängen (aus [Wün07])

3.2.2. Plausible Mechaniksimulationen

In Abgrenzung zu Mehrkörpersystemen (Unterabschnitt A.1.1) und Finite-Elemente-Methoden (Unterabschnitt A.1.2) sei eine *plausible Msimulation* in Anlehnung an [Ebe10] und [Mir96] als

> "die zeitlich diskrete Simulation der Dynamik starrer Körper unter dem Einfluss externer Kräfte und Momente nach den Gesetzmäßigkeiten der Newton'schen Mechanik unter Verwendung numerischer Methoden innerhalb für den interaktiven Betrieb geeigneter Zeitintervalle"

definiert.

Mechanisch plausibel soll im Folgenden im Folgenden die Eigenschaft virtueller Objekte bezeichnen, sich entsprechend der Gesetzmäßigkeiten der klassischen Mechanik zu verhalten, ohne jedoch hierbei eine vollständige Abbildung aller physikalischen Eigenschaften realer Objekte zu bieten.

Vorrangig bedeutet dies, dass sich räumlich ausgedehnte Objekte nicht gegenseitig durchdringen dürfen, und dass deren Bewegungsverhalten unter dem Einfluss von Kräften und Momenten durch die mathematischen Modelle der klassischen Mechanik bestimmt wird.

Allgemein beschränkt sich die Mächtigkeit der verwendeten Modellierung auf eine makroskopische Betrachtungsweise: So werden etwa mesoskopische Effekte wie in der Realität auftretende minimale Verformung solider Objekte beim Auftreffen auf andere solide Objekte vernachlässigt. Die Modellierung von Haft- und Gleitreibung sind im Allgemeinen stark vereinfachte Annäherungen an reale Gegebenheiten, wie sie etwa Gegenstand der Tribologie sind. Auch werden etwa thermodynamische Effekte, wie sie bei elastischer oder plastischer Verformung von Objekten auftreten, gänzlich vernachlässigt. Dies stellt im Vergleich zu FEM-Simulationen eine noch weitergehende Einschränkung der im zugrunde liegenden Modell berücksichtigten Eigenschaften dar, trägt jedoch der Berücksichtigung der strengeren Laufzeit-Anforderungen Rechnung, die an echtzeitfähige Starrkörpersimulationen im Gegensatz zu FEM- oder MKS-Systemen gestellt werden.

Während bei Letzteren die Präzision des zugrunde liegenden Modells im Vordergrund steht und die Prioritäten bei der Bewertung von Simulationsergebnissen bei der Übertragbarkeit auf beziehungsweise der Verifikation von realen mechanischen Systemen liegen, ist bei Ersteren die Leistungsfähigkeit des Simulationssystems in Hinblick auf die benötigte Rechenzeit pro Iteration das wesentliche Leistungsmerkmal.

3.3. Starrkörpersimulationen

Die numerisch diskrete Simulation von Mehrkörpersystemen zerfällt in zwei sequentielle Teilschritte, die jeweils auf den geometrischen und mechanischen Eigenschaften der simulierten Objekte operieren: Die *Kollisionserkennung* und die *Kollisionsbehandlung*.

Abbildung 3.2 zeigt die schematische Übersicht der einzelnen Teilaufgaben im sequentiellen Ablauf pro Iteration einer Starrkörpersimulation.

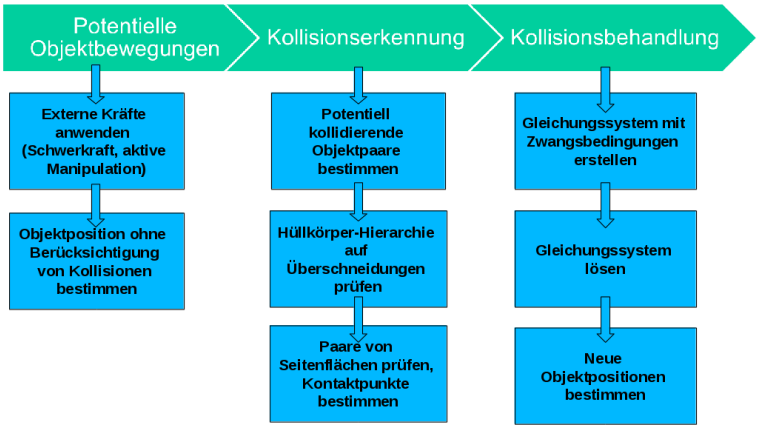


Abbildung 3.2.: Einzelne Schritte pro Iteration einer echtzeitfähigen Starrkörpersimulation (nach [Erl05] und [MC94]); Die Vorwegnahme potentieller Objektbewegungen ohne Berücksichtigung von auftretenden Kollisionen kann als Teilschritt der Kollisionserkennung angesehen werden.

Die zur nachfolgenden Diskussion von Kollisionserkennungs-Algorithmen in Kapitel 4 und Kapitel 5 notwendigen Grundlagen werden im Folgenden eingeteilt nach der einzelnen Teilschritten in einer Iteration einer echtzeitfähigen Starrkörpersimulation diskutiert:

- Abschnitt 3.4 bis Abschnitt 3.5 stellen Grundlagen zur *Vorfilter*- und *Objektpaar*-Phase vor: Schnitt-Tests zwischen Hüllkörpern und Elementen von Dreiecks-Netzen.
- Abschnitt 3.6 und Abschnitt 3.7 diskutieren das Zusammenwirken zwischen Erkennung und Behandlung von Kollisionen, und die Relevanz der Verwaltung von Kontaktpunkt-Konfigurationen für die Stabilität und Plausibilität einer Starrkörpersimulation.

3.3.1. Begriffsbestimmung

Als Präzisierung der Begriffsbestimmung in Unterabschnitt 3.2.2 gliedert sich eine Starrkörpersimulation in zwei Hauptaufgabengebiete:

- Die *Kollisionserkennung*: In diesem ersten Schritt werden die Geometrien simulierter Körper auf gegenseitige Berührung oder Überschneidung überprüft. Die Kollisionserkennung befasst sich vorrangig mit rein geometrischen Fragestellungen: Ob Objekte überhaupt innerhalb eines Simulationsschrittes kollidieren können, an welchen Stellen der zugeordneten Geometrien dies geschieht, und wann es zum ersten Kontakt zwischen Objekten kommt. Dementsprechend sind die verwendeten Hilfsmittel vorwiegend dem Bereich der Computergrafik und der algorithmischen Geometrie entliehen. Ergebnis dieses Verarbeitungsschrittes sind die räumlichen Koordinaten von Berührungs-Punkten zwischen Körpern in der Simulation, an denen Zwangsbedingungen wie Kontakt- und Reibungskräfte angelegt werden können, die das mechanische Verhalten der jeweiligen Objekte beeinflussen.
- Die *Kollisionsbehandlung*: In diesem zweiten Schritt werden ausgehend von den Ergebnissen der Kollisionserkennung die mechanischen Reaktionen der Objekte in der Simulation bestimmt. Anhand neu entstandener, bestehender oder sich lösender Kontaktkonfigurationen zwischen Objekten werden Positionen und Bewegungs-

Geschwindigkeiten der Objekte nach den Gegebenheiten der Newton'schen Mechanik modifiziert. Eine weitere essentielle Aufgabe ist es in der Kollisionsbehandlung, potentielle Durchdringungen zwischen Geometrien durch die Verwendung geeigneter Kollisionsreaktionen (je nach Funktionsweise der zugrunde liegenden Verfahren Kontakt-Impulse oder -Kräfte) zu verhindern oder wieder rückgängig zu machen.

3.3.2. Die Kollisionserkennung

Dieser erste Teilschritt einer Starrkörpersimulation ist die am schwierigsten zu optimierende Aufgabe in Starrkörpersimulationen. Der Rechenzeitaufwand für die Bewältigung der zu lösenden Aufgabe hängt von verschiedenen Faktoren ab, die in Konkurrenz zueinander stehen:

- Der erste Einflussfaktor ist die Anzahl der Geometrien in einer simulierten Szene.
- Ein weiterer Faktor ist die Nähe verschiedener Objekte zueinander.
- Schließlich hat auch die Gruppierung von Objekten in der simulierten Szene Einfluss auf den Laufzeitbedarf der Kollisionserkennung.

Neben der globalen Verteilung von Objekten in der Szene ist es vor allem die spezifische Struktur der verwendeten Geometrien, die wesentlichen Einfluss auf den Aufwand der Berechnung von Kollisionen zwischen verschiedenen Objekten hat. Abhängig von der gewählten Modellierung von Geometrien sind eine große Anzahl von Rechenoperationen notwendig, um zwei Geometrien vollständig auf mögliche Kollisionen zu überprüfen. Dieser Aufwand vergrößert sich noch, sofern man die Bewegung von Objekten innerhalb eines Simulations-Zeitschrittes mit berücksichtigen möchte oder muss, anstatt Objektkollisionen nur zu einem einzelnen Zeitpunkt am Ende des jeweiligen Zeitintervalls zu prüfen.

Um den Laufzeitbedarf pro Simulationsschritt möglichst niedrig zu halten, ist es einerseits notwendig, die Anzahl tatsächlich auf Kollisionen zu prüfender Geometrien möglichst weit zu reduzieren, und andererseits die detaillierte Überprüfung einzelner Kombinationen von Geometrien auf gegenseitige Kollisionen so effizient wie möglich zu gestalten. Die detaillierte Überprüfung von Geometrien erfolgt dabei üblicherweise paarweise.

Aus diesem Grund unterteilt sich die Kollisionserkennung üblicherweise wiederum in zwei Teilschritte:

- Die *Vorfilter-Phase* (engl. *Broadphase*, Abschnitt 3.4): Unnötige detaillierte Kollisionstests können für Objekte ausgeschlossen werden, die zu weit voneinander entfernt sind (oder sich in ihren Bewegungsbahnen im betrachteten Zeitintervall nicht nahe genug kommen), um überhaupt kollidieren zu können.
- Die *Objektpaar-Phase* (engl. *Narrow-Phase*, Abschnitt 3.5): Alle Objektpaare, für die die Vorfilter-Phase potentiell die Möglichkeit einer Kollision aufgezeigt hat, werden detailliert auf gegenseitige Kollisionen überprüft.

3.3.3. Die Kollisionsbehandlung

Die Kollisionsbehandlung hat zum einen die Aufgabe, neu entstehende Kollisionen zwischen Körpern in der Simulation aufzulösen; hier kann es aufgrund der zeitdiskreten Funktionsweise

von Simulationssystemen zu kurzzeitigen gegenseitigen Durchdringungen von Kollisionsgeometrien kommen, die durch das Anwenden geeigneter Gegenreaktionen auf die beteiligten Objekte aufgelöst werden müssen.

Zum anderen muss die Kollisionsbehandlung in der Lage sein, bleibende Kontakte zwischen Körpern korrekt beizubehalten, wobei zusätzlich noch das Reibungsverhalten an den Kontaktpunkten simuliert werden muss.

Die Ergebnisse der Kollisionsbehandlung bestimmen die neuen Positionen, Geschwindigkeiten und anliegende Kräfte für alle Objekte in der Szene nach der Ausführung einer Iteration der Simulation.

Eine weitere Aufgabe der Kollisionsbehandlung besteht in der Behandlung zusätzlicher Bewegungsbeschränkungen zwischen Körpern in der Simulation, die unabhängig von Kollisionen zwischen bewegten Körpern in der Simulation spezifiziert sind: Gelenkverbindungen zwischen Paaren von Körpern. Verfahren zur Behandlung von Gelenkverbindungen werden im Rahmen dieser Arbeit jedoch nicht schwerpunktmäßig behandelt; in Unterabschnitt A.2.5 werden diese kurz diskutiert.

3.4. Die Vorfilter-Phase (Broadphase)

Broadphase-Verfahren können in zwei verschiedenen Abschnitten der Kollisionserkennung verwendet werden:

- Zum einen angewandt auf die komplette simulierte Szene. Im Fall der szenenweiten Kollisionserkennung werden üblicherweise Hüllkörper, die individuelle Objekte in der Szene komplett umschließen, in einer räumlichen Partitionierung angeordnet und es wird anschließend geprüft, welche Hüllkörper sich gemeinsam in einer oder mehrerer Regionen der räumlichen Partitionierung befinden. Eine zweite Alternative in der szenenweiten Vorfilter-Phase besteht darin, Hüllkörper von Objekten anhand ihrer Koordinaten-Intervalle entlang der einzelnen Achsen eines gemeinsamen 3D-Koordinatensystems zu sortieren und nur diejenigen Paare von Objekten weiter zu berücksichtigen, deren Hüllkörper sich entlang mindestens einer Achse überschneiden.
- Zum anderen angewandt auf Paare potentiell kollidierender Objekte. Geht es um die Überprüfung von Objektpaaren auf gegenseitige Berührung, so können Hüllkörper-Hierarchien als vereinfachte Repräsentationen für komplexe Objekte in einer Simulation dienen, um den späteren Berechnungsaufwand in der Objektpaar-Phase zu reduzieren. Eine solche Hüllkörper-Hierarchie ist so aufgebaut, dass die oberste Ebene der Hierarchie das komplette Objekt einschließt, und die weiter unten in der Hierarchie angesiedelten Ebenen sukzessive immer kleinere Teile des gesamten Objektes umschließen, bis dessen kleinste Konstruktionselemente erreicht sind, oder das Volumen der Hüllgeometrien ein bestimmtes minimales Volumen unterschreitet. Bei der Überprüfung auf gegenseitige Kontakte zwischen einem Objektpaar werden sukzessive alle Ebenen der beiden Hüllkörper-Hierarchien auf Überschneidung überprüft. Die Überprüfung tiefer in der Hierarchie liegender Ebenen wird nur dann fortgesetzt, wenn es auf weiter oben liegenden Ebenen eine Überschneidung zwischen den Hüllgeometrien der entsprechenden Teile der Hüllkörperhierarchien gab.

Neben Hüllkörper-Hierarchien ist auch die Verwendung von *Raumpartitionierungs-Schemata* möglich, in denen für einzelne Geometrien nicht an diese angepasste Hüllgeometrien berechnet werden: Stattdessen wird eine Geometrie denjenigen Subvolumina einer Raumpartitionierung zugeordnet, in denen sie enthalten ist. Raumpartitionierungs-Schemata werden in Kapitel 5 in Zusammenhang mit GPU-basierter Kollisionserkennung verwendet werden, jedoch nicht als Datenstruktur zur Ermittlung von Überschneidungen zwischen Objekten, sondern als Hilfsmittel zur effizienten Verarbeitung von Punkt-Primitiven.

3.4.1. Hüllgeometrien

Als Alternativen für Hüllgeometrien sind eine Vielzahl elementarer und komplexer gestalteter Geometrien möglich; die am häufigsten verwendeten Varianten sind:

- Hüll-Kugeln ([TC96], [OD99], [WZ09b], [BCW11])
- Achsen-orientierte Hüllquader (engl. *Axis Aligned Bounding Box*, AABB; [Zac95], [Ber97])
- Objekt-orientierte Hüllquader (engl. *Object-aligned Bounding Box*, OBB; [GLM96], [LGLM99])
- Hüll-Polyeder (engl. *k-Distance Oriented Polygon*, k-DOP; [KHM+98], [Zac98])

Während bei einfach strukturierten Hüllgeometrien die gegenseitige Überprüfung auf Überlappung eine einfach zu berechnende Form hat, besteht häufig das Problem mangelnder Anpassung der Hüllgeometrie an die zugrundeliegende tatsächliche Geometrie. Dementsprechend kann es dazu kommen, dass die Vorfilter-Phase eine große Anzahl möglicher Objektkollisionen meldet, für die tatsächlich aber keine tatsächliche Kollision besteht. In der Objektpaar-Phase können deswegen unnötig viele detaillierte Kollisionsberechnungen angestoßen werden. Demgegenüber ist bei komplexer gestalteten Hüllgeometrien die Berechnung auf gegenseitige Überlappung weitaus komplexer, aufgrund der besseren Annäherung an die tatsächliche Geometrie ist die Wahrscheinlichkeit jedoch geringer, dass in der Objektpaar-Phase ein unnötiger detaillierter Kollisionstest durchgeführt werden muss.

Abbildung 3.3 zeigt eine Aufstellung einiger häufig verwendeter Typen von Hüllgeometrien.

3.4.1.1. Achsen-orientierte Hüllquader (AABB)

Die Normalen der Seitenflächen eines achsen-orientierten Hüllquaders sind an den Koordinaten-Achsen eines gegebenen Koordinatensystems ausgerichtet; dies bedeutet, dass eine AABB neu berechnet werden muss, sofern die von ihr approximierte Geometrie eine Rotationsbewegung vollführt.

Der wesentliche Vorteil von AABBs ist der einfache Schnitt-Test, der nur einen komponentenweisen Vergleich der Koordinaten von zwei Punkten, die die Ausdehnung einer AABB beschreiben, erfordert (Algorithmus 1).

Trotz der Notwendigkeit, AABBs bei Rotationsbewegungen neu an der umschlossenen Geometrie ausrichten zu müssen, sind AABBs und AABB-Hierarchien aufgrund der einfachen Schnittberechnungen ein in den meisten gängigen Starrkörpersimulations-Lösungen (bei-

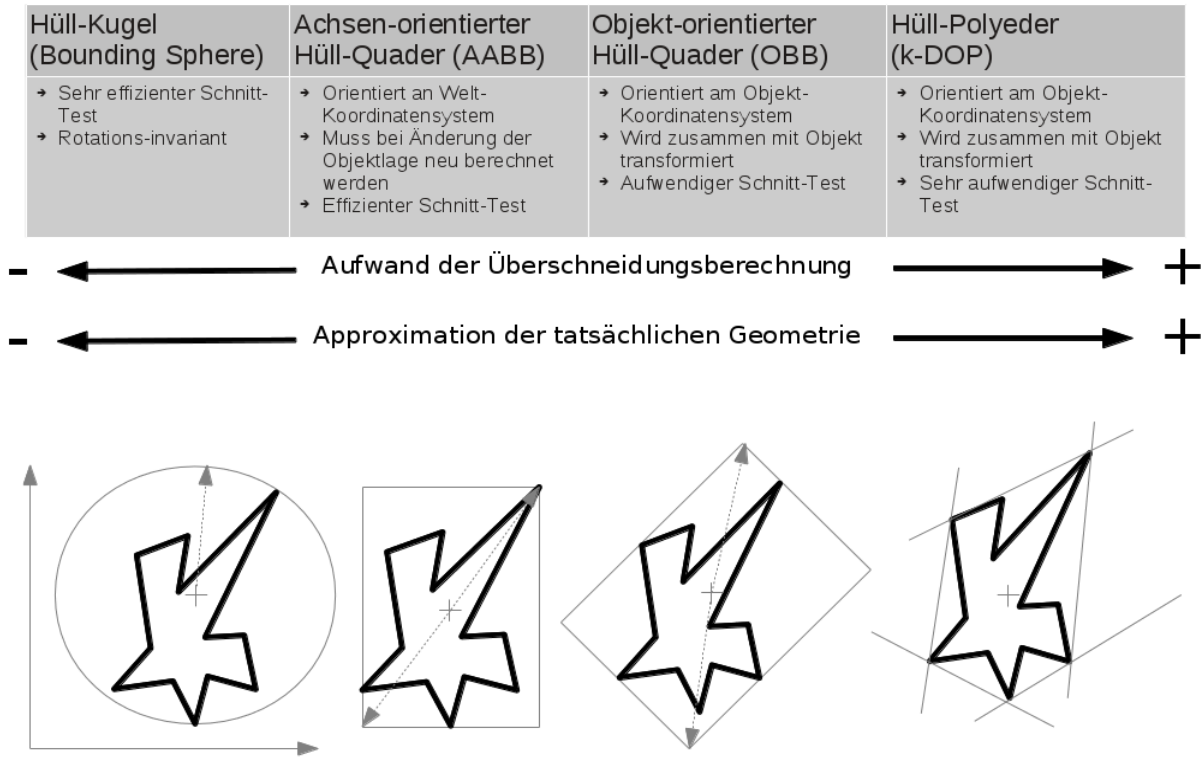


Abbildung 3.3.: Einige häufig verwendete Arten von Hüllgeometrien: Die konkurrierenden Größen bei der Wahl von Hüllgeometrien sind die Komplexität des Tests auf Überschneidung gegenüber der Güte der Approximation der umschlossenen Geometrie als Verhältnis zwischen deren Volumen und dem Volumen der Hüllgeometrie.

spielsweise [Cou13], [NVi11], [Hav11]) vorhandener Standard-Mechanismus in der Vorfilter-Phase beziehungsweise als Hüllkörper-Hierarchie in der Objektpaar-Phase.

Für verformbare Kollisionsgeometrien sind AABB-Hierarchien oft die einzige in Frage kommende Alternative für die Objektpaar-Phase, da die Neuberechnung von Hierarchien mit anderen Arten von Hüllkörpern wesentlich aufwendiger ist als die für AABBs.

Für die Neuberechnung einer AABB kommen unterschiedliche Strategien in Frage ([Eri05, Kapitel 4.2]):

- Berechnung einer AABB anhand einer Hüllkugel: Da eine Hüllkugel rotations-invariant ist, entfällt hier die Notwendigkeit, eine komplette Neuausrichtung der AABB bei Rotationsbewegungen vorzunehmen. Der Nachteil dieser Methode besteht aber dementsprechend in einer sehr schlechten Approximation der umschlossenen Geometrie.
- Die Berechnung einer optimal angepassten AABB anhand aller Eckpunkte einer umschlossenen Geometrien.
- Ermitteln der Extrema aus der Menge der Eckpunkte einer umschlossenen Geometrie mittels Hill-Climbing: Mit einer geeigneten Datenstruktur, die den Zugriff auf direkt benachbarte Eckpunkte erlaubt, können die in alle sechs Richtungen jeweils maximal vom Objekt-Mittelpunkt entfernten Eckpunkte durch eine Maxima-Suche ermittelt

Algorithmus 1 : Schnitt-Test für zwei AABBs

Input : P, Q
if $P.max[0] < Q.min[0] \vee P.min[0] > Q.max[0]$ **then**
  Überschneidung;
if $P.max[1] < Q.min[1] \vee P.min[1] > Q.max[1]$ **then**
  Überschneidung;
if $P.max[2] < Q.min[2] \vee P.min[2] > Q.max[2]$ **then**
  Überschneidung;
Keine Überschneidung;

und so die Ausdehnung einer AABB ermittelt werden. Dieses Verfahren funktioniert allerdings nur für konvexe Geometrien.

- Berechnung einer neuen AABB anhand der vorherigen, rotierten AABB.

3.4.1.2. Objekt-ausgerichtete Hüllquader (OBB)

Eine OBB bietet im Gegensatz zu einer AABB die Möglichkeit, das Hüllvolumen mit einer Orientierung zu versehen. Diese Erweiterung erlaubt nicht nur eine bessere Approximation einer umschlossenen Geometrie, sondern vermeidet auch die Notwendigkeit der Aktualisierung einer OBB nach Rotationen.

Gegenüber AABBs bedeutet dies allerdings einen höheren Speicherplatzbedarf zur Speicherung der OBB-spezifischen Rotationsmatrix (neben dem Mittelpunkt des OBB-Volumens und der Ausdehnung in Richtung der Koordinaten-Achsen), und ebenso einen deutlich aufwendigeren Schnitt-Test im Vergleich zu AABBs.

Das Grundprinzip der effizientesten Test-Variante basiert auf dem sogenannten Separating-Axis-Theorem (SAT). Dieses Theorem basiert wiederum auf dem Trennungssatz ([Wer97]), als dessen Konsequenz zwei konvexe Geometrien sich dann nicht überschneiden, wenn eine Hyperebene (Gerade in 2D, Ebene in 3D) existiert, für die die beiden konvexen Geometrien in unterschiedlichen Halbräumen liegen. Als Nachweis für die Existenz einer Hyperebene genügt die Existenz einer separierenden Achse, die orthogonal zu einer möglichen Hyperebene liegt. Die separierende Achse ist in den für die Kollisionserkennung relevanten Fällen (2D und 3D) immer eine Gerade.

Projiziert man Geometrie-Elemente der zu testenden konvexen Objekte auf entsprechende mögliche separierende Achsen, und überschneiden sich die resultierenden Intervalle auf einer der separierenden Achsen nicht, so überschneiden sich auch die beiden Geometrien als Gesamtes nicht (Abbildung 3.4). Ist im Umkehrschluss für alle Geometrieelemente auf deren möglichen separierenden Achsen eine Überschneidung vorhanden, so überschneiden sich auch die Geometrien selbst.

Um zwei OBBs auf eine mögliche Überschneidung zu testen, sind insgesamt 15 SAT-Tests nötig ([Got00]):

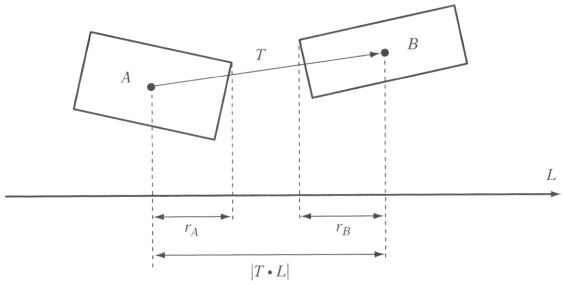


Abbildung 3.4.: Schnitt-Test anhand des Separating-Axis-Theorems: Für ein Paar aus OBBs werden maximal 15 separate Separating-Axis-Tests benötigt, um eine Überschneidung feststellen zu können.

- Jeweils drei SAT-Tests für die Achsen des objekt-spezifischen Koordinatensystems jeder OBBs
- Neun weitere Tests die möglichen Kreuzprodukte aus den Kombinationen der Koordinaten-Achsen beider OBBs

Zur Berechnung einer möglichst optimal an eine umschlossene Geometrie angepassten OBB werden meist Verfahren auf Basis der Hauptkomponentenanalyse (PCA) verwendet. Die durch eine Hauptkomponentenanalyse ermittelte Kovarianz-Matrix der Eckpunkte einer Eingangsgeometrie bildet dabei das OBB-spezifische Koordinatensystem, und das Baryzentrum der Punktmenge den Mittelpunkt der OBB. Eine Möglichkeit zur Bestimmung der Ausdehnungen der OBB besteht in der Folge beispielsweise in der Bestimmung der jeweils am weitesten entfernten Eckpunkte von den Ebenen, die von jeweils zwei der Achsen des OBB-spezifischen Koordinatensystems aufgespannt werden. Weitere Details zu unterschiedlichen Methoden zur Berechnung von optimalen OBBs sind etwa [Got00] oder [BHP01] zu entnehmen.

3.4.1.3. Hüll-Kugeln

Auf Hüll-Kugeln wird im Rahmen dieser Arbeit nicht ausführlicher eingegangen: Obwohl Kugeln einen sehr effizienten Schnitt-Test und einen sehr geringen Speicherplatzbedarf haben, ist die durch sie mögliche Annäherung an die zugrundeliegende Geometrie oft nur schlecht.

Für Hüllkugel-Hierarchien gibt es eine Reihe von Ansätzen, die

- entweder über eine Minimierung der Volumen-Differenz zwischen Hüllkugel und umschlossener Geometrie ([OD99], [TC96])
- oder über die Konstruktion einer Hüllkörper-Hierarchie, die nur das Innere einer umschlossenen Geometrie umfasst ([WZ09b], [WZ09a])

versuchen, den wesentlichen Nachteil von Hüllkugel-Hierarchien zu kompensieren. Obwohl diese Techniken vielversprechende Ansätze darstellen und die Leistungscharakteristiken vor allem im Fall von [WZ09b] beeindruckend sind, so gibt es doch zum aktuellen Zeitpunkt beispielsweise keine Implementierung von Hüllkugel-Hierarchien für GPU-Prozessoren. Es muss außerdem beachtet werden, dass die Erzeugung von optimal angepassten Hüllkugel-Hierarchien mit enormem Aufwand bei der Vorberechnung der Hüllkörper-Hierarchien verbunden ist.

3.4.1.4. k-DOPs

Ein k-DOP (*D*iscrete *O*riented *P*olygon, k bezeichnet die Anzahl verwendeter Seitenflächen je Hüllkörper) wird durch die Umfassung der ursprünglichen Geometrie mit Paaren von Ebenen gebildet, wobei jedes Ebenenpaar eine gemeinsame Normale aufweist. Der durch jedes Ebenenpaar eingeschlossene Teilraum beschreibt dabei die maximale Ausdehnung der umschlossenen Geometrie in Richtung der Normalen eines Ebenen-Paars. Die Besonderheit bei der Konstruktion von k-DOPs besteht darin, dass für alle Hüllgeometrien dieselben Normalen-Vektoren zur Bestimmung der umschließenden Ebenen-Paare verwendet werden, und nur die Abstände der Ebenen zum Ursprung des Bezugs-Koordinatensystems gespeichert werden müssen. k-DOP-Hierarchien bieten sowohl einen effizienten Schnitt-Test als auch eine sehr gute Anpassung an die zugrundeliegende Geometrie ([KHM+98], [Zac98], [Z+00]): Die Anpassung an die zugrundeliegende Geometrie ist bei k-DOPs mit zunehmender Anzahl an Seitenflächen besser, allerdings ist der benötigte Speicher pro Hüllkörper vergleichsweise groß. Schnitt-Tests für k-DOPs sind ähnlich effizient wie bei AABBs, jedoch steigt der Speicherplatzbedarf je Hüllvolumen mit zunehmender Anzahl von Seitenflächen an. Des weiteren sind k-DOPs nicht rotations-invariant, so dass wie bei AABBs nach Änderung der Orientierung eines Objekts umgebende k-DOPs neu berechnet werden müssen. Obwohl k-DOPs vorteilhafte Charakteristiken aufweisen und im Rahmen der Entwicklungs-Arbeit zu dieser Dissertation untersucht wurden, wurde OBB-Hierarchien gegenüber k-DOP-Hierarchien der Vorzug gegeben, da deren Speicherbedarf gegenüber k-DOPs konstant ist, und die Approximation der zugrunde liegenden Geometrie der von k-DOPs mit niedriger Facetten-Zahl entspricht.

3.4.2. Hüllkörper-Hierarchien

Für komplex strukturierte Geometrien mit großen Menge von Seitenflächen ist es in der Objektpaar-Phase nicht praktikabel, jedes Paar von Seitenflächen aus zwei Geometrien einzeln auf mögliche Kontakte zu untersuchen. Bei der detaillierten Überprüfung komplex gestalteter Kollisionsgeometrien sind daher Hüllkörper-Hierarchien als Datenstrukturen zur Beschleunigung der Suche nach potentiell kollidierenden Teilen der involvierten Modelle im Einsatz.

Hierbei gibt es unterschiedliche Möglichkeiten, die Hierarchien zweier in einem Objektpaar-Test überprüfter Modelle paarweise zu durchlaufen; der benötigte Laufzeitaufwand für die komplette Auswertung beider Hierarchien hängt dabei von mehreren Faktoren ab:

- Der Lage beziehungsweise der Abstand der beiden Modelle zueinander: Je näher sich zwei Modelle sind, desto wahrscheinlicher werden Überlappungen der Hüllkörperhierarchien.
- Der Tiefe und der Verzweigungsgrad der Hierarchien: Beide sind abhängig davon, auf welche Weise die Hüllkörper-Hierarchien erzeugt werden.
- Die Qualität der Approximation an Teile der zugrunde liegenden geometrischen Modelle: Diese ist abhängig von der gewählten Art von Hüllgeometrien.
- Der verwendeten Suchstrategie: In welcher Reihenfolge werden Knoten auf derselben Ebene der Hierarchien durchlaufen?

- Bedingt durch die Tiefe und den Verzweigungsgrad der Hüllkörper-Hierarchien: Welcher Grad an Parallelität ist bei der Traversierung potentiell möglich?

Bedingt durch die Restriktionen, die die baum-basierte Struktur von Hüllkörper-Hierarchien jedem verwendeten Suchverfahren aufzwingt, ist der Zeitaufwand für die Traversierung von Hüllkörper-Hierarchien nur schwer zu kontrollieren beziehungsweise im Voraus abzuschätzen: Der Verzweigungsgrad einer Hüllkörper-Hierarchie schränkt so beispielsweise ein, wie viele Elemente der nächst tieferen Ebene in der Hierarchie parallel auf Überlappung geprüft werden können. Als weitere schwer vorhersagbare Größe sei noch die Problematik genannt, dass die tatsächlich erreichte Tiefe beim Durchlaufen einer Hüllkörper-Hierarchie (und damit die Anzahl durchzuführender Überlappungs-Tests) nicht abgeschätzt werden kann; dies kann im Extremfall dazu führen, dass ein Objektpaar-Test vorzeitig abgebrochen wird, wenn die Kollisionserkennung strengen Laufzeitschranken Rechnung tragen muss.

Zwei Gesichtspunkte sind bei der Betrachtung von Hüllkörperhierarchien besonders wichtig: Die Konstruktionsweise, und die Art der Traversierung bei der Suche nach möglicherweise überlappenden Teilen von Hierarchien.

3.4.3. Konstruktion von Hüllkörper-Hierarchien

Für die Konstruktion von Hüllkörper-Hierarchien kommen drei verschiedene Ansätze in Frage (Abbildung 3.5):

- Der Top-Down-Ansatz: Hier wird zuerst ein Hüllkörper für die gesamte Geometrie berechnet, der dann sukzessive in kleinere Hüllkörper unterteilt wird, bis ein definiertes Abbruch-Kriterium (beispielsweise ein minimales Volumen) erreicht, oder jede Seitenfläche der umschlossenen Geometrie mit einem Hüllkörper versehen ist.
- Der Bottom-Up-Ansatz: Zuerst wird für jede Seitenfläche der umschlossenen Geometrie ein Hüllkörper berechnet. Diese Hüllkörper werden solange zu weiteren Hüllkörpern kombiniert, bis die gesamte Eingangsgeometrie umschlossen wird.
- Die Insertion-Methode: Ein Hüllvolumen für jede Seitenfläche wird sukzessive in die Hierarchie eingefügt; diese wird nach dem Einfügen neu berechnet oder adaptiert.

3.4.3.1. Der Top-Down-Ansatz

Diese Methode wird rekursiv angewandt; nach der Aufteilung eines Hüllvolumens wird die Berechnung der Hierarchie für die neu entstandenen Sub-Volumina auf dieselbe Weise fortgesetzt. Die Aufteilung erfolgt durch die Partitionierung des betrachteten Volumens anhand einer Hyperebene (Gerade in 2D, Ebene in 3D); die Menge der verbleibenden Primitive der Eingabegeometrie werden nach Zugehörigkeit zu den Halbräumen aufgeteilt. Die zu berücksichtigenden Einflussgrößen beim Top-Down-Verfahren sind:

- Die Partitionierungs-Strategie (Median, Minimierung des Volumens der Kind-Hüllkörper, Minimierung der Überlappung der Kind-Hüllkörper)
- Die Abbruch-Kriterien (Anzahl umschlossener Primitive, minimales Volumen der Kind-Hüllkörper, Tiefe der Hierarchie)

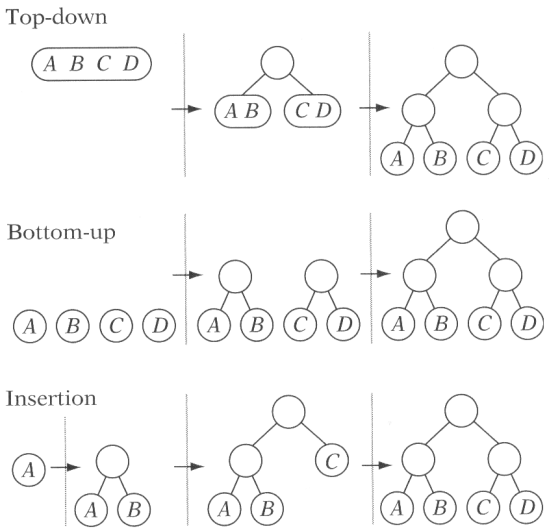


Abbildung 3.5.: Verschiedene Konstruktionsverfahren für Hüllkörper-Hierarchien ([Eri05]): Die Bottom-up-Methode ist zwar das einfachste Verfahren, kann jedoch zur Erzeugung unnötig großer Hüllkörper in höheren Hierarchie-Ebenen führen. Die Top-Down-Methode erzeugt besser angepasste Hüllkörper, erfordert jedoch höheren Aufwand bei der Aufteilung der verbleibenden Geometrie. Die Insertion-Methode ist als einziges Verfahren zur Anwendung auf sich verändernde Geometrien geeignet, ohne dass eine komplette Neuberechnung einer Hierarchie erforderlich wird.

- Die Auswahl der partitionierenden Hyper-Ebene und die Zuordnung verbleibender Primitive

3.4.3.2. Der Bottom-up-Absatz

Der Bottom-Up-Ansatz ([Omo89], [GSB99]) ist im Gegensatz dazu eine iterative Methode; hier ist die Wahl des Paares aus Knoten, die in einer Iteration durch einen neuen Eltern-Knoten kombiniert werden sollen (anhand eines möglichst geringen Volumens des neuen Eltern-Knotens), das wichtigste Auswahlkriterium. Neben einer erschöpfenden Suche durch die Berechnung möglicher Volumina mit allen verfügbaren anderen Knoten sind auch verschiedene optimierende Verfahren vorgeschlagen worden, die graphentheoretische Hilfsmittel wie die Berechnung eines minimalen Spannbaums verwenden, um mögliche Gruppierungen von Objekten in neuen Eltern-Knoten zu bestimmen.

3.4.3.3. Die Insertion-Methode

Bei der Insertion-Methode ([GS87]) werden neue Elemente einer Hüllkörper-Hierarchie durch sukzessives Einfügen in einen anfangs leere Hierarchie berechnet; die Positionierung innerhalb der bestehenden Hierarchie erfolgt üblicherweise so, dass das Volumen der Hierarchie durch das Einfügen eines neuen Elements möglichst wenig wächst. Eine Konsequenz hiervon ist, dass räumlich ausgedehntere Teile einer Eingangsgeometrie näher an der Wurzel der berechneten Hierarchie eingefügt werden, während kleinere Elemente tiefer in der Hierarchie liegen. Die Insertion-Methode ist für sich verändernde Geometrien besser geeignet als die beiden anderen Methoden: Eine Hüllkörper-Hierarchie kann mit ihrer Hilfe bei Änderungen

in umschlossenen Geometrien dynamisch neu berechnet werden, während bei den beiden anderen angesprochenen Methoden eine komplette Neuberechnung erforderlich ist.

3.4.4. Traversierung von Hüllkörper-Hierarchien

Bei der Traversierung von Hüllkörper-Hierarchien sind zwei Kriterien zu berücksichtigen:

- Die Wahl der Suchstrategie: Tiefen-, Breiten, oder Best-First-Suche
- Die Methode beim Absteigen durch die Hierarchien:
  - Eine Hierarchie vor der anderen
  - Simultaner Abstieg
  - Abwechselnder Abstieg
  - Selektion anhand des umschlossenen Volumens

3.4.4.1. Suchstrategien

Tiefen-Suche Diese Suchstrategie ist, in Kombination mit einer Such-Heuristik (angedeutet in Algorithmus 2) zur Optimierung der Abstiegsreihenfolge, die am häufigsten eingesetzte Strategie zur Traversierung von Hüllkörper-Hierarchien.

Funktion `DescendA`

Input : t, u

```
1 // 1. Hierarchie bevorzugt
2 return !IsLeaf(t);
3 // 2. Hierarchie bevorzugt
4 return IsLeaf(u);
5 // Bevorzuge Hierarchie mit größerem Volumen des obersten Hüllkörpers
6 return (!IsLeaf(b) ∨ (!IsLeaf(t) ∧ (SizeOfBV(t) ≧ SizeOfBV(u)));
```

Breiten- und Best-First-Suche Eine reine Breiten-Suche ist im Hinblick auf Hüllkörperhierarchien im Nachteil gegenüber einer Tiefen-Suche, da sie gegenüber dieser einen hohen Speicherplatzbedarf für einen Stapelspeicher aufweist, der zur Traversierung einer Hüllkörper-Hierarchie mittels dieses Suchverfahrens nötig ist. Eine Anwendung für die Breiten-Suche besteht allerdings in Kollisionserkennungs-Systemen, die zu beliebigen Zeitpunkten einer Simulations-Iteration in der Lage sein müssen, Informationen über mögliche Kollisionen zu bestimmen. Hier ist die Breiten-Suche eine gute Alternative, um die Zeit für die Suche nach möglichen Kollisionen gleichmäßig über Teile von Hüllkörper-Hierarchien zu verteilen, anders als bei der Tiefen-Suche.

Die Best-First-Suche versucht, die Reihenfolge besuchter Knoten anhand eines bestimmten Ziel-Kriteriums zu ordnen; da für Hüllkörper-Hierarchien ein solches Kriterium nicht in Form eines bestimmten Ziel-Knotens angegeben werden kann, sondern höchstens in Form einer Heuristik während der Traversierung besteht (z. B. erst Hierarchie-Teilen mit größerem

Volumen traversieren), ist die Erweiterung der Tiefen-Suche mit Heuristiken dieser Art die übliche Vorgehensweise. Die Best-First-Suche ist im Gegensatz zur Tiefen- und Breiten-Suche ein *informiertes Suchverfahren*, das sich nicht ausschließlich an der Struktur eines Baums orientiert, sondern Auswahlkriterien zur Festlegung des Verlaufs einer Suche verwendet.

3.4.4.2. Abstiegs-Regeln

Abstiegs-Regeln bestimmen, in welcher Reihenfolge beziehungsweise mit welcher Priorität bei einer paarweisen Überprüfung von Hüllkörper-Hierarchien Teile einer oder beider beteiligten Hierarchien auf mögliche Überschneidungen überprüft werden. Wird jeweils eine der Hierarchien bevorzugt traversiert, so kann der Aufwand der Traversierung sehr hoch werden, sofern eine der Hierarchien den Wurzel-Knoten der anderen komplett schneidet, beziehungsweise die Hierarchie sehr tief ist. Wird die Abstiegsreihenfolge etwa durch den Vergleich der verbleibenden Volumina von Hüllkörpern oder die Größe des Schnittvolumens zwischen Hüllkörpern bestimmt, wird dieser Aufwand vermindert: Diese Vorgehensweise gleicht die Nachteile einer blinden Suche aus, die nur die Baum-Struktur selbst berücksichtigt. Sind Volumen-Berechnungen eine aufwendige Operation, so kann alternativ eine simultane Traversierung verwendet werden, die rekursiv immer Knoten aus derselben Tiefe beider betrachteter Hierarchien überprüft. Dies stellt einen Kompromiss zwischen den prioritäts-basierten Abstiegsregeln, bei denen versucht wird, das verbleibende zu überprüfende Volumen schnell zu minimieren, und den blinden Abstiegs-Regeln dar, die sich nur an den Hierarchie-Strukturen orientieren.

3.4.5. Verfahren der Vorfilter-Phase

Repräsentativ sollen nun in kurzer Form zwei Verfahren betrachtet werden, die für die szenenweite Berechnung von Kollisionen zwischen Objekten eingesetzt werden können: Eines der Verfahren verwendet eine inkrementell konstruierte AABB-Hierarchie, das zweite beruht auf der Sortierung der Intervalle von AABBs entlang der Achsen eines Bezugs-Koordinatensystems.

3.4.5.1. Dynamischer AABB-Baum

Dynamische AABB-Bäume werden in der Bullet Physics Engine ([Cou12]) in der Vorfilter-Phase verwendet; hier werden zwei unterschiedliche AABB-Bäume für bewegte und unbewegte Objekte benutzt. Das erlaubt die parallele Überprüfung zwischen bewegten sowie bewegten und unbewegten Objekten. Die Implementierung nutzt als zusätzliche Optimierung die Möglichkeit, die AABB-Bäume zu rebalancieren, sofern sich eine ungleiche Verteilung von Kind-Knoten in verschiedenen Teilen der Hierarchie ergeben sollte. Werden AABBs aufgrund von Objektbewegungen neu positioniert, so erfolgt eine Adaption der AABB-Hierarchie nur für den Fall, dass die Repositionierung außerhalb der bisherigen Eltern-AABB erfolgt. Als weitere Maßnahme zur Kompensation von numerischen Toleranzen werden AABBs um einen Toleranzfaktor erweitert; daneben bieten diese dynamischen Bäume die Gelegenheit, Objektbewegungen durch Extrusionen von AABBs zu berücksichtigen, indem das Volumen einer AABB unter Berücksichtigung der momentanen Eigenbewegung erweitert wird.

3.4.5.2. Sweep and Prune (SAP)

Das Sweep and Prune-Verfahren ([Bar92], [CLMP95]) beruht auf dem Ansatz, eine sortierte Liste der Achsenabschnitte aller AABBs in einer Simulation entlang aller drei räumlichen Dimensionen zu führen. Um mögliche Kollisionen zwischen AABBs zu ermitteln, genügt nun ein Iterieren durch die Intervall-Listen entlang aller drei Achsen: Eine zu überprüfende AABB kann mittels einer binären Suche schnell mit anderen AABBs abgeglichen werden. Dieses Verfahren ist generell sehr stabil gegenüber Objektbewegungen über wenige Iterationen der Simulation hinweg: Die Sortierung der Liste bleibt über kurze Zeit größtenteils erhalten, sofern die Simulation nicht sich sehr schnell bewegende Objekte enthält. Ein Nachteil des Verfahrens ist jedoch die Tatsache, dass eng benachbarte Objektgruppen unter Umständen zu häufigen Neu-Sortierungen entlang einer oder mehrerer Intervall-Listen führen können, da in diesem Fall selbst geringe Eigenbewegungen zu großen Veränderungen in Intervall-Reihenfolgen führen können.

3.5. Die Objektpaar-Phase (Narrow-Phase)

Die Objektpaar-Phase dient dazu, die exakte Anzahl und Lage von Kontaktpunkten zwischen Objektpaaren zu bestimmen, die als Eingangsdaten zur Etablierung und Aufrechterhaltung von Kontakten in der mechanischen Modellierung der Szene dienen. Dazu werden alle Objektpaare, die in der Vorfilter-Phase als potentiell kollidierend markiert worden sind, anhand ihrer detaillierten Modelle auf Kontakte zwischen ihren Bestandteilen (Eckpunkten, Kanten und Seitenflächen) berechnet.

Narrow-Phase-Verfahren werden zum einen dadurch unterschieden, ob sie in der Lage sind, die Bewegung von Objekten zu berücksichtigen (sogenannte “Continuous Collision Detection”-Verfahren (CCD)), oder ob sie nur in der Lage sind, statische Objektkonfigurationen ohne Objektbewegung zu verarbeiten.

Ein weiteres Unterscheidungsmerkmal besteht darin, ob ein Verfahren direkt auf den topologischen Daten der verwendeten Objektgeometrie operiert, oder ob wiederum vereinfachende Geometrien als temporäre oder vorberechnete Hilfskonstrukte verwendet werden, um die Verarbeitungsgeschwindigkeit des Verfahrens zu erhöhen.

Da Objektpaar-Tests üblicherweise nur auf sehr kleinen Elementen eines Modells operieren, ist der tatsächliche Berechnungsaufwand für die Bestimmung von Kontaktpunkten zwischen einzelnen Paaren (oder kleinen Gruppen) von Seitenflächen vergleichsweise gering, sofern die Eigenbewegung der betroffenen Modelle nicht berücksichtigt werden muss und nur eine statische Momentaufnahme der Szene auf Kollisionen berechnet werden soll. Vielmehr ist die Gesamtzahl tatsächlich durchgeführter Paartests für das Laufzeitverhalten der Objektpaar-Phase entscheidend. Ein wichtiges Kriterium ist daher die Vermeidung unnötiger Paartests durch ein vorhergehendes Broadphase-Verfahren, ebenso wie die Möglichkeit, einen unnötig durchgeführten Paartest bereits frühzeitig abbrechen zu können.

An dieser Stelle soll nur kurz auf den Schnitt-Test zwischen Paaren von Dreiecken eingegangen werden; eine ausführlichere Diskussion einer möglichen Implementierung eines solchen Tests wird in Abschnitt B.2 diskutiert. Ebenso wird dort eine weitere Variante eines auf

die Überprüfung konvexer Geometrien spezialisierten Algorithmus für die Objektpaar-Phase erläutert: Der GJK-Algorithmus.

3.5.1. Schnitt-Test zwischen Paaren von Seitenflächen

Direkt auf der topologischen Beschreibungen operierende Paartest-Verfahren verwenden Ecken, Kanten und Seitenflächen eines geometrischen Modells, um Berührungs- oder Schnittpunkte zwischen Paaren von Seitenflächen verschiedener Geometriemodelle zu bestimmen. Die übliche Konfiguration sind hierbei Paare von Dreiecken als Elemente von Dreiecksnetzen.

Für die Berechnung möglicher Kontaktpunkte zwischen einem Paar von Dreiecken sind unterschiedliche Verfahren vorgeschlagen worden: Eine Alternative ist die Verwendung von insgesamt elf Separating-Axis-Tests, je einer in Richtung der Dreiecks-Normalen, und neun weitere für die möglichen Kreuzprodukte der Dreiecks-Kanten.

Eine weitere Alternative ist eine direkte Schnittberechnung zwischen Dreieck und Geradenabschnitt für alle möglichen Kombinationen aus den Kanten und einem Dreieckspaar; dieser Test ist allerdings nicht robust gegenüber koplanaren Dreiecken.

[Möl97], [Hel97] und [DG02] jedoch schlagen effizientere Tests vor, die nur die beiden Normalen als mögliche separierende Achsen testen, danach jedoch überprüfen, ob jeweils alle drei Punkte beider Dreiecke im selben Halbraum relativ zu den jeweiligen von den Dreiecken aufgespannten Ebenen liegen. Ist dies der Fall, so berühren sich die Dreiecke nicht. Andernfalls wird der resultierende Geradenabschnitt als Schnitt der beiden Ebenen berechnet und geprüft, ob dieser Geradenabschnitt innerhalb eines der Dreiecke zum Liegen kommt, entweder direkt als Schnitt zwischen einem Dreieck und einem Geradenabschnitt, oder über Projektion der Schnitt-Intervalle auf eine der Koordinaten-Achsen des Bezugs-Koordinatensystems.

3.6. Schnittstelle zwischen Kollisionserkennung und -behandlung: Kontakt-Punkte

Die Kollisionserkennung ist eine Aufgabe, die mittels geometrischer Berechnungen unter Zuhilfenahme der linearen Algebra bewältigt wird. Im Gegensatz dazu verwendet die Kollisionsbehandlung die Gesetze der Newton'schen Mechanik, um das mechanische Verhalten simulierter Objekte zu bestimmen.

Das Kollisionserkennung liefert die geometrischen Eingangsgrößen für die Berechnung von Kontaktkräften beziehungsweise -impulsen, die als Reaktion auf auftretende Kontakte zwischen Objekten von der Kollisionsbehandlung bestimmt werden. Diese geometrischen Größen bestehen üblicherweise aus einer Liste von punktförmigen Kontakten; zu jedem bestimmten Kontaktpunkt zwischen zwei Objekten gehören (Abbildung 3.6):

- Die Koordinaten des Kontaktpunkts selbst
- Sofern sich beide Geometrien überschneiden: Die Eindringtiefe am Kontaktpunkt
- Die Kontakt-Normale, entgegen derer die Reaktion auf den Kontakt zwischen den Objekten erfolgt

Im Folgenden wird von polygonalen Geometrien ausgegangen, deren Seitenflächen konvex sind.

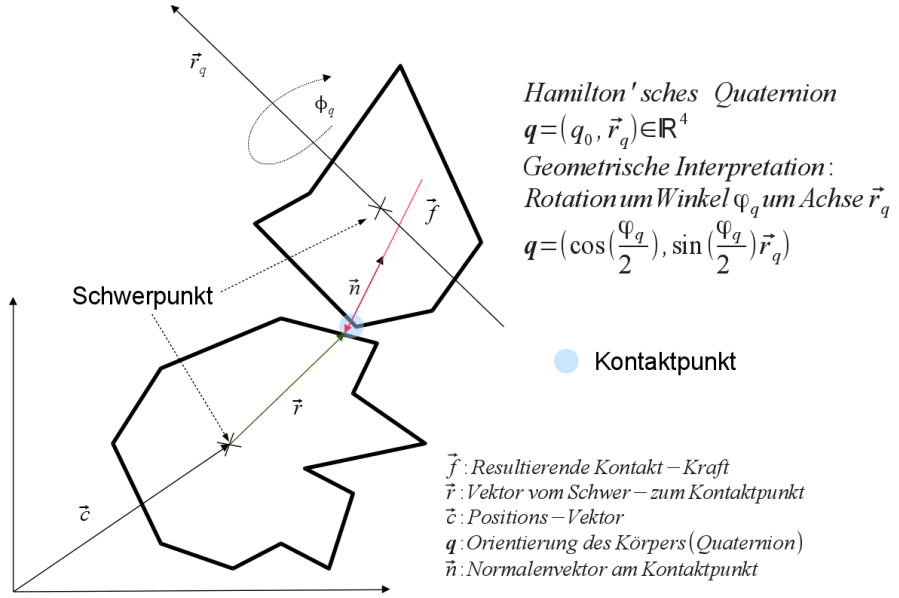


Abbildung 3.6.: Beteiligte Größen an Kontaktpunkten zwischen zwei Starrkörpern

Die Approximation von Flächen-Kontakten durch einzelne Kontaktpunkte erscheint zunächst als eine sehr weitgehende Vereinfachung. Der Argumentation aus [Buc99, Kapitel 8] folgend sei jedoch angemerkt, dass reale Oberflächen auf mikroskopischer Ebene keinesfalls planar aufeinander liegen, sondern durch eine große Anzahl „punktförmiger“ Kontakte auf molekularer beziehungsweise atomarer Ebene interagieren. Daher ist es nachvollziehbar, dass eine solche Vereinfachung für temporal und numerisch diskrete Starrkörpersimulationen durchaus eine gangbare Alternative darstellt (wie in Abbildung 3.7 angedeutet).

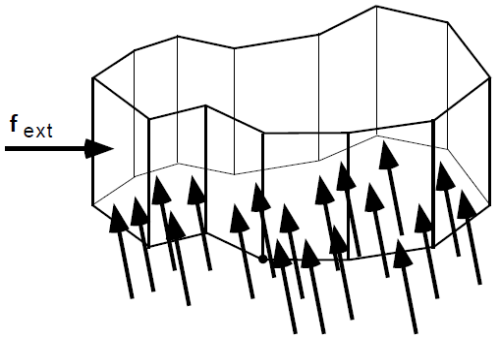


Abbildung 3.7.: Approximation von Flächen-Kontakten durch Punkt-Kontakte ([Buc99, Kapitel 8]): Ein flächenhafter Kontakt zwischen zwei Objekten wird durch eine Menge einzelner Kontaktpunkte ersetzt.

Obwohl sich [Buc99] mit seinen Untersuchungen und der darauf aufbauenden Argumentation auf das Reibungsverhalten von Objekten konzentriert (siehe auch Unterabschnitt A.2.5), ist diese Argumentation ebenso eine gute Rechtfertigung für die Verwendung punktförmiger Kontakte als Approximation für Flächen-Kontakte.

Folgende Fragen sind bei der Nutzung punktförmiger Kontakte bei polygonaler Geometrien in Kollisionserkennungs-Algorithmen zu berücksichtigen:

- Welche Art von Kontakt-Konfigurationen können prinzipiell zwischen polygonalen Geometrien auftreten?
- Wie können diese Kontakt-Konfigurationen beschrieben werden?
- Wie müssen solche Kontakt-Konfigurationen beschaffen sein, um redundante Kontaktpunkte vermeiden zu können?

Eine ausführlichere Diskussion der Verwendung und möglicher Umsetzungen punktförmiger Kontkte zwischen polygonalen Geometrien ist in Abschnitt A.3 als Teil von Anhang A zu finden. Die folgenden Ausführungen sollen anhand der Diskussion dreier Alternativen zur Auswahl von Kontaktpunkten zur Beschreibung von Kontaktkonfigurationen zwischen polygonalen Objekten die besondere Wichtigkeit der Resultate der Kollisionserkennung für die Stabilität und Plausibilität des mechanischen Verhaltens einer Starrkörpersimulation verdeutlichen.

3.7. Minimale Kontaktpunkt-Konfigurationen

Die Verwaltung und Verfolgung minimaler Kontaktpunkt-Konfigurationen zur Laufzeit einer Mechaniksimulation ist von großer Wichtigkeit für die Stabilität der Simulation selbst (Vermeidung von Durchdringungen) und die Plausibilität des von der Kollisionsbehandlung erzeugten mechanischen Verhaltens (möglichst vollständige Erkennung aller Kontaktregionen) simulierter Objekte.

3.7.1. Relevanz von Kontaktpunkt-Konfigurationen für die Stabilität einer Simulation

Wie sich die Verwaltung von Kontaktpunkt-Konfigurationen, also etwa

- die Auswahl von Kontaktpunkten, die in der Kollisionsbehandlung verwendet werden sollen
- die Aufrechterhaltung bestehender Kontakt-Bedingungen
- das Lösen von Kontakten

auf die Stabilität einer Simulation auswirken können, lässt sich mit Hilfe eines Beispiels, das im Rahmen des in Abschnitt 2.1 beschriebenen Industrieprojekts untersucht worden ist, aufzeigen: Das Test-Szenario bestand aus einem einfachen Versuchsaufbau, in dem eine (vereinfacht modellierte) Schraube nur unter Einfluss der Schwerkraft in eine Verschraubungs-Öffnung eines zweiwändigen Lagerbocks fiel; Ziel dieses Versuchs war, die Stabilität der Kontaktbehandlung bei umschließenden Kontakten zu überprüfen.

Wie in Abbildung 3.8 zu sehen, ist die Kollisionsbehandlung der Bullet Physics Engine nicht in der Lage, unter diesen Bedingungen eine stabile Kontaktbehandlung zu gewährleisten: Anstatt nach Erreichen einer stabilen Ruhelage dauerhaft in dieser zu verbleiben, verursacht die Kollisionsbehandlung im Bereich der umschließenden Kontakte in den Öffnungen des Lagerbocks ein instabiles Verhalten der Schraube, das im weiteren Verlauf der Simulation sogar dazu führt, dass sich die beiden Objekte überschneiden.

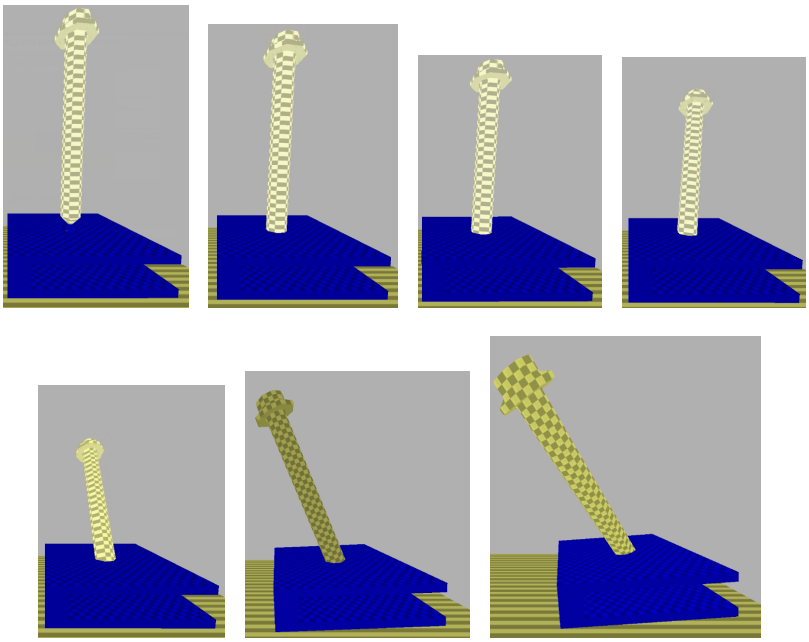


Abbildung 3.8.: Verlauf des Schraube-Lagerbock-Tests mit der Bullet Physics Engine: Der Kollisionsbehandlung gelingt es nicht, im späteren Verlauf des Versuchs eine stabile Ruhelage der Schraube in der Verschraubungs-Öffnung des Lagerbocks zu etablieren. Die Geometrien beginnen, sich gegenseitig zu durchdringen, und die Schraube beginnt zu kippen.

Die Ursache für ein derartiges Verhalten einer Simulation ist allerdings nicht ausschließlich die Kollisionsbehandlung. Problematisch ist im Fall der Bullet Physics Engine auch, dass die Kollisionserkennung nach Überschreiten einer bestimmten Anzahl von erkannten Kontakten zwischen einem Paar von Geometrien keine weiteren Kontaktpunkte mehr an die Kollisionsbehandlung weitergibt. Dadurch kann es dazu kommen, dass eine mechanische Reaktion an einem Teil einer Geometrie zwar korrekterweise Überschneidungen verhindert, es dafür an anderen Stellen der Geometrie dafür zu neuen Durchdringungen kommt, oder sich bereits bestehende Durchdringungen verschlimmern. Abbildung 3.9 skizziert dieses Problem für das Schraube-Lagerbock-Experiment aus Abbildung 3.8: Die Ausgangssituation in Abbildung 3.9a führt durch die Auflösung eines der Kontaktpunkte dazu, dass die Schraube in die entgegengesetzte Richtung ausweicht, was im weiteren Verlauf in Abbildung 3.9b zu einer neuen Durchdringung führt. Die mechanische Reaktion auf diese führt nun dazu, dass die Schraube zu kippen bedingt, was in der Folge zu immer schwerwiegenderen wechselseitigen Durchdringungen führt (Abbildung 3.9c). Schließlich kann durch diese Wechselwirkungen instabiles Verhalten wie das in Abbildung 3.8 gezeigte auftreten.

Dagegen ist die Simulations-Software Veo, die das von Buck vorgeschlagene Verfahren zur Verwaltung von Kontaktpunkten verwendet, zur Etablierung einer stabilen Ruhelage der im Lagerbock steckenden Schraube in der Lage, wie Abbildung 3.10 verdeutlicht.

Die Herangehensweise von Buck beruht darauf, dass pro Iteration der Simulation jeweils nur der erste neu erkannte Kontaktpunkt an die Kollisionsbehandlung weitergegeben wird. Kann keine Lösung des zugrunde liegenden Gleichungssystems zur Bestimmung mechanischer Reaktionen gefunden werden, so wird der zuletzt erkannte Kontaktpunkt wieder aus der Liste registrierter Kontakte entfernt, und die Simulation auf den letzten bekannten konstisten-

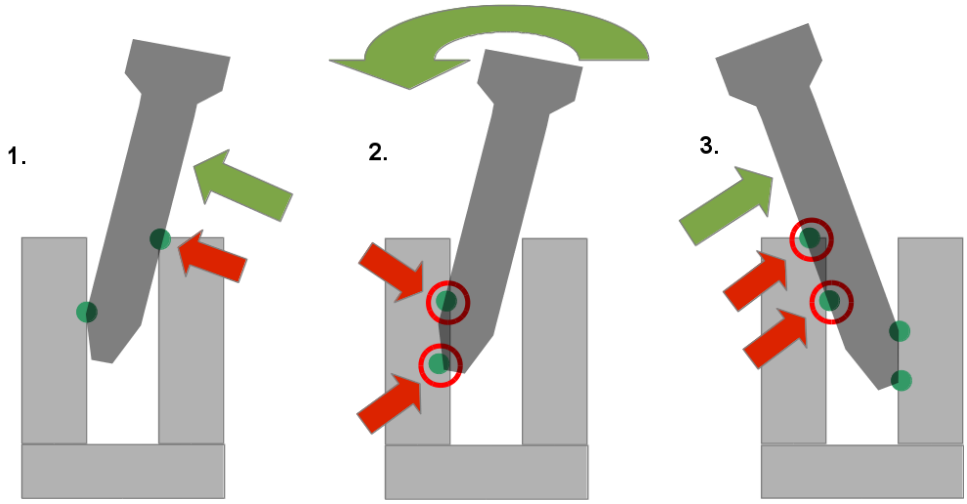


Abbildung 3.9.: Exemplarischer Verlauf einer umschließenden Kontakt-Konfiguration (Kontakt-Impulse als rote, die resultierende Bewegungsrichtung des Objekts als grüne Pfeile): Diese exemplarische Sequenz von Kollisionsreaktionen führt zu einer Wechselwirkung zwischen unterschiedlichen Kontaktpunkt-Regionen, die bei der Korrektur bestehender Durchdringungen zu neuen Überschneidungen entgegengesetzt zur Bewegungsrichtung führt.

ten Zustand zurückgesetzt, indem die Objektbewegungen der letzten Iteration rückgängig gemacht werden.

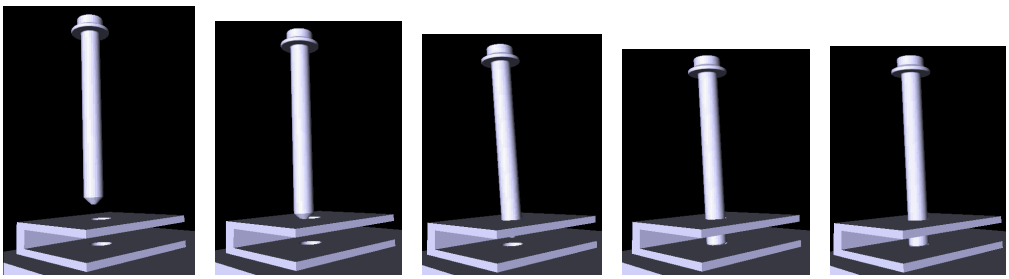


Abbildung 3.10.: Verlauf des Schraube-Lagerbock-Tests mit der Mechaniksimulation Veo ([Buc99], [Eck98]): Die hier verwendete Methode zur Kontaktbehandlung konstruiert explizit in jeder Iteration eine minimale Menge von Kontaktpunkten, unter anderem indem nach erfolgter Kollisionserkennung nur ein neuer Kontaktpunkt je Iteration registriert und durch die Kollisionsbehandlung die numerische Lösbarkeit des zugrunde liegenden Gleichungssystems sichergestellt wird. Ist diese nicht gegeben, so wird die Simulation auf den Zustand der vorigen Iteration zurückgesetzt.

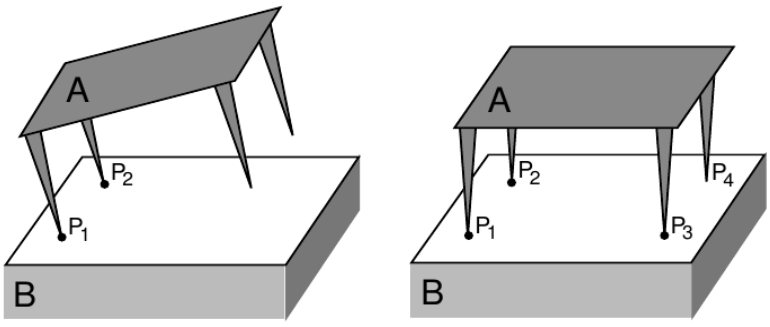
Die folgenden Abschnitte gehen daher nochmals auf unterschiedliche Möglichkeiten zur Behandlung von Kontaktpunkten ein, und zwar anhand verschiedener Alternativen, die in folgenden Mechaniksimulationen zum Einsatz kommen:

- Wie von Buck [Buc99] und Eckstein [Eck98] vorgeschlagen
- Die Kontaktpunkt-Erzeugung in der Bullet Physics Engine [Cou12]
- Alternative Kontaktpunkt-Modelle aus dem SOFA-Framework [INR13]

3.7.2. Von Buck vorgeschlagener Ansatz

Buck [Buc99, Kapitel 5] und Eckstein [Eck98] verfolgen den Ansatz, je Iteration einer Simulation nur jeweils einen einzelnen neu erkannten Kontaktpunkt zwischen zwei Objekten zur Menge bestehender Kontaktpunkte hinzuzufügen. Begründet wird dieser Ansatz wie folgt: Der erste Kontakt zwischen einem Paar polygonaler Objekte erfolgt im Allgemeinen an einem einzelnen Kontaktpunkt. Um an diesem ersten Kontaktpunkt eine Durchdringung der Objekte zu verhindern, muss eine mechanische Reaktion in Form einer Kontaktkraft erfolgen, und es ergibt sich eine neue, nicht-redundante Bewegungsbeschränkung (Gleichung A.12) für jedes der beteiligten Objekte. Es wird von Buck weiter argumentiert, dass das Auftreten eines neuen Kontaktpunktes selbst als Indiz dafür betrachtet wird, dass durch ihn eine neue, nicht-redundante Bewegungsbeschränkung entsteht.

Durch die strikte Einschränkung bei der Auswahl neu entstehender Kontaktpunkte auf einen einzelnen Kontaktpunkt pro Iteration wird gewährleistet, dass alle neu entstehenden Bewegungsbeschränkungen, die neben dem ersten Kontakt zwischen Objekten entstehen können, redundante Bewegungsbeschränkungen sind, durch die entsprechende mechanische Reaktionen zur Vermeidung von Objekt-Durchdringungen erfolgen. Werden bei der Kollisionserkennung mehr als sechs Kontaktpunkte für ein Objektpaar ermittelt, so werden nur die ersten (beziehungsweise, abhängig von der wechselseitigen Objektbewegung, maximal) sechs registrierten Kontaktpunkte für die Berechnung der mechanischen Reaktion verwendet. An weiteren eventuell registrierten Kontaktpunkten wird trotz zutreffender Kontaktbedingung keine weitere Bewegungsbeschränkung angesetzt. Abbildung 3.11 zeigt ein Beispiel für eine so gefundene Kontaktpunkt-Konfiguration: Obwohl an Punkt P_4 in Abbildung 3.11b eine Kontaktbedingung erfüllt wäre, wird dieser Kontaktpunkt nicht weiter bei der Bestimmung der mechanischen Reaktion berücksichtigt. Eine Ruhelage des Objekts würde sich im weiteren Verlauf der Simulation durch die alternierende Entstehung und Entfernung von Bewegungsbeschränkungen beispielsweise an P_1 und P_4 ergeben.



(a) Ausgangssituation **(b)** Minimale Konfiguration

Abbildung 3.11.: Erzeugung einer minimalen, stabilen Kontaktpunkt-Konfiguration (nach [Buc99]): Das gezeigte Objekt hat bei Erreichen einer stabilen Kontakt-Konfiguration noch einen Rotations- und zwei Translations-Freiheitsgrade. Zur Aufrechterhaltung dieser Kontaktkonfiguration sind nur drei Kontaktpunkte nötig, weswegen bei P_4 kein Kontaktpunkt entsteht. In weiteren Iterationen kann es in einer solchen Situation bedingt durch numerischen Ungenauigkeiten dazu kommen, dass alternierend zwischen P_1 und P_4 Kontaktpunkte entstehen beziehungsweise entfernt werden.

Diese Vorgehensweise hat den wesentlichen Vorteil, minimale Kontaktpunkt-Konfigurationen explizit konstruieren zu können und die Einführung nicht-redundanter Kontaktbedingungen in die Berechnung des mechanischen Verhaltens simulierter Objekte gezielt zu vermeiden.

Andererseits wird vom Autor selbst auf einen Nachteil dieser Methode verwiesen: Es wird die Verfügbarkeit einer ausreichend performanten Kollisionserkennungs-Komponente angenommen. Durch die Beschränkung auf maximal einen neuen Kontaktpunkt pro Iteration ist es nötig, eine höhere Anzahl an Iterationen in der Kollisionserkennung bewältigen zu können, da nach Erzeugung einer neuen Kontaktbedingung und der gegebenenfalls dadurch entstehenden Veränderungen im Bewegungsverhalten von Objekten die Szene erneut auf mögliche Kollisionen überprüft werden muss.

3.7.3. Ansätze in der Bullet Physics Engine

Coumans ([Cou11], [Cou10]) benutzt zur Verwaltung von Kontaktpunkt-Konfigurationen in der Bullet Physics Engine ([Cou12]) Methoden, die die gleichzeitige Entstehung mehrerer neuer Kontaktpunkte zwischen Objekten ermöglicht.

Die erste Methode nutzt Objektpaaren zugeordnete Zwischenspeicher, die innerhalb eines bestimmten Toleranzbereiches bestehende Kontaktpunkte erst entfernen, wenn deren Abstand zum anderen Objekt außerhalb des Toleranzbereiches liegt. Die Entstehung neuer Kontaktpunkte basierend auf bereits im Zwischenspeicher enthaltenen Kontaktpunkten innerhalb des assoziierten Toleranzbereichs wird zusätzlich beschränkt.

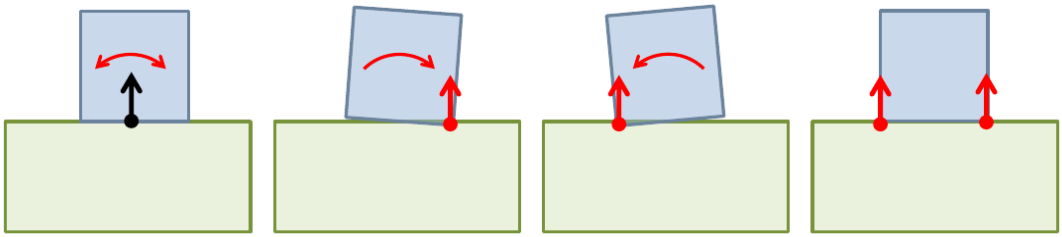


Abbildung 3.12.: Erzeugung von Kontaktpunkten durch die Variation einer Flächen-Normalen (nach [Cou10]): Zur Erzeugung weiterer möglicher Kontaktpunkte wird die räumliche Orientierung eines Objekts relativ zur Auflagefläche minimal variiert.

Abbildung 3.12 zeigt ein vereinfachtes Beispiel für eine der von Coumans beschriebenen Methoden, um mittels der Ausnutzung von Toleranzen, in diesem Fall der Variation einer Flächen-Normalen, mögliche neue Kontaktpunkte zwischen Geometrien zu erzeugen: Indem eines der Objekte geringfügig zur Flächen-Normalen der Seitenfläche des zweiten Objekts, auf der es aufliegt, rotiert wird, können mögliche Kandidaten für neue Kontaktpunkte ermittelt werden.

Die zweite Methode in Bullet benutzte Methode zur Verwaltung von Kontaktpunkten nutzt eine Polygon-Clipping-Methode (den Sutherland-Hodgman-Algorithmus, [SH74]), um Kontaktpunkte aus dem Schnitt der Kanten eines konvexen Polygons mit einem anderen (konvexen oder konkaven) Polygon zu generieren. Dabei werden die Eckpunkte des zugeschnittenen

Polygons sowie alle Schnittpunkte des zugeschnittenen Polygons mit den Kanten des Schnitt-Polygons als Kandidaten für Kontaktpunkte betrachtet. Ein Beispiel ist in Abbildung 3.13 dargestellt.

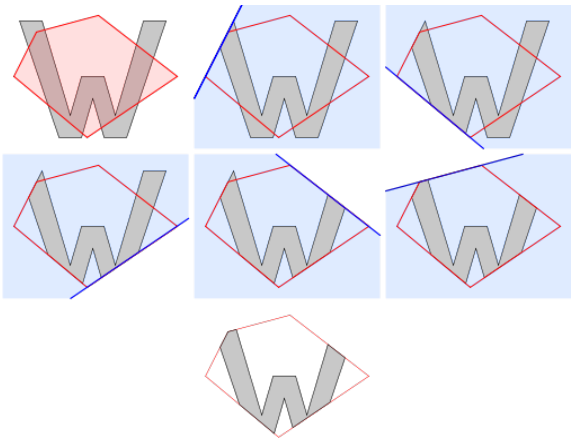


Abbildung 3.13.: Berechnung der Schnittmenge zweier Polygone mittels des Sutherland-Hodgman-Algorithmus (nach [Cou11]): Neue mögliche Kontaktpunkte resultieren aus den Schnittpunkten der Kanten der beiden Polygone.

Im Gegensatz zu der von Buck vorgeschlagenen Methode erzeugen die in der Bullet Physics Engine eingesetzten Verfahren meistens mehr Kontaktpunkt-Kandidaten, als zur Aufrechterhaltung von Bewegungsbeschränkungen prinzipiell nötig wären. Coumans schlägt daher einige Kriterien vor, die zur Beschränkung der tatsächlich in der Kollisionsbehandlung verwendeten Kontaktpunkte verwendet werden können, beispielsweise die Beschränkung der Anzahl von Kontaktpunkten pro Zwischenspeicher oder etwa die Beschränkung auf das Hinzufügen von Kontaktpunkten mit der größten Eindringtiefe.

3.7.4. Ansätze in SOFA und mit dessen Hilfe implementierter Verfahren

Die standardmäßig im SOFA-Framework für polygonale Kollisions-Geometrien verwendete Methode zur Verwaltung von Kontaktpunkten ([Fau09], [All07], [SDC08]) erstellt für jedes Paar aus Dreiecksnetz-Elementen, für das die Objektpaar-Phase einen Abstand unterhalb eines konfigurierbaren Toleranzbereichs eine mögliche Berührung meldet, einen entsprechenden Kontaktpunkt. Diese Vorgehensweise ist zwar nicht so restriktiv wie der von Buck verfolgte Ansatz, benötigt aber andererseits keine expliziten Maßnahmen zur Reduktion der Anzahl in der Kollisionsbehandlung verwendeter Kontaktpunkte, wie es bei den beschriebenen Vorgehensweisen in der Bullet Physics Engine der Fall ist. Die Überprüfung einzelner Dreiecksnetz-Elemente erfolgt dabei wie in Unterabschnitt B.2.2 beschrieben anhand Schnittberechnungen zwischen Paaren von Kanten und Kombinationen aus Ecken und Flächen einzelner Paare von Dreiecken.

Einen anderen Ansatz zur Kontaktpunkt-Verwaltung verfolgen Allard und Faure in [AFC+10] und [FBJF08]. Auf dieses Kollisionserkennungs-Verfahren selbst wird in Abschnitt B.3 genauer eingegangen. Diese Vorgehensweise verzichtet auf die explizite Bestimmung punktför-

miger Kontakte, und nutzt stattdessen direkt die Eckpunkten einer polygonalen Geometrie als Ansatzpunkte für mechanische Reaktionen.

An welchen Eckpunkten das Anwenden einer Kontaktkraft notwendig ist, wird durch die Berechnung des Schnitt-Volumens sich überschneidender Teile von Kollisionsgeometrien bestimmt (Abbildung 3.14). Proportional zum Schnitt-Volumen beziehungsweise der wechselseitigen Eindringtiefe eines Objektpaars werden an den Eckpunkten aller Seitenflächen, die sich innerhalb eines Schnitt-Volumens befinden, Kontaktkräfte entgegengesetzt zur Richtung des Eindringens angesetzt.

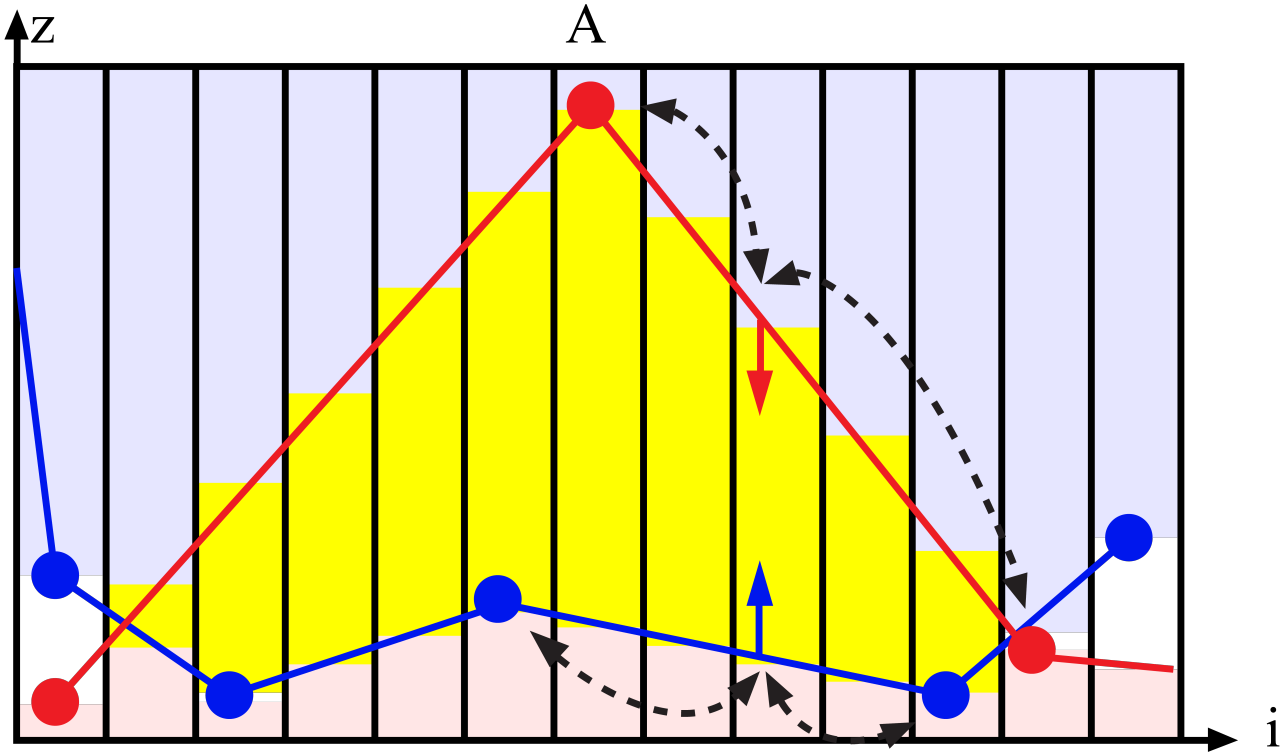


Abbildung 3.14.: Anwendung von Kontakt-Kräften proportional zum Schnitt-Volumen (gelb hinterlegt): Abhängig von der in diesem Verfahren explizit in Kauf genommenen Tiefe der wechselseitigen Durchdringung werden Kontaktkräfte bestimmt, die aus der volumetrischen Repräsentation proportional an umliegenden Eckpunkten der polygonalen Oberflächenbeschreibung angewandt werden. Das Verfahren selbst wird in Abschnitt B.3 diskutiert. (nach [FBJF08])

KAPITEL 4

ANALYSE EXISTIERENDER ANSÄTZE ZUR KOLLISIONSERKENNUNG

Kapitel 4 beschäftigt sich mit dem Stand der Technik bei Kollisionserkennungs-Verfahren.

Abschnitt 4.1 zeigt unterschiedliche Funktionsweisen von Kollisionserkennungs-Verfahren auf, wobei unterschiedliche Varianten von GPU-basierte Verfahren und deren Implementierungen im Vordergrund stehen.

Abschnitt 4.2 bis Abschnitt 4.4 erläutern anhand existierender Implementierungen die jeweilige Funktionsweise sowie die Vor- und Nachteile verschiedener Arten von Kollisionserkennungs-Verfahren.

4.1. Überblick

Die im Folgenden betrachteten Kollisionserkennungs-Verfahren in der Objektpaar-Phase lassen sich grundsätzlich wie folgt einordnen:

- *Polygon-basierte Verfahren*: Polygonale Oberflächenbeschreibung in Kombination mit daraus erstellten Hüllkörper-Hierarchien, wie in Abschnitt 3.4 und Abschnitt 3.5 vorgestellt.
- *Voxel-basierte Verfahren*: Diese verwenden ein über ein Objekt gelegtes, strukturiertes Gitter zusammen mit Punktwolken-Repräsentationen von Objekten. Hier werden für jedes Objekt zwei Datenstrukturen aus der ursprünglichen polygonalen Geometrie erzeugt: Eine strukturiertes Gitter als Partitionierung des Objektvolumens, und eine Punktwolke aus Punkten, die auf der Oberfläche der ursprünglichen polygonalen Geometrie liegen. Für den Kollisionstest wird die Punktwolke eines Objekts in das strukturierte Gitter des anderen Objekts eingeordnet und so bestimmt, ob und welche Teile der getesteten Geometrie in der Nähe der Oberfläche eines anderen Objektes liegen.

Bildraum-basierte Verfahren stellen eine weitere Klasse von Methoden zur Kollisionserkennungs-Verfahren dar, die mögliche Überschneidungen zwischen Objekten anhand aus verschiedenen Perspektiven erzeugten Ansichten ermitteln: Diese Verfahren arbeiten nicht auf Basis der eigentlichen Objektbeschreibungen, sondern bestimmen anhand aus verschiedenen Perspektiven durchgeführten Verdeckungs-Tests ein gegebenenfalls vorhandenes Schnitt-Volumen zwischen zwei Geometrien. Abschnitt B.3 in Anhang B beschäftigt sich mit zwei Implementierungen, die nach diesem Prinzip funktionieren. Eine ausführliche Diskussion bildraum-basierter Verfahren wird an dieser Stelle dagegen nicht erfolgen, da die in Abschnitt B.3 ausführlicher beschriebenen Nachteile dieser Verfahren in Widerspruch zu den in Abschnitt 3.1 formulierten Anforderungen für die Implementierung eines eigenen Verfahrens für den Einsatz in robotik-orientierten Anwendungen für Starrkörpersimulationen stehen.

Die für dieses und die folgenden Kapitel getroffene Auswahl von Kollisionserkennungsverfahren ist eine bewusste Eingrenzung auf einen der wichtigsten Aspekte in Starrkörpersimulationen in der Industrie- und Service-Robotik: Die performante Kollisionserkennung für konkave, bewegte Dreiecksnetze.

Die in den folgenden Abschnitten angesprochenen Verfahren sind gut dazu geeignet, die Stärken und Schwächen verschiedener Herangehensweisen an das Problem der Kollisionserkennung zu illustrieren. Auch sollen diese als Ausgangsbasis für den Versuch dienen, die Stärken verschiedener Verfahren zu kombinieren, ohne gleichzeitig deren Schwächen in Kauf nehmen zu müssen. Dabei wird besonders die Kombination einer adaptierten Hüllkörper-Hierarchie in Kombination mit voxel-basierten Verfahren im kommenden Kapitel im Vordergrund stehen, und wie diese für den Einsatz auf Grafik-Prozessoren adaptiert werden können.

Dabei wird für jede Klasse von Verfahren speziell auf die Konstruktion, Traversierung und gegebenenfalls Aktualisierung der verwendeten Datenstrukturen, die elementaren Kollisionstests in der Objektpaar-Phase und (sofern nicht bereits in Abschnitt 3.6 vorgestellt) die verwendete Kontaktpunkt-Verwaltung eingegangen.

Hinsichtlich des Einsatzes der im Folgenden angesprochenen Verfahren auf Grafik-Prozessoren ist zu bemerken, dass für polygon-basierte Verfahren einige Implementierungen existieren; bild-basierte Verfahren im Sinn der vorgestellten Definition sind aufgrund ihrer Funktionsweise ausschließlich auf der Basis von Grafik-Hardware umgesetzt. Für voxel-basierte Verfahren wie dem in Unterabschnitt 4.4.3 vorgestellten Voxmap-Pointshell-Algorithmus sind dem Autor keine GPU-basierten Implementierungen bekannt.

4.2. Polygon-basierte Verfahren

Polygon-basierte Verfahren bezeichnen im Folgenden die Kombination polygonaler Geometriebeschreibungen mit auf Basis der Geometriebeschreibungen erstellter Hüllkörper-Hierarchien. Der Fokus des folgenden Abschnitts soll dabei nicht auf den eigentlichen Paar-Tests einzelner Elemente polygonaler Geometrien liegen, sondern sich nochmals eingehender mit den Charakteristiken von Hüllkörper-Hierarchien, wie etwa deren Aufbau und Traversierung, beschäftigen. Dabei wird versucht, anhand einiger konkreter Beispiele aufzuzeigen, wie sich verschiedene Aspekte auf das Laufzeitverhalten bei der Traversierung auswirken, und

welche Möglichkeiten in existierenden Veröffentlichungen und Implementierungen genutzt wurden und werden, um dieses positiv zu beeinflussen.

4.2.1. Bisherige Arbeiten

Als mögliche Varianten von Hüllkörper-Hierarchien wurden im Lauf der vergangenen 30 Jahre viele Varianten vorgeschlagen. Eine beispielhafte Auswahl möglicher Ansätze umfasst (siehe auch Unterabschnitt 3.4.1):

- Hüllkugeln ([Qui94], [Hub95], [GB96], [TC96], [OD99], [WZ09b])
- Achsen-orientierte Hüllquader (AABBs) ([Zac95], [Ber97], [Z+00])
- Objekt-orientierte Hüllquader (OBBs) ([GLM96], [LMM10])
- Konvexe Polyeder (k-DOPs) ([Zac98], [KHM+98], [KA98])
- Ansätze mit einer Kombination verschiedener Hüllkörper-Arten ([BCG+96]) oder anderer Hüllkörper-Formen ([He99])

Für detaillierte Informationen zu den genannten und weiteren Ansätzen bieten beispielsweise folgende Quellen ausführliches Material:

- Eckstein legt in [Eck98, Kapitel 3, Kapitel 4] ausführlich Kriterien und Gütemaße für die optimierte Top-Down-Konstruktion von Hüllkörper-Hierarchien und Schnitt-Test zwischen Hüllkörpern (Hüllkugeln, AABBs, OBB) dar.
- Erleben geht in [Erl05, Kapitel 15] ebenso ausführlich auf die Konstruktion und Traversierung von Hüllkörper-Hierarchien ein, und behandelt auch verformbare Geometrien, inklusive der Überprüfung von elastischen Geometrien auf Überschneidungen mit sich selbst.
- Ericson beschäftigt sich in [Eri05, Kapitel 6] ebenfalls mit den verschiedenen Aspekten der Konstruktion und Traversierung von Hüllkörper-Hierarchien.
- Weller behandelt in [Wel13, Kapitel 2] ebenfalls dieses Thema als Teil einer Übersicht über etablierte Verfahren in der Kollisionserkennung.

4.2.2. Kostenabschätzung

Unter anderem nach [KK86] und [Hub95] sollte eine Hüllkörper-Hierarchie folgende Kriterien erfüllen:

- Baum-Knoten eines Teilbaums der Hierarchie sollten räumlich nahe beieinander liegen
- Der Hüllkörper jedes Baum-Knotens sollte ein möglichst kleines Volumen haben
- Das summierte Volumen aller Hüllkörper der Hierarchie sollte möglichst klein sein
- Die Überlappung zwischen Hüllkörpern auf der selben Ebene der Hierarchie sollte möglichst klein sein
- Hüllkörper auf höheren Ebenen der Hierarchie sollten bei der Traversierung bevorzugt behandelt werden

- Die Baumstruktur einer Hierarchie sollte bezüglich der Knoten-Struktur und der umschlossenen Geometrieteile möglichst gut balanciert sein

Diese Anforderungen beschreiben die verschiedenen Einflüsse auf die Laufzeit-Effizienz einer Hüllkörper-Hierarchie und betreffen:

- Die Konstruktion (Unterabschnitt 3.4.3), also einerseits die Methode zur Berechnung möglichst gut approximierter Hüllkörper, und andererseits die entstehende Baum-Struktur der Hüllkörper-Hierarchie, abhängig von:
  - Baum-Tiefe
  - Verzweigungsgrad
  - Balance
- Die Traversierung (Unterabschnitt 3.4.4), abhängig von:
  - Der Art des Traversierungs-Verfahrens (uninformierte Suche versus heuristische Suche)
  - Die verwendete Abstiegs-Regel
  - Die Reihenfolge beim Traversieren (simultan versus abwechselnd)

Die Laufzeit-Effizienz einer Hüllkörper-Hierarchie lässt sich entsprechend der soeben genannten Faktoren nach [WHG84], [GLM96], [KHM+98], [Eck98], [He99] in Gestalt einer Kosten-Funktion ausdrücken (nach der Notation von [Eri05, Abschnitt 6.1.2]):

$$T = N_V C_V + N_P C_P + N_U C_U + C_O \tag{4.1}$$

mit N_V als Anzahl durchgeführter Hüllkörper-Schnitttests und C_V als Kosten eines Schnitttests, N_P als Anzahl durchgeführter Schnitttests zwischen Paaren von Seitenflächen und C_P als Kosten eines solchen Schnitttests, N_U und C_U als Anzahl von und Kosten pro Aktualisierung eines Hüllkörpers, und mit C_O als Aufwand für Operationen wie die Transformation eines Hüllkörpers in das Koordinatensystem eines anderen.

Die Kosten eines Paar-Tests zwischen Seitenflächen sind unabhängig von den restlichen Faktoren dieser Kostenabschätzung, alle anderen stellen konkurrierende Ziele dar. So ist die Balance zwischen Komplexität der Schnitt-Tests zwischen Hüllkörpern und dem Grad der Approximation der zugrunde liegenden Geometrie einer der schwierigsten Kompromisse, die beim Umgang mit Hüllkörper-Hierarchien berücksichtigt werden müssen.

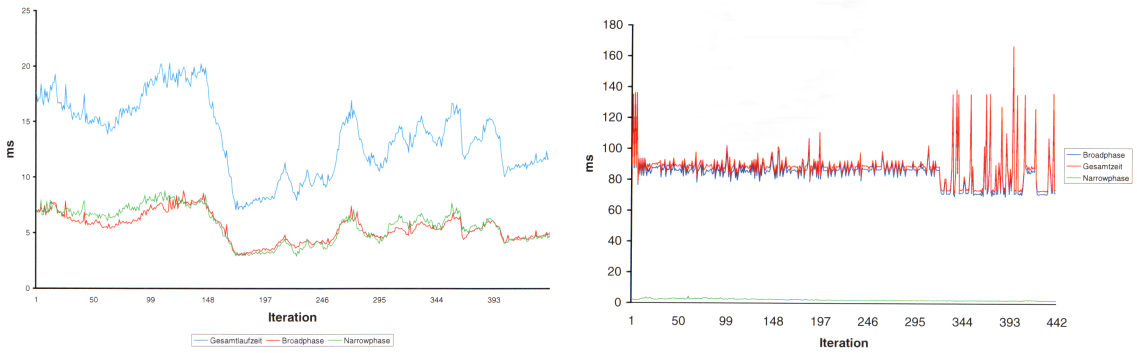
Als bester Kompromiss zwischen Berechnungsaufwand für den Schnitt-Test und Effizienz beim Ausschließen von Hierarchie-Teilen werden für polygonale Geometrien von vielen Autoren objekt-orientierte Hüllquader genannt ([GLM96], [LGLM99], [Zac98]). Außerdem haben OBBs den Vorteil, ein wesentlich besseres Verhältnis zwischen dem Volumen der umschlossenen Geometrie und dem Volumen der Hüllkörper zu haben als AABBs oder Hüllkugeln. Diese Eigenschaft wird in Kapitel 5 ausgenutzt werden.

4.2.3. Laufzeit-Verhalten

Anstatt sich wie die zuvor genannten Autoren primär letztlich auf eine Optimierung der Kosten-Funktion zu konzentrieren, soll als Motivation für die Entwicklung des in Kapitel 5 vorgestellten eigenen Ansatzes zur Kollisionserkennung ein anderer Weg eingeschlagen werden. Dazu ist zuerst eine kurze Betrachtung zweier praktischer Beispiele für den Einsatz von Hüllkörper-Hierarchien nützlich:

- Das erste Beispiel zeigt zwei polygon-basierte Ansätze im Vergleich, wie sie im Rahmen der in Abschnitt 2.1 beschriebenen Projektstudie im Bereich der virtuellen Inbetriebnahme eingesetzt worden sind: Der in der Bullet Physics Engine für die Kollisionserkennung zwischen konkaven, bewegten Dreiecksnetzen verwendete Kollisionserkennungs-Algorithmus GIMPACT ([Leo07]), und der Ansatz nach Eckstein und Buck ([Eck98], [Buc99]) aus der Virtual-Reality-Applikation Veo.
- Das zweite Beispiel betrifft die GPU-basierte Implementierung auf der Basis von OBB-Bäumen in gProximity ([LMM10]).

Das Laufzeitverhalten von GIMPACT und Veo ist in den Diagrammen in Abbildung 4.1 und Abbildung 4.2a skizziert. Abbildung 4.1 zeigt das Laufzeitverhalten der Algorithmen im Schraube-Lagerbock-Szenario aus Unterabschnitt 3.7.1, Abbildung 4.2a für ein anderes Experiment mit zwei aufeinander liegenden Antriebs-Kegelrädern (Abbildung 4.2a).

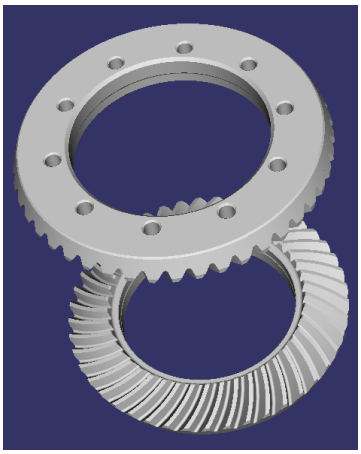


(a) GIMPACT

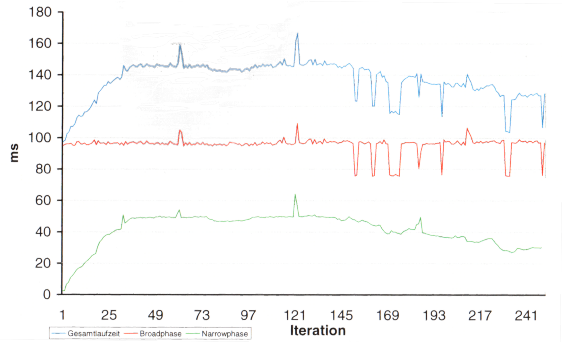
(b) Veo (nach Buck [Buc99] und Eckstein [Eck98])

Abbildung 4.1.: Laufzeit-Verhalten von GIMPACT (Abbildung 4.1a) und Veo (Abbildung 4.1b) im Schraube-Lagerbock-Experiment aus Unterabschnitt 3.7.1 (nach [Bec10]): Die Traversierung der Hüllkörper-Hierarchien der Modelle benötigt bei beiden Kollisionserkennungs-Verfahren einen wesentlichen Anteil der Gesamtlaufzeit (etwa 70 Prozent bei GIMPACT, bei Veo sogar 90 Prozent und mehr). Die Anzahl tatsächlich durchgeführter Paar-Tests wird dadurch effektiv verringert; jedoch ändert dies nichts an der Tatsache, dass beide Verfahren für diese Szene Laufzeiten beanspruchen, die jenseits interaktiver Laufzeitgrenzen liegen.

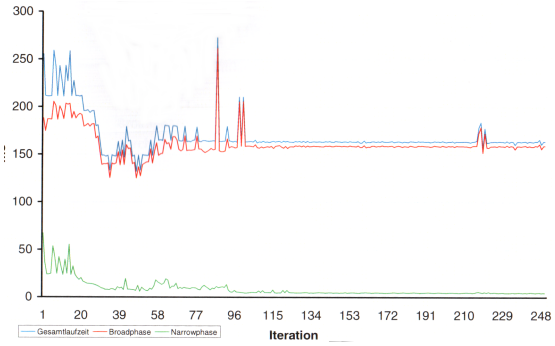
Beachtet man jeweils die Anteile der Hierarchie-Traversierung (rote Kennlinien in den Diagrammen) und den Paar-Tests zwischen einzelnen Seitenflächen (grüne Kennlinien), so ist bei beiden Algorithmen (in besonderem Maß aber bei Veo) zu erkennen, dass die Traversierung einen mindestens gleich großen oder sogar den wesentlichen Anteil an der benötigten Gesamtlaufzeit pro Iteration der Simulation benötigt. Dies ist im Fall von Veo ein Indiz für die Effektivität der verwendeten Konstruktions- und Traversierungs-Methoden, da im Verlauf



(a) Verwendete Modelle



(b) GIMPACT



(c) Veo (nach Buck [Buc99] und Eckstein [Eck98])

Abbildung 4.2.: Laufzeit-Verhalten von GIMPACT (Abbildung 4.2b) und Veo (Abbildung 4.2c) mit zwei aufeinander fallenden Antriebs-Kegelrädern (Abbildung 4.2a, nach [Bec10]): Auch in diesem Versuch beansprucht die paarweise Traversierung von Hüllkörper-Hierarchien bei den beiden Verfahren einen wesentlichen Anteil der gesamten verbrauchten Laufzeit pro Iteration.

der Paartest-Phase ein wesentlicher Anteil der Seitenflächen der jeweiligen Geometrie-Paare im Lauf der Traversierung der Hüllkörper-Hierarchie ausgeschlossen werden kann.

Dieselbe Beobachtung gilt auch für das Laufzeitverhalten von gProximity (in Abbildung 4.3), sowohl bei der Kollisionserkennung mit als auch ohne Berücksichtigung von Objektbewegungen. Der größte prozentuale Anteil an der benötigten Laufzeit wird durch die Traversierungs-Phase verbraucht, in allen gezeigten Konfigurationen etwa zwischen 70 und 80 Prozent der Gesamtlaufzeit. Auch hier liegt die Schlussfolgerung nahe, dass die verwendeten OBB- und AABB-Hierarchien effektiv dazu in der Lage sind, die Anzahl tatsächlich benötigter Paar-Tests zwischen Seitenflächen stark zu verringern.

Dennoch stellt sich die Frage, ob und inwiefern die Traversierungs-Phase in geeigneter Form adaptiert werden kann, um ihren wesentlichen Anteil an der nötigen Gesamtlaufzeit für einen Objektpaar-Test zu verringern und so eine weitere Zeitersparnis zu erreichen. Dabei soll im weiteren Verlauf nicht auf die Verwendung von Hüllkörpern verzichtet werden. Stattdessen ist das Ziel, eine alternative Konstruktions-Methode anhand polygonaler Geometriebeschreibungen zu finden, die gut für die Verwendung auf GPUs geeignet ist, dabei aber einen besseres Laufzeitverhalten aufweist als GPU-basierte Hüllkörper-Hierarchien, wie sie hier beschrieben

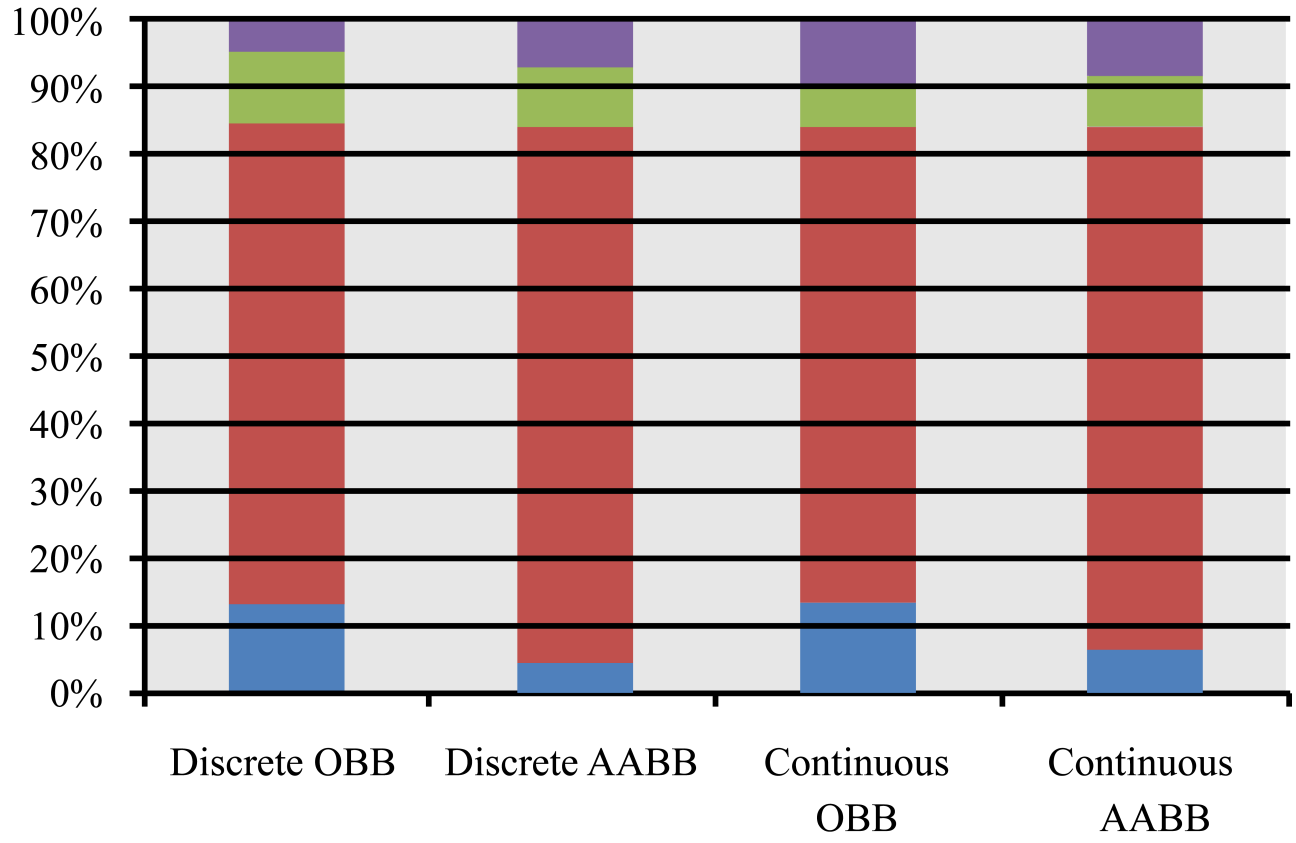


Abbildung 4.3.: Laufzeitverhalten von gProximity (Unterabschnitt 4.3.1, [LMM10]): Betrachtet man die reinen prozentualen Anteile der einzelnen Schritte des Verfahrens, so ist es auch bei diesem Verfahren die paarweise Traversierung von Hüllkörper-Hierarchien (in der Abbildung rot markiert), die den größten Laufzeitaufwand verursacht.

worden sind. Um mögliche Ansatzpunkte für solche Adaptionen zu erläutern, sind nochmals einige Aspekte der Konstruktion und Traversierung von Hüllkörper-Hierarchien zu beachten.

4.2.4. Konstruktion von Hüllkörper-Hierarchien

Bei der Konstruktion von Hüllkörper-Hierarchien in Hinblick auf den Einsatz mit GPUs ist es der Verzweigungsgrad einer Hüllkörper-Hierarchie, der im Wesentlichen bestimmt, wie effektiv der Abstieg durch die Hierarchie parallelisiert werden kann.

Für einen Hierarchie vom Verzweigungsgrad k mit n Blattknoten ergibt sich die Zahl innerer Knoten

$$\frac{n-1}{k-1}$$

gegenüber der Gesamtzahl aller Knoten

$$\frac{nk-1}{k-1}$$

Je höher der Verzweigungsgrad eines Baums also ist, desto weniger innere Knoten sind erforderlich, und die Höhe der Hierarchie verringert sich. Dem gegenüber steht das Problem, dass pro Hierarchie-Ebene umso mehr Aufwand für paarweise Schnitttests zwischen Hüllkörpern investiert werden muss, um alle Kombinationen aus Knoten derselben Hierarchie-Ebene aus zwei Hierarchien zu testen.

Die überwiegende Mehrheit von Hüllkörper-Hierarchien sind in Form binärer Bäume umgesetzt. Binäre Bäume sind einfacher zu konstruieren und in Datenstrukturen umzusetzen als Bäume mit höherem Verzweigungsgrad. Als Beispiel sei die Top-Down-Konstruktion genannt: Um für die Konstruktion der nächsten in der Hierarchie tiefer liegenden Ebene von Hüllkörpern das Volumen des Eltern-Knoten aufzuteilen, muss nur eine einzelne trennende Ebene ermittelt werden; k verschiedene Objekte können dagegen bereits auf $2^{k-1}-1$ verschiedene Weisen in zwei Teilmengen unterteilt werden. Diese Zahl möglicher Partitionen steigt exponentiell mit der Anzahl der Unterteilungen, was die Konstruktion von Teilbäumen höherer Grade (abgesehen von einfach strukturierten Unterteilungs-Schemata) zu aufwendig macht.

4.2.5. Traversierung von Hüllkörper-Hierarchien

Daneben wird von einigen Autoren ([KHM+98], [KA98]) über die verwendete Abstiegsregel argumentiert: Betrachtet man für eine balancierte Hierarchie vom Grad k mit n Blattknoten, so ergibt sich für die Abstiegsregel „A vor B“ ein Laufzeitaufwand proportional zu $klog_k(n)$, für einen simultanen Abstieg dagegen proportional zu $k^2log_d(k)$. Ersterer wird bei $k = 2.718$ minimal, letzterer dagegen bei $k = 2$, was für die Verzweigungsgrade 2 und 3 spricht.

Hierarchien mit binärem Verzweigungsgrad sind demnach die bei Weitem am öftesten verwendete Variante von Hüllkörper-Hierarchien, und Bäumen mit höherem Verzweigungsgrad scheint im Allgemeinen wenig Aufmerksamkeit gewidmet zu werden ([Wel13], [ZL03]). Mezger ([MKE03]) führt an, dass durch die Verwendung von Hierarchien mit höheren Verzwei-

gungsgraden zwar die Anzahl benötigter Schnitttests zwischen Hüllvolumina gleich bleibt, sich jedoch durch die Verringerung der Tiefe der Hierarchien bei der Traversierung mittels rekursiver Verfahren die Rekursionstiefe halbiert.

Lauterbach ([LMM10]) führt für die Bestimmung des minimalen Abstands zwischen zwei polygonalen Geometrien, die ebenfalls von gProximity unterstützt wird, ein anderes Argument an, das für die Verwendung von Bäumen mit höherem Verzweigungsgrad spricht: Im Unterschied zur reinen Kollisionserkennung können bei dieser Aufgabe Hierarchie-Teile nicht von weiteren Tests ausgeschlossen werden, da die Bestimmung des minimalen Abstands letztendlich immer auf einem Paar von Blattknoten erfolgt. Außerdem ist es notwendig, die zwischenzeitlichen Abschätzungen des minimalen Abstands, die während einer Traversierung ermittelt worden sind, für unterschiedliche Hierarchie-Teile regelmäßig abzugleichen, um die weitere gezielte Suche zu ermöglichen. Daher werden für die Bestimmung des minimalen Abstands in gProximity Hierarchien mit einem höheren Verzweigungsgrad eingesetzt; die Autoren geben dabei aufgrund empirischer Erfahrungen Bäume mit dem Verzweigungsgrad acht als optimal an.

4.3. GPU-basierte Umsetzungen polygon-basierter Verfahren

Das zentrale Problem bei der Verwendung von GPUs für polygon-basierte Verfahren ist die Parallelisierbarkeit der Hierarchie-Traversierung. Bedingt durch die Struktur einer Hierarchie in Kombination mit dem gewählten Traversierungs-Verfahren ist der Grad der möglichen Parallelisierbarkeit bei Hüllkörper-Hierarchien eingeschränkt, und die in Hardware umgesetzte Warteschlangen-Verwaltung von GPU-Prozessoren (wie in Abschnitt C.1 beschrieben) erlaubt keinen expliziten Eingriff in die Planung der Reihenfolge noch auszuführender Berechnungen. Des weiteren ist abhängig von der wechselseitigen Lage eines Paars von polygonalen Objekten nicht vorhersagbar, welche Teile der zugehörigen Hüllkörper-Hierarchien im Lauf der Traversierung von weiteren Tests ausgeschlossen werden können, und somit nicht a priori planbar, wie viele Hüllkörper-Tests im Lauf der Traversierung genau durchgeführt werden müssen.

Daher ist es nötig, einen Weg zu finden, eine möglichst gleichmäßige Auslastung aller Kerne eines GPU-Prozessors zu gewährleisten, um vom potentiellen Leistungsgewinn durch massiv parallele Berechnungen profitieren zu können.

Paar-Tests für Seitenflächen sind hiervon in wesentlich geringerem Maß betroffen, da hier keine Abhängigkeiten in der Reihenfolge der einzelnen Tests mehr besteht, wie sie durch die Struktur und das gewählte Traversierungs-Verfahren bei Hüllkörper-Hierarchien gegeben sind.

Ein weiteres Problem bei der Nutzung von GPUs ist die Datenübertragung zwischen den Speicherbereichen des Gast-Systems und einer GPU: Obwohl neuere GPU-Plattformen einen direkten Zugriff auf den Arbeitsspeicher des Gast-Systems unterstützen, ist die Zugriffsgeschwindigkeit sowohl bei lesenden als auch schreibenden Zugriffen um ein Vielfaches geringer als beim Zugriff auf den GPU-eigenen Arbeitsspeicher. Daher ist darauf zu achten, den Datentransfer zwischen Gast-System und GPU in möglichst gebündelten Zugriffen durchzu-

führen, und während der Ausführung einzelner Kernel so weit möglich auf weitere Speicherzugriffe im Gast-System zu verzichten.

Mögliche Alternativen für die Warteschlangen-Verwaltung und Aufgaben-Planung sowie den Datenaustausch zwischen Gast-System und GPU sollen im Folgenden anhand existierender Implementierungen aufgezeigt werden. Dazu werden folgende unter teilweiser oder ausschließlicher Verwendung von GPU-Prozessoren umgesetzte polygon-basierte Verfahren im Anschluss betrachtet:

- gProximity ([LMM10]), das Hierarchie-Traversierung (OBB-Bäume) und Paartests von Seitenflächen GPU-basiert umsetzt.
- HPCCD ([KHH+09]), ein hybrides CPU-/GPU-Verfahren. Die Hierarchie-Traversierung (AABB-Bäume) wird CPU-basiert unter Nutzung von Multi-Threading ausgeführt, Paartests von Seitenflächen GPU-basiert.
- Eine Fallstudie der Hochschule für Technik Stuttgart ([EMSW13]), in der unterschiedliche Ansätze zur Warteschlangen-Verwaltung für GPU-gestützte Kollisionserkennungs-Verfahren verglichen werden.

4.3.1. gProximity

gProximity nutzt OBB-Hierarchien für eine rein GPU-basierte Kollisionserkennung ohne Berücksichtigung von Objektbewegungen, und unterstützt darüber hinaus die Kollisionserkennung unter Berücksichtigung von Objektbewegungen mittels extrudierter Hüllkörper, die das in einem Zeitintervall überstrichene Volumen approximieren.

Obwohl die statische Kollisionserkennung hier im Vordergrund steht, ist eine kurze Erläuterung der Kollisionserkennung unter Berücksichtigung von Objektbewegungen angebracht, die in Kapitel 7 nochmals kurz in Zusammenhang mit dem eigenen Verfahren aufgegriffen werden soll.

Als Hüllkörper für die dynamische Kollisionserkennung dienen sogenannte Rectangular Swept Spheres (RSS, [LGLM00]), die aus der Minkowski-Summe (Unterabschnitt B.2.1) eines Rechtecks und einer Kugel bestehen (Abbildung 4.4). Die Schnittberechnung erfolgt über die Bestimmung die Abschätzung des Abstands zwischen Paaren von Kanten der in einem RSS eingebetteten Rechtecke mittels deren Voronoi-Regionen. Für die Konstruktion und Neuberechnung von OBB- und RSS-Hierarchien im Fall verformbarer Geometrien oder zur Anpassung an erfolgte Bewegungen verwendet gProximity einen zweistufigen Ansatz ([LGS+09]), der die Gruppierung von Seitenflächen der Eingangsgeometrie und die anschließende Berechnung der Hüllkörper-Hierarchie voneinander getrennt ausführt. Da die Zuordnung von Teilen der Eingangsgeometrie vor der Berechnung der Hierarchie bereits festgelegt ist, kann letztere problemlos parallel durchgeführt werden, da durch die vorhergehende Aufteilungs-Phase die Partitionierung der Hüllkörper-Volumina nicht erst während der Konstruktions-Phase erfolgt.

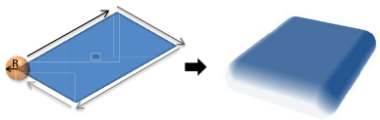


Abbildung 4.4.: Konstruktion einer Rectangular Swept Sphere (RSS): Eine RSS entsteht aus der Minkowski-Summe einer Seitenfläche und einer Kugel.

4.3.1.1. Warteschlangen-Verwaltung

Um eine GPU möglichst effektiv auslasten zu können, schlagen die Autoren von gProximity einen iterativen Ansatz zur Traversierung vor, der pro Ausführungs-Einheit einen Teilbaum der Hierarchie nicht komplett traversiert, sondern zu überprüfende Paare von Kind-Knoten eines aktuell betrachteten Paares aus Hüllkörpern in eine Warteschlangen-Datenstruktur einreiht. In der Warteschlange befindliche Paar-Tests werden in einem zweiten Kernel-Aufruf neu ausbalanciert, sobald die Bearbeitung einer Hierarchie-Ebene aufgrund der nicht mehr gegebenen Verfügbarkeit weiterer Streaming-Prozessoren nicht mehr weiter parallelisiert werden kann, oder die Beendigung aller auszuführenden Paartests einer Hierarchie-Ebene festgestellt wird (siehe Abbildung 4.5).

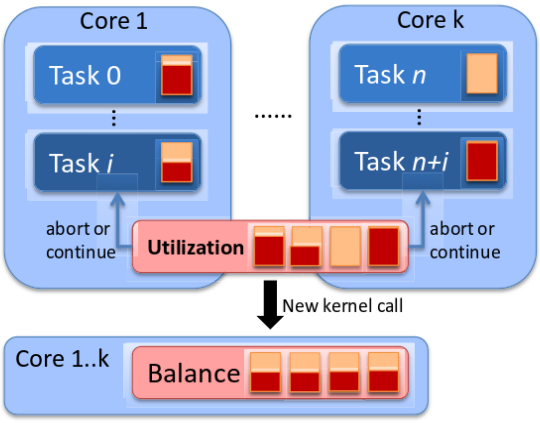
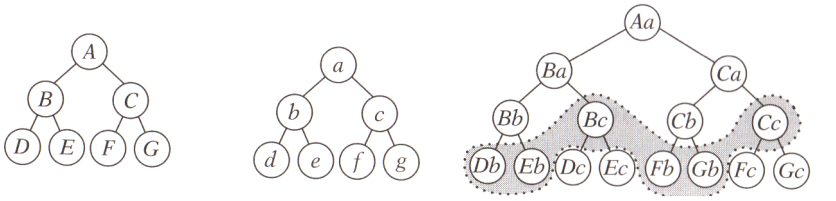


Abbildung 4.5.: Warteschlangen-Verwaltung in gProximity ([LMM10]): Die paarweise Traversierung von Teilen der Hüllkörper-Hierarchien in einem Objektpaar-Test wird parallelisiert abgearbeitet, bis in jeder (unabhängigen) Ausführungseinheit eine von zwei Abbruchbedingungen erreicht wird: Entweder sind keine weiteren Paartests mehr möglich, da keine Überschneidung festgestellt oder die Blattknoten-Ebene erreicht wurde, oder die Anzahl von Tests überschreitet die Länge der Warteschlange einer Ausführungseinheit. In einem separaten Kernel-Aufruf werden alle Warteschlangen rebalanciert, und eine weitere Iteration der Traversierung angestoßen, bis alle Paartests vollständig verarbeitet worden sind.

Diese Vorgehensweise vermeidet die Notwendigkeit, während der parallelen Ausführung von Paartests eine Synchronisierung zwischen verschiedenen Kernel-Instanzen vornehmen zu müssen, die sich negativ auf die Parallelisierung auswirken würde, und vermeidet durch die explizite Rebalancierung eine ungleichmäßige Auslastung. Ebenso werden exzessive Schreib- und Lesevorgänge auf den Speicher des Gast-Systems vermieden, da neben der Übertragung der Geometriebeschreibungen und der Hüllkörper-Hierarchien in den Speicher einer GPU

nur die Indizes von Hüllkörper-Paaren transferiert werden müssen, die von individuellen Kernel-Instanzen in die Warteschlange eingereiht werden.

Um die Verarbeitung der ersten Paartests in den obersten Ebenen eines Paars von Hüllkörper-Hierarchien besser parallelisieren zu können, schlagen die Autoren den Einsatz des sogenannten Front-Tracking-Ansatzes vor ([LC98], [KHM+98]): Anstatt in jeder Iteration einer Simulation alle Paare von Hüllkörper-Hierarchien komplett zu traversieren, wird die Tatsache ausgenutzt, dass sich Objekte zwischen zwei Iterationen aufgrund der normalerweise benutzten geringen Zeitschritt-Weite von wenigen Millisekunden nur vergleichsweise geringfügig relativ zueinander bewegen, und daher die Ergebnisse einer vorhergehenden Traversierung als Ausgangspunkt für eine erneute Traversierung in der nächsten Iteration benutzt werden können.



(a) Hüllkörper-Hierarchie 1 **(b)** Hüllkörper-Hierarchie 2 **(c)** BVTT mit Front (schattiert)

Abbildung 4.6.: Funktionsweise des Front-Tracking: Alle möglichen Kombinationen von Paar-Tests zwischen Hüllkörpern aus zwei Hüllkörper-Hierarchien werden in einem Traversierungs-Baum expandiert. Anhand dieser Expansion können diejenigen Paare von Hüllvolumina festgehalten werden, in denen pro Hierarchie-Ebene zuletzt keine Überschneidung zwischen Teil-Hierarchien des geprüften Objektpaars festgestellt worden ist. Diese Front von Knoten kann als Ausgangspunkt für eine erneute paarweise Traversierung in der nächsten Iteration einer Simulation genutzt werden, eine entsprechend geringe Positions- und Ausrichtungs-Änderung der Objekte vorausgesetzt.

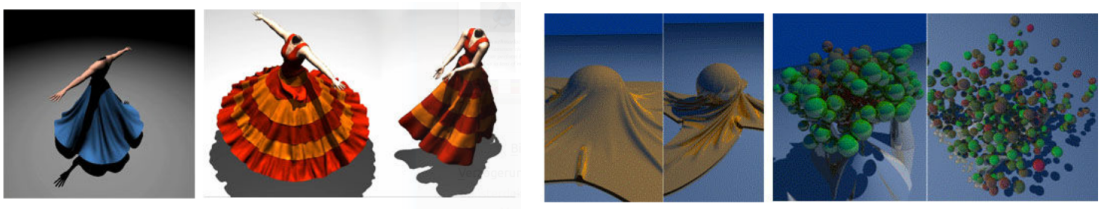
Dazu wird ein Traversierungs-Baum erstellt (Bounding Volume Traversal Tree, BVTT, Abbildung 4.6), der alle Kombinationen durchgeführter Paartests in einer Traversierung beinhaltet. Anhand eines BVTT kann eine Menge von Hüllkörper-Paaren ermittelt werden, die in der letzten Traversierung als nicht überlappend festgestellt worden sind. Diese Knoten-„Front“ dient dann als Start-Menge für eine erneute Traversierung, in der zielgerichtet die Front selbst sowie alle unterhalb im BVTT liegenden Paartests (sowie unter bestimmten Bedingungen die Eltern-Knoten der Front im BVTT) geprüft werden können, anstatt erneut eine vollständige Traversierung zu starten.

Dies hat für einen GPU-basierten Algorithmus vor allem für große Hüllkörper-Hierarchien den Vorteil, eine wesentlich bessere Parallelisierbarkeit zur ermöglichen, da die Front umso tiefer im BVTT zu liegen kommt, je näher sich ein Paar aus Objekten kommt.

4.3.1.2. Laufzeitverhalten

Zur Beurteilung der Leistungsfähigkeit ihres Ansatzes haben die Autoren von gProximity einige Beispiel-Szenarien verwendet, die das Verhalten von Stoff an animierten Avataren (Abbildung 4.7a) und im Zusammenspiel mit einer starren Geometrie (einer rotierenden Kugel,

über der ein Stofftuch ausgebreitet wird), sowie eine Vielkörper-Simulation (Abbildung 4.7b) betreffen.



(a) Elastische Geometrien

(b) Elastische vs. starre Geometrie, Viel-Körper-Simulation

Abbildung 4.7.: Beispiel-Szenarien für gProximity

Die Autoren von gProximity geben für ihren Ansatz Leistungsdaten an, die fast durchgängig im Bereich interaktiver Laufzeiten liegen: Sowohl die Erstellung und Neuberechnung (Tabelle 4.1) als auch die Traversierung (Tabelle 4.2) von Hüllkörper-Hierarchien (von den Autoren jeweils für AABBs und OBBs verglichen), sowie die Paartests zwischen Seitenflächen werden von gProximity (abgesehen von der Vielkörper-Simulation) deutlich unter 40 Millisekunden absolviert. Das ist ein deutliches Indiz dafür, dass GPU-basierte Kollisionserkennungs-Verfahren bei effektiver Parallelisierbarkeit von Berechnungen dazu in der Lage sind, einen enormen Leistungsgewinn gegenüber CPU-basierten Ansätzen (auch mit Nutzung von Multithreading) zu ermöglichen.

Modell	Anzahl Seitenflächen	Konstruktion		Neuberechnung	
		AABB	OBB	AABB	OBB
Stoff-Simulation 1	49000	22	27	1,73	4,6
Stoff-Simulation 2	40000	20	24	1,4	3,9
Stoff-Simulation 3	92000	31	39	2,5	7,9
Mehrkörper-Simulation	146000	56	68	3,2	11,5

Tabelle 4.1.: Laufzeitbedarf (Millisekunden) von gProximity für Konstruktion und Neuberechnung von Hüllkörper-Hierarchien (aus [LMM10])

Die Autoren geben die von ihnen erzielte Leistungsverbesserung gegenüber CPU-basierten Ansätzen mit einem Faktor zwischen 2 und 5 an, unter Berücksichtigung unterschiedlicher verwendeter Hardware-Architekturen.

4.3.2. Hybrid Parallel Continuous Collision Detection (HPCCD)

HPCCD ([KHH+09]) ist im Gegensatz zu gProximity ein hybrider Ansatz, der die Traversierung und die Aktualisierung von Hüllkörper-Hierarchien (AABB-Bäume) mittels Multithreading auf den Prozessoren des Gast-Systems ausführt, und Paartests zwischen Seitenflächen auf GPUs auslagert (Abbildung 4.8).

Basierend auf den Ergebnissen der Traversierung eines Paars von Hüllkörper-Hierarchien werden die für die Paartests zwischen Seitenflächen nötigen Index-Daten inkrementell über

Modell	statische Erkennung		dynamische Erkennung	
	AABB	OBB	AABB	OBB
Stoff-Simulation 1	27	22	37	34
Stoff-Simulation 2	17	22	26	29
Stoff-Simulation 3	28	36	42	38
Mehrkörper-Simulation	78	70	91	74

Tabelle 4.2.: Laufzeitbedarf (Millisekunden) von gProximity für die Traversierung von Hüllkörper-Hierarchien

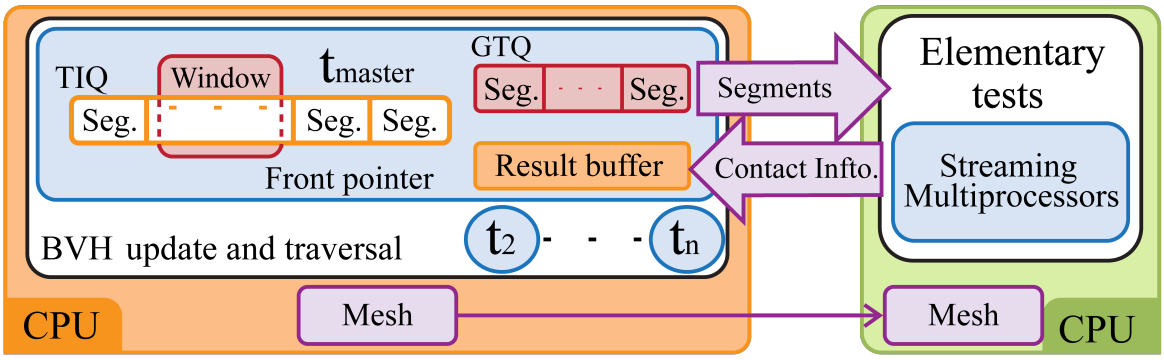


Abbildung 4.8.: Die Funktionsweise von HPCCD (aus [KHH+09]): Die paarweise Traversierung und die Aktualisierung von Hüllkörper-Hierarchien werden CPU-gestützt von mehreren Threads ausgeführt, während Paartests zwischen Seitenflächen GPU-gestützt erfolgen. Zur Minimierung der bidirektionalen Datenübertragung zwischen Gast-System und GPU-Prozessoren werden nötige Paartests in einer Warteschlange gesammelt. Erst wenn eine ausreichende Anzahl von Paartests erreicht wird, erfolgt die Übertragung der nötigen Geometrie- und Indexdaten auf die GPU, die diese Tests anschließend ausführt und deren Ergebnisse wiederum in einer einzigen Übertragung zurück in den Speicherbereich des Gast-Systems transferiert.

einen zentralen Scheduler-Thread auf die GPU übertragen, wo diese Paartests mittels dynamischer Kollisionserkennung durchgeführt werden.

Der zentrale Scheduler-Thread verwaltet einen Speicherbereich (Triangle Index Queue, TIQ in Abbildung 4.8), in dem jedem Thread in der Traversierung ein bestimmtes Segment zur Hinterlegung zu prüfender Paare aus Seitenflächen zugeteilt ist. Sobald eine bestimmte Anzahl von Segmenten gefüllt ist, werden diese für die Paartests in den Speicherbereich der GPU transferiert. Der Scheduler-Thread verwaltet darüber hinaus weitere Speicherbereiche, in denen der Verarbeitungs-Status von Segmenten durch die GPU verfolgt wird, und in denen die Ergebnisse der detaillierten Kollisions-Tests auf der GPU zurück in den Speicherbereich des Gast-Systems übertragen werden.

Die Art und Weise der Traversierung von Hüllkörper-Hierarchien in HPCCD erfordert eine andere Task-Planung, als sie gProximity einsetzt: Anstatt iterativ zwischen Traversierung und Balancierung zu wechseln, nutzt HPCCD einen pool-basierten Ansatz, in dem einzelne CPU-Threads entweder auf die Zuteilung neuer Hüllkörper-Tests warten, oder aber selbst Hüllkörper-Tests an wartende Threads delegieren können. HPCCD verwaltet die CPU-basierte Task-Planung also dezentral, und nicht wie gProximity gebündelt in einer separaten Rebalancierungs-Phase.

4.3.2.1. Warteschlangen-Verwaltung

Wie bei gProximity besteht auch bei HPCCD das Problem, wie bei der Traversierung eine möglichst gute Parallelisierung erreicht werden kann, und das vor allem in den höchsten Ebenen einer Hüllkörper-Hierarchie. Dazu wird bei HPCCD die initiale Aufgabenverteilung an einzelne CPU-Threads so vorgenommen, dass die Wurzelknoten der von jedem Thread zu bearbeitenden Hierarchie-Teile keine gemeinsamen Eltern-Knoten haben. Somit bestehen keine strukturellen Abhängigkeiten zwischen den verschiedenen Ausführungs-Einheiten mehr, wie in Abbildung 4.9 illustriert.

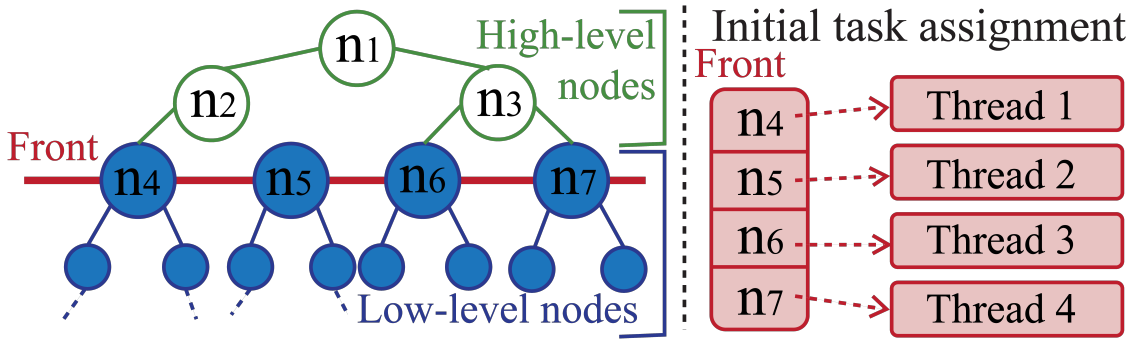


Abbildung 4.9.: Die Warteschlangen-Verwaltung von HPCCD (aus [KHH+09]): Anstatt wie gProximity Front-Tracking zu verwenden, um die initiale Aufgabenverteilung bei einer paarweisen Traversierung festzulegen, werden Teil-Hierarchien zwischen Threads so aufgeteilt, dass sie über keine gemeinsamen Eltern-Knoten verfügen. So werden wechselseitige Abhängigkeiten zwischen einzelnen Ausführungseinheiten vermieden. Die verbleibenden, obersten Ebenen einer Traversierung werden dabei durch einen einzelnen Thread ausgeführt.

Bei der dynamischen Neuverteilung von Hüllkörper-Tests zwischen einzelnen CPU-Threads dient die Anzahl von Seitenflächen in einem Teilbaum einer Hierarchie als Indiz für Aufgaben, die am besten an andere Threads delegiert werden können. Die Neuverteilung erfolgt dabei erst, nachdem ein Thread die Traversierung eines Teilbaums beendet hat; dadurch wird die wechselseitige Abhängigkeit von Threads minimiert, und abgesehen von der Kommunikation zur Datenübertragung von einem überlasteten zu einem verfügbaren Thread sind keine weiteren Maßnahmen zur Synchronisierung zwischen einzelnen CPU-Threads in der Traversierungs-Phase nötig.

Die Synchronisierung zwischen Gast-System und GPU-Prozessoren wird über einen separaten Kommunikations-Thread verwaltet, der abhängig vom Fortschritt der Hüllkörper-Traversierung eine Auftrags-Warteschlange verwaltet, die Indizes von potentiell kollidierenden Seitenflächen-Paaren enthält. Erst wenn in der Auftrags-Warteschlange genügend Segmente durch die Hierarchie-Traversierung ausgefüllt worden sind (CPU-seitig noch genutzte, nur teilweise gefüllte Segmente werden nicht an die GPU weitergeleitet), wird die Paartest-Phase für Seitenflächen auf der GPU angestoßen. Der Kommunikations-Thread gibt Segmente in der Auftrags-Warteschlange wieder frei, sobald die GPU die zugehörigen Berechnungen beendet hat. Außerdem werden Paartests für Seitenflächen nur dann GPU-basiert durchgeführt, wenn die Anzahl gefüllter Segmente der Auftrags-Warteschlange nicht die Anzahl auf einer GPU zur Verfügung stehender Streaming-Prozessoren überschreitet.

Modell	Anzahl Seitenflächen	dynamische Erkennung
Stoff-Simulation	92000	23,2
Vielkörper-Simulation 1	32000	6,8
Vielkörper-Simulation 2	146000	53,8
Zerbrechende Statue	252000	53,6

Tabelle 4.3.: Laufzeitbedarf (Millisekunden) von HPCCD

4.3.2.2. Laufzeitverhalten

Zur Beurteilung der Leistungsfähigkeit stellen die Autoren Leistungsdaten aus vier Beispiel-Szenarios zur Verfügung (Tabelle 4.3), von denen zwei auch von den Autoren von gProximity ([LMM10]) benutzt worden sind (die Interaktion eines Stofftuchs mit einer rotierenden Kugel, und eine Vielkörper-Simulation), und deren Vergleich mit HPCCD von Lauterbach auch dort angesprochen wird. Daneben wurde mit HPCCD noch ein weiteres, für ein Kollisionserkennungs-Verfahren anspruchsvolles Szenario mit einem komplexen, zerbrechenden Objekt durchgeführt (Abbildung 4.10). Insgesamt ist im Vergleich mit gProximity festzustellen, dass bei den direkt vergleichbaren Szenarien HPCCD ein besseres Laufzeitverhalten als gProximity aufweist, und auch die Simulation eines zerbrechenden Objekts mit einer Szenengröße von über 250000 Dreiecken beinahe in interaktiver Laufzeit umgesetzt werden kann.

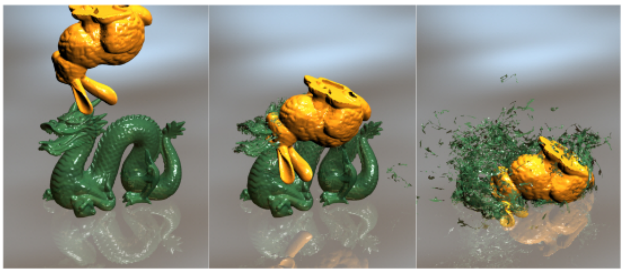


Abbildung 4.10.: Beispiel-Szenario aus HPCCD (zerbrechende Statue, [KHH+09])

Allerdings bleibt anzumerken, dass die Autoren von gProximity ihren Algorithmus auf einem System mit vier CPU-Kernen und nur mit einer einzigen GPU getestet haben, während das beste Laufzeitverhalten von HPCCD auf einer Plattform mit acht CPU-Kernen und zwei GPU-Einheiten ermittelt worden ist.

Die Autoren von HPCCD geben die erzielte Laufzeitverbesserung gegenüber konventionellen, CPU-basierten Ansätzen mit einem Faktor zwischen 8 und 13, und gegenüber früherer GPU-basierter Ansätze mit einem Faktor zwischen 5 und 10 an.

4.3.3. Ansatz der Hochschule für Technik Stuttgart

Erbes [EMSW13] und seine Mit-Autoren stellen in ihrer Veröffentlichung eine andere Anwendung für hochperformante Kollisionserkennungs-Verfahren vor, als die beiden zuvor vorgestellten Verfahren gProximity und HPCCD zum Ziel hatten: Bei den beiden Letzteren stand

die Leistungsfähigkeit von Paar-Tests im Mittelpunkt, für Anwendungen wie Vielkörper-Simulationen oder in der Computer-Animation. Hier steht die Kollisionserkennung als Teilaufgabe der Trajektorien-Planung in der Robotik im Vordergrund.

Die grundsätzliche Aufgabe ist zwar die Gleiche: Die Ermittlung potentieller Kollisionen zwischen einem Paar polygonal modellierter Geometrien. Hier steht jedoch nicht die Laufzeit für einen einzelnen Paartest im Vordergrund, sondern die effiziente statische Kollisionserkennung zwischen einem stationären und einem bewegten Objekt, für das eine Vielzahl alternativer Positionen und Orientierungen relativ zum stationären Objekt auf Kollisionsfreiheit überprüft werden muss.

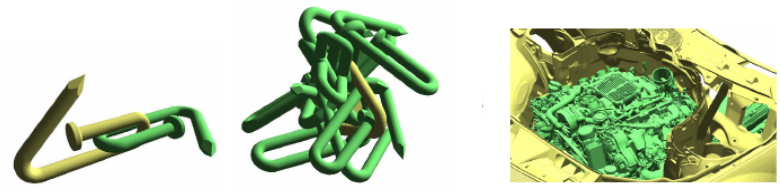
Das zu untersuchende Problem wird von den Autoren wie folgt beschrieben: $\mathbb{A}$ und $\mathbb{B}$ sind zwei polygonale Geometrien in der Form von Dreiecksnetzen, $Bvh_{\mathbb{A}}$ und $Bvh_{\mathbb{B}}$ zugehörige Hüllkörper-Hierarchien (OBB-Bäume). $\mathbb{B}$ ist unbewegt, während $\mathbb{A}$ unterschiedlichen affinen Transformationen $\mathbf{Q} \in \mathbb{R}^3 \times SO(3)$ unterworfen ist. Die Aufgabenstellung besteht nun darin, einen Kollisions-Test zwischen $\mathbb{B}$ und $\mathbf{Q} \cdot \mathbb{A}$ durchzuführen und damit die Kollisionsfreiheit einer bestimmten Positionierung beider Objekte zueinander nachzuweisen oder zu widerlegen. Als Traversierungs-Regel wird von den Autoren eine Tiefen-Suche verwendet.

Daher ergeben sich in dieser Anwendung zusätzliche Freiheitsgrade für die Parallelisierbarkeit von Berechnungen, die von den Autoren mittels mehrerer Alternativen zur Aufteilung von Berechnungen zwischen individuellen GPU-Ausführungseinheiten untersucht wurden:

- Eine 1:1-Entsprechung zwischen einem Thread auf der GPU und einem Paar-Tests (*KernelA* in Abbildung 4.12)
- Die Ausführung mehrerer Paar-Tests pro Thread auf der GPU (*KernelB* in Abbildung 4.12)
- Die Verwendung mehrerer GPU-Threads pro Paar-Test (*KernelC* in Abbildung 4.12)
- Die Gruppierung mehrerer GPU-Threads zur Ausführung mehrerer Paar-Tests pro Thread-Gruppe (*KernelD* in Abbildung 4.12)

4.3.3.1. Laufzeitverhalten

Im Unterschied zu den vorhergehenden Abschnitten ist es bei den von Ebers implementierten Verfahren hilfreich, zuerst das Laufzeitverhalten der verschiedenen Parallelisierungs-Schemata zu untersuchen, und erst im Anschluss auf die einzelnen Varianten einzugehen.

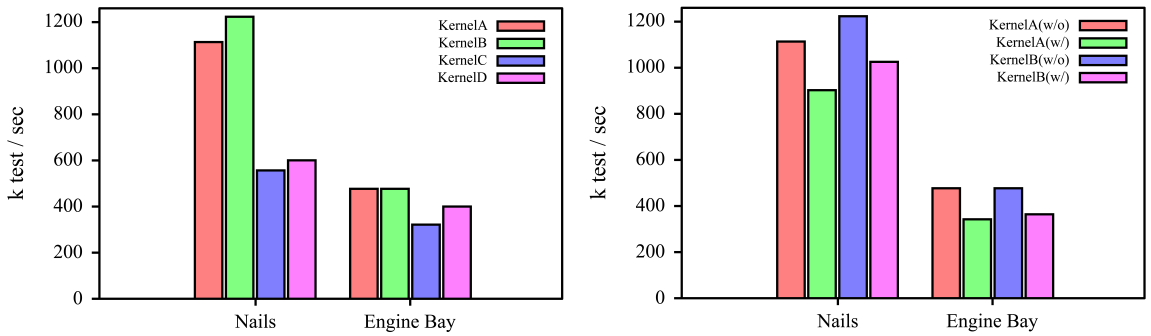


(a) Gebogene Nägel

(b) Motorblock und Motorraum

Abbildung 4.11.: Von Ebers ([EMSW13]) verwendete Beispiel-Szenarien

Als Beispiel-Szenarien wurden die in Abbildung 4.11 gezeigten Umgebungen verwendet: Verschiedene Kombinationen ineinander steckender, gebogener Nägel sowie das Ausbauen eines Motorblocks aus einem Motorraum.



(a) Reine Hierarchie-Traversierung **(b)** Mit Paar-Tests für Seitenflächen

Abbildung 4.12.: Laufzeitverhalten verschiedener Parallelisierungs-Schemata ([EMSW13]): Die Parallelisierung anhand einzelner Paartests beziehungsweise mehrerer Paartests pro Kernel-Instanz weist ein gegenüber komplexer organisierten Parallelisierungs-Schemata deutlich besseres Laufzeitverhalten auf.

Das Laufzeitverhalten der einzelnen Varianten ist in Abbildung 4.12a für die Traversierung der OBB-Hierarchien angegeben, in Abbildung 4.12b unter zusätzlicher Berücksichtigung von Paartests für Seitenflächen. Auffällig ist, dass die Varianten *KernelA* und *KernelB* in beiden verglichenen Anwendungsfällen eine deutlich höhere Anzahl Tests pro Sekunde bewältigen können als *KernelC* und *KernelD*.

Vergleicht man die unterschiedlichen Ansätze zur Parallelisierung in den vier von Ebers verwendeten Varianten, so ist bei den ersten beiden Alternativen (*KernelA* und *KernelB*) festzustellen, dass die Parallelisierung entlang einzelner beziehungsweise kleiner Teile der Eingabe-Menge von Transformationen $\mathbf{Q}$ erfolgt: Eine GPU-Thread bearbeitet exklusiv die Hierarchie-Traversierung einer einzelnen Eingabe-Transformation (*KernelA*, Algorithmus 3), beziehungsweise (sofern nicht bereits durch andere Threads aus demselben Warp bearbeitet) sequentiell mehrere Eingabe-Transformationen. Der Status der Überprüfung einer Eingabe-Transformation wird in *KernelA* von jedem Thread an einer dedizierten Position in einem globalen Speicherbereich abgelegt, es bestehen für diese Variante daher keine wechselseitigen Abhängigkeiten zwischen einzelnen Threads.

Die beiden anderen Alternativen (*KernelC* und *KernelD*) hingegen verwenden mehrere GPU-Threads für die Hierarchie-Traversierung einzelner Eingabe-Transformationen: *KernelC* parallelisiert einzelne Separating-Axis-Tests beim Schnitt-Test von OBB-Paaren (Algorithmus 4). Bei der Variante *KernelD* teilt sich eine Gruppe aus 32 Threads die Bearbeitung von acht verschiedenen Eingabe-Transformationen, wobei pro Eingabe-Transformation von mehreren Threads gleichzeitig Hüllkörper-Tests vorgenommen werden können.

KernelC ist die Variante mit den größten Abhängigkeiten zwischen einzelnen Threads einer Thread-Gruppe, wobei hier immer 16 Threads für einen Paartest zwischen zwei OBBs zusammengefasst werden. Ein Thread einer Gruppe ist für die Verwaltung des Stapelspeichers während der Traversierung verantwortlich, und transformiert eine der jeweils aktuell betrachteten OBBs entsprechend der bearbeiteten Eingabe-Transformation. Die anderen 15 Threads

Algorithmus 3 : KernelA

```
Input : Q, bvhA, bvhB
// Liste von Transformationen, Wurzel-Knoten von Hüllkörper-Hierarchien A und B
qIdx = blockDim · blockIdx + threadIdx;
S.start = globalStackArray + qIdx · STACK_SIZE;
Initialize();
while S ≠ ∅ ∧ col = 0 do
    (A, B) = S.pop();
    if Intersect(A(q), B) then
        col = 1;
    else
        S.push(children(A, B));
result[qIdx] = col;
```

der Gruppe berechnen jeweils einen Separating-Axis-Test. Das Ergebnis wird in einer von allen Threads einer Gruppe gemeinsam verwendeten Ergebnis-Variable festgehalten.

Im Unterschied zu *KernelA* ist bei *KernelC* das Problem, dass der schreibende Zugriff auf die Ergebnis-Variable synchronisiert werden muss, und dass jeder Thread einer Thread-Gruppe separate Lese-Zugriffe auf die Daten von OBBs im globalen Speicher durchführen muss, um seinen individuellen Separating-Axis-Test durchzuführen. Ebenso muss das Auslesen der Indizes eines Paares aus OBBs durch den Verwaltungs-Thread einer Thread-Gruppe synchronisiert werden. Auch von den Autoren selbst wird diese Variante der Parallelisierung daher als zu feingranular angesehen: Die Notwendigkeit mehrerer Synchronisations-Punkte zwischen Threads, die am selben Paartest zweier OBBs beteiligt sind, neutralisiert so den potentiellen Laufzeitgewinn, der durch eine Parallelisierung erreicht werden könnte.

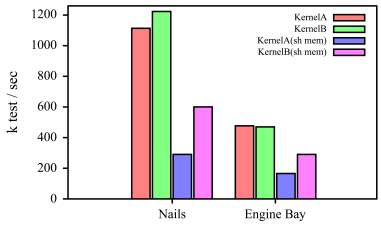
Ein letzter wichtiger Punkt, der von Erbes angesprochen wird, betrifft die Verwendung unterschiedlicher Speichertypen, die in CUDA-Architekturen zur Verfügung stehen (Abbildung 4.14):

- Register-Speicher (wenige Kilobyte): Zwischenspeicher für Variablen, sichtbar nur auf Thread-Ebene.
- Thread-lokaler Speicher (im Bereich einiger hundert Kilobyte): Sichtbar nur auf Thread-Ebene, Zugriff um Faktor 100-1000 langsamer als auf Register-Speicher.
- Gemeinsamer Speicher (unter 100 Kilobyte): Sichtbar nur auf Block-Ebene, Zugriff um Faktor 100 schneller als auf globalen Speicher. Zugriffe müssen zwischen Threads synchronisiert werden, sofern zwei oder mehr Threads auf dieselbe Speicheradresse zugreifen.
- Globaler Speicher (im Gigabyte-Bereich): Gemeinsamer Speicherbereich für alle Threads und Blöcke. Zugriff um Faktor 100 langsamer als auf lokalen Speicher. Zugriffe müssen zwischen Threads synchronisiert werden. Globaler Speicher wird vom Gast-System aus verwaltet und ist als einzige Speicher-Kategorie in CUDA-Architekturen vom Gast-System selbst les- und schreibbar.

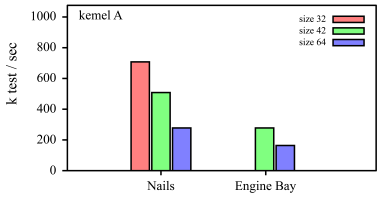
Algorithmus 4 : KernelC

Input : $Q, bvhA, bvhB$

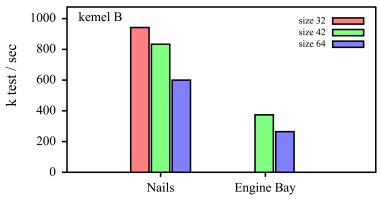
```
1  // Liste von Transformationen, Wurzel-Knoten von Hüllkörper-Hierarchien A und B
2  groupSize = 16;
3  groupsPerBlock = blockDim / groupSize;
4  tIdx = threadIdx mod groupSize;
5  gIdx = threadIdx / groupSize;
6  qIdx = groupsPerBlock · blockIdx + gIdx;
7  S.start = globalStackArray + qIdx · STACK_SIZE;
8  __shared__ volatile col;
9  __shared__ volatile OBB A, B;
10 Initialize();
11 while 1 do
12     if tIdx = 0 then
13         if S = ∅ then
14             col = 1;
15             break;
16         else
17             (idxA, idxB) = S.pop();
18     if tIdx < 15 then
19         (A, B) = loadVolumes(idxA, idxB);
20     if tIdx = 0 then
21         A = A(q);
22     __shared__ cut = 1;
23     if tIdx < 15 then
24         n = getSepAxis(A, B, tIdx);
25         if sepAxisProject(A, B, n) then
26             cut = 0;
27     if tIdx = 0 ∧ cut = 1 then
28         if A.isLeaf() ∧ B.isLeaf() then
29             col = 1;
30             break;
31         else
32             S.push(children(A, B));
33 result[qIdx] = col;
```



(a) Mit und ohne globalem Stapelspeicher



(b) Unterschiedliche Stapelspeicher-Größen, Variante A



(c) Unterschiedliche Stapelspeicher-Größen, Variante B

Abbildung 4.13.: Laufzeitverhalten unter Nutzung eines globalen Stapelspeichers ([EMSW13])

Die Autoren haben als weiteres Experiment den Speicherort des Stapelspeichers, der für die Tiefen-Suche bei der Traversierung von OBB-Hierarchien verwendet wird, an zwei verschiedenen Speicher-Stufen innerhalb der CUDA-Architektur (gemeinsamer versus globaler Speicher) und in unterschiedlichen Größen (zwischen 32 und 64 Kilobyte) variiert. Vergleicht man die Leistungsfähigkeit unter Verwendung lokalen Speichers gegenüber der Verwendung globalen Speichers (Abbildung 4.13a), so ist zu sehen, dass die Verwendung eines Stapelspeichers im gemeinsamen Speicher trotz dessen weitaus höherer Zugriffsgeschwindigkeit die Leistungsfähigkeit der beiden getesteten Verfahren enorm verschlechtert. Dies ist damit zu begründen, dass bei der Reservierung von Speicherbereichen im gemeinsamen Speicher eines Thread-Blocks die Anzahl gleichzeitig ausgeführter Thread-Blocks sinkt, je größer der reservierte Speicherbereich pro Thread-Block wird. Selbst wenn die Stapelspeicher-Größe pro Thread-Block verringert wird, um eine höhere Anzahl Thread-Blöcke parallel ausführen zu können, ist in den von Erbes getesteten Verfahren zur Hüllkörper-Traversierung kein besseres Laufzeit-Verhalten gegenüber der Verwendung globalen Speichers mit wesentlich höherer Latenz zu erreichen (Abbildung 4.13b, Abbildung 4.13c).

Dies verdeutlicht, wie CUDA-basierte Hardware mittels massiv paralleler Ausführung von Programmen Latenzen in Speicher-Zugriffen kompensieren kann. Allerdings wird dadurch einer der möglichen Nachteile GPU-basierter Berechnungen deutlich, auf die während der Entwicklung GPU-basierten Programm-Codes zu achten ist: Einen Kompromiss zwischen Parallelität der Ausführung und Häufigkeit sowie Umfang von Speicherzugriffen auf Speicherbereiche mit hoher Latenz zu finden. Daher ist es wichtig, sowohl die Übertragung von Daten zwischen Gast-System und GPUs und zurück zu minimieren, als auch auf die Lokalität von Speicherzugriffen während der Laufzeit einzelner Kernel-Instanzen zu achten.

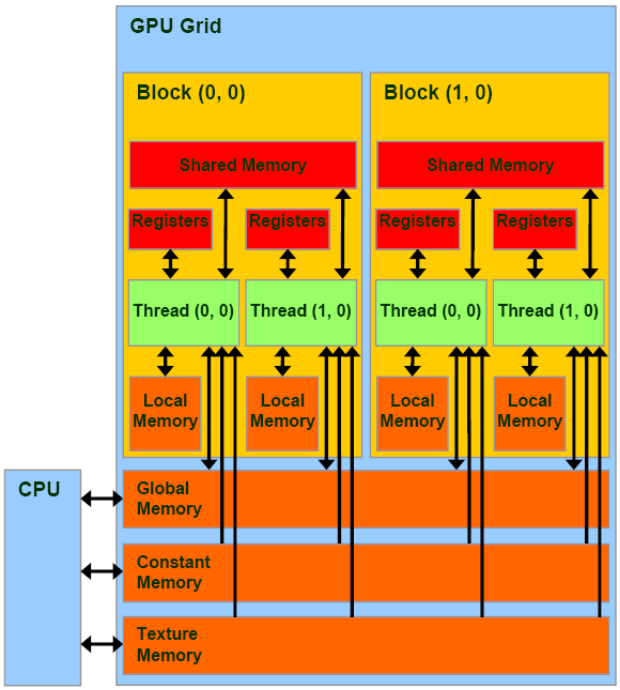


Abbildung 4.14.: Speicher-Modell in CUDA ([van11])

4.3.4. Diskussion

4.3.4.1. Beobachtungen zum Laufzeitverhalten

Die überwiegende Mehrheit von Verfahren zur Kollisionserkennung basiert auf der Verwendung von polygonalen Geometriebeschreibungen in Kombination mit Hüllkörper-Hierarchien zur Reduzierung der Anzahl von Paar-Tests zwischen Seitenflächen. Analysiert man basierend auf einer Aufwandsabschätzung (Unterabschnitt 4.2.2) das Laufzeitverhalten sowohl CPU- als auch GPU-gestützter Implementierungen (Unterabschnitt 4.2.3), so ist es die paarweise Traversierung von Hüllkörper-Hierarchien, die im Verhältnis zu tatsächlich durchgeführten Paartests zwischen Seitenflächen einen mindestens so großen, in vielen Fällen sogar größeren Anteil an der benötigten Rechenzeit pro Iteration einer Simulation aufweist. Diese Beobachtung stimmt grundsätzlich mit der Aufgabenstellung an dies Hüllkörper-Hierarchien überein: Der Vermeidung einer Laufzeitkomplexität von $O(n^2)$ bei einer umfassenden Ausführung von Paar-Tests zwischen allen möglichen Kombinationen von Seitenflächen eines Paares polygonaler Geometrien.

Dem steht bei der GPU-basierten Umsetzung die Problematik gegenüber, eine Möglichkeit zur effizienten Parallelisierung einer paarweisen Traversierung zu finden. Die in Unterabschnitt 4.3.1 bis Unterabschnitt 4.3.3 vorgestellten Verfahren verfolgen dabei unterschiedliche Herangehensweisen:

- Eine komplett GPU-seitige, nicht synchronisierte Expansion der Traversierung in Kombination mit einer Rebalancierungs-Phase bei gProximity.
- Ein hybrider Ansatz in Form einer mehrkern-unterstützten, CPU-seitigen Expansion der Traversierung in Kombination mit GPU-gestützter Ausführung bei HPCCD.

- Bedingt durch den Unterschied in der Aufgabenstellung zur Pfadplanung bei Erbes und Mit-Autoren: Der Verzicht auf die Synchronisierung innerhalb einer paarweisen Traversierung, die gegenüber Implementierungen mit mehreren GPU-Threads oder Thread-Gruppen pro Traversierung ein enorm besseres Laufzeitverhalten zeigen.

Eine weitere, hier nicht näher betrachtete Variante liefert Tang in [TMLT11]: Hier wird ein front-tracking basierter Ansatz verwendet, um basierend auf der Kollisionsfront aus einer vorhergehenden Iteration direkt mit den Eltern- und Kind-Knoten im BVTT-Baum in der Traversierung zu beginnen, anstatt in jeder Iteration eine komplette Hüllkörper-Traversierung vorzunehmen.

Als zweiter wichtiger Aspekt neben der Minimierung von Abhängigkeiten zwischen Ausführungseinheiten verdeutlichen die vorgestellten Verfahren die besondere Bedeutung einer effizienten Speicherverwaltung im Hinblick auf Lese- und Schreibzugriffe zwischen den Speicherbereichen des Gast-Systems und dem eigenen Speicherbereich einer GPU. Da die Datenübertragung zwischen diesen verschiedenen Speichersystemen unter Umständen um den Faktor 100 bis 1000 langsamer erfolgt als der Zugriff auf den geräte-eigenen Speicher, ist ebenso auf eine Minimierung des Datentransfers zwischen Gast-System und GPU zu achten:

- gProximity löst dieses Problem, indem die Warteschlangen-Verwaltung während einer Hierarchie-Traversierung komplett im Speicherbereich der GPU erfolgt.
- HPCCD bewältigt dieses Problem durch eine gastsystem-seitige Warteschlange, in der Anforderungen für Paartests so lange zusammengefasst werden, bis eine bestimmte Speicherblock-Größe erreicht wird, die dann in einem einzigen Transfer in den Speicherbereich der GPU übertragen wird.

Die von Erbes gemachten Versuche zeigen weiterhin, wie sich die Verwendung verschiedener Speicherbereiche innerhalb einer GPU selbst auf das Leistungsverhalten von Kollisionserkennungs-Verfahren auswirken kann: Die Nutzung eines thread-lokalen Speicherbereichs für die Verwaltung eines Stapelspeichers für eine Tiefen-Suche hatte allein aufgrund der vorhandenen Speicher-Latenz einen massiven Leistungsabfall zur Folge, der die potentiell zu erzielenden Laufzeit-Vorteile durch ein für CPU-Implementierungen besser geeignetes Suchverfahren komplett neutralisiert hat.

Was den Transfer der Datenstrukturen für Hüllkörper-Hierarchien und polygonale Geometriebeschreibungen selbst betrifft, ist in allen vorgestellten Verfahren jedoch ein Aspekt nur am Rand oder sogar gar nicht berücksichtigt worden: Die meisten Verfahren gehen davon aus, dass die Geometrie-Daten selbst a priori in den GPU-Speicherbereich transferiert werden, und es während der Laufzeit einer Simulation nicht notwendig ist, zusätzliche Geometrie-Daten nachzuladen.

4.3.4.2. Zur Struktur von Hüllkörper-Hierarchien

Verbunden mit der potentiell erreichbaren Parallelisierung im Verlauf der Traversierung existiert noch ein zweiter Freiheitsgrad in Form des Verzweigungsgrades einer Hüllkörper-Hierarchie: Die Mehrheit der vorgestellten Verfahren beschränkt sich hier auf binäre Bäume, als einzige Ausnahme weicht gProximity für die Berechnung des geringsten Abstands zwischen zwei Geometrien davon ab, indem es Bäume vom Verzweigungsgrad acht verwendet,

um einen höheren Parallelisierungsgrad bei der kompletten Traversierung einer Hüllkörper-Hierarchie zu erreichen.

Trotz einfacherer Verfahren zur Konstruktion und Traversierung von binären Bäumen ist die Benutzung von Hüllkörper-Hierarchien mit höherem Verzweigungsgrad eine Alternative, die sich grundsätzlich sehr gut für massiv parallel arbeitende Kollisionserkennungs-Verfahren eignet: Wechselseitige Abhängigkeiten zwischen Ausführungseinheiten sind bei solchen Datenstrukturen nur noch bedingt gegeben.

Abseits von der Struktur einer Hüllkörper-Hierarchie, die an sich Auswirkungen auf das in GPU-Architekturen verwendete, hardware-immantente Scheduling hat, ist die Ablauf-Logik in einem GPU-Kernel an sich ein weiterer Gesichtspunkt, der beachtet werden muss: Da bei einer Hüllkörper-Traversierung generell nicht a priori absehbar ist, wie viel Laufzeit die Traversierung einer Teil-Hierarchie benötigen wird, müssen Maßnahmen ergriffen werden, um eine ungleichmäßige Auslastung unterschiedlicher Thread-Gruppen auf einer GPU zu vermeiden. Bei gProximity geschieht dies wiederum über eine explizite Rebalancierung der Traversierung von Teil-Hierarchien, während HPCCD dieses Problem dadurch zu umgehen versucht, indem GPU-basierte Berechnungen erst in einer gastsystem-seitigen Warteschlange gruppiert werden, bevor sie tatsächlich auf der GPU angestoßen werden.

Beim in Kapitel 5 vorgestellte Ansatz wird diesem Problem auf eine andere Art begegnet: Anstatt tiefe Hüllkörper-Hierarchien paarweise zu traversieren, werden Kollisionsgeometrien zwar ebenfalls mit Hüllkörpern umschlossen, die allerdings in sehr flachen Hierarchien mit einem sehr hohen Verzweigungsgrad angeordnet sind. Die individuellen Hüllkörper umschließen dabei nicht nur einzelne Seitenflächen, sondern größere Teile einer Kollisionsgeometrie. Kollisions-Tests werden dementsprechend nicht mehr in Form einzelner Paar-Tests, sondern für größere Gruppen von Seitenflächen vorgenommen. Die paarweise Traversierung von Hüllkörper-Hierarchien wird so durch eine sehr leicht parallelisierbare, zwischen einzelnen Gruppen von Seitenflächen unabhängig erfolgende Verarbeitung ersetzt.

4.3.4.3. Zum Leistungsumfang der vorgestellten Beispiele

Die Unterstützung elastischer oder verformbarer Geometrien sowie die Fähigkeit zu dynamischer Kollisionserkennung (unter Berücksichtigung von Objekt-Eigenbewegungen während einer Simulations-Iteration) sind für viele Anwendungsbereiche wichtig und hilfreich. Betrachtet man jedoch die erforderliche Funktionalität, wie sie etwa in Szenarien im Bereich der virtuellen Inbetriebnahme (Abschnitt 2.1) vorkommen, so ist in Abgrenzung zu anderen Anwendungsbereichen für echtzeitfähige Starrkörpersimulationen Folgendes festzustellen:

1. Die in den vorhergehenden Abschnitten vorgestellten, GPU-basierten Kollisionserkennungs-Verfahren beschäftigen sich nahezu ausschließlich mit Szenen, in denen nur einige wenige zwar komplex strukturierte Objekte interagieren, nicht jedoch mit Umgebungen, in denen eine größere Anzahl von Objekten interagiert.
2. Die Anwendungsgebiete von Simulationen aus der Industrie- und Service-Robotik, die ihrerseits zur Simulation mechanischen Verhaltens in Echtzeit Starrkörpersimulations-Pakete aus dem Bereich der Unterhaltungs- und Spiele-Industrie heranziehen, sind zwar durchaus zur Behandlung von Szenen mit einer größeren Anzahl von Objekten geeignet,

dabei müssen jedoch abhängig von der konkreten Aufgabenstellung Einschränkungen in der geometrischen Komplexität von Kollisions-Geometrien in Kauf genommen werden.

Da die performante Kollisionserkennung zwischen komplex strukturierten Geometrien in moderater Anzahl im Vordergrund dieser Arbeit steht, wird der Schwerpunkt beim Entwurf eines eigenen Kollisionserkennungs-Verfahrens auf der statischen Kollisionserkennung für starre Körper liegen. Die mögliche Erweiterung des Funktionsumfangs dieses Verfahren um die Unterstützung verformbarer Körper und die Integration dynamischer Kollisionserkennung wird kurz in Kapitel 7 behandelt werden.

4.4. Raumpartitionierungs-basierte Verfahren

Eine weitere Art von Verfahren für die Kollisionserkennung, auf die bisher noch nicht ausführlich eingegangen worden ist, sind *Raumpartitionierungs-Schemata*. Bei diesen Verfahren wird ein Bereich des dreidimensionalen Raums in disjunkte Teilregionen zerlegt, und Objekte in einer Simulation werden in Bezug auf eine solche Partitionierung mit der oder den Teilregionen assoziiert, mit denen eine Überschneidung besteht. Im Gegensatz zu Hüllkörper-Hierarchien passt sich also die Partitionierung einer Raum-Region nicht der Struktur einer Eingabe-Geometrie an, sondern ordnet Eingabe-Geometrien in eine bestehende Partitionierung ein.

Als mögliche Datenstrukturen für Raumpartitionierungs-Schemata stehen verschiedene Alternativen zur Verfügung:

- Raum-Gitter: Regelmäßig, adaptiv oder hierarchisch
- Bäume: Octree, k-d-Bäume, BSP-Bäume (Binary Space Partitioning)
- Sortierverfahren wie Sweep and Prune (SAP, Unterunterabschnitt 3.4.5.2)

Raumpartitionierungs-Schemata werden in vielen Fällen für die Vorfilter-Phase eingesetzt. Allerdings gibt es auch Verfahren mit ähnlichem Funktions-Prinzip, die für die Verwendung in der Objektpaar-Phase geeignet sind, wie der Voxmap-Pointshell-Algorithmus (Unterabschnitt 4.4.3).

4.4.1. Gitter-Strukturen

Gitter-Strukturen erlauben die schnelle Indizierung von einzelnen Teilregionen, ebenso wie einen schnellen Zugriff auf Nachbarzellen. Nur Objekte, die in den gleichen Zellen eines Gitters liegen, müssen in detaillierteren Kollisionstests berücksichtigt werden.

Während der Zugriff auf Elemente eines Gitters und die Suche nach potentiell kollidierenden Objekten oder Objekt-Teilen sehr effizient ist, so gibt es für Gitter-Strukturen jedoch auch einige Nachteile, die ihre Nützlichkeit vor allem in der Objektpaar-Phase einschränken:

- Die Wahl der Zellengröße: Die Größe einzelner Zellen entscheidet darüber, wie gut ein Gitter dazu in der Lage ist, eine eindeutige Zuordnung assoziierter Objekte zu gewährleisten: Sind Objekte im Verhältnis zu Gitterzellen sehr klein, können unter Umständen viele Objekte nur wenigen Zellen zugeordnet sein. Das führt wiederum

dazu, dass viele unnötige Paar-Tests durchgeführt werden. Ist andererseits die Größe von Objekten im Verhältnis zu Gitterzellen sehr groß, so müssen Objekte vielen Zellen zugeordnet werden, was wiederum die Anzahl unnötiger Paartests erhöhen kann. Diesen Problemen kann zwar durch die Verwendung adaptiver Gitter, die etwa in Bereichen mit hoher Objekt-Dichte ein Teilgitter mit kleineren Gitterzellen verwenden oder durch hierarchische Gitter, die für denselben Raumbereich mehrere Gitter-Strukturen mit unterschiedlichen Zellengrößen verwenden, begegnet werden; das erhöht jedoch den Aufwand der Verwaltung der Datenstrukturen und erhöht den Speicherplatzbedarf.

- Die Zuordnung von Objekten: Für komplex strukturierte Geometrien kann die Bestimmung der Zuordnung zu Gitterzellen sehr aufwendig sein, da potentiell für jedes einzelne Geometrie-Element die Einordnung in eine Gitterstruktur bestimmt werden muss. Daher wird die Zuordnung oft nur auf Basis von Hüllkörpern vorgenommen, wie Hüllkugeln oder AABBs, für die sich die Überschneidung mit Gitterzellen einfach bestimmen lässt.
- Der Speicherplatzbedarf: Abhängig von der Größe der unterteilten Raumregion, der Granularität der Unterteilung und der Anzahl oder geometrischen Komplexität behandelter Objekte kann der Speicherplatzbedarf einer Gitterstruktur sehr groß werden. Auch diesem Problem kann mit Hilfe speichereffizienter Implementierungen, die ohne die vollständige Allokation aller Gitterzellen im Speicher (etwa durch die Verwendung von Hashing) funktionieren, begegnet werden; allerdings steht auch hier der größeren Speichereffizienz eine aufwendigere Verwaltung und Aktualisierung der verwendeten Datenstrukturen gegenüber.
- Aktualisierung nach Objektbewegungen: Wenn sich Objekte bewegen, muss deren Zuordnung zu Gitterzellen neu bestimmt und aktualisiert werden.

4.4.2. Baum-Datenstrukturen in der Raumpartitionierung

Neben Gitter-Strukturen existieren auch Baum-Datenstrukturen zur Anwendung als Raumpartitionierungen. Anders als bei Hüllkörper-Hierarchien werden hier jedoch nicht Hüllvolumina hierarchisch angeordnet: Vielmehr werden zur Erstellung einer Raumpartitionierung Baumknoten solange anhand von Partitionierungsregeln erzeugt, bis eine Raumregion vollständig umschlossen ist, beziehungsweise bis zum Erreichen eines bestimmten minimalen Baumknoten-Volumens partitioniert wurde.

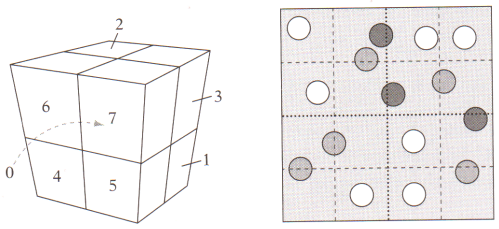
Im Gegensatz zu den bisher behandelten Hüllkörper-Hierarchien ist es jedoch bei diesen Baum-Strukturen so, dass sich die mit inneren und Blatt-Knoten assoziierten Volumina nicht an die umschlossenen Geometrie-Teile oder Geometrien adaptieren: Entweder werden diese einem Kind-Volumen eindeutig zuordnet, oder sie werden mit allen geschnittenen Kind-Volumina assoziiert. Darüber hinaus gibt es sogar die Variante, eine Eingangs-Geometrie entsprechend ihrer Zugehörigkeit zu verschiedenen Volumina in der Baum-Struktur aufzuteilen und damit die topologische Struktur der Eingangs-Geometrie zu verändern.

Die verschiedenen Arten von Bäumen in Raumpartitionierungs-Schemata unterscheiden sich in der Form der Partitionierungsregel, die zur Unterteilung der umschlossenen Raumregion während der Konstruktion eines Baumes genutzt wird. Die verschiedenen Partitionie-

rungsregeln unterscheiden sich dabei in der Flexibilität der Auswahlmöglichkeiten der zur Unterteilung verwendeten geometrischen Primitive (Geraden in 2D, Ebenen in 3D).

4.4.2.1. Octree

Octrees ([Mea80], [Mea82]) beziehungsweise Quad-Trees unterteilen eine Raumregion rekursiv für jeden inneren Knoten im Baum in acht beziehungsweise vier gleich große Teilregionen. Als Hüllkörper für den Wurzelknoten wird ein achsenorientierter Würfel verwendet (Abbildung 4.15a).



(a) Struktur eines Octree mit Indizierung von Kindknoten

(b) Einordnung von Objekten in einen Quad-Tree

Abbildung 4.15.: Die Struktur von Octrees und die Einordnung von Objekten in einen Quad-Tree ([Eri05, Kapitel 7]): In Abbildung 4.15b sind Objekte schattiert hinterlegt, die nicht eindeutig einem Blattknoten im Baum zugeordnet werden können, und daher von mehreren Blattknoten referenziert werden müssen.

Eine Raumregion wird jeweils entlang der x-,y-, und z-Ebene in zwei Halbräume unterteilt; die Parameter der Ebenen müssen dabei nicht explizit in der Datenstruktur des Baumes festgehalten werden. Die geometrische Struktur der Baumknoten ergibt sich einfach durch die Halbierung der Abmessungen des Eltern-Knotens entlang aller drei Koordinaten-Achsen. Diese Art der Aufteilung ist bei räumlich ausgedehnten Objekten nicht sehr flexibel: Es kommt bedingt durch die Konstruktion von Octrees sehr häufig dazu, dass Objekte mit mehreren Knoten im Baum assoziiert werden müssten (Abbildung 4.15b). In solchen Fällen kann ein Objekt entweder aufgeteilt werden, oder es wird mit demjenigen inneren Octree-Knoten assoziiert, in dem es noch als Ganzes enthalten ist. Diese Problematik, und die Verwaltung bewegter Objekte bei der erneuten Einordnung nach einer Objektbewegung sind gravierende Nachteile für die Verwendung von Octrees in der Kollisionserkennung beim Umgang mit polygonal beschriebenen Geometrien (und allgemein mit räumlich ausgedehnten Objekten).

Diese Einschränkung gilt allerdings nicht für Punkte als Primitive: Da hier konzeptionell keine Probleme mit der räumlichen Ausdehnung des geometrischen Primitivs bei der Einordnung in einen Octree gegeben sind, sind Octrees ein optimales Werkzeug bei der Indizierung auch großer, unstrukturierter Punktmengen. Diese Eigenschaft wird im weiteren Verlauf der Arbeit (Kapitel 5) noch eine wichtige Rolle spielen.

Auch weisen Octrees aufgrund ihres regelmäßigen Aufbaus bezüglich des benötigten Speicherplatzes sehr günstige Eigenschaften auf: So können etwa die Kindknoten eines inneren

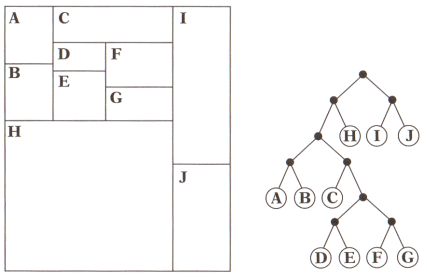
Knoten im Baum über ein Bit-Feld kodiert werden, ohne eine komplexere Datenstruktur allokieren zu müssen, die explizite Verweise auf Kindknoten enthält.

Einige Beispiele für den Einsatz von Octrees in Anwendungsfeldern außerhalb der Kollisionserkennung finden sich etwa in:

- Noborio und Mitautoren ([NFA88]): Ein frühes Beispiel für die Anwendung von Octrees zur Pfadplanung in der Robotik
- OctoMap ([HWB+13]): Eine octree-basiertes Verfahren zur Umgebungs-Modellierung in der Robotik
- Sparse Voxel Octrees ([LK11]): Octree-basiertes Raycasting-Verfahren in der Computergrafik

4.4.2.2. k-d-Bäume

k-d-Bäume ([Ben75], [FBF77]) benutzen eine flexiblere Partitionierungsregel: Anstatt einer Halbierung entlang jeder Koordinaten-Achse wie beim Octree erfolgt die Aufteilung bei k-d-Bäumen pro Baum-Ebene nur entlang einer einzelnen Koordinaten-Achse, und das Verhältnis der Partitionierung ist dabei nicht fixiert (Abbildung 4.16a, Abbildung 4.16b). Allerdings stehen die Ebenen, die in jeder Baum-Ebene die Unterteilung in Kind-Volumina bestimmen, immer orthogonal zueinander, beziehungsweise orthogonal zu den Achsen des Bezugs-Koordinatensystems. Die Normalen-Richtung und das Partitionierungs-Verhältnis muss als Teil der Datenstruktur eines k-d-Baums festgehalten werden.



(a) Partitionierung **(b)** Struktur des Baums

Abbildung 4.16.: Die Struktur eines k-d-Baums in 2D ([Eri05, Kapitel 7]): Die Partitionierungsregel, welche den Baum zyklisch entlang der Koordinaten-Achsen eines Bezugs-Koordinatensystems unterteilt, ist gut bei entlang der Aufteilung der dritten Hierarchie-Ebene in Abbildung 4.16b zu erkennen: Die Aufteilung zwischen den Knoten I und J und dem restlichen linken Teilbaum erfolgt entlang der Y-Achse, die Trennung zwischen I und J selbst entlang der X-Achse.

Für k-d-Bäume sind bei Octrees angesprochenen Probleme bei der Einordnung von Objekten in die Baumstruktur nicht so schwerwiegend, da aufgrund der flexibleren Partitionierungsregeln eine bessere Adaption der Baumstruktur an Eingangs-Geometrien möglich ist, und die Beschränkung auf eine partitionierende Ebene pro Hierarchie-Stufe erleichtert die Verwaltung und Traversierung der Baumstruktur.

Dennoch sind k-d-Bäume grundsätzlich mit denselben Nachteilen behaftet wie Octrees, was die Verwendung in der Kollisionserkennung mit räumlich ausgedehnten Objekten betrifft.

Wesentlich besser geeignet sind k-d-Bäume wiederum für Operationen auf Punkt-Mengen, wie etwa die schnellen Bestimmung der Zugehörigkeit eines Punktes zu einer bestimmten Raumregion, die Lokalisierung der nächsten Nachbarn von Punkten, oder die Ermittlung aller Punkte innerhalb einer bestimmten Raumregion. Daher sind k-d-Bäume ebenso wie Octrees besonders für den Umgang mit Punktmengen geeignet.

4.4.2.3. BSP-Bäume

BSP-Bäume ([FKN80]) sind eine Generalisierung von k-d-Bäumen: Die partitionierenden Ebenen müssen nicht mehr orthogonal zu den Achsen eines Bezugs-Koordinatensystems liegen, sondern können beliebig positioniert und orientiert sein (Abbildung 4.17). Ansonsten entspricht die Funktionsweise dieser Datenstruktur der von k-d-Bäumen, mit denselben Vor- und Nachteilen.

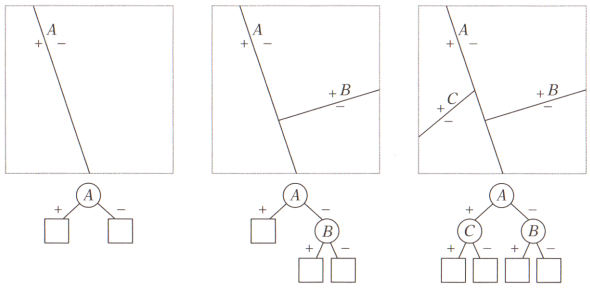


Abbildung 4.17.: Struktur eines BSP-Baums in 2D ([Eri05, Kapitel 8]): Die Partitionierungsregel verzichtet gegenüber k-d-Bäumen auf einer Orientierung der Partitionierung anhand der Koordinaten-Achsen des Bezugs-Koordinatensystems, und erlaubt damit eine beliebige Orientierung.

Die verwendete Partitionierungs-Regel erlaubt zwar eine nahezu beliebig flexible Anpassung an die Struktur von Eingangs-Geometrien, bedeutet aber auch einen wesentlich höheren Laufzeitaufwand bei der Konstruktion und vor allem bei der Adaption der Baumstruktur an Änderungen zur Laufzeit. Daher werden BSP-Bäume generell nur für statische beziehungsweise unbewegliche Geometrien verwendet, für die ein BSP-Baum vor Start einer Simulation berechnet wird.

Aus diesem Grund werden BSP-Bäume im weiteren Verlauf der Arbeit nicht weiter im Zusammenhang mit Aufgaben in der Kollisionserkennung betrachtet, da sich die in Kapitel 5 beschriebenen eigenen Verfahren besonders auf die Kollisionserkennung zwischen bewegten Objekten konzentrieren.

Die ursprüngliche Anwendung von BSP-Bäumen war die ansichts-unabhängige Berechnung gegenseitiger Verdeckungen in dreidimensionalen Szenen. Während der Rasterisierung einer Szene ist es mit Hilfe eines vorberechneten BSP-Baums einfach möglich, die Sortierung geometrischer Primitive in Bezug auf eine bestimmte Ansicht einer Szene zu ermitteln und somit festzulegen, welche Geometrien bei der Rasterisierung berücksichtigt werden müssen.

4.4.3. Der Voxmap-Pointshell-Algorithmus

Basierend auf dem Prinzip der Raumpartitionierung ist der Voxmap-Pointshell-Algorithmus ([MPT99], [MPT06]) eine Variante zur Kollisionserkennung, die im Gegensatz zur Mehrheit der bislang vorgestellten Verfahren nicht auf der direkten Verwendung von polygonalen Oberflächenbeschreibungen basiert. Anstatt ein Objekt über eine Kombination aus verschiedenen geometrischen Primitiven (Ecken, Kanten, Flächen) zu spezifizieren, wird in diesem Verfahren eine duale Repräsentation von Objekten in einer Simulation verwendet:

- Statische Geometrien in einer Szene werden mittels Raumpartitionierung zu einem einzigen Geometrie-Ensemble zusammengefasst: Der *Voxmap* (Volume Occupancy Map).
- Bewegte Objekte in einer Szene werden über eine Punkthülle approximiert, die mittels hoch aufgelöstem Sampling der Objekt-Oberfläche erzeugt wird: Die *Point-Shell* oder *Punkt-Hülle*.

Der Algorithmus ist nach den Autoren für die Simulation eines einzelnen bewegten Objekts in einer komplex strukturierten, aber im Rahmen der Simulation statischen Umgebung konzipiert. Dabei ist das Verfahren in der Lage, im einstelligen Millisekunden-Bereich Kollisionen zwischen dem bewegten, von einem Benutzer über ein haptisches Eingabegerät gesteuerten Objekt und der statischen Szene zu detektieren. Der ursprüngliche Einsatzbereich des Algorithmus sind Zugänglichkeits-Untersuchungen anhand eines virtuellen Modells, wie sie etwa bei der Planung von Wartungs-Vorgängen erforderlich sind. Dabei wird davon ausgegangen, dass die simulierten Vorgänge unter Berücksichtigung einer minimalen Toleranz bei der Berechnung und Auflösung von Kontakten zwischen Objekten durchgeführt werden können.

Der Kollisions-Test in diesem Verfahren besteht in jeder Iteration darin, die Punkt-Hülle des bewegten Objekts in die Voxmap der statischen Umgebung einzuordnen (Abbildung 4.18a). Jedem Punkt aus der Punkt-Hülle ist zusätzlich eine Kontakt-Normale zugeordnet, deren Richtung aus der Krümmung der Oberfläche der Eingangs-Geometrie in der Umgebung des Punktes bestimmt wird. Sofern ein Kontakt zwischen einem Voxel nahe der Oberfläche der Eingabe-Geometrie und einem Element der Punkt-Hülle festgestellt wird, so wird für den betreffenden Punkt eine Eindring-Tiefe ermittelt: Durch den Mittelpunkt des Voxels wird eine Tangential-Ebene gelegt, deren Normale mit der dem Punkt assoziierten Kontakt-Normalen identisch ist. Die Eindring-Tiefe wird durch den Abstand des Punktes zur Tangential-Ebene bestimmt (Abbildung 4.18b).

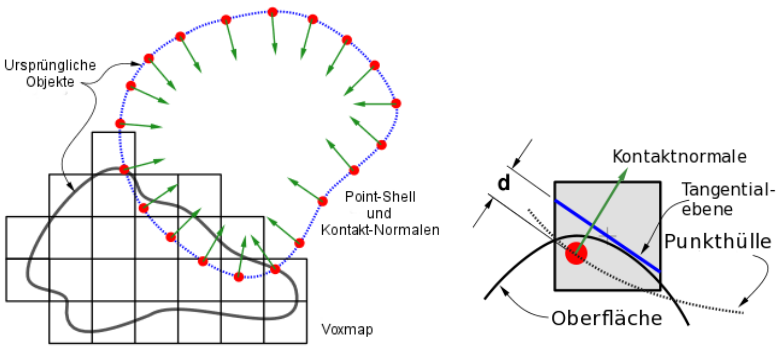
Die Kontakt-Kraft ergibt sich proportional zur einer konfigurierbaren Repulsion K_{ff} mit der Eindring-Tiefe d nach dem Hooke'schen Federgesetz in Richtung der Kontakt-Normalen zu

$$\vec{f} = -\vec{d} \cdot K_{ff}$$

.

4.4.3.1. Die Voxmap

Die Erstellung der Voxmap-Datenstruktur bedient sich eines regulären Gitters, das für ein statisches Objekt erzeugt wird. Anhand des Gitters wird eine Klassifikation jedes Voxels in innere, Oberflächen-, oberflächen-nahe und äußere Voxel vorgenommen. Als Nachbar-



(a) Der Kollisions-Test

(b) Ermittlung einer Kontaktpunkt-Konfiguration

Abbildung 4.18.: Der Voxmap-Pointshell-Algorithmus ([MPT99]): Der Kollisions-Test des Verfahrens überprüft, ob und in welchen Oberflächen- oder oberflächen-nahen Zellen der Raumpartitionierung des statischen Teils der Szene Elemente aus der Punkt-Hülle eines aktiv bewegten Objekts liegen. Anhand des Abstands zur Tangential-Ebene der approximierten Oberfläche kann eine Eindring-Tiefe bestimmt werden, die zusammen mit der einem Punkt assoziierten Kontakt-Normalen einen Kontaktpunkt spezifiziert.

Voxel von Oberflächen-Voxeln werden alle Voxel klassifiziert, die eine gleiche Kante oder Seitenfläche einer Eingabe-Geometrie beinhalten.

Eine Alternative zur Ermittlung dieser Klassifikation wäre beispielsweise die Verwendung des Marching-Cube-Algorithmus ([LC87]). Sagardia beschreibt in [SHPH08] eine andere Vorgehensweise: Hier wird ausgehend von einer polygonalen Eingabe-Geometrie zuerst ein Schnitt-Test der AABB jeder Seitenfläche mit dem regulären Gitter vorgenommen; alle Voxel, die innerhalb einer AABB liegen, werden anschließend einem detaillierteren Schnitt-Test mit der Seitenfläche selbst unterzogen und entsprechend der relativen Lage zur Seitenfläche klassifiziert.

Würde eine Voxmap vollständig im Speicher allokiert werden, wäre der Speicherplatzbedarf für Szenen, wie sie von den Autoren des Original-Algorithmus beschrieben werden, enorm. Daher wird eine modifizierte Octree-Datenstruktur vorgeschlagen: Die Tiefe des erzeugten Octree wird auf drei Ebenen beschränkt, und anstatt nur acht Kindknoten in jeder Ebene zu erzeugen, wird von den Autoren vorgeschlagen, einen Baum mit 2^{3N} Kindknoten mit $N \in 1, 2, \ldots, 5$ zu verwenden. Außerdem wird die minimale und maximale Größe des Volumens von Knoten im Baum beschränkt: Die minimale Größe entspricht dabei der Größe der Blatt-Voxel, und die maximale Größe ergibt sich implizit aus der auf drei Ebenen beschränkten Höhe der Baum-Struktur.

4.4.3.2. Die Punkt-Hülle

Die Punkt-Hülle wird vom ursprünglichen Algorithmus erzeugt, indem alle Mittelpunkte von Oberflächen-Voxeln des bewegten Objekts in die Punkt-Hülle aufgenommen werden. Als weitere Methode zur Bestimmung einer Punkt-Hülle schlägt Sagardia ([SHPH08]) eine Alternative vor, die als nichtlineares Optimierungsproblem formuliert ist: Für die Mit-

telpunkte aller Oberflächen-Voxel werden dabei diejenigen Punkte auf der Oberfläche der Eingabe-Geometrie ermittelt, die ersteren jeweils am nächsten liegen.

Der entscheidende Vorteil einer Punkt-Hülle in Kombination mit einer Raumpartitionierung besteht im sehr effizienten Schnitt-Test zwischen den beiden Datenstrukturen: Anders als bei auf polygonalen Geometrien basierenden und mittels Hüllkörper-Hierarchien beschleunigten Kollisionerkennungs-Verfahren ist beim Voxmap-Pointshell-Algorithmus die Verwendung eines „Brute Force"-Ansatzes zum Testen jedes Punktes aus einer Punkt-Hülle gegen die Voxmap unproblematisch.

Das ist in durch die Form des Kollisions-Tests begründet, für den pro zu testendem Punkt nur folgende Schritte notwendig sind:

- Das Berechnen des Knoten-Index in der Voxmap, in der ein Punkt liegt.
- Überprüfen, ob es sich beim berechneten Knoten-Index um ein oberflächen-nahes oder Oberflächen-Voxel handelt. Dazu ist ein Zugriff auf das zum Knoten-Index gehörende Voxel in der Voxmap-Datenstruktur erforderlich.

Der Knoten-Index k_p eines Punktes p ergibt sich für einen Octree etwa durch:

$$k_p = \begin{pmatrix} \frac{p_x - x_{min}}{s_v} \\ \frac{p_y - y_{min}}{s_v} \\ \frac{p_z - z_{min}}{s_v} \end{pmatrix} \tag{4.2}$$

mit s_v als Voxel-Größe und $(x_{min}, y_{min}, z_{min})$ als minimale Koordinaten des Wurzelknotens des Octree.

Der Zugriff auf ein dem Knoten-Index k zugeordnetes Voxel kann beispielsweise über eine rekursive Suche im Octree erfolgen, wie in Algorithmus 6 (für einen konventionellen Octree) skizziert.

Funktion `findLeafRecursive`

Input : $p_x, p_y, p_z, depthMask, rootNode$

1 // Index des gesuchten Voxels, Bit-Maske zur Indizierung der Baum-Tiefe, Wurzelknoten des Octree

2 $i_{child} = (((!!(p_x \ \& \ depthMask)) \ll 2) | ((!!(p_y \ \& \ depthMask)) \ll 1) | (!!(p_z \ \& \ depthMask)))$;

3 $n_{child} = \text{rootNode.child}(i_{child})$;

4 **if** $n_{child}.valid \wedge n_{child}.nodeType == INNER$ **then**

5 findLeafRecursive$(p_x, p_y, p_z, \frac{depthMask}{2}, n_{child})$;

6 **else**

7 return n_{child};

Die in Algorithmus 6 skizzierte Suche nach Blattknoten in einem Octree ist besonders für Bäume mit künstlich beschränkter Höhe, wie von den Autoren des Original-Algorithmus vorgeschlagen, eine sehr performante Such-Operation. Obwohl sie für jedes Element der Punkt-Hülle einmal durchgeführt werden muss, ist der Laufzeitaufwand dieses Kollisions-

Algorithmus 6 : findLeaf()

Input : p_x, p_y, p_z

```
1 // Index des gesuchten Voxels
2 findLeafRecursive(p_x, p_y, p_z, depth_mask, root_node);
```

Tests gering genug, um einen Betrieb der Kollisionserkennung im einstelligen Millisekunden-Bereich zu erlauben.

Da eine Punkt-Hülle keine topologisch geschlossene Oberfläche darstellt, wird die gegenseitige Kollisionserkennung zwischen mehreren aktiv bewegten Objekten (also Kollisions-Tests zwischen zwei Punkt-Hüllen) nicht unterstützt. Ebenso ist ein expliziter Toleranzabstand in Größe der Voxel-Auflösung zwischen statischer Umgebung und bewegten Objekten eine Einschränkung, die explizit von den Autoren für den Algorithmus gemacht wird.

Eine mögliche Parallelisierung des Verfahrens wird von den Autoren des Original-Algorithmus bereits angedacht; Lawlor ([LK02]) beschreibt eine mögliche Implementierung eines sehr ähnlichen Verfahrens für Mehrkern-Architekturen.

4.4.4. Diskussion

Voxel-basierte Verfahren basieren auf einem Ansatz, der durch die Reduktion eines Kollisionstests auf die Einordnung von Punkt-Primitiven in eine Raumpartitionierung sehr attraktive Laufzeit-Charakteristiken aufweist. Das Laufzeitverhalten des Voxmap-Pointshell-Algorithmus, der zum Zeitpunkt der Veröffentlichung des Verfahrens im Jahr 1999 bereits in der Lage war, Kollisionserkennung in einer komplex strukturierten Umgebung mit Iterations-Zeiten im Bereich weniger Millisekunden durchzuführen, zeigt das Leistungspotenzial dieser Art von Verfahren. Wie bereits von den Autoren des ursprünglichen Algorithmus angesprochen, sind voxel-basierte Verfahren prinzipiell sehr gut für eine parallelisierte Ausführung geeignet: Durch den Wegfall einer Vorfilter-Phase, wie sie bei polygon-basierten (und teilweise auch bei bildraum-basierten) Verfahren benötigt wird, sind prinzipiell keine gesonderten Maßnahmen wie zur Verwaltung und Rebalancierung von Warteschlangen nötig, um eine optimale Auslastung einer massiv parallelen Prozessor-Plattform erreichen zu können. Die Eingangs-Datenmenge in Form einer Punkt-Wolke kann bei nur lesendem Zugriff auf die Datenstruktur einer Raumpartitionierung in disjunkte Teil-Bereiche zerlegt werden, die unabhängig voneinander hinsichtlich ihrer Position innerhalb der Raumpartitionierung überprüft werden können.

Allerdings gibt es auch zwei gravierende Nachteile, die voxel-basierte Ansätze gegenüber polygon-basierten Verfahren haben:

- Der Speicherbedarf für die Daten der Raumpartitionierung der Punkt-Hülle: Zwar kann mittels geeigneter Techniken zur Reduzierung tatsächlich im Speicher allokierter Voxel (etwa durch einen Octree mit beschränkter Höhe), die Beschränkung der pro Voxel angelegten Nutzdaten (nur zwei Bit pro Voxel zur Codierung der Klassifikation in der ursprünglichen Version des Voxmap-Pointshell-Algorithmus) und die Variation der Voxel-Größe (also der Veränderung der voluminalen Auflösung) der Speicherplatzbedarf einer Raumpartitionierung verringert werden. Jedoch bleibt der Speicherplatz-

bedarf ein beschränkender Faktor für voxel-basierte Algorithmen. Das selbe Problem gilt generell auch für Punkt-Hüllen, ist beim Voxmap-Pointshell-Algorithmus jedoch deswegen nicht entscheidend, weil ein aktiv manipuliertes Objekt im Verhältnis zur Gesamtgröße der statischen Umgebung typischerweise eine weitaus geringere geometrische Komplexität und damit auch eine wesentlich kleinere Anzahl von Seitenflächen in seiner ursprünglichen Geometrie aufweist, die in Form einer Punkt-Wolke erfasst werden muss.

- Die Beschränkung auf eine statische Umgebung und ein einziges (oder wenige) durch Benutzer-Eingaben manipulierte Objekte: Dadurch entfällt zwar die Notwendigkeit, eine Raumpartitionierung bei Veränderungen zu aktualisieren; andererseits ist es so nicht möglich, Szenen mit einem höheren Anteil an extern manipulierten, aber auch frei beweglichen Objekten zu simulieren.

Die Einschränkung hinsichtlich des Speicherplatzbedarfs für eine Raumpartitionierungs-Datenstruktur ist hierbei im Hinblick auf eine GPU-gestützte Umsetzung eines voxel-basierten Verfahrens ein besonders schwerwiegendes Problem. Es sollte allerdings nicht außer acht gelassen werden, dass es in Form baum-basierter Datenstrukturen wie Octrees oder k-d-Bäumen möglich ist, eine Raumpartitionierung speichereffizient in serialisierter Form, also nicht unter Verwendung expliziter Speicher-Verweise auf individuelle Kind-Knoten, zu erzeugen und zu traversieren. Das gilt besonders für Octrees, die dank ihres symmetrischen Aufbaus etwa die räumlichen Dimensionen innerer Knoten nicht explizit als Teil ihrer Datenstruktur erfassen müssen. Ebenso wenig ist es bei Octrees nötig, einzelne Kind-Oktanten in inneren Knoten des Baums explizit als eigene Teil-Bäume im Speicher zu allokieren: Ob ein Kind-Oktant im Baum existiert, kann durch einer Bit-Maske mit acht Bit indiziert werden, die (abgesehen von im Baum mit abgelegten Nutzdaten) als einziger Bestandteil pro Octree-Knoten gespeichert werden muss. Darüber hinaus ist die Lokalisierung der Position eines Punktes innerhalb eines Octree wie bereits erwähnt sehr einfach möglich.

Aus diesem Grund wird in Kapitel 5 der Versuch unternommen, den Kollisions-Test des Voxmap-Pointshell-Algorithmus und damit verbunden dessen Datenstrukturen in einer für eine GPU-gestützte Implementierung geeigneten Weise zu adaptieren.

KAPITEL 5

DER EIGENE ANSATZ ZUR KOLLISIONSERKENNUNG

Kapitel 5 beschäftigt sich mit alternativen Anwendungsmöglichkeiten für die in Kapitel 4 vorgestellten Verfahren zur Kollisionserkennung. Im Vordergrund steht dabei der Ansatz, die Vorteile voxel-basierter Verfahren mit denen polygon-basierter Verfahren so zu verbinden, dass die jeweiligen Nachteile beider Verfahrensklassen weitgehend umgangen werden können. Entscheidend wird dabei die Adaption von Datenstrukturen, Speicherzugriffen und Kollisions-Tests für den Einsatz auf GPU-Prozessoren sein.

Abschnitt 5.1 stellt die generelle Herangehensweise des eigenen Ansatzes vor und beschäftigt sich mit die Gemeinsamkeiten mit und den Unterschieden zu den in Kapitel 4 vorgestellten Verfahren.

In Abschnitt 5.2 wird die Vorgehensweise zur Partitionierung von polygonalen Geometriebeschreibungen vorgestellt, die im Rahmen des eigenen Ansatzes als Alternative zur Verwendung von Hüllkörper-Hierarchien verwendet wird.

Abschnitt 5.3 behandelt die Erstellung und Nutzung von punkt-basierten Oberflächenbeschreibungen im Rahmen des eigenen Ansatzes ähnlich zu dem in Unterabschnitt 4.4.3 diskutierten Voxmap-Pointshell-Algorithmus.

Raumpartitionierungen als zweite im eigenen Verfahren verwendete Datenstruktur und deren GPU-basierte Erstellung werden in Abschnitt 5.4 diskutiert.

In Abschnitt 5.5 wird die verwendete Implementierung für Octrees behandelt, und die nötigen Modifikationen für den Einsatz im Rahmen des eigenen Ansatzes zur Kollisionserkennung beschrieben.

Abschnitt 5.6 stellt dieselben Themen in Bezug auf die verwendete k-d-Baum-Implementierung vor.

5.1. Überblick

Das Ziel des im Folgenden vorgestellten Kollisionserkennungs-Verfahrens ist die Kombination von Objekt-Repräsentationen mittels polygonaler Geometriebeschreibungen und der

effizienten Vorfilterung potentiell kollidierender Geometrieteile mittels flacher Hüllkörper-Hierarchien mit hohem Verzweigungsgrad mit der Effizienz des Kollisions-Tests zwischen Punkt-Primitiven und Raumpartitionierungen.

Die finale Bestimmung konkreter Kontaktpunkte zwischen Geometrien soll dabei auf die gleiche Weise wie bei polygonalen Kollisionserkennungsverfahren erfolgen, indem Paare von Seitenflächen auf mögliche Kontakte geprüft werden. Die Bestimmung potentiell kollidierender Seitenflächen soll dagegen nicht ausschließlich durch die paarweise Traversierung von Hüllkörper-Hierarchien erfolgen, sondern basierend auf einer Kombination aus speziell strukturierten Hüllkörper-Verbunden und der Verwendung eines Kollisions-Tests, der ähnlich arbeitet wie im Voxmap-Pointshell-Algorithmus.

Allerdings dient der verwendete Kollisions-Test auf Basis von Punkt-Wolken und Raumpartitionierungen nicht zur direkten Berechnung von Kontaktpunkten, sondern als Indikator bei der Suche nach Regionen in einem Paar von Geometrien, die potentiell kollidierende Seitenflächen enthalten.

Die Motivation hinter dieser hybriden Vorgehensweise lässt sich durch eine genauere Betrachtung der Probleme begründen, die bei polygon-basierten Kollisionserkennungs-Verfahren beim Einsatz auf GPU-Prozessoren auftreten können (diskutiert in Unterabschnitt 4.3.4):

- Die *Parallelisierbarkeit* der Traversierung von Hüllkörper-Hierarchien.
- Die *Gewährleistung einer optimalen Auslastung* von GPU-Prozessoren, sowohl im Hinblick auf die erreichbare Parallelisierbarkeit, als auch auf die nicht a priori planbare Laufzeit bei der Traversierung unterschiedlicher Teil-Hierarchien.
- Die Übertragung von Geometrie- und laufzeit-generierten Daten zwischen den Speicherbereichen des Gast-Systems und der GPU und die dabei zu berücksichtigende *Speicher-Latenz*.
- Der *Speicherplatz-Bedarf* für Geometriebeschreibungen und Hüllkörper-Hierarchien in Szenen mit einer größeren Zahl von Objekten.

Bei voxel-basierte Verfahren sind folgende Nachteile (diskutiert in Unterabschnitt 4.4.4) bei der Verwendung solcher Ansätze zu beachten:

- Der *Speicherplatz-Bedarf* für Raumpartitionierungen ist ebenso wie bei polygon-basierten Verfahren beim Einsatz auf GPUs ein beschränkender Faktor.
- Die *Aktualisierung einer Raumpartitionierung* nach Objektbewegungen ist eine potentiell aufwendige Operation.
- Speziell beim Voxmap-Pointshell-Algorithmus: Die *Beschränkung auf eine einzige bewegte Geometrie* innerhalb einer statischen Umgebung.

Auf die Verwendung polygonaler Geometriebeschreibungen soll im Folgenden nicht verzichtet werden, da diese aufgrund ihrer Spezifikation als oberflächen-basierte Geometriebeschreibung die höchste Präzision (innerhalb der begrenzten Genauigkeit von binären Fließkommazahl-Formaten) und damit die beste Approximation eines geometrischen Modells bereitstellen. Die übliche duale Beschreibung einer Kollisions-Geometrie durch ein Dreiecks-Netz in Kombination mit einer umschließenden Hüllkörper-Hierarchie wird jedoch im Folgenden um zwei Repräsentationen ergänzt, die durch die Datenstrukturen des Voxmap-Pointshell-Algorithmus inspiriert ist. Im Gegensatz zu diesem Algorithmus werden diese jedoch nicht getrennt für

ein aktiv bewegtes Objekt und eine statische Umgebung angelegt; vielmehr werden für jedes Objekt eine eigene Punkt-Hülle und eine eigene Raumpartitionierung in einem objektspezifischen Koordinatensystem erzeugt und zusätzlich zur polygonalen Eingabe-Geometrie gespeichert.

Auch die Struktur der Hüllkörper-Hierarchien für Objekte wird anders angelegt als die durch konventionelle Konstruktionsweisen entstehende Struktur mit einem Blatt-Knoten pro Seitenfläche einer Eingabe-Geometrie: Die Gruppierung von Teil-Geometrien wird größere Teil-Strukturen zusammenfassen, die im Fall einer potentiellen Kollision mit einer anderen Teil-Struktur einem detaillierteren Paar-Test unterzogen werden, anstatt die Anzahl möglicher Kollisionen erst mittels einer Hierarchie-Traversierung einzugrenzen. Diese Idee mag zunächst widersprüchlich erscheinen, da Hüllkörper-Hierarchien genau für diese Aufgabe konzipiert sind. Allerdings zeigt die Analyse GPU-basierter Implementierungen polygonbasierter Ansätze in Abschnitt 4.3 (gProximity und HPCCD), dass bei der Traversierung von Hüllkörper-Hierarchien zusätzliche Maßnahmen erforderlich sind, um eine effektive Aufgabenverteilung für die massiv parallele Traversierung gewährleisten zu können. Umgekehrt zeigen die Versuche von Erbes in Unterabschnitt 4.3.3, dass Parallelisierungs-Schemata mit keiner oder allenfalls geringer Kopplung von unabhängig voneinander ablaufenden Hüllkörper-Traversierungen in ihrem Laufzeitverhalten den Schemata überlegen sind, die im Gegensatz dazu eine Parallelisierung von Berechnungen innerhalb derselben Hierarchie-Traversierung vornehmen.

Aufgrund dieser Beobachtungen soll der eigene Ansatz auf die Berechnung vollständiger Hüllkörper-Hierarchien verzichten. Stattdessen werden Hüllkörper-Strukturen benutzt, die in einer sehr flachen Hierarchie mit einem hohen Verzweigungsgrad gruppiert sind. Diese Restrukturierung nutzt die Vorteile von Hüllkörper-Hierarchien bei der schnelleren Lokalisierung potentiell kollidierender Teil-Geometrien, und stellt gleichzeitig sicher, dass auf eine aufwendige Warteschlangen-Verwaltung in der Traversierungs-Phase verzichtet werden kann.

Was die Funktionsweise einer in dieser Art modifizierten Hüllkörper-Hierarchie für den Einsatz auf einer GPU zusätzlich attraktiv macht, ist die Verringerung der Anzahl durchzuführender Paar-Tests von inneren Knoten, und eine größere Menge zu überprüfender geometrischer Primitive in den einzelnen Blatt-Knoten. Auch das mag zunächst widersprüchlich erscheinen, da es eigentlich die Aufgabe einer Hüllkörper-Hierarchie ist, die Anzahl nötiger Paar-Tests zwischen elementaren Geometrie-Elementen in einem Dreiecks-Netz zu minimieren.

Der hybride Ansatz von HPCCD, der die Hüllkörper-Traversierung CPU-gestützt und nur Paar-Tests zwischen Dreiecken GPU-gestützt durchführt, ist allerdings ein aufschlussreiches Gegenbeispiel dafür: Um eine hohe Anzahl von Kopiervorgängen zwischen Gast-System und GPU zu verhindern, werden bei HPCCD durchzuführende Paar-Tests zuerst in einer Warteschlange gesammelt. Erst nachdem eine gewisse Mindest-Menge an Paar-Tests in der Warteschlange vorhanden ist, werden diese „en bloc“ in den Speicher einer GPU kopiert, und die Berechnung tatsächlicher Kontakte wird parallelisiert pro Dreiecks-Paar angestoßen.

Inspiriert durch diese blockweise Verarbeitung von Paar-Tests soll der eigene Ansatz eine ähnliche Gruppierung größerer *Konglomerate* von geometrischen Primitiven verfolgen. Im Unterschied zu HPCCD soll die Gruppierung von Paar-Tests allerdings nicht erst im Lauf einer Hüllkörper-Traversierung vorgenommen werden, sondern gewissermaßen imma-

nent bereits während der Konstruktion der Hüllkörper-Hierarchien von Objekten erfolgen. Das bringt den Vorteil mit sich, den Umfang von GPU-gestützten Paar-Tests in einer Art beeinflussen zu können, die sich an den topologischen und strukturellen Gegebenheiten der Eingabe-Geometrien orientiert, anstatt durch die gewählte Abstiegs-Regel und die Art der Traversierung beim Abstieg durch ein Paar von Hüllkörper-Hierarchien bestimmt zu werden, oder wie im Fall von Front-Tracking die Erzeugung und Verwaltung eines BVTT-Baums notwendig zu machen, der noch dazu potentiell für jeden Paar-Test zwischen zwei Objekten separat im Speicher gehalten werden müsste.

Ein *Konglomerat* sei im Folgenden definiert als

> Teilmenge s_i der Menge aller Seitenflächen S einer polygonalen Geometriebeschreibung, wobei die Vereinigung aller Konglomerate, die aus der Partitionierung einer solchen Geometriebeschreibung hervorgehen, wiederum alle Seitenflächen der erzeugenden Geometrie umfassen soll: $\cup_{i=1,2,...,n} s_i = S$.

Die Restrukturierung einer Hüllkörper-Struktur in der soeben beschriebenen Weise führt andererseits zu einem neuen Problem: Wie kann die effiziente Verarbeitung detaillierter Kollisions-Test innerhalb eines größeren Konglomerats aus Geometrie-Primitiven sichergestellt werden, wenn die Reduktion auf Tests zwischen einzelnen Paaren von Seitenflächen, wie sie durch konventionelle Hüllkörper-Hierarchien erzwungen wird, wegfällt?

An dieser Stelle soll die Vorgehensweise des Voxmap-Pointshell-Algorithmus zum Einsatz kommen: Anstatt alle möglichen Kombinationen von Seitenflächen aus zwei Konglomeraten zu testen, wird an dieser Stelle ein voxel-basierter Ansatz gewählt. Jedes Konglomerat ist zusätzlich mit einer Punkt-Hülle und einer Raumpartitionierung versehen, die allerdings nicht die gesamte Eingangs-Geometrie umfassen, sondern nur auf Basis der im Konglomerat erfassten Seitenflächen erzeugt wird. Dies ist die zweite Stufe des eigenen Ansatzes, auf der durch die Wahl eines sehr effizient parallelisierbaren Verfahrens dafür gesorgt werden soll, dass die massiv parallele Arbeitsweise von GPUs bestmöglich ausgenutzt werden kann: Anstatt mit Paar-Tests für Dreiecken zu operieren, kann die detaillierte Suche nach möglichen Kontaktpunkten gewissermaßen auf eine „Nearest-Neighbour“-Suche zurückgeführt werden.

Diese Möglichkeit entsteht dadurch, dass die Punkt-Hülle eines Konglomerats in die Raumpartitionierung eines anderen Konglomerats eingeordnet wird. Die Assoziation von Punkten aus der Punkt-Hülle des ersten Konglomerats in einem Paar-Test mit Voxeln aus dem zweiten Konglomerat, in denen Punkte auf dessen Oberfläche registriert sind, dient dabei als Indikator für die Durchführung eines Paar-Tests auf Basis der zugehörigen polygonalen Geometrie-Elemente. Im Unterschied zum Voxmap-Pointshell-Algorithmus dient die voxelbasierte Suche nach möglichen Berührungen also als zusätzliche Vorfilter-Phase, und nicht als finaler Kollisions-Test zur Ermittlung tatsächlicher Kontaktpunkte.

Das führt wiederum zu einer weiteren Frage, die die potentielle Anzahl von elementaren Kollisions-Tests betrifft: Polygon-basierte Verfahren führen Paar-Tests zwischen Dreiecken wie in Unterabschnitt 3.5.1 beschrieben durch, beispielsweise über sechs Ecke-Fläche- und neun Kante-Kante-Tests (Algorithmus 17), oder über das Separating-Axis-Theorem aus. Raumpartitionierungs-Verfahren müssen dagegen eine um ein Vielfaches höhere Zahl an elementaren Kollisions-Tests in Form der Einordnung aller Punkte einer Punkt-Hülle in eine Raumpartitionierung bewältigen. Um diesem Problem zu begegnen, werden als Datenstrukturen für die Raumpartitionierung innerhalb von Konglomeraten alternativ Octrees oder

k-d-Bäume verwendet, für die es (wie in Unterunterabschnitt 4.4.3.1 und Unterunterabschnitt 4.4.3.2 beschrieben) sehr einfache Möglichkeiten gibt, die Einordnung von Punkten in die jeweilige Baumstruktur zu bestimmen.

Der Ablauf des eigenen Verfahrens stellt sich für einen Kollisions-Test zwischen einem Paar von Geometrien in der Objektpaar-Phase also wie folgt dar:

1. Überprüfung aller Hüllkörper-Paare der Konglomerate aus beiden Geometrien
2. Für alle überlappenden Hüllkörper-Paare: Einordnung der Punkt-Hülle eines Konglomerats in die Raumpartitionierung des anderen Konglomerats
3. Für alle Punkte aus der Punkt-Hülle eines Konglomerats, die in oberflächen-nahen Voxeln der Raumpartitionierung des anderen liegen: Bestimmung der zugehörigen Seitenflächen aus der polygonalen Geometriebeschreibung
4. Durchführen von Paar-Tests für Seitenflächen für die bestimmten Dreiecks-Paare

Die folgenden Schritte müssen für die Umsetzung des beschriebenen Verfahrens umgesetzt werden:

1. Die Gruppierung von Geometrie-Teilen zu Konglomeraten
2. Eine speichereffiziente und leicht rekonstruierbare Repräsentation für Punkt-Hülle von Konglomeraten finden
3. Eine speichereffiziente, schnell zu erstellende und effizient durchsuchbare Datenstruktur für Raumpartitionierungen von Konglomeraten finden
4. Die Adaption eines voxel-basierten Kollisions-Tests für den Einsatz auf GPUs

Der weitere Verlauf von Kapitel 5 wird sich mit diesen verschiedenen Aspekten wie folgt beschäftigen:

1. Abschnitt 5.2 diskutiert verschiedene Möglichkeiten, wie Teile polygonaler Eingabe-Geometrien zu Konglomeraten kombiniert werden können. Zu den möglichen Ansätzen gehören etwa näherungsweise konvexe Dekompositionen oder Clustering.
2. Abschnitt 5.3 beschäftigt sich mit der verwendeten Datenstruktur für und der Erzeugung von Punkt-Hüllen.
3. Abschnitt 5.4 bis Abschnitt 5.6 stellen die im eigenen Ansatz verwendeten Datenstrukturen für Raumpartitionierungen vor: GPU-basierte Implementierungen für Octrees und k-d-Bäume.

5.2. Zur Erzeugung von Konglomeraten

Die erste Fragestellung, die bei der Umsetzung des im vorigen Abschnitt 5.1 skizzierten Verfahrens gelöst werden muss, ist die Unterteilung von polygonalen Eingangs-Geometrien auf eine Weise, die die Entstehung einer tiefen Hierarchie mit niedrigem Verzweigungsgrad verhindert. Gleichzeitig soll eine zu feingranulare Aufteilung einer Eingangs-Geometrie, in der Konglomerate nur wenige Seitenflächen enthalten, möglichst vermieden werden.

Als mögliche Varianten zur Segmentierung polygonaler Geometrien sind neben Hüllkörper-Hierarchien, welche von der Mehrheit existierender Kollisionserkennungs-Verfahren verwendet werden, auch alternative Ansätze in der Computergrafik beziehungsweise aus der analytischen Geometrie anwendbar:

- Konvexe Zerlegung von Polyedern
- Clustering-Techniken zur Segmentierung von Punkt-Wolken

Eine weitere Möglichkeit zur Segmentierung von polygonalen Geometrien, angelehnt an die Verwendung von Octrees mit beschränkter Höhe wie bei Voxmap-Pointshell, oder auch an die Verschiebung des Einstiegs in eine paarweise Hierarchie-Traversierung wie beim Front-Tracking, wäre die Gruppierung von Geometrie-Elementen anhand deren Enthaltensein in Hüllkörpern in hohen Ebenen einer Hierarchie.

Die praktische Anwendung der im Folgenden vorgestellten Verfahren wird als Teil von Kapitel 6 vorgestellt werden.

5.2.1. Konvexe Zerlegung

Diese Techniken zerlegen beliebig strukturierte Polyeder in disjunkte, konvexe Teil-Mengen. Das Bestimmen einer konvexen Zerlegung mit einer minimalen Anzahl konvexer Teil-Polyeder ist ein NP-hartes Problem ([Cha84]); Verfahren zur konvexen Zerlegung verwenden daher andere Kriterien als die Erzeugung einer minimalen Anzahl von erzeugten konvexen Polyedern.

Im Rahmen dieser Arbeit hat sich die Unterscheidung anhand eines Kriteriums als praktikabel erwiesen, das eine Unterscheidung zwischen volumen-treuen und approximierenden konvexen Zerlegungen macht:

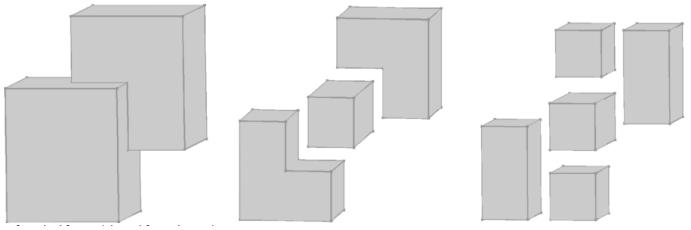
- Volumen-treue Zerlegungen erzeugen konvexe Teil-Polyeder, deren Vereinigungsmenge wieder exakt den ursprünglichen Polyeder ergibt.
- Approximierende Zerlegungen erlauben die Erzeugung von konvexen Teil-Polyedern, deren Vereinigungsmenge einen dem ursprünglichen Polyeder ähnlichen Polyeder ergibt (innerhalb einer benutzerdefinierten Toleranz).

Eigentlich ist die Verwendung einer approximierenden Zerlegung für Simulations-Szenarien mit hohen Anforderungen an die geometrische Präzision von simulierten Objekten kontraproduktiv; dies ist im Zusammenhang mit der diskutierten Erzeugung einer Gruppierung von Teil-Geometrien jedoch nicht problematisch, da die Elemente einer approximierenden Zerlegung einfach als Hüllkörper verwendet werden können, um im Rahmen einer Vorverarbeitung von Eingabe-Geometrien die Zugehörigkeit von Seitenflächen zu einzelnen Elementen der approximierenden Zerlegung zu bestimmen.

5.2.1.1. Volumen-treue Zerlegung

Das in CGAL ([Hac13]) nach Hachenberger ([Hac07]) verwendete Verfahren fügt etwa neue Seitenflächen an allen Kanten eines Polyeders ein. Als Kriterium für eine Zerlegung dient der Innenwinkel zwischen zwei Seitenflächen: Ist dieser Winkel größer als 180 Grad, so wird ent-

lang entlang einer Kante, an der zwei solche Seitenflächen anschließen, eine neue Seitenfläche eingefügt (Abbildung 5.1).



(a) Erster Schritt **(b)** Zweiter Schritt **(c)** Ergebnis der Zerlegung

Abbildung 5.1.: Konvexe Zerlegung eines Polyeders nach Hachenberger ([Hac13]), [Hac07])

Die Zerlegung erfolgt dabei in zwei Schritten. Dem Zerlegungs-Kriterium entsprechende Kanten werden in zwei Gruppen unterteilt, um die eindeutige Bestimmung neuer Seitenflächen gewährleisten zu können: Neue Seitenflächen werden dabei nur orthogonal zu einer der Ebenen des Bezugs-Koordinatensystems (der x-y-Ebene in der CGAL-Implementierung) eingefügt, und parallel zur x-y-Ebene liegende Kanten werden daher in einer zweiten Iteration separat behandelt. Als Ergebnis einer konvexen Zerlegung erzeugt dieses Verfahren maximal $O(r^2)$ konvexe Teil-Polyeder, wobei r die Anzahl von Kanten ist, an denen zwei Seitenflächen des ursprünglichen Polyeders einen Innenwinkel von mehr als 180 Grad aufweisen. Für genauere Informationen zur Funktionsweise dieses Verfahrens sei auf [Hac07] verwiesen.

Diese als Teil von CGAL implementierte Methode verändert die topologische Struktur eines Polyeders: Es werden neue Seitenflächen eingefügt, was die Konnektivität des Polyeders und auch die Anzahl von Kanten und Ecken verändert. Die Methode ist aber volumen-treu: Das heißt, das von den durch die Zerlegung entstehenden Polyeder umschlossene Volumen bleibt nach einer konvexen Zerlegung unverändert, und die Vereinigungsmenge aller konvexen Teile führt wieder zu dem vom ursprünglichen Polyeder umschlossenen Volumen zurück.

5.2.1.2. Approximierende Zerlegung

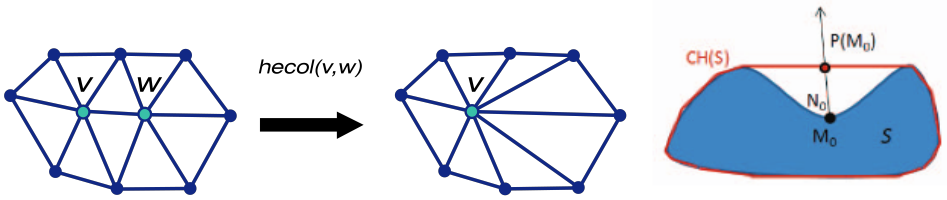
Eine Alternative zu volumen-treuen konvexen Zerlegungen sind Verfahren, die Abweichungen von der Struktur eines ursprünglichen Polyeders erlauben, indem sie nicht nur dessen topologische Struktur verändern, sondern bei der Partitionierung des Ursprungs-Polyeders Ecken, Kanten oder Flächen abweichend von der Oberfläche des ursprünglichen Polyeders einfügt und nicht nur eine Aufteilung des von diesem umschlossenen Volumens vornimmt, so wie es bei dem Verfahren nach Hachenberger der Fall war.

Verschiedene Ansätze für approximierende konvexe Zerlegungen werden etwa in [KT03] und [LA06], [LA08] beschrieben; im Rahmen dieser Arbeit wurde jedoch das von Mamoud ([MG09]) beschriebene und als quelloffene Implementierung verfügbare Verfahren HACD (*H*ierarchical *A*pproximate *C*onvex *D*ecomposition) eingesetzt.

HACD erzeugt eine annähernd konvexe Zerlegung eines triangulierten Polyeders S basierend auf dessen *dualen Graph* S^*, indem Kanten in diesen Graph anhand eines Maßes für die

Konkavität der Umgebung eines betrachteten Eckpunkts entfernt und die Eckpunkte der Kanten zu einem einzigen Eckpunkt kombiniert werden.

Der duale Graph eines Polyeders enthält für jedes Dreieck eines Polyeders einen Knoten, und die Kanten-Menge des Graphen verbindet alle Knoten miteinander, die im Polyeder eine gemeinsame Kante haben.



(a) Entfernung einer Kante (Half-Edge-Collapse) **(b)** Abschätzung der Konkavität

Abbildung 5.2.: Verfahrens-Schritte in HACD (aus [MG09])

Die Zusammenfassung zweier Knoten aus dem dualen Graph erfolgt über einen sogenannten Half-Edge-Collapse (Abbildung 5.2a): Alle Kanten eines aus dem Graphen entfernten Knoten w werden mit einem neu entstehenden Knoten v verbunden. Bei jeder Zusammenfassung wird für einen neu entstehenden Knoten v eine (anfänglich leere) Liste $A(v)$ mit Vorläufer-Knoten aktualisiert mit

$$A(v) \leftarrow A(v) \cup A(w) \cup w \tag{5.1}$$

Welche Knoten des dualen Graphen (und entsprechend welche Dreiecke des ursprünglichen Polyeders) zusammengefasst werden, wird durch eine Kostenfunktion bestimmt, die den Grad der Konkavität $C(S(v,w))$ der Oberfläche $S(v,w)$ mit dem Längenverhältnis E_{shape} in der Umgebung zweier zu verbindender Knoten v, w wie folgt abschätzt:

$$S(v,w) = A(v) \cup A(w) \cup v, w \tag{5.2}$$

$$E_{shape} = \frac{\rho^2(S(v,w))}{4\pi\sigma(S(v,w))} \tag{5.3}$$

$$C(S(v,w)) = \arg\max_{M \in S} \|M - P(M)\| \tag{5.4}$$

mit ρ als Umfang (Summe der Länge der äußeren Kanten aller zu $S(v,w)$ gehörenden Dreiecke) und σ als Fläche aller zu $S(v,w)$ gehörenden Dreiecke.

Die Konkavität der Umgebung eines Knotens aus dem dualen Graphen ergibt sich durch die maximale Differenz des Vektors zwischen der Projektion $P(M)$ eines Eckpunkts M und dem Eckpunkt selbst aus der Umgebung $S(v,w)$ auf deren konvexe Hülle $CH(S)$ (Abbildung 5.2b).

Das Kostenmaß für die Reduktion einer Kante $E(v,w)$ aus dem dualen Graph ergibt sich damit zu

$$E(v,w) = \frac{C(S(v,w))}{D} + \alpha E_{shape}(v,w) \tag{5.5}$$

mit D als Normalisierungs-Faktor, gewählt als die Länge der Diagonale der AABB von $S(v,w)$, und α als Skalierungsfaktor für den Einfluss des Längenverhältnisses auf die Kos-

tenfunktion. In jeder Iteration des Verfahrens wird eine Reduktion für die Kante im dualen Graphen mit dem geringsten Kostenmaß ausgeführt. Eine neue Zerlegung $\Pi(n) = \left\{\pi_1^n, \pi_2^n, \ldots, \pi_{K(n)}^n\right\}$ mit

$$\forall k \in 1, \ldots, K(n) : \pi_k^n = p_k^n = p_k^n \cup A(p_k^n) \tag{5.6}$$

mit $p_k^n \in S^*$ als die Knoten des dualen Graphs wird bestimmt. Das Verfahren wird solange iteriert, bis alle k Teile der entstandenen Zerlegung eine Konkavität $C(\pi_k^n)$ kleiner als eine benutzerdefinierte Toleranz ϵ aufweisen.

Der Parameter α in Gleichung 5.5 wird von den Autoren als

$$\alpha = \frac{\epsilon}{D \cdot 10} \tag{5.7}$$

festgelegt. Die Begründung für diese Wahl des Parameters wird damit angegeben, dass in den ersten Iterationen des Verfahrens die einzelnen Teile der entstehenden Zerlegung ein Konkavitäts-Maß nahe 0 aufweisen, und das Längenverhältnis E_{shape} die Kostenfunktion dominiert, und damit die Zerlegung in möglichst kompakte Oberflächen erfolgt. Dies ändert sich im späteren Verlauf der Zerlegung, da die Größe der einzelnen Teile der Zerlegung wächst, und damit deren Konkavität kontinuierlich steigt. Da das Verfahren bevorzugt den ursprünglichen Polyeder möglichst in große Teile zerlegen soll, sorgt diese Wahl des Skalierungsfaktors α dafür, dass die Konkavität einzelner Teile einer Zerlegung und nicht das Längenverhältnis der Umgebung die Wahl der Partitionierung in den letzten Iterationen des Verfahrens dominiert.

Im Vergleich zu früheren Ansätzen zur approximierende konvexen Zerlegung (konkret zu [LA06]) erzielt HACD eine hinsichtlich der Anzahl entstehender konvexer Teile eine besser optimierte konvexe Zerlegung, was dieses Verfahren im Hinblick auf die beabsichtigte Verwendung als Unterteilungs-Schema zur Festlegung von Konglomeraten zu einer sehr gut geeigneten Alternative im Rahmen des vorgestellten, eigenen Kollisionserkennungs-Verfahrens macht.

5.2.2. Clustering

Die zweite im Rahmen der Arbeit untersuchte Möglichkeit zur Segmentierung einer polygonalen Eingangs-Geometrie ist durch einen Anwendungsbereich aus der Robotik mit einer sehr ähnlichen Aufgabenstellung (allerdings mit anderer Zielstellung) motiviert: Der Suche nach Strukturen in Punkt-Wolken, oft bestehend aus den Messdaten abstands-messender Sensoren wie Laser-Scannern oder Microsofts Kinect-Kamera. Es wird dabei versucht, Teilbereiche der unstrukturierten Eingabe-Daten gemäß bestimmter Muster zu gruppieren, wie etwa zu Linien, Ebenen, oder gleichförmigen geometrischen Körpern wie Kugeln, Zylindern oder Quadern. Eine weitere Möglichkeit besteht darin, Regionen von Punkt-Wolken anhand des wechselseitigen Abstands von Punkten in disjunkte Teile zu gruppieren.

Anwendungen in der Robotik nutzen diese Verfahren zur Objekterkennung, beispielsweise zur Erkennung von Boden und Wänden in geschlossenen Räumen, oder zur Erkennung von speziell geformten Objekten wie Gläsern, Flaschen oder Verpackungen.

Die in dieser Arbeit eingesetzten Clustering-Techniken zur Segmentierung von Punkt-Wolken sind Teil der quelloffenen Point Cloud Library (PCL, [RC11]). Die verwendeten Clustering-Techniken sind dabei zwei Kategorien zuzuordnen:

- Basierend auf dem RANSAC-Algorithmus ([FB81]) und darauf aufbauenden Weiterentwicklungen wie MLESAC ([TZ00]) oder PROSAC ([CM05])
- Basierend auf sogenanntem Euklidischem Clustering ([Rus09])

5.2.2.1. RANSAC-basierte Clustering-Techniken

RANSAC (*Ran*dom *Sa*mple *C*onsensus) und verwandte Verfahren versuchen, für eine Menge von Messdaten (im Fall der PCL dreidimensionale Punkte) eine Modellkurve zu ermitteln und dabei den Einfluss einzelner gestreuter Messwerte, die eine mögliche Modellkurve verzerren würden, möglichst weit einzuschränken.

Um eine bestmöglich an Messdaten angepasste Modellreihe zu erzeugen, geht RANSAC iterativ wie in Algorithmus 7 skizziert vor.

RANSAC bestimmt in jeder Iteration ein sogenanntes *Consensus Set*, das als Mittel zur Beurteilung der Anpassung möglicher Modellkurven an eine Menge von Messdaten dient. Dazu wird zuerst eine mögliche Modellreihe durch eine zufällige Auswahl von Messwerten erzeugt; es werden dabei nur so viele Messwerte benutzt, wie nötig sind, um die gesuchte Art von Messkurve zu parametrisieren. Nach der Erstellung einer möglichen Modellkurve wird für alle weiteren Messdaten der Abstand zu dieser bestimmt; jeder Messwert, der eine benutzerdefinierten Abstand unterschreitet, wird in das Consensus Set der aktuellen Iteration aufgenommen. Nach Beendigung der Suche wird die Modellkurve mit der höchsten Anzahl an beinhalteten Messpunkten als beste Hypothese ausgewählt.

Algorithmus 7 : RANSAC

Input : M, ϵ, N, C_{min}

1 // Eingegebene Messdaten, maximal zulässige Abweichung eines Messwerts, Anzahl der Iterationen, Mindestanzahl Elemente für Consensus Set
2 $C \leftarrow \emptyset$;
3 **for** $1 \rightarrow N$ **do**
4 Wähle zufällig: Menge $P \in M$;
5 Bestimme eine Modellreihe aus P;
6 Bestimme $Q \in M : q \in M : dist(q, P) \leq \epsilon$;
7 **if** $\|Q\| \geq C_{min}$ **then**
8 $C \leftarrow Q$;
9 $C_{best} = c \in Q : max(\|c\|) \in Q$;

Für die Segmentierung von Punktwolken werden RANSAC-basierte Algorithmen selbst wiederum iterativ verwendet: Nach der Ermittlung des jeweils besten Consensus Set in einer Iteration werden alle zu diesem gehörenden Punkte aus der ursprünglichen Punktwolke entfernt; dies wird solange fortgesetzt, bis keine sinnvolle weitere Segmentierung (beispielsweise

weil kein Consensus Set mit der erforderlichen Mindestgröße mehr aus der verbleibenden Punktwolke extrahiert werden kann) mehr möglich ist.

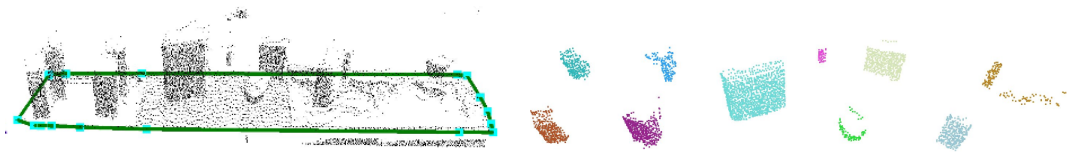
Hinsichtlich der Erzeugung einer möglichen Partitionierung einer polygonalen Eingangs-Geometrie in Konglomerate ist zu bemerken, dass die Fähigkeit von RANSAC-basierten Verfahren, Abweichungen in Eingabedaten herausfiltern zu können, nicht der eigentliche Grund für deren Verwendung darstellt: Die in Abschnitt 5.3 vorgestellte Repräsentation für Punktwolken wird aus Punkten gebildet, die in einem regelmäßigen Raster pro Seitenfläche einer Eingangs-Geometrie ermittelt werden. Damit ist die Existenz von durch Messfehler bedingten abweichenden Elementen in den Eingabedaten konstruktionsbedingt ausgeschlossen.

Auch ist im Gegensatz zu konvexen Zerlegungen die Flexibilität RANSAC-basierten Clusterings dahingehend eingeschränkt, dass beispielsweise das Clustering basierend auf zusammenhängenden, planaren Teilen einer Eingangs-Geometrie bei fortgesetzter Anwendung auf eine Punkt-Wolke, die wie in Abschnitt 5.3 beschrieben strukturiert ist, Cluster für einzelne Seitenflächen einer Eingangs-Geometrie erzeugen würde. Das widerspricht dem zuvor formulierten Ziel, die Anzahl der Elemente einer Partitionierung gegenüber konventionellen Hüllkörper-Hierarchien wesentlich verringern zu können. Daher wird für die Bestimmung möglicher Konglomerate mit Hilfe der in der PCL verfügbaren, RANSAC-basierten Methoden eine minimale Anzahl erfasster Seitenflächen pro Segment vorausgesetzt.

5.2.2.2. Euklidisches Clustering

Euklidisches Clustering ([Rus09]) verfolgt einen Ansatz, der dem eines Flood-Fill-Algorithmus ähnelt (Algorithmus 8): Eine Punkt-Wolke wird anhand der wechselseitigen Abstände von Punkten zueinander gruppiert; für einen betrachteten Punkt werden alle noch nicht bereits einem Cluster zugeordnete Punkte mit einem Abstand geringer als ein benutzerdefinierter Radius dem gerade aktiven Cluster zugeordnet.

Eine Cluster-Bildung wird abgeschlossen, sofern es in der durch den benutzerdefinierten Radius definierten Umgebung keinen weiteren noch nicht zugeordneten Punkt gibt. Der Algorithmus fährt auf dieselbe Weise fort, bis keine neuen Cluster mehr aus der ursprünglichen Punktwolke erzeugt werden können. Abbildung 5.3 zeigt ein Beispiel für die Anwendung dieses Verfahrens.



(a) Ursprüngliche Punkt-Wolke **(b)** Extrahierte Cluster

Abbildung 5.3.: Euklidisches Clustering (aus [Rus09])

Wie im Fall RANSAC-basierten Clusterings muss auch bei diesem Verfahren in Bezug auf die Erzeugung möglicher Konglomerate zur Zusammenfassung von Teilen einer Eingabe-Geometrie auf die Anzahl erfasster Seitenflächen pro Cluster geachtet werden, um nicht eine Segmentierung mit unerwünscht hoher Anzahl von Elementen zu erreichen.

Algorithmus 8 : Euklidisches Clustering

Input : P, d_{th}

```
1  // Punktwolke, maximaler Radius eines Clusters
2  C ← ∅;
3  Q ← ∅;
4  k-d-Baum t_kd für P berechnen;
5  while ∃p_i ∈ P : p_i ∉ C do
6      for p_i ∈ P do
7          Q ← p_i;
8          for p_i ∈ Q do
9              Suche t_j ∈ T_i^k in t_kd mit ‖p_i − t_j‖ ≤ d_th;
10             for t_j ∈ T_i^k do
11                 if t_j ∉ C then
12                     Q ← t_j;
13             if ∀p_i ∈ Q : p_i ∈ C then
14                 C ← Q;
15                 Q ← ∅;
```

5.2.3. Partitionierung polygonaler Geometrien mit Hilfe von Hüllkörper-Hierarchien

Eine dritte Möglichkeit zur Segmentierung einer polygonalen Geometrie bietet sich durch die Verwendung der inneren Knoten einer Hüllkörper-Hierarchie. Ähnlich den in Kapitel 4 beschriebenen Techniken zur parallelen Traversierung von Hüllkörper-Hierarchien wie dem Front-Tracking, das mögliche Teilbäume für eine erneute Traversierung zwischenspeichert, können Geometrie-Teile anhand ihrer Zugehörigkeit zu verschiedenen Teilbäumen einer Hüllkörper-Hierarchie zu Konglomeraten gruppiert werden. Im Unterschied zu konvexen Zerlegungen und den vorgestellten Clustering-Techniken orientiert sich eine solche Partitionierungs-Technik nicht an den topologischen Charakteristiken einer polygonalen Geometrie, sondern an der durch das für die Erstellung der Hierarchie verwendete Konstruktions-Verfahren gebildeten Aufteilung.

Das Kriterium zur Bestimmung einer Partitionierung besteht in der Verwendung der inneren Knoten einer Hierarchie-Ebene nahe des Wurzelknotens einer Hierarchie, und der Zuordnung aller darin enthaltenen Seitenflächen zu einem Konglomerat. Hier muss bei der Wahl der Hierarchie-Ebene auf das Verhältnis zwischen der pro innerem Knoten enthaltenen Seitenflächen und der Gesamtzahl von Seitenflächen in der umschlossenen Geometrie geachtet werden. Ebenso ist eine eventuell bestehende Unbalanciertheit einer Hüllkörper-Hierarchie problematisch: Wenn sich die von verschiedenen Teilbäumen umschlossene Anzahl von Seitenflächen stark unterscheidet, kann sich das ebenfalls negativ auf eine Partitionierung auswirken, die sich ausschließlich an einer einzigen Hierarchie-Ebene orientiert. Hier wäre es als Gegenmaßnahme beispielsweise möglich, Teilbäume mit einer größeren Anzahl Seitenflächen weiter zu segmentieren, und aus Kind-Knoten weitere Konglomerate zu erzeugen.

5.2.4. Kombinierte Verwendung der Verfahren

Die alleinige Verwendung eines der zuvor beschriebenen Verfahren zur Erzeugung von Konglomeraten ist nicht in allen Fällen möglich oder vorteilhaft: Beispielsweise sind RANSAC-basierte Verfahren durch die spezifische Art von geometrischen Strukturen (wie Ebenen) für manche polygonale Geometrien nicht in der Lage, eine sinnvolle Segmentierung zu bestimmen. Andererseits ist eine (approximierende) konvexe Zerlegung zwar grundsätzlich dazu in der Lage, eine topologisch fehlerfrei spezifizierte polygonale Geometrie vollständig zu segmentieren: Die Zerlegung kann im Zweifelsfall Segmente aus einzelnen Seitenflächen erzeugen, da ein einzelnes Dreieck konvex ist. Das steht allerdings wie bereits erwähnt im Widerspruch mit dem Ziel, Segmentierungen mit einer wesentlich kleineren Anzahl von Knoten zu erzeugen, als sie Hüllkörper-Hierarchien typischerweise besitzen.

Um diesen Problemen zu begegnen, könnten die verschiedenen Verfahren in der Implementierung des eigenen Ansatzes sukzessive auf Eingabe-Geometrien angewendet werden. Ein zusätzlicher Schritt zur Berücksichtigung von Geometrie-Teilen, die eventuell nicht durch eines der zuvor angewendeten Verfahren erfasst worden sind, müsste dabei als weitere Maßnahme erfolgen:

1. RANSAC-basierte Clustering-Verfahren beziehungsweise Euklidisches Clustering
2. Approximierende konvexe Dekomposition
3. Berechnung von umschließenden OBBs für alle übrigen Seitenflächen

Diese Anwendungs-Reihenfolge wäre der beste Kompromiss zwischen der Anzahl erzeugter Konglomerate und der Robustheit gegenüber schwierig zu erfassender topologischer Gegebenheiten.

5.3. Datenstruktur für Punkt-Hüllen

In Abschnitt 5.1 wurde die Verwendung von Punkt-Hüllen im eigenen Ansatz zur Kollisionserkennung bereits kurz diskutiert. Der folgende Abschnitt beschäftigt sich nun ausführlicher mit der Erzeugung, Speicherung und Verwendung von Punkt-Wolken als Werkzeug zur Lokalisierung potentiell kollidierender Geometrie-Elemente im Paar-Test zwischen zwei Konglomeraten.

Die verwendete Datenstruktur für Punkt-Hüllen muss in der Lage sein, zwei an sich konkurrierende Ziele erfüllen zu können:

- *Speicher-Effizienz*: Da im Gegensatz zum ursprünglichen Voxmap-Pointshell-Algorithmus für jede Eingabe-Geometrie eine eigene Punkt-Hülle erzeugt werden muss, wäre der Speicherbedarf für die unkomprimierte Speicherung der vollständigen Koordinaten jedes einzelnen Punktes sehr hoch.
- Eine *ausreichend dichte Approximation* der zugrunde liegenden polygonalen Geometriebeschreibung: Im ursprünglichen Algorithmus wird dies beispielsweise durch die Erzeugung der Punkt-Hülle auf Grundlage einer Partitionierung eines Objekts in der Voxel-Größe der Voxmap gewährleistet. Der eigene Ansatz muss in ähnlicher Weise

für eine Abdeckung der ursprünglichen Objekt-Oberfläche sorgen, die jede Seitenfläche beziehungsweise Kante in der erzeugten Punkt-Hülle adäquat repräsentiert.

Zur Gewährleistung einer speichereffizienten Repräsentation verzichtet die im Folgenden vorgestellte Datenstruktur auf die explizite Speicherung von Koordinaten-Tripeln für jedes Element einer Punkt-Hülle: Stattdessen bedient sie sich einer sehr kompakten Darstellung in Form einer Bit-Maske. Des weiteren erfolgt die Auswahl von Oberflächen-Punkten nicht mit Hilfe einer Raumpartitionierung, sondern wird durch ein pro Seitenfläche einer Eingabe-Geometrie erzeugtes adaptives Raster gebildet. Diese Kombination aus einer sehr gleichförmig strukturierten Oberflächen-Approximation und einer speichereffizienten Repräsentation erlaubt die einfache und schnelle Rekonstruktion der Koordinaten von Oberflächen-Punkten für die Verwendung in Kollisions-Tests.

5.3.1. Erzeugung von Punkt-Hüllen

Als Ausgangspunkt für die Berechnung einer Punkt-Hülle dient wie im ursprünglichen Voxmap-Pointshell-Algorithmus die polygonale Geometriebeschreibung eines Objekts. Allerdings wird für die hier vorgestellte Vorgehensweise nicht auf eine Raumpartitionierung oder eine Formulierung als quadratisches Optimierungsproblem zurückgegriffen, wie es bei den in Unterunterabschnitt 4.4.3.2 beschriebenen Verfahren der Fall ist. Vielmehr wird zur Erzeugung der Punkt-Hülle auf eine einfachere Vorgehensweise zurückgegriffen, wie in Algorithmus 9 angedeutet.

Für jedes Dreieck einer Geometrie wird zuerst das umgebende Rechteck berechnet; basierend auf dem Verhältnis zwischen Breite und Höhe dieses Rechtecks wird ein rektilineares Gitter über das Rechteck gelegt, dessen Kreuzungspunkte darauf überprüft werden, ob sie innerhalb der Fläche des gerade betrachteten Dreiecks liegen. Ist dies der Fall, so wird in einer dem Dreieck zugeordneten Bit-Maske der entsprechende Bit-Index als aktiv markiert. Der Ursprungs-Punkt $g_0 = (0, 0)$ des rektilinearen Gitters fällt dabei mit dem gemäß der polygonalen Geometriebeschreibung der Eingabe-Geometrie ersten Eckpunkt jedes Dreiecks zusammen.

Abbildung 5.4 zeigt einige Beispiele für die Struktur von mit Hilfe des beschriebenen Verfahrens berechneten Punkt-Hüllen.

Eine derartige Konstruktion ermöglicht es, auf die separate Transformation jedes Elements der Punkt-Hülle bei Objektbewegungen zu verzichten: Da die Bit-Maske relativ zu den Koordinaten eines Eckpunkts des zugeordneten Dreiecks ausgerichtet ist, verändern sich die Koordinaten der per Bit-Maske kodierten Elemente der Punkt-Hülle relativ zu den Koordinaten ihres Bezugspunktes nicht.

Die Unterstützung verformbarer Geometrien durch das beschriebene Verfahren wäre mit einer Erweiterung zur Neuberechnung der Rasterisierung einzelner Seitenflächen realisierbar: Die Bestimmung des umgebenden Rechtecks und die Bestimmung innerhalb eines davon umschlossenen Dreicks liegender Gitterpunkte ist mittels Verwendung einfacher geometrischer Prädikate umsetzbar, und muss nur für solche Seitenflächen vorgenommen werden, deren Eckpunkte sich bei einer Verformung verschieben, oder deren topologische Konnektivität sich zwischen zwei Iterationen einer laufenden Simulation verändert hat.

Algorithmus 9 : Verfahren zur Erzeugung einer Punkt-Hülle

Input : G

```
1  // polygonale Geometriebeschreibung
2  GRID_SIZE = 4;
3  TILES_PER_SIDE = 2;
4  for Dreiecke t_i ∈ G do
5      int64_t bm[4];
6      for j = 0 → 4 do
7          bm[j] = 0;
8      Berechne umgebendes Rechteck R;
9      Berechne Anzahl der Zellen in Länge l_i und Breite b_i für ein rektilineares Gitter über R;
10     Berechne Länge l_z und Breite b_z einer Gitterzelle von R;
11     for i = 0 → l_i do
12         for j = 0 → b_i do
13             t_x = i / GRID_SIZE;
14             t_y = j / GRID_SIZE;
15             if pointInTriangle(t_i[0] + i · l_z + j · b_z) then
16                 ti_x = 0;
17                 if i < GRID_SIZE then
18                     ti_x = i;
19                 else
20                     ti_x = i% GRID_SIZE;
21                 ti_y = 0;
22                 if j < GRID_SIZE then
23                     ti_y = j;
24                 else
25                     ti_y = j% GRID_SIZE;
26                 bm_target = (ti_x · GRID_SIZE + ti_y;
27                 bm_shift = 2^(bm_target);
28                 bm[TILES_PER_SIDE · t_x + t_y| = bm_shift;
```

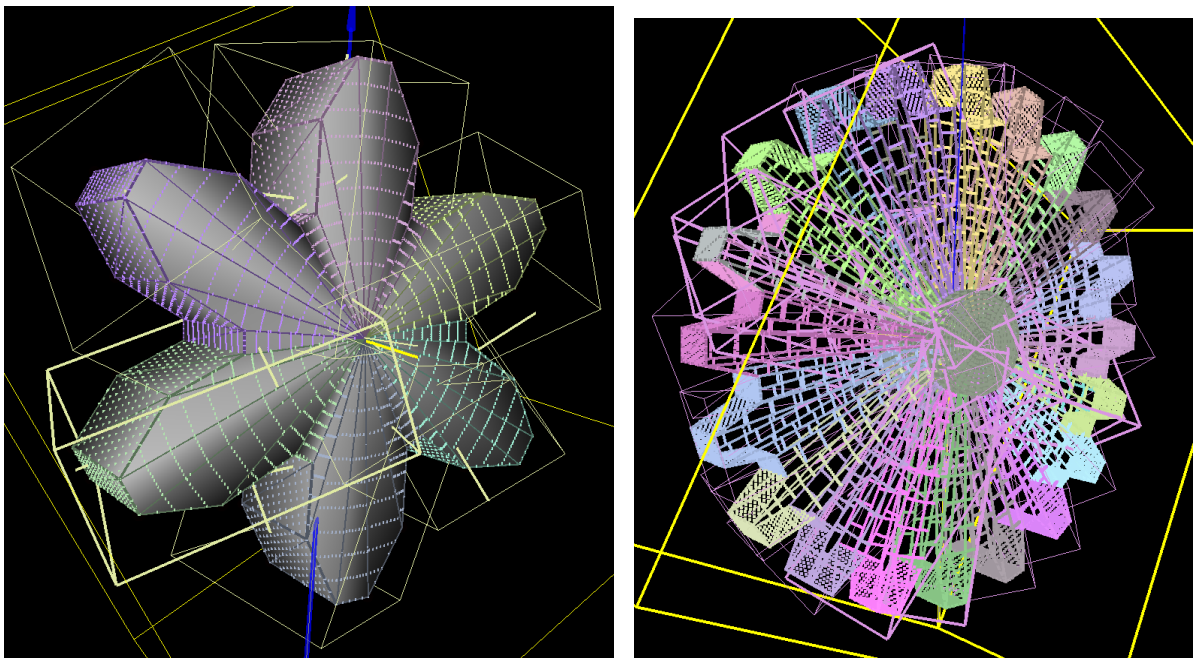


Abbildung 5.4.: Beispiele für Punkt-Hüllen, die anhand der strukturierten Rasterisierung von Seitenflächen in polygonalen Oberflächenbeschreibungen erstellt werden

5.3.2. Datenstruktur zur Speicherung von Punkt-Hüllen

Die Erstellung einer Punkt-Hülle auf Basis eines rektilinearen Gitters erlaubt es, gegenüber der vollständigen Speicherung von Punkt-Koordinaten eine wesentliche Einsparung beim benötigten Speicherplatz zu erreichen. Bei einer vollständigen Speicherung skaliert der Speicherbedarf einer Punkt-Hülle mit n Punkten

$$(3 \cdot \text{sizeof}((\text{float}|\text{double})) \cdot n \tag{5.8}$$

Bei bei der komprimierten Speicherung einer Punkt-Hülle für eine Geometrie mit m Dreiecken mittels Bit-Maske ergibt sich der Speicherbedarf zu

$$(4 \cdot \text{sizeof}(\text{int64\_t}) + 2 \cdot \text{sizeof}(\text{float}|\text{double})) \cdot m \tag{5.9}$$

Während der Speicherplatzbedarf bei der vollständigen Speicherung von Punkt-Koordinaten linear mit der Anzahl in einer Punkt-Hülle enthaltener Punkte wächst, wächst er bei der komprimierten Speicherung nur linear mit der Anzahl der Seitenflächen eines Dreiecks-Netzes.

Die Höhe und Breite des Gitters ist auf 16 mal 16 Zellen begrenzt, was die Speicherung von maximal 256 möglichen Oberflächen-Punkten pro Seitenfläche erlauben würde. Für jeden im Dreieck liegenden Punkt wird in einer Bit-Maske, die in vier 64 Bit großen Ganzzahl-Variablen des Datentyps unsigned long oder int64_t abgebildet ist, ein Bit entsprechend der ermittelten Position im Gitter aktiv gesetzt. Zusätzlich wird pro Dreieck ein Paar aus Fließkomma-Werten benötigt, das die Zellengröße des rektilinearen Gitters in x- und y-Richtung angibt.

Die Datenstruktur für die Repräsentation der Geometrie- und Punkt-Hüllen-Daten eines Konglomerats im GPU-Speicher ist wie in Algorithmus 10 gezeigt strukturiert. Sie folgt dem „Structure-of-Arrays"-Ansatz, wie er in Anhang C als Teil des Vielkörper-Problems benutzt wird. Die einzelnen Daten-Elemente umfassen:

- Die Koordinaten der Eckpunkte des Dreiecks-Netzes als Feld von dreidimensionalen Vektoren aus Fließkomma-Zahlen.
- Die Indizes der Dreiecke im Dreiecks-Netz als Feld von dreidimensionalen Vektoren aus Ganzzahl-Werten.
- Die Bit-Masken jedes Dreiecks als Feld von jeweils vier 64 Bit großen Ganzzahl-Werten.
- Die Gitter-Größe jedes Dreiecks als Feld von zweidimensionalen Vektoren aus Fließkomma-Zahlen.

Algorithmus 10 : Datenstruktur eines Dreiecks-Netzes zusammen mit dessen komprimierter Punkt-Hülle

```
1 unsigned int numTriangles;
2 vec3* vertexList;
3 int3* indexList;
4 ulonglong4* bitMasks;
5 float2* gridSize;
```

Die Daten der Bit-Masken jedes Dreiecks können wie die Vertex- und Index-Listen eines Dreiecks-Netzes, wie sie typischerweise zur Beschreibung von Geometrien in Kollisionserkennungs-Verfahren eingesetzt werden, in einem kontinuierlichen Speicherbereich allokiert und abgelegt werden, zusammen mit den Zellengrößen des zugrunde liegenden Gitters. Die Wahl von vier jeweils 64 Bit großen Ganzzahl-Variablen (zusammen mit zwei Fließkomma-Werten mit 32 Bit) ist in einer weiteren Hinsicht vorteilhaft: Datentransfers in den Speicherbereich einer GPU sind für Block-Größen von 64 und 128 Bit optimiert. Die Gesamtgröße der zusätzlich in einer Geometriebeschreibung abgelegten Daten zur Erfassung einer Punkt-Hülle ergibt sich so zu einem Vielfachen dieser Größen, was sich positiv auf die Übertragungs-Geschwindigkeit beim Transfer zwischen dem Speicher des Gast-Systems und einer GPU auswirkt.

5.3.3. Zugriff auf die Datenstruktur und Rekonstruktion vollständiger Punkt-Koordinaten

Der Zugriff auf die zu einzelnen Seitenflächen eines Dreiecks-Netzes gehörende Bit-Maske erfolgt über den Index eines Dreiecks, wie er sich durch dessen Position in der polygonalen Geometriebeschreibung ergibt, in Kombination mit der Multiplikation eines Versatz-Wertes, wie er durch die Anzahl der Elemente des jeweiligen Datums gegeben ist (beispielsweise 4 für das Bit-Masken-Feld).

Dieses Adressierung-Schema entspricht den Anforderungen der von GPU-Architekturen verwendeten, hardware-basierten Warteschlangen-Verwaltung: Basierend auf der Adresse eines Threads innerhalb eines Ausführungs-Gruppe auf der GPU kann eine Kernel-Instanz auf

die Geometrie-Beschreibung samt assoziierter Punkt-Hüllen-Attribute durch eine einfache Feld-Referenzierung zugreifen.

Die Rekonstruktion der Koordinaten eines Punktes der Punkt-Hülle erfolgt in zwei einfachen Schritten: Als Teil einer zweifach verschachtelten Schleife, die eine Bit-Maske bis zur ihrer maximalen Größe von 16 mal 16 expandiert, wird für jedes Bit mittels einer binären Und-Operation überprüft, ob der durch dieses Bit repräsentierte Punkt als innerhalb des zugehörigen Dreiecks liegend markiert ist. Um nun die vollen 3D-Koordinaten des Punktes zu rekonstruieren, muss nur ein Versatz-Vektor in der Ebene des zugehörigen Dreiecks addiert werden, der in x- und y-Richtung des repräsentierten rektilinearen Gitters ein Vielfaches der Breite beziehungsweise Höhe einer Zelle des Gitters zu den Koordinaten des ersten Eckpunkts des Dreiecks beträgt.

5.4. Datenstrukturen für Raumpartitionierungen

Der folgende Abschnitt beschäftigt sich mit den beiden Varianten von Raumpartitionierungen, die im Rahmen des eigenen Verfahrens zur Umsetzung des ersten Teils detaillierter Kollisionstests zwischen Paaren von Konglomeraten untersucht worden sind: Octrees und k-d-Bäume. Neben den eigentlichen Datenstrukturen der beiden Varianten werden auch die Besonderheiten der verwendeten CUDA-basierten Implementierungen behandelt:

- Für Octrees: Die im Quellcode der Point Cloud Library ([RC11]) verfügbare Implementierung.
- Für k-d-Bäume: Die in der quelloffenen Bibliothek FLANN ([MD10], [ML12]) verfügbare Implementierung.

Ein weiterer Themenbereich betrifft die Unterstützung punkt-basierter Abfragen, und die Indizierung der Position von Punkten innerhalb der jeweiligen Baumstrukturen. Da die Verwendung von punkt-basierten Abfragen eine wesentlich höhere Anzahl zu verarbeitender geometrischer Primitive als bei der kombinierten Verwendung von Hüllkörper-Hierarchien bedeutet, muss im Gegenzug die Effizienz der einzelnen elementaren Kollisionstests wesentlich höher sein als im Fall kombinierter Hüllvolumina- und Seitenflächen-Paartests.

Diese Anforderung erfüllt die Kombination GPU-basierter Operationen auf der Basis von Octrees beziehungsweise k-d-Bäumen mit der Effizienz von Such- und Indizierungs-Operationen dieser Baum-Varianten. Ein wesentliches Element bei dieser Kombination ist dabei die abgewandelte Verwendung von Octrees oder k-d-Bäumen bei der Einordnung von Punkt-Wolken als Teil eines Kollisionstests: Üblicherweise werden diese Baumstrukturen unter anderem als Indizierungs-Hilfen für Suchoperationen in einer Punkt-Wolke eingesetzt. Der eigene Kollisionstest berechnet diese Baumstrukturen zwar ebenso für diese Verwendung, nutzt bei einer Suche auf Basis eines solchen Baums aber nicht die ursprüngliche Punkt-Wolke, auf deren Basis ein Baum berechnet worden ist, sondern ordnet die Punkt-Wolke eines anderen Objekts in einem Paartest der Baumstruktur des betrachteten Objekts zu. Diese Änderung betrifft dabei aber nicht die grundsätzliche Funktionsweise von Suchvorgängen basierend auf Punkt-Primitiven, sondern verwendet eine andere Datenquelle für diese Suchvorgänge: Anstatt die gleiche Eingabe-Geometrie zu verarbeiten, wird in einem Paartest zwischen Konglomeraten die Punkt-Wolke des ersten Konglomerats in einen auf Basis des zweiten Konglomerats

erzeugten Octree oder k-d-Baum eingeordnet. Da die Blatt-Knoten beziehungsweise deren Eltern-Knoten der betreffenden Bäume konstruktionsbedingt Punkte auf der Oberfläche der ursprünglichen polygonalen Geometrie umschließen, dient die Einordnung von Oberflächen-Punkten der anderen Geometrie in oberflächen-nahe Knoten als Indiz für möglicherweise auftretende Kollisionen zwischen den umschlossenen Seitenflächen.

Dabei ist zu betonen, dass ein Kollisions-Test dieser Art nicht die Struktur eines Octree oder k-d-Baums verändert, da die Punkt-Wolke des ersten Konglomerats als Teil des Tests nicht dauerhaft im betreffenden Baum des zweiten Konglomerats abgelegt wird. Vielmehr ist es ausreichend, nur die Position eines Punktes in Bezug auf den Baum zu ermitteln und als Indiz für eine möglicherweise auftretende Kollision festzuhalten. Diese Eigenschaft ist vor allem in Bezug auf Octrees vorteilhaft, da es für diese Art von Raumpartitionierung eine Indizierungs-Möglichkeit gibt, die mit wenigen Fließkommazahl-Operationen pro Koordinate und der Erstellung eines bit-basierten Index auskommt, um die Position eines Punktes in Bezug auf einen Octree festzuhalten.

5.5. Zur Verwendung von Octrees

Die im Rahmen der praktischen Umsetzung des eigenen Verfahrens verwendete Octree-Implementierung weist bedingt durch ihre GPU-spezifische Ausrichtung einige Unterschiede zu den in Unterabschnitt 4.4.2 und Unterunterabschnitt 4.4.3.1 angesprochenen Octree-Datenstrukturen auf. Die wichtigsten beiden Unterschiede bestehen in der Repräsentation der Baumstruktur selbst, sowie in der Index-Berechnung für die Bestimmung des Octree-Knotens, in den ein Punkt als einziges von dieser Implementierung unterstütztes geometrisches Primitiv eingeordnet werden kann.

Die Baumstruktur selbst wird nicht in Form einer hierarchischen Repräsentation abgelegt, in der ein Eltern-Knoten explizit Verweise auf seine Kind-Knoten enthält. Dagegen wird ein Octree hier als Feld-Datenstruktur in einem kontinuierlich allokierten Speicherbereich zusammen mit der assoziierten Punkt-Wolke abgelegt, wobei folgende Daten-Elemente als Teil der Datenstruktur während der Erzeugung eines Octree gespeichert werden:

1. Ein Feld mit den Koordinaten der Punkte der erzeugenden Punkt-Wolke
2. Ein Feld, das die Knoten des Octree beinhaltet
3. Ein Feld mit Indizes der jeweiligen Eltern-Knoten jedes Knoten im Octree
4. Ein Feld mit den zu den Knoten gehörenden Morton-Schlüsseln
5. Zwei Felder mit Indizes für den Anfang und das Ende von Intervallen im Feld mit den Koordinaten der Punkte, die gemeinsam in einem Blatt-Knoten liegen
6. Die Anzahl der Knoten im Octree
7. Die Ausdehnung des Wurzelknotens des Octree als Koordinaten des minimalen und maximalen Eckpunktes

5.5.1. Index-Berechnung bei der Einordnung eines Punktes: Der Morton-Schlüssel

Die gleichförmige Struktur eines Octree erlaubt eine alternative Darstellung der Position eines Knotens durch einen binären Index in Form von *Morton-Schlüsseln* ([MOR66]). Während es die Struktur anderer zuvor betrachteter Baum-Strukturen wie etwa bei Hüllkörper-Hierarchien nötig macht, die geometrische Struktur von Hüllvolumina separiert von der eigentlichen Baumstruktur zu spezifizieren, ist der Octree ein Sonderfall diesbezüglich: Da Kind-Knoten immer in acht gleich große Kind-Volumina aufgeteilt werden, wobei die Seitenlänge eines Kind-Knoten der Hälfte der Seitenlänge seines Eltern-Knoten entspricht, müssen die Dimensionen eines Knotens nicht explizit erfasst werden. Es genügt, die Kantenlänge des Wurzelknotens explizit zu speichern: Aus dieser kann anhand der Tiefe eines Knotens im Baum dessen räumliche Ausdehnung einfach rekonstruiert werden.

Um die Baum-Struktur selbst zu erfassen, bietet die regelmäßige Struktur des Octree wiederum die Möglichkeit, die Position eines Knotens im Baum über eine binäre Indizierung zu adressieren. Dafür wird ein Tupel aus drei Bit pro Ebene des Baums gebraucht, um den Oktanten zu spezifizieren, in dem sich ein Knoten befindet: Jedes Bit legt für eine Ebene jeweils fest, ob sich der Oktant eines Kind-Knotens (oder des Knotens) selbst über oder unter der xy-, xz- und yz-Ebene (im Bezugs-Koordinatensystem des Octree gesehen) befindet. Konkateniert man die 3-Bit-Tupel jeder Ebene bis zur maximalen Tiefe eines Octree, so lässt sich auf diese Weise eine eindeutige Adressierung für jeden Knoten im Octree erstellen.

Bezüglich des Speicherbedarfs eines Octree ist eine solche Repräsentation für die Koordinaten eines Knotens im Baum sehr vorteilhaft, da sich vergleichbar zur in Unterabschnitt 5.3.2 vorgestellten Repräsentation für Punkt-Hüllen die 3-Bit-Tupel eines solchen Schlüssels bitweise als Elemente einer Ganzzahl-Variable ablegen lassen (Algorithmus 11 und Algorithmus 12). Dadurch entsteht zwar der Nachteil, dass die maximale Tiefe eines Octree durch die Größe eines Ganzzahl-Datentyps beschränkt ist (beispielsweise auf zehn Ebenen bei der Verwendung eines 32-Bit-Datentyps). Dieser Nachteil wird aber durch die sehr schnelle Bestimmbarkeit eines Morton-Schlüssels ausgeglichen. Außerdem muss ein Octree, der die Punkt-Hülle eines Konglomerats erfasst, nicht so beschaffen sein, dass jeder einzelne Punkt aus der Punkt-Hülle in einem separaten Blatt-Knoten des Octree erfasst wird. Diese Tatsache lässt die Nutzung eines in der Höhe beschränkten Octree zu.

Algorithmus 11 zeigt, wie die einzelnen Bit-Komponenten einer Koordinate sukzessive in die Bits einer Ganzzahl-Variable integriert werden (mittels $\oplus$ als binäres XOR, $\ll$ als Bit-Shift-Operator und & als binärem Und). Ein Aufruf der Funktion resultiert in einem 10 Bit langen Schlüssel für eine Koordinate mit zwei Bit Abstand.

Die Addition dreier Aufrufe der Funktion in Algorithmus 11 kombiniert mit einer weiteren Bit-Shift-Operation ergibt schließlich den Morton-Schlüssel eines Knotens im Octree (Algorithmus 12).

5.5.2. Zur Erzeugung eines Octrees

Ein Octree wird durch die Implementierung aus der PCL nicht nur GPU-unterstützt durchsucht, sondern auch erstellt. Dazu wird die CUDA-basierte Algorithmen-Bibliothek Thrust

Funktion `InterleaveBits`

Input : i

// Koordinate entlang einer Dimension

Output : m

// Morton-Schlüssel, kodiert als Bit-Komponenten einer Ganzzahl (Integer) //

$m = ---------- 9876543210$

$m = (i \oplus (i \ll 16))$ & 0xff0000ff;

// $m = -------98 ---------- 76543210$

$m = (i \oplus (i \ll 8))$ & 0x0300f00f;

// $m = -------98 --------7654 --------3210$

$m = (i \oplus (i \ll 4))$ & 0x030c30c3;

// $m = -------98 ----76 ----54 ----32 ----10$

$m = (i \oplus (i \ll 2))$ & 0x09249249;

// $m = -----9 --8 --7 --6 --5 --4 --3 --2 --1 --0$

Algorithmus 12 : MortonKey3()

Input : x, y, z

// Index eines Knoten im Octree

Output : n

// Morton-Schlüssel

$n = (\text{InterleaveBits}(z) \ll 2) + (\text{InterleaveBits}(y) \ll 1) + \text{InterleaveBits}(x)$;

([cud13]) genutzt, die häufig gebrauchte Algorithmen (Transformation, Sortieren) über eine iterator-basierte Schnittstelle ähnlich der C++ Standard Template Library (STL) anbietet.

Bevor die eigentliche Octree-Struktur ermittelt wird, erfolgt zuerst eine Erstellung von Morton-Schlüsseln für alle Punkte aus der erzeugenden Punkt-Wolke, gefolgt von einer Neu-Sortierung der Punkt-Koordinaten anhand der Morton-Schlüssel.

Die Konstruktion eines Octree setzt sich aus folgenden Schritten zusammen:

1. Allokieren des Speicherbereichs für den Octree, inklusive der erzeugenden Punktwolke. Der betreffende Speicherbereich wird als zweidimensionales Feld kontinuierlich allokiert Die Anzahl der Spalten dieses Felds entspricht der Anzahl der Punkte in der erzeugenden Punkt-Wolke; für jedes Element der Datenstruktur wird eine eigene Zeile des Feldes reserviert.
2. Berechnen des minimalen und maximalen Punkts des umgebenden Wurzelknotens des Octree
3. Berechnen des Morton-Schlüssels für jeden Punkt in der erzeugenden Punkt-Wolke
4. Sortieren der Elemente der erzeugenden Punkt-Wolke nach ihrem Morton-Schlüssel
5. Bestimmung der Zuordnung von Punkten zu Octree-Blattknoten

Die Größe des resultierenden Octree ist durch zwei Faktoren beschränkt: Die Begrenzung des möglichen Adressraums der Morton-Schlüssel auf eine maximale Baum-Tiefe von zehn Ebenen, und die maximal GPU-seitig allokierbare Speichergröße, wobei die Beschränkung

der Baum-Tiefe die schwerwiegendere Einschränkung bei der Konstruktion eines Octree darstellt. Um die Erfassung möglichst großer Punktwolken zu ermöglichen, wird nicht jeder einzelne Punkt der erzeugenden Punktwolke separat in einem Blattknoten des entstehenden Octree erfasst. Stattdessen wird die Expansion eines Teilbaums abgebrochen, sobald die in einem Oktant erfasste Anzahl von Punkten um eine bestimmte Konstante (implementierungsbedingt 32 Punkte) unterschritten werden würde.

Die Neu-Sortierung der erzeugenden Punktwolke entsprechend der durch die Morton-Schlüssel erzeugte Ordnungsrelation erlaubt den Verzicht auf eine separate Suche beim Zugriff auf Punkt-Koordinaten bei der Durchführung von Suchvorgängen auf Grundlage des erzeugten Octree.

5.6. Zur Verwendung von k-d-Bäumen

Die durch FLANN ([MD10]) bereit gestellte k-d-Baum-Implementierung hat gegenüber der Verwendung von Octrees den Vorteil, dass k-d-Bäume eine wesentlich kompaktere Approximation einer Eingabe-Geometrie erlauben: Durch die flexibleren Subdivisions-Regeln ist es bei der Konstruktion dieser Baum-Variante wie in Unterunterabschnitt 4.4.2.2 bereits kurz angedeutet möglich, das Verhältnis zwischen dem Volumen des k-d-Baums und dem von einer Eingangs-Geometrie umschlossenen Volumen gegenüber Octrees um ein Vielfaches zu reduzieren. Dem gegenüber steht als Nachteil die im Vergleich zu Octrees in Kombination mit Morton-Schlüsseln aufwendigere Suche nach der Zuordnung eines Punktes zu einem Blattknoten in einem k-d-Baum: Während mit Morton-Schlüsseln eine extrem effiziente Adressierung der Position eines Punkt-Primitivs innerhalb eines Octrees möglich ist, muss bei k-d-Bäumen die Baumstruktur während einer Suche explizit traversiert werden, da die Bestimmung des Enthalten-Seins eines Punktes in inneren beziehungsweise Blatt-Knoten eines k-d-Baums unumgänglich ist.

Ein k-d-Baum wird in einer Datenstruktur mit folgenden Elementen abgelegt:

1. Drei Felder mit den Koordinaten der Punkte der erzeugenden Punkt-Wolke (ein Feld je Dimension)
2. Drei Felder mit den Indizes der jeweiligen Koordinaten-Werte von Punkten (um ein Neusortieren der Koordinaten-Felder vermeiden zu können)
3. Drei Felder mit den Indizes der Baum-Knoten, mit denen Punkte jeweils assoziiert sind
4. Ein Feld mit den Indizes der Eltern-Knoten jedes Knotens im k-d-Baum
5. Ein Feld mit den Indizes des linken Kind-Knotens jedes Knotens im Baum (sofern dieser existiert). Die Indizes werden dabei so verwaltet, dass der rechte Kind-Knoten jedes Baum-Knotens direkt nach dem Index des linken Kind-Knotens abgelegt wird.
6. Jeweils ein Feld mit den Koordinaten der minimalen und maximalen Ecken der AABBs aller Baum-Knoten

5.6.1. Zur Erzeugung eines k-d-Baums

Um einen k-d-Baum für eine Punktwolke GPU-unterstützt zu erzeugen, geht die Implementierung von FLANN wie folgt vor:

1. Allokieren von Speicherbereichen entsprechend der Anzahl von Elementen der Punktwolke
2. Sortieren der Elemente der Punktwolke nach aufsteigenden Koordinaten entlang der x-, y-, und z-Achsen des Bezugs-Koordinatensystems
3. Solange noch Baum-Knoten existieren, für die eine Aufteilung möglich ist:
   a) Sofern die Anzahl in einem Knoten enthaltener Punkte größer ist als ein vorgegebener Zielwert: Den Knoten entlang der längsten Achse der umgebenden AABB aufteilen.
   b) Für jeden Punkt aus der Punktwolke: Ermitteln, ob dieser in einem in der aktuellen Iteration aufgeteilten Knoten liegt. Wenn ja, als zugehörig zu einem neuen Kind-Knoten (rechter oder linker Hand) markieren.
   c) Neuordnen der Elemente der Punktwolke, so dass alle jedem Baum-Knoten zugeordneten Punkte kontinuierlich in den Index-Feldern, die die Zugehörigkeit eines Punktes zu einem Knoten festhalten, liegen.

Wie auch bei der in Abschnitt 5.5 verwendeten Vorgehensweise, den Speicherbereich mit den Koordinaten der Elemente einer Punktwolke nicht selbst zu manipulieren (beispielsweise durch Neusortieren), sondern stattdessen ein index-basiertes Zugriffs-Schema zu verwenden, bietet die k-d-Baum-Implementierung von FLANN ebenso den Vorteil, bei Suchvorgängen ein sortiertes Index-Feld nutzen zu können.

Der Speicherbedarf für die Erfassung eines k-d-Baums ist aufgrund dessen gegenüber einem Octree unregelmäßigeren Partitionierungsweise höher: So muss für jeden Knoten im Baum eine separate AABB erfasst werden; ebenso ist zur Speicherung der Aufteilung von Knoten die Erfassung der Koordinatenachse, entlang der ein Knoten aufgeteilt wurde, und des Verhältnisses der Aufteilung notwendig.

5.6.2. Einordnung von Punkten in k-d-Bäumen

Um den Baum-Knoten zu lokalisieren, in dem ein gegebener Punkt in einem k-d-Baum enthalten ist, ist pro Ebene des Baums nur ein einfacher Koordinaten-Vergleich notwendig: Ausgehend vom Wurzelknoten muss entlang der Koordinatenachse, entlang der der Knoten bei der Erstellung des Baums ursprünglich geteilt worden ist, nur die jeweilige Koordinate eines Punktes mit dem als Teil der Knoten-Parameter abgelegten Separations-Wert, verglichen werden. Die Suche wird rekursiv im rechten oder linken Teilbaum fortgesetzt, bis ein Blatt-Knoten erreicht wird. Andernfalls bricht der Suchvorgang ab, wenn die entsprechende Koordinate des gesuchten Punktes außerhalb des vom k-d-Baum erfassten Volumens liegt.

Neben der Lokalisierung einzelner Punkte innerhalb eines k-d-Baums sind auch Bereichs-Suchen wie die Ermittlung aller Punkte, die zu einem gegebenen Punkt einen vorgegebenen maximalen Abstand haben, möglich: Hier wird ein Suchvorgang nicht abgebrochen,

sobald ein Blatt-Knoten erreicht wird. Zusätzlich kann dann über die Hüllvolumina von Eltern-Knoten ermittelt werden, ob umgebende Knoten ebenfalls im räumlichen Bereich der Suchanfrage liegen, und mittels Backtracking die Suche in benachbarten Teilbäumen fortgesetzt werden.

KAPITEL 6

UMSETZUNG UND RESULTATE

Kapitel 6 zeigt die Ergebnisse von durchgeführten Experimenten mit den in Kapitel 5 vorgestellten alternativen Verfahren zur Kollisionserkennung.

Abschnitt 6.1 stellt zuerst kurz die GPU-basierte Implementierung des eigenen Ansatzes vor.

Abschnitt 6.2 diskutiert das für das eigene Verfahren gewählte Parallelisierungs-Schema.

Abschnitt 6.3 erläutert die technischen Rahmenbedingungen, unter denen das eigene Kollisionserkennungsverfahren im Rahmen der vorliegenden Arbeit getestet wurde.

Abschnitt 6.4 beschäftigt sich mit Szenarien, die zur Erprobung des eigenen Verfahrens verwendet wurden, und dem resultierenden Laufzeitverhalten des Verfahrens unter verschiedenen Randbedingungen.

Abschnitt 6.5 diskutiert diese Ergebnisse schließlich in Relation zu der in Kapitel 5 behandelten Funktionsweise des eigenen Verfahrens.

6.1. Details zur GPU-basierten Implementierung des Verfahrens

Die Implementierung des eigenen Verfahrens folgt der üblichen Unterteilung in eine vorhergehende Vorfilter-Phase sowie der anschließenden Ausführung von Paar-Tests zwischen potentiell kollidierenden Objekten. Diese ist wiederum unterteilt in Paar-Tests zwischen den Hüllkörpern von Konglomeraten aus den jeweiligen Objekten sowie die Ermittlung potentiell kollidierender Seitenflächen mittels der Einordnung der Punkthülle eines Konglomerats in die Raumpartitionierung eines zweiten Konglomerats. Als Resultat dieser Einordnung können Paare von Seitenflächen bestimmt werden, die nahe aneinander liegen und im letzten Schritt des Verfahrens vollständig auf möglichen Überschneidungen hin überprüft werden müssen.

Im Unterschied zu rein polygon-basierten Verfahren (Abschnitt 4.2) und reinen Raumpartitionierungs-Verfahren wie Voxmap-Pointshell (Abschnitt 4.4) setzt die Imple-

mentierung des eigenen Verfahrens darauf, die jeweiligen Vorteile dieser unterschiedlichen Verfahrens-Klassen in bestmöglicher Kombination einzusetzen.

Polygonale Geometriebeschreibungen und daraus erstellte Hüllkörper-Hierarchien bieten den Vorteil, potentiell kollidierende Geometriebereiche schnell einzugrenzen, und anhand Paar-Tests auf der Ebene von Seitenflächen Kontaktpunkte präzise bestimmen zu können. Problematisch hinsichtlich des Einsatzes in Kombination mit GPU-Prozessoren ist der nötige Aufwand, eine optimale Parallelisierung der paarweisen Traversierung zweier Hüllkörper-Hierarchien gewährleisten zu können. Ebenso ist das Übertragen der Datenstrukturen von Hüllkörper-Hierarchien und der eigentlichen Geometrie-Daten in den Speicherbereich eines GPU-Prozessors ein Nachteil, da nicht im Voraus genau vorhersagbar ist, welche Teile von Geometriebeschreibungen und Hierarchien im Verlauf der paarweisen Traversierung tatsächlich benötigt werden.

Raumpartitionierungs-Verfahren sind dem gegenüber prinzipiell sehr gut für den Einsatz auf massiv parallelen Prozessor-Architekturen geeignet: Der eigentliche Kollisions-Test besteht nur aus der Einordnung von Punkt-Primitiven in eine Raumpartitionierung. Dabei bestehen im Gegensatz zur paarweisen Hüllkörperhierarchie-Traversierung keine Abhängigkeiten bezüglich der Reihenfolge der durchgeführten Kollisions-Tests, weder hinsichtlich der hierarchischen Anordnung noch hinsichtlich der gewählten Abstiegsregel. Damit wird die gleichzeitige Ausführung einer großen Anzahl von Einordnungs-Operationen möglich, da die Operation darüber hinaus keinen schreibenden Zugriff auf die Daten einer Raumpartitionierung selbst benötigt.

Der Nachteil beim Einsatz von Raumpartitionierungen ist der gegenüber polygonalen Oberflächenbeschreibungen größere Speicherplatzbedarf für die Erfassung von Punkt-Primitiven und Raumpartitionierungen selbst. Obwohl der Octree bei Letzteren hierin aufgrund seiner Konstruktionsweise eine Ausnahme darstellt, gilt dies für andere Varianten von Raumpartitionierungs-Verfahren in weitaus größerem Maß: Zwar ist der Speicherplatzbedarf bei Hüllkörper-Hierarchien pro Baum-Knoten an sich je nach gewählter Hüllkörper-Variante höher als beispielsweise für die Spezifikation eines Knoten in einem k-d-Baum. Da die Anzahl von zur Erstellung einer Raumpartitionierung verwendeten Primitiven in Form einer Punktwolke jedoch wesentlich höher ist als bei polygonalen Geometriebeschreibungen etwa in Form einer Vertex-Index-Liste, ist die Zahl von Baum-Knoten in einer Raumpartitionierung insgesamt wesentlich größer als bei Hüllkörper-Hierarchien.

Um nun die Vorzüge beider Ansätze nutzen zu können, bedient sich das eigene Verfahren einer Kombination aus den jeweiligen Stärken polygon-basierter und Raumpartitionierungs-Verfahren, ergänzt um alternative Möglichkeiten zur Partitionierung großer Kollisionsgeometrien, wie sie in Abschnitt 5.2 vorgestellt worden sind. Besonders geeignet für die Erzeugung von Partitionierungen, die in Bezug auf die Anzahl enthaltener Seitenflächen eine möglichst kleine Abweichung voneinander aufweisen, haben sich dabei approximierende konvexe Zerlegungen wie durch HACD (Unterabschnitt 5.2.1) implementiert, erwiesen.

Das eigene Verfahren übernimmt aus polygon-basierten Verfahren die Funktionsweise abschließender Tests zwischen Paaren von Seitenflächen unverändert. Wie durch den Vergleich unterschiedlicher Varianten zur Bestimmung von Kontaktpunkt-Konfigurationen in Abschnitt 3.6 festzustellen war, erlaubt die Ermittlung von Kontaktpunkten anhand polygonaler Oberflächenbeschreibungen die bestmögliche Bestimmung nicht-redundanter Kontaktpunkt-

Mengen, im Gegensatz zu Kontaktmodellen wie dem des Voxmap-Pointshell-Algorithmus (Abbildung 4.18 in Unterabschnitt 4.4.3): Das dort verwendete Kontaktpunkt-Modell ist nicht für die Verwendung in impuls- oder kontaktkraft-basierten Lösungsverfahren gedacht, die hinsichtlich der Plausibilität und numerischen Stabilität einer Mechaniksimulation mit einer größeren Anzahl aktiv bewegter Objekte konzipiert sind.

Als Ersatz für die paarweise Traversierung von Hüllkörper-Hierarchien benutzt das eigene Verfahren Hüllkörper-Verbunde mit nur einer Hierarchie-Ebene und einem entsprechend vielfach größeren Verzweigungsgrad. Diese Entscheidung begründet sich durch den Verwaltungsaufwand für die Koordination einer paarweisen Hüllkörper-Traversierung, wie sie im Fall von gProximity durch eine Rebalancierung von Tests zwischen Teilbäumen und bei HPCCD durch eine gastsystem-seitige Warteschlange umgesetzt werden.

Obwohl die genannten Verfahren selbst sehr laufzeiteffizient sind und durch die Autoren jeweils gezeigt worden ist, dass sie sowohl zur Simulation größerer Objekt-Mengen als auch von Szenarien mit komplexem mechanischem Verhalten wie plastisch verformbaren oder zerbrechenden Objekten in der Lage sind, so ist bei beiden Verfahren eine bestimmte Art von Objekt-Konfigurationen nicht vertreten: Simulationen mit einer größeren Anzahl detailliert konstruierter Geometrien, wie sie Gegenstand der in Kapitel 2 beschriebenen Anwendungen in der Robotik sind. Für diese Anwendungen ist es wichtig, einen ausgewogenen Kompromiss zwischen der maximal in oder nahe Echtzeit simulierbarer Objekte und deren geometrischer Präzision bieten zu können.

Um diesen Kompromiss zu erreichen, müssen bei der GPU-basierten Umsetzung eines Kollisionserkennungs-Verfahrens folgende Faktoren berücksichtigt werden:

- Sicherstellen eines hohen Parallelisierungs-Grads bei der Ausführung eines Kollisionserkennungs-Verfahrens, und damit verbunden einer möglichst gleich verteilten Auslastung von GPU-Prozessoren.
- Minimierung des nötigen Datentransfers zwischen den Speicherbereichen des Gastsystems und GPU-Prozessoren.
- Gewährleisten einer möglichst unterbrechungsfreien Ausführung GPU-basierter Berechnungen; verbunden damit ist die weitgehende Vermeidung eines mehrfachen Wechsels zwischen CPU- und GPU-basierten Berechnungen pro Iteration.

Für die erste und zweite Phase (Vorfilter- und Objektpaar-Phase) des eigenen Kollisionserkennungs-Verfahrens wird daher folgender Ansatz gewählt: Die Vorfilter-Phase wird rein CPU-basiert ausgeführt, ebenso der erste von zwei Schritten im Objektpaar-Test. Der eigentliche Objektpaar-Test zur Ermittlung potentiell kollidierender Paare von Seitenflächen wird komplett GPU-basiert ausgeführt, ebenso wie die abschließenden Paar-Tests zwischen Seitenflächen zur präzisen Bestimmung von Kontaktpunkten.

Die Vorfilter-Phase erfolgt dabei auf Basis objekt-ausgerichteter Hüllquader (OBBs), wobei diese im Gegensatz zu konventionellen Hüllkörper-Hierarchien nur zweistufig und nicht in der üblichen Modellierung als Baum angeordnet sind: Die für die szenenweite Vorfilter-Phase verwendete OBB einer Geometrie wird dabei als Hüllquader aller Hüllquader der Konglomerate gebildet, die für eine Kollisionsgeometrie berechnet worden sind. Wird während der szenenweiten Vorfilter-Phase eine Überschneidung zwischen zwei übergeordneten OBBs festgestellt,

so werden für das betreffende Objektpaar in einem nächsten Schritt alle konglomerat-eigenen OBBs auf mögliche Überschneidungen überprüft.

Jedes Paar von Konglomeraten, deren OBBs sich überschneiden, wird im dritten Schritt des Verfahrens für einen GPU-basierten Paar-Test vorgemerkt. Eine Liste der Indizes aller potentiell kollidierenden Paare von Konglomeraten wird in den Speicherbereich der GPU übertragen, ebenso wie die Daten aller Konglomerate, die noch nicht im GPU-Speicher vorhanden sind.

Der GPU-basierte Paar-Test wird schließlich durch die Verwendung der Punkthülle und der Raumpartitionierung jeweils eines Konglomerats in einem Paar-Test eingeleitet: Die Punkthülle eines Konglomerats wird aus der in Abschnitt 5.3 diskutierten komprimierten Darstellung expandiert und in das lokale Koordinatensystem des zweiten Konglomerats transformiert. Für jeden Punkt der Punkthülle des ersten Konglomerats wird dessen Einordnung in den Octree oder k-d-Baum des zweiten Konglomerats berechnet. Sofern ein Punkt in den Bereich eines Blattknotens eingeordnet wird, so wird das zugehörige Flächenpaar als Kandidat für einen präzisen Paar-Test im letzten Schritt des Verfahrens vermerkt. Um ohne eine separate Suche nach der Seitenfläche auszukommen, die jeweils einem Punkt oder einem Blattknoten zugeordnet ist, sind sowohl die einzelnen Punkte der Punkthülle eines Konglomerats als auch die Blattknoten in Octrees beziehungsweise k-d-Bäumen mit dem Index der Seitenfläche versehen, mit der sie während der Konstruktion der jeweiligen Datenstruktur assoziiert waren.

Zur Einordnung von Punkt-Primitiven eines Konglomerats in die Raumpartitionierung eines anderen werden die in Abschnitt 5.5 beziehungsweise Abschnitt 5.6 vorgestellten Verfahren verwendet: Morton-Schlüssel für Octrees, die rekursive Suche nach einem passenden Blattknoten im k-d-Baum. Diese Suchoperationen sind durch ihre massiv parallele Ausführung auf GPU-Hardware sehr gut dazu geeignet, als Alternative für die in gProximity und HP-CCD GPU-basiert umgesetzte paarweise Hüllkörperhierarchie-Traversierung beziehungsweise warteschlangen-basierte Front-Tracking-Methode zu dienen. Auch wenn die Anzahl der durchgeführten elementaren Kollisions-Tests wesentlich größer ist als bei der Kombination von paarweisen Hüllkörper-Tests mit anschließenden Paar-Tests zwischen Seitenflächen, so ist die Struktur eines einzelnen Kollisions-Tests wesentlich einfacher: So ist die Berechnung eines Morton-Schlüssels für einen Punkt mit wenigen Fließkommazahl-Divisionen und bitweisen Operationen zu implementieren.

Kombiniert mit der massiv parallelen Ausführung mittels eines GPU-Prozessors sowie dem Umstand, dass sich aus der Einordnung einer Punkthülle in eine Raumpartitionierung ebenso die räumliche Nähe von Seitenflächen eines Paars von Geometrien rekonstruieren lässt wie bei der Verwendung von Hüllkörper-Hierarchien, ist die abgewandelte Benutzung von Suchverfahren aus dem Bereich der Punktwolken-Verarbeitung eine vielversprechende Alternative zu anderen GPU-basierten Kollisionserkennungs-Verfahren, wie sie in Abschnitt 4.2 und Abschnitt B.3 diskutiert werden.

Nicht nur hinsichtlich der effizienten Funktionsweise elementarer Kollisions-Tests auf Basis von Punkthüllen und Octrees oder k-d-Bäumen birgt das vorgeschlagene eigene Verfahren ein großes Leistungspotential in sich: Da die Aufteilung der pro Iteration durchzuführenden Paar-Tests nicht erst während der GPU-seitigen Ausführung ergibt, sondern auf der Basis von Konglomerat-Paaren vor Beginn der GPU-basierten Berechnungen erfolgt, ist

der Verwaltungsaufwand zur Koordination des Ablaufs paralleler Berechnungen auf einer GPU weitaus geringer als etwa bei gProximity oder HPCCD. Ebenso ist ein weiterer positiver Nebeneffekt dieser Entkopplung zwischen Vorfilter-Phase und paarweisen Konglomerat-Tests der Wegfall von wechselseitigen Abhängigkeiten während der Laufzeit von Kernel-Instanzen, welche nach den in Unterabschnitt 4.3.3 gezeigten Fallstudien anhand unterschiedlicher Parallelisierungs-Schemata einen negativen Einfluss auf die Leistungsfähigkeit von Kollisionserkennungs-Verfahren haben: Die Ausführung einzelner Paar-Tests zwischen jeweils zwei Konglomeraten erfordert einerseits keinerlei schreibende Zugriffe auf die jeweiligen Datenstrukturen, und andererseits läuft sie jeweils völlig entkoppelt von gleichartigen Operationen zwischen anderen Paaren von Konglomeraten ab.

Betrachtet man den Zusammenhang zwischen raumpartitionierungs-basierten Kollisionserkennungs-Verfahren wie Voxmap-Pointshell und anderen, nicht auf der Verwendung polygonaler Geometriebeschreibungen basierenden Verfahren, so ist ein weiterer wichtiger Unterschied zum hier vorgestellten eigenen Verfahren erkennbar: Letzteres bedient sich Raumpartitionierungen und Punkthüllen nicht als geometrischer Primitive zur Erkennung möglicher Kontaktpunkte, sondern ausschließlich als Hilfsmittel zur Suche nach räumlich benachbarten Paaren von Seitenflächen in Konglomeraten. Endgültige Kontaktpunkte werden wie bereits beschrieben in einem abschließenden Schritt auf Basis einer polygonalen Oberflächenbeschreibung ermittelt. Damit sind die Vorteile beider Verfahrensklassen auf neue Weise vereint: Die Leistungsfähigkeit punkt-basierter Kollisionsberechnungen wird als Ersatz für Hüllkörper-Hierarchien eingesetzt, während gleichzeitig die Vorteile polygon-basierter Oberflächenbeschreibungen für die präzise Berechnung minimaler Kontaktpunkt-Konfigurationen und den daraus resultierenden Vorzüge für die Plausibilität und numerische Stabilität der nachgelagerten Mechaniksimulation erhalten bleiben.

6.2. Das Parallelisierungs-Schema des eigenen Ansatzes

Die Zusammenstellung von Aufgaben für die GPU-basierten Berechnungen des Verfahrens, sowie die Koordination von Kernel-Aufrufen und der Datenübertragung zwischen GPU und Gast-System sind weitere zu lösende Probleme bei der Umsetzung des eigenen Verfahrens. Diese sollen im folgenden Abschnitt erläutert werden.

6.2.1. Zur Zusammenstellung von Berechnungs-Aufgaben für die GPU: Arbeitseinheiten

Die Struktur von Objektpaar-Tests, die sich durch die Strukturierung von Kollisionsgeometrien mittels Konglomeraten im eigenen Verfahren ergibt, erlaubt im Gegensatz zur paarweisen Traversierung von Hüllkörper-Hierarchien die simultane Ausführung einer großen Anzahl von Kollisions-Tests zwischen Paaren von Komglomeraten: Bei Hüllkörper-Hierarchien erfordert die Koordination der GPU-basierten Ausführung einer paarweisen Traversierung, die von der Struktur der Hierarchien, der Traversierungs-Reihenfolge und der Anzahl sich überschneidender Hüllvolumina pro Hierarchie-Ebene bestimmt wird, wie in Unterabschnitt 4.3.1 und Unterabschnitt 4.3.2 eine Ergänzung durch einen expliziten Rebalancierungs-Schritt oder

durch eine gastsystem-seitige Warteschlangenverwaltung für Paartests zwischen Seitenflächen: Nur so kann eine möglichst gleichmäßige Auslastung einer GPU im Verlauf der paarweisen Traversierung sichergestellt werden.

Für das eigene Verfahren ist ebenfalls eine Warteschlangenverwaltung nötig, die allerdings nicht auf die Ebene einzelner Objektpaar-Tests beschränkt ist: Durch die Verwendung einer flachen Hierarchie mit hohem Verzweigungsgrad zur Zusammenfassung der Hüllvolumina einzelner Konglomerate in Kollisionsgeometrien können auch Konglomerat-Paare aus verschiedenen von der szenenweiten Vorfilter-Phase ermittelten Objektpaaren simultan auf mögliche Kollisionen geprüft werden. Das erlaubt der Warteschlangenverwaltung des eigenen Verfahrens, auf einer zweiten Ebene neben der parallelisierten Ausführung der Einordnung von Punkt-Hüllen in Octrees in einem Konglomeratpaar-Test parallelisierte Berechnungen anzustoßen: Die simultane Ausführung mehrerer Objektpaar-Tests, sowie die Separierung von Objektpaar-Tests über mehrere Iterationen der GPU-basierten Teile des eigenen Verfahrens hinweg.

Allerdings gibt es zwei Einflußfaktoren, auf die im Zusammehang mit der Warteschlangenverwaltung des eigenene Verfahrens geachtet werden sollte:

1. Abhängig von der Anzahl der Objekte in einer Simulation, deren geometrischer Struktur und der relativen Positionierung zueinander kann die Gesamtzahl zu testender Konglomerat-Paare zwischen verschiedenen Iterationen einer Simulation stark schwanken.
2. GPU-Prozessoren können nur eine bestimmte maximale Anzahl von Kernel-Instanzen pro Streaming-Prozessor und Grid gleichzeitig ausführen.

Daher wird für das eigene Verfahren eine Warteschlangenverwaltung verwendet, die Konglomeratpaar-Tests unter Berücksichtigung dieser beiden Einflußfaktoren in *Arbeitseinheiten* gemäß der Vorgehensweise in Abbildung 6.1 aufteilt.

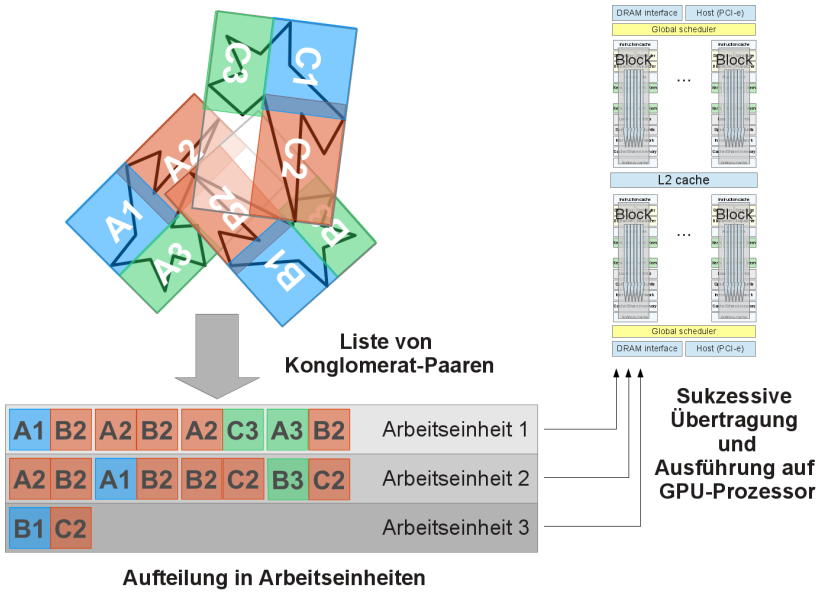


Abbildung 6.1.: Skizze des Parallelisierungs-Schemas des eigenen Verfahrens

Eine Liste aller durchzuführenden Konglomeratpaar-Tests wird in Arbeitseinheiten unterteilt, deren Größe sich anhand der vom verwendeten GPU-Prozessor maximal gleichzeitig ausführbaren Kernel-Instanzen orientiert. Diese Arbeitseinheiten werden zusammen mit den Datenstrukturen der zu testenden Konglomerate (sofern diese nicht bereits aus einer früheren Iteration bereits resident im Speicher der GPU sind) in den Speicher der GPU kopiert, und ein übergeordneter Kernel, der die Ausführung der octree-basierten Tests sowie die Überprüfung von Seitenflächen-Paaren koordiniert, wird für jede der zusammengestellten Arbeitseinheiten CPU-seitig aufgerufen.

6.2.2. Zur Koordination von Kernel-Aufrufen

Zur Durchführung der Laufzeitmessungen in Abschnitt 6.4 wurde die Fähigkeit der CUDA-Version 5.0 ausgenutzt, die es bereits auf einer GPU aktiven Kernel-Instanzen erlaubt, andere Kernel GPU-seitig zu starten (Dynamic Parallelism[1]). Diese Funktion erlaubt es übergeordneten Kernel-Instanzen, ihrerseits sukzessive Kernel-Instanzen zu starten, die für die Durchführung der octree-basierten Suche nach potentiell kollidierenden Seitenflächen sowie der Bestimmung von Kontaktpunkten anhand Paaren von Seitenflächen eines Konglomerat-Paars verantwortlich sind.

Die für die Laufzeitmessungen verwendete Implementierung war dabei nur mit einer statischen maximalen Größe für eine Arbeitseinheit realisiert; die Daten der Konglomerat-Paare in den Szenarien zur Laufzeitmessungen waren dabei im Rahmen der Vorverarbeitungsphase zur Erstellung von Konglomeraten bereits im Speicherbereich der GPU allokiert. Aus diesem Grund konzentrieren sich die gemachten Laufzeitmessungen auf die Leistungsfähigkeit des eigenen Verfahrens beschränkt auf einzelne Paar-Tests zwischen Kollisionsgeometrien. Die Erweiterung des Verfahrens zur Nutzung mehrerer Arbeitseinheiten pro Iteration der Simulation mit variabler Größe der Arbeitseinheiten (auch zur Verteilung der Berechnungen auf verschiedene GPU-Prozessoren, sofern verfügbar) erfordert jedoch nur Modifikationen an der gastsystem-seitigen Warteschlangenverwaltung: Die Gestaltung der GPU-basierten Berechnungen ermöglicht es, durchzuführende Konglomeratpaar-Tests flexibel nach der Anzahl verfügbarer GPU-Prozessoren und deren jeweiligen hardware-seitigen Fähigkeiten zu verteilen.

6.2.3. Zum Datentransfer zwischen GPU und Gast-System

Die Übertragung von Daten zwischen Gastsystem und GPU setzt sich aus folgenden Typen von Datenstrukturen zusammen:

1. Die Geometriebeschreibungen, Raumpartitionierungen und Punkt-Hüllen von Konglomeraten.
2. Listen von Konglomeratpaaren als Bestandteile von Arbeitseinheiten.
3. Listen von ermittelten Kontaktpunkten beziehungsweise der Indizes von als kollidierend erkannten Seitenflächen-Paaren unterschiedlicher Konglomerate.

[1] `http://developer.download.nvidia.com/assets/cuda/files/CUDADownloads/TechBrief_Dynamic_Parallelism_in_CUDA.pdf`

Arbeitseinheiten und Listen von Kontaktpunkten müssen bei jeder Durchführung der Kollisionserkennung neu zusammengestellt und auf in den Speicherbereich einer GPU übertragen, beziehungsweise nach Abschluss der Kollisionserkennung wieder in den Speicherbereich des Gastsystems zurück kopiert werden.

Datenstrukturen von Konglomeraten dagegen können resident auf der GPU gehalten werden und müssen nicht in jeder Iteration erneut in den Speicherbereich einer GPU übertragen werden. Obwohl in den Szenarien für Laufzeitmessungen in Abschnitt 6.4 eine statische Allokierung der Konglomerate aller verwendeten Kollisionsgeometrien ausreichend war (da der Speicherbedarf für die vom eigenen Verfahren bestimmten Konglomerate durch den auf der verwendeten GPU verfügbaren Speicher unproblematisch abgedeckt wurde), ist eine solche statische Allokierung bei Grafikkarten mit wenig eigenem Speicher oder bei der Verwendung umfangreicherer Kollisionsgeometrien keineswegs für alle denkbaren Szenarien praktikabel.

Für eine dynamische Verwaltung der im Speicherbereich der GPU residenten Daten von Konglomeraten wären verschiedene Strategien denkbar, die einerseits mit etablierten Methoden der Informatik, wie sie etwa bei der Verwaltung von Cache-Speichern eingesetzt werden, andererseits unter Ausnutzung der durch die Kollisionsbehandlung verfügbaren Informationen über den Bewegungsverlauf simulierter Objekte operieren könnten:

- Eine einfache Maßnahme wäre die Erweiterung der Datenstruktur von Konglomeraten um einen Zähler, der jedes Mal erhöht wird, wenn ein Konglomerat während einer Iteration der Kollisionserkennung in mindestens einem Paartest mit einem anderen Konglomerat verwendet wird. Sofern dieser Zähler über eine festgelegte Zahl von Simulationsschritten nicht mehr inkrementiert worden ist, kann ein solches Konglomerat aus dem Speicherbereich der GPU entfernt werden (entsprechend einer „Not recently used“-Strategie).
- Alternativ wären auch Strategien möglich, die nicht nur die absolute Zugriffshäufigkeit im Verhältnis zur Gesamtzahl von Iterationen berücksichtigen: Beispielsweise wären auch „Least Recently Used“-Strategien in verschiedenen Varianten denkbar, die die Entfernung von Konglomeraten aus dem residenten Speicher einer GPU über komplexere Kriterien bestimmen.
- Eine andere Variante wäre die zusätzliche Verwendung eines Kollisionserkennungs-Algorithmus, der die Eigenbewegung von simulierten Objekten berücksichtigt, wie es etwa bei gProximity (Unterabschnitt 4.3.1) über Rectangle Swept Spheres ([LGLM00]) der Fall ist. Für Objekte, die möglicherweise in einer der nächsten Iterationen miteinander kollidieren könnten, wäre aufgrund dieser Vorwärts-Simulation eine vorweg genommene Übertragung der zugehörigen Konglomerate in den Speicherbereich einer GPU möglich.

6.3. Technische Rahmenbedingungen

Nach der theoretischen Funktionsweise der GPU-basierten Implementierung des vorgestellten Verfahrens zur Kollisionserkennung in Abschnitt 6.1 soll der folgende Abschnitt die technischen Rahmenbedingungen der Implementierung erläutern:

- Die allgemeine Integration in das SOFA-Framework
- Eine kurze Beschreibung der Implementierung des gewählten Verfahrens zur Erzeugung von Konglomeraten
- Die Implementierung der CPU-basierten Vorfilter-Phase
- Die GPU-basierte Verwendung von Punkthüllen und Raumpartitionierungen, sowie die abschließenden Paar-Tests für Seitenflächen

6.3.1. Integration in SOFA

Das SOFA-Framework ermöglicht die Erweiterung vordefinierter Schnittstellen zur Integration von Kollisionserkennungs- und Kollisionsbehandlungs-Verfahren auf der Basis einer plugin-basierten Architektur. Als Teil der Spezifikation der Eigenschaften simulierter mechanischer Objekte kann für jedes Objekt unter anderem festgelegt werden, welche Art von Kollisionsmodellen auf Basis der polygonalen Beschreibung der verwendeten Kollisionsgeometrie erstellt werden sollen. Zur Auswahl zueinander kompatibler Kollisionsmodelle müssen Entwickler die zulässigen Kombinationen für Paar-Tests manuell als Teil der Initialisierung der Kollisionsmodelle spezifizieren.

Als Teil der Implementierung des eigenen Verfahrens wurden zwei Varianten von Kollisionsmodellen implementiert:

- Ein Modell auf Basis von OBB-Bäumen als Referenz-Modell auf Basis der Implementierung von gProximity.
- Ein Modell auf Basis von Punkthüllen und Raumpartitionierungen, in dem das eigene Verfahren zur Kollisionserkennung realisiert worden ist.

Die Software-Komponente, die im SOFA-Framework die Ausführung der Kollisionserkennung verwaltet, beruht auf der üblichen Zweiteilung zwischen Vorfilter- und Objektpaar-Phase, wobei für die Vorfilter-Phase auf Basis der jeweiligen Geometriebeschreibung eines Objektes jeweils ein AABB-Baum berechnet wird. Da das eigene Modell jedoch von der Verwendung von Hüllkörper-Hierarchien im konventionellen Sinn absieht, musste für die Integration in die Kollisionserkennung von SOFA eine eigene Komponente entwickelt werden, die während der Ausführung der Vorfilter-Phase durch SOFA abhängig von der Art des Kollisionsmodells entweder die in SOFA integrierten oder die eigenen Kollisionsmodell-Implementierungen verwendet.

Im Kontext der Objektpaar-Phase weicht die eigene Implementierung durch das gewählte Vorgehen ab, die szenenweite Vorfilter-Phase und die paarweise Überprüfung von Konglomerat-Paaren im selben Teilschritt vorzunehmen: Zuerst die Überprüfung der OBBs von ganzen Objekten, mit anschließendem Paar-Test aller konglomerat-spezifischen OBBs, sofern eine Überschneidung festgestellt wird. Das SOFA-Framework unterteilt die szenenweite Überprüfung und Objektpaar-Tests im Gegensatz dazu strikt in zwei separate Teilschritte.

Ebenso gibt es bei der Umsetzung der Objektpaar-Tests selbst Unterschiede zur Vorgehensweise im SOFA-Framework: Die Architektur der Kollisionserkennungs-Komponente ist entsprechend der Funktionsweise von Hüllkörperhierarchien umgesetzt. Die von SOFA verwendete Warteschlangenverwaltung geht davon aus, dass beim paarweisen Abstieg durch

Hüllkörper-Hierarchien Einträge für sukzessive kleinere Teilbäume erzeugt und iterativ abgearbeitet werden können. Das eigene Verfahren nutzt für diesen Schritt die massiv parallele Ausführung auf einem GPU-Prozessor und damit eine hardware-basierte Warteschlangenverwaltung. Daher wurde bei der Implementierung des eigenen Verfahrens auch die Schnittstelle der standardmäßig in SOFA eingesetzten Warteschlangenverwaltung für Objektpaar-Tests umgangen: Stattdessen wird im eigenen Verfahren direkt die GPU-basierte Berechnung für individuelle Konglomerat-Paare eingeleitet, sobald die Objektpaar-Phase selbst abgeschlossen ist.

6.3.2. Die Partitionierung von Kollisionsgeometrien

Die quelloffene Implementierung von HACD wurde im Rahmen der Arbeit ohne besondere Modifikationen zur Vorverarbeitung von Kollisionsgeometrien verwendet: Die aus von HACD vorgenommenen Zerlegung resultierenden Subgeometrien werden jedoch nicht selbst als Kollisionsgeometrien verwendet. HACD garantiert, dass die ursprüngliche Geometrie durch die einzelnen Teile der Zerlegung komplett umschlossen wird. Diese Eigenschaft wird für die Neustrukturierung der ursprünglichen Geometrien ausgenutzt: Alle Dreiecke einer Eingangs-Geometrie werden jeweils daraufhin überprüft, zu welcher Teil-Geometrie der Zerlegung sie gehören. So werden individuelle Seitenflächen jeweils einem Konglomerat zugeordnet, das auf der Grundlage der approximierenden konvexen Zerlegung von HACD bestimmt wird.

Im Zug der Integration von HACD als Werkzeug zur Zerlegung von Kollisionsgeometrien hat sich dieses Verfahren gegenüber den anderen in Abschnitt 5.2 beschriebenen Alternativen als bester Kompromiss zwischen Vollständigkeit der Zerlegung und des Größenverhältnisses von Konglomeraten zueinander (gemessen an der Anzahl von Seitenflächen pro Konglomerat) erwiesen. Aus diesem Grund wird HACD auch als bevorzugtes Verfahren für die Vorverarbeitung der in Abschnitt 6.4 benutzten Kollisionsgeometrien verwendet.

6.3.3. CPU-seitige Umsetzung der Vorfilter-Phase

Die Vorfilter-Phase sowie der erste von zwei Teilschritten der Objektpaar-Phase des eigenen Verfahrens läuft wie zuvor beschrieben exklusiv CPU-seitig ab. Dabei obliegt die Planung der Vorfilter-Phase der Kollisionserkennungs-Komponente des SOFA-Framework, der Einstieg in die Objektpaar-Phase wird jedoch unabhängig von den durch SOFA bereitgestellten Mechanismen vom eigenen Verfahren selbst übernommen.

6.3.3.1. Integration in die Vorfilter-Phase von SOFA

Zur Durchführung der Vorfilter-Phase wird durch SOFA jedes Objekt-Paar in einer Simulation anhand pro Iteration aktualisierter AABBs auf mögliche Überschneidungen analysiert. Das eigene Verfahren integriert sich in diese Überprüfung durch eine leicht modifizierte Implementierung der Planungskomponente für die Vorfilter-Phase: Anhand des Typs der überprüften Kollisionsmodelle wird entweder das Standardverhalten von SOFA verwendet, oder die Vorfilter-Phase des eigenen Verfahrens angestoßen, sofern beide überprüften Modelle eines Paar-Tests in der Vorfilter-Phase Instanzen der eigenen Kollisionsmodelle darstellen. In

diesem Fall wird anstatt der paarweisen Traversierung zweier AABB-Hierarchien die zuvor bereits erwähnte Überprüfung anhand Paaren von OBBs jeder paarweisen Kombination aus Konglomeraten eines Objekt-Paars ausgeführt.

6.3.3.2. Thread-basierte Ausführung der Bestimmung potentiell kollidierender Konglomerate

Um Tests zwischen Konglomerat-Paaren performant ausführen zu können, wurde eine multithreading-basierte Herangehensweise gewählt. Da die Paar-Tests wiederum nur lesenden Zugriff auf die Parameter der Hüllkörper von Konglomeraten benötigen, ist eine Aufteilung in folgender Art möglich: Pro Thread wird ein einzelner Hüllkörper eines Konglomerats gegen alle Hüllkörper der Konglomerate des anderen an einem Paar-Test beteiligten Objektes geprüft. Diese Herangehensweise berücksichtigt zwar beispielsweise nicht, dass ein Paar-Test zwischen zwei Konglomeraten nur einmal durchgeführt werden müsste, da jedes Paar aus Konglomeraten zweimal in unterschiedlichen Threads geprüft wird. Jedoch ist festzuhalten, dass die Anzahl nötiger Paar-Tests selbst unter diesen Umständen immer noch wesentlich geringer ist, als es bei einer paarweisen Traversierung von Hüllkörper-Hierarchien der Fall wäre. Ebenso ist zu beachten, dass die Ausführung von Paar-Tests anhand eines solchen Schemas keinerlei wechselseitige Abhängigkeiten zwischen den einzelnen Ausführungseinheiten beinhaltet. Dies ist ein vorteilhafter Nebeneffekt des abgewandelten Aufbaus der im eigenen Ansatz verwendeten, sehr flachen Hüllkörper-Verbände mit sehr hohem Verzweigungsgrad.

Die Koordination der Ausführung der erfolgt auf Basis eines Thread-Pools: Je zu überprüfendem Konglomerat wird ein Thread zur Abarbeitung der Konglomeratpaar-Tests gestartet. Die Ausführung der Threads untereinander muss dabei nicht koordiniert werden. Der Thread-Pool wartet die Beendigung der Ausführung aller gestarteten Threads ab, bevor die Liste durchzuführender GPU-basierter Konglomeratpaar-Tests (von Duplikaten bereinigt) in den Speicherbereich der GPU transferiert und die Objektpaar-Phase gestartet wird.

6.3.4. GPU-seitige Umsetzung der Objektpaar-Phase

Sobald die Liste durchzuführender Konglomeratpaar-Tests feststeht, werden die Daten der zu überprüfenden Konglomerate, welche noch nicht im Speicherbereich der GPU resident sind, auf die GPU kopiert. Danach wird für jedes zu prüfende Konglomerat-Paar mit der Einordnung der Punkthülle eines Konglomerats in die Raumpartitionierung des anderen am Paar-Test beteiligten Konglomerats begonnen.

6.3.4.1. Raumpartitionierungen und Punkthüllen

Hierfür wird zuerst die vollständige Punkthüllen-Repräsentation aus der komprimierten Darstellung als Bitmasken in Koordinaten-Tupel vorgenommen, wie in Unterabschnitt 5.3.3 beschrieben. Als Nächstes werden die Punkt-Koordinaten jedes Punktes des ersten Konglomerats in das objekt-spezifische Koordinatensystem der Geometrie transformiert, zu der das zweite Konglomerat in einem Paar-Test gehört. Jeder Punkt in objekt-spezifischen Koordinaten wird dann entsprechend der in Abschnitt 5.4 diskutierten Methoden (Morton-

Schlüssel für Octrees, Koordinatenvergleich für k-d-Bäume) in die Raumpartitionierung des zweiten Konglomerats eingeordnet. Anhand des Seitenflächen-Index, der für jeden Punkt einer Punkthülle aus der unbenutzten vierten Koordinaten-Komponente der Eckpunkte der Dreiecke eines Konglomerats, und für die Raumpartitionierung in den Blattknoten der jeweils verwendeten Baumstruktur hinterlegt ist, wird ein Paar von Seitenflächen als potentiell kollidierend angenommen, sofern ein Punkt aus der Punkthülle des ersten Konglomerats in einem Blattknoten des zweiten Konglomerats zu liegen kommt. Die so ermittelten paarweisen Indizes von Seitenflächen werden in ein Datenfeld im Speicherbereich der GPU hinterlegt (wiederum bereinigt um Duplikate).

6.3.4.2. Paar-Tests für potentiell kollidierende Seitenflächen-Paare

Sobald alle Kernel-Instanzen für die paarweise Überprüfung von Konglomeraten ihre Arbeit beendet haben, folgt im letzten Schritt der Berechnungen die exakte Bestimmung von Kontaktpunkten anhand eines Paar-Tests für Seitenflächen anhand der polygonalen Oberflächenbeschreibungen, die in jedem Konglomerat als Teil dessen Datenstruktur in Form zweier Datenfelder (eines für die Koordinaten der Eckpunkte, eines für die Indizes der im Konglomerat enthaltenen Dreiecke) hinterlegt sind.

Die Methode zum Schnitt-Test zwischen Paaren von Dreiecken ist eine modifizierte Variante der in Unterabschnitt 3.5.1 behandelten Vorgehensweise, die insgesamt 15 Schnittberechnungen anhand der einzelnen Ecken und Kanten von zwei Dreiecken (sechs Ecke-Fläche- und neun Kante-Kante-Tests) pro Dreiecks-Paar durchführt. Die Ausführung der abschließenden Seitenflächen-Tests folgt dabei demselben Prinzip, wie es etwa auch in der Implementierung von gProximity zum Einsatz kommt: Im Rahmen eines separaten Berechnungsprozesses anhand der Liste der Paar-Indizes, die während der vorhergehenden Konglomeratpaar-Tests erstellt worden ist. Die Eckpunkt-Koordinaten und Indizes der zu überprüfenden Dreiecke werden in kontinuierlich allokierten Speicherbereichen im GPU-Speicher abgelegt und die Ausführung der Kernel-Instanzen zur Schnittberechnung angestoßen. Die Kernel-Instanzen greifen dabei über ihren durch die hardware-basierte Warteschlangenverwaltung von CUDA bestimmten Index auf die Eckpunkt- und Index-Daten des zu überprüfenden Dreiecks-Paars in den zuvor allokierten Speicherbereichen zu. Die Ergebnisse der Berechnung werden in den Speicherbereich zurückgeschrieben, in dem ursprünglich alle zu überprüfenden Paare von Dreiecken hinterlegt waren. Zur Indikation einer festgestellten Kollision werden die Indizes der entsprechenden Dreiecks-Paare negiert.

6.3.5. Zur Leistungsmessung verwendete Hardware und Laufzeitumgebung

Die in Abschnitt 6.4 und Abschnitt 6.5 diskutierten Test-Szenarien, die zur praktischen Erprobung des eigenen Ansatzes zur Kollisionserkennung genutzt wurden, wurden als Szenenbeschreibungen im XML-basierten Dateiformat des SOFA-Frameworks erstellt. Die Kollisionsgeometrien selbst wurden im Wavefront OBJ-Format in Gestalt von Eckpunkt- und Index-Listen (Unterabschnitt B.1.3) gespeichert. Die für das eigene Verfahren notwendigen

Laufzeit-Daten (Punkt-Hüllen und Octrees) wurden wie in Kapitel 5 beschrieben ausschließlich als Teil von Vorverarbeitungs-Schritten berechnet.

Der verwendete Arbeitsplatz-Rechner war mit einem Intel i5 760 Quad-Core-Prozessor[2] mit 2,8 GHz Taktfrequenz mit 8 GB Arbeitsspeicher ausgerüstet. Der Rechner war mit einer GeForce GTX 780 Ti[3] als GPU-Prozessor ausgerüstet (mit einem Grafikkarten-Speicher von 3 GB), die in einem PCIe 2.0-Steckplatz verbaut war. Die GPU verfügt über 2880 CUDA-Kerne und unterstützt den CUDA Computing-Standard 3.5.

Die Leistungsmessung selbst wurde mit einem Linux-Betriebssystem (Ubuntu 13.10) unter Verwendung der SOFA-Version 1.0 (Release Candidate 1) durchgeführt. SOFA wurde mit der GNU C Compiler Suite Version 4.8.1 als Debug-Version übersetzt. Als CUDA-Laufzeitumgebung wurde die CUDA-Runtime in der Version 5.5 verwendet.

6.4. Verglichene Szenarien und Beurteilung des Laufzeitverhaltens des eigenen Verfahrens

Es wurden verschiedene Beispiel-Szenarien entworfen, um das Laufzeitverhalten des eigenen Verfahrens in Simulationen mit unterschiedlichen Objektgrößen und -konfigurationen geeignet beurteilen zu können. Die gewählten Szenarien haben das Ziel, das eigene Verfahren in unterschiedlichen Situationen zu beurteilen:

1. Ein Objektpaar mit insgesamt etwa 1500 Seitenflächen in einer räumlich nahen Kontakt-Konfiguration (Unterabschnitt 6.4.1).
2. Eine Szene mit zehn Objekten mit insgesamt etwa 4500 Seitenflächen mit jeweils fünf bewegten und fünf statischen Objekten (Unterabschnitt 6.4.2).
3. Eine Szene mit drei Objekten (ebenfalls in einer räumlich nahen Kontakt-Konfiguration), mit insgesamt etwa 7500 Seitenflächen. Für diesen Versuch wurde die CPU-basierte Paartest-Phase mittels OBBs jedoch deaktiviert und es wurden alle möglichen Kombinationen von Konglomeraten in Paartests zwischen den Objekten in der Szene in jeder Iteration gegeneinander getestet (Unterabschnitt 6.4.3).
4. Ein unbewegtes Objektpaar mit zwei identischen Kollisionsgeometrien mit jeweils etwa 10000 Seitenflächen, die jeweils im Koordinatenursprung positioniert sind (ebenfalls Unterabschnitt 6.4.3). Bei diesem Szenario wurden ebenfalls alle möglichen Paare von Konglomeraten in Paartests berücksichtigt.

Die ersten beiden Versuche dienen zur Beurteilung des Laufzeitaufwandes in Szenen sowohl mit einem einzelnen überprüften Objektpaar pro Iteration, als auch in einer Szene mit mehreren bewegten Objekten, die mit statischen Bestandteilen in der Szene kollidieren, und in der pro Iteration der Simulation multiple Paartests durchzuführen sind.

Der dritte und vierte Versuch konzentrieren sich dagegen auf die Beurteilung des nötigen Laufzeitaufwands mit Fokus auf den GPU-basierten Teil des eigenen Verfahrens, bestehend aus der Ermittlung potentiell kollidierender Seitenflächen anhand von Octrees und Punkt-

[2] `http://ark.intel.com/products/48496/Intel-Core-i5-760-Processor-8M-Cache-2_80-GHZ`
[3] `http://www.nvidia.com/gtx-700-graphics-cards/gtx-780ti`

Hüllen sowie Paartests zwischen Seitenflächen. In beiden Versuchen wurden dafür statische Szenen ohne Objektbewegung verwendet. Im dritten Versuch wurden hierzu drei Objekte nahe zueinander positioniert, und für Paartests zwischen Kollisionsgeometrien alle möglichen Kombinationen von Konglomeraten verwendet. Dies soll das Laufzeitverhalten der GPU-basierten Berechnungen bei der Überprüfung von Konglomerat-Paaren aufzeigen, jedoch ohne dass sich die überprüften Konglomerat-Paare notwendigerweise (teilweise oder vollständig) überschneiden. Im vierten Versuch wurden hingegen zwei identische Kollisionsmodelle an denselben räumlichen Koordinaten instanziiert, um eine Abschätzung des Laufzeitverhaltens sowohl unter der Überprüfung aller möglichen Konglomeratpaare in einem Paartest als auch der vollständigen Ausführung von Octree-Tests bei der Überdeckung von Punkt-Hüllen und Octrees identischer Kollisionsgeometrien zu erhalten.

6.4.1. Simulation mit einem Objektpaar

Das erste Szenario beschränkt sich auf ein einzelnes Objektpaar (dargestellt in Abbildung 6.2a) in einer räumlich nahen Ausgangsposition. Die verwendeten 3D-Modelle sind in den Beispiel-Szenarien des SOFA-Frameworks enthalten. Die Laufzeit des eigenen Verfahrens wurde über 250 Iterationen der Simulation hinweg einmal mit einem bewegten und einem statischen Objekt (Abbildung 6.3a), und einmal ohne Objektbewegung in einer statischen Szene ohne Objektbewegung gemessen (Abbildung 6.3b).

(a) Drahtgitter-Modell **(b)** Kollisionsgeometrien

Abbildung 6.2.: Modelle und Kollisionsgeometrien der im ersten Experiment verwendeten Objekte

Die Simulation ohne Berücksichtigung von Objektbewegungen wurde als Referenz für den direkten Vergleich mit einer Ausführung des selben Szenarios unter Berücksichtigung von Objektbewegungen ausgeführt.

In der Simulation mit Objektbewegung bewegte sich das Modell des Caduceus nur unter Einfluß der Schwerkraft in Richtung der negativen Z-Achse. In diesem Experiment war die Kollisionsbehandlung der Simulation nicht aktiviert, um im Verlauf des Experiments eine möglichst große Änderung der Kontaktsituation durch eine zunehmende Überschneidung zwischen den beiden Kollisionsgeometrien zu erzeugen. Das Ziel dieser Maßnahme war es zu ermitteln, ob eine Änderung der Anzahl zu überprüfender Geometrie-Teile zu erkennbaren Abweichungen bei den GPU-unterstützten Berechnungen des eigenen Verfahrens würde.

Ohne Berücksichtigung von Objektbewegungen zeigen die GPU-unterstützten Berechnungen ein konstant stabiles Laufzeitverhalten mit einer Gesamtlaufzeit von jeweils unter einer

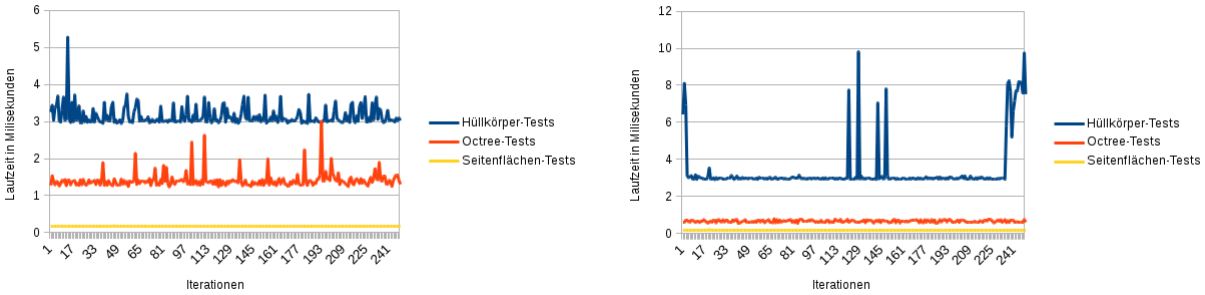


Abbildung 6.3.: Laufzeitverhalten des eigenen Verfahrens in zwei Ausführungen der Simulation aus Abbildung 6.2 über 250 Iterationen hinweg

Millisekunde, wobei die octree-basierten Berechnungen etwa 75 Prozent des gesamten Laufzeitaufwandes beanspruchen. Mit einem bewegten Objekt in der Szene verdoppelt sich der Laufzeitanteil der octree-basierten Berechnungen im Durchschnitt auf 1,5 Millisekunden je Iteration, während sich der Laufzeitanteil der Seitenflächen-Tests in diesem Szenario nicht wesentlich verändert.

Die Laufzeit der CPU-basierten Hüllkörper-Tests verhält sich im Vergleich zwischen statischer und bewegter Szene (abgesehen von einzelnen Abweichungen bei den Messungen in der statischen Szene) erwartungsgemäß: In der statischen Szene ändert sich die relative Position einzelner OBBs zueinander nicht; damit sind in jeder Iteration der Simulation dieselben Hüllquader-Paare zu überprüfen, und auch die Anzahl der nötigen SAT-Tests ändert sich in dieser Situation nicht. Im Gegensatz dazu verursacht die Bewegung eines Objekts Änderungen sowohl in der Anzahl zu überprüfender Hüllquader-Paare als auch in in der Anzahl nötiger SAT-Tests pro Paartest, was in einer durchschnittlich höheren Laufzeit der CPU-basierten Berechnungen resultiert.

6.4.2. Simulation mit mehreren bewegten Objekten

Das zweite gewählte Szenario besteht aus insgesamt zehn Modellen von Zahnrädern, die wie in den Abbildung 6.4a und Abbildung 6.4b gezeigten Ausgangspositionen arrangiert waren. Jedes der Modelle besteht aus 300 bis 500 Seitenflächen; für das zweite Experiment waren jeweils fünf Modelle als statische Objekte und als frei bewegte Objekte konfiguriert. In diesem Versuch war die Kollisionsbehandlung von SOFA aktiviert, um das Verhalten des eigenen Verfahrens unter Einbeziehung mechanischem Verhaltens zeigen zu können.

Die bewegten und statischen Instanzen der verwendeten Modelle sind dabei so positioniert, dass jeweils ein statisches und ein bewegtes Objekt in unmittelbarer Nähe zueinander liegen. Die bewegten Objekte unterlagen während der Ausführung der Simulation wiederum nur der Schwerkraft.

Wie in den Diagrammen in Abbildung 6.5 zu sehen ist, erfolgt die Ausführung octree-basierter Berechnungen in allen sechs Kombinationen der dargestellten Paartests innerhalb maximal zehn Millisekunden, im Durchschnitt benötigen diese Berechnungen etwa sieben

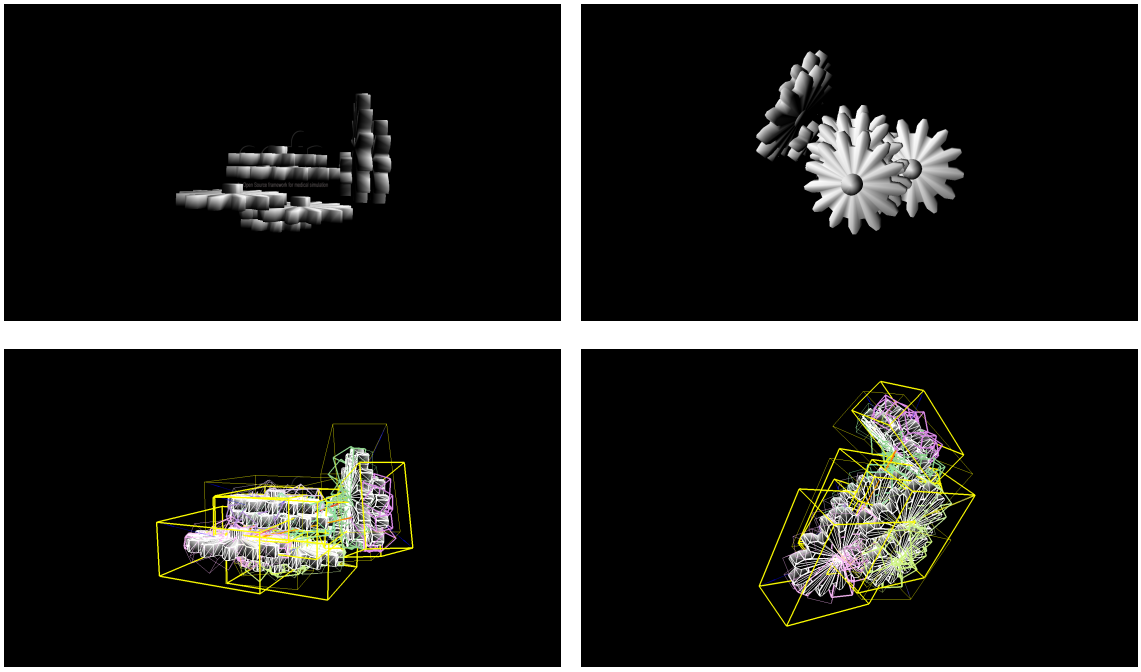


Abbildung 6.4.: Modelle und Kollisionsgeometrien der im zweiten Experiment verwendeten Objekte

Millisekunden. Die Überprüfung von Seitenflächen-Paaren benötigen in dieser Simulation mit Laufzeiten zwischen etwa 0,3 und 1,5 Millisekunden.

Der Laufzeitbedarf von Hüllquader-Tests pro Objektpaar-Test bewegt sich in diesem Szenario etwa im selben Bereich wie im ersten Experiment mit einem einzelnen Objektpaar.

6.4.3. Simulationen zur Beurteilung des ungünstigsten Laufzeitverhaltens

Um das Laufzeitverhalten des eigenen Verfahrens für im Fall extremer Rahmenbedingungen beurteilen zu können, wurden zwei unterschiedliche Szenarien angelegt:

1. In der ersten Variante wurde die Überprüfung von Hüllquader-Paaren absichtlich abgeschaltet, und stattdessen eine vollständige Überprüfung aller Konglomerat-Paare in einem Objektpaar-Test erzwungen.
2. In der zweiten Variante wurden zusätzlich dazu die verwendeten Kollisionsgeometrien an denselben Koordinaten platziert, um neben der vollständigen Überprüfung aller Konglomerat-Paare auch noch deren vollständige räumliche Überdeckung zu gewährleisten.

6.4.3.1. Versuch mit erzwungener Überprüfung aller Konglomeratpaare

Für den Versuch mit einer erzwungenen Überprüfung aller Konglomerat-Paare ohne räumliche Überlagerung der verwendeten Kollisionsgeometrien wurde die in Abbildung 6.6 gezeigte Objektkonfiguration verwendet. Die Szene wurde für die Laufzeit des Experiments als sta-

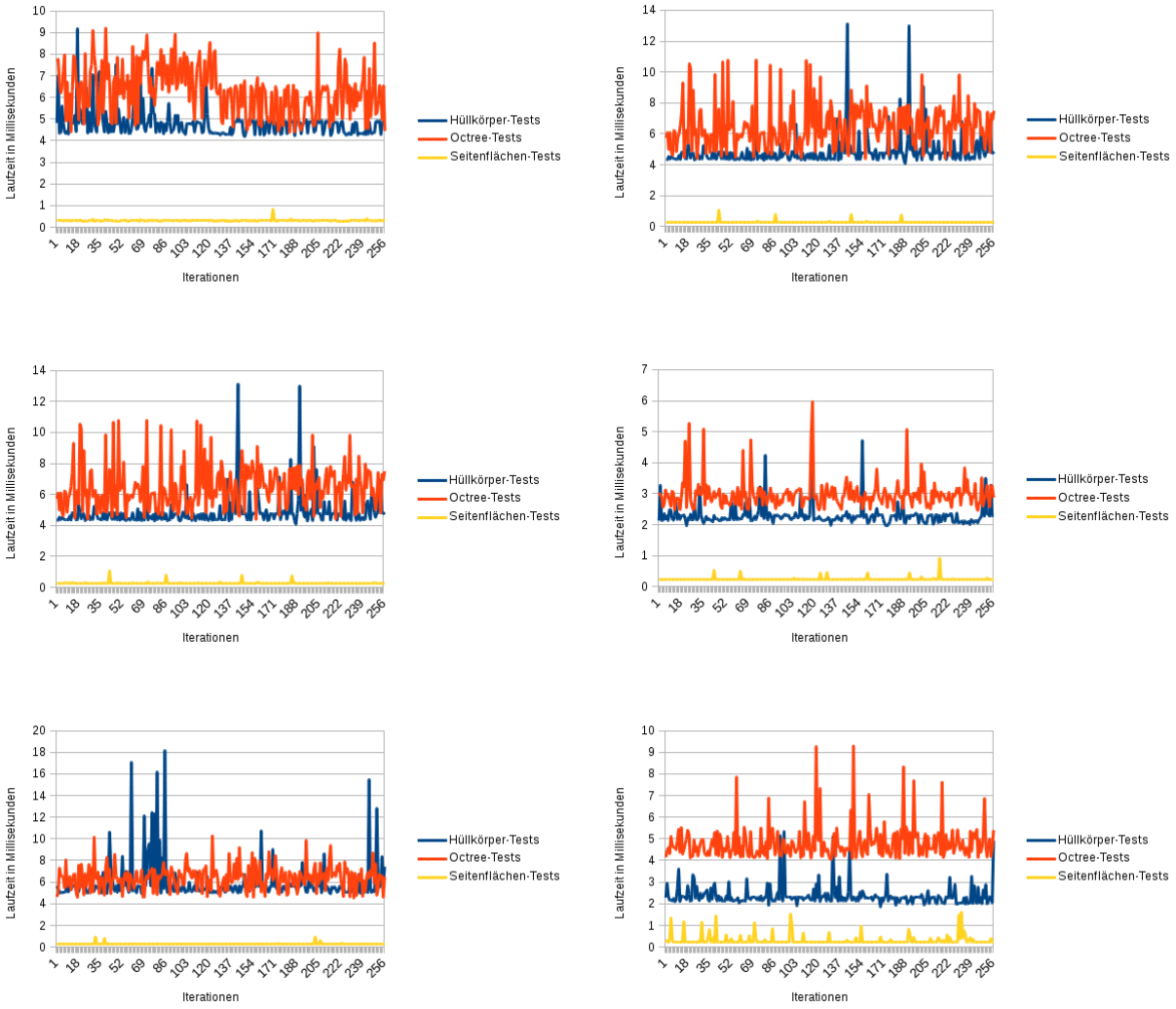


Abbildung 6.5.: Laufzeitverhalten des eigenen Verfahrens mit den Kollisionsgeometrien aus Abbildung 6.4 über 250 Iterationen hinweg. Jedes Diagramm erfasst jeweils den Laufzeitbedarf von einzelnen Objektpaar-Tests zwischen Kollisionsgeometrien, die über alle 250 Iterationen hinweg an Kollisionstests beteiligt waren.

tisch konfiguriert, da Objektbewegungen und damit Änderungen der Anzahl überlappender Hüllquader keinen Einfluß auf den Laufzeitaufwand der Kollisionserkennung gehabt hätte: Die erzwungene Durchführung aller möglichen Kombinationen aus Konglomeratpaaren in diesem Experiment hebt den Unterschied zwischen einer statischen Szene und einer Szene mit bewegten Objekten im Rahmen der Kollisionserkennung auf.

Die in Abbildung 6.6b dargestellten Kollisionsmodelle bestehen aus jeweils etwa 1300 Dreiecken für die Modelle der beiden Schraubenzieher, und etwa 4500 Dreiecken für das Modell des Kurbelwellen-Gehäuses.

Die in Abbildung 6.7 gezeigten Laufzeitdiagramme für die zwei in der statischen Szene durchgeführten Objektpaar-Tests (je ein Schraubenzieher-Modell mit dem Kurbelwellen-Gehäuse) zeigen die für die Einordnung der Konglomerate der Schraubenzieher-Modelle in die Konglomerate des Kurbelwellen-Gehäuses verbrauchte Laufzeit. Die gegenüber den beiden vorher-

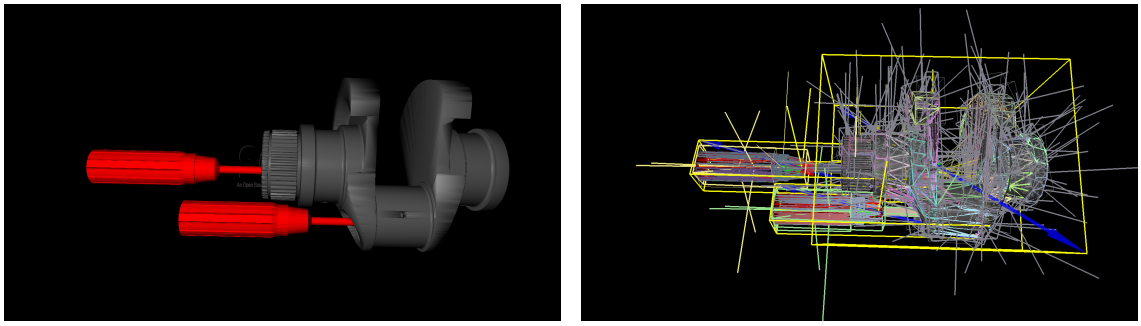


Abbildung 6.6.: Modelle und Kollisionsgeometrien der im dritten Experiment verwendeten Objekte

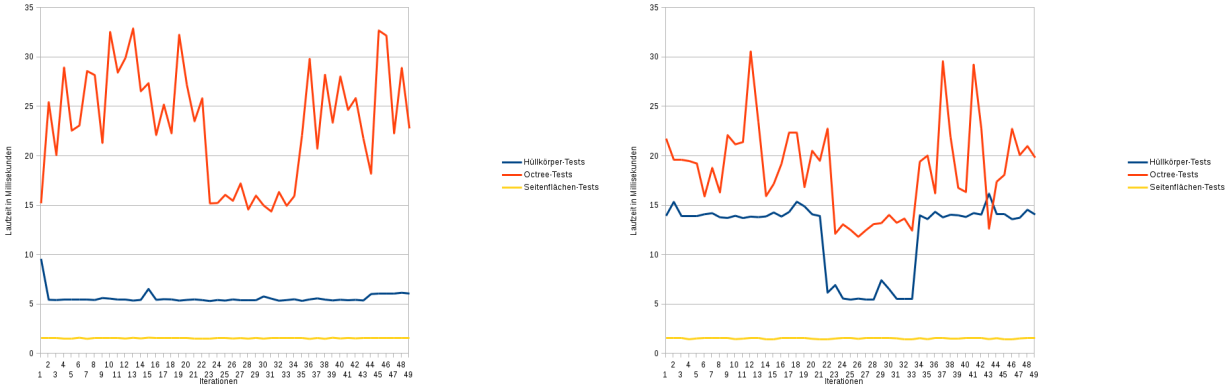


Abbildung 6.7.: Laufzeitverhalten des eigenen Verfahrens mit den Kollisionsgeometrien aus Abbildung 6.6 über 50 Iterationen hinweg.

gehenden Experimenten um den Faktor 2 bis 2,5 höhere Laufzeit erklärt sich dabei durch die jeweilige Aufteilung der Kollisionsmodelle in Konglomerate: Während die Schraubenzieher-Geometrie in nur vier Konglomeraten zerlegt wurde, besteht das Modell des Kurbelwellen-Gehäuses aus insgesamt 71 Konglomeraten. Dadurch ergibt sich bedingt durch die im eigenen Verfahren gewählte Parallelisierungs-Schema eine höhere Anzahl auf der GPU auszuführender Kernel-Instanzen gegenüber den beiden vorhergehenden Szenarien.

Dennoch bleibt festzuhalten, dass die octree-basierten Berechnungen als entscheidende Phase des eigenen Verfahrens trotz der in diesem Experiment künstlich herbeigeführten Situation, in der alle Konglomeratpaare ohne Rücksicht bestehende oder nicht bestehende Überlappungen zwischen deren Hüllkörpern trotzdem in der Lage ist, die Kollisionserkennung nur mit linear ansteigendem Laufzeitbedarf bewältigen zu können.

6.4.3.2. Versuch mit einem Paar vollständig überlappender Kollisionsgeometrien

Das letzte durchgeführte Experiment wurde unter Verwendung des in Abbildung 6.8a dargestellten Modells eines KUKA-Roboterarms durchgeführt. Dafür wurden zwei identische Instanzen desselben Modells übereinander instanziiert, und gleichzeitig die Überprüfung aller möglichen Konglomeratpaare erzwungen.

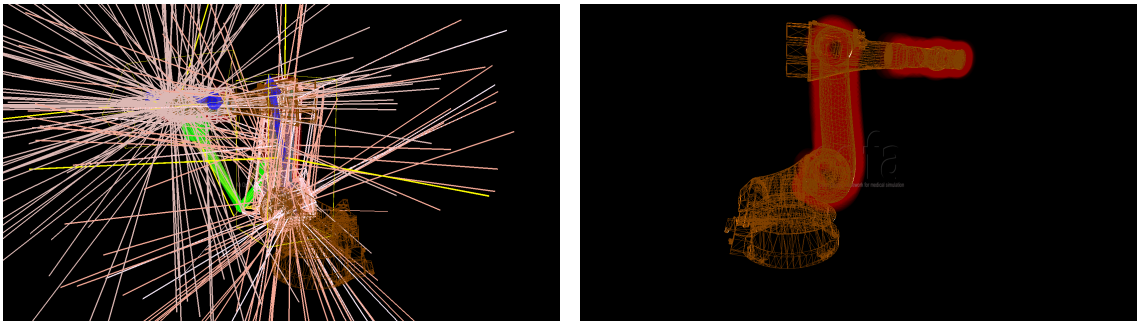


Abbildung 6.8.: Modell und Kollisionsgeometrie des im vierten Experiment verwendeten Objekts

Die in Abbildung 6.9 gezeigten Laufzeitdiagramme beschränken sich auf Objektpaar-Tests, die zwischen Paaren der in Abbildung 6.8b rot hinterlegten Segmenten des Modells durchgeführt wurden. Auch in diesem Experiment wurde wiederum eine statische Szene betrachtet. Durch die explizit in diesem Experiment erzwungene Überprüfung aller möglichen Kombinationen von Konglomeratpaaren im Objektpaar-Test ergeben sich insgesamt drei mögliche Kombinationen: Jeweils eine Kollisionsgeometrie mit sich selbst (Abbildung 6.9a und Abbildung 6.9b), plus ein Paartest zwischen beiden Kollisionsgeometrien (Abbildung 6.9c).

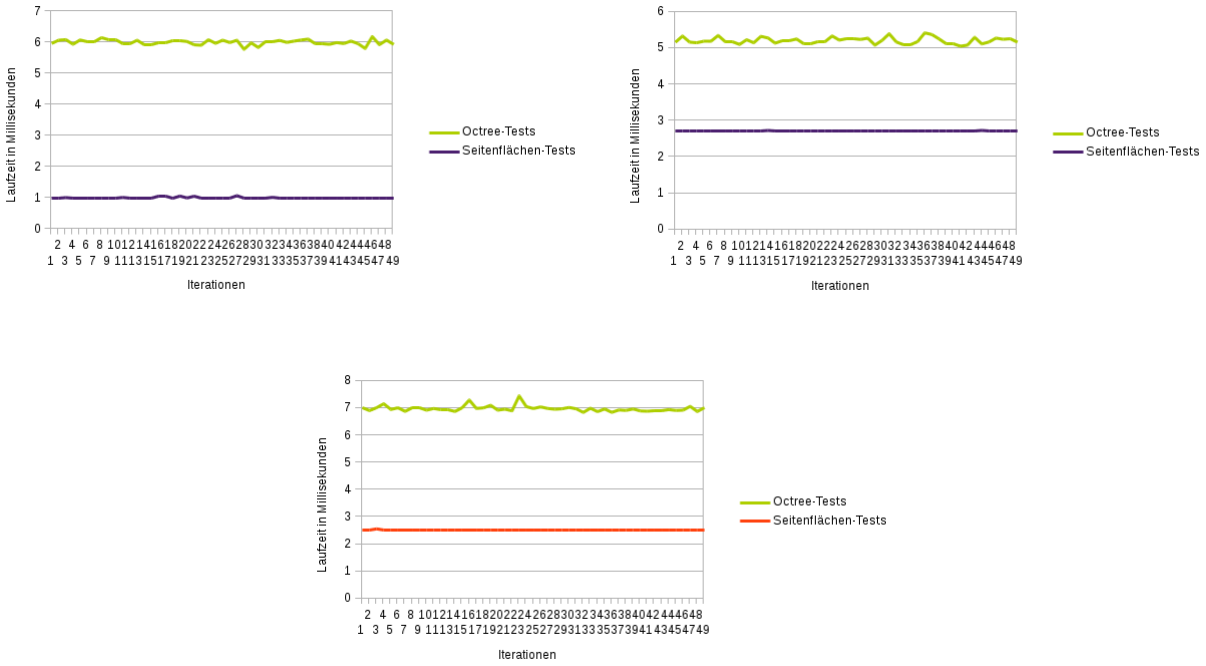


Abbildung 6.9.: Laufzeitverhalten des eigenen Verfahrens mit der Kollisionsgeometrie aus Abbildung 6.8 über 50 Iterationen hinweg

Hinsichtlich des Laufzeitverhaltens in diesem Experiment ist zu bemerken, dass sowohl die Objektpaar-Tests zwischen identischen Geometrien als auch der Objektpaar-Test zwischen zwei unterschiedlichen Kollisionsgeometrien ein vergleichbares Laufzeitverhalten aufweisen: Im Gegensatz zum vorherigen Experiment weisen die verwendeten Kollisionsgeometrien ei-

ne um einen nur geringen Faktor voneinander abweichende Anzahl an Seitenflächen und damit an Konglomeraten auf, was sich aufgrund des Parallelisierungs-Schemas des eigenen Verfahrens günstig auf die Anzahl insgesamt ausgeführter Kernel-Instanzen auswirkt.

6.5. Diskussion der Ergebnisse

Die in Abschnitt 6.4 vorgestellten Szenarien zeigen die Fähigkeit des eigenen Verfahrens, sowohl bei Kollisionsgeometrien unterschiedlicher Größe und Komplexität als auch unter verschiedenen Randbedingungen bei seiner Benutzung, grundsätzlich gut innerhalb der Grenzen der Anforderungen für interaktiven Betrieb mit Iterationszeiten unter 40 Millisekunden zu funktionieren.

Die octree-basierte Ermittlung potentiell kollidierender Paare von Seitenflächen, die im eigenen Verfahren in Kombination mit der Partitionierung von Kollisionsgeometrien in Konglomerate die paarweise Traversierung von Hüllkörper-Hierarchien ersetzt, erfüllt dabei in einem Objektpaartest die in Kapitel 1 und Kapitel 2 aufgestellten Anforderungen hinsichtlich des Laufzeitaufwandes.

In diesem Zusammenhang sollen nochmals die wichtigsten Besonderheiten des in Abschnitt 6.2 diskutierten Parallelisierungs-Schemas sowie der verwendeten Datenstrukturen für Konglomerate im eigenen Verfahren betont werden: Dieses erlaubt es im Gegensatz zu paarweisen Traversierung von Hüllkörper-Hierarchien, Paare von Konglomeraten verschiedener Objektpaare unabhängig voneinander auf mögliche Kollisionen zu überprüfen.

Ermöglicht wird das einerseits durch die Gestaltung der Datenstrukturen, die für die Repräsentation von Konglomeraten im GPU-basierten Teil des eigenen Verfahrens verwendet werden: Diese enthalten neben Octrees und Punkthüllen auch die Eckpunkt- und Index-Listen der Seitenflächen, die in einem Konglomerat enthalten sind. So kann sowohl die Ermittlung potentiell kollidierender Seitenflächen als auch die Überprüfung von Seitenflächen-Paaren zur Ermittlung von Kontaktpunkten durch eine einzige Kernel-Instanz bewerkstelligt werden. Bei der Verwendung von Hüllkörper-Hierarchien (wie bei gProximity in Unterabschnitt 4.3.1 und HPCCD in Unterabschnitt 4.3.2) ist dagegen eine Trennung der Vorfilter- und Paartest-Phase für Seitenflächen notwendig.

Die Separierung von Kollisionsgeometrien anhand disjunkter Konglomerate (wie in Abschnitt 5.2 diskutiert) erlaubt andererseits die Zusammenstellung von Arbeitseinheiten für die GPU-basierte Berechnung, die nicht notwendigerweise zu denselben Paaren von Kollisionsgeometrien gehören müssen, welche im Rahmen der szenenweiten Vorfilter-Phase als potentiell kollidierend detektiert worden sind. Die Restriktionen, die bei Hüllkörper-Hierarchien durch deren Struktur und die verwendete Traversierungs-Methode hinsichtlich Anzahl und Reihenfolge durchzuführender Paartests bestehen, sind bei der vom eigenen Verfahren verwendeten Organisation von Arbeitseinheiten daher nicht gegeben. Die Restrukturierung der Elemente von Kollisionstests in Form von Konglomeraten durch das eigene Verfahren erlaubt so potentiell die Zusammenfassung mehrerer Paartests in einer gemeinsamen Arbeitseinheit, was eine neue Möglichkeit zur Parallelisierung oberhalb der Ebene einzelner Objektpaar-Tests ermöglicht: Während gProximity beispielsweise für jeden Objektpaar-Test einen separaten Aufruf eines CUDA-Kernels benötigt, kann das eigene Verfahren zum Einen die Anzahl

von CPU-seitigen CUDA-Kernel-Aufrufen verringern, und zum Anderen potentiell mehrere Objektpaar-Tests gleichzeitig pro Arbeitseinheit absolvieren, abhängig von der Gesamtzahl zu überprüfender Konglomerat-Paare je Iteration einer Simulation.

Die Ausführung der abschließenden Paartests für Seitenflächen im eigenen Verfahren benötigt konsistent in allen Laufzeitmessungen aus Abschnitt 6.4 nur maximal ein Viertel der Laufzeit der octree-basierten Berechnungen, und ist damit keine kritische Einflussgröße auf die benötigte Gesamtlaufzeit des Verfahrens.

6.5.1. Einschränkungen bei der Umsetzung des eigenen Verfahrens

Bei der Implementierung des eigenen Verfahrens waren einige Faktoren zu berücksichtigen, die potentielle weitere Optimierungen nicht berücksichtigt haben, welche den Laufzeitaufwand des Verfahrens noch weiter hätten verringern können. Auf diese soll hier noch kurz eingegangen werden.

6.5.1.1. Zum Aufbau von Octree-Datenstrukturen und Punkt-Hüllen

Bei der Erstellung von Octrees auf Basis von Punkt-Hüllen wurden für die in Abschnitt 6.4 benutzten Objekte eine einheitliche Größe für die Kantenlänge von Blattknoten benutzt. Ebenso wurden die Kanten von Seitenflächen bei der Erstellung von Punkt-Hüllen von Konglomeraten eine konstante Anzahl von fünf gleichmäßig über die Länge einer Kante hinweg verteilten Punkten verwendet. Die Kombination dieser Faktoren hat dazu geführt, dass Seitenflächen beziehungsweise deren Kanten nicht ausschließlich von Blattknoten eines Octrees umhüllt werden. Das könnte bei ausschließlicher Suche nach Octree-Blattknoten bei der Einordnung von Punkten aus Punkt-Hüllen anderer Objekte dazu führen, dass potentielle Kontakte zwischen Seitenflächen nicht detektiert werden. Um diesem Problem zu begegnen, werden bei der Einordnung von Punkt-Hüllen nicht nur Blattknoten, sondern auch innere Knoten eines Octree innerhalb eines Radius, der einem Vielfachen der verwendeten Kantenlänge von Octree-Blattknoten entspricht, zur Indikation möglicher Kollisionen zwischen Paaren von Seitenflächen aus unterschiedlichen Konglomeraten verwendet.

6.5.1.2. Zur Zusammenstellung von Arbeitseinheiten

Die Zusammenstellung von Arbeitseinheiten, die im Rahmen der Versuche aus Abschnitt 6.4 verwendet worden ist, beschränkte sich jeweils auf die Berücksichtigung von Konglomeraten aus einzelnen Paartests. Das zur Partitionierung von Kollisionsgeometrien benutzte Verfahren (HACD, Unterabschnitt 5.2.1) hat für die in den Versuchen verwendeten Kollisionsgeometrien Konglomerate mit einer Größe zwischen 20 und 100 Seitenflächen erzeugt, was selbst bei dem in Unterabschnitt 6.4.3 verwendeten Modell eines Sechs-Achs-Roboters mit etwa 10000 Dreiecken nur eine dreistellige Gesamtzahl von Konglomeraten erzeugt hat. Daher wurde für die durchgeführten Experimente jeweils nur ein einzelne Arbeitseinheit je Ausführung der Kollisionserkennung benötigt, und eine weitere Unterteilung der Menge je Iteration zu überprüfender Konglomerat-Paare auf mehr als eine Arbeitseinheit wurde daher nicht angestoßen.

6.5.2. Zusätzliche Optimierungen des eigenen Verfahrens

Die Behebung der in Unterabschnitt 6.5.1 geschilderten Einschränkungen hinsichtlich der statischen Größe von Blattknoten in Octrees und der Einbeziehung innerer Knoten bei der Einordnung von Punkt-Hüllen sowie mögliche Maßnahmen zur Verringerung der Anzahl durchgeführter Einordnungs-Operationen von Punkten in Octrees sollen noch kurz skizziert werden.

6.5.2.1. Integration erweiterter Attribute für Octrees

Eine mögliche Erweiterung der verwendeten Octree-Datenstrukturen, vergleichbar mit der Klassifizierung von Voxeln im Voxmap-Pointshell-Algorithmus (Oberflächen- und oberflächen-nahe Voxel; [MPT06]), wäre in Form sogenannter Distanz-Felder (engl. Distance Maps, [TKH+04], [FSG03], [FPRJ00]) möglich.

Diese sind eine Ergänzung voxel-basierter Datenstrukturen um Angaben zum Abstand einer Zelle in einer Raumpartionierung um den Abstand zu Zellen, in denen Teile einer approximierten polygonalen Oberfläche enthalten sind. Die Integration eines Abstandswertes bei der Einordnung von Punkten in Octrees könnte als frühzeitiges Abbruchkriterium herangezogen werden. Die Verwendung von Morton-Schlüsseln zur Lokalisierung des Octree-Knotens, in dem ein eingeordneter Punkt enthalten ist, liefert diese Information aufgrund der Konstruktionsweise von Morton-Schlüsseln implizit als Teil von deren Adressierungs-Schema: Jedes Bit-Tripel adressiert eine Octree-Ebene; sofern ein Punkt in einem inneren Knoten zu liegen kommt, bestimmt die Anzahl von Bit-Tripeln ohne gesetztes Adress-Bit den Abstand eines Punktes zur Blattknoten-Ebene eines Octree.

6.5.2.2. Variation der Größe von Blattknoten

Eine weitere Möglichkeit zur besseren Adaption eines Octree an die von einem Konglomerat umschlossenen Seitenflächen besteht in der Variation der Kantenlänge von Blattknoten. Anstatt wie in der für die Laufzeitmessungen verwendeten einfachen Implementierung eine statische Kantenlänge zu verwenden, könnte diese beispielsweise abhängig von der Länge der AABB eines Konglomerats (da deren zugehörige Octrees zentriert im Koordinaten-Ursprung konstruiert werden) gewählt werden. Die Adaption der Dimensionen des Octree an das Längen- und Breitenverhältnis der erzeugenden Punkt-Hülle anhand eines solchen Kriteriums könnte dazu beitragen, die Tiefe von Octrees in Konglomeraten zu verringern, da weniger Blattknoten erzeugt werden, je größer die Kantenlänge eines Blattknotens gewählt wird, ohne dass gleichzeitig die erzeugende Punkt-Hülle verändert wird.

6.5.2.3. Verringerung der Anzahl von Punkt-Octree-Tests zwischen Paaren von Konglomeraten

Bei der Einordnung von Punkt-Hüllen in Octrees wurde für bei den Versuchen aus Abschnitt 6.4 grundsätzlich die komplette Punkt-Hülle eines Konglomerats expandiert. Kombiniert mit der Verwendung von Distanz-Feldern kann die komplette Expansion der Punkt-Hülle eines Konglomerats jedoch vermieden werden: So könnte etwa durch die sukzessive

Prüfung der Vertices und von Punkten auf den Kanten einer Seitenfläche ein Gradient der Abstandswerte bestimmt werden. Die Überprüfung von Punkten innerhalb einer Seitenfläche könnte etwa dann übersprungen werden, wenn keiner der zuvor überprüften Eck- oder Kanten-Punkte in einem Oberflächen- oder oberflächennahen Voxel liegen. Ebenso könnte die Expansion der Punkt-Hülle einer Seitenfläche übersprungen werden, wenn keiner der Eck- oder Kanten-Punkte einem oberflächennahen Voxel zugeordnet wird.

KAPITEL 7

MÖGLICHE ERWEITERUNGEN

Das folgende Kapitel diskutiert in kurzer Form mögliche Erweiterungen und Anpassungen der in Kapitel 5 und Kapitel 6 vorgestellten Kollisionserkennungs-Verfahren.

Die in Kapitel 4 behandelten Kollisionserkennungs-Verfahren sind im Gegensatz zur Implementierung des eigenen Verfahrens, wie sie Gegenstand von Kapitel 5 war, zum Umgang mit sich verformenden oder sogar zerbrechenden Objekten imstande. Abschnitt 7.1 skizziert die erforderlichen Erweiterungen, die für das eigene Verfahren notwendig wären, um Änderungen in Form und Topologie in Kollisionsgeometrien geeignet unterstützen zu können.

Ein weiterer Aspekt, der im Verlauf der vorhergehenden Kapitel weitgehend unberücksichtigt geblieben ist, betrifft die *dynamische Kollisionserkennung* unter Berücksichtigung der Eigenbewegung simulierter Objekte. In den meisten Implementierungen wird von konstanten Zeitschritten im Bereich weniger Millisekunden ausgegangen, zu denen jeweils eine statische Momentaufnahme der simulierten Szene auf Kollisionen überprüft wird. Um variable Zeitschritte mit größerer Dauer unterstützen zu können, und sich daraus ergebende Probleme wie etwa in Objektkonfigurationen in Abbildung 7.1 zu vermeiden, sind manche Kollisionserkennungs-Verfahren dahingehend erweitert, dass sie die Eigenbewegung von Objekten während der Kollisionserkennung berücksichtigen können, sowohl in der Vorfilter-Phase als auch auf der Ebene von Paar-Tests zwischen einzelnen Seitenflächen. Abschnitt 7.2 erläutert, wie das eigene Verfahren zur Unterstützung dynamischer Kollisionserkennung erweitert werden könnte.

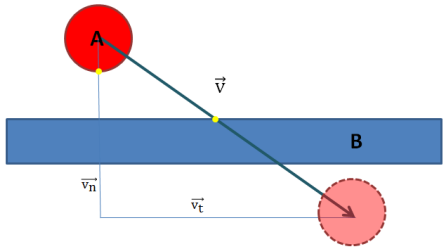


Abbildung 7.1.: Nicht registrierte Kollision bei Objektbewegung mit hoher Geschwindigkeit (nach [Lin11]): Ein Kollision eines sich schnell bewegendes Objektes mit einem anderen Objekt kann in der Kollisionserkennung übersehen werden, sofern die zwischen zwei Iterationen zurückgelegte Entfernung ausreicht, das erste Objekt vollständig jenseits des zweiten Objekts zu positionieren.

7.1. Elastische Objekte - verformbare Geometrien

Das in Kapitel 5 und Kapitel 6 diskutierte eigene Verfahren zur Kollisionserkennung hat sich im Rahmen der vorliegenden Arbeit bisher auf die Unterstützung starrer Körper beschränkt. Diese vereinfachende Annahme ist vorteilhaft für eine effiziente Implementierung, da sich die Manipulation der verwendeten Datenstrukturen auf eine Vorverarbeitung zur Durchführung der Partitionierung von Geometrien sowie die Berechnung von Raumpartitionierungen und Punkthüllen beschränkt. Andererseits bedeutet diese Annahme aber eine große Einschränkung des Funktionsumfangs, vor allem im Vergleich zu Verfahren wie gProximity oder HPCCD, die auch zur Simulation komplexen Verformungsverhaltens oder sogar zerbrechender Objekte in der Lage sind.

Der Umgang mit plastisch oder elastisch verformbaren beziehungsweise mit zerbrechenden Objekten erfordert, die zur Kollisionserkennung genutzten Datenstrukturen während der Laufzeit der Simulation an neue geometrische oder topologische Gegebenheiten der zugrunde liegenden polygonalen Geometrien anpassen zu können. Hier sind zwei Szenarien zu unterscheiden:

- Plastische oder elastische Verformung: Die Koordinaten von Eckpunkten einer Geometrie ändern sich, aber die topologische Struktur, also Anzahl und Konnektivität der einzelnen Seitenflächen, bleibt unverändert.
- Mechanische Änderungen mit Auswirkung auf die topologische Struktur (Zerbrechen, Zerschneiden): Hierbei ändert sich zusätzlich die topologische Struktur der Geometrie.

7.1.1. Modifikationen der Vorfilter-Phase

Bei polygon-basierten Verfahren implizieren diese Veränderungen an Geometrien die Notwendigkeit der Neuberechnung zugehöriger Hüllkörper-Hierarchien. Generell fällt in diesem Fall die Wahl der verwendeten Hüllkörper auf einfachere geometrische Strukturen, die sich aufgrund ihrer Konstruktionsverfahren sehr schnell neu berechnen lassen, oder die ohnehin nach Lageveränderungen der umschlossenen Geometrie neu berechnet werden müssen: Achsen- oder objekt-orientierte Hüllquader (AABBs beziehungsweise OBBs).

Für das eigene Verfahren wäre bezüglich der verwendeten Hüllkörper auf der Ebene einzelner Konglomerate eine Neuberechnung in analoger Weise notwendig. Allerdings wäre diese bedingt durch den Verzicht auf tief verschachtelte Hierarchien, abgesehen von einem alle Konglomerate umschließenden Hüllkörper, deutlich einfacher und damit mit weniger Laufzeitaufwand verbunden als eine Rekonstruktion von Hüllkörper-Hierarchien basierend auf Top-down, Bottom-up- oder Insertion-Methoden (Unterabschnitt 3.4.3), obwohl diese Problematik für Hüllkörper-Hierarchien etwa durch die nur teilweise Neuberechnung von Hierarchien umgangen werden kann.

7.1.2. Modifikationen der Objektpaar-Phase

Zur Adaption der pro Konglomerat verwendeten Raumpartitionierungen an geometrische Veränderungen wäre die erneute Anwendung der in Unterabschnitt 5.5.2 und Unterab-

schnitt 5.6.1 beschriebenen Konstruktionsverfahren für Octrees und k-d-Bäume bei plastischen oder elastischen Verformungen denkbar. Die erneute GPU-basiert implementierte Berechnung dieser Raumpartitionierungen ist sehr laufzeiteffizient, und könnte über die gezielte Rekonstruktion von Teilbäumen (ähnlich wie dies für Hüllkörper-Hierarchien möglich ist) noch weiter beschleunigt werden.

Komplexer in der Umsetzung wäre dagegen die Unterstützung von Veränderungen der topologischen Struktur: Hier kann nicht mehr davon ausgegangen werden, dass nach einer solchen Veränderung eine Zusammenfassung eines Verbunds aus Seitenflächen noch sinnvoll beibehalten werden kann. Beispielsweise kann im Fall eines zerbrechenden Objekts die räumliche Nähe der in einem Konglomerat zusammengefassten Seitenflächen nicht mehr gegeben sein, was vor allem für die Struktur einer auf Grundlage eines in dieser Art „degenerierten“ Konglomerats berechneten Raumpartitionierungen negative Auswirkungen hätte. Um topologische Veränderungen trotzdem unterstützen zu können, würde sich eine dynamische Erzeugung neuer Konglomerate etwa an Hand der Konnektivität von Seitenflächen in Gestalt gemeinsamer Eckpunkte und Kanten anbieten: Nach der Separation eines ursprünglichen Konglomerats können aus den neu entstandenen Verbünden von Seitenflächen neue Konglomerate zur Laufzeit erstellt werden, die das ursprüngliche Konglomerat ersetzen. Diese Vorgehensweise wäre möglich, da für die Partitionierung einer Geometrie in Untereinheiten durch das eigene Verfahren keine besonderen Voraussetzungen, beispielsweise in Gestalt einer Beschränkung auf konvexe Geometrien wie im GJK-Algorithmus, gemacht werden. Zwar wäre eine solche Aufteilung in neue Konglomerate beispielsweise ohne Berücksichtigung des Verhältnisses der Anzahl erfasster Seitenflächen pro Konglomerat nicht optimal, jedoch trotz dieser Nachteile eine gangbare Variante zur Nachrüstung dieser Funktionalität für das eigene Verfahren.

Um die Punkthülle eines Konglomerats nach Veränderungen der Struktur oder Topologie der ursprünglichen Geometrie zu aktualisieren, bietet der gewählte Ansatz bei der Konstruktion von Punkthüllen einen wesentlichen Vorteil: Sowohl bei Änderungen der Koordinaten von Eckpunkten als auch topologischen Veränderungen kann die Punkthülle pro Seitenfläche anhand des in Unterabschnitt 5.3.1 vorgestellten Verfahrens neu erzeugt und die zugehörige Bit-Maske betroffener Seitenflächen aktualisiert werden. Diese Aktualisierung könnte einfach als zusätzlicher Schritt in die Neuberechnung von Konglomeraten integriert werden.

7.2. Dynamische Kollisionserkennung

Die Erweiterung des eigenen Verfahrens um die Berücksichtigung von Objektbewegungen als Teil der eigentlichen Kollisionserkennung würde Modifikationen an zwei Stellen erfordern, um als Teil der Vorfilter-Phase die Bewegungen von Konglomeraten berücksichtigen zu können, und als Teil der Tests von Seitenflächen-Paaren die Bewegung individueller Dreiecks-Paare.

7.2.1. Modifikationen der Vorfilter-Phase

Als Teil der Vorfilter-Phase ist die Verwendung von extrudierten Hüllvolumina, die entlang der Trajektorie eines Objekts eine konservative Abschätzung der während einer Iteration erfolgenden Eigenbewegung anhand des von diesem überstrichenen Volumens vornehmen, mög-

lich. Gleiches gilt für den zweiten Schritt der CPU-basierten Überprüfung von konglomeratspezifischen Hüllvolumina. Eine Variante hierzu wäre der Einsatz der in Unterabschnitt 4.3.1 kurz erwähnten Rectangle Swept Spheres (RSS). Günstig würde sich hier wiederum die Eigenschaft der Hüllkörper-Strukturen des eigenen Verfahrens auswirken, die mit wesentlich weniger Hüllvolumina operieren, als es konventionelle Hüllkörper-Hierarchien tun.

7.2.2. Modifikationen der Objektpaar-Phase

Die Integration in die paarweise Überprüfung von Konglomeraten könnte sich eines Interpolations-Schemas als Teil der Einordnung individueller Punkte aus der jeweiligen Punkthülle eines Konglomerats als zusätzlicher Teilschritt bedienen: Ausgehend von der Trajektorie eines Objekts könnte die Position jedes Punktes nicht nur einmalig zur Einordnung verwendet werden, sondern in mehreren Iterationen jeweils nach Aktualisierung anhand der extrapolierten Bewegungsrichtung erneut in die Raumpartitionierung des anderen Konglomerats in einem Paar-Test eingeordnet werden. Dieser Ansatz würde sich nahtlos in die zuvor beschriebene Funktionsweise des statischen Kollisions-Test einfügen, da außer der Extrapolation der Trajektorie und der mehrfachen Ausführung der Einordnung eines Punkts während eines Simulationsschritts keine weiteren Modifikationen an der grundlegenden Methode des statischen Anwendungsfalls notwendig wären.

Für die Überprüfung von Seitenflächen-Paaren könnte wiederum eine Erweiterung der Methode für die statische Überprüfung unverändert übernommen werden, da das eigene Verfahren an dieser Stelle ausschließlich auf die polygonale Geometriebeschreibung eines Objekts zurückgreift, und deswegen keine zusätzlichen Erweiterungen vorgenommen werden müssten.

KAPITEL 8

ZUSAMMENFASSUNG

Im Zuge der Diskussion praxisnaher Anwendungen von echtzeitfähigen Starrkörpersimulationen im Bereich der Robotik in Kapitel 2 wurde die Kollisionserkennung zwischen komplex strukturierten Geometrien als derjenige Teilbereich identifiziert, der aufgrund seiner zentralen Bedeutung im Gesamtkontext eines solchen Simulationssystems großen Einfluss auf dessen Funktionsumfang und Nützlichkeit für unterschiedliche Aufgabenstellungen ausübt. Die Laufzeit-Effizienz einer Kollisionserkennungs-Komponente beeinflusst dabei in einer Starrkörpersimulation als wichtigste Einflussfaktoren:

1. Die *geometrische Präzision*: Je komplexer Geometrien strukturiert sind, umso mehr Laufzeitaufwand wird für die Suche nach möglichen Kontakten zwischen Objekten benötigt. Angesichts der gestellten Anforderung, eine Simulation in oder zumindest nahe Echtzeit betreiben zu können, sind mögliche Alternativen entweder die Verwendung einfach strukturierter Kollisionsgeometrien, oder die Optimierung der verwendeten Algorithmen.
2. Die *mechanische Plausibilität*: Diese ist eng mit der geometrischen Präzision verwendeter Kollisionsgeometrien verbunden. Unterscheiden sich etwa die zur Visualisierung und zur Kollisionserkennung verwendeten Modelle in einer simulierten Umgebung voneinander, so beeinträchtigen diese Abweichungen die Realitätsnähe einer Simulation erheblich, indem die fundamentale Eigenschaft realer massiver Objekte, sich nicht gegenseitig durchdringen zu können, buchstäblich sichtbar verletzt wird. Obwohl solche Abweichungen für bestimmte Anwendungsbereiche noch hinnehmbar sein mögen, so sind sie doch für Simulationsanwendungen wie die im Rahmen der diskutierten Projektstudie geplante Nutzung eines virtuellen Prototyps zur Unterstützung der Entwicklung von Steuerungsprogrammen für die spätere Übertragung auf reale Roboter keinesfalls akzeptabel: Würde ein Umgebungsmodell mit von der späteren realen Arbeitsumgebung abweichender geometrischer Struktur für diese Aufgabe verwendet werden, so wäre ein mit Hilfe eines virtuellen Modells entwickeltes Steuerungsprogramm nutzlos, da seine Funktionsfähigkeit von Gegebenheiten abhängen würde, die in der realen Entsprechung einer Arbeitsumgebung nicht vorhanden sind.

3. Die *mechanische Stabilität*: Diese manifestiert sich in der erfolgreichen Vermeidung des Auftretens mechanischen Verhaltens in einer Simulation, das etwa durch numerische Instabilitäten im Lösungsverfahren der Kollisionsbehandlung oder das Übersehen vorhandener Objektberührungen oder -überschneidungen durch die Kollisionserkennung auftreten kann. Ein Beispiel für eine solche Situation wurde in Unterabschnitt 3.7.1 diskutiert: Hier ist es der verwendeten Starrkörpersimulation nicht gelungen, eine stabile Ruhelage eines Objektpaars in einer gegenseitig umschließenden Kontaktsituation herzustellen. Verursacht wurde dies dadurch, dass die Kollisionserkennungs-Komponente der Starrkörpersimulation nicht in der Lage war, eine Menge von Kontaktpunkten zu detektieren, die die vorliegende Kontaktsituation korrekt hätte wiedergeben können.

Aufgrund dieser Beobachtungen ist die wichtigste Problemstellung bei den vorgestellten Projekten (vor allem im Fall der Projektstudie zur virtuell unterstützten Entwicklung und Erprobung von Steuerungsprogrammen für Industrieroboter, Abschnitt 2.1) die Präzision bei der Modellierung von Kollisionsgeometrien, die zur Bestimmung möglicher Berührungen oder Überschneidung zwischen Objekten in einer Simulation verwendet werden. Verbindet man diese Schlussfolgerung mit der Notwendigkeit, eine Simulation innerhalb enger Laufzeitgrenzen betreiben zu können, so ergibt sich als Konsequenz die dem weiteren Verlauf der vorgelegten Arbeit zugrundeliegende Motivation zur Entwicklung eines Kollisionserkennungs-Verfahrens, das im Unterschied zu ebenfalls analysierten existierenden Verfahren (Kapitel 4) besondere Anforderungen, die sich im Kontext von Aufgabenstellungen aus der Robotik ergeben, berücksichtigt:

1. Die Unterstützung von simulierten Umgebungen mit einer moderaten Anzahl detailliert modellierter Objekte: Die in Kapitel 2 sowie Kapitel 3 diskutierten Starrkörpersimulationen nutzen Kollisionserkennungs-Verfahren, die für die optimale Unterstützung einer großen Menge einfach strukturierter Kollisionsgeometrien konzipiert sind. Komplex strukturierte polygonale Geometrien werden von diesen Software-Paketen nur eingeschränkt unterstützt; vor allem der Laufzeitbedarf (Unterabschnitt 4.2.3) ist in Bezug auf die Eignung zum Betrieb in interaktiver Laufzeit problematisch. Die in Unterabschnitt 4.3.1 bis Unterabschnitt 4.3.3 vorgestellten polygon-basierten Verfahren, die alle für die Nutzung massiv paralleler Prozessorarchitekturen (GPUs) ausgelegt sind, besitzen die Fähigkeit zur Verarbeitung großer polygonaler Objekte innerhalb weniger Millisekunden. Allerdings sind die Beispiel-Szenen, anhand derer die benötigten Laufzeiten durch die jeweiligen Autoren ermittelt worden sind, größtenteils nur aus einem einzigen Paar von Objekten zusammengesetzt. Das im Rahmen der vorliegenden Arbeit implementierte Verfahren versucht dagegen, einen möglichst ausgewogenen Kompromiss zwischen diesen beiden Arten von Anwendungsfällen zu etablieren.
2. Spezifische Berücksichtigung von Szenen mit mehreren bewegten Objekten: Neben polygon-basierten Verfahren existiert eine weitere Art von Kollisionserkennungs-Verfahren, die auf der Grundlage von Raumpartitionierungen und Punkthüllen operieren. Als Vertreter dieser Art voxel-basierter Verfahren wurde der Voxmap-Pointshell-Algorithmus in Unterabschnitt 4.4.3 genauer analysiert. Der Vorteil solcher Verfahren liegt in der effizienten Natur der elementaren Kollisions-Tests, die gegenüber der Vorgehensweise polygon-basierter Verfahren weitaus weniger Laufzeitaufwand erfordern. Diese Eigenschaft ermöglicht ebenfalls die Simulation komplex strukturierter Arbeitsumgebungen in Zeitintervallen im einstelligen Millisekunden-Bereich. Allerdings besteht in

Bezug auf die Struktur dieser Umgebungen ein gravierender Nachteil: Der Großteil der Arbeitsumgebung wird als statisch angenommen und in einer einzigen Datenstruktur zusammengefasst, während nur eines oder wenige von Anwendern kontrollierte Objekte tatsächlich aktiv in der Szene bewegt werden können. Das eigene Verfahren versucht, die effizienten Laufzeiteigenschaften dieser Verfahren mit der Unterstützung mehrerer, auch unabhängig von interaktiver Manipulation bewegter Objekte zu kombinieren.

3. Eignung zum Betrieb innerhalb interaktiver Laufzeitgrenzen unter den beiden zuvor genannten Voraussetzungen: Das Konzept des eigenen Verfahrens versucht, sowohl die nötige Laufzeit zur Durchführung der Kollisionserkennung selbst als auch den Aufwand für die Koordination der einzelnen Schritte des Verfahrens (wie den Datentransfer von und zum GPU-Prozessor) so weit wie möglich zu minimieren. Die Effizienz des Verfahrens selbst soll wird durch die Verwendung von punkthüllen-basierten Operationen im Rahmen von Objektpaar-Tests gewährleistet werden: Diese ersetzen die von anderen Verfahren genutzte paarweise Traversierung von Hüllkörper-Hierarchien, die bei den in Abschnitt 3.6 und Abschnitt 4.2 behandelten Implementierungen als der Teilschritt identifiziert werden konnte, der gemessen an der Gesamtlaufzeit einer Iteration den größten Laufzeitanteil verbraucht, und zudem bei der Portierung auf GPU-Prozessoren eine aufwendige Warteschlangenverwaltung erfordert. Die Notwendigkeit einer solchen Warteschlangenverwaltung wird im eigenen Verfahren dadurch umgangen, dass ein Objektpaar-Test durch die Separierung von Geometrien in Konglomerate auf der Basis von Paaren dieser Konglomerate durchgeführt werden kann. Zwischen einzelnen Instanzen von Konglomerat-Paartests bestehen, anders als bei der paarweisen Traversierung von Hüllkörper-Hierarchien, keine wechselseitigen Abhängigkeiten während der Laufzeit.
4. Vermeidung expliziter Annahmen oder Einschränkungen hinsichtlich der Struktur und Eigenschaften verwendeter Kollisionsgeometrien: Ein weiterer Vorteil des eigenen Verfahrens ergibt sich durch die Konstruktionsweise der verwendeten Partitionierung von Eingangs-Geometrien auf der Basis von Konglomeraten. Das erlaubt die Benutzung des Verfahrens ohne die Notwendigkeit für Anwender, manuell Kollisionsgeometrien konstruieren zu müssen. Wichtiger noch ist die Tatsache, dass diese Vorgehensweise ohne eine Abweichung zwischen Kollisionsgeometrien und den ursprünglichen Modellen auskommt: Das erlaubt schließlich die Erfüllung einer der wichtigsten Anforderungen an Kollisionserkennungs-Algorithmen, die unter anderem in Abschnitt 3.1 formuliert worden war.

Die in Kapitel 6 durchgeführten Experimente zur Erprobung der Implementierung des eigenen Verfahrens haben gezeigt, dass die gewählte Vorgehensweise des Verfahrens, die paarweise Traversierung von Hüllkörper-Hierarchien in Objektpaar-Tests durch eine Kombination aus flexibler Restrukturierung von Kollisionsgeometrien in Form von Konglomeraten und der Verwendung von Raumpartitionierungen (Octrees) und Punkt-Hüllen zur Lokalisierung potentiell kollidierender Paare aus Seitenflächen zu ersetzen, die Vorteile massiv paralleler Prozessor-Architekturen sehr gut nutzen kann:

1. Der Verzicht auf tief geschachtelte Hüllkörper-Hierarchien und deren Substitution durch Konglomerate als disjunkte Teilstrukturen von Kollisionsgeometrien erlaubt potentiell die simultane Ausführung mehrerer Objektpaar-Tests.

2. Da in einem Konglomerat die nötigen Datenstrukturen für die Ermittlung potentiell kollidierender Seitenflächen (Octrees und Punkt-Hüllen) als auch die Eckpunkt- und Index-Daten der in einem Konglomerat enthaltenen Seitenflächen zusammegefasst sind, können beide Schritte des eigenen Verfahrens im selben Kernel zusammengefasst werden. Damit sind keine separaten Maßnahmen zur Sicherstellung einer gleichmäßigen Nutzung parallel laufender Berechnungseinheiten nötig, wie sie bei auf Hüllkörper-Hierarchien basierenden Verfahren (beispielsweise gProximity oder HPCCD, Abschnitt 4.2) notwendig sind, um diese im Lauf der paarweisen Traversierung von Hüllkörper-Hierarchien gewährleisten zu können.
3. Obwohl sich die im Rahmen dieser Arbeit vorgestellten Laufzeitmessungen auf die Ebene individueller Objektpaar-Tests beschränkt haben, erlaubt die Funktionsweise des eigenen Verfahrens hinsichtlich seiner Warteschlangenverwaltung (sowohl als Teil der Konfiguration als auch als Teil der Ausführung GPU-basierter Berechnungen) eine flexible Zusammenstellung von Arbeitseinheiten abhängig von der Leistungsfähigkeit als auch der Anzahl verfügbarer GPU-Prozessoren, und ist damit besser in der Lage als auf Hüllkörper-Hierarchien basierende Verfahren, die jeweils zur Verfügung stehende GPU-Hardware optimal auszunutzen.

Als Resultate dieser Arbeit sind schließlich folgende Kernaussagen festzuhalten:

1. Die Untersuchung des Einsatzes von echtzeitfähigen Starrkörpersimulationen im Kontext von Problemstellungen in der Industrie- und Service-Robotik, und Herleitung der Anforderungen an entsprechende Software-Lösungen in Hinblick auf Robotik-Anwendungen.
2. Die Analyse existierender Kollisionserkennungs-Verfahren, die sowohl im Rahmen der vorgestellten Praxisanwendungen eingesetzt wurden, als auch GPU-basierender Implementierungen, die als Resultate aktueller Forschungs- und Entwicklungsarbeiten im Bereich der Kollisionserkennung entstanden sind.
3. Der Entwurf und die Implementierung eines GPU-basierten Kollisionserkennungs-Verfahrens, das einen hybriden Ansatz aus polygon-basierten Geometriebeschreibungen und voxel-basierten Verfahren verwendet und damit die Vorteile beider Verfahrensklassen in einer Weise vereint, die sich gut für den Einsatz auf massiv parallelen Prozessorarchitekturen eignet.

ANHANG A

MECHANIKSIMULATIONEN

A.1. Simulationstechnologien

Um die unterschiedlichen Ansätze für Mechaniksimulationen gegeneinander abzugrenzen, ist eine präzisere Begriffsbestimmung hilfreich. Dies betrifft unter anderem die Unterscheidung verschiedener technologischer Ansätze anhand des Laufzeitverhaltens und der Struktur der jeweils zugrunde liegenden Problemstellungen.

Gegenstand dieser Arbeit ist die Kollisionserkennung für echtzeitfähige Starrkörpersimulationen, bei denen die plausible Abbildung des mechanischen Verhaltens von Objekten im dreidimensionalen Raum im Vordergrund steht. Im Gegensatz dazu stehen Simulationsanwendungen, die mit wesentlich detaillierteren Modellen mechanischen Verhaltens arbeiten, und bei denen der Laufzeitbedarf einer Simulation im Vergleich zur Genauigkeit der Modellierung nur eine untergeordnete Rolle spielt.

A.1.1. Mehrkörpersystem (MKS)

Woehrnle [Woe11] definiert ein Mehrkörpersystem als Verbund

> aus massebehafteten starren Körpern, deren Bewegungen durch Bindungen geometrisch beschränkt sind und auf die verteilte und diskrete Kräfte und Momente einwirken.

Ein schematisches Beispiel eines Mehrkörpersystems zeigt Abbildung A.1.

Da eine analytische Lösung für das Verhalten von mehreren Objekten unter gegenseitigem Kontakt im Allgemeinen nicht in geschlossener Form zu berechnen ist, steht bei Mehrkörpersystemen die Lösung von Problemen aus der klassischen Mechanik mit Methoden der Numerik und der Verwendung von zeitdiskreten Simulationsmethoden im Mittelpunkt. Anders als bei echtzeitfähigen Starrkörpersimulationen spielt in Mehrkörpersystemen die Berechenbarkeit einer Lösung für das jeweils vorliegende Mehrkörpersystem in Realzeit keine

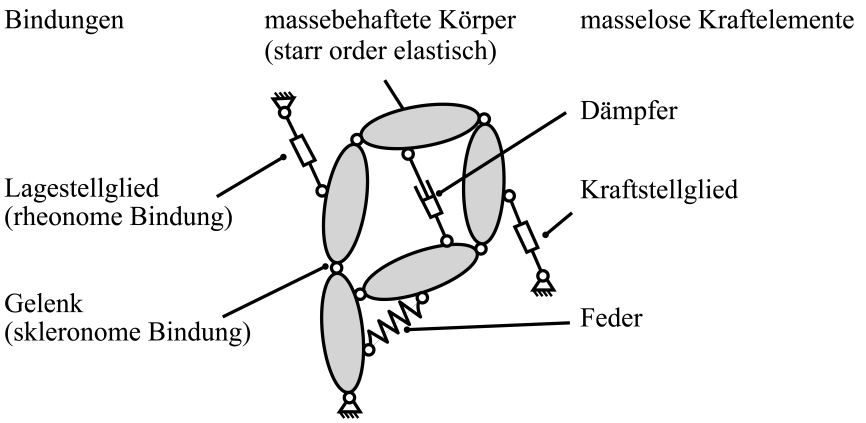


Abbildung A.1.: Ein Mehrkörpersystem (Beispiel nach [Woe11])

Rolle. Des weiteren steht auch nicht die detaillierte Modellierung von Kontakten zwischen Geometrien in den modellierten Systemen im Vordergrund, sondern vielmehr das kinematische beziehungsweise dynamische Verhalten des Gesamtsystems. Abbildung A.2 zeigt ein Mehrkörpermodell eines Roboterarmes, das mit dem MKS-Programm SIMPACK ([Sim13]) erstellt worden ist. Dieses Modell verzichtet auf die Verwendung des geometrischen Aufbaus des zugrunde liegenden Roboterarmes, und abstrahiert dessen Kinematik stattdessen über entsprechend dimensionierte Achsen und Rotationsfreiheitsgrade.

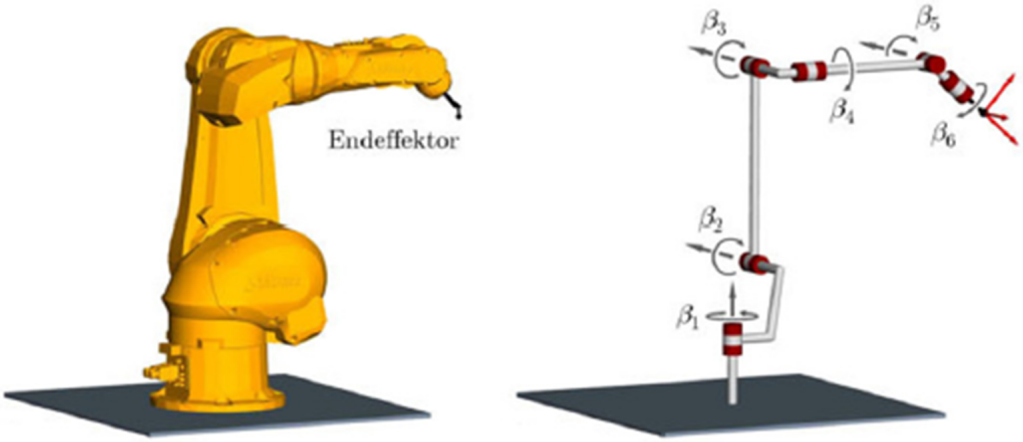


Abbildung A.2.: Mehrkörpermodell eines Roboterarms (aus [Woe11])

A.1.2. Finite-Elemente-Methode (FEM)

Eine andere Spezialisierung im Bereich mechanischer Simulationen stellt die *Finite-Elemente-Methode (FEM)* dar. Generell handelt es sich bei der FEM um ein numerisches Verfahren zur Lösung partieller Differentialgleichungen. Die Anwendbarkeit von FE-Methoden beschränkt

sich nicht allein auf mechanische Problemstellungen, sondern ist ebenso für die Simulationen in anderen Bereichen wie etwa der Thermodynamik, der Strömungslehre oder der Elektrotechnik anwendbar.

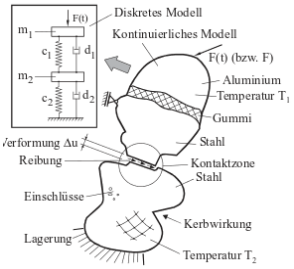


Abbildung A.3.: Abstraktion realer Gegebenheiten in mechanischen Simulationen (nach [Kle12])

Klein ([Kle12]) definiert die FEM nach Clough ([Clo60]) als

> Modellvorstellung eines Kontinuums als eine Zusammensetzung von Teilbereichen (finiten Elementen). In jedem Teilbereich wird das Elementverhalten durch einen Satz von Ansatzfunktionen beschrieben, die die Verschiebungen, Dehnungen und Spannungen in diesem Teilbereich wiedergeben.

Ein wesentliches Ziel der FEM ist es, eine das jeweilige Problem beschreibende Differentialgleichung in ein lineares Gleichungssystem zu überführen. Die zugrundeliegende Problemdomäne (etwa ein geometrisches Modell) muss vor der Berechnung in ein finites Modell überführt werden, für dessen einzelne Unterteilungen (Stab-, Scheiben-, Schalen-, oder Volumen-Elemente) die eigentliche numerische Berechnung durchgeführt wird.

In Abbildung A.3 ist das generelle Prinzip der Abstraktion von realen Gegebenheiten bei der Modellerstellung für Mechaniksimulationen angedeutet: Materialeigenschaften und -Besonderheiten wie Dichte-Variationen, Verformungsarbeit, Reibungsverhalten oder thermische Eigenschaften werden entweder nicht berücksichtigt, oder werden innerhalb eines simulierten Objektes subsumiert: Im gezeigten Beispiel wird beispielsweise für einen aus mehreren Materialien (Stahl, Gummi, Aluminium) bestehenden Körper eine einheitliche Dichte- und damit Massenverteilung angenommen. Für das zweite Objekt werden etwa die durch Einschlüsse lokal veränderten Materialeigenschaften ebenfalls nicht im diskreten Modell berücksichtigt.

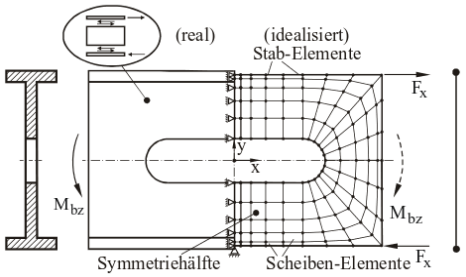


Abbildung A.4.: Vereinfachte dargestellte Erstellung eines FEM-Modells (nach [Kle12])

Abbildung A.4 illustriert die Erstellung eines FEM-Netzmodells am Beispiel eines Doppel-T-Trägers mit einer Aussparung in Längsrichtung: Angedeutet ist eine Zerlegung in Stab-

beziehungsweise Schalenelemente unter Ausnutzung einer Bauteil-Symmetrie. Die Berechnung anliegender Momentenbelastungen erfolgt in diesem Beispiel auf einem verkleinerten FEM-Modell.

Bei der FEM steht die analytisch hoch präzise Simulation mechanischer Vorgänge im Vordergrund, wobei die verwendeten geometrischen Modelle nicht nur als polygonale Oberflächenbeschreibungen, sondern als polyhedrale Volumina spezifiziert sind. So ist einerseits die Anzahl verwendeter geometrischer Primitive je modelliertem Objekt wesentlich höher, andererseits ist jedoch die zur Verfügung stehende Laufzeit nicht an die strengen Beschränkungen interaktiver Anwendungen gebunden.

A.2. Grundlagen der Kollisionsbehandlung

Als letzter Bestandteil einer Starrkörpersimulation soll zum Abschluss dieses Abschnitts noch ein Überblick über Verfahren zur Kollisionsbehandlung gegeben werden.

A.2.1. Die Problemstellung

Die Bewegung eines einzelnen Starrkörpers lässt sich in der Schreibweise der Newton-Euler-Gleichungen (oder *Lagrange-Formulierung*) als:

$$\frac{\partial}{\partial t}\begin{pmatrix}\vec{r}\\ \mathbf{q}\\ \vec{v}\\ \omega\end{pmatrix} = \begin{pmatrix}\vec{v}\\ \frac{1}{2}\omega\mathbf{q}\\ \vec{a}\\ \alpha\end{pmatrix} \tag{A.1}$$

ausdrücken.

Hierbei ist:

- $\vec{r}$ die Position des Schwerpunktes
- $\mathbf{q}$ Die räumliche Orientierung als Quaternion
- $\vec{v}$ die translative Geschwindigkeit
- ω die Winkelgeschwindigkeit
- $\vec{a}$ die translative Beschleunigung
- α die Winkelbeschleunigung

Dabei ergeben sich $\vec{a}$ und α zu:

$$\vec{a} = \frac{\mathbf{F}}{m} \tag{A.2}$$

$$\alpha = \mathbf{I}^{-1}(\tau + \omega \times \mathbf{I}\omega) \tag{A.3}$$

mit der Masse m, dem Trägheitstensor $\mathbf{I}$, der Summe der auf den Starrkörper einwirkenden Kräfte $\mathbf{F}$ und dem Drehmoment τ

Alternativ dazu ist die *Hamilton'sche Formulierung* möglich als:

$$\frac{\partial}{\partial t}\begin{pmatrix}\vec{r}\\ \mathbf{q}\\ \mathbf{P}\\ \mathbf{L}\end{pmatrix} = \begin{pmatrix}\vec{v}\\ \frac{1}{2}\omega\mathbf{q}\\ \mathbf{F}\\ \tau\end{pmatrix} \tag{A.4}$$

Hier ist $\mathbf{P}$ der Impuls und $\mathbf{L}$ der Drehimpuls, der auf den Starrkörper einwirkt. $\vec{v}$ und ω ergeben sich zu:

$$\vec{v} = \frac{\mathbf{P}}{m} \tag{A.5}$$

$$\omega = \mathbf{I}^{-1}\mathbf{L} \tag{A.6}$$

Die Verfahren zur numerischen Integration der Bewegung von Starrkörpern, die im Folgenden kurz vorgestellt werden, lassen sich in drei Kategorien einteilen:

1. Die sogenannte Penalty-Methode, bei der die Auflösung von Durchdringungen zwischen Körpern durch ein Feder-Masse-System modelliert wird.
2. Impuls-basierte Methoden: Hier wird jede Art von Interaktionen zwischen Körpern durch Mikrokollisionen modelliert.
3. Zwangsbedingungs-basierte Methoden: Diese Methoden berechnen explizit Kontakt- und Reaktionskräfte zwischen Körpern durch Lösen der zugrunde liegenden Bewegungsgleichungen.

A.2.2. Die Penalty-Methode

Zur Umsetzung von Kontaktkräften werden bei der Penalty-Methode ([MW88]) Federn-Analogien mit Ruhelagen-Länge Null an Durchdringungen zwischen Objekten in der Simulation verwendet (Abbildung A.5). Dabei steigt die erzeugte Federkraft proportional zur Eindringtiefe an. Die auf einen Körper einwirkenden Kräfte und Momente ergeben sich dadurch zu

$$\mathbf{F} = f_{ext} + \sum_{i=1}^{n}(f^i_{feder})$$

$$\tau = \tau_{ext} + \sum_{i=1}^{n}(\tau^i_{feder})$$

Die Berechnung der Kontakt-Kräfte f_{feder} kann beispielsweise über ein Ersatzmodell als kritisch gedämpfter harmonischer Oszillator erfolgen, um unerwünschtes Schwingungs-Verhalten, das bei dem angesprochenen Feder-Modell normalerweise auftreten würde, weitgehend zu unterdrücken.

Der Vorteil der Penalty-Methode liegt in deren Verständlichkeit und einfachen Umsetzbarkeit. Sie ist vor allem bei der Simulation elastischer Körper, wie etwa dem Verformungsverhalten von textilen Stoffen, ein häufig verwendetes Modell der Kollisionsbehandlung.

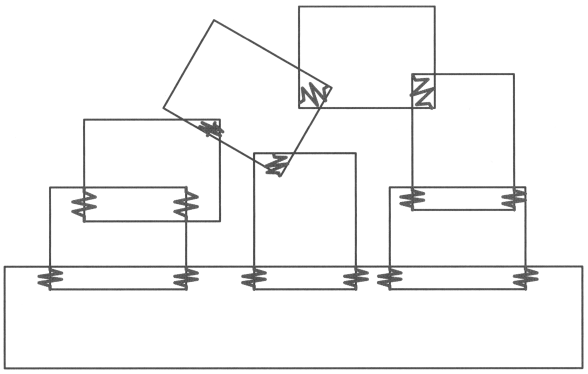


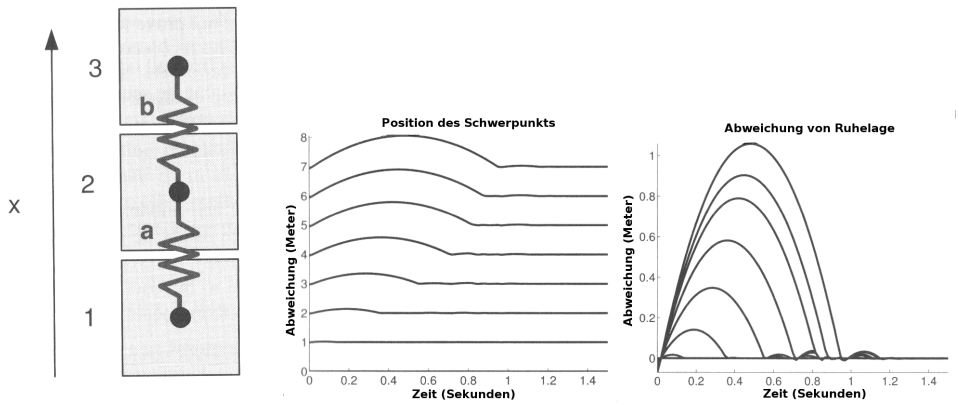
Abbildung A.5.: Das Wirkprinzip der Penalty-Methode in der Kollisionsbehandlung ([Erl05, Kapitel 5]): Federn-Analogien werden an Kontaktpunkten angesetzt, solange eine Durchdringung von Objekten erkannt wird.

Ein wesentlicher Nachteil der Penalty-Methode liegt in der Natur der Modellierung eines Kontaktes als harmonischer Oszillator begründet. Zwar ergeben sich ähnliche Probleme auch für die in den folgenden Abschnitten angesprochenen Kollisionsbehandlungs-Verfahren, die Penalty-Methode ist aber besonders anfällig für diese Art von Problemen: Das Auftreten von *sekundären Schwingungen*, die im Extremfall zur Destabilisierung einer Objekt-Konfiguration führen können, wie sie exemplarisch in Abbildung A.6a aufgezeigt ist. Die Ursache dieses fehlerhaften Verhaltens ist in einem Resonanzeffekt zu suchen, der durch die Kompensation von Durchdringungen im unteren Teil eines Objektstapels beginnt, und sich sukzessive entlang des Stapels fortpflanzt; Abbildung A.6b zeigt die Abweichung von der Gleichgewichtslage für einen Stapel von sieben Würfeln identischer Masse, die bei einer Kantenlänge von einem Meter mit einer anfänglichen gegenseitigen Durchdringung von einem Zentimeter positioniert worden sind. Die maximale auftretende Abweichung von der idealen Ruhelage beträgt für den obersten Würfel im Stapel einen Meter, und die auftretende Sekundärschwingung tritt über mehr als eine Sekunde hinweg auf, bevor erneut eine Ruhelage auftritt. Ein zweiter Nachteil der Penalty-Methode besteht in der nötigen Kalibrierung der System-Parameter für das zugrundeliegende Modell aus harmonischen Oszillatoren in Kombination mit numerischen Lösungsverfahren für dieses Modell.

Weitergehende Informationen zu Anwendungen der Penalty-Methode und zu Alternativen für die Modellierung als harmonischer Oszillator sind beispielsweise in [MZ90], [Hir02], oder [JV03] zu finden.

A.2.3. Zwangsbedingungs-basierte Verfahren

Bei zwangsbedingungs-basierten Verfahren besteht eine Unterscheidung zwischen Kollisionen (ein bewegtes Objekt kollidiert mit einen stationären Hindernis) und bleibenden Kontakten (ein Objekt liegt in Ruhelage auf einem anderen).



(a) Objekt-Stapel **(b)** Positionsabweichungen in einem Objektstapel

Abbildung A.6.: Problematisches Verhalten bei der Penalty-Methode: Durch sekundäre Schwingungen bewegen sich einzelne Quader in einem Stapel aus Quadern (Abbildung A.6a), obwohl sie eigentlich eine stabile Ruhelage einnehmen sollten. Das Problem verstärkt sich, je weiter oben ein Quader im Stapel positioniert ist, da sich die Kollisionsreaktionen von Quadern unterhalb sukzessive Richtung oberes Ende des Stapels fortsetzt.

Für Kollisionen wird angenommen, dass sie nur eine sehr kleine Zeitdauer haben (einen sehr kleinen Bruchteil der Zeitschrittweite der Simulation); diese werden daher mittels Kraftstößen oder Reaktionsimpulsen aufgelöst. Für bleibende Kontakte mit einer Dauer über mehrere Iterationen der Simulation hinweg wird dagegen eine Kontaktkraft berechnet, die in Richtung der Kontakt-Normalen angesetzt wird. Kontaktkräfte sind Randbedingungen unterworfen: So darf eine Kontaktkraft nur dann wirken, solange sich zwei kollidierende Objekte aufeinander zu bewegen.

Für die Bestimmung der an bleibenden Kontakten anzusetzenden Kontaktkräfte gibt es eine Reihe unterschiedlicher Verfahren:

- Als quadratisches Programmierungsproblem, gelöst mittels heuristischer Verfahren ([Bar89]).
- Als lineares oder nichtlineares Komplementaritätsproblem ((Non-)Linear Complementary Problem, LCP/NCP) bei dem zusätzliche Randbedingungen für die Aufrechterhaltung eines bestehenden Kontaktpunkts in die Formulierung des zugrunde liegenden Gleichungssystems einfließen ([Löt82], [Löt84], [CC92]).
- Als Optimierungsproblem, in dem versucht wird, der Zielposition eines frei bewegten Körpers unter Berücksichtigung im Bewegungsverlauf entstehender Kollisionen möglichst nahe zu kommen ([MS01] und [SM04]).

Die Formulierung als lineares Komplementaritätsproblem (LCP) stellt als Randbedingungen für die Beschleunigung a und Kontaktkraft f an einem Kontaktpunkt auf:

$$a \geqq 0, f \geqq 0, f \cdot a = 0 \tag{A.7}$$

Diese Bedingungen stellen sicher, dass sich ein Objektpaar entweder von alleine auseinander bewegt, oder andernfalls eine Kontaktkraft anliegt, die die relative Beschleunigung der Körper am Kontaktpunkt auf Null reduziert.

Zu Beginn werden die Kontaktkräfte für alle Kontaktpunkte auf Null initialisiert. Dann wird für jeden Kontaktpunkt eine Kontaktkraft berechnet, die die Bedingungen aus Gleichung A.7 erfüllt. Ist für einen bestehenden Kontaktpunkt $a < 0$ (d. h. es würde zu einer Durchdringung kommen), so muss die anzuwendende Kontaktkraft korrigiert und für alle zuvor betrachteten Kontaktpunkte erneut überprüft werden, ob die Bedingungen aus Gleichung A.7 noch zutreffen.

Die Verwendung eines LCP zur numerischen Lösung der Newton-Euler-Gleichungen ist in [Erl05, Kapitel 7.1 und 7.2] detailliert beschrieben. Ein Problem bei der Verwendung LCP-basierter Ansätze ist, dass bei zusätzlicher Berücksichtigung von Reibungsverhalten unter Umständen keine oder keine eindeutige Lösung des aufgestellten Gleichungssystems gefunden werden kann.

Buck [Buc99, Kapitel 6] verwendet eine ähnliche Formulierung in Form eines auf ein lineares Gleichungssystem zurückgeführtes nichtlineares Komplementaritätsproblem, die im Fall einer nicht-konsistenten Lösung die Simulation auf den letzten konsistenten Zustand zurücksetzt.

A.2.4. Impulsbasierte Methoden

Bei impulsbasierten Methoden ([Hah88], [Mir96], [ST96]) werden keine expliziten Zwangsbedingungen für Kontaktpunkte formuliert. Stattdessen wird an Kontaktpunkten zwischen Körpern ein Reaktionsimpuls angelegt, der die gegenseitige Durchdringung der beteiligten Körper verhindert. Bei impulsbasierten Methoden muss nicht zwischen Kollisionen und bleibenden Kontakten unterschieden werden. Bleibende Kontakte werden durch Mikrokollisionen umgesetzt: In jedem Zeitschritt wird an einem bleibenden Kontaktpunkt ein neuer Reaktionsimpuls angelegt, der für die Aufrechterhaltung des bleibenden Kontaktes sorgt. Im Gegensatz zu zwangsbedingungs-basierten Ansätzen erfolgt die Bestimmung und sukzessive Anwendung von Kontaktimpulsen nicht simultan, sondern sequentiell. So ist pro Objekt-Paar zwar nur ein Kontaktimpuls gleichzeitig zu berechnen. Allerdings kann die unmittelbare Geschwindigkeits-Veränderung, die einem Körper durch einen Kontaktimpuls aufgeprägt wird, zu sekundären Kollisionen führen. Deswegen muss die Kollisionsbehandlung theoretisch so lange iterativ fortgesetzt werden, biss alle Kollisionen aufgelöst sind.

Dies ist jedoch aus Laufzeitgründen oft nicht möglich, so dass bei der Implementierung impulsbasierter Ansätze Vorkehrungen getroffen werden müssen, um etwa Endlos-Schleifen (die durch degenerierte Kontakt-Konfigurationen entstehen können) oder Resonanzeffekte (ähnlich den bei der Penalty-Methode in Unterabschnitt A.2.2 geschilderten) zu vermeiden. Beispielsweise können Reaktionsimpulse unterdrückt werden, wenn die relative Geschwindigkeit eines Kontaktpunktes in Richtung der Kontakt-Normalen unterhalb einer Toleranzgrenze liegt (wie bei [MC96]), oder die Kontaktbehandlung kann sortiert nach Kontaktpunkten mit der jeweils geringsten relativen Geschwindigkeit erfolgen (wie in [CR98]). Eine weitere Möglichkeit besteht in der Unterteilung der Kollisionsbehandlung in mehrere Teilschritte, in denen zunächst nur die Objekt-Geschwindigkeiten entsprechend der ermittelten Kontaktpunkte angepasst, und in einer weiteren Iteration ausgehend von der neuen Objektbewegung entstehende sekundäre Kontakte separat mittels Reaktionsimpulsen behandelt werden (wie bei [GBF03], [Gue06]).

Gängige echtzeitfähige Starrkörpersimulationen wie ODE, Bullet, Havok oder PhysX verwenden impulsbasierte Methoden im Rahmen der Kollisionsbehandlungs-Phase. Eine umfassendere Behandlung impulsbasierter Ansätze findet sich etwa in [Eri05, Kapitel 6] oder [Ebe10, Kapitel 6].

A.2.5. Weitere Aufgaben der Kollisionsbehandlung

Neben der Bestimmung des Bewegungsverlaufs von Starrkörpern unter dem Einfluss von externen und Kontakt-Kräften obliegen der Kollisionsbehandlung noch weitere Aufgaben, wie beispielsweise:

- Die Behandlung von Bewegungsbeschränkungen zwischen Objekten in Form von Gelenken mit unterschiedlichen Translations- und Rotations-Freiheitsgraden
- Die Simulation von Reibungsverhalten

A.2.5.1. Bewegungsbeschränkungen zwischen Objekten

Kontakt-Punkte sind *uniliterale Zwangsbedingungen*: Es gilt eine$\geqq$-Relation in Bezug auf den Betrag anliegender Kontaktkräften (Gleichung A.7). Dagegen sind Zwangsbedingungen, die nicht durch gegenseitigen Kontakte bedingt sind, *bilaterale Zwangsbedingungen*: Es gilt eine $=$-Relation zwischen den Kräften, die auf zwei durch ein Gelenk verbundene Objekte ausgeübt werden.

Unterschieden wird zwischen:

- *Holonomen Zwangsbedingungen*, beschrieben durch implizite Funktionen (abhängig von der Lage eines Körpers und der Zeit) der Form

$$\mathbf{C}(\vec{x}, t) = 0 \tag{A.8}$$

  Holonome Zwangsbedingungen reduzieren die Freiheitsgrade in einem Mehrkörpersystems. Sie sind zur Beschreibung von Gelenken zwischen Objekten geeignet.

- *Nicht-holonomen Zwangsbedingungen*: Diese lassen sich nicht in ausschließlich in Abhängigkeit von Lage und Zeit beschreiben; bespielsweise hängen sie zusätzlich von der Geschwindigkeit betroffener Objekte ab, oder liegen als Ungleichungen vor:

$$\mathbf{C}(\vec{x}, \vec{v}, t) = 0 \tag{A.9}$$

  In diesem Sinn sind auch Kontakt-Punkte nicht-holonome Zwangsbedingungen.

Es existieren verschiedene Lösungsansätze zur Umsetzung von Zwangsbedingungen. Die folgende Zusammenfassung orientiert sich an den Klassifikationen in [Ben07, Abschnitt 2.1] und [Eri05, Abschnitt 7.5 bis 7.9].

Mit Hilfe der Penalty-Methode Beschrieben über implizite Funktionen, die von Zeit, Objektlage, und -geschwindigkeit abhängig sein können. Für holonome Bedingungen bestimmt sich die anzusetzende Kraft zur Einhaltung einer Zwangsbedingung $\mathbf{C}$ durch

$$\mathbf{F}_p = -\alpha \mathbf{J}^T(\Omega^2 \mathbf{C} + 2\Omega\mu\dot{\mathbf{C}} + \ddot{\mathbf{C}})$$

Die konstanten Parameter α, μ, Ω können dabei als Stärke, Eigenfrequenz und Dämpfung einer Feder interpretiert werden.

Die Methode reduzierter Koordinaten Diese unterstützt nur holonome Zwangsbedingungen. Wie in Abschnitt A.3 (Gleichung A.12) verringern m Zwangsbedingungen die Freiheitsgrade eines Mehrkörpersystems, so dass die n *allgemeinen Koordinaten* eines solchen Systems auch mittels *reduzierter Koordinaten* in der Form

$$x_i = x_i(q_1, \cdots, q_{n-m}), i \in [1, \cdots, n]$$

beschrieben werden kann. Die allgemeinen Koordinaten des Systems werden zur Lösung des entstehenden Gleichungssystems durch die reduzierten Koordinaten ersetzt und Bewegungsgleichungen bezüglich der reduzierten Koordinaten q_i aufgestellt, die dann mit Hilfe von Verfahren zur numerischen Integration gelöst werden können.

Die Lagrange-Faktoren-Methode Bei dieser Methode werden Zwangsbedingungen mittels interner Kräfte umgesetzt. Ist der Zustand eines Objekts mittels eines Lage-Vektors $\vec{x}$ (Position und Orientierung kombiniert) und eines Geschwindigkeits-Vektors $\vec{v}$ erfasst, und Masse m und Trägheitstensor J in einer Matrix der Form

$$M(t) = \begin{pmatrix} m\mathbf{I}^3 & \mathbf{0} \\ \mathbf{0} & J(t) \end{pmatrix} \tag{A.10}$$

zusammengefasst, so lässt sich eine Bewegungsgleichung für die auf einen Körper einwirkende Kraft als gewöhnliche Differentialgleichung 2. Ordnung in der Form

$$\mathbf{F} = \frac{\partial^2}{\partial^2 t}(\mathbf{M}(t) \cdot \vec{x}(t))$$

aufstellen.

Aus

$$\frac{\partial}{\partial t}\vec{p}(t) = F$$
$$\frac{\partial}{\partial t}\vec{x}(t) = \mathbf{M}^{-1}(t)\vec{p}(t) = \vec{v}(t)$$

lässt sich wiederum mittels numerischer Integration der neue Zustand eines Körpers nach einem diskreten Zeitschritt $t_0 + \Delta t$ ausgehend von einem bekannten Zustand zum Zeitpunkt t_0 bestimmen.

A.2.5.2. Reibungsverhalten

Die Funktionsweise der Simulation von Reibungsverhalten soll sich auf die kurze Erläuterung einer möglichen Umsetzung anhand der Penalty-Methode beschränken. Abbildung A.7a zeigt das Prinzip des Ansatzes: Um gemäß dem Coulombschen Reibungsgesetz Haft- beziehungsweise Gleit-Reibung umzusetzen, wird zwischen zwei sequentiell registrierten Kontaktpunkten wiederum eine Feder-Analogie platziert, die im Fall von Haftreibung die Bewegung eines Objekts entlang der Oberfläche eines anderen neutralisiert, im Fall von Gleit-Reibung der Bewegungsrichtung entgegengesetzt eine Reibungskraft ausübt. Für die Gleit-Reibung wird der Ansatzpunkt der Feder der Objektbewegung angepasst nachgeführt. Der Betrag der Reibungskraft ergibt sich mit dem Ansatzpunkt $\mathbf{a}$ und einem aktiven Kontaktpunkt $\mathbf{p}$ zu

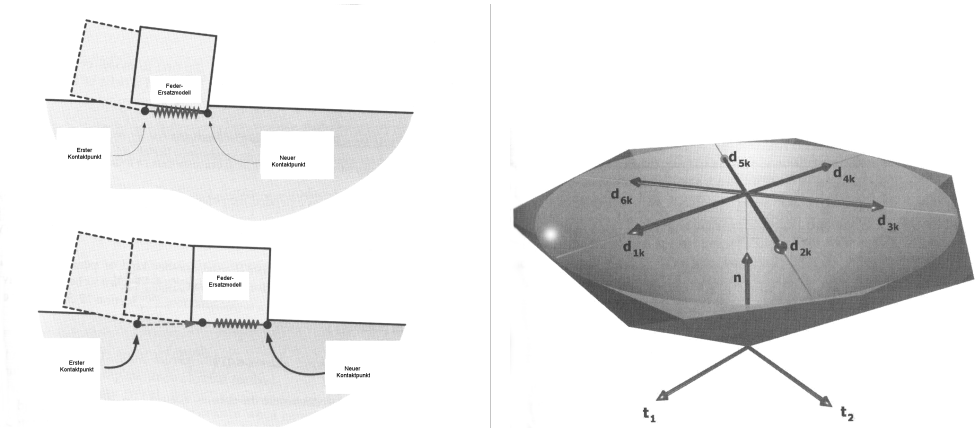
$$F_{reibung} = k(\mathbf{a} - \mathbf{p})$$

mit der Federkonstante k. Als Übergangs-Bedingung zwischen Haft- und Gleitreibung dient

$$\|F_{reibung}\| \leqq \mu \|F_{normale}\|$$

mit dem Reibungskoeffizienten μ und der Kontaktkraft $F_{normale}$ entgegen der Richtung der Kontakt-Normalen. Kommt es zum Übergang zwischen Haft- und Gleitreibung, so ergibt sich die Reibungskraft zu

$$\|F_{reibung}\| = \mu \|F_{normale}\|$$



(a) Reibungsverhalten in der Penalty-Methode

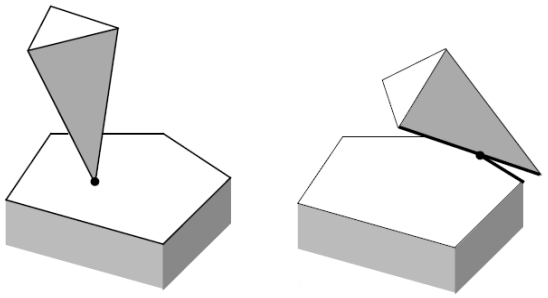
(b) Approximation eines Reibungs-Kegels

Abbildung A.7.: Modellierung von Reibung in Starrkörpersimulationen (nach [Eri05, Kapitel])

Abbildung A.7b skizziert eine mögliche Konfiguration eines komplexeren Reibungsmodells: Der Reibungs-Kegel an einem Kontaktpunkt wird durch eine Reibungs-Pyramide approximiert. Hier werden auftretende Reibungskräfte nicht nur durch einzelne Verbindungen zwischen Kontaktpunkten aus der Kontakt-Historie eines Objekts repräsentiert: Stattdessen werden tangential zur Kontakt-Ebene (aufgespannt durch $\vec{t_1}$ und $\vec{t_2}$ in Abbildung A.7b) radial Reibungskräfte entlang $d_{1k}, \ldots, d_{6k}$ in Abbildung A.7b angesetzt.

A.3. Kontakt-Konfigurationen zwischen polygonalen Objekten

Betrachtet man alle möglichen Kombinationen aus Kontakten zwischen Ecken, Kanten und Seitenflächen (Tabelle A.1), so wird deutlich, dass sich alle auftretenden Kombinationen auch in degenerierten Fällen mittels Kontakten zwischen einem Eckpunkt und einer Fläche (Abbildung A.8a) beziehungsweise einem Paar aus Kanten (Abbildung A.8b) ersetzen lassen.



(a) Kontakt zwischen Ecke und Fläche

(b) Kontakt zwischen zwei Kanten

Abbildung A.8.: Kontakt-Konfigurationen zwischen Elementen von polygonalen Geometrien (aus [Buc99, Kapitel 4])

Diese Tatsache macht sich etwa eine Variante für den Schnitt-Test zwischen zwei Dreiecken zunutze, die in Unterabschnitt B.2.2 im Detail diskutiert wird.

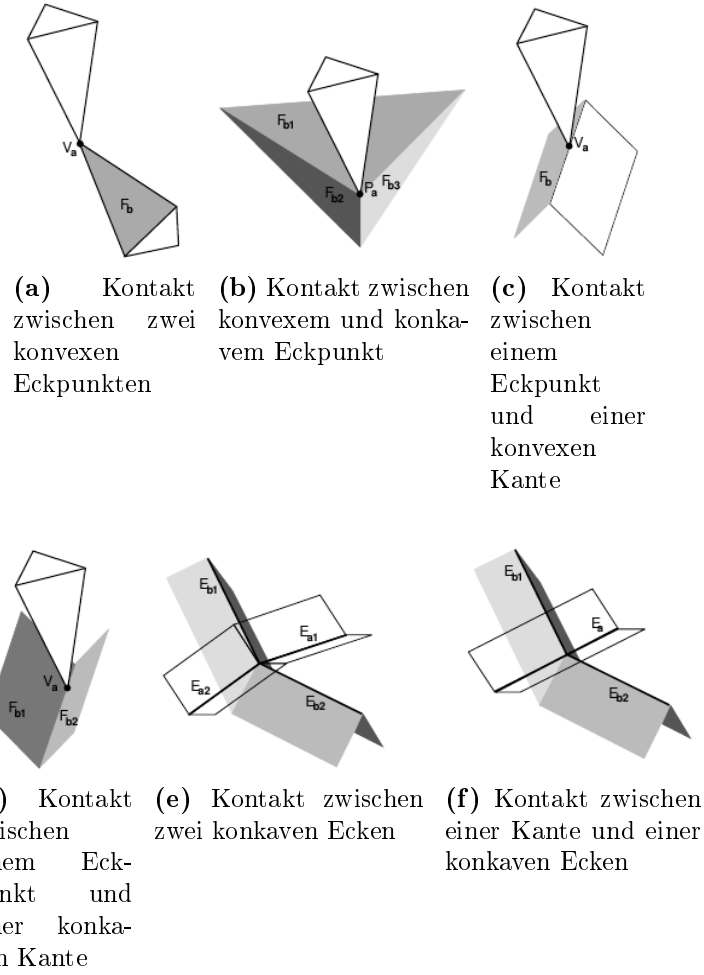
Art des Kontakts	Ersatz-Konfiguration
Ecke-Fläche	Ecke-Fläche (Abbildung A.8a)
Kante-Kante	Kante-Kante (Abbildung A.8b)
Fläche-Kante	Zwei Kontaktpunkte (Ecke-Fläche oder Kante-Kante) (Abbildung A.11a)
Fläche-Fläche	Drei Kontaktpunkte (Ecke-Fläche oder Kante-Kante) (Abbildung A.11b)
konvexe Ecke - konvexe Ecke	Ecke-Fläche oder Kante-Kante (Abbildung A.9a)
konvexe Ecke - konkave Ecke	3 Ecke-Fläche-Kontakte (Abbildung A.9b)
konvexe Ecke - konvexe Kante	Ecke-Fläche oder Kante-Kante (Abbildung A.9c)
konvexe Ecke - konkave Kante	2 Ecke-Fläche-Kontakte (Abbildung A.9d)
Sattelpunkt - Sattelpunkt	4 Kante-Kante-Kontakte (Abbildung A.9e)
Sattelpunkt - konvexe Kante	2 Kante-Kante-Kontakte (Abbildung A.9f)

Tabelle A.1.: Mögliche Kontakt-Konfigurationen zwischen polygonalen Geometrien (nach [Buc99, Kapitel 4])

Tabelle A.1 fasst die Ersatzmodelle für weitere mögliche Kombinationen aus Kontakten zwischen Elementen polygonaler Geometriemodelle zusammen.

Kontakte zwischen zwei aufeinander liegenden Seitenflächen oder zwischen einer Fläche und einer aufliegenden Kante können durch eine gleichwertige Kombination aus Ecke-Fläche- und

Kante-Kante-Kontakte beziehungsweise durch zwei Ecke-Fläche-Kontakte an den Endpunkten einer aufliegenden Kante ersetzt werden (Abbildung A.10a und Abbildung A.10b).



(a) Kontakt zwischen zwei konvexen Eckpunkten

(b) Kontakt zwischen konvexem und konkavem Eckpunkt

(c) Kontakt zwischen einem Eckpunkt und einer konvexen Kante

(d) Kontakt zwischen einem Eckpunkt und einer konkaven Kante

(e) Kontakt zwischen zwei konkaven Ecken

(f) Kontakt zwischen einer Kante und einer konkaven Ecken

Abbildung A.9.: Ersatz-Modelle für weitere Kontakt-Situationen zwischen Elementen polygonaler Geometrien (aus [Buc99, Kapitel 4])

Die Kontaktbedingung zwischen einem konvexen Polygon und einer Kante lässt sich als Schnittmenge einer Ebene und eines Geradenabschnitts zwischen zwei Punkten a_1 und a_2 formulieren. Gilt für diese zwei Punkte:

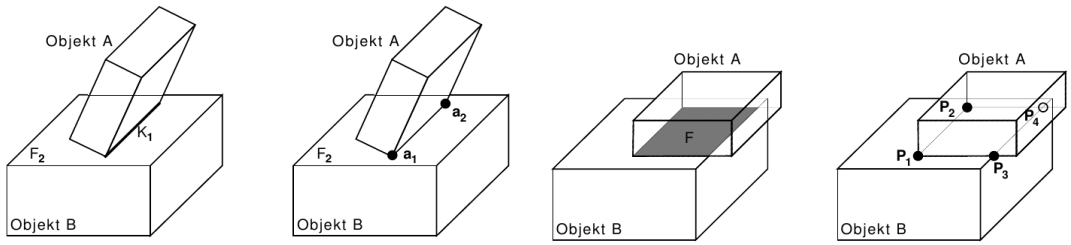
$$\vec{n}_E^T a_1 - d_E = 0 \wedge \vec{n}_E^T a_2 - d_E = 0$$

Dann gilt für alle Punkte auf einem durch a_1 und a_2 begrenzten Geradenabschnitt:

$$\vec{n}_E^T(a_1 + \lambda(a_2 - a_1)) - d_E = 0 \forall \lambda \in [0,1] \tag{A.11}$$

mit einer Ebene E in Normalen-Form:

$$E : \vec{n}_E^T \vec{x} = d_E$$



(a) Kontakte zwischen Kante und Fläche **(b)** Kontakte zwischen zwei Flächen

Abbildung A.10.: Ersetzen von Kante-Fläche- und Fläche-Fläche-Kontakte durch Kontakt-Punkte (aus [Buc99, Kapitel 4])

(schraffiert in Abbildung A.13a). Somit lässt sich ein Kontakt zwischen einer Kante und einer Fläche durch zwei punktförmige Kontakte ersetzen.

Eine Kontaktbedingung zwischen zwei konvexen Polygonen lässt sich mit einer ähnlichen Überlegung finden: Betrachtet man die sich ergebende Schnittfläche in einer solchen Kontakt-Konfiguration (Abbildung A.11b), so ist diese wiederum ein konvexes Polygon. Ein Algorithmus zur Berechnung des Schnitt-Polygons zweier konvexer Polygone wird beispielsweise in [O'R08, Kapitel 7.6] vorgestellt.

Ob ein Punkt innerhalb eines konvexen Polygons liegt, lässt sich beispielsweise mit Hilfe von Algorithmus 15 ermitteln.

Funktion `Cross2D`

1 // Pseudo-Kreuzprodukt von $\vec{v}$ und $\vec{w}$
  Input : $\vec{v}, \vec{w}$
  Output : b
2 $b = v.y() \cdot w.x() - v.x() \cdot w.y()$;
3 **return** b;

Funktion `VectorOnWhichSide2D`

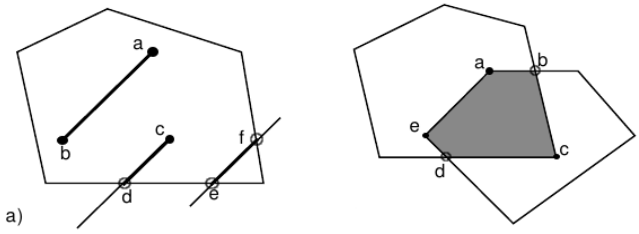
1 // Orientiert sich $\vec{w}$ rechts- oder linksseitig zu $\vec{v}$?
  Input : $\vec{v}, \vec{w}$
  Output : s
2 $cp = \text{Cross2D}(\vec{v}, \vec{w})$;
3 **if** $cp > 0$ **then**
4 $s = \text{LEFT}$;
5 **else if** $cp < 0$ **then**
6 $s = \text{RIGHT}$;
7 **else**
8 $s = \emptyset$;
9 **return** s;

Algorithmus 15 : PointInConvexPolygon

```
1  // Verfahren zur Überprüfung, ob ein Punkt innerhalb eines konvexen Polygons liegt
   Input : p, 𝕍
2  ps = ∅;
3  cs = ∅;
4  for i = 0 → length(𝕍) do
5      a = V[n], b = V[n + 1]%(𝕍);
6      as = b − a;
7      ap = b − p;
8      cs = VectorOnWhichSide2D(as, ap);
9      if cs = ∅ then
10         return false;
11     else if ps = ∅ then
12         ps = cs;
13     else if ps ≠ cs then
14         return false;
15     return true;
```

Die Menge aller Punkte P_i, die in der Schnittfläche enthalten sind, lässt sich so ausdrücken als:

$$P_i = \{p \parallel \text{PointInConvexPolygon}(p) = \text{ true}\}$$



(a) Kontakte zwischen Kante und Fläche

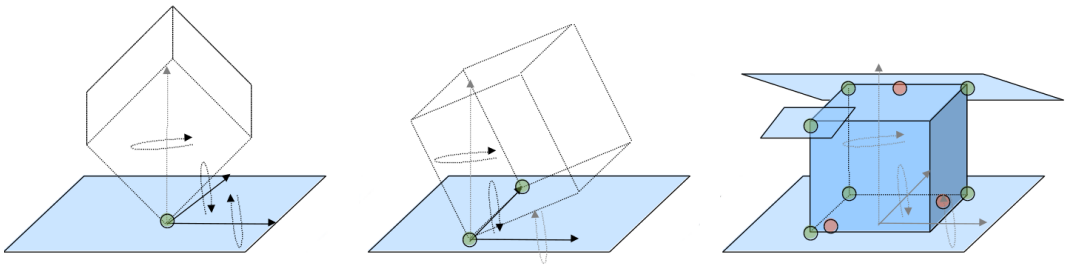
(b) Kontakte zwischen zwei Flächen

Abbildung A.11.: Ersatz-Modelle für Kontakte zwischen Kanten und Flächen (aus [Buc99, Kapitel 4])

Analog zum Fall eines Kontakts zwischen einer Kante und einem Polygon lässt sich für den Kontakt zwischen zwei konvexen Polygonen argumentieren, dass sich eine solche Kontakt-Fläche durch die Menge aller Punkte auf den Kanten des Schnitt-Polygons beschreiben lässt (wie in Gleichung A.11). Ebenso kann nun für jede einzelne Kante des Schnitt-Polygons ein Kante-Fläche-Kontakt verwendet werden, um die Umfassung der Kontaktregion zu erfassen. Auch kann jeder dieser Kante-Fläche-Kontakte wieder durch zwei einzelne Kontakt-Punkte ersetzt werden.

Allerdings ist diese Kontaktkonfiguration redundant, da mehr als drei Kontaktpunkte in einem Fläche-Fläche-Kontakt nicht mehr zur Verringerung der Bewegungs-Freiheitsgrade eines

Objektes in 3D führen. Abbildung A.12 zeigt dies an einem einfachen Beispiel: Sobald ein Starrkörper mehr als sechs Beschränkungen durch Kontakte mit anderen Objekten unterliegt, beinhaltet eine solche Kontaktpunkt-Konfiguration redundante Restriktionen (Abbildung A.12c).



(a) Beschränkung eines Freiheitsgrads **(b)** Beschränkung zweier Freiheitsgrade **(c)** Redundante Beschränkungen

Abbildung A.12.: Freiheitsgrade bei unterschiedlichen Kontaktpunkt-Konfigurationen

Formal ausgedrückt entspricht jeder der unabhängigen Freiheitsgrade eines in 3D bewegten Objektes einer Randbedingung in einem Sechs-Komponenten-Vektor

$$\vec{m} = (m_1, \ldots, m_6)^T \tag{A.12}$$

Jeder nicht redundante Kontaktpunkt entspricht einer skalaren Bedingung $m_{1|\ldots|6} = 0$ und reduziert die Anzahl unabhängiger Freiheitsgrade um 1. Ein redundanter Kontaktpunkt entspricht dagegen einer linear abhängigen Bedingung.

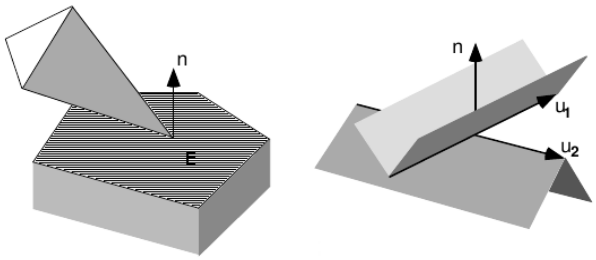
Die Bestimmung von Kontaktpunkt-Konfigurationen spielt eine sehr wichtige Rolle für die Stabilität einer Mechaniksimulation, etwa bei Objekten in Ruhelage oder aufeinander liegenden Objekten, aber auch bei der Behandlung umschließender Kontakt-Situationen. In diesem Zusammenhang ist die möglichst vollständige Analyse einer simulierten Umgebung auf Kontakte zwischen Objekten eine der wichtigsten Anforderungen an eine Kollisionserkennungs-Komponente: Sofern eine solche nicht in der Lage ist, eine Szene vollständig auf Kollisionen zwischen Objekten zu überprüfen, ist die Kollisionsbehandlung nicht in der Lage, Durchdringungen zwischen Objekten zu verhindern, wenn keine Informationen über betreffende Kontaktsituationen spezifiziert sind.

A.3.1. Bestimmung von Kontakt-Normalen

Bevor exemplarisch auf verschiedene Alternativen zur Bestimmung und Verfolgung von Kontakt-Konfigurationen eingegangen wird, ist noch eine weitere Eingangsgröße bei Kontaktpunkt-Konfigurationen, die für die Mechaniksimulation zur Bestimmung von Kontakt-Kräften beziehungsweise -Impulsen von Wichtigkeit ist, zu berücksichtigen: Die Kontakt-Normale, welche die Richtung bestimmt, in der eine mechanische Reaktion bei gegenseitigen Objekt-Kontakten wirken soll.

Bei Ecke-Fläche- und Kante-Kante-Kontakten ist es möglich, eine eindeutige Kontaktnormale zu definieren:

- Bei Ersteren ist die Kontaktnormale gleich der Flächen-Normale (Abbildung A.13a).
- Bei Letzteren lässt sich diese durch das Kreuzprodukt der Kanten-Richtungen bestimmen (Abbildung A.13b).



(a) Ecke-Fläche-Kontakt **(b)** Kante-Kante-Kontakt

Abbildung A.13.: Kontaktnormalen bei Ecke-Fläche- und Kante-Kante-Kontakten

Für einen Ecke-Fläche-Kontakt zwischen zwei Objekten G_1 und G_2 ergibt sich eine Kontakt-Bedingung mit

- $\hat{v}$: Beteiligter Eckpunkt von G_1
- $E_2 : \vec{\hat{n}}^T \mathbf{x} = \hat{d}$: Ebene, in der die beteiligte Ebene von G_2 liegt
- $t_1, t_2 \in \mathbb{R}^3$: Translations-Vektoren zur Transformation aus den lokalen Objekt- in ein gemeinsames Weltkoordinaten-System
- $R_1, R_2 \in \mathbb{R}^{3\times3}$: Rotationsmatrizen zur Transformation aus den lokalen Objekt- in ein gemeinsames Weltkoordinaten-System

zu

$$c = R_1 \cdot \hat{v} + t_1 \tag{A.13}$$
$$\vec{n} = R_2 \cdot \vec{\hat{n}} \tag{A.14}$$
$$w = \vec{\hat{n}} R_2^T (R_1 \hat{v} + t_1 - t_2) - \hat{d} \tag{A.15}$$

mit dem *Kontaktpunkt* c, der *Kontakt-Normalen* $\vec{n}$ und dem *Kontakt-Abstand* w.

Für einen Kante-Kante-Kontakt zwischen zwei Objekten G_1 und G_2 ist eine Kontakt-Bedingung mit

- Kante K_1 aus G_1 mit den Anfangs-/Endpunkten v_1 und v_2
- Kante K_2 aus G_2 mit den Anfangs-/Endpunkten w_1 und w_2
- $t_1, t_2 \in \mathbb{R}^3$: Translations-Vektoren zur Transformation aus einem lokalen Objekt- in ein gemeinsames Weltkoordinaten-System
- $R_1, R_2 \in \mathbb{R}^{3\times3}$: Rotationsmatrizen zur Transformation aus den lokalen Objekt- in ein gemeinsames Weltkoordinaten-System

durch

$$\vec{n} = R_1(w_1 - v_1) \times R_2(w_2 - v_2) \tag{A.16}$$

$$w = \frac{\vec{n}^T(v_1 - v_2)}{\|\vec{n}\|} \tag{A.17}$$

mit der *Kontakt-Normalen* $\vec{n}$ und dem Kontakt-Abstand w. Die Koordinaten des Kontaktpunkts c ergeben sich beispielsweise als ein Resultat der Anwendung von Algorithmus 19.

ANHANG B

GEOMETRISCHE GRUNDLAGEN

B.1. Geometrie-Repräsentation

Dieser Abschnitt gibt einen Überblick über verschiedene Möglichkeiten zur Darstellung der Geometrie dreidimensionaler Objekte. Herausgehoben werden sollen dabei polygonale Oberflächenbeschreibungen, die vor allem durch ihre weit verbreitete Nutzung in der Computergrafik als eng zur Kollisionserkennung verwandtem Gebiet für praktisch alle existierenden Kollisionserkennungsverfahren sowohl als Eingabe-Daten als auch als grundlegende Datenstruktur für die Bestimmung von Kontakten zwischen Objekten dienen.

Es ist hilfreich, unterschiedliche Alternativen zur Repräsentation eines dreidimensionalen Objekts anhand der Funktionsweise der jeweiligen Verfahren zu klassifizieren. Beispielsweise bei Stroud ([Str06]) findet sich ein nützliches Klassifizierungs-Schema: Danach lassen sich Verfahren unterteilen in oberflächen- und volumen-gebundene Ansätze:

- Mengentheoretische Ansätze: Constructive Solid Geometry (CSG), volumengebunden
- Generative Modellierung: Modellierung durch Manipulation eines sog. Generators, so erzeugt etwa das Verschieben (Extrudieren) einer Kurve im Raum eine Freiformfläche; sowohl oberflächen- als auch volumengebunden
- Zellen-basiert: Diskretisierung eines Objektes etwa mittels eines strukturierten Raumgitters, volumengebunden
- Begrenzungsflächenmodelle/Boundary Representation (B-Rep): Oberflächengebundene Darstellung der Hülle eines Objektes mit Hilfe parameterisierter Flächen oder Kurven
- Polygonisierung: Oberflächengebundene Darstellung der Objekthülle mittels planarer Polygone

Eine vergleichbare Klassifizierung findet sich auch in [LG98], dargestellt in Abbildung B.1.

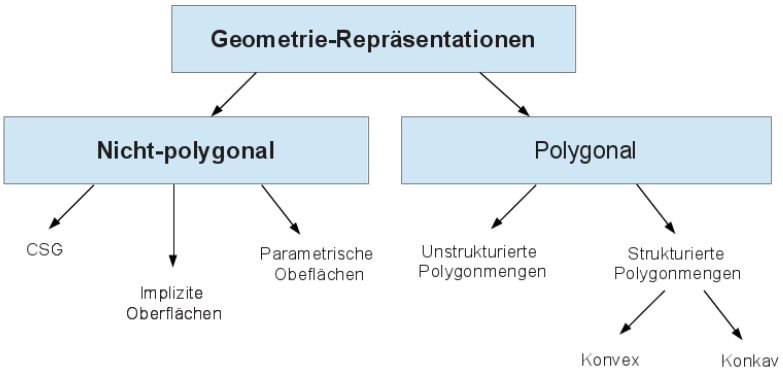


Abbildung B.1.: Klassifikation von Geometrie-Repräsentationen nach Lin und Gottschalk ([LG98])

Im Folgenden soll zur Abgrenzung gegenüber polygonalen Oberflächenbeschreibungen und zellen-basierten Verfahren, welche beide im weiteren Verlauf genutzt werden, kurz auf den mengentheoretischen Ansatz und die generelle oberflächengebundene Darstellung eingegangen werden, und deren Vor- und Nachteile im Hinblick auf die Verwendung in der Kollisionserkennung erläutert werden.

B.1.1. Begrenzungsflächenmodelle (Boundary Representations)

B.1.1.1. Die Datenstruktur

Begrenzungsflächenmodelle beschreiben ein Objekt durch eine Menge von Flächen-Elementen, die zusammen dessen Oberfläche umschließen.

Für die Spezifikation der einzelnen Seitenflächen ist zwischen zwei unterschiedlichen Arten von Abbildungs-Vorschriften zu unterscheiden:

- Implizit definierte Oberflächen: Diese werden durch Funktionen der Form $f : \mathbb{R}^3 \mapsto \mathbb{R}$ definiert, wobei die implizite Oberfläche selbst durch die Punkte definiert ist, die die Bedingung $f(x, y, z) = 0$ erfüllen.
- Parametrisch definierte Oberflächen: Diese werden durch Funktionen der Form $f : \mathbb{R}^2 \mapsto \mathbb{R}^3$ definiert, entsprechend der Abbildung aus einer Ebene in den dreidimensionalen Raum.

Parametrisch definierte Oberflächen werden mittels stückweise definierter Kurven definiert; die bekanntesten Varianten dieser Klasse von sind B-Splines beziehungsweise Non-Uniform Rational B-Splines (NURBS).

Am Beispiel von Beziér-Kurven soll kurz die Funktionsweise von parametrisch definierten Oberflächen erläutert werden. Im Anschluß daran folgt eine kurze Diskussion von Ansätzen zur Kollisionserkennung auf Basis von parametrisch definierten Oberflächen. B-Splines und NURBS verwenden andere Basisfunktionen, bedienen sich aber ansonsten desselben Prinzips.

Eine Beziér-Kurve wird durch ein Kontrollpolynom der Form

$$P(t) = \sum_{i=1}^{N} B_i J_{n,i}(t) \tag{B.1}$$

mit

$$J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{B.2}$$

definiert, wobei B_i Punkten auf dem Kontrollpolynom der Beziér-Kurve entspricht und $J_{n,i}$ die Basisfunktionen des Kontrollpolynoms sind. Der Grad des Kontrollpolynoms liegt immer um eins niedriger als die Anzahl der Kontrollpunkte für eine Beziér-Kurve. Der erste und letzte Punkt auf dem Kontroll-Polygon (B_0 und B_n) liegen koinzident auf dem ersten und letzten Punkt der Beziér-Kurve. Abbildung B.2 zeigt eine Beziér-Kurve mit ihrem Kontrollpolynom und dessen Kontrollpunkten.

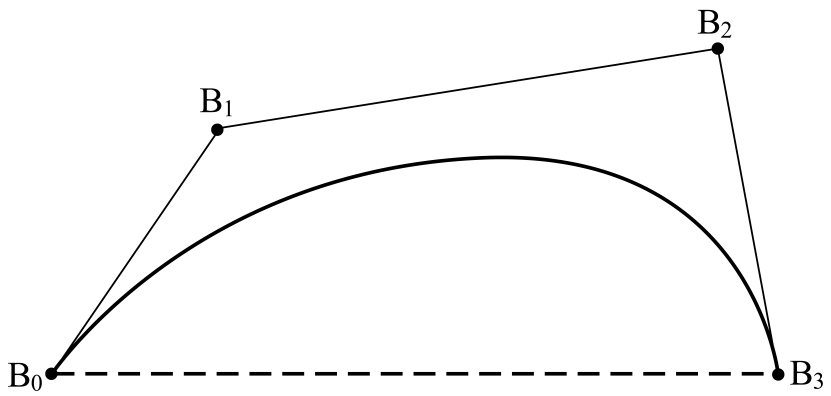


Abbildung B.2.: Eine Beziér-Kurve mit ihrem Kontroll-Polygon

Eine Beziér-Fläche entsteht durch die Kombination zweier Beziér-Polynome (verdeutlicht in Abbildung B.3); die Kontrollpunkte zur Interpolation der beiden Beziér-Kurven sind nun in einem Kontroll-Netz über die modellierte Oberfläche verteilt.

Die Fläche ist durch eine Funktion der Form

$$Q(u,w) = \sum_{i=0}^{n} \sum_{j=0}^{n} B_{i,j} J_{n,i}(u) K_{m,j}(w) \tag{B.3}$$

definiert, wobei $B_(i,j)$ den Kontrollpunkten und $J_{n,i}$ und $K_{m,j}$ die jeweiligen Basisfunktionen der Beziér-Kurven entlang der Fläche sind; deren Form entspricht der in Gleichung B.1 und Gleichung B.2.

Weitere Details zu den für B-Splines und NURBS verwendeten Basisfunktionen sowie den Stetigkeits-Eigenschaften von parameterisierten Kurven und Flächen (wichtig bei der Modellierung von Oberflächen aus mehreren, stückweise definierten Flächen) sind [Rog01] zu entnehmen.

B.1.1.2. Anwendungen in der Kollisionserkennung

Verfahren zur Kollisionserkennung auf Basis von parameterisierten Oberflächen werden von Lin und Gottschalk ([LG98]) in unterschiedliche Klassen eingeteilt, die sich meist unterschiedlicher Arten von Subdivisions-Schemata bedienen, um nach möglichen Überschneidungen zu suchen.

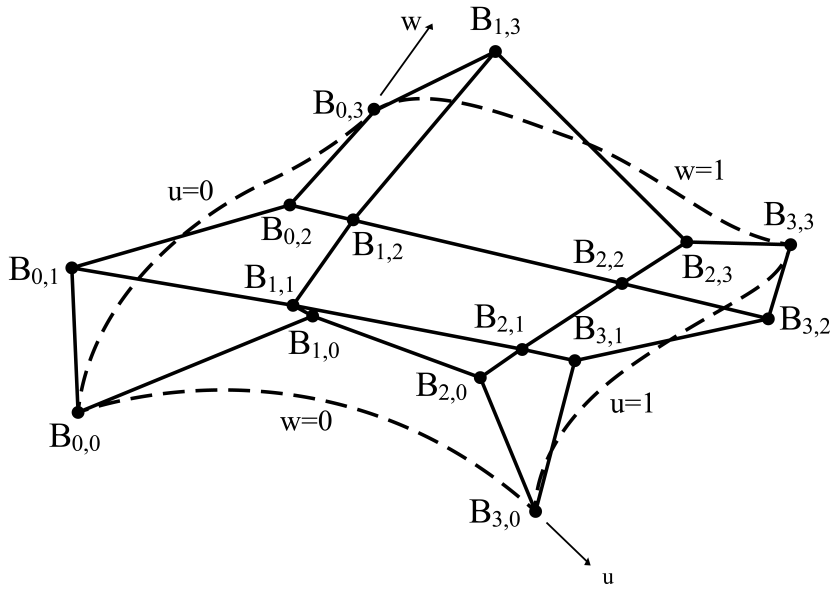


Abbildung B.3.: Ein Beziér-Patch mit Kontrollpunkten

Stellvertretend sei kurz ein in [VBZ90] und [SWF+93] vorgestelltes Verfahren erläutert: Dieses ermöglicht die Erkennung von Kontakten zwischen sich bewegenden und verformbaren parameterisierten Oberflächen, konkreter einer Abbildung der Form

$$f : (u, v) \times t \mapsto \mathbb{R}^3$$

unterworfenen Flächenabschnitten. Lipschitz-Bedingungen werden genutzt, um die Ausdehnung einer parameterisierten Oberfläche im Bildraum zu bestimmen, und das Ergebnis dieser Berechnung wird benutzt, um einen Hüllkörper für die betrachtete Teil-Oberfläche zu berechnen.

Eine Lipschitz-Bedingung sagt aus, dass

$$\|\vec{f}(\vec{u}_2) - \vec{f}(\vec{u}_1)\| = L\|\vec{u}_2 - \vec{u}_1\|$$

für einen Faktor L für eine Teil-Region der parametrischen Oberfläche $\vec{f}(\vec{u})$ gilt.

Abbildung B.4 skizziert diesen Zusammenhang und zeigt eine Hüllkugel, die durch die Anwendung einer Lipschitz-Bedingung bestimmt wurde.

Mögliche in Kontakt befindliche Regionen einer Oberfläche werden durch Schnitt-Tests zwischen den bestimmten und adaptive Subdivision mittels eines k-d-Baums eingegrenzt, bis Abschnitte der betrachteten Oberflächen erreicht sind, die eine bestimmten minimalen Abstand zueinander unterschreiten.

Die Laufzeiteigenschaften des soeben beschriebenen Verfahrens (ebenso wie die verwandter Ansätze, die direkt auf parameterisierten Oberflächen als Geometrie-Repräsentation basieren) werden jedoch unter anderem von Lin und Gottschalk als nicht geeignet für den Einsatz in interaktiven Anwendungen beschrieben. Ebenso ist zu bemerken, dass die Veröffentlichungen zu Verfahren dieser Art bereits beträchtliche Zeit zurückliegen, und innerhalb der letzten Jahre in ähnlicher Form nicht wieder aufgegriffen worden sind.

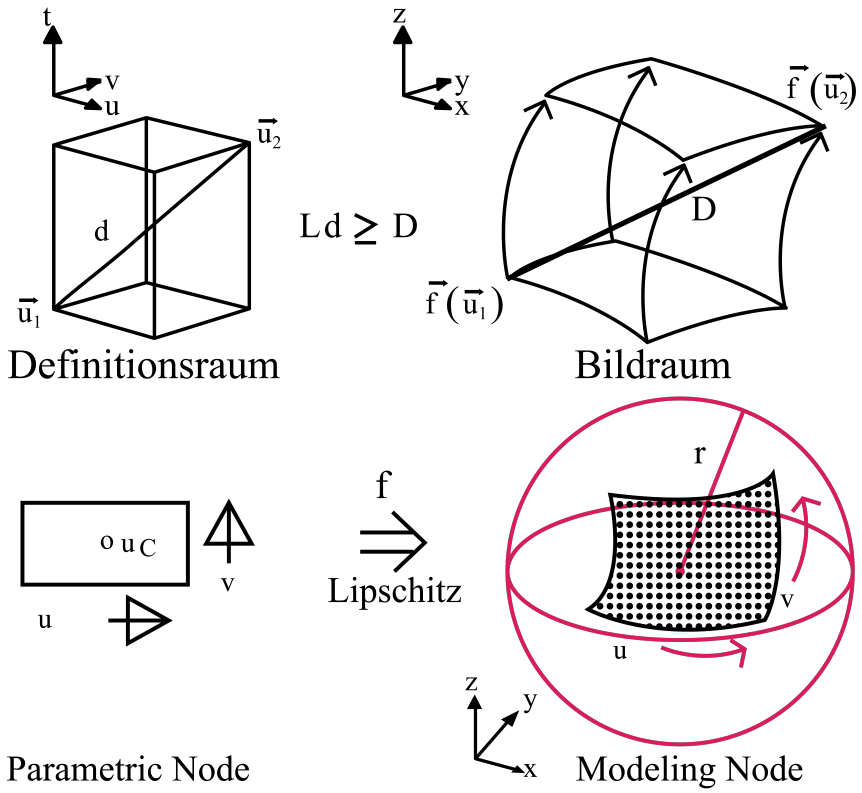


Abbildung B.4.: Lipschitz-Bedingung und daraus resultierende Hüllkugel

Greß, Guthe und Klein stellen in [GGK06] einen Ansatz vor, der sich ebenfalls eines Subdivisions-Schemas bedient, das von einem Algorithmus zur grafischen Darstellung von NURBS-Flächen gesteuert wird ([GBK05]). Auch hier werden anhand des Subdivisions-Schemas sukzessiv feiner strukturierte Teile einer Hüllkörper-Hierarchie berechnet; die Eingrenzung weiter zu untersuchender Teilbereiche der Szene erfolgt hier wie im Fall des zuvor beschriebenen Algorithmus aus [VBZ90] anhand eines NURBS-basierten Geometriemodells. Allerdings dient das NURBS-Modell hier nur zur Erzeugung der Hüllkörper-Hierarchie, während die weitere Berechnung von Kontaktpunkten auf ein polygon-basiertes Modell zurückgreift.

B.1.2. Mengentheoretische Ansätze: Constructive Solid Geometry (CSG)

B.1.2.1. Die Datenstruktur

Ein CSG-Modell entsteht durch die Anwendung von Mengen-Operatoren (Vereinigung, Durchschnitt, Komplement etc.) auf eine Kombination von verschiedenen geometrischen Primitiven (etwa Kugel, Quader, Zylinder), die über eine hierarchische Anordnung sukzessive die Struktur der gewünschten Geometrie spezifizieren.

Der Vorteil einer CSG-Repräsentation liegt in der intuitiven Konstruktionsweise, mittels der Objekte durch "Durchbohren"/"Aussparen" (Durchschnitt/Komplement) und "Zusammen-

fassen" (Vereinigung) auf eine Weise aufgebaut werden können, die Anwendern aus dem alltäglichen Erfahrungsbereich vertraut ist. Jedoch sind bestimmte Operationen wie das Erzeugen abgerundeter Kanten nur sehr schwierig darstellbar.

Die entstehende Hierarchie aus Geometrien und darauf angewendeten Mengen-Operationen (wie in Abbildung B.5 rechts zu sehen) bietet sich grundsätzlich als Datenstruktur für Aufgaben der in Kollisionserkennung an, denn durch die Konstruktionsweise bedingt gibt ein CSG-Baum unmittelbar eine Zerlegung des durch ihn konstruierten Objektes in Teil-Volumen an. Es bietet sich theoretisch an, durch die Erzeugung des Durchschnitts zweier CSG-Bäume Kontakte oder Überschneidungen zwischen den modellierten Geometrien zu ermitteln. Jedoch würde dies ein effizientes Verfahren (engl. *Null-Object Detection*, siehe etwa [Hof89] oder [RV89]) für die Bestimmung einer leeren bzw. nicht-leeren Schnittmenge zweier CSG-Bäume erfordern, welches jedoch auch beim aktuellen Stand der Forschung noch nicht zur Verfügung steht.

B.1.2.2. Anwendungen in der Kollisionserkennung

Es wurde eine Reihe von Verfahren zur Kollisionserkennung unter Zuhilfenahme von CSG-Bäumen vorgeschlagen (etwa [PZ95], [PC01], [SLY96]); diese benutzen eine CSG-Datenstruktur jedoch nicht als alleinige Datenbasis, sondern berechnen etwa wie [PC01] eine hybride Daten-Struktur aus CSG-Baum und einer Boundary Representation. CSG-Bäume werden hier verwendet, um zu bestimmen, ob Eckpunkte von Polyedern innerhalb des von einem CSG-Baum umschlossenen Volumens liegen.

Zusammenfassend ist zu sagen, dass unter anderem aufgrund des Fehlens eines effizienten Verfahrens zur Null-Object Detection, aber auch durch die nur schwierig mögliche Konvertierung einer CSG-Repräsentation in oberflächen-basierte Repräsentationen CSG-Bäume zwar prinzipiell interessante Eigenschaften für den Einsatz in hybriden Datenstrukturen aufweisen (wie in [PC01] ausgeführt), aber für sich allein gesehen keine weite Verbreitung im Bereich von Kollisionserkennungsverfahren haben. Daher werden CSG-basierte Verfahren im weiteren Verlauf der Diskussion nicht weiter berücksichtigt werden.

B.1.3. Polygonale Oberflächenbeschreibungen, Dreiecksnetze

In der Computergrafik sind dreidimensionale Körper nicht als massive Volumina, sondern als Verbunde aus Polygonen, auch *Polygon-Netze* genannt, modelliert; diese Modellierung erfasst nur die äußere Hülle eines Objektes, nicht das von ihm umschlossene Volumen selbst.

Diese Art der Geometrie-Repräsentation ist die im Bereich der Computergrafik übliche Ansatz bei der Modellierung von dreidimensionalen Objekten: Moderne Grafik-Hardware ist für die Verarbeitung polygonaler Geometrien optimiert. Dies ist durch die besondere Eignung dieser geometrischen Strukturen für Z-Puffer- (oder Raycasting-) Verarbeitung bei der Rasterisierung dreidimensionaler Szenen für die Darstellung auf zweidimensionalen Ausgabegeräten zur Lösung des Verdeckungsproblems zu erklären. Neben den gegenüber anderen Geometrie-Repräsentationen vorhandenen Vorteilen bei der Umsetzung von Verdeckungs- oder Schnittberechnungen ermöglichen polygonale Modelle, und hier im Besonderen Dreiecksnetze, die effiziente Behandlung von Beleuchtungsberechnungen, Textu-

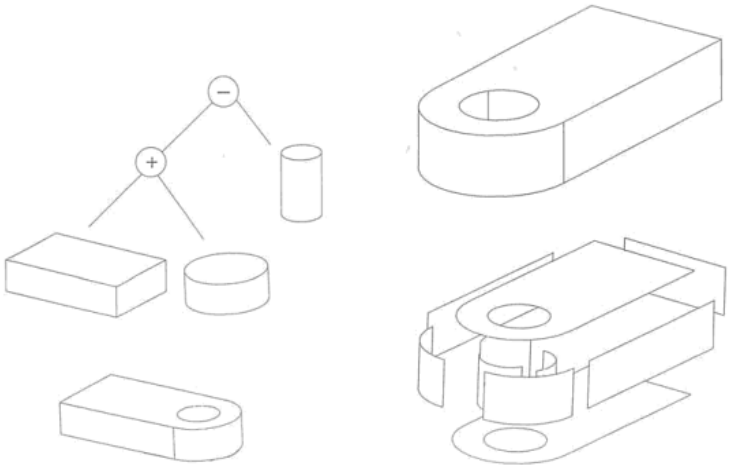


Abbildung B.5.: Ein Objekt in Boundary-Representation (rechte Skizze) und als CSG-Baum (linke Skizze)

rierung und Shader-Prozeduren, welche parallel zum Verdeckungsproblem die wichtigsten Werkzeuge für die Erzeugung realistischer oder realitätsnaher visueller Umgebungen in der 3D-Grafik darstellen.

Daraus resultiert die besondere Unterstützung dieser Art der Geometrie-Repräsentation durch die am weitesten verbreiteten Standard-Bibliotheken OpenGL und DirectX: Im Kern beider Systemen sind Polygon-Netze die wichtigste Möglichkeit zur Repräsentation von Geometrien in 3D.

B.1.3.1. Die Datenstruktur

In der elementaren Geometrie wird ein Polygon-Netz als lücken- und überlappungsfreie Partition einer Oberfläche definiert. Graphentheoretisch ist ein Polygon-Netz ein ungerichteter Graph ohne mehrfache Kanten. Die einzelne Polygone sind jeweils zyklische Teilgraphen.

Ein Eckpunkt eines Polygons wird im englischen Sprachgebrauch als *Vertex* (Plural Vertices) bezeichnet, eine Kante als *Edge*, und eine Seitenfläche als *Face* (deutsch Facette). Abbildung B.6 zeigt die Elemente eines Polygon-Netzes zusammen mit ihren Bezeichnungen.

Die Alternativen zur Spezifikation eines Polygon-Netzes unterscheiden sich hinsichtlich der Ausführlichkeit topologischer Informationen, also der Informationen über die Konnektivität einzelner Netz-Elemente, die in der jeweiligen Beschreibungs-Form hinterlegt werden. Naturgemäß steigt der Speicherbedarf eines Formats mit der Ausführlichkeit der enthaltenen Topologie-Informationen, während bei Formaten mit weniger hinterlegten Informationen der Aufwand für die Rekonstruktion der topologischen Struktur eines Polygon-Netzes durch Such- und Traversierungs-Operationen steigt.

Folgende unterschiedliche Möglichkeiten existieren, um ein Polygon-Netz zu spezifizieren:

- Als Vertex/Vertex-Liste
- Als Knotenliste oder Vertex-/Face-Liste

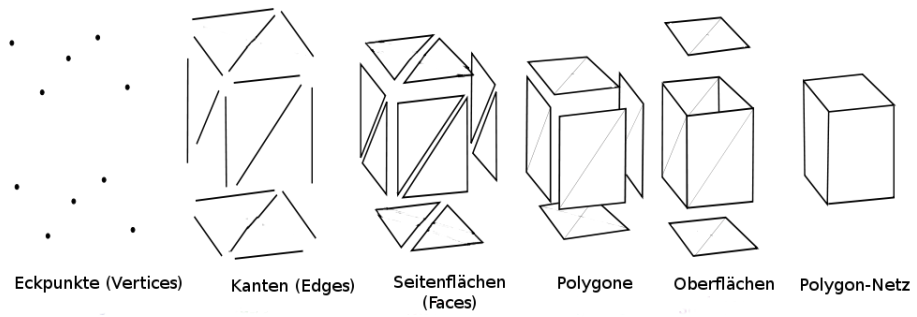


Abbildung B.6.: Elemente von Polygon-Netzen am Beispiel einer einfachen Geometrie

Format	Verweise für Abbildung eines Würfels	Direkte Traversierungs-Operationen
Face/Vertex	24	$Facette \rightarrow Eckpunkt$
Winged-Edge	192	$Facette \leftrightarrow Kante$ $Kante \leftrightarrow Eckpunkt$
Half-Edge	144	$Facette \leftrightarrow Halbkante$ $Halbkante \leftrightarrow Eckpunkt$ $Kante \leftrightarrow Halbkante$
Vertex/Vertex	24 (48 mit adjazente Kanten)	$Vertex \rightarrow Vertex$ $(Vertex \rightarrow Halbkante)$

Tabelle B.1.: Speicherbedarf und Traversierungs-Operationen verschiedener Datenformate für Polygon-Netze (nach [Smi06])

- Als Kantenliste
- Als Winged-Edge-Liste
- Als Half-Edge-Liste

Tabelle B.1 stellt die genannten Datenformate mit ihrem Speicherbedarf und den möglichen Traversierungen dar; die Angaben zum Speicherbedarf beziehen sich auf die Geometrie in Abbildung B.6.

Eine *Vertex/Vertex-Liste* erfasst neben einer Liste aller Eckpunkte für jeden Vertex eines Polygon-Netzes alle mit ihm über Kanten direkt verbundenen Vertices. Dieses Format erlaubt zwar die effiziente Ausführung von Veränderungen der Netz-Struktur und benötigt nur wenig Speicherplatz, jedoch erfasst es topologische Informationen über die strukturellen Elemente eines Netzes, die durch vollständige Suchen über alle Vertices und deren Nachbarn rekonstruiert werden müssen. Eine interessante erwähnenswerte Eigenschaft der Vertex-/Vertex-Repräsentation ist, dass sie im Gegensatz zu den anderen hier genannten Repräsentationen eine Algebra für Traversierungs-, Abfrage-, und Editier-Operationen definiert (siehe [Smi06]). Dieser Formalismus erweitert die Eignung dieses Formats für die Anwendung über Polygon-Netze hinaus, beispielsweise für die physikalische Modellierung von biologischen Wachstums-Prozessen. Für die Anwendung in der Kollisionserkennung ist

jedoch festzustellen, dass die Erfassung erweiterter topologischer Informationen Vorrang vor der effizienten Modifizierbarkeit der topologischen Struktur eines Polygon-Netzes hat. Sowohl für starre als auch verformbare Kollisionsgeometrien gilt, dass sich zwar prinzipiell die Position von Teilen eines Polygon-Netzes verändern können, nicht aber deren Konnektivität: Die Verformung eines Körpers führt in den verwendeten Modellierungen nicht zum Einfügen oder Entfernen von einzelnen Kanten oder Seitenflächen.

Winged-Edge- und *Half-Edge-Datenstrukturen* erfassen das größte Spektrum topologischer Informationen aller hier genannten Formate: Die Winged-Edge-Liste erfasst zusätzlich zu den in einer Kantenliste verfügbaren Informationen noch alle von Anfangs- und Endpunkten jeder Kante abgehenden Nachbarkanten. Half-Edge-Listen ersetzen Verweise auf von Vertices ausgehenden Kanten durch ein paar gerichteter Halb-Kanten.

Eine *Knotenliste* beinhaltet eine Liste aller Vertices eines Polygon-Netzes sowie eine Liste, die die Zugehörigkeit der Vertices zu einzelnen Facetten des Polygon-Netzes erfasst. Eine *Kantenliste* speichert im Gegensatz dazu alle Kanten eines Polygon-Netzes in einer separaten Liste, und eine Seitenfläche wird über Verweise auf die Einträge ihrer Kanten in der Kantenliste definiert. Für jede Kante werden Verweise auf Anfangs- und Endpunkt sowie die zugehörigen Facetten abgelegt.

Die Knotenliste benötigt bedingt durch die kompakte Darstellung weniger Speicherplatz als andere Formate, unterstützt jedoch durch den Verzicht auf weitergehende topologische Informationen bestimmte Arten von Suchanfragen nicht sehr effizient.

Variationen dieser beiden Darstellungs-Schemata sind die am meisten benutzten Datenstrukturen für Algorithmen in der Computergrafik und auch der Kollisionserkennung. Sie bieten einen guten Kompromiss zwischen Speicherverbrauch und effizienten Suchanfragen beim Zugriff auf beziehungsweise bei der Suche nach Netz-Elementen.

Zusätzlich ist zu bemerken, dass beim Zugriff auf einzelne Elemente eines Polygon-Netzes während der Laufzeit einer Simulation das Netz in seiner Gesamtheit nicht in jedem einzelnen Zugriff auf Netz-Elemente in seiner Gesamtheit traversiert werden muss: So ist etwa in der letzten Phase, der Berechnung von Kollisionspunkten, immer nur der Zugriff auf Paare oder kleine Gruppen von Seitenflächen aus einem Paar von Polygon-Netzen erforderlich, da weite Teile der Netz-Strukturen durch die Traversierung von Hüllkörper-Geometrien bereits zuvor aus der detaillierten Überprüfung ausgeschlossen werden können. Ähnliches gilt auch für die Adaption von Hüllkörper-Hierarchien, wie etwa im Fall von AABB-Bäumen: Die Rekonstruktion der Hüllkörperhierarchie kann so erfolgen, dass immer nur auf kleine Teile eines ganzen Polygon-Netzes zugegriffen werden muss. Hüllkörper für größere Teile eines Polygon-Netzes werden üblicherweise aus den von ihnen umschlossenen tiefer in der Hierarchie liegenden Hüllkörpern berechnet, und nicht durch eine erneute Traversierung der zugrundeliegenden Polygon-Netz-Elemente.

B.1.3.2. Anwendungen in der Kollisionserkennung

Die Fragestellung des Verdeckungsproblems, und ebenso die Funktionsweise des Raycasting, beschäftigt sich mit derselben Fragestellung wie die Kollisionserkennung: Der räumlichen Lage von Objekte zueinander. Obwohl diese Problemstellung in der Kollisionserkennung einge-

hender behandelt wird als im Bereich der Visualisierung, ist die Ähnlichkeit der elementaren Verdeckungs- beziehungsweise Schnittberechnungen doch unübersehbar.

Beispielsweise existiert eine ganze Reihe bild-basierter Kollisionserkennungsverfahren, die anhand von Verdeckungs-Berechnungen im Z-Puffer mögliche Überschneidungen zwischen polygonalen Objekten ermitteln können.

Raycasting-basierte Verfahren sind im Bereich der Kollisionserkennung zwar nicht so weit verbreitet wie Methoden, die unmittelbar auf polygonalen Oberflächen funktionieren. Die Schnittberechnung zwischen Halbgeraden und Polygonen an sich ist jedoch beispielsweise bei der Berechnung von Kollisionen unter Berücksichtigung (stückweise linearer) Objektbewegungen relevant.

Dreiecksnetze als Spezialfall von Polygon-Netzen sind im Bereich der Computergrafik und auch im Bereich der Kollisionserkennung von besonderer Bedeutung. Zusätzlich zu den bereits genannten Vorteilen im Bereich der 3D-Grafik (siehe Unterabschnitt B.1.3) existieren effiziente Verfahren zur Bestimmung von Kontakten zwischen Ecken, Kanten und Seitenflächen für Paare von Dreiecken.

B.2. Schnittberechnungen in der Objektpaar-Phase

Die folgenden beiden Abschnitte erläutern als Ergänzung zu Abschnitt 3.5 zuerst den GJK-Algorithmus als ein Beispiel für Algorithmen, die unter Nutzung spezieller Eigenschaften von Objekten (hier in Gestalt einer Beschränkung auf konvexe Geometrien) zu einer effizienten Berechnung möglicher Kontakte zwischen Geometrien in der Lage sind. Ebenso wird eine Verfahren zur exakten Berechnung von Kontaktpunkten zwischen zwei Dreiecken behandelt.

B.2.1. Der GJK-Algorithmus: Kollisionserkennungs-Verfahren für konvexe Geometrien

Konvexe Geometrien weisen gegenüber konkaven Geometrien besondere Charakteristiken auf, die von auf diesen Typ von Geometrien spezialisierte Algorithmen zur effizienten Berechnung von Kontakten ausgenutzt werden können. Zwei Vertreter solcher Algorithmen sind V-Clip [Mir98] und der Gilbert-Johnson-Keerthi (GJK)-Algorithmus [GJK88]. Vor allem der GJK-Algorithmus wird hierbei häufig von aktuellen Starrkörpersimulations-Paketen verwendet.

Der GJK-Algorithmus bestimmt das Vorliegen einer Überschneidung zwischen zwei konvexen Geometrien dadurch, indem er prüft, ob der Ursprung eines gemeinsamen Koordinatensystems in der Minkowski-Differenz der beiden Eingangsgeometrien liegt.

Die Minkowski-Summe und die Minkowski-Differenz zweier Punktmengen A und B sind definiert als:

$$A \oplus B = \{a + b | a \in A, b \in B\} \tag{B.4}$$

$$A \ominus B = A \oplus (-B) \tag{B.5}$$

Der Algorithmus bildet sogenannte Simplices aus Teilmengen der Minkowski-Differenz: Ein 0-Simplex entspricht einem Punkt, ein 1-Simplex einer Kante, ein 2-Simplex einem Dreieck und ein 3-Simplex einem Tetraeder. Simplices werden so erstellt beziehungsweise verändert, dass pro Iteration des Algorithmus ein weiterer Punkt aus der Minkowski-Differenz zu einem bestehenden Simplex hinzugefügt wird, dessen Abstand zum Ursprung des gemeinsamen Koordinatensystems geringer ist als der aller anderen Punkte aus der Minkowski-Differenz. Anders ausgedrückt liegt ein solcher Punkt in Richtung des Ursprungs am weitesten vom in der jeweiligen Iteration bestehenden Simplex entfernt.

Algorithmus 16 zeigt die Funktionsweise; ein Beispiel ist in Abbildung B.7 illustriert.

Algorithmus 16 : Der GJK-Algorithmus

Input : P, Q
// Konvexe Geometrien P und Q (in gemeinsamem Koordinatensystem, z. B. Q transformiert in lokales Koordinatensystem von P)
Wähle $s \in P \ominus Q$;
$t = max(S_c(P \ominus Q))$;
Simplex $S = t$;
while $s \neq t$ **do**
    $t = max(S_c(P \ominus Q))$;
    $S \rightarrow t$;
if $\mathbf{O} \in S$ **then**
    Überschneidung;
else
    Keine Überschneidung;

Die Abstands-Bestimmung ist Aufgabe eines Sub-Algorithmus, mit dessen Hilfe bestimmt wird, wie der bestehende Simplex durch einen neuen Punkt ergänzt werden, beziehungsweise welche Punkte aus dem bestehenden Simplex entfernt werden müssen. Dieser Sub-Algorithmus verwendet üblicherweise Verfahren aus der linearen Algebra, um das einem Punkt am nächsten liegenden Element in den verschiedenen Simplex-Typen zu bestimmen, wie in Abbildung B.8 für ein Dreieck anhand der Lage eines Punktes in einer bestimmten Voronoi-Region angedeutet.

B.2.2. Details zur Schnittberechnung zwischen zwei Dreiecken

Ein Weg, der in vielen Starrkörpersimulations-Paketen eingesetzt wird, ist eine Kombination aus sechs Ecke-Fläche- und neun Kante-Kante-Tests (Algorithmus 17): Es wird zuerst die Lage der Eckpunkte eines Dreiecks jeweils gegen das andere Dreieck geprüft (Algorithmus 18). Im Anschluß erfolgt dann eine Schnittberechnung zwischen allen möglichen Kombinationen aus Kanten-Paaren der beiden Dreiecke (Algorithmus 19).

Der Test zwischen dem Eckpunkt eines Dreiecks und dem anderen Dreieck in einem Paartest berechnet die baryzentrischen Koordinaten des getesteten Eckpunkts in Bezug auf das zweite Dreieck, und gibt den Eckpunkt beziehungsweise die Koordinaten auf den Kanten

Algorithmus 17 : TriangleTriangleIntersection

Input : $t_1, t_2, alarmDist$

1 // Zu testende Dreiecke, Toleranz-Abstand für Kontakt
2 $dist^2 = alarmDist * alarmDist$;
3 $p_1 = t_1.p_1$;
4 $p_2 = t_1.p_2$;
5 $p_3 = t_1.p_3$;
6 $p_n = t_1.normal$;
7 $q_1 = t_2.p_1$;
8 $q_2 = t_2.p_2$;
9 $q_3 = t_2.p_3$;
10 $q_n = t_2.normal$;
11 doIntersectionTrianglePoint($dist^2, q_1, q_2, q_3, q_n, p_1$);
12 doIntersectionTrianglePoint($dist^2, q_1, q_2, q_3, q_n, p_2$);
13 doIntersectionTrianglePoint($dist^2, q_1, q_2, q_3, q_n, p_3$);
14 doIntersectionTrianglePoint($dist^2, p_1, p_2, p_3, p_n, q_1$);
15 doIntersectionTrianglePoint($dist^2, p_1, p_2, p_3, p_n, q_2$);
16 doIntersectionTrianglePoint($dist^2, p_1, p_2, p_3, p_n, q_3$);
17 doIntersectionLineLine($dist^2, p_1, p_2, q_1, q_2$);
18 doIntersectionLineLine($dist^2, p_1, p_2, q_2, q_3$);
19 doIntersectionLineLine($dist^2, p_1, p_2, q_3, q_1$);
20 doIntersectionLineLine($dist^2, p_2, p_3, q_1, q_2$);
21 doIntersectionLineLine($dist^2, p_2, p_3, q_2, q_3$);
22 doIntersectionLineLine($dist^2, p_2, p_3, q_3, q_1$);
23 doIntersectionLineLine($dist^2, p_3, p_1, q_1, q_2$);
24 doIntersectionLineLine($dist^2, p_3, p_1, q_2, q_3$);
25 doIntersectionLineLine($dist^2, p_3, p_1, q_3, q_1$);

Algorithmus 18 : TriangleePointIntersection

Input : $t_1, q, alarmDist$
// Zu testendes Dreieck, Punkt, minimaler Toleranz-Abstand
Output : $closestPoint, contactNormal$
$\vec{AB} = t_1.p_2 - t_1.p_1$;
$\vec{AC} = t_1.p_3 - t_1.p_1$;
$\vec{AQ} = q - t_1.p_1$;
$A = \nvDash$;
$\vec{b} = (0, 0)$;
$A[0][0] = \vec{AB} \cdot \vec{AB}$;
$A[1][1] = \vec{AC} \cdot \vec{AC}$;
$A[0][1] = A[1][0] = \vec{AB} \cdot \vec{AC}$;
$b[0] = \vec{AQ} \cdot \vec{AB}$;
$b[1] = \vec{AQ} \cdot \vec{AC}$;
$det = \text{determinant}(A)$;
$\alpha = \frac{(b[0] \cdot A[1][1] - b[1] \cdot A[0][1])}{det}$;
$\beta = \frac{(b[1] \cdot A[0][0] - b[0] \cdot A[1][0])}{det}$;
if $\alpha < 0.000001 \vee \beta < 0.000001 \vee \alpha + \beta > 0.999999)$ **then**
    $p_{AB} = \frac{b[0]}{A[0][0]}$;
    $p_{AC} = \frac{b[1]}{A[1][1]}$;
    if $p_{AB} < 0.000001 \wedge p_{AC} < 0.0000001$ **then**
        $\alpha = 0.0$;
        $\beta = 0.0$;
    else if $p_{AB} < 0.999999 \wedge \beta < 0.000001$ **then**
        $\alpha = p_{AB}$;
        $\beta = 0.0$;
    else if $p_{AC} < 0.999999 \wedge \alpha < 0.000001$ **then**
        $\alpha = 0.0$;
        $\beta = p_{AC}$;
    else
        $p_{BC} = \frac{(b[1] - b[0] + A[0][0] - A[1][1])}{(A[0][0] + A[1][1] - 2 * A[0][1])}$;
        if $p_{BC} < 0.000001$ **then**
            $\alpha = 1.0$;
            $\beta = 0.0$;
        else if $p_{BC} > 0.999999$ **then**
            $\alpha = 0.0$;
            $\beta = 1.0$;
        else
            $\alpha = 1.0 - p_{BC}$;
            $\beta = p_{BC}$;
$\vec{p} = p_1 + \vec{AB} \cdot \alpha + \vec{AC} \cdot \beta$;
$\vec{pq} = q - \vec{p}$;
if $\|\vec{pq}\| \geqq alarmDist$ **then**
    Kein Kontakt;
else
    $closestPoint = q$;
    $contactNormal = \vec{pq}$;

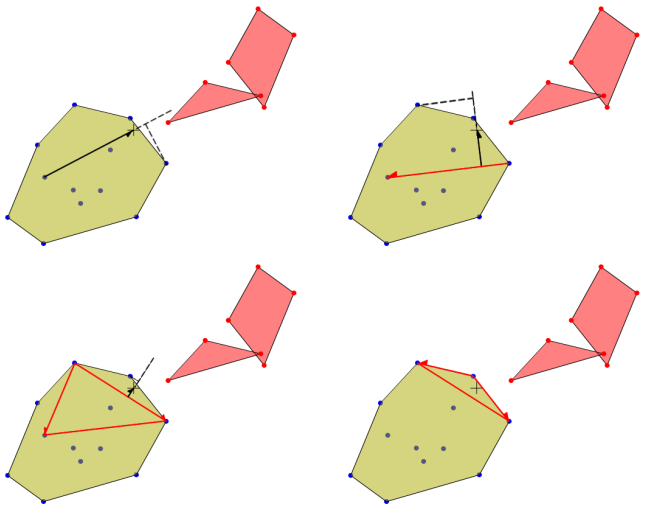


Abbildung B.7.: GJK-Algorithmus in 2D

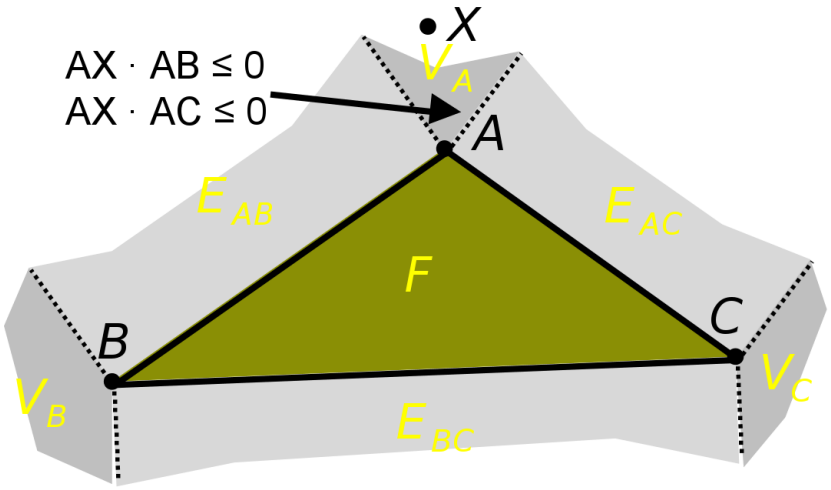


Abbildung B.8.: Nächstliegendes Element eines Dreiecks zu einem Punkt anhand der Voronoi-Regionen des Dreiecks

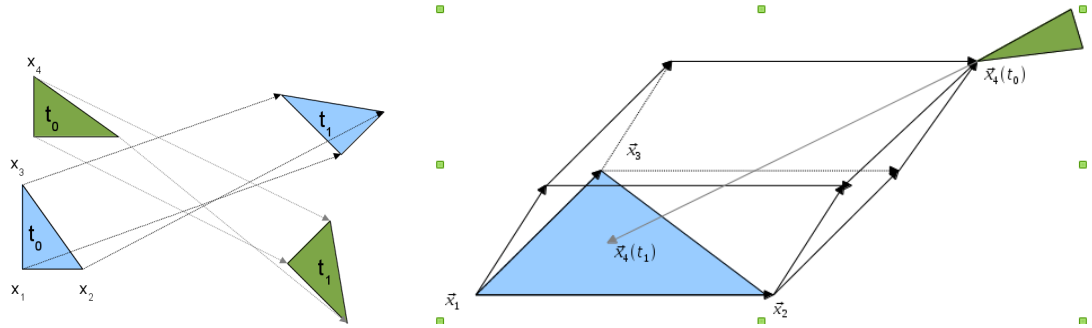
des zweiten Dreiecks an, der dem getesteten Eckpunkt des ersten Dreiecks am nächsten liegt. Ein tatsächlicher Kontakt wird nur dann erzeugt, wenn die Distanz zwischen dem getesteten Eckpunkt und dem ihm am nächsten liegenden Punkt des zweiten Dreiecks einen Toleranzbereich unterschreitet.

Der Test zwischen Paaren von Kanten zweier Dreiecke berechnet die beiden Punkte auf den jeweilgen Kanten, die den geringsten Abschnitt zueinander aufweisen. Dabei werde zwei Fälle unterschieden:

Sind die beiden Kanten (innerhalb eines Toleranzbereichs) koplanar, werden die Endpunkte der Kanten wechselweise aufeinander projiziert. Sofern die Projektion zweier Endpunkte auf die jeweils andere Kante innerhalb des Geradenabschnitts liegt, der durch die andere Kante gebildet wird, so findet sich der Punkt mit dem geringsten Abstand zur anderen Kante jeweils auf der Hälfte des Intervalls zwischen einem Endpunkt der anderen Kante und der Projektion des Eckpunkts (p und q in Abbildung B.10).

Algorithmus 19 : EdgeEdgeIntersection

```
1  // Ermittelt die zwei Punkte auf zwei Geradenabschnitten, die zueinander den geringsten Abstand haben
   Input : p_0, p_1, q_0, q_1
2  // Start- und Endpunkte der Geradenabschnitte p, q
   Output : P, Q
3  // Die einander am nächsten liegenden Punkte
4  A\vec{B} = p_1 - p_0;
5  C\vec{D} = q_1 - q_0;
6  A\vec{C} = q_0 - p_0;
7  A = 0;
8  \vec{b} = (0, 0);
9  A[0][0] = A\vec{B} \cdot A\vec{B}, A[1][1] = C\vec{D} \cdot C\vec{D};
10 A[0][1] = A[1][0] = -C\vec{D} \cdot A\vec{B};
11 b[0] = A\vec{B} \cdot A\vec{C}, b[1] = -C\vec{D} * \cdot A\vec{C};
12 det = determinant(A);
13 nr_{AB} = \|A\vec{B}\|^2, nr_{CD} = \|C\vec{D}\|^2;
14 \alpha = 0.5, \beta = 0.5;
15 if det \neq 0 then
16     \alpha = \frac{(b[0] \cdot A[1][1] - b[1] \cdot A[0][1])}{det};
17     \beta = \frac{(b[1] \cdot A[0][0] - b[0] \cdot A[1][0])}{det};
18 else
19     A\vec{D} = q_1 - p_0, C\vec{B} = p_1 - q_0;
20     c_{proj} = \frac{b[0]}{nr_{AB}};
21     d_{proj} = \frac{(A\vec{B} \cdot A\vec{D})}{nr_{AB}};
22     a_{proj} = \frac{b[1]}{nr_{CD}};
23     b_{proj} = \frac{(C\vec{D} \cdot C\vec{B})}{nr_{CD}};
24     if c_{proj} \geq 0 \wedge c_{proj} \leq 1 then
25         if d_{proj} > 1 then
26             \alpha = \frac{(1.0 + c_{proj})}{2}, \beta = \frac{b_{proj}}{2};
27         else if d_{proj} < 0 then
28             \alpha = \frac{c_{proj}}{2}, \beta = \frac{(1 + a_{proj})}{2};
29         else
30             \alpha = \frac{(c_{proj} + d_{proj})}{2}, \beta = 0.5;
31     else if d_{proj} \geq 0 \wedge d_{proj} \leq 1 then
32         if c_{proj} < 0 then
33             \alpha = \frac{d_{proj}}{2}, \beta = \frac{(1 + a_proj)}{2};
34         else
35             \alpha = \frac{(1 + d_{proj})}{2}, \beta = \frac{b_{proj}}{2};
36     else
37         if c_{proj} \cdot d_{proj} < 0 then
38             \alpha = 0.5, \beta = \frac{(a_{proj} + b_{proj})}{2};
39         else
40             if c_{proj} < 0 then
41                 \alpha = 0;
42             else
43                 \alpha = 1;
44             if a_{proj} < 0 then
45                 \beta = 0;
46             else
47                 \beta = 1;
48     P = p_0 + A\vec{B} \cdot \alpha;
49     Q = q_0 + C\vec{D} \cdot \beta;
50     return;
51 if \alpha < 0 then
52     \alpha = 0;
53     \beta = \frac{(C\vec{D} \cdot (p_0 - q_0))}{nr_{CD}};
54 else if \alpha > 1 then
55     \alpha = 1;
56     \beta = \frac{(C\vec{D} \cdot (p_1 - q_0))}{nr_{CD}};
57 if \beta < 0 then
58     \beta = 0;
59     \alpha = \frac{(A\vec{B} \cdot (q_0 - p_0))}{nr_{AB}};
60 else if \beta > 1 then
61     \beta = 1;
62     \alpha = \frac{(A\vec{B} \cdot (q_1 - p_0))}{nr_{AB}};
63 if \alpha < 0 then
64     \alpha = 0;
65 else if \alpha > 1 then
66     \alpha = 1;
67 P = p_0 + A\vec{B} \cdot \alpha;
68 Q = q_0 + C\vec{D} \cdot \beta;
```



(a) Test unter Berücksichtigung von Objektbewegung

(b) Test Eckpunkt gegen Fläche

Abbildung B.9.: Objektpaar-Test für Dreiecke

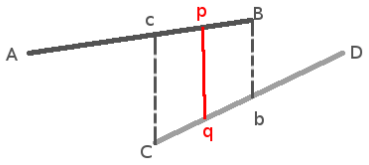


Abbildung B.10.: Kante-Kante-Kontakt: Ko-planarer Fall

Andernfalls wird das Paar von Punkten mit dem geringsten Abstand zueinander wie in Abbildung B.11 skizziert berechnet. Mit den Größen aus Abbildung B.11 lässt sich ein Gleichungssystem zur Ermittlung der Faktoren s und t (entsprechen α und β in Algorithmus 19) aufstellen:

$$\begin{pmatrix} a & -b \\ c & -e \end{pmatrix} \cdot \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} -c \\ -f \end{pmatrix}, a = d_1 \cdot d_1, b = d_1 \cdot d_2, c = d_1 \cdot r, e = d_2 \cdot d_2, f = d_2 \cdot r, d = ae - b^2 \tag{B.6}$$

Etwa mit Hilfe der Regel von Cramer ergeben sich s und t zu:

$$s = \frac{(bf - ce)}{d} \tag{B.7}$$

$$t = \frac{(af - bc)}{d} \tag{B.8}$$

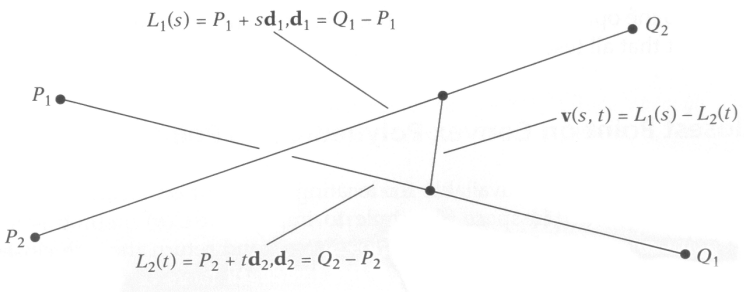


Abbildung B.11.: Kante-Kante-Kontakt: Genereller Fall

B.3. Bildraum-basierte Verfahren

Eine andere Möglichkeit, Grafik-Prozessoren für Kollisionserkennungs-Verfahren zu verwenden, ist die Nutzung bildraum-basierter Techniken: Anstatt auf Basis polygonaler Geometriebeschreibungen in Kombination mit Hüllkörper-Hierarchien zu arbeiten (wie in Abschnitt 4.2), können sich potentiell berührende oder überschneidende Geometrien daraufhin untersucht werden:

- Wie sie sich aus unterschiedlichen Positionen gesehen gegenseitig verdecken
- Ob und wie sich ein Schnitt-Volumen zwischen ihnen ergibt

Bildraum-basierte Verfahren sind für jegliche Art rasterisierbarer Geometrien geeignet; sie unterstützen daher neben statischen auch verformbare und unstrukturierte Dreiecks-Netze, aber auch NURBS- und Bezier-Patches (Abschnitt B.1), und funktionieren auch ohne zusätzliche geometrische Datenstrukturen wie Hüllkörper-Hierarchien.

Ihre Schwachstelle besteht allerdings darin, dass es sich nur um approximierende Verfahren handelt: Ihre geometrische Genauigkeit ist durch die Auflösung der Bildpuffer, die zur Rasterisierung der verwendeten Geometrien verwendet werden, beschränkt. Das hat beispielsweise zur Folge, dass Kollisionen zwischen Objekten nicht oder fälschlicherweise erkannt werden: Etwa wenn sehr kleine Objekte weit von der verwendeten Kamera-Position entfernt liegen und daher bei der angewendeten perspektivischen Projektion auf wenige oder sogar nur ein Pixel im Bildpuffer abgebildet werden. Auch entstehen Probleme mit Polygonen, deren Flächen-Normale nahezu rechte Winkel zur Blickrichtung der Kamera einnehmen, und die als „auf einer Kante stehend“ nicht im erzeugten Bild erscheinen. Verfahren, die verdeckungsbasiert arbeiten, sind gegenüber solchen Objekt-Konfigurationen empfindlicher als Verfahren, die mögliche Schnitt-Volumina berechnen. Für beide Arten von bildraum-basierten Verfahren gilt jedoch die genannte Einschränkung hinsichtlich der Auflösung der verwendeten Bildpuffer.

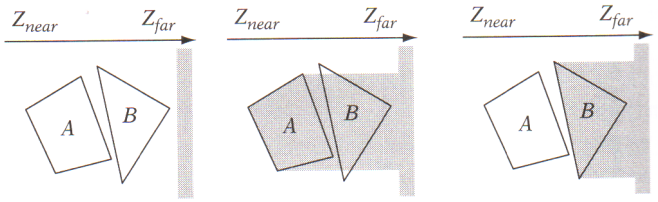
Da die möglichst präzise Berechnung möglicher Kollisionen das Hauptanliegen der vorliegenden Arbeit ist, sollen bildraum-basierte Verfahren nur anhand zweier Beispiele vorgestellt werden:

- CULLIDE ([GRLM03], [GLM05], [GKLM07]) als Vertreter eines verdeckungs-basierten Ansatzes (Unterabschnitt B.3.1)
- Ein auf Layered Depth Images ([SGHS98]) basierendes Verfahren ([TKH+04], [ZH07], [FBJF08]) als Vertreter schnittvolumen-basierter Verfahren (Unterabschnitt B.3.2)

B.3.1. Mittels Verdeckungs-Berechnung

Das Prinzip der Verdeckungs-Berechnung ist in Abbildung B.12 (für konvexe Objekte) zu sehen: Zwei Objekte werden (zuerst mit aktiviertem Z-Puffer und dann mit aktiviertem Verdeckungs-Test) nacheinander rasterisiert. Sind nach dem Rasterisieren des zweiten Objekts keine zum ersten Objekt gehörenden Pixel im Z-Puffer verzeichnet, so verdeckt das zweite Objekt dieses vollständig und liegt damit vor dem ersten Objekt. Sollte dies nicht der Fall sein (Abbildung B.12b), wird ein zweiter Rasterisierungsvorgang mit der gleichen Vorgehensweise, aber vertauschter Objekt-Reihenfolge durchgeführt (Abbildung B.12c). Nur

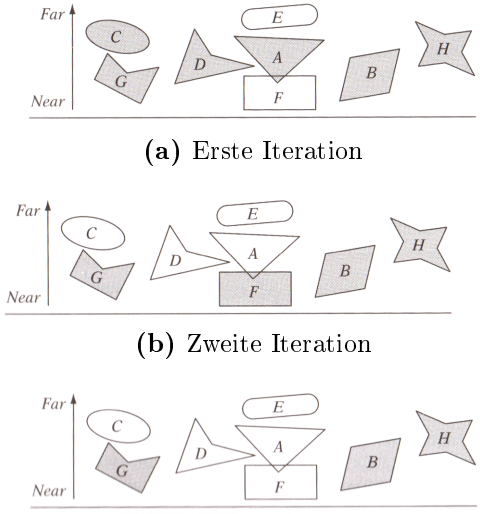
wenn in beiden Rasterisierungs-Durchgängen jeweils eine teilweise Verdeckung festgestellt wird, liegt eine potentielle Überschneidung zwischen den geprüften Objekten vor.



(a) Initialisierung **(b)** Erste Rasterisierung **(c)** Zweite Rasterisierung

Abbildung B.12.: Bestimmen gegenseitiger Verdeckung mittels Z-Puffer ([Eri05, Kapitel 13])

CULLIDE erweitert dieses Prinzip über ein Objekt-Paar hinaus. Das Verfahren ermittelt eine potentiell kollidierende Teilmenge (engl. Potentially Colliding Set, PCS) aller Objekte $S = O_1, O_2, \ldots, O_n$ in einer Szene. Dazu wird wiederum eine zweistufige Rasterisierung der Szene durchgeführt, indem einmal alle Objekte in der Reihenfolge $O_1, O_2, \ldots, O_n$ (Abbildung B.13a) und dann in der Reihenfolge $O_n, O_{n-1}, \ldots, 0_1$ (Abbildung B.13b) rasterisiert werden, und dabei deren Sichtbarkeit gegenüber allen anderen Objekten wie im zuvor beschriebenen Paar-Test festgehalten wird. Ein Objekt O_i, das gegenüber $O_1, \ldots, O_{i-1}$ und $O_{i+1}, \ldots, O_n$ jeweils vollständig sichtbar ist, kann von weiteren Kollisions-Tests ausgeschlossen werden. Dies gilt ebenso für alle vor (in der ersten Rasterisierung) beziehungsweise nach (in der zweiten Rasterisierung) einem als voll sichtbar eingestuften Objekt ebenfalls als voll sichtbar erkannte Objekte.



(a) Erste Iteration

(b) Zweite Iteration

(c) Potentiell kollidierende Objekte

Abbildung B.13.: Funktionsweise von CULLIDE ([Eri05, Kapitel 13]); voll sichtbare Objekte sind schattiert

Indem dieses Verfahren in mehreren Iterationen beziehungsweise aus verschiedenen Blickrichtungen auf eine Szene durchgeführt wird, kann das PCS immer weiter eingeschränkt werden. Dabei ist das Verfahren sowohl auf eine gesamte Szene in einer Vorfilter-Phase als auch in einer Objektpaar-Phase anwendbar, wenn die einzelnen Dreiecke von Dreiecks-Netzen selbst

als individuelle Geometrien betrachtet werden. In dieser Interpretation ermittelt CULLIDE alle möglicherweise kollidierenden Dreiecks-Paare, für die dann noch Paar-Tests zur exakten Bestimmung von Kontaktpunkten durchgeführt werden müssen.

Obwohl die eingangs des Abschnitts erwähnten Vorteile auch für CULLIDE gelten, so hat das Verfahren neben der Beschränkung durch die Auflösung der verwendeten Bildpuffer noch eine weitere Schwachstelle: Die Anzahl benötigter Rasterisierungs-Durchläufe. Für jedes überprüfte Objekt muss die betrachtete Szene (oder ein Ausschnitt) zweimal neu rasterisiert werden. Obwohl dabei Beleuchtungseigenschaften und Texturierung deaktiviert sind, stellt dies trotzdem hohe Leistungsanforderungen an die verwendete Grafik-Hardware. Zwar kann die Anzahl nötiger Rasterisierungs-Durchläufe durch eine Hierarchisierung der Szene (ähnlich wie bei Hüllkörper-Hierarchien) vermindert werden, indem beispielsweise zuerst die AABBs von Objekten rasterisiert werden, bevor mit der Überprüfung einzelner Objekt-Teile oder Seitenflächen begonnen wird, jedoch führt dies trotzdem zu einem ähnlichen Sachverhalt, wie er auch bei der Analyse des Verhaltens polygon-basierter Verfahren hinsichtlich der Laufzeitanteile der einzelnen Teilschritte der Algorithmen zu beobachten war: So wie die meisten der untersuchten polygon-basierten Verfahren einen wesentlichen Anteil ihrer gesamten Laufzeit pro Iteration in der Traversierungs-Phase verbrauchen, ist dies bei Verfahren wie CULLIDE für die benötigten Rasterisierungen einer Szene der Fall.

B.3.2. Mittels Layered Depth Images

Diese Art von Verfahren ermittelt das Schnitt-Volumen zwischen Objekten unter Zuhilfenahme sogenannter Layered Depth Images (LDI, Abbildung B.14), einer geschichteten Rasterisierung eines Objekts, mittels der das Innere eines Objekts durch die Suche nach den Übergängen zwischen Objekt-Äußerem und -Innerem approximiert werden kann. Dazu werden sogenannte Stencil-Puffer verwendet, in denen bei der Rasterisierung nur diejenigen Teile eines Objekts registriert werden, die bis zu einer bestimmten Tiefe in z-Richtung sichtbar sind. Durch mehrmaliges Rasterisieren entsteht so eine geschichtete Diskretisierung eines Objekts entlang einer Sichtachse, wie es schon der englischsprachige Name der verwendeten Datenstruktur ausdrückt.

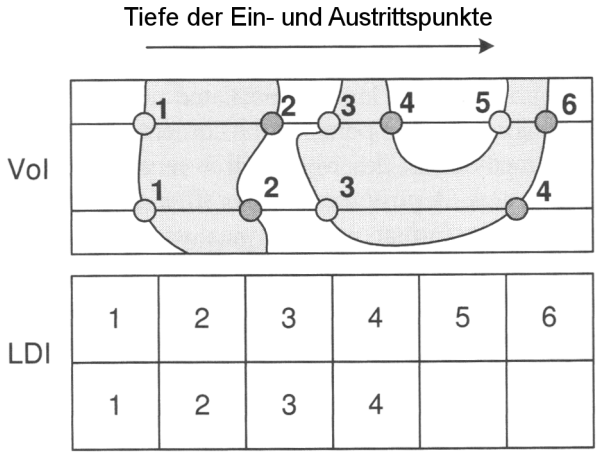
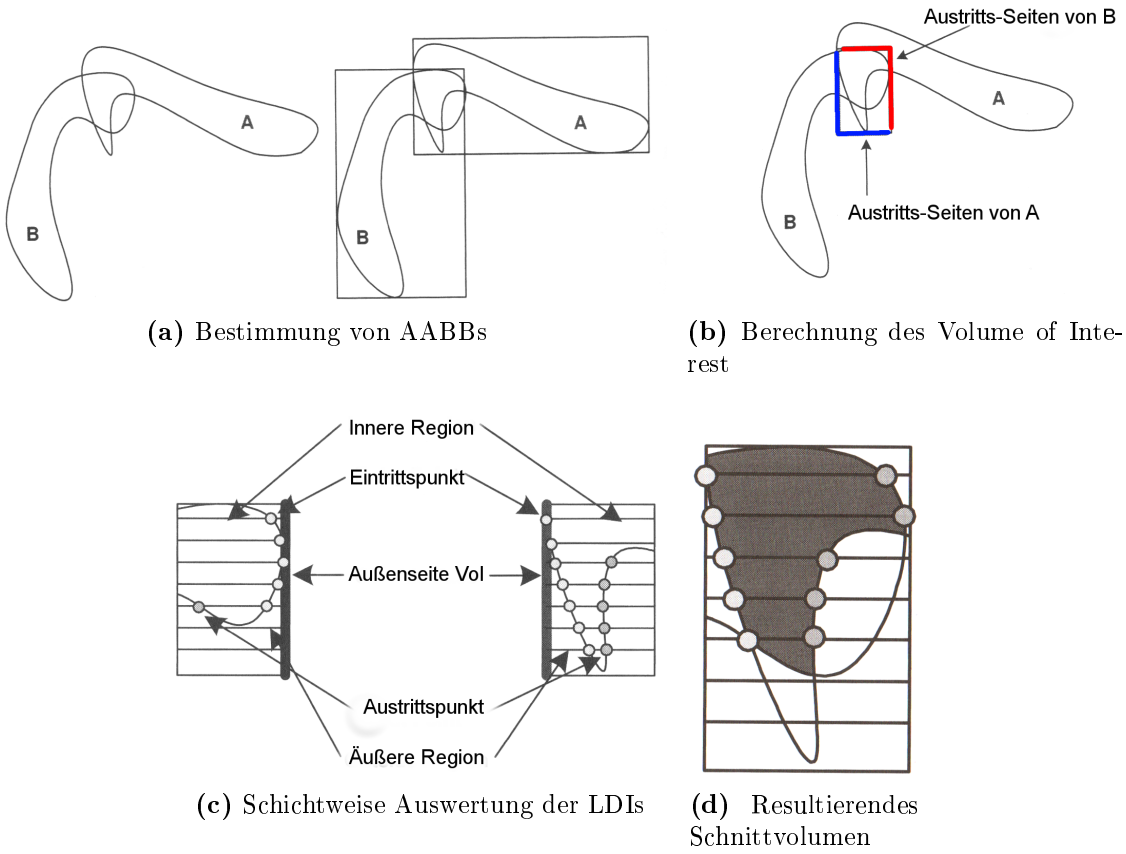


Abbildung B.14.: Die Struktur eines Layered Depth Image (LDI, nach [Erl05, Kapitel 17])

Die Kollisionserkennung mittels LDIs funktioniert wiederum zweistufig: Zuerst werden anhand der AABBs von Objekt-Paaren mögliche Schnitt-Volumina bestimmt (Abbildung B.15a). Sofern eine Überschneidung der AABBs vorliegt, bestimmt diese die Größe des zu berechnenden Schnittvolumen-Bereichs, und damit die Größe der für beide Objekte zu erstellenden LDIs (Abbildung B.15b). Dann wird für jedes Objekt ein separates LDI erstellt (Abbildung B.15c). Zur Bestimmung des Schnitt-Volumens pro Schicht in einem fusionierten LDI (Abbildung B.15d) wird dazu wechselseitig für jeden Ein-und Austrittspunkt im LDI eines Objekts im LDI des zweiten Objekts überprüft, ob dieser in der entsprechenden gleichen Schicht des anderen LDI innerhalb des Intervalls zwischen einem Ein- und Austrittspunkt liegt. Ist dies der Fall, so liegt eine Überschneidung vor, und die Tiefe der Durchdringung zwischen den beteiligten Objekten kann anhand der Intervall-Länge angenähert bestimmt werden.



(a) Bestimmung von AABBs

(b) Berechnung des Volume of Interest

(c) Schichtweise Auswertung der LDIs

(d) Resultierendes Schnittvolumen

Abbildung B.15.: Die einzelnen Schritte eines LDI-basierten Verfahrens ([Erl05, Kapitel 17])

Wie bei CULLIDE ist es notwendig, eine Szene beziehungsweise ein Objektpaar aus verschiedenen (zueinander orthogonalen) Perspektiven mittels LDIs zu rasterisieren, um eine Abschätzung der Schnitt-Intervalle entlang aller drei Raumrichtungen zu erhalten, und daraus ein angenähertes Schnitt-Volumen berechnen zu können.

Für LDI-basierte Verfahren gilt neben den Einschränkungen, die bereits im Zusammenhang mit CULLIDE zu beobachten waren, dass die verwendeten Objekte eine komplett geschlossene Hülle aufweisen müssen, da sonst die Unterscheidung zwischen Objekt-Äußerem und -Innerem nicht mehr zuverlässig erfolgen kann.

B.3.3. Diskussion

Obwohl bildraum-basierte Verfahren hinsichtlich der Unterstützung verformbarer und sogar variabler Geometrien (deren Topologie sich zur Laufzeit verändert) und durch den zumindest teilweise möglichen Verzicht auf umfangreiche Hüllkörper-Hierarchien eine sehr vielversprechende Herangehensweise sind, ist die Beschränkung der möglichen Präzision bei der Kollisionserkennung durch die zur Verfügung stehende Auflösung von Bildpuffern, in denen die Rasterisierung von einzelnen (oder Gruppen von) Primitiven zur Verdeckungs-Berechnung erfolgt, eine enorme Restriktion. Das in Kapitel 5 vorgestellte eigene Kollisionserkennungs-Verfahren hat zum Ziel, wie polygon-basierte Verfahren auf Basis von polygonalen Geometriebeschreibungen zu arbeiten und prinzipiell ohne verfahrens-immanente Beschränkungen funktionieren zu können.

ANHANG C

LEISTUNGSFÄHIGKEIT NUMERISCHER BERECHNUNGEN AUF CPU- UND GPU-PROZESSOREN

Ein Schwerpunkt der Arbeit sind GPU-basierte Verfahren zur Kollisionserkennung. Eine kurze Analyse der technischen Besonderheiten und Vorteile von Grafik-Prozessoren ist daher für das Verständnis der in Kapitel 4 und Kapitel 5 behandelten GPU-basierten Verfahren zur Kollisionserkennung hilfreich.

Es folgt im nächsten Abschnitt C.1 ein allgemeiner Überblick über den allgemeinen Aufbau einer GPU sowie die zugehörigen Programmiersprache und -schnittstellen.

Um die Vorteile von GPU-Prozessoren gegenüber konventionellen CPU-Implementierungen zu verdeutlichen, soll im übernächsten Abschnitt C.2 ein Problem aus der klassischen Mechanik betrachtet werden: Das Vielkörper-Problem.

C.1. Architektur von GPU-Prozessoren

Im Gegensatz zu konventionellen CPU-Architekturen, die auf die Minimierung von Latenz-Zeiten sowohl bei Speicherzugriffen als auch bei der Ausführung von Operationen hin optimiert sind, ist bei GPU-Prozessoren das Erzielen einer hohen Durchsatzleistung das wichtigste Kriterium.

Die Steuerwerke der meisten modernen Prozessoren beinhalten beispielsweise Mechanismen, um Assembler-Instruktionen neu anzuordnen oder parallel auszuführen, oder aber auch spekulativ mit der Ausführung von Schleifen fortzufahren, ohne die Überprüfung etwaiger vorhandener Abbruchbedingungen im ablaufenden Programm abzuwarten. Diese Maßnahmen dienen dazu, Verzögerungen innerhalb eines in einem CPU-Kern ablaufenden Threads zu minimieren. GPU-Architekturen verzichten weitgehend auf solche komplexen Maßnahmen in

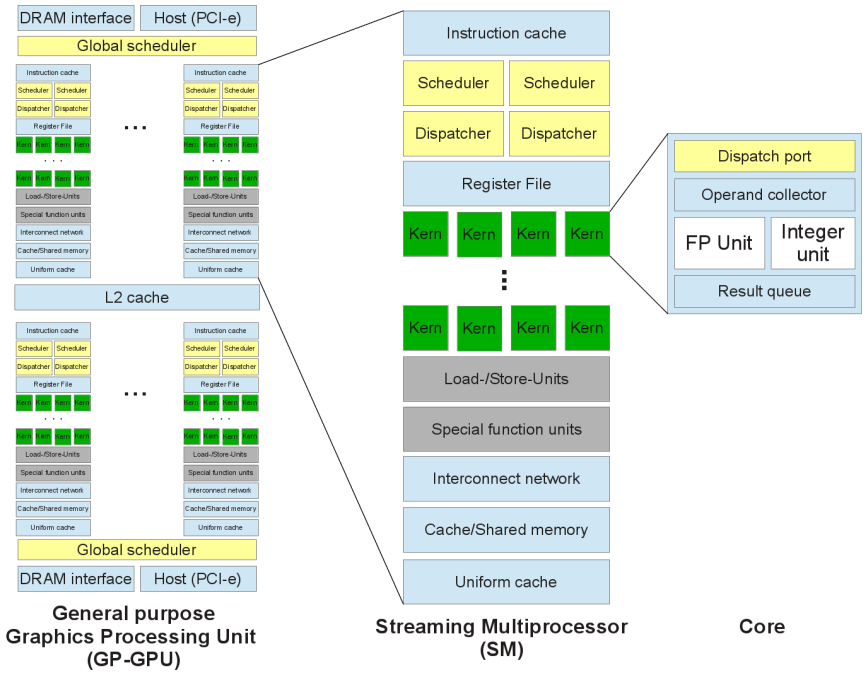


Abbildung C.1.: Schematische Darstellung der Hardware-Architektur einer GPU

den Steuerwerken eines Streaming-Prozessors. Anstelle der Optimierung der Ausführungsgeschwindigkeit pro Kern tritt eine massiv parallele Ausführung von identischen Instruktionen auf einer gegenüber konventionellen Prozessoren vielfach höheren Anzahl von Rechenwerken. Dadurch sind GPU-Prozessoren weniger anfällig für Verzögerungen bei Speicherzugriffen und länger laufenden Instruktionen in einzelnen GPU-Kernen.

In Abbildung C.1 ist der schematische Aufbau einer GPU skizziert: Eine GPU ist aus einer Reihe von Prozessor-Einheiten aufgebaut (engl. „Streaming Multiprocessors“), von denen jeder neben den eigentlichen Prozessor-Kernen (engl. „Cores“, 32 bei der Fermi-Architektur von NVidia) unter anderem eigene Scheduling- und Cache-Einheiten besitzt. Jeder Prozessor-Kern enthält wiederum die eigentlichen ALU- und Fließkomma-Einheiten.

Die in Abbildung C.1 skizzierte Hardware-Architektur spiegelt den Grundgedanken der massiv parallelen Ausführung von Programmabläufen wider: Jeder Kern in einem Streaming-Prozessor führt den gleichen Programm-Code aus, nur auf unterschiedlichen Bereichen der gegebenen Eingangsdaten.

Diese Funktionsweise liegt auch den Programmiersprachen-Erweiterungen durch CUDA zugrunde: Auf einem Grafik-Prozessor ausgeführter Programm-Code wird mittels spezieller Aufrufe aus dem auf der CPU ausgeführten Hauptprogramm heraus gestartet. Mittels des Aufrufs einer CUDA-Funktion wird die Anzahl der zu benutzenden Ausführungseinheiten auf der GPU festgelegt. Die GPU führt dann den aufgerufenen Kernel auf einer entsprechenden Anzahl von Prozessoren aus, wobei das Scheduling beziehungsweise die Ausführungsreihenfolge für den Anwender transparent erfolgt.

Die Architektur von Streaming-Prozessoren hat unmittelbare Auswirkungen auf die Programmierschnittstellen und Spracherweiterungen, mit deren Hilfe Programme für den Einsatz auf GPUs erstellt werden.

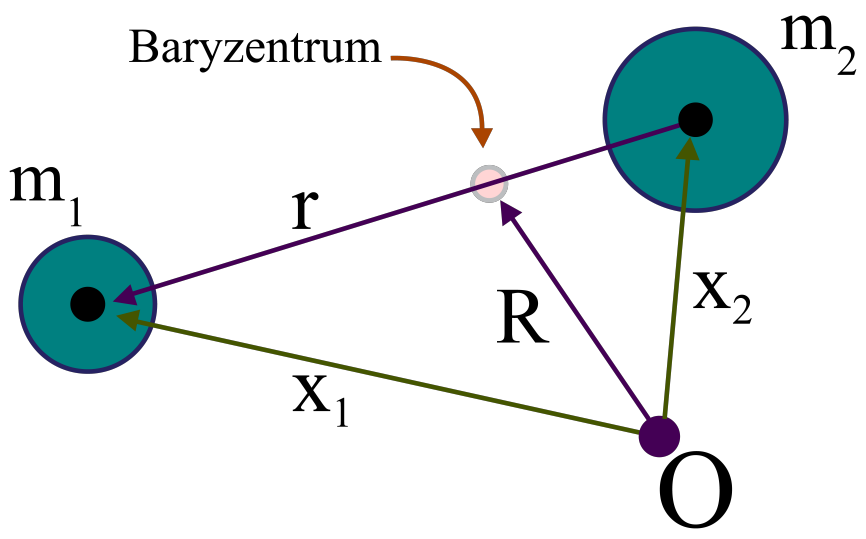


Abbildung C.2.: Schematische Darstellung des Zweikörper-Problems

C.2. Iterative Mehrkörpersimulation auf GPUs

Das Vielkörper-Problem betrachtet die Bewegung dreier oder mehrerer Körper unter wechselseitigem gravitativem Einfluss. Für zwei Körper ergeben sich die wechselseitig aufeinander ausgeübten Kräfte (mit den Größen aus Abbildung C.2) zu

$$\vec{F}_{12}(\vec{x}_1, \vec{x}_2) = -Gm_1m_2\frac{\vec{x}_1 - \vec{x}_2}{\|\vec{x}_1 - \vec{x}_2\|^3}$$
$$\vec{F}_{21}(\vec{x}_1, \vec{x}_2) = -Gm_1m_2\frac{\vec{x}_2 - \vec{x}_1}{\|\vec{x}_1 - \vec{x}_2\|^3}$$

mit Positionsveränderungen pro Zeitintervall als

$$\vec{x}_1 = \vec{R} + \frac{m_2}{m_1 + m_2}\vec{r}$$
$$\vec{x}_2 = \vec{R} + \frac{m_1}{m_1 + m_2}\vec{r}$$

Für drei oder mehr Körper ist es nicht möglich, eine algebraische Lösung für dieses Problem zu formulieren ([Poi92, Kapitel 1]). Vielmehr ist die Verwendung numerischer Ansätze wie einer iterativen Mehrkörpersimulation erforderlich, um eine näherungsweise Lösung berechnen zu können ([Sti13], [Vla13]).

Das im Folgenden vorgestellte Beispiel wählt hierzu einen sehr einfachen Einsatz für die Berechnung der pro Körper wirkenden Gravitationskräfte und die Aktualisierung der Geschwindigkeit bzw. Position jedes Körpers nach einem (diskreten) Zeitschritt der Simulation: Die gesamte auf einen Körper einwirkende Kraft wird als Summe der von allen anderen Körpern in der Simulation ausgehenden Gravitationskräfte berechnet, und die Aktualisie-

rung der Geschwindigkeit eines Körpers erfolgt durch eine einfache lineare Approximation innerhalb eines Zeitschrittes als Multiplikation der einwirkenden Kraft mit der Länge des Zeitschrittes.

Algorithmus 20 : Positions- und Geschwindigkeits-Aktualisierung eines einzelnen Körpers: CPU-Implementierung

```
1  // Aktualisierung des Partikels i
   Input : i, dt, nParticles
2  #define SOFTENING 1e^-10
3  Fx = 0.0f, Fy = 0.0f, Fz = 0.0f;
4  x = p.x[i], y = p.y[i], z = p.z[i];
5  #pragma simd
6  #pragma vector aligned
7  for j = 0 → nParticles do
8      dx = p.x[j] − x;
9      dy = p.y[j] − y;
10     dz = p.z[j] − z;
11     drSquared = dx ∗ dx + dy ∗ dy + dz ∗ dz + SOFTENING;
12     drPowerN32 = 1.0f/(drSquared ∗ sqrtf(drSquared));
13     Fx+ = dx ∗ drPowerN32;
14     Fy+ = dy ∗ drPowerN32;
15     Fz+ = dz ∗ drPowerN32;
16 p.vx[i]+ = Fx ∗ dt;
17 p.vy[i]+ = Fy ∗ dt;
18 p.vz[i]+ = Fz ∗ dt;
```

Algorithmus 20 zeigt die Implementierung dieser einfachen numerischen Integration für konventionelle Prozessoren, ergänzt um OpenMP-spezifische Compiler-Direktiven zur compilerunterstützten Integration paralleler Abläufe bereits während der Übersetzungs-Zeit.

Algorithmus 21 zeigt dieselbe numerische Integration als CUDA-Programm implementiert.

Beiden Implementierungen ist die Datenstruktur gemeinsam, in denen Position und Geschwindigkeit jedes Körpers in der Simulation abgelegt sind. Um Verzögerungen bei Speicherzugriffen zu vermeiden und einem Compiler die Optimierung der Assembler-Anweisungen im erzeugten Programm zu erleichtern, wurde hier eine Variante gewählt, die die jeweiligen Vektor-Komponenten in an einem Stück allokierten Speicherbereichen ablegt, und nicht in einer Datenstruktur mit entsprechenden Variablen, die für jeden Körper separat instantiiert und damit für eine Fragmentierung der in den Schleifen zur Summierung der einwirkenden Kräfte auszulesenden Werte im Speicher sorgen würde.

Eine weitere Optimierung, die beiden gezeigten Implementierungen gemeinsam ist, stellt die Addition einer sehr klein gewählten Fließkommazahl ($SOFTENING$) im Term, der den quadratischen Abstand zweier Körper berechnet. Die Überlegung hierbei ist, eine Überprüfung zu vermeiden, die normalerweise nötig wäre, um die Einbeziehung des aktuell betrachteten Körpers selbst in die Summierung aller einwirkenden Kräfte zu vermeiden ($i = j$). Diese würde einen Compiler normalerweise dazu veranlassen, eine Zerlegung des Schleifen-Durchlaufs

in einen Bereich $i < j$ und $i > j$ durchzuführen, was den nötigen Laufzeitaufwand für die Verarbeitung einer Iteration erheblich erhöhen würde. Durch die Addition eines kleinen konstanten Terms wird hier verhindert, dass in der nachfolgenden Berechnung der Inversen des Abstands-Terms eine Division durch Null erfolgt. Diese künstliche „Verfälschung“ spielt in der anschließenden Integration keine Rolle, da die Distanz eines Körpers zu sich selbst 0 beträgt und daher die Aktualisierung „mit sich selbst“ vermieden wird.

Dies sind bereits zwei Beispiele dafür, wie durch das geschickte Ausnutzen numerischer Besonderheiten und die Optimierung der Anordnung der Eingangsdaten im Speicher dafür gesorgt werden kann, dass verzögerten Speicherzugriffen und compiler-spezifischen Problemen bei der Parallelisierung von Ausführungs-Pfaden vorgebeugt werden kann.

Algorithmus 21 : Positions- und Geschwindigkeits-Aktualisierung eines einzelnen Körpers: Implementierung als CUDA-Kernel

```
1  #define SOFTENING 1e^-10
   Input : pos, vel, dt, nParticles
2  i = blockDim.x + blockIdx.x + threadIdx.x;
3  if i < nParticles then
4      iPos = pos[i];
5      F = {0.0f, 0.0f, 0.0f};
6      for tile = 0 → gridDim.x do
7          shared spos[BLOCK_SIZE];
8          spos[threadIdx.x] = pos[tile * blockDim − X + threadIdx.x];
9          syncthreads();
10         #pragma unroll
11         for j = 0 → BLOCK_SIZE do
12             dx = spos[j].x − iPos.x;
13             dy = spos[j].y − iPos.y;
14             dz = spos[j].z − iPos.z;
15             drSquared = dx * dx + dy * dy + dz * dz + SOFTENING;
16             drPowerN32 = rsqrtf(drSquared);
17             drPowerN32 = drPowerN32 * drPowerN32 * drPowerN32;
18             F.x = dx * drPowerN32;
19             F.y = dy * drPowerN32;
20             F.z = dz * drPowerN32;
21         syncthreads();
22     vel[i].x+ = dt * F.x;
23     vel[i].y+ = dt * F.y;
24     vel[i].z+ = dt * F.z;
```

Einer der wesentlichen Unterschiede in der CUDA-Implementierung ist in Zeile 2 in Algorithmus 21 zu sehen: Während bei der konventionellen Implementierung der Index des zu aktualisierenden Körpers explizit (in einer nicht gezeigten äußeren Schleife) als Parameter an die Aktualisierungs-Routine übergeben werden muss, ergibt sich dieser im Fall der CUDA-Variante implizit durch die Zuweisung eines Elementes aus den Eingabedaten

an einen bestimmten Kern innerhalb eines Streaming-Prozessors. Die Variablen *blockDim*, *blockIdx* und *threadIdx* sind CUDA-spezifische Spracherweiterungen, die zur Laufzeit eines CUDA-Programms durch die Ausführungsumgebung bereitgestellt werden.

Die angesprochene Index-Berechnung zeigt sehr gut, wie das Zusammenspiel zwischen Hard- und Software bei GPU-unterstützten Berechnungen funktioniert: Anstatt die Ausführungs-Reihenfolge und Scheduling explizit gestalten zu müssen, wie es typischerweise beim Umgang mit Nebenläufigkeit bei konventionellen Prozessoren der Fall ist, werden diese Aufgaben mit Unterstützung der Hardware vorgenommen beziehungsweise durch die Programmierschnittstelle abstrahiert.

Die grundlegende Ausführungs-Einheit auf einer GPU ist ein *Thread*. Die von einem Thread ausgeführten Berechnungen werden im Quellcode eines *Kernels* spezifiziert. Die Ausführung eines Kernels erfolgt in einer Gruppe von Threads, wobei jeder Thread wie bereits angedeutet einen anderen Bereich der Eingabedaten bearbeitet.

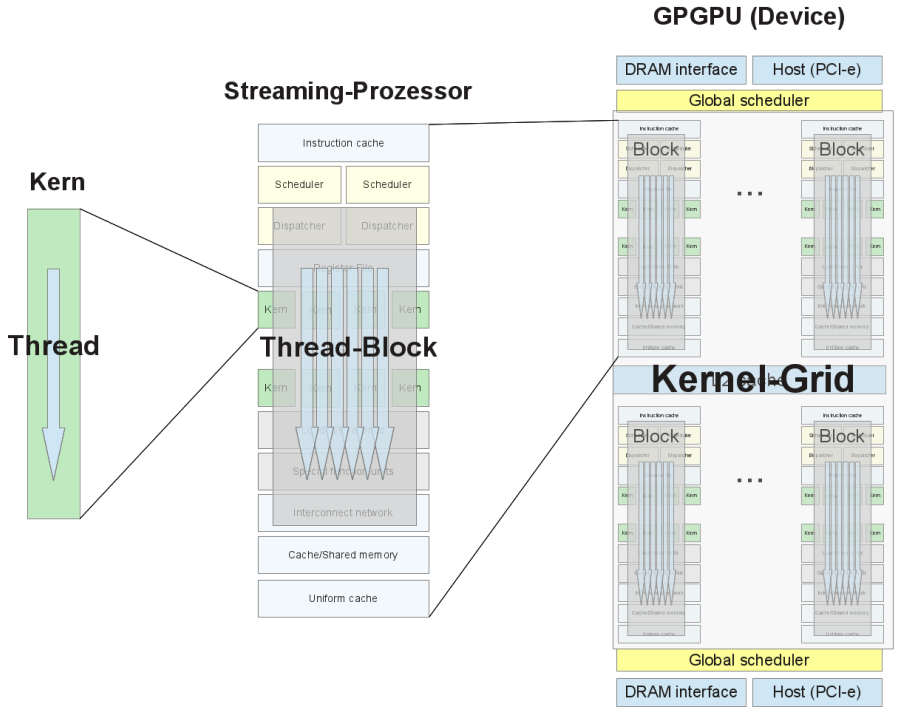
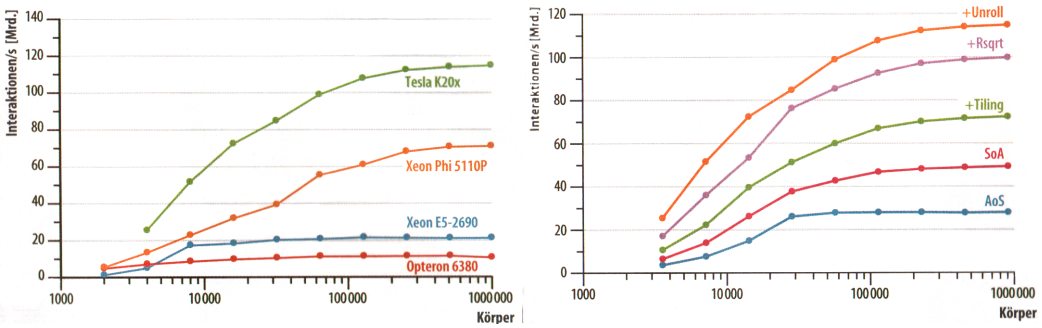


Abbildung C.3.: Hierarchische Strukturierung der Ausführung von Threads auf einer GPU

Die weitere Gruppierung von Ausführungseinheiten auf der GPU ist in Abbildung C.3 zu sehen: Threads werden zu Thread-Blöcken gruppiert, die jeweils einem Streaming-Prozessor zugeordnet werden. Abhängig von den Speicheranforderungen eines Thread-Blocks und dem verfügbaren Speicher auf einem Streaming-Prozessor können dabei auch mehrere (nebenläufige) Thread-Blöcke auf einem Streaming-Prozessor verarbeitet werden. Thread-Blöcke werden schließlich weiter zu Gittern (engl. „Grid“) gruppiert, welche die Berechnungen für einen bestimmten aktiven Kernel durchführen. Ein Gitter ist dabei an eine spezifische GPU gebunden.

Abbildung C.4a zeigt anhand des Vielkörper-Problems ermittelte Leistungsdaten unterschiedlicher aktueller CPU- und GPU-Varianten. Dabei sind die Anzahl simulierter Körper



(a) Leistungsfähigkeit verschiedener CPU- und GPU-Plattformen

(b) Leistungsfähigkeit der NVidia Tesla K20X bei unterschiedlichen Optimierungen

Abbildung C.4.: Leistungsdaten von CPU- und GPU-Plattformen anhand des Vielkörper-Problems

und die jeweils erreichte Anzahl berechneter Interaktionen pro Sekunde aufgetragen. Die Leistung der beiden gezeigten konventionellen Prozessoren im Diagramm (Intel Xeon E5 und AMD Opteron) liegt dabei deutlich unter denen des Intel Xeon Phi (ein massiv paralleler Coprozessor für die Xeon-Serie) und der NVidia Tesla. Auch wenn für massiv parallele Prozessor-Plattformen die Entwicklung neuer beziehungsweise Adaption existierender Programme besondere Sorgfalt bei der Berücksichtigung architektur-spezifischer Besonderheiten (wie beispielsweise die zuvor angesprochene Anordnung von Eingabedaten im Speicher, oder die geschickte Umformulierung zu überprüfender Bedingungen in einem Schleifen-Konstrukt) nötig ist, und es daneben viele Probleme gibt, die nicht oder nicht ohne großen Portierungs-Aufwand parallelisierbar berechenbar sind, so zeigen diese Leistungsdaten doch eindrucksvoll, welchen potentiellen Leistungsgewinn die Verwendung von GPU-Prozessoren bieten kann.

Abbildung C.4b zeigt den Effekt verschiedener Optimierungen sowohl an den verwendeten Datenstrukturen als auch durch die Verwendung plattform-spezifischer mathematischer Funktionen, jeweils für eine NVidia Tesla-GPU:

- Die zuvor beschriebene Verwendung kontinuierlich allokierter Speicherbereiche (in Abbildung C.4b als *SoA*, engl. „Structure of Arrays“, bezeichnet) gegenüber einer Datenstruktur, die die Attribute jedes simulierten Körpers separat erfasst und für jeden davon separat instantiiert wird (in Abbildung C.4b als *AoS*, engl. „Array of Structures“ bezeichnet) führt beispielsweise bereits zu einem Anstieg der zu erreichenden Interaktionen pro Sekunde um etwa 40%.
- Die Verwendung einer optimierten Funktion zur Berechnung des Kehrwertes einer Quadratwurzel (*rsqrt* in Abbildung C.4b) erhöht die Leistungsfähigkeit der Berechnung nach Ausschöpfung der Optimierungen der Datenstrukturen beispielsweise um weitere 40%.
- Die Vermeidung einer separaten Überprüfung auf die Berechnung des Einflusses eines Körpers auf sich selbst (*Unroll* in Abbildung C.4b) schließlich bringt einen weiteren Leistungszuwachs um etwa 15%.

Dieses Beispiel für den Einsatz von GPU-Prozessoren an einem zwar einfach formulierten und strukturierten numerischen Problem mit vereinfachten Annahmen führt sehr anschau-

lich vor, wie effizient und damit im Kontext der Kollisionserkennung effektiv massiv parallele Algorithmen potentiell sein können. Es ist jedoch bereits an diesem einfachen Beispiel zu erkennen, dass bei der Implementierung von numerischen Verfahren für GPUs sorgfältig vorgegangen werden muss, und mitunter eine Kombination aus Modifikationen an den verwendeten Algorithmen und der Umgehung plattform-spezifischer Probleme beziehungsweise der Ausnutzung von Besonderheiten der Hardware-Architektur nötig ist, um potentiell mögliche Leistungsgewinne auch tatsächlich ausnutzen zu können.

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTE DER ALGORITHMEN

LITERATURVERZEICHNIS

[AFC+10] Allard, Jérémie ; Faure, François ; Courtecuisse, Hadrien ; Falipou, Florent ; Duriez, Christian ; Kry, Paul G.: Volume contact constraints at arbitrary resolution. In: *ACM SIGGRAPH 2010 papers*. New York, NY, USA : ACM, 2010 (SIGGRAPH '10). – ISBN 978–1–4503–0210–4, 82:1–82:10

[All07] Allard, Jérémie: *SOFA Collision Pipeline*. `http://www.sofa-framework.org/docs/SOFA-collision-pipeline-Sept07.pdf`. Version: 2007

[Bar89] Baraff, David: Analytical methods for dynamic simulation of non-penetrating rigid bodies. In: *ACM SIGGRAPH Computer Graphics* 23 (1989), Nr. 3, 223–232. `http://dl.acm.org/citation.cfm?id=74356`

[Bar92] Baraff, David: Dynamic simulation of non-penetrating rigid bodies. In: *Computer Graphics (SIGGRAPH'92)* (1992), 303–308. `http://www.cs.cmu.edu/~baraff/papers/thesis-part1.ps.Z`

[BCG+96] Barequet, Gill ; Chazelle, Bernard ; Guibas, Leonidas J. ; Mitchell, Joseph S. ; Tal, Ayellet: BOXTREE: A hierarchical representation for surfaces in 3D. In: *Computer Graphics Forum* Bd. 15 Wiley Online Library, 1996, S. 387–396

[BCW11] Brunnett, G. ; Coquillart, S. ; Welch, G.: *Virtual Realities: Dagstuhl Seminar 2008*. Springer Vienna, 2011 (SpringerLink : Bücher). `http://books.google.de/books?id=YREwmfLR5MYC`. – ISBN 9783211991787

[Bec10] Becker, Irina: *Gegenüberstellung und Evaluation zweier Kollisionserkennungsalgorithmen*. 2010

[Ben75] Bentley, Jon L.: Multidimensional binary search trees used for associative searching. In: *Communications of the ACM* 18 (1975), Nr. 9, S. 509–517

[Ben07] Bender, Jan: *Impulsbasierte Dynamiksimulation von Mehrkörrpersystemen in der virtuellen Realität*. Karlsruhe, Universität Fridericiana zu Karlsruhe, Diss., 2007. `http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/2558`

[Ber97] Bergen, Gino van d.: Efficient collision detection of complex deformable models using AABB trees. In: *Journal of Graphics Tools* 2 (1997), Nr. 4, S. 1–13

[BHP01] Barequet, Gill ; Har-Peled, Sariel: Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. In: *Journal of Algorithms* 38 (2001), Nr. 1, S. 91–109

[Buc99] Buck, Matthias: *Simulation interaktiv bewegter Objekte mit Hinderniskontakten*. Saarbrücken, Universität des Saarlands, Diss., 1999. `http://scidok.sulb.uni-saarland.de/volltexte/2004/169/pdf/MatthiasBuck_ProfDrGuenterHotz.pdf`

[CC92] Cottle, Richard W. ; Chang, Yow-Yieh: Least-index resolution of degeneracy in linear complementarity problems with sufficient matrices. In: *SIAM Journal on Matrix Analysis and Applications* 13 (1992), Nr. 4, S. 1131–1141

[Cha84] Chazelle, Bernard: Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. In: *SIAM Journal on Computing* 13 (1984), Nr. 3, S. 488–507

[CLMP95] Cohen, Jonathan D. ; Lin, Ming C. ; Manocha, Dinesh ; Ponamgi, Madhav: I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In: *Proceedings of the 1995 symposium on Interactive 3D graphics* ACM, 1995, 189–ff

[Clo60] Clough, R. W.: The finite element method in plane stress analysis. In: ASCE Conference Electronic Computation (Hrsg.): *Proceedings of the ASCE Conference Electronic Computation*. Pittsburgh, PA, 1960

[CM05] Chum, Ondrej ; Matas, Jiri: Matching with PROSAC-progressive sample consensus. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* Bd. 1 IEEE, 2005, S. 220–226

[CML13a] CMLabs Simulations: *Construction, Mining and Forestry*. http://www.vxsim.com/en/applications/heavy_equipment/index.php. Version: 2013

[CML13b] CMLabs Simulations: *Vortex Physics Engine*. http://www.vxsim.com. Version: 2013

[Cou10] Coumans, Erwin: *Contact Generation*. http://bullet.googlecode.com/files/GDC10_Coumans_Erwin_Contact.pdf. Version: 2010

[Cou11] Coumans, Erwin: *Contact generation between 3D convex meshes*. http://www.altdevblogaday.com/2011/05/13/contact-generation-between-3d-convex-meshes. Version: 2011

[Cou12] Coumans, Erwin: *Bullet Physics Engine*. http://bulletphysics.org/wordpress. Version: 2012

[Cou13] Coumans, Erwin: *GPU Rigid Body Simulation*. https://bullet.googlecode.com/files/GDC2013_ErwinCoumans_GPU_rigid_body_simulation.pdf. Version: 2013

[CR98] Chatterjee, Anindya ; Ruina, Andy: A new algebraic rigid-body collision law based on impulse space considerations. In: *TRANSACTIONS-AMERICAN SOCIETY OF MECHANICAL ENGINEERS JOURNAL OF APPLIED MECHANICS* 65 (1998), S. 939–951

[Cry11] Crytek GmbH: *CryEngine*. http://crytek.com/cryengine/cryengine3/overview. Version: 2011

[cud13] NVIDIA Corporation: *Thrust*. http://docs.nvidia.com/cuda/thrust/. Version: 2013

[DG02] Devillers, Olivier ; Guigue, Philippe: *Faster Triangle-Triangle Intersection Tests*. http://www.inria.fr/rrrt/rr-4488.html. Version: 2002

[Ebe10] Eberly, David H.: *Game physics*. 2nd. Amsterdam, Boston : Morgan Kaufmann, 2010 http://www.worldcat.org/oclc/733937388. – ISBN 978–0–12–374903–1

[Ecc11] ECCERobot Project Consortium: *ECCERobot Project*. http://eccerobot.org/. Version: 2011

[Eck98] Eckstein, Jens: *Echtzeitfähige Kollisionserkennung für Virtual Reality Anwendungen*. Saarbrücken, Universität des Saarlands, Diss., 1998. http://scidok.sulb.uni-saarland.de/volltexte/2004/179/pdf/JensEckstein_ProfDrGuenterHotz.pdf

[EMSW13] Erbes, Rainer ; Mantel, Anja ; Schömer, Elmar ; Wolpert, Nicola: Parallel Collision Queries on the GPU. Version: 2013. http://dx.doi.org/10.1007/978-3-642-35893-7_8. In: Keller, Rainer (Hrsg.) ; Kramer, David (Hrsg.) ; Weiss, Jan-Philipp (Hrsg.): *Facing the Multicore-Challenge III* Bd. 7686. Springer Berlin Heidelberg, 2013. – DOI 10.1007/978–3–642–35893–7_8. – ISBN 978–3–642–35892–0, 84–95

[Eri05] Ericson, Christer: *Real-time collision detection: Christer Ericson*. Amsterdam : Elsevier and Morgan Kaufmann, 2005 http://www.worldcat.org/oclc/439755683. – ISBN 978–1558607323

[Erl05] Erleben, Kenny: *Stable, Robust, and Versatile Multibody Dynamics Animation*, Diss., 2005. http://image.diku.dk/kenny/download/erleben.05.thesis.pdf

[Eur07] Europa-Parlament: *Seventh Framework Programme (2007 to 2013): 1982/2006/EC.* http://europa.eu/legislation_summaries/energy/european_energy_policy/i23022_en.htm. Version: 2007

[Fau09] Faure, François: *Contact Modeling.* http://www.sofa-framework.org/docs/vr09-contacts.pdf. Version: 2009

[FB81] Fischler, Martin A. ; Bolles, Robert C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In: *Communications of the ACM* 24 (1981), Nr. 6, S. 381–395

[FBF77] Friedman, Jerome H. ; Bentley, Jon L. ; Finkel, Raphael A.: An algorithm for finding best matches in logarithmic expected time. In: *ACM Transactions on Mathematical Software (TOMS)* 3 (1977), Nr. 3, S. 209–226

[FBJF08] Faure, François ; Barbier, Sébastien ; Jérémie, Allard ; Florent, Falipou: *Image-based Collision Detection and Response between Arbitrary Volume Objects.* Version: 2008. http://www-ljk.imag.fr/Publications/Basilic/com.lmc.publi.PUBLI_Inproceedings@11a3a80d87d_19a029e/main.pdf

[Fea08] Featherstone, R.: *Robot Dynamics Algorithms.* Springer, 2008 (Kluwer international series in engineering and computer science: Robotics). http://books.google.de/books?id=UjWbvqWaf6gC. – ISBN 9780387743158

[Fis10] Fiser, Daniel: *libccd: Library for collision detection between convex shapes.* http://libccd.danfis.cz/about. Version: 2010

[FKN80] Fuchs, Henry ; Kedem, Zvi M. ; Naylor, Bruce F.: On Visible Surface Generation by a Priori Tree Structures. In: *SIGGRAPH Comput. Graph.* 14 (1980), Juli, Nr. 3, 124–133. http://dx.doi.org/10.1145/965105.807481. – DOI 10.1145/965105.807481. – ISSN 0097–8930

[FPRJ00] Frisken, Sarah F. ; Perry, Ronald N. ; Rockwood, Alyn P. ; Jones, Thouis R.: Adaptively sampled distance fields: a general representation of shape for computer graphics. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* ACM Press/Addison-Wesley Publishing Co., 2000, S. 249–254

[FSG03] Fuhrmann, Arnulph ; Sobotka, Gerrit ; Gross, Clemens: Distance fields for rapid collision detection in physically based modeling. In: *Proceedings of GraphiCon 2003*, 2003, 58–65

[GB96] Greenspan, M. ; Burtnyk, N.: Obstacle count independent real-time collision avoidance. In: *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on* Bd. 2, 1996. – ISSN 1050–4729, S. 1073–1080 vol.2

[GBF03] Guendelman, Eran ; Bridson, Robert ; Fedkiw, Ronald: Nonconvex rigid bodies with stacking. In: *ACM Transactions on Graphics (TOG)* 22 (2003), Nr. 3, S. 871–878

[GBK05] Guthe, Michael ; Balázs, Aákos ; Klein, Reinhard: GPU-based trimming and tessellation of NURBS and T-Spline surfaces. In: *ACM Transactions on Graphics (TOG)* Bd. 24 ACM, 2005, S. 1016–1023

[Ger12a] Gerkey, Brian: *Player/Stage.* http://playerstage.sourceforge.net/index.php?src=index. Version: 2012

[Ger12b] Gerkey, Brian P.: *ROS: Robot Operating System.* http://www.ros.org/wiki. Version: 2012

[GGK06] Gress, Alexander (Hrsg.) ; Guthe, Michael (Hrsg.) ; Klein, Reinhard (Hrsg.): *GPU-based Collision Detection for Deformable Parameterized Surfaces.* Bd. *3.* 2006

[GJK88] Gilbert, Elmer G. ; Johnson, Daniel W. ; Keerthi, S S.: A fast procedure for computing the distance between complex objects in three-dimensional space. In: *Robotics and Automation, IEEE Journal of* 4 (1988), Nr. 2, 193–203. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=2083

[GKLM07] Govindaraju, Naga K. ; Kabul, Ilknur ; Lin, Ming C. ; Manocha, Dinesh: Fast continuous collision detection among deformable models using graphics processors. In: *Computers & Graphics* 31 (2007), Nr. 1, S. 5–14

[GLM96] Gottschalk, Stefan ; Lin, Ming C. ; Manocha, Dinesh: OBBTree: a hierarchical structure for rapid interference detection. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* ACM, 1996, S. 171–180

[GLM05] Govindaraju, Naga K. ; Lin, Ming C. ; Manocha, Dinesh: Quick-cullide: Fast inter-and intra-object collision culling using graphics hardware. In: *Virtual Reality, 2005. Proceedings. VR 2005. IEEE*, 2005

[Got00] Gottschalk, Stefan: *Collision queries using oriented bounding boxes*, The University of North Carolina, Diss., 2000

[GRLM03] Govindaraju, Naga K. ; Redon, Stephane ; Lin, Ming C. ; Manocha, Dinesh: CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* Eurographics Association, 2003, S. 25–32

[GS87] Goldsmith, Jeffrey ; Salmon, John: Automatic creation of object hierarchies for ray tracing. In: *Computer Graphics and Applications, IEEE* 7 (1987), Nr. 5, S. 14–20

[GSB99] García, Miguel A. ; Sappa, Angel D. ; Basanez, Luis: Efficient generation of object hierarchies from 3d scenes. In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on* Bd. 2 IEEE, 1999, S. 1359–1364

[Gue06] Guendelman, Eran: *Physically-Based Simulation of Solids and Solid-Fluid Coupling*, Stanford University, Diss., 2006

[Hac07] Hachenberger, Peter: Exact Minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra in convex pieces. In: *Algorithms–ESA 2007*. Springer, 2007, S. 669–680

[Hac13] Hachenberger, Peter: Convex Decomposition of Polyhedra. In: *CGAL User and Reference Manual*. 4.2. CGAL Editorial Board, 2013. – http://www.cgal.org/Manual/4.2/doc_html/cgal_manual/packages.html#Pkg:ConvexDecomposition3

[Hah88] Hahn, James K.: Realistic animation of rigid bodies. In: *SIGGRAPH Comput. Graph.* 22 (1988), Juni, Nr. 4, 299–308. http://dx.doi.org/10.1145/378456.378530. – DOI 10.1145/378456.378530. – ISSN 0097–8930

[Hav11] Havok.com Inc: *Havok Physics*. http://www.havok.com/uploads/Havok_Physics_2011.pdf. Version: 2011

[He99] He, Taosong: Fast collision detection using QuOSPO trees. In: *Proceedings of the 1999 symposium on Interactive 3D graphics* ACM, 1999, S. 55–62

[Hel97] Held, Martin: Erit—a collection of efficient and reliable intersection tests. In: *Journal of Graphics Tools* 2 (1997), Nr. 4, 25–44. http://www.cosy.sbg.ac.at/~held/papers/jgt98.ps.gz

[Hir02] Hirota, Gentaro: *An Improved Finite Element Contact Model for Anatomical Simulations*, University of North Carolina, Chapel Hill, Diss., 2002. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.908&rep=rep1&type=pdf

[Hof89] Hoffmann, Christoph M.: *Geometric and solid modeling: An introduction*. San Mateo, Calif : Morgan Kaufmann, 1989 http://www.worldcat.org/oclc/19922242. – ISBN 1558600671

[Hub95] Hubbard, Philip M.: Collision detection for interactive graphics applications. In: *Visualization and Computer Graphics, IEEE Transactions on* 1 (1995), Nr. 3, 218–230. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=466717

[HWB+13] Hornung, Armin ; Wurm, Kai M. ; Bennewitz, Maren ; Stachniss, Cyrill ; Burgard, Wolfram: OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. In: *Autonomous Robots* (2013). http://dx.doi.org/10.1007/s10514-012-9321-0. – DOI 10.1007/s10514–012–9321–0. – Software available at http://octomap.github.com

[INR13] INRIA: *SOFA: Simulation Open Framework Architecture.* http://www.sofa-framework.org. Version: 2013

[JV03] Jansson, Johan ; Vergeest, Joris S.: Combining deformable-and rigid-body mechanics simulation. In: *The Visual Computer* 19 (2003), Nr. 5, 280–290. http://link.springer.com/article/10.1007/s00371-002-0187-6

[KA98] Konecny, Petr ; An, Zad: *Bounding Volumes in Computer Graphics.* Brno, Czech Republic, Diss., 1998. http://decibel.fi.muni.cz/HCILAB/publications/thesis.ps.gz

[KHH+09] Kim, Duksu ; Heo, Jae-Pil ; Huh, Jaehyuk ; Kim, John ; Yoon, Sung-eui: HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs. In: *Computer Graphics Forum* Bd. 28 Wiley Online Library, 2009, 1791–1800

[KHM+98] Klosowski, James T. ; Held, Martin ; Mitchell, Joseph S. ; Sowizral, Henry ; Zikan, Karel: Efficient collision detection using bounding volume hierarchies of k-DOPs. In: *Visualization and Computer Graphics, IEEE Transactions on* 4 (1998), Nr. 1, S. 21–36

[KK86] Kay, Timothy L. ; Kajiya, James T.: Ray tracing complex scenes. In: *ACM SIGGRAPH Computer Graphics* Bd. 20 ACM, 1986, 269–278

[Kle12] Klein, Bernd: *FEM: Grundlagen und Anwendungen der Finite-Element-Methode im Maschinen- und Fahrzeugbau : mit 12 Fallstudien und 20 Übungsaufgaben.* 9., verb. und erw. Wiesbaden : Springer Vieweg, 2012 http://www.worldcat.org/oclc/782975388. – ISBN 978–3–8348–1603–0

[Koe12] Koenig, Nate: *Gazebo: Gazebo is a 3D multi-robot simulator with dynamics. It is capable of simulating articulated robot in complex and realistic environments.* http://www.gazebosim.org. Version: 2012

[KSG11] Krammer, Petra ; Schlick, Jochen ; Gorecky, Dominic: VISTRA – Virtuelle Simulation und Training in der Montage und im Service. Version: 2011. http://www.vistra-project.eu/cms/htdocs/index.php?id=7. In: Springer Automotive Media (Hrsg.): *Tagungsband ATZproduktion-Fachtagung.* 2011

[KT03] Katz, Sagi ; Tal, Ayellet: *Hierarchical mesh decomposition using fuzzy clustering and cuts.* ACM, 2003

[Kuk12] Kuka Gmbh: *Kuka SimPro.* http://www.kuka-robotics.com/germany/de/products/software/kuka_sim/kuka_sim_detail/PS_KUKA_Sim_Pro.htm. Version: 2012

[LA06] Lien, Jyh-Ming ; Amato, Nancy M.: Approximate convex decomposition of polygons. In: *Computational Geometry* 35 (2006), Nr. 1, S. 100–123

[LA08] Lien, Jyh-Ming ; Amato, Nancy M.: Approximate convex decomposition of polyhedra and its applications. In: *Computer Aided Geometric Design* 25 (2008), Nr. 7, 503–522. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.151.8227&rep=rep1&type=pdf

[LC87] Lorensen, William E. ; Cline, Harvey E.: Marching cubes: A high resolution 3D surface construction algorithm. In: *ACM Siggraph Computer Graphics* Bd. 21 ACM, 1987, S. 163–169

[LC98] Li, Tsai-Yen ; Chen, Jin-Shin: Incremental 3d collision detection with hierarchical data structures. In: *Proceedings of the ACM symposium on Virtual reality software and technology* ACM, 1998, S. 139–144

[Leo07] Leon, Francisco: *GIMPACT - Geometric tools for VR.* http://gimpact.sourceforge.net. Version: 2007

[LG98] Lin, Ming ; Gottschalk, Stefan: Collision detection between geometric models: A survey. In: *Proc. of IMA Conference on Mathematics of Surfaces* Bd. 1, 1998, S. 602–608

[LGLM99] Larsen, Eric ; Gottschalk, Stefan ; Lin, Ming C. ; Manocha, Dinesh: *Fast Proximity Queries with Swept Sphere Volumes.* Version: 1999. http://gamma.cs.unc.edu/SSV/ssv.pdf

[LGLM00] Larsen, Eric ; Gottschalk, Stefan ; Lin, Ming C. ; Manocha, Dinesh: Fast distance queries with rectangular swept sphere volumes. In: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on* Bd. 4 IEEE, 2000, 3719–3726

[LGS+09] Lauterbach, C. ; Garland, M. ; Sengupta, S. ; Luebke, D. ; Manocha, D.: Fast BVH Construction on GPUs. In: *Computer Graphics Forum* 28 (2009), Nr. 2, 375–384. http://dx.doi.org/10.1111/j.1467-8659.2009.01377.x. – DOI 10.1111/j.1467–8659.2009.01377.x. – ISSN 1467–8659

[Lin11] Linneweber, Thorben: *Continuous Collision Detection*. http://jitter-physics.com/wordpress/?tag=continuous-collision-detection. Version: 2011

[LK02] Lawlor, Orion S. ; Kalée, Laxmikant V.: A voxel-based parallel collision detection algorithm. In: *Proceedings of the 16th international conference on Supercomputing* ACM, 2002, 285–293

[LK11] Laine, Samuli ; Karras, Tero: Efficient sparse voxel octrees. In: *Visualization and Computer Graphics, IEEE Transactions on* 17 (2011), Nr. 8, S. 1048–1059

[LMM10] Lauterbach, C. ; Mo, Q. ; Manocha, Dinesh: *gProximity: Hierarchical GPU-based Operations for Collision and Distance Queries*. Version: 2010. http://gamma.cs.unc.edu/GPUCOL/gProximity.pdf

[Löt82] Lötstedt, Per: Mechanical systems of rigid bodies subject to unilateral constraints. In: *SIAM Journal on Applied Mathematics* 42 (1982), Nr. 2, 281–296. http://epubs.siam.org/doi/abs/10.1137/0142022

[Löt84] Lötstedt, Per: Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. In: *SIAM journal on scientific and statistical computing* 5 (1984), Nr. 2, 370–393. http://epubs.siam.org/doi/abs/10.1137/0905028

[MC94] Mirtich, B. ; Canny, J. F.: *Impulse-based dynamic simulation*. Version: 1994. http://www.cs.berkeley.edu/~jfc/papers/94/ibds94.pdf

[MC96] Mirtich, Brian ; Canny, JF: Hybrid simulation: combining constraints and impulses. In: *Proceedings of First Workshop on Simulation and Interaction in Virtual Environments*, 1996

[MD10] Marius Muja ; David G. Lowe: Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. Version: 2010. http://people.cs.ubc.ca/~mariusm/uploads/FLANN/flann_visapp09.pdf. In: Ranchordas, AlpeshKumar (Hrsg.): *Computer vision, imaging and computer graphics* Bd. 68. Berlin : Springer, 2010. – ISBN 9783642118395

[Mea80] Meagher, D. J. R.: *Octree Encoding: a New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. 1980 http://books.google.de/books?id=CgRPOAAACAAJ

[Mea82] Meagher, D. J. R.: *Octree Generation: Analysis and Manipulation*. Defense Technical Information Center, 1982 http://books.google.de/books?id=keGDnQEACAAJ

[MG09] Mamou, K. ; Ghorbel, F.: A simple and efficient approach for 3D mesh approximate convex decomposition. In: *Image Processing (ICIP), 2009 16th IEEE International Conference on*, 2009. – ISSN 1522–4880, S. 3501–3504

[Mic12a] Michel, Olivier: *Webots: the mobile robotics simulation software*. http://www.cyberbotics.com. Version: 2012

[Mic12b] Microsoft: *Robotics Developer Studio*. http://www.microsoft.com/robotics/#About. Version: 2012

[Mir96] Mirtich, B.: *Impulse-based Dynamic Simulaton of Rigid Body Systems*. Berkeley, University of California, Berkeley, CA, USA, Diss., 1996. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.783

[Mir98] Mirtich, Brian: V-Clip: Fast and robust polyhedral collision detection. In: *ACM Transactions on Graphics (TOG)* 17 (1998), Nr. 3, 177–208. http://dl.acm.org/citation.cfm?id=285860

[MKE03] Mezger, Johannes ; Kimmerle, Stefan ; Etzmuss, Olaf: Hierarchical techniques in collision detection for cloth animation. (2003). https://dspace.zcu.cz/bitstream/handle/11025/1617/G97.pdf?sequence=1

[ML12] Muja, Marius ; Lowe, David G.: Fast Matching of Binary Features. In: *Computer and Robot Vision (CRV)*, 2012, S. 404–410

[Möl97] Möller, Tomas: A fast triangle-triangle intersection test. In: *Journal of graphics tools* 2 (1997), Nr. 2, 25–30. http://www.ce.chalmers.se/staff/tomasm/pubs/tritri.pdf

[MOR66] MORTON, G. M.: A computer oriented geodetic data base and a new technique in file sequencing. Ottawa, 1966. – Forschungsbericht

[MPT99] McNeely, William A. ; Puterbaugh, Kevin D. ; Troy, James J.: Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling. In: Rockwood, Alyn (Hrsg.): *Computer graphics*. New York, N.Y : Association for Computing Machinery, 1999. – ISBN 0–201–48560–5, S. 401–408

[MPT06] McNeely, William A. ; Puterbaugh, Kevin D. ; Troy, James J.: *Voxel-Based 6-DOF Haptic Rendering Improvements*. Version: 2006. http://www.haptics-e.org/Vol_03/he-v3n7.pdf

[MS01] Milenkovic, Victor J. ; Schmidl, Harald: Optimization-based animation. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* ACM, 2001, 37–46

[MW88] Moore, Matthew ; Wilhelms, Jane: Collision Detection and Response for Computer Animation. In: *SIGGRAPH Comput. Graph.* 22 (1988), Juni, Nr. 4, 289–298. http://dx.doi.org/10.1145/378456.378528. – DOI 10.1145/378456.378528. – ISSN 0097–8930

[MZ90] McKenna, Michael ; Zeltzer, David: Dynamic simulation of autonomous legged locomotion. In: *ACM SIGGRAPH Computer Graphics* Bd. 24 ACM, 1990, 29–38

[NFA88] Noborio, Hiroshi ; Fukuda, Shozo ; Arimoto, Suguru: Fast interference check method using octree representation. In: *Advanced robotics* 3 (1988), Nr. 3, S. 193–212

[NVi11] NVidia Corporation: *PhysX*. http://www.nvidia.com/object/physx_new.html. Version: 2011

[OD99] O'SULLIVAN, CAROL ; Dingliana, John: Real-time collision detection and response using sphere-trees. (1999). http://www.tara.tcd.ie/jspui/bitstream/2262/64112/1/spheres.PDF

[Omo89] Omohundro, Stephen M.: Five Balltree Construction Algorithms. 1989. – Forschungsbericht

[O'R08] O'Rourke, Joseph: *Computational geometry in C*. 2. Ed., 11. print. Cambridge [u.a.] : Cambridge Univ. Pr., 2008. – XIII, 376 S.. – ISBN 978–0–521–64010–7

[PC01] Poutrain, Kai ; Contensin, Magali: Dual B-Rep-CSG collision detection for general polyhedra. In: *Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on* IEEE, 2001, 124–133

[Poi92] Poincaré, Henri: *Les méthodes nouvelles de la mécanique céleste*. Paris, Frankreich : Gauthier-Villars et fils, 1892

[PZ95] Purgathofer, Werner ; Zeiller, Michael: CSG Based Collision Detection. In: *Graphics and Robotics*. Springer, 1995, S. 59–72

[Qui94] Quinlan, S.: Efficient distance computation between non-convex objects. In: *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, 1994, S. 3324–3329 vol.4

[RC11] Rusu, Radu B. ; Cousins, Steve: 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 9-13 2011

[Rog01] Rogers, David F.: *An introduction to NURBS : with historical perspective*. 2001. – 1–344 S. ISSN 1558606696

[Ros11] Rossdeutscher, Mario: *Entwicklung eines Verfahrens zum Programmtest in der robotergestützten Montage*. Cottbus, Technische Universität Cottbus, Diss., 2011

[Rus09] Rusu, Radu B.: *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*, Computer Science department, Technische Universitaet Muenchen, Germany, Diss., October 2009. http://files.rbrusu.com/publications/RusuPhDThesis.pdf

[RV89] Rossignac, Jaroslaw R. ; Voelcker, Herbert B.: Active zones in CSG for accelerating boundary evaluation, redundancy elimination, interference detection, and shading algorithms. In: *ACM Transactions on Graphics* 8 (1989), S. 51–87

[SDC08] Saupin, Guillaume ; Duriez, Christian ; Cotin, Stephane: Contact Model for Haptic Medical Simulations. In: Bello, Fernando (Hrsg.) ; Edwards, P.J.Eddie (Hrsg.): *Biomedical Simulation* Bd. 5104. Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–70520–8, S. 157–165

[SGHS98] Shade, Jonathan ; Gortler, Steve ; He, Li-wei ; Szeliski, Richard: *Layered Depth Images*. Version: 1998. http://grail.cs.washington.edu/projects/ldi/ldi.pdf

[SH74] Sutherland, Ivan E. ; Hodgman, Gary W.: Reentrant polygon clipping. In: *Commun. ACM* 17 (1974), Januar, Nr. 1, 32–42. http://dx.doi.org/10.1145/360767.360802. – DOI 10.1145/360767.360802. – ISSN 0001–0782

[SHPH08] Sagardia, M. ; Hulin, T. ; Preusche, C. ; Hirzinger, G.: *Improvements of the Voxmap-PointShell Algorithm — Fast Generation of Haptic Data-Structures*. Version: 2008. http://www.robotic.dlr.de/fileadmin/robotic/hulin/publications/Sagardia_VoxmapPointshellGeneration_2008.pdf

[Sim13] Simpack AG: *Simpack*. http://www.simpack.com. Version: 2013

[SLY96] Su, CJ ; Lin, FH ; Yen, BP: An adaptive bounding object based algorithm for efficient and precise collision detection of CSG-represented virtual objects. In: *Proc. of Symposium on virtual reality in manufacturing research and education* Citeseer, 1996

[SM04] Schmidl, Harald ; Milenkovic, Victor J.: A fast impulsive contact suite for rigid body simulation. In: *IEEE Transactions on Visualization and Computer Graphics* (2004), 189–197. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.159.26

[Smi06] Smith, Colin: *On vertex-vertex systems and their use in geometric and biological modelling*. University of Calgary, 2006 http://algorithmicbotany.org/papers/smithco.dis2006.pdf

[Smi07] Smith, Russell: *ODE: Open Dynamics Engine*. http://www.ode.org. Version: 2007

[ST96] Stewart, David E. ; Trinkle, Jeffrey C.: AN IMPLICIT TIME-STEPPING SCHEME FOR RIGID BODY DYNAMICS WITH INELASTIC COLLISIONS AND COULOMB FRICTION. In: *International Journal for Numerical Methods in Engineering* 39 (1996), Nr. 15, S. 2673–2691

[Sti13] Stiller, Andreas: Von Körpern und Kräften. In: *c't - Magazin für Computertechnik* 12 (2013), S. 182–189

[Str06] Stroud, Ian: *Boundary representation modelling techniques*. London : Springer, 2006 http://www.worldcat.org/oclc/209949567. – ISBN 978–1–84628–312–3

[SWF+93] Snyder, John M. ; Woodbury, Adam R. ; Fleischer, Kurt ; Currin, Bena ; Barr, Alan H.: Interval Methods for Multi-Point Collisions between Time-Dependent Curved Surfaces. In: *Computer Graphics*, 1993, S. 321–334

[TC96] Tzafestas, Costas ; Coiffet, Philippe: Real-time collision detection using spherical octrees: virtual reality application. In: *Robot and Human Communication, 1996., 5th IEEE International Workshop on* IEEE, 1996, 500–506

[TKH+04] Teschner, M. ; Kimmerle, S. ; Heidelberger, W. ; Zachmann, Gabriel ; Raghupati, L. ; Fuhrmann, Arnulph ; Cani, M.-P ; Faure, François ; Magnenat-Thalmann, N. ; Strasser, W. ; Volino, P.: *Collision Detection for Deformable Objects*. Version: 2004. http://www-evasion.imag.fr/Publications/2004/TKZHRFCFMS04/STARmain.pdf

[TMLT11] Tang, Min ; Manocha, Dinesh ; Lin, Jiang ; Tong, Ruofeng: Collision-Streams: Fast GPU-based collision detection for deformable models. In: *I3D '11: Proceedings of the 2011 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2011, 63–70

[TZ00] Torr, Philip H. ; Zisserman, Andrew: MLESAC: A new robust estimator with application to estimating image geometry. In: *Computer Vision and Image Understanding* 78 (2000), Nr. 1, S. 138–156

[van11] van Oosten, Jeremiah: *Cuda Memory Model*. http://3dgep.com/?p=2012. Version: 2011

[VBZ90] Von Herzen, Brian ; Barr, Alan H. ; Zatz, Harold R.: Geometric collisions for time-dependent parametric surfaces. In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1990 (SIGGRAPH '90). – ISBN 0–89791–344–2, 39–48

[Vis12] Visual Components: *3D Realize*. http://www.visualcomponents.com/Products/3DRealize. Version: 2012

[Vla13] Vladimirov, Andey; Karpusenko V.: Test-driving Intel Xeon Phi coprocessors with a basic N-body simulation / Colfax Research. Version: 2013. http://research.colfaxinternational.com/file.axd&2013\/1\/Colfax_Nbody_Xeon_Phi.pdf. 2013. – Forschungsbericht

[WAB+13] Wittmeier, Steffen ; Alessandro, Cristiano ; Bascarevic, Nenad ; Dalamagkidis, Konstantinos ; Devereux, David ; Diamond, Alan ; Jäntsch, Michael ; Jovanovic, Kosta ; Knight, Rob ; Marques, Hugo G. u. a.: Toward anthropomimetic robotics: Development, simulation, and control of a musculoskeletal torso. In: *Artificial life* 19 (2013), Nr. 1, 171–193. http://www6.in.tum.de/Main/Publications/wittmeier2013a.pdf

[Wel13] Weller, René: *New geometric data structures for collision detection and haptics*. Springer, 2013 http://dx.doi.org/10.1007/978-3-319-01020-5

[Wer97] Werner, D.: *Funktionalanalysis*. Springer-Verlag GmbH, 1997 (Springer-Lehrbuch). http://books.google.de/books?id=kIw5PwAACAAJ. – ISBN 9783540619048

[WHG84] Weghorst, Hank ; Hooper, Gary ; Greenberg, Donald P.: Improved computational methods for ray tracing. In: *ACM Transactions on Graphics (TOG)* 3 (1984), Nr. 1, S. 52–69

[Win11] Winkler, Lutz: *Robot3D: Open Source Modular Swarm Robot Simulation Engine*. https://launchpad.net/robot3d. Version: 2011

[WJD+11] Wittmeier, Steffen ; Jantsch, M ; Dalamagkidis, Konstantinos ; Rickert, Markus ; Marques, Hugo G. ; Knoll, Alois: Caliper: a universal robot simulation framework for tendon-driven robots. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on* IEEE, 2011, 1063–1068

[Woe11] Woernle, Christoph: *Mehrkörpersysteme: Eine Einführung in die Kinematik und Dynamik starrer Körper // Eine Einführung in die Kinematik und Dynamik von Systemen starrer Körper*. Berlin, New York : Springer, 2011 http://www.worldcat.org/oclc/742515296. – ISBN 978–3–642–15981–7

[Wün07] Wünsch, Georg: *Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme*. München, Technische Universität München, Diss., 2007

[Wün13] Wünsch, Georg Dr.-Ing.: *industrialPhysics*. http://www.machineering.de. Version: 2013

[WW10] Winkler, Lutz ; Wörn, Heinz: Modular Robot Simulation. In: Levi, Paul (Hrsg.) ; Kernbach, Serge (Hrsg.): *Symbiotic multi-robot organisms*. Berlin : Springer, 2010. – ISBN 978–3–642–11691–9, S. 133–163

[WZ09a] Weller, Rene ; Zachmann, Gabriel: A unified approach for physically-based simulations and haptic rendering. In: *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games* ACM, 2009, S. 151–159

[WZ09b] Weller, Rene ; Zachmann, Gabriel: Inner sphere trees for proximity and penetration queries. In: *Robotics: Science and Systems* Bd. 2, 2009

[Z+00] Zachmann, Gabriel u. a.: *Virtual Reality in Assembly Simulation: Collision Detection, Simulation Algorithms, and Interaction Techniques.* Fraunhofer-IRB-Verlag, 2000 http://www2.in.tu-clausthal.de/~zach/papers/zachmann_diss_book.pdf

[Zac95] Zachmann, Gabriel: The boxtree: Exact and fast collision detection of arbitrary polyhedra. In: *First Workshop on Simulation and Interaction in Virtual Environments (SIVE 95)*, 1995

[Zac98] Zachmann, Gabriel: Rapid collision detection by dynamically aligned DOP-trees. In: *Virtual Reality Annual International Symposium, 1998. Proceedings., IEEE 1998* IEEE, 1998, 90–97

[ZH07] Zink, Nico ; Hardy, Alexandre: *Cloth Simulation and Collision Detection using Geometry Images.* Version: 2007. http://www.ahardy.za.net/Home/cloth/afrigraph_2007_final.pdf?attredirects=0

[ZL03] Zachmann, Gabriel ; Langetepe, Elmar: Geometric Data Structures for Computer Graphics. Version: Juli 2003. http://www.gabrielzachmann.org. In: *Proc. of ACM SIGGRAPH.* ACM Transactions of Graphics, Juli 2003

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

_____

(Fabian Aichele)