

Continuous Deformable Collision Detection



Knoblauch Daniel

Semesterarbeit
Winter 2004/05

Computer Graphics Laboratory
D-INFK
ETH Zürich

Prof. Dr. Markus Gross
Bruno Heidelberger

Continuous Collision Detection

Einleitung

Interaktive Simulationen werden mit einem möglichst hohen Zeitschritt getaktet. Diese Diskretisierung der Zeit hat nun weitreichende Auswirkungen auf die einzelnen Komponenten des Systems. So kann unter Umständen eine Kollision zwischen zwei Objekten unerkannt bleiben, falls sich die Kollision nur genau zwischen zwei Zeitpunkten ereignet. Die Ursache von verpassten Kollisionen ist meist die hohe relative Geschwindigkeit der involvierten Objekte im Zusammenpiel mit dem erwünscht grossen Zeitschritt.

Aufgabe

Im Rahmen dieser Semesterarbeit soll ein Verfahren entwickelt werden, welches trotz diskreter Zeitschritte der Simulation keine Kollision verpasst. Speziell soll dieses Verfahren auch für deformierbare, volumetrische Modelle geeignet sein.

Aufgabenstellung

Folgende Aufgaben sollen bearbeitet werden:

- Einarbeitung in bekannte Verfahren für *Continuous Collision Detection*, wie auch in *DefColStudio*, das bestehende System zur Simulation von deformierbaren Objekten.
- Entwurf (oder Auswahl) eines geeigneten Verfahrens, welches den speziellen Anforderungen der physikalisch deformierbaren Objekte entspricht.
- Implementierung des Verfahrens und Integration in die bestehende Simulationsumgebung.
- Entwicklung verschiedener Testszenarien zur Überprüfung des Funktionalität und Robustheit des Verfahrens.
- Erstellen eines schriftlichen Berichts über die Arbeit.

Bemerkungen

Die Semesterarbeit steht unter der Obhut von Prof. Markus Gross und wird von Bruno Heidelberger betreut.

Ausgabe: Oktober 2004

Abgabe: Februar 2005

Abstract

Collision detection is a fundamental geometric problem that arises in diverse graphical applications like CAD/CAM, dynamic simulations, robotics and computer games. The most used approaches are discrete algorithms. They can have problems detecting collisions, if time intervals or relative velocities are too large. This semester thesis tries to avoid this problems by analysing a continuous, deformable collision detection.

Based on deformable objects an analytical approach has been chosen, where the collisions are determined between the surface points and triangles. In difference to the discrete algorithm, the continuous algorithm does not only investigate the collision existence but also the collision time and the position. To avoid the expensive, analytical calculation, a preprocessing procedure based on hash tables was introduced. This reduced the investigation to significant collision pairs.

The continuous deformable collision detection has been implemented and compared to the discrete collision detection. From the results it was apparent, that the continuous collision detection garanties the detection of all possible collisions. Efficiency is lost, but a collision detection is introduced that finds all collisions and additionally provides the collision time and position.

Zusammenfassung

In der Computergraphik stellt die Kollisionsdetektion ein grundlegendes Problem dar. Ihr Anwendungsgebiet erstreckt sich über Bereiche wie CAD/CAM, dynamische Simulationen, Robotik und Computerspiele. Durch den bis anhin meist angewendeten diskreten Ansatz können Probleme auf Grund grosser Zeitintervalle oder grosser relativer Geschwindigkeiten auftreten. Um eine fehlerlose Kollisionsdetektion zu garantieren, wird in dieser Semesterarbeit die kontinuierliche, deformierbare Kollisionsdetektion untersucht.

Auf Grund der deformierbaren Objekte wurde ein analytischer Ansatz gesucht, der die Kollisionen zwischen Oberflächenpunkten und Oberflächendreiecken ermittelt. Im Unterschied zur diskreten Kollisionsdetektion kann nicht nur eine Kollision erkannt werden, sondern es wird zusätzlich der genaue Zeitpunkt und Ort des Aufpralls ermittelt. Um die aufwendige, analytische Berechnung so selten wie möglich durchführen zu müssen, wurde ein Hashingverfahren eingeführt, welches die Kollisionsdetektion nur auf signifikante Kollisionspaare anwendet.

Die kontinuierliche, deformierbare Kollisionsdetektion wurde implementiert und mit der diskreten Kollisionsdetektion verglichen. Die Tests ergaben, dass die kontinuierliche Variante garantieren kann, dass alle Kollisionen entdeckt werden. Daraus folgten Einbussen in der Effizienz. Es wurde somit ein Ansatz geschaffen, der alle Kollisionen entdeckt und zusätzlich den Zeitpunkt sowie den Ort des Aufpralls ausgibt.

Inhaltsverzeichnis

1 Einleitung und Motivation	1
1.1 DefCol-Projekt	1
1.2 Motivation	1
1.3 Übersicht	3
2 Grundlagen	5
2.1 Kollisionsdetektion	5
2.1.1 Diskrete Kollisionsdetektion	5
2.1.2 Kontinuierliche Kollisionsdetektion	8
2.1.3 Evaluierete kontinuierliche Kollisionsdetektionen	9
2.2 Kollisionsdetektion im DefCol-Projekt	10
3 Realisierung	11
3.1 Übersicht	11
3.2 Hashing von bewegten Punkten und Dreiecken	11
3.2.1 Bewegte Punkte	11
3.2.2 Bewegte Dreiecke	12
3.3 Vorverarbeitung der Kollisionspaare	13
3.3.1 BoundingBox	13
3.3.2 Punkt in BoundingBox	13
3.3.3 Vermeidung mehrfacher Kollisionsdetektion	14
3.4 Kontinuierliche Kollisionsdetektion	15
3.4.1 Bisektion	15
3.4.2 Kontinuierliche Kollisionsgleichung	16
3.4.3 Existenztest	18
4 Resultate und Ausblick	21
4.1 Vergleich der kontinuierliche und diskrete Kollisionsdetektion	21
4.1.1 Einfache Simulation mit tiefen, relativen Geschwindigkeiten	21
4.1.2 Einfache Simulation mit hohen, relativen Geschwindigkeiten	23
4.1.3 Einfache Simulation mit hohen Geschwindigkeiten	24
4.1.4 Komplexe Simulation mit tiefen relativen Geschwindigkeiten	25
4.2 Fazit und Ausblick	27
5 Referenzen	29

1

Einleitung und Motivation

In diesem Kapitel werden wir eine kleine Einführung in das DefCol-Projekt geben und anschliessend die Gründe erläutern, weshalb wir uns entschieden haben die kontinuierliche Kollisionsdetektion genauer unter die Lupe zu nehmen. Zum Schluss werden wir noch einen Überblick über die Arbeit geben.

1.1 DefCol-Projekt

Das Ziel des DefCol-Projektes ist eine Entwicklungsumgebung für interaktive Simulationen und deformierbare Objekte bereitzustellen.

Die Eigenschaften der Objekte in dieser Simulation werden durch ihr Modell und ihre Struktur beschrieben. Ausserdem beschreiben die zwei Faktoren Mesh und Material die Oberfläche dieser Objekte. Die Struktur gibt die geometrische Form durch Tetraeder und Punkte an. Im Modell werden die Grundwerte wie zum Beispiel die Festigkeit oder Dämpfung der Objekte initialisiert. Diese Objekte können nun zu Szenen zusammengelegt werden, in denen die globalen Kräfte der Simulationen angegeben werden können. Auf diese Weise können wir mit einem modularen Ansatz unsere Simulationen zusammensetzen. Wir sehen ein solche Szene mit drei Objekten in Abbildung 1.1.

Während der Simulation wirken die verschiedenen globalen Kräfte auf die Objekte. Ausserdem werden die Deformationen der Objekte, die durch die Wechselwirkung der Grundwerte der Objekte und den äusseren Kräften entstehen, errechnet und simuliert. Ein weiterer wichtiger Bestandteil der Simulationen ist die Kollisionsdetektion und deren Antwort. Wir werden uns in unserer Arbeit auf die kontinuierliche Kollisionsdetektion konzentrieren.

Wir haben also eine Umgebung die es erlaubt, verschiedene Szenen zu simulieren. Da wir aber eine interaktive Simulation wollen, ist es dem Betrachter möglich interaktive Kräfte in der Simulation wirken zu lassen und die Reaktion auf diese Eingriffe in Echtzeit mitzuerleben. Dies setzt wiederum voraus, dass die angewendeten Ansätze oder Algorithmen echtzeitkompatibel sind.

1.2 Motivation

Kollisionsdetektion ist ein grundlegendes geometrisches Problem, welches in verschiedenen Applikationen wie CAD/CAM, dynamische Simulationen, Robotik, virtuellen

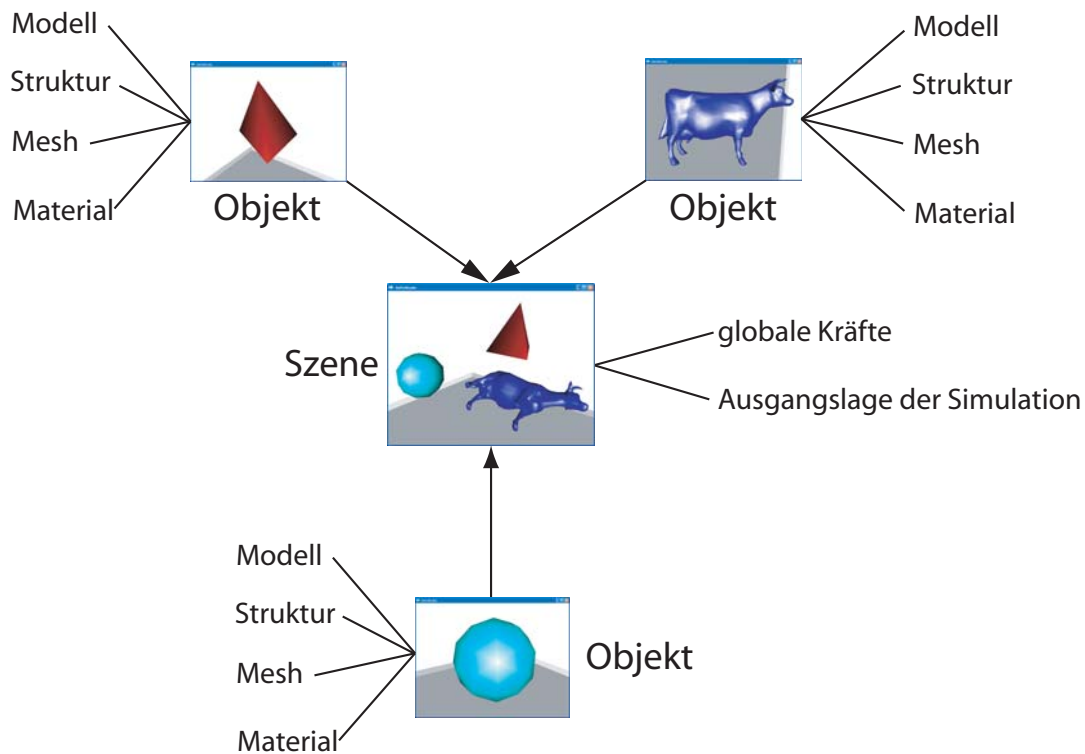


Abbildung 1.1: Wir sehen eine Szene mit drei Objekten: einem Tetraeder, einer Kugel und einer Kuh.

Umgebungen und Computerspielen eingesetzt wird. Darum wurden schon sehr viele Untersuchungen zur Lösung dieses Problems durchgeführt. Das Ziel dieser Arbeiten ist, Kollisionen von komplexen Körpern in Simulationen zu erkennen. Um dies zu erreichen werden diese Körper meistens in einfachere Unterkörper wie Punkte und Dreiecke oder Tetraeder unterteilt. Bei diesen einfacheren Gebilden werden dann die Kollisionen ermittelt. Wir sehen in Abbildung 1.2 eine solche Vereinfachung.

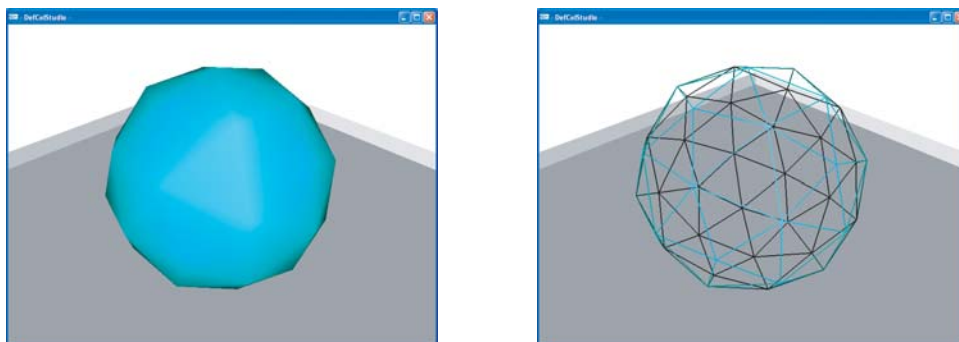


Abbildung 1.2: Links sehen wir eine Kugel und rechts die Vereinfachung in Dreiecke und deren Punkte.

Die meisten dieser Arbeiten stützen sich aber auf einen diskreten Ansatz, der nur in bestimmten Zeitintervallen nach Kollisionen sucht. Dieser Ansatz ist einfach und effizient, hat aber den grossen Nachteil, dass man auf Grund eines zu grossen Zeitintervalls

Kollisionen übersehen kann. Wir können einen solchen Fall in Abbildung 2.2 sehen. Dies kann man zwar durch die Verkleinerung des Zeitintervalls vermindert werden, dadurch verliert man aber teilweise den Vorteil der Effizienz. Aus diesem Grund haben wir uns vorgenommen einen kontinuierlichen Ansatz zu finden, welcher unsere Echtzeitsimulation ergänzen könnte. Ein kontinuierlicher Ansatz setzt nicht auf die stichprobenartige Suche nach Kollisionen, sondern betrachtet die ganze Zeitspanne und verpasst somit keine einzige Kollision. Dies führt natürlich zu mehr Rechenaufwand, was in unserer Echtzeitsimulation ein grosser Nachteil darstellt. Darum haben wir versucht eine Variante zu finden, die trotzdem ausreichend schnell für unsere Anwendung ist. Es existieren bereits ein paar wenige Arbeiten zu diesem Thema, die aber fast ausschliesslich mit starren Körpern arbeiten. Wir haben aber in unserem DefCol-Projekt deformierbare Objekte, welche den Aufwand noch vergrössern. Wir haben versucht aus den verschiedenen Ansätzen für starre Körper einen Weg zu finden, der es uns erlaubt, eine kontinuierliche deformierbare Kollisionsdetektion einzuführen.

1.3 Übersicht

Wir werden im Kapitel 2 die Grundlagen für unseren kontinuierlichen Kollisionsalgorithmus sehen. Dazu gehören eine Einführung in die diskrete Kollisionsdetektion in Section 2.1.1 sowie eine Diskussion verschiedener Ansätze der kontinuierlichen Kollisionsdetektion in Section 2.1.2 und Section 2.1.3. Ausserdem lernen wir in Section 2.1.1 eine Variante des Preprocessings kennen, die sich auf Hashtabellen stützt.

In Section 3.2 betrachten wir unsere Preprocessingverfahren genauer und passen sie unseren Bedürfnissen an. Ausserdem stellen wir zusätzliche Preprocessingverfahren in Section 3.3 vor. Im Section 3.4 führen wir dann den endgültigen kontinuierlichen Kollisionsalgorithmus ein. Zum Schluss werden wir in Section 4.1 noch den kontinuierlichen und den diskreten Kollisionsalgorithmus auf ihre Stärken und Schwächen testen und mit verschiedenen Simulationen ihre Performanz und Korrektheit überprüfen. Aus den gewonnen Erkenntnissen ziehen wir in Section 4.2 unser Fazit und stellen noch ein paar denkbare nachfolgende Arbeiten vor.

2

Grundlagen

In diesem Kapitel möchten wir verschiedene, schon im Vorfeld unserer Arbeit entwickelte Algorithmen zur diskreten sowie kontinuierlichen Kollisionsdetektion betrachten, ihre Anwendbarkeit in unserem Projekt prüfen und selber hergeleitete Ideen vorstellen. Zum Schluss betrachten wir die bis jetzt gebräuchliche diskrete Kollisionsdetektion im DefCol-Projekt

2.1 Kollisionsdetektion

Es gibt im wesentlichen zwei verschiedene Arten der Kollisionsdetektion. Man unterscheidet dabei zwischen den diskreten und den kontinuierlichen Kollisionsalgorithmen. Bei den diskreten Algorithmen werden die Tests nur zu bestimmten Zeitpunkten ausgeführt. Die kontinuierlichen hingegen betrachten den ganzen Zeitablauf. Es wird schnell ersichtlich, welches die Vor- und Nachteile der verschiedenen Ansätze sind. Die diskreten Kollisionsdetektionsalgorithmen haben den Vorteil, dass sie effizient eine geschehene Kollision detektieren können. Man kann jedoch auf Grund der nur stichprobenartigen Tests nicht garantieren, dass man auch wirklich alle Kollisionen entdeckt. Die kontinuierlichen bauen hingegen darauf auf, dass sie zu jedem Zeitpunkt eine Kollision detektieren können, was einen viel grösseren rechnerischen Aufwand verursacht. In den beiden folgenden Abschnitten wollen wir diese beiden Arten der Kollisionsdetektion genauer untersuchen.

2.1.1 Diskrete Kollisionsdetektion

In diesem Abschnitt werden wir ein paar Methoden der diskreten Kollisionsdetektion anschauen. Die grundlegende Idee dieser Methoden ist, alle Tests in bestimmten Zeitintervallen durchzuführen, was wir in Abbildung 2.1 sehen können. Je kleiner man diese Testintervalle macht, umso weniger Informationen verliert man, denn die Wahrscheinlichkeit, dass etwas zwischen diesen Intervallen unbemerkt passiert, wird kleiner. An diesen bestimmten Zeitpunkten muss nun bestimmt werden, ob eine Kollision stattgefunden hat. In Abbildung 2.2 sehen wir einen Fall, in dem die Kollision nicht entdeckt wird. Um diese diskrete Idee einzuführen, wurden mehrere Methoden hergeleitet.

Die erste Methode stützt sich auf Bounding Volumes, das heisst, dass man um die Körper, die getestet werden müssen, einen einfachen Körper legt, welcher erlaubt Durchdringungen mit anderen Körpern einfach zu berechnen. Zu diesen Körpern gehören zum Beispiel Kugeln [Hub96]. Als erstes wird getestet, ob sich solche Bounding Volumes durchdringen. Ist dies der Fall, muss auf die aufwändigere Berechnung der tatsächlichen

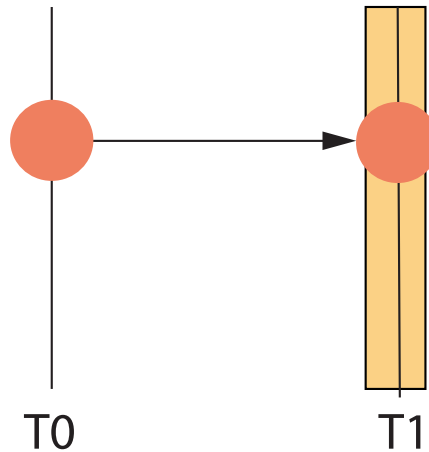


Abbildung 2.1: Diskrete Kollisionsdetektion: Im Zeitpunkt T_1 hat eine Kollision stattgefunden und wurde detektiert

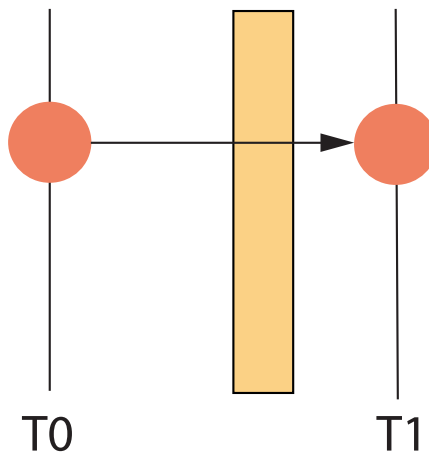


Abbildung 2.2: Diskrete Kollisionsdetektion: Kollision zwischen Zeitpunkt T_0 und T_1 wurde übersehen.

Durchdringung der ursprünglichen Körper zurückgegriffen werden. Mit diesen Bounding Volumes verhindert man, dass diese aufwändige Berechnung mit jedem möglichen Kollisionspaar gemacht werden muss. Diese Methode kann erweitert werden, in dem Bounding Volume Hierarchien benutzt werden. In solchen Fällen, werden die Körper hierarchisch in Bounding Volumes aufgeteilt. Wir sehen Beispiele von Bounding Volumes in Abbildung 2.3. Der Test wird dann auf die grössten Bounding Volumes der zu testenden Körper angewendet. Schneiden sich diese, werden die betroffenen untergeordneten Bounding Volumes auf eine Durchdringung getestet. Dieser Vorgang wird fortgesetzt, bis man bei den eigentlichen Körpern ankommt. Nun folgt ein Test auf die Durchdringung, welcher aufgrund der Aufteilung der Körper viel einfacher von statten gehen müsste. Damit aber mit diesen Methoden gearbeitet werden kann, müssen die Bounding Volumes und ihre allfälligen Hierarchien schon im voraus berechnet werden. Bei statischen Objekten ist dies kein Problem, da ihre Form unverändert bleibt und man nur die Position und Ausrichtung abändern muss. Will man aber mit deformierbaren Objekten arbeiten, so müssen diese Strukturen in jedem Zeitschritt neu berechnet werden, da sich auch die Form und somit auch die ganzen Bounding Volumes und Hierarchien verändern.

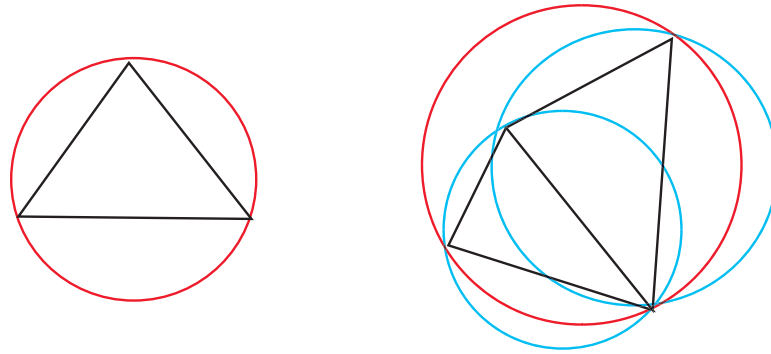


Abbildung 2.3: Bounding Volumes: links sehen wir ein einfaches Bounding Volume und rechts eine Bounding Volume Hierarchie

Aus diesem Grund wird versucht Methoden zu finden, durch die man so gut wie möglich mit Hilfe einer speziellen Datenstruktur die Berechnung von Durchdringungen ermitteln kann. Eine dieser Methoden beruht auf dem Hashingprinzip. Dabei wird die Simulationsumgebung in ein dreidimensionales Gitternetz aufgeteilt. Nun kann bestimmt werden in welchen der Gitterzellen die Körper vorhanden sind. In

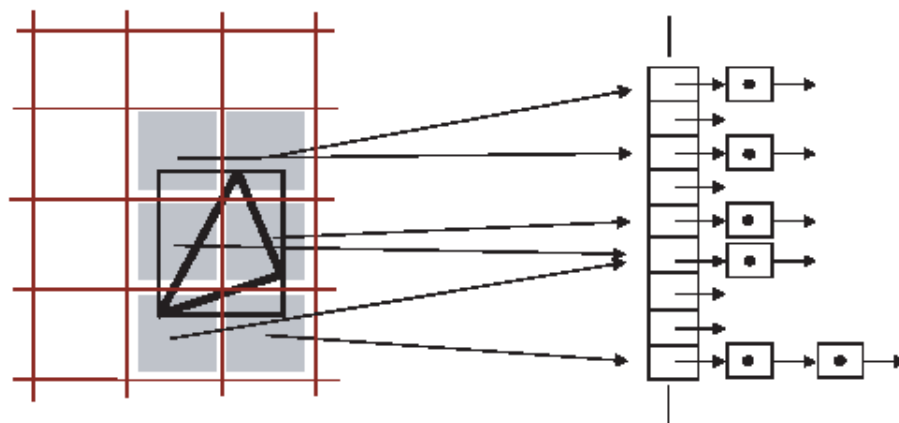


Abbildung 2.4: Hashingverfahren: Links sehen wir das Objekt im 2D-Raum und rechts die Projektion in die Hashingtabelle.

Abbildung 2.4 sehen wir ein Beispiel für den zweidimensionalen Raum. Sind in einer Zelle mehrere Körper vorhanden, so ist es möglich, dass dort eine Kollision stattfindet. Ist dies der Fall, so testet man die betroffenen Körper auf eine Durchdringung. Es wird ersichtlich, dass je kleiner die Gitternetze sind, umso weniger Tests notwendig sind, die auf keine Durchdringung resultieren. Aus diesem Grund versucht man die Zellen so klein wie möglich zu machen. Da aber die Verwaltung eines solchen Netzes sehr aufwendig werden kann, führt man das Hashingverfahren ein. Dieses Verfahren projiziert das Gitternetz in eine eindimensionale Datenstruktur. Dies wird mit einer bestimmten Hashingformel durchgeführt [Teso3]. Diese Projektion ist aber nicht eindeutig und deshalb ist es möglich, dass man zwei Körper testet, die nicht in der Nähe voneinander liegen. Aus diesem Grund muss man einen Trade-off zwischen der Größe der Zellen und der Hashingstruktur finden [Teso3]. Man hat zwar bei dieser Methode auch eine Vorverarbeitung der Körper, welche aber effizient gemacht werden kann und so zu einer Beschleunigung der Problemlösung führt.

2.1.2 Kontinuierliche Kollisionsdetektion

Nachdem wir die diskreten Ansätze der Kollisionsdetektion betrachtet haben, wollen wir uns nun den Kontinuierlichen zuwenden. In diesem Bereich wurde bis jetzt noch nicht so ausgiebig geforscht. Wir möchten hier einen Überblick über die schon bestehenden Arbeiten geben.

Es gibt eine Gruppe von Methoden, die auf starre Körper anwendbar sind. Eine davon wird in Redons paper [Redoo] vorgestellt. Die Bewegung der Körper wird hier in Rotationen und Translationen aufgeteilt, voraus verschiedene Fälle der Bewegungen und Berechnungen entstehen. Bei diesem Vorgehen werden die Kollisionen von Gerade/Gerade und Punkt/Gerade berechnet. Nach Aussagen von Redon ist dies eine geeignete Methode zur interaktiven Manipulation von sich langsam bewegenden Objekten. Des weiteren gibt es auch von Redon ein weiteres Verfahren, das im Paper [Redo2] vorgestellt wird. In dieser Methode wird mit Bounding Volumes gearbeitet und zwar mit OBBs. In beiden Ansätzen, wird ein beliebige aber konstante und starre Bewegung der Körper angenommen und umgesetzt. Es gibt noch einen weiteren Ansatz von Redon, in dem mit Swept Volumes [Redo4] gearbeitet wird. In dieser Methode wird angenommen, dass man alles Gelenkmodelle hat.

Ein ganz anderer Ansatz präsentiert Chin-Shyung in seinem Paper [Chigg]. Hier wird eine approximative, kontinuierliche Methode vorgestellt. Die Kollisionen werden hier zwischen Punkten und Dreiecken der Körper ermittelt. Eine Grundbedingung ist, dass sich der Punkt und das Dreieck aufeinander zubewegen. Um die Kollision zu berechnen, hält man zunächst den Punkt fest und lässt ihn mit dem Dreieck kollidieren. So bekommt man einen ersten Zeitpunkt t_1 . Nun hält man das Dreieck fest und lässt den Punkt damit kollidieren, was zu t_2 führt. Mit den beiden so erhaltenen Zeitpunkten und der Annahme von linearer Fortbewegung können wir mit folgender Gewichtung die Kollisionszeit berechnen:

$$t = \frac{t_1 \cdot t_2}{t_1 + t_2} \quad (2.1)$$

Gilt $0 \leq t \leq 1$ so ist t eine mögliche Lösung. Es muss aber noch getestet werden, ob der daraus resultierende Kollisionspunkt auf dem Kollisionsdreieck liegt. Ist dies der Fall, so hat man eine Kollision zwischen diesen beiden Körpern gefunden.

Als letztes möchten wir noch eine weitere Methode vorstellen, in der die Kollision analytisch berechnet wird. Auch hier betrachtet man die Kollision zwischen Punkten und Dreiecken. Durch lösen von Gleichung (2.2), können wir den Zeitpunkt einer Kollision bestimmen. Welcher in Abbildung 2.5 graphisch dargestellt ist.

$$P + V \cdot t = P_0 + V_0 \cdot t + ((P_1 - P_0) + (V_1 - V_0) \cdot t) \cdot u + ((P_2 - P_0) + (V_2 - V_0) \cdot t) \cdot v \quad (2.2)$$

Wobei P der Punkt ist und V seine Geschwindigkeit. P_0, P_1, P_2 sind die Punkte des Dreiecks und V_0, V_1, V_2 sind die dazugehörigen Geschwindigkeiten. u und v sind die Gewichtungen der baryzentrischen Koordinaten. Diese Gleichung (2.2) kann durch eine Bisektionsmethode gelöst werden. Gilt $0 \leq t \leq 1$ so haben wir eine mögliche Lösung. Wir müssen auch hier noch testen, ob der Kollisionspunkt tatsächlich auf dem Kollisionsdreieck zu liegen kommt.

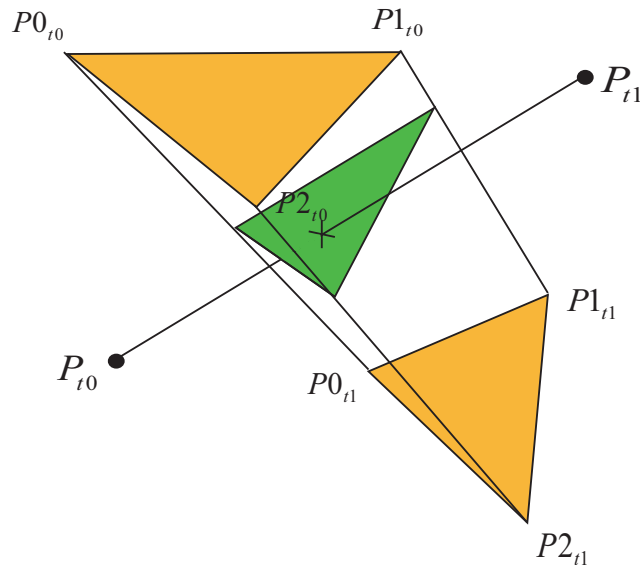


Abbildung 2.5: Wir sehen das Dreieck und den Punkt am Intervallanfang t_0 und am Intervallende t_1 . Das grüne Dreieck ist das Kollisionsdreieck, auf dem auch der Kollisionspunkt liegt.

2.1.3 Evaluierbare kontinuierliche Kollisionsdetektionen

Im Abschnitt 2.1.2 haben wir alle kontinuierlichen Kollisionsdetektionen beschrieben, die wir in unseren Recherchen gefunden haben. Da wir aber eine kontinuierliche, deformierbare Kollisionsdetektion entwickeln wollen, müssen wir die Ansätze die auf starren Körpern basieren verwerfen.

Wir haben beschlossen die analytische Methode sowie den Ansatz von Chin-Shyung [Chigg] zu implementieren und zu evaluieren. Die Chin-Shyung Methode mussten wir verwerfen, da die Annahme, dass die Punkte und Dreiecke aufeinander zugehen, im allgemeinen nicht zutrifft und wir somit keine ansprechenden Resultate erreichen konnten. Aus der Idee der Gewichtung von verschiedenen Zeiten, haben wir eine weitere Möglichkeit entwickelt. Wir nahmen an, dass die Dreiecke und deren Flächen sich in einem einzelnen Intervall im Grossen und Ganzen parallel bewegen. Wir berechneten die Zeitpunkte t_y , t_x der Durchstosspunkte der Geraden zwischen dem Ausgangspunkt und dem Endpunkt des in der Kollisionsdetektion involvierten einzelnen Punktes mit der Anfangs- und Schlussfläche des Dreiecks. Dies können wir in Abbildung 2.6 sehen. Da die Punkte sowie die Flächen sich genau im gleichen Zeitintervall bewegen und wir annahmen, dass sie sich linear und konstant bewegen, konnten wir aus den errechneten Zeitpunkten der Durchstösse der Geraden mit den Flächen und mit dem Verhältnis in der Gleichung (2.3) den Zeitpunkt der Kollision berechnen.

$$t = \frac{-t_y}{t_y - t_x + 1} \quad (2.3)$$

Leider entstanden Probleme, wenn einer der Zeitpunkte t_y oder t_x negativ war. Dies ist darauf zurückzuführen, dass wir im wesentlichen nur die Geschwindigkeit des Punktes in Betracht zogen, nicht aber die der Fläche. Aus diesem Grund konnte aus dem Bereich der negativen Zeiten keine korrekte Berechnung erfolgen.

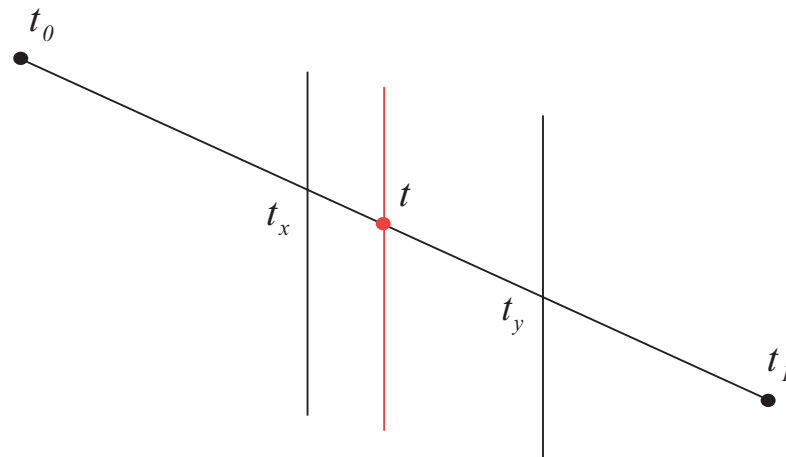


Abbildung 2.6: Wir sehen in dieser Abbildung die Ebenen des Dreiecks zu den Durchstoßzeitpunkten t_x und t_y . Wobei t_y für das Dreieck die Ausgangsposition ist und t_x die Endposition in diesem Zeitintervall darstellen.

Von diesem Ansatz nahmen wir aber die Idee mit, dass wir die Punkte nicht unbedingt mit den Dreiecken sondern auch mit den dazugehörigen Flächen schneiden können. Dies führt zu einer Einsparung von Ressourcen in der Kollisionszeitpunktberechnung. Man muss zwar testen ob das Resultat im Kollisionsdreieck liegt, dies war aber auch bis anhin erforderlich.

2.2 Kollisionsdetektion im DefCol-Projekt

Hier möchten wir die bereits bestehende diskrete Kollisionsdetektion des DefCol-Projektes vorstellen. Das Ziel des DefCol-Projektes ist es, eine interaktive Simulation von deformierbaren Körpern zur Verfügung zu stellen. Aus diesem Grund hat man sich für ein Hashingverfahren [Teso3] entschieden, bei dem die Punkte, wie auch die Tetraeder der verschiedenen Körper in einer Hashingtabelle auf Grund ihrer Position in der Umgebung gespeichert werden. Alle Punkte die in einer solchen Zelle liegen, werden auf ein Eindringen in die Tetraeder der gleichen Zelle getestet. Wird ein Eindringen festgestellt, so wird eine Kollisionsreaktion [Heio4] auf Grund der Tiefe des Eindringens ermittelt und ausgeführt. Wir entschieden uns diesen Aufbau beizubehalten und haben uns bei der Einführung unserer kontinuierlichen Kollisionsdetektion an diesem Vorgehen orientiert.

3

Realisierung

Diese Kaptiel erläutert, wie wir unsere kontinuierliche Kollisionsdetektion angewendet und implementiert haben. Wir werden dazu Schrittweise vorgehen und alle wichtigen Aspekte einzeln analysieren.

3.1 Übersicht

Wir haben uns an der diskreten Kollisionsdetektion orientiert und mussten als erstes ermitteln auf welcher Abstraktionsebene wir die Kollisionsdetektion durchführen wollten. In der diskreten Kollisionsdetektion werden die Kollisionen zwischen den Tetraeder und Punkten der Körper ermittelt. Da wir in der kontinuierlichen Variante auch noch die Bewegung der Körper in die Hashtabelle einbringen mussten, haben wir uns entschieden, die Kollision zwischen den Oberflächpunkten und den Oberflächendreiecken der Körper zu ermitteln. Um diese Bewegung zu erreichen, haben wir anstatt der Punkte Geraden eingefügt, die vom Anfangs- bis zum Endpunkt der Bewegung eines Punktes führten. Die Dreiecke haben wir mit dem gleichen Grundgedanken in die Hashtabelle eingefügt. Wir haben nach einem Körper gesucht, der den ganzen Raum abdeckt über den das Dreieck im Intervall gleitet. Dies war möglich, weil wir annahmen, dass zwischen den Intervallen alle Bewegungen linear sind. Nun mussten wir ermitteln, in welchen Hashzellen bewegte Punkte sowie bewegte Dreiecke vorhanden waren und konnten mit dem analytischen Verfahren den allfälligen Kollisionszeitpunkt ermitteln. Da das Hashingverfahren nicht eindeutig ist, musste kontrolliert werden, ob der Kollisionszeitpunkt auch wirklich zu einer gültigen Lösung führte. Dafür mussten wir testen, ob der ermittelte Kollisionspunkt auf dem Kollisionsdreieck lag. War dies der Fall, so hatten wir eine gültige Kollision gefunden.

3.2 Hashing von bewegten Punkten und Dreiecken

Das Hashingverfahren musste an die kontinuierliche Methode angepasst werden. Das heisst, dass wir ausser der Position der Punkte und Dreiecke auch noch ihre Bewegung einbringen mussten. Dies haben wir folgendermassen gelöst:

3.2.1 Bewegte Punkte

Nach der Annahme, dass die Bewegungen zwischen den Intervallen immer linear sind, konnten wir die Bewegung des Punktes hinzufügen, indem wir aus dem Punkt eine Gerade machten. Nun standen wir vor dem Problem, wie wir bestimmen konnten in welchen Gitterzellen diese Gerade vorhanden ist. Wir konnten dank eines Voxel

Traversieralgorithmen von Amanatides [Ama87] diese Gitterzellen ermitteln und die bewegten Punkte dementsprechend einfügen. Einen solchen bewegten Punkt und die von ihm beeinflussten Zellen können wir in Abbildung 3.1 sehen.

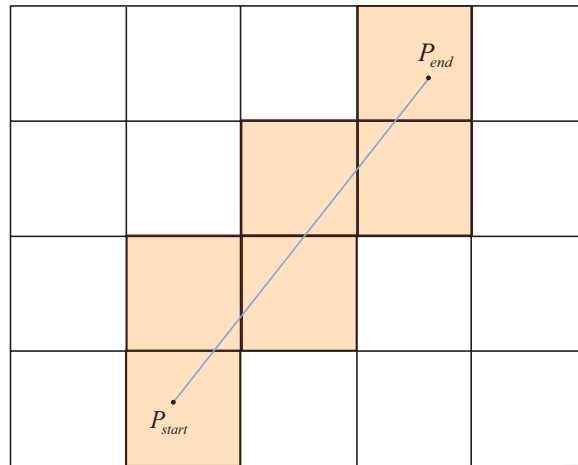


Abbildung 3.1: Hashingverfahren: Einfügen eines bewegten Punktes.

3.2.2 Bewegte Dreiecke

Das Einfügen der Dreiecke in die Hashtabelle stellte ein grösseres Problem dar, da wir mit einer beliebigen linearen Bewegung der drei Punkte rechnen mussten. Die einfachste Methode, die wir gefunden haben, war, dass wir eine Boundingbox über das Anfangs- und das Enddreieck gelegt haben. Das heisst, dass wir einfach mit den maximalen und minimalen Koordinatenwerten der Dreieckspunkte eine Boundingbox erstellt haben. Dies führte zu einem grösseren Volumen, das getestet werden musste, hatte aber den Vorteil, dass wir mit sehr wenig Rechenaufwand ein angemessenes Resultat erhalten konnten. Dieses Vorgehen können wir in Abbildung 3.2 sehen.

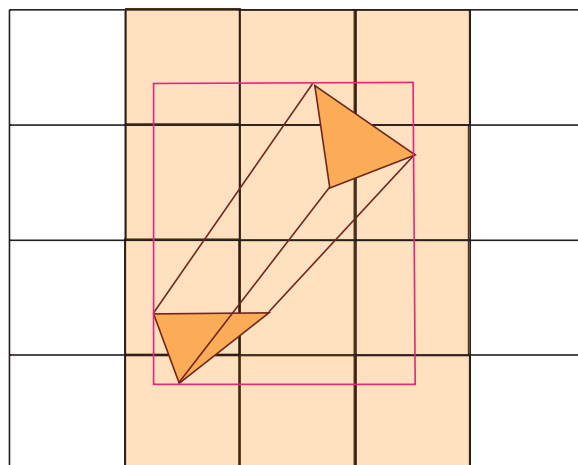


Abbildung 3.2: Hashingverfahren: Einfügen eines bewegten Dreieckes mit Hilfe einer Boundingbox.

Beim zweiten Ansatz haben wir probiert, das Volumen des umschliessenden Körpers zu verkleinern. Wir haben versucht, einen Schlauch um unser bewegtes Dreieck zu legen. Mit der Annahme, dass wir den grössten Abstand r zwischen zwei benachbarten Punkten der Körper einer Simulation kennen, haben wir nach einer solchen Lösung gesucht. Die erste Idee war, dass man um die Verbindungen der Anfangs- und Endpunkte des beweg-

ten Dreiecks einen Schlauch mit Radius r legten und dann das resultierende Schnittvolumen dieser drei Schläuche als den umschliessenden Körper betrachteten. Dies hat sich aber als ziemlich ineffizient erwiesen. Aus diesem Grund haben wir beschlossen nur einen Schlauch um die Verbindung zwischen Anfangs- und Endpunkt des Mittelpunktes des Dreieckes zu legen. Ein Beispiel können wir in Abbildung 3.3 sehen. Dank der

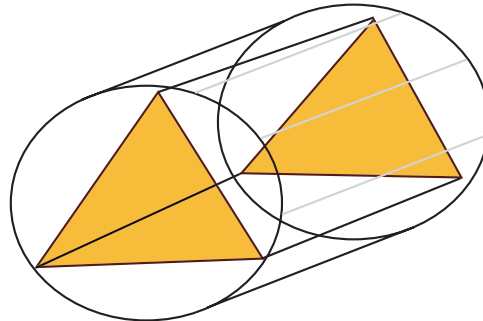


Abbildung 3.3: Schlauch um das bewegte Dreieck. Als Mittelachse wird die Verbindung der Dreiecksmittelpunkte genommen.

Annahme, dass wir r kennen konnten wir garantieren, dass sich das ganze bewegte Dreieck in diesem Schlauch befand. Wir hatten zwar mehr Volumen als im Ansatz mit den drei Schläuchen, aber dafür einen viel geringeren Rechenaufwand. Die Implementation dieses Ansatzes hat leider gezeigt, dass auch dieser Aufwand zur Berechnung des umschliessenden Körpers im Vergleich mit der Kollisionsdetektion viel zu gross ausfiel und die Simulation abbremste. Darum sind wir wieder auf unseren aller ersten Ansatz mit der Boundingbox über das Anfangs- und Enddreieck zurückgekehrt.

3.3 Vorverarbeitung der Kollisionspaare

Das Ziel des Hashverfahrens ist, nur Körperpaare zu testen, die wirklich eine Kollision haben könnten. Da aber das Hashverfahren nicht eindeutig ist und wir mit Zellen arbeiten, die aufgrund ihrer Lage und Grösse auch zwei Körper beinhalten können, die nicht kollidieren, bekommen wir immer noch viele Kollisionspaare, die gar keine sind. Da die Berechnung der endgültigen Kollisionsdetektion aufwendig ist, möchten wir noch mehr dieser fälschlichen Kollisionspaare ausschliessen. In diesem Abschnitt werden die Verfahren gezeigt, die wir eingesetzt haben, um dies zu erreichen.

3.3.1 BoundingBox

Wie in den diskreten Kollisionsmethoden kann auch hier eine BoundingBox erstellt werden. Diese BoundingBox wird über das ganze bewegte Dreieck gelegt. Der einfachste Weg dies zu tun ist, dass man mit den Maxima und Minima des bewegten Dreiecks im Koordinatensystem diese BoundingBox erstellt. Dies ist eine einfache und effiziente Art und Weise zum Ziel zu kommen.

3.3.2 Punkt in BoundingBox

Wenn wir eine BoundingBox über das bewegte Dreieck haben, so können wir testen, ob der dazugehörige bewegte Punkt sich jemals in diesem Bereich aufhält. Da wir testen müssen, ob der Punkt im gegebenen Zeitintervall die BoundingBox passiert, berechnen wir für jede Koordinate den Durchstosszeitpunkt mit dem Minimum und Maximum der

BoundingBox. Erhält man zwei negative Werte oder zwei Werte über Eins, so kann man sagen, dass sich dieser Punkt nie in der BoundingBox befindet. Sind die beiden Werte negativ, so bewegt sich der Punkt für diese Koordinate nur im Bereich vor dem Minimum. Sind beide Werte über Eins, so bewegt sich der Punkt nur im Bereich nach dem Maximum. Als ein weiterer Fall kann betrachtet werden, was passiert, wenn sich der Punkt in eine Koordinatenrichtung gar nicht bewegt. Trifft dies zu, muss er von Anfang an zwischen dem Minimum und dem Maximum dieser Koordinatenrichtung liegen, da er sonst auf Grund seiner Bewegung gar nie in die BoundingBox eindringen kann. Macht man diese Tests für alle Koordinatenrichtungen, kann man bestimmen, ob zwischen dem bewegten Punkt und dem bewegten Dreieck eine Kollision möglich ist.

3.3.3 Vermeidung mehrfacher Kollisionsdetektion

Durch das Einfügen der Punkte und Dreiecke mit Rücksichtnahme auf die Bewegung, kann es sein, dass diese Punkte und Dreiecke in mehrere Gitterzellen eingefügt werden. Es ist offensichtlich, dass es passieren kann, dass der gleiche Punkt und das gleiche Dreieck in mehreren Gitterzellen als potenzielle Kollisionspartner enthalten sind. Dies ist zum Beispiel der Fall, wenn sie den gleichen Weg beschreiten, aber nur zeitlich verschoben sind. Wir können dies in Abbildung 3.4 sehen. Nun wollen wir aber nicht in jeder

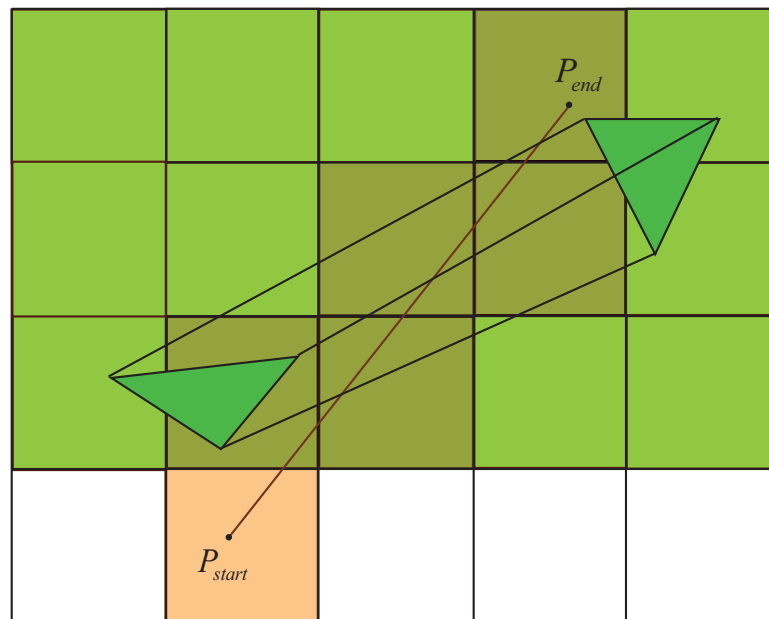


Abbildung 3.4: Die hellgrünen Felder sind durch das Dreieck belegt. Die rosaroten durch den Punkt und die dunkelgrünen von Punkt und Dreieck.

dieser Gitterzellen die ausführliche Kollisionsdetektion berechnen. Aus diesem Grund merken wir uns, wenn ein Punkt und ein Dreieck miteinander getestet wurden und können so das mehrfache Testen verhindern. Wir brauchen für diesen Vorgang zwar mehr Speicherplatz, aber verglichen mit dem Performanzgewinn durch die Einsparung des aufwendigen Kollisionstests wird das ohne Probleme wettgemacht.

3.4 Kontinuierliche Kollisionsdetektion

In diesem Abschnitt wollen wir die analytische Kollisionsdetektion vorstellen und analysieren. Will man diese Detektion berechnen, so muss eine Gleichung fünften Grades gelöst werden. Dies haben wir mit der Bisektionsmethode gelöst. Hat man einen potenziellen Zeitpunkt einer Kollision ermittelt, so muss noch getestet werden, ob dieser überhaupt ein korrektes Resultat darstellt. Diese Vorgehensweisen werden wir nun Schritt für Schritt beschreiben und erklären.

3.4.1 Bisektion

Hat man eine Funktion mit einem Grad höher als drei, so ist es nicht möglich mit einem analytischen Verfahren die Nullstellen zu bestimmen. Aus diesem Grund wurden verschiedene Verfahren entwickelt, die eine Approximation einer Nullstelle bestimmen können. Wir haben uns hier für das Bisektionsverfahren entschieden, da es mit erfüllten Anfangsbedingungen eine korrekte Lösung garantiert.

Die Idee des Bisektionverfahrens ist, dass wenn ein Nullpunkt vorhanden ist, man links und rechts davon ein positiver und ein negativer Teil sein muss. Hat man also einen Ausgangspunkt, indem man einen negativen und positiven Wert hat, so muss dazwischen ein Nullpunkt sein. Halbiert man nun das Intervall zwischen dem negativen und positiven Wert und testet diesen neuen Wert, so ersetzt man den positiven Wert durch den neuen Wert, sofern er auch positiv ist und sonst, falls er nicht Null ist, durch den negativen. So kann man sich durch Bisektion dem Nullpunkt immer mehr annähern. Die Ermittlung des neuen Punktes sehen wir in Abbildung 3.5. Als Abbruchbedingung kann man ein

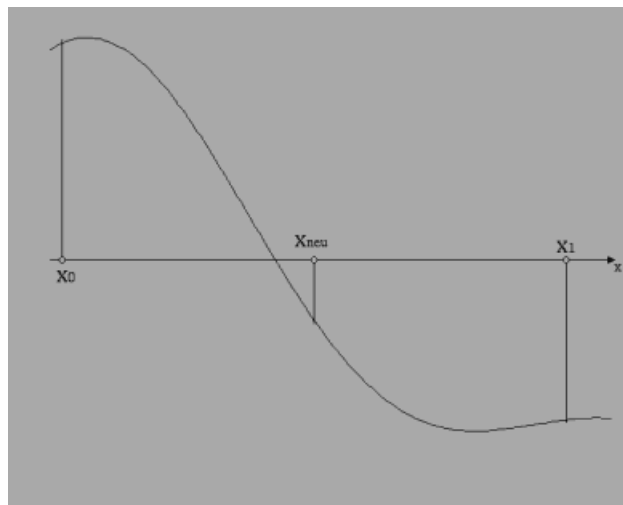


Abbildung 3.5: Bisektion: Der neue Punkt wird in diesem Fall zum neuen x_1 .

Mindestintervall angeben. Natürlich ist die Suche auch beendet, wenn der Nullpunkt gefunden ist. Um dieses Verfahren ausführen zu können, muss man aber eine solche Anfangsbedingung wie oben beschrieben erfüllen. In unserem Fall haben wir die Werte von Intervallanfang $t_A = 0$ und vom Intervallende $t_E = 1$ verglichen. Hatten beide das gleiche Vorzeichen, so haben wir t_E um 0,1 verkleinert und wieder die Vorzeichen der dazugehörigen Werte verglichen. Dies haben wir so lange gemacht, bis $t_E < 0$ galt oder die Vorzeichen der Werte verschieden waren. War die Abbruchbedingung $t_E < 0$ so wurde angenommen, dass es keine Kollision geben konnte. Ansonsten wurde mit den Werten t_A und t_E die Bisektion begonnen.

3.4.2 Kontinuierliche Kollisionsgleichung

Um den Zeitpunkt jeder Kollision zweier Körper in einem bestimmten Intervall zu detektieren, muss man einen kontinuierlichen Ansatz nehmen. Eine Möglichkeit ist, die Gleichung (3.1) zu lösen.

$$\mathbf{P} + \mathbf{V} \cdot t = \mathbf{P}_0 + \mathbf{V}_0 \cdot t + ((\mathbf{P}_1 - \mathbf{P}_0) + (\mathbf{V}_1 - \mathbf{V}_0) \cdot t) \cdot \mathbf{u} + ((\mathbf{P}_2 - \mathbf{P}_0) + (\mathbf{V}_2 - \mathbf{V}_0) \cdot t) \cdot \mathbf{v} \quad (3.1)$$

Wobei \mathbf{P} für den Punkt und \mathbf{V} für seine Geschwindigkeit stehen. Die Punkte des Dreiecks werden durch $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$ und ihre Geschwindigkeiten durch $\mathbf{V}_0, \mathbf{V}_1, \mathbf{V}_2$ dargestellt. Die beiden Unbekannten u und v stehen für die Gewichtungen der Vektoren $\mathbf{P}_0\mathbf{P}_1$ und $\mathbf{P}_0\mathbf{P}_2$. t steht für den Zeitpunkt der ermittelten potenziellen Kollision. Ordnen wir diese Gleichung um und betrachten die einzelnen Koordinaten, so erhalten wir drei lineare Gleichungen [Moo88]

$$\begin{aligned} \mathbf{a} \cdot \mathbf{u} + \mathbf{b} \cdot \mathbf{v} + \mathbf{c} \cdot t &= \mathbf{d} \\ \mathbf{e} \cdot \mathbf{u} + \mathbf{f} \cdot \mathbf{v} + \mathbf{g} \cdot t &= \mathbf{h} \\ \mathbf{i} \cdot \mathbf{u} + \mathbf{j} \cdot \mathbf{v} + \mathbf{k} \cdot t &= \mathbf{l} \end{aligned} \quad (3.2)$$

wobei

$$\begin{aligned} \mathbf{a} &= (\mathbf{P}_{1x} - \mathbf{P}_{0x}) + t \cdot (\mathbf{V}_{1x} - \mathbf{V}_{0x}) \\ \mathbf{b} &= (\mathbf{P}_{2x} - \mathbf{P}_{0x}) + t \cdot (\mathbf{V}_{2x} - \mathbf{V}_{0x}) \\ \mathbf{c} &= -\mathbf{V}_x \\ \mathbf{d} &= \mathbf{P}_x - (\mathbf{P}_{0x} + t \cdot \mathbf{V}_{0x}) \\ \mathbf{e} &= (\mathbf{P}_{1y} - \mathbf{P}_{0y}) + t \cdot (\mathbf{V}_{1y} - \mathbf{V}_{0y}) \\ \mathbf{f} &= (\mathbf{P}_{2y} - \mathbf{P}_{0y}) + t \cdot (\mathbf{V}_{2y} - \mathbf{V}_{0y}) \\ \mathbf{g} &= -\mathbf{V}_y \\ \mathbf{h} &= \mathbf{P}_y - (\mathbf{P}_{0y} + t \cdot \mathbf{V}_{0y}) \\ \mathbf{i} &= (\mathbf{P}_{1z} - \mathbf{P}_{0z}) + t \cdot (\mathbf{V}_{1z} - \mathbf{V}_{0z}) \\ \mathbf{j} &= (\mathbf{P}_{2z} - \mathbf{P}_{0z}) + t \cdot (\mathbf{V}_{2z} - \mathbf{V}_{0z}) \\ \mathbf{k} &= -\mathbf{V}_z \\ \mathbf{l} &= \mathbf{P}_z - (\mathbf{P}_{0z} + t \cdot \mathbf{V}_{0z}) \end{aligned} \quad (3.3)$$

Die \mathbf{P}_{iw} und \mathbf{V}_{iw} sind die Positionen und Geschwindigkeiten der Komponenten der Punkte des Dreiecks und die \mathbf{P}_w und \mathbf{V}_w sind die Positionen und Geschwindigkeiten der Komponenten des Punktes. Die Geschwindigkeiten stellen die Distanz des Weges dar, die ein Punkt in einem Intervall hinter sich bringt. Dieses Gleichungssystem kann auf Gleichung (3.4) vereinfacht werden.

$$\frac{(\mathbf{c}t(\mathbf{j}e - \mathbf{i}f) + \mathbf{f}a(\mathbf{t}k - \mathbf{l}) + \mathbf{f}d\mathbf{i} + \mathbf{h}(\mathbf{j}a - \mathbf{i}b) + \mathbf{e}(\mathbf{b}l - \mathbf{d}j - \mathbf{t}k\mathbf{b}) - \mathbf{t}g(\mathbf{j}a + \mathbf{i}b))\mathbf{c}}{\mathbf{i}(\mathbf{g}b - \mathbf{c}f) + \mathbf{k}(\mathbf{a}f - \mathbf{b}e) + \mathbf{j}(\mathbf{c}e - \mathbf{g}a)} = 0 \quad (3.4)$$

Dies ist nun eine Gleichung dritten Grades. Auf Grund der in unserem Fall komplizierten analytischen Lösung bedienen wir uns vorerst der Bisektionsmethode und zwar über dem Intervall $0 \leq t \leq 1$.

Möchte man diese Gleichung analytisch lösen, so ist die Formel von Cardano ein möglicher Ansatz. Hat man die Gleichung (3.5) so setzt man $y = t + b/3a$ und es entsteht die Gleichung (3.6). Die Anzahl der Lösungen hängt von der Diskriminanten $D = q^2 + p^3$ ab. Ist $D < 0$ so erhält man drei verschiedene reelle Lösungen. Gilt $D > 0$, ergeben sich eine reelle und zwei komplexe Lösungen. Ist $D = 0$ und $p = q = 0$ so erhalten wir für alle drei Lösungen Null. Ist aber $p^3 = -q^2 \neq 0$, so erhalten wir zwei Lösungen. Die Lösungen erhalten wir durch die Gleichungen (3.7). Wobei $u = \sqrt[3]{-q + \sqrt{D}}$, $v = \sqrt[3]{-q - \sqrt{D}}$ und $f_{1,2}$ die Lösungen der Gleichung $f^2 + f + 1 = 0$, d.h. $f_{1,2} = 0.5(-1 \pm i\sqrt{3})$ sind.

$$at^3 + bt^2 + ct + d = 0 \quad (3.5)$$

$$y^3 + 3py + 2q = 0 \quad \text{mit} \quad 3p = \frac{3ac - b^2}{3a^2} \quad 2q = \frac{2b^3}{27a^3} - \frac{bc}{3a^2} + \frac{d}{a} \quad (3.6)$$

$$y_1 = u + v \quad y_2 = f_1 u + f_2 v \quad y_3 = f_1 v + f_2 u \quad (3.7)$$

Betrachtet man diese Gleichung (3.4) genauer, wird ersichtlich, dass der Nenner Null sein kann. Ist dies der Fall, so bekommen wir ein nichtssagendes Resultat. Aus diesem Grund mussten wir die Fälle suchen, in denen der Nenner Null werden konnte. Unsere Berechnungen haben ergeben, dass der Nenner Null ist, wenn der Punkt P sich nicht bewegt. Ist dies der Fall, so sind c , g und k gleich Null und somit auch der Nenner. Um dies zu umgehen, mussten wir einen anderen Weg finden, um t zu berechnen. Wir haben uns dazu entschieden folgendes Gleichungssystem 3.8 zu lösen

$$\begin{aligned} P_x + V_x \cdot t &= P_{0x} + V_{0x} + (P_{10x} + V_{10x} \cdot t) \cdot u + (P_{20x} + V_{20x} \cdot t) \cdot w \\ P_y + V_y \cdot t &= P_{0y} + V_{0y} + (P_{10y} + V_{10y} \cdot t) \cdot u + (P_{20y} + V_{20y} \cdot t) \cdot w \\ P_z + V_z \cdot t &= P_{0z} + V_{0z} + (P_{10z} + V_{10z} \cdot t) \cdot u + (P_{20z} + V_{20z} \cdot t) \cdot w \end{aligned} \quad (3.8)$$

Wobei $P_{10} = P_1 - P_0$ und $P_{20} = P_2 - P_0$ sowie $V_{10} = V_1 - V_0$ und $V_{20} = V_2 - V_0$ gelten. Wir haben dieses Gleichungssystem nach u und dann nach w aufgelöst. Da durch dieses Auflösen wieder Divisionen entstanden, mussten wir testen, ob bei diesen Divisionen wieder durch Null geteilt wird. Aus diesem Grund haben wir einen Algorithmus entwickelt, der die verschiedenen Arten der Auflösung dieser Gleichung betrachtet und die erste nimmt, bei der nie durch Null geteilt wird. Als Beispiel zeigen wir in der Gleichung (3.9) die Auflösung nach u in den z -Koordinaten und die anschließende Auflösung nach w in den y -Koordinaten.

$$\begin{aligned} w &= \frac{P_y + V_y \cdot t - (P_{0y} + V_{0y} \cdot t) - (P_{10y} + V_{10y} \cdot t) \cdot \frac{P_z + V_z \cdot t - (P_{0z} + V_{0z} \cdot t)}{P_{10z} + V_{10z} \cdot t}}{(P_{20y} + V_{20y} \cdot t) - (P_{10y} + V_{10y} \cdot t) \cdot \frac{P_{20z} + V_{20z} \cdot t}{P_{10z} + V_{10z} \cdot t}} \\ u &= \frac{P_z + V_z \cdot t - (P_{0z} + V_{0z} \cdot t) - (P_{20z} + V_{20z} \cdot t) \cdot w}{P_{10z} + V_{10z} \cdot t} \end{aligned} \quad (3.9)$$

Die beiden Nenner dieser Gleichungen müssen ungleich Null sein. Sind sie es doch, so muss eine andere Kombination der Zeilen des Gleichungssystems zur Lösung beigezogen werden. Mit dieser Variante gehen wir sicher, dass wir mit unserer Berechnung keine unendlich grossen Zahlen bekommen.

Wir haben jetzt also den Spezialfall gesehen, in dem sich der Punkt nicht bewegt. Nun wollen wir den Fall betrachten, in dem sich das Dreieck nicht bewegt. Bewegt sich der Punkt auch nicht, so können wir gleich sagen, dass es keine Kollision geben kann. Ausserdem können wir bei Dreiecken, die parallel zu einer Koordinatenfläche liegen den allfälligen Durchstosszeitpunkt wie in Gleichung (3.10) für die z-Koordinate einfach berechnen.

$$t = \frac{P_{0z} - P_z}{V_z} \quad (3.10)$$

Dieser Zeitpunkt ist zwar nicht unbedingt der Zeitpunkt des Schnittpunktes mit dem Dreieck, sondern der Zeitpunkt des Schnittpunktes mit der Ebene der Dreiecks. Nun muss nur noch getestet werden, ob dieser Schnittpunkt nicht nur auf der Ebene sondern auch im Dreieck liegt. Im dritten Fall haben wir ein beliebiges Dreieck, das sich nicht bewegt und einen Punkt, der sich beliebig bewegt. Wir haben also Gleichung (3.11):

$$\mathbf{P} + \mathbf{V} \cdot t = \mathbf{P}_0 + \mathbf{P}_1 \cdot u + \mathbf{P}_2 \cdot w \quad (3.11)$$

Da nur noch V abhängig von t ist, können wir diese Gleichung lösen, indem wir die Koordinatengleichung der durch das Dreieck aufgespannten Ebene erstellen und t so bestimmen, dass der Abstand der Punktes P zur Zeit t zur Ebene gleich Null ist.

Bei allen Varianten gilt, dass wenn man ein Resultat für t erhält für das $0 \leq t \leq 1$ gelten muss. Ist dies der Fall, so hat man eine mögliche Kollision gefunden. Es muss aber trotzdem noch ermittelt werden, ob dieses Resultat auch wirklich eine Kollision zwischen dem Punkt und dem Dreieck darstellt. Dies führen wir mit einem Existenztest durch. Dieser überprüft ob der Kollisionspunkt, der sich aus t ergibt auch wirklich auf dem Kollisionsdreieck liegt.

3.4.3 Existenztest

Es gibt verschiedene Arten, wie man ermitteln kann, ob ein Punkt in einem Dreieck liegt. Wir haben zwei davon genauer betrachtet.

Die erste Methode testet, ob der gesuchte Punkt auf der richtigen Seite aller drei Begrenzungslinien des Dreiecks liegt. Um dies zu ermitteln, berechnet man die Koordinatenform der Geraden und testet, ob der gesuchte Punkt auf der gleichen Seite liegt wie der gegenüber liegende Punkt des Dreiecks. Hat man die Koordinatenform der Geraden, so muss man nur noch die Punkte einsetzen und vergleichen, ob sie das gleiche Vorzeichen haben. Ist dies bei allen drei Geraden der Fall, so liegt der gesuchte Punkt im Dreieck. Wir mussten bei unseren Tests feststellen, dass diese Methode durch die numerische Berechnung zu grosse Abweichungen hatte.

Aus diesem Grund haben wir uns für die Methode mit den baryzentrischen Koordinaten entschieden. Mit den baryzentrischen Koordinaten kann man ermitteln, ob ein Punkt in einem Dreieck liegt, in dem man den Punkt durch die Gewichtung der drei Punkte des Dreiecks erreicht. Diese Vorgehen können wir in Abbildung 3.6 sehen. Sind diese drei Gewichtungen alle zwischen Null und Eins und addieren sich auf Eins auf, dann liegt der Punkt im Dreieck. Man muss also die Gleichungen (3.12) lösen.

$$\mathbf{P} = a \cdot \mathbf{P}_0 + b \cdot \mathbf{P}_1 + c \cdot \mathbf{P}_2 \quad (3.12)$$

Teilen wir das Ganze auf die einzelnen Koordinaten auf, so haben wir ein Gleichungssystem mit drei Unbekannte und drei Gleichungen, was einfach gelöst werden kann. Zum

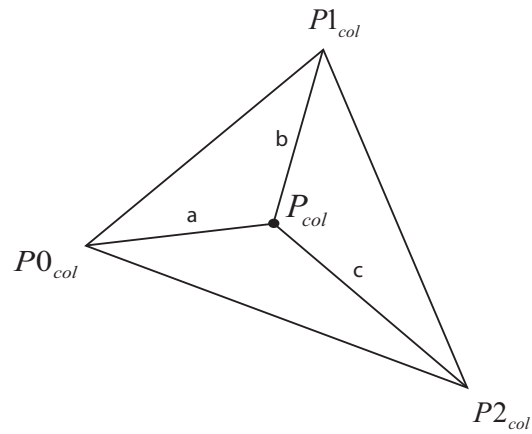


Abbildung 3.6: Der Kollisionspunkt wird durch die Dreieckspunkte mit den Gewichtungen a, b, c dargestellt.

Schluss muss nur noch getestet werden, ob a, b und c zwischen Null und Eins liegen und ob $a + b + c = 1$ gilt. Ist dies der Fall, so haben wir eine gültige Kollision gefunden.

4

Resultate und Ausblick

In diesem Kapitel werden wir unseren kontinuierlichen Kollisionsalgorithmus testen und mit dem vorher schon vorhandenen diskreten Kollisionsalgorithmus vergleichen. Wir werden uns dabei auf die Performanz aber auch auf die qualitativen Resultate konzentrieren. Im letzten Abschnitt wird ein Fazit über diese Arbeit verfasst und mögliche weiterführende Arbeiten vorgestellt und motiviert.

4.1 Vergleich der kontinuierliche und diskrete Kollisionsdetektion

In diesem Abschnitt werden wir aufgrund verschiedener Simulationen die Vor- und Nachteile des diskreten sowie des kontinuierlichen Ansatzes zeigen.

4.1.1 Einfache Simulation mit tiefen relativen Geschwindigkeiten

Wir werden in diesem Test ein Tetraeder mit einer relativ zur Intervalllänge langsamen Geschwindigkeit auf eine feste Membran fallen lassen. Dabei wird die Zeit gemessen, die der diskrete und der kontinuierliche Kollisionsalgorithmus für die Ausführung der Kollisionsdetektion, der Deformation und der Kollisionsantwort über alle Objekte in einem Zeitintervall brauchen. Die Kollisionsdetektion umfasst im kontinuierlichen Ansatz das Vorgehen, das wir im Kapitel 3 gesehen haben und uns einen genauen Kollisionszeitpunkt sowie den Kollisionspunkt zurückgibt. Im diskreten Ansatz wird das Eindringen eines Punktes in ein Tetraeder ermittelt und das Eindringen bejaht oder verneint. Wir wissen also nicht genau, wann die Kollision passiert und auch nicht wo. Die Deformation wird in beiden Ansätzen gleich gehandhabt doch wollen wir hier nicht genauer darauf eingehen. Für die ersten Tests haben wir auch die Kollisionsantwort vereinfacht. Wir stoppen die Simulation sobald eine Kollision entdeckt wird und zwar an dem Ort an dem sie ermittelt wurde, das heisst, dass im kontinuierlichen Fall auf dem korrekten Kollisionspunkt gestoppt wird und im diskreten Fall am Schluss des Intervalls, in dem die Kollision entdeckt wurde.

In Abbildung 4.1 sehen wir die Ausgangslage für unseren Test. Wir lassen das Tetraeder fallen und erwarten, dass der Aufprall auf die Membran bemerkt wird und die Simulation stoppt. In dieser Simulation werden 484 Oberflächendreiecke in die Kollisionsdetektion verwickelt sein. Wir lassen nun diesen Test mit beiden Algorithmen laufen und wir sehen in Abbildung 4.2 die Endresultate. Es ist ersichtlich, dass beide Algorithmen die Kollision entdeckt haben und am korrekten Ort angehalten haben. Nun wollen wir einen Blick auf die aufgewendete Zeit werfen. In Abbildung 4.3 sind die gemessenen Zeiten in einem logarithmischen Plot dargestellt. Wir sehen, dass der konti-

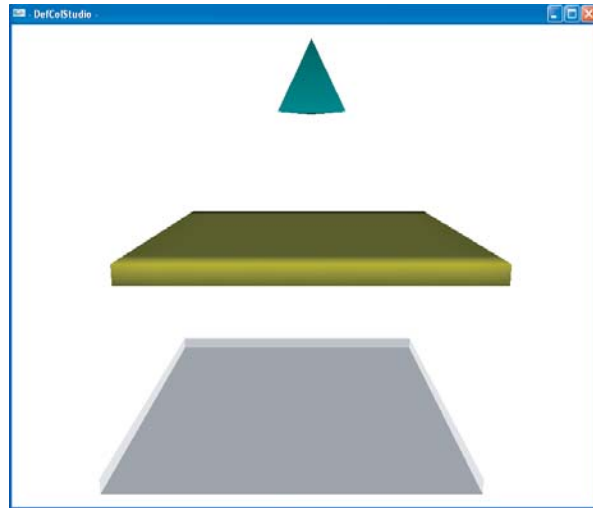


Abbildung 4.1: Tetraeder-Membran-Test: Ausgangslage.

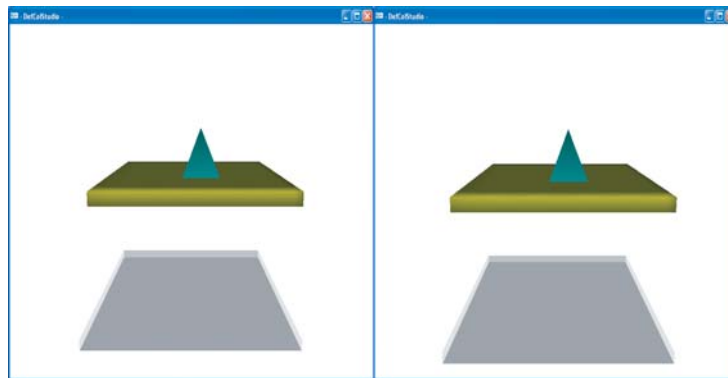


Abbildung 4.2: Tetraeder-Membran-Test: Simulationsresultat für kontinuierlichen (links) und diskreten Ansatz (rechts) mit einem relativ langsam fallenden Tetraeder.

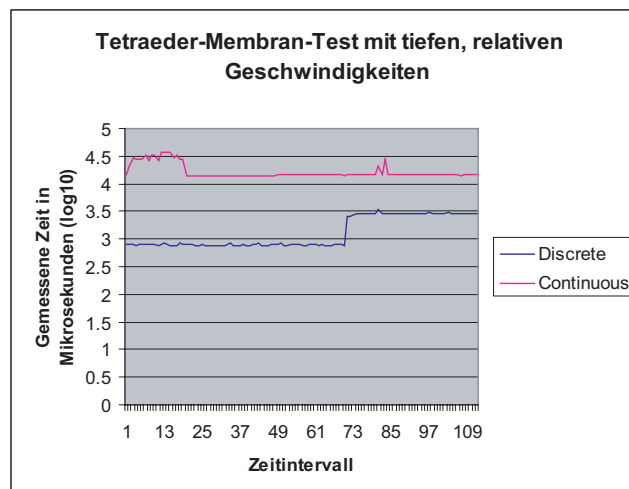


Abbildung 4.3: Tetraeder-Membran-Test: Logarithmischer Plot der gemessenen Zeit über die Zeitintervalle.

nuierliche Algorithmus immer langsamer ist. Die Kollision findet im Bereich von Zeitintervall 80 statt. Der kontinuierlichen Ansatz führt die Detektion nur einmal pro Punkt-

Dreieck-Paar durch. Der diskrete Ansatz jedoch ermittelt das Eindringen des Punktes in das Tetraeder immer wieder, bis er wieder ausgetreten ist. In Tabelle 4.1 sind die maximalen und minimalen Werte dieser Messung ersichtlich. Aus den ermittelten Durchschnitten wird hergeleitet, dass der diskrete Ansatz im Durchschnitt etwa 11 mal schneller ist als der kontinuierliche. Es darf nicht vergessen werden, dass im kontinuierlichen Ansatz nach dieser Zeit mehr Informationen zur Verfügung steht als mit dem diskreten Ansatz. Wir wissen nämlich genau wo und zu welchem Zeitpunkt die Kollision stattfindet.

Tabelle 4.1: Tetraeder-Membran-Test: relativ langsame Bewegungen

	Minimum	Maximum	Durchschnitt
Kontinuierlicher Algorithmus	13706 μ s	36518 μ s	17080 μ s
Diskreter Algorithmus	753 μ s	3369 μ s	1565 μ s

4.1.2 Einfache Simulation mit hohen relativen Geschwindigkeiten

Es wird noch einmal die gleiche Simulation wie im letzten Abschnitt durchgeführt. Dieses mal werden wir aber unser Tetraeder mit einer relativ zur Intervalllänge hohen Geschwindigkeit fallen lassen. Das Ziel dieser Beschleunigung ist zu zeigen, dass der diskrete Kollisionsalgorithmus mit schnellen Bewegungen, genauer gesagt mit schnellen Bewegungen relativ zur Zeitintervallgröße, Probleme bekommen kann. Die Ausgangslage bleibt gleich und wir werden auch die gleichen Messungen durchführen. In Abbildung 4.4 sehen wir das optische Resultat dieser Kollisionsdetektion. Es ist ersicht-

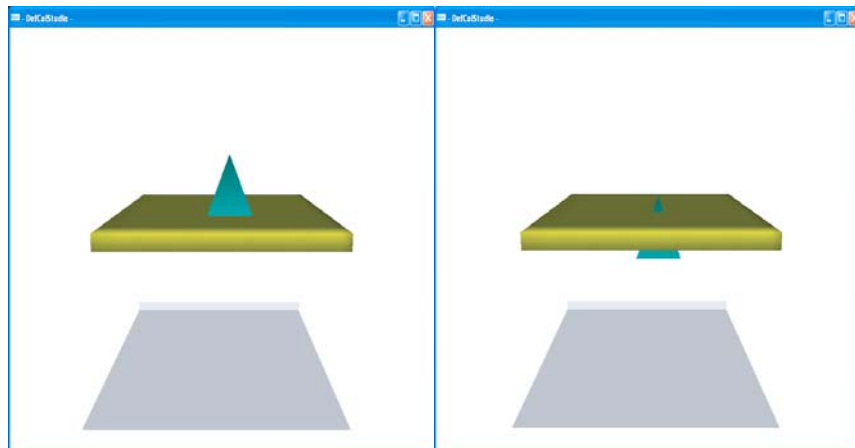


Abbildung 4.4: Tetraeder-Membran-Test: Simulationsergebnis für kontinuierlichen (links) und diskreten Ansatz (rechts) mit einem relativ schnell fallenden Tetraeder.

lich, dass der kontinuierliche Algorithmus wieder auf das gleiche Resultat kommt wie schon mit dem langsamen Tetraeder. Der diskrete Ansatz hat die Kollision zwar erkannt, aber die Durchdringung ist schon so weit fortgeschritten, dass es schwer werden könnte eine angemessene Kollisionsantwort durchzuführen. Schalten wir die ansonsten gebräuchliche Kollisionsantwort für den diskreten Kollisionsalgorithmus ein, so sehen wir, dass das Tetraeder trotz erkannter Kollision durch die Membran fällt. In diesem Fall

ist die Information der Durchdringung ohne genauen Zeitpunkt oder Ort nicht von Nutzen. Wir können jetzt noch nicht sagen, ob es im kontinuierlichen Ansatz eine genug gute Kollisionsantwort geben wird, die auf diese Situation angemessen reagiert. Es kann aber gesagt werden, dass es mit der zusätzlichen Information, einfacher ist die Reaktion in die richtige Richtung zu entwickeln. Nun wollen wir aber trotzdem noch die Zeitmessungen betrachten, die im logarithmischen Plot in Abbildung 4.5 zu sehen ist. Wieder

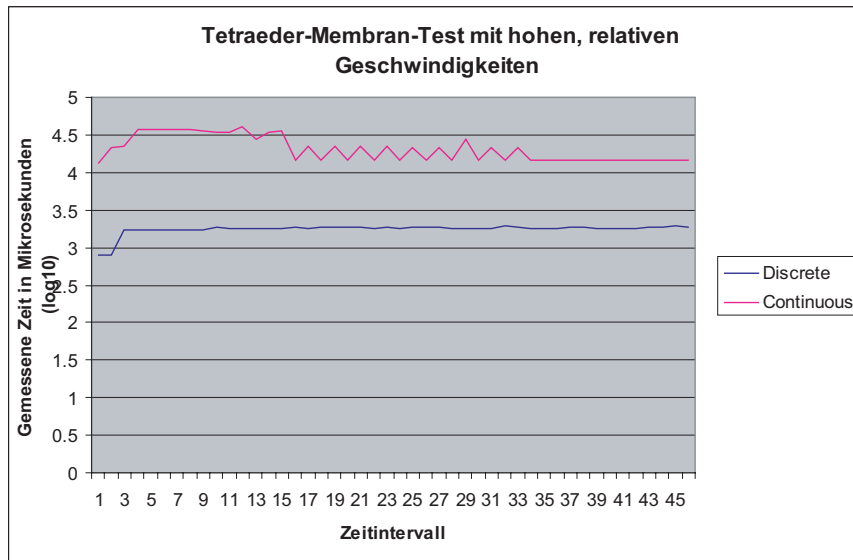


Abbildung 4.5: Tetraeder-Membran-Test: Logarithmischer Plot der gemessenen Zeit über die Zeitintervalle.

können wir daraus folgern, dass der kontinuierliche Ansatz langsamer ist als der diskrete. Betrachten wir die Eckdaten in Tabelle 4.2 sieht man, dass der diskrete Algorithmus im

Tabelle 4.2: Tetraeder-Membran-Test: relativ schnelle Bewegungen

	Minimum	Maximum	Durchschnitt
Kontinuierlicher Algorithmus	13480 μs	40742 μs	17958 μs
Diskreter Algorithmus	784 μs	2003 μs	1810 μs

Durchschnitt etwa 10 mal schneller ist. Auch hier muss dieses Resultat mit Rücksicht auf die zusätzlichen Informationen des kontinuierlichen Resultates und des weiter oben gezeigten optischen Resultates interpretiert werden.

4.1.3 Einfache Simulation mit hohen Geschwindigkeiten

Diesmal werden wir die Simulation mit einer Geschwindigkeit durchführen, die garantiert, dass die Kollision des Tetraeders mit der Membran zwischen zwei Tests im diskreten Algorithmus liegt. Darum können wir davon ausgehen, dass die Kollision vom diskreten Ansatz nicht erkannt wird. Wir wollen nun untersuchen, was für ein Resultat wir mit dem kontinuierlichen Ansatz erhalten. Dafür gehen wir wieder genau gleich vor. Für das optische Resultat stoppen wir die Simulation sobald das Tetraeder durch die Membran durchgefallen ist, damit wir den Unterschied der Resultate der Ansätze in Abbildung 4.6 sehen können. Wir sehen, dass das Tetraeder beim diskreten Fall durch die

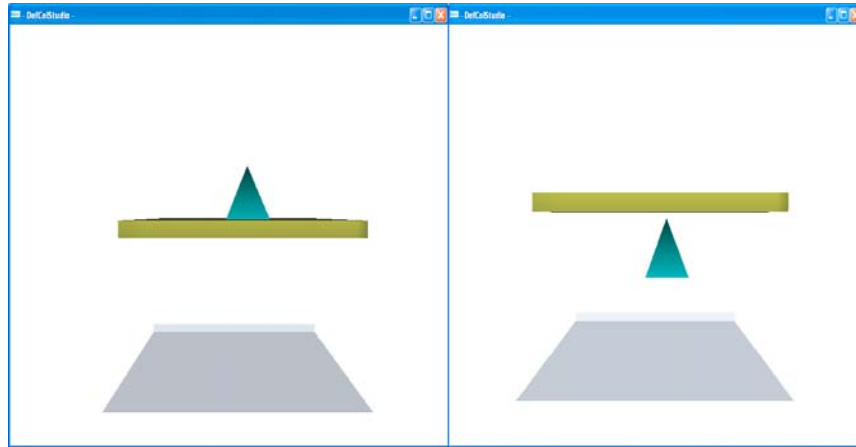


Abbildung 4.6: Tetraeder-Membran-Test: Simulationsresultat für kontinuierlichen (links) und diskreten Ansatz (rechts) mit einem schnell fallenden Tetraeder

Membran durchfällt. Im kontinuierlichen wird aber die Kollision erkannt und das Tetraeder hält auf der Membran. Wir können also dieses Problem aus dem diskreten Ansatz durch den kontinuierlichen Ansatz lösen. Nun wollen wir auch noch die Zeitmessungen betrachten, obwohl diese mit Vorsicht zu geniessen sind, da im diskreten Fall keine Kollision erkannt wurde und somit die aufwendigeren Berechnung nie durchgeführt werden mussten. Im kontinuierlichen Fall aber mussten alle Berechnungen durchgeführt werden und darum haben wir hier eine noch grössere Abweichung in der Effizienz der beiden Ansätze. Dies können wir im logarithmischen Plot in Abbildung 4.7 sehen. Die

Tabelle 4.3: Tetraeder-Membran-Test: hohen Geschwindigkeiten

	Minimum	Maximum	Durchschnitt
Kontinuierlicher Algorithmus	13385 μs	40386 μs	18866 μs
Diskreter Algorithmus	(752 μs)	(859 μs)	(782 μs)

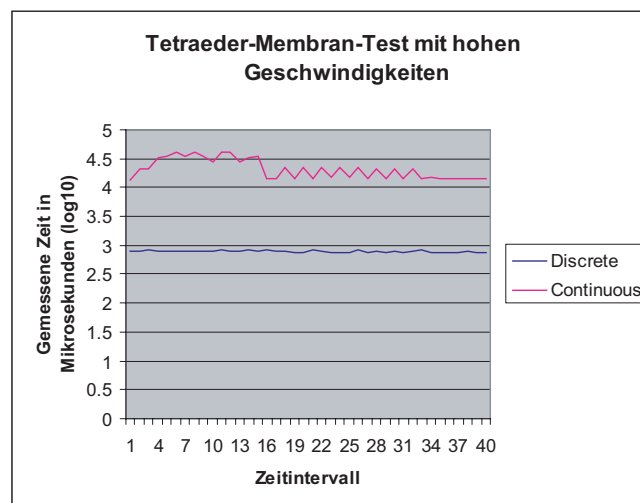


Abbildung 4.7: Tetraeder-Membran-Test: Logarithmischer Plot der gemessenen Zeit über die Zeitintervalle.

Eckdaten sehen wir in Tabelle 4.3. Aus diesen folgt, dass der diskrete Ansatz in diesem Beispiel im Durchschnitt 24 mal schneller ist.

4.1.4 Komplexe Simulation mit tiefen relativen Geschwindigkeiten

Wir werden nun noch das Verhalten der Algorithmen mit einer komplexeren Simulation betrachten und ihre Resultate miteinander vergleichen. Dazu wird eine Simulation durchgeführt, in der fünf Teddybären aufeinander fallen gelassen werden, die zusammen 2860 Oberflächendreiecke haben. Dies sind etwa sechs mal mehr interagierende Dreiecke als in der einfachen Simulation. Wir sehen in Abbildung 4.8 die Anfangsposition



Abbildung 4.8: Anfangsposition der Simulation mit fünf Teddybären die zusammen 2860 Oberflächendreiecke enthalten.

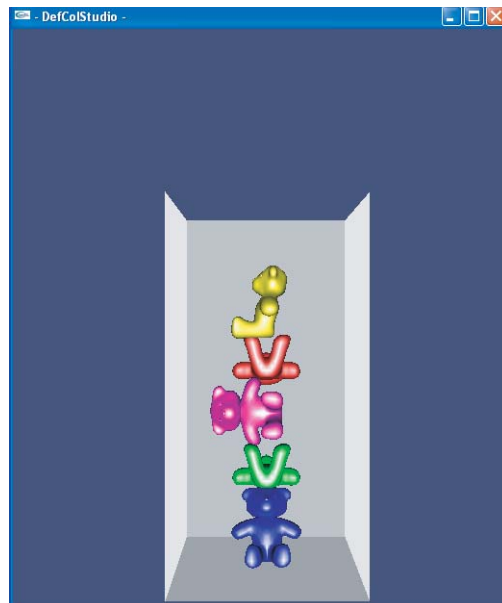


Abbildung 4.9: Endstadium der Bären nach ihrem Fall und der Einfrierung nach einer entdeckten Kollision.

der Objekte. Wir lassen die Simulation laufen und die Bären fallen durch die Gravitationskraft senkrecht nach unten. Sobald eine Kollision entdeckt wird, werden die interagieren-

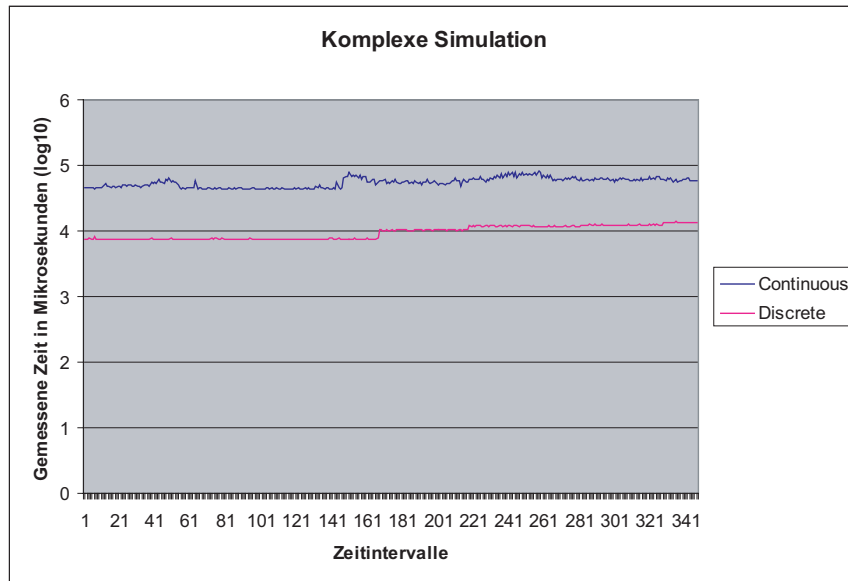


Abbildung 4.10: Komplexe Simulation: Logarithmischer Plot der gemessenen Zeit über die Zeitintervalle

den Bären gestoppt und in dieser Position festgehalten. Das optische Resultat der kontinuierlichen Kollisionsdetektion kann in Abbildung 4.9 betrachtet werden, welches genau gleich aussieht wie das der diskreten Kollisionsdetektion. Es wurden alle Kollisionen erkannt und somit ist Kollisionserkennung nicht von der Komplexität der Simulation abhängig. Wir haben für diese Simulation die gleiche Zeitmessung angewandt und auf einen logarithmischen Plot aufgetragen, den wir in Abbildung 4.10 sehen können. Aus diesem Plot und aus der Tabelle 4.4 können wir herauslesen, dass der kontinuierliche

Tabelle 4.4: Komplexe Simulation

	Minimum	Maximum	Durchschnitt
Kontinuierlicher Algorithmus	43684 μs	81341 μs	55547 μs
Diskreter Algorithmus	7312 μs	13803 μs	9724 μs

Algorithmus auch in dieser Simulation langsamer ist. Der diskrete Kollisionsalgorithmus ist im Durchschnitt etwa sechs mal schneller als die kontinuierliche Variante. Weiter können wir sehen, dass bei dieser Anzahl an Oberflächendreiecken das Preprocessing einen grossen Anteil an der gemessenen Zeit ausmacht. Und so die Kollisionen nicht mehr so fest ins Gewicht fallen wie bei einfacheren Simulationen. Da aber die Simulationen während der Perioden ohne Kollisionen schneller laufen, können wir sagen, dass das Preprocessing zwar teuer ist, aber uns vor noch viel grösseren Kosten mit der Kollisionsdetektion bewahrt.

4.2 Fazit und Ausblick

Wir haben in dieser Arbeit eine kontinuierliche Kollisionsdetektion eingeführt. Dazu haben wir verschiedene Ansätze aus früheren Arbeiten sowie eigene Ideen angeschaut, zum Teil implementiert und getestet. Aus den gewonnenen Erkenntnissen haben wir uns

für den in diesem Bericht vorgestellten Algorithmus entschieden. Da unsere Kollisionsdetektion aufwendig ist, haben wir durch Einführen verschiedener Vorverarbeitungsverfahren versucht alle irrelevanten Kollisionspaare herauszufiltern. So konnten wir unnötige Kosten beseitigen. Aus unseren Tests konnten wir schliessen, dass wir ein Verfahren entwickelt haben, dass langsamer ist als das vorherige diskrete Verfahren. Wir konnten aber zeigen, dass unser neues Verfahren, die Nachteile des diskreten Verfahrens aufhebt und genauere Angaben über die Kollisionszeit und deren Ort geben kann.

Bei unserer Implementation haben wir die Lösung durch das Bisektionsverfahren errechnet. Da die Gleichung dritten Grades ist, kann man sie aber auch analytisch ermitteln. Ein Ansatz dafür wurde im Abschnitt 3.4.2 erläutert. In einer weiteren Arbeit könnte dieses Vorgehen genauer analysiert und implementiert werden.

Wir sind bei unseren Arbeiten auf Spezialfälle gestossen, in denen wir nicht die Kollision zwischen einem Dreieck und einem Punkt ermittelt haben, sondern die Kollision zwischen der Dreiecksebene und dem Punkt. Dies konnten wir machen, da wir im Nachhinein sowieso getestet haben, ob der Kollisionspunkt auch wirklich im Kollisionsdreieck liegt. In diesen Spezialfällen haben wir die Kollision mit der Hilfe der Koordinatengleichung der Dreiecksebene ermittelt. Wir haben getestet wann der Abstand des Punktes und der Ebene gleich Null ist. Es wäre denkbar, dass man eine Form findet, in der man diesen Ansatz nicht nur in Spezialfällen sondern im allgemeinen einsetzen kann. Dies wäre sicher ein Bereich, der weiter untersucht werden sollte.

Durch unseren kontinuierlichen Kollisionsalgorithmus haben wir eine Variante entwickelt, die einem nicht nur sagt, dass eine Kollision stattgefunden hat, sondern man erhält zudem den genauen Zeitpunkt und somit auch den genauen Ort der Kollision. Die Frage ist, ob man diese zusätzlichen Informationen gewinnbringend in eine Kollisionsresponse einbauen kann. Dies ist mit Sicherheit eine nicht ganz einfache aber spannende Erweiterung unserer Arbeit, die untersucht werden sollte.

5

Referenzen

[Moo88]

Matthew Moore and Jane Wilhelms:
Collision Detection and Response for Computer Animation.
SIGGRAPH '88, Atlanta, August 1-5, 1988.

[Teso3]

Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomeranets and Markus Gross:
Optimized Spatial Hashing for Collision Detection of Deformable Objects.
VMV 2003 Munich, Germany, November 19-21, 2003

[Heio4]

B.Heidelberger, M. Teschner, R. Keiser, M. Mueller, M. Gross:
Consistent Penetration Depth Estimation for Deformable Collision Response.
Proc. Vision, Modeling, Visualization VMV'04
Standford, USA, pp. 339-346, November 16-18, 2004

[Redoo]

S. Redon, A. Kheddar and S. Coquillart:
An Algebraic Solution to the Problem of Collision Detection for Rigid Polzhedral Objects.
Proceedings of the 2000 IEEE International Conference on Robotics & Automation
San Francisco, CA - April 2000

[Redo2]

Stephan Redon, Abderrahmane Kheddar and Sabine Coquillart:
Fast Continuous Collision Detection between Rigid Bodies
EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel

[Redo4]

Stephan Redon, Young J. Kim, Ming C. Lin and Dinesh Manocha:
Fast Continuous Collision Detection for Articulated Models
ACM Symposium on Solid Modeling and Applications (2004).

[Chi99]

Chin-Shyurng Fahn and Jui-Lung Wang:
Efficient Time-Interrupted and Time-Continuous Collision Detection Among Polyhedral Objects in Arbitrarz Motion.
Journal of Information Science and Engineering 15, 769-799 (1999).

[Ama87]

John Amanatides and Andrew Woo:
A Fast Voxel Traversal Algorithm for Ray Tracing.
EUROGRAPHICS'87, G. Marechal (Editor)
Elsevier Science Publisher B.V. (North-Holland).

[Hub87]

Philipp M. Hubbard:
Approximating polyhedra with spheres for time-critical collision detection.
ACM Transactions on Graphics, 15(3):179-210, 1996