Numerische Simulationen für granulare Medien

Diplomarbeit
von
Alexander Schinner
aus
Wasserburg am Inn

durchgeführt an der
Naturwissenschaftlichen Fakultät II – Physik
der Universität Regensburg
unter Anleitung von Prof. Dr. Ingo Morgenstern

korrigierte Fassung
September 1995

Entgegenkommen

Die ewig Unentwegten und Naiven
Ertragen freilich unsre Zweifel nicht.
Flach sei die Welt, erklären sie uns schlicht,
Und Faselei die Sage von den Tiefen.

Denn sollt es wirklich andre Dimensionen
Als die zwei guten, altvertrauten geben,
Wie könnte da ein Mensch noch sicher wohnen,
Wie könnte da ein Mensch noch sorglos leben?

Um also Frieden zu erreichen,
So laßt uns eine Dimension denn streichen!

Denn sind die Unentwegten wirklich ehrlich,
Und ist das Tiefensehen so gefährlich,
Dann ist die dritte Dimension entbehrlich.

Magister Ludi Josef Knecht
aus „Das Glasperlenspiel“ von H.Hesse

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

1.1 Allgemeines

Sand ist eines der am wenigsten beachteten, aber fast allgegenwärtigen Dinge unserer Umwelt. Auch in der Physik ist „einfacher“ Sand, oder genauer sind „granulare Medien“ ein noch wenig untersuchtes Gebiet. Daß dies so ist, liegt an der Komplexität, die in der scheinbaren „Einfachheit“ verborgen ist. Eine analytische Lösung verbietet sich von selbst, denn die Art der Wechselwirkung, Reibungsprozesse und Teilchenzahl bilden ein unüberwindliches Hindernis. Aber auch die statistische Physik bietet noch keine Erklärungsmodelle für einen Sandhaufen. Somit besteht heute das Wissen über granulare Medien fast ausschließlich in Erfahrungswerten von Ingenieuren die Schüttwinkel und Raumausfüllung kennen. Man kennt zwar Phänomene wie die Prozesse in Halden, Flußverhalten, Raumausfüllung, das Verhalten unter Vibration, Lawinen und viele andere, genaues Wissen über sie gibt es aber nicht.
Durch das Aufkommen von digitalen Rechenanlagen wurde es möglich, die Natur zu simulieren. Allerding war die Rechenleistung bis vor wenigen Jahren zu gering, um in einem vernünftigen Rahmen die genaue Untersuchung granularer Medien zu ermöglichen. Um Rechenzeit zu sparen, und trotzdem eine akzeptable Zahl von Sandkörnern zu simulieren, war man gezwungen, drastische Vereinfachungen hinzunehmen. So wird fast immer in zwei Dimensionen gearbeitet und zum Teil sogar die freie Bewegung durch eine Bewegung auf einem Gitter eingeschränkt.
Da die zeitintensivste Berechnung normalerweise die der Kollision der Partikel ist, wurde auch deren Form vereinfacht. Meist wird im 2-dimensionalen mit Kreisen, inzwischen aber auch mit Polyedern gearbeitet. Im Bereich der 3-dimensionalen Simulationen sind bisher fast nur Kugeln oder Ellipsoide verwendet worden.
Ziel dieser Arbeit ist es, die Grundlagen für eine realistischere Simulation in drei Dimensionen zu finden.
Es wurde eine dreidimensionale Simulation mit konvexe Polyeder möglichst hoher Flächenzahl angestrebt. Eine steigende Partikelzahl soll die Simulation nicht nachteilig beeinflussen, das Laufzeitverhalten soll linear sein. Spätere Erweiterungen in Richtung Parallelität o.ä. sollen möglich sein. Daraus ergeben sich Probleme verschie-

denster Art. Allgemeine Algorithmen für die Berechnung der wichtigsten Größen wie Volumen, Schwerpunkt und Trägheitstensor für unregelmäßige Polyeder müssen entwickelt werden. Die numerischen Fehler bei der Beschreibung der Rotation mit Matrizen müssen umgangen werden. Das Hauptproblem stellt die Kollisionsdetektion dar, da hier bei bisherigen Simulationen die meiste Rechenzeit verschenkt wurde.

1.2 Aufgaben der Simulation

Granulare Medien zeigen eine große Bandbreite von Eigenschaften, die sich vielfältig von denen anderer Materialien unterscheiden. Eine genaue Zuordnung zu Festkörpern *oder* Flüssigkeiten ist nicht sinnvoll. Die wichtigsten dieser Eigenschaften und einige Simulationsergebnissen sollen vorgestellt werden.

1.2.1 Schüttwinkel (Angle of Repose)

Im Gegensatz zu Flüssigkeiten ist die Oberfläche granularer Medien nicht eben, sondern es können sich verschiedene Strukturen ausbilden. Dies kommt daher, daß sich bis zum einem bestimmten Winkel Θ_m die einzelnen Körner stabil anordnen können. Der zweite wichtige Winkel Θ_r heißt Schüttwinkel oder genauer „angle of repose“. Er beschreibt, unter welchem Winkel ein Sandhaufen zur Ruhe kommt. Die Winkel an der Oberfläche eines Sandhaufens liegen dann immer zwischen Θ_m und Θ_r.

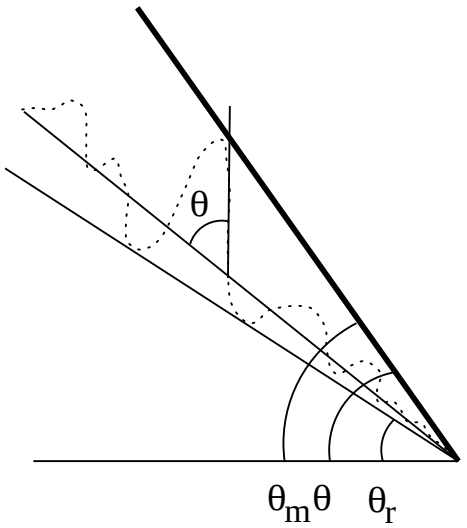

Abbildung 1.1: Die Oberfläche eines Sandhaufens. Die Winkel Θ_r und Θ_m sind eingezeichnet, der gemessene Winkel Θ wird über die lokale Ableitung bestimmt und liegt zwischen Θ_r und Θ_m.

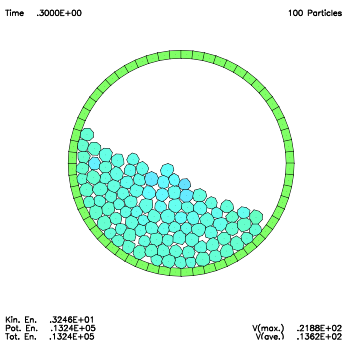

Abbildung 1.2: Zweidimensionale Simulation zu Bestimmung von Θ_r. Die Trommel dreht sich im Uhrzeigersinn.

Eine Möglichkeit, diesen Winkel zu bestimmen ist es, Sand in ein Gefäß zu füllen und dann eine Seitenwand zu entfernen. Ebenso ist es möglich, den Sand in einer rotierenden Trommel wie in Abbildung 1.2 anzusammeln und dann den Winkel zu messen.
Um Θ_m zu bestimmen, läßt man Sandkörner auf einen Haufen fallen, und zwar wird immer dann ein Sandkorn eingefügt, wenn die Maximalgeschwindigkeit der Sandkörner unter eine gewisse Grenze fällt; man hält also den Sandhaufen immer am Abrutschen und füllt entsprechend nach.
Gerade die Bestimmung von Θ_r ist eine der wichtigsten Prüfsteine für eine Simulation. Der theoretische Zusammenhang zwischen dem Koeffizienten der Haftreibung μ und Θ_r ist durch $\mu = \tan \Theta_r$ gegeben.

1.2.2 Verhalten bei Vibration

In einer oben offenen Schachtel werden die Sandkörner plaziert. Der Boden der Schachtel vibriert, daß heißt die Z-Koordinate wird durch

$$\mathbf{z}(t) = A \cdot \sin(2\pi f t) \tag{1.1}$$

bestimmt. Hierbei ist dann die Dichte in den verschiedenen Schichten interessant. Im Experiment wurde in den oberen Schichten von Clément et al. [CR90] eine „fluidized state“ beobachtet, dies allerdings am quasi-zweidimensionalen Modell von 300 Stahlscheiben. In den unteren Teilen hingegen waren die Sandkörner in ihrer Bewegung auf einen kleinen Bereich beschränkt. Ebenso können Konvektionszellen beobachtet werden, wenn man die Amplitude A des Bodens von der x-Koordinate abhängig macht oder indem nicht das ganze Gefäß vibriert, sondern nur der Boden als eine Art Kolben arbeitet.
Ebenfalls interessant sind in diesem Zusammenhang *vibrating convoyer belts*. Dabei schwingt dann der Untergrund nicht parallel zur Richtung der Gravitationskraft, sondern um einen kleinen Winkel verdreht. Nur granulare Medien lassen sich auf diesen

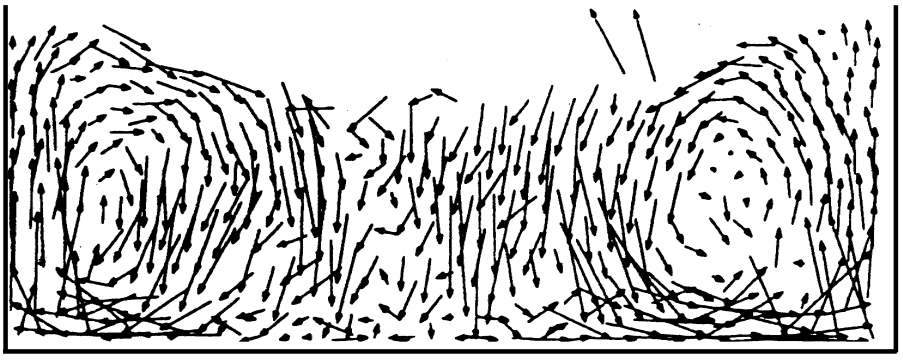

Abbildung 1.3: Konvektionszellen in einer Schachtel, die senkrechten Wände sind fest, die Vibrationsfrequenz ist $f = 20$ Hz. (aus [Her94])

„Fließbändern" befördern, sie sind in der Pharmazeutischen Industrie sehr verbreitet, um Pillen zu transportieren. Wenn die Vibrationsfrequenz klein ist, bleiben die Pillen auf dem Untergrund liegen, und bewegen sich nur auf Ellipsen. Nimmt die Frequenz zu, wird die Bewegung sinusförmig und bei entsprechend hohen Frequenzen werden dann die Kurven immer flacher, bis dann ein fast waagrechter Fluß beobachtet werden kann (siehe Abbildung 1.4).

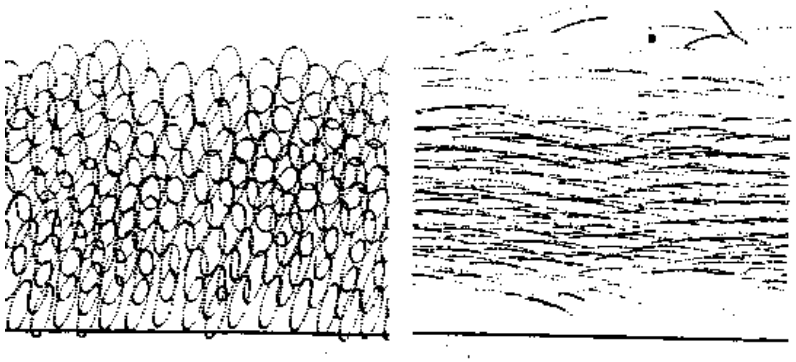

Abbildung 1.4: Trajektorien der Sandkörner bei der Simulation eines *vibrating conveyer belt*. Links liegt die Vibrationsfrequenz bei 10 Hz, die Teilchen bleiben auf dem Band liegen und folgen dessen elliptischer Bewegung. Rechts schwingt die Grundplatte mit 80 Hz, die Partikel bewegen sich fast waagrecht vorwärts.

Betrachtet man Studentenfutter als granulares Medium, dann stellt sich gelegentlich

die Frage: *Warum liegt die Paranuß immer oben?*
Wenn ein Gemisch aus unterschiedlich großen Partikeln gleicher Dichte geschüttelt wird, dann wandern die großen Partikel langsam nach oben. Dieser interessante Effekt heißt *size segregation*. Als mögliches Erkärungsmodell kann man die Konvektionszellen betrachten, liegt nämlich ein großes Partikel an der Oberfläche, dann benötigt es deutlich länger, wieder an Rand der Konvektionszelle zu kommen und dort unterzutauchen. Für die Simulation der Vibration sind die wichtigen Untersuchungsparameter dann das Größenverhältnis der Partikel und die Frequenz und Amplitude der Vibration. Die interessanten Größen sind dann die Art und Form der Konvektionszellen, die Bewegung der „großen" Teilchen und anderes.

1.2.3 Fluß durch einen Trichter oder eine Röhre

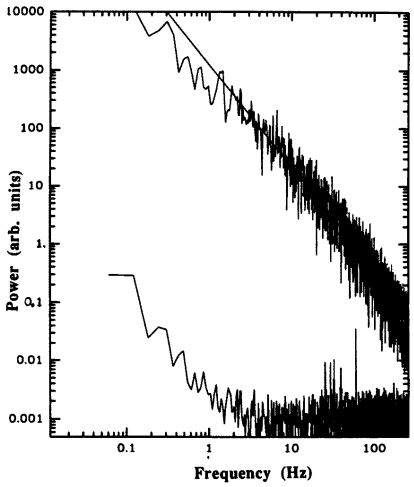

Abbildung 1.5: Doppellogarithmischer Plot der Kraft zwischen Sandkorn und Wand gegen die Frequenz. Die obere Kurve wurde bei durchfließendem Sand aufgenommen, sie zeigt über zwei Zehnerpotenzen hinweg lineares Verhalten. Die untere Kurve zeigt das Rauschen der Versuchsapperatur, der Trichter war befüllt, aber unten verschlossen.(nach [BB93]).

Der Fluß granularer Medien ist ein interessantes Gebiet, obwohl dies auf den ersten Blick nicht so scheint. So ist es z.B. möglich, daß Silos nach Jahren plötzlich durch Schockwellen im Inneren zusammenbrechen [Her94]. Als nächstes stellt sich die Frage, warum ein Trichter verstopfen kann, obwohl seine Austrittsöffnung größer ist als die durchfließenden Partikel?
Beim Durchfluß durch eine Röhre oder einen Trichter kann man Schwankungen der Dichte entlang der Flußrichtung beobachten, die über lange Zeit hinweg stabil bleiben.

Untersucht man den zeitlichen Verlauf der Kraft, die auf die Wände wirkt, zeigt sich wie in Abbildung 1.5 im Spektrum der Kraft ein typisches $\frac{1}{f^\alpha}$ Verhalten. Erst bei sehr steilen Wänden eines Trichters geht das Spektrum in ein weißes Rauschen über.

1.2.4 Die vorliegende Arbeit

In der vorliegenden Arbeit werden die Grundlagen für eine dreidimensionale Simulation untersucht. Durch die Analyse der existierenden Arbeiten konnten die „Schwachstellen" bisheriger Algorithmen festgestellt werden. Es wurde nun versucht, Alternativen zu erarbeiten, die diese Einschränkungen überwinden.
Bei der Darstellung von Rotationen wurden Rotationsmatrizen durch Quaternionen ersetzt. Diese bieten ein deutlich stabileres numerisches Verhalten.
Die Körper der Simulation werden durch konvexe Polyeder dargestellt. Es mußten Verfahren entwickelt werden, mit denen es möglich ist, Masse, Schwerpunkt und Trägheitstensor eines unregelmäßigen Körpers explizit zu berechnen.
Als Hauptproblem gilt bei Simulationen immer der hohe Zeitbedarf. Es zeigt sich allerdings, daß in den meisten bisherigen Arbeiten für die Kollisionsdetektion Algorithmen verwendet werden, deren Zeitbedarf mit $\mathcal{O}(n^k)$ geht. Das Hauptgewicht der Arbeit lag darauf, dieses Problem zu umgehen. Es konnten Algorithmen aus den Bereichen der Computational-Geometry und der Virtual-Reality in die Physik übertragen werden. Mit diesen Algorithmen ist es möglich, die Kollision aller Sandkörner mit $\mathcal{O}(n)$ zu finden, wobei das einzelne Sandkorn beliebig komplex sein kann.
Noch nicht behandelt wurde die Berechnung der Kraft zwischen zwei kollidierenden Partikeln und deren Haft- und Gleitreibung. Die Berechnung der Kraft soll aus dem Overlap der zwei Polyeder erfolgen. Dazu soll noch ein Algorithmus entwickelt werden, der bei kleinen Eindringtiefen mit Hilfe der Graphentheorie sehr schnell Ergebnisse liefert.

Kapitel 2

Mechanik starrer Körper

2.1 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen dargestellt, die benötigt werden, um die Bewegung eines starren Körpers unter dem Einfluß von Kraft und Drehmoment zu beschreiben.
Es wird, wie in der theoretischen Physik üblich, zwischen einen ortsfesten Koordinatensystem $\mathbf{K}$ und einem für jedes Sandkorn S_n individuellen, Koordinatensystem $\overline{\mathbf{K}}_\mathbf{n}$ unterschieden. Im weiteren wird statt $\overline{\mathbf{K}}_\mathbf{n}$ nur $\overline{\mathbf{K}}$ geschrieben. Es wird vorrausgesetzt, daß alle Formeln in gleicher Weise für jedes Partikel gelten.

1. Das „raumfeste“ Koordinatensystem $\mathbf{K}$ ist für alle Körper identisch, es ist ein Inertialsystem.

2. Das „körperfeste“ Koordinatensystem $\overline{\mathbf{K}}$ ist mit dem Körper fest verbunden, es ist kein Inertialsystem. Es wird so gewählt, daß

$$\int \mathrm{d}^3x \, \mathbf{x}\varrho(x) = 0 \tag{2.1}$$

   gilt. Somit liegt der Schwerpunkt im Ursprung von $\overline{\mathbf{K}}$.

Die Bewegung eines starren Körpers besteht dann aus einer Translation des Schwerpunktes und einer Rotation um den Koordinatenursprung von $\overline{\mathbf{K}}$. Die Winkelgeschwindikeit $\boldsymbol{\omega}(t)$ der Rotation ist parallel zur momentanen Drehachse, diese geht in $\overline{\mathbf{K}}$ durch $(0,0,0)$.
Durch drei körperfeste Punkte $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$, die nicht auf einer Geraden liegen, ist die Orientierung eines Körpers fest vorgegeben. Somit kann der Körper bei homogener Verformung maximal neun Freiheitsgrade haben, diese sind drei Translations-, drei Rotations– und drei Deformationsfreiheitsgrade. Da außerdem der Abstand zwischen den Punkten fest ist

$$|\mathbf{r}_1 - \mathbf{r}_2, | = \text{const} \quad |\mathbf{r}_1 - \mathbf{r}_3, | = \text{const} \quad |\mathbf{r}_2 - \mathbf{r}_3, | = \text{const}, \tag{2.2}$$

entfällt natürlich die Deformation. Der Lage des Körper ist also durch die Koordinaten des Schwerpunktes in **K** sowie die Richtung der Drehachse und die Größe des Drehwinkels vollständig beschrieben.
Damit ist die Geschwindigkeit **v** eines körperfesten Punktes im Inertialsystem durch

$$\mathbf{v} = \mathbf{v}_0 + \boldsymbol{\omega} \times \mathbf{x} \tag{2.3}$$

mit

$$\begin{aligned} \mathbf{v}_0 &= \text{Geschwindigkeit des Koordinatenursprungs von } \overline{\mathbf{K}} \text{ in } \mathbf{K} \\ \boldsymbol{\omega} &= \text{Winkelgeschwindigkeit im Inertialsystem} \\ \mathbf{x} &= \text{Ortsvektor des Punktes in } \overline{\mathbf{K}} \end{aligned} \tag{2.4}$$

gegeben.
Für die kinetische Energie gilt

$$\begin{aligned} T &= \frac{1}{2}\int \mathrm{d}^3x\ \varrho(x)(\mathbf{v}_0 + \boldsymbol{\omega}\times\mathbf{x}) \\ &= \frac{1}{2}\mathbf{v}_0^2 \underbrace{\int \mathrm{d}^3x\ \varrho(x)}_{\text{Gesamtmasse M}} + (\mathbf{v}_o \times \boldsymbol{\omega}) \underbrace{\int \mathrm{d}^3x\ \mathbf{x}\varrho(x)}_{\equiv 0} + \\ &+ \frac{1}{2}\int \mathrm{d}^3x\ \varrho(x)\omega^\mu \left[\mathbf{x}^2\delta_{\mu\nu} - x_\mu x_\nu\right]\omega^\nu \end{aligned} \tag{2.5}$$

Man kann die kinetische Energie also zerlegen in $T_{trans} = \frac{1}{2}M\mathbf{v}_0^2$ und $T_{rot} = \frac{1}{2}\boldsymbol{\omega}\mathbf{I}\boldsymbol{\omega}$, dabei ist **I** der durch

$$I_{(\mu\nu)} = \int \mathrm{d}^3x\ \varrho(x)\left[\mathbf{x}^2\delta_{\mu\nu} - x_\mu x_\nu\right] \tag{2.6}$$

gegebene Trägheitstensor[1].
Der Drehimpuls kann in den Drehimpuls des Schwerpunktes und den relativen Drehimpuls zerlegt werden. Nur der Schwerpunktsanteil ist dabei von der Wahl des raumfesten Koordinatensystem **K** abhängig. Der Relativdrehimpuls ist durch

$$\mathbf{L} = \int \mathrm{d}^3x\ \varrho(x)\mathbf{x}\times\dot{\mathbf{x}} \tag{2.7}$$

gegeben. Nach 2.3 ist $\dot{\mathbf{x}} = \boldsymbol{\omega}\times\mathbf{x}$, somit gilt

$$\mathbf{L} = \int \mathrm{d}^3x\ \varrho(x)\mathbf{x}\times(\boldsymbol{\omega}\times\mathbf{x}) = \int \mathrm{d}^3x\ \varrho(x)\left[\mathbf{x}^2\boldsymbol{\omega} - (\mathbf{x}\cdot\boldsymbol{\omega})\mathbf{x}\right] = \mathbf{I}\boldsymbol{\omega} \tag{2.8}$$

Man kann dann auch $T_{rot} = \frac{1}{2}\boldsymbol{\omega}\mathbf{I}\boldsymbol{\omega}$ schreiben.
Für die Berechnung der allgemeinen Bewegung starrer Körper werden zwei Vektorgleichungen benötigt, mit denen man die Schwerpunktsbewegung und die Drehung um den Schwerpunkt beschreiben kann.

[1] Zur numerischen Berechnung: siehe Kapitel 3.5

2.2 Die Bewegungsgleichungen

Zuerst soll die Bewegung des Schwerpunktes betrachtet werden. Da das Integral über die Kräfte zwischen den differentiellen Massenelementen $\equiv 0$ ist, bewegt sich der Schwerpunkt so, als ob die gesamte Masse M des starren Körpers in ihm vereinigt wäre und die resultierende Kraft $F_{res} = \sum_{\alpha=1}^{n} F_\alpha$ aller äußeren Kräfte an ihm angreifen würde. Bezeichnet man $\mathbf{p} = M\dot{\mathbf{x}}$ als den Gesamtimpuls, dann gilt

$$\frac{\mathrm{d}}{\mathrm{d}t}\,\mathbf{p} = \mathbf{F}_{res} \quad \text{oder} \quad \ddot{\mathbf{x}} = \frac{\mathbf{F}_{res}}{M}. \tag{2.9}$$

Nach dem Drehimpulssatz

$$\frac{\mathrm{d}}{\mathrm{d}t}\,\mathbf{L} = \mathbf{D} \tag{2.10}$$

ist die zeitliche Änderung des Drehimpulses gleich dem resultierenden äußeren Drehmoment. Dabei bezieht man sich auf das körperfeste Koordinatensystem $\overline{\mathbf{K}}$. In der Darstellung des starren Körpers als N Massenpunkte ist dabei

$$\begin{aligned} \mathbf{L} &= \sum_{i=1}^{N} m_i x^{(i)} \times \dot{x}^{(i)} \\ \mathbf{D} &= \sum_{i=1}^{N} x^{(i)} \times \dot{\mathbf{F}}^{(i)} \end{aligned} \tag{2.11}$$

.

Für die Darstellung der zeitlichen Änderung von $\mathbf{L}$ ist die Eulersche Gleichung hilfreich:

$$\dot{\mathbf{L}} = \mathbf{I}\dot{\boldsymbol{\omega}} + (\boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega}); \tag{2.12}$$

dabei beziehen sich alle Größen auf das körperfeste Koordinatensystem.

2.3 Quaternionen als Ersatz für Rotationsmatrizen

2.3.1 Die Nachteile von Rotationsmatrizen in der Simulation

Der in der theoretischen Physik übliche Ansatz, Rotationen mit Hilfe von Rotationsmatrizen $\mathbf{R}$ darzustellen, ist im Bereich der numerischen Simulationen nicht vorteilhaft: Durch die iterativen Lösungsmethoden für die Gleichung

$$\dot{\mathbf{R}}(t) = f(\mathbf{R}(t)) \tag{2.13}$$

und die dabei auftretenden numerischen Fehler entsteht ein „numerical drift“ [Bar93]. Dabei werden kleine Fehler über viele Zeitschritte aufsummiert und führen dann zu dem beschriebenen Verhalten.

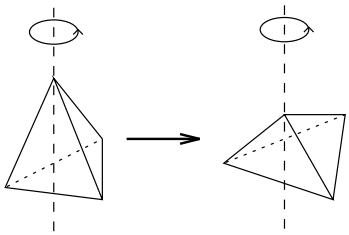

Abbildung 2.1: Deformation eines rotierenden Tetraeders

Um dies darzustellen, wurde ein regelmäßiges Tetraeder simuliert, das sich um eine Achse dreht, die durch eine Ecke verläuft und senkrecht auf der gegenüberliegenden Fläche steht. Es wirkten keine externen Drehmomente oder Kräfte auf diesen Körper. Vier verschiedene Zeitschritte ($0.1s$, $0.01s$, $0.001s$, und $0.0001s$) wurden berechnet und ausgewertet. In den Abbildungen 2.2, 2.3 und 2.4 ist jeweils der Abstand einer oder mehrerer Ecken eines Tetraeders zu dessen Schwerpunkt aufgetragen. So trat bei einer Schrittweite von 0.1s nach einer Simulationszeit von $> 1.5s$ eine Größenänderung um den Faktor 10^{120} auf[2] (siehe Abbildung 2.4). Abbildung 2.2 zeigt, daß auch in realistischeren Fällen mit deutlich kleineren Zeitschritten eine stetige Vergrößerung des Körpers beobachtet werden kann. Doch auch hier tritt, wie man in Abbildung 2.3 detailliert sieht, nach einiger Zeit, abhängig von der Schrittweite, eine „numerische Katastrophe“ ein. Das Problem ist aber nicht durch eine Skalierung zu lösen, wie am Graphen 2.3 zu erkennen ist. Man sieht, daß hier drei Ecken des Tetraeders schneller nach außen abwandern als die vierte Ecke, durch die die Rotationsachse verläuft. Diese Verformung wird in Abbildung 2.1 nochmals schematisch dargestellt.

2.3.2 Quaternionen als Rotation

Als Alternative zu Rotationsmatrizen kann man Rotationen mit Quaternionen

$$\mathbf{q} = [w\ ,\ (x, y, z)] \tag{2.14}$$

darstellen. Sie bieten ein deutlich besseres numerisches Verhalten. Quaternionen können auf verschiedene äquivalente Weise definiert werden. Es ist inzwischen[3] üblich, die Zahlen x, y, z als Vektor zu bezeichnen, und w als Skalar. Die Verknüpfungen zwischen Quaternionen können dann auf der Basis von Skalar- und Kreuzprodukt definiert werden. Die Definitionen für Quaternionen, den Beweis für die Äquivalenz von Quaternionen und Matrizen im Hinblick auf die Rotation, die Bestimmung von $\mathbf{R}$ aus $\mathbf{q}$ und umgekehrt, sowie einige nützliche Formeln finden sich im Anhang A.

[2]Dieses Verhalten ist sicherlich auf die Wahl der extrem großen Zeitschritte von $0.1s$ zurückzuführen, doch soll diese Simulation nur die Gefahr aufzeigen, die Rotationsmatrizen darstellen können.

[3]Ursprünglich wurden sie von Hamilton als erweiterte komplexe Zahlen in der Form $w+\mathbf{i}x+\mathbf{j}y+\mathbf{k}z$, definiert, wobei $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$, $\mathbf{ij} = \mathbf{k} = -\mathbf{ji}$ mit $w, x, y, z \in \mathbb{R}$ gilt.

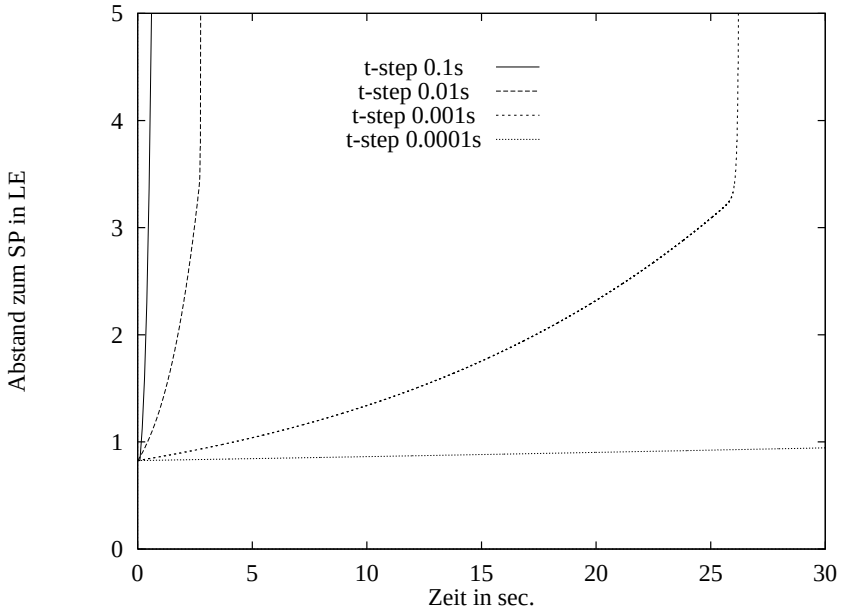

Abbildung 2.2: Abstände einer Ecke zum Schwerpunkt des Tetraeders bei verschiedenen Zeitschritten

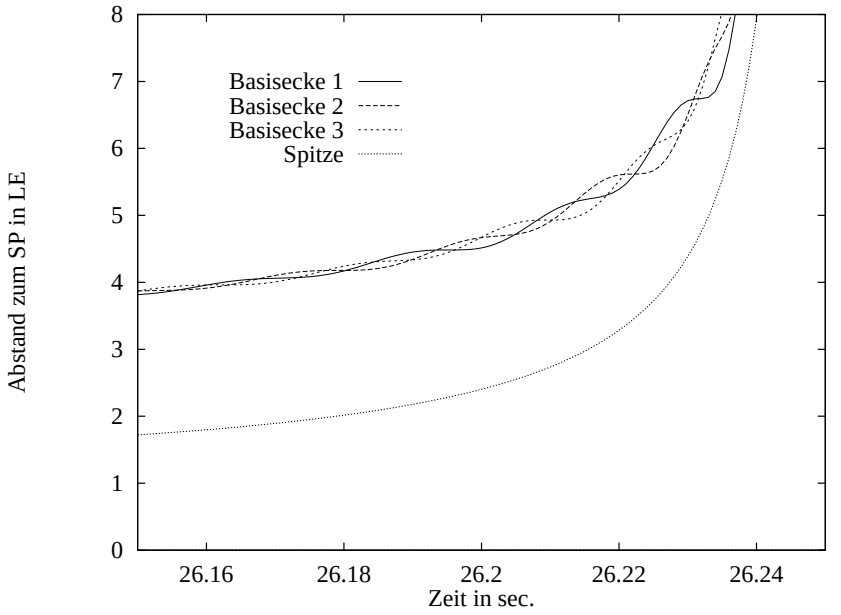

Abbildung 2.3: Detailansicht des „numerical drift“ bei einer Schrittweite von $0.001s$

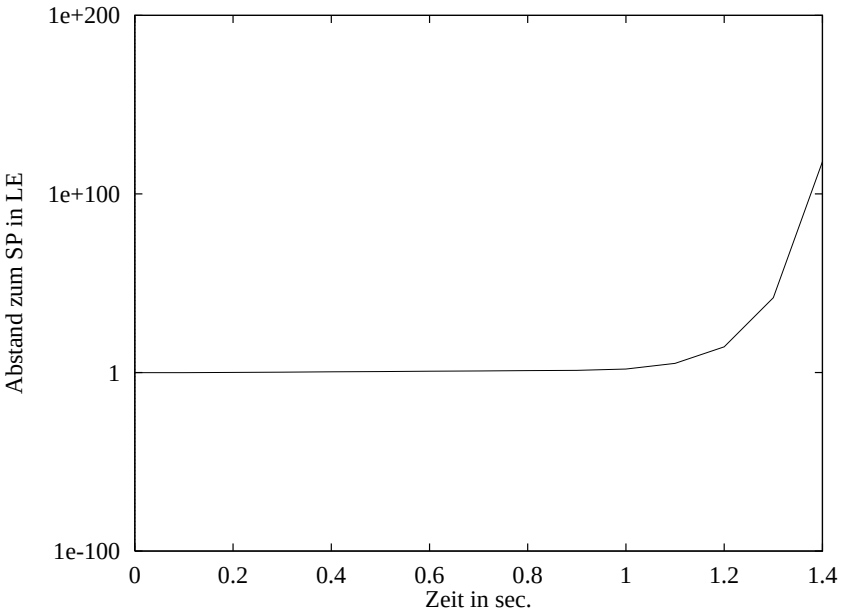

Abbildung 2.4: Abstände einer Ecke eines rotierenden Tetraeders bei 0.1 s Zeitschritt. Durch das Aufsummieren der Fehler in der Rotationsmatrix nimmt die Größe des Tetraeders mit dem Faktor 10^{120} zu.

2.3.3 Zeitableitung von Quaternionen

Ziel dieser Rechnung ist es, eine Differentialgleichung für $\mathbf{q}$ zu erhalten, aus der sich während der Simulation dann die Orientierung im Raum bestimmen läßt.
Die Winkelgeschwindigkeit $\boldsymbol{\omega}(t)$ gibt bekanntlich an, daß ein Körper um die $\boldsymbol{\omega}(t)$-Achse mit $|\boldsymbol{\omega}(t)|$ rotiert. Wenn man nun von einem Körper mit konstanter Winkelgeschwindigkeit $\boldsymbol{\omega}$ ausgeht, dann wird die Rotation des Körpers nach einem kurzen Zeitintervall Δt durch das Quaternion

$$\left[\cos\frac{|\boldsymbol{\omega}(t)|\,\Delta t}{2}\ ,\ \sin\frac{|\boldsymbol{\omega}(t)|\,\Delta t}{2}\frac{\boldsymbol{\omega}(t)}{|\boldsymbol{\omega}(t)|}\right] \quad \text{(siehe Anhang A)} \tag{2.15}$$

beschrieben. Nun soll $\dot{\mathbf{q}}(t)$ zu einem beliebigen Zeitpunkt t_0 bestimmt werden. Zum Zeitpunkt $t_0 + \Delta t$, wobei Δt klein gewählt wird, kann dann die Orientierung des Körpers dadurch genähert werden, daß man zuerst eine Rotation mit $\mathbf{q}(t_0)$ausführt. Dann wird mit $\boldsymbol{\omega}(t_0)$ über den Zeitraum Δt um die Achse $\frac{\boldsymbol{\omega}(t)}{|\boldsymbol{\omega}(t)|}$ gedreht. Durch die Kombination der beiden Drehungen ergibt sich

$$\mathbf{q}(t_0+\Delta t) = \left[\cos\frac{|\boldsymbol{\omega}(t)|\,\Delta t}{2}\ ,\ \sin\frac{|\boldsymbol{\omega}(t)|\,\Delta t}{2}\frac{\boldsymbol{\omega}(t)}{|\boldsymbol{\omega}(t)|}\right]\mathbf{q}(t_0) \tag{2.16}$$

und mit der Substitution $t = t_0 + \Delta t$,bzw.$\Delta t = t - t_0$ dann

$$\mathbf{q}(t) = \left[\cos\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2}\ ,\ \sin\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2}\frac{\boldsymbol{\omega}(t)}{|\boldsymbol{\omega}(t)|}\right]\mathbf{q}(t_0). \tag{2.17}$$

Nun wird Formel 2.17 zum Zeitpunkt t_0 abgeleitet. Da $\mathbf{q}(t_0)$ konstant ist, muß nur $\left[\cos\frac{|\boldsymbol{\omega}(t)|(t-t_0)}{2}\ ,\ \sin\frac{|\boldsymbol{\omega}(t)|(t-t_0)}{2}\frac{\boldsymbol{\omega}(t)}{|\boldsymbol{\omega}(t)|}\right]$ zur Zeit $t = t_0$ betrachtet werden.

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\cos\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2} &= -\frac{|\boldsymbol{\omega}(t)|}{2}\sin\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2} = -\frac{|\boldsymbol{\omega}(t)|}{2}\sin 0 = 0\\ \frac{\mathrm{d}}{\mathrm{d}t}\sin\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2} &= \frac{|\boldsymbol{\omega}(t)|}{2}\cos\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2} = \frac{|\boldsymbol{\omega}(t)|}{2}\cos 0 = \frac{|\boldsymbol{\omega}(t)|}{2}\end{aligned} \tag{2.18}$$

Somit ergibt sich

$$\begin{aligned}\dot{\mathbf{q}}(t) &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\left[\cos\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2}\ ,\ \sin\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2}\frac{\boldsymbol{\omega}(t)}{|\boldsymbol{\omega}(t)|}\right]\mathbf{q}(t_0)\right)\\ &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\left[\cos\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2}\ ,\ \sin\frac{|\boldsymbol{\omega}(t)|\,(t-t_0)}{2}\frac{\boldsymbol{\omega}(t)}{|\boldsymbol{\omega}(t)|}\right]\right)\mathbf{q}(t_0)\\ &= \left[0\ ,\ \frac{|\boldsymbol{\omega}(t)|}{2}\frac{\boldsymbol{\omega}(t)}{|\boldsymbol{\omega}(t)|}\right]\mathbf{q}(t_0)\\ &= \frac{1}{2}\left[0\ ,\ \boldsymbol{\omega}(t_0)\right]\mathbf{q}(t_0)\end{aligned} \tag{2.19}$$

Der Term $[0\ ,\ \boldsymbol{\omega}(t_0)]\,\mathbf{q}(t_0)$ wird nach Anhang A mit $[\boldsymbol{\omega}(t_0)\mathbf{q}(t_0)]$ abgekürzt[4], der letzte Term kann dann als

$$\dot{\mathbf{q}}(t) = \frac{1}{2}\left[\boldsymbol{\omega}(t)\mathbf{q}(t)\right] \tag{2.20}$$

geschrieben werden.
Nun muß $\ddot{\mathbf{q}}$ bestimmt werden.

$$\begin{aligned}\ddot{\mathbf{q}}(t) &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{2}\left[\boldsymbol{\omega}(t)\mathbf{q}(t)\right]\right)\\ &= \frac{1}{2}\left(\left[\dot{\boldsymbol{\omega}}(t)\mathbf{q}(t)\right] + \left[\boldsymbol{\omega}(t)\dot{\mathbf{q}}(t)\right]\right)\end{aligned} \tag{2.21}$$

Durch Einsetzen von Formel 2.20 in 2.21 erhält man[5]

$$\ddot{\mathbf{q}}(t) = \frac{1}{2}\left(\left[\dot{\boldsymbol{\omega}}(t)\mathbf{q}(t)\right] + \frac{1}{2}\left[\boldsymbol{\omega}(t)\cdot\left[\boldsymbol{\omega}(t)\mathbf{q}(t)\right]\right]\right) \tag{2.22}$$

Die Zeitableitung $\dot{\boldsymbol{\omega}}(t)$ ergibt sich aus der Eulerschen Gleichung

$$\dot{\mathbf{L}} = \mathbf{I}\dot{\boldsymbol{\omega}} + (\boldsymbol{\omega}\times\mathbf{I}\boldsymbol{\omega}), \tag{2.23}$$

[4] Bei unvorsichtiger Rechnung kann diese Abkürzung zur Verwechslung von Quaternion $[\boldsymbol{\omega}(t)\ ,\ 0]$ und $\boldsymbol{\omega}(t)$ führen, stellt aber sonst kein Problem dar.

[5] *Vorsicht!* $\boldsymbol{\omega}(t)\cdot[\boldsymbol{\omega}(t)\ ,\ \mathbf{q}(t)]$ darf nicht als $\boldsymbol{\omega}(t)^2\mathbf{q}(t)$ geschrieben werden, da $[\boldsymbol{\omega}(t)\ ,\ \mathbf{q}(t)]$ ein Quaternion darstellt. Es wäre möglich, zuerst $[\boldsymbol{\omega}(t)\ ,\ 0]\cdot[\boldsymbol{\omega}(t)\ ,\ 0]$ zu berechnen, das Ergebnis ist aber ein Quaternion und kein Vektor $\in\mathbb{R}^3$.

mit $\dot{\mathbf{L}} = \mathbf{D}$ folgt

$$\dot{\boldsymbol{\omega}}(t) = \mathbf{I}^{-1}(t)\left(\mathbf{D}(t) + (\mathbf{I}(t)\boldsymbol{\omega}(t) \times \boldsymbol{\omega}(t))\right). \tag{2.24}$$

Löst man Formel 2.20 nach $\boldsymbol{\omega}$ auf[6]

$$\boldsymbol{\omega}(t) = [0\ ,\ \boldsymbol{\omega}(t)] = 2\dot{\mathbf{q}}(t)\mathbf{q}^*(t). \tag{2.25}$$

und setzt in 2.24 ein, dann erhält man

$$\dot{\boldsymbol{\omega}}(t) = \mathbf{I}^{-1}(t)\left(\mathbf{D}(t) + 4(\mathbf{I}(t)\dot{\mathbf{q}}(t)\mathbf{q}^*(t) \times \dot{\mathbf{q}}(t)\mathbf{q}^*(t))\right). \tag{2.26}$$

Durch Einsetzen der Gleichungen 2.25 und 2.26 in 2.22 ergibt sich

$$\ddot{\mathbf{q}}(t) = \frac{1}{2}\left(\left[\underbrace{\left(\mathbf{I}^{-1}(t)\left(\mathbf{D}(t) + 4(\mathbf{I}(t)\dot{\mathbf{q}}(t)\mathbf{q}^*(t) \times \dot{\mathbf{q}}(t)\mathbf{q}^*(t))\right)\right)}_{\dot{\boldsymbol{\omega}}(t)}\mathbf{q}(t)\right] + \frac{1}{2}\left[\underbrace{2\dot{\mathbf{q}}(t)\mathbf{q}^*(t)}_{\boldsymbol{\omega}(t)} \cdot \left[\underbrace{2\dot{\mathbf{q}}(t)\mathbf{q}^*(t)}_{\boldsymbol{\omega}(t)}\mathbf{q}(t)\right]\right]\right) \tag{2.27}$$

Da $\mathbf{I}(t)$ und $\mathbf{I}^{-1}(t)$ noch von $\mathbf{q}$ abhängig sind, werden diese auf die Trägheitstensoren im Koordinatensystem $\mathbf{K}$ umgeformt. Dabei ist $\mathbf{R}_q(t)$ eine Rotationsmatrix, für die für beliebige Vektoren $\mathbf{x}$ zu einer beliebigen Zeit t gilt[7]:

$$\mathbf{R}_q(t)\mathbf{x} = \mathbf{q}(t)\,[0\ ,\ \mathbf{x}]\,\mathbf{q}^*(t) = \mathbf{q}(t)\mathbf{x}\mathbf{q}^*(t) \tag{2.28}$$

$$\mathbf{I}(t) = \mathbf{R}_q(t)\mathbf{I}{\mathbf{R}_q}^T(t) \tag{2.29}$$

$$\mathbf{I}^{-1}(t) = \mathbf{R}_q(t)\mathbf{I}_{body}^{-1}{\mathbf{R}_q}^T(t) \tag{2.30}$$

Somit ist $\mathbf{R}_q(t)$ nur die Matrixdarstellung für die Rotation $\mathbf{q}(t)\mathbf{x}\mathbf{q}^*(t)$.
Insgesamt hat man dann

[6] Hier und im folgenden wird statt $[0\ ,\ \boldsymbol{\omega}]$ nur $\boldsymbol{\omega}$ geschrieben.

[7] Eine direkte Transformation von Matrizen mit Hilfe von Quaternionen konnte nicht gefunden werden.

Kapitel 3

Simulationsmethoden

Wenn die Geometrie eines Körpers festgelegt ist, müssen verschiedene charakteristische Größen berechnet werden. Zuerst werden der Schwerpunkt und die Masse berechnet. Damit kann durch eine einfache Koordinatenverschiebung der Ursprung des körperfesten Koordinatensystem $\overline{\mathbf{K}}$ in den Schwerpunkt gelegt und der Trägheitstensor berechnet werden.

3.1 Die Form der Körper

Die einzelnen Sandkörner der Simulation werden durch konvexe Polyeder dargestellt. Die Einschränkung auf konvexe Körper erscheint annehmbar, da es dadurch möglich war, Algorithmen mit sehr gutem Zeitverhalten zu verwenden.

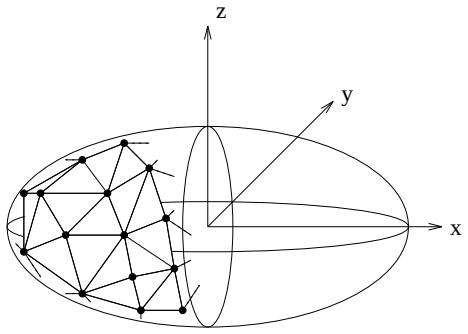

Abbildung 3.1: Generierung der Körper. Die Eckpunkte liegen auf einem Ellipsoiden.

Die Oberflächenelemente eines Polyeders sind immer Dreiecke. Da aber die einzelnen Dreiecke parallel sein können, ist es möglich beliebig komplexe Polygone als Oberflächenelemente zu erzeugen. Der Rechenzeitbedarf steigt dadurch nicht, da die verwendeten Algorithmen entweder völlig unabhängig von der Gestalt des Körpers

sind, oder aber nur von der Eckenzahl abhängen. Die Zahl der Ecken des Polyeders ändert sich aber durch die Zerlegung nicht.

Um „schöne“ Sandkörner zu erhalten, werden die Polyeder in einen Ellipsoid eingepaßt, d.h. die Ecken des Polyeders liegen auf der Oberfläche eines Ellipsoiden. Weiterhin hat dieses Verfahren den Vorteil, daß man die Ellipsoid-Methode zu Bestimmung von Bounding–Boxes verwenden kann (siehe 4.1.3.2). Es ist somit recht einfach zu bewerkstelligen, daß man ähnliche Sandkörner in einer Simulation hat, die Größe kann über die Halbachsen des Ellipsoid leicht variiert werden und die Ecken werden auf der Oberfläche leicht verschoben. Die Beschränkung auf die Oberfläche von Ellipsoiden ist aber nicht sehr „streng“, da sie nur auf Teilchen zutrifft, die automatisch generiert werden. Alle Teilchen, die selbst konstruiert werden, unterliegen dieser Beschränkung nicht.

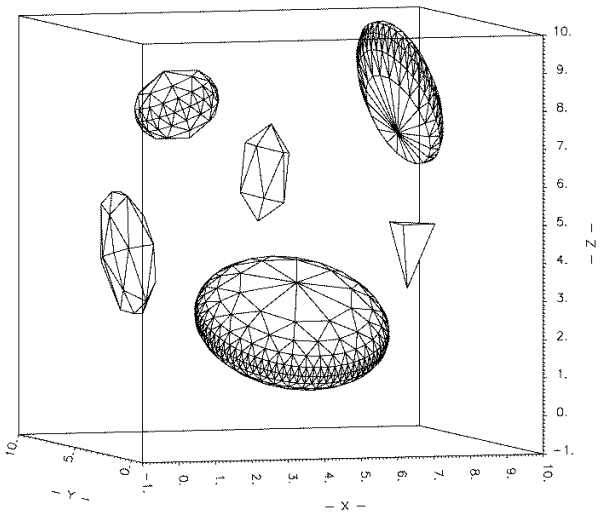

Abbildung 3.2: Dargestellt sind verschiedene Partikel mit denen in der Simulation gearbeitet wurde. Das einfachste Teilchen ist das Tetraeder (rechts). Der Kompliziertest ist ein 462-flächler (unten).

Im weiteren haben Polyeder also immer die folgenden Eigenschaften:

1. Die Polyeder sind konvex.

2. Die Eckpunkte liegen auf der Oberfläche eines Ellipsoiden.

3. Die Oberflächenelemente sind Dreiecke.

3.2 Die Zerlegung des Polyeders in geeignete geometrische Objekte

Bei Masse, Schwerpunkt und Trägheitstensor ist es notwendig, ein oder mehrmals über d^3r zu integrieren. Die Berechnungen sind erst nach Programmstart möglich, da Gestalt und Größe der Polyeder nicht von vornherein festgelegt sind. Eine Monte-Carlo-Simulation nach dem simple-sampling-Schema

$$\int \mathrm{d}V \approx V\left(\langle f\rangle \pm \sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}}\right) \tag{3.1}$$

erscheint nicht sinnvoll, da die Bestimmung aller zehn Integrale (Masse, dreimal für den Schwerpunkt und sechsmal für den Trägheitstensor[1]) mit ausreichender Genauigkeit zu lange dauern würde. Es müßte für jeden zufällig gewählten Punkt bestimmt werden, ob er innerhalb oder außerhalb des Polyeders liegt, dies ist bei hoher Flächenzahl zu aufwendig.
Die andere naheliegende Möglichkeit wäre, da die Polyederecken auf einem Ellipsoid liegen, einfach die entsprechenden Formeln für dieses zu verwenden. Da aber besonders Wert darauf gelegt wurde, *keine unnötigen* Einschränkungen zu machen, sondern größtmögliche Realitätsnähe zu erreichen, wurde diese Näherung nicht in Betracht gezogen. Stattdessen werden die Polyeder in einzelne Pyramiden mit bestimmter Form zerlegt, für die dann eine analytische Lösung berechnet werden konnte. Diese müssen dann noch in geeigneter Weise wieder „zusammengefügt“ werden.
Das Ziel ist es, Pyramiden mit folgender Gestalt zu erhalten: Wenn $\mathcal{P}_1$,$\mathcal{P}_2$, $\mathcal{P}_3$,$\mathcal{P}_4$ die Ecken der Pyramide sind, so müssen sie bei folgenden Koordinaten liegen:

$$\begin{aligned}\mathcal{P}_1 &= (0,0,0)\\ \mathcal{P}_2 &= (a,0,0)\\ \mathcal{P}_3 &= (a,d,0)\\ \mathcal{P}_3 &= (a,b,c)\end{aligned} \tag{3.2}$$

Die Stirnfläche durch $(\mathcal{P}_2,\mathcal{P}_3,\mathcal{P}_4)$ ist somit parallel zur Y-Z-Ebene, die Basisfläche $(\mathcal{P}_1,\mathcal{P}_2,\mathcal{P}_3)$ liegt in der X-Y-Ebene.
Es sei das Polyeder durch N Dreiecke $D_i, i = 1,\ldots,N$ begrenzt, die Ecken seinen $E_j^{(D_i)}, j = 1,2,3$. Außerdem liege der Ursprung innerhalb des konvexen Polyeders. Dann läßt sich das Polyeder P in N Pyramiden P_i' zerlegen. Dabei ist dann D_i die Basisfläche, der Ursprung die Pyramidenspitze und die Ecken haben dann als Koordinaten $[(0,0,0), E_1^{(D_i)}, E_2^{(D_i)}, E_3^{(D_i)}]$.

[1] Für den Trägheitstensor müssen nur sechs Elemente unter Ausnutzung der Symetrieeigenschaften berechnet werden, ansonsten sind neun Integrale notwendig

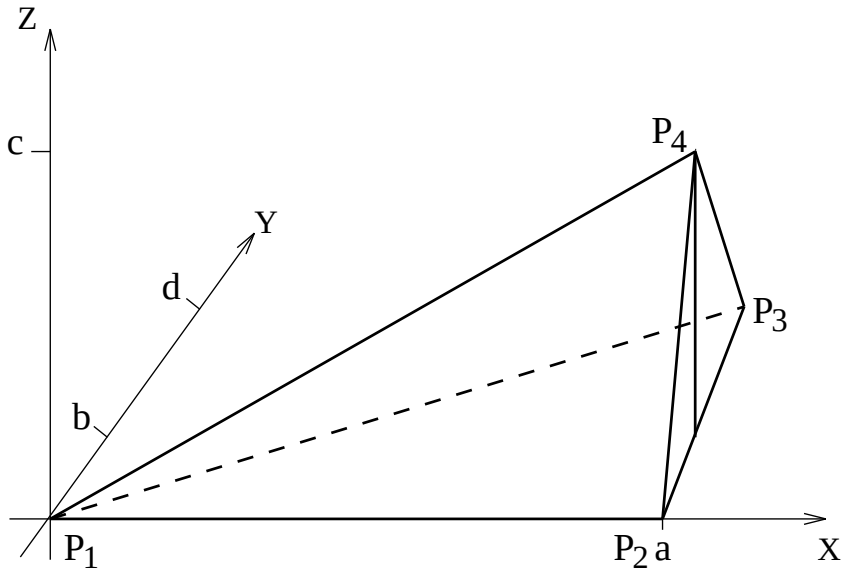

Abbildung 3.3: Darstellung der Teilpyramide, für die der Schwerpunkt und der Trägheitstensor bestimmt wird. Die Pyramide ist nach 3.2 durch a, b, c, d bestimmt.

Jede Pyramide P_i' muß weiter zerlegt werden. Dazu fällt man das Lot durch den Ursprung auf die Ebene, in der D_i liegt; der Schnittpunkt beider ist L_i. Damit sind dann drei Pyramiden $P_{i,j}', j = 1, 2, 3$ pro P_i' mit den Eckpunkten

$$[(0,0,0), L_i, E_1^{(D_i)}, E_2^{(D_i)}]\,,\ [(0,0,0), L_i, E_2^{(D_i)}, E_3^{(D_i)}]\,,\ [(0,0,0), L_i, E_1^{(D_i)}, E_3^{(D_i)}]$$

bestimmt. Dabei bildet die Strecke $\overline{((0,0,0), L_i)}$ mit $\overline{L_i, E_1^{(D_i)}}$, $\overline{L_i, E_2^{(D_i)}}$, $\overline{L_i, E_3^{(D_i)}}$ jeweils einen rechten Winkel. Somit kann nun eine Rotationsmatrix $\mathbf{R}_{i,j}$ berechnet werden, die $P_{i,j}'$ so dreht, daß die Bedingungen aus Formel 3.2 erfüllt sind. Die zu drehenden Koordinatenachsen sind (siehe Abbildung 3.4):

$$\begin{aligned} \mathbf{x} &= \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} & \mathbf{x}' &= L_i \\ \mathbf{y} &= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} & \mathbf{y}' &= E_1^{(D_i)} - L_i \\ \mathbf{z} &= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} & \mathbf{z}' &= \mathbf{x}' \times \mathbf{y}'. \end{aligned} \tag{3.3}$$

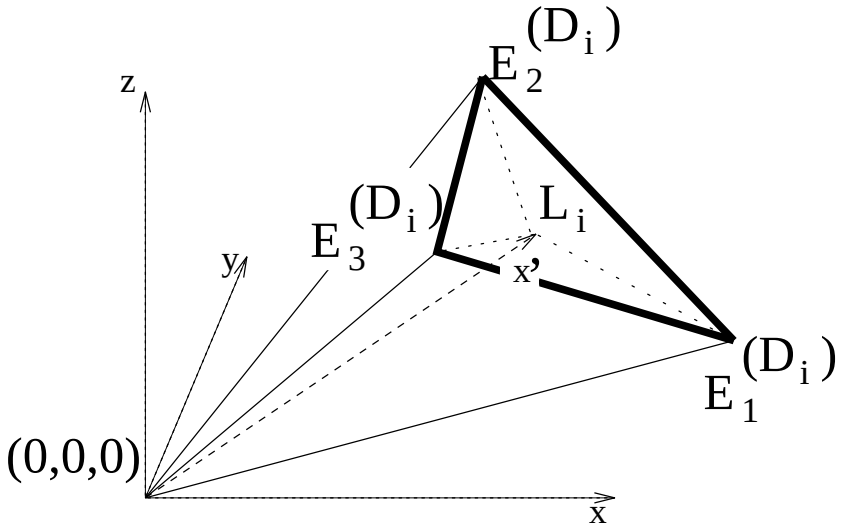

Abbildung 3.4: Nachdem das Polyeder in Pyramiden mit D_i als Grundfläche zerlegt wurde, werden diese noch einmal durch das Lot L_i unterteilt. Für die nachfolgende Drehung bildet L_i immer die x'-Achse. Die y' und z' Achse werden dann für die drei Teilpyramiden nach Formel 3.3 bestimmt.

Die Rotationsmatrix ist dann bestimmt durch

$$\mathbf{R}_{i,j} = \begin{bmatrix} \cos\left(\sphericalangle\left(\mathbf{x},\mathbf{x}'\right)\right) & \cos\left(\sphericalangle\left(\mathbf{x},\mathbf{y}'\right)\right) & \cos\left(\sphericalangle\left(\mathbf{x},\mathbf{z}'\right)\right) \\ \cos\left(\sphericalangle\left(\mathbf{y},\mathbf{x}'\right)\right) & \cos\left(\sphericalangle\left(\mathbf{y},\mathbf{y}'\right)\right) & \cos\left(\sphericalangle\left(\mathbf{y},\mathbf{z}'\right)\right) \\ \cos\left(\sphericalangle\left(\mathbf{z},\mathbf{x}'\right)\right) & \cos\left(\sphericalangle\left(\mathbf{z},\mathbf{y}'\right)\right) & \cos\left(\sphericalangle\left(\mathbf{z},\mathbf{z}'\right)\right) \end{bmatrix} \tag{3.4}$$

Durch Einsetzten und Auflösen[2] ergibt sich:

$$\mathbf{R}_{i,j} = \begin{bmatrix} \frac{L_{i,x}}{|L_i|} & \frac{E_{1,x}^{(D_i)} - L_{i,x}}{\sqrt{-2\left(E_1^{(D_i)}\cdot L_i\right) + \left|E_1^{(D_i)}\right|^2 + |L_i|^2}} & \frac{L_{i,y}\,E_{1,z}^{(D_i)} - L_{i,z}\,E_{1,y}^{(D_i)}}{\sqrt{-\left(E_1^{(D_i)}\cdot L_i\right)^2 + |L_i|^2\left|E_1^{(D_i)}\right|^2}} \\ \frac{L_{i,y}}{|L_i|} & \frac{E_{1,y}^{(D_i)} - L_{i,y}}{\sqrt{-2\left(E_1^{(D_i)}\cdot L_i\right) + \left|E_1^{(D_i)}\right|^2 + |L_i|^2}} & \frac{L_{i,z}\,E_{1,x}^{(D_i)} - L_{i,x}\,E_{1,z}^{(D_i)}}{\sqrt{-\left(E_1^{(D_i)}\cdot L_i\right)^2 + |L_i|^2\left|E_1^{(D_i)}\right|^2}} \\ \frac{L_{i,z}}{|L_i|} & \frac{E_{1,z}^{(D_i)} - L_{i,z}}{\sqrt{-2\left(E_1^{(D_i)}\cdot L_i\right) + \left|E_1^{(D_i)}\right|^2 + |L_i|^2}} & \frac{L_{i,x}\,E_{1,y}^{(D_i)} - L_{i,y}\,E_{1,x}^{(D_i)}}{\sqrt{-\left(E_1^{(D_i)}\cdot L_i\right)^2 + |L_i|^2\left|E_1^{(D_i)}\right|^2}} \end{bmatrix} \tag{3.5}$$

So läßt sich jede Teilpyramide $P'_{i,j}$ so drehen, daß man $P_{i,j}$ erhält, welche durch die vier Parameter a, b, c, d (wie aus Abb. 3.3 ersichtlich) beschrieben ist und den Bedingungen

[2]im Anhang E.2 findet sich ein MAPLE 5 Release 3-Programm, das diese Berechnungen ausführt und passenden Fortran-Code erzeugt.

3.2 entspricht. Im weiteren werden sich die Berechnungen von Masse, Schwerpunkt und Trägheitstensor immer auf diese $P_{i,j}$ stützen.

3.3 Volumen und Masse

Die Masse eines Körpers wird mit

$$m_{ges} = \int \mathrm{d}^3 r \; \varrho(r) \quad \text{bzw. bei Massenpunkten} \quad m_{ges} = \sum_{i=1}^{N} m_i \tag{3.6}$$

beschrieben. Da $\varrho(r) = const$ angenommen wird, reicht es, zuerst das Volumen zu berechnen. Das Gesamtvolumen V_{ges} des Polyeders ist die Summe der Volumina der Pyramiden,also

$$V_{ges} = \sum_{i=1}^{N} \sum_{j=1}^{3} V_{i,j}. \tag{3.7}$$

Das Volumen der einer Teilpyramide $P_{i,j}$ ergibt sich durch das Spatprodukt

$$V_{i,j=}\frac{1}{6} * \left(\begin{pmatrix} a \\ 0 \\ 0 \end{pmatrix} \times \begin{pmatrix} a \\ d \\ 0 \end{pmatrix} \right) * \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \frac{1}{6} acd. \tag{3.8}$$

Somit ist die Gesamtmasse $m_{ges} = V_{ges} \cdot \varrho$ und die Masse einer Teilpyramide $m_{i,j} = V_{i,j} \cdot \varrho$.

3.4 Schwerpunkt

Der Schwerpunkt liegt bei

$$\mathbf{x}_{\mathrm{SP}} = \frac{\int \mathrm{d}^3 r \; \mathbf{r}}{V} \quad \text{bzw. für Massenpunkte bei} \quad \mathbf{x}_{\mathrm{SP}} = \frac{\sum_{i=1}^{N} V_i \mathbf{x^{(i)}}}{\sum_{i=1}^{N} V_i}, \tag{3.9}$$

wobei wieder $\varrho(r) = const = \varrho$ angenommen wird.
Der Schwerpunkt eines Teilelementes $P_{i,j}$ ergibt sich durch die Integration

$$\begin{aligned} \mathbf{x}_{i,j}^{SP} &= \frac{1}{V_{i,j}} \cdot \left(\int_0^a \int_{\frac{bx}{a}}^{\frac{xd}{a}} \int_0^{\frac{\left(y-\frac{xd}{a}\right)c}{(b-d)}} \mathbf{x} dz \, dy \, dx + \int_0^a \int_0^{\frac{bx}{a}} \int_0^{\frac{cy}{b}} \mathbf{x} dz \, dy \, dx \right) = \\ &= \frac{1}{4} \begin{pmatrix} 3a \\ b+d \\ c \end{pmatrix} \end{aligned} \tag{3.10}$$

Diese so gefundenen $\mathbf{x}_{i,j}^{SP}$ müssen dann noch ins das Koordinatensystem des Polyeders über $\mathbf{x'}_{i,j}^{SP} = \mathbf{R}_{i,j}\mathbf{x}_{i,j}^{SP}$ zurückgedreht werden. Dann kann der Schwerpunkt durch Gleichung 3.9 bestimmt werden.

$$\mathbf{x}^{SP} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{3} m_{i,j}\mathbf{x'}_{i,j}^{SP}}{m_{ges}} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{3} m_{i,j}\mathbf{R}_{i,j}\mathbf{x}_{i,j}^{SP}}{m_{ges}} \tag{3.11}$$

3.5 Trägheitstensor

Diese Berechnungen werden erst durchgeführt, wenn der Schwerpunkt des Polyeders in $(0,0,0)$ liegt. Dabei gilt allgemein für den Trägheitstensor

$$I_{(\mu\nu)} = \int \mathrm{d}^3x\ \varrho(x)\left[\mathbf{x}^2\delta_{\mu\nu} - x_\mu x_\nu\right] \tag{3.12}$$

oder

$$\mathbf{I} = \int \mathrm{d}^3x\ \varrho(x) \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -xy & z^2+x^2 & -yz \\ -xz & -yz & x^2+y^2 \end{bmatrix}. \tag{3.13}$$

Auch hier soll wieder zuerst der Trägheitstensor von $P_{i,j}$ für $\varrho(r) = const$ bestimmt werden.

$$\begin{aligned}\mathbf{I}_{i,j} = &\int_0^a \int_{\frac{bx}{a}}^{\frac{xd}{a}} \int_0^{\frac{\left(y-\frac{xd}{a}\right)c}{(b-d)}} \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -xy & z^2+x^2 & -yz \\ -xz & -yz & x^2+y^2 \end{bmatrix} dz\,dy\,dx + \\ &+ \int_0^a \int_0^{\frac{bx}{a}} \int_0^{\frac{cy}{b}} \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -xy & z^2+x^2 & -yz \\ -xz & -yz & x^2+y^2 \end{bmatrix} dz\,dy\,dx\end{aligned} \tag{3.14}$$

Ausintegriert ist dies:

$$\mathbf{I}_{i,j} = \frac{1}{60}\,c\,a\,d \begin{bmatrix} b^2+bd+c^2+d^2 & -2\,a\,(b+d) & -2\,ca \\ -2\,a\,(b+d) & c^2+6\,a^2 & -\frac{1}{2}c\,(2\,b+d) \\ -2\,ca & -\frac{1}{2}c\,(2\,b+d) & b^2+bd+6\,a^2+d^2 \end{bmatrix} \tag{3.15}$$

$\mathbf{I}_{i,j}$ wird dann mit Hilfe des Steinerschen Satzes

$$I'_{(\mu\nu)} = \sum_{\sigma\tau=1}^{3} R_{(\mu\sigma)}R_{(\nu\tau)}(I_{(\sigma\tau)} + m[\mathbf{a}^2\delta_{\sigma\tau} - a_{\sigma\tau}]) \tag{3.16}$$

in das Schwerpunktssystem des Polyeders transformiert.Dabei ist **a** der Vektor zwischen dem Ursprung und dem Schwerpunkt der Pyramide $P_{i,j}$

Da der Trägheitstensor in der Massendichte $\varrho(r)$ linear ist, ist er additiv. Das bedeutet, daß der Trägheitstensor eines Körpers, der aus mehreren starren Teilkörpern zusammengefügt ist, gleich der Summe der einzelnen Trägheitstensoren ist, und somit

$$\mathbf{I}_{ges} = \sum_{i=1}^{N} \sum_{j=1}^{3} \mathbf{I}'_{i,j} \tag{3.17}$$

gilt.

Die so berechneten Massen, Schwerpunkte und Trägheitstensoren aller Partikel werden dann benötigt, um Kräfte und Drehmomente bei der Kollision zu berechnen und die Bewegungsgleichungen zu lösen.

3.6 Klassische Lösungsverfahren für Differentialgleichungen und ihre Probleme

Die wohl einfachste aller Methoden, um Differentialgleichungen der Form $\dot{y} = f(y)$ zu lösen, ist das Eulersches Verfahren. Ausgehend vom Startwert $x_0 = x(t_0)$ wird $x(t_0+h)$ zum Zeitpunkt $t_0 + h$ berechnet, dabei ist h die Schrittweite:

$$x(t_0 + h) = x(t_0) + h * \dot{x}(t_0) \tag{3.18}$$

Man macht also nur einen Schritt in Richtung der 1. Ableitung. So einfach diese Methode erscheint, so unbrauchbar ist sie für eine realistische Simulation. Der Grund, warum dieses Lösungsverfahren trotzdem immer wieder erwähnt wird, ist, daß sich an ihm alle Probleme, die die numerische Lösung von Differentialgleichungen mit sich bringt, leicht beschreiben lassen. Da sich in der weiterführenden Literatur [PTVF92, Hen68, Gea71, WB93, Gar94] sorgfältigste Untersuchungen dieses Verfahrens in Bezug auf Konvergenz, Stabilität, Fehlerabschätzungen etc. finden, soll nur in Abbildung 3.5 das Grundprinzip dargestellt werden und in Abbildung 3.6 die wichtigsten Fehler dargestellt werden.

Sehr viel genauer und zuverlässiger ist das Runge-Kutta-Verfahren. Die Grundidee dabei ist es, durch Berechnung an verschieden Positionen im Intervall $[t, t + h]$ eine genauere Bestimmung von δx zu ermöglichen. Das am meisten verwendete im Bereich

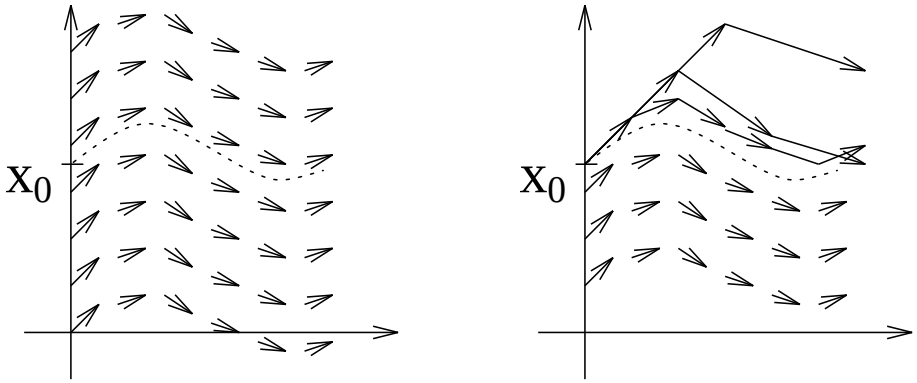

Abbildung 3.5: Der Funktionsverlauf wird durch einen Polygonzug approximiert. Bei unterschiedlichen Schrittweiten folgen die einzelnen Schritte in Richtung der Ableitung dem wirklichen Kurvenverlauf mehr oder weniger genau.

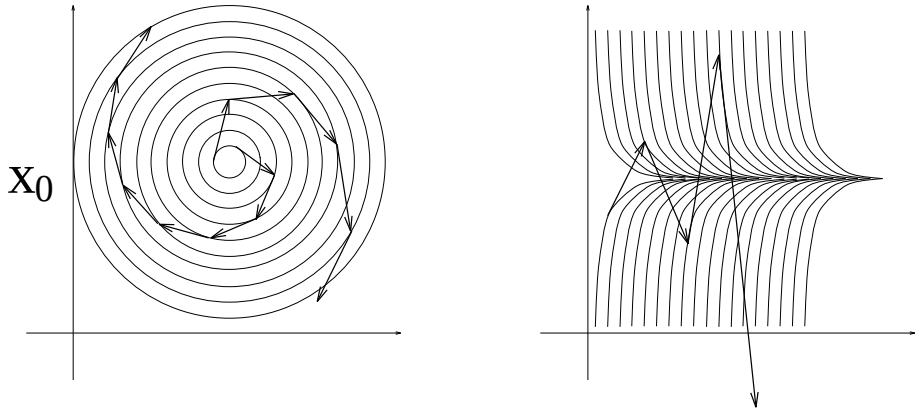

Abbildung 3.6: Eigentlich sollte die Funktion links ein geschlossener Kreis sein, aber mit dem Eulerverfahren wird man immer eine Spirale beschreiben. Rechts sieht man, daß das Verfahren instabil werden kann, wenn die Schrittweite zu groß wird.

der Computational Physics ist immer noch das Runge-Kutta-Verfahren 4. Ordnung.

$$\begin{aligned} k_1 &= hf(x_0, t_0) \quad \text{Euler!} \\ k_2 &= hf\left(x_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}\right) \\ k_3 &= hf\left(x_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}\right) \\ k_4 &= hf\left(x_0 + k_3, t_0 + h\right) \\ x(t_0 + h) &= x_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \end{aligned} \tag{3.19}$$

Man sieht, welches Problem bei der Simulation granularer Medien auftritt, wenn das Runge-Kutta-Verfahren Anwendung findet: Es wäre notwendig, für einen Zeitschritt viermal eine vollständige Berechnungen aller Kollisionen und der daraus resultierenden Kräfte und Drehmomente durchzuführen. Da dieser Vorgang aber sehr viel mehr Rechenzeit als der ODE[3]-Solver benötigt und damit den Hauptanteil an der Gesamtlaufzeit hat, kann dieses Verfahren aus Effizienzgründen keine Anwendung finden. Auf die Herleitung und genaue Beschreibung dieses und anderer Runge-Kutta-Verfahren [4] höherer Ordnung wird deshalb nicht eingegangen werden.

3.7 6-Value Gear-Predictor-Corrector-Methode

Diesem Absatz sei ein Zitat aus den „Numerical recipes“ [PTVF92] vorangestellt:

> For many scientific users, fourth-order Runge-Kutta is not just the first word on ODE integrates, but the last word as well. In fact, you can get pretty far on this old workhorse, especially if you combine it with an adaptive stepsize algorithm. Keep in mind, however, that the old workhorses's last trip may well be take you to the poorhouse: Bulirsch-Stör or predictor-corrector methods can be very much more efficient for problems where very high accuracy is a requirement. Those methods are the high-strung racehorse. Runge-Kutta is for ploughing the fields.

Gegeben sei die Differentialgleichung $y^{(l)} = f(t, y, y', y'', \cdots, y^{(l-1)})$ und ein $k < l$. Dann sei $\mathbf{y}$ ein Vektor, der y und die ersten k Ableitungen von y enthält, und die Form

$$\mathbf{y} = \begin{pmatrix} y \\ y' \\ y'' \\ \vdots \\ y^{(k)} \end{pmatrix} \tag{3.20}$$

hat. B sei eine $(k+1) \times (k+1)$ Matrix, sowie $\mathbf{c}$ ein Vektor mit der $(k+1)$ Elementen. Dann kann nach [Gea71] jede Multivalue-[5]Predictor-Corrector-Methode in der Form

[3] ODE (ordinary differential equation) = gewöhnliche Differentialgleichung

[4] Ein sehr übersichtliche Beschreibung und Herleitung sowie Beispielprogramme in MatLab findet sich in [Gar94].

[5] Hier werden nur die sog. Multivalue-Verfahren betrachtet. Bei Multistep-Verfahren erfolgt der Predictorschritt mit Hilfe der letzten Schritte der Lösungsverfahren. Im Vergleich zu den Multivalue-Verfahren haben sie den Nachteil, nur sehr schwer für ein „Adaptive Stepsize“ geeignet zu sein. Außerdem müssen vor Beginn der Simulationen z.B. mit Hilfe von Runge-Kutta Startwerte für die „letzten“ Schritte bestimmt werden, damit das Verfahren korrekt beginnen kann. Zudem ist es sehr aufwendig, während des Laufs die Ordnung des Verfahrens zu ändern. In [Hen68, Gea71] wird noch erwähnt, daß bei Multivalue meist die Koeffizienten c kleiner sind als bei Multistep, dies kommt der Genauigkeit in Bezug auf Rundungsfehler zugute.

$$\begin{aligned} \mathbf{y}_{n,(0)} &= B\mathbf{y}_{n-1} \quad \text{Predictor} \\ \mathbf{y}_{n,(m+1)} &= \mathbf{y}_{n,(m)} + \mathbf{c}G(\mathbf{y}_{n,(m)}) \quad m \geq 0 \quad \text{Corrector} \end{aligned} \tag{3.21}$$

geschrieben werden. Das Prinzip des Verfahrens ist es, zuerst mit Hilfe der ersten k Ableitungen von y einen neuen Wert für diese Ableitungen vorauszusagen (Predictor). Dann wird mit Hilfe von $f(t, y, y', y'', \cdots, y^{(l-1)})$ an dieser vorhergesagten Stelle eine Korrektur für diese Werte berechnet (Corrector), dies kann m–mal geschehen. Auffällig ist, daß der Predictor*nicht* von der zu lösenden Differentialgleichung abhängt, sondern nur von $\mathbf{y}$.
Für den Predictor beim 6-Value-Gear-Predictor-Corrector geht man von der Taylorentwicklung

$$f(a+h) = \sum_{n=0}^{\infty} \frac{h^n}{n!} f^{(n)}(a) \tag{3.22}$$

aus. Dabei werden dann $f(t)$,$\dot{f}(t)$ bis $f^{(4)}(t)$ jeweils entwickelt, wobei immer nach $f^{(5)}$ abgebrochen wird.

$$\begin{aligned} f(t+\delta t) &= f(t) + \delta t \frac{\mathrm{d}f(t)}{\mathrm{d}t} + \frac{\delta t^2}{2}\frac{\mathrm{d}^2 f(t)}{\mathrm{d}t^2} + \frac{\delta t^3}{6}\frac{\mathrm{d}^3 f(t)}{\mathrm{d}t^3} + \frac{\delta t^4}{24}\frac{\mathrm{d}^4 f(t)}{\mathrm{d}t^4} + \frac{\delta t^5}{120}\frac{\mathrm{d}^5 f(t)}{\mathrm{d}t^5} \\ \frac{\mathrm{d}f(t+\delta t)}{\mathrm{d}t} &= \frac{\mathrm{d}f(t)}{\mathrm{d}t} + \delta t \frac{\mathrm{d}^2 f(t)}{\mathrm{d}t^2} + \frac{\delta t^2}{2}\frac{\mathrm{d}^3 f(t)}{\mathrm{d}t^3} + \frac{\delta t^3}{6}\frac{\mathrm{d}^4 f(t)}{\mathrm{d}t^4} + \frac{\delta t^4}{24}\frac{\mathrm{d}^5 f(t)}{\mathrm{d}t^5} \\ \frac{\mathrm{d}^2 f(t+\delta t)}{\mathrm{d}t^2} &= \frac{\mathrm{d}^2 f(t)}{\mathrm{d}t^2} + \delta t \frac{\mathrm{d}^3 f(t)}{\mathrm{d}t^3} + \frac{\delta t^2}{2}\frac{\mathrm{d}^4 f(t)}{\mathrm{d}t^4} + \frac{\delta t^3}{6}\frac{\mathrm{d}^5 f(t)}{\mathrm{d}t^5} \\ &\vdots \\ \frac{\mathrm{d}^5 f(t+\delta t)}{\mathrm{d}t^5} &= \frac{\mathrm{d}^5 f(t)}{\mathrm{d}t^5} \end{aligned} \tag{3.23}$$

Die einzelnen Ableitungen werden dann zeitskaliert, so daß mit $\mathbf{r}_0 = f(t)$ gilt

$$\mathbf{r}_n = \frac{\delta t^n}{n!}\frac{\mathrm{d}^n \mathbf{r}_0}{\mathrm{d}t^n}, \tag{3.24}$$

also $\mathbf{r}_1(t) = \delta t \frac{\mathrm{d}\mathbf{r}_0}{\mathrm{d}t}$,$\mathbf{r}_2(t) = \frac{\delta t^2}{2}\frac{\mathrm{d}^2\mathbf{r}_0}{\mathrm{d}t^2}$ usw. Damit läßt sich dann das Gleichungssystem 3.23 in der Form

$$\begin{pmatrix} \mathbf{r}_0^p(t+\delta t) \\ \mathbf{r}_1^p(t+\delta t) \\ \mathbf{r}_2^p(t+\delta t) \\ \mathbf{r}_3^p(t+\delta t) \\ \mathbf{r}_4^p(t+\delta t) \\ \mathbf{r}_5^p(t+\delta t) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 3 & 6 & 10 \\ 0 & 0 & 0 & 1 & 4 & 10 \\ 0 & 0 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{r}_0(t) \\ \mathbf{r}_1(t) \\ \mathbf{r}_2(t) \\ \mathbf{r}_3(t) \\ \mathbf{r}_4(t) \\ \mathbf{r}_5(t) \end{pmatrix} \tag{3.25}$$

schreiben. Dabei ist die Matrix die Pascalsche Dreiecksmatrix, die allgemein bestimmt ist durch

$$P_{i,j} = \begin{cases} 1 & \text{falls } i = 1 \\ 0 & \text{falls } i > j \\ P_{i,j-1} + P_{i-1,j-1} & \text{sonst} \end{cases} \tag{3.26}$$

mit $i, j = 1, \ldots, n$.
Damit ist der Predictor passend zu Gleichung 3.21 bestimmt.

p	k	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
1	3	$\frac{5}{12}$	1	$\frac{1}{2}$					
	4	$\frac{3}{8}$	1	$\frac{3}{4}$	$\frac{1}{6}$				
	5	$\frac{251}{720}$	1	$\frac{11}{12}$	$\frac{1}{3}$	$\frac{1}{24}$			
	6	$\frac{95}{288}$	1	$\frac{25}{24}$	$\frac{35}{72}$	$\frac{5}{48}$	$\frac{1}{120}$		
	7	$\frac{19087}{60480}$	1	$\frac{137}{120}$	$\frac{5}{8}$	$\frac{17}{96}$	$\frac{1}{40}$	$\frac{1}{720}$	
	8	$\frac{5257}{17280}$	1	$\frac{49}{40}$	$\frac{203}{270}$	$\frac{49}{192}$	$\frac{7}{144}$	$\frac{7}{1440}$	$\frac{1}{5040}$
2	4	$\frac{1}{6}$	$\frac{5}{6}$	1	$\frac{1}{3}$				
	5	$\frac{19}{120}$	$\frac{3}{4}$	1	$\frac{1}{2}$	$\frac{1}{12}$			
	6	$\frac{3}{20}$	$\frac{251}{360}$	1	$\frac{11}{18}$	$\frac{1}{6}$	$\frac{1}{60}$		
	7	$\frac{863}{6048}$	$\frac{665}{1008}$	1	$\frac{25}{36}$	$\frac{35}{144}$	$\frac{1}{24}$	$\frac{1}{360}$	
	8	$\frac{1925}{14112}$	$\frac{19087}{30240}$	1	$\frac{137}{180}$	$\frac{5}{16}$	$\frac{17}{240}$	$\frac{1}{120}$	$\frac{1}{2520}$
3	5	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{5}{4}$	1	$\frac{1}{4}$			
	6	$\frac{3}{80}$	$\frac{19}{40}$	$\frac{9}{8}$	1	$\frac{3}{8}$	$\frac{1}{20}$		
	7	$\frac{221}{5040}$	$\frac{9}{20}$	$\frac{251}{240}$	1	$\frac{11}{24}$	$\frac{1}{10}$	$\frac{1}{120}$	
	8	$\frac{2185}{46368}$	$\frac{863}{2016}$	$\frac{95}{96}$	1	$\frac{25}{48}$	$\frac{49}{336}$	$\frac{1}{48}$	$\frac{1}{840}$
4	6	$\frac{1}{30}$	$\frac{1}{10}$	1	$\frac{5}{3}$	1	$\frac{1}{5}$		
	7	$\frac{16}{630}$	$\frac{3}{20}$	$\frac{19}{20}$	$\frac{3}{2}$	1	$\frac{3}{10}$	$\frac{1}{30}$	
	8	$\frac{11}{630}$	$\frac{221}{1260}$	$\frac{9}{10}$	$\frac{251}{180}$	1	$\frac{11}{30}$	$\frac{1}{15}$	$\frac{1}{210}$

Tabelle 3.1: In dieser Tabelle sind die Korrekturfaktoren für Gear-Predictor-Corrector angegeben. p gibt die Ordnung der zu lösenden Differentialgleichung an, k die Ordnung des Predictor-Correctors.

Beim Gear-PC wird nur einmal korrigiert. Dies hat für die Simulation dann den Vorteil, daß auch die Kraftberechnung nur einmal durchgeführt wird.
Jetzt ist zu beachten, daß der Korrekturschritt abhängig von der Ordnung der zu lösenden Differentialgleichung ist.

Es wird $\Delta\mathbf{r}$ berechnet, das den Unterschied zwischen dem aus Gleichung 3.25 vorhergesagten $\mathbf{r}_n^p$ und dem Wert, der durch Anwendung der DGL auf die vorhergesagten Werte berechnet wurde, angibt.
Für eine Differentialgleichung der Form

$$\dot{\mathbf{r}} = f(\mathbf{r}) \qquad \Delta\mathbf{r} = \mathbf{r}_1^c - \mathbf{r}_1^p \qquad \mathbf{r}_1^c = f(\mathbf{r}_0^p) \tag{3.27}$$

$$\ddot{\mathbf{r}} = f(\mathbf{r}) \quad \text{ist} \quad \Delta\mathbf{r} = \mathbf{r}_2^c - \mathbf{r}_2^p \quad \text{mit} \quad \mathbf{r}_2^c = f(\mathbf{r}_0^p). \tag{3.28}$$

$$\ddot{\mathbf{r}} = f(\mathbf{r}, \dot{\mathbf{r}}) \qquad \Delta\mathbf{r} = \mathbf{r}_2^c - \mathbf{r}_2^p \qquad \mathbf{r}_2^c = f(\mathbf{r}_0^p, \mathbf{r}_1^p) \tag{3.29}$$

Dann ist der Korrekturschritt durch

$$\begin{pmatrix} \mathbf{r}_0^c(t+\delta t) \\ \mathbf{r}_1^c(t+\delta t) \\ \mathbf{r}_2^c(t+\delta t) \\ \mathbf{r}_3^c(t+\delta t) \\ \mathbf{r}_4^c(t+\delta t) \\ \mathbf{r}_5^c(t+\delta t) \end{pmatrix} = \begin{pmatrix} \mathbf{r}_0^p(t+\delta t) \\ \mathbf{r}_1^p(t+\delta t) \\ \mathbf{r}_2^p(t+\delta t) \\ \mathbf{r}_3^p(t+\delta t) \\ \mathbf{r}_4^p(t+\delta t) \\ \mathbf{r}_5^p(t+\delta t) \end{pmatrix} + \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix} \Delta\mathbf{r} \tag{3.30}$$

gegeben.
Die Parameter c sind aus Tabelle 3.1 ersichtlich. Im Fall der DGL der Form $\ddot{\mathbf{r}} = f(\mathbf{r}, \dot{\mathbf{r}})$ wird in [AT86] vorgeschlagen, beim 5-Value-PC für c_0 den Wert $\frac{19}{120}$ durch $\frac{19}{90}$, und beim 6-Value-PC $\frac{3}{20}$ durch $\frac{3}{16}$ zu ersetzten.

Kapitel 4

Kollisionsdetektion

Die Überprüfung, ob sich zwei Sandkörner berühren, ist bei der Simulation der entscheidende Faktor. Wenn nun die Simulation soweit ist, daß also die Dynamik eines einzelnen Sandkorns unter dem Einfluß von Gravitation und evtl. anderen Kräften berechnet werden kann, dann kommt der Teil, der am sorgfältigsten untersucht werden muß. Es gilt folgende Probleme zu beachten, die auftreten, wenn ein „einfacher" Algorithmus verwendet wird.

- Der Rechenaufwand der Kollisionsbestimmungen steigt mit $\mathcal{O}(n^2)$.
- Der Rechenaufwand der Kollisionsbestimmung zwischen zwei Polyedern steigt mit $\mathcal{O}((n_V + n_E + n_F)^2)$.

Dabei wird jedes Polyeder mit jedem anderen geschnitten. Bei den einzelnen Kollisionsprüfungen wiederum wird jeder Bestandteil des einen Sandkorns mit allen des anderen verglichen. Dies ist sicherlich die am leichtesten zu realisierende Lösung des Problems, aber eben auch die schlechteste. Denn das Zeitverhalten ist extrem ungünstig, die in jedem einzelnen Zeitschritt zu leistende Rechenarbeit ist enorm. Deswegen sollte man sich an den Grundsatzhalten „Denkzeit $\geq$ Rechenzeit" halten. Eine wesentlich bessere Methode ist es, das Problem schrittweise zu lösen, d.h. mit schnellen ungenauen Tests zu beginnen, und erst danach zeitaufwendigere Tests, die aber sicherer arbeiten, zu verwenden. Der allgemein übliche Ansatz ist es dabei, zuerst „Bounding–Boxes " zu definieren, die aufgrund ihrer vereinfachten Geometrie sehr schnell zu prüfen sind, und erst, wenn sich diese schneiden, auch die Körper selbst zu prüfen. Dies bringt einen wesentlichen Zeitgewinn.
Doch auch hiermit ist noch nicht das Optimum erreicht. Zum einen bleibt die Schwierigkeit, daß auch die paarweise Überprüfung der Bounding–Boxes mit $\mathcal{O}(n^2)$ arbeiten, wenn auch mit einem wesentlich geringeren Zeitfaktor. Zum anderen bleibt das Problem der Kollisionsdetektion zwischen zwei Polyedern bestehen. Deswegen erschien es wichtig, noch deutlich bessere Algorithmen zu finden, um später mit ausreichender Komplexität simulieren zu können. Für die Überprüfung der Bounding–Box wurde auf einem Algorithmus von David Baraff aufgebaut [Bar93]; für die Kollisionsdetektion fand ein Distanzalgorithmus von Ming C. Lin Anwendung. Diese Algorithmen

sind zwar sehr komplex, insbesondere der zweitgenannte, bieten aber beide ein hervorragendes Zeitverhalten. Im Fall der Bounding–Boxes findet man $\mathcal{O}(n)$, bei der Abstandsbestimmung sogar $\mathcal{O}(c)$ mit $c = const.$

4.1 Bounding–Boxes

Allgemein kann man B_A als Bounding–Box von A bezeichnen, wenn der Körper A innerhalb von B_A liegt, also $A \subset B_A$ und B_A eine für einen Algorithmus günstigere Form hat. Das Wort „Box“ suggeriert zwar quaderförmige Gestalt, trotzdem sind auch andere Geometrien sinnvoll. Gebräuchlich sind Quadrate, Rechtecke und Kreise in 2 Dimensionen und Quader, Würfel und Kugeln in 3 Dimensionen. Außerdem ist es günstig, wenn die Bounding–Box so klein wie möglich ist.
Der Sinn einer Bounding–Box liegt darin, ein schnelleres Prüfen auf *NICHT*-Kollision zu ermöglichen. Denn gibt es keinen Punkt $x \in \mathbb{R}^3$ mit $x \in B_A \wedge x \in B_{A'}$, dann kann es wegen $A \subset B_A$,$A' \subset B_{A'}$ auch keinen Punkt $x \in A \wedge x \in A'$ geben. Somit ist in diesem Fall eine genauere Untersuchung auf Kollision nicht nötig. Allerdings gibt es Punkte, für die $x \in \mathbb{R}^3$ für die $x \in B_A \wedge x \in B_{A'}$ gilt, aber $x \notin A \vee x \notin A'$. Es können sich also die Bounding–Boxes schneiden, ohne daß sich die darin enthaltenen Körper berühren. Für diesen Fall sind dann weitere Berechnungen notwendig.
Als Faustregel kann man sagen, daß eine Bounding–Box immer so klein wie möglich und so groß wie nötig sein sollte. Es kann unter Umständen sinnvoll sein, die Box größer zu wählen als die von der Geometrie her gesehen „optimale“ Gestalt. So wäre es denkbar, wenn in der Simulation die Polyeder von der Gestalt sind, daß die Ecken auf einem Ellipsoid liegen, als Bounding–Boxes Ellipsoide zu verwenden. Dann würde man jede Box gegen jede andere prüfen und im Fall, daß sich die Ellipsoide berühren, dann die entsprechenden Körper im Inneren zu prüfen. Im Vergleich zu einer direkten Prüfung würde dies eine deutliche Verbesserung darstellen, doch ist eine Zeitersparnis auch noch an anderer Stelle möglich. Zuerst soll also ein Algorithmus vorgestellt werden, der in $\mathcal{O}(n)$ (mit n als Zahl der Polyeder) die Kollisionen zwischen allen Bounding–Box es prüfen kann. Dann sollen verschiedene Möglichkeiten vorgestellt werden, um die zu diesem Algorithmus passenden Bounding–Box zu berechnen. Außerdem sollen die jeweiligen Vor- und Nachteile und das Zeitverhalten untersucht werden.

4.1.1 Eindimensionaler Fall

Zuerst soll der Algorithmus am eindimensionalen Fall untersucht werden. Die n Bounding–Boxes stellen also Intervall I_i mit $i = 1..n$ auf der Koordinatenachse dar. Das einzelne Intervall kann durch $[b_i, e_i]$ beschrieben werden, wobei $b_i, e_i \in \mathbb{R}$ und $b_i < e_i$. Nun kollidieren zwei Boxen i und j, wenn für die Intervalle $[b_i, e_i]$,$[b_j, e_j]$ mit $i \neq j$ gilt

$$b_i \in [b_j, e_j] \vee e_i \in [b_j, e_j]. \tag{4.1}$$

Sortiert man b_i, b_j, e_i, e_j der Größe nach in aufsteigender Reihenfolge und sieht sich die *zulässigen* Kombinationen genau an, ist leicht zu erkennen, daß immer dann eine Kollision vorliegt, wenn im Bereich eines Intervalls der Anfang oder das Ende eines anderen Intervalls liegt.

b_j, e_j, b_i, e_i	Keine Kollision
b_i, e_i, b_j, e_j	Keine Kollision
b_i, b_j, e_i, e_j	Kollision
b_i, b_j, e_j, e_i	Kollision
b_j, b_i, e_j, e_i	Kollision
b_j, b_i, e_i, e_j	Kollision

Tabelle 4.1: Zulässige Kombinationen, wenn die Intervallgrenzen von $[b_i, e_i]$, $[b_j, e_j]$ in aufsteigender Reihenfolge sortiert werden. Alle anderen Konfigurationen sind wegen $b_k < e_k$ nicht möglich.

In gleicher Weise kann man für n Intervalle verfahren. Zuerst werden alle b_i und e_i mit $i = 1..n$ in aufsteigender Reihenfolge in eine Liste L sortiert.

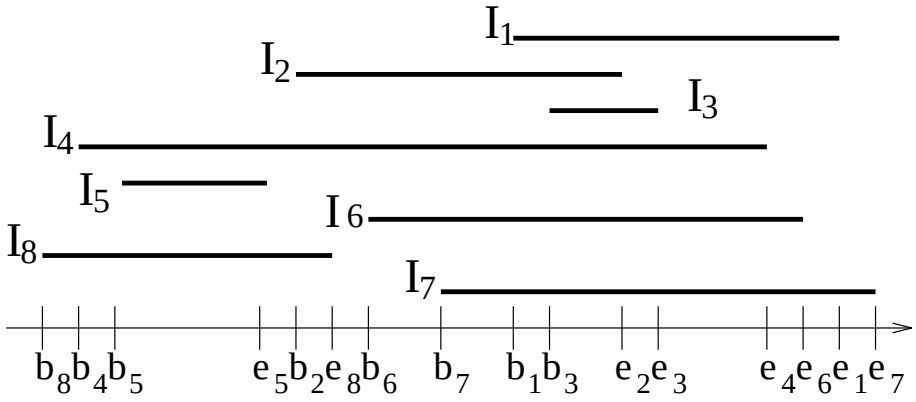

Abbildung 4.1: Im eindimensionalen sind Bounding–Boxes nur Intervalle mit dem Anfang b_i und dem Ende e_i. Sortiert man die b_i und e_i der Größe nach, kann man alle Kollisionen ablesen.

Dann wird eine zweite, im Moment noch leere Liste L_{akt} erzeugt, in der „aktuelle Intervalle“ gespeichert werden können. Danach wird die Liste, beginnend mit dem kleinsten Eintrag, abgearbeitet. Ist der Eintrag vom Typ b_i, so wird das Intervall i in L_{akt} eingetragen. Außerdem kollidieren alle Intervalle aus L_{akt} mit dem Intervall i. Ist der Eintrag allerdings ein e_i, wird i aus L_{akt} entfernt. Ist der letzte Eintrag der Liste abgearbeitet, dann sind alle Kollisionen entdeckt. Einen Ablauf für die Konfiguration aus Abbildung 4.1 ist in Tabelle 4.2 zu finden. Der Zeitbedarf für das Einsortieren

i	L_i	Aktion	L_{akt}(alt)	gef. Kollisionen
1	b_8	8 in L_{akt} einf.	ø	ø
2	b_4	4 in L_{akt} einf.	8	(4;8)
3	b_5	5 in L_{akt} einf.	$8 \rightarrow 4$	(5;8), (5;4)
4	e_5	5 in L_{akt} lösch.	$8 \rightarrow 4 \rightarrow 5$	keine Prüfung
5	b_2	2 in L_{akt} einf.	$8 \rightarrow 4$	(2;8) , (2;4)
6	e_8	8 in L_{akt} lösch.	$8 \rightarrow 4 \rightarrow 2$	keine Prüfung
7	b_6	6 in L_{akt} einf.	$4 \rightarrow 2$	(6;4), (6;2)
8	b_7	7 in L_{akt} einf.	$4 \rightarrow 2 \rightarrow 6$	(7;4), (7;2), (7;6)
9	b_1	1 in L_{akt} einf.	$4 \rightarrow 2 \rightarrow 6 \rightarrow 7$	(1;4), (1;2), (1;6), (1;7)
10	b_3	3 in L_{akt} einf.	$4 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 1$	(3;4), (3;2), (3;6), (3;7), (3;1)
11	e_2	2 in L_{akt} lösch.	$4 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 1 \rightarrow 3$	keine Prüfung
12	e_3	3 in L_{akt} lösch.	$4 \rightarrow 6 \rightarrow 7 \rightarrow 1 \rightarrow 3$	keine Prüfung
13	e_4	4 in L_{akt} lösch.	$4 \rightarrow 6 \rightarrow 7 \rightarrow 1$	keine Prüfung
14	e_6	6 in L_{akt} lösch.	$6 \rightarrow 7 \rightarrow 1$	keine Prüfung
15	e_1	1 in L_{akt} lösch.	$7 \rightarrow 1$	keine Prüfung
16	e_7	7 in L_{akt} lösch.	7	keine Prüfung

Tabelle 4.2: Für die Liste L=($b_8 \rightarrow b_4 \rightarrow b_5 \rightarrow e_5 \rightarrow b_2 \rightarrow e_8 \rightarrow b_6 \rightarrow b_7 \rightarrow b_1 \rightarrow b_3 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_6 \rightarrow e_1 \rightarrow e_7$) werden alle 19 Kollisionen { (4;8), (5;8), (5;4), (2;8) , (2;4), (6;4), (6;2)}, (7;4), (7;2), (7;6), (1;4), (1;2), (1;6), (1;7), (3;4), (3;2), (3;6), (3;7), (3;1) } gefunden.

von b und e in die Liste L geht mit $\mathcal{O}(n \log n)$. Das Abarbeiten der Liste L und das Bearbeiten von L_{akt} erfolgt mit $\mathcal{O}(n)$. Die Bestimmung der k Overlaps geht mit $\mathcal{O}(k)$. Insgesamt hat man also $\mathcal{O}(n \log n + k)$. Dies ist ein optimaler Algorithmus [Bar93], um den ersten Schritt dieses Problems zu lösen.

Der vom Rechenaufwand her gesehen ungünstigste Vorgang ist mit $\mathcal{O}(n \log n)$ der Sortieralgorithmus. Es ist aber möglich, in den weiteren Zeitschritten noch einmal schneller zu arbeiten, denn man kann ausnützen, daß die simulierten Körper pro Zeitschritt ihre Lage nur geringfügig verändern. Da sich die Konfigurationen also nur langsam ändern, ist ja die Liste L_{alt} des letzten Zeitschrittes immer noch „fast richtig“. Hat sich z.B. die Konfiguration aus Abbildung 4.1 weiterentwickelt und sieht jetzt wie in Abbildung 4.2 aus, dann hat sich zwar die die Größe der Boxes verändert und die Lage der Endpunkte, aber die Liste L hat bis auf eine Änderung denselben Aufbau. Es ist also nur eine Vertauschung notwendig.

Für das Problem des Sortierens in bereits vorsortierten Listen ist eine Methode mit dem Namen *Insertion-sort* entwickelt worden. Damit kann dann mit Hilfe von Permutationen dann aus L_{alt} sehr effizient L bestimmt werden. Eine genaue Beschreibung von Insertion-sort findet sich in Anhang B. Im Prinzip wird dabei ein Element solange zum Anfang der Liste bewegt, bis es größer ist als der Vorgänger. Wenn das Ende der

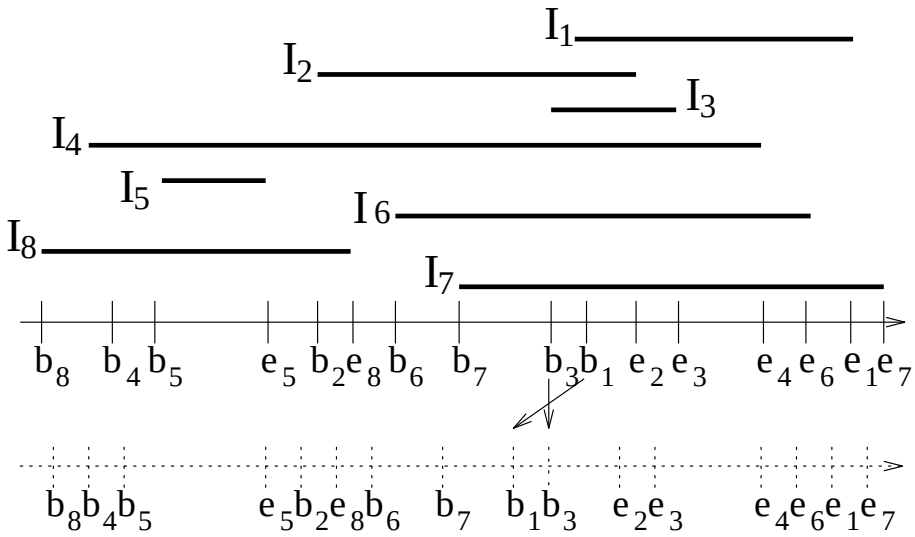

Abbildung 4.2: Obwohl sich gegenüber 4.1 alle Intervalle verschoben haben, haben nur b_1 und b_3 ihren Reihenfolge vertauscht.

Liste erreicht ist, ist sie sortiert. Der Rechenzeitbedarf ist dann $\mathcal{O}(n+c)$, wobei c die Zahl der notwendigen Vertauschungen ist. Eine weitere Verbesserung ist im Hinblick auf die Overlapliste möglich. Hat man im letzten Schritt die Overlaps in eine Matrix $O_{i,j}$ mit $O_{i,j} = O_{j,i}$ gespeichert, wird sich im letzten Beispiel nur der Eintrag in $O_{1,3}$ und $O_{3,1}$ ändern. Wenn zwei Intervalle, die sich im letzten Zeitschritt überlappten, dies nicht mehr tun, müssen b_i und e_j oder aber b_j und e_i ein- oder mehrfach vertauscht worden sein. Wird die Matrix $O_{i,j}$ also schon während des Sortierens geändert, ist sie mit dem Zeitaufwand $\mathcal{O}(n+c)$ wieder auf den aktuellen Stand zu bringen. Da die Zahl der Änderungen des Overlapstatus k und die Zahl der Vertauschungen in der Liste c in einer physikalischen Simulation mit ausreichend kleinen Schritten fast gleich ist, ist die zusätzlich anfallende Arbeit $\mathcal{O}(c-k)$ zu vernachlässigen. Somit ist es also möglich, in einer Dimension alle Kollisionen in der Zeit $\mathcal{O}(n+c)$ zu finden. Der Algorithmus wird umso besser, je weniger Änderungen vorzunehmen sind; in einem Sandpile, der zur Ruhe kommt, ist der Zeitbedarf hierfür minimal.

4.1.2 Dreidimensionaler Fall

Im Prinzip kann man im dreidimensonalen Fall ähnlich wie in einer Dimension vorgehen. Die Bounding–Boxes müssen hier Quader sein, deren Kanten parallel zu den Koordinatenachsen sind. Dann gibt es also drei Intervalle $[b_i^{(x)}, b_i^{(x)}]$,$[b_i^{(y)}, b_i^{(y)}]$ und $[b_i^{(z)}, b_i^{(z)}]$. Der erste Schritt wird wieder sein, die Intervalle in drei Listen einzuordnen. Dabei ist also je eine Liste für die Intervallgrenzen $b_i^{(x)}$ und $e_i^{(x)}$ auf der X–Achse und dasselbe für die Y– und Z–Achse. Dieser Schritt hat, wenn man genauso wie im eindimensionalen

Fall vorgeht, den Zeitbedarf $\mathcal{O}(n+c)$ [1]. Tritt ein Wechsel zwischen zwei Koordinaten i und j auf (egal welche Koordinatenachse), so werden die Bounding–Boxes geprüft, ob sie ihren Overlapstatus verändert haben. Da dieser Vorgang einen konstanten Zeitbedarf hat, werden alle Veränderungen von $O_{i,j}$ mit $\mathcal{O}(n+c)$ gefunden. Auch hier ist die zusätzliche Arbeit, die entsteht, wenn Werte in der Liste getauscht werden, ohne daß sich in $O_{i,j}$ etwas ändert, mit $\mathcal{O}(c-k)$ anzugeben und damit vernachlässigbar.

4.1.3 Bestimmung der Bounding–Boxes

Die Bounding–Box müssen, um für den vorgestellten Algorithmus brauchbar zu sein, Quader sein. Dabei müssen die Kanten des Quaders parallel zu einer Koordinatenachse liegen. Es sollen nun zwei Methoden vorgestellt werden, mit denen in der vorliegenden Simulation Bounding–Boxes berechnet werden können. Dabei ist die erste Methode universell verwendbar, während die zweite auf Besonderheiten dieser Simulation beruht, durch Abwandlungen aber auch zu verallgemeinern ist.

4.1.3.1 Min-Max-Methode

Wenn die K Eckpunkte des Polyeders, um den die Bounding–Box bestimmt werden soll, die Koordinaten (x_i, y_i, z_i) mit $i = 1, \ldots, K$ haben, dann werden die Intervallgrenzen $[b_i^{(x)}, b_i^{(x)}]$,$[b_i^{(y)}, b_i^{(y)}]$ und $[b_i^{(z)}, b_i^{(z)}]$ wie folgt bestimmt:

$$\begin{aligned} b_i^{(x)} &= \min(x_1, x_2, \ldots, x_k) \\ e_i^{(x)} &= \max(x_1, x_2, \ldots, x_k) \\ b_i^{(y)} &= \min(y_1, y_2, \ldots, y_k) \\ e_i^{(y)} &= \max(y_1, y_2, \ldots, y_k) \\ b_i^{(z)} &= \min(z_1, z_2, \ldots, z_k) \\ e_i^{(z)} &= \max(z_1, z_2, \ldots, z_k) \end{aligned} \tag{4.2}$$

Dies ist die kleinste mögliche Bounding–Box im gegebenen Koordinatensystem. Für jedes $b^{(x)} > b_i^{(x)}$ gibt es nach 4.2 eine Ecke mit den Koordinaten (x_m, y_m, z_m), für die $x_m < b^{(x)}$ gilt. Ebenso kann es kein gültiges $e^{(x)} < e_i^{(x)}$ geben. Analoges gilt dann auch für die Y– und Z–Achse.
Gibt es nun N Polyeder mit je maximal K Ecken, dann ist der Aufwand zur Berechnung der Bounding–Boxes $\mathcal{O}(N * K)$.

4.1.3.2 Ellipsoid-Methode

Da in der Simulation für diese Arbeit die Polyeder so gewählt wurden, daß alle Eckpunkte (x_i, y_i, z_i) auf einem Ellipsoid liegen, ist es auch möglich, sozusagen eine

[1] Da bei der $\mathcal{O}$-Schreibweise Vorfaktoren weggelassen werden, schreibt man also nicht $3 \cdot \mathcal{O}(n+c)$, sondern nur $\mathcal{O}(n+c)$, obwohl der Sortiervorgang in 3 Dimensionen die dreifache Zeit benötigt (sieh e auch Anhang E.1).

Bounding–Box um die Bounding–Box zu legen. Man bestimmt analytisch, welche Ausdehnung ein Quader haben muß, um ein Ellipsoid bekannter Größe, dessen Halbachsen *nicht* parallel zu den Koordinatenachsen sind, aufnehmen zu können.
Gegeben sei das Ellipsoid im körperfesten Koordinatensystem $\overline{\mathbf{K}}$ mit der Gleichung

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1. \tag{4.3}$$

Des weiteren ist die Matrix $\mathbf{R}$ bekannt, die einen Vektor $\overline{\mathbf{k}}$ mittels $\mathbf{Rk} = \mathbf{k}$ ins raumfeste System transformiert[2].
Damit hat das Ellipsoid im ortsfesten Koordinatensystem die Gleichung

$$\begin{aligned}\frac{(\mathbf{R}_{1,1}x + \mathbf{R}_{1,2}y + \mathbf{R}_{1,3}z)^2}{a^2} + \frac{(-\mathbf{R}_{1,2}x + \mathbf{R}_{2,2}y + \mathbf{R}_{2,3}z)^2}{b^2} + \\ + \frac{(-\mathbf{R}_{1,3}x - \mathbf{R}_{2,3}y + \mathbf{R}_{3,3}z)^2}{c^2} = 1.\end{aligned} \tag{4.4}$$

Mit

$$\begin{aligned}
n_1 &= \frac{2\,\mathbf{R}_{1,1}\mathbf{R}_{1,2}}{a^2} - \frac{2\,\mathbf{R}_{1,2}\mathbf{R}_{2,2}}{b^2} + \frac{2\,\mathbf{R}_{1,3}\mathbf{R}_{2,3}}{c^2} \\
n_2 &= \frac{2\,\mathbf{R}_{1,1}\mathbf{R}_{1,3}}{a^2} - \frac{2\,\mathbf{R}_{1,2}\mathbf{R}_{2,3}}{b^2} - \frac{2\,\mathbf{R}_{1,3}\mathbf{R}_{3,3}}{c^2} \\
n_3 &= \frac{2\,\mathbf{R}_{1,2}\mathbf{R}_{1,3}}{a^2} + \frac{2\,\mathbf{R}_{2,2}\mathbf{R}_{2,3}}{b^2} - \frac{2\,\mathbf{R}_{2,3}\mathbf{R}_{3,3}}{c^2} \\
n_4 &= \frac{\mathbf{R}_{1,1}{}^2}{a^2} + \frac{\mathbf{R}_{1,2}{}^2}{b^2} + \frac{\mathbf{R}_{1,3}{}^2}{c^2} \\
n_5 &= \frac{\mathbf{R}_{1,2}{}^2}{a^2} + \frac{\mathbf{R}_{2,2}{}^2}{b^2} + \frac{\mathbf{R}_{2,3}{}^2}{c^2} \\
n_6 &= \frac{\mathbf{R}_{1,3}{}^2}{a^2} + \frac{\mathbf{R}_{2,3}{}^2}{b^2} + \frac{\mathbf{R}_{3,3}{}^2}{c^2}
\end{aligned} \tag{4.5}$$

läßt sich die Formel 4.4 in der Form

$$n_1\,x\,y + n_2\,x\,z + n_3\,y\,z + n_4\,x^2 + n_5\,y^2 + n_6\,z^2 \tag{4.6}$$

darstellen. Im weiteren wird sich die Berechnung nur auf die Bestimmung der Maxima auf der X–Achse beschränken. Die Berechnungen für die Y– und Z–Achse verlaufen analog[3].

[2] Es wird hier mit Rotationsmatrizen gearbeitet, da die Rechnung mit Quaternionen deutlich mehr Multiplikationen erfordert und damit zeitaufwendiger ist.

[3] Im Anhang E.2 findet sich ein Programm für MAPLE 5 Release 3, das diese Berechnungen durchführen kann.

Löst man 4.6 nach x auf, so erhält man zwei Lösungen, die die „untere“ und die „obere“ Hälfte des Ellipsoids bestimmen.

$$\begin{aligned} x_{\pm} = & \frac{-n_1 y - n_2 z}{2n_4} \pm \\ & \pm \frac{\sqrt{{n_1}^2 y^2 + 2n_1 y n_2 z + {n_2}^2 z^2 - 4n_4 n_3 yz - 4n_4 n_5 y^2 - 4n_4 n_6 z^2 + 4n_4}}{2n_4} \end{aligned} \tag{4.7}$$

Für diese Gleichungen sind nun die Extremwerte zu bestimmen. Dazu werden die Ableitungen $\frac{\mathrm{d}x_{\pm}}{\mathrm{d}y}$ und $\frac{\mathrm{d}x_{\pm}}{\mathrm{d}z}$ bestimmt und gleich Null gesetzt. Die Lösungen dieses Gleichungssystems sind dann die gesuchten Maximalwerte.
Mit

$$\Omega = {n_1}^2 y^2 + 2n_1 y n_2 z + {n_2}^2 z^2 - 4n_4 n_3 yz - 4n_4 n_5 y^2 - 4n_4 n_6 z^2 + 4n_4$$

ergibt sich

$$\begin{aligned} \frac{\mathrm{d}x_{\pm}}{\mathrm{d}y} &= \mp \frac{\pm n_1 \sqrt{\Omega} - {n_1}^2 y - n_1 n_2 z + 2n_4 n_3 z + 4n_4 n_5 y}{2n_4 \sqrt{\Omega}} = 0 \\ \frac{\mathrm{d}x_{\pm}}{\mathrm{d}z} &= \mp \frac{\pm n_2 \sqrt{\Omega} - n_1 y n_2 - {n_2}^2 z + 2n_4 n_3 y + 4n_4 n_6 z}{2\, n_4 \sqrt{\Omega}} = 0. \end{aligned} \tag{4.8}$$

Die Lösungen

$$\begin{aligned} y_1 &= (2n_1 n_6 - n_2 n_3)\, \Gamma \\ z_1 &= -(n_1 n_3 - 2n_2 n_5)\, \Gamma \end{aligned} \tag{4.9}$$

und

$$\begin{aligned} y_2 &= -(2n_1 n_6 - n_2 n_3)\, \Gamma \\ z_2 &= (n_1 n_3 - 2n_2 n_5)\, \Gamma \end{aligned} \tag{4.10}$$

dieser zwei Gleichungssysteme 4.8, wobei

$$\Theta = {n_1}^2 n_6 - n_1 n_2 n_3 + n_5 {n_2}^2 + n_4 {n_3}^2 - 4n_4 n_6 n_5 \quad \text{und} \quad \Gamma = \frac{\sqrt{\frac{\Theta}{{n_3}^2 - 4n_6 n_5}}}{2\Theta}$$

ist, werden verwendet, um durch Einsetzen in Gleichung 4.7 die Maximalwerte

$$\begin{aligned} x_1 &= \left({n_3}^2 + 4n_6 n_5\right) \Gamma \\ x_2 &= -x_1 \end{aligned} \tag{4.11}$$

für x zu bestimmen.

Jetzt muß noch berücksichtigt werden, daß das raumfeste Koordinatensystem **K** und das körperfeste $\overline{\mathbf{K}}$ um den Vektor (x_0, y_0, z_0) verschoben sind. Damit sind die Grenzen der Bounding–Box

$$\begin{aligned} b^{(x)} &= x_0 - |x_1| & e^{(x)} &= x_0 + |x_1| \\ b^{(y)} &= y_0 - |y_1| & e^{(y)} &= y_0 + |y_1| \\ b^{(z)} &= y_0 - |z_1| & e^{(z)} &= z_0 + |z_1| \end{aligned} \tag{4.12}$$

.

Somit lassen sich die Berechnungen für eine Bounding–Box um einen Körper mit K Ecken in konstanter Zeit ausführen, die Berechnung bei allen N geht demnach mit $\mathcal{O}(c)$ mit $c = const.$

4.1.3.3 Vergleich

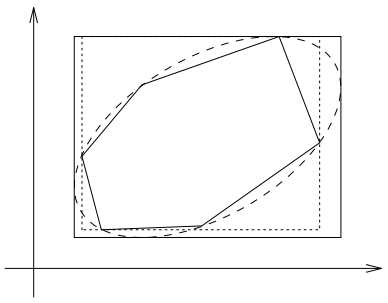

Abbildung 4.3: Die Ecken des Polygons liegen auf dem Rand eines Ellipsoiden. Die minimale Bounding–Box ist gestrichelt eingetragen. Die aus der Ellipse berechnete Bounding–Box wir mit einer durchgezogenen Linie dargestellt, sie ist am rechten Rand deutlich zu groß.

Es stellt sich nun die Frage, welche der beiden Möglichkeiten zur Bestimmung der Bounding–Boxes schneller ist. Die Bestimmung über die Min-Max-Methode hat den Vorteil, daß sie die kleinstmöglichen Intervalle b und e findet, mit zunehmender Komplexität des einzuschließenden Körpers wird der Zeitbedarf aber immer größer.
Die Bestimmung über das Ellipsoid ist hingegen von der Form des eingeschlossenen Körpers unabhängig und wird in konstanter Zeit berechnet. Dies hat gleichzeitig den Nachteil, daß die Bounding–Box im allgemeinen[4] zu groß bestimmt wird (sieh e Abbildung 4.3). Somit nimmt die Zahl der zusätzlichen Kollisionsprüfungen zu.
Um beide Methoden vergleichen zu können, wurden die einzelnen Laufzeiten bei verschiedenen Eckenzahlen bestimmt. Außerdem wurde das Größenverhältnis der

[4]Nur wenn zufällig je ein Eckpunkt auf dem Berührungspunkten Ellipsoid–Quader liegt, dann ist die Größe minimal (dies ist z.B an der Oberkante in Abbildung 4.3 der Fall).

Ecken	$\frac{V_{\text{Ellipsoid}}}{V_{\text{Min-Max}}}$	$\sigma\left(\frac{V_{\text{ellipsoid}}}{V_{\text{Min-Max}}}\right)$	Ellipsoid μs	gewichtet μs	Min-Max μs
4	3.1	0.344	1.7	149	2.65
6	1.28	0.28	1.9	21.5	2.92
12	1.3	0.14	1.7	22.7	4.36
18	1.13	0.105	1.6	10.7	5.70
22	1.16	0.078	2.2	13.4	6.80
42	1.09	0.038	2.6	8.86	11.6
52	1.08	0.039	2.6	8.18	13.7
62	1.058	0.037	1.9	6.00	16.6
72	1.061	0.0307	2.3	6.61	18.9
202	1.045	0.0294	2.2	5.35	48.3
302	1.043	0.029	3.6	6.61	73.3
462	1.02	0.0154	3.1	4.50	109

Tabelle 4.3: Vergleich der benötigten Rechenzeit für die Berechnung einer Bounding–Box mit der Min-Max Methode und der Ellipsoid-Methode. Die gewichtete Rechenzeit wurde über $\left(\frac{V_{Ellipsoid}}{V_{Min-Max}} - 1\right) * 70\mu s$ berechnet.

Bounding–Boxes bestimmt. Dabei mußte berücksichtigt werden, daß dieses Verhältnis für eine feste Eckenzahl nicht konstant ist, sondern von der aktuellen Lage des umschlossenen Körpers beeinflußt wird, es wurde also auch die Standardabweichung bestimmt. Dann nimmt man an, daß pro Schritt $\frac{V_{\text{Ellipsoid}}}{V_{\text{Min-Max}}} - 1$ zusätzliche Kollisionprüfungen notwendig sind[5]. Für die einzelne Kollisionsprüfung wurde nach Tabelle 4.4 ein Zeit von $70\mu s$ angenommen. Die einzelnen Verhältnisse und Zeiten sind in Tabelle 4.3 aufgeführt, die Zeiten sind zusätzlich im Graphen 4.4 aufgetragen. Man sieht, daß der Zeitbedarf für die Min-Max-Methode linear ansteigt, während die reine Rechenzeit für die Ellipsoid-Methode konstant ist. Wenn allerdings die zusätzliche Rechenzeit für Kollisionsprüfungen eingerechnet wird, sieht man, daß es zwei Fälle gibt. Bei sehr kleiner Eckenzahl ist die Min-Max-Methode deutlich schneller, da bei der Alternative sehr viele zusätzliche Kollisionsbestimmungen notwendig werden. Wird die Eckenzahl aber sehr groß, und nähert sich die Form des Körpers damit dem Ellipsoid an, so geht das Größenverhältnis der Bounding–Boxes gegen 1. Man hat fast nur noch die Rechenzeit der Ellipsoid-Methode, diese liegt deutlich unter dem entsprechend vieler Vergleiche. Der Schnittpunkt beider Kurven ist bei ca. 50 Ecken, in diesem Bereich muß dann in der Simulation ausgetestet werden, welche der beiden Methoden das bessere Zeitverhalten liefert. Es wurden beide Methoden in das Programm eingebaut, die Auswahl erfolgt über einen Befehl `BBOX` in der Parameterdatei. (sieh e hierzu Anhang D)

[5]In der Simulation dürfte der wirkliche zusätzliche Zeitbedarf geringer sein, da immer einzelne Partikel keine Nachbarn haben, z.B. wenn sie gerade frei fallen. Liegt der Sandhaufen schon ziemlich dicht, wird man fast immer eine Kollision feststellen, auch hier ist dann die Größe der Bounding–Box nicht mehr ausschlaggebend.

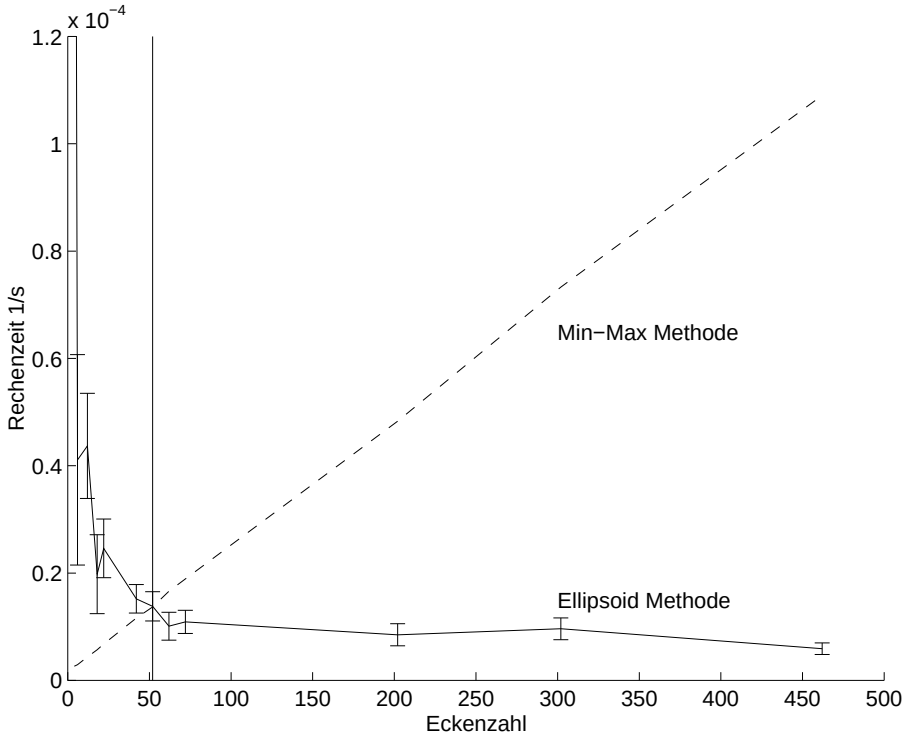

Abbildung 4.4: Vergleich der Rechenzeit für die Bounding–Box Berechnung nach der Ellipsoid-Methode und der Min-Max-Methode. Der Schnittpunkt beider Kurven liegt bei 52 Ecken. Links der Trennlinie ist die Min-Max Methode besser, rechts davon die Ellipsoid-Methode.

4.2 closest-feature-Algorithmus

In diesem Kapitel soll ein schneller Algorithmus [Lin93] vorgestellt werden, mit dem es möglich ist, den Abstand zweier Polyeder zu berechnen. Gleichzeitig werden auch die beiden einander am nächsten liegenden Bestandteile bestimmt.

Der Vorteil dieser Methode besteht darin, daß die Berechnung unabhängig von der Komplexität der beteiligten Körper ist, sie arbeiten mit $\mathcal{O}(\text{const})$. Die Grundidee ist es, das Wissen aus dem letzten Zeitschritt der Simulation zu verwenden und damit einen sehr guten Ausgangspunkt für die neuerliche Rechnung zu haben.

Es wird im nächsten Abschnitt die Grundidee für die Bestimmung der nächsten Punkte dargestellt. Dann wird der Begriff Voronoizelle erläutert und deren Berechnung vorgestellt. Danach müssen einige geometrische Grundlagen diskutiert werden, um dann auf die Implementierung eingehen zu können. Abschließend wird dann noch das Laufzeitverhalten untersucht und mit dem der „einfachen“ Algorithmen verglichen.

4.2.1 „Find and Track“ für die nächsten Punkte

Das Konzept des Algorithmus ist recht einfach: um den Abstand zweier Polyeder d_{AB} zu bestimmen, beginnt man mit einem beliebigen Paar $f_A \in A$ und $f_B \in B$ und prüft, ob die nächsten Punkte auf f_A und f_B liegen. Da A und B konvexe Polyeder sind, ist für diesen Test nur noch das Wissen über die an f_A und f_B grenzenden Elemente notwendig. Das Ganze ist also ein lokaler Test, der von der Komplexität von A und B *nicht* abhängt. Der Test arbeitet in zwei Schritten. Zuerst werden zwei Punkte auf f_A bzw. f_B bestimmt, die den kürzesten Abstand zwischen diesen beiden Elementen haben. Dann wird geprüft, ob der Punkt auf f_A in der Voronoiregion von f_B liegt und umgekehrt. Ist dies der Fall, so entsprechen diese zwei Punkte der Bedingung 4.16. Ist dies nicht der Fall, so wird f_A oder f_B durch einen seiner Nachbarn ersetzt und alles wiederholt. Bei dieser Auswahl ist garantiert, daß die neu gewählten f_A und f_B näher zusammenliegen als die vorhergehenden.
Da sich die Lage der Polyeder bei einer vernünftigen physikalischen Simulation nicht schlagartig ändert, ist meistens der Algorithmus schon nach einer Iteration beendet. Ist dies nicht der Fall, sind selten mehr als zwei Iterationen notwendig. Ist der Abstand 0, so berühren sich f_A und f_B und man hat eine Ausgangsbasis für die Bestimmung des konvexen Schnittpolyeders.

4.2.1.1 Terminologie und Definitionen

In diesem Anschnitt sollen einige Begriffe erläutert werden, die zum Verständnis des closest-feature-Algorithmus und dessen Implementierung hilfreich sind.
Sei A ein Polyeder. Dann besteht A aus verschiedenen Bestandteilen f_1,...,f_n, wobei n die Gesamtzahl aller Bestandteile ist, die Ecken, Kanten und Flächen als Bestandteile bezeichnet werden und f,k,e deren Anzahl ist. Somit ist $n = f + k + e$. Dabei gilt:

$$\begin{aligned} \cup f_i = A, i = 1, \ldots, n \\ f_i \cap f_j = \varnothing, i \neq j \end{aligned} \tag{4.13}$$

Für die einzelnen Kanten wird die „winged-edge“ Darstellung, die in Abbildung 4.5 gezeigt wird, verwendet. Dabei wird der Anfangspunkt (*Tail*) und Endpunkt (*Head*) verwendet, die Strecke selbst zeigt von T_E zu H_E. Die zwei Flächen, die Coboundaries sind, werden nach ihrer Lage beim Blick von Tail in Richtung Head als „linke“ und „rechte Fläche“ bezeichnet.
Flächen werden durch die Gleichung

$$ax + by + cz + d = 0 \tag{4.14}$$

dargestellt. Dabei hat der Vektor $n = (a, b, c)$ die Länge eins und $-d$ ist der Abstand der Ebene zum Ursprung des Koordinatensystems.
Dasjenige Paar von Elementen ist das am nächsten liegende, welches die zwei sich am nächsten liegenden Punkte enthält. Wenn A und B zwei konvexe, kompakte Polyeder mit dem euklidischen Abstand

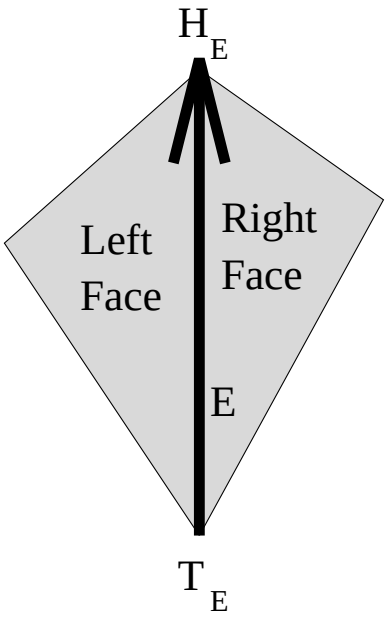

Abbildung 4.5: „winged edge“ Darstellung einer Kante

$$d_{AB} = \inf_{p \in A, q \in B} |p - q| \tag{4.15}$$

und $P_A \in A$,$P_B \in B$

$$d_{AB} = |P_A - P_B| \tag{4.16}$$

sind, dann sind P_A und P_B die sich am nächsten liegende Punkte von A und B.

4.2.2 Voronoizellen

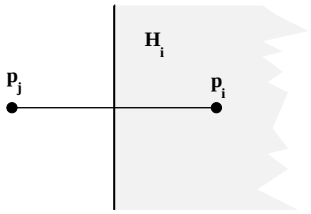

Abbildung 4.6: Halbebene H_i als Voronoizelle

Voronoiregionen bilden einen zentralen Punkt beim closest-feature-Algorithmus . Deswegen wird zuerst auf die Voronoidiagramme im Allgemeinen eingegangen, um dieses Konzept dann von Punkten auf Polyederbestandteile und vom $\mathbb{R}^2$ auf den $\mathbb{R}^3$ auszuweiten. Das Problem „Gegeben ist eine Menge S von N Punkten im $\mathbb{R}^2$; welches ist die Menge aller Punkte (x, y) in Bezug auf $p_i \in S$, die näher an p_i als als an jedem anderen Punkt $p_{j\neq i}$ liegen? " wird durch die Konstruktion eines Voronoidiagramms gelöst.

Dabei wird die Ebene in einzelne Regionen aufgeteilt, wobei jede Region V_i zum Punkt p_i gehört und alle Punkte umfaßt, die am nächsten zu p_i liegen. Wie ist nun diese Region zu konstruieren?

Gegeben seien zwei Punkte p_i und p_j. Offensichtlich ist die Menge aller Punkte die in V_i liegen, die Halbebene $H_i(p_j, p_j)$ mit $p_i \in H_i(p_j, p_j)$. $H_i(p_j, p_j)$ ist begrenzt durch die Mittelsenkrechte auf die Strecke $\overline{p_j p_j}$ (sieh e Abbildung 4.6) Bei mehr als zwei Punkten ist V_i der Durchschnitt von $N-$ Halbebenen $H_{j\neq i}$. Wird dies für alle Punkte p_{1-N} ausgeführt, erhält man ein nichtreguläres Gitter, das Voronoidiagramm. Jeder Punkt der Ebene liegt innerhalb mindestens eines Voronoipolygons. Wenn also ein Punkt $(x, y) \in V_i$ liegt, ist p_i der am nächsten liegende Punkt in Bezug auf (x, y). Gilt für einen Punkt $(x, y) \in V_i$ und $(x, y) \in V_j$ mit $i \neq j$, dann liegt (x, y) auf der Grenze und hat zu p_i und p_j den gleichen Abstand. Wird dieses Konzept nun auf den $\mathbb{R}^3$ erweitert, so sind die Voronoiregionen die Schnitte von Halbräumen, also konvexe (evtl. an einer Seite offene) Polyeder. Für den closest-feature-Algorithmus muß dieses Konzept noch einmal erweitert werden, und zwar in Bezug auf die Punkte p_i. Diese werden durch Punkte (Ecken des eines Polyeders), Strecken (Kanten eines Polyeders) und Polygone (Flächen eines Polyeders) ersetzt. Die dann zu findenden Voronoiregionen zu einem Polyederbestandteil sind also eine Punktmenge *außerhalb* des Polyeders, die näher an diesem Bestandteil liegt als zu allen anderen. Die Voronoiregionen formen also Teilräume um das Polyeder und sind im Falle konvexer Polyeder durch Ebenen begrenzt. In den nächsten drei Abschnitten soll nun erklärt werden, wie die Voronoiregionen zu einer Polygonecke, -kante und -fläche berechnet werden.

4.2.2.1 Voronoiregion einer Ecke und Akzeptanzkriterium

Gegeben sei die Ecke $\mathcal{E}$ eines Polyeders. In diesem Punkt treffen dann N Kanten $\mathcal{K}_{1,N}$ zusammen. Die Voronoizelle $V_\mathcal{E}$ wird dann durch N Ebenen $E_{1,N}$ begrenzt, für die gilt:

$$\mathcal{E} \in E_i \quad \text{und} \quad \mathcal{K}_i \perp E_i \quad \text{für alle} \quad i = 1, \ldots, N \tag{4.17}$$

Die Voronoiregion hat also die Form einer unten offenen, unendlich hohen Pyramide (sieh e auch Abbildung 4.7).

Wenn $\mathbf{u}$ ein Vektor der Länge 1 in Richtung der Kante $\mathcal{K}_i$ ist, und $\mathcal{E}$ die Koordinaten

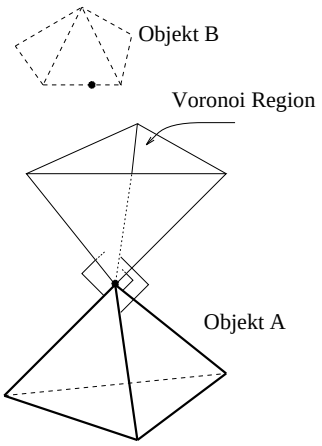

Abbildung 4.7: Voronoiregion einer Ecke auf einem konvexen Polyeder

$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$ hat, dann ist Ebene E_i bestimmt durch:

$$\pm \left(\mathbf{u} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} - \mathbf{u} \cdot \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \right) = 0 \tag{4.18}$$

Welches Vorzeichen wirklich gültig ist, wird *vor* Start der Simulation getestet, da die Gleichungen der Ebenen so bestimmt werden müssen , daß ein Punkt, der in der Voronoiebene liegt, oberhalb aller Ebenen $E_{1,N}$ liegt. Somit wird während der Simulation die Gleichung nur noch mit $+1$ oder -1 multipliziert. Liegt ein Punkt P unterhalb einer der Ebenen E_k, so ist der Abstand $\overline{\mathcal{K}_k P}$ kleiner als $\overline{\mathcal{E} P}$. Somit wird $\mathcal{E}$ durch $\mathcal{K}_k$ ersetzt, und der closest-feature-Algorithmus wird noch einmal aufgerufen.

4.2.2.2 Voronoiregion einer Kante und Akzeptanzkriterium

Gegeben ist nun die Kante $\mathcal{K}$. Sie wird begrenzt durch zwei Ecken $\mathcal{E}_{Tail}$ und $\mathcal{E}_{Head}$, und zwei Flächen $\mathcal{F}_{right}$ und $\mathcal{F}_{left}$. (Definitionen nach Abschnitt 4.2.1.1). Es sind also vier Gleichungen zu bestimmen, die ein Prisma wie in Abbildung 4.8 ergeben.
Die Gleichungen derjenigen zwei Ebenen, die nur durch $\mathcal{E}_{Tail}$ oder $\mathcal{E}_{Head}$ laufen, werden analog zum Abschnitt 4.2.2.1 berechnet. Liegt der zu prüfende Punkt P unterhalb einer der Ebenen, so wird im nächsten Durchlauf $\mathcal{E}_{Tail}$ bzw. $\mathcal{E}_{Head}$ statt $\mathcal{K}$ verwendet. Die beiden anderen Ebenen müssen nun so berechnet werden, daß die Kante in beiden Ebenen liegt und die einzelne Ebene senkrecht zu der jeweiligen angrenzenden Fläche steht. Sei $\mathbf{u}$ wieder der Einheitvektor entlang $\mathcal{K}$ und $\mathbf{n}$ der Normalenvektor auf der angrenzenden Fläche. Außerdem habe $\mathcal{E}_{Tail}$ die Koordinaten $\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$. Dann lautet

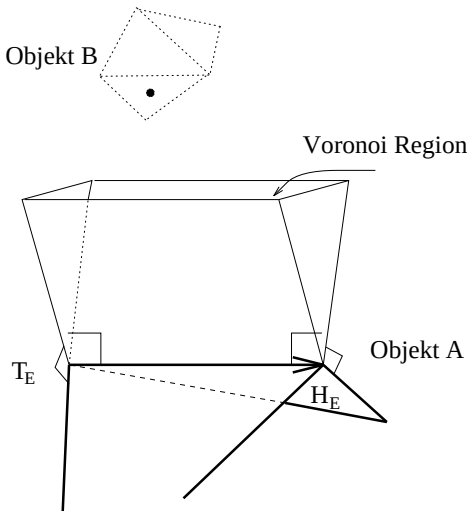

Abbildung 4.8: Voronoiregion einer Ecke auf einem konvexen Polyeder

die Gleichung für die Ebene E:

$$\pm\left((\mathbf{n}\times\mathbf{u})\cdot\begin{pmatrix}x\\y\\z\end{pmatrix}-(\mathbf{n}\times\mathbf{u})\cdot\begin{pmatrix}p_1\\p_2\\p_3\end{pmatrix}\right)=0 \tag{4.19}$$

Auch hier wird analog zu den Ecken das Vorzeichen für alle 4 Voronoiebenen vorher bestimmt.
Schlägt der Test für P und E_{right} fehl, wird der Algorithmus mit $\mathcal{F}_{rigth}$ fortgesetzt, entsprechendes für E_{left} und $\mathcal{F}_{left}$

4.2.2.3 Voronoizelle einer Fläche

Bei der Voronoiregion einer Fläche $\mathcal{F}$ ist zu berücksichtigen, daß die Flächen der in der Simulation verwendeten Polyeder Dreiecke sind. Somit entfällt das von M.C. Lin vorgeschlage Unterteilen komplexer Flächen in Dreiecke oder Vierecke. Es sind somit nur die Gleichungen von 3 Ebenen zu bestimmen, die senkrecht zur Fläche stehen, und entlang einer der drei Kanten $\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3$ verlaufen. Die Rechnung selbst verläuft analog zur der in Formel 4.19. Bei einer Ebene ist allerdings zu beachten, daß die Voronoiregion nur außerhalb des Polyeders verläuft. Somit muß noch die Gleichung der Fläche $\mathcal{F}$ berechnet und dann geprüft werden, ob P oberhalb dieser Ebene liegt. Ist dies nicht der Fall, so liegt entweder eines der Elemente des zweiten Polyeders näher an $\mathcal{F}$ oder aber es handelt sich um eine Kollision. In diesem Fall können aber $\mathcal{F}$ und $\mathcal{P}$ ein lokales Minimum des Abstands haben und ein Wechsel zu einem Nachbarelement würde den Abstand wieder vergrößern. Deswegen muß eine Routine aufgerufen werden, die nach einem kürzeren Abstand sucht. Danach kann mit dem allgemeinen Algorithmus fortgefahren werden.

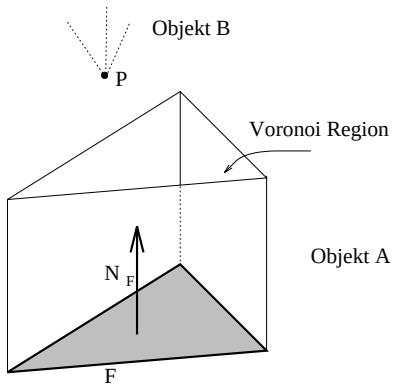

Abbildung 4.9: Voronoiregion einer Fläche auf einem konvexen Polyeder

4.2.2.4 Prüfung zweier Ecken

Dies ist der einfachste Fall. Seien die Ecken $\mathcal{E}_1$ und $\mathcal{E}_2$. Die zwei zu untersuchenden Punkte $p_1 \in \mathcal{E}_1$ und $p_2 \in \mathcal{E}_3$ sind die Ecken selbst und die Vorgehenweise ist dann folgende:

1. Berechnung der Voronoizelle V_2
2. Prüfung, ob $p_1 \in V_2$. Wenn nicht: neues p_1 wählen und Algorithmus erneut abarbeiten.
3. Berechnung der Voronoizelle V_1
4. Prüfung, ob $p_2 \in V_1$. Wenn nicht: neues p_2 wählen und Algorithmus erneut abarbeiten.
5. $\mathcal{E}_1$ und $\mathcal{E}_2$ erfüllen die Bedingung4.16
6. Abstand p_1, p_2 berechnen

4.2.2.5 Prüfung einer Ecke und einer Kante

Sind eine Ecke $\mathcal{E}_1$ und eine Kante $\mathcal{K}_1$ gegeben, müssen die zu untersuchenden Punkte $p_1 \in \mathcal{E}_1$ und $p_2 \in \mathcal{K}_1$ bestimmt werden. Dabei ist $p_1 = \mathcal{E}_1$ und $\mathbf{p}_2$ wird wie folgt berechnet, wenn die Kante von $T_E = (T_x, T_y, T_z)$ nach $H_E = (H_x, H_y, H_z)$ geht. Dann ist der gesuchte Punkt p_2

$$p_2 = T_E + \min(1, \max(0, \frac{(p_1 - T_e) \cdot \mathbf{e}}{|\mathbf{e}|^2})) \tag{4.20}$$

wobei $\mathbf{e} = H_E - T_E$ ist. Dabei wird mit dem Term $\lambda = \frac{(p_1 - T_e) \cdot \mathbf{e}}{|\mathbf{e}|^2}$ die Projektion von p_1 auf eine Gerade durch T_E und H_E berechnet. Der entsprechende Punkt auf der Strecke

erfolgt durch Reduktion von λ in das Intervall $[0,1]$. Dann kann eine Prüfung wie in Abschnitt 4.2.2.4 durchgeführt werden.

4.2.2.6 Nächste Punkte einer Ecke und einer Fläche

Sind eine Ecke $\mathcal{E}_1$ und eine Fläche $\mathcal{F}_2$ gegeben, muß ebenso wie in Abschnitt 4.2.2.5 nur $p_s \in \mathcal{F}_2$ explizit berechnet werden, da p_1 wieder mit $\mathcal{E}_1$ identisch ist. Sei $V = (p_{1_x}, p_{1_y}, p_{1_z}, 1)$ und die Gleichung der Ebene $ax + by + cz + d = 0$, wobei $\mathbf{n} = (a,b,c)$ ein Einheitsvektor ist und d der Abstand der Ebene zum Ursprung. Dann ist mit $N_f = (\mathbf{n}, -d) = (a,b,c,-d)$ und $\mathbf{n}_f = (\mathbf{n}, 0)$ der Punkt p_2 gegeben durch

$$p_2 = V - (V \cdot N_f)\mathbf{n}_f. \tag{4.21}$$

Danach wird wieder wie in Abschnitt 4.2.2.4 verfahren.

4.2.2.7 Nächste Punkte zweier Kanten

Sind zwei Kanten $\mathcal{K}_1$ und $\mathcal{K}_2$ mit Head H_1 und T_1 bzw. H_2 und T_2 gegeben, dann müssen die beiden Fälle $\mathcal{K}_1 \| \mathcal{K}_2$ und $\mathcal{K}_1 \nparallel \mathcal{K}_2$ unterscheiden werden. Ist $\mathcal{K}_1 \nparallel \mathcal{K}_2$, dann wird $\mathbf{e}_1$ als $\mathbf{e}_1 = H_1 - T_2$ und $\mathbf{e}_2$ als $\mathbf{e}_2 = H_2 - T_2$ definiert. Die gesuchten Punkte $\mathbf{p}_1$ auf $\mathbf{e}_1$ und $\mathbf{p}_2$ auf $\mathbf{e}_2$ sind dann gegeben durch

$$\begin{aligned} p_1 &= H_1 + s(T_1 - H_1) = H_1 - \mathbf{e}_1 \\ p_2 &= H_2 + u(T_2 - H_2) = H_2 - \mathbf{e}_2, \end{aligned} \tag{4.22}$$

wobei s und u Parameter zwischen 0 und 1 sind, und die relative Lage auf den Strecken angeben. Wenn $\mathbf{n} = p_1 - p_2$ ist, dann muß $|\mathbf{n}|$ der minimale Abstand zwischen den Kanten sein. Somit muß $\mathbf{n}$ orthogonal zu $\mathbf{e}_1$ und $\mathbf{e}_2$ sein, wir erhalten also

$$\begin{aligned} \mathbf{n} \cdot \mathbf{e}_1 &= (p_1 - p_2) \cdot \mathbf{e}_1 = 0 \\ \mathbf{n} \cdot \mathbf{e}_2 &= (p_1 - p_2) \cdot \mathbf{e}_2 = 0. \end{aligned} \tag{4.23}$$

Durch Einsetzen von 4.22 in 4.23 und Auflösen ergibt sich für s und u:

$$\begin{aligned} s &= \frac{(\mathbf{e}_1 \cdot \mathbf{e}_2)\,[(H_1 - H_2) \cdot \mathbf{e}_2] - (\mathbf{e}_2 \cdot \mathbf{e}_2)\,[(H_1 - H_2) \cdot \mathbf{e}_1]}{((\mathbf{e}_1 \cdot \mathbf{e}_2)(\mathbf{e}_1 \cdot \mathbf{e}_2)) - ((\mathbf{e}_1 \cdot \mathbf{e}_1)(\mathbf{e}_2 \cdot \mathbf{e}_2))} \\ u &= \frac{(\mathbf{e}_1 \cdot \mathbf{e}_1)\,[(H_1 - H_2) \cdot \mathbf{e}_2] - (\mathbf{e}_1 \cdot \mathbf{e}_2)\,[(H_1 - H_2) \cdot \mathbf{e}_1]}{((\mathbf{e}_1 \cdot \mathbf{e}_2)(\mathbf{e}_1 \cdot \mathbf{e}_2)) - ((\mathbf{e}_1 \cdot \mathbf{e}_1)(\mathbf{e}_2 \cdot \mathbf{e}_2))} \end{aligned} \tag{4.24}$$

Auch in diesem Fall muß sichergestellt sein, daß $s, u \in [0,1]$. Es darf hier allerdings nicht, wie M.C.Lin in [Lin93] vorschlägt, s und u über

$$\begin{aligned} s' &= \min(1, \max(0, s) \\ u' &= \min(1, \max(0, u) \end{aligned} \tag{4.25}$$

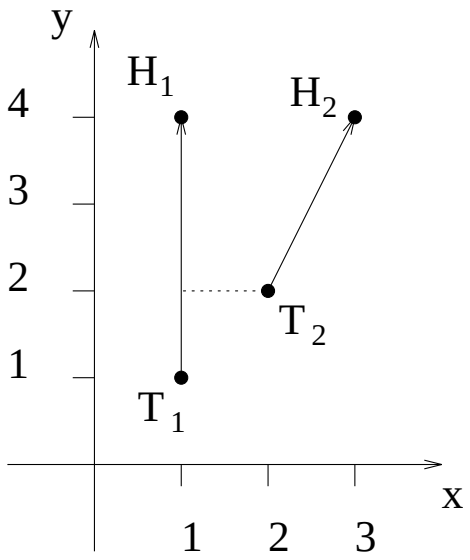

Abbildung 4.10: In diesem Fall versagt die Berechnung der nächsten Punkte zweier Kanten nach M.C.Lin. Es sind zusätzliche Fallunterscheidungen nötig.

abgeschnitten werden.
Hat man zum Beispiel $H_1 = \begin{pmatrix} 1 \\ 4 \\ 0 \end{pmatrix}, T_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$, $H_2 = \begin{pmatrix} 3 \\ 4 \\ 0 \end{pmatrix}, T_2 = \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix}$. Somit ist nach 4.24 $s = \frac{4}{3}$ und $u = 2$ und nach Formel 4.25 folgt dann $s = 1$ und $u = 1$. Damit wären T_1 und T_2 die Punkte mit dem kürzesten Abstand. Da dies offensichtlich falsch ist, müssen für den Fall, daß entweder $s \notin [0,1]$ oder $u \notin [0,1]$ weitere Überprüfungen erfolgen. Dabei muß dann für $s > 1$ ($s < 1$) der Abstand von T_1 (H_1) zur anderen Strecke bestimmt werden. Analog wird mit u verfahren und der kürzeste Abstand wird dann verwendet. Damit erhält man das korrekte Ergebnis, falls $\mathcal{K}_1 \not\parallel \mathcal{K}_2$.
Im Fall $\mathcal{K}_1 \| \mathcal{K}_2$ sind grundsätzlich zwei Möglichkeiten zu unterscheiden. Entweder es gibt (sieh e Abbildung 4.11) wie in Fall A unendlich viele Lösungen oder wie in Fall B nur eine. Ist die Lösung nicht eindeutig, werden die Punkte in der Mitte des gemeinsamen Bereiches ausgewählt. Die Berechnung selbst erfolgt durch Projektion der Endpunkte auf die jeweilig andere Strecke und dann durch Fallunterscheidungen.
Sind die Punkte $\mathbf{p}_1$ und $\mathbf{p}_2$ bestimmt, kann analog wie in Abschnitt 4.2.2.4 verfahren werden.

4.2.2.8 Nächste Punkte einer Kante und einer Fläche

Sind eine Ecke $\mathcal{E}_1$ und eine Fläche $\mathcal{F}_2$ zu untersuchen, muß zuerst entschieden werden, ob die Kante und die Fläche parallel sind. Ist dies nicht der Fall, so ist entweder

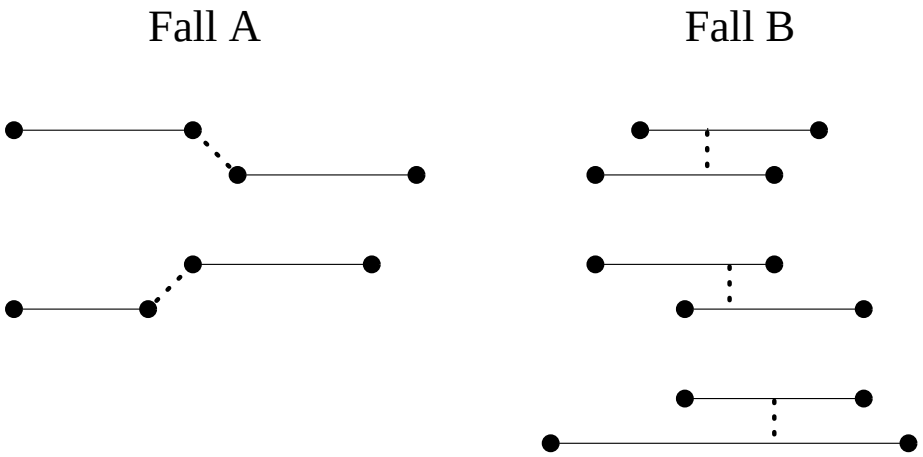

Abbildung 4.11: Dargestellt werden die 5 Fälle bei der Bestimmung der nächsten Punkte zwischen parallelen Flächen. Im Fall A sind die Punkte eindeutig, im Fall B nicht. Hier sollten dann zwei Punkte gewählt werden, die in der Mitte des Überlappbereiches liegen.

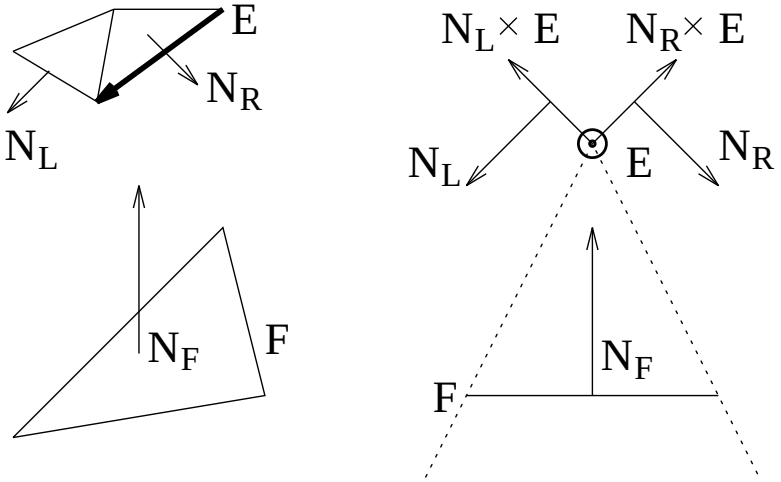

Abbildung 4.12: Darstellung der Lage einer Kante und einer Fläche für die Abstandsberechnung.

einer der Eckpunkte der Strecke und die Fläche das nächste zu untersuchende Paar, oder aber die Kante und eine der Begrenzungskanten des Dreiecks. Der erstgenannte Fall tritt ein, wenn Head oder Tail in der Voronoiregion der Fläche liegen. Ansonsten werden als nächstes die Kante $\mathcal{E}_1$ und die zu dieser am nächsten liegende Kante $\mathcal{E}_\mathcal{F}$ untersucht. Es ist also nicht möglich, daß in diesem Fall $\mathcal{E}_1$ und $\mathcal{F}_2$ die gesuchten

Teile der Polyeder sind. Wenn $\mathcal{E}_1 \| \mathbf{F}_2$ gilt, dann erfüllen sie 4.16, wenn folgende drei Bedingungen zutreffen:

1. Die Kante muß die Voronoiregion der Fläche schneiden.

2. Die Flächennormale von $\mathcal{F}_2$ muß zwischen den Flächennormalen der Begrenzungsflächen von $\mathcal{E}_1$ liegen (sieh e hierzu Abbildung 4.12).

3. Die Kante muß oberhalb von $\mathcal{F}_2$ liegen.

4.2.2.9 Nächste Punkte zweier Flächen

Dies ist der Fall, der am seltensten auftritt. Hierbei muß zuerst geprüft werden, ob $\mathcal{F}_1$ und $\mathcal{F}_2$ parallel sind. Ist dies der Fall, so wird geprüft, ob die Projektion der Fläche $\mathcal{F}_1$ auf die Ebene, die $\mathcal{F}_2$ enthält, mit $\mathcal{F}_2$ überlappt. Ist dies der Fall, und liegt jede Ebene über der anderen, so sind $\mathcal{F}_1$ und $\mathcal{F}_2$ das gesuchte Paar, sie haben den kleinsten Abstand. Sind hingegen $\mathcal{F}_1$ und $\mathcal{F}_2$ nicht parallel, sucht man zuerst das Element $f_1 \in \mathcal{F}_1$, das den kleinsten Abstand zur Ebene $\mathcal{F}_2$ hat und umgekehrt. Wenn nun f_1 und $\mathcal{F}_2$ das entsprechende Akzeptanzkriterium erfüllen, werden diese als nächstes getestet, das gleiche erfolgt mit f_2 und $\mathcal{F}_1$. Ansonsten werden alle 9 möglichen Eckenpaaren überprüft und es wird mit demjenigen weitergearbeitet, das den kürzesten Abstand hat.
Stellt sich heraus, daß eine Ebene vollständig unter der anderen liegt, muß ein „völlig neues“ Paar f_1,f_2 gesucht werden, da ansonsten $\mathcal{F}_1$ und $\mathcal{F}_2$ ein lokales Minimum des Abstands bilden können. (Eine genaue Beschreibung aller möglichen Sonderfälle findet sich im Anhang C)

4.2.3 Preprocessing und Datenstrukturen

Der entscheidende Faktor im Hinblick auf die Rechengeschwindigkeit des closest-feature-Algorithmus ist eine sorgfältige Implementierung der Datenstrukturen. Jede für eine Berechnung nötige Information sollte umgehend zur Verfügung stehen. Andererseits sollte keine Berechnung mehrfach erfolgen oder nicht benötigt werden. Es ist also sinnvoll, drei verschiedenen Kategorien an Berechnungen zu unterscheiden:

1. Informationen, die während der ganzen Simulation erhalten bleiben, so z.B. das Netzwerk der einzelnen Polyeder und die logische Struktur der Voronoizellen.

2. Informationen, die bei jedem Zeitschritt vollständig und neu berechnet werden müssen. Dies sind vor allem die Koordinaten der Ecken im raumfesten Koordinatensystem $\mathbf{K}$.

3. Informationen, die nur benötigt werden, um $f_1 \in A$ und $f_2 \in B$ zu überprüfen, z.B. die Gleichungen der Voronoizellen, Flächennormalen und anderes.

4.2.3.1 Datenstrukturen, Implementation und Laufzeitverhalten

Die Datenstruktur ist das Wichtigste an der Implementierung eines Algorithmus. Ein falsch gewählter Aufbau kann die Rechenzeit erheblich vergrößern und die Übersichtlichkeit der Implementierung reduzieren[6]. Deswegen sind folgende Punkte zu beachten:

- Die Koordinaten der Ecken im raumfesten Koordinatensystem **K** müssen zu jedem Zeitschritt neu berechnet werden. Obwohl diese Koordinaten auch für Berechnungen im Zusammenhang mit Ecken, Kanten und Flächen benötigt werden, sollen die Daten nur an einer Stelle gespeichert sein. Somit ist nur ein Speicherzugriff nötig. Dies wird dadurch realisiert, daß die Strukturen für Kanten und Flächen nur Referenzen auf die jeweiligen Eckkoordinaten speichern. Diese sind während der gesamten Simulation unverändert.

- Jedes Element des Polyeders benötigt schnellen Zugriff auf die Information über seine Nachbarn. Auch dies wird durch eine dynamische Verkettung über Zeiger realisiert. Diese Berechnung ist nur zu Beginn der Simulation notwendig und kann dann unverändert belassen werden.

- Die Strukturen der Voronoizellen müssen Zugriff auf diejenigen Elemente haben, mit deren Hilfe die entsprechende Ebenengleichung berechnet werden kann. Außerdem ist es vorteilhaft, wenn dort auch steht, welches Element als nächstes verwendet werden soll, wenn die Prüfung fehlschlägt.

Um dies alles zu gewährleisten, wird vor Beginn der Simulation für jedes Partikel ein Netz dynamisch angelegt, in dem alle erforderlichen Verknüpfungen realisiert werden. Dies geschieht zum größten Teil über verkettete Listen. Eine Skizze der aufgebauten Strukturen findet sich im Anhang C. Der Aufbau dieser Strukturen ist zwar von der Ordnung $\mathcal{O}(n)$, muß aber nur ein einziges Mal vor der Simulation durchgeführt werden und ist somit für die Gesamtzeit der Simulation unerheblich.

4.2.3.2 Laufzeitverhalten

Zur Bestimmung der durchschnittlichen Rechenzeit für die Berechnung des Abstands zwischen zwei Polyedern wurden verschiedene Geometrien verwendet. Das einfachste Polyeder war ein Tetraeder, das komplizierteste war ein fast kugelförmiges Gebilde mit 462 Ecken, 1380 Kanten und 920 Flächen. Es wurde immer der Abstand zwischen zwei gleichen Körpern berechnet, wobei der eine um die X-Achse rotierte, der andere um die Y-Achse. Bei einer Simulationszeit von 100 Sekunden mit einem Zeitschritt von 0.001 Sekunden wurde der Algorithmus 100000 mal abgearbeitet. Dabei zeigt sich, daß die Rechenzeit im Bereich von 70 μs liegt.
Versucht man, den Abstand zweier Polyeder durch direktes Vergleichen aller Elemente zu bestimmen, so ergibt sich schon im einfachsten Fall, dem des Tetraeders, bereits

[6] Da der Algorithmus ca. 3500 Zeilen lang ist, muß auf Übersichtlichkeit und Kommentare viel Wert gelegt werden.

Ecken	Kanten	Flächen	Ges.	1 Z.	2 Z.	3 Z.	4 Z.	5 Z.	6- Z.	μs
4	6	4	14	98290	1430	259	17	2	1	63
6	12	8	26	99803	196	0	0	1	1	83
12	30	20	62	99497	465	36	0	1	1	71
18	48	32	98	99703	296	0	0	0	1	69
22	60	40	122	99702	222	75	0	0	1	72
42	120	80	242	99164	737	97	1	0	1	76
52	150	100	302	99563	292	144	0	0	1	70
62	180	120	362	99544	430	25	0	0	1	60
72	210	140	422	99486	436	77	0	0	1	63
82	240	160	482	99442	549	8	0	0	1	70
102	300	200	602	99361	630	8	0	0	1	78
202	600	400	1202	98907	1057	35	0	0	1	74
302	900	600	1802	98271	1667	57	4	0	1	78
462	1380	920	2762	97661	2217	112	5	3	2	58

Tabelle 4.4: Zahl der Zyklen, die der closest-feature-Algorithmus braucht, um bei 100000 Berechnungen den minimalen Abstand zu finden.

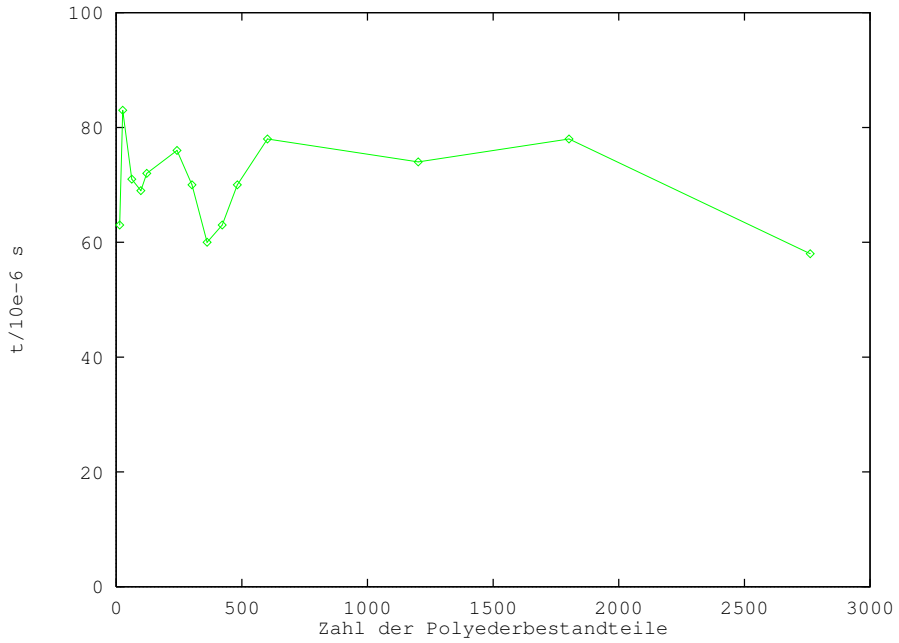

Abbildung 4.13: Laufzeitverhalten des closest-feature-Algorithmus

eine Rechenzeit von ca. 3 ms, bei einem Icosaeder 55 ms. Vielleicht ließe sich dieser Algorithmus durch geschickte Programmierung um den Faktor 2–10 beschleunigen,

im Vergleich zu 70 μs erscheint dies aber unwichtig. Denn beim Tetraeder muß der Abstand $14 \cdot 14 = 196$-mal berechnet werden, bei einem Icosaeder steigt diese Zahl dann auf 3644 Vergleiche und bei dem komplexesten Körper auf $2762^2 = 7628644$ Vergleiche. Es tritt also nicht der Fall ein, daß der Algorithmus mit dem scheinbar schlechteren Zeitverhalten $\mathcal{O}(n^2)$ besser ist, weil sein Zeitbedarf für kleine n im Vergleich zum scheinbar besseren $\mathcal{O}(n)$ oder $\mathcal{O}(const)$ geringer ist (sieh e auch Anhang E.1).
Somit ist es möglich, durch Kombination des closest-feature-Algorithmus mit Bounding–Boxes alle Kollisionen der Sandkörner mit $\mathcal{O}(n + k)$ zu bestimmen, d.h. die Simulationszeit wird linear mit der Zahl der zu simulierenden Sandkörner wachsen. Der closest-feature-Algorithmus liefert im Fall der Kollision einen Startpunkt für die Bestimmung des Overlappolygons. Aus dem Overlappolygon sollen dann die bei der Kollision auftretenden Kräfte und Drehmomente berechnet werden.

Kapitel 5

Zusammenfassung

In dieser Arbeit wurden die Grundlagen von Simulationen für granulare Medien untersucht. Es wurden Verfahren zu Berechnung von Schwerpunkt und Trägheitstensor unregelmäßiger konvexer Polyeder entwickelt. Um numerische Fehler bei der Darstellung von Rotationen zu vermeiden, wurden Quaternionen verwendet, die eine deutlich stabilere und zuverlässigere Simulation erlauben. Als Differentialgleichungslöser wurde der Gear-Predictor-Corrector verwendet, der in Bezug auf Genauigkeit und Energieerhaltung große Vorteile bietet. Außerdem benötigt man nur eine Kraftberechnung pro Zeitschritt. Für das Problem der Kollisionsprüfung wurde eine mehrstufig Lösung entwickelt. Als erster Schritt wurde ein Verfahren für Bounding–Boxes eingeführt, das mit $\mathcal{O}(n+k)$ arbeitet. Für die Bestimmung der Bounding–Boxes wurde ein Verfahren basierend auf einem Ellipsoid entwickelt, das die Berechnung in konstanter Zeit, unabhängig von der Komplexität des eingeschlossenen Polyeders, erlaubt. Für den nächsten Schritt wurde der closest-feature-Algorithmus implementiert und untersucht. Damit ist es möglich, unabhängig von Form und Lage zweier beliebig komplizierter Polyeder, deren Abstand in konstanter Zeit zu bestimmen. Diese Zeit liegt mit ca. 70 μs sogar unter der, die für den direkten Vergleich zweier Tetraeder benötigt wird.

Die Hauptursache für die Probleme bisheriger Simulationen liegen im Bereich der verwendeten Algorithmen, die meistens mit $\mathcal{O}(n^\alpha)$ arbeiten. Eine Simulation, deren Zeitbedarf nur linear mit der Zahl der simulierten Körper ansteigt, scheint möglich. Außerdem spricht nichts gegen die Verwendung von komplexeren Polyedern, man ist nicht auf Kugeln beschränkt.

In Zukunft soll die Kraftberechnung aus dem Overlap zweier Polyeder erfolgen. In zwei Dimensionen konnte dieses Verfahren schon von M. Bauernfeind erfolgreich getestet werden. Dadurch wird dann auch die Haft- und Gleitreibung korrekt modelliert. Dann kann man auch die Einschränkungen beurteilen, die bei bisherigen Simulationen gemacht wurden. Darüberhinaus kann dann auch der Einfluß der Zahl der Dimensionen auf bisherige Simulationen bestimmt werden.

Um die Rechenzeit noch weiter zu verkürzen, soll der Code noch parallelisiert werden [Mat94]. Zukünftig werden sich dann anspruchsvolle Simulationen mit einer großen Zahl komplexer Teilchen verwirklichen lassen. Ein großes Interesse besteht an der Un-

tersuchung von Trichtern. Die Frage ist, ob sich das Verstopfen eines Trichters durch Dichteschwankungen o.ä. ankündigt, und ob es möglich ist, das Verstopfen dann noch zu verhindern. Ebenso soll der Einfluß der Form des Trichters auf das Durchflußverhalten untersucht werden.
Als Fernziel soll eine Simulation mit konkaven Polyedern entwickelt werden und außerdem soll versucht werden, feuchte granulare Medien zu simulieren.

Anhang A

Quaternionen

A.1 Definitionen

Quaternionen können auf verschiedene äquivalente Weisen definiert werden. Ursprünglich wurden sie von Hamilton als erweiterte komplexe Zahlen in der Form $w+\mathbf{i}x+\mathbf{j}y+\mathbf{k}z$, definiert, wobei $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$, $\mathbf{ij} = \mathbf{k} = -\mathbf{ji}$ mit $w, x, y, z \in \mathbb{R}$ gilt. Das eigentlich Neue ist die Nichtkommutativität der Multiplikation.
Es ist inzwischen üblich, die Zahlen x, y, z als Vektor zu bezeichnen, und w als Skalar. Die Verknüpfungen zwischen Quaternionen werden dann auf der Basis von Skalar- und Kreuzprodukt definiert.
Verschiedene Darstellungsmöglichkeiten für Quaternionen:

$$\begin{aligned}\mathbf{q} &\stackrel{def}{=} [w\ ,\ \mathbf{v}] \quad ;\mathbf{v} \in \mathbb{R}^3, w \in \mathbb{R} \\ &= [w\ ,\ (x,y,z)] \quad ;x,y,z,w \in \mathbb{R} \\ &= [w,x,y,z] \quad ;x,y,z,w \in \mathbb{R} \\ &= w + \mathbf{i}x + \mathbf{j}y + \mathbf{k}z; w,x,y,z \in \mathbb{R}, \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1, \mathbf{ij} = \mathbf{k} = -\mathbf{ji}, w,x,y,z \in \mathbb{R}\end{aligned} \tag{A.1}$$

Vektoren $\mathbf{v} \in \mathbb{R}^3$ und Skalare können in der Form

$$\begin{aligned} s &= [s\ ,\ \mathbf{0}] \\ \text{und} \quad \mathbf{v} &= [0\ ,\ \mathbf{v}] \end{aligned} \tag{A.2}$$

geschrieben werden.
Die Addition erfolgt komponentenweise

$$\mathbf{q} + \mathbf{q}\,' = [\mathbf{v}\ ,\ w] + [\mathbf{v}'\ ,\ w'] \stackrel{def}{=} [\mathbf{v} + \mathbf{v}'\ ,\ w + w'] \tag{A.3}$$

Die Multiplikation ist durch

$$\mathbf{q} \cdot \mathbf{q}\,' = [w\ ,\ \mathbf{v}] \cdot [w'\ ,\ \mathbf{v}'] \stackrel{def}{=} [ww' - \mathbf{v} \cdot \mathbf{v}'\ ,\ v \times v' + w\mathbf{v}' + w'\mathbf{v}] \tag{A.4}$$

gegeben,wobei

$$\begin{aligned}(\mathbf{pq})\mathbf{q}' &= \mathbf{p}(\mathbf{qq}') \\ s\mathbf{q} = \mathbf{q}s &= [s\ ,\ \mathbf{0}]\,[w\ ,\ \mathbf{v}] = [sw\ ,\ s\mathbf{v}]\end{aligned} \tag{A.5}$$

und sehr wichtig für Formel 2.22

$$\mathbf{vv}' = [0\ ,\ \mathbf{v}]\,[0\ ,\ \mathbf{v}'] = [-\mathbf{v}\cdot\mathbf{v}'\ ,\ \mathbf{v}\times\mathbf{v}'] \tag{A.6}$$

gilt.
Über das Konjugierte $\mathbf{q}^*$

$$\mathbf{q}^* = [w\ ,\ \mathbf{v}]^* \stackrel{def}{=} [w\ ,\ -\mathbf{v}] \tag{A.7}$$

und die Norm $\mathrm{N}(\mathbf{q})$

$$\mathrm{N} \stackrel{def}{=} \mathbf{qq}^* = \mathbf{q}^*\mathbf{q} = w^2 + \mathbf{v}\cdot\mathbf{v} = w^2 + x^2 + y^2 + z^2 \tag{A.8}$$

mit den Eigenschaften

$$\begin{aligned}(\mathbf{q}^*)^* &= \mathbf{q} \\ (\mathbf{pq})^* &= \mathbf{q}^*\mathbf{p}^* \\ (\mathbf{p}+\mathbf{q})^* &= \mathbf{q}^* + \mathbf{p}^* \\ \mathrm{N}(\mathbf{qq\ '}) &= \mathrm{N}(\mathbf{q})\mathrm{N}(\mathbf{q\ '}) \\ \mathrm{N}(\mathbf{q}^*) &= \mathrm{N}(\mathbf{q})\end{aligned} \tag{A.9}$$

wird das Inverse q^{-1}

$$q^{-1} = \frac{\mathbf{q}^*}{\mathrm{N}(\mathbf{q})} \tag{A.10}$$

definiert.
Quaternionen mit $\mathrm{N}(\mathbf{q}) = 1$ heißen Einheitsquaternionen.

A.2 Quaternionen und Rotationen

Um den Zusammenhang zwischen Quaternionen und Rotationen darzustellen, muß folgendes bewiesen werden.
Sei p ein Punkt im dreidimensionalen Raum, dargestellt durch ein Quaternion; $p = [w\ ,\ \mathbf{v}] = [w\ ,\ (x,y,z)]$. Außerdem sei $\mathbf{q}$ ein Quaternion ungleich Null.Dann gilt:

1. Das Produkt $\mathbf{q}p\mathbf{q}^{-1}$ führt $p = [w\ ,\ \mathbf{v}]$ nach $p' = [w\ ,\ \mathbf{v}']$ über, wobei $\mathrm{N}(\mathbf{v})$ gleich $\mathrm{N}(\mathbf{v}')$ ist

2. Jedes Vielfache $\neq 0$ von $\mathbf{q}$ gibt dasselbe Ergebnis

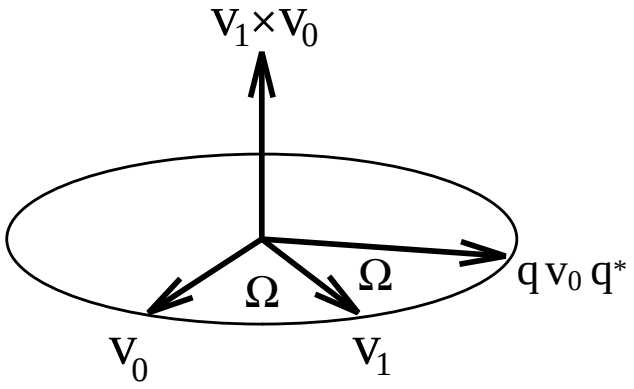

Abbildung A.1: Darstellung der Lage von $\mathbf{v}_0$ und $\mathbf{v}_1$ für den Beweis der Rotation mit Quaternionen.

3. Wenn $\mathrm{N}(\mathbf{q}) = 1$ ist, dann ist $\mathbf{q} = [\cos\Omega\ ,\ \hat{\mathbf{v}}\sin\Omega]$ eine Rotation um $\hat{\mathbf{v}}$ mit dem Winkel 2Ω

Zu 2:
Das Inverse von $s\mathbf{q}$ ist $\mathbf{q}^{-1}s^{-1}$ und die Multiplikation mit einem Skalar kommutiert. Damit gilt

$$s\mathbf{q}p\mathbf{q}^{-1}s^{-1} = \mathbf{q}p\mathbf{q}^{-1}ss^{-1} = \mathbf{q}p\mathbf{q}^{-1} \tag{A.11}$$

Man kann also o.B.d.A annehmen, daß $\mathbf{q}$ ein Einheitsquaternionen ist. Somit kann man statt mit $\mathbf{q}p\mathbf{q}^{-1}$ mit $\mathbf{q}p\mathbf{q}^{*}$ arbeiten.
Zu 1:
Zuerst soll gezeigt werden, daß w durch $\mathbf{q}p\mathbf{q}^{*}$ nicht verändert wird. Dazu wird $2\mathrm{S}(q) = \mathbf{q} + \mathbf{q}^{*}$ definiert, wobei S dazu dient, den skalaren Anteil von $\mathbf{q}$ extrahieren; es ist also $\mathrm{S}(p) = w$. Wegen

$$\begin{aligned} 2\mathrm{S}(\mathbf{q}p\mathbf{q}^{*}) = (\mathbf{q}p\mathbf{q}^{*}) + (\mathbf{q}p\mathbf{q}^{*})^{*} = \mathbf{q}p\mathbf{q}^{*} + \mathbf{q}p^{*}\mathbf{q}^{*} = \\ = \mathbf{q}\,(p + p^{*})\,\mathbf{q}^{*} = \mathbf{q}\,(2\mathrm{S}(p))\,\mathbf{q}^{*} = 2\mathrm{S}(p) \end{aligned} \tag{A.12}$$

wird aus $[w\ ,\ \mathbf{v}]$ durch $\mathbf{q}p\mathbf{q}^{*}$ dann $[w\ ,\ \mathbf{v}']$. Da die Multiplikation keinen Einfluß auf die Norm hat, ist $\mathrm{N}(p) = \mathrm{N}(p')$ und da w unverändert bleibt, ist dann $\mathrm{N}(\mathbf{v}) = \mathrm{N}(\mathbf{v}')$
Zu 3:
Gegeben seien, wie in Abbildung A.1 zu sehen, zwei Vektoren $\mathbf{v}_0$,$\mathbf{v}_1$, es gelte:

$$\begin{aligned} \mathrm{N}(\mathbf{v}_0) = \mathrm{N}(\mathbf{v}_1) &= 1 \\ \cos\Omega &= \mathbf{v}_0 \cdot \mathbf{v}_1 \\ \hat{\mathbf{v}} &= \frac{(\mathbf{v}_0 \times \mathbf{v}_1)}{|\mathbf{v}_0 \times \mathbf{v}_1|} \\ q \overset{def}{=} \mathbf{v}_1\mathbf{v}_0^{*} &= [\mathbf{v}_0 \cdot \mathbf{v}_1\ ,\ \mathbf{v}_0 \times \mathbf{v}_1] = [\cos\Omega\ ,\ \hat{\mathbf{v}}\sin\Omega]\,. \end{aligned} \tag{A.13}$$

Nun kann man zeigen daß $(\mathbf{q}\mathbf{v}_0\mathbf{q}^*)\mathbf{v}_1^* = \mathbf{v}_1\mathbf{v}_0^*$ gilt.

$$\begin{aligned}(\mathbf{q}\mathbf{v}_0\mathbf{q}^*)\mathbf{v}_1^* &= (\mathbf{q}\mathbf{v}_0(\mathbf{v}_1\mathbf{v}_0^*)^*)\mathbf{v}_1^* = \\ &= \mathbf{q}\mathbf{v}_0(\mathbf{v}_0\mathbf{v}_1^*))\mathbf{v}_1^* = \\ &= \mathbf{q}(\mathbf{v}_0\mathbf{v}_0)(\mathbf{v}_1^*\mathbf{v}_1^*) = \\ &= \mathbf{q}(-1)(-1) = \\ &= \mathbf{q} = \mathbf{v}_0\mathbf{v}_1^*\end{aligned} \tag{A.14}$$

Dies bedeutet, daß $\mathbf{q}\mathbf{v}_0\mathbf{q}^*$ in derselben Ebene wie $\mathbf{v}_0$ und $\mathbf{v}_1$ liegt und mit $\mathbf{v}_1$ den Winkel Ω einschließt. Zusätzlich kann man aus

$$(\mathbf{q}\mathbf{v}_1\mathbf{q}^*)(\mathbf{q}\mathbf{v}_0\mathbf{q}^*)^* = (\mathbf{q}(\mathbf{q}\mathbf{v}_0)\mathbf{q}^*)(\mathbf{q}\mathbf{v}_0\mathbf{q}^*)^* = (\mathbf{q}(\mathbf{q}\mathbf{v}_0\mathbf{q}^*)(\mathbf{q}\mathbf{v}_0\mathbf{q}^*)^* = \mathbf{q} \tag{A.15}$$

folgern, daß auch $(\mathbf{q}\mathbf{v}_1\mathbf{q}^*)$ in dieser Ebene liegt und mit $(\mathbf{q}\mathbf{v}_0\mathbf{q}^*)$ den Winkel Ω bildet. Somit dreht q die Vektoren $\mathbf{v}0$,$\mathbf{v}1$ um 2Ω mit $\hat{\mathbf{v}}$ als Rotationsachse. Nun muß die Wirkung von $\mathbf{q}$ auf $\hat{\mathbf{v}}$ untersucht werden. Wegen

$$\begin{aligned}\mathbf{q}\hat{\mathbf{v}} &= [\cos\Omega\ ,\ \hat{\mathbf{v}}\sin\Omega]\,[0\ ,\ \hat{\mathbf{v}}] = [-\hat{\mathbf{v}}\sin\Omega\hat{\mathbf{v}}\ ,\ \hat{\mathbf{v}}\cos\Omega] = \\ &= [0\ ,\ \hat{\mathbf{v}}]\,[\cos\Omega\ ,\ \hat{\mathbf{v}}\sin\Omega] = \hat{\mathbf{v}}\mathbf{q}\end{aligned} \tag{A.16}$$

gilt dann $\mathbf{q}\hat{\mathbf{v}}\mathbf{q}^* = \hat{\mathbf{v}}\mathbf{q}\mathbf{q}^* = \hat{\mathbf{v}}$, also wird $\hat{\mathbf{v}}$ nicht verändert, und die Bezeichnung als Rotationsachse ist gerechtfertigt.
Nun muß man auf einen beliebigen Vektor p verallgemeinern.
Da sich $p = s_0\mathbf{v}_0 + s_1\mathbf{v}_0 + s_2\hat{\mathbf{v}}$ darstellen läßt, ergibt sich aufgrund der Bilinearität dann

$$\begin{aligned}\mathbf{q}(s_0\mathbf{v}_0 + s_1\mathbf{v}_0 + s_2\hat{\mathbf{v}})\mathbf{q}^* &= \mathbf{q}(s_0\mathbf{v}_0)\mathbf{q}^* + \mathbf{q}(s_1\mathbf{v}_0) + q(s_2\hat{\mathbf{v}})\mathbf{q}^* = \\ &= s_0\mathbf{q}(\mathbf{v}_0)\mathbf{q}^* + s_1\mathbf{q}(\mathbf{v}_0) + s_2q(\hat{\mathbf{v}})\mathbf{q}^*,\end{aligned} \tag{A.17}$$

die Einzelterme aber wurden bereits untersucht. Somit dreht also $\mathbf{q}$ jeden Vektor mit 2Ω um $\hat{\mathbf{v}}$.
Die Kombination zweier Rotationen, wobei zuerst mit $\mathbf{q}_1$ dann mit $\mathbf{q}_2$ gedreht wird, ist durch $\mathbf{q} = \mathbf{q}_1\mathbf{q}_2$ gegeben, da

$$\mathbf{q}_1(\mathbf{q}_2\mathbf{p}\mathbf{q}_2^*)\mathbf{q}_1^* = (\mathbf{q}_1\mathbf{q}_2)\mathbf{p}(\mathbf{q}_2\mathbf{q}_1)^* = \mathbf{q}\mathbf{p}\mathbf{q}^* \tag{A.18}$$

gilt.

A.3 Umwandlung Quaternion↔Rotationsmatrix

Die Matrix $\mathbf{R}$, die die gleiche Drehung wie $\mathbf{q} = [w\ ,\ (x, y, z)]$ ausführt, ist durch

$$\mathbf{R} = \begin{bmatrix} 1-2(y^2+z^2) & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & 1-2(x^2+z^2) & 2(yz-wx) \\ 2(xz-wy) & 2(yz+wx) & 1-2(x^2+y^2) \end{bmatrix} \tag{A.19}$$

gegeben.
Umgekehrt kann man $\mathbf{q} = [w\ ,\ (x, y, z)]$ aus $\mathbf{R}$ folgendermaßen bestimmen:

$$\begin{aligned} w &= \frac{1}{2}\sqrt{(\mathrm{Tr}\,\mathbf{R}+1)} \\ x &= \frac{R_{3,2}}{\sqrt{(\mathrm{Tr}\,\mathbf{R}+1)}} \\ y &= \frac{R_{1,3}}{\sqrt{(\mathrm{Tr}\,\mathbf{R}+1)}} \\ z &= \frac{R_{2,1}}{\sqrt{(\mathrm{Tr}\,\mathbf{R}+1)}}. \end{aligned} \tag{A.20}$$

Anhang B

Insertion-sort

Insertion-sort ist ein elementarer Algorithmus zum Sortieren von Datensätzen. Es entspricht weitestgehend dem Verfahren, das der Mensch beim Sortieren von Spielkarten verwendet.
Es wird jetzt angenommen, daß man so sortieren will, daß am Ende die Daten in aufsteigender Reihenfolge vorliegen; für absteigende Sortierung läuft das Verfahren analog. Man arbeitet mit zwei Bereichen, einer, in dem die bereits sortierten Daten stehen und einem, der die noch einzusortierenden Datensätze enthält.[1]
Man entnimmt der unsortierten Liste das Element, das in der Liste ganz vorne steht. Man vergleicht die nun nacheinander mit den Elementen aus der sortierten Liste, wobei man bei den größten Elementen beginnt. Ist das neu einzuordnende dann größer als das Vergleichsobjekt, wird es nach diesem eingefügt.
Nach Sedgewick [Sed92] benötigt Insertion-sort im Durschnitt bei zufällig verteilten Daten $\frac{N^2}{4}$ Vergleiche und $\frac{N^2}{8}$ Austauschoperationen. Das Zeitverhalten geht mit $\mathcal{O}(n^2)$, dies ist in Abbildung B.2 eindeutig erkennbar. Liegen die Daten in „falscher“ Reihenfolge vor, muß also aus einer absteigenden Zahlenfolge eine aufsteigende gemacht werden, sind $\frac{N^2}{2}$ Vergleiche und $\frac{N^2}{4}$ Vertauschungen notwendig. Zwar geht auch hier der Zeitaufwand mit $\mathcal{O}(n^2)$, die wirkliche Rechenzeit ist aber genau doppelt so groß. Es ist zu bedenken, daß $\mathcal{O}(n)$ immer ohne Vorfaktoren angegeben wird, siehe auch Anhang E.1.
Der für die Simulation interessanteste Fall ist allerdings eine „fast“ geordnete Liste. Dazu soll als Grenzfall die geordnete Liste betrachtet werden. Für jedes Element muß nur mit seinem Vorgänger verglichen werden, Austauschoperationen sind nicht nötig. Der Rechenzeitbedarf ist also von der Ordnung $\mathcal{O}(n)$. Bei der nicht vollständig geordneten Liste nimmt man an, daß insgesamt für alle Elemente noch zusätzlich k Vergleiche und Vertauschungen am Listenende benötigt werden, dabei sollte k nicht größer als einige n werden, und so klein wie möglich sein.[2] Der Zeitbedarf geht dann mit $\mathcal{O}(n+k)$, ist also wie man auch in Abbildung B.3 sieht, linear. Für diesen Fall ist

[1] Im Programm selber trennt man diese beiden nicht mehr strikt, sondern verwendet entweder zwei Teile eines Arrays, oder man teilt die linear verkettete Liste geeignet.

[2] $\frac{k}{n}$ gibt dann an, um wieviele Plätze ein Element im Durchschnitt falsch liegt.

Insertion-sort jedem „besseren“, komplexen Verfahren wie z.B Quicksort, Heap Sort o.ä deutlich überlegen.
Insertion-sort implementiert mit FORTRAN 90.

```
program inssort
implicit none
cend -&--1---------2---------3---------4---------5---------6
c Variablendeklarationen
integer,parameter :: maximal=10000
integer :: N
double precision,dimension(:),pointer :: liste
integer cnt
c Vorarbeiten
allocate(liste(maximal+1))     ! Speicherplatz belegen
liste(1)=-32000  ! Sentinel
N=100                          ! Zahl der zu sortierenden Elemente
c Hauptprogramm
do cnt=1,N
       liste(cnt+1)=cnt+((4.0*rand())-2.0) ! Fast sortiert
c  liste(cnt+1)=maximal-cnt ! schlechteste Sortierung
c  liste(cnt+1)=100000*rand()             ! zufällig verteilt
   end do
call sort(liste,N)  ! Die Sortierung selbst
c Ausgabe
write (*,*) 'Sortierte Liste'
do cnt=1,N
 write (*,*) liste(cnt+1)
   end do
c Programmende
```

```
contains
com -----------------------------------------------------
subroutine sort(liste,N)
double precision,dimension(:),pointer :: liste
integer N
c first written on 06/30/1995
c rev 1.0
cend -&--1---------2--------3---------4---------5---------6
c lokale Variablen
double precision vgl_element    ! Einzusortierendes Element
integer cnt,pos                 ! Zaehler
do cnt=3,N+1 ! Schleife ab dem 2. Listenelemente
 pos=cnt ! von hier aus rueckwaerts testen
 vgl_element=liste(cnt) ! das muss einsortiert werden
 do while(vgl_element.le.liste(pos))
  pos=pos-1                     ! korrekten Platz suchen
       end do
 liste(pos+2:cnt)=liste(pos+1:cnt-1) !eine Position nach rechts
 liste(pos+1)=vgl_element
end do
end subroutine sort

end program inssort
```

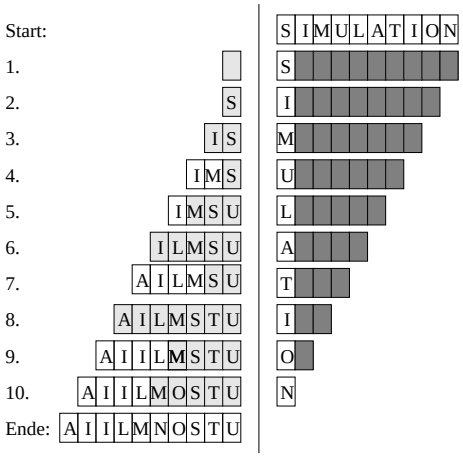

Abbildung B.1: Hier wird schematisch Insertion-sort dargestellt. Die Buchstaben des Wortes „SIMULATION“ sollen geordnet werden. Links der Trennlinie ist der bereits sortierte Bereich. In jedem Schritt muß dort mit Hellgrau unterlegten Buchstaben verglichen werden, bis der korrekte Platz gefunden wurde. Rechts sind die noch einzuordnenden Zeichen, die schwarzen Kästchen sind im jeweiligen Schritt noch unwichtig.

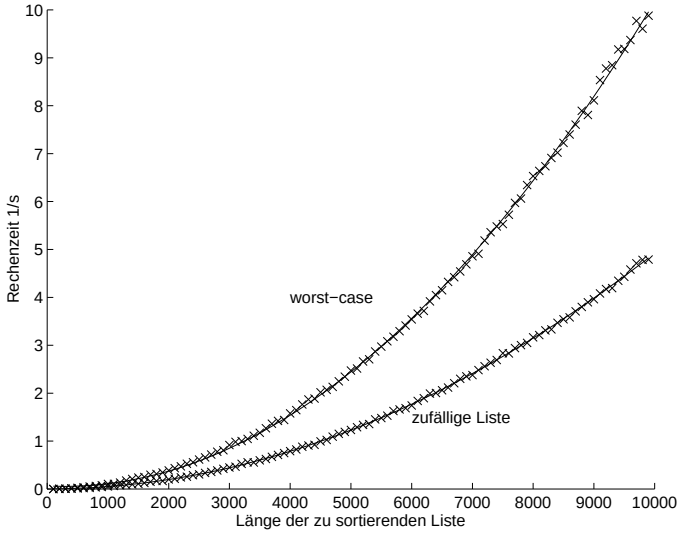

Abbildung B.2: Laufzeitverhalten von Insertion-sort: Sortiert wurden verschieden lange Listen in aufsteigende Reihenfolge sortiert. Dabei waren die zu sortierenden Listen entweder zufällig belegt, oder im „worst-case“ abfallend angeordnet. Sehr gut ist die Ordnung $\mathcal{O}(n^2)$ des Laufzeitverhaltens zu erkennen.

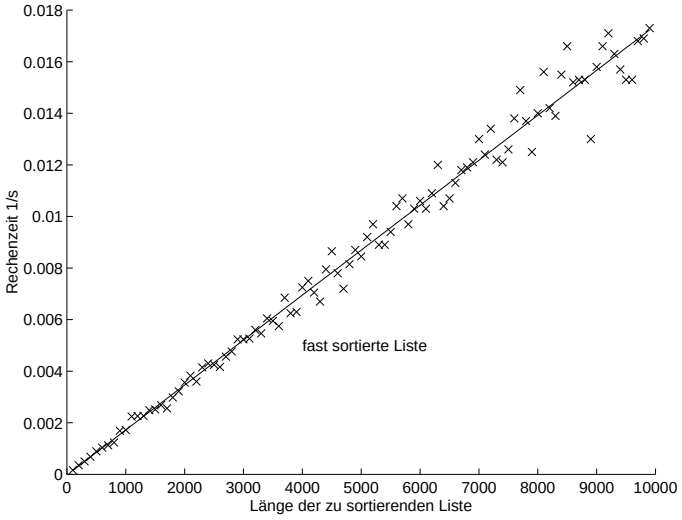

Abbildung B.3: Im Fall einer fast sortierten Liste ist das Laufzeitverhalten von Insertion-sort fast linear.

Anhang C

closest-feature-Algorithmus

Das nachfolgende FORTRAN 90-Programmfragment ist die Kernroutine des closest-feature-Algorithmus und stammt aus dem Programm, das währenden der Diplomarbeit entwickelt wurde:

```
com ------------------------------------------------------------
subroutine closest_features(particle1,cf1,particle2,cf2,
     #  distance,time_step)
type (polyeder) particle1,particle2
type (feature) cf1,cf2
double precision distance
integer time_step
c function :
c first written on 5/4/1995
c rev 1.0
c rev 1.1 Einf"urung von time_step
cend -&--1---------2--------3---------4---------5---------6----
c lokale Variablen
 integer cycle_cnt
 double precision dummy
 cycle_cnt=0
 distance = -infinity
 do while (distance.lt.0.0)
  cycle_cnt=cycle_cnt+1
  if (cycle_cnt.eq.100) then
   call cycle_check
  end if
  select case(cf1%feature_type)
  case (T_VERT)
   select case(cf2%feature_type)
   case (T_VERT)
c Fall 1: particle1->vertex und particle2->vertex
    call check_vertex_vertex(cf1%p_vertex,cf2%p_vertex,
     #     distance,time_step,
     #     cf1%feature_type,cf1%p_vertex,cf1%p_edge,cf1%p_face,
     #     cf2%feature_type,cf2%p_vertex,cf2%p_edge,cf2%p_face)
   case (T_EDGE)
c Fall 2: particle1->vertex und particle2->edge
    call check_vertex_edge(cf1%p_vertex,cf2%p_edge,
     #     distance,time_step,
     #     cf1%feature_type,cf1%p_vertex,cf1%p_edge,cf1%p_face,
     #     cf2%feature_type,cf2%p_vertex,cf2%p_edge,cf2%p_face)
   case (T_FACE)
c Fall 3: particle1->vertex und particle2->face
    call check_vertex_face(cf1%p_vertex,cf2%p_face,
     #     distance,time_step,
     #     cf1%feature_type,cf1%p_vertex,cf1%p_edge,cf1%p_face,
     #     cf2%feature_type,cf2%p_vertex,cf2%p_edge,cf2%p_face)
   end select
  case (T_EDGE)
   select case(cf2%feature_type)
   case (T_VERT)
c Fall 4: particle1->edge und particle2->vertex
    call check_vertex_edge(cf2%p_vertex,cf1%p_edge,
     #     distance,time_step,
     #     cf2%feature_type,cf2%p_vertex,cf2%p_edge,cf2%p_face,
     #     cf1%feature_type,cf1%p_vertex,cf1%p_edge,cf1%p_face)
   case (T_EDGE)
c Fall 5: particle1->edge und particle2->edge
    call check_edge_edge(cf1%p_edge,cf2%p_edge,
     #     distance,time_step,
     #     cf1%feature_type,cf1%p_vertex,cf1%p_edge,cf1%p_face,
     #     cf2%feature_type,cf2%p_vertex,cf2%p_edge,cf2%p_face)
   case (T_FACE)
c Fall 6: particle1->edge und particle2->face
    call check_edge_face(cf1%p_edge,cf2%p_face,
     #     particle2,
     #     distance,time_step,
     #     cf1%feature_type,cf1%p_vertex,cf1%p_edge,cf1%p_face,
     #     cf2%feature_type,cf2%p_vertex,cf2%p_edge,cf2%p_face)
   end select
  case (T_FACE)
   select case(cf2%feature_type)
   case (T_VERT)
c Fall 7: particle1->face und particle2->vertex
    call check_vertex_face(cf2%p_vertex,cf1%p_face,
     #     distance,time_step,
     #     cf2%feature_type,cf2%p_vertex,cf2%p_edge,cf2%p_face,
     #     cf1%feature_type,cf1%p_vertex,cf1%p_edge,cf1%p_face)
   case (T_EDGE)
c Fall 8: particle1->face und particle2->edge
    call check_edge_face(cf2%p_edge,cf1%p_face,
     #     particle1,
     #     distance,time_step,
     #     cf2%feature_type,cf2%p_vertex,cf2%p_edge,cf2%p_face,
     #     cf1%feature_type,cf1%p_vertex,cf1%p_edge,cf1%p_face)
   case (T_FACE)
c Fall 9: particle1->face und particle2->face
    call check_face_face(cf1%p_face,cf2%p_face,
     #     particle1,particle2,
     #     distance,time_step,
     #     cf1%feature_type,cf1%p_vertex,cf1%p_edge,cf1%p_face,
     #     cf2%feature_type,cf2%p_vertex,cf2%p_edge,cf2%p_face)
   end select
 end select
end do
end subroutine closest_features
```

Dieses Flußdiagramm stellt die Grundstruktur für die einzelnen Tests beim closest-feature-Algorithmus dar:

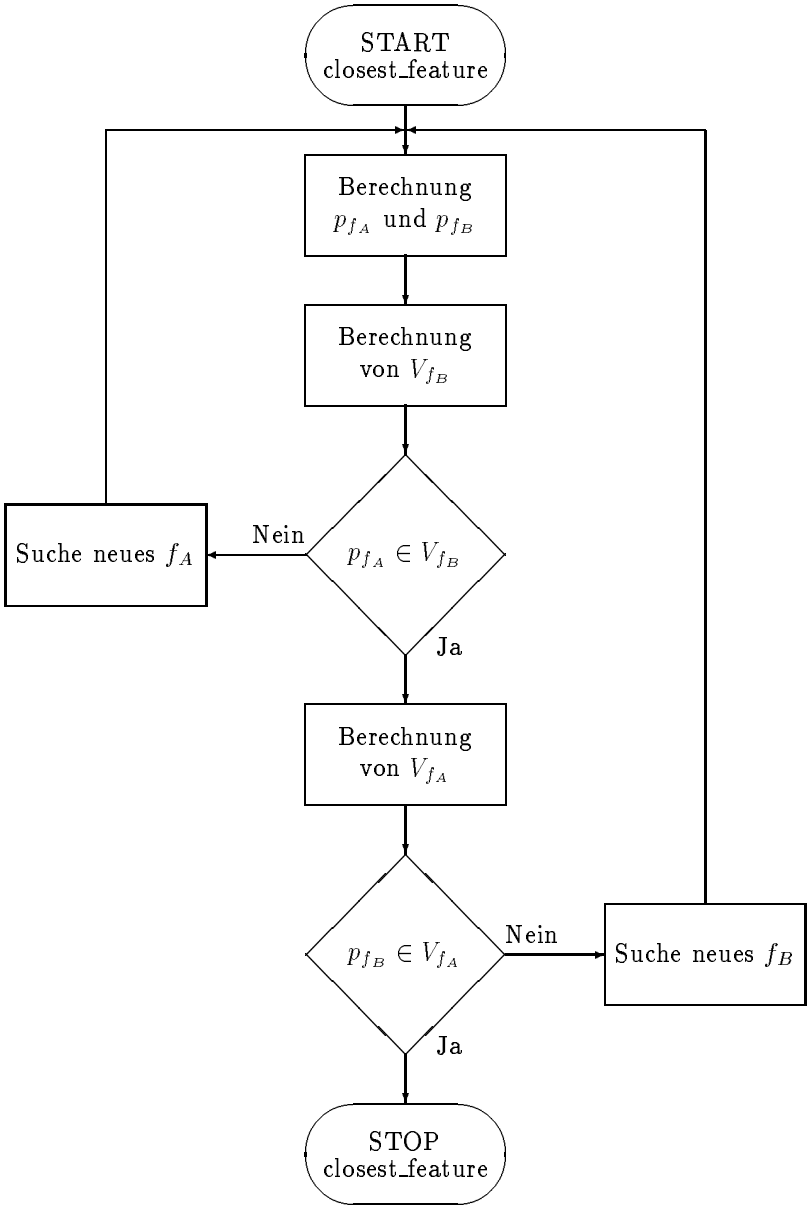

Abbildung C.1: Flußdiagramm für den closest-feature-Algorithmus . Dabei sind f_A f_B Elemente der Polyeder A B , V_{f_A},V_{f_B} die zu f_A bzw. f_B gehörigen Voronoiregionen, und p_{f_A},p_{f_B} die sich nächsten Punkte auf f_A und f_B .

Die nachfolgenden Beschreibungen stammen aus [Lin93], sie enthalten alle Sonderfälle, die zu berücksichtigen sind, wenn es notwendig ist, zwei Flächen, oder eine Fläche und eine Ecke zu prüfen:

Given an edge E and a face F, the algorithm first checks **IF E AND F ARE PARALLEL**, i.e. the two endpoints H_E and T_E are equi-distant from the face F (i.e their signed distance is the same). the *signed* distance between H_E and F can be computed easily by $H_E \cdot N_F$ where N_F is F's unit outward normal and similarly for the signed distance between F and H_T

THEN, when the head H_E and the tail T_E of the edge E are equi-distant from the face F, the subroutine checks **IF E INTERSECTS F'S PRISM REGION**, (i.e. the region bounded by F's constraint planes constructed from the edges in F's boundary).

THEN, if E intersects the prism region of F, the algorithm checks **IF THE HEAD H_E (OR TAIL T_E OF E LIES ABOVE THE FACE** F.

THEN, it will check **IF THE FACE OUTWARD NORMAL N_F IS "BOUNDED"BY THE NORMALS OF THE EDGE'S LEFT AND RIGHT FACES**, say N_L and N_R respectively, to verify that the face F lies inside the Voronoi region of the Edge E, not in the Voronoi region of E's left or right face (F_L and F_R). This is done by checking if $(N_L \times \vec{E}) \cdot N_f > 0$ and $(\vec{E} \times N_L) \cdot N_f > 0$, where N_F is the outward normal of the face F, $\vec{E}$ is the edge vector, and N_R,N_L denote the outward normal of the edge's right and left face respectively.

THEN, the edge E and the face F will be returned as the closest feature pair, since E and F lie within the Voronoi region of each other.

ELSE, it return the right or left face (F_L and F_R) of E, which ever is closer to F

ELSE, when the head H_E (implies E also) lies beneath F, then it will invoke a subroutine *e-find-min* which finds the closest feature on the polyhedron B containing F to E by enumerating all th features on B. Then the algorithm will return E and this new feature f_B as the next candidates for the closest feature pair verification

ELSE, E does not intersect the prism defined by F's boundary edges, then there is at least one edge of F's boundary which is closer to E than the interior of F is to E (by the definition of Voronoi region). So, when the test failed, it calls a constant time routine (which takes time proportional to the constant number of faces boundary) and returns the edge E and the closest edge or vertex, say f_F, on the boundary to E.

ELSE, the algorithm checks **IF ONE END POINT LIES ABOVE THE FACE** F **AND THE OTHER LIES BENEATH** F.

THEN, if so there is one edge on F's boundary which is closer to E than the interior of F. So, the algorithm will find the closest edge or vertex f_B, which has the minimum distance to E among all edges and vertices in $F's$ boundary, and return (E,f_b) as the next candidate pair.

ELSE, one endpoint H_E(or T_E) of E is closer (the magnitude $|V_e \cdot N_F|$ instead of the signed distance is used for comparison here) to F than the other endpoint, the algorithm will check **IF THE CLOSER ENDPOINT** V_E **LIES INSIDE OF** $F'S$ **PRISM REGION** (bounded by constraint planes generated by the edges in F's boundary), $Cell_F$, as in the vertex-face case.

THEN, if closer endpoint E_E lies inside of F's prism region, then the algorithm will next verify **IF THIS ENDPOINT LIES ABOVE THE FACE** F (to detect a local minimum of distance function).

THEN, if V_E (the closer endpoint to F) is above F the algorithm next verifies **IF THE CLOSEST POINT** P_F **ON** F **TO** V_E **SATISFIES** V_E**'S APPLICABILITY CONSTRAINTS.**

THEN, if so the algorithm returns V_E and F as the closest feature pair.

ELSE some constraint plane in V_E's Voronoi cell $Cell_{V_E}$ is violated. This constraint plane is generated by an edge, say E', in V_E's coboundary. This edge E' is closer to F, so the algorithm will return (E',F) as the next candidate pair.

ELSE, if the closer endpoint V_E of E lies beneath F, then the algorithm finds the closest feature F_B on the polyhedron B to V_E. The algorithm return (V_e,f_B as the next candidate feature pair.

ELSE, if the closer endpoint V_E of E to F lies outside of $F's$ prism region, then there is one edge on F's boundary which is closer to E than the interior of F. So, the algorithm will find the closest edge or vertex f_B, which has the minimum distance to E among all edges and vertices in $F's$ boundary, and return (E,f_B) as the next candidate pair

END

(0) Given two faces, F_A and F_B, the algorithm first checks **IF THEY ARE PARALLEL**.

THEN, if they are parallel, it invokes a subroutine which runs in constant time (proportional to the product of numbers of boundaries between two faces, which is a constant after subdivision) to check **IF THE ORTHOGONAL PROJECTION OF** F_A **ON THE PLANE OF** F_B **OVERLAPS** F_B (i.e F_A intersects the prism region of F_B).

(1) **THEN**, if they overlapped, the algorithm next checks **IF** P_A (implies F_A as well, since the two faces are parallel) **LIES ABOVE** F_B, where P_A is one of the points on F_A and its projection down to F_B is in the overlap region.

(2) **THEN**, if P_A (thus F_A) lies above F_B, the algorithm next checks **IF** P_B **THUS** F_B**) ALSO LIES ABOVE** F_A, where P_B is one of the points and its projection down to F_B is in the overlap region.

(3) **THEN**, if P_B (thus F_B) is above F_A and vice versa, we have a pair of closest points (P_A,P_B) satisfy the respective applicability constraints of F_B and F_A. so, P_A and P_B are the closest points on A and B; and the features containing them, F_A and F_B, will be returned as the closest features.

(4) **ELSE**, if P_B (thus F_B) lies beneath F_A, then the algorithm finds the closest feature f_A on polyhedron A containing F_A to F_B by enumerating all features of A to search for the feature which has minimum distance to F_B. So, the new features (f_A,F_B) is guaranteed to be closer than (F_A,F_B).

(5) **ELSE**, if P_A (thus F_A) lies beneath F_B then the algorithm finds the closest feature f_B on polyhedron B containing F_B to F_A and returns (F_A,f_b) as the next candidate feature pair. This pair of new features has to be closer than F_A to F_B since f_b has the shortest distance among all features on B to F_A.

(6) **ELSE**, if F_A and F_B are parallel but the projection of F_A does not overlap F_B, a subroutine enumerates all possible combination of edge–pairs (one from the boundary of each face respectively) and computes the distance between each edge–pair to find a pair of closest edges which has the minimum distance among all. This new pair of edges (E_A,E_B) is guaranteed to be closer than the two faces, since two faces F_A and F_B do not contain their boundaries.

(7) **ELSE**, id F_A and F_B are not parallel, this implies there is at least one vertex or edge on F_A closer to F_B than F_A to F_B *or* vice versa. So, the algorithm first finds the closest vertex or edge on F_A to the plane containing F_B, Φ_B.

(8) comVERTEX: if there is only a single closest vertex V_A to the plane containing F_B, V_A is closer to F_B than F_A is to F_B since two faces are not parallel and their boundaries must be closer to each other than their interior. Next the algorithm checks if V_A lies within the voronoiregion of F_B.

(9) **THEN**, if so V_A and F_B will be returned as the next candidate pair.

(10) **ELSE**, the algorithm finds the closest vertex (or edge) on F_B to Φ_A, and proceeds as before.

(11) comVERTEX: if there is only a single vertex V_B closest to the plane Φ_A containing F_A, the algorithm checks if V_B lies inside of F_A's Voronoi region bounded by the constraint planes generated by the edges in F_A's boundary and above F_A.

(12) **THEN**, if V_B lies inside of F_A's Voronoi region, then F_A and V_B will naturally be the next candidate pair. This new pair of features is closer than the previous pair as already analyzed in (10).

(13) **ELSE**, the algorithm performs a search of all edges from the boundary of F_A and F_B and returns the closest pair (E_A,E_B) as the next candidate feature pair.

(14) **EDGE:**, similarly if the closest feature on F_B to Φ_A is an edge E_B, the algorithm once again checks if E_B cuts F_A's prism to determine whether F_A and E_B are the next candidate features.

(15) **THEN**, the algorithm checks **IF THE END POINT OF E_B LIES ABOVE F_A.**

(16) **THEN**, the new pair of candidate features is F_A and E_B since E_B lies within F_A's Voronoi region.

(17) **ELSE**, the algorithm searches for the closest feature f_A which has the minimum distance among all other features on A to E_B and returns (f_A,E_B) as the next pair of candidate features.

(18) **ELSE**, if not, then this implies that neither face contains the closest points since neither F_A nor F_B's Voronoi regions contain a closest point from the other's boundary. So, the algorithm will enumerate all the edge–pairs in the boundaries of F_A and F_B to find the pair of edges which is closest in distance. This new pair of edges (E_A,E_B) is guaranteed to be closer than two faces.

(19) **EDGE:** if there are two closest vertices on F_A to Φ_B, then the edge E_A containing the two vertices is closer to Φ_B than F_A is. Next, the algorithm checks **IF E_A INTERSECTS THE PRISM REGION OF F_B**, as in the edge-face case.

(20) **THEN**, the algorithm checks **IF THE END POINT OF** E_A **LIES ABOVE** F_B.

(21) **THEN**, if so the edge E_A and F_B will be returned as the next candidate pair.

(22) **ELSE**, the algorithm searches for the closest feature f_b which has the minimum distance among all other features on B to E_A and returns (E_A, f_b) as the next pair of candidate features.

(23) **ELSE**, if this is not the case, then the algorithm finds the closest vertex (or edge) on F_B to the plane containing F_A, say Φ_A and proceeds as before, from block (11) to block (18).

END

Nachfolgend sind die grundsätzlichen Dateistrukturen und deren Verknüpfungen dargestellt:

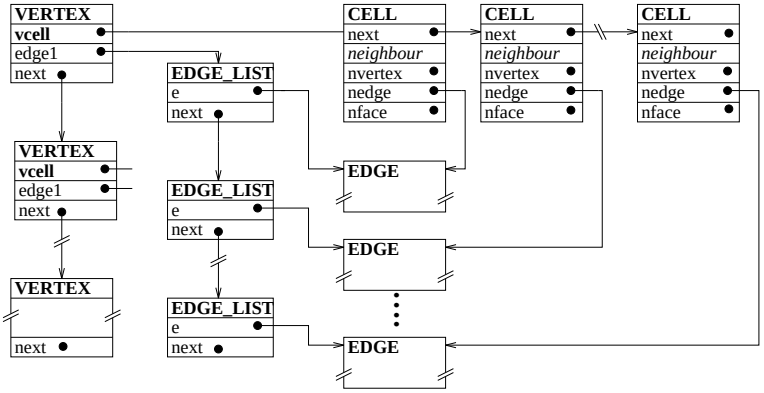

Abbildung C.2: Datenstruktur einer Ecke. Pfeile repräsentieren Pointer, Boxen stellen Strukturen dar. Die zusätzliche verkettete Liste der Kanten ist nötig, da die Zahl der Kanten, die in einer Ecke zusammentreffen, nicht festgelegt ist.

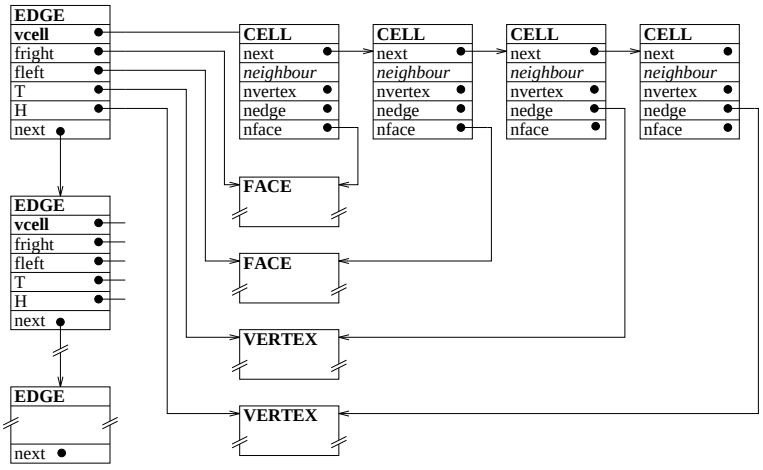

Abbildung C.3: Datenstruktur einer Kante. Pfeile repräsentieren Pointer, Boxen stellen Strukturen dar.

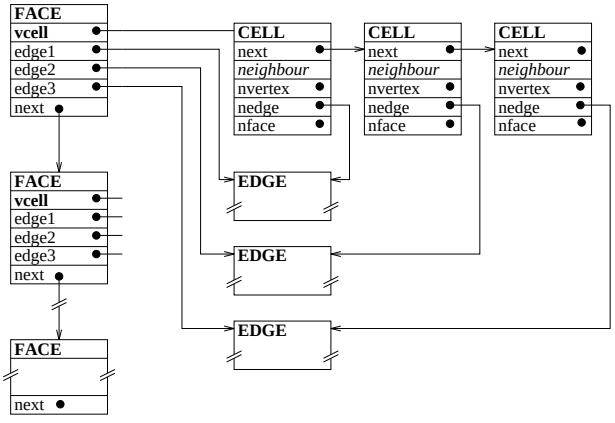

Abbildung C.4: Datenstruktur einer Fläche. Pfeile repräsentieren Pointer, Boxen stellen Strukturen dar.

Anhang D

Befehlsworte in der Enviroment-Datei

Zulässige Variablen im `GRAFIK`–Bereich

`STD_FONT`	-1		Schriftart für KOS,etc.	Tab.: D.2
`STD_SIZE`	4	mm	Standartschrifthöhe	≥ 0
`TITEL_X`	240	mm	X-Koordinate der Titelzeile	
`TITEL_Y`	265	mm	Y-Koordinate der Titelzeile	
`TITEL_FONT`	-1		Schriftart für die Titelzeile	Tab.: D.2
`TITEL_SIZE`	8	mm	Schrifthöhe für die Titelzeile	≥ 0
`TEXT_LEFT`	26	mm	siehe Abb.: D.1	
`TEXT_TOP`	250	mm	siehe Abb.: D.1	
`TEXT_DIST`	10	mm	siehe Abb.: D.1	
`TEXT_FONT`	-1		Schriftart für Meßwerte	Tab.: D.2
`TEXT_SIZE`	4	mm	Schrifthöhe für Meßwerte	≥ 0
`V_COLOR`	1		`0` alle Teilchen werden in derselben Farbe gezeichnet `1` $v = v_{min}$ bis $v = v_{max}$ `2` linear bis ca. $v \geq 10\frac{mm}{s}$ `3` linear bis ca. $v \geq 100\frac{mm}{s}$ `4` linear bis ca. $v \geq 1000\frac{mm}{s}$ `5` linear bis ca. $v \geq 10000\frac{mm}{s}$ `6` $\log_{10}$ bis ca. $v \geq 10000\frac{mm}{s}$ `7` $\log_2$ bis ca. $v \geq 64\frac{mm}{s}$	$0-7$
`SCR_XMIN`	100	mm	X_{min} für den Zeichenbereich von gr3	?
`SCR_YMIN`	30	mm	Y_{min} für den Zeichenbereich von gr3	?
`SCR_XMAX`	370	mm	X_{max} für den Zeichenbereich von gr3	?
`SCR_YMAX`	255	mm	Y_{max} für den Zeichenbereich von gr3	?

Fortsetzung auf der nächsten Seite

Fortsetzung der letzten Seite				
Name	Std.	Eh.	Erklärung	zulässig
AXS_XMIN	-1	LE	X_{min} für den Wertebereich der KO-Achsen	$\leq$AXS_XMAX
AXS_YMIN	-1	LE	Y_{min} für den Wertebereich der KO-Achsen	$\leq$AXS_YMAX
AXS_ZMIN	-1	LE	Z_{min} für den Wertebereich der KO-Achsen	$\leq$AXS_ZMAX
AXS_XMAX	20	LE	X_{max} für den Wertebereich der KO-Achsen	$\geq$AXS_XMIN
AXS_YMAX	20	LE	Y_{max} für den Wertebereich der KO-Achsen	$\geq$AXS_YMIN
AXS_ZMAX	20	LE	Z_{max} für den Wertebereich der KO-Achsen	$\geq$AXS_ZMIN
SCL_XMIN	-1	LE	X_{min} für die Beschriftung der KO-Achsen	$\leq$SCL_XMAX
SCL_YMIN	-1	LE	Y_{min} für die Beschriftung der KO-Achsen	$\leq$SCL_YMAX
SCL_ZMIN	-1	LE	Z_{min} für die Beschriftung der KO-Achsen	$\leq$SCL_ZMAX
SCL_XMAX	20	LE	X_{max} für die Beschriftung der KO-Achsen	$\geq$SCL_XMIN
SCL_YMAX	20	LE	Y_{max} für die Beschriftung der KO-Achsen	$\geq$SCL_YMIN
SCL_ZMAX	20	LE	Z_{max} für die Beschriftung der KO-Achsen	$\geq$SCL_ZMIN
X_ANGLE	0	rad	Drehung des KOS um die X-Achse bei der Darstellung	$0 - 2\pi$
Y_ANGLE	0	rad	Drehung des KOS um die Y-Achse bei der Darstellung	$0 - 2\pi$
Z_ANGLE	0	rad	Drehung des KOS um die Z-Achse bei der Darstellung	$0 - 2\pi$
X_ROTSPEED	0	$\frac{rad}{s}$	Winkelgeschw. des KOS um die X-Achse bei der Darstellung	≥ 0
Y_ROTSPEED	0	$\frac{rad}{s}$	Winkelgeschw. des KOS um die Y-Achse bei der Darstellung	≥ 0
Z_ROTSPEED	0	$\frac{rad}{s}$	Winkelgeschw. des KOS um die Z-Achse bei der Darstellung	≥ 0
PLOTTYPE	0		0=ALL alle Linien werden gezeigt 1=HID Hidden Line Algorithmus 2=MIX verdeckte Linien	$0 - 2$
Zulässige Variablen im GLOBAL–Bereich				
BBOX	1		1= Ellipsoid Methode für BB 0= Min-Max Methode für BB	0,1
RAND_SEED	4711		Seed für den Zufallszahlengenerator	
TIME_START	0.0	s	Beginn der Zeitmessung	
TIME_ENDE	0.0	s	Ende der Zeitmessung	
TIME_STEP	0.1	s	Schrittweite der Zeitmessung	
MESS_TIMES	1		Zeitschritte zwischen zwei Messungen 0 = Nur Messung der Anfangskonfiguration < 0 = Keine Messung	
MESS_TIMES	1		Zeitschritte zwischen zwei Bildschirmupdates 0 = Nur Darstellung der Anfangskonfiguration	

Fortsetzung auf der nächsten Seite

Fortsetzung der letzten Seite				
Name	Std.	Eh.	Erklärung $< 0 =$ Keine Messung	zulässig
SAND_XO	0	LE	X-Koordinate der l.u. Ecke des Sandquaders	
SAND_YO	0	LE	Y-Koordinate der l.u. Ecke des Sandquaders	
SAND_ZO	0	LE	Z-Koordinate der l.u. Ecke des Sandquaders	
SAND_XSIZE	10	LE	Breite des Sandquaders	
SAND_YSIZE	10	LE	Tiefe des Sandquaders	
SAND_ZSIZE	10	LE	Höhe des Sandquaders	
SAND_A_LEN	1	LE	Länge der 1. Halbachse des Ellipsoids	
SAND_B_LEN	1	LE	Länge der 1. Halbachse des Ellipsoids	
SAND_C_LEN	1	LE	Länge der 1. Halbachse des Ellipsoids	
SAND_DIST	2	LE	Abstände der Sandkörner zu Beginn	
Zulässige Variablen im GEOMETRY–Bereich				
RHO	1	$\frac{kg}{mm^2}$	Dichte des Wandstückes	
VIS_TRIS	*ntri*		Zahl der Dreiecke, die sichtbar sind	$0 - ntri$

Tabelle D.1: Befehlsworte in der Enviroment-Datei. Mit diesen Parametern kann das gesamte Programm gesteuert werden. Es können die verschiedenen Darstellungen gewählt und die Konstanten festgelegt festgelegt werden, ohne daß eine Neucompilieren notwendig ist.

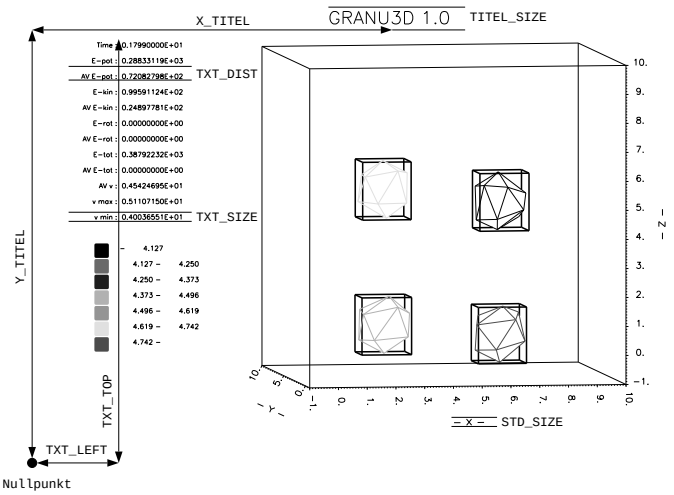

Abbildung D.1: Bedeutung der Parameter, mit denen die Darstellung am Bildschirm beeinflußt werden kann.

Fontname	proportional	proportional kursiv	äquidistant	äquidistant kursiv
Standart	-1	-101	-201	-301
Standart Thick	-2	-102	-202	-302
Standart Filled	-51	-151	-251	-351
Roman	-3	-103	-203	-303
Roman Thick	-4	-104	-204	-304
Italic	-5	-105	-205	-305
Italic Thick	-6	-106	-206	-306
Script	-7	-107	-207	-307
Script Thick	-8	-108	-208	-308
Gothic English	-9	-109	-209	-309
Gothic German	-10	-110	-210	-310
Gothic Italian	-11	-111	-211	-311
Greek Sonderzeichen	-13	-113	-213	-313

Tabelle D.2: GKS-Fonts

Anhang E

Verschiedenes

E.1 $\mathcal{O}$-Terminologie

$\mathcal{O}$ ist das Landausche Größenordnungssymbol, das bedeutet, daß nur die höchste Potenz angegeben wird, von der eine Funktion abhängig ist, nicht die niedrigeren Potenzen oder Vorfaktoren. $\mathcal{O}$ besagt dann, daß Terme niedrigerer Ordnung vernachlässigt werden. l ist im folgenden die Systemgröße.
Rechenregeln:

$$\begin{aligned} \mathcal{O}(l^m + l^n) &= \mathcal{O}(l^{\max(m,n)}) \\ \mathcal{O}(l^m) * \mathcal{O}(l^n) &= \mathcal{O}(l^{m+n}) \\ \mathcal{O}(l^m + \text{const}) &= \mathcal{O}(l^m) \end{aligned} \tag{E.1}$$

Vorsicht: Die Aussagen der Big-O-Terminologie gelten nur für die Größenordnung, nicht für die absolute Rechenzeit. Bestimmend ist oft der Vorfaktor. Dadurch kann manchmal die Verwendung von „theoretisch“ ungünstigeren Algorithmen zu schnelleren Programmen führen, wenn l klein ist.

E.2 Programme für MAPLE

Der folgende Code berechnet die maximale Ausdehnung eines Ellipsoids in Richtung der Koordinatenachsen.

```
> # MAPLE-Programm zur Berechnung der Minima und Maxima eines Ellipsoids
> # in Bezug auf die Koordinatenachsen
> restart;
> with(plots):with(linalg):with(plots):
> # Variablendeklarationen
> loes:=vector(6):erg:=vector(6):gl:=vector[6]:
> n1:='n1':n2:='n2':n3:='n3':n4:='n4':n5:='n5':n6:='n6':
```

```
> xs:=vector(3):x:=vector(3):ri:=matrix(3,3):r:=matrix(3,3):
> ri:=transpose(r):
> # Antisymmetrie der Rotationsmatrix
> ri[1,2]:=-ri[2,1]:ri[1,3]:=-ri[3,1]:ri[3,2]:=-ri[2,3]:
> # Transformiertes Ellipsoid
> xs:=multiply(ri,x):
> gl:=xs[1]**2/a**2+xs[2]**2/b**2+xs[3]**2/c**2;
> gl:=expand(gl,{x[1],x[2],x[3]}):
> glx:=n1*x[1]*x[2]+n2*x[1]*x[3]+n3*x[2]*x[3]+n4*x[1]^2+n5*x[2]^2+n6*x[3]^2;
> # Vereinfachung und Bestimmung von n1,...,n6
> match(gl=glx,{x[1],x[2],x[3]},'lsg'):
> # Auflösen der Ellipsoid-Gleichung nach X,Y,Z
> glx1:=solve(glx=1,x[1]):
> glx2:=solve(glx=1,x[2]):
> glx3:=solve(glx=1,x[3]):
> # Bestimmung der Gleichungssysteme für die Maximalwerte
> glsysx11:={simplify(diff(glx1[1],x[2]))=0,simplify(diff(glx1[1],x[3]))=0}:
> glsysx12:={simplify(diff(glx1[2],x[2]))=0,simplify(diff(glx1[2],x[3]))=0}:
> glsysx21:={simplify(diff(glx2[1],x[1]))=0,simplify(diff(glx2[1],x[3]))=0}:
> glsysx22:={simplify(diff(glx2[2],x[1]))=0,simplify(diff(glx2[2],x[3]))=0}:
> glsysx31:={simplify(diff(glx3[1],x[1]))=0,simplify(diff(glx3[1],x[2]))=0}:
> glsysx32:={simplify(diff(glx3[2],x[1]))=0,simplify(diff(glx3[2],x[2]))=0}:
> # Bestimmung der Lösungen der obengenannten Gleichungssysteme
> loes[1]:=solve(glsysx11,{x[2],x[3]}):
> loes[2]:=solve(glsysx12,{x[2],x[3]}):
> loes[3]:=solve(glsysx21,{x[1],x[3]}):
> loes[4]:=solve(glsysx22,{x[1],x[3]}):
> loes[5]:=solve(glsysx31,{x[1],x[2]}):
> loes[6]:=solve(glsysx32,{x[1],x[2]}):
> # Ersetzten der Abkürzungen n1,...,n6
> assign(lsg);
> x:=vector(3):assign(loes[1]):erg[1]:=(simplify(glx1[1])):
> x:=vector(3):assign(loes[2]):erg[2]:=(simplify(glx1[2])):
> x:=vector(3):assign(loes[3]):erg[3]:=(simplify(glx2[1])):
> x:=vector(3):assign(loes[4]):erg[4]:=(simplify(glx2[2])):
> x:=vector(3):assign(loes[5]):erg[5]:=(simplify(glx3[1])):
> x:=vector(3):assign(loes[6]):erg[6]:=(simplify(glx3[2])):
> # Generierung des FORTRAN- Programms
> fortran(erg,optimized,mode=double);
> # PROGRAMMENDE
```

Mit dem folgenden Programm berechnet Maple Volumen, Schwerpunkt und Trägheitstensor einer Pyramide, die die Bedingungen aus Gleichung 3.3 entspricht. Außerdem wird die Rotationsmatrix 3.5 bestimmt.

```
>  with(linalg):
> # Deklarationen
```

```
> r:=['x','y','z']:
> sp:=vector(3):
> jx:=matrix([[y*y+z*z,-x*y,-x*z],[-x*y,z*z+x*x,-y*z],[-x*z,-y*z,x*x+y*y]]):
> a:='a':b:='b':c:='c':d:='d':
> tmp:=(y-d*x/a)*c/(b-d):
> # Volumen (Massen) Berechnung
> simplify(int(int(int(1,z=0..c*y/b),y=0..b*x/a),x=0..a)):
> mass:=simplify("+int(int(int(1,z=0..tmp),y=b/a*x..d/a*x),x=0..a));
> # Schwerpunkt
> erg1:=map(va->int(int(int(va,z=0..c*y/b),y=0..b*x/a),x=0..a),r):
> erg2:=map(va->int(int(int(va,z=0..tmp),y=b/a*x..d/a*x),x=0..a),r):
> for i from 1 to 3 do sp[i]:=simplify((erg1[i]+erg2[i])/(mass1+mass2)) od;
> mat_j:=matrix(3,3):
> vorfaktor:=1/60*c*a*d:
> for i from 1 to 3 do
>  for j from 1 to 3 do
>    int(int(int(jx[i,j],z=0..tmp),y=b/a*x..d/a*x),x=0..a) ;
>    mat_j[i,j]:="+int(int(int(jx[i,j],z=0..c*y/b),y=0..b*x/a),x=0..a);
>    mat_j[i,j]:=simplify(evalm(mat_j[i,j]/vorfaktor));
>  od;
> od;
> vorfaktor*evalm(mat_j);
> # Rotationsmatrix berechnen
> # Bestimmung der Koordinatenachsen
> # E_1{(D_i)}
> x1_vec:=vector([x1,y1,z1]):
> x2_vec:=vector([x2,y2,z2]):
> # L_i
> xp_vec:=vector([xp,yp,zp]):
> koor1[1]:=vector([1,0,0]):
> koor1[2]:=vector([0,1,0]):
> koor1[3]:=vector([0,0,1]):
> koor2[1]:=evalm(xp_vec):
> koor2[2]:=evalm(x1_vec-xp_vec):
> koor2[3]:=evalm(crossprod(xp_vec,(x1_vec-xp_vec))):
> t:=matrix(3,3):
> for i from 1 to 3 do
>  for j from 1 to 3 do
>   t[i,j]:=cos(angle(koor1[i],koor2[j])):
>  od;
> od;
> print(t);
> # PROGRAMMENDE
```

E.3 FORTRAN 90

In diesem Anschnitt sollen nur kurz die wichtigsten Neuheiten von FORTRAN 90 darstellen, um einem FORTRAN-Programmierer das Studium des Quellcodes zu erleichtern. Auffällig ist als erstes, daß die Variablennamen keiner Längenbeschränkung mehr unterliegen. Es auch können Kommentare in einer Programmzeile stehen, alles nach einem Ausrufezeichen wird vom Compiler nicht mehr ausgewertet.

```
c Deklaration eines Feldes. "::" trennt die Variablen von der Deklaration
      real,dimension(100,3) :: SCHWERPUNKTE ! Schwerpunktskoordinaten
```

Des weiteren beherrscht FORTRAN 90 nun auch Strukturen wie in C. Damit sind dann sehr viel übersichtlichere Programme möglich.

```
c Deklaration einer Struktur mit dem Namen "BEISPIEL"
c Variablen- und Strukturnamen werden hier GROSSGESCHRIEBEN, obwohl
c FORTRAN 90 nicht case-sensitiv ist.
      type (BEISPIEL)
       integer                :: VARIABLE1
       type (UNTERSTRUKTUR)   :: STRUKTUR
      end type                              ! Ende der Typendeklaration
c Deklaration einer Struktur mit dem Name "UNTERSTRUKTUR"
      type (UNTERSTRUKTUR)
       integer,dimension(3) :: VEKTOR
      end type
      type (beispiel) :: DUMMY
c mit % kann auf Elemente von Strukturen zugegriffen werden
      DUMMY%VARIABLE1 = 1
c Durch Kombination von mehreren % ist auch Zugriff
c auf Strukturen in Strukturen möglich
      DUMMY%STRUKTUR%VEKTOR = (/1.0,0.0,0.0/)
      call unterprogramm(DUMMY)
c Definition eines Unterprogramms
      subroutine unterprogramm(PARAMETER1)
      type (beispiel) :: PARAMETER1
c Ausgabe aller Element 1 bis 100 (1:100)
       write (*,*) 3*PARAMETER1%FELD1(1:100)
      end subroutine
```

Die wichtigste Neuerung sind Pointer, damit lassen sich dynamische verkettete Listen erzeugen. Ziele von Pointern müssen mit dem Kennwort `target` deklariert werden.

```
       integer, pointer :: PTR      ! Ein Pointer
       integer,target   :: VARIABLE ! Eine Variable
       PTR => VARIABLE              ! Zuweisung der Adresse von "variable"
       PTR = 42                     ! Zuweisung an die Adresse in "ptr"
       write (*,*) 'The answer is ',VARIABLE,'.'    ! Gibt 42 aus
```

Literaturverzeichnis

[AT86] M.P. Allan and D.J. Tildesley. *Computer Simulation of Liquids.* Oxford University Press, 1986.

[Bar93] David Baraff. Rigid body simulation. In *Course 60, An Introduction to Physically Based Modelling*, pages H1–H68. ACM Siggraph, 1993.

[BB93] G. W. Baxter and R.P. Behringer. *Pattern Formation and Complexity in Granular Flow*, chapter 3. Springer Verlag, 1993.

[BE92] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 23–90. World Scientific, Singapore, 1992.

[CR90] E. Clement and J. Rajchenbach. 133. *Europhys. Lett. 16*, 1990.

[FPP85] Michael Ian Shamos Franco P. Preparata. *Computational Geometry.* Springer-Verlag, 1985.

[Gar94] Alejandro L. Garcia. *Numerical Methods for Physics.* Prentice Hall, 1994.

[Gea71] C. William Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, Inc., 1971.

[GKV86] C. Gerthsen, H. Kneser, and H. Vogel. *Physik.* Springer Verlag, 15. edition, 1986.

[Hen68] Peter Henrici. *Discrete Variables Methods in Ordinary Differential Equations.* John Wiley & Sons, Inc., 1968.

[Her94] H.J. Herrmann. Simulating granular media on the computer. In *Third Granada Lectures in Computational Physics*, 1994.

[INB91] K.A. Semendjajew I. N. Bronstein. *Taschenbuch der Mathematik.* Verlag Harri Deutsch, Thun und Frakfurt/Main, 1991.

[KMS+88] Otto Kerner, Joseph Maurere, Jutta Steffens, Thomas Thode, and Rudolf Voller. *Vieweg Mathematik Lexikon.* Viewweg, 1988.

[Kop91] Helmut Kopka. *LaTeXEine Einführung.* Addison-Wesly, 1991.

[Kuc89] Horst Kuchling. *Taschenbuch der Physik.* Verlag Harri Deutsch, Thun und Frankfurt/Main, 12. edition, 1989.

[LC91] M.C. Lin and J.F. Canny. A fast algorithm for incremental distance calculation. In *International Conference on Robotics and Automation*, pages 1008–1014. IEEE, 1991.

[Lin93] M.C. Lin. *Efficient Collision Detection for Animation and Robotics.* PhD thesis, University of Californa at Berkeley, December 1993.

[LM93] M.C. Lin and D. Manocha. *Interference detection between curved objects for computer animation*, pages 43–57. Models and Techniques in Computer Animation. Springer-Verlag, 1993.

[Mat94] Hans-Georg Matuttis. Parallele vektoren. *mc-extra*, August 1994. Beilage zu DOS International.

[O'R93] J. O'Rourke. Computational geometry column 18. *Internat. J. Comput. Geom. Appl.*, 3:107–113, 1993. Also in SIGACT News 24:1 (1993), 20–25.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in Fortran.* Cambridge University Press, 1992.

[Sch90] F. Scheck. *Mechanik.* Springer-Verlag, 1990.

[Sed92] Sedgewick. *Algorithmen.* Addison-Wesley, 1992.

[WB93] Andrew Witkin and David Baraff. Differential equation basics. In *Course 60, An Introduction to Physically Based Modelling*, pages H1–H68. ACM Siggraph, 1993.

Danksagungen

Mein Dank gilt an dieser Stelle all jenen, die zum Enstehen vorliegender Arbeit beigetragen haben.
Bei Herrn Prof. Dr. Ingo Morgenstern möchte ich mich ganz herzlich für die Aufnahme in seiner Arbeitsgruppe bedanken. Ich danke ihm auch für die Möglichkeit, dieses interessante Thema zu bearbeiten, für sein Interesse am Verlauf der Arbeit und seine Ideen zur industriellen Anwendung.
Herrn Prof. Dr. Uwe Krey danke ich für seine Hilfsbereitschaft und die Möglichkeit der Teilnahme an seinem Lehrstuhlseminar.
Hans-Georg Matuttis sei für seine Betreuung gedankt. Hätte er mich nicht vor manchem möglichen Fehler und „unsinnigem“ Paper gewarnt, ich hätte viel mehr Unsinn gemacht.
Den Insassen des „Doktorandenzimmers“ Markus Bauernfeind, Werner Fettes und Thomas Hußlein und dem Rest der Arbeitgruppe Morgenstern sei für die gute Zusammenarbeit und die Beantwortung mancher dummer Frage beim Mittagessen, beim Kaffetrinken und sonst wann gedankt.
Robert Hildebrandt danke ich für das Korrekturlesen des Skripts, ohne ihn wäre wohl so manches Komma nicht an seinem Platz,.
Nun zu Dir, Severin: Du weißt gar nicht, für wieviel ich bei Dir in der Schuld stehe. Du hast mir bei so mancher Hürde nicht nur in diesem Studium geholfen, vom ersten Matheübungsblatt bis zur Diplomarbeit war unsere Zusammenarbeit einfach wichtig.
Danken muß ich auch meinem Vater, der das Ende meines Studiums nicht mehr miterleben durfte und ganz besonders meiner Mutter, die mir die finanzielle Seite des Studiums vom Halse hielt und immer da war, wenn ich sie gebraucht habe.

HERZLICHEN DANK!