Impulsbasierte Dynamik starrer Körper

Stephan Jost

18. Oktober 1997

Diplomarbeit nach einem Thema von Prof. Dr. Günter Hotz
in der Technischen Fakultät, Fachbereich 14 – Informatik,
der Universität des Saarlandes, Saarbrücken

Hiermit versichere ich an Eides Statt, daß ich diese Arbeit nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Saarbrücken, 1997

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Mit der fortschreitenden Entwicklung von immer leistungsfähigeren Rechnern werden auch *Simulationen* für Wissenschaft und Technik zunehmend interessanter. Simulationen versuchen, den für eine Anwendung relevanten Teil der Realität möglichst genau anzunähern und damit Prognosen für das Verhalten des simulierten Systems in der Zukunft zu stellen bzw. das Verhalten in der Vergangenheit zu rekonstruieren, um damit das System besser verstehen zu können.

Die Einsatzgebiete für Simulationen sind sehr vielseitig. Einige der bekanntesten Anwendungen sind z.B. Design und Testen neuer Produkte in der Konstruktions- und Fertigungstechnik, Simulation von Montagevorgängen, Testen z.B. von Fahrverhalten, Aerodynamik und Sicherheit (*Crashtests*) in der Automobilindustrie sowie in Luft- und Raumfahrt und die möglichst realitätsnahe Konstruktion von virtuellen Welten (*Virtual Reality*).

Der Einsatz von Simulationen erlaubt die Aufhebung der Trennung von Entwurfsphase und Produktionsplanung. Schon in der Entwurfsphase können frühzeitig Designfehler entdeckt und korrigiert werden, während die Techniker bereits eine visuelle Vorstellung von dem fertigen Produkt haben und auf leichte Montierbarkeit und Demontierbarkeit (wichtig für Herstellung und Wartung) achten können, was sich positiv auf die geometrische Auslegung und Anordnung der Bauteile auswirkt und damit später die Anwendung von teuerem Spezialwerkzeug beim Montieren bzw. Demontieren unnötig macht. Der wichtigste Vorteil an einer Simulation ist die einfache Manipulierbarkeit von Systemparametern und damit die schnelle Möglichkeit zu Änderungen und Optimierungen am Design.

Um Simulationen sinnvoll einsetzen zu können, müssen wir allerdings hohe Anforderungen an physikalische Korrektheit und Effizienz stellen. Je korrekter unser Modell ist, desto aussagekräftiger sind die Ergebnisse der Simulation. Außerdem sind Simulationen nur effizient, wenn sie in interaktiver Zeit ablaufen, d.h. wenn man die Auswirkungen von Änderungen direkt sehen und gegebenenfalls darauf reagieren kann. Damit sparen Simulationen Geld, da Prototypen erst relativ spät in der Entwurfsphase gebaut werden müssen, und vor allem Zeit, da Fehler frühzeitig erkannt und beseitigt werden können. Außerdem tragen Simulationen zum besseren Verständnis der simulierten Systeme bei.

Im folgenden wollen wir uns etwas näher mit Montagesimulationen und *Virtual Reality* beschäftigen. Am Lehrstuhl von Prof. Dr. Hotz wurde von Dr. Schömer bereits eine Montagesimulation entwickelt. Das $\mathbb{IGOR}$ -System erlaubt die Simulation von Geometrie und

Kinematik physikalischer Systeme. Bisher fehlte aber die Einbeziehung der Dynamik, d.h. die Betrachtung der Kräfte, die Bewegungen in physikalischen Systemen auslösen.

In Simulationen ist aber möglichst große Realitätsnähe sehr wichtig, da Fehler sehr teuer werden können, wenn sie erst bei der realen Anwendung festgestellt werden (z.B. Raumfahrt, Crashtests bei Autos, Produktion von fehlerhaften Bauteilen usw.). In unserem Fall wären mögliche Fehler, die bei Vernachlässigung der Dynamik auftreten können, z.B. das fehlende Umkippen eines Roboterarms, wenn sein Schwerpunkt nicht mehr über der Auflagefläche liegt, das schwerelose Schweben eines Bauteils, das von einem Greifer losgelassen wird und die fehlende Reaktion auf mögliche Kollisionen.

Die vorliegende Arbeit beschäftigt sich nun mit der Einbeziehung einer impulsbasierten Dynamik in die Simulation. Dabei kann es auch zur Interaktion zwischen den simulierten Objekten kommen, was die Erkennung solcher Kollisionen und die physikalisch möglichst korrekte Reaktion darauf notwendig macht.

Die zu simulierenden Objekte modellieren wir als starre Polyeder, d.h. sie haben eine geschlossene Oberfläche, die aus ebenen Flächen zusammengesetzt ist, und es ist keine Verformung erlaubt. Polyeder bedeuten keine starke Einschränkung, da sich jedes reale Objekt beliebig nah durch Polyeder annähern läßt.

Gliederung und Resultate

Unsere Simulation verläuft grob in vier Phasen. In der ersten Phase wird unser System *initialisiert*. Dazu zählt die Berechnung von Volumen (und damit Masse), Schwerpunkt und Trägheitsmatrix der einzelnen Objekte beim Einlesen des Szenarios. Anschließend können die Objekte noch auf die zum Startzeitpunkt gewünschte Position und Orientierung gebracht werden. Die übrigen drei Phasen werden dann nach dem Starten der Simulation in jedem Schritt innerhalb einer Schleife wiederholt. Zuerst wird die eigentliche *Bewegung* durch Integration der von den wirkenden Kräften abhängenden Bewegungsgleichungen berechnet und ausgeführt. Als nächstes testen wir mit der *Kollisionserkennung*, ob im simulierten Zeitintervall Kollisionen zwischen Objekten des Systems aufgetreten sind. Ist das der Fall, so müssen wir auf die Kollisionen physikalisch korrekt reagieren. Diese Aufgabe übernimmt in unserem Fall eine *impulsbasierte Kollisionsreaktionskomponente*. Dieser Ablauf legt die Gliederung der Arbeit in die folgenden vier Kapitel nahe.

Im *ersten Kapitel* leiten wir Formeln zur Berechnung des Volumens und der ersten beiden Momente von d-dimensionalen Polytopen her. Die vorgestellte Methode zerlegt das Polytop in d-Simplices und diese wiederum in Ursprungssimplices. Sie erwartet als Eingabe die triangulierte Oberflächenbeschreibung des Polyeders. Daher werden noch einige einfache Verfahren zur Triangulierung von Polygonen kurz erläutert.

Das *zweite Kapitel* beschreibt die Herleitung der Bewegungsgleichungen für einen starren Polyeder. Dazu beschreiben wir zunächst den Zustand des Objektes durch die Angabe von

Masse, Trägheit, Position und Orientierung sowie Dreh- und linearem Impuls. Die zeitliche Änderung dieser Parameter, beschrieben in Form von Differentialgleichungen, erlaubt es uns, daraus die Parameter des Objektes zu einem späteren Zeitpunkt zu berechnen. Zur Lösung dieser Differentialgleichungen haben wir keine geschlossenen Formeln. Wir müssen also ein numerisches Verfahren, in unserem Fall Runge-Kutta vierter Ordnung mit variabler Schrittweitenanpassung, verwenden. Um die Zahl der zu lösenden Differentialgleichungen möglichst gering zu halten, stellen wir die Orientierung nicht als Matrix dar (neun Parameter), sondern als Einheitsquaternion (vier Parameter).

Wenn sich die Objekte frei im Raum bewegen können, ist die Möglichkeit zur Interaktion zwischen den Objekten gegeben. Mit der Erkennung solcher Kontakte beschäftigt sich das *dritte Kapitel*. Unsere Kollisionserkennung erhält als Eingabe den Zustand des Systems zu zwei festen Zeitpunkten. Durch Interpolation des dazwischen liegenden Weges müssen wir nun feststellen, ob in diesem Schritt ein Kontakt aufgetreten ist. Wir führen die Kollisionserkennung auf unterster Ebene zwischen den Bestandteilen wie Ecken, Kanten und Flächen jedes Objektpaares durch. Dabei können wir die Zahl der zu testenden Kollisionstypen von neun auf vier reduzieren. Es wäre unnötig, die Kollisionstests auf unterster Ebene für alle Bestandteilpaare aller Objektpaare auch in jedem Schritt durchzuführen, wenn die betreffenden Objekte weit voneinander entfernt sind. Daher benutzen wir einige Heuristiken zur Steigerung der Effizienz. Wir berechnen die kleinsten einschließenden Kugeln unserer Objekte und führen die eigentliche Kollisionserkennung nur durch, wenn die Kugeln kollidieren. Eine Zwischenstufe wäre dabei noch das Auflösen *einer* der beiden Kugeln und das Durchführen von Kollisionstests zwischen der verbleibenden Kugel und den Bestandteilen des anderen Objektes. Abschließend werden noch einige Erweiterungsmöglichkeiten wie z.B. *Hüllkörperhierarchien* vorgestellt.

Das *vierte Kapitel* schließlich beschäftigt sich mit der Reaktion auf Kollisionen. Zu Beginn werden mehrere Ansätze miteinander verglichen und erläutert, warum wir uns für einen impulsbasierten Ansatz entschieden haben. Der folgende Abschnitt beschäftigt sich dann konkret mit der Berechnung der *nach* dem Stoß auf die an einer Kollision beteiligten Objekte wirkenden Impulse. Anschließend gehen wir näher auf die Behandlung von *mehreren gleichzeitigen* Kontakten ein. Wir entwickeln eine Methode, den Impuls für ein bewegtes Objekt, das mit beliebig vielen nichtbewegbaren kollidiert, auszurechnen. Diese Methode läßt sich auch auf beliebige Kontaktsituationen zwischen mehreren Objekten erweitern. Der verbleibende Teil des Kapitels enthält einen Ausblick auf mögliche Erweiterungen der Reaktionskomponente.

Abschließend beschreibt der *Anhang* die Benutzerschnittstelle zu der Implementierung der beschriebenen Dynamiksimulation, die auf dem IGOR -System basiert. Ein eigener Abschnitt ist dabei der Anbindung der *Spacemouse* an das System gewidmet. Sie ermöglicht die Manipulation aller sechs Freiheitsgrade im dreidimensionalen Raum gleichzeitig und erlaubt damit eine viel intuitivere Steuerung als z.B. Maus oder Tastatur.

Danksagungen

Ich möchte mich bei Herrn Prof. Dr. Günter Hotz für die Vergabe der Diplomarbeit und die interessante Themenstellung bedanken. Mein besonderer Dank gilt Herrn Dr. Elmar Schömer für die individuelle Betreuung und viele konstruktive Diskussionen und Anregungen.
Außerdem danke ich allen Korrekturlesern und den Kommilitonen und Mitarbeitern an diesem Lehrstuhl für technische und TEXnische Unterstützung sowie das hervorragende Arbeitsklima.
Insbesondere aber danke ich meinen Eltern, die mir das Studium erst ermöglicht haben.

Kapitel 1

Grundlagen

1.1 Momente eines Polyeders

Um die Wirkung von Kräften und Impulsen auf starre Körper zu untersuchen, benötigt man Informationen über deren Masse, die Lage des Schwerpunktes und die Massenverteilung innerhalb des Körpers. Die Trägheitsmatrix stellt eine Verallgemeinerung des Massenbegriffs dar und charakterisiert die Massenverteilung innerhalb von starren Körpern, sofern sie homogen ist, in ausreichendem Maße zur Bestimmung der bei Rotationen auftretenden Drehmomente und -impulse.

Die Polyeder werden in unserer Datenstruktur über ihre Berandung beschrieben, d.h. als geometrische Informationen über die Lage der Eckpunkte und Richtung der Flächennormalen und als topologische Informationen über Nachbarschaftsbeziehungen wie Zugehörigkeit der Punkte zu Kanten, Verkettung der Kanten untereinander und Zuordnung der Flächen zu den berandenden Kanten. Aus diesen Informationen können wir unter der Annahme, daß die Masse homogen verteilt ist, Volumen, Schwerpunkt und Trägheitsmatrix eines Polyeders bestimmen.

Im folgenden wird die Berechnung der Momente durch Zerlegung des Polyeders in Simplices vorgestellt. Eine alternative Methode, die z.B. in dem dieser Arbeit vorausgehenden Fortgeschrittenenpraktikum [7] verwendet wurde, wäre die Umwandlung der erhaltenen Volumenintegrale mittels des Gauß'schen Satzes in Integrale über die Oberfläche des Polyeders.

1.1.1 Masse, Schwerpunkt, Trägheitsmatrix

Allgemein lauten die Momente z.B. nach J. Craig[8] für Dimension drei:

$$\text{Masse:} \qquad m = \int_V \rho(\mathbf{x})dV \tag{1.1}$$

$$\text{Schwerpunkt:} \qquad \mathbf{S} = \frac{1}{m}\int_V \mathbf{x}\rho(\mathbf{x})dV \tag{1.2}$$

$$\text{Trägheitsmatrix:} \qquad \mathbf{I} = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{pmatrix} \tag{1.3}$$

$$\begin{array}{lll} \text{mit:} & I_{xx} = \int_V (\mathbf{y}^2+\mathbf{z}^2)\rho(\mathbf{x})dV & I_{xy} = \int_V \mathbf{xy}\rho(\mathbf{x})dV \\ & I_{yy} = \int_V (\mathbf{x}^2+\mathbf{z}^2)\rho(\mathbf{y})dV & I_{yz} = \int_V \mathbf{yz}\rho(\mathbf{y})dV \\ & I_{zz} = \int_V (\mathbf{x}^2+\mathbf{y}^2)\rho(\mathbf{z})dV & I_{xz} = \int_V \mathbf{xz}\rho(\mathbf{z})dV \\ & \text{(Massenträgheitsmomente)} & \text{(Massenträgheitsprodukte)} \end{array}$$

Unter der Annahme, daß die Masse innerhalb des Körpers homogen verteilt ist, gilt $\rho(\mathbf{x}) = \rho \quad \forall \mathbf{x} \in$ Polyeder $\mathcal{P}$. Die Formeln (1.1) bis (1.3) haben gemeinsam, daß ein Volumenintegral der Form $\int_V f(\mathbf{x})dV$ berechnet werden muß mit

$f(\mathbf{x}) = 1$	für das Volumen
$f(\mathbf{x}) = x_i$	für das erste Moment bzgl. x_i
$f(\mathbf{x}) = x_i x_j$	für das zweite Moment bzgl. x_i und x_j $\quad (i, j \in \{1, \dots, d\})$.

Die im folgenden beschriebene Methode basiert auf der von E. Schömer in der Vorlesung *Simulation mechanischer Systeme*[19] vorgestellten Methode zur Berechnung des Volumens.

1.1.2 Berechnung für einen d-dimensionalen Simplex

Definition 1.1. *Seien $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_d \in \mathbb{R}^d$ und bilden ein Rechtssystem. Dann gilt:*

$$d\text{-Simplex } \Delta(\mathbf{0}, \mathbf{a}_1, \dots, \mathbf{a}_d) := \{\mathbf{x} \mid \mathbf{x} = \sum_{i=1}^{d} \lambda_i \mathbf{a}_i, \sum_{i=1}^{d} \lambda_i \leq 1, \lambda_i \geq 0\}$$

Der (Ursprungs-)Simplex bildet die konvexe Hülle der Punkte $\mathbf{0}, \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_d$.

Satz 1.1 (Momente eines Ursprungs-Simplex). *Für das Volumen und die beiden ersten Momente eines Ursprungs-Simplex* $(\mathbf{0}, \mathbf{a}_1, \dots, \mathbf{a}_d)$ *gelten:*

$$V = \int_V 1 dV = \frac{1}{d!}\det(\mathbf{A}) \tag{1.4}$$

$$M_i = \int_V x_i dV = \frac{|\mathbf{A}_i|}{(d+1)!}\det(\mathbf{A}) \tag{1.5}$$

$$M_{ij} = \int_V x_i x_j dV = \frac{|\mathbf{A}_i||\mathbf{A}_j| + \mathbf{A}_i {\mathbf{A}_j}^T}{(d+2)!}\det(\mathbf{A}) \tag{1.6}$$

Die Matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$ *besteht aus den Spaltenvektoren* $\mathbf{a}_1, \dots, \mathbf{a}_d$*;* $\mathbf{A}_i$ *bezeichnet die* $i-$*te Zeile von Matrix* $\mathbf{A}$ *und* $|\mathbf{A}_i|$ *die Summe der Komponenten der* $i-$*ten Zeile.*

Beweis.

$$\begin{aligned}
\int_V f(\mathbf{x})dV &= \int_{x_1}\int_{x_2}\cdots\int_{x_d} f(\mathbf{x})dx_d\cdots dx_2 dx_1 \\
&= \int_0^1 \int_0^{1-\lambda_1} \cdots \int_0^{1-\sum_{i=1}^{d-1}\lambda_i} f(\mathbf{x})\, \det(\frac{\partial g}{\partial \lambda_1}, \dots, \frac{\partial g}{\partial \lambda_d})\, d\lambda_d \cdots d\lambda_1 && (1.7)\\
&= \int_0^1 \cdots \int_0^{1-\sum_{i=1}^{d-1}\lambda_i} f(\mathbf{x})\, \det(\mathbf{a}_1, \dots, \mathbf{a}_d)\, d\lambda_d \cdots d\lambda_1 && (1.8)\\
&= \det(\mathbf{A}) \int_0^1 \cdots \int_0^{1-\sum_{i=1}^{d-1}\lambda_i} f(\mathbf{x})\, d\lambda_d \cdots d\lambda_1
\end{aligned}$$

Gleichung (1.7) erhalten wir, indem wir den Substitutionssatz (siehe z.B. ENDL/LUH[9]) mit

$$g : \mathbb{R}^d \to \mathbb{R}^d, \mathbf{x} = g(\boldsymbol{\lambda}),\ \mathbf{x}, \boldsymbol{\lambda} \in \mathbb{R}^d, \text{ und } g'(\boldsymbol{\lambda}) = \sum_{i=1}^d \frac{\partial g}{\partial \lambda_i} d\lambda_i$$

anwenden. (1.8) ergibt sich wegen $g(\boldsymbol{\lambda}) = \mathbf{A}\boldsymbol{\lambda} = \sum_{i=1}^d \lambda_i \mathbf{a}_i$ und damit $\frac{\partial g}{\partial \lambda_i} = \mathbf{a}_i$. $\det(\frac{\partial g}{\partial \lambda_1}, \dots, \frac{\partial g}{\partial \lambda_d})$ ist die Determinante der zu der linearen Transformation gehörenden Jacobi-Matrix.

Wir substituieren nun $z_i := 1 - \sum_{j=1}^i \lambda_i$ für $i = 1, \dots, d-1$, d.h. $\lambda_i = z_{i-1} - z_i$ für $i = 2, \dots, d$ und $\lambda_1 = 1 - z_1$.

Fall 1: Volumen:

$$V = \det(\mathbf{A}) \int\limits_0^1 \int\limits_0^{z_1} \cdots \int\limits_0^{z_{d-1}} 1 dz_d \cdots dz_1$$

$$\text{„innerstes“ Integral: } \int\limits_0^{z_{d-1}} 1 dz_d = [z_d]_0^{z_{d-1}} = z_{d-1}$$

$$\int\limits_0^{z_{d-2}} z_{d-1} dz_{d-1} = \left[\frac{1}{2} z_{d-1}^2\right]_0^{z_{d-2}} = \frac{1}{2} {z_{d-2}}^2$$

$$\vdots$$

$$\text{„äußerstes“ Integral: } \int\limits_0^1 \frac{1}{(d-1)!} {z_1}^{d-1} dz_1 = \left[\frac{1}{d!} z_1^d\right]_0^1 = \frac{1}{d!}$$

$$\Rightarrow V = \frac{1}{d!} \det(\mathbf{A})$$

Fall 2: erstes Moment bzgl. x_i:

$$M_i = \int\limits_V x_i dV = \det(\mathbf{A}) \int\limits_0^1 \int\limits_0^{z_1} \cdots \int\limits_0^{z_{d-1}} x_i dz_d \cdots dz_1$$

$$\int\limits_0^{z_{d-1}} x_i \, dz_d = \int\limits_0^{z_{d-1}} \sum_{j=1}^{d} a_{ij} \lambda_j dz_d = \int\limits_0^{z_{d-1}} \sum_{j=2}^{d} a_{ij}(z_{j-1} - z_j) + a_{i1}(1 - z_1) \, dz_d$$

$$= \left[\Big(a_{i1}(1 - z_1) + \sum_{j=2}^{d-1} a_{ij}(z_{j-1} - z_j) + a_{id} z_{d-1} \Big) z_d - \frac{1}{2} a_{id} z_d^2 \right]_0^{z_{d-1}}$$

$$= \left[\Big(a_{i1}(1 - z_1) + \sum_{j=2}^{d-2} a_{ij}(z_{j-1} - z_j) + a_{i(d-1)}(z_{d-2} - z_{d-1}) + a_{id} z_{d-1} \Big) z_d - \frac{1}{2} a_{id} z_d^2 \right]_0^{z_{d-1}}$$

$$= \frac{1}{2} a_{id} z_d^2 - a_{i(d-1)} z_{d-1}^2 + \Big(a_{i1}(1 - z_1) + \sum_{j=2}^{d-2} a_{ij}(z_{j-1} - z_j) + a_{i(d-1)} z_{d-2} \Big) z_{d-1}$$

$$\int\limits_0^{z_{d-2}} \Big(\frac{1}{2} a_{id} z_d^2 - a_{i(d-1)} z_{d-1}^2 + \Big(a_{i1}(1 - z_1) + \sum_{j=2}^{d-2} a_{ij}(z_{j-1} - z_j) + a_{i(d-1)} z_{d-2} \Big) z_{d-1} \Big) dz_{d-1}$$

$$= \Bigg[\Big(a_{i1}(1 - z_1) + \sum_{j=2}^{d-3} a_{ij}(z_{j-1} - z_j) + a_{i(d-1)} z_{d-2} + a_{i(d-2)}(z_{d-3} - z_{d-2}) \Big) \frac{1}{2} z_{d-1}^2$$

$$+ \frac{1}{6} a_{id} z_{d-1}^3 - \frac{1}{3} a_{i(d-1)} z_{d-1}^3 \Bigg]_0^{z_{d-2}}$$

$$
\begin{aligned}
&= \frac{1}{6} a_{id} z_{d-2}^3 - \frac{1}{3} a_{i(d-1)} z_{d-2}^3 + \Big(a_{i1}(1 - z_1) + \sum_{j=2}^{d-3} a_{ij}(z_{j-1} - z_j) + a_{i(d-2)} z_{d-3} \Big) \frac{1}{2} z_{d-2}^2 \\
&\quad + \frac{1}{2} a_{i(d-1)} z_{d-2}^3 - \frac{1}{2} a_{i(d-2)} z_{d-2}^3 \\
&= \Big(\cdots \Big) \frac{1}{2} z_{d-2}^2 + \frac{1}{6} a_{id} z_{d-2}^3 + \frac{1}{6} a_{i(d-1)} z_{d-2}^3 - \frac{1}{2} a_{i(d-2)} z_{d-2}^3 \\
&\qquad \vdots \\
&\int_0^1 \Big(\sum_{j=2}^{d} \frac{1}{d!} a_{ij} z_1^d - \frac{1}{(d-1)!} a_{i1} z_1^d + a_{i1} \frac{1}{(d-1)!} z_1^{(d-1)} \Big) dz_1 \\
&= \Big[\sum_{j=2}^{d} \frac{1}{(d+1)!} a_{ij} z_1^{d+1} - \frac{1}{(d-1)!(d+1)} a_{i1} z_1^{d+1} + a_{i1} \frac{1}{d!} z_1^d \Big]_0^1 \\
&= \frac{1}{(d+1)!} \sum_{j=1}^{d} a_{ij} = \frac{1}{(d+1)!} |\mathbf{A}_i| \\
\Rightarrow M_i &= \frac{|\mathbf{A}_i|}{(d+1)!} \det(\mathbf{A})
\end{aligned}
$$

Fall 3: zweites Moment bzgl. x_i und x_j:

$$
\begin{aligned}
M_{ij} &= \int_V x_i x_j dV = \det(\mathbf{A}) \int_0^1 \int_0^{z_1} \cdots \int_0^{z_{d-1}} x_i x_j dz_d \cdots dz_1 \\
&= \det(\mathbf{A}) \int_0^1 \int_0^{z_1} \cdots \int_0^{z_{d-1}} \Big(\sum_{k=2}^{d} a_{ik}(z_{k-1} - z_k) + a_{i1}(1 - z_1) \Big) \\
&\qquad\qquad \Big(\sum_{l=2}^{d} a_{jl}(z_{l-1} - z_l) + a_{j1}(1 - z_1) \Big) dz_d \cdots dz_1
\end{aligned}
$$

Die Integrale kann man nun analog zu Fall 1 und 2 von innen nach außen auflösen und erhält dann:

$$
\begin{aligned}
M_{ij} &= \frac{\det(\mathbf{A})}{(d+2)!} \Big(\Big(\sum_{k=1}^{d} a_{ik} \sum_{l=1}^{d} a_{jl} \Big) + \sum_{k=1}^{d} a_{ik} a_{jk} \Big) \\
&= \frac{|\mathbf{A}_i||\mathbf{A}_j| + \mathbf{A}_i \mathbf{A}_j^T}{(d+2)!} \det(\mathbf{A})
\end{aligned}
$$

Die Formeln für das Volumen und die beiden ersten Momente wurden zusätzlich noch mit dem Algebrasystem Maple für den hier relevanten Fall $d = 3$ verifiziert. $\square$

LIEN/KAJIYA zeigen in [12] für den Fall $d = 3$ eine Herleitung für beliebige Funktionen f der Form

$$f(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \sum_{n_1, n_2, n_3} \mathbf{x}_1^{n_1} \mathbf{x}_2^{n_2} \mathbf{x}_3^{n_3}.$$

1.1.3 Momente für beliebige d-dimensionale Polytope

Satz 1.1 gilt bisher nur für d-Simplices im Ursprung, läßt sich aber auf beliebige d-Simplices verallgemeinern.

Wir betrachten den Simplex $\Delta(\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_d)$, wobei die Vektoren $\mathbf{a}_1 - \mathbf{a}_0, \ldots, \mathbf{a}_d - \mathbf{a}_0$ ein Rechtssystem bilden (wenn das nicht gilt, numeriert man die $\mathbf{a}_i$ entsprechend um). Man „verschiebt" also den Simplex so, daß $\mathbf{a}_0$ im Ursprung liegt. Nun kann man Satz 1.1 anwenden und für die Momente gilt:

$$M(Simplex(\mathbf{a}_0, \ldots, \mathbf{a}_d)) = m(\mathbf{A}) \det(\mathbf{a}_1 - \mathbf{a}_0, \ldots \mathbf{a}_d - \mathbf{a}_0) = m(\mathbf{A}) \det \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \mathbf{a}_0 & \mathbf{a}_1 & \cdots & \mathbf{a}_d \end{pmatrix}$$

Der Faktor $m(\mathbf{A})$ wird dabei für das jeweilige Moment entsprechend gewählt (z.B. $\frac{|\mathbf{A}_i|}{(d+1)!}$ für das erste Moment, $\mathbf{A}$ besteht dabei aus den Spaltenvektoren $\mathbf{a}_1 - \mathbf{a}_0, \ldots, \mathbf{a}_d - \mathbf{a}_0$). Es gilt: Simplex $\Delta(\mathbf{a}_0, \ldots, \mathbf{a}_d) = \bigcap_{i=0}^{d} H_i^-$ mit

$$H_i^- = \left\{ \mathbf{x} | \det \begin{pmatrix} 1 & \cdots & 1 & 1 & 1 & \cdots & 1 \\ \mathbf{a}_0 & \cdots & \mathbf{a}_{i-1} & \mathbf{x} & \mathbf{a}_{i+1} & \cdots & \mathbf{a}_d \end{pmatrix} \geq 0 \right\}$$

. Im positiven Halbraum H_i^+ befinden sich analog die Punkte $\mathbf{x}$, für die die Determinante kleiner oder gleich Null ist. Die Punkte $\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}, \mathbf{a}_{i+1}, \ldots, \mathbf{a}_d$ liegen auf der $(d-1)$-dimensionalen Hyperebene H_i und bilden dort einen $(d-1)$-dimensionalen Randsimplex von $\Delta(\mathbf{a}_0, \ldots, \mathbf{a}_d)$, während $\mathbf{a}_i$ im negativen Halbraum $H_i^- \setminus H_i$ liegt.

Definition 1.2 (Orientierter Simplex). *Ein $(d-1)$-dimensionaler Randsimplex $\Delta(\mathbf{a}_1, \ldots, \mathbf{a}_d)$ eines d-dimensionalen Polytopes $\mathcal{P}$ heißt* positiv orientiert *genau dann, wenn:*

$$\forall \varepsilon, \forall \mathbf{x} \in \Delta(\mathbf{a}_1, \ldots, \mathbf{a}_d) : (H^- \setminus H) \cap \mathcal{P} \cap \mathcal{S}_\varepsilon(\mathbf{x}) \neq \emptyset,$$

mit $H^- \backslash H = \left\{ \mathbf{x} | \det \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \mathbf{x} & \mathbf{a}_1 & \cdots & \mathbf{a}_d \end{pmatrix} > 0 \right\}$, $\mathcal{S}_\varepsilon(\mathbf{x}) =$ *d-dim. Kugel um* $\mathbf{x}$ *mit Radius* ε.

Satz 1.2 (Momente eines d-dim. Polytopes). *Der Rand $\partial\mathcal{P}$ des Polytops $\mathcal{P}$ sei vollständig in $(d-1)$-dim. positiv orientierte Simplices zerlegt. Dann gilt:*

$$M(\mathcal{P}) = \sum_{\substack{\Delta(\mathbf{a}_1, \ldots, \mathbf{a}_d) \subset \partial\mathcal{P} \\ \textit{positiv orientiert}}} m(\mathbf{A}) \det(\mathbf{a}_1, \ldots, \mathbf{a}_d)$$

Beweis. Wir zeigen den Satz zunächst für den Fall, daß $\mathcal{P}$ ein d-Simplex $\Delta(\mathbf{b}_0, \ldots, \mathbf{b}_d)$ in allgemeiner Lage ist.

Der Simplex Δ hat $(d+1)$ Randsimplices der Dimension $(d-1)$. Die Eckpunkte der Randsimplices seien dabei so numeriert, daß die Simplices positiv orientiert sind. Die Randsimplices bilden mit dem Ursprung $(d+1)$ neue, d-dimensionale Simplices der Form $\Delta_i(\mathbf{b}_0 i, \ldots, \mathbf{b}_{i-1}, \mathbf{0}, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_d)$, $(i = 0, \ldots, d)$.

Ein Integral über den Simplex Δ ist gleich der Summe der Integrale über die neu erzeugten Simplices $\Delta_0, \ldots, \Delta_d$, jeweils mit dem passenden Vorzeichen $S(\Delta_i)$ [12]:

$$M(\Delta) = \int_{\Delta} f(\mathbf{x}) dv = \sum_{\Delta_0 \ldots \Delta_d} S(\Delta_i) \int_{\Delta_i} f(\mathbf{x}) dv \tag{1.9}$$

Der Wert der Signumfunktion S ist dabei abhängig von der Lage des zugrunde liegenden Randsimplex zum Ursprung:

$$S(\Delta_i) = \begin{cases} 1 & \text{falls der Ursprung im negativen Halbraum des Randsimplex liegt} \\ 0 & \text{falls der Ursprung } \textit{auf} \text{ der Ebene durch den Randsimplex liegt} \\ -1 & \text{sonst} \end{cases}$$

Die im letzten Abschnitt hergeleiteten Formeln (1.4)–(1.6) gelten nur unter der Voraussetzung, daß $\Delta_i(0, \mathbf{a}_1, \ldots, \mathbf{a}_d)$ ein Rechtssystem bildet. Das ist gerade dann der Fall, wenn $S(\Delta_i) = 1$ gilt:

$$\det \begin{pmatrix} 1 & \cdots & 1 & 1 & 1 & \cdots & 1 \\ \mathbf{b}_0 & \cdots & \mathbf{b}_{i-1} & \mathbf{0} & \mathbf{b}_{i+1} & \cdots & \mathbf{b}_d \end{pmatrix} = (-1)^i \det \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 & \cdots & 1 \\ \mathbf{0} & \mathbf{b}_0 & \cdots & \mathbf{b}_{i-1} & \mathbf{b}_{i+1} & \cdots & \mathbf{b}_d \end{pmatrix}$$

Ansonsten hat man den Vorfaktor (-1), der sich bei vorausgesetzt positiv orientiertem Randsimplex gerade mit dem Vorzeichen $S(\Delta_i)$ aufhebt. Wir erhalten also:

$$M(\Delta) = \sum_{\Delta_0 \ldots \Delta_d} S(\Delta_i) \int_{\Delta_i} f(\mathbf{x}) dv = \sum_{\substack{\Delta(\mathbf{a}_1, \ldots, \mathbf{a}_d) \subset \partial \mathcal{P} \\ \text{positiv orientiert}}} m(\mathbf{A}) \det(\mathbf{a}_1, \ldots, \mathbf{a}_d) \tag{1.10}$$

Wenn $\mathcal{P}$ ein beliebiges Polytop ist, zerlege man $\mathcal{P}$ vollständig in d-dimensionale Simplices und berechne M als die Summe über die Momente dieser Simplices. Dabei heben sich die Summanden für alle inneren $(d-1)$-dim. Randsimplices auf, da jeder innere $(d-1)$-Simplex gerade zweimal in entgegengesetzter Orientierung vorkommt. Jede innere „Fläche“ gehört ja zu *zwei* Ursprungssimplices, die in *verschiedenen* Halbräumen bzgl. der „Fläche“ liegen. Es verbleiben also nur die Summanden für die $(d-1)$-dimensionalen Simplices des *Randes* von $\mathcal{P}$. □

Abbildung 1.1 zeigt die Berechnung von Momenten am Beispiel eines Polytops der Dimension $d = 2$. Das Polygon $\mathcal{P}(\mathbf{v}_0, \ldots, \mathbf{v}_4)$ hat als $(d-1)$-dimensionale Randsimplices

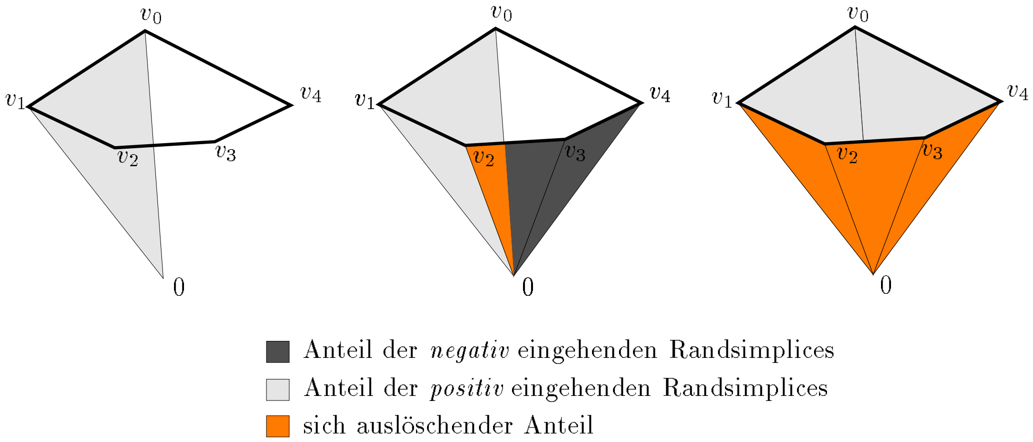

Abbildung 1.1: Momentberechnung als Summe über die Momente der aus den Randsimplices erzeugten Ursprungssimplices.

die Kanten $(\mathbf{v}_i, \mathbf{v}_{(i+1 \mod n)})$ mit $i = 0, \ldots, 4$. Diese bilden mit dem Ursprung Dreiecke, über die dann aufsummiert wird. Die Schraffierungen bezeichnen jeweils das Vorzeichen, mit dem der entsprechende Summand in die Berechnung eingeht. Die Abbildung zeigt von links nach rechts drei Stufen der Summation. Links sehen wir den Anteil für Kante $(\mathbf{v}_0, \mathbf{v}_1)$, in der Mitte für die Randsimplices $(\mathbf{v}_0, \mathbf{v}_1)$, $(\mathbf{v}_2, \mathbf{v}_3)$ und $(\mathbf{v}_3, \mathbf{v}_4)$ und ganz rechts für alle fünf Kanten.

1.2 Implementierung

1.2.1 Verwendete Datenstrukturen

Wir beschreiben unsere Objekte über die Berandung, d.h. als Informationen über die Ecken, Kanten und Flächen des Randes von Polyeder $\mathcal{P}$. Die Flächen und Kanten enthalten neben den geometrischen Informationen wie den Koordinaten von Quelle und Ziel der Kanten bzw. der Lage der Ebene durch die Fläche auch topologische Daten wie z.B. Adjazenzbeziehungen oder Umkehr-, Folge- und Vorgängerkante.

Algorithmus 1 zeigt die hier relevanten Bestandteile der für die Berechnung der Momente benutzten Klassen *Polyhedron*, *Face* und *Edge*. *Polyhedron* ist ein Abkömmling der allgemeineren Klasse *Part* und enthält zusätzlich die geometrische Beschreibung des Objektes in Form von Listen der enthaltenen Ecken, Kanten und Flächen.

Jede Fläche (Klasse *Face*) enthält neben Informationen über die Lage im Raum (Klasse *Plane*) auch einen Zeiger auf eine der sie begrenzenden Kanten. Die Kanten (Klasse *Edge*) sind flächenweise zyklisch miteinander verkettet (*Edge *pred, *succ*) und erlauben jeweils den Zugriff auf Quelle und Ziel der Kante, also auf die Koordinaten der Eckpunkte der

Algorithmus 1 Datenstruktur

```
class Polyhedron: public Part
{ public:
        list<Vertexptr> vertexlist;
        list<Edgeptr> edgelist;
        list<Faceptr> facelist;

        Polyhedron();
        Polyhedron(Polyhedron *);
        ~Polyhedron();
};
class Face
{ public:
        Edge *adj_edge;
        Plane plane;
        int id;

        Face() {};
};
class Edge
{ public:
        Vertex *source, *target;
        Face *adj_face;
        Edge *rev_edge, *pred, *succ;
        int id;

        Edge() {};
};
```

Fläche. Dabei gilt die „Rechte-Hand-Regel“: Wenn die Finger der rechten Hand der Richtung der verketteten Kanten folgen, zeigt der Daumen in Richtung der Flächennormalen (nach „außen“ bzgl. des Körpers).

1.2.2 Triangulierung

Die Momente werden für jedes Polyeder direkt beim Einlesen des Objektes berechnet. Für die Berechnung der Momente braucht man zwar die Zerlegung der Oberfläche in Dreiecke, aber für die spätere Simulation (speziell die Kollisionserkennung) ist es effizienter, möglichst wenige Flächen zu haben. Daher führen wir die Triangulierung nur auf einer Kopie des Polyeders aus. Auf dem eigentlichen Polyeder werden nur konkave Polygone in konvexe zerlegt.

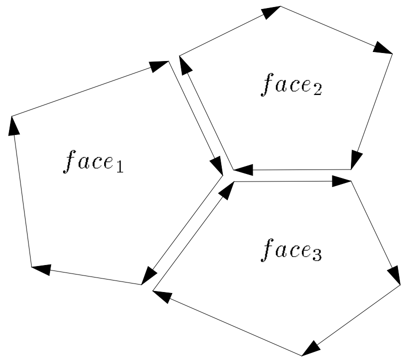

Abbildung 1.2: Auszug aus der Oberfläche eines Polyeders. Jede ungerichtete Kante ist als zwei entgegengesetzt gerichtete Kanten dargestellt, die damit eindeutig einer Fläche zuzuordnen sind. Die Flächennormalen zeigen dabei jeweils „in die Bildebene“ hinein.

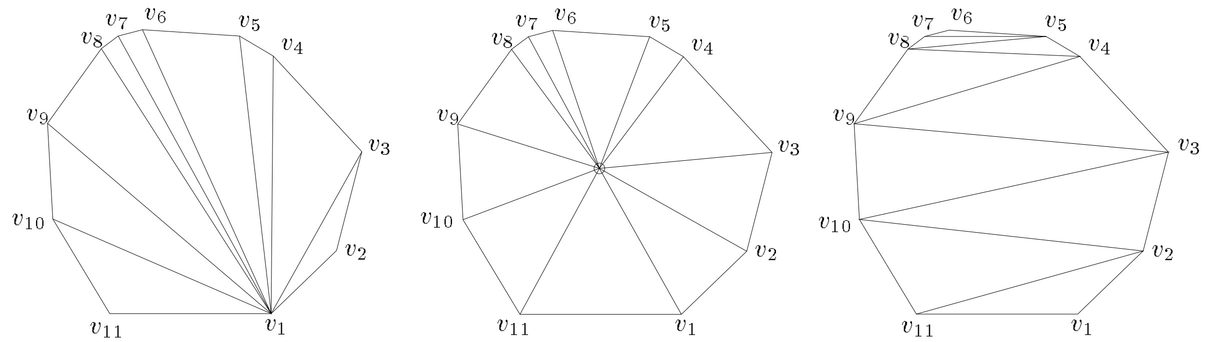

Abbildung 1.3: Beispiel für die drei vorgestellten Arten der Triangulierung.

Die Triangulierung kann auf verschiedene Arten erfolgen. Gegeben sei jeweils Polygon $\mathcal{P}(v_1, ..., v_n)$, wobei die Ecken gegen den Uhrzeigersinn durchnumeriert sind. $\mathcal{P}$ darf auch konkav sein.

- Zerlegung des Polygons in die $n-2$ Dreiecke $\mathcal{P}_i(v_1, v_i, v_{i+1}) \quad i = 2, \ldots, n-1$. Diese Methode erzeugt evtl. degenerierte (sehr lange, dünne) Dreiecke, was sich negativ auf die numerische Genauigkeit auswirkt.

- Numerisch stabiler ist es, zuerst einen neuen Punkt v_0 (*Steiner-Punkt*) als arithmetisches Mittel der Koordinaten der Eckpunkte zu berechnen und $\mathcal{P}$ dann in die Dreiecke $\mathcal{P}_i(v_0, v_i, v_{i+1}), i = 1, \ldots, n$, und $\mathcal{P}_n(v_0, v_n, v_1)$ zu zerlegen. Bei dieser Methode erhält man n Dreiecke im Vergleich zu $n-2$ Dreiecken im ersten Fall und braucht außerdem für die Triangulierung zwei Durchläufe (ein Durchlauf ist notwendig, um den „Mittelpunkt“ zu berechnen).

- Die Zerlegung von $\mathcal{P}$ in die Dreiecke $\mathcal{P}_{2i-1}(v_{n-i+1}, v_i, v_{i+1})$ mit $i = 1, \ldots, \lceil \frac{n-2}{2} \rceil$ und

$\mathcal{P}_{2i}(v_{i+1}, v_{n-i+1}, v_{n-i})$ mit $i = 1, \ldots, \lfloor \frac{n-2}{2} \rfloor$ braucht wie Methode 1 nur einen Durchlauf und erzeugt $n-2$ Dreiecke, ist aber numerisch robuster und vereinigt damit die Vorteile der beiden vorigen Methoden.

1.2.3 Momente

Die Momente selbst werden in einer Schleife über alle Flächen (Dreiecke, da wir auf einer triangulierten Kopie arbeiten) des Polyeders berechnet. Für jede Fläche verfolgen wir die Kette der anliegenden Kanten und erhalten die Eckpunkte der Fläche. Daraus berechnen wir dann nach den in den vorigen Abschnitten vorgestellten Formeln Volumen, Schwerpunkt und Trägheitstensor des Polyeders. Die Laufzeit des Algorithmus ist linear in der Anzahl der Dreiecke und damit linear in der Komplexität des ursprünglichen Polyeders.

Das vorgestellte Verfahren kann numerisch instabil werden, wenn wir das Polyeder in degenerierte Ursprungssimplices zerlegen. Das passiert, wenn der Abstand der Eckpunkte des $(d-1)$-dimensionalen Randsimplex zum Ursprung sehr groß ist im Vergleich zum Abstand der Eckpunkte untereinander (d.h. man erhält einen sehr langen, dünnen Simplex). Dieses Problem können wir aber umgehen, indem wir das Polyeder vor der Berechnung in den Ursprung verschieben.

Kapitel 2

Dynamik starrer Körper

Für die Simulation der Bewegung starrer Körper im Raum benötigt man Informationen über den *Zustand*, d.h. die Position und Orientierung der Objekte, und die zeitliche Änderung des Zustands. Die Zustandsänderung wird hervorgerufen durch die auf den Körper wirkenden Kräfte bzw. Drehmomente (z.B. Gravitation, Wind, magnetische Felder usw.). Die hier entwickelten Bewegungsgleichungen sind Differentialgleichungen erster Ordnung und beruhen auf einem Ansatz von Baraff[4], erweitert von Schömer[19]. Das Lösen der Differentialgleichungen erlaubt es uns, aus dem Zustand eines Objektes zu einem bestimmten Zeitpunkt den Zustand nach einem Intervall zu ermitteln.

Da es zu unseren Gleichungen keine geschlossenen Lösungsformeln gibt, benutzen wir einen numerischen Differentialgleichungslöser. Wir wählen das in [17] vorgestellte *Runge-Kutta-Verfahren* mit variabler Schrittweitenkontrolle. Es erhält als Eingabe die Zustandsvariablen zu einem Startzeitpunkt sowie Informationen über die Berechnung der ersten Ableitungen und liefert den Zustand zum Endzeitpunkt.

2.1 Herleitung der Bewegungsgleichungen

2.1.1 Position und Orientierung

Um den für uns relevanten Zustand eines Massenpunktes im dreidimensionalen Raum zum Zeitpunkt t eindeutig zu bestimmen, genügen drei Werte, nämlich x-, y- und z-Koordinate bezüglich eines Referenzsystems (z.B. des Weltkoordinatensystems). Für einen beliebigen starren Körper ist die Bestimmung des aktuellen Zustandes etwas schwieriger. Da ein beliebiger Körper eine räumliche Ausdehnung hat und sich um sich selbst drehen kann, benötigt man neben der *Position* eines Referenzpunktes noch die aktuelle *Orientierung*. Als Referenzpunkt bietet sich der Schwerpunkt an, da seine Position unter Rotation stabil bleibt (Rotationen werden bei uns um die Hauptachsen des Körpers ausgeführt, die sich im

Schwerpunkt schneiden). Die Orientierung eines Objektes kann man als die Transformation der Punkte bzgl. des körpereigenen Koordinatensystems ins Weltkoordinatensystem auffassen, wenn beide Systeme den gleichen Ursprung haben. Das bedeutet nichts anderes als die Transformation der Koordinatenachsen des Körpers selbst ins Weltkoordinatensystem. Diese Transformation stellt einen Basiswechsel dar und läßt sich daher als Orthonormalmatrix darstellen unter der Annahme, daß die beiden Basen aus orthonormalen Vektoren bestehen. Transformationen mit Orthonormalmatrizen haben die Eigenschaft, daß Längen und Winkel erhalten bleiben (*starre Transformation*). Die Spaltenvektoren der Basen sind daher die normierten Koordinatenachsen der betreffenden Systeme $\{A\}$ und $\{B\}$. Die Rotationsmatrix selbst besteht aus den Koordinatenachsen von System $\{B\}$ relativ zu System $\{A\}$. Notation:

$$ {}^{A}_{B}\mathbf{R} = \left[{}^{A}\mathbf{x}_{B}\,{}^{A}\mathbf{y}_{B}\,{}^{A}\mathbf{z}_{B}\right] $$

Ein weiterer Vorteil der Orthonormalmatrizen ist die leichte Berechenbarkeit der Inversen. Es gelten $\mathbf{R}^{-1} = \mathbf{R}^{T}$ und $\det(\mathbf{R}) = 1$.

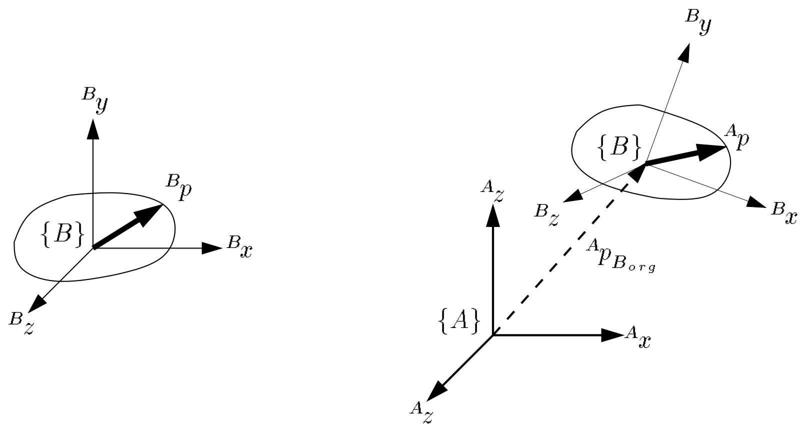

Abbildung 2.1: Punkt $\mathbf{p}$ bzgl. des körpereigenen Koordinatensystems $\{B\}$ und bzgl. des Systems $\{A\}$.

Der Ursprung von System $\{B\}$ kann nach der Änderung der Orientierung auch aus dem Ursprung von System $\{A\}$ verschoben werden. Abbildung 2.1 zeigt links den Punkt $\mathbf{p}$, dargestellt bzgl. des körpereigenen Koordinatensystems $\{B\}$, das seinen Ursprung im Schwerpunkt des Objektes hat, und rechts den gleichen Punkt bzgl. des Systems $\{A\}$. Man erhält ${}^{A}\mathbf{p}$, indem man ${}^{B}\mathbf{p}$ mit ${}^{A}_{B}\mathbf{R}$ um den Ursprung von $\{B\}$ dreht und dann um den Vektor ${}^{A}\mathbf{p}_{B_{org}}$ verschiebt. ${}^{A}\mathbf{p}_{B_{org}}$ ist die Position des Ursprungs von System $\{B\}$, dargestellt in Koordinaten bzgl. System $\{A\}$.

$$ {}^{A}\mathbf{p} = {}^{A}_{B}\mathbf{R}\,{}^{B}\mathbf{p} + {}^{A}\mathbf{p}_{B_{org}} \tag{2.1} $$

Auf diese Weise werden auch Bewegungen beschrieben. Die Bewegung eines Objektes (d.h. aller Punkte des Objektes) ist ja nichts anderes als eine zeitabhängige Änderung von

Position und Orientierung des körpereigenen Koordinatensystems relativ zum Weltkoordinatensystem.

2.1.2 Lineare Geschwindigkeit und Winkelgeschwindigkeit

Als nächstes benötigen wir die zeitliche Änderung des Zustandes. Die Positionsänderung des Objekts läßt sich beschreiben als die Positionsänderung des Schwerpunktes $\mathbf{c}$, also als seine *lineare Geschwindigkeit.*

$$\dot{\mathbf{c}}(t) = \frac{d\mathbf{c}(t)}{dt} = \mathbf{v}(t)$$

Satz 2.1 (Satz von Euler). *Jede Orientierung $\{B\}$ relativ zu $\{A\}$, beschrieben durch eine Transformationsmatrix ${}^A_B\mathbf{R}$, kann man durch eine Rotation um eine feste Achse $\mathbf{r}$ und einen Winkel ϕ erhalten.*

Die Drehung eines Punktes $\mathbf{p}(t)$ läßt sich also mit Hilfe eines Vektors $\boldsymbol{\omega}(t)$ beschreiben. Die *Länge* von $\boldsymbol{\omega}(t)$ gibt die Anzahl der Umdrehungen pro Zeiteinheit an (also die *Winkelgeschwindigkeit*) und die *Richtung* von $\boldsymbol{\omega}(t)$ die Richtung der Rotationsachse durch den Schwerpunkt des Objektes. Dadurch bleibt die Position des Schwerpunktes bei der Drehung stabil.

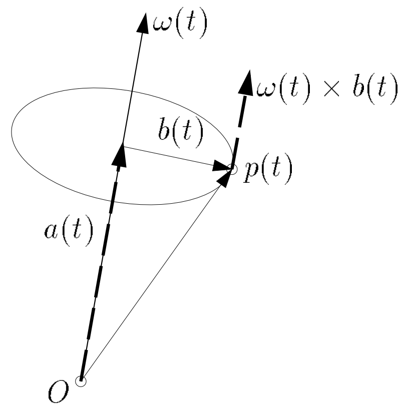

Abbildung 2.2: Winkelgeschwindigkeit $\boldsymbol{\omega}(t)$ eines Punktes $\mathbf{p}(t)$

Wie Abbildung 2.2 zeigt, ergibt sich die zeitliche Änderung von $\mathbf{p}(t)$ durch $\boldsymbol{\omega}(t) \times \mathbf{b}(t)$. $\mathbf{p}(t)$ „läuft“ durch die Drehung entlang des Kreises, immer senkrecht zur Drehachse $\boldsymbol{\omega}(t)$ und zum Radius $\mathbf{b}(t)$. Die Geschwindigkeit beträgt $|\boldsymbol{\omega}(t) \times \mathbf{b}(t)| = |\boldsymbol{\omega}(t)||\mathbf{b}(t)|$, da $\boldsymbol{\omega}(t)$ und $\mathbf{b}(t)$ senkrecht zueinander stehen. Wir erhalten also

$$\begin{aligned}\dot{\mathbf{p}}(t) &= \boldsymbol{\omega}(t) \times \mathbf{b}(t) = \boldsymbol{\omega}(t) \times \mathbf{b}(t) + \boldsymbol{\omega}(t) \times \mathbf{a}(t) \\ &= \boldsymbol{\omega}(t) \times (\mathbf{b}(t) + \mathbf{a}(t)) = \boldsymbol{\omega}(t) \times \mathbf{p}(t), \qquad (2.2)\end{aligned}$$

mit $\boldsymbol{\omega}(t) \times \mathbf{a}(t) = 0$. Dies gilt für jeden beliebigen Punkt des Körpers, insbesondere für seine Koordinatenachsen (d.h. die Spaltenvektoren der Matrix $\mathbf{R}$). Wie man leicht nachrechnen kann, hat die Linksmultiplikation eines Vektors $\mathbf{p}$ mit der schiefsymmetrischen Matrix $\omega^\times$ den gleichen Effekt wie die Bildung des Kreuzproduktes $\boldsymbol{\omega} \times \mathbf{p}$.

$$\boldsymbol{\omega}^\times \mathbf{p} = \begin{pmatrix} 0 & -\boldsymbol{\omega}_z & \boldsymbol{\omega}_y \\ \boldsymbol{\omega}_z & 0 & -\boldsymbol{\omega}_x \\ -\boldsymbol{\omega}_y & \boldsymbol{\omega}_x & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}_x \\ \mathbf{p}_y \\ \mathbf{p}_z \end{pmatrix} = \begin{pmatrix} \boldsymbol{\omega}_x \\ \boldsymbol{\omega}_y \\ \boldsymbol{\omega}_z \end{pmatrix} \times \begin{pmatrix} \mathbf{p}_x \\ \mathbf{p}_y \\ \mathbf{p}_z \end{pmatrix} = \boldsymbol{\omega} \times \mathbf{p}.$$

Damit vereinfacht sich die Darstellung der Ableitung von $\mathbf{R}$ zu

$$\dot{\mathbf{R}} = \boldsymbol{\omega}^\times \mathbf{R}, \tag{2.3}$$

da man nicht mehr wie beim Kreuzprodukt drei getrennte Gleichungen für die Koordinatenachsen braucht.

Bei Translation *und* Rotation erhalten wir die Geschwindigkeit eines Punktes ${}^A\mathbf{p}$ durch Differenzieren von Gleichung (2.1), wobei System $\{A\}$ das körpereigene System zum Zeitpunkt t ist und $\{B\}$ das körpereigene System zum Zeitpunkt 0 (das man so wählt, daß der Schwerpunkt im Ursprung liegt und die Körperachsen an den Achsen des Weltkoordinatensystems ausgerichtet sind). $\mathbf{p}(0)$ bezeichnet also die Position von Punkt $\mathbf{p}$ relativ zum Schwerpunkt des Körpers zum Zeitpunkt 0.

$$\begin{aligned} \dot{\mathbf{p}}(t) &= \dot{\mathbf{R}}\mathbf{p}(0) + \dot{\mathbf{c}}(t) = \boldsymbol{\omega}^\times \mathbf{R}\mathbf{p}(0) + \mathbf{v}(t) = \boldsymbol{\omega}^\times(\mathbf{R}\mathbf{p}(0) + \mathbf{c}(t) - \mathbf{c}(t)) + \mathbf{v}(t) \\ &= \boldsymbol{\omega}^\times((\mathbf{R}\mathbf{p}(0) + \mathbf{c}(t)) - \mathbf{c}(t)) + \mathbf{v}(t) = \boldsymbol{\omega}^\times(\mathbf{p}(t) - \mathbf{c}(t)) + \mathbf{v}(t) \end{aligned} \tag{2.4}$$

2.1.3 Quaternionen

Eine andere Möglichkeit zur Darstellung von Rotationen als die 3×3-Matrix $\mathbf{R}$ sind *Einheitsquaternionen*. Ein *Hamiltonsches Quaternion* $\mathfrak{q}$ ist ein Quadrupel, bestehend aus dem Skalar $\mathbf{q}_s$ und dem Vektor $\mathbf{q}$ mit

$$\mathbf{q}_s = \cos\frac{\phi}{2} \text{ und } \mathbf{q} = \mathbf{k}\sin\frac{\phi}{2}.$$

$\mathbf{k}$ ist dabei die Drehachse und ϕ der Drehwinkel. Nach Satz 2.1 genügen Achse und Drehwinkel, um eine Rotation zu definieren.

Satz 2.2. *Die Abbildung* $\mathrm{rot}(\mathbf{k}, \phi) : \mathbb{R}^3 \to \mathbb{R}^3$ *mit* $\mathbf{p} \mapsto \left(\mathfrak{q} \cdot (0, \mathbf{p})^T \cdot \mathfrak{q}^*/\mathfrak{q}\mathfrak{q}^*\right)$ *und* $\mathfrak{q}(\mathbf{k}, \phi) := (\cos\frac{\phi}{2}, \mathbf{k}\sin\frac{\phi}{2})$ *entspricht einer Rotation des Vektors* $\mathbf{p}$ *um Achse* $\mathbf{k}$ *um den Winkel* ϕ*. Mit* $|\mathfrak{q}| = 1$ *gilt hierbei* $\mathfrak{q}\mathfrak{q}^* = 1$*.* $\mathfrak{q}^*$ *ist das zu* $\mathfrak{q}$ *konjugierte Element, das man durch Negieren der Vektorkomponente erhält.*

Ein Beweis zu diesem Satz befindet sich z.B. in der Diplomarbeit von M. Welsch[23].

Rechenregeln:

Addition	$\lambda(\mathfrak{p}+\mathfrak{q}) = \lambda\begin{pmatrix}\mathbf{p}_s\\ \mathbf{p}\end{pmatrix} + \lambda\begin{pmatrix}\mathbf{q}_s\\ \mathbf{q}\end{pmatrix} = \begin{pmatrix}\lambda(\mathbf{p}_s+\mathbf{q}_s)\\ \lambda(\mathbf{p}+\mathbf{q})\end{pmatrix}$
Multiplikation	$\mathfrak{p}\mathfrak{q} = \begin{pmatrix}\mathbf{p}_s\\ \mathbf{p}\end{pmatrix}\begin{pmatrix}\mathbf{q}_s\\ \mathbf{q}\end{pmatrix} = \begin{pmatrix}\mathbf{p}_s\mathbf{q}_s - \mathbf{p}^T\mathbf{q}\\ \mathbf{p}_s\mathbf{q}+\mathbf{q}_s\mathbf{p}+\mathbf{p}\times\mathbf{q}\end{pmatrix}$
neutrales Element der Multiplikation	$\mathfrak{e} = \begin{pmatrix}1\\ \mathbf{0}\end{pmatrix}$
konjugiertes Element von $\mathfrak{q} = \begin{pmatrix}\mathbf{q}_s\\ \mathbf{q}\end{pmatrix}$	$\mathfrak{q}^* = \begin{pmatrix}\mathbf{q}_s\\ -\mathbf{q}\end{pmatrix}$
inverses Element von $\mathfrak{q} = \begin{pmatrix}\mathbf{q}_s\\ \mathbf{q}\end{pmatrix}$	$\mathfrak{q}^{-1} = \frac{\mathfrak{q}^*}{\lvert\mathfrak{q}\rvert}$
Rotation mit Quaternionen	$\begin{pmatrix}0\\ \mathbf{p}(t)\end{pmatrix} = \frac{\mathfrak{q}\,(0,\mathbf{p}(0))^T\,\mathfrak{q}^*}{\mathfrak{q}\mathfrak{q}^*}$
Hintereinanderausführung	$\begin{pmatrix}0\\ \mathbf{p}(t)\end{pmatrix} = \mathfrak{q}_2\mathfrak{q}_1\begin{pmatrix}0\\ \mathbf{p}(0)\end{pmatrix}\mathfrak{q}_1^*\mathfrak{q}_2^*$

Die Quaternionen bilden mit der üblichen Vektoraddition im $\mathbb{R}^4$ und der oben definierten Multiplikation einen nicht kommutativen Ring mit Eins. Alle Elemente außer $(0,\mathbf{0})$, dem neutralen Element der Addition, sind invertierbar. Ein solches algebraisches Objekt nennt man auch *Schiefkörper*.

Für *Einheits*quaternionen kommt noch die Einheitsbedingung hinzu.

$$\mathfrak{q}\mathfrak{q}^* = \mathbf{q}_s^2 + \mathbf{q}_x^2 + \mathbf{q}_y^2 + \mathbf{q}_z^2 = 1 \tag{2.5}$$

Die Einheitsquaternionen bilden unter der oben definierten Multiplikation eine Gruppe.

Bei der Verwendung von Quaternionen treten allerdings Probleme auf.

- Die gleiche Rotation kann durch zwei verschiedene Quaternionen dargestellt werden. Eine Rotation um Achse $\mathbf{k}$ um den Winkel ϕ entspricht der Rotation um Achse $-\mathbf{k}$ um den Winkel $-\phi$. Welche von beiden Varianten gewählt wird, hängt von der Implementierung ab und macht für die Simulation keinen Unterschied, da die effektive Rotation für beide Fälle identisch ist.

- Umgekehrt werden z.B. die Rotation um 0^o und die Rotation um 360^o beide durch das Quaternion $\mathfrak{q} = (1,\mathbf{0})^T$ dargestellt. Die Orientierung des Körpers ist in beiden Fällen zwar gleich, aber die Rotation unterscheidet sich in der Simulation (z.B. für die Kollisionserkennung). Dieses Problem können wir aber umgehen, indem wir die Schrittweite bei der Rotation genügend klein wählen (dadurch werden beide Rotationen unterscheidbar).

Einheitsquaternionen haben gegenüber Matrizen mehrere Vorteile. Zum einen hat man nur noch vier Parameter statt neun, d.h. der Differentialgleichungslöser muß pro Simulationsschritt fünf Gleichungen weniger lösen. Der Hauptvorteil liegt aber in der numerischen Stabilität. Bei jedem Simulationsschritt können Rundungsfehler auftreten, d.h. die Matrix ist möglicherweise nicht mehr orthonormal und damit keine exakte Rotationsmatrix mehr. Für den rotierten Körper kann das eine Verzerrung in sich selbst bedeuten. Für das Quaternion bedeuten Rundungsfehler evtl. den Verlust der Einheitslänge. Dies läßt sich aber leicht durch Normalisieren des Quaternions korrigieren.

Algorithmus 2 Matrix → Quaternion

$\mathbf{q}_s^2 = \frac{1}{4}(1 + r_{xx} + r_{yy} + r_{zz})$
if $\mathbf{q}_s^2 > \epsilon$ **then**
  $\mathbf{q}_s = \sqrt{\mathbf{q}_s^2}$
  $\mathbf{q}_x = (r_{yz} - r_{zy})/4\mathbf{q}_s$
  $\mathbf{q}_y = (r_{zx} - r_{xz})/4\mathbf{q}_s$
  $\mathbf{q}_z = (r_{xy} - r_{yx})/4\mathbf{q}_s$
else
  $\mathbf{q}_s = 0$
  $\mathbf{q}_x^2 = -\frac{1}{2}(r_{yy} + r_{zz})$
  if $\mathbf{q}_x^2 > \epsilon$ **then**
    $\mathbf{q}_x = \sqrt{\mathbf{q}_x^2}$
    $\mathbf{q}_y = r_{xy}/2\mathbf{q}_x$
    $\mathbf{q}_z = r_{xz}/2\mathbf{q}_x$
  else
    $\mathbf{q}_x = 0$
    $\mathbf{q}_y^2 = \frac{1}{2}(1 - r_{zz})$
    if $\mathbf{q}_y^2 > \epsilon$ **then**
      $\mathbf{q}_y = \sqrt{\mathbf{q}_y^2}$
      $\mathbf{q}_z = r_{yz}/2\mathbf{q}_y$
    else
      $\mathbf{q}_y = 0$
      $\mathbf{q}_z = 1$
    end if
  end if
end if

Da die Rotationsmatrix aber weiterhin noch benötigt wird (z.B. für die Berechnung der aktuellen Trägheitsmatrix $\mathbf{I}(t)$), müssen wir Quaternionen in Matrizen umrechnen können und umgekehrt. Für Quaternionen in Abhängigkeit von $\mathbf{R}$ gilt:

$$\mathbf{q}_s = \frac{1}{2}\sqrt{1 + r_{xx} + r_{yy} + r_{zz}}, \quad \mathbf{q}_x = \frac{r_{zy} - r_{yz}}{4\mathbf{q}_s}, \quad \mathbf{q}_y = \frac{r_{xz} - r_{zx}}{4\mathbf{q}_s}, \quad \mathbf{q}_z = \frac{r_{yx} - r_{xy}}{4\mathbf{q}_s}$$

Diese Art der Berechnung macht natürlich keinen Sinn, wenn der Skalaranteil $\mathbf{q}_s = 0$ wird (bei $\phi = 180^o$). Wir gehen dann nach der von SHOEMAKE[21] vorgeschlagenen Sonderfallbehandlung vor. Ist unser berechnetes $\mathbf{q}_s$ nahe bei Null, so nullieren wir $\mathbf{q}_s$ und drücken dann $\mathbf{q}_x^2$ in Abhängigkeit von $\mathbf{q}_y$ und $\mathbf{q}_z$ aus. Die Einträge der Matrix $\mathbf{R}$ vereinfachen sich in Gleichung (2.6) durch das Verschwinden von $\mathbf{q}_s$, so daß wir die Beziehung

$$\mathbf{q}_x^2 = -\frac{1}{2}(r_{yy} + r_{zz})$$

erhalten. Gilt nun $\mathbf{q}_x^2 > \varepsilon$, so ergeben sich $\mathbf{q}_y$ und $\mathbf{q}_z$ in einfacher Weise in Abhängigkeit von $\mathbf{q}_x$, wobei $\mathbf{q}_x$ im Nenner steht. Ist $\mathbf{q}_x^2 \leq \varepsilon$, so nullieren wir auch $\mathbf{q}_x$ und können $\mathbf{q}_y$ in Abhängigkeit von r_{zz} ausdrücken. Ist auch $\mathbf{q}_y$ nahe bei Null, so setzen wir $\mathbf{q}_s$, $\mathbf{q}_x$ und $\mathbf{q}_y$ auf Null und $\mathbf{q}_z$ wird Eins. Etwas ausführlicher wird die dargestellte Sonderfallbehandlung von Algorithmus 2 beschrieben.

Umgekehrt ergibt sich für $\mathbf{R}$, parametrisiert mit $\mathfrak{q}$:

$$\mathbf{R} = \begin{pmatrix} 2\mathbf{q}_s^2 + 2\mathbf{q}_x^2 - 1 & 2(\mathbf{q}_y\mathbf{q}_x - \mathbf{q}_z\mathbf{q}_s) & 2(\mathbf{q}_z\mathbf{q}_x + \mathbf{q}_y\mathbf{q}_s) \\ 2(\mathbf{q}_x\mathbf{q}_y + \mathbf{q}_z\mathbf{q}_s) & 2\mathbf{q}_s^2 + 2\mathbf{q}_y^2 - 1 & 2(\mathbf{q}_z\mathbf{q}_y - \mathbf{q}_x\mathbf{q}_s) \\ 2(\mathbf{q}_x\mathbf{q}_z - \mathbf{q}_y\mathbf{q}_s) & 2(\mathbf{q}_y\mathbf{q}_z + \mathbf{q}_x\mathbf{q}_s) & 2\mathbf{q}_s^2 + 2\mathbf{q}_z^2 - 1 \end{pmatrix} \tag{2.6}$$

Während der Simulation variiert die aktuelle Orientierung mit der Zeit. Wenn wir Gleichung (2.6) nach t differenzieren, erhalten wir:

$$\dot{\mathbf{R}} = 2 \begin{pmatrix} 2(\mathbf{q}_s\dot{\mathbf{q}}_s + \mathbf{q}_x\dot{\mathbf{q}}_x) & \dot{\mathbf{q}}_y\mathbf{q}_x + \mathbf{q}_y\dot{\mathbf{q}}_x - \dot{\mathbf{q}}_z\mathbf{q}_s - \mathbf{q}_z\dot{\mathbf{q}}_s & \dot{\mathbf{q}}_z\mathbf{q}_x + \mathbf{q}_z\dot{\mathbf{q}}_x + \dot{\mathbf{q}}_y\mathbf{q}_s + \mathbf{q}_y\dot{\mathbf{q}}_s \\ \dot{\mathbf{q}}_x\mathbf{q}_z + \mathbf{q}_x\dot{\mathbf{q}}_y + \dot{\mathbf{q}}_z\mathbf{q}_s + \mathbf{q}_z\dot{\mathbf{q}}_s & 2(\mathbf{q}_s\dot{\mathbf{q}}_s + \mathbf{q}_y\dot{\mathbf{q}}_y) & \dot{\mathbf{q}}_z\mathbf{q}_y + \mathbf{q}_z\dot{\mathbf{q}}_y - \dot{\mathbf{q}}_x\mathbf{q}_s - \mathbf{q}_x\dot{\mathbf{q}}_s \\ \dot{\mathbf{q}}_x\mathbf{q}_z + \mathbf{q}_x\dot{\mathbf{q}}_y - \dot{\mathbf{q}}_z\mathbf{q}_s - \mathbf{q}_z\dot{\mathbf{q}}_s & \dot{\mathbf{q}}_y\mathbf{q}_z + \mathbf{q}_y\dot{\mathbf{q}}_z + \dot{\mathbf{q}}_x\mathbf{q}_s + \mathbf{q}_x\dot{\mathbf{q}}_s & 2(\mathbf{q}_s\dot{\mathbf{q}}_s + \mathbf{q}_z\dot{\mathbf{q}}_z) \end{pmatrix} \tag{2.7}$$

Nun wollen wir eine Differentialgleichung für die zeitliche Änderung von Rotationen in Quaternionennotation herleiten. Ausgangsbasis ist Gleichung (2.3), aufgelöst nach $\boldsymbol{\omega}^\times$.

$$\boldsymbol{\omega}^\times = \mathbf{R}^T\dot{\mathbf{R}} \tag{2.8}$$

Wenn wir nun (2.6) und (2.7) einsetzen, erhalten wir für $\boldsymbol{\omega}_x, \boldsymbol{\omega}_y$ und $\boldsymbol{\omega}_z$ komplexe Ausdrücke, die sich aber mit Beziehung (2.5) und deren Ableitung

$$\mathbf{q}_s\dot{\mathbf{q}}_s + \mathbf{q}_x\dot{\mathbf{q}}_x + \mathbf{q}_y\dot{\mathbf{q}}_y + \mathbf{q}_z\dot{\mathbf{q}}_z = 0$$

stark vereinfachen lassen und schließlich

$$\begin{pmatrix} 0 \\ \boldsymbol{\omega}_x \\ \boldsymbol{\omega}_y \\ \boldsymbol{\omega}_z \end{pmatrix} = 2 \begin{pmatrix} \mathbf{q}_s & \mathbf{q}_x & \mathbf{q}_y & \mathbf{q}_z \\ -\mathbf{q}_x & \mathbf{q}_s & \mathbf{q}_z & -\mathbf{q}_y \\ -\mathbf{q}_y & -\mathbf{q}_z & \mathbf{q}_s & \mathbf{q}_x \\ -\mathbf{q}_z & \mathbf{q}_y & -\mathbf{q}_x & \mathbf{q}_s \end{pmatrix} \begin{pmatrix} \dot{\mathbf{q}}_s \\ \dot{\mathbf{q}}_x \\ \dot{\mathbf{q}}_y \\ \dot{\mathbf{q}}_z \end{pmatrix}$$

ergeben. Die erste Zeile der Matrix ist dabei einfach die Ableitung der Einheitsbedingung (2.5). Wie man leicht nachrechnen kann, ist die Determinante der Matrix immer -1, d.h.

die Matrix ist invertierbar und wir können die Gleichung nach $\dot{\mathfrak{q}}$ auflösen.

$$\dot{\mathfrak{q}} = \begin{pmatrix} \dot{\mathbf{q}}_s \\ \dot{\mathbf{q}}_x \\ \dot{\mathbf{q}}_y \\ \dot{\mathbf{q}}_z \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \mathbf{q}_s & -\mathbf{q}_x & -\mathbf{q}_y & -\mathbf{q}_z \\ \mathbf{q}_x & \mathbf{q}_s & -\mathbf{q}_z & \mathbf{q}_y \\ \mathbf{q}_y & \mathbf{q}_z & \mathbf{q}_s & -\mathbf{q}_x \\ \mathbf{q}_z & -\mathbf{q}_y & \mathbf{q}_x & \mathbf{q}_s \end{pmatrix} \begin{pmatrix} 0 \\ \boldsymbol{\omega}_x \\ \boldsymbol{\omega}_y \\ \boldsymbol{\omega}_z \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 \\ \boldsymbol{\omega}_x \\ \boldsymbol{\omega}_y \\ \boldsymbol{\omega}_z \end{pmatrix} \begin{pmatrix} \mathbf{q}_s \\ \mathbf{q}_x \\ \mathbf{q}_y \\ \mathbf{q}_z \end{pmatrix} = \frac{1}{2}\mathfrak{w}\mathfrak{q} \tag{2.9}$$

Damit haben wir Differentialgleichung (2.3), ausgedrückt mit Quaternionen statt Rotationsmatrizen.

Eine genauere Betrachtung der Darstellungen von Rotationen mit den Umrechnungen zwischen den Beschreibungen findet sich z.B. bei Shoemake[21].

2.1.4 Kraft und Drehmoment

Bisher haben wir uns nur mit dem Zustand und der Art der Bewegung von starren Körpern beschäftigt, nicht aber mit den Kräften (z.B. Gravitationskraft usw.), die die betrachteten Bewegungen verursachen. Wir müssen nun auch noch die jeweils zu einem bestimmten Zeitpunkt t auf einen Punkt $\mathbf{x}_i$ einwirkenden Kräfte mit einbeziehen.

Definition 2.1. *Zwei entgegengesetzt gleiche Kräfte* $\mathbf{F}$ *und* $-\mathbf{F}$*, die an zwei Punkten eines Körpers mit dem Abstandsvektor* $\mathbf{r}$ *angreifen, bezeichnet man als* Kräftepaar.

Unabhängig vom Bezugspunkt bewirkt das Kräftepaar das *Drehmoment*

$$\mathbf{D} = \mathbf{r} \times \mathbf{F}.$$

An einem starren Körper angreifende Kräfte können immer durch eine an einem beliebigen Punkt angreifende Kraft und ein Kräftepaar ersetzt werden (vgl. z.B. Greiner[10]).

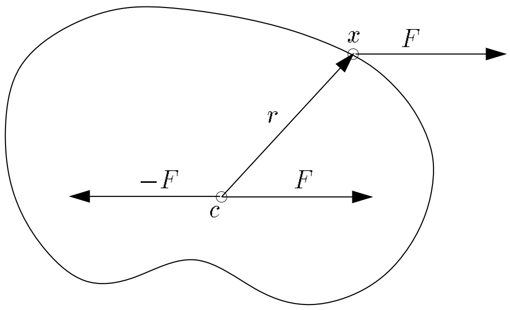

Abbildung 2.3: In Punkt $\mathbf{x}$ angreifende Kraft $\mathbf{F}$, ersetzt durch ein Kräftepaar und die im Schwerpunkt angreifende Kraft $\mathbf{F}$.

Abbildung 2.3 zeigt die Kraft $\mathbf{F}$, die am Punkt $\mathbf{x}$ angreift. Wir ändern nichts an der Gesamtkraft, wenn wir zusätzlich im Schwerpunkt die Kräfte $\mathbf{F}$ und $-\mathbf{F}$ wirken lassen.

Die in Punkt $\mathbf{x}$ angreifende Kraft und die im Schwerpunkt angreifende Kraft $-\mathbf{F}$ bilden über den Vektor $\mathbf{r}$ ein Kräftepaar nach obiger Definition. Übrig bleibt dann noch Kraft $\mathbf{F}$, die im Schwerpunkt angreift.

Wirken nun mehrere Kräfte gleichzeitig, so gilt für die resultierende linear auf den Ursprung wirkende Kraft

$$\mathbf{F} = \sum_i \mathbf{F}_i$$

und für das insgesamt auf den Körper wirkende Drehmoment

$$\mathbf{D} = \sum_i \mathbf{r}_i \times \mathbf{F}_i + \sum_i \mathbf{D}_i.$$

Der Vektor $\mathbf{r}$ ist dabei die Differenz aus dem Ortsvektor von $\mathbf{x}$ und den Koordinaten des Schwerpunktes $\mathbf{c}$. Die $\mathbf{D}_i$ sind direkt im Schwerpunkt ansetzende Drehmomente. Solche Drehmomente könnten z.B. durch einen Motor erzeugt werden.

Die Gravitation läßt sich nun einfach simulieren als eine auf den Schwerpunkt wirkende externe lineare Kraft $\mathbf{F} = m\mathbf{g}$ mit Drehmoment $\mathbf{D} = 0$, wobei $\mathbf{g}$ in Richtung der negativen $\mathbf{z}$-Achse zeigt.

2.1.5 Linearer Impuls und Drehimpuls

Für den *linearen Impuls* eines Objektes gilt

$$\mathbf{p}(t) = m\mathbf{v}(t). \tag{2.10}$$

Durch Differenzieren erhalten wir mit *Newtons Gleichung*

$$\dot{\mathbf{p}}(t) = m\mathbf{a}(t) = \mathbf{F}(t). \tag{2.11}$$

Analog zu Gleichung (2.10) definieren wir für den *Drehimpuls*

$$\mathbf{L}(t) = \mathbf{I}(t)\boldsymbol{\omega}(t). \tag{2.12}$$

Der Trägheitstensor beschreibt die Massenverteilung innerhalb des Körpers relativ zum Schwerpunkt und wirkt als „Skalierungsfaktor" für die Beziehung zwischen Drehimpuls und Winkelgeschwindigkeit. $\mathbf{I}(t)$ ist von der aktuellen Orientierung abhängig. Die Hauptträgheitsachsen bleiben bei starren Körpern bzgl. des körpereigenen Systems konstant und drehen sich mit dem Objekt im Weltkoordinatensystem. Wir erhalten also den jeweils zum Zeitpunkt t aktuellen Trägheitstensor durch

$$\mathbf{I}(t) = \mathbf{R}(t)\mathbf{I}_0\mathbf{R}(t)^T, \tag{2.13}$$

wobei $\mathbf{I}_0$ die Trägheit im Ausgangszustand beschreibt. Wenn wir Gleichung (2.13) differenzieren, erhalten wir eine ähnliche Beziehung für das Drehmoment wie Gleichung (2.11) für lineare Kräfte.

$$\begin{aligned}\dot{\mathbf{L}}(t) &= \mathbf{I}(t)\dot{\boldsymbol{\omega}}(t) + \dot{\mathbf{I}}(t)\boldsymbol{\omega}(t) = \mathbf{I}(t)\dot{\boldsymbol{\omega}}(t) + \frac{d}{dt}\left(\mathbf{R}(t)\mathbf{I}_0\mathbf{R}(t)^T\right)\boldsymbol{\omega}(t)\\ &= \mathbf{I}(t)\dot{\boldsymbol{\omega}}(t) + \left(\dot{\mathbf{R}}(t)\mathbf{I}_0\mathbf{R}(t)^T + \mathbf{R}(t)\mathbf{I}_0\dot{\mathbf{R}}(t)^T\right)\boldsymbol{\omega}(t)\\ &= \mathbf{I}(t)\dot{\boldsymbol{\omega}}(t) + \boldsymbol{\omega}^\times(t)\mathbf{R}(t)\mathbf{I}_0\mathbf{R}(t)^T\boldsymbol{\omega}(t) - \mathbf{R}(t)\mathbf{I}_0\mathbf{R}(t)^T\boldsymbol{\omega}^\times(t)\boldsymbol{\omega}(t)\\ &= \mathbf{I}(t)\dot{\boldsymbol{\omega}}(t) + \boldsymbol{\omega}^\times(t)\mathbf{R}(t)\mathbf{I}_0\mathbf{R}(t)^T\boldsymbol{\omega}(t) = \mathbf{D}(t)\end{aligned} \tag{2.14}$$

Die letzte Gleichheit ergibt sich aus *Eulers Gleichung*.

2.1.6 Zusammenfassung

Der Zustand eines Körpers kann zu jedem Zeitpunkt t durch folgende Parameter bestimmt werden.

$$\begin{aligned} m &= \text{Masse des Körpers}\\ \mathbf{I}_0 &= \text{Trägheitsmatrix im Ausgangszustand}\\ \mathbf{c} &= \text{Position des Schwerpunktes}\\ \mathfrak{q} &= \text{Orientierung in Quaternionenschreibweise}\\ \mathbf{p} &= \text{linearer Impuls}\\ \mathbf{L} &= \text{Drehimpuls}\end{aligned}$$

Masse und Trägheitsmatrix (bzgl. des Körpersystems) werden einmal zu Beginn der Simulation berechnet und bleiben dann konstant. Damit verbleiben dreizehn variable Parameter, die pro Simulationsschritt neu bestimmt werden müssen.

Außerdem verwenden wir noch einige daraus abgeleitete Hilfsvariablen.

$$\begin{aligned} \mathbf{R} &= \frac{(\mathbf{q}_s^2 - \mathbf{q}^2)\mathbf{E} + 2\mathbf{q}\mathbf{q}^T + 2\mathbf{q}_s\mathbf{q}^\times}{\mathbf{q}_s^2 + \mathbf{q}^2} && \text{Rotationsmatrix}\\ \mathbf{v} &= \frac{1}{m}\mathbf{p} && \text{lineare Geschwindigkeit}\\ \boldsymbol{\omega} &= \mathbf{R}\mathbf{I}_0^{-1}\mathbf{R}^T\mathbf{L} && \text{Winkelgeschwindigkeit}\end{aligned}$$

$\mathbf{F}$: gesamte auf den Körper einwirkende Kraft

$\mathbf{D}$: Gesamtdrehmoment, das auf den Körper einwirkt

Folgende Differentialgleichungen definieren die zeitliche Änderung dieser Parameter

$$\begin{aligned} \dot{\mathbf{c}} &= \mathbf{v} & \dot{\mathfrak{q}} &= \frac{1}{2}\mathfrak{w}\mathfrak{q}\\ \dot{\mathbf{p}} &= \mathbf{F} & \dot{\mathbf{L}} &= \mathbf{D}\end{aligned}$$

und erlauben damit die Bestimmung des Zustandes zum Zeitpunkt $t + \Delta t$. Wir müssen also mit dem im folgenden Abschnitt beschriebenen numerischen Verfahren pro Simulationsschritt dreizehn Differentialgleichungen lösen.

2.2 Runge-Kutta-Verfahren

Da wir für die in den letzten Abschnitten hergeleiteten Differentialgleichungen keine geschlossenen Lösungsformeln haben, müssen wir auf numerische Verfahren zur Integration *gewöhnlicher Differentialgleichungen* zurückgreifen. Gewöhnliche Differentialgleichungen oder ODE (von *ordinary differential equations*) enthalten im Unterschied zu *partiellen Differentialgleichungen* nur Ableitungen nach *einer* Variablen. Unsere Simulation gehört zur Klasse der *Anfangswertprobleme*, d.h. gegeben sind die Funktionswerte an einem Startpunkt, und wir wollen daraus die Funktionswerte an einem Endpunkt bzw. einer Reihe von Punkten nach bestimmten Intervallen errechnen. Wir verwenden hier die *Runge-Kutta-Methode*, die z.B. in *Numerical Recipes in C* [17] genauer vorgestellt wird. Sie kombiniert die Informationen aus mehreren *Euler-Schritten*, um eine Taylorreihe höherer Ordnung zur Bestimmung des Funktionswertes nach einem bestimmten Intervall herzuleiten. Runge-Kutta hat gegenüber anderen Verfahren wie beispielsweise der *Richardson-Extrapolation* oder *predictor-corrector Methoden* den Vorteil, daß es ein sehr robustes Verfahren darstellt, das auch bei weniger „glatt“ verlaufenden Funktionen meist recht gute Ergebnisse liefert und außerdem relativ leicht zu implementieren ist. Allerdings sind die anderen Verfahren oft effizienter. Dieses Problem läßt sich aber durch die Verwendung von variablen Schrittweiten mildern. Bei glattem Verlauf kann man die Schrittweite bei vorgegebener Genauigkeit vergrößern und bei stark schwankenden Funktionswerten bei Bedarf wieder verringern.

2.2.1 ODE erster Ordnung

Numerische Integrationsverfahren behandeln nur Gleichungen erster Ordnung. Daher müssen Differentialgleichungen höherer Ordnung zuerst in eine Menge von Differentialgleichungen erster Ordnung umgewandelt werden. Dies erreichen wir durch die Einführung neuer Variablen. Die neuen Variablen sind dabei einfach Ableitungen voneinander bzw. von der ursprünglichen Variable.

Beispiel 2.1. Die DGL zweiter Ordnung

$$\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} = r(x)$$

kann ersetzt werden durch zwei DGL erster Ordnung:

$$\frac{dy}{dx} = z(x)$$
$$\frac{dz}{dx} = r(x) - q(x)z(x)$$

□

Wir erhalten also eine Menge von N Differentialgleichungen erster Ordnung der Form:

$$\frac{dy_i(x)}{dx} = f_i(x, y_1, \ldots, y_N), \text{ für } i = 1, \ldots, N$$

Die rechten Seiten der Gleichungen (d.h. die f_i), sind nicht nur von der Variablen x alleine abhängig, sondern auch von den jeweiligen Funktionswerten y_j.

Es kann dabei vorkommen, daß die neuen Hilfsvariablen in ihrem Funktionswert stark schwanken, obwohl die ursprünglichen Variablen einen glatten Verlauf zeigen. Um solches singuläres Verhalten (und damit große Rundungsfehler) zu mildern, kann es nützlich sein, die Hilfsvariablen durch zusätzliche Faktoren bei der Definition abzuändern.

In unserem Fall sind solche Maßnahmen nicht notwendig, da nur Differentialgleichungen ersten Grades vorkommen.

2.2.2 Eulers Methode

Die grundsätzliche Idee ist, in der Ableitungsinformation $\frac{dy(x)}{dx} = y'(x) = f(x, y)$ Zähler und Nenner durch diskrete Schritte Δy und Δx darzustellen und dann mit Δx zu multiplizieren. Damit erhalten wir die Änderung des Funktionswertes bei „Schrittweite“ Δx.

$$y_{n+1} = y_n + \Delta y = y_n + f(x_n, y_n)\Delta x$$

In Abbildung 2.4 bezeichnet $y_i'(x_j)$ jeweils die i-te Berechnung der Ableitung im j-ten Simulationsschritt bzgl. Startpunkt x_j. Der exakte Funktionswert wäre

$$y_{n+1} = y_n + hf(x_n, y_n) + O(h^2), \quad \text{mit } h = \Delta x = x_{n+1} - x_n.$$

Man hat also nach Eulers Methode die Funktion in erster Ordnung angenähert und damit einen Fehlerterm zweiter Ordnung in h. Diese Methode ist im Vergleich zu anderen Methoden weder sehr genau noch sonderlich stabil, aber sie bildet die Basis für alle weiteren Ansätze zur Lösung von Anfangswertproblemen.

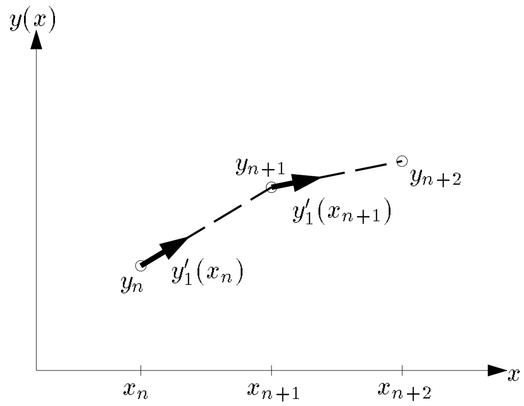

Abbildung 2.4: Eulers Methode. Der neue Funktionswert wird mit der Auswertung der Ableitung am Startpunkt gefunden.

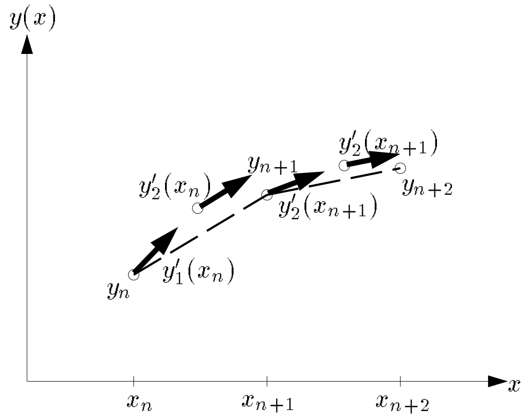

Abbildung 2.5: Mittelpunktmethode

2.2.3 Mittelpunktmethode

Bei der Mittelpunktmethode oder auch Runge-Kutta zweiter Ordnung machen wir mit der Ableitungsinformation an der Stelle x_n zuerst einen „versuchsweisen" Euler-Schritt zur Mitte des Intervalls. An dem so erhaltenen Punkt bilden wir wieder die Ableitung und machen mit dem erhaltenen Wert einen „echten" Schritt von x_n nach x_{n+1}.

$$\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \frac{1}{2}\,h, y_n + \frac{1}{2}\,k_1) \\
y_{n+1} &= y_n + k_2 + O(h^3) = y_n + hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hf(x_n, y_n)) + O(h^3)
\end{aligned}$$

Damit haben wir den Fehlerterm zweiter Ordnung eliminiert und erreichen so Genauigkeit zweiter Ordnung.

2.2.4 Runge-Kutta vierter Ordnung

Auf obige Weise lassen sich die Fehlerterme beliebiger Ordnung eliminieren. In der Praxis kommt am häufigsten das Runge-Kutta-Verfahren vierter Ordnung vor. Hier benutzen wir für den „echten" Schritt eine mit unterschiedlichen Koeffizienten gewichtete Kombination aus den verschiedenen Ableitungsinformationen.

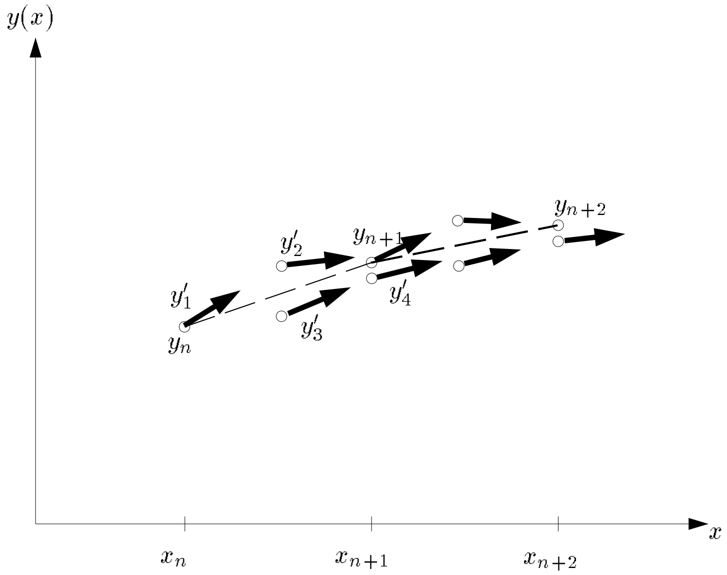

Abbildung 2.6: Runge-Kutta vierter Ordnung

Zuerst werten wir die Ableitung am Startpunkt aus und machen mit der so gewonnenen Information einen *halben* Schritt zur Intervallmitte. Die hier erhaltene Ableitungsinformation verwenden wir für einen zweiten Schritt zur Intervallmitte, wo wir ein weiteres Mal die Ableitung bestimmen. Mit dieser Information machen wir nun einen *ganzen* Schritt. An dem so erhaltenen Punkt errechnen wir ein viertes Mal die Ableitung.

$$\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \frac{h}{2}h, y_n + \frac{k_1}{2}) \\
k_3 &= hf(x_n + \frac{h}{2}h, y_n + \frac{k_2}{2}) \\
k_4 &= hf(x_n + h, y_n + k_3)
\end{aligned}$$

Der endgültige Funktionswert ergibt sich dann aus einer Kombination der so gewonnen Ableitungsinformationen. Am häufigsten finden sich in der Praxis folgende Koeffizienten:

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)$$

Nach demselben Schema könnte man auch noch Fehlerterme höherer Ordnung eliminieren, aber der zusätzliche Aufwand für das Verfahren nimmt stärker zu als der Gewinn in der Genauigkeit rechtfertigt. Es ist in diesem Fall effizienter, Runge-Kutta vierter Ordnung mit kleinerer Schrittweite zu verwenden.

2.2.5 Variable Schrittweite

Um eine akzeptable Genauigkeit zu erhalten, müssen wir bei stark schwankenden Funktionswerten eine sehr kleine Schrittweite benutzen. Es wäre aber ineffizient, diese Schrittweite für die gesamte Simulation — also auch „glatte“ Funktionsabschnitte — zu verwenden. Wir passen die Schrittweite daher an den Funktionsverlauf an und wählen die Schrittweite immer so groß wie möglich, um noch die gewünschte Genauigkeit zu erzielen. Damit haben wir möglichst große Effizienz, aber auch die vorher spezifizierte Genauigkeit in schwierigen Funktionsabschnitten. Wir brauchen daher eine Abschätzung für den gemachten Fehler, um zu entscheiden, ob wir die Schrittweite verringern müssen oder sicher erhöhen können.

Eine weit verbreitete Technik ist die Schrittverdoppelung (*step doubling*), d.h. wir machen jeden Schritt zweimal (einmal als vollständigen Schritt und einmal als zwei halbe Schritte) und vergleichen die erhaltenen Ergebnisse. Das bedeutet bei Runge-Kutta vierter Ordnung einen Aufwand von elf Ableitungsauswertungen (ein ganzer Schritt und zwei halbe, wobei die erste Auswertung für beide gleich ist) im Vergleich zu acht Auswertungen (zwei Schritte mit halber Schrittweite). Was bekommen wir für diesen Mehraufwand ?

Seien $y(x+2h)$ das exakte Ergebnis für den Schritt von x nach $x+2h$ und y_1 (ein ganzer Schritt) und y_2 (zwei halbe Schritte) die berechneten Ergebnisse. Dann erhalten wir:

$$y(x+2h) = y_1 + (2h)^5\Phi + O(h^6) + \cdots \tag{2.15}$$

$$y(x+2h) = y_2 + 2(h^5)\Phi + O(h^6) + \cdots \tag{2.16}$$

Dabei liegt Φ nach der Taylorreihenentwicklung in der Größenordnung von $\frac{y^{(5)}(x)}{5!}$. Der Faktor $(2h)^5$ kommt von der Schrittweite $2h$ und der Faktor $2(h^5)$ von zwei Schritten der Größe h, wobei die Schrittweite nach obiger Fehlerabschätzung jeweils in fünfter Potenz in die Gleichung eingeht.
Ein Maß für den gemachten Fehler ist die Differenz aus den beiden errechneten Ergebnissen.

$$\Delta = y_2 - y_1 = (32-2)h^5\Phi \tag{2.17}$$

In (2.15) eingesetzt erhalten wir dann

$$y(x+2h) = y_2 + \frac{\Delta}{15} + O(h^6),$$

also einen Term mit Genauigkeit fünfter Ordnung. Da wir nun den Zusammenhang unserer Fehlerabschätzung mit dem exakten und errechneten Ergebnis kennen, brauchen wir noch einen Zusammenhang zwischen der Schrittweite h und unserer Fehlerabschätzung. Die gesuchte Schrittweite h_0 ist gerade die Schrittweite, bei der wir unsere gewünschte Genauigkeit Δ_0 erhalten würden. In Gleichung (2.17) geht h in fünfter Potenz ein. Wir erhalten also aus dem Verhältnis von gewünschter und tatsächlicher Genauigkeit Δ_1 bei Schrittweite h_1 die neue Schrittweite h_0 als

$$h_0 = h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{0.2} . \tag{2.18}$$

Für $\Delta_0 > \Delta_1$ sagt uns die Gleichung, um welchen Faktor wir die Schrittweite für den *nächsten* Schritt sicher erhöhen können und für $\Delta_0 < \Delta_1$, mit wie stark verringerter Schrittweite wir den aktuellen Schritt *wiederholen* müssen.

Δ_0 ist aber keine einzelne Zahl, sondern ein Vektor aus den gewünschten Genauigkeiten für alle Differentialgleichungen unserer Menge. Die neue Schrittweite muß sich also nach dem „schlimmsten“ Fall richten, also so klein gewählt sein, daß für *jede* Gleichung die gewünschte Genauigkeit erreicht wird.

Die gewünschte Genauigkeit kann dabei für jede beteiligte Gleichung auf verschiedene Arten definiert werden:

1. absolut spezifizierte Genauigkeit (z.B. auf die fünfte Nachkommastelle)

2. Genauigkeit relativ zum aktuellen Funktionswert (z.B. sinnvoll bei stark in der Größenordnung variierenden Funktionswerten)

3. Genauigkeit in Abhängigkeit von Schranken (z.B. nützlich bei zwischen Extremwerten oszillierenden Funktionen)

4. Genauigkeit in Abhängigkeit von der Schrittweite. Δ_0 gibt den Fehler pro Schritt an, aber die Fehler können sich auch über mehrere Schritte aufaddieren, d.h. bei kleiner Schrittweite hat man mehr Schritte und damit potentiell einen größeren Fehler.

Wenn nun aber Δ_0 mit h skaliert ist, stimmt der Exponent 0.2 nicht mehr. Wenn die Schrittweite auf einen neuen Wert h_1 gesenkt wird und Δ_1 das neue h_1 als Faktor enthält, kürzt sich ein h_1 in Gleichung (2.18) weg und der richtige Exponent wäre 0.25. Da wir im voraus nicht wissen, ob die Genauigkeit mit der Schrittweise skaliert ist und der Unterschied zwischen beiden Exponenten nicht zu groß ist, bietet sich folgende praktikable Lösung an:

$$\begin{aligned} h_0 &= S h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{0.20} && \text{für} \Delta_0 \geq \Delta_1 \\ h_1 &= S h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{0.25} && \text{für} \Delta_0 < \Delta_1 \end{aligned}$$

Der Sicherheitsfaktor S ist etwas kleiner als 1.0. Er sorgt dafür, daß die Schrittweite immer etwas kleiner gewählt wird als unbedingt nötig, um zu verhindern, daß die Fehlerabschätzung — durch mögliche Rundungsfehler verursacht — eine weitere Verringerung der Schrittweite erzwingt.

Das Runge-Kutta-Verfahren ist zwar sehr robust, aber in degenerierten Fällen kann es vorkommen, daß die Fehlerabschätzung immer schief geht. Der Simulationsschritt wird mit immer kleinerer Schrittweite wiederholt, führt aber nie zum Ziel. Deshalb sollte man verhindern, daß die Schrittweite in den Bereich der Rechengenauigkeit sinkt und sich damit Rundungsfehler zu stark auswirken.

2.3 Implementierung

2.3.1 Modifikation der Genauigkeitsanforderung

Pro Simulationsschritt lösen wir dreizehn Differentialgleichungen für jedes beteiligte Objekt. Um den dabei gemachten Fehler möglichst gering zu halten, passen wir die Genauigkeitsanforderungen für die Fehlerabschätzung der Schrittweitenkontrolle individuell an die jeweilige Funktion an (vgl. Abschnitt 2.2).

Durch die eventuell vorkommenden Kollisionen und die damit verbundenen starken Schwankungen der Funktionswerte läßt sich der Verlauf der Funktionen, die den Zustand der Objekte und die darauf wirkenden Impulse beschreiben, kaum vorhersagen. Darüber hinaus können wir bei linearem Impuls, Drehimpuls und den Koordinaten des Schwerpunkts auch keine genauen Aussagen über den Wertebereich oder obere/untere Schranken machen. Daher bietet es sich an, die gewünschte Genauigkeit hier relativ zum jeweiligen Funktionswert zu spezifizieren.

Die Orientierung dagegen wird als Einheitsquaternion dargestellt. Die jeweiligen Komponenten liegen also zwischen -1 und $+1$. Eine Skalierung in Abhängigkeit des Funktionswertes führt vor allem in der Nullnähe zu einem Scheitern der Fehlerabschätzung und damit zu einer weiteren Verkleinerung der Schrittweite. Hier ist es günstiger, die Genauigkeitsanforderung in Abhängigkeit der Schranken (hier betragsmäßig 1) anzugeben, d.h. absolute Genauigkeit statt relativer.

2.3.2 Bewegungsfunktion

Algorithmus 3 zeigt grob den Ablauf eines Bewegungsschrittes. Während der Simulation können wir ein Objekt (*active object*) direkt über die SpaceMouse manipulieren. Die Auslenkung der SpaceMouse simuliert dabei eine zu diesem Zeitpunkt auf das Objekt wirkende externe Kraft. Außerdem lassen sich die Knöpfe der SpaceMouse mit verschiedenen Funktionen belegen. In unserem Fall sind das z.B. Ende der Simulation, Nullierung des

Algorithmus 3 Ablauf eines Bewegungsschrittes

for all $poly \in moved\_list$ **do**
  if $poly$ is active **then**
    read_spacemouse($button, \Delta force, \Delta torque$)
    if $button$ = ENDE **then**
      end of simulation
    else if $button$ = RESET_ANG_MOM **then**
      reset angular momentum to starting value
    else if $button$ = RESET_IMPULSE **then**
      reset impulse to starting value
    else
      $poly \rightarrow force += \Delta force$
      $poly \rightarrow torque += \Delta torque$
    end if
  end if
  $poly \rightarrow q(0) = poly \rightarrow q(1)$
  $poly \rightarrow centre(0) = poly \rightarrow centre(1)$
  $poly \rightarrow L(0) = poly \rightarrow L(1)$
  $poly \rightarrow p(0) = poly \rightarrow p(1)$

  do_runge_kutta_step($q(1), centre(1), L(1), p(1)$)
      {solve the DGL to get the actual state}
  transform($poly, O, o$)
      {do the actual movement, O and o are the new orientation and barycentre}

  if $poly$ is active **and** $button$ = NONE **then**
    $poly \rightarrow force -= \Delta force$
    $poly \rightarrow torque -= \Delta torque$
  end if
end for

Drehimpulses oder Bewegung des Schwerpunktes zurück in den Ursprung und Nullierung des wirkenden Impulses.

Für die Kollisionserkennung (vgl. nächstes Kapitel) benötigen wir die Daten über Zustand und wirkende Impulse *vor* und *nach* dem aktuellen Bewegungsschritt. Daher kopieren wir diese Daten, bevor wir den Differentialgleichungslöser aufrufen und damit die aktuelle Position und Orientierung sowie linearen Impuls und Drehimpuls erhalten.

Dann führen wir die daraus resultierende Bewegung aus (*transform*) und restaurieren Kraft und Drehmoment für das aktive Objekt. Das ist notwendig, weil die durch das Eingabegerät simulierten Kräfte jeweils nur für einen Zeit*punkt* wirken und keine ständig vorhandenen Kräfte, wie z.B. die Gravitation, darstellen sollen. Da wir mit der Hand nicht so genau steuern können, daß wir eine Auslenkung nur für einen Zeit*punkt*, nicht aber für einen

---

Algorithmus 4 eigentliche Bewegungsfunktion

---

transform($poly, O, o$)
{
  for all $v \in poly \rightarrow vertexlist$ **do**
    $v.v(0) = v.v(1)$
    $v.v(1) = O * (v_0 - centre_0) + o$
  end for
  $m.v(0) = m.v(1)$
  $m.v(1) = O * (m_0 - centre_0) + o$

  for all $f \in poly \rightarrow facelist$ **do**
    $f.normal(0) = f.normal(1)$
    $f.n(0) = f.n(1)$
    $f.normal(1) = T * f.normal_0$
    $f.n(1) = f.n_0 + f.n(1) * t - f.normal_0 * centre_0$
  end for
}

Zeit*raum* haben, erhalten wir in der Praxis Auslenkungen für die Simulations-Zeitpunkte innerhalb eines Intervalls. Die Auslenkung erreicht den Vollausschlag im Mittelteil des Intervalls und flacht zu den Randbereichen hin ab, was in etwa dem realistischen Verlauf einer an einem Objekt extern angreifenden Kraft entsprechen würde. Die fehlende Stetigkeit in unserem Kraftverlauf ist durch die Simulation zu diskreten Zeitpunkten bedingt.

Algorithmus 4 zeigt den vereinfachten Ablauf der Bewegungsfunktion *transform*. Die Matrix $\mathbf{O}$ und der Vektor $\mathbf{o}$ bezeichnen die aktuelle Orientierung und die aktuelle Lage des Schwerpunktes des Objektes. Wir gehen die Liste aller Eckpunkte durch und sichern die Koordinaten aus dem vorigen Schritt in $v.\mathbf{v}(0)$, da sie noch zur Kollisionserkennung benötigt werden. Dann errechnen wir mittels Orientierung und Position des Schwerpunktes die aktuellen Koordinaten aus den Koordinaten zum Startzeitpunkt. Dieselbe Prozedur führen wir auch mit den Koordinaten des Mittelpunktes der kleinsten einschließenden Kugel durch. Anschließend aktualisieren wir noch für alle Flächen des Polyeders die Normale und den Abstand zum Ursprung. Sie bestimmen durch die Gleichung

$$\mathbf{normal}(t)^T\mathbf{x}(t) = \mathbf{n}(t)$$

die Ebene, die durch die Fläche führt.

Kapitel 3

Kollisionserkennung

Aufgabe der Kollisionserkennung in der Dynamiksimulation ist es, sämtliche Kollisionen der Objekte untereinander festzustellen, damit Durchdringung durch entsprechende Gegenmaßnahmen verhindert werden kann (siehe Kollisionsreaktion in Kapitel 4). Außerdem soll die Kollisionserkennungskomponente die notwendigen Daten wie Kontaktpunkt, -ebene und -zeit an die Reaktionskomponente liefern.
Als erstes werden wir uns im folgenden mit der allgemeinen Vorgehensweise bei Bewegungen, die durch Differentialgleichungen beschrieben sind, beschäftigen. Anschließend geht es um die Kollisionserkennung auf unterster Ebene, also lokal zwischen den Bestandteilen (*features*) wie Ecken, Kanten und Flächen von Polyedern untereinander. Dann folgen einige einfache Heuristiken zur Verbesserung der Effizienz mit einem Ausblick auf weitere Möglichkeiten zur Effizienzsteigerung.

3.1 Allgemeine Vorgehensweise

Exakte Kollisionserkennung betrachtet die Bewegung von Objekten normalerweise als Überlagerung von Translationen und Rotationen, d.h. die Bahnkurven lassen sich in geschlossener Form als Trajektorien angeben. Somit kann man exakt Zeitpunkt und Ort des frühesten Kontaktes zwischen den beteiligten Objekten berechnen.
In unserem Fall aber ist die Bewegung durch Differentialgleichungen beschrieben, die wir nur zu diskreten Zeitpunkten lösen. Wir erhalten also den Zustand des Systems nur zu festen Zeitpunkten, nicht aber die zugehörigen Bahnkurven.

3.1.1 Geometrische Kohärenz und Interpolation der Bahnkurven

Das numerische Lösen der Differentialgleichungen liefert uns den Zustand des Systems nur an festen Stützstellen, nicht aber auf dem dazwischen liegenden Weg. Unter Ausnutzen der *geometrischen Kohärenz* können wir den Weg aber bei hinreichend kleinem Abstand der Stützstellen interpolieren, ohne das Ergebnis zu stark zu verfälschen. Geometrische Kohärenz bedeutet hier, daß sich die Lage der Objekte zueinander in einem Zeitschritt nur unwesentlich ändert. Man kann die Bahnkurve also als Folge von Interpolationspolynomen hinreichend genau annähern.

Interpolationsproblem (z.B. nach BRONSTEIN[6]) Im Intervall $[a, b]$ seien die *Stützstellen* $a \leq x_0 < \cdots < x_n \leq b$ gegeben, zusammen mit den zugehörigen Funktionswerten y_i für $i = 0, \ldots, n$.
Gesucht wird das Interpolationspolynom $I_n(x)$ von Grad $\leq n$ mit $I_n(x_i) = y_i$ für $0 \leq i \leq n$. Sind alle x_i paarweise verschieden, so gibt es immer *genau ein* solches Polynom.

Form von Lagrange $I_n(x) = \sum_{i=0}^{n} y_i L_i(x)$ mit $L_i = \prod_{i \neq j} \frac{x-x_j}{x_i-x_j}$

Interpolationsfehler Falls f hinreichend oft differenzierbar ist, gilt:
$f(x) - I_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1) \cdots (x - x_n), \quad \xi \in [a, b]$ ist dabei abhängig von x.

Für unsere Simulation wählen wir aus verschiedenen Gründen die lineare Interpolation:

- Für den Fall $n = 1$ läßt sich das Interpolationspolynom sehr schnell berechnen.
- Bei der linearen Interpolation bleibt der Speicherplatzaufwand relativ gering. Für einen Simulationsschritt müssen wir uns in der verwendeten Datenstruktur den gesamten Zustand wie Lage, Orientierung und wirkende Kräfte an den $n+1$ Stützstellen merken. Für den linearen Fall müssen wir also zwei „Kopien“ sämtlicher Zustandsvariablen verwalten und in *jedem* Schritt aktualisieren, was für höhere Ordnung entsprechenden Mehraufwand bedeutet.
- Der lineare Fall ist einfach zu implementieren.
- Um die gleiche Genauigkeit zu erzielen, ist es erheblich aufwendiger, Interpolation höherer Ordnung zu benutzen als für den linearen Fall die Schrittweite entsprechend zu verringern.

3.1.2 Distanzberechnung und Durchdringungstoleranz

Da wir die Kollisionstests nicht mit den als Überlagerung von Translationen und Rotationen beschriebenen Bahnkurven der Objekte durchführen können, benutzen wir eine Distanzberechnungsstrategie. Wir berechnen pro Simulationsschritt jeweils den kürzesten Abstand zwischen je zwei Objekten. Wenn der berechnete Abstand negativ ist (Durchdringung!)

und betragsmäßig eine vorher bestimmte *Durchdringungstoleranz* ε überschreitet, meldet die Kollisionserkennung einen Kontakt und berechnet die Kontaktdaten. Die Toleranz ungleich Null ist nötig, um zu verhindern, daß die Kollisionserkennung (durch Rundungsfehler verursacht) evtl. fälschlicherweise einen Kontakt meldet, wenn zwei Objekte sich dicht aneinander vorbei bewegen (vgl. Abbildung 3.1), aber *nicht* berühren.
Die Toleranz ε sollte einerseits möglichst klein gewählt werden, um jede *tatsächliche* Kollision auch wirklich zu erkennen, andererseits aber auch in der Größenordnung deutlich über eventuellen Rundungsfehlern liegen, um fälschlich „erkannte" Kontakte zu vermeiden. Die exakte Wahl ist abhängig vom geometrischen Aufbau und der Art und Geschwindigkeit der Bewegung der beteiligten Objekte. In der Praxis hat sich ein Wert im Bereich von 10^{-7} bis 10^{-5} als guter Kompromiß erwiesen.

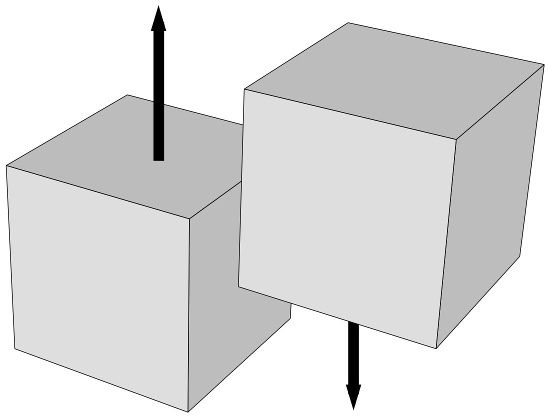

Abbildung 3.1: Beispiel für zwei Polyeder, die sich jeweils in Pfeilrichtung dicht aneinander vorbei bewegen, sich aber *nicht* berühren.

3.1.3 Lokalität und Ausschließen von Möglichkeiten

Wir betrachten Kollisionserkennung lokal, d.h. eine Kollision zweier Polyeder ist gleichbedeutend mit der Kollision einer Ecke, Kante oder Fläche des einen Polyeders mit einem Bestandteil des anderen Polyeders. Wir führen die Kollisionserkennung also paarweise über den *features* der Objekte durch und erhalten den kleinsten Abstand zweier Objekte als den kleinsten Abstand zwischen jeweils einem *feature* der beteiligten Objekte. Damit genügt es, Kollisionserkennung auf unterster Ebene mit Ecken, Kanten und Flächen zu betreiben.

Wiederum unter Ausnutzen der geometrischen Kohärenz können wir Tests auf bestimmte geometrische Konstellationen, die bei hinreichend kleinem Zeitschritt Δt nicht auftreten, weglassen. Dazu gehören z.B. Kollisionen mit der „Innenseite" von Polyedern oder die Kollision mit konkaven Kanten. Voraussetzung für die Kollision mit einer Innenseite ist gegenseitige Durchdringung der Objekte. Das kann aber nicht passieren, da es vorher zu einer Kollision der „Außenseiten" kommen muß. Konkave Kanten könnten höchstens mit

Ecken oder parallelen Kanten des zweiten Objektes kollidieren. Diese Fälle werden aber als Sonderfall der Ecke-Fläche-Kollision behandelt. Ansonsten kommt es *vor* der Kollision mit einer konkaven Kante immer zum Kontakt mit einer adjazenten Fläche der Kante.

3.2 Kollisionserkennung auf unterster Ebene

Als Eingabe für die Kollisionserkennung haben wir das auf Kollision zu testende *feature*-Paar der beiden Objekte, jeweils *vor* und *nach* dem aktuellen Simulationsschritt. Aufgabe der Kollisionserkennung ist es, damit zu entscheiden, ob während des letzten Simulationsschrittes eine Kollision stattgefunden hat. Ist es zu einem Kontakt gekommen, müssen die Daten des Kontaktes berechnet werden, die die *Kollisionsreaktion* benötigt. Das sind:

1. Pointer auf die am Kontakt beteiligten *features* und die zugehörigen *Objekte*, um einen späteren Zugriff auf die Zustandsbeschreibung der Objekte zu ermöglichen.

2. Der *Zeitpunkt* der Kollision, um den Zeitpunkt der *frühesten* Kollision pro Simulationsschritt zu identifizieren und mittels Interpolation den Zustand des Systems zu diesem Zeitpunkt t_c herzustellen.

3. Der exakte *Punkt* des Kontaktes (im folgenden Abschnitt werden wir alle Kontaktarten auf Punktkontakte reduzieren).

4. Die *Kontaktebene*, senkrecht zu der der lineare Impuls beim Stoß sein Vorzeichen ändert.

Da wir die exakte Lage der *features* nur zu Beginn ($t = 0$) und Ende ($t = 1$) des Schrittes haben und den Weg dazwischen linear interpolieren, können wir den Kollisionszeitpunkt nicht exakt bestimmen. Wir berechnen t_c aus den minimalen Abständen der *features* zu den Zeitpunkten $t = 0$ und $t = 1$.

Ist $d(1)$ größer als $d(0)$, so wächst der Abstand zwischen den *features* und es kommt zu keiner Kollision. Wenn $d(1)$ größer als $-\varepsilon$ ist, dann ist die Durchdringungstoleranz noch nicht überschritten und es wird keine Kollision gemeldet. War schon $d(0)$ kleiner als $-\varepsilon$, so müßte schon im vorigen Schritt eine Kollision erkannt worden sein und es kommt nun zu Durchdringung. Da Durchdringung von der Reaktionskomponente verhindert wird, kommt dieser Fall nicht vor und dient nur als Sicherheitsabfrage.

Die Distanz der beiden *features* zu einem Zeitpunkt $0 \leq t \leq 1$ erhalten wir durch lineare Interpolation als:

$$d(t) = d(0) + t(d(1) - d(0)) \tag{3.1}$$

Gesucht wird nun der Zeitpunkt t_c, für den die Distanz genau Null ist, d.h. die *features* sich gerade berühren. Mit $d(t_c) = 0$ erhalten wir aus Gleichung (3.1) den Zeitpunkt der Kollision:

$$t_c = \begin{cases} 0 & \text{falls } d(0) - d(1) < \varepsilon \text{ (um } t < 0 \text{ zu verhindern)} \\ \frac{d(0)}{d(0)-d(1)} & \text{sonst} \end{cases} \tag{3.2}$$

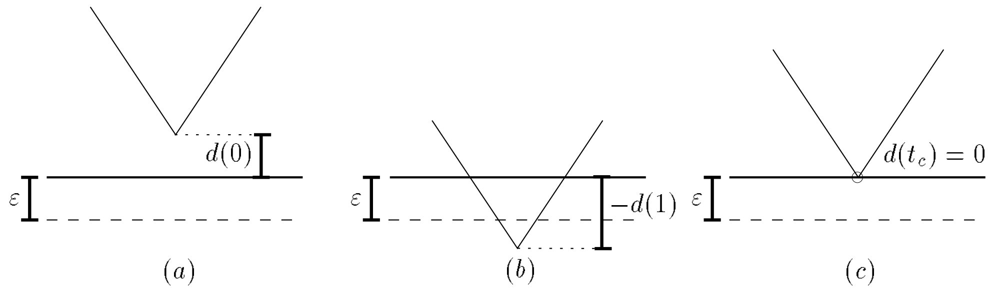

Abbildung 3.2: Distanzen zu den Zeitpunkten $t = 0$, $t = 1$ und $t = t_c$ für eine Beispielkollision.

3.2.1 Reduktion auf vier Typen der Kollisionserkennung

Polyeder sind aufgebaut aus Ecken, Kanten und Flächen. Für die elementaren Kollisionstests müßten wir also eigentlich $3^2 = 9$ Typen von *feature*-Paare betrachten. Diese Zahl läßt sich aber reduzieren.

1. Da die symmetrischen Fälle jeweils identisch sind, brauchen wir sie nicht doppelt zu betrachten. Damit bleiben nur noch sechs Fälle.

2. Kollisionen zwischen Flächen und Kanten direkt zu erkennen ist sehr schwierig. Nach Baraff[1] genügt es aber, die End- bzw. Randpunkte der *Kontaktregion* zu betrachten. Die Kontaktregion eines Fläche-Kante-Kontaktes sind ein oder mehrere Liniensegmente, deren Endpunkte durch Ecke-Fläche- bzw. Kante-Kante-Kontakte bestimmt werden. Mehrere Liniensegmente können evtl. bei nichtkonvexen Flächen vorkommen. Sind Quelle oder Ziel der Kante an der Kollision beteiligt (vgl. Abb. 3.3a), so erhalten wir die entsprechenden Ecke-Fläche-Kontakte. Wenn die Kante über den Rand der Fläche hinausragt, so stellen wir außerdem Kante-Kante-Kontakte mit den entsprechenden Kanten, die die Fläche begrenzen, fest (vgl. Abb. 3.3b und 3.3c).

   Abbildung 3.3 zeigt die möglichen Typen der Ecke-Fläche-Kontakte für den Fall konvexer Flächen. Für nichtkonvexe Flächen erhalten wir evtl. noch zusätzlich eine

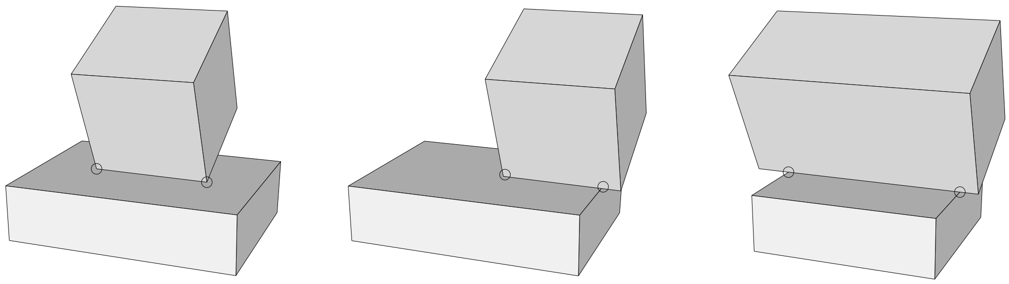

Abbildung 3.3: Verschiedene Typen der Kollision einer Kante mit einer (konvexen) Fläche.

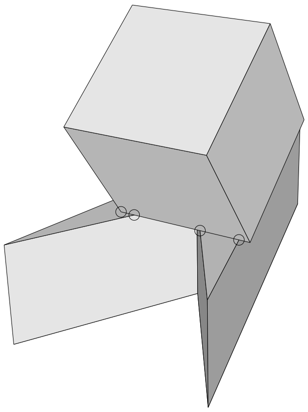

Abbildung 3.4: Beispiel für eine Fläche-Kante-Kollision bei nichtkonvexer Fläche.

gerade Anzahl von Kante-Kante-Kontakten, nämlich dort wo die Kante die Fläche „verläßt“ und wieder „betritt“ (siehe z.B. Abb. 3.4).

Da wir schon beim Einlesen alle nichtkonvexen Flächen in konvexe zerlegen, kommen bei uns nur die Fälle aus Abbildung 3.3 vor.

3. Ähnlich wie in Fall 2 können wir auch Fläche-Fläche-Kontakte darstellen als eine Menge von Ecke-Fläche- und Kante-Kante-Kontakten. Die Kontaktregion ist ein (evtl. nichtkonvexes) Polygon. Die für uns interessanten Randpunkte der Kontaktregion erhalten wir als Ecke-Fläche-Kontakte für die Eckpunkte der einen Fläche, die die andere Fläche berühren, und als Kante-Kante-Kontakte für sich berührende Begrenzungskanten der jeweiligen Flächen (vgl. z.B. Abb. 3.5).

Damit müssen wir jedes Objektpaar also noch auf Kante-Kante- und Ecke-Fläche-Kontakte und die degenerierten Fälle der Ecke-Kante- und Ecke-Ecke-Kollision testen. Kontaktpunkt ist dabei jeweils der beteiligte Eckpunkt bzw. der Schnittpunkt der beteiligten Kanten.

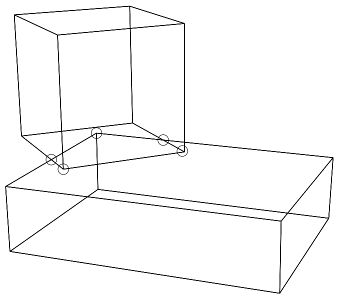

Abbildung 3.5: Beispiel für eine Fläche-Fläche-Kollision und die Kontaktpunkte, die sich aus den Ecken der Kontaktregion ergeben.

3.2.2 Kante-Kante-Kollision

Gegeben seien die Kanten e_1 und e_2 mit den Quellen $\mathbf{a}_1$ bzw. $\mathbf{a}_2$ und den Zielen $\mathbf{b}_1$ bzw. $\mathbf{b}_2$, jeweils zu den Zeitpunkten $t = 0$ und $t = 1$. Der so definierte Zeitschritt sei dabei so klein, daß wir unter Ausnutzung der Kohärenz annehmen können, daß sich die Lage der Endpunkte in einem Schritt hinreichend wenig ändert. Wie in Kapitel 1 beschrieben, wird jede Kante des Objekts als zwei entgegengesetzt gerichtete Kanten verwaltet, damit jeder gerichteten Kante eindeutig die adjazente Fläche zugeordnet werden kann.

Neben den Kanten selbst betrachten wir außerdem die adjazenten Flächen, um effizient und robust über die Lage der beiden Kanten zueinander entscheiden zu können. Die zweite adjazente Fläche ist dabei die der umgekehrt gerichteten Kante zugeordnete adjazente Fläche. Der so entstehende „Keil" wird eindeutig definiert durch die Kante e_i, die Normalen $\mathbf{n}_{f_i}$ und $\mathbf{n}_{f_i^{rev}}$ der adjazenten Flächen und die Vektoren $\mathbf{n}_{e_i}$ und $\mathbf{n}_{e_i^{rev}}$, die senkrecht auf e_i *und* der jeweiligen Flächennormalen stehen und von der Kante weg zeigen (vgl. zur Notation Abb. 3.6).

$$\mathbf{n}_{e_i} = \mathbf{n}_{f_i} \times (\mathbf{b}_i - \mathbf{a}_i)$$
$$\mathbf{n}_{e_i^{rev}} = (\mathbf{b}_i - \mathbf{a}_i) \times \mathbf{n}_{f_i^{rev}}$$

Wenn zwei Kanten nicht parallel sind, definieren sie eindeutig die Richtung einer Ebene im Raum. Den Normalenvektor dieser *Trennebene* erhalten wir einfach durch

$$\mathbf{n} = (\mathbf{b}_1 - \mathbf{a}_1) \times (\mathbf{b}_2 - \mathbf{a}_2)$$

Um unsere Kollisionserkennung möglichst schnell und effizient zu machen, testen wir einfache Bedingungen, die Kollisionen ausschließen, möglichst früh. Nur für Kantenpaare, die diese Tests passiert haben, führen wir die eigentliche Abstandsberechnung (vgl. Anfang von Abschnitt 3.2) auf den durch die Kanten verlaufenden Geraden durch. Zu dem über die Distanz errechneten *möglichen* Kollisionszeitpunkt t_c überprüfen wir dann noch, ob sich wirklich die Kanten oder nur die zugehörigen Geraden schneiden.

Wir führen den Kante-Kante-Kollisionstest in der folgenden Reihenfolge durch und wissen, daß es zu keiner Kollision gekommen sein kann, sobald die erste dieser Bedingungen erfüllt ist:

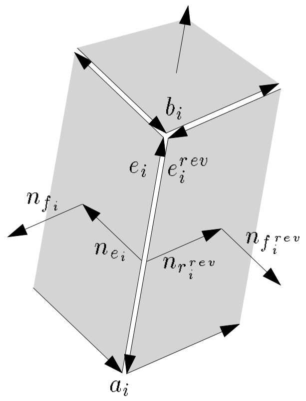

Abbildung 3.6: Bei Kante-Kante-Kollisionstests verwendete Notation

1. Ist eine der beteiligten Kanten konkav?
2. Liegt einer der Keile zu *keinem* Zeitpunkt zwischen $t = 0$ und $t = 1$ vollständig auf einer Seite der Trennebene?
3. Liegen beide Keile auf derselben Seite der Trennebene?
4. Sind die Kanten (fast) parallel?
5. Wächst die Distanz zwischen den Geraden, war $d(0)$ schon kleiner als $-\varepsilon$, oder ist die Distanz zum Zeitpunkt $t = 1$ größer als $-\varepsilon$?
6. Liegt einer der Lotfußpunkte an der kürzesten Verbindungsstrecke der Geraden *„neben"* dem anderen Liniensegment?

Wie testen wir nun im einzelnen die Bedingungen?

1. Für konkave Kanten gilt:
$$\det(\mathbf{b}_i - \mathbf{a}_i, \mathbf{n}_{f_i}, \mathbf{n}_{f_i^{rev}}) < 0$$
2. Um über die Lage des Keiles zu der Trennebene zu entscheiden, definieren wir für die Flächen f_i und f_i^{rev} eine Funktion
$$s(f_i(t)) = \mathbf{n}^T \mathbf{n}_{e_i} \text{ bzw. } s(f_i^{rev}(t)) = \mathbf{n}^T \mathbf{n}_{e_i^{rev}}$$
Ihr Vorzeichen erlaubt es uns, den Halbraum bzgl. der Trennebene festzustellen, in dem die Fläche zum Zeitpunkt t liegt.

   Gilt nun
$$\begin{aligned} & s(f_i(0)) < 0 \wedge s(f_i^{rev}(0)) > 0 \wedge s(f_i(1)) < 0 \wedge s(f_i^{rev}(1)) > 0 \\ \vee & s(f_i(0)) > 0 \wedge s(f_i^{rev}(0)) < 0 \wedge s(f_i(1)) > 0 \wedge s(f_i^{rev}(1)) < 0, \end{aligned}$$

d.h. liegen f_i und f_i^{rev} zu den Zeitpunkten $t = 0$ und $t = 1$ jeweils auf verschiedenen Seiten der Trennebene, so kann bei hinreichend glattem Verlauf von $s(f_i(t))$ und $s(f_i^{rev}(t))$ (Kohärenz!) keine Kollision auftreten.

3. Um zu testen, ob beide Keile auf der *gleichen* Seite der Trennebene liegen, benutzen wir die gleiche Funktion $s(f_i(t))$.
   Gilt
   $$\begin{aligned} &s(f_1(0)) = s(f_1^{rev}(0)) = s(f_1(1)) = s(f_1^{rev}(1)) \\ =&s(f_2(0)) = s(f_2^{rev}(0)) = s(f_2(1)) = s(f_2^{rev}(1)), \end{aligned}$$
   so liegen die beiden Keile auf der gleichen Seite der Trennebene und die zugehörigen Kanten können nicht kollidieren.

4. Der Fall von (fast) parallelen Kanten wird in Abschnitt 3.2.4 behandelt und hier nicht näher untersucht. Wenn Kanten fast parallel sind, ist die Länge ihres Kreuzproduktes sehr klein. Wir benutzen also
   $$((\mathbf{b}_1 - \mathbf{a}_1) \times (\mathbf{b}_2 - \mathbf{a}_2))^T((\mathbf{b}_1 - \mathbf{a}_1) \times (\mathbf{b}_2 - \mathbf{a}_2)) = \mathbf{n}^T\mathbf{n} < \varepsilon$$
   als Kriterium für fast parallele Kanten.

5. Erst jetzt berechnen wir die Distanz zwischen den durch die Kanten verlaufenden Geraden. Den Abstand erhalten wir als den Abstand der Lotfußpunkte der Geraden aufeinander. Das Lot steht senkrecht auf beiden Geraden und entspricht in seiner Länge gerade dem kürzesten Abstand zweier Punkte auf den Geraden in Normalenrichtung der Trennebene. Dieser Abstand
   $$\delta(t) = \frac{\mathbf{n}^T(t)}{|\mathbf{n}(t)|}(\mathbf{a}_1(t) - \mathbf{a}_2(t)), \quad t \in \{0, 1\}$$
   ist vorzeichenbehaftet: Falls $s(f_i(t)) + s(f_i^{rev}(t)) > 0$ ist, bedeutet $\mathbf{n}^T(t)(\mathbf{a}_1(t) - \mathbf{a}_2(t)) > 0$, daß der Keil zu $e_1(t)$ den Keil zu $e_2(t)$ nicht schneidet. Falls $s(f_i(t)) + s(f_i^{rev}(t)) < 0$, bedeutet $\mathbf{n}^T(t)(\mathbf{a}_1(t) - \mathbf{a}_2(t)) < 0$, daß sich die Keile nicht schneiden. d definieren wir daher als:
   $$d(t) = \operatorname{sign}\big(s(f_i(t)) + s(f_i^{rev}(t))\mathbf{n}^T(t)(\mathbf{a}_1(t) - \mathbf{a}_2(t)\big), \quad t \in \{0, 1\} \tag{3.3}$$

6. Um zu entscheiden, ob sich die *Kanten* oder nur die Geraden schneiden, müssen wir herausfinden, ob die Lotfußpunkte an der Stelle mit dem kürzesten Abstand jeweils *auf* den Kanten liegen. An der Stelle des kürzesten Abstands δ muß gelten:
   $$\mathbf{a}_1 + \lambda_1(\mathbf{b}_1 - \mathbf{a}_1) + \delta\mathbf{n} = \mathbf{a}_2 + \lambda_2(\mathbf{b}_2 - \mathbf{a}_2) \tag{3.4}$$

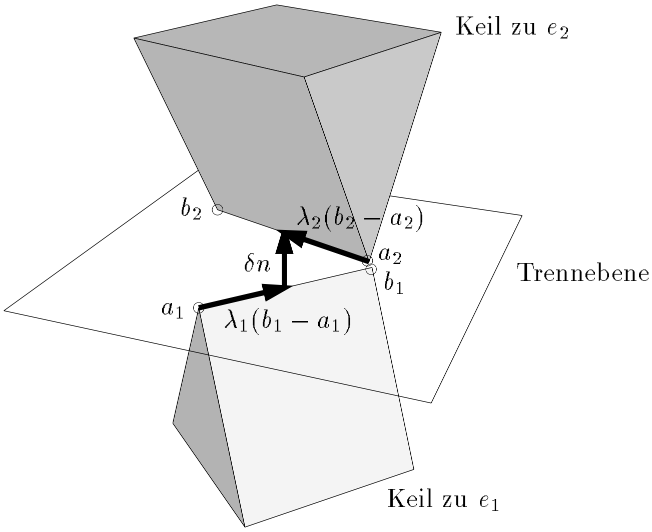

Abbildung 3.7: Zu den Kanten e_1 und e_2 gehörende Keile und Erläuterung zu Gleichung 3.4

Kramer'sche Regel:

$$\alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} = \mathbf{d}$$
$$\alpha = \frac{\det(\mathbf{d},\mathbf{b},\mathbf{c})}{\det(\mathbf{a},\mathbf{b},\mathbf{c})}, \quad \beta,\gamma \text{ analog definiert}$$

Aus Gleichung 3.4 erhalten wir

$$\lambda_1(\mathbf{b}_1 - \mathbf{a}_1) - \lambda_2(\mathbf{b}_2 - \mathbf{a}_2) + \delta\mathbf{n} \; = (\mathbf{a}_2 - \mathbf{a}_1)$$

Wir lösen mit der Kramer'schen Regel nach λ_1 bzw. λ_2 auf

$$\begin{aligned}
\lambda_1 &= \frac{\det(\mathbf{a}_2 - \mathbf{a}_1, \mathbf{a}_2 - \mathbf{b}_2, \mathbf{n})}{\det(\mathbf{b}_1 - \mathbf{a}_1, \mathbf{a}_2 - \mathbf{b}_2, \mathbf{n})} = \frac{(\mathbf{a}_2 - \mathbf{a}_1)^T((\mathbf{a}_2 - \mathbf{b}_2)\times\mathbf{n})}{(\mathbf{b}_1 - \mathbf{a}_1)^T((\mathbf{a}_2 - \mathbf{b}_2)\times\mathbf{n})} \\
&= \frac{(\mathbf{b}_2 - \mathbf{a}_2)^T(\mathbf{n}\times(\mathbf{a}_2 - \mathbf{a}_1))}{\mathbf{n}^2} \\
\lambda_2 &= \frac{\det(\mathbf{b}_1 - \mathbf{b}_2, \mathbf{a}_2 - \mathbf{a}_1, \mathbf{n})}{\det(\mathbf{b}_1 - \mathbf{a}_1, \mathbf{a}_2 - \mathbf{b}_2, \mathbf{n})} = \frac{(\mathbf{b}_1 - \mathbf{a}_1)^T(\mathbf{n}\times(\mathbf{a}_2 - \mathbf{a}_1))}{\mathbf{n}^2}
\end{aligned}$$

Die Lotfußpunkte liegen auf den Kanten selbst, falls $0 \leq \lambda_1, \lambda_2 \leq 1$. Die Werte $\lambda_i(t_c)$ zum Kollisionszeitpunkt t_c erhalten wir durch Interpolation:

$$\lambda_i(t_c) = \lambda_i(0) + t_c(\lambda_i(1) - \lambda_i(0))$$

Dabei kann es vorkommen, daß die interpolierten $\lambda_i(t_c)$ zwischen 0 und 1 liegen, die λ_i zu den Zeitpunkten 0 und 1 aber nicht. Dieser Fall kann bei hinreichend kleinem

Zeitschritt eigentlich nur vorkommen, wenn die Distanz zwischen den Kanten noch zu groß für eine Kollision ist. Eigentlich dürfte das Kantenpaar damit den vorigen Test nicht passiert haben, aber da die Berechnung der Normalen der Trennebene vorzeichenbehaftet ist, kann es unter ungünstigen geometrischen Bedingungen (vgl. Abb. 3.8) passieren, daß sich die Normalenrichtung um mehr als 90° ändert und das Kantenpaar damit den Distanztest passieren läßt.

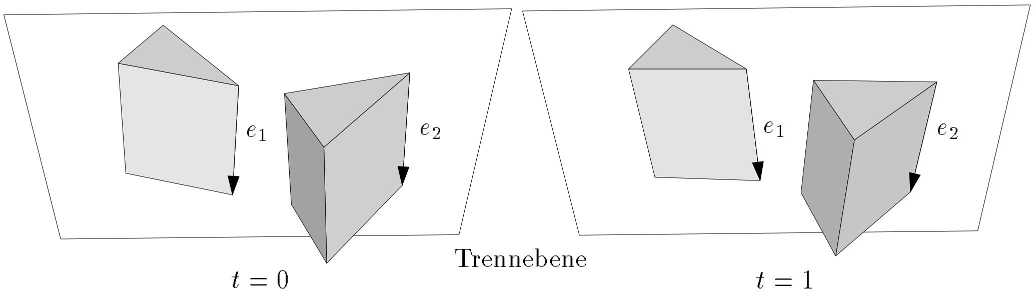

Abbildung 3.8: Beide Keile liegen jeweils vollständig auf *einer* Seite der Trennebene. Zum Zeitpunkt $t = 0$ zeigt die Normale aus der Bildebene heraus, bei $t = 1$ in die Ebene hinein. Die Richtung der Normalen ändert sich also um mehr als 90°. Dadurch kann das Kantenpaar bei ungünstiger Konstellation fälschlicherweise den Distanztest passieren.

Wir lösen das Problem, indem wir prüfen, ob der Winkel zwischen $\mathbf{n}(0)$ und $\mathbf{n}(1)$ größer als 90° ist (d.h. $\mathbf{n}^T(0)\mathbf{n}(1) < 0$), wenn die λ_i zu den Zeitpunkten 0 oder 1 außerhalb des Intervalls $[0, 1]$ liegen. Der Test auf den Winkel alleine genügt nicht, weil mit einem Winkel größer als 90° durchaus „echte" Kollisionen auftreten können (vgl. Abb. 3.9).

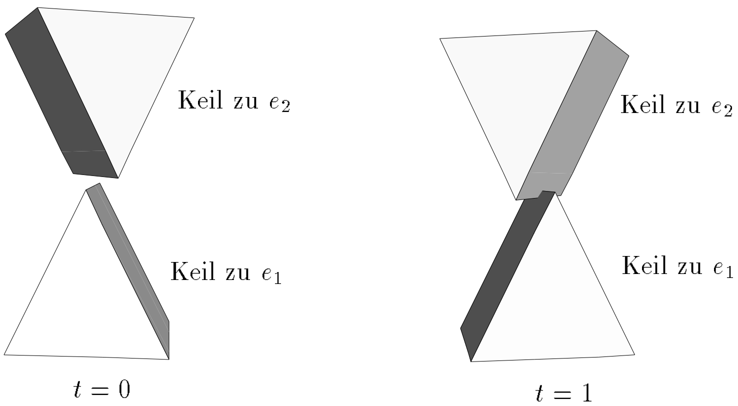

Abbildung 3.9: Bei $t = 0$ zeigt die Normale in Richtung des Keils zu e_1, bei $t = 1$ zeigt die Normale auf den Keil zu e_2. Der Winkel zwischen $\mathbf{n}(0)$ und $\mathbf{n}(1)$ ist also größer als 90°, wir haben aber trotzdem eine „echte" Kollision.

Wenn wir nun eine Kollision erkannt haben, müssen wir noch die zu einer weiteren Verarbeitung notwendigen Daten berechnen. Den Kontaktpunkt interpolieren wir als

$$\begin{aligned}
\mathbf{p}(0) &= \mathbf{a}_2(0) + \lambda_2(0)(\mathbf{b}_2(0) - \mathbf{a}_2(0)) \\
\mathbf{p}(1) &= \mathbf{a}_2(1) + \lambda_2(1)(\mathbf{b}_2(1) - \mathbf{a}_2(1)) \\
\mathbf{p}(t_c) &= \mathbf{p}(0) + t_c(\mathbf{p}(1) - \mathbf{p}(0)).
\end{aligned} \tag{3.5}$$

Die Reaktionskomponente setzt voraus, daß die Normale der Kollisionsebene immer in Richtung von Objekt 1 zeigt. Durch Interpolieren erhalten wir für $\mathbf{n}(t_c)$:

$$\begin{aligned}
\mathbf{n}(t_c) &= \begin{cases} -s(f_2(t_c))\big(\mathbf{n}(0) + t_c(\mathbf{n}(1) - \mathbf{n}(0))\big) & \text{falls } s(f_2(t_c)) \neq 0 \\ -s(f_2^{rev}(t_c))\big(\mathbf{n}(0) + t_c(\mathbf{n}(1) - \mathbf{n}(0))\big) & \text{sonst} \end{cases} \\
\mathbf{n}(t_c) &= \frac{\mathbf{n}(t_c)}{|\mathbf{n}(t_c)|}
\end{aligned}$$

3.2.3 Ecke-Fläche-Kollision

Gegeben seien die Ecke $\mathbf{b}$ und die Fläche f mit den Eckpunkten $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k$, die in der Ebene mit der Gleichung $\mathbf{n}^T\mathbf{x} = n_0$ liegt, jeweils zu den Zeitpunkten $t = 0$ und $t = 1$. Wir können davon ausgehen, daß f konvex ist, da wir schon beim Einlesen (vgl. Kapitel 1) alle nichtkonvexen Flächen in konvexe zerlegen. Die Abstände der Punkte $|\mathbf{a}_i(1) - \mathbf{a}_i(0)|$ und $|\mathbf{b}(1) - \mathbf{b}(0)|$ seien dabei hinreichend klein.

Wir berechnen $d(0)$ und $d(1)$ als den Abstand von Punkt $\mathbf{b}$ zu der Ebene durch f.

$$d(t) = \mathbf{n}^T(t)\mathbf{b}(t) - n_0, \quad t \in \{0, 1\} \tag{3.6}$$

Ist $d(t) \geq 0$, so liegt $\mathbf{b}(t)$ im positiven Halbraum der durch $f(t)$ definierten Ebene.

1. Gilt $d(0) - d(1) < 0$, so entfernen sich die Objekte voneinander.
2. Gilt $d(1) > -\varepsilon$, so ist die Toleranz für eine zulässige Durchdringung noch nicht überschritten.
3. Gilt $d(0) < -\varepsilon$, so lag $\mathbf{b}(0)$ schon auf der „Rückseite" der zu f gehörenden Fläche.
4. Gelten die vorigen Bedingungen nicht, so sind die Distanzvoraussetzungen für eine Kollision erfüllt, und wir interpolieren den (möglichen) Kollisionszeitpunkt t_c (vgl. Abschnitt 3.2).

Um zu entscheiden, ob wir eine Kollision entdeckt haben, müssen wir noch testen, ob sich Punkt $\mathbf{b}$ zur Zeit t_c im *Extrusionskörper* von Fläche f befunden hat.

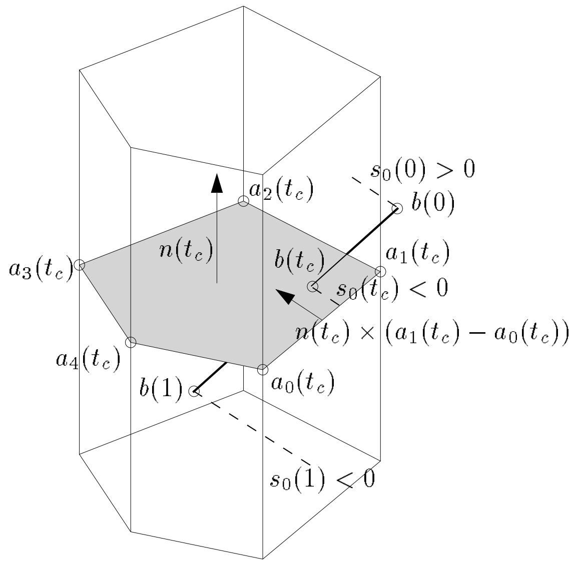

Abbildung 3.10: Extrusionskörper zur konvexen Fläche $f(t_c)$. Die Vorzeichen der $s_i(t_c)$ zeigen an, auf welcher Seite bezüglich $f(t_c)$ der jeweiligen Kante der Punkt $\mathbf{b}(t_c)$ liegt.

Ein Punkt liegt im Extrusionskörper einer (konvexen) Fläche, wenn er auf der „Innenseite“ aller die Fläche berandenden Kanten liegt. Wir definieren dazu die Funktion

$$s_i(t) = \big(\mathbf{n}(t) \times (\mathbf{a}_{(i+1)\mathrm{mod}k}(t) - \mathbf{a}_i(t))\big)^T (\mathbf{b}(t) - \mathbf{a}_i(t)), \quad t \in \{0,1\}, i \in \{0,1,\ldots,k-1\}. \tag{3.7}$$

Die Funktionswerte der s_i zum Zeitpunkt t_c erhalten wir mittels Interpolation:

$$s_i(t_c) = s_i(0) + t_c(s_i(1) - s_i(0)) \tag{3.8}$$

Ist $s_i(t_c) < 0$ für *alle* $i \in \{0,1,\ldots,k-1\}$, so haben wir eine Kollision erkannt. Gilt $|s_i(t_c)| < \varepsilon$ für eine bzw. zwei der Begrenzungskanten von $f(t_c)$, so haben wir eine Kollision mit dem Rand festgestellt. Diesen Fall behandeln wir als Ecke-Kante- bzw. Ecke-Ecke-Kollision in Abschnitt 3.2.4.

Den Kontaktpunkt und die Normale der Kollisionsebene erhalten wir im Fall der Ecke-Fläche-Kollision einfach aus dem für den Zeitpunkt t_c interpolierten Eckpunkt $\mathbf{b}(t_c)$ bzw. der Normalen der beteiligten Fläche $\mathbf{n}(t_c)$.

$$\mathbf{p}(t_c) = \mathbf{b}(0) + t_c(\mathbf{b}(1) - \mathbf{b}(0)) \tag{3.9}$$
$$\mathbf{n}(t_c) = \mathbf{n}(0) + t_c(\mathbf{n}(1) - \mathbf{n}(0)) \tag{3.10}$$

3.2.4 Ecke-Kante- und Ecke-Ecke-Kollision

Innerhalb der Ecke-Fläche-Kollision haben wir festgestellt, ob der Punkt $\mathbf{b}$ mit dem Rand von Fläche f (bzw. mit Toleranz ε vom Rand entfernt) kollidiert. Haben wir $|s_j(t_c) < \varepsilon|$

für zwei Begrenzungskanten $(\mathbf{a}_{i-1}, \mathbf{a}_i)$ und $(\mathbf{a}_i, \mathbf{a}_{i+1})$ festgestellt, so haben wir eine Ecke-Ecke-Kollision zwischen $\mathbf{b}$ und $\mathbf{a}_i$. Gilt $|s_j(t_c) < \varepsilon|$ nur für eine Kante $(\mathbf{a}_i, \mathbf{a}_{i+1})$, so haben wir eine Ecke-Kante-Kollision zwischen $\mathbf{b}$ und $(\mathbf{a}_i, \mathbf{a}_{i+1})$.

Nun müssen wir noch beurteilen, ob es wirklich zu einer Kollision kommt oder ob die Ecke $\mathbf{b}$ sich mit allen adjazenten Flächen f' sicher an der Kante $(\mathbf{a}_i, \mathbf{a}_{i+1})$ bzw. der Ecke $\mathbf{a}_i$ vorbeibewegen kann. Um das zu entscheiden, definieren wir zunächst die relative Bewegungsrichtung von $\mathbf{b}$ und dem Berührpunkt $\mathbf{p}$ als

$$\mathbf{v} = (\mathbf{b}(1) - \mathbf{b}(0)) - (\mathbf{p}(1) - \mathbf{p}(0)).$$

Trifft $\mathbf{b}$ auf $\mathbf{a}_i$, so können wir $\mathbf{p}(t) = \mathbf{a}_i(t)$ setzen. Trifft $\mathbf{b}$ aber auf die Kante, so erhalten wir $\mathbf{p}(0)$ und $\mathbf{p}(1)$ wie folgt:

$$\begin{aligned}
\mu(t_c) &= \frac{(\mathbf{a}_{i+1}(t_c) - \mathbf{a}_i(t_c))^T(\mathbf{b}(t_c) - \mathbf{a}_i(t_c))}{(\mathbf{a}_{i+1}(t_c) - \mathbf{a}_i(t_c))^2} \\
\mathbf{p}(0) &= \mathbf{a}_i(0) + \mu(t_c)(\mathbf{a}_{i+1}(0) - \mathbf{a}_i(0)) \\
\mathbf{p}(1) &= \mathbf{a}_i(1) + \mu(t_c)(\mathbf{a}_{i+1}(1) - \mathbf{a}_i(1))
\end{aligned}$$

Dabei bezeichnet $\mu(t_c)$ das Verhältnis, in dem $\mathbf{p}$ die Kante zum Zeitpunkt t_c teilt. Damit gibt $\mathbf{p}(t_c)$ also den Berührpunkt auf der Kante an. Die Punkte $\mathbf{a}_i$, $\mathbf{a}_{i+1}$ und $\mathbf{b}$ zum Zeitpunkt t_c erhalten wir wie üblich durch lineare Interpolation.

Wir suchen nun ein Kriterium, um festzustellen, ob sich eine zu $\mathbf{b}$ adjazente Fläche $f' = (\ldots, \mathbf{b}_1, \mathbf{b}, \mathbf{b}_2, \ldots)$ ohne Kollision an der Kante bzw. der Ecke vorbeibewegen kann. Wir betrachten für jedes f' das Vorzeichen von

$$\underbrace{\big((\mathbf{a}_{i+1}(t_c) - \mathbf{a}_i(t_c)) \times \mathbf{v}\big)^T}_{\mathbf{g}(t_c)} \underbrace{\big(\lambda_1(\mathbf{b}_1(t_c) - \mathbf{b}(t_c)) + \lambda_2(\mathbf{b}_2(t_c) - \mathbf{b}(t_c))\big)}_{\mathbf{h}(t_c)}, \quad \lambda_1, \lambda_2 > 0. \tag{3.11}$$

$\mathbf{h}(t_c)$ ist für $\lambda_1, \lambda_2 > 0$ der Vektor von Punkt $\mathbf{b}(t_c)$ zu einem Punkt im Innern des Schnittes der beiden negativen Halbebenen zu den an $\mathbf{b}$ anliegenden Kanten von f'. $\mathbf{g}(t_c)$ steht senkrecht auf der beteiligten Kante $(\mathbf{a}_i, \mathbf{a}_{i+1})$ und der relativen Bewegungsrichtung $\mathbf{v}$.

Gibt es also für eine zu $\mathbf{b}$ adjazente Fläche f' ein Paar $\lambda_1, \lambda_2 > 0$, für das (3.11) größer als Null ist, so haben wir eine Fläche gefunden, die sich *nicht* sicher (ohne Kollision) an der Kante $(\mathbf{a}_i, \mathbf{a}_{i+1})$ vorbei bewegen kann. Intuitiv gibt das Vorzeichen von $\mathbf{g}(t_c)^T\mathbf{h}(t_c)$ an, in *welchen* Halbraum bzgl. der Ebene durch $(\mathbf{a}_i(t_c), \mathbf{a}_{i+1}(t_c))$ und $\mathbf{v}(t_c)$ der Vektor $\mathbf{h}(t_c)$ einer zu $\mathbf{b}(t_c)$ adjazenten Fläche zeigt.

Abbildung 3.12 zeigt in Teil (a) eine Kontaktsituation, bei der wir *keine* Kollision melden. Die Kante $(\mathbf{a}_i, \mathbf{a}_{i+1})$ zeigt in die Bildebene hinein. Es gilt $\mathbf{g}^T\mathbf{h} < 0$ für alle zu $\mathbf{b}$ adjazenten Flächen, die Objekte können sich gefahrlos aneinander vorbei bewegen. In Teil (b) sieht die Situation anders aus: $\mathbf{b}$ und $(\mathbf{a}_i, \mathbf{a}_{i+1})$ bewegen sich aufeinander zu (siehe $\mathbf{v}$). Damit wird $\mathbf{g}^T\mathbf{h}_{f''} > 0$, die Bedingung (3.12) ist also für mindestens eine Fläche erfüllbar (bei passender Wahl der λ_i). Wir melden hier eine Kollision zurück.

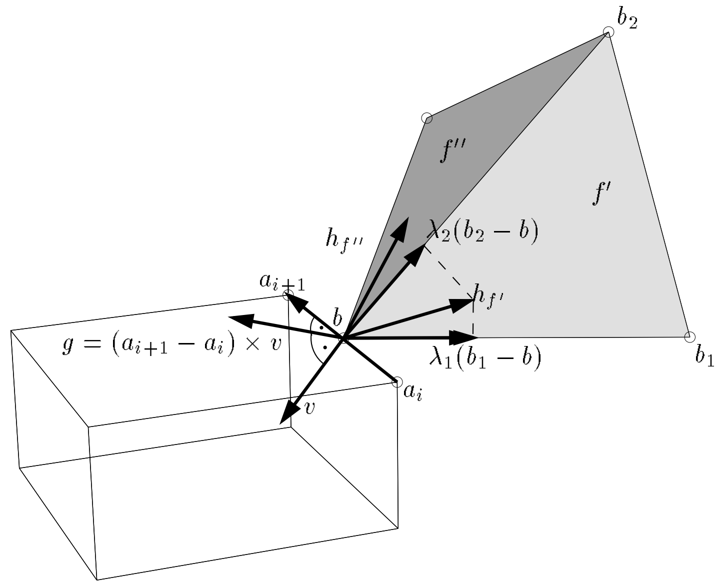

Abbildung 3.11: Notation bei Ecke-Kante- bzw. Ecke-Ecke-Kontakten.

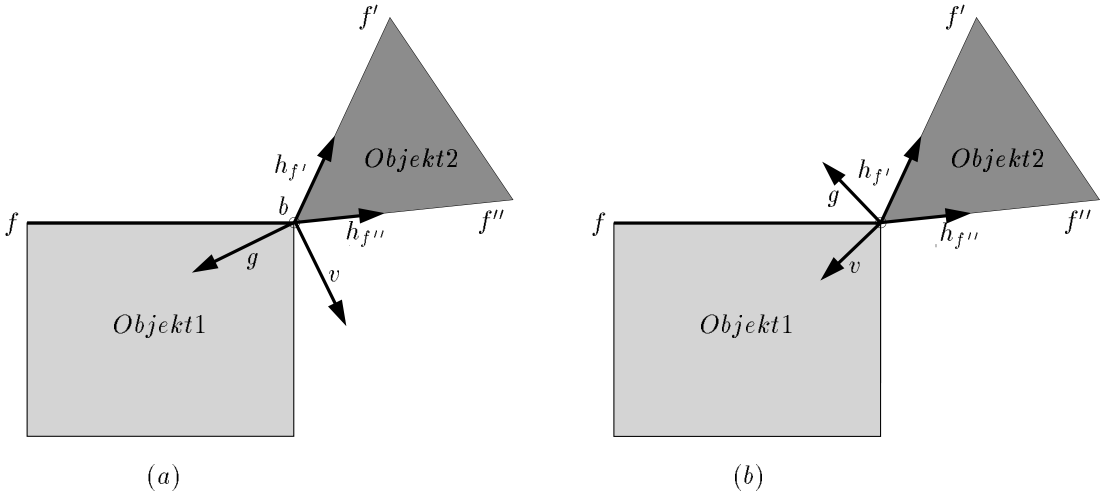

Abbildung 3.12: Beispiele für Ecke-Kante-Kontakte: In Teil (a) können sich die Objekte sicher aneinander vorbei bewegen, obwohl es zur Berührung kommt, in Teil (b) kommt es zu einer *echten* Kollision

Hatten wir einen Ecke-Ecke-Kontakt anstatt des Ecke-Kante-Kontaktes, so muß für eine tatsächliche Kollision zusätzlich die analoge Bedingung für Kante $(\mathbf{a}_{i-1}, \mathbf{a}_i)$ erfüllt sein.

Einen Sonderfall haben wir, falls der Kontakt schon zum Zeitpunkt $t = 0$ bestand. Ein Kontakt zum Zeitpunkt 0 ist ein bestehender Kontakt bzw. ein Gleitkontakt. Ein Gleitkontakt, dessen Kontaktpunkt sich zum Rand der Fläche bewegt, löst sich, sobald er den

Rand erreicht hat. Diesen Fall des sich lösenden Kontaktes wollen wir *nicht* als Kollision deklarieren und brauchen daher noch die zusätzliche Bedingung

$$\exists\ f' = (\dots, \mathbf{b}_1, \mathbf{b}, \mathbf{b}_2, \dots) \text{ und } \lambda_1, \lambda_2 > 0: \\ \mathbf{n}(0)^T \big(\lambda_1(\mathbf{b}_1(0) - \mathbf{b}(0)) + \lambda_2(\mathbf{b}_2(0) - \mathbf{b}(0))\big) > 0.$$

Diese Bedingung bedeutet, daß zumindest eine der zu $\mathbf{b}$ adjazenten Flächen noch eine positive Komponente in Richtung der Normalen der Fläche haben muß, mit deren Rand $\mathbf{b}$ zum Zeitpunkt $t = 0$ in Berührung ist. Ist das nicht mehr der Fall, so löst sich der Kontakt und $\mathbf{b}$ bewegt sich ohne weitere Kollision an der Kante/Ecke des anderen Objektes vorbei.

Abbildung 3.13 zeigt in Teil (a) einen Gleitkontakt zum Zeitpunkt 0, den wir noch als Kollision betrachten ($\mathbf{n}^T(0)\mathbf{h}_{f'}(0) < 0$). Den Kontakt in Teil (b) dagegen sehen wir schon als gelöst an ($\mathbf{n}^T(0)\mathbf{h}(0) > 0$ für alle an $\mathbf{b}(0)$ adjazenten Flächen von Objekt 2).

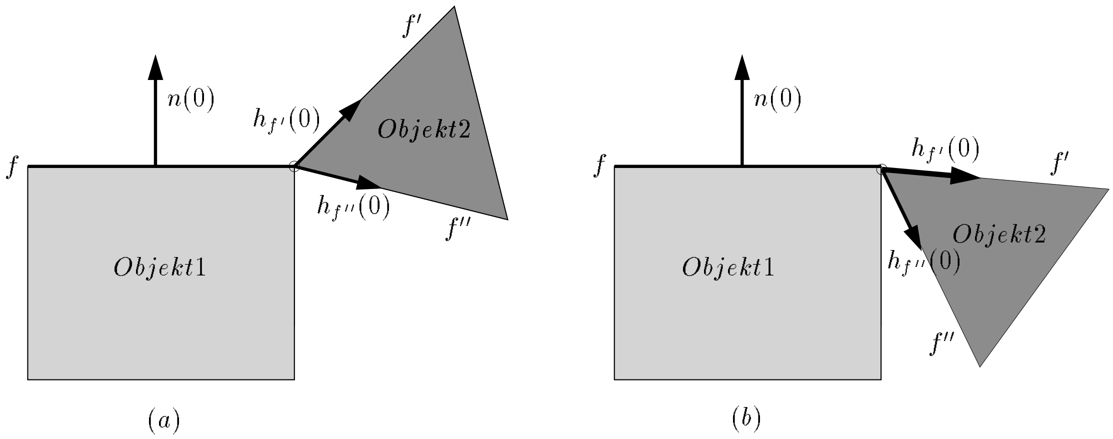

Abbildung 3.13: Sich am Rand der Fläche lösender Gleitkontakt, erkannt über Bedingung (3.14).

Die Bewegung wird also genau dann zu einer Kollision führen, wenn es eine zu $\mathbf{b}$ adjazente Fläche f' gibt, so daß gilt:

$$\exists \lambda_1, \lambda_2 > 0:$$

$$\big((\mathbf{a}_{i+1}(t_c) - \mathbf{a}_i(t_c)) \times \mathbf{v}\big)^T \big(\lambda_1(\mathbf{b}_1(t_c) - \mathbf{b}(t_c)) + \lambda_2(\mathbf{b}_2(t_c) - \mathbf{b}(t_c))\big) > 0 \tag{3.12}$$

$$\big((\mathbf{a}_i(t_c) - \mathbf{a}_{i-1}(t_c)) \times \mathbf{v}\big)^T \big(\lambda_1(\mathbf{b}_1(t_c) - \mathbf{b}(t_c)) + \lambda_2(\mathbf{b}_2(t_c) - \mathbf{b}(t_c))\big) > 0 \tag{3.13}$$

$$\mathbf{n}(t_c)^T \big(\lambda_1(\mathbf{b}_1(t_c) - \mathbf{b}(t_c)) + \lambda_2(\mathbf{b}_2(t_c) - \mathbf{b}(t_c))\big) > 0 \tag{3.14}$$

Die Bedingungen (3.12)–(3.14) werden für den Fall des Ecke-Ecke-Kontaktes mit $t_c = 0$ benötigt. Haben wir einen Ecke-Kante-Kontakt, so lassen wir Bedingung (3.13) weg, ist $t_c > 0$, so lassen wir (zusätzlich) Bedingung (3.14) weg.

Wir erhalten also pro Fläche, die zu $\mathbf{b}$ adjazent ist, für jede der bis zu drei Bedingungen eine Gleichung der Form:

$$\exists \lambda_1, \lambda_2 > 0 : \forall i : \alpha_i \lambda_1 + \beta_i \lambda_2 > 0 \tag{3.15}$$

Wir müssen das so erhaltene homogene Ungleichungssystem nicht lösen, sondern nur entscheiden, ob es eine zulässige Lösung gibt. Da nur die *Richtung* von $\mathbf{h}$ entscheidend ist und nicht die Länge, genügt es, nur den Schnitt des Zulässigkeitsbereiches mit dem Einheitsquadrat zu betrachten ($0 \leq \lambda_1, \lambda_2 \leq 1$). Den Zulässigkeitsbereich erhalten wir aus den zu den Bedingungen (3.15) gehörenden Geradengleichungen (vgl. Abbildung 3.14).

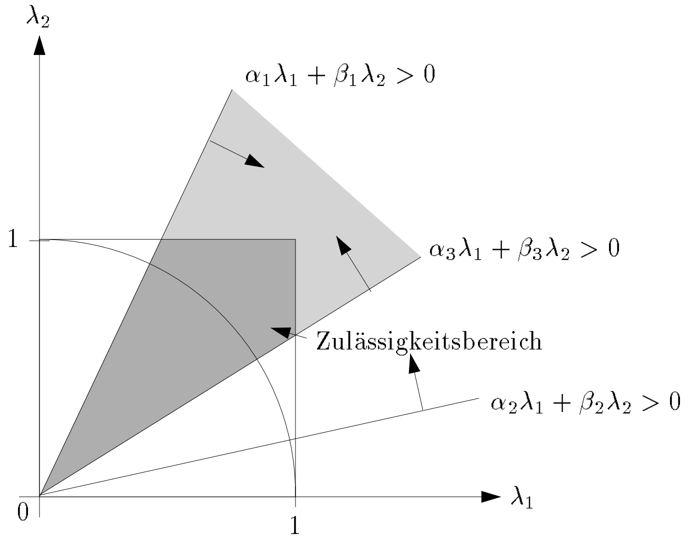

Abbildung 3.14: Zulässigkeitsbereich eines homogenen linearen Ungleichungssystems im Einheitsquadrat.

Da wir nun auch für Ecke-Kante- bzw. Ecke-Ecke-Kontakte sicher entscheiden können, ob es zu einer Kollision kommt oder nicht, müssen wir nun noch Kontaktebene und Kontaktpunkt bestimmen. Den Kontaktpunkt haben wir schon als $\mathbf{p}(t_c)$ bestimmt. Die Normale der Kontaktebene ist aber in diesem Fall nicht wohldefiniert und das korrekte physikalische Verhalten der beiden Objekte damit nichtdeterministisch, solange es degenerierte Kontaktpunkte gibt. Nach Baraff[1] ist die Berechnung der Kontaktkraft für einen degenerierten Kontakt ein NP-vollständiges Problem. Das können wir vermeiden, indem wir den Kontakt deterministisch machen, d.h. als Ecke-Fläche-Kontakt betrachten und die Kontaktdaten (Normale der Trennebene) entsprechend bestimmen. In unserem Fall erhalten wir die degenerierten Kontakte als Sonderfall im Rahmen der Ecke-Fläche-Kollisionserkennung und wählen daher als Trennebene die Ebene durch die zugehörige Fläche mit Richung auf Objekt 1.

3.3 Heuristiken

Bei großen Szenarien, d.h. vielen Objekten mit großer Komplexität, stellt sich die Kollisionserkennung als *„Bottleneck“* der Simulation heraus. Wir müssen bisher *jedes feature*-Paar eines *jeden* Objektpaares auf Kollision testen. In der Praxis ist es aber so, daß sich die meisten Objekte nie oder nur für kurze Zeit in „gefährlicher“ Nähe zueinader befinden. Wir suchen also effiziente Heuristiken, die schnell feststellen, ob sich Objekte überhaupt so nahe sind, daß es zu einer Kollision kommen kann, und erst dann die eigentliche Kollisionserkennung aufrufen. Es bieten sich *Hüllkörper* an wie z.B. die im folgenden vorgestellte *kleinste umschließende Kugel* (*smallest enclosing sphere*, SES). Hüllkörper sollen einen möglichst schnellen Kollisionstest erlauben und das Objekt dabei möglichst genau annähern.

Im folgenden wird eine *3-Stufen-Kollisionserkennung* beschrieben. Wenn die einschließenden Kugeln eines Objekt-Paares kollidieren, lösen wir *eine* der Kugeln auf (die Kugel, deren Volumen im Verhältnis zum Volumen des Objektes am schlechtesten ist). Erhalten wir auch hier eine oder mehrere Kollisionen, so lösen wir auch die zweite Kugel auf und führen die oben vorgestellte *low-level* Kollisionserkennung nur für die in Frage kommenden *features* des zuerst aufgelösten Objektes durch. In Frage kommen dabei alle *features*, für die in der zweiten Stufe eine Kollision mit der zweiten Kugel festgestellt worden ist.

3.3.1 Kugelheuristik

Berechnung der SES

Wenn wir einen konvexen Hüllkörper zu einem Objekt berechnen wollen, genügt es, die Hülle über die Eckpunkte der konvexen Hülle zu bestimmen. Wir wollen die Menge $\mathcal{P}$ der Eckpunkte so in eine Kugel einpacken, daß der größte Abstand eines Punktes aus $\mathcal{P}$ zu dem gewählten Mittelpunkt $\mathbf{c}$ minimiert wird. Mit dem Radius minimieren wir auch gleichzeitig das Volumen der einschließenden Kugel und nähern das Objekt so möglichst gut an. Wir definieren $\mathbf{r}(\mathbf{c})$ als den Radius der kleinsten einschließenden Kugel zum Mittelpunkt $\mathbf{c}$ und $r_i(\mathbf{c}) = |\mathbf{p}_i - \mathbf{c}|$ als den Abstand von $\mathbf{c}$ zum Punkt $\mathbf{p}_i \in \mathcal{P}$. Dann erhalten wir für den Radius der SES:

$$r_{min} = \min_{\mathbf{c}\in\mathbb{R}} \mathbf{r}(\mathbf{c}) \quad \text{mit } \mathbf{r}(\mathbf{c}) = \max_{i=1,\dots,n} r_i(\mathbf{c})$$

Den optimalen Mittelpunkt $\mathbf{c}$ approximieren wir mit numerischen Methoden. Wir starten mit einem beliebigen Punkt (z.B. dem Schwerpunkt) und konstruieren dann iterativ bessere Lösungen. In der im $\mathbb{ICOR}$ -System bereits implementierten Berechnung der SES wird der Downhill Simplex Algorithmus von Nelder und Mead[11] verwendet.

Kugel-Kugel-Kollisionstest

Seien $\mathbf{a}(t)$ und $\mathbf{b}(t)$ die Mittelpunkte der beiden Kugeln (für $t \in \{0, 1\}$) und r_a und r_b die jeweiligen Radien. Wenn sich die beiden Kugeln berühren, gilt

$$|\mathbf{a}(t) - \mathbf{b}(t)| = r_a + r_b.$$

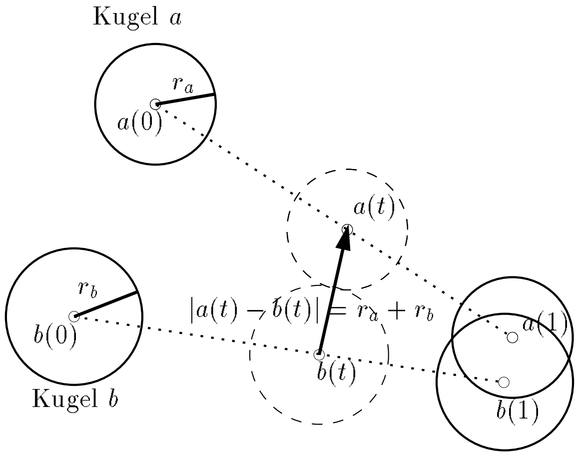

Abbildung 3.15: Berührung der *kleinsten einschließenden Kugeln* zweier Objekte zum Zeitpunkt t.

1. Als erstes testen wir nun, ob sich die Kugeln schon zum Zeitpunkt $t = 0$ durchdringen. Das ist der Fall für

   $$(\mathbf{a}(0) - \mathbf{b}(0))^2 < (r_a + r_b)^2.$$

   Wir melden in diesem Fall also eine Kollision.

2. $(\mathbf{b}(0) - \mathbf{a}(0))^T(\mathbf{a}(1) - \mathbf{a}(0)) > 0$ bedeutet, daß $\mathbf{a}$ sich während des Simulationsschrittes an $\mathbf{b}$ annähert. Gilt also

   $$\big((\mathbf{b}(0) - \mathbf{a}(0))^T(\mathbf{a}(1) - \mathbf{a}(0)) + (\mathbf{a}(0) - \mathbf{b}(0))^T(\mathbf{b}(1) - \mathbf{b}(0))\big) < 0,$$

   so entfernen sich $\mathbf{a}$ und $\mathbf{b}$ und wir erhalten in diesem Schritt keine Kollision.

3. Trifft keiner der vorigen Fälle zu, so müssen wir noch entscheiden, ob es im aktuellen Simulationsschritt zu einer Kollision kommt. Wenn es zur Kollision kommt, so muß es einen Zeitpunkt $0 \leq t \leq 1$ geben, zu dem sich die Oberflächen der Kugeln gerade berühren. Das ist genau dann der Fall, wenn die zu

   $$\Big|\big(\mathbf{a}(0) + t(\mathbf{a}(1) - \mathbf{a}(0))\big) - \big(\mathbf{b}(0) + t(\mathbf{b}(1) - \mathbf{b}(0))\big)\Big| = r_a + r_b \tag{3.16}$$

gehörende quadratische Gleichung eine Lösung mit $0 \leq t \leq 1$ hat. Wir brauchen dazu nicht die exakten Lösungen zu berechnen (Wurzelberechnung!). Es genügt nach Schömer und Thiel[18] eine Vorzeichenbetrachtung der Koeffizienten, um zu entscheiden, ob es eine Lösung zwischen 0 und 1 gibt. Wenn wir Gleichung 3.16 quadrieren und umformen, erhalten wir eine Gleichung der Form

$$\begin{aligned} at^2 + bt + c = 0 \quad \text{mit } a &= \big((\mathbf{a}(1) - \mathbf{a}(0)) - (\mathbf{b}(1) - \mathbf{b}(0))\big)^2 \\ b &= 2\big((\mathbf{a}(1) - \mathbf{a}(0)) - (\mathbf{b}(1) - \mathbf{b}(0))\big)\big(\mathbf{a}(0) - \mathbf{b}(0)\big) \\ c &= (\mathbf{a}(0) - \mathbf{b}(0))^2 - (r_a + r_b)^2. \end{aligned}$$

Falls $b^2 - 4ac \geq 0$, hat die Gleichung reelle Lösungen. Wir machen dann eine Fallunterscheidung nach a:

(a) $a = 0$: Wir erhalten eine Lösung für t zwischen 0 und 1, wenn folgende Bedingung erfüllt ist:

$$(b \geq 0 \wedge b + c \geq 0 \wedge c \leq 0) \vee (b \leq 0 \wedge b + c \leq 0 \wedge c \geq 0)$$

(b) $a > 0$: Hier lautet die Bedingung für eine Lösung zwischen 0 und 1

$$\begin{aligned} &\big((b \leq 0 \vee c \leq 0) \wedge (2a + b \geq 0 \wedge a + b + c \geq 0)\big) \\ \vee &\big((b \leq 0 \wedge c \geq 0) \wedge (2a + b \geq 0 \vee a + b + c \leq 0)\big). \end{aligned}$$

(c) $a < 0$: Gibt es eine Lösung zwischen 0 und 1, so muß gelten:

$$\begin{aligned} &\big((b \geq 0 \wedge c \leq 0) \wedge (2a + b \leq 0 \vee a + b + c \geq 0)\big) \\ \vee &\big((b \geq 0 \vee c \geq 0) \wedge (2a + b \leq 0 \wedge a + b + c \leq 0)\big). \end{aligned}$$

Wir können uns diese Bedingungen veranschaulichen anhand der Funktionswerte und Steigungen an den Stellen 0 und 1 der zur Gleichung gehörenden Parabel.

3.3.2 Auflösen einer Kugel

Nachdem nun eine Berührung/Durchdringung der einschließenden Kugeln festgestellt ist, lösen wir die Kugel, die das eingehüllte Objekt schlechter annähert, auf. Ein Maß für die Güte der Näherung ist z.B. das Verhältnis zwischen dem Volumen des Objekts und dem der zugehörigen Kugel. Die dabei verbleibende Kugel testen wir dann auf Kollisionen mit den Ecken, Flächen und Kanten des zweiten Objekts. Die Kollision eines *features* mit der zweiten Kugel ist eine Voraussetzung für die Kollision des *features* mit einem *feature* des anderen Objekts. Wir müssen die oben beschriebenen *low-level*-Kollisionstests aber mit *allen features* machen, die mit der Kugel in Kollision sind, nicht nur mit dem, das als erstes mit der Kugel in Kontakt gerät.

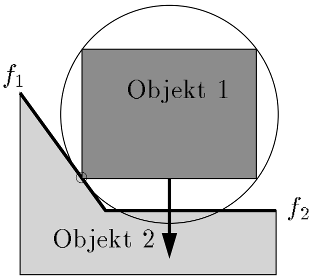

Abbildung 3.16: Beispiel für zwei Kugel-Fläche-Kontakte. Der Kontakt mit f_1 bedeutet auch einen Kontakt der beiden Objekte selbst, der Kontakt mit f_2 (noch) nicht.

Abbildung 3.16 zeigt die umschließende Kugel zu Objekt 1, die mit den Flächen f_1 und f_2 von Objekt 2 in Kontakt kommt. Der Kontakt der Kugel mit f_2 kommt bei der Bewegung von Objekt 1 in Pfeilrichtung zwar zuerst zustande, bedeutet aber keinen Kontakt der Objekte selbst. Der Kontakt der Kugel mit f_1 dagegen erfolgt erst später, ist aber ein echter Kontakt der beiden Objekte.

Ecke-Kugel-Kollisionstest

Sei $\mathbf{a}(t)$ der Mittelpunkt der Kugel um Objekt 1 mit dem Radius r_a und $\mathbf{b}(t)$ die auf Kollision zu testende Ecke von Objekt 2. Der Kollisionstest läuft ähnlich ab wie der Kugel-Kugel-Test.

1. Bei Durchdringung schon zum Zeitpunkt $t = 0$ gilt:
$$(\mathbf{a}(0) - \mathbf{b}(0))^2 < r_a^2$$
2. Wenn sich $\mathbf{b}$ und die Kugel voneinander entfernen und damit im aktuellen Simulationsschritt nicht kollidieren können, gilt die Bedingung
$$\big((\mathbf{b}(0) - \mathbf{a}(0))(\mathbf{a}(1) - \mathbf{a}(0)) + (\mathbf{a}(0) - \mathbf{b}(0))(\mathbf{b}(1) - \mathbf{b}(0))\big) < 0.$$
3. Trifft keiner der vorigen Fälle zu, müssen wir noch folgende Gleichung analog zum letzten Abschnitt auf eine Lösung zwischen 0 und 1 untersuchen.
$$\Big|\mathbf{a}(0) + t(\mathbf{a}(1) - \mathbf{a}(0)) - \big(\mathbf{b}(0) + t(\mathbf{b}(1) - \mathbf{b}(0))\big)\Big| = r_a$$

Fläche-Kugel-Kollisionstest

Sei $\mathbf{a}(t)$ der Mittelpunkt der Kugel um Objekt 1 mit dem Radius r_a und $\mathbf{n}^T(t)\mathbf{x} = n_0(t)$ die Gleichung der Ebene durch die (konvexe) Fläche f mit den Eckpunkten $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k$. Der Kollisionstest kombiniert Elemente aus der Kugel-Kugel-Kollision und der Ecke-Fläche-Kollision.

1. Gilt $\mathbf{n}^T(0)\mathbf{a}(0) - n_0(0) < -r_a$, so liegt die Kugel schon zu Beginn des aktuellen Schrittes ganz auf der „Innenseite“ der Fläche, und es kann zu keiner Kollision kommen.

2. Gilt $\mathbf{n}^T(0)\mathbf{a}(0) - n_0(0) < r_a$, so haben wir evtl. schon zu Beginn eine Durchdringung. Wir müssen dazu noch entscheiden, ob die Kollision mit der Ebene innerhalb oder außerhalb der Fläche stattfindet. Liegt der mögliche Kontaktpunkt (bzw. der auf der gleichen Gerade in Normalenrichtung der Ebene liegende Mittelpunkt der Kugel) im Extrusionskörper der Fläche, so melden wir eine Kollision zurück. Diesen Test führen wir analog zum Ecke-Fläche-Kollisionstest aus.

3. Bei Berührung der Kugel mit der Ebene hat der Mittelpunkt der Kugel gerade Abstand r_a von der Ebene:

   $$\big(\mathbf{n}(0) + t(\mathbf{n}(1) - \mathbf{n}(0))\big)^T \big(\mathbf{a}(0) + t(\mathbf{a}(1) - \mathbf{a}(0))\big) - \big(n_0(0) + t(n_0(1) - n_0(0))\big) = r_a$$

   Hat die daraus resultierende quadratische Gleichung

   $$\begin{aligned} 0 = &t^2\big((\mathbf{n}(1) - \mathbf{n}(0))^T(\mathbf{a}(1) - \mathbf{a}(0))\big) \\ &+t\big(\mathbf{a}(0)^T(\mathbf{n}(1) - \mathbf{n}(0)) + \mathbf{n}(0)^T(\mathbf{a}(1) - \mathbf{a}(0)) - (n_0(1) - n_0(0))\big) \\ &+\big(\mathbf{n}(0)^T\mathbf{a}(0) - n_0(0) - r_a\big) \end{aligned}$$

   eine Lösung zwischen 0 und 1, so kommt es im aktuellen Zeitschritt zum Kontakt. Wir müssen nun noch analog zum letzten Punkt feststellen, ob der Kontakt mit der Ebene innerhalb oder außerhalb der Fläche stattfindet.

Kante-Kugel-Kollisionstest

Zum Kante-Kugel-Kollisionstest benutzen wir einige Elemente, die schon bei der Kante-Kante-Kollision vorgestellt wurden. Seien $\mathbf{a}$ und $\mathbf{b}$ Quelle und Ziel der Kante und $\mathbf{m}$ der Mittelpunkt der Kugel, jeweils zum Zeitpunkt $t = \{0, 1\}$. Den zu Kante $(\mathbf{a}(t), \mathbf{b}(t))$ gehörenden Keil berechnen wir auf gleiche Art wie schon beschrieben.

1. Zuerst testen wir, ob die Kugel auf der „richtigen“ Seite des Keiles liegt

   $$\begin{aligned} &(\mathbf{m}(t) - \mathbf{a}(t))^T\mathbf{n}_f(t) \geq 0, \quad t = \{0, 1\} \\ &(\mathbf{m}(t) - \mathbf{a}(t))^T\mathbf{n}_{f^{rev}}(t) \geq 0 \end{aligned}$$

   und ob es wirklich zur Kollision mit der Kante kommen kann und nicht mit einer der adjazenten Flächen

   $$\begin{aligned} &(\mathbf{m}(t) - \mathbf{a}(t))^T\mathbf{n}_e(t) \leq 0 \\ &(\mathbf{m}(t) - \mathbf{a}(t))^T\mathbf{n}_{e^{rev}}(t) \leq 0. \end{aligned}$$

   Ist nicht jeweils mindestens eine dieser Bedingungen erfüllt, kann es nicht zur Kollision kommen.

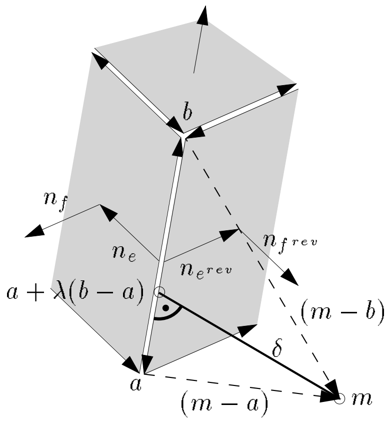

Abbildung 3.17: Notation beim Ecke-Kugel-Kontakt

2. Nun wollen wir den Abstand des Mittelpunktes der Kugel von der Kante berechnen. Dazu brauchen wir zuerst die Ebene mit Normale in umgekehrter Richtung des Lotes von $\mathbf{m}$ auf die Gerade durch $(\mathbf{a}, \mathbf{b})$.

$$\begin{aligned} \mathbf{n}(t) &= \big((\mathbf{b}(t) - \mathbf{a}(t)) \times (\mathbf{m}(t) - \mathbf{a}(t))\big) \times (\mathbf{b}(t) - \mathbf{a}(t)), \quad t = \{0, 1\} \\ \mathbf{n}(t) &= \operatorname{sign}\big((\mathbf{m}(t) - \mathbf{a}(t))^T \mathbf{n}(t)\big)\mathbf{n}(t) \\ \mathbf{n}(t) &= \frac{\mathbf{n}(t)}{|\mathbf{n}(t)|} \end{aligned}$$

Der Abstand zur Kante ergibt sich dann aus

$$d(t) = ((\mathbf{m}(t) - \mathbf{a}(t))^T \mathbf{n}(t), \quad t = \{0, 1\}.$$

Sind $d(0)$ und $d(1)$ größer als der Radius r, so kann es nicht zur Kollision kommen. Ansonsten ermitteln wir aus $d(0)$ und $d(1)$ den möglichen Kollisionszeitpunkt t_c (ähnlich wie in Gleichung (3.2)).

$$t_c = \begin{cases} 0 & \text{falls } d(0) - d(1) < \varepsilon \\ \frac{d(0) - r}{d(0) - d(1)} & sonst \end{cases}$$

Bei $d(0) - d(1) < \varepsilon$ haben wir schon zu Beginn des Schrittes evtl. eine Durchdringung. In diesem Fall setzen wir den Kollisionszeitpunkt auf $t_c = 0$ und gehen dann zu Punkt (3).

3. Jetzt müssen wir noch entscheiden, ob der Lotfußpunkt des Lotes von $\mathbf{m}$ auf die Gerade durch die Kante zum Zeitpunkt t_c *auf* dem Liniensegment $(\mathbf{a}, \mathbf{b})$ liegt. Wir lösen dazu die (überbestimmte) lineare Gleichung

$$\mathbf{a}(t_c) + \lambda(\mathbf{b}(t_c) - \mathbf{a}(t_c)) + \delta\mathbf{n}(t_c) = \mathbf{m}(t_c).$$

Erhalten wir eine eindeutige Lösung mit $0 \leq \lambda \leq 1$, so liegt der Lotfußpunkt vom Mittelpunkt der Kugel aus *auf* der Kante selbst. Der Wert von δ bestimmt dann, ob wir eine Durchdringung bzw. Berührung haben ($\delta \leq r$) oder nicht ($\delta > r$).

3.4 Mögliche Erweiterungen

Bei großen Szenarien erweist sich die Kollisionserkennung als langsamste Komponente der Simulation. Deshalb werden große Anstrengungen unternommen, um die Kollisionserkennung zu beschleunigen. Im folgenden Abschnitt werden einige interessante Ansätze aus der Literatur vorgestellt.

3.4.1 Hierarchien von Hüllkörpern

Wenn man physikalische Systeme betrachtet, stellt man fest, daß die meisten Objekte sich nie oder nur über einen kurzen Zeitraum so nahe sind, daß es überhaupt zur Kollision kommen kann. Daher werden effizient auf Kollisionen zu testende Hüllkörper, wie z.B. die oben vorgestellten *kleinsten einschließenden Kugeln*, verwendet. Die zeitaufwendigen exakten Tests müssen dann nur für Objektpaare, die sich nahe sind, ausgeführt werden. Außer Kugeln werden z.B. achsenparallele Boxen (besonders schnell zu testen) oder nach den Hauptachsen des Objekts ausgerichtete Boxen (nähern das Objekt besonders gut an) verwendet. Da wir nur relativ kleine Systeme betrachten, sind für unseren Fall einfache Hüllkörper völlig ausreichend. Bei großen Szenarien aber bieten sich für diese drei Hüllkörpertypen Hierarchien an, die meist als Bäume verwaltet werden. Hier einige ausgewählte Beispiele:

1. Der *Box-Tree* von ZACHMANN und FELGER[24] ist ein Beispiel für eine Hüllkörperhierarchie, die auf achsenparallenen Boxen aufbaut. Hier wird ein Baum aufgebaut, in dessen inneren Knoten achsenparallele Boxen abgespeichert werden und in dessen Blättern die eigentlichen Polygone, die die Polyeder begrenzen. Die Söhne sind dabei immer in der Vater-Box enthalten. Da die Zuteilung zu einer Box einfach nach Koordinaten vorgenommen wird, sind Kollisionstests mit achsenparallelen Boxen sehr schnell, allerdings werden die umschlossenen Objekte nicht sehr genau angenähert.

2. Bei den *Oriented Bounded Box Trees* von GOTTSCHALK, MIN und MANOCHA[20] wird eine ähnliche Hierarchie aufgebaut. Hier werden die umhüllten Objekte allerdings möglichst dicht durch eine Box angenähert. Die Boxen sind damit nicht mehr achsenparallel und können sich evtl. überlappen. Die Zuordnung zu den umhüllenden Boxen bleibt aber eindeutig. Die genauere Annäherung der umschließenden Boxen erlaubt ein schnelleres Erreichen der eigentlichen Objekte durch den Baum (wir brauchen weniger Hierarchiestufen), wird aber mit einer Verlangsamung des Kollisionstests im Vergleich zu achsenparallelen Boxen erkauft. In dieser Arbeit wird

allerdings durch geschickte Wahl der Koordinatenachsen eine zu vorausgegangenen Ansätzen deutlich höhere Geschwindigkeit erzielt.

3.4.2 Closest Features Algorithmus

Die bisherigen Ansätze versuchen, die Zahl der pro Schritt exakt zu testenden Objektpaare zu verringern. An einem anderen Punkt setzt der *Closest Features Algorithmus* von LIN und CANNY an, den z.B. MIRTICH in seiner Dissertation [15] beschreibt. Hier wird die Zahl der pro Objektpaar zu testenden *feature*-Paare reduziert. Für je zwei Polyeder wird das Paar der nächsten *features* (Ecken, Kanten oder Flächen) verwaltet. Dabei wird ausgenutzt, daß die aktuellen nächsten *features* sehr wahrscheinlich in der Nähe der vorherigen liegen. Wir müssen also jeweils nur so lange die Distanz der vorherigen *closest features* und der jeweils dazu benachbarten *features* testen, bis wir das neue Paar mit der kleinsten Distanz gefunden haben, nicht aber sämtliche *feature*-Paare der beiden Polyeder. Eine wesentlicher Nachteil dieses Ansatzes ist die Beschränkung auf *konvexe* Polyeder. Im nichtkonvexen Fall genügt es nicht, nur die Nachbarn zu betrachten.

3.5 Implementierung

3.5.1 Kollisionserkennung

Wir wollen nicht, daß alle Objekte eines Szenarios bewegbar sind, d.h. Kräften wie z.B. der Gravitation unterliegen. Solche Objekte wären z.B. Wände, Decke und Boden von dargestellten Gebäuden. Daher teilen wir alle Objekte in bewegliche (mobile) und nicht bewegbare (fixierte) Objekte ein. Die Kollisionstests erfolgen in einer Schleife über alle Objektpaare, wobei wir keine Kollisionstests zwischen fixierten Objekten machen müssen.

Die Funktion zur Erkennung von Kontakten zwischen einem Polyederpaar ist dabei wie in Algorithmus 5 definiert. Da jede Kante in der Datenstruktur zweimal vorkommt (in entgegengesetzter Richtung), wird die Kante-Kante-Kollisionserkennung nur für *eine* dieser Kanten aufgerufen.

3.5.2 Datenstrukturen

Jedes Polyeder enthält in seiner Datenstruktur einen Zeiger auf die Liste der Kontakte, an denen es zur Zeit beteiligt ist. Kontakte zwischen bewegbaren und fixierten Objekten werden in die Liste des bewegbaren Polyeders eingetragen. Bei zwei bewegbaren Objekten befindet sich der Kontakt in der Liste des Objekts, das in der Objektliste zuerst kommt. Die Liste enthält alle Daten des Kontaktes, die zur Bearbeitung in der Kollisions-Reaktions-Komponente notwendig sind.

Algorithmus 5 Kollisionserkennung zwischen zwei Polyedern

```
detect_collision(poly1, poly2)
{
  if not ses_collision(poly1, poly2) then
    no collision detected
  else {enclosing spheres collide}
    for all v ∈ poly1→vertexlist do
      for all f ∈ poly2→facelist do
        if vertex_face_coll(v,f) then
          append detected contact to contactlist of poly1
        end if
      end for
    end for

    for all f ∈ poly1→facelist do
      for all v ∈ poly2→vertexlist do
        if vertex_face_coll(v,f) then
          append detected contact to contactlist of poly1
        end if
      end for
    end for

    for all e1 ∈ poly1→edgelist with e1^rev not yet tested do
      for all e2 ∈ poly1→edgelist with e2^rev not yet tested do
        if edge_edge_coll(v,f) then
          append detected contact to contactlist of poly1
        end if
      end for
    end for
  end if
}
```

Wie in Abschnitt 3.1.1 bereits erwähnt, benötigen wir von allen zeitabhängigen Größen der Polyeder zwei „Kopien“, die den Zustand *vor* bzw. *nach* dem aktuellen Zeitschritt wiederspiegeln. Aus diesen beiden Zuständen können wir den Zustand zum errechneten Kollisionszeitpunkt linear interpolieren. Für die Koordinaten der Eckpunkte, des Schwerpunktes und des Mittelpunktes der kleinsten einschließenden Kugel und für die Normalen der Flächen, die das Objekt begrenzen, verwalten wir drei „Kopien“. Die dritte Kopie enthält den Zustand des Objektes zum Start der Simulation. Die aktuelle Position und Orientierung des Objektes wird immer aus dem Startzustand berechnet, um die Akkumulation von Rechenfehlern und damit evtl. eine Verzerrung des Objektes zu vermeiden.

Algorithmus 6 Datenstruktur zur Verwaltung der Kontakten

```
class Contact
{
    public:
        Vec3 point;
            {Kontaktpunkt}
        Vec3 normal;
            {Normale der Kontaktebene (zeigt zu Objekt1)}
        double t;
            {Zeitpunkt des Kontaktes}

        Part *part2;
            {Pointer auf das zweite beteiligte Objekt}
        int type;
            {Typ des Kontaktes}

        union{Vertex *vertex1; Edge *edge1; Face *face1;};
            {Pointer auf das beteiligte feature von Objekt1}
        union{Face *face2; Edge *edge2; Vertex *vertex2;};
            {Pointer auf das beteiligte feature von Objekt2}

        Contact *next;
            {Pointer auf das nächste Listenelement}

        Contact() {}
};
```

Kapitel 4

Impulsbasierte Kollisionsreaktion

Die Simulation der Bewegung eines isolierten Körpers in einem physikalischen System ist relativ einfach, sofern er nicht mit anderen Körpern interagiert und man nur die Einflüsse von externen Kräften wie Gravitation, Luftwiderstand oder vom Benutzer interaktiv verursachten Kräften betrachtet. Die Interaktion verschiedener Objekte eines Systems ist abhängig von der Modellierung der Kontakte zwischen den Körpern. Kontakte lassen sich nicht so genau und eindeutig modellieren wie die Dynamik eines isolierten Körpers. Dabei müssen wir abwägen zwischen Effizienz und Genauigkeit unseres Modells. Da unsere Simulation interaktiv sein soll, müssen wir einige Vereinfachungen machen:

- Wir beschränken uns auf *starre Körper*, d.h. wir erlauben weder Flüssigkeiten noch makroskopisch verformbare Materialien wie z.B. Textilien oder Gummi.

- Wir verzichten auf die Simulation von *Reibung*. Nach BARAFF[2] ist die Berechnung der Kontaktkräfte mit Reibung ein NP-vollständiges Problem. In einem interaktiven System wäre die Simulation von Reibung also nur mit Vereinfachungen des Reibungsmodells (z.B. des empirischen Reibungsmodells von COULOMB) möglich.

- Wir modellieren die Wirkung der *Elastizität* zwischen zwei Körpern, d.h. die mikroskopische Verformung am Kontaktpunkt sowie die dabei stattfindende Impulsübertragung als einzelnen Koeffizienten e. Weil wir uns auf starre Körper beschränken, können wir zwar keine Verformung modellieren, aber durch den Koeffizienten die daraus resultierenden Impulsanteile nach dem Stoß.

Aufgabe der Kollisionsreaktion ist es, Durchdringung an den von der Kollisionserkennung gemeldeten Kontakten zu verhindern. Um das zukünftige Verhalten eines physikalischen Systems vorherzusagen, kommt es darauf an, möglichst realitätsnah zu bestimmen, wie eine Kollision die beteiligten Objekte beeinflußt.

4.1 Vergleich verschiedener Ansätze

Die bisherigen Ansätze lassen sich alle grob in drei Kategorien einteilen. Jede davon hat Vor- und Nachteile in Bezug auf Effizienz und Genauigkeit. Das von uns benutzte impulsbasierte Modell erlaubt die Berechnung der Kollisionsreaktion in interaktiver Zeit und versucht, dabei möglichst große Genauigkeit zu erreichen.

4.1.1 *Constraint*-basierter Ansatz

Wenn es zum Kontakt zwischen Objekten kommt, verhindern hier *Constraints* eine Durchdringung. Diese kann man formulieren als Ungleichungen in Abhängigkeit von den Beschleunigungen der Objekte. Eine solche Ungleichung könnte z.B. sicherstellen, daß sich ein Eckpunkt, der im Extrusionskörper einer Fläche liegt, immer im nicht-negativen Halbraum der durch die Fläche verlaufenden Ebene befindet. Das können wir testen, indem wir den Punkt in die Ungleichung einsetzen, deren Vorzeichen die Lage des Punktes bzgl. der Ebene beschreibt. Wir unterscheiden zwei Methoden der Kontaktmodellierung mittels *Constraints*.

Penalty-Methode

Eine der frühesten Arbeiten zu diesem Ansatz stammt von Moore und Wilhelms[16]. Diese Methode verhindert Durchdringung nicht, sondern setzt am Kontaktpunkt eine harte Feder ein, die die Objekte wieder zu trennen versucht. Je stärker die Durchdringung ist, desto größer ist die gewählte Federkonstante. Wenn sich die Objekte wieder trennen, wird die Feder entfernt. Diese Methode ist aber nicht sehr exakt und benötigt für akzeptable Genauigkeit viele Iterationen. Der Grund dafür sind z.B. große Federkonstanten, die die zu integrierenden Gleichungen numerisch instabil machen. Die Federkonstante muß auch an das jeweilige Objektpaar angepaßt werden (schwere Objekte dringen z.B. bei gleicher Beschleunigung tiefer ein als leichte). Außerdem müssen wir berücksichtigen, daß die Kollisisonstests periodisch ausgeführt werden, und nicht zum *exakten* Kollisionszeitpunkt. Es ist also oft schon zur Durchdringung gekommen. Das heißt, daß wir typischerweise etwas zu große Kontaktkräfte errechnen. Allerdings ist dieses Verfahren relativ einfach zu implementieren und läßt sich auf nicht-starre Körper ausdehnen. Die *Penalty*-Methode ist also absolut ausreichend für Simulationen, die nur korrekt *aussehen* müssen und keine große Genauigkeit benötigen.

Analytische Methode

Der erste, der analytische Methoden zur Berechnung von Kontaktkräften angewendet hat, war Lötstedt[13]. Aus den *Constraints*, die Durchdringung verhindern sollen, wird ein

Lineares Komplementaritätsproblem (LCP) aufgestellt, dessen Lösung die dazu notwendigen Kontaktkräfte angibt. Ein LCP ist ähnlich aufgebaut wie ein lineares Programm, enthält aber eine zusätzliche Komplementaritätsbedingung und hat allgemein die Form:

$$\begin{aligned} \mathbf{a} = \mathbf{A}\mathbf{f} - \mathbf{b} &\geq 0 \\ \mathbf{f} &\geq 0 \\ \mathbf{f}^T\mathbf{a} &= 0 \end{aligned}$$

Dabei wird $\mathbf{A}$ bestimmt durch Massen, Trägheitsmomente und Kontaktgeometrie der beteiligten Körper, $\mathbf{a}$ ist der verallgemeinerte Beschleunigungsvektor (aus den Normalkomponenten der linearen und Winkelbeschleunigung), und $\mathbf{f}$ bestimmt die Normalkomponenten der Kräfte an den jeweiligen Kontaktpunkten. Die Bedingungen $\mathbf{a} \geq 0$ und $\mathbf{f} \geq 0$ verhindern, daß sich die Körper an den Kontaktpunkten aufeinanderzubewegen (und durchdringen) bzw. daß die Kontaktkräfte die Körper zusammenhalten wollen. Die Komplementaritätsbedingung $\mathbf{f}^T\mathbf{a} = 0$ schließlich bedeutet, daß bei einer relativen Beschleunigung von Null eine Kontaktkraft wirkt und der Kontakt bestehen bleibt. Ist die relative Beschleunigung größer als Null, so wirkt keine Kontaktkraft mehr, und der Kontakt löst sich. LCPs kann man z.B. mit dem Lemke-Algorithmus lösen, wobei aber die Effizienz der bekannten Algorithmen mit steigender Anzahl der Kontakte sinkt.

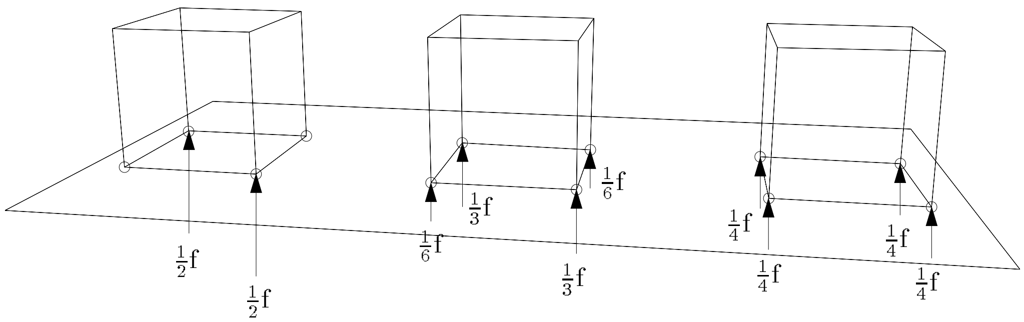

Abbildung 4.1: Beispiel für verschiedene Konfigurationen der Kontaktkräfte in einer degenerierten Kontaktsitualtion

Baraff[3] hat festgestellt, daß die Lösung außerdem bei Reibung oder degenerierten Kontaktsituationen nicht eindeutig ist. Abbildung 4.1 zeigt drei von unendlich vielen möglichen Konfigurationen der Kontaktkräfte an den vier Kontakten, aber alle drei Möglichkeiten ergeben die gleiche resultierende Beschleunigung für den Körper. Im Beisein von Reibung sind auch die Beschleunigungen nicht mehr eindeutig. Der Grund dafür liegt in der Beschränkung auf ideale (starre) Körper und der Verwendung des Reibungsmodells von Coulomb.

Ein weiterer Nachteil der analytischen Methode ist die Abhängigkeit von Kontaktkohärenz. Aus Effizienzgründen (und wegen der fehlenden Eindeutigkeit der Kräfte bei Reibung) ist es nicht möglich, mit jedem Schritt ein „neues“ LCP zu lösen. Man muß von den im vorherigen Schritt berechneten Resultaten ausgehen.

4.1.2 Impulsbasierter Ansatz

Bei impulsbasierten Ansätzen wird Durchdringung nicht durch *Constraints* verhindert, sondern alle Kontaktsituationen werden durch Kollisionen in den Kontaktpunkten simuliert. Zwischen zwei Kollisionen bewegen sich die Objekte auf ballistischen Bahnkurven, die nur durch die Wirkung von Gravitation und evtl. anderen externen Kräften beeinflußt werden.

An dem klassischen Beispiel einer Kugel, die eine Rampe hochrollt, auf einer ballistischen Bahn durch die Luft fliegt, mehrfach auf dem Boden springt, evtl. ein Stück über den Boden gleitet und dann weiterrollt, kann man gut den Unterschied zwischen den verschiedenen Ansätzen sehen. Während eine solche Simulation für *Constraint*-basierte Ansätze wegen der ständig wechselnden Kontaktsituation und dem damit verbundenen Modellwechsel sehr aufwendig ist, werden alle vorkommenden Arten von Kontakten in den impulsbasierten Ansätzen einfach als an den Kontaktpunkten angreifende Impulse behandelt.

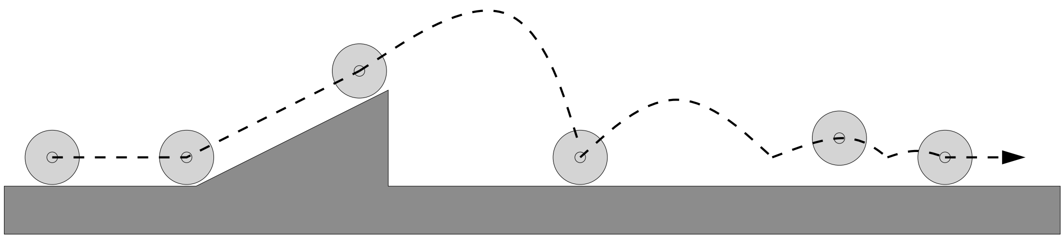

Abbildung 4.2: Verschiedene Kontaktsituationen am Beispiel einer Kugel, die über eine Rampe springt.

Für die Bewegung selber gibt es keine *Constraints*, sie läßt sich also relativ leicht durch Integration der jeweiligen Bewegungsgleichungen berechnen.

Wird während der Bewegung eine Kollision erkannt, so berechnet die Kollisionsreaktionskomponente ein im Kontaktpunkt angreifendes Impulspaar, das für sofortige Änderung der Geschwindigkeiten sorgt (Vorzeichenwechsel in Normalenrichtung der Kontaktebene). Dabei geht man davon aus, daß Geschwindigkeiten/Impulse *vor* und *nach* der Kollision über algebraische Gleichungen miteinander in Beziehung stehen, wobei auch Reibung (z.B. nach dem Coulomb'schen Modell), Energieverlust während der Kollision und Elastizität der beteiligten Objekte mit einbezogen werden müssen.

Eine große Schwäche des impulsbasierten Ansatzes ist die Simulation von ruhenden und gleitenden Kontakten. Ein Objekt, das eigentlich auf einer festen Unterlage ruht, erfährt ständig neue Kollisionen und wird durch die damit hervorgerufenen Impulse wieder davon getrennt. Das führt zu einem Zittern des Objektes an der Oberfläche der Unterlage.

Wir können das Zittern so gering machen, daß es am Bildschirm nicht mehr wahrnehmbar ist (Bewegung kleiner als ein Pixel), aber wir müssen trotzdem in jedem Schritt eine Kollisionsbehandlung von eigentlich ruhenden Kontakten durchführen.

4.1.3 Warum impulsbasierter Ansatz?

Ein ausführlicher Vergleich der verschiedenen Ansätze findet sich z.B. bei MIRTICH[15]. Hier daher nur die wichtigsten Gründe, den impulsbasierten Ansatz zu wählen, erläutert.

- Die analytische Methode benötigt *Kontaktkohärenz*, um in interaktiver Geschwindigkeit arbeiten zu können. Die Notwendigkeit, von früheren Lösungen auszugehen, ist speziell wegen der nicht immer zu garantierenden Eindeutigkeit der berechneten Kontaktkräfte problematisch.

- Der impulsbasierte Ansatz zeichnet sich durch große *Robustheit* aus. Es werden immer eindeutige Impulse für jedes Objekt berechnet, im Gegensatz zu der fehlenden Eindeutigkeit der Kontaktkräfte bei analytischen Ansätzen. Die Impulsverteilung auf die einzelnen Kontakte ist bei degenerierten Kontaktsituationen zwar auch nicht eindeutig, aber alle möglichen Verteilungen resultieren in den gleichen Impulsen für die beteiligten Objekte. Außerdem gibt es beim impulsbasierten Ansatz keine numerisch instabilen Differentialgleichungen wie bei der *Penalty*-Methode.

- Im Gegensatz zur *Penalty*-Methode kommt es nicht wirklich zur Durchdringung der kollidierenden Objekte. Außerdem ist die Kontaktbehandlung unabhängig vom Gewicht der beteiligten Objekte.

- Alle Kontakte können *lokal* am jeweiligen Kontaktpunkt betrachtet werden. Wir benötigen dazu nicht den globalen Zustand des Systems, sondern nur die Daten der jeweiligen Kontakte, d.h. Kontaktpunkt, Normale der Trennebene und Kontaktzeit.

- Wir können alle Kontaktmodi (gleiten, kollidieren, ruhen, rollen) mit dem *gleichen* Kontaktmodell behandeln. Es sind keine Übergänge zwischen verschiedenen Modellen notwendig. Das erlaubt auch die gleichzeitige Behandlung von gleichzeitig stattfindenden Kontakten und verlangt keine getrennte Reaktion je nach Kontaktmodus.

Neben diesen Vorzügen hat der impulsbasierte Ansatz aber auch eine große Schwäche:

- Die Behandlung von ruhenden bzw. gleitenden Kontakten ist unzureichend. Solche Kontakte müssen nach der Behandlung wieder gelöst und im folgenden Schritt wieder neu erkannt und behandelt werden.

Nachdem wir nun die verschiedenen möglichen Ansätze kurz miteinander verglichen haben, beschäftigen wir uns im folgenden mit der Entwicklung einer auf Impulsen basierenden Kollisionsreaktionskomponente.

4.2 Vorgehensweise bei Einzelkontakten

Ein Simulationsschritt besteht grob aus folgenden Phasen:

1. Differentialgleichungen integrieren und die berechneten Bewegungen durchführen.
2. Kollisionserkennung ausführen und Punkt, Ebene und Zeit der Kontakte bestimmen.
3. Zeitpunkt der *frühesten* Kollision feststellen und System auf den Zustand zu diesem Zeitpunkt t_c bringen. Damit werden evtl. später im gleichen Schritt stattfindende Kollisionen nicht verpaßt, sondern im folgenden Schritt erkannt und behandelt.
4. Übertragenen Impuls und die daraus resultierende Änderung der Winkel- und linearen Geschwindigkeiten bestimmen.
5. Impulse und Drehimpulse *nach* der Kollision berechnen.
6. Kollidierte Objekte auf Sicherheitsabstand voneinander bringen.

Zur Berechnung der Kollision machen wir einige Annahmen:

1. Infinitesimal kleine Kollisionszeit:
   - Die Konfiguration der Objekte des Systems ändert sich während der Kollision nicht.
   - Der Effekt, den die beiden an der Kollision beteiligten Objekte $\mathcal{P}_1$ und $\mathcal{P}_2$ aufeinander ausüben, kann durch einen Impuls beschrieben werden, der senkrecht auf der Kontaktebene zwischen $\mathcal{P}_1$ und $\mathcal{P}_2$ steht und definitionsgemäß in Richtung von $\mathcal{P}_1$ zeigt.
2. Anforderungen an den berechneten Impuls:
   - Der berechnete Impuls soll eine Durchdringung der Objekte verhindern.
   - Die aus dem Impuls resultierenden relativen Kontaktgeschwindigkeiten sollen die Objekte voneinander wegbewegen. Die relative Kontaktgeschwindigkeit errechnet sich aus der Differenz der Geschwindigkeiten der beiden Objekte im Kontaktpunkt.

$$\mathbf{u} = \mathbf{u}_2 - \mathbf{u}_1$$
$$\mathbf{u}_i = \mathbf{v}_i + \boldsymbol{\omega}_i \times \mathbf{r}_i \quad i \in \{1,2\}$$

     Die Kontaktgeschwindigkeiten $\mathbf{u}_i$ ergeben sich nach Gleichung (2.4). $\mathbf{r}_i$ ist der Vektor vom Schwerpunkt des Objektes zum Kontaktpunkt.

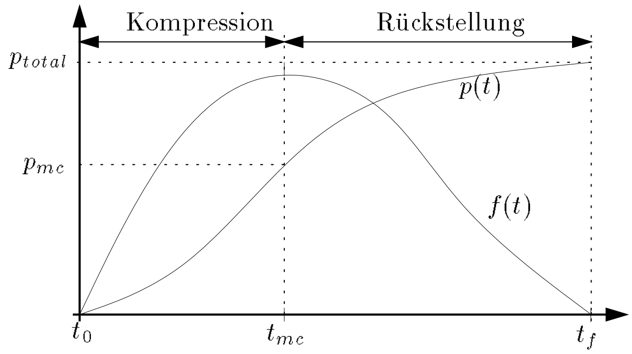

Abbildung 4.3: Unterteilung einer Kollision in Kompressions- und Rückstellungsphase nach der Poisson'schen Hypothese.

3. Poisson'sche Hypothese:

$\mathbf{p}_{total}$ ist die Normalkomponente des Gesamtimpulses, der während der Kollision von $\mathcal{P}_2$ auf $\mathcal{P}_1$ übertragen wird, während $\mathbf{p}_{mc}$ die Normalkomponente dieses Impulses bis zum Zeitpunkt der maximalen Kompression darstellt. t_0 und t_f sind Anfangs- und Endpunkt der Kollision, während t_{mc} den Zeitpunkt der maximalen Kompression bezeichnet. Zu diesem Zeitpunkt verschwindet die relative Kontaktgeschwindigkeit in Normalenrichtung. Die Funktionen $f(t)$ und $p(t) = \int f(t)$ beschreiben den Verlauf der Kontaktkraft und des übertragenen Impulses während der Kollision. Die Poisson'sche Hypothese

$$\mathbf{p}_{total} = (1 + e)\mathbf{p}_{mc} \tag{4.1}$$

setzt $\mathbf{p}_{total}$ und $\mathbf{p}_{mc}$ zueinander in Beziehung. Der Faktor e ist materialabhängig und bezeichnet das Verhältnis der Impulsübertragung. Er liegt im Intervall $[0, 1]$ und heißt Koeffizient der Rückstellung oder auch Elastizitätskoeffizient. Für $e = 1$ haben wir einen *vollkommen elastischen Stoß*, d.h. der Impuls wird vollständig übertragen, und keine Energie geht verloren. Mit $e = 0$ ist der Stoß *plastisch* bzw. *vollkommen unelastisch*. Die Objekte trennen sich nicht mehr nach der Kollision. Es genügt, jeweils nur die Komponente des Impulses zu betrachten, die in Normalenrichtung der Trennebene wirkt. Die Tangentialkomponente verschwindet für den reibungsfreien Fall (vgl. auch CANNY und MIRTICH[5]).

4.3 Impulsberechnung bei Einzelkontakten

Wir wollen nun für einen einzelnen Kontakt die Impulsübertragung von Objekt 2 auf Objekt 1 in algebraische Gleichungen fassen. Nach dem Impulserhaltungssatz muß im verlustfreien Fall die Summe der Impulse aller Objekte des Systems vor und nach dem Stoß gleich sein. Für einen speziellen Kontaktpunkt müssen wir nur die Impulse der beiden beteiligten Objekte betrachten, alle anderen Objekte bleiben davon unbeeinflußt. Seien $\mathbf{v}_i^b$

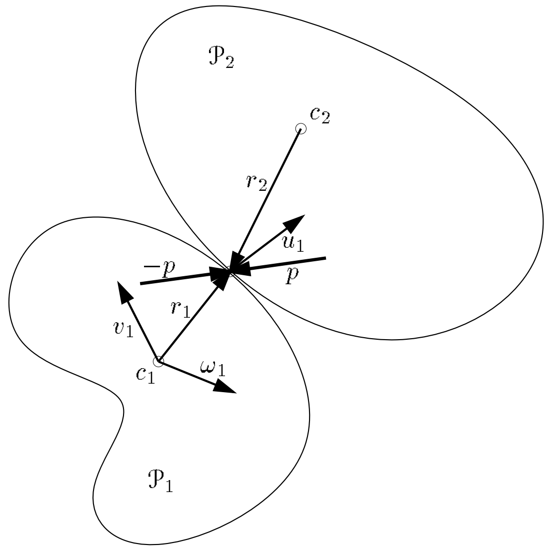

Abbildung 4.4: Mögliche Kollision zwischen zwei Objekten.

und $\boldsymbol{\omega}_i^b$ lineare und Winkelgeschwindigkeit des Objektes i — jeweils zum Zeitpunkt $t = t_c$ — vor (*before*) und $\mathbf{v}_1^a$ und $\boldsymbol{\omega}_i^a$ nach (*after*) dem Stoß. $\mathbf{p}$ ist der übertragene Impuls. Dann erhalten wir für die Impulse nach dem Stoß:

$$m_1\mathbf{v}_1^a = m_1\mathbf{v}_1^b + \mathbf{p} \tag{4.2}$$
$$m_2\mathbf{v}_2^a = m_2\mathbf{v}_2^b - \mathbf{p} \tag{4.3}$$
$$\mathbf{I}_1\boldsymbol{\omega}_1^a = \mathbf{I}_1\boldsymbol{\omega}_1^b + \mathbf{r}_1 \times \mathbf{p} \tag{4.4}$$
$$\mathbf{I}_2\boldsymbol{\omega}_2^a = \mathbf{I}_2\boldsymbol{\omega}_2^b - \mathbf{r}_2 \times \mathbf{p} \tag{4.5}$$

Der übertragene Impuls zeigt definitionsgemäß in Richtung von Objekt 1. Die Geschwindigkeiten erhalten wir mit

$$\mathbf{v}_i^b = \frac{\mathbf{p}_i^b}{m_i}$$
$$\boldsymbol{\omega}_i^b = \mathbf{I}_i^{-1}\mathbf{L}_i^b$$

aus den Impulsen zur Zeit $t = t_c$, die wir durch Lösen der zugehörigen Differentialgleichungen berechnet haben. Damit haben wir noch dreizehn Unbekannte. Dreizehn und nicht fünfzehn Variablen, weil $\mathbf{p}$ nur einen Normalenanteil hat und der Tangentialanteil verschwindet (vgl. oben). Für $\mathbf{p}$ gilt also

$$\mathbf{p} = |\mathbf{p}|\mathbf{n} =: p\mathbf{n}.$$

Da wir zur Bestimmung dieser Variablen aber nur zwölf Gleichungen haben, fehlt noch eine Beziehung. McMillan beschreibt in [14] die Impulsübertragung für den bei uns relevanten reibungsfreien Fall. Beim vollkommen plastischen Stoß ist die Komponente der relativen Kontaktgeschwindigkeit in Normalenrichtung $\mathbf{n}$ der Trennebene im Moment der maximalen Kompression gleich Null. Die Tangentialkomponenten dagegen bleiben durch den Stoß unverändert. Wir erhalten also die Beziehung:

$$\mathbf{n}^T(\mathbf{u}_2^a - \mathbf{u}_1^a) = \mathbf{n}^T\big(\mathbf{v}_2^a + \boldsymbol{\omega}_2^a \times \mathbf{r}_2 - (\mathbf{v}_1^a + \boldsymbol{\omega}_1^a \times \mathbf{r}_1)\big) = 0$$

Hier setzen wir die aus den Gleichungen (4.2) – (4.5) erhaltenen Geschwindigkeiten ein und erhalten schließlich:

$$\begin{aligned} 0 &= \mathbf{n}^T\Big(\mathbf{v}_2^b - \frac{p\mathbf{n}}{m_2} + \big(\boldsymbol{\omega}_2^b - \mathbf{I}_2^{-1}(\mathbf{r}_2 \times p\mathbf{n})\big) \times \mathbf{r}_2 - \mathbf{v}_1^b - \frac{p\mathbf{n}}{m_1} - \big(\boldsymbol{\omega}_1^b - \mathbf{I}_1^{-1}(\mathbf{r}_1 \times p\mathbf{n})\big) \times \mathbf{r}_1\Big) \\ &= \mathbf{n}^T\big(\mathbf{v}_2^b + \boldsymbol{\omega}_2^b \times \mathbf{r}_2 - (\mathbf{v}_1^b + \boldsymbol{\omega}_1^b \times \mathbf{r}_1)\big) \\ &\quad - p\big(\frac{1}{m_1} + \frac{1}{m_2} + (\mathbf{r}_2 \times \mathbf{n})\mathbf{I}_2^{-1}(\mathbf{r}_2 \times \mathbf{n}) + (\mathbf{r}_1 \times \mathbf{n})\mathbf{I}_1^{-1}(\mathbf{r}_1 \times \mathbf{n})\big) \end{aligned}$$

Diese Gleichung lösen wir nun nach $\mathbf{p}$ auf und erhalten:

$$p = \frac{\mathbf{n}^T\big(\mathbf{v}_2^b + \boldsymbol{\omega}_2^b \times \mathbf{r}_2 - (\mathbf{v}_1^b + \boldsymbol{\omega}_1^b \times \mathbf{r}_1)\big)}{\big(\frac{1}{m_1} + \frac{1}{m_2} + (\mathbf{r}_2 \times \mathbf{n})\mathbf{I}_2^{-1}(\mathbf{r}_2 \times \mathbf{n}) + (\mathbf{r}_1 \times \mathbf{n})\mathbf{I}_1^{-1}(\mathbf{r}_1 \times \mathbf{n})\big)} \tag{4.6}$$

Im unelastischen Fall entspricht der so berechnete Wert für den bis zum Zeitpunkt der maximalen Kompression übertragenen Impuls dem insgesamt übertragenen Impuls, da sich die Objekte nach dem Stoß nicht mehr trennen und keine weitere Impulsübertragung mehr stattfindet. Für den elastischen Fall dagegen müssen wir den in Gleichung (4.6) berechneten Impuls noch mit $(e+1)$ multiplizieren, um den gesamten übertragenen Impuls zu erhalten. e ist dabei der Rückstellungskoeffizient. Diese Formel entspricht damit genau der Aussage der Poisson'schen Hypothese (4.1).

Die Impulse *nach* dem Stoß, mit denen wir den nächsten Simulationsschritt beginnen, erhalten wir dann analog zu den Gleichungen (4.2) – (4.5) als

$$\begin{aligned} \mathbf{p}_1^a &= \mathbf{p}_1^b + p\mathbf{n} \\ \mathbf{p}_2^a &= \mathbf{p}_2^b - p\mathbf{n} \\ \mathbf{L}_1^a &= \mathbf{L}_1^b + \mathbf{r}_1 \times (p\mathbf{n}) \\ \mathbf{L}_2^a &= \mathbf{L}_2^b - \mathbf{r}_2 \times (p\mathbf{n}). \end{aligned}$$

Die obige Formel zur Impulsberechnung gilt im Prinzip auch, wenn eines der beiden Objekte fixiert ist und wir für dieses Objekt von unendlich großer Masse und Trägheit ausgehen. Da wir im Rechner aber keine unendlich großen Zahlen darstellen können, betrachten wir den Fall der Impulsübertragung zwischen bewegtem und unbewegbaren Objekten gesondert.

Die Geschwindigkeiten für fixierte Objekte sind wegen der unendlich großen Masse und Trägheit immer gleich Null, unabhängig von der (endlichen) Größe des Impulses. Unter diesen Annahmen vereinfacht sich die Berechnung des übertragenen Impulses:

$$\mathbf{p} = (1+e)\frac{\mathbf{n}^T\big(0 - (\mathbf{v}_1^b + \boldsymbol{\omega}_1^b \times \mathbf{r}_1)\big)}{\big(\frac{1}{m_1} + 0 + 0 + (\mathbf{r}_1 \times \mathbf{n})\mathbf{I}_1^{-1}(\mathbf{r}_1 \times \mathbf{n})\big)} = (1+e)\frac{-\mathbf{n}^T(\mathbf{v}_1^b + \boldsymbol{\omega}_1^b \times \mathbf{r}_1)}{\big(\frac{1}{m_1} + (\mathbf{r}_1 \times \mathbf{n})\mathbf{I}_1^{-1}(\mathbf{r}_1 \times \mathbf{n})\big)}$$

Die Impulse *nach* dem Stoß erhalten wir dann analog zum obigen Vorgehen. Der auf das fixierte Objekt übertragene Impuls bewirkt dabei wegen der unendlich großen Dichte keine Bewegung.

4.4 Erweiterung auf mehrere Kontakte

Die Kontaktbehandlung im vorigen Abschnitt geht davon aus, daß es nur Einzelkontakte gibt, die sich nicht gegenseitig beeinflussen, also nie mehrere Kontakte zwischen sich beeinflussenden Objekten exakt *gleichzeitig* auftreten. Diese Annahme ist für die meisten Kontaktsituationen ausreichend, da Kontakte im Laufe der Simulation im allgemeinen nicht wirklich gleichzeitig auftreten, sondern (zumindest geringfügig) zeitversetzt. Es gibt aber auch Situationen, in denen die Behandlung von Mehrfachkontakten notwendig wird.

Das passiert zum Beispiel, wenn ein Objekt mit einer senkrecht auf der Bewegungsrichtung stehenden Fläche auf eine parallele Fläche eines zweiten Objektes trifft. Solche Situationen kommen hauptsächlich bei vorgegebenen Startbedingungen vor, später im Laufe der Simulation ist exakte Parallelität sehr viel seltener.

Ein weiterer Fall von Mehrfachkontakten sind ruhende oder gleitende Fläche-Fläche-Kontakte. Werden Stöße simuliert, die nicht vollkommen elastisch sind, kommt es durch den Energieverlust bei Objekten, die auf eine feste Unterlage (Tisch, Boden usw.) treffen, eventuell irgendwann zu Mehrfachkontakten, deren Zeitdifferenz immer kleiner wird und schließlich unter die Rechengenauigkeit sinkt. Solche Kontakte werden dann als gleichzeitige Mehrfachkontakte erkannt. Im folgenden werden wir uns mit der Behandlung solcher Mehrfachkontakte beschäftigen.

Zunächst beschränken wir uns auf Mehrfachkontakte zwischen bewegten und fixierten Objekten. Das macht die Betrachtung der Kontakte besonders einfach, da die Daten der fixierten Objekte nur in Form der Koordinaten des Kontaktpunktes in die Gleichungen eingehen. Der Ansatz läßt sich aber problemlos auf beliebig komplizierte Kontaktsituationen erweitern.

Die relativen Kontaktgeschwindigkeiten am i-ten Kontakt ergeben sich nach dem letzten Abschnitt, wobei sich die Formel vereinfacht, da die Geschwindigkeiten für das zweite (unbewegbare) Objekt gleich Null sind:

$$\begin{aligned}\mathbf{u}_i^b &= \mathbf{v}^b + \boldsymbol{\omega}^b \times \mathbf{r}_i \\ \mathbf{u}_i^a &= \mathbf{v}^a + \boldsymbol{\omega}^a \times \mathbf{r}_i\end{aligned}$$

In den Kontaktpunkten gilt für die Normalkomponenten der Kontaktgeschwindigkeiten nach dem vorigen Abschnitt mit verschwindenden Geschwindigkeiten der fixierten Objekte:

$$\begin{aligned} \mathbf{n}_i^T \mathbf{u}_i^a &= \mathbf{n}_i^T (\mathbf{u}_i^b + \Delta \mathbf{u}_i) = -e \mathbf{n}_i^T \mathbf{u}_i^b \\ \Longleftrightarrow \mathbf{n}_i^T \Delta \mathbf{u}_i &= -(1+e) \mathbf{n}_i^T \mathbf{u}_i^b \end{aligned} \tag{4.7}$$

Für $\Delta \mathbf{u}_i$ gilt wegen des Distributivgesetzes für Addition und Kreuzprodukt

$$\Delta \mathbf{u}_i = \Delta \mathbf{v} + \Delta \boldsymbol{\omega} \times \mathbf{r}_i, \tag{4.8}$$

wobei sich die gesamte Änderung der linearen Geschwindigkeit $\Delta \mathbf{v}$ als die Summe der in den k Kontaktpunkten jeweils in Normalenrichtung der Trennebenen auftretenden Geschwindigkeiten ergibt.

$$\Delta \mathbf{v} = \sum_{1 \le i \le k} \frac{p_i}{m} \mathbf{n}_i = \frac{1}{m} \sum_{1 \le i \le k} p_i \mathbf{n}_i \tag{4.9}$$

Analog ergibt sich $\Delta \boldsymbol{\omega}$ als Summe der aus den Rotationen in den Kontaktpunkten resultierenden Winkelgeschwindigkeiten:

$$\Delta \boldsymbol{\omega} = \sum_{1 \le i \le k} \mathbf{I}^{-1} (\mathbf{r}_i \times p_i \mathbf{n}_i) = \mathbf{I}^{-1} \sum_{1 \le i \le k} (\mathbf{r}_i \times p_i \mathbf{n}_i) \tag{4.10}$$

Die p_i bezeichnen jeweils die nach dem Stoß an den Kontaktpunkten in Normalenrichtung angreifenden Impulsanteile.

Diese Formeln wollen wir nun zur Berechnung der Impulse in eine geschlossene Form bringen. Wir definieren den verallgemeinerten Geschwindigkeitsvektor $\Delta \mathbf{u} := \begin{pmatrix} \Delta \mathbf{v} \\ \Delta \boldsymbol{\omega} \end{pmatrix} \in \mathbb{R}^6$ und erhalten nach (4.9) und (4.10) die Gleichung:

$$\begin{aligned} \Delta \mathbf{u} &= \mathbf{M}^{-1} \mathbf{J} \mathbf{p} \qquad (4.11) \\ \text{mit } \mathbf{M} &= \begin{pmatrix} m\mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \in \mathbb{R}^{6 \times 6} \\ \mathbf{J} &= \begin{pmatrix} \mathbf{n}_1 & \mathbf{n}_2 & \cdots & \mathbf{n}_k \\ \mathbf{r}_1 \times \mathbf{n}_1 & \mathbf{r}_2 \times \mathbf{n}_2 & \cdots & \mathbf{r}_k \times \mathbf{n}_k \end{pmatrix} \in \mathbb{R}^{6 \times k} \\ \mathbf{p} &= (p_1, \ldots, p_k)^T \in \mathbb{R}^k \end{aligned}$$

Die Matrix $\mathbf{M}$ stellt die verallgemeinerte Massenmatrix dar mit der 3×3-Einheitsmatrix $\mathbf{E}$, und $\mathbf{p}$ ist der Vektor der in den Kontaktpunkten wirkenden Impulsanteile in Normalenrichtung. Matrix $\mathbf{J}$ beschreibt die geometrische Konstellation, also die jeweiligen Normalenrichtungen und über $\mathbf{r}_i$ die Lage des Kontaktpunktes bzgl. des Schwerpunktes.

Aus (4.7) folgt damit das Gleichungssystem:

$$\mathbf{J}^T \mathbf{M}^{-1} \mathbf{J} \mathbf{p} = -(1+e) \mathbf{J}^T \mathbf{u}^b, \quad \mathbf{u}^b = \begin{pmatrix} \mathbf{v}^b \\ \boldsymbol{\omega}^b \end{pmatrix} \tag{4.12}$$

Es bleibt dazu noch zu zeigen, daß

$$\mathbf{J}^T \Delta\mathbf{u} = \begin{pmatrix} \mathbf{n}_1^T \Delta\mathbf{u}_1 \\ \vdots \\ \mathbf{n}_k^T \Delta\mathbf{u}_k \end{pmatrix} \text{ bzw. } \mathbf{J}^T \mathbf{u} = \begin{pmatrix} \mathbf{n}_1^T \mathbf{u}_1^b \\ \vdots \\ \mathbf{n}_k^T \mathbf{u}_k^b \end{pmatrix}$$

gilt. Zeilenweise erhalten wir mit der Beziehung $\mathbf{a}^T(\mathbf{b} \times \mathbf{c}) = (\mathbf{c} \times \mathbf{a})^T \mathbf{b}$ jeweils:

$$(\mathbf{J}^T \Delta\mathbf{u})_i = \mathbf{n}_i^T \Delta\mathbf{v} + (\mathbf{r}_i \times \mathbf{n}_i)^T \Delta\boldsymbol{\omega} = \mathbf{n}_i^T \Delta\mathbf{v} + \mathbf{n}_i^T(\boldsymbol{\omega} \times \mathbf{r}_i) = \mathbf{n}_i^T \Delta\mathbf{u}_i$$

□

Analog läßt sich die zweite Beziehung zeigen.

Um die gesuchten Impulse zu erhalten, suchen wir nun ein Verfahren, um Gleichungssysteme der Form $\mathbf{A}\mathbf{x} = \mathbf{b}$ zu lösen. In unserem Fall entspricht der Vektor $\mathbf{b}$ der rechten Seite von Gleichung (4.12), der gesuchte Lösungsvektor $\mathbf{x}$ dem Impulsvektor $\mathbf{p}$, und für $\mathbf{A}$ gilt:

$$\mathbf{A} := \mathbf{J}^T \mathbf{M}^{-1} \mathbf{J} = \left(\frac{1}{m} \mathbf{n}_i^T \mathbf{n}_j + (\mathbf{r}_i \times \mathbf{n}_i)^T \mathbf{I}^{-1} (\mathbf{r}_j \times \mathbf{n}_j) \right)_{(i,j)} \tag{4.13}$$

Unter anderen finden sich die Algorithmen zu den beiden folgenden Verfahren z.B. bei Press et al. in *Numerical Recipes in C* [17].

Cholesky-Zerlegung

Als einfachstes und schnellstes Verfahren bietet sich die Cholesky-Zerlegung an. Matrix $\mathbf{A}$ wird zerlegt in das Produkt aus der unteren Dreiecksmatrix $\mathbf{L}$ und deren Transponierte. Es gilt:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

$$\mathbf{L}_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} \mathbf{L}_{ik}^2 \right)^{\frac{1}{2}} \qquad \mathbf{L}_{ji} = \frac{1}{\mathbf{L}_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} \mathbf{L}_{ik} \mathbf{L}_{jk} \right)$$

Da $\mathbf{L}$ eine Dreiecksmatrix ist, ist das Rückwärtseinsetzen zur Bestimmung von $\mathbf{x}$ besonders einfach. Die Cholesky-Zerlegung verlangt allerdings, daß die Matrix $\mathbf{A}$ *symmetrisch* und *positiv definit* ist. Die Symmetrie ist in unserem Fall immer erfüllt, aber für überbestimmte Kontaktsituationen (redundante Kontakte) ist $\mathbf{A}$ nicht positiv definit. Zum Beispiel ist die Kontaktregion eines Fläche-Fläche-Kontaktes durch drei Kontaktpunkte eindeutig bestimmt. Haben wir aber mehr als drei Kontakte gleichzeitig (z.B. bei einer Rechteckfläche), ist die Matrix $\mathbf{A}$ singulär und damit nicht mehr positiv definit.

Singulärwert-Zerlegung

Die Singulärwert-Zerlegung oder *Singular Value Decomposition* (SVD) ist ein numerisch sehr robustes Verfahren, das auch für singuläre und numerisch schlecht konditionierte Matrizen eine Lösung berechnet. Die SVD basiert auf dem folgenden Satz aus der linearen Algebra:

Satz 4.1. *Jede $M \times N$-Matrix $\mathbf{A}$ mit $M \geq N$ kann geschrieben werden als ein Produkt aus einer spalten-orthonormalen Matrix $\mathbf{U} \in \mathbb{R}^{M \times N}$, einer $M \times N$-Diagonalmatrix mit nichtnegativen Elementen (*singular values*) und der Transponierten einer orthonormalen Matrix $\mathbf{V} \in \mathbb{R}^{N \times N}$.*

$$\mathbf{A} = \mathbf{U} \begin{pmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_N \end{pmatrix} \mathbf{V}^T$$

In unserem Fall sind die Matrix $\mathbf{A}$ und damit auch $\mathbf{U}$, $\mathbf{W}$ und $\mathbf{V}$ quadratisch. Daher läßt sich die Inverse als

$$\mathbf{A}^{-1} = \mathbf{V}\mathbf{W}^{-1}\mathbf{U}^T, \quad \mathbf{W}^{-1} = \begin{pmatrix} \frac{1}{w_1} & & \\ & \ddots & \\ & & \frac{1}{w_N} \end{pmatrix}$$

schreiben. Ein Maß für die numerische Konditionierung der Matrix ist das Verhältnis aus größtem und kleinstem Singulärwert. Für singuläre Matrizen (mindestens eines der w_i Null), ist die Konditionierungszahl unendlich. Im Fall einer singulären Matrix $\mathbf{A}$ sind die Begriffe *Kern* und *Bild* der linearen Abbildung zu $\mathbf{Ax} = \mathbf{b}$ für die Berechnung einer Lösung wichtig. Der Kern ist der Unterraum aller Vektoren $\mathbf{x}$, die durch Multiplikation mit $\mathbf{A}$ auf $\mathbf{0}$ abgebildet werden. Außerdem gibt es einen Unterraum aus Vektoren $\mathbf{b}$, die durch $\mathbf{A}$ „erreicht" werden können, d.h. für die es Vektoren $\mathbf{x}$ gibt, die durch Multiplikation mit $\mathbf{A}$ darauf abgebildet werden. Dieser Unterraum heißt Bild der Abbildung und seine Dimension heißt *Rang*. Die Summe von Rang und Dimension des Kerns ergibt gerade die Dimension N der Abbildung.

Wenn $\mathbf{b} = \mathbf{0}$ ist, so ist jede Spalte von $\mathbf{V}$, deren zugehöriges w_j Null ist, eine Lösung. Im für uns relevanten Fall $\mathbf{b} \neq \mathbf{0}$ hat die Gleichung genau dann eine Lösung $\mathbf{x}$, wenn $\mathbf{b}$ im Bild von $\mathbf{A}$ liegt. Ist $\mathbf{A}$ allerdings singulär, so ist die Lösung nicht eindeutig. Jede Linearkombination aus Vektoren des Kerns, addiert zu einer Lösung $\mathbf{x}$, ergibt ebenfalls eine Lösung des Gleichungssystems. Wie schon in Abschnitt 4.1.1 und bei MIRTICH[15] gesehen, gibt es bei redundanten Kontakten viele mögliche Verteilungen der Kontaktkräfte bzw. -impulse auf die Kontaktpunkte, die allerdings alle die gleiche resultierende Bewegung der beteiligten Objekte zur Folge haben. Die fehlende Eindeutigkeit der Impulse beeinträchtigt also nicht die Eindeutigkeit der berechneten Geschwindigkeiten. Wir können daher eine beliebige der möglichen Lösungen wählen. Wir entscheiden uns wegen der größeren numerischen Robustheit für die Lösung $\mathbf{x}$, die die Länge $|\mathbf{x}|$ minimiert. Dieses $\mathbf{x}$ erhalten wir, indem wir für $\frac{1}{w_j}$ Null einsetzen, wenn $w_j = 0$ gilt. Der Vektor $\mathbf{x}$ errechnet sich dann aus

$$\mathbf{x} = \mathbf{V}\mathbf{W}^{-1}(\mathbf{U}^T\mathbf{b}) \tag{4.14}$$

Der Beweis hierzu findet sich bei Press et al.[17].

In einem realen physikalischen System kann man einem Objekt zu jedem Zeitpunkt, also auch vor bzw. nach Kollisionen, Winkel- und lineare Geschwindigkeit bzw. Dreh- und linearen Impuls zuordnen. Das bedeutet, daß in unserer Simulation das Gleichungssystem immer (mindestens) eine Lösung haben muß. Durch Rundungsfehler könnte es aber vorkommen, daß $\mathbf{b}$ nicht *genau* im Bildbereich von $\mathbf{A}$ liegt. Der von uns verwendete Algorithmus „konstruiert" in diesem Fall aus Gleichung (4.14) eine Lösung. Der so konstruierte Vektor $\mathbf{x}$ löst das Gleichungssystem $\mathbf{A}\mathbf{x} = \mathbf{b}$ zwar nicht genau, berechnet aber einen Vektor, der $|\mathbf{A}\mathbf{x} - \mathbf{b}|$ minimiert. Der Beweis funktioniert ähnlich wie der vorige und steht wiederum bei Press[17].

Ein weiteres numerisches Problem gibt es für Werte w_j, die so klein sind, daß sich Rundungsfehler auswirken können. Da wir in $\mathbf{W}^{-1}$ mit dem Kehrwert dieser w_j rechnen, können sich diese Fehler deutlich bemerkbar machen. Es ist in einem solchen Fall robuster, die w_j zu nullieren. Das bedeutet im Prinzip, daß die entsprechende Gleichung zur Berechnung des Vektors $\mathbf{x}$ nicht benutzt wird. Sie wäre wegen den sich eventuell auswirkenden Rundungsfehlern ohnehin bestenfalls nutzlos, könnte aber andererseits die korrekte Lösung des Gleichungssystems beeinträchtigen.

Mehrfachkontakte zwischen beliebigen Objekten

Die Behandlung von Mehrfachkontakten ist bisher noch nicht optimal. Für das in Abbildung 4.5 dargestellte System z.B. würde es nicht ausreichen, nur jeweils *Paare* von Objekten zu betrachten. In der gezeigten Situation beeinflussen sich alle Objekte gegenseitig. Die Behandlung von Mehrfachkontakten kann beliebig kompliziert werden, da es in der Realität keine Beschränkung der Anzahl der Kontakte oder beteiligten Objekte gibt. Das System aus Abbildung 4.6 z.B. würde sich beliebig erweitern lassen und damit mit wachsender Zahl der beteiligten Objekte komplexer werden, da eine gleichzeitige Betrachtung *aller* beteiligten Objekte erforderlich ist, um jeweils die korrekten Impulse bzw. Geschwindigkeiten berechnen zu können.

Das oben beschriebene Gleichungssystem wird so einfach, weil darin nur Daten wie Masse, Trägheitsmatrix und Geschwindigkeiten des bewegten Objektes vorkommen. Die entsprechenden Daten des unbeweglichen Objektes verschwinden wegen der nullierten Geschwindigkeiten. Wir können diese Methode aber auch auf Kontaktsituationen zwischen beliebig vielen bewegbaren *und* fixierten Objekten erweitern.

Sei N die Anzahl der an der Kontaktsituation beteiligten Objekte und k die Zahl der Kontakte. Wir gehen aus von der an einem Kontaktpunkt zwischen den beiden beteiligten

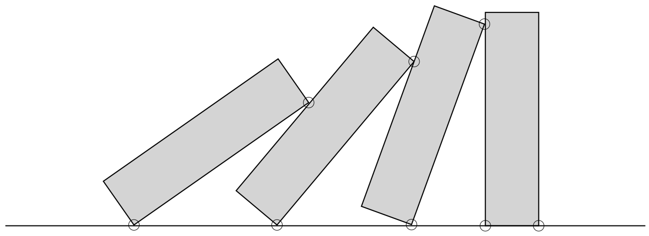

Abbildung 4.5: Reihe fallender Dominosteine, wobei sich alle Steine über die Kontakte gegenseitig beeinflussen und die Kollisionsbehandlung nicht mehr nur jeweils für Objektpaare betrachtet werden kann.

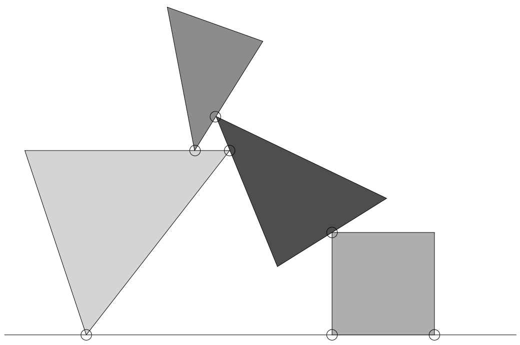

Abbildung 4.6: Kontaktsituationen können beliebig schwierig werden, wenn sich mehrere Objekte gegenseitig beeinflussen und gleichzeitig betrachtet werden müssen.

Objekten geltenden Beziehung

$$\mathbf{n}^T\mathbf{u}_j^a - \mathbf{n}^T\mathbf{u}_i^a = -e\mathbf{n}^T(\mathbf{u}_j^b - \mathbf{u}_i^b) \text{ mit } i, j \in \{1, \ldots, N\},$$

die wir umformen zu

$$\mathbf{n}^T(\Delta\mathbf{u}_j - \Delta\mathbf{u}_i) = -(1+e)\mathbf{n}^T(\mathbf{u}_j^b - \mathbf{u}_i^b).$$

Für die Berechnung der Geschwindigkeiten gilt analog zu (4.9) und (4.10):

$$\mathbf{v}_i = \frac{1}{m_i}\sum_{1\leq l\leq k} S(i,l)p^{(l)}\mathbf{n}^{(l)} \qquad\qquad \boldsymbol{\omega}_i = \mathbf{I}_i^{-1}\sum_{1\leq l\leq k} \mathbf{r}_i^{(l)} \times S(i,l)p^{(l)}\mathbf{n}^{(l)}$$

$$\text{mit: } S(i,l) = \begin{cases} 1 & i \text{ an Kontakt } l \text{ beteiligt und } \mathbf{n}^{(l)} \text{ zeigt in Richtung von } i \\ -1 & i \text{ an Kontakt } l \text{ beteiligt und } \mathbf{n}^{(l)} \text{ zeigt } \textit{nicht} \text{ in Richtung von } i \\ 0 & \text{Objekt } i \text{ ist nicht an Kontakt } l \text{ beteiligt} \end{cases}$$

Das bedeutet, daß jeweils für genau zwei Objekte i und j der Summand des Kontaktes l nicht gleich Null ist und $S(i,l) = -S(j,l)$ gilt.

Für alle k Kontakte zusammengefaßt erhalten wir das Gleichungssystem:

$$\mathbf{J}^T\mathbf{M}^{-1}\mathbf{J}\mathbf{p} = -(1+e)\mathbf{J}^T\mathbf{u}^b$$

$$\text{mit: } \mathbf{M}^{-1} = \begin{pmatrix} \frac{1}{m_1}\mathbf{E} & & & & \mathbf{0} \\ & \mathbf{I}_1^{-1} & & & \\ & & \ddots & & \\ & & & \frac{1}{m_N}\mathbf{E} & \\ \mathbf{0} & & & & \mathbf{I}_N^{-1} \end{pmatrix} \in \mathbb{R}^{6N\times 6N}$$

$$\mathbf{p} = (p_1, \dots, p_k)^T \in \mathbb{R}^k$$

$$\mathbf{u}^b = \left(\mathbf{v}_1^{b\,T}, \boldsymbol{\omega}_1^{b\,T}, \dots, \mathbf{v}_N^{b\ T}, \boldsymbol{\omega}_N^{b\ T}\right)^T \in \mathbb{R}^{6N},$$

Für die Jacobi-Matrix $\mathbf{J} \in \mathbb{R}^{6N\times k}$ gilt:

$$\begin{pmatrix} \mathbf{J}_{3(2i-1)-2,l} \\ \mathbf{J}_{3(2i-1)-1,l} \\ \mathbf{J}_{3(2i-1),l} \end{pmatrix} = S(i,l)\mathbf{n}^{(l)} \qquad i = 1, \dots, N$$

$$\begin{pmatrix} \mathbf{J}_{3(2i)-2,l} \\ \mathbf{J}_{3(2i)-1,l} \\ \mathbf{J}_{3(2i),l} \end{pmatrix} = S(i,l)(\mathbf{r}_i^{(l)} \times \mathbf{n}^{(l)}) \qquad l = 1 \dots, k$$

Das Gleichungssystem können wir analog zu Gleichung (4.13) lösen. Das Verfahren funktioniert auch für fixierte Objekte. In diesem Fall nullieren wir die entsprechenden Einträge in Matrix $\mathbf{M}^{-1}$.

4.5 Mögliche Erweiterungen

Unser Modell bietet noch einige Ansatzpunkte zur Erweiterung.

- Wie schon zu Beginn erwähnt, ist die Behandlung von ruhenden bzw. gleitenden Kontakten generell in impulsbasierten Modellen ein Problem. Es gibt keine wirklich ruhenden Kontakte, d.h. Kontakte mit einer relativen Kontaktgeschwindigkeit von Null. Wie im Abschnitt über die Implementierung näher erklärt wird, müssen bleibende Kontakte als Folge kleiner Stöße behandelt werden (um Bewegung in Tangentialrichtung zu ermöglichen). Die Behandlung von bleibenden Kontakten ist dagegen eine große Stärke von analytischen Ansätzen. Denkbar wäre hier, wie z.B. in der Dissertation von Mirtich[15] vorgeschlagen, ein Modellwechsel zu einem analytischen Ansatz, wenn die Geschwindigkeit in Normalenrichtung einen Toleranzwert unterschreitet, ab dem Kontakte sich nicht wieder sofort lösen sollen.

- Außerdem wäre die Einbeziehung von Reibung wünschenswert. „Echte" Reibung (und nicht nur einfach eine Verlangsamung der Bewegung) erfordert allerdings Informationen über Materialeigenschaften wie Oberflächenbeschaffenheit, Elastizität und so weiter. Hier treten dann noch zusätzliche Probleme auf wie z.B. die Übergänge zwischen den einzelnen Reibungsformen (Übergang von Haftreibung zu Gleitreibung und umgekehrt, Rollreibung, sich wieder lösende Kontakte).

4.6 Implementierung

Der erste folgende Abschnitt skizziert grob den Ablauf eines Schrittes der Kollisionserkennung und der Reaktionskomponente. Beide sind eng miteinander verbunden, da die Erkennung die relevanten Daten für die korrekte Reaktion auf Kollisionen liefert. Die folgenden Abschnitte gehen dann näher auf die einzelnen Teile der Reaktionskomponente ein.

4.6.1 Hauptschleife

Algorithmus 7 Hauptschleife Kollisionsbehandlung

```
detect_collision()
      {Kollisionserkennung und Aufbau der Kontaktlisten}
if new contact then
  find_tmin()
        {Identifizieren von Zeitpunkt und Typ des frühesten Kontaktes}
  for all poly ∈ movedlist do
    partial_transform(poly, t_min)
          {alle Objekte auf den Zustand zum Zeitpunkt t = t_min bringen}
  end for

  if type = moved_moved_contact then
    moved_moved_contact(t_min)
  else
    moved_fixed_contact(t_min)
          {moved-fixed-multi-contact}
  end if

  for all poly ∈ movedlist do
    security_distance(normal)
          {alle Objekte auf Sicherheitsabstand in Richtung normal bringen}
  end for
end if
```

Algorithmus 7 zeigt den Ablauf eines Schrittes von Kollisionserkennung und -reaktion. Die Kollisionserkennung mit dem Aufbau der Kontaktlisten erfolgt wie im letzten Kapitel beschrieben. Wurde ein Kontakt erkannt, so suchen wir nun Zeitpunkt und Typ des frühesten Kontaktes innerhalb des letzten Simulationsintervalls. Als nächstes bringen wir alle Objekte durch Interpolieren auf den Zustand zu diesem Zeitpunkt t_{min}. Auf diese Art stellen wir sicher, daß keine im gleichen Simulationsintervall später auftretenden Kontakte übergangen werden, weil wir jeweils nur die frühesten Kontakte behandeln. Der Typ des frühesten Kontaktes (bewegt–bewegt oder bewegt–unbewegt) bestimmt nun die weitere Art der Kontaktbehandlung, d.h. im wesentlichen die Berechnung des bzw. der Impulse nach dem Stoß. Der letzte Teil der Kollisionsbehandlung bringt die Objekte auf einen *Sicherheitsabstand* voneinander, um Durchdringung im folgenden Simulationsschritt sicher verhindern zu können.

4.6.2 Mehrfachkontakte zwischen bewegt-unbewegt

In einer Schleife über die Kontaktlisten aller bewegbaren Objekte löschen wir alle Kontakte, die nicht innerhalb einer Toleranz Δt nach dem frühesten Kontakt des aktuellen Simulationsintervalls liegen. Wenn zum Beispiel ein Objekt bei nicht vollkommen elastischen Stößen auf eine Unterlage fällt, kommt es zu vielen Einzelkontakten, die zeitlich immer dichter zusammen liegen. Das Objekt „zittert“ und die Zeitdifferenz zwischen zwei Kontakten wird immer kleiner bis sie schließlich unter die Rechengenauigkeit sinkt. Erst dann würden Kontakte als gleichzeitig angesehen. Das hat den Nachteil, daß die Simulationsschrittweite sehr klein wird. Es wird also pro Schritt nur wenig Fortschritt erzielt und die Simulation wird langsam. Erlauben wir aber eine Toleranz für die gleichzeitige Behandlung von Kontakten, so wird dieser Effekt vermindert. Die Simulation läuft mit gleichmäßigerer Geschwindigkeit ab und die minimale Simulationsschrittweite sinkt nicht unter den Toleranzwert Δt. Die Berechnung des Impulses nach dem Stoß erfolgt nun wie schon in Abschnitt 4.4 ausführlich beschrieben. Die Funktion *multi_contact* benutzt die Kontaktliste des übergebenen Polyeders *poly*, um die Matrix $\mathbf{A}$ und den Vektor $\mathbf{b}$ des zu lösenden Gleichungssystems aufzubauen.

4.6.3 Kontakt zweier bewegbarer Objekte

In einer Schleife über die Kontaktlisten aller Objekte aus der Liste der bewegbaren Objekte löschen wir alle Kontakte außer dem zum frühesten Kollisionszeitpunkt t_{min} stattfindenden. Mit den so erhaltenen Kontaktdaten führen wir die Berechnung der Impulse nach dem Stoß analog zu Abschnitt 4.3 durch.

4.6.4 Ruhende Kontakte und Sicherheitsabstand

Die Behandlung von ruhenden bzw. gleitenden Kontakten stellt bei impulsbasierten Ansätzen ein großes Problem dar. Solche Kontakte erkennt man an der verschwindenden relativen Kontaktgeschwindigkeit in Normalenrichtung der Trennebene. In unserer Simulation verwalten wir zur Zeit keine solchen Kontakte. Bestehende Kontakte werden gelöst und im nächsten Simulationsschritt als neue Kontakte erkannt. Das erreichen wir, indem wir unsere kollidierenden Objekte auf einen *Sicherheitsabstand* voneinander bringen. Dazu verschieben wir das Objekt, in dessen Kontaktliste sich die aktuellen Kontakte befinden, in Richtung der Resultierenden aus den jeweiligen Normalen. Die Größe des Sicherheitsabstandes ist von der Art der Simulation abhängig. Einerseits soll der Abstand so groß sein, daß die Objekte garantiert getrennt werden (und somit im nächsten Schritt die erneute Kollision erkannt wird), andererseits aber so klein, daß das Lösen der Kontakte und das damit verbundene „Zittern" bei der verwendeten Bildschirmauflösung nicht sichtbar ist.

Eine gesonderte Behandlung von ruhenden/gleitenden Kontaktes ist nicht auf einfache Art möglich. Nullierter (bzw. sehr kleiner) Normalenanteil der relativen Kontaktgeschwindigkeit reicht als Kriterium nicht aus. Da nach wie vor Kräfte wie z.B. die Gravitation auf „ruhende" Objekte wirken (notwendig, um Gleitkontakte zu realisieren), müßte für solche Kontakte die Kontaktgeschwindigkeit jeweils explizit nulliert werden, um Durchdringung der Objekte zu verhindern. Auf diese Art wäre es aber unmöglich, sich wieder lösende Kontakte zu erkennen (z.B. ein Objekt, das von der schiefen Ebene gleitet). Eine sehr aufwendige Alternative zu der verwendeten Methode wäre bei solchen Kontakten der Wechsel zu einem analytischen Kontaktmodell.

Anhang A

Benutzerschnittstelle

Die Implementierung des in dieser Arbeit vorgestellten Simulationssystems stellt eine Erweiterung der am Lehrstuhl entwickelten Montagesimulation $\mathbb{ICOR}$ dar. Neben den Datenstrukturen und dem Datenformat für $\mathbb{ICOR}$ -Objekte wurde vor allem die Benutzerschnittstelle erweitert. Insbesondere ist nun auch die Steuerung mit der SpaceMouse möglich. Die SpaceMouse ist ein Eingabegerät, das die Manipulation von sechs Freiheitsgraden gleichzeitig zuläßt. Sie erlaubt dem Benutzer somit eine viel intuitivere Interaktion mit dem System als dies mit Maus oder Tastatur möglich wäre.

Zusätzlich implementiert und eingebunden wurden Algorithmen zur Berechnung der Momente der Polyeder und die Bewegung von Objekten eines physikalischen Systems über darauf wirkende Kräfte wie z.B. die Gravitation. Zur korrekten Simulation waren dann noch eine neue Kollisionserkennung notwendig, die mit der Zustandsbeschreibung des Systems *vor* und *nach* dem aktuellen Bewegungsschritt arbeitet, sowie die impulsbasierte Kollisionsreaktionskomponente. Die beiden letzten Teile bilden den Hauptanteil der Implementierung.

A.1 Anbindung der SpaceMouse

Die SpaceMouse ist ein von der DLR (Deutsche Forschungsgesellschaft für Luft- und Raumfahrt e.V. [22]) entwickeltes Eingabegerät, das die intuitive Manipulation von bis zu sechs Freiheitsgraden gleichzeitig erlaubt. Sie benötigt keine eigene Stromversorgung und wird über die serielle Schnittstelle angeschlossen. Über eine Sensorkappe steuern wir mit den Fingerspitzen einer Hand ein opto-elektronisches Meßsystem, das die Handbewegung in absolute Werte für die sechs Freiheitsgrade umsetzt und zur Weiterverarbeitung an den Rechner sendet. Diese Freiheitsgrade entsprechen im Raum drei translatorischen (verschieben bzw. skalieren) und drei rotatorischen Freiheitsgraden (Drehung um die Koordinatenachsen oder Hauptachsen des manipulierten Objektes). Außerdem besitzt die SpaceMouse noch in ihrer Funktion frei belegbare Knöpfe.

A.1.1 Schnittstelle

Die SpaceMouse schreibt periodisch den Zustand der Sensoren in einen Puffer. Aus dem Puffer kann dann mittels der mitgelieferten Software jeweils der erste Datensatz ausgelesen werden. Dieser Datensatz besteht aus den Werten x, y und z für die translatorischen Auslenkungen, den Variablen X, Y und Z für die rotatorischen Auslenkungen und evtl. der in der Variablen *button* zurückgelieferten Nummer des gedrückten Knopfes. Diese Werte lassen sich durch Skalierung an die jeweilige Anwendung anpassen. Die Knöpfe lassen sich den eigenen Anforderungen entsprechend umbelegen, sind aber schon mit einigen nützlichen Funktionen für exaktere Steuerung vorbelegt.

1. *Trans on/off :* Abschalten der translatorischen Auslenkungen erlaubt die exaktere Rotation ohne Positionsänderung.
2. *Rot on/off :* Nullierung der rotatorischen Auslenkungen zur exakteren Positionierung bei unveränderter Orientierung.
3. *Dominant on/off :* Exaktere Steuerung der jeweils größten Auslenkung durch Nullieren aller anderen Auslenkungen.
4. *Zeroing :* Direktes Abschalten *aller* Auslenkungen.
5. *Gain trans :* Hardwaremäßige Skalierung der translatorischen Auslenkungen (beschleunigen bzw. abbremsen von Bewegungen)
6. *Gain rot :* Skalierung der rotatorischen Auslenkungen.
7. *Nullradius :* Bestimmung des Toleranzwertes. Unterhalb dieses Wertes werden Auslenkungen ignoriert.
8. *Set default :* Ausgangseinstellungen der SpaceMouse wiederherstellen.

A.1.2 Verwendung in unserer Simulation

- Position des Betrachters verändern :
  Die Bewegung eines Punktes im Raum hat 3 Freiheitsgrade. Also bleiben die 3 übrigen Manipulationsmöglichkeiten unbelegt. Wir können den Blickpunkt auf das Szenario zu bzw. davon weg bewegen, um das Szenario 'herumlaufen' oder uns über bzw. unter das Objekt bewegen. Das *zoomen* definieren wir dabei als translatorische Bewegung, die beiden anderen Operationen als rotatorische.

- Position/Orientierung des jeweils aktiven Objektes verändern :
  Mit den translatorischen Auslenkungen bewegen wir das Objekt in Richtung der Koordinatenachsen, mit den rotatorischen drehen wir das Objekt um die in den Schwerpunkt verlagerten Koordinatenachsen. Wir können hier also alle 6 Freiheitsgrade

eines Polyeders im Raum gleichzeitig manipulieren. Auf Knopfdruck können wir außerdem zwischen Objektmanipulation und Blickpunktmanipulation umschalten.

- Simulation der Kräfte, die auf ein Objekt wirken :
  Auch hier verwenden wir alle 6 Freiheitsgrade. Die translatorischen Auslenkungen stehen für im Schwerpunkt des Objekts in Richtung der Koordinatenachsen angreifende Kräfte und die rotatorischen für Drehmomente bezüglich durch den Schwerpunkt verlaufenden Achsen, parallel zu den Koordinatenachsen.

A.1.3 Probleme

- Die Zeitintervalle, nach denen die SpaceMouse in den Puffer schreibt bzw. die Anwendung den Puffer ausliest, sind im allgemeinen nicht synchronisiert. Typischerweise wird schneller in den Puffer geschrieben, als wir die Datensätze benötigen, so daß wir beim Auslesen nicht die *aktuelle* Auslenkung erhalten, sondern eine *früher* festgestellte. Damit läuft die angezeigte Auslenkung der 'echten' mit der Zeit immer weiter hinterher. Umgehen können wir das Problem, indem wir bei jedem Auslesen den *gesamten* Puffer lesen und alle Datensätze außer dem aktuellen ignorieren.

- Ein weiteres Problem erhalten wir beim Drücken eines der Knöpfe der SpaceMouse während der Simulation. Es wird nicht nur die Nummer des gedrückten Knopfes zurückgegeben, sondern auch die Werte der Auslenkungen zerstört. Daher bauen wir noch einen Test ein, der die Auslenkungswerte bei gedrückten Knöpfen herausfiltert.

- Einerseits ist die Steuerung mit der SpaceMouse sehr intuitiv und schnell, da wir alle Freiheitsgrade gleichzeitig manipulieren können. Andererseits ist es sehr schwierig, ganz exakt zu steuern und festgelegte Situationen herzustellen, z.B. eine bestimmte Position/Orientierung. Erleichternd wirken dabei die in Abschnitt A.1.1 vorgestellten Funktionen.
  In unserem Fall nutzen wir die Vorzüge der SpaceMouse, erlauben aber alternativ dazu auch exakte Steuerung über die Maus, um jeweils einzelne Freiheitsgrade zu manipulieren bzw. absolute Eingabewerte (z.B. Koordinatenangaben) in Textfenstern.

A.2 Erweiterung der Montagesimulation

Beim Einlesen des Szenarios wird neben dem Aufbau der Datenstrukturen die im ersten Kapitel beschriebene Berechnung von Volumen, Schwerpunkt und Trägheitsmatrix der Objekte durchgeführt.

Anschließend können wir die Objekte noch in Position und Orientierung manipulieren, um den gewünschten Startzustand der Simulation herzustellen. Abbildung A.2 zeigt die Eingabemaske, die die wahlweise Steuerung mit der Maus über die Pfeilknöpfe, mit der

Tastatur über die Texteingabefelder oder mit der SpaceMouse erlaubt. Bei den ersten beiden Methoden hat jeder der sechs Freiheitsgrade seinen eigenen Eingabebereich und kann gezielt einzeln manipuliert werden, während die SpaceMouse die Steuerung von mehreren (allen) Freiheitsgraden gleichzeitig erlaubt. Außerdem können wir über die SpaceMouse auch wahlweise den Blickpunkt auf das Szenario verändern.

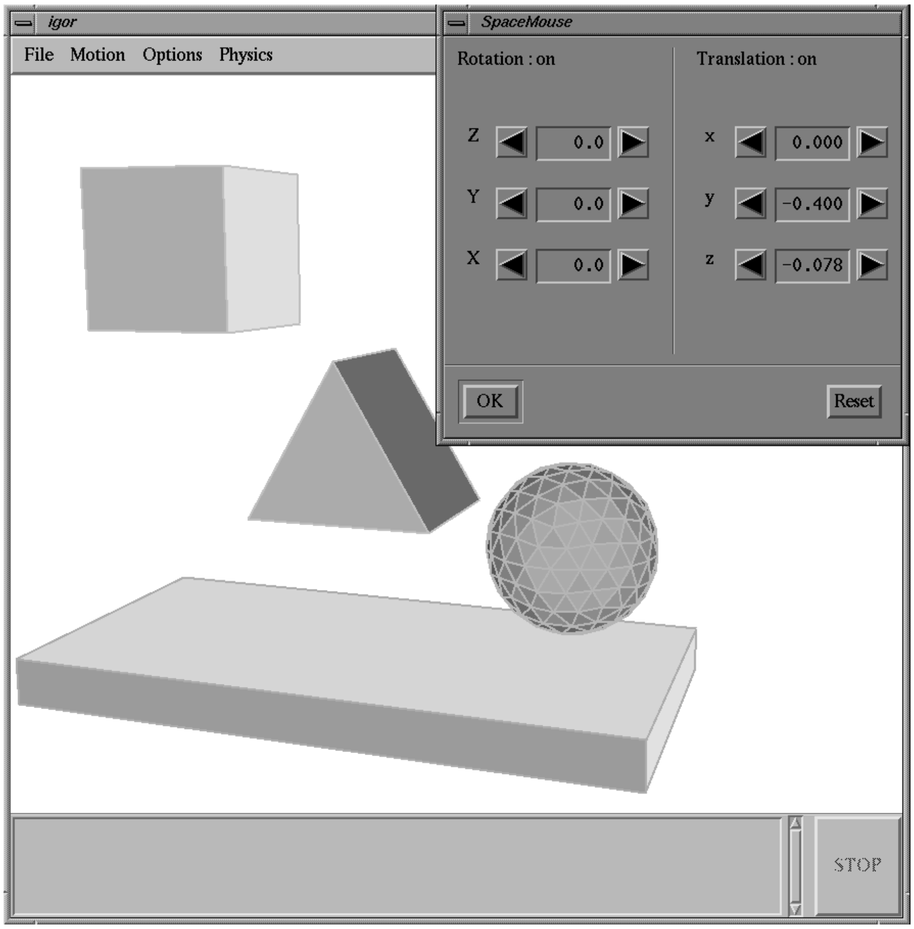

Abbildung A.1: Maske zur Eingabe von Position und Orientierung

Abbildung A.2 zeigt die Maske, die dem Benutzer nach dem Starten der Simulation die Manipulation einiger Parameter erlaubt. Neben dem Stoppen, Wiederaufnehmen und Beenden der Simulation können wir hier auch die Elastizität der simulierten Objekte und die Größe der auf das Szenario wirkenden Gravitationskraft steuern. Außerdem können wir mit der SpaceMouse Kräfte bzw. Drehmomente simulieren, die zu einem Zeitpunkt zusätzlich auf das jeweils aktive Objekt wirken.

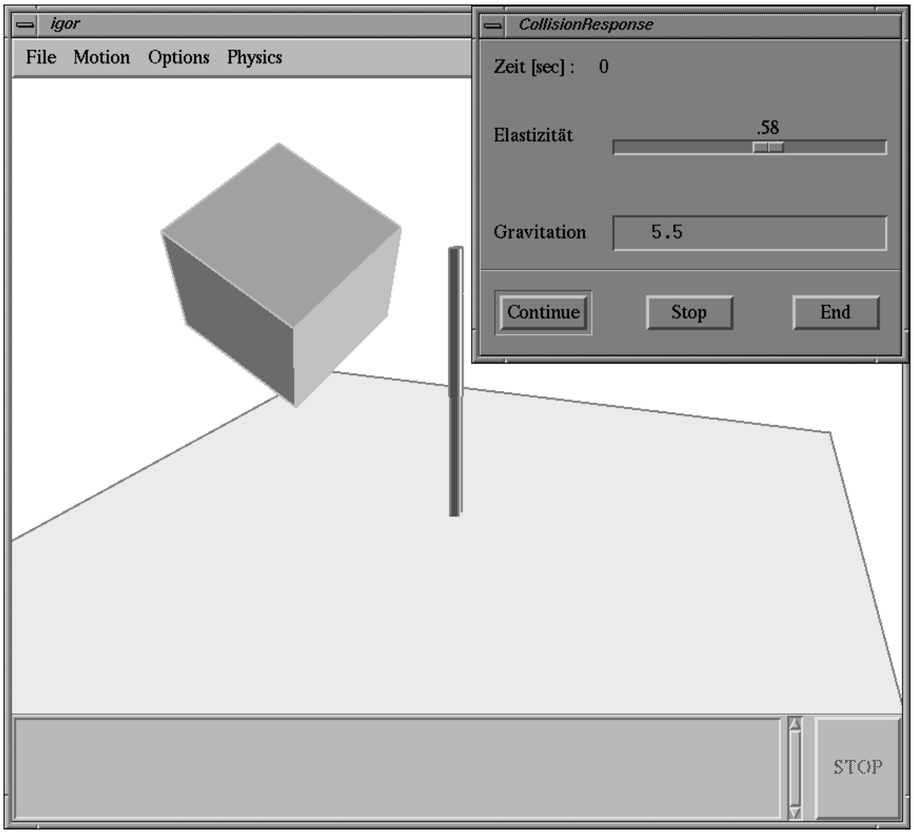

Abbildung A.2: Maske zur Änderung von Gravitation und Elastizität

Literaturverzeichnis

[1] David Baraff. Analythical methods for dynamic simulation of non-penetrating rigid bodies. *SIGGRAPH Computer Graphics*, 23:223–232, 1989.

[2] David Baraff. Coping with friction for non-penetrating rigid body simulation. *SIGGRAPH Computer Graphics*, 25:31–40, 1991.

[3] David Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies.* Ph.d.thesis, Department of Computer Science, Cornell University, March 1992.

[4] David Baraff. An introduction to physically based modeling, 1997.

[5] B.Mirtich and J.Canny. Impulse-based dynamic simulation. *Proceedings of Workshop on Algorithmic Foundations of Robotics*, February 1994.

[6] I.N. Bronstein and K.A. Semendjajew. *Taschenbuch der Mathematik.* Teubner Verlagsgesellschaft, Stuttgart - Leipzig, 1957.

[7] J. Comes and S. Jost. Berechnung der Momente eines Polyeders und Dynamiksimulation am Beispiel eines Doppelpendels, 1995.

[8] John J. Craig. *Introduction to Robotics.* Addison Wesley Publ. Company, 1986.

[9] K. Endl and W. Luh. *Analysis II.* Akademische Verlagsgesellschaft, Wiesbaden, 1987.

[10] Walter Greiner. *Theoretische Physik — Mechanik II.* Verlag Harry Deutsch, Thun und Frankfurt am Main, 1986.

[11] J.A.Nelder and R.Mead. Downhill simplex algorithm. *Computer Journal*, 7:305–309, 1965.

[12] Sheue ling Lien and J.T. Kajiya. A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *IEEE Computer Graphics and Applications*, 4:35–41, 1984.

[13] Per Lötstedt. Coulomb friction in two-dimensional rigid body systems. *Zeitschrift für Angewandte Mathematik und Mechanik*, 61:605–15, 1981.

[14] W.D. MacMillan. *Dynamics of Rigid Bodies.* Dover, New York, 1960.

[15] Brian Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems.* Ph.D.thesis, University of California, Berkeley,California, December 1996.

[16] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22:289–298, 1988.

[17] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, Cambridge, 1988.

[18] Elmar Schömer and Christian Thiel. Efficient collision detection for moving polyhedra. Technical Report MPI-I-94-147, MPI für Informatik, Saarbrücken, Germany, September 1994.

[19] Elmar Schömer. Simulation mechanischer Systeme, 1995.

[20] S.Gottschalk, M.C.Lin, and D.Manocha. OBBTree : A hierarchical structure for rapid interference detection. *ACM SIGGRAPH'96*, 1996.

[21] K. Shoemake. Animating rotation with quaternion curves. *SIGGRAPH Computer Graphics*, 19:245–254, 1985.

[22] SPACE CONTROL, Company for 3D Systems, Rudolf-Diesel-Straße 3,82205 Gilchingen, Germany, URL: http//:www.op.dlr.de/FF-DR-RS/SC/. *SpaceMouse: User's Manual Version 5.3.*

[23] Markus Welsch. Exakte geometrische Kollisionserkennung. Diplomarbeit, Universität des Saarlandes, Saarbrücken, Germany, January 1996.

[24] Zachmann and Felger. The BoxTree : Enabling real-time and exact collision detection of arbitrary polyhedra. *Proc. SIVE'95*, pages 104–113, 1995.