Exakte geometrische Kollisionserkennung

Markus Welsch

10. Januar 1996

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, im Januar 1996

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Die enorm ansteigende Leistungsfähigkeit moderner Hardware erlaubt die Entwicklung von immer realitätsnäheren *Simulationen* – der virtuellen Nachbildung realer Vorgänge. Ausschlaggebend für die Aussagekraft eines Simulationsergebnisses ist die Genauigkeit des einer Simulation zugrundeliegenden Modells. Solche Modelle beschreiben den für eine Anwendung relevanten Ausschnitt der realen Welt – die darin enthaltenen Objekte und deren Abhängigkeiten – in mathematischer Weise. Simulationen werden in den verschiedensten Bereichen eingesetzt, um z.B. Vorgänge zu optimieren. Hierzu wird – vergleichbar mit einer Versuchsreihe in der Physik – eine Folge von Simulationen durchgeführt, deren Ergebnisse durch Änderung entsprechender Simulationsparameter mit den Vorstellungen des Benutzers zur Deckung gebracht werden sollen (Crashtestsimulation). Die Verwendung von Simulationen kann zu erheblichen Kosten- und Zeiteinsparungen führen, wenn es zum Beispiel darum geht, den Prototypen eines neuen Produktes zu entwickeln und zu testen: bevor in der Automobilindustrie ein realer Prototyp eines neu entwickelten Fahrzeugs angefertigt wird, werden zunächst zahlreiche Simulationen auf der Basis eines virtuellen Prototyps (*software prototyping*) durchgeführt, die Aufschluß über das Fahrverhalten, die Aerodynamik, die Sicherheit der Insassen bei einem Unfall uvm. geben sollen. Am Rechner entworfene *virtuelle Prototypen* weisen wegen der leichten Manipulierbarkeit der Systemparameter gegenüber realen Prototypen eine weitaus höhere Flexibilität auf: so können Entwurfsfehler schon in den Anfangsstadien der Entwicklung erkannt und beseitigt werden. Darüber hinaus können auch Gefahrengründe für den Einsatz rein computerbasierter Simulationen sprechen (Flugsimulator).

Ein konkretes Anwendungsgebiet für Simulationen stellt die Objektmontage durch Roboter dar. Dort gilt es, meist relativ einfach geartete Teile maschinell exakt zusammenzufügen. Zu Optimierungszwecken können mittels rechnerunterstützter Simulationen sehr schnell verschiedene Varianten des Montageprozesses durchgespielt werden, ohne eine prototypische Anlage aufbauen bzw. ändern zu müssen. Ein populärer Modellierungsansatz für die Objekte der realen Welt sieht hierbei deren geometrische Beschreibung durch starre Polyeder vor, was wegen der Ebenheit der Oberflächen und Geradlinigkeit der Flächenbegrenzungen die Entwicklung praktikabler und effizienter Algorithmen ermöglicht. Als Bewegungen läßt man Translationen, Rotationen sowie deren überlagerte Bewegungsformen zu. Eine fundamentale Bedingung, die der Simulationsprozeß nun erfüllen muß, um dem Anspruch, die Realität nachzubilden, gerecht zu werden, ist die Verhinderung der Durchdringung der Objekte; ein solches System muß also in der Lage sein, Kollisionen zwischen Objekten während ihrer Bewegungen zu erkennen und den Benutzer darauf aufmerksam zu machen, da dieser ansonsten dazu gezwungen wäre, den Montagevorgang optisch zu kontrollieren, was im Gesamtsystem allerdings eine Unsicherheitsquelle darstellen würde. Die hierzu bisher praktizierte naive Vorgehensweise sah die Umsetzung eines solchen Kollisionsproblems in ein geometrisches Schnittproblem und dessen numerische Lösung auf der Basis von Datentypen beschränkter Genauigkeit (Gleitkommaarithmetik) vor.

Die praktische Implementierung geometrischer Algorithmen stellt im allgemeinen ein schwieriges Problem dar, da numerische und kombinatorische Informationen konsistent gehalten werden müssen; die hiermit verbundenen Schwierigkeiten werden in [MN94] an einem einfachen Beispiel aufgezeigt: bei Verwendung von hardwaremäßig implementierten Datentypen beschränkter Genauigkeit zur Berechnung geometrischer Probleme ist es einfach möglich, die Gleichung zweier Geraden anzugeben, die sich mehrfach schneiden. Übertragen auf unsere Problemstellung der geometrischen Kollisionserkennung kann dies im Verlauf der Berechnungen durchaus zu falschen Entscheidungen bzgl. des Auftretens bzw. Vorhandenseins einer Kollision kommen. Durch aufwendige Fehleranalysen und -abschätzungen kann zwar erzwungen werden, daß stets *sichere* Entscheidungen getroffen werden (vgl. pessimistisches Rechnen), doch läßt sich in diesen Fällen keine *exakte* Entscheidung treffen: so kann eine Kollision erkannt werden, die nicht vorhanden ist. Einzige Alternative zu diesem Ansatz, gerade dann, wenn es um sicherheitskritische Anwendungen geht, die kein – auch noch so kleines – Restrisiko erlauben, ist das *exakte Rechnen*.

Die naive Interpretation des Begriffes *exaktes Rechnen* beinhaltet zum einen die exakte Berechnung aller auftretenden numerischen Werte und zum anderen das Treffen ausschließlich fehlerfreier Entscheidungen. Da der erst genannte Punkt im allgemeinen ein unlösbares Problem darstellt (exakte Berechnung von $\sqrt{2}$), zum Treffen fehlerfreier Entscheidungen andererseits jedoch auch keine unbegrenzte Präzision nötig ist, revidieren wir die naive Definition des Begriffes *exaktes Rechnen*, indem wir damit von nun an die Tatsache bezeichnen, daß die einem Problem zugrundeliegende, berechnete kombinatorische Struktur exakt sein muß, die hiermit assoziierten numerischen Werte jedoch nicht unbedingt. Ein Ansatz zur Umsetzung dieser Interpretation ist die Idee der präzisionsabhängigen Berechnung, die auf der Basis eines softwaremäßig implementierten, ganzzahligen Datentyps (nahezu) beliebiger[1] Genauigkeit durchgeführt wird: hierbei wird in Abhängigkeit von der Genauigkeit der Eingabedaten die zur exakten Lösung eines Problems benötigte Präzision berechnet. In [SSW95] wurde so zum Beispiel gezeigt, daß exakte Kollisionsberechnungen in einem einfacheren Modell, in dem lediglich ein einzelnes, translatorisch oder rotatorisch bewegtes Objekt erlaubt wird, mit $14L + 22$-facher Genauigkeit durchgeführt werden kann, wenn L die maximale

[1] Die Länge der darstellbaren Zahlen ist durch den vorhandenen Speicherplatz beschränkt. Für unsere Anwendungen kann jedoch davon ausgegangen werden, daß dies nicht der limitierende Faktor sein wird.

Länge der Eingabedaten bezeichnet. Das Problem der exakten geometrischen Kollisionserkennung für mehrere gleichzeitig bewegte Objekte erfordert, wie wir im Verlauf der Arbeit erkennen werden, das Hantieren mit noch wesentlich längeren Zahlen, was sich in einer höheren Laufzeit der Verfahren niederschlägt. Dennoch gewinnt *exaktes Rechnen* in der *Computational Geometry* stark an Bedeutung, was nicht zuletzt in der enorm ansteigenden Leistungsfähigkeit heutiger (und zukünftiger) Hardware bei sinkenden Preisen begründet liegt. Verliert der Trade-Off zwischen Exaktheit und Rechengeschwindigkeit weiterhin an Brisanz, so wird *exaktes Rechnen* in Zukunft in der Informatik einen weitaus höheren Stellenwert einnehmen.

Gliederung und Resultate

Ziel der Modellierung eines Systems ist die Reduktion der einfließenden Modellparameter auf ein notwendiges Minimum bei gleichzeitiger Wahrung der notwendigen Realitätstreue und Aussagekraft. Im *ersten Kapitel* werden wir daher ein einfaches Modell konstruieren, das es uns erlaubt, den für unsere Zwecke interessanten Ausschnitt der realen Welt effizient nachzubilden und exakte Kollisionsberechnungen durchzuführen. Hierbei betrachten wir die Objekte als starre Körper und modellieren sie durch Polyeder. Als Bewegungen lassen wir Translationen und Rotationen sowie deren überlagerte Bewegungsformen zu. Als Konfigurationen unserer Objekte werden wir aus Gründen der Exaktheit 7-Tupel über dem euklidischen Ring der ganzen Zahlen $\mathbb{Z}$ konstruieren. Diese Tatsache erfordert im Verlauf der Verfahren zur Kollisionsberechnung für die Ausführung bestimmter Operationen (Division, Wurzelziehen) spezielle Vorgehensweisen, um den Ring der ganzen Zahlen nicht zu verlassen. Entscheidend für die Ganzzahligkeit und Kompaktheit der Konfigurationen ist die Beschreibung von Orientierungen bzw. Rotationen durch Quaternionen.

Im *zweiten Kapitel* werden wir erkennen, daß sich ein vollständiger Kollisionstest zwischen zwei polyhedral modellierten Objekten in vier geometrisch kompakt beschreibbare Testkategorien aufteilen läßt. Die hierdurch erreichte Reduktion der Komplexität des Ausgangsproblems durch Zerlegung in überschaubarere Teilprobleme – eine nicht nur in der Informatik verbreitete Technik zur Problemlösung – ermöglicht eine effiziente Beschreibung der jeweiligen Kollisionsbedingungen. Die generelle Vorgehensweise sieht hierbei eine Projektion der betrachteten räumlichen Szene in eine geeignete Standardebene vor. Aufgrund der zeitlich parametrisierten Objektkonfigurationen ergibt sich in allen Fällen ein polynomielles Ungleichungssystem im Zeitparameter t der Form:

$$\begin{array}{rrcl} & f(t) & = & 0 \end{array} \Big\} \text{ potentielle Kollisionszeitpunkte}$$

$$\left.\begin{array}{rrcl} \wedge & g_1(t) & > & 0 \\ \wedge & & \vdots & \\ \wedge & g_n(t) & > & 0 \end{array}\right\} \begin{array}{l}\text{Einschränkung der potentiellen} \\ \text{Kollisionszeitpunkte auf reale}\end{array}$$

Lediglich die Anzahl der Nebenbedingungen variiert unter den diversen Kollisionsarten. Die Koeffizientenlängen der Polynome können hierbei unter Umständen um den Faktor 7 anwachsen. Um nun zu entscheiden ob während einer vorgegebenen Bewegung, die über dem Zeitintervall $[0, 1]$ parametrisiert ist, eine Kollision zwischen den Objekten auftritt, gilt es von einem solchen polynomiellen Ungleichungssystem zu entscheiden ob es dort eine Lösung besitzt.

In *Kapitel 3* werden wir zwei Verfahren zur exakten Entscheidung obigen Problems vorstellen:

- **das Pedersen-Verfahren**
  Dieses Verfahren berechnet auf der Basis *Sturmscher Ketten* die Anzahl der Nullstellen eines Polynoms, die einer Anzahl von polynomiellen Nebenbedingungen genügen, explizit.

- **das Uspensky-Verfahren**
  Dieses Verfahren separiert die Nullstellen der polynomiellen Hauptbedingung in Intervalle mit rationalen Grenzen; durch Auswertung der Nebenbedingungen an solchen läßt sich mittels ausgefeilter Strategien eine Entscheidung über die Existenz einer Lösung des gesamten polynomiellen Ungleichungssystems treffen.

Beide Verfahren zeichnen sich dadurch aus, daß nach einer endlichen Anzahl algebraischer Operationen auf den ganzzahligen Koeffizienten der Polynome eine *exakte* Entscheidung über das Vorhandensein bzw. die Anzahl von Nullstellen im Intervall $(0, 1)$ getroffen werden kann. Ausführlich werden wir die Komplexität der Verfahren betrachten, wobei wir feststellen werden, daß die Zahlenlängen in den Zwischenrechnungen unter Umständen erheblich anwachsen können, was den Einsatz spezieller Zahlenpakete zur Darstellung *beliebig* großer, ganzer Zahlen notwendig macht.

Die in den Kapiteln 2 und 3 geschilderten Vorgehensweisen und Verfahren zur exakten geometrischen Kollisionserkennung wurden im Rahmen dieser Arbeit in das von Dr. Elmar Schömer am Lehrstuhl von Prof. Dr. Günter Hotz entwickelte $\mathbb{IGOR}$-System zur interaktiven Montage- und Robotersimulation integriert. Dies stellt insofern eine wesentliche und wichtige Erweiterung dar, als Kollisionsberechnungen bisher lediglich auf numerischer Basis durchgeführt wurden und durchaus falsche Ergebnisse liefern konnten. In *Kapitel 4* beschreiben wir die für die Implementierung der Verfahren zunächst notwendigen Erweiterungen im Bereich algebraischer Ausdrücke und Strukturen. Um die Verfahren für einen Einsatz in der Praxis konkurrenzfähig machen zu können, werden verschiedene Vorgehensweisen zur Erhöhung der Effizienz betrachtet, die zum Teil drastische Laufzeiteinsparungen erzielen:

- der Einsatz einfacher Heuristiken zur Reduzierung der Anzahl der mittels in Kapitel 3 beschriebenen Verfahren exakt zu untersuchenden polynomiellen Ungleichungssysteme. Hierbei machen wir uns die Tatsache zu nutze, anhand schnell zu berechnender Kriterien Aussagen über das Verhalten eines Polynoms im Intervall $(0, 1)$ machen und so evtl. eine teurere, exakte Lösung verwerfen zu können.

Im Rahmen eines vollständigen, exakten Kollisionstest bringt dies in der Regel erhebliche Laufzeiteinsparungen mit sich.

- die Auswahl der zur Darstellung großer Zahlen zu verwendenden Zahlenpakete, die aufgrund der Effizienz der implementierten arithmetischen Funktionen durchaus starke Auswirkungen auf die Laufzeit der eingesetzten Verfahren haben kann.
- der Einsatz spezieller Techniken (z.B. *reference counting*) zur Ressourcen-Einsparung.

Zum Abschluß des Kapitels werden wir zur Untermauerung der Ergebnisse Laufzeitvergleiche zwischen den aus auszuwählendem Verfahren, Datentyp und auszuwählender Heuristik möglichen Kombinationen angeben. Hierbei wird sich zeigen, daß eine mit obigen Vorgehensweisen optimierte *exakte Lösung* einer rein *numerischen Lösung* in ihrer Laufzeit durchaus recht nahe kommen kann. Dies verdeutlicht, daß *exaktes Rechnen* zusammen mit Optimierungstechniken und wachsender Leistungsfähigkeit der Hardware immer konkurrenzfähiger wird, zumindest dann, wenn es sich nicht um laufzeitkritische Anwendungen handelt.

In *Anhang A* stellen wir weitere nützliche mathematische Fakten zur Verfügung, die wir im Verlauf der Arbeit benutzen werden.

Um dem Leser einen Eindruck von der Komplexität des Problems der exakten geometrischen Kollisionserkennung zu vermitteln, wird in *Anhang B* ein sich aus einer einfachen Bewegung zweier einfacher polyhedraler Objekte ergebendes polynomielles Ungleichungssystem und dessen weitere Entwicklung im Verlauf der Berechnungen angegeben.

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 hervorragende individuelle Betreuung. Seine freundliche Unterstützung, seine Geduld und seine wertvollen Ratschläge trugen wesentlich zum Gelingen dieser Arbeit bei.

Thomas Chadzelek gilt mein Dank für die ausgezeichnete Zusammenarbeit im $\mathbb{ICOR}$ -Projekt und die fruchtbaren Diskussionen und Anregungen. Allen Kommilitonen und Mitarbeitern an diesem Lehrstuhl danke ich für das hervorragende Arbeitsklima.

Insbesondere aber danke ich meinen Eltern, die mir ein Studium überhaupt erst ermöglichten; ihnen widme ich diese Arbeit.

Kapitel 1

Grundlagen der geometrischen Kollisionserkennung

In diesem Kapitel werden wir das mathematische Handwerkszeug zur Bearbeitung des Problems der *exakten geometrischen Kollisionserkennung* bereitstellen. Dazu zählen Begriffe wie Polyeder, Translationen und Rotationen, Konfigurationen von Objekten und vieles mehr. Zunächst müssen wir uns jedoch mit dem Begriff des *mathematischen Modells* beschäftigen, einer Abstraktion von der realen Welt, die es uns ermöglicht, komplexe Sachverhalte und Vorgänge rechnerunterstützt in effizienter Weise zu simulieren.

1.1 Das Weltszenario

Zur effizienten Beschreibung und Untersuchung komplexer Sachverhalte und Vorgänge aus unserer realen Welt ist es unumgänglich, das betrachtete System auf ein notwendiges Minimum an einfließenden Faktoren zu reduzieren. Die Vielzahl der Elemente, sowie die Kompliziertheit und Vielfalt der Funktionsprozesse realer Systeme gestatten es nicht, absolut adäquate Modelle für sie aufzustellen; meist ist es sogar unmöglich alle gegenseitigen Beziehungen der Systemelemente zu erkennen und zu verstehen. Oberstes Ziel bei der Modellierung ist es demnach, durch Auswahl der kleinsten Teilmenge von Variablen und Relationen, eine formale (mathematische) Beschreibung des realen Systems zu finden, die den nötigen Grad der Annäherung an die Wirklichkeit besitzt.

Mathematische Modelle spielen in der Informatik eine entscheidende Rolle, bilden sie doch die Basis für Simulationen, der Technik, Probleme iterativ und experimentell zu lösen; hierbei werden Vorgänge aus der realen Welt nachgebildet. Erbringt eine solche Simulation kein zufriedenstellendes Ergebnis, wird sie in der Regel nach Änderung einiger Modellparameter wiederholt. Simulationen werden überall dort eingesetzt, wo es zu teuer, zu gefährlich, zu zeitaufwendig oder einfach unmöglich ist, Versuchsreihen in der realen Welt durchzuführen: so ist computerunterstützte Modellierung und Simulation beispielsweise oft die einzige Möglichkeit beim Entwurf von Prototypen (*software prototyping*).

Ein weiteres spezielles Anwendungsgebiet, aus dem diese Arbeit hervorging, ist die *Montagesimulation*: hier wird das Zusammensetzen, Zerlegen und Bewegen von Objekten mittels Rechnerunterstützung nachgebildet. Als konkrete Anwendung stelle man sich eine Fertigungszelle vor, in der mehrere Roboter bei der Erstellung eines Produktes zusammenarbeiten: Einzelteile müssen hierbei von den Greifern der Roboter an bestimmten Stellen aufgenommen, transportiert und wieder abgesetzt werden. Als Bewegungsarten der Objekte kommen Translationen und Rotationen in Betracht; auch überlagerte Bewegungen sind möglich. Entscheidend für den Erfolg des Montageprozesses ist die Präzision der Bewegungen, vor allem dann, wenn zwei genau aufeinander abgestimmte Werkstücke zusammengefügt werden sollen. Da das Problem der Eigenkoordination der Roboter zum Erreichen eines beabsichtigten Ziels sehr komplex ist, soll der grobe Fertigungsablauf und seine Koordination von außen vorgegeben werden: dem System werden zur Lösung des Problems sozusagen Hilfestellungen gegeben, indem man ihm Zwischenkonfigurationen der beteiligten Objekte zu verschiedenen Zeitpunkten des Montagevorgangs angibt. Aufgabe des Simulationssystems soll es nun sein, die so indirekt vorgegebenen Bewegungen (Konfigurationsübergänge) *exakt* auf Kollisionsfreiheit zu überprüfen. Tritt eine Kollision auf, kann diese dann evtl. durch einfache Änderungen in den Vorgaben beseitigt werden.

1.2 Die mathematische Modellierung des Weltszenarios

Wie wir an der Beschreibung des Weltszenarios schon erkennen können, werden bei seiner Umsetzung in ein mathematisches Modell folgende Aspekte von entscheidender Bedeutung sein:

- *exakte* Modellierung der Objekte
- *exakte* Modellierung der Bewegungen
- *exakte* Modellierung des zeitlichen Ablaufs

1.2.1 Modellierung der Objekte

Im nun folgenden Abschnitt werden wir Objektklassen ausführlich beschreiben, die in der konstruktiven Geometrie wegen ihrer praktischen Bedeutung und ihrer effizienten Beschreibungsmöglichkeit von besonderem Interesse sind und die wir aus diesen Gründen zur Modellierung unserer Objekte heranziehen werden.

Definition 1.1:
Eine Folge von n Strecken $\overline{\mathbf{v_0 v_1}}$, $\overline{\mathbf{v_1 v_2}}$, …, $\overline{\mathbf{v_{n-1} v_n}}$ des $\mathbb{R}^3$, von denen keine zwei benachbarten derselben Gerade angehören, heißt ein **Streckenzug** *s. Die Strecken $\overline{\mathbf{v_i v_{i+1}}}$ heißen Kanten e_i, ihre Endpunkte $\mathbf{v_i}$ Ecken von s. Ein Streckenzug s heißt:*

- **geschlossen**, *wenn* $\mathbf{v_0} = \mathbf{v_n}$, *sonst offen.*
- **einfach**, *wenn je zwei nichtbenachbarte Seiten keinen gemeinsamen Punkt haben, sonst überschlagen.*
- **eben**, *wenn alle Ecken* $\mathbf{v_i}$ *in derselben Ebene, der Trägerbene* Σ *von s, liegen, sonst räumlich.*

Definition 1.2 :
Ein ebener einfacher und geschlossener Streckenzug s heißt **Polygon**. *Mit* **int(p)** *bezeichnet man die von einem Polygon p umschlossene Region ohne deren Begrenzung, mit* **bound(p)** *die Begrenzung der von p umschlossenen Region.*

Definition 1.3 :
Ein Polygon p heißt **positiv orientiert**, *wenn beim Durchlaufen der gerichteten Kanten* e_i *von p die Punkte von int(p) links (mathematisch positiver Drehsinn) von* e_i *liegen.*

Durch die Verwendung positiv orientierter Polygone ist es möglich, solche lediglich durch Angabe ihrer Eckpunkte im Gegenuhrzeigersinn eindeutig zu beschreiben. Aufgrund dieser Tatsache lassen sich Polygone programmiertechnisch gesehen effizient durch ein Feld von Punkten darstellen.

Bezeichnung 1.4 :
Seien $\mathbf{a}$ *und* $\mathbf{b}$ *Punkte des* $\mathbb{R}^3$. *Mit* $l_{\mathbf{ab}}$ *bezeichnen wir das gerichtete Liniensegment von* $\mathbf{a}$ *nach* $\mathbf{b}$, *mit* $L_{\mathbf{ab}}$ *die entsprechende Gerade.*

Die Lage eines Punktes bzgl. einer solchen gerichteten Kante läßt sich gemäß [CLR90] effizient berechnen:

Satz 1.5 :
Sei E eine Ebene im $\mathbb{R}^3$ *mit dem Normalenvektor* $\mathbf{n}$. *Ist die Blickrichtung des Betrachters diesem Normalenvektor entgegengesetzt, so läßt sich die Lage eines Punktes* $\mathbf{a} \in E$ *bzgl. einer gerichteten Kante* $l_{\mathbf{v_i v_{i+1}}} \in E$ *folgt bestimmen:*

- $\mathbf{a}$ *liegt links von* $l_{\mathbf{v_i v_{i+1}}}$ $\iff \mathbf{n}^T\Big((\mathbf{v_{i+1}} - \mathbf{v_i}) \times (\mathbf{a} - \mathbf{v_i})\Big) > 0$
  $\iff det[\mathbf{n}, \mathbf{v_{i+1}} - \mathbf{v}_i, \mathbf{a} - \mathbf{v_i}] > 0$
- $\mathbf{a}$ *liegt rechts von* $l_{\mathbf{v_i v_{i+1}}}$ $\iff \mathbf{n}^T\Big((\mathbf{v_{i+1}} - \mathbf{v_i}) \times (\mathbf{a} - \mathbf{v_i})\Big) < 0$
  $\iff det[\mathbf{n}, \mathbf{v_{i+1}} - \mathbf{v_i}, \mathbf{a} - \mathbf{v_i}] < 0$

Dieser Satz läßt sich durch Anwendung der *Rechte-Hand-Regel* verifizieren.

Definition 1.6:
Eine **konvexe polyhedrale Region** *im* $\mathbb{R}^3$ *ist der Schnitt endlich vieler abgeschlossener Halbräume; durch endliche Vereinigung davon ergibt sich eine* **polyhedrale Region**. *Ein* **Polyeder** *schließlich ist eine polyhedrale Region, die beschränkt und zusammenhängend ist sowie gleich dem Abschluß ihres offenen Inneren.*

Diese Definition hat zur Folge, daß ein Polyeder eine kompakte Teilmenge des $\mathbb{R}^3$ ist, begrenzt durch ebene Flächen, die ihrerseits Polygone sind. Zur Beschreibung des polyhedralen Volumenobjektes läge es nun nahe, anzugeben, wie das Polyeder durch Schnitt und Vereinigung aus Halbräumen entstanden ist. Für die weitere algorithmische Verwendung ist es jedoch günstiger, lediglich den Rand des Polyeders, seine Oberfläche, zu betrachten. Eine solche Darstellung wird auch *boundary representation* genannt.

Damit läßt sich ein Polyeder $\mathcal{P}$ durch die Menge seiner Ecken, Kanten und Flächen sowie deren Nachbarschaftsbeziehungen beschreiben. Als seine Beschreibungskomplexität $|\mathcal{P}|$ bezeichnet man die Anzahl aller Ecken, Kanten und Flächen: $|E| + |V| + |F|$.

Definition 1.7:
Ein (zusammenhängendes) Polyeder, bei dem in jeder Ecke sämtliche Kanten einen einzigen Kantenzyklus bilden, heißt **gewöhnlich**. *Ein gewöhnliches Polyeder heißt genau dann* **einfach**, *wenn sein Rand eine einfach zusammenhängende nur aus einfachen Polygonen bestehende Polyederfläche ist.*

Nach einem Satz von Euler gilt für einfache Polyeder $|E| + |F| = |V| + 2$; als deren Beschreibungskomplexität ergibt sich daraus:

$$|\mathcal{P}| = 2\,|V| + 2 = \Theta(|V|).$$

Die von den Flächen eines Polyeders $\mathcal{P}$ beschränkte Region heißt *Inneres*, die unbeschränkte Region *Äußeres* des Polyeders. Zu jeder Polyederfläche f bestimmen wir ihren Normalenvektor $\mathbf{n_f}$ als den der zugehörigen Ebene, der in das Polyederäußere gerichtet ist (siehe Abb. 1.1). In der Montage werden in der Regel einfache Werkteile zu komplexeren zusammengesetzt. Da wir das Problem der *exakten* Kollisionserkennung im Raum auf geometrischem Wege anpacken wollen, liegt es für die Modellierung der Objekte daher nahe, diese durch starre einfache Polyeder zu beschreiben. Dies kann, wie wir gesehen haben, in effizienter Weise geschehen; zudem lassen sich real vorkommende Körper beliebig genau durch solche Polyeder approximieren. Nicht-konvexe Objekte können wir aus konvexen Polyedern durch Vereinigung modellieren.

Um *exakte* Berechnungen zu ermöglichen, ist es notwendig, daß bereits die Eingabedaten - also Ausgangs- und Zwischenlage der Objekte - präzise sind: dies läßt sich beispielsweise dadurch erreichen, daß man nur ganzzahlige Koordinaten für die Polyedereckpunkte erlaubt. Die zu erreichende Eingabegenauigkeit bestimmt hierbei die Größe dieser Zahlen.

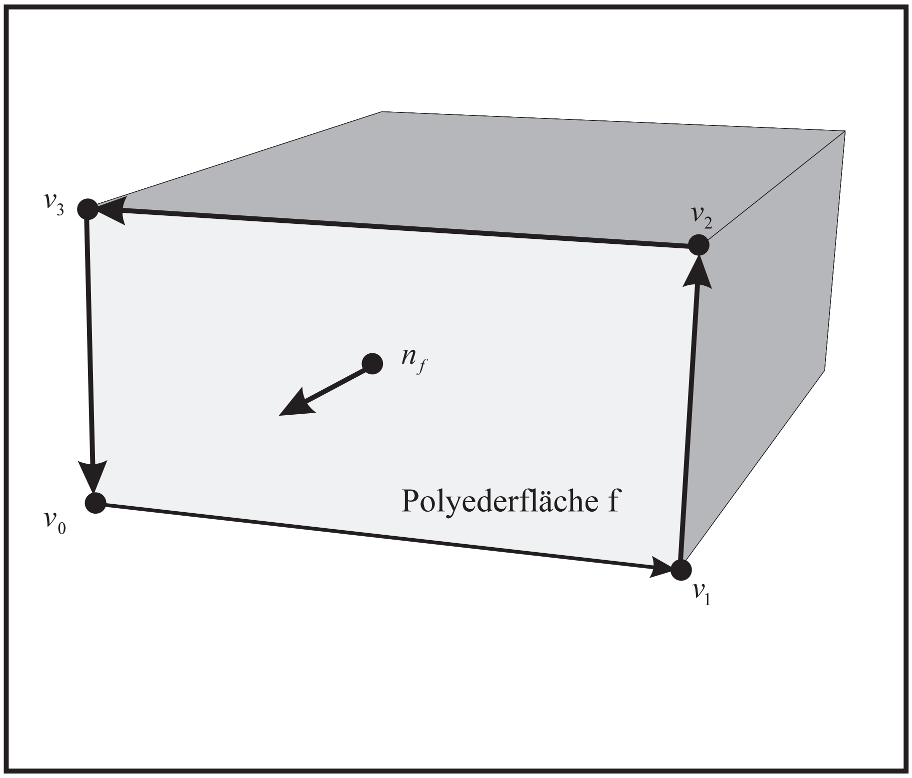

Abbildung 1.1: BESCHREIBUNG EINER POLYEDERFLÄCHE

1.2.2 Modellierung der Konfigurationen und Bewegungen

Betrachten wir im folgenden ein polyhedral modelliertes Objekt $\mathcal{P}$, das sich frei im Raum bewegen kann. Erinnern wir uns an die Beschreibung unserer Montagezelle: sollte dort ein Objekt aus einer Ausgangs- in eine Ziellage gebracht werden, so geschah dies durch Angabe einer Folge von Zwischenlagen zu bestimmten Zeitpunkten. Ein Übergang zwischen diesen Zwischenlagen, den das System eigenständig ausführen soll, entspricht dann einer Bewegung. Eine solche Zwischenlage - auch Konfiguration genannt - soll hierbei Position und Orientierung des Objektes bzgl. seiner Ausgangslage spezifizieren. Da die Art der Repräsentation der Objektkonfigurationen wesentliche Auswirkungen auf die Laufzeit und die Exaktheit der später folgenden Algorithmen mit sich bringt, müssen wir um eine möglichst kompakte und effiziente Parametrisierung bemüht sein. Im nächsten Abschnitt werden wir deshalb u.a. eine nicht alltägliche Darstellungsweise für Rotationen von Vektoren kennenlernen: die Quaternionen.

1.2.2.1 Die Lage von Objekten im Raum

Im folgenden gehen wir davon aus, daß die Koordinaten unserer Objekte bzgl. eines globalen Referenzkoordinatensystems, dem sogenannten *Weltkoordinatensystem*, vorliegen; dabei ist es, wie wir später oft sehen werden, von Vorteil, nur *Orthonormalsysteme* zu betrachten.

Um die Lage eines starren Objektes im Raum zu beschreiben, müssen wir die Position und die Orientierung des Objektes bzgl. des Weltkoordinatensystems angeben. Hierzu ordnen wir jedem Objekt ein *körpereigenes Koordinatensystem* zu und beschreiben die relative Lage der beiden Systeme zueinander. Da wir lediglich Rotationen um eine feste Achse zulassen, die durch den Ursprung des körpereigenen Koordinatensystems verläuft, ist bei der Wahl seiner Lage der Objekttyp von entscheidender Bedeutung:

- bei Objekten, die z.B. als Teil eines Roboters modelliert werden (z.B. Roboterarm), sollte der Ursprung im entsprechenden Gelenk liegen

- bei der Modellierung von im Raum frei beweglichen Objekten ist es sinnvoll, den Ursprung in den Schwerpunkt des Objektes zu verlegen

Zur Festlegung der Position ist es ausreichend, den Verschiebungsvektor zwischen den beiden Koordinatenursprüngen anzugeben. Der Startpunkt dieses Vektors ist dabei der Ursprung des Weltkoordinatensystems.

Mit der aktuellen Orientierung eines starren Körpers im Raum bezeichnet man die Orientierung des körperfesten Koordinatensystems relativ zum globalen Weltkoordinatensystem bzgl. dessen die Objekte definiert sind; dieser Übergang entspricht einer Basistransformation, die sich durch eine orthonormale 3×3-Matrix beschreiben läßt. Jede Orientierungsänderung eines starren Körpers läßt sich gemäß einem Satz von Euler durch eine Rotation um eine geeignete Achse erreichen. Die Beschreibung von Orientierungen durch solche 3×3-Matrizen ist redundant, denn zur eindeutigen Festlegung der Orientierung eines Dreibeins genügen 3 Parameter (z.B. Eulerwinkel). Eine weitere Alternative stellen die *Quaternionen* dar. Hier benutzt man zur Spezifikation der Orientierung eine Rotationsachse $\mathbf{r}$ und einen Winkel α. Da diese Beschreibungsmöglichkeit nicht so populär ist, werden wir ihr einen eigenen Abschnitt zukommen lassen.

1.2.2.2 Beschreibung von Orientierungen durch Quaternionen

Die Menge $\mathbb{R}^4$ bildet mit der üblichen Vektoraddition und einer geeignet gewählten Multiplikation einen nichtkommutativen Ring mit Eins, in dem alle Elemente außer der Null invertierbar sind. Diese algebraische Struktur heißt auch Schiefkörper. Die Elemente dieses Schiefkörpers bezeichnet man auch als *Hamiltonsche Quaternionen*. Diese bieten - wie wir in diesem Abschnitt noch sehen werden - eine elegante Darstellungsmöglichkeit für Orientierungen und Rotationen. Ein Quaternion $\mathfrak{q} = (q_0, q_1, q_2, q_3)^T \in \mathbb{R}^4$ werden wir in Zukunft als $\mathfrak{q} = (q_0, \mathbf{q})$ schreiben, wobei q_0 der Skalaranteil und $\mathbf{q}= (q_1, q_2, q_3)^T$ der Vektoranteil des Quaternions $\mathfrak{q}$ genannt wird. Dies ergibt in natürlicher Weise Inklusionen von $\mathbb{R}$ und $\mathbb{R}^3$ in die Menge $\mathbb{R}^4$ der Quaternionen. Als Multiplikation definiert man die bilineare Zuordnung

$$\mathfrak{p} \cdot \mathfrak{q} = (p_0, \mathbf{p}) \cdot (q_0, \mathbf{q}) := (p_0 q_0 - \mathbf{p}^T \mathbf{q}, p_0 \mathbf{q} + q_0 \mathbf{p} + \mathbf{p} \times \mathbf{q}) \tag{1.1}$$

Das zu einem Quaternion $\mathfrak{q}$ *konjugierte* Quaternion $\mathfrak{q}^*$ erhält man durch Negation des Vektoranteils: $\mathfrak{q} = (q_0, \mathbf{q}) \mapsto \mathfrak{q}^* = (q_0, -\mathbf{q})$. Den Vektoranteil eines Quaternions $\mathfrak{q}$ bezeichnen wir mit $[\mathfrak{q}]$. Der skalare Wert $\mathfrak{q} \cdot \mathfrak{q}^* = q_0^2 + \mathbf{q}^2$ entspricht dem Quadrat der Länge des Quaternions $\mathfrak{q}$ unter der euklidischen Metrik. Quaternionen, die die Bedingung $\mathfrak{q} \cdot \mathfrak{q}^* = 1$ erfüllen, heißen *Einheitsquaternionen*; sie bilden mit der oben definierten Multiplikation eine Gruppe und nehmen - wie dem folgenden Satz zu entnehmen ist - für unsere Anwendungen eine Sonderstellung ein:

Satz 1.8:
Die Abbildung $R_{\mathbf{r},\alpha} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ *mit* $\mathbf{x} \mapsto [\mathfrak{x}'] = [\mathfrak{q} \cdot (0, \mathbf{x}) \cdot \mathfrak{q}^*]$ *und* $\mathfrak{q} = \mathfrak{q}_{\mathbf{r},\alpha} = (cos(\frac{\alpha}{2}), sin(\frac{\alpha}{2})\mathbf{r})$ *entspricht einer Rotation des Vektors* $\mathbf{x}$ *um die Achse* $\mathbf{r}$ *um den Winkel* α. *Mit* $|\mathbf{r}| = 1$ *gilt hierbei* $\mathfrak{q} \cdot \mathfrak{q}^* = 1$.

Beweis:
Zunächst werden wir das doppelte Quaternionenprodukt gemäß 1.1 expandieren:

$$\begin{aligned}
\mathfrak{q} \cdot (0, \mathbf{x}) \cdot \mathfrak{q}^* &= \Big(cos(\frac{\alpha}{2}), sin(\frac{\alpha}{2})\mathbf{r}\Big) \cdot \Big(0, \mathbf{x}\Big) \cdot \Big(cos(\frac{\alpha}{2}), -sin(\frac{\alpha}{2})\mathbf{r}\Big) \\
&= \Big(cos(\frac{\alpha}{2}), sin(\frac{\alpha}{2})\mathbf{r}\Big) \cdot \Big(sin(\frac{\alpha}{2})\mathbf{x}^T\mathbf{r}, cos(\frac{\alpha}{2})\mathbf{x} + sin(\frac{\alpha}{2})\mathbf{r} \times \mathbf{x}\Big) \\
&= \Big(- sin^2(\frac{\alpha}{2})\mathbf{r}^T(\mathbf{r} \times \mathbf{x}), cos^2(\frac{\alpha}{2})\mathbf{x} + 2sin(\frac{\alpha}{2})cos(\frac{\alpha}{2})\mathbf{r} \times \mathbf{x} + \\
&\quad sin^2(\frac{\alpha}{2})\mathbf{r}^T\mathbf{x}\mathbf{r} + sin(\frac{\alpha}{2})cos(\frac{\alpha}{2})\mathbf{r} \times \mathbf{x} + sin^2(\frac{\alpha}{2})\mathbf{r} \times (\mathbf{r} \times \mathbf{x})\Big) \\
&= \Big(0, cos^2(\frac{\alpha}{2})\mathbf{x} + 2sin(\frac{\alpha}{2})cos(\frac{\alpha}{2})\mathbf{r} \times \mathbf{x} + sin^2(\frac{\alpha}{2})\mathbf{r}^T\mathbf{x}\mathbf{r} + \\
&\quad sin(\frac{\alpha}{2})cos(\frac{\alpha}{2})\mathbf{r} \times \mathbf{x} + sin^2(\frac{\alpha}{2})(\mathbf{r}^T\mathbf{x}\mathbf{r} - \mathbf{x})\Big) \\
&= \Big(0, (cos^2(\frac{\alpha}{2}) - sin^2(\frac{\alpha}{2}))\mathbf{x} + 2sin(\frac{\alpha}{2})cos(\frac{\alpha}{2})\mathbf{r} \times \mathbf{x} + 2sin(\frac{\alpha}{2})\mathbf{r}^T\mathbf{x}\mathbf{r}\Big) \\
&= \Big(0, cos(\alpha)\mathbf{x} + sin(\alpha)\mathbf{r} \times \mathbf{x} + (1 - cos(\alpha))\mathbf{r}^T\mathbf{x}\mathbf{r}\Big) \\
&= (0, \mathbf{x}')
\end{aligned}$$

Bei der Umformung der trigonometrischen Terme haben wir deren in Abschnitt A.4 beschriebene Zusammenhänge ausgenutzt. Die vorletzte Zeile beschreibt - wie wir aus Formel A.7 wissen - eine Rotation des Punktes $\mathbf{x}$ um die Achse $\mathbf{r}$ und den Winkel α im Gegenuhrzeigersinn. Damit ist Satz 1.8 bewiesen. ■

Die Tatsache, daß die Einheitsquaternionen mit der zugehörigen Multiplikation die Gruppenstruktur besitzen, ermöglicht es uns, hintereinanderausgeführte Rotationen ein und desselben Objektes durch das Produkt der zugehörigen Quaternionen zu beschreiben. Bezeichnen wir mit $R_\mathfrak{q}$ eine Rotation $R_{\mathbf{r},\alpha}$ um eine feste Achse $\mathbf{r}$ und einen Winkel α, wie sie durch die Abbildungsvorschrift 1.8 im Quaternionenkalkül beschrieben werden kann, so gilt:

$$R_\mathfrak{p} \circ R_\mathfrak{q} = R_{\mathfrak{p} \cdot \mathfrak{q}}$$

Die Darstellungsweise von Orientierungen durch Quaternionen definiert einen Homomorphismus

$$\varphi : S^3 \rightarrow SO(3)$$

vom Raum S^3 der Einheitsquaternionen in die Gruppe $SO(3)$ der Rotationen im $\mathbb{R}^3$. Daß es sich dabei nicht um einen Isomorphismus handeln kann, ergibt sich bereits aus der Tatsache, daß die beiden Quaternionen $\mathfrak{q}$ und $-\mathfrak{q}$ die gleiche Rotation beschreiben. $\mathfrak{q}$ und $\mathfrak{q}^*$ repräsentieren zueinander inverse Rotationen.
Expandieren wir den durch die Abbildungsvorschrift in Satz 1.8 gegebenen Ausdruck

$$\begin{aligned} \mathfrak{x}' &= \mathfrak{q} \cdot \mathfrak{x} \cdot \mathfrak{q}^* \\ &= \begin{bmatrix} q_0 \\ \mathbf{q} \end{bmatrix} \circ \begin{bmatrix} 0 \\ \mathbf{x} \end{bmatrix} \circ \begin{bmatrix} q_0 \\ -\mathbf{q} \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ (q_0^2 - \mathbf{q}^2)\mathbf{x} + 2(\mathbf{q}^T\mathbf{x})\mathbf{q} + 2q_0(\mathbf{q} \times \mathbf{x}) \end{bmatrix} \end{aligned}$$

so erhalten wir für den aus der Rotation des Vektors $\mathbf{x}$ hervorgegangenen Vektor $\mathbf{x}'$:

$$\mathbf{x}' = (q_0^2 - \mathbf{q}^2)\mathbf{x} + 2(\mathbf{q}^T\mathbf{x})\mathbf{q} + 2q_0(\mathbf{q} \times \mathbf{x})$$

Dieser Vorgang läßt sich eleganter, wie in affinen Räumen üblich, durch Multiplikation des Vektors $\mathbf{x}$ mit einer orthonormalen Matrix $\mathbf{Q}$ beschreiben:

$$\mathbf{x}' = \mathbf{Q} \cdot \mathbf{x}$$

Für die Matrix $\mathbf{Q}$ gilt hierbei:

$$\mathbf{Q} = (q_0^2 - \mathbf{q}^2) \cdot \mathbf{E} + 2\mathbf{q}\mathbf{q}^T + 2q_0\mathbf{Q}^\times \tag{1.2}$$

$\mathbf{Q}^\times$ bezeichnet die gemäß A.1 aus den Komponenten des Vektors $\mathbf{q}$ aufgebaute Matrix, deren Produkt mit dem Vektor $\mathbf{x}$ als Ergebnis gerade das Vektorprodukt $\mathbf{q} \times \mathbf{x}$ liefert.

Ein wesentlicher Vorteil der Repräsentation von Orientierungen bzw. Rotationen durch Quaternionen liegt in der eleganten und kompakten Beschreibungsmöglichkeit; doch ist es an dieser Stelle auch notwendig, sich über etwaige Schwierigkeiten bei deren Benutzung klar zu werden:

- Jede Rotation bzw. Orientierung eines Objektes kann durch zwei verschiedene Quaternionen repräsentiert werden, da eine Rotation um die Achse $\mathbf{r}$ und den Winkel α das gleiche Ergebnis wie die Rotation um die Achse -$\mathbf{r}$ und den Winkel $-\alpha$ liefert; für die hierzu korrespondierenden Quaternionen $\mathfrak{q}_{\mathbf{r},\alpha}$ und $\mathfrak{q}_{-\mathbf{r},-\alpha}$ gilt, wie sich anhand obiger Abbildungsvorschrift nachvollziehen läßt: $\mathfrak{q}_{-\mathbf{r},-\alpha} = -\mathfrak{q}_{\mathbf{r},\alpha}$.

- Orientierungen und Rotationen sind nicht exakt das gleiche: eine Rotation eines Objektes um 360° ist unter dem Aspekt der Kollisionsberechnung verschieden von einer Rotation um 0°; beide werden jedoch durch dasselbe Quaternion $\mathfrak{q} = (1, 0, 0, 0)$ beschrieben. Um Rotationen eindeutig durch Quaternionen zu repräsentieren, sind daher dazwischenliegende Kontrollpunkte erforderlich.

Kehren wir nun von diesen allgemeinen Betrachtungen zu unserer speziellen Anwendung zurück: des *exakten* Rechnens wegen verwenden wir zur Beschreibung unserer polyhedral modellierten Objekte nur ganzzahlige Koordinaten. Bei der Beschreibung von Rotationen durch ganzzahlige *Nicht-Einheitsquaternionen* müssen wir deren Länge berücksichtigen:

$$\begin{aligned}
\mathfrak{x}' &= \frac{\mathfrak{q}}{|\mathfrak{q}|} \cdot \mathfrak{x} \cdot \frac{\mathfrak{q}^*}{|\mathfrak{q}^*|} \\
&= \frac{\mathfrak{q} \cdot \mathfrak{x} \cdot \mathfrak{q}^*}{\mathfrak{q} \cdot \mathfrak{q}^*} \\
&= \frac{1}{q_0^2 + \mathbf{q}^2} \cdot \mathfrak{q} \cdot \mathfrak{x} \cdot \mathfrak{q}^*
\end{aligned}$$

In Matrixschreibweise erhalten wir mit $\mathbf{Q}$ aus 1.2:

$$\mathbf{x}' = \frac{\mathbf{Q}}{q_0^2 + \mathbf{q}^2} \cdot \mathbf{x}$$

Damit wir ein Objekt uneingeschränkt aus einer beliebig möglichen Ausgangs- in jede beliebig mögliche Zielorientierung bringen können, ist es notwendig, ein *ganzzahliges* Quaternion angeben zu können, das gemäß 1.8 die hierzu erforderliche Rotation beschreibt:

Satz 1.9:
Seien $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^3$ *mit* $|\mathbf{a}| = |\mathbf{b}|$. *Dann gilt:*

$$\exists\, \mathfrak{q} \in \mathbb{Z}^4 \ \mathit{mit}\ \mathbf{b} = \left[\frac{\mathfrak{q} \cdot (0, \mathbf{a}) \cdot \mathfrak{q}^*}{\mathfrak{q} \cdot \mathfrak{q}^*}\right]$$

Beweis:
Seien $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^3$ zwei Vektoren mit $|\mathbf{a}| = |\mathbf{b}|$. Wollen wir den Vektor $\mathbf{a}$ durch Rotation um eine feste Achse mit dem Vektor $\mathbf{b}$ zur Deckung bringen, so kommt als Rotationsachse $\mathbf{r}$ der Normalenvektor der Ebene in Frage, in der sich diese Drehung vollzieht: $\mathbf{r} = \mathbf{a} \times \mathbf{b}$. Trivialerweise gilt: $\mathbf{r} \in \mathbb{Z}^3$.

Sei $\mathfrak{r} = (r_0, \mathbf{r})$ das Quaternion, mittels dessen die Rotation beschrieben wird. Dann gilt:

$$\begin{array}{rrcl}
 & \mathbf{b} & = & \frac{1}{r_0^2+\mathbf{r}^2} \cdot ((r_0^2 - \mathbf{r}^2)\mathbf{a} + 2(\mathbf{r}^T\mathbf{a})\mathbf{r} + 2r_0\mathbf{r} \times \mathbf{a}) \\
\Longleftrightarrow & \mathbf{b}r_0^2 + \mathbf{b}\mathbf{r}^2 & = & r_0^2\mathbf{a} - \mathbf{r}^2\mathbf{a} + 2\mathbf{r}\mathbf{r}^T\mathbf{a} + 2r_0\mathbf{r} \times \mathbf{a} \\
\Longleftrightarrow & r_0^2(\mathbf{b}-\mathbf{a}) - 2r_0(\mathbf{r} \times \mathbf{a}) & = & -\mathbf{r}^2(\mathbf{a}+\mathbf{b}) + 2\mathbf{r}(\mathbf{r}^T\mathbf{a}) \\
\Longrightarrow & r_0^2\mathbf{a}^T(\mathbf{b}-\mathbf{a}) - \underbrace{2r_0\mathbf{a}^T(\mathbf{r} \times \mathbf{a})}_{=0} & = & -\mathbf{r}^2\mathbf{a}^T(\mathbf{a}+\mathbf{b}) + 2(\mathbf{r}^T\mathbf{a})^2 \\
\Longleftrightarrow & r_0^2 & = & \frac{-\mathbf{r}^2(\mathbf{a}^2+\mathbf{a}^T\mathbf{b})+2(\mathbf{a}^T\mathbf{r})^2}{\mathbf{a}^T(\mathbf{b}-\mathbf{a})} \\
\Longleftrightarrow & r_0^2 & = & \frac{(\mathbf{a}\times\mathbf{b})^2\mathbf{a}^T(\mathbf{a}+\mathbf{b})+2(\mathbf{a}^T(\mathbf{a}\times\mathbf{b}))^2}{\mathbf{a}^T(\mathbf{a}-\mathbf{b})} \\
\Longleftrightarrow & r_0^2 & = & \frac{(\mathbf{a}\times\mathbf{b})^2\mathbf{a}^T(\mathbf{a}+\mathbf{b})}{\mathbf{a}^T(\mathbf{a}-\mathbf{b})} \\
\Longleftrightarrow & r_0^2 & = & \frac{(\mathbf{a}^2\mathbf{b}^2-(\mathbf{a}^T\mathbf{b})^2)\mathbf{a}^T(\mathbf{a}+\mathbf{b})}{\mathbf{a}^T(\mathbf{a}-\mathbf{b})} \\
\Longleftrightarrow & r_0^2 & = & \frac{((\mathbf{a}^2)^2-(\mathbf{a}^T\mathbf{b})^2)\mathbf{a}^T(\mathbf{a}+\mathbf{b})}{\mathbf{a}^T(\mathbf{a}-\mathbf{b})} \\
\Longleftrightarrow & r_0^2 & = & \frac{(\mathbf{a}^2-\mathbf{a}^T\mathbf{b})(\mathbf{a}^2+\mathbf{a}^T\mathbf{b})(\mathbf{a}^2+\mathbf{a}^T\mathbf{b})}{(\mathbf{a}^2-\mathbf{a}^T\mathbf{b})} \\
\Longleftrightarrow & r_0^2 & = & (\mathbf{a}^2+\mathbf{a}^T\mathbf{b})(\mathbf{a}^2+\mathbf{a}^T\mathbf{b}) \\
\Longleftrightarrow & r_0^2 & = & (\mathbf{a}^2+(\mathbf{a}^T\mathbf{b}))^2 \\
\Longleftrightarrow & r_0 & = & \pm(\mathbf{a}^2+(\mathbf{a}^T\mathbf{b}))
\end{array}$$

Bei den obigen Umformungen haben wir ausgenutzt, daß $\mathbf{a}^T(\mathbf{a} \times \mathbf{b}) = 0$. Desweiteren wurde die Lagrangesche Identität A.3 angewendet. Aufgrund der Tatsache, daß sich der Punkt $\mathbf{a}$ auf einer Kreisbahn um die Rotationsachse $\mathbf{r}$, die durch den Ursprung verläuft, bewegt, gilt, wie man sich schnell überlegt, $\mathbf{a}^T(\mathbf{b}-\mathbf{a}) \neq 0$. Anhand der durchgeführten Umformungen läßt sich die Ganzzahligkeit von r_0 erkennen.
Insgesamt haben wir gezeigt, daß $\mathfrak{r} \in \mathbb{Z}^4$. ∎

Nachdem wir wissen, wie wir im Rahmen unseres Modelles Orientierungen bzw. Rotationen mittels ganzzahliger Quaternionen repräsentieren können, sind wir nun in der Lage, die Konfigurationen unserer Objekte zu definieren.

1.2.2.3 Konfigurationen und parametrisierte Konfigurationsübergänge

Im letzten Abschnitt haben wir erfahren, wie die augenblickliche Lage eines Objektes im Raum durch Angabe

- eines *Positionsvektors*, der die Verschiebung des Objektes gegenüber seiner Ausgangslage charakterisiert, und
- eines *Quaternions*, das die Rotation des Objektes um eine feste Achse und einen bestimmten Winkel beschreibt,

repräsentiert werden kann. Die Konfiguration eines Objektes wird innerhalb unseres Modells demnach durch ein 7-Tupel $\mathcal{C} \in \mathbb{Z}^7$ festgelegt.

Um die Konfigurationen mehrerer Objekte zu verschiedenen Zeitpunkten in einfacher Weise spezifizieren zu können, wollen wir folgende Notation einführen:

Bezeichnung 1.10:
Mit $\mathcal{C}_i(k) = (\mathbf{o}_i^k, \mathfrak{p}_i^k)$ bezeichnen wir die Konfiguration des Polyeders $\mathcal{P}_i$ zu einem Zeitpunkt $k \in \mathbb{N}$. Hierbei gilt:

- $\mathbf{o}_i^k \in \mathbb{Z}^3$ *spezifiziert die Verschiebung des Objektes gegenüber seiner Ausgangsposition*
- $\mathfrak{p}_i^k \in \mathbb{Z}^4$ *spezifiziert die Orientierung des Objektes bzgl. seiner Ausgangsorientierung*

Die *Bewegung* eines polyhedralen Objektes $\mathcal{P}_i$ ergibt sich nun als stetiger Übergang zwischen zwei zeitlich aufeinanderfolgenden Konfigurationen $\mathcal{C}_i(k)$ und $\mathcal{C}_i(k+1)$. Dieser Übergang geschieht durch Angabe einer stetigen, über dem Zeitintervall $[0,1]$ parametrisierten Interpolationsfunktion, so daß sich eine Zwischenlage wie folgt berechnen läßt:

$$\mathcal{C}_i(k+t) \ = I(t)(\mathcal{C}_i(k), \mathcal{C}_i(k+1)) \text{ mit } 0 \le t \le 1$$

Anfangs- und Endpunkt der Interpolationsfunktion I müssen die entsprechenden Konfigurationen sein, d.h. es muß gelten:

- $I(0)(\mathcal{C}_i(k), \mathcal{C}_i(k+1)) = \mathcal{C}_i(k)$ und
- $I(1)(\mathcal{C}_i(k), \mathcal{C}_i(k+1)) = \mathcal{C}_i(k+1)$.

Ist der zeitliche Abstand der interaktiv vorgegebenen Konfigurationen $\mathcal{C}_i(k)$ eines Objektes $\mathcal{P}_i$ nicht zu groß, was i.a. eine nicht allzu gravierende Änderung der Lage des Objektes zwischen zwei aufeinanderfolgenden Konfigurationen impliziert, so scheint eine *lineare Interpolation* zur Berechnung der Zwischenkonfigurationen durchaus berechtigt zu sein:

$$\begin{aligned} \mathcal{C}_i(k+t) &= \mathcal{C}_i(k) \ + t\,(\mathcal{C}_i(k+1) - \mathcal{C}_i(k)) \text{ mit } 0 \le t \le 1 \\ &= \begin{bmatrix} \mathbf{o}_i^k \\ \mathfrak{p}_i^k \end{bmatrix} + t \cdot \left(\begin{bmatrix} \mathbf{o}_i^{k+1} \\ \mathfrak{p}_i^{k+1} \end{bmatrix} - \begin{bmatrix} \mathbf{o}_i^k \\ \mathfrak{p}_i^k \end{bmatrix} \right) \end{aligned}$$

Im nun folgenden Teil der Arbeit werden wir stets die *lineare Interpolation* benutzen. Dies läßt sich durch das Wesen unserer Modellierung durchaus begründen und bringt einige Vorteile mit sich:

- einfache Berechnung der Zwischenkonfigurationen mit linearer Zeitabhängigkeit

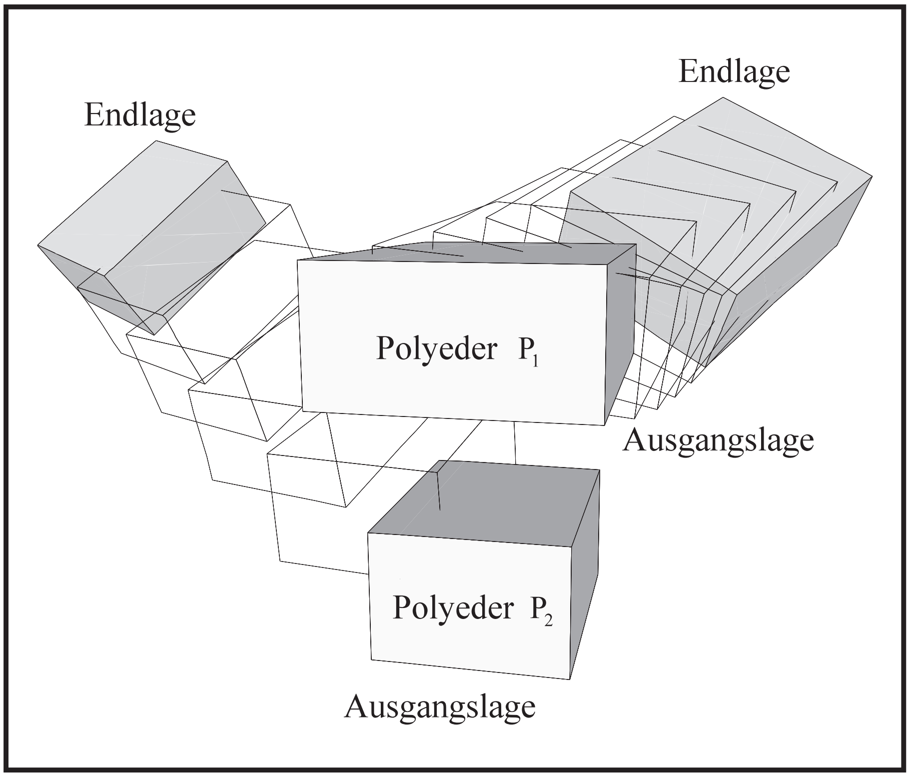

Abbildung 1.2: linear interpolierte Zwischenkonfigurationen zweier gleichzeitig bewegter Objekte

- niedrigerer Grad der polynomiellen Gleichungen, die die Kollisionsbedingungen beschreiben

Allerdings möchten wir erwähnen, daß alle vorgestellten Verfahren sowie alle implementierten Algorithmen auch für Interpolationspolynome höheren Grades funktionieren. Eine lineare Interpolation zwischen zwei Punkten eines Raumes beliebiger Dimension beschreibt die kürzeste Verbindungsstrecke zwischen ihnen in diesem Raum. Die Konfigurationen unserer betrachteten Objekte sind ganzzahlige 7-Tupel, die aus einem 3-Tupel (Position) und einem 4-Tupel (Orientierung) zusammengesetzt sind. Betrachten wir nun ein Objekt $\mathcal{P}_i$, das während des Übergangs aus der Konfiguration $\mathcal{C}_i(k)$ in die Konfiguration $\mathcal{C}_i(k+1)$ sowohl seine Position, als auch seine Orientierung ändert: die lineare Interpolation im 3-dimensionalen Raum der Translationen beschreibt in diesem Fall offensichtlich die Punkte der Strecke $\overline{\mathbf{o}_i^k\mathbf{o}_i^{k+1}}$, so daß sich das Objekt auch tatsächlich geradlinig aus seiner Ausgangs- in seine Endposition bewegt.

Bei der linear parametrisierten Beschreibung des Übergangs aus einer Ausgangsorientierung $\mathfrak{p}_i^k$ in eine Zielorientierung $\mathfrak{p}_i^{k+1}$ ist dem nicht so: Orientierungen werden mittels Einheitsquaternionen beschrieben, die die Oberfläche der 4-dimensionalen Einheitskugel $\mathcal{K}^4$ bilden. In [Cha95] wurde gezeigt, daß Großkreise in $\mathcal{K}^4$ und die Lagen eines Körpers während Volldrehungen einander entsprechen. Die Drehung eines Objektes um eine *feste* Achse mit *konstanter* Winkelgeschwindigkeit wird durch die Quaternionendarstellung auf eine Geodäte des $\mathcal{K}^4$ abgebildet. Als direkter Weg aus einer Start- in eine Zielorientierung wird jeweils der kürzere Großkreisbogen angesehen, der beide Punkte enthält. Nun stehen wir allerdings vor dem Problem, daß ein solcher stetiger Weg im $\mathcal{K}^4$ nicht linear beschreibbar ist. Die lineare Interpolation zwischen zwei Punkten der 4-dimensionalen Einheitskugel liefert lediglich einen stetigen Weg durch ihr Inneres. Bei dieser Darstellungsweise der Zwischenkonfigurationen treten zwei Fragen auf:

- Bleibt die Rotationsachse bei der linearen Interpolation zwischen zwei Orientierungen fest?
- Wie ändert sich die Rotationsgeschwindigkeit des Objektes, wenn der kürzeste Weg auf der Oberfläche der 4-dimensionalen Einheitskugel $\mathcal{K}^4$ zwischen zwei Punkten durch einen geradlinigen Weg durch ihr Inneres ersetzt wird?

Satz 1.11 :
Seien $\mathfrak{p}$ und $\mathfrak{q}$ zwei Einheitsquaternionen, die die Orientierung eines Objektes zu verschiedenen Zeitpunkten darstellen. Dann beschreiben die auf der Strecke

$$\mathfrak{r}(t) = \mathfrak{p} + t \cdot (\mathfrak{q} - \mathfrak{p}) \text{ mit } t \in [0,1] \tag{1.3}$$

liegenden Quaternionen eine Rotation des Objektes um eine feste Achse.

Beweisidee:
Sei $\mathbf{x}$ ein Punkt eines Objektes, das sich aus der Orientierung $\mathfrak{p}$ in die Orientierung $\mathfrak{q}$ bewegt. Wie wir gesehen haben, läßt sich dieser Vorgang wie folgt beschreiben:

$$\begin{aligned} \mathfrak{r}(t) &= \begin{pmatrix} r_0(t) \\ \mathbf{r}(t) \end{pmatrix} = \mathfrak{p} + t \cdot (\mathfrak{q} - \mathfrak{p}) \text{ mit } t \in [0,1] \\ \Longrightarrow \mathbf{x}(t) &= \frac{1}{r_0(t)^2 + \mathbf{r}(t)^2}\Big((r_0(t)^2 - \mathbf{r}(t)^2)\mathbf{x} + 2(\mathbf{r}(t)^T\mathbf{x})\mathbf{r}(t) + 2r_0(t)(\mathbf{r}(t) \times \mathbf{x})\Big) \\ &= \mathbf{R}(t) \cdot \mathbf{x} \end{aligned} \tag{1.4}$$

Hierbei gilt: $\mathbf{R}(t) \cdot \mathbf{R}(t)^T = \mathbf{E}$ und $det(\mathbf{R}(t)) = 1$. Für die momentane Rotationsachse bzw. -geschwindigkeit ω gilt laut [Cra89]:

$$\begin{aligned} \frac{\partial \mathbf{x}(t)}{\partial t} &= \omega(t) \times \mathbf{x}(t) \\ &= \Omega(t) \cdot \mathbf{x}(t) \end{aligned} \tag{1.5}$$

Der Vektor $\omega(t)$ beschreibt die Rotationsachse, seine Länge die Rotationsgeschwindigkeit. $\Omega(t)$ bezeichnet die gemäß A.1 definierte Matrix zur multiplikativen Beschreibung des Vektorproduktes. Desweiteren gilt:

$$\frac{\partial \mathbf{x}(t)}{\partial t} = \frac{\partial \mathbf{R}(t)}{\partial t} \cdot \mathbf{x} \tag{1.6}$$

Gleichsetzen von 1.5 und 1.6 ergibt dann:

$$\begin{aligned}
\Omega(t) \cdot \mathbf{x}(t) &= \frac{\partial \mathbf{R}(t)}{\partial t} \cdot \mathbf{x} \\
\Omega(t) \cdot \mathbf{R}(t) \cdot \mathbf{x} &= \frac{\partial \mathbf{R}(t)}{\partial t} \cdot \mathbf{x} \\
\Omega(t) \cdot \mathbf{R}(t) &= \frac{\partial \mathbf{R}(t)}{\partial t} \\
\Omega(t) &= \frac{\partial \mathbf{R}(t)}{\partial t} \cdot \mathbf{R}(t)^T \qquad (1.7)
\end{aligned}$$

Eine Expansion dieses Ausdrucks mit dem Algebrasystem MAPLE zeigt, daß die Einträge $\omega_i(t)$, $i \in \{1,2,3\}$, der Matrix

$$\Omega(t) = \frac{2}{|\mathfrak{r}(t)|^2} \cdot \begin{bmatrix} 0 & -\omega_3(t) & \omega_2(t) \\ \omega_3(t) & 0 & -\omega_1(t) \\ -\omega_2(t) & \omega_1(t) & 0 \end{bmatrix} \qquad (1.8)$$

gerade dem Vektoranteil des Quaternionenproduktes $\mathfrak{q} \cdot \mathfrak{p}^*$ entsprechen: $\omega(t) = [\mathfrak{q} \cdot \mathfrak{p}^*]$. Damit ist die Richtung des Vektors $\omega(t)$ unabhängig vom Zeitparameter t; lediglich seine Länge variiert in Abhängigkeit von der Länge des zeitlich interpolierten Quaternions $\mathfrak{r}(t) = \mathfrak{p} + t \cdot (\mathfrak{q} - \mathfrak{p})$. Dies bedeutet, daß

- die Rotationsachse bei linearer Interpolation zwischen zwei Orientierungen eines Objektes fest bleibt
- die Rotationsgeschwindigkeit nahezu konstant bleibt, wenn das interpolierte Quaternion *fast* die Länge 1 besitzt. Verläuft die interpolierte Verbindungsstrecke nahe am Nullpunkt vorbei, nimmt die Rotationsgeschwindigkeit allerdings *dramatisch* (Grenzwert ∞) zu.

■

Wollte man eine Rotation mit konstanter Winkelgeschwindigkeit beschreiben, müßte man – wie wir oben schon gesehen haben – die Bahn zwischen zwei Punkten auf der Oberfläche der vierdimensionalen Einheitskugel – dem Quaternionenraum – ganzzahlig angeben. Da dies nur in den wenigsten Fällen gelingt wird man sich mit nahezu konstanter – zumindest aber beschränkter – Rotationsgeschwindigkeit begnügen: hierzu machen wir uns die Tatsache zu Nutze, daß wir Rotationen auch durch Verwendung von vier dreidimensionalen Würfeln $W_i, i \in \{1,2,3,4\}$, im Quaternionenraum repräsentieren können. Jedes Einheitsquaternion wird wie folgt auf die Oberfläche eines bestimmten Würfels W_i projiziert: sei

$$\mathfrak{q} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}$$

ein Einheitsquaternion. Weiterhin sei

$$\begin{aligned} q_{max} &= max\{\ q_0, q_1, q_2, q_3\ \} \\ &= q_i \text{ für ein } i \in \{0,1,2,3\} \end{aligned}$$

Dann liegt das Quaternion

$$\begin{aligned} \mathfrak{q}' &= \begin{pmatrix} q_0/q_{max} \\ q_1/q_{max} \\ q_2/q_{max} \\ q_3/q_{max} \end{pmatrix} \\ &= \begin{pmatrix} q_0/q_i \\ q_1/q_i \\ q_2/q_i \\ q_3/q_i \end{pmatrix} \text{ für ein } i \in \{0,1,2,3\} \end{aligned}$$

im i-ten Würfel W_i. Ein Einheitsquaternion kann in mehrere Würfel projiziert werden, wenn mehrere Koordinaten den maximalen Wert besitzen. In dem Fall, daß

- zwei Koordinaten den maximalen Wert besitzen, wird das Einheitsquaternion auf die Flächen zweier verschiedener Würfel projiziert.
- drei Koordinaten den maximalen Wert besitzen, wird das Einheitsquaternion auf die Kanten dreier verschiedener Würfel projiziert.
- vier Koordinaten den maximalen Wert besitzen, wird das Einheitsquaternion auf die Ecken aller vier Würfel projiziert.

Diese Vorgehensweise erlaubt eine eindeutige Identifikation der Würfelflächen und liefert einen dreidimensionalen topologischen Raum, der homomorph zu P^3 und zu SO^3, der Gruppe der dreidimensionalen Rotationen, ist. Um diese Art der Beschreibung zu verdeutlichen, werden wir uns zunächst auf niederdimensionale Teilräume des Quaternionenraums zurückziehen. In Abbildung 1.3 werden die Punkte des Einheitskreises im $\mathbb{R}^2$ auf die Punkte zweier Liniensegmente abgebildet. Abbildung 1.4 zeigt das hierzu korrespondierende Szenario des $\mathbb{R}^3$: die Punkte der Einheitskugel können eindeutig auf drei geeignete Flächen des Einheitswürfels projiziert werden. Anstatt nun die Bahn auf der Kugeloberfläche zwischen zwei Punkten zeitlich parametrisiert zu beschreiben, geben wir einen Weg auf der Würfeloberfläche zwischen den korrespondierenden Projektion an, der bei Bedarf die jeweilige Flächen über Kanten bzw. Ecken hinweg wechseln kann. Kennt man diese Punkte, ist eine lineare Parametrisierung dieses Weges sehr einfach. Die Länge der so beschriebenen Quaternionen ist Element des Intervalls $[1,2]$, so daß die Rotationsgeschwindigkeit der hierdurch beschriebenen Rotation gemäß 1.8 zumindest beschränkt ist; in günstigen Fällen – der Weg darf nicht in die Nähe von Ecken oder Kanten geraten – ist sie sogar nahezu konstant.

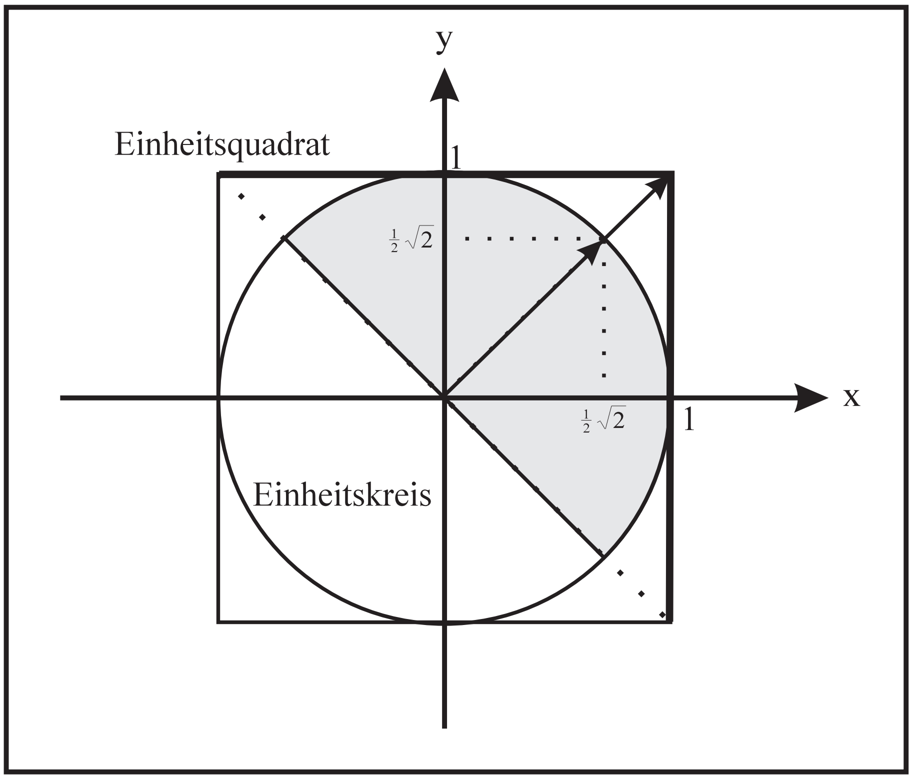

Abbildung 1.3: Prinzip der 2-Lines-Representation im $\mathbb{R}^2$

Um bei Verwendung dieser Repräsentationsmöglichkeit der Ganzzahligkeit unserer Konfigurationen Rechnung zu tragen, strecken wir die so erhaltenen Quaternionen um einen von der zu erreichenden Eingabegenauigkeit abhängigen Faktor.

Zusammenfassend stellen wir nun fest, daß sich die zeitabhängige Position und Orientierung eines Punktes $\mathbf{x}$, der sich über dem Zeitintervall $[0, 1]$ aus einer Startkonfiguration $\mathcal{C}_s = (\mathbf{o}_s, \mathfrak{p}_s)$ in eine Zielkonfiguration $\mathcal{C}_z = (\mathbf{o}_z, \mathfrak{p}_z)$ bewegt, wie folgt angeben läßt: sei

$$\mathfrak{r}(t) \quad = \quad \mathfrak{p}_s + t \cdot (\mathfrak{p}_z - \mathfrak{p}_s) = \begin{bmatrix} r_0(t) \\ \mathbf{r}(t) \end{bmatrix}$$

Die zeitabhängige Orientierungsmatrix ergibt sich nach dem bisher Gesagten zu:

$$\begin{aligned}
\mathbf{R}(t) &= \frac{1}{r_0(t)^2 + \mathbf{r}(t)^2}\Big((r_0(t)^2 - \mathbf{r}(t)^2)\mathbf{E} + 2\mathbf{r}(t)\mathbf{r}(t)^T + 2r_0(t)\mathbf{R}(t)^\times\Big) \\
&= \frac{\mathbf{R}'(t)}{N(t)} \text{ mit} \\
N(t) &= r_0(t)^2 + \mathbf{r}(t)^2 \text{ und} \\
\mathbf{R}'(t) &= \Big((r_0(t)^2 - \mathbf{r}(t)^2)\mathbf{E} + 2\mathbf{r}(t)\mathbf{r}(t)^T + 2r_0(t)\mathbf{R}(t)^\times\Big)
\end{aligned}$$

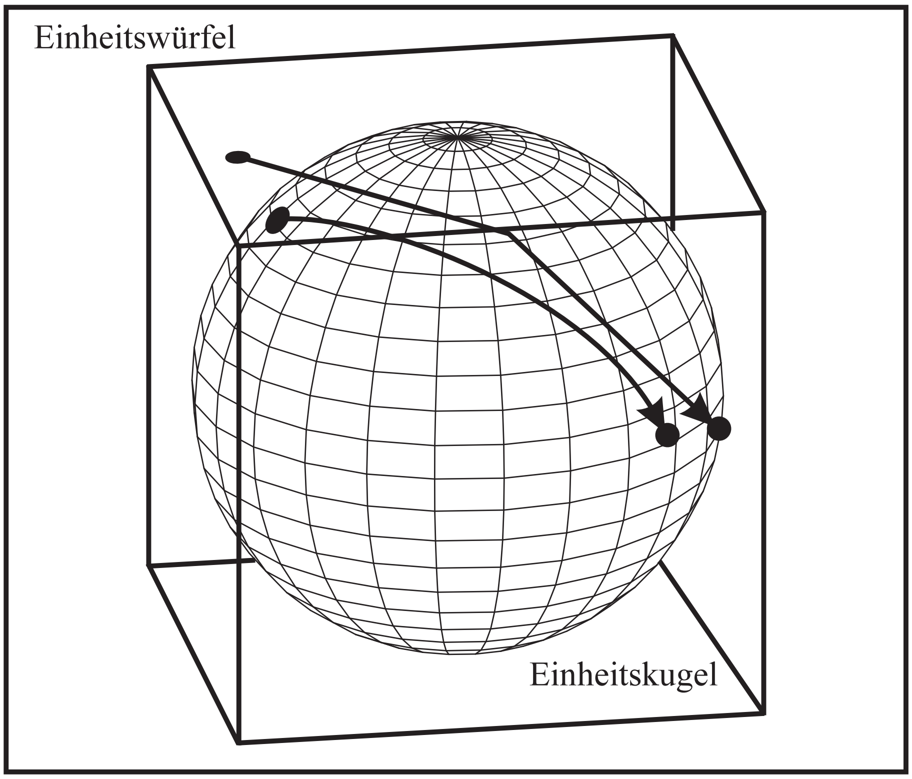

Abbildung 1.4: Prinzip der 3-Faces-Representation im $\mathbb{R}^3$

Der zeitabhängige Positionsvektor des bewegten Punktes $\mathbf{x}$ berechnet sich als:

$$\mathbf{o}(t) = \mathbf{o}_s + t(\mathbf{o}_z - \mathbf{o}_s)$$

Zusammen läßt sich dann die zeitabhängige Position und Orientierung des Punktes $\mathbf{x}$ mittels des folgenden kompakten Vektor-Matrizenausdrucks berechnen:

$$\mathbf{x}(t) = \mathbf{R}(t) \cdot \mathbf{x} + \mathbf{o}(t) \tag{1.9}$$
$$= \frac{1}{N(t)} \cdot \mathbf{R}'(t) \cdot \mathbf{x} + \mathbf{o}(t) \tag{1.10}$$

1.3 Zusammenfassung

In diesem Kapitel haben wir die mathematischen Grundlagen zur exakten geometrischen Kollisionsberechnung zusammengestellt. Von entscheidender Bedeutung war hierbei die Erstellung eines Modells, das effiziente Kollisionsberechnungen ermöglicht. Neben der Modellierung der realen Körper durch Polyeder stand eine ganzzahlige Konfigurationsbeschreibung der Objekte sowie deren Bewegungen (zeitlich parametrisierte Konfigurationsübergänge) im Mittelpunkt. Wesentlichen Anteil an der Kompaktheit der Objektkonfigurationen hatte die Beschreibung von Rotationen durch Quaternionen.

Kapitel 2

Exakte geometrische Kollisionserkennung

Das Problem der Kollisionserkennung oder der Kontaktbestimmung zwischen zwei oder mehreren Objekten ist grundlegend in der Computeranimation, der Computersimulation (z.B. virtual reality, virtuelles Prototyping), CAD-Anwendungen, der Bewegungsplanung und vielen anderen Bereichen. Je nach Einsatzgebiet ist eine mehr oder weniger hohe Präzision bei der Entscheidungsfindung bzgl. des Vorhandenseins von Kollision oder Nicht-Kollision erforderlich. So ist es z.B. bei sicherheitskritischen Anwendungen (Raumfahrt) nicht akzeptabel, auf Verfahren zurückzugreifen, die ein - wenn auch noch so kleines - potentielles Risiko für eine Fehlentscheidung in sich bergen. Für solche Einsatzgebiete sind Verfahren gefragt, die in jedem Fall die *richtige* Entscheidung treffen: die Grundlagen hierfür werden wir in diesem Kapitel schaffen, in dem wir *exakte* Kollisionsbedingungen für zwei bewegte polyhedrale Objekte herleiten.

2.1 Kollisionen zwischen Polyedern

Um eine effiziente Kollisionsberechnung zu ermöglichen, beschränken wir die erlaubten Körper auf Polyeder; diese lassen sich in einfacher Weise durch ihre Oberflächenstruktur, d.h. ihre Ecken, Kanten und Flächen beschreiben (*boundary representation*). Zudem können reale Körper beliebig genau durch kompakte Polyeder approximiert werden, wenn der Aufwand dabei auch recht hoch werden kann. Unser Interesse zielt nun auf die Fragestellung, wann zwei bewegte Polyeder, für die zu den vorgegebenen Zeitpunkten $k \in \mathbb{N}$

$$\mathcal{P}_i\ (k) \cap \mathcal{P}_j(k) = \emptyset$$

gilt, kollidieren:

Definition 2.1:
Zwei Polyeder $\mathcal{P}_i$ und $\mathcal{P}_j$ kollidieren während ihrer über dem Zeitintervall $(0,1)$ parametrisierten Bewegung aus ihren Konfigurationen zum Zeitpunkt k in die zum Zeitpunkt

$k+1$, falls sie in Kontakt geraten, d.h. falls gilt:

$$\begin{array}{rl} \exists t \in (0,1): & bound(\mathcal{P}_i(k+t)) \cap bound(\mathcal{P}_j(k+t)) \neq \emptyset \\ \wedge & int(\mathcal{P}_i(k+t)) \cap int(\mathcal{P}_j(k+t)) = \emptyset \end{array}$$

Wenn zwei Polyeder $\mathcal{P}_i$ und $\mathcal{P}_j$ miteinander in Kontakt kommen, ist grundsätzlich jeweils eine Fläche der beiden Polyeder an der Kontaktsituation beteiligt. Betrachtet man die Kontaktmöglichkeiten jedoch genauer, ist zu erkennen, daß bei jedem möglichen Kontakt einer der beiden folgenden Fälle

- Kante-Kante-Kontakt
- Ecke-Fläche-Kontakt

vorliegen muß, was auch in [Eck95] gezeigt wird:

Satz 2.2:
Seien $\mathcal{P}_i$ und $\mathcal{P}_j$ zwei beliebige Polyeder. $\mathcal{P}_i$ und $\mathcal{P}_j$ sind genau dann miteinander in Kontakt, wenn gilt:

1. $int(\mathcal{P}_i) \cap int(\mathcal{P}_j) = \emptyset$
2. (a) $\exists v \in V_{\mathcal{P}_i}\ (V_{\mathcal{P}_j}),\ f \in F_{\mathcal{P}_j}\ (F_{\mathcal{P}_i})$ *mit* $v \in f$
   oder

   (b) $\exists e_1 \in E_{\mathcal{P}_i},\ e_2 \in E_{\mathcal{P}_j}$ *mit* $e_1 \cap e_2 \neq \emptyset$

Durch diese Beobachtung und aufgrund der Oberflächenbeschreibung unserer polyhedral modellierten Objekte sind wir nun in der Lage, den Kollisionstest zweier Polyeder auf geometrische Schnittprobleme zurückzuführen. Wie wir diese geometrische Kollisionserkennung *exakt* durchführen können, werden wir im Anschluß an den folgenden Abschnitt, in dem die Problemstellung noch einmal spezifiziert und formalisiert wird, sehen.

2.2 Formulierung der Problemstellung

Gegeben sei nun ein Szenario, in dem sich zwei Objekte gleichzeitig bewegen. In unserem Modell geht die Bewegung eines Objektes durch Interpolation zwischen zwei Konfigurationen aufeinanderfolgender Zeitpunkte hervor. Für jedes der beiden Objekte sei seine Konfiguration zu diskreten Zeitpunkten bekannt. Mit $\mathcal{C}_i(k) = (\mathbf{o}_i^k, \mathfrak{p}_i^k)$ bezeichnen wir hierbei die Konfiguration (Position und Orientierung) des Objektes i zum Zeitpunkt $k \in \mathbb{N}_0$. Nach Voraussetzung gilt weiterhin, daß $\mathcal{C}_i(k) \in \mathbb{Z}^7$. Den Übergang von $\mathcal{C}_i(k)$ nach $\mathcal{C}_i(k+1)$ erhalten wir durch eine geeignete polynomielle Interpolation (z.B. lineare Interpolation). Für $t \in (0,1)$ gilt dann:

$$\mathcal{C}_i(k+t) = I(t)(\mathcal{C}_i(k), \mathcal{C}_i(k+1))$$

Die Frage, mit der wir uns im folgenden Kapitel beschäftigen, lautet, ob zwei polyhedrale Objekte $\mathcal{P}_i$ und $\mathcal{P}_j$ während ihrer Bewegung im Zeitintervall $(k, k+1)$ kollidieren:

$$\exists\, t \in (0,1) \text{ mit } \mathcal{P}_i(k+t) \cap \mathcal{P}_j(k+t) \neq \emptyset \text{ ?} \tag{2.1}$$

Wie wir sehen werden, ist es zur Beantwortung dieser Fragestellung nicht notwendig, die eventuell vorhandenen Kollisionszeitpunkte explizit zu berechnen. Zur Untersuchung der einzelnen Kollisionsarten wird ein zweistufiges Verfahren zur Entscheidung des Problems vorgestellt:

1. Beschreibung aller potentiellen Kollisionszeitpunkte durch eine polynomielle Gleichung der Form $f(t) = 0$. In diesem Schritt wird das ursprünglich räumliche Problem auf ein ebenes reduziert.

2. Beschränkung der in (1.) erfaßten Zeitpunkte durch polynomielle Ungleichungen vom Typ $g_i(t) > 0$ auf reale Kollisionszeitpunkte.

2.3 Exakte Kollisionserkennung

Wie wir in Satz 2.2 gesehen haben, gibt es für zwei polyhedrale Objekte folgende Kollisionstypen:

- Kante-Kante-Kollision
- Punkt-Fläche-Kollision

Um in entarteten Fällen (siehe Abb. 2.1) frei von Fallunterscheidungen und sonstigen *unangenehmen* Nebenerscheinungen zu werden, betrachten wir diese, nämlich

- Punkt-Kante-Kontakt und
- Punkt-Punkt-Kontakt,

gesondert. Ein Kollisionstest für zwei Polyeder $\mathcal{P}_i$ und $\mathcal{P}_j$ besteht demnach aus vier einzelnen, voneinander unabhängigen Tests.

In den nun folgenden Abschnitten, in denen wir die einzelnen Kollisionsarten genauer betrachten wollen, seien zwei sich bewegende Polyeder $\mathcal{P}_1$ und $\mathcal{P}_2$ sowie deren Konfigurationen zu zwei aufeinanderfolgenden Zeitpunkten k und $k+1$ gegeben (siehe Modellbeschreibung). Zwischenkonfigurationen zu einem Zeitpunkt $k+t$ $(0 < t < 1)$ errechnen wir der Einfachheit halber mittels linearer Interpolation:

$$\mathcal{C}_i(k+t) \quad = \quad \mathcal{C}_i(k) \; + t \cdot (\mathcal{C}_i(k+1) - \mathcal{C}_i(k))$$

Dadurch erreichen die später zu lösenden polynomiellen Gleichungssysteme einen niedrigeren Grad. Liegen die Zeitpunkte, zu denen Zwischenkonfigurationen durch den Programmierer vorgegeben werden, dicht genug beisammen bzw. sind die Konfigurationsänderungen in dem dazwischenliegenden Zeitintervall nicht *allzu* gravierend,

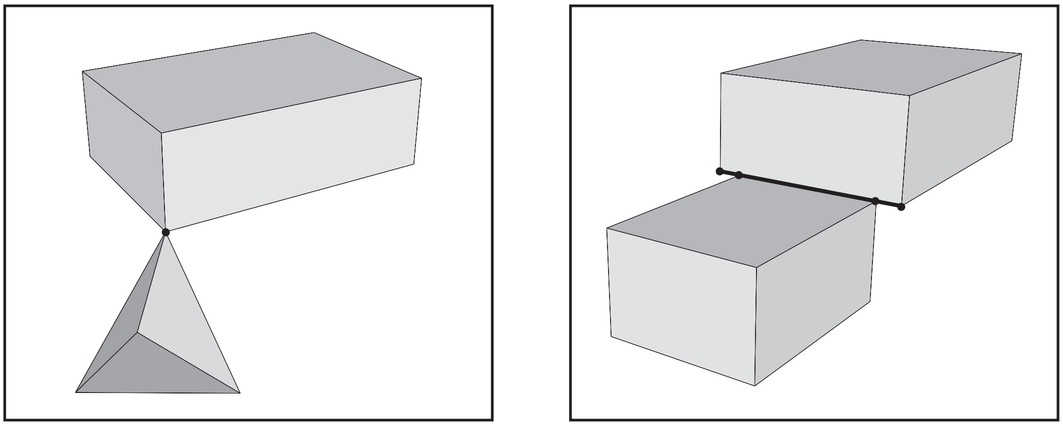

Abbildung 2.1: ZWEI ENTARTETE KONTAKTSITUATIONEN

so erscheint die Verwendung der linearen Interpolation auch durchaus sinnvoll. In einer solchen über dem Zeitparameter t linear parametrisierten Zwischenkonfiguration $\mathcal{C}_i(k+t)$ erreichen die Einträge der Rotationsmatrix $\mathbf{R_i}(t)$ den Grad 2; die Komponenten des aktuellen Positionsvektors $\mathbf{o_i}(t)$ besitzen den Grad 1.

2.3.1 Kante-Kante-Kollision

Sei $l_{\mathbf{ab}}(t)$ die Kante des bewegten Polyeders $\mathcal{P}_1$ mit den Endpunkten $\mathbf{a}(t)$ und $\mathbf{b}(t)$ und $l_{\mathbf{cd}}(t)$ die Kante des bewegten Polyeders $\mathcal{P}_2$ mit den Endpunkten $\mathbf{c}(t)$ und $\mathbf{d}(t)$. Für die bewegten Ecken unserer beiden Polyeder $\mathcal{P}_1$ und $\mathcal{P}_2$ gilt nach Formel 1.9 bzw. 1.10:

$$\begin{aligned}
\mathbf{a}(t) &= \mathbf{R_1}(t)\cdot\mathbf{a}+\mathbf{o_1}(t) = \frac{\mathbf{R'_1}(t)}{N_1(t)}\cdot\mathbf{a}+\mathbf{o_1}(t)\\
\mathbf{b}(t) &= \mathbf{R_1}(t)\cdot\mathbf{b}+\mathbf{o_1}(t) = \frac{\mathbf{R'_1}(t)}{N_1(t)}\cdot\mathbf{b}+\mathbf{o_1}(t)\\
\mathbf{c}(t) &= \mathbf{R_2}(t)\cdot\mathbf{c}+\mathbf{o_2}(t) = \frac{\mathbf{R'_2}(t)}{N_2(t)}\cdot\mathbf{c}+\mathbf{o_2}(t)\\
\mathbf{d}(t) &= \mathbf{R_2}(t)\cdot\mathbf{d}+\mathbf{o_2}(t) = \frac{\mathbf{R'_2}(t)}{N_2(t)}\cdot\mathbf{d}+\mathbf{o_2}(t)
\end{aligned}$$

Wir leiten nun Kollisionsbedingungen für zwei solche Liniensegmente während ihrer Bewegung in einem Zeitintervall her. Sei die Bewegung der Polyeder im Zeitintervall $(k,k+1)$ mittels $t\in(0,1)$ parametrisiert.

2.3.1.1 Potentielle Kollisionen von Liniensegmenten

Eine notwendige Bedingung dafür, daß sich zwei Liniensegmente schneiden, besteht darin, daß sich die korrespondierenden Geraden schneiden. Dazu müssen die vier Endpunkte der Liniensegmente gemeinsam in einer Ebene liegen; diese Bedingung läßt sich

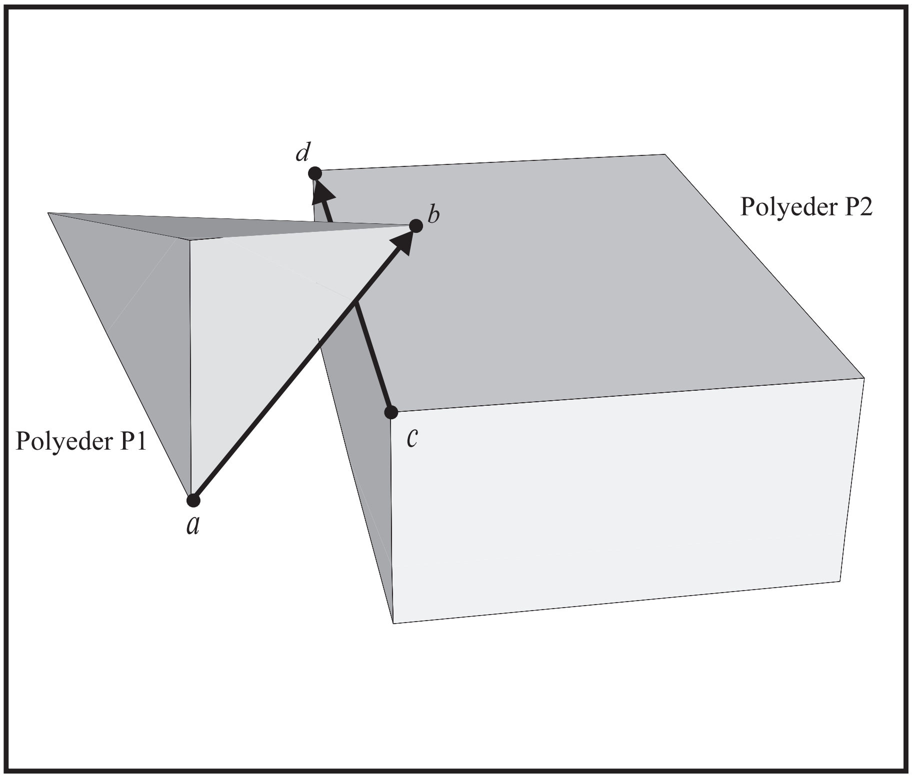

Abbildung 2.2: KANTE-KANTE-KOLLISION

gemäß Satz A.1 durch folgende Gleichung fassen

$$
\begin{aligned}
& det \begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{a}(t) & \mathbf{b}(t) & \mathbf{c}(t) & \mathbf{d}(t) \end{bmatrix} &= 0 \\
\Longleftrightarrow\quad & (\mathbf{d}-\mathbf{c})^T \mathbf{R_2}(t)^T \mathbf{R_1}(t)(\mathbf{a}\times\mathbf{b}) + (\mathbf{c}\times\mathbf{d})^T \mathbf{R_2}(t)^T \mathbf{R_1}(t)(\mathbf{b}-\mathbf{a}) + \\
& (\mathbf{o_1}(t)-\mathbf{o_2}(t))^T(\mathbf{R_1}(t)(\mathbf{b}-\mathbf{a})\times \mathbf{R_2}(t)(\mathbf{d}-\mathbf{c})) &= 0 \\
\Longleftrightarrow\quad & (\mathbf{d}-\mathbf{c})^T \frac{\mathbf{R'_2}(t)}{N_2(t)}^T \frac{\mathbf{R'_1}(t)}{N_1(t)}(\mathbf{a}\times\mathbf{b}) + (\mathbf{c}\times\mathbf{d})^T \frac{\mathbf{R'_2}(t)}{N_2(t)}^T \frac{\mathbf{R'_1}(t)}{N_1(t)}(\mathbf{b}-\mathbf{a}) + \\
& (\mathbf{o_1}(t)-\mathbf{o_2}(t))^T(\frac{\mathbf{R'_1}(t)}{N_1(t)}(\mathbf{b}-\mathbf{a})\times \frac{\mathbf{R'_2}(t)}{N_2(t)}(\mathbf{d}-\mathbf{c})) &= 0 \\
\Longleftrightarrow\quad & (\mathbf{d}-\mathbf{c})^T \mathbf{R'_2}(t)^T \mathbf{R'_1}(t)(\mathbf{a}\times\mathbf{b}) + (\mathbf{c}\times\mathbf{d})^T \mathbf{R'_2}(t)^T \mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a}) + \\
& \underbrace{(\mathbf{o_1}(t)-\mathbf{o_2}(t))^T}_{Grad\ 1} \underbrace{(\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a}))}_{Grad\ 2} \times \underbrace{(\mathbf{R'_2}(t)(\mathbf{d}-\mathbf{c}))}_{Grad\ 2} &= 0
\end{aligned}
$$

deren Expansion zu einer polynomiellen Gleichung der Art $f(t) = 0$ vom Grad 5 führt. Aufgrund der Abhängigkeit der Laufzeit der Verfahren zur Kollisionsberechnung von der Länge der auftretenden Zahlen – eine untere Schranke für die Multiplikation zweier L-Bit-Zahlen ist $L\ log\ L$ – werden wir die maximalen Koeffizientenlängen L_{max} der zu berechnenden Polynome unter Berücksichtigung der Eingabelänge L_{Ein} betrachten.

$$\begin{aligned} L_{max} &= length\Big((\mathbf{d}-\mathbf{c})^T \mathbf{R}_2'(t)^T \mathbf{R}_1'(t)(\mathbf{a}\times\mathbf{b}) + (\mathbf{c}\times\mathbf{d})^T \mathbf{R}_2'(t)^T \mathbf{R}_1'(t)(\mathbf{b}-\mathbf{a}) + \\ &\qquad (\mathbf{o_1}(t)-\mathbf{o_2}(t))^T (\mathbf{R}_1'(t)(\mathbf{b}-\mathbf{a})) \times (\mathbf{R}_2'(t)(\mathbf{d}-\mathbf{c}))\Big) \\ &= max\Big\{length((\mathbf{d}-\mathbf{c})^T \mathbf{R}_2'(t)^T \mathbf{R}_1'(t)(\mathbf{a}\times\mathbf{b})), \\ &\qquad length((\mathbf{c}\times\mathbf{d})^T \mathbf{R}_2'(t)^T \mathbf{R}_1'(t)(\mathbf{b}-\mathbf{a})), \\ &\qquad length((\mathbf{o_1}(t)-\mathbf{o_2}(t))^T (\mathbf{R}_1'(t)(\mathbf{b}-\mathbf{a})) \times (\mathbf{R}_2'(t)(\mathbf{d}-\mathbf{c})))\Big\} + c \\ &= max\{\ 7\,L_{Ein}\ ,\ 7\,L_{Ein}\ ,\ 7\,L_{Ein}\ \} + c \\ &= 7\,L_{Ein} + c \end{aligned}$$

Expandiert man obige Gleichung vollständig, entsteht ein Ausdruck aus zahlreichen Summanden, die jeweils maximal die Länge $7\,L_{Ein}$ besitzen. Durch deren Summation wächst die Länge des gesamten Ausdrucks um eine additive Konstante c, die logarithmisch in der Anzahl der auftretenden Summanden ist.

Die Nullstellen des Polynoms $f(t)$ im Intervall $(0,1)$ beschreiben die Zeitpunkte, zu denen die zu den Liniensegmenten $l_{\mathbf{ab}}(t)$ und $l_{\mathbf{cd}}(t)$ gehörigen Geraden $L_{\mathbf{ab}}(t)$ und $L_{\mathbf{cd}}(t)$ während ihrer über dem Zeitintervall $(0,1)$ parametrisierten Bewegung kollidieren bzw. parallel verlaufen – sich also gemeinsam in einer Ebene befinden. Da in der Regel nicht alle Kollisionszeitpunkte der Geraden auch tatsächlich Kollisionszeitpunkte der Liniensegmente sind, müssen wir diese durch Angabe geeigneter Bedingungen im Intervall $(0,1)$ noch weiter einschränken.

2.3.1.2 Einschränkung der potentiellen auf reale Kollisionen

Durch die Beschreibung der potentiellen Kollisionszeitpunkte der zu Geraden ausgedehnten Liniensegmente mittels einer polynomiellen Gleichung vom Typ $f(t) = 0$ haben wir das räumliche Ausgangsproblem auf ein ebenes reduziert: wir müssen uns nun mit der Frage beschäftigen, wann zwei Liniensegmente, die zu einem Zeitpunkt $t_{pot-Col} \in (0,1)$ gemeinsam in einer beliebigen Ebene liegen – also $f(t_{pot-Col}) = 0$ – sich tatsächlich schneiden. Hierbei schließen wir den Fall, daß die beiden Liniensegmente in zwei Endpunkten übereinstimmen, aus.

Eine mögliche Vorgehensweise sieht die Projektion der vorhandenen, *ebenen* Szene in eine *geeignete* Standardebene vor. Die Projektion darf die Szene jedoch nicht entarten lassen, d.h. Normalenvektor der Standardebene und Normalenvektor der Szenenebene dürfen nicht senkrecht aufeinander stehen: in diesem Fall würde die Projektion zu einem Liniensegment (im ungünstigsten Fall sogar zu einem Punkt) degenerieren. Bezeichnen wir mit $\overline{\mathbf{a}(t)}, \overline{\mathbf{b}(t)}, \overline{\mathbf{c}(t)}$ und $\overline{\mathbf{d}(t)}$ die in die Standardebene mit dem Normalenvektor $\mathbf{e}$ projizierten Endpunkte der Liniensegmente $l_{\mathbf{ab}}(t)$ und $l_{\mathbf{cd}}(t)$, so gilt:

$$\overline{l_{\mathbf{ab}}(t)} \cap \overline{l_{\mathbf{cd}}(t)} \neq \emptyset \quad \Rightarrow \quad l_{\mathbf{ab}}(t) \cap l_{\mathbf{cd}}(t) \neq \emptyset$$

Zwei Liniensegmente $\overline{l_{\mathbf{ab}}(t)}$ und $\overline{l_{\mathbf{cd}}(t)}$ schneiden sich in der Ebene, wenn gilt:

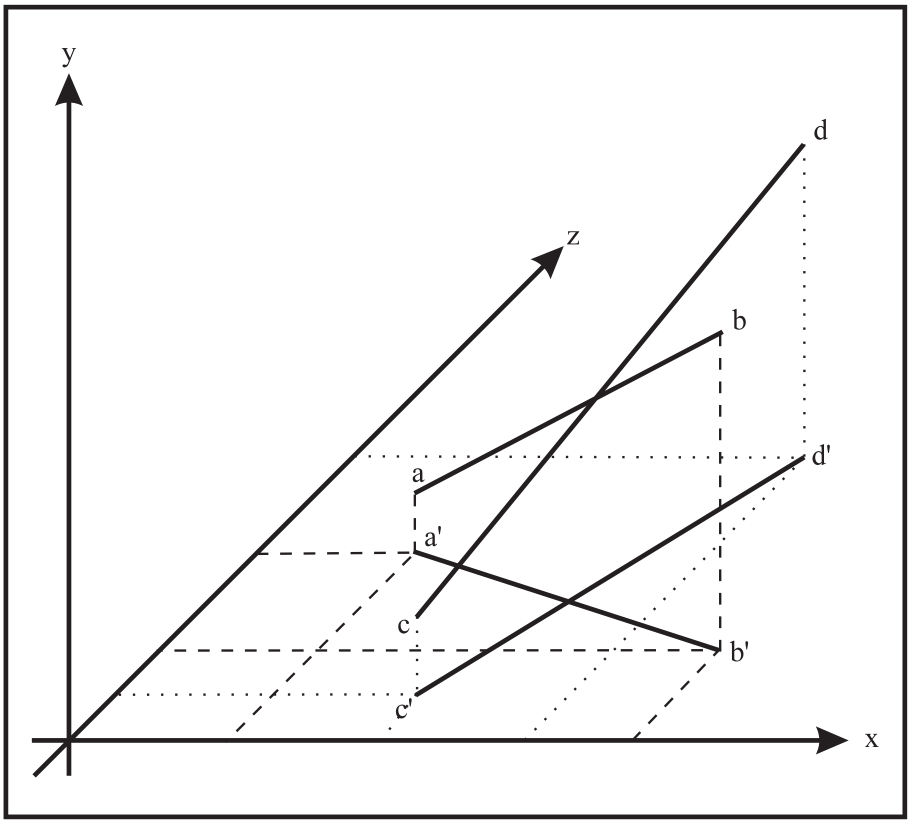

Abbildung 2.3: GEEIGNETE PROJEKTION DER SZENE IN EINE STANDARDEBENE

1. $\overline{\mathbf{a}(t)}$ und $\overline{\mathbf{b}(t)}$ liegen bzgl. der Geraden durch die Punkte $\overline{\mathbf{c}(t)}$ und $\overline{\mathbf{d}(t)}$ in verschiedenen Halbebenen und

2. $\overline{\mathbf{c}(t)}$ und $\overline{\mathbf{d}(t)}$ liegen bzgl. der Geraden durch die Punkte $\overline{\mathbf{a}(t)}$ und $\overline{\mathbf{b}(t)}$ in verschiedenen Halbebenen

Dies führt, wie wir in Kapitel 1 bereits gesehen haben, zu folgenden Bedingungen:

$$\begin{aligned} \mathbf{e}^T\Big((\mathbf{b}(t)-\mathbf{a}(t))\times(\mathbf{c}(t)-\mathbf{a}(t))\Big) &> 0 \\ \wedge\ \mathbf{e}^T\Big((\mathbf{b}(t)-\mathbf{a}(t))\times(\mathbf{d}(t)-\mathbf{a}(t))\Big) &< 0 \\ \wedge\ \mathbf{e}^T\Big((\mathbf{d}(t)-\mathbf{c}(t))\times(\mathbf{b}(t)-\mathbf{c}(t))\Big) &> 0 \\ \wedge\ \mathbf{e}^T\Big((\mathbf{d}(t)-\mathbf{c}(t))\times(\mathbf{a}(t)-\mathbf{c}(t))\Big) &< 0 \end{aligned}$$

Wegen der implizit zugrundegelegten Richtung der Kanten müssen - sollten diese Bedingungen nicht erfüllt sein - auch die hierzu symmetrischen Bedingungen

$$\begin{aligned} \mathbf{e}^T\Big((\mathbf{b}(t)-\mathbf{a}(t))\times(\mathbf{c}(t)-\mathbf{a}(t)\Big) &< 0 \\ \wedge\ \mathbf{e}^T\Big((\mathbf{b}(t)-\mathbf{a}(t))\times(\mathbf{d}(t)-\mathbf{a}(t))\Big) &> 0 \end{aligned}$$

$$\begin{aligned} \wedge\, \mathbf{e}^T\Big((\mathbf{d}(t)-\mathbf{c}(t))\times(\mathbf{b}(t)-\mathbf{c}(t))\Big) &< 0\\ \wedge\, \mathbf{e}^T\Big((\mathbf{d}(t)-\mathbf{c}(t))\times(\mathbf{a}(t)-\mathbf{c}(t))\Big) &> 0 \end{aligned}$$

überprüft werden. Exemplarisch möchten wir nun die erste Ungleichung expandieren:

$$\begin{aligned}
& \mathbf{e}^T\Big((\mathbf{b}(t)-\mathbf{a}(t))\times(\mathbf{c}(t)-\mathbf{a}(t))\Big) > 0\\
\Longleftrightarrow\quad & \mathbf{e}^T\Big((\mathbf{a}(t)\times\mathbf{b}(t))-\mathbf{c}(t)\times(\mathbf{b}(t)-\mathbf{a}(t))\Big) > 0\\
\Longleftrightarrow\quad & \mathbf{e}^T\Big((\mathbf{R_1}(t)\mathbf{a}+\mathbf{o_1}(t))\times(\mathbf{R_1}(t)\mathbf{b}+\mathbf{o_1}(t))-\\
& (\mathbf{R_2}(t)\mathbf{c}+\mathbf{o_2}(t))\times((\mathbf{R_1}(t)\mathbf{b}+\mathbf{o_1}(t))-(\mathbf{R_1}(t)\mathbf{a}+\mathbf{o_1}(t)))\Big) > 0\\
\Longleftrightarrow\quad & \mathbf{e}^T\Big((\mathbf{R_1}(t)\mathbf{a})\times(\mathbf{R_1}(t)\mathbf{b})+(\mathbf{R_1}(t)\mathbf{a}\times\mathbf{o_1}(t)+\\
& \mathbf{o_1}(t)\times(\mathbf{R_1}(t)\mathbf{b})-(\mathbf{R_2}(t)\mathbf{c}+\mathbf{o_2}(t))\times(\mathbf{R_1}(t)(\mathbf{b}-\mathbf{a}))\Big) > 0\\
\Longleftrightarrow\quad & \mathbf{e}^T\Big(\mathbf{R_1}(t)(\mathbf{a}\times\mathbf{b})+\mathbf{o_1}(t)\times(\mathbf{R_1}(t)(\mathbf{b}-\mathbf{a}))+\\
& (\mathbf{R_1}(t)(\mathbf{b}-\mathbf{a}))\times(\mathbf{R_2}(t)\mathbf{c}+\mathbf{o_2}(t))\Big) > 0\\
\Longleftrightarrow\quad & \mathbf{e}^T\Big(\frac{\mathbf{R'_1}(t)}{N_1(t)}(\mathbf{a}\times\mathbf{b})+\mathbf{o_1}(t)\times(\frac{\mathbf{R'_1}(t)}{N_1(t)}(\mathbf{b}-\mathbf{a}))+\\
& (\frac{\mathbf{R'_1}(t)}{N_1(t)}(\mathbf{b}-\mathbf{a}))\times(\frac{\mathbf{R'_2}(t)}{N_2(t)}\mathbf{c}+\mathbf{o_2}(t))\Big) > 0\\
\Longleftrightarrow\quad & \mathbf{e}^T\Big((\mathbf{R'_1}(t)(\mathbf{a}\times\mathbf{b}))N_2(t)+(\mathbf{o_1}(t)\times(\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a})))N_2(t)+\\
& (\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a}))\times(\mathbf{R'_2}(t)\mathbf{c})+((\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a}))\times\mathbf{o_2}(t))N_2(t)\Big) > 0
\end{aligned}$$

Man erhält eine polynomielle Ungleichung mit folgenden Eigenschaften:

- Grad 5 im Zeitparameter t:
  Bei der angenommenen linearen Interpolation zwischen den vorgegebenen Konfigurationen besitzen die Einträge der Matrizen $\mathbf{R'_i}(t)$ und ihre Nenner $N_i(t), i \in \{1,2\}$, den Grad 2, die Vektorkomponenten von $\mathbf{o_i}(t), i \in \{1,2\}$ den Grad 1. Für den Grad Deg_{max} des gesamten polynomiellen Ausdrucks gilt dann:

$$\begin{aligned}
Deg_{max} &= grad\Big((\mathbf{R'_1}(t)(\mathbf{a}\times\mathbf{b}))N_2(t)+(\mathbf{o_1}(t)\times(\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a})))N_2(t)+\\
&\qquad (\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a}))\times(\mathbf{R'_2}(t)\mathbf{c})+((\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a}))\times\mathbf{o_2}(t))N_2(t)\Big)\\
&= max\Big\{grad((\mathbf{R'_1}(t)(\mathbf{a}\times\mathbf{b}))N_2(t)),\\
&\qquad grad((\mathbf{o_1}(t)\times(\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a})))N_2(t)),\\
&\qquad grad((\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a}))\times(\mathbf{R'_2}(t)\mathbf{c})),\\
&\qquad grad(((\mathbf{R'_1}(t)(\mathbf{b}-\mathbf{a}))\times\mathbf{o_2}(t))N_2(t))\Big\}\\
&= max\{4,5,4,5\}\\
&= 5
\end{aligned}$$

- sechsfache Länge der Koeffizienten gegenüber der Eingabe:
  Die zeitabhängigen Orientierungsmatrizen und deren Nenner besitzen - wie sich anhand von Gleichung 1.4 leicht nachvollziehen läßt - zweifache Koeffizientenlänge gegenüber der Eingabelänge L_{Ein}; die zeitabhängigen Positionsvektoren dagegen nur einfache. Für die maximale Koeffizientenlänge L_{max} der polynomiellen Ungleichung erhält man daher:

$$\begin{aligned}
L_{max} &= length\Big((\mathbf{R_1'}(t)(\mathbf{a}\times\mathbf{b}))N_2(t) + (\mathbf{o_1}(t)\times(\mathbf{R_1'}(t)(\mathbf{b}-\mathbf{a})))N_2(t) + \\
&\qquad (\mathbf{R_1'}(t)(\mathbf{b}-\mathbf{a}))\times(\mathbf{R_2'}(t)\mathbf{c}) + ((\mathbf{R_1'}(t)(\mathbf{b}-\mathbf{a}))\times\mathbf{o_2}(t))N_2(t)\Big) \\
&= max\Big\{length((\mathbf{R_1'}(t)(\mathbf{a}\times\mathbf{b}))N_2(t)), \\
&\qquad length((\mathbf{o_1}(t)\times(\mathbf{R_1'}(t)(\mathbf{b}-\mathbf{a})))N_2(t)), \\
&\qquad length((\mathbf{R_1'}(t)(\mathbf{b}-\mathbf{a}))\times(\mathbf{R_2'}(t)\mathbf{c})), \\
&\qquad length(((\mathbf{R_1'}(t)(\mathbf{b}-\mathbf{a}))\times\mathbf{o_2}(t))N_2(t))\Big\} + c \\
&= max\{\ 5\,L_{Ein}\ ,\ 6\,L_{Ein}\ ,\ 6\,L_{Ein}\ ,\ 6\,L_{Ein}\ \} + c \\
&= 6\,L_{Ein} + c
\end{aligned}$$

Expandiert man obige Gleichung vollständig, entsteht ein Ausdruck aus zahlreichen Summanden, die jeweils maximal die Länge $6\,L_{Ein}$ besitzen. Durch deren Summation wächst die Länge des gesamten Ausdrucks um eine additive Konstante c, die logarithmisch in der Anzahl der auftretenden Summanden ist.

Die übrigen Bedingungen lassen sich in analoger Weise zu Ungleichungen obigen Typs umformen. Die Tatsache also, daß sich zwei bewegte Liniensegmente, die zu einem Zeitpunkt t in einer Ebene liegen, tatsächlich schneiden, läßt sich demnach durch vier polynomielle Ungleichungen beschreiben.

Damit haben wir das Ausgangsproblem - nämlich zu entscheiden, ob zwei bewegte Liniensegmente miteinander kollidieren - auf das Problem reduziert, von einem polynomiellen Gleichungssystem der Form

$$\begin{array}{rrcll}
 & f(t) & = & 0 & \big\}\ \text{potentielle Kollisionszeitpunkte} \\
\wedge & g_1(t) & > & 0 & \\
\wedge & g_2(t) & > & 0 & \\
\wedge & g_3(t) & > & 0 & \\
\wedge & g_4(t) & > & 0 &
\end{array}\left.\vphantom{\begin{array}{c}0\\0\\0\\0\end{array}}\right\}\ \begin{array}{l}\text{Bedingungen für Schnitt zweier}\\ \text{Liniensegmente in der Ebene}\end{array}$$

entscheiden zu müssen, ob es im Intervall $(0, 1)$ Lösungen besitzt.

Bei eingehender Untersuchung der verbleibenden Kollisionstypen wird sich - wie wir in den folgenden Abschnitten noch sehen werden - ebenfalls ein solches polynomielles

Gleichungssystem ergeben. Wenn wir also Kollisionserkennung *exakt* betreiben wollen, wird es notwendig sein, Verfahren zur *exakten* Beantwortung der folgenden Frage anzugeben:

$$\boxed{\#\{t \in (0,1) \,|\, f(t) = 0 \;\wedge\; g_i(t) > 0 \;\forall\, 1 \leq i \leq l\} \;>\; 0\,?}$$

Doch dazu mehr in Kapitel 3.

2.3.2 Punkt-Kante-Kollision

Im nun folgenden Abschnitt untersuchen wir, wann ein bewegter Punkt $\mathbf{a}(t)$ eines Polyeders $\mathcal{P}_1$ auf ein bewegtes Liniensegment $l_{\mathbf{cd}}(t)$ mit den Endpunkten $\mathbf{c}(t)$ und $\mathbf{d}(t)$ eines Polyeders $\mathcal{P}_2$ trifft.

2.3.2.1 Potentielle Kollisionen von Punkt-Kante

Wie im Fall der Kante-Kante-Kollision expandieren wir das Liniensegment $l_{\mathbf{cd}}(t)$ zunächst zu einer Geraden $L_{\mathbf{cd}}(t) = \mathbf{c}(t) + \lambda(\mathbf{d}(t) - \mathbf{c}(t))$, $\lambda \in \mathbb{R}$, durch die Punkte $\mathbf{c}(t)$ und $\mathbf{d}(t)$ und fragen uns, wann der bewegte Punkt $\mathbf{a}(t)$ auf die bewegte Gerade trifft. Dies ist dann der Fall, wenn der Abstand des Punktes zur Geraden gerade null ist:

$$\begin{array}{rrcl}
 & dist(\mathbf{a}(t), L_{\mathbf{cd}}(t)) & = & 0 \\
\Longleftrightarrow & \frac{|(\mathbf{d}(t)-\mathbf{c}(t))\times(\mathbf{a}(t)-\mathbf{c}(t))|}{|(\mathbf{d}(t)-\mathbf{c}(t))|} & = & 0 \\
\Longleftrightarrow & |(\mathbf{d}(t)-\mathbf{c}(t))\times(\mathbf{a}(t)-\mathbf{c}(t))| & = & 0 \\
\Longleftrightarrow & |\mathbf{d}(t)\times\mathbf{a}(t)-\mathbf{c}(t)\times\mathbf{a}(t)-\mathbf{d}(t)\times\mathbf{c}(t)| & = & 0 \\
\Longleftrightarrow & |(\mathbf{d}(t)-\mathbf{c}(t))\times\mathbf{a}(t)+\mathbf{c}(t)\times\mathbf{d}(t)| & = & 0
\end{array}$$

Sei $\mathbf{u}(t) = (\mathbf{d}(t) - \mathbf{c}(t)) \times \mathbf{a}(t) + \mathbf{c}(t) \times \mathbf{d}(t)$. Es gilt:

$$|\mathbf{u}(t)| = 0 \quad \Longleftrightarrow \quad u_x(t)^2 + u_y(t)^2 + u_z(t)^2 = 0 \tag{2.2}$$

$$\begin{array}{rcl}
\mathbf{u}(t) & = & (\mathbf{d}(t)-\mathbf{c}(t))\times\mathbf{a}(t)+\mathbf{c}(t)\times\mathbf{d}(t) \\
 & = & (\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{d}+\mathbf{o_2}(t)-\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{c}-\mathbf{o_2}(t)\times(\frac{\mathbf{R_1'}(t)}{N_1(t)}\mathbf{a}+\mathbf{o_1}(t))+ \\
 & & (\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{c}+\mathbf{o_2}(t))\times(\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{d}+\mathbf{o_2}(t)) \\
 & = & (\frac{\mathbf{R_2'}(t)}{N_2(t)}(\mathbf{d}-\mathbf{c}))\times(\frac{\mathbf{R_1'}(t)}{N_1(t)}\mathbf{a})+(\frac{\mathbf{R_2'}(t)}{N_2(t)}(\mathbf{d}-\mathbf{c}))\times\mathbf{o_1}(t)+ \\
 & & (\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{c})\times(\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{d})+(\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{c})\times\mathbf{o_2}(t)+\mathbf{o_2}(t)\times(\frac{\mathbf{R_2'}(t)}{N_2(t)}(\mathbf{d}-\mathbf{c}))
\end{array}$$

$$
\begin{aligned}
&= (\tfrac{\mathbf{R_2'}(t)}{N_2(t)}(\mathbf{d}-\mathbf{c})) \times (\tfrac{\mathbf{R_1'}(t)}{N_1(t)}\mathbf{a}) + (\tfrac{\mathbf{R_2'}(t)}{N_2(t)}(\mathbf{d}-\mathbf{c})) \times \mathbf{o_1}(t) + \tfrac{\mathbf{R_2'}(t)}{N_2(t)}(\mathbf{c}\times\mathbf{d}) + \\
&\quad \mathbf{o_2}(t) \times (\tfrac{\mathbf{R_2'}(t)}{N_2(t)}(t)(\mathbf{d}-\mathbf{c})) \\
&= (\mathbf{R_2'}(t)(\mathbf{d}-\mathbf{c})) \times (\mathbf{R_1'}(t)\mathbf{a}) + ((\mathbf{R_2'}(t)(\mathbf{d}-\mathbf{c})) \times \mathbf{o_1}(t))N_1(t) + \\
&\quad (\mathbf{R_2'}(t)(c \times d))N_1(t) + (\mathbf{o_2}(t) \times (\mathbf{R_2'}(t)(\mathbf{d}-\mathbf{c})))N_1(t)
\end{aligned}
$$

Die einzelnen Komponenten des Vektors $\mathbf{u}$ bestehen aus Polynomen im Zeitparameter t vom Grad 5, deren Koeffizienten sechsfache Länge gegenüber der Eingabe aufweisen. Wollte man nun die Länge dieses Vektors gemäß 2.2 auf 0 überprüfen, erhielte man eine polynomielle Gleichung vom Grad 10 – dies entspräche dem doppelten Grad aller bisher erhaltenen Polynome.

Eine alternative Möglichkeit um zu entscheiden, ob die Länge des Vektors $\mathbf{u}$ zu einem Zeitpunkt $t \in (0,1)$ null wird, eröffnet uns der folgende Satz aus der Algebra:

Satz 2.3:
Seien f und g zwei Polynome in einer Veränderlichen. Dann gilt:

$$\exists\, t_0 \in \mathbb{C} \; mit \; f(t_0) = 0 \;\wedge\; g(t_0) = 0 \quad \Longleftrightarrow \quad ggt(f,g) \neq 1$$

Angewendet auf den Vektor $\mathbf{u}$ gilt dann:

$$\exists\, t_0 \in \mathbb{C} \text{ mit } |\mathbf{u}(t_0)| = 0 \Leftrightarrow \exists\, t_0 \in \mathbb{C} \text{ mit } \mathbf{u}(t_0) = \vec{0} \Leftrightarrow ggt(u_x(t), u_y(t), u_z(t)) \neq 1$$

Die Auswertung des ggT dieser 3 Vektorkomponenten liefert auf jeden Fall eine polynomielle Hauptbedingung in t vom Grad ≤ 5. Eine zusätzliche Schwierigkeit resultiert hierbei jedoch aus der Tatsache, daß wir der Exaktheit wegen nur mit ganzzahligen Ausdrücken operieren, die Berechnung des ggT mittels des Euklidischen Algorithmus allerdings die Division erfordert. Um dieses Problem umgehen zu können, müssen wir einen speziellen Algorithmus implementieren, der den ggT berechnet, ohne den Ring der ganzen Zahlen $\mathbb{Z}$ zu verlassen. Genaueres hierzu in Kapitel 4 im Abschnitt über die implementierte Polynom-Klasse.

2.3.2.2 Einschränkung der potentiellen auf reale Kollisionen

Durch die Lösungen $t \in (0,1)$ der polynomiellen Gleichung $f(t) = 0$, die wir im vorigen Abschnitt hergeleitet haben, können wir die Zeitpunkte beschreiben, zu denen der bewegte Punkt $\mathbf{a}$ auf der bewegten Gerade $L_{\mathbf{cd}}$ liegt. Wir müssen nun Bedingungen herleiten, die die Lage des Punktes auf das eigentliche Liniensegment $l_{\mathbf{cd}}$ einschränken.

Lemma 2.4:
Gegeben sei ein Liniensegment $l_{\mathbf{cd}}$ mit den Endpunkten $\mathbf{c}$ und $\mathbf{d}$, sowie eine Gerade $L_{\mathbf{cd}}$ durch die Punkte $\mathbf{c}$, $\mathbf{d} \in \mathbb{R}^3$. Für einen Punkt $\mathbf{a} \in \mathbb{R}^3$ gilt:

$$Falls\; \mathbf{a} \in L_{\mathbf{cd}} \; und\; falls\; \exists i \in \{x,y,z\} \; mit\; c_i < a_i < d_i \Longrightarrow \mathbf{a} \in l_{\mathbf{cd}}.$$

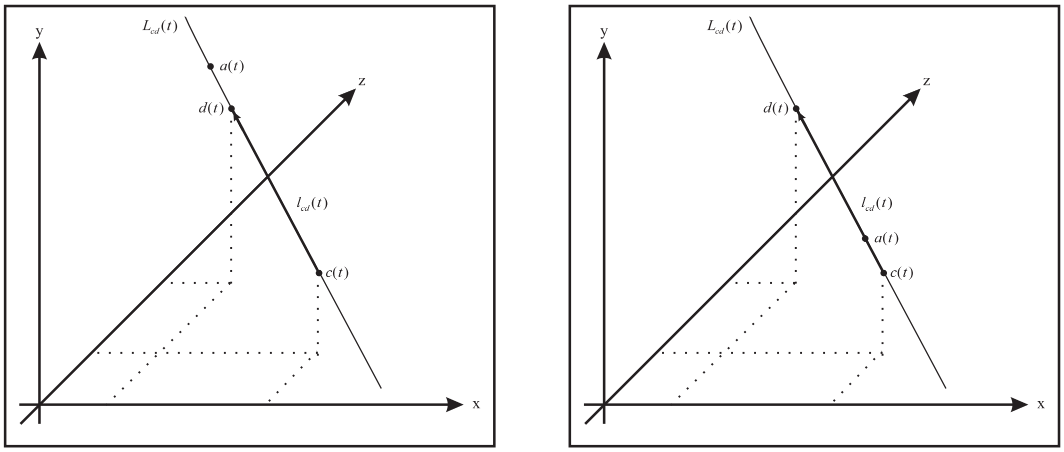

Abbildung 2.4: LAGE PUNKT-LINIENSEGMENT (GERADE)

Damit der bewegte Punkt $\mathbf{a}(t)$ nun auch tatsächlich mit dem bewegten Liniensegment $l_{\mathbf{cd}}(t)$ kollidiert, sind neben der Hauptbedingung $f(t) = 0$ gemäß obigen Lemmas folgende Nebenbedingungen notwendig:

$$\begin{array}{rcl} c_x(t) < a_x(t) < d_x(t) & \vee & c_x(t) > a_x(t) > d_x(t) \\ \vee\; c_y(t) < a_y(t) < d_y(t) & \vee & c_y(t) > a_y(t) > d_y(t) \\ \vee\; c_z(t) < a_z(t) < d_z(t) & \vee & c_z(t) > a_z(t) > d_z(t) \end{array}$$

Im folgenden werden wir nun eine solche Ungleichungskette, z.B. $\mathbf{e}^T\,\mathbf{c}(t) < \mathbf{e}^T\,\mathbf{a}(t) < \mathbf{e}^T\,\mathbf{d}(t)$ mit einem geeigneten Standardeinheitsvektor $\mathbf{e}$, genauer untersuchen; dazu zerlegen wir sie in ihre ursprünglichen Ungleichungen, die wir getrennt expandieren wollen. Für den ersten Teil der Ungleichungskette

$$\begin{array}{rrcl} & \mathbf{e}^T\,\mathbf{c}(t) & < & \mathbf{e}^T\,\mathbf{a}(t) \\ \Longleftrightarrow & \mathbf{e}^T\Big(\mathbf{a}(t) - \mathbf{c}(t)\Big) & > & 0 \\ \Longleftrightarrow & \mathbf{e}^T\Big(\big(\frac{\mathbf{R}'_\mathbf{1}(t)}{N_1(t)}\mathbf{a} + \mathbf{o_1}(t)\big) - \big(\frac{\mathbf{R}'_\mathbf{2}(t)}{N_2(t)}\mathbf{c} + \mathbf{o_2}(t)\big)\Big) & > & 0 \\ \Longleftrightarrow & \mathbf{e}^T\Big((\mathbf{R}'_\mathbf{1}(t)\mathbf{a})N_2(t) + \mathbf{o_1}(t)N_1(t)N_2(t) - & & \\ & (\mathbf{R}'_\mathbf{2}(t)\mathbf{c})N_1(t) - \mathbf{o_2}(t)N_1(t)N_2(t)\Big) & > & 0 \\ \Longleftrightarrow & \mathbf{e}^T\Big((\mathbf{R}'_\mathbf{1}(t)\mathbf{a})N_2(t) - (\mathbf{R}'_\mathbf{2}(t)\mathbf{c})N_1(t) + & & \\ & (\mathbf{o_1}(t) - \mathbf{o_2}(t))N_1(t)N_2(t)\Big) & > & 0 \end{array}$$

ergibt sich eine polynomielle Nebenbedingung der Art $g(t) > 0$ mit Grad 5, deren Koeffizienten fünffache Länge gegenüber den Eingabedaten besitzen. Durch analoge

Vorgehensweise erhält man für den zweiten Teil der Ungleichungskette:

$$\begin{aligned} & \mathbf{e}^T \mathbf{a}(t) && < && \mathbf{e}^T \mathbf{d}(t) \\ \iff \quad & \mathbf{e}^T \Big((\mathbf{R_2'}(t)\mathbf{d}) N_1(t) - (\mathbf{R_1'}(t)\mathbf{a}) N_2(t) + \\ & (\mathbf{o_2}(t) - \mathbf{o_1}(t)) N_1(t) N_2(t) \Big) && > && 0 \end{aligned}$$

Die Frage also, ob ein Punkt $\mathbf{a}$ und ein Liniensegment $l_{\mathbf{ab}}$ während ihrer gleichzeitigen Bewegung kollidieren, läßt sich formal durch ein polynomielles Gleichungssystem der Art

$$\begin{array}{rrcll} & f(t) & = & 0 & \big\} \text{ potentielle Kollisionszeitpunkte} \\ \wedge & g_1(t) & > & 0 & \multirow{2}{*}{$\Big\}$ Bedingungen für Schnitt eines Liniensegmentes und eines Punktes in der Ebene} \\ \wedge & g_2(t) & > & 0 & \end{array}$$

ausdrücken, von dem es zu entscheiden gilt, ob es für ein $t \in (0,1)$ eine Lösung besitzt.

2.3.3 Punkt-Fläche-Kollision

Sei $\mathbf{a}(t)$ eine Ecke des sich bewegenden Polyeders $\mathcal{P}_1$ und $f_{\mathbf{v_{i_1}} \dots \mathbf{v_{i_s}}}(t)$ eine Fläche des sich bewegenden Polyeders $\mathcal{P}_2$, die durch die in Kapitel 1 vereinbarte Reihenfolge ihrer Eckpunkte $\mathbf{v_{i_1}} \dots \mathbf{v_{i_s}}$ beschrieben werde.

2.3.3.1 Potentielle Kollisionen von Punkt-Fläche

Zur Beschreibung der potentiellen Kollisionszeitpunkte einer bewegten Ecke und einer bewegten Fläche dehnen wir die Fläche zunächst zu einer Ebene $H_f(t)$ aus (Ebene mit dem Normalenvektor der Fläche und dem entsprechenden Abstand) und fragen uns, ob der bewegte Punkt $\mathbf{a}(t)$ zu einem Zeitpunkt $t \in (0,1)$ in der Ebene $H_f(t)$ liegt. Wie in Anhang A.3 hergeleitet wird, berechnet sich die Gleichung der bewegten Ebene wie folgt:

$$\begin{aligned} & \mathbf{n}(t)^T \mathbf{a}(t) &=& \; n_0(t) \\ \iff \quad & (\mathbf{R_2}(t)\mathbf{n})^T (\mathbf{R_1}(t)\mathbf{a} + \mathbf{o_1}(t)) &=& \; n_0 + \mathbf{o_2}(t)^T (\mathbf{R_2}(t)\mathbf{n}) \\ \iff \quad & \mathbf{n}^T \mathbf{R_2}(t)^T \mathbf{R_1}(t)\mathbf{a} + \mathbf{n}^T \mathbf{R_2}(t)(\mathbf{o_1}(t) - \mathbf{o_2}(t)) &=& \; n_0 \\ \iff \quad & \mathbf{n}^T \frac{\mathbf{R_2'}(t)}{N_2(t)}^T \frac{\mathbf{R_1'}(t)}{N_1(t)} \mathbf{a} + \mathbf{n}^T \frac{\mathbf{R_2'}(t)}{N_2(t)} (\mathbf{o_1}(t) - \mathbf{o_2}(t)) &=& \; n_0 \\ \iff \quad & \mathbf{n}^T \mathbf{R_2'}(t)^T \mathbf{R_1'}(t)\mathbf{a} + \mathbf{n}^T \mathbf{R_2'}(t)(\mathbf{o_1}(t) - \mathbf{o_2}(t)) N_1(t) &=& \; n_0 N_1(t) N_2(t) \end{aligned}$$

Der Normalenvektor $\mathbf{n}$ der Fläche $f_{\mathbf{v_{i_1}} \dots \mathbf{v_{i_s}}}$ läßt sich als $\mathbf{n} = \mathbf{v_{i_j}} \times \mathbf{v_{i_k}} + \mathbf{v_{i_k}} \times \mathbf{v_{i_l}} + \mathbf{v_{i_l}} \times \mathbf{v_{i_j}}$ aus geeigneten Ecken $\mathbf{v_{i_j}}, \mathbf{v_{i_k}}, \mathbf{v_{i_l}}$ der Fläche konstruieren. Für den Parameter n_0 erhält man $\mathbf{v_{i_j}}^T (\mathbf{v_{i_k}} \times \mathbf{v_{i_l}})$.

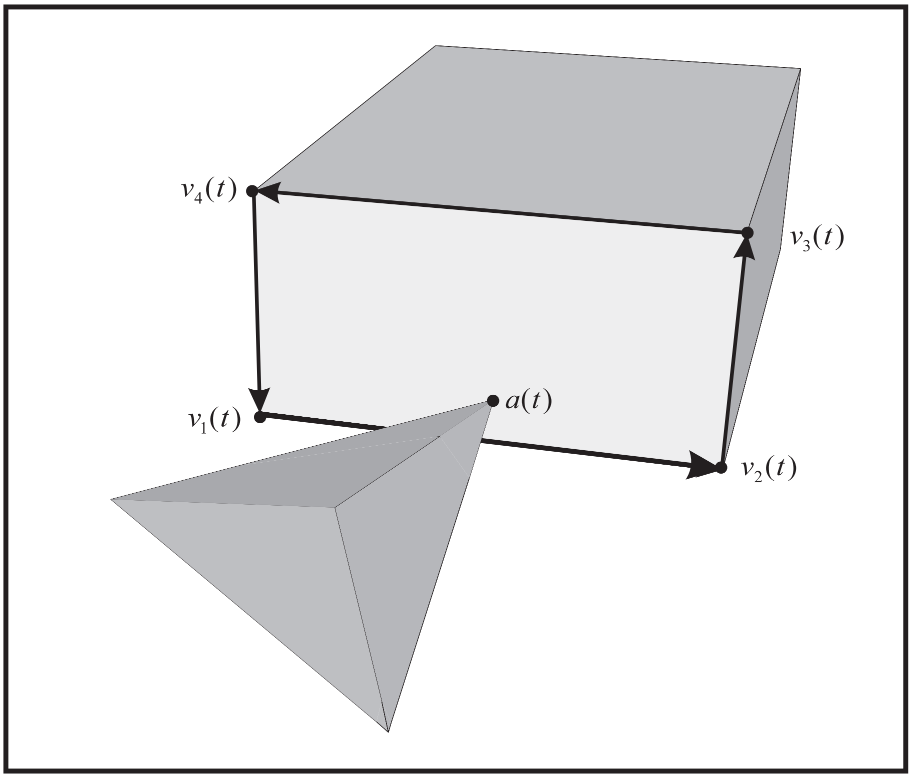

Abbildung 2.5: PUNKT-FLÄCHE-KOLLISION

Diese polynomielle Gleichung vom Typ $f(t) = 0$, die den Grad 5 besitzt und deren Koeffizienten gegenüber der Eingabe siebenfache Länge aufweisen, beschreibt eine für die Punkt-Fläche-Kollision notwendige, aber nicht hinreichende Bedingung. In einem nächsten Schritt müssen wir aus den so erhaltenen potentiellen Kollisionszeitpunkten die realen durch Angabe zusätzlicher Bedingungen selektieren.

2.3.3.2 Einschränkung der potentiellen auf reale Kollisionen

Liegen ein bewegter Punkt $\mathbf{a}(t)$ und eine bewegte Fläche $f_{\mathbf{v_{i_1}}\ldots\mathbf{v_{i_s}}}(t)$ zu einem Zeitpunkt $t_{pot-Col} \in (0,1)$ gemeinsam in einer Ebene, d.h. gilt $f(t_{pot-Col}) = 0$, so muß – soll eine Kollision der zugehörigen polyhedralen Objekte $\mathcal{P}_1$ und $\mathcal{P}_2$ vorliegen – der Punkt zum Zeitpunkt $t_{pot-Col}$ innerhalb des Flächensegmentes liegen. Um diese Tatsache effizienter überprüfen zu können, beschränken wir uns bei unseren Betrachtungen auf konvexe Flächen und projizieren die zum Zeitpunkt $\underline{t_{pot-Col}\ ebene}$ räumliche Szene in eine geeignete Standardebene. Bezeichnen wir mit $\overline{\mathbf{a}(t)}, \overline{\mathbf{v_{i_1}}(t)}, \ldots, \overline{\mathbf{v_{i_s}}(t)}$ die in die Standardebene mit dem Normalenvektor $\mathbf{e}$ projizierten Punkte, so gilt:

$$\overline{\mathbf{a}(t)} \cap \overline{f_{\mathbf{v_{i_1}}\ldots\mathbf{v_{i_s}}}(t)} \neq \emptyset \quad\Longrightarrow\quad \mathbf{a}(t) \cap f_{\mathbf{v_{i_1}}\ldots\mathbf{v_{i_s}}}(t) \neq \emptyset$$

Die Kanten der Fläche seien bzgl. des Flächennormalenvektors mathematisch positiv orientiert. Man erhält die Fläche als Schnitt der linken Halbebenen, die sich durch

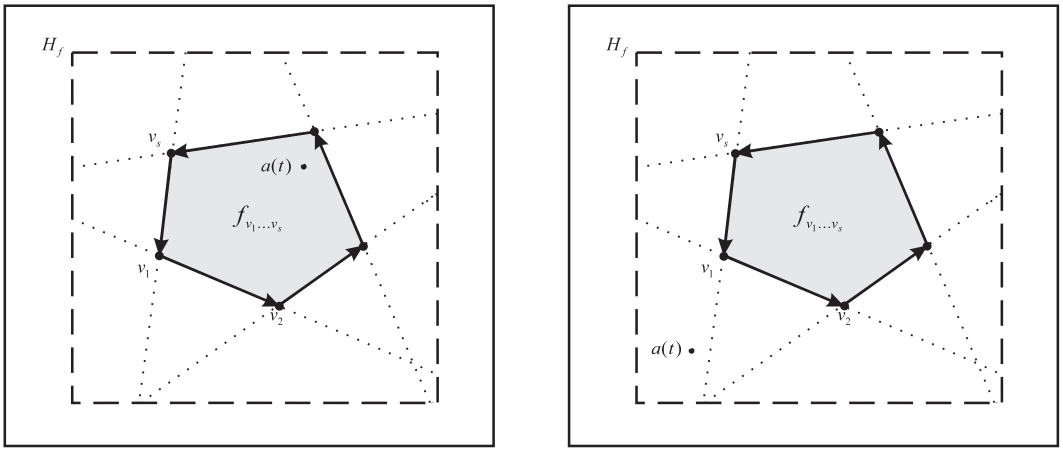

Abbildung 2.6: Lage Punkt-Fläche

Verlängerung der Begrenzungskanten zu Geraden ergeben. Soll der Punkt $\mathbf{a}(t_{pot-Col})$ innerhalb der Fläche – also im Schnitt der Halbebenen – liegen, so muß er *links* von allen zu Geraden verlängerten Flächenkanten liegen. Dies gilt jedoch nur dann, wenn durch die Projektion die Orientierung der Kanten nicht verändert wird. Ansonsten ist *links* durch *rechts* zu ersetzen. Bezeichnen wir mit $\mathbf{e}$ den Normalenvektor der Projek-

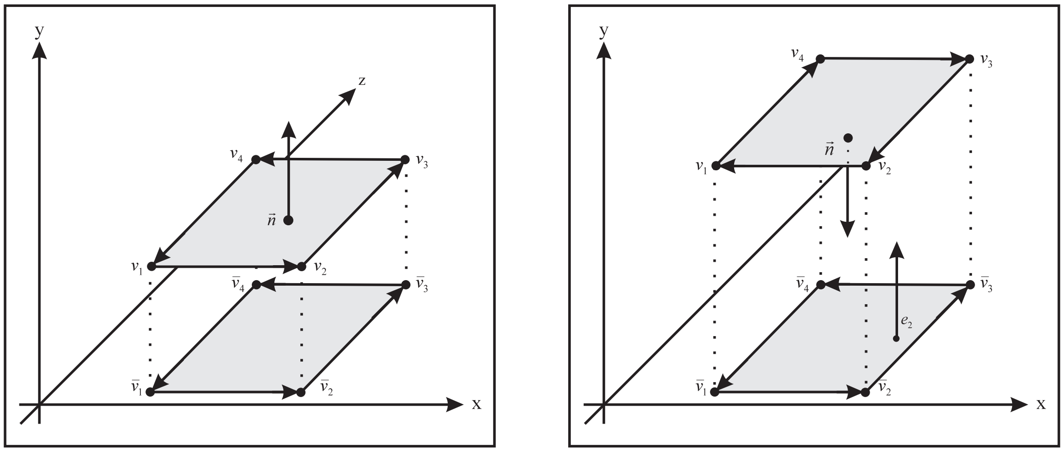

Abbildung 2.7: Orientierungswechsel nach Projektion

tionsebene und mit $\mathbf{n_{f(t)}}$ den Normalenvektor der bewegten Fläche, so kann man sich überlegen:

- Die Orientierung der Flächenkanten bleibt nach der Projektion in eine Standardebene unverändert, wenn $0^o \leq \angle(\mathbf{e}, \mathbf{n_f}) < 90^o \Longrightarrow \mathbf{e}^T \mathbf{n_f} > 0$.

- Die Orientierung der Flächenkanten wird nach der Projektion in eine Standardebene umgekehrt, wenn $90^o < \angle(\mathbf{e}, \mathbf{n_f}) \leq 180^o \Longrightarrow \mathbf{e}^T \mathbf{n_f} < 0$.

- Die Szene entartet durch die Projektion, wenn $\angle(\mathbf{e}, \mathbf{n_f}) = 90^o$, weshalb wir eine solche Projektion nicht zulassen.

Mit Hilfe dieser Überlegungen sind wir nun in der Lage, mittels des Vorzeichens des Skalarproduktes $\mathbf{e}^T\mathbf{n_{f(t)}}$ nach der Projektion eine Unterscheidung von *links* und *rechts* in der Standardebene (Normalenvektor e) gemäß Satz 1.5 vorzunehmen. Angewendet auf den Fall unserer projizierten Fläche $\overline{f_{\mathbf{v_{i_1}}\ldots\mathbf{v_{i_s}}}}$ mit s Kanten ergeben sich dann $s+1$ Nebenbedingungen:

$$
\begin{array}{rrcl}
 & \mathbf{e}^T\mathbf{n_{f(t)}} & > & 0 \\
\wedge & \mathbf{e}^T\Big((\overline{\mathbf{v_2}(t)} - \overline{\mathbf{v_1}(t)}) \times (\overline{\mathbf{a}(t)} - \overline{\mathbf{v_1}(t)})\Big) & > & 0 \\
\wedge & \mathbf{e}^T\Big((\overline{\mathbf{v_3}(t)} - \overline{\mathbf{v_2}(t)}) \times (\overline{\mathbf{a}(t)} - \overline{\mathbf{v_2}(t)})\Big) & > & 0 \\
 & & \vdots & \\
\wedge & \mathbf{e}^T\Big((\overline{\mathbf{v_s}(t)} - \overline{\mathbf{v_{s-1}}(t)}) \times (\overline{\mathbf{a}(t)} - \overline{\mathbf{v_{s-1}}(t)})\Big) & > & 0 \\
\wedge & \mathbf{e}^T\Big((\overline{\mathbf{v_1}(t)} - \overline{\mathbf{v_s}(t)}) \times (\overline{\mathbf{a}(t)} - \overline{\mathbf{v_s}(t)})\Big) & > & 0 \\
 & \text{oder} & & \\
 & \mathbf{e}^T\mathbf{n_{f(t)}} & < & 0 \\
\wedge & \mathbf{e}^T\Big((\overline{\mathbf{v_2}(t)} - \overline{\mathbf{v_1}(t)}) \times (\overline{\mathbf{a}(t)} - \overline{\mathbf{v_1}(t)})\Big) & < & 0 \\
\wedge & \mathbf{e}^T\Big((\overline{\mathbf{v_3}(t)} - \overline{\mathbf{v_2}(t)}) \times (\overline{\mathbf{a}(t)} - \overline{\mathbf{v_2}(t)})\Big) & < & 0 \\
 & & \vdots & \\
\wedge & \mathbf{e}^T\Big((\overline{\mathbf{v_s}(t)} - \overline{\mathbf{v_{s-1}}(t)}) \times (\overline{\mathbf{a}(t)} - \overline{\mathbf{v_{s-1}}(t)})\Big) & < & 0 \\
\wedge & \mathbf{e}^T\Big((\overline{\mathbf{v_1}(t)} - \overline{\mathbf{v_s}(t)}) \times (\overline{\mathbf{a}(t)} - \overline{\mathbf{v_s}(t)})\Big) & < & 0
\end{array}
$$

Exemplarisch möchten wir eine solche Nebenbedingung expandieren:

$$
\begin{array}{rcl}
\mathbf{e}^T\Big((\mathbf{v_{i+1}}(t) - \mathbf{v_i}(t)) \times (\mathbf{a}(t) - \mathbf{v_i}(t))\Big) & > & 0 \\
\iff \mathbf{e}^T\Big((\mathbf{R_2}(t)\mathbf{v_{i+1}} + \mathbf{o_2}(t) - \mathbf{R_2}(t)\mathbf{v_i} - \mathbf{o_2}(t)) \times & & \\
(\mathbf{R_1}(t)\mathbf{a} + \mathbf{o_1}(t) - \mathbf{R_2}(t)\mathbf{v_i} - \mathbf{o_2}(t))\Big) & > & 0 \\
\iff \mathbf{e}^T\Big((\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_{i+1}} - \frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) \times (\frac{\mathbf{R_1'}(t)}{N_1(t)}\mathbf{a} + \mathbf{o_1}(t) - \frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i} - \mathbf{o_2}(t))\Big) & > & 0 \\
\iff \mathbf{e}^T\Big((\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_{i+1}}) \times (\frac{\mathbf{R_1'}(t)}{N_1(t)}\mathbf{a}) + (\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_{i+1}}) \times \mathbf{o_1}(t) - & & \\
(\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_{i+1}}) \times (\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) - (\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_{i+1}}) \times \mathbf{o_2}(t) - & & \\
(\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) \times (\frac{\mathbf{R_1'}(t)}{N_1(t)}\mathbf{a}) - (\frac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) \times \mathbf{o_1}(t) - & &
\end{array}
$$

$$
\begin{aligned}
&(\tfrac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) \times (\tfrac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) + (\tfrac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) \times \mathbf{o_2}(t)\Big) \quad > \quad 0 \\
\Longleftrightarrow \quad & \mathbf{e}^T\Big((\tfrac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_{i+1}}) \times (\tfrac{\mathbf{R_1'}(t)}{N_1(t)}\mathbf{a}) + (\tfrac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_{i+1}}) \times \mathbf{o_1}(t) - \\
&\tfrac{\mathbf{R_2'}(t)}{N_2(t)}(\mathbf{v_{i+1}} \times \mathbf{v_i}) - (\tfrac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_{i+1}}) \times \mathbf{o_2}(t) - \\
&(\tfrac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) \times (\tfrac{\mathbf{R_1'}(t)}{N_1(t)}\mathbf{a}) - (\tfrac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) \times \mathbf{o_1}(t) + \\
&(\tfrac{\mathbf{R_2'}(t)}{N_2(t)}\mathbf{v_i}) \times \mathbf{o_2}(t)\Big) \quad > \quad 0 \\
\Longleftrightarrow \quad & \mathbf{e}^T\Big(((\mathbf{R_2'}(t)\mathbf{v_{i+1}}) \times (\mathbf{R_1'}(t)\mathbf{a})) + ((\mathbf{R_2'}(t)\mathbf{v_{i+1}}) \times \mathbf{o_1}(t))N_1(t) - \\
&(\mathbf{R_2'}(t)(\mathbf{v_{i+1}} \times \mathbf{v_i}))N_1(t) - ((\mathbf{R_2'}(t)\mathbf{v_{i+1}}) \times \mathbf{o_2}(t))N_1(t) - \\
&((\mathbf{R_2'}(t)\mathbf{v_i}) \times (\mathbf{R_1'}(t)\mathbf{a})) - ((\mathbf{R_2'}(t)\mathbf{v_i}) \times \mathbf{o_1}(t))N_1(t) + \\
&((\mathbf{R_2'}(t)\mathbf{v_i}) \times \mathbf{o_2}(t))N_1(t)\Big) \quad > \quad 0
\end{aligned}
$$

Diese polynomielle Ungleichung im Zeitparameter t besitzt bei Verwendung linearer Interpolation für die zeitliche Beschreibung der Konfigurationsübergänge den Grad 5. Gegenüber den Eingabedaten wächst die Länge ihrer Koeffizienten um den Faktor 6. Somit erhalten wir auch im Fall der Punkt-Fläche-Kollision ein polynomielles Ungleichungssystem der Form

$$
\begin{array}{rrcl}
 & f(t) & = & 0 \quad \} \text{ potentielle Kollisionszeitpunkte} \\
\wedge & g_1(t) & > & 0 \quad \} \text{ Bedingung zur Definition von } \textit{rechts} \text{ und } \textit{links} \text{ in der Projektionsebene} \\
\wedge & g_2(t) & > & 0 \\
 & & \vdots & \quad \text{Bedingungen für den Schnitt eines Punktes und einer Fläche in der Ebene} \\
\wedge & g_s(t) & > & 0 \\
\wedge & g_{s+1}(t) & > & 0
\end{array}
$$

von dem wir *exakt* entscheiden müssen, ob es für $t \in (0,1)$ eine Lösung besitzt.

2.3.4 Punkt-Punkt-Kollision

Sei $\mathbf{a}(t)$ eine Ecke des sich bewegenden Polyeders $\mathcal{P}_1$ und $\mathbf{b}(t)$ eine Ecke des sich bewegenden Polyeders $\mathcal{P}_2$. Zwei bewegte Punkte des $\mathbb{R}^3$ kollidieren genau dann, wenn

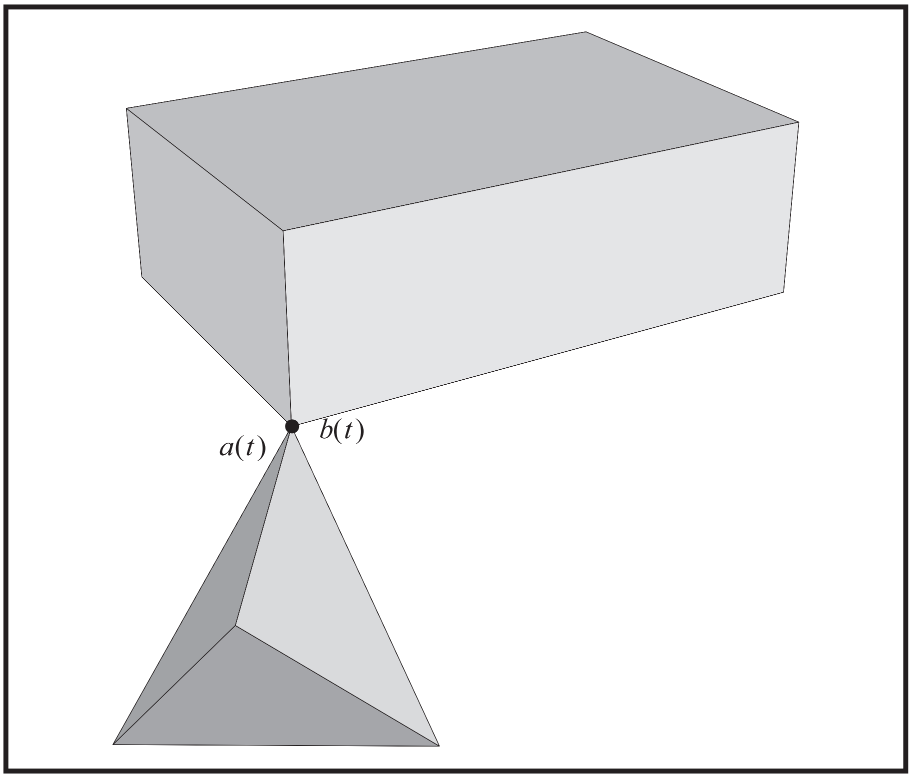

Abbildung 2.8: Punkt-Punkt-Kollision

sie in ihren Koordinaten übereinstimmen, d.h. wenn gilt:

$$
\begin{array}{rrcl}
 & \mathbf{a}(t) & = & \mathbf{b}(t) \\
\Longleftrightarrow & \mathbf{f}(t) := \mathbf{a}(t) - \mathbf{b}(t) & = & \vec{0} \\
\Longleftrightarrow & f_x(t) := a_x(t) - b_x(t) & = & 0 \\
 & \wedge\ f_y(t) := a_y(t) - b_y(t) & = & 0 \\
 & \wedge\ f_z(t) := a_z(t) - b_z(t) & = & 0 \\
\Longleftrightarrow & f'(t) := f_x(t)^2 + f_y(t)^2 + f_z(t)^2 & = & 0
\end{array}
$$

Untersucht man den Vektor $\mathbf{f}(t)$ mit seinen polynomiellen Komponenten $f_i(t)$ genauer, erhält man:

$$
\begin{array}{rrcl}
 & \mathbf{f}(t) & = & \mathbf{a}(t) - \mathbf{b}(t) \\
 & & = & \left(\frac{\mathbf{R'_1}(t)}{N_1(t)}\mathbf{a} + \mathbf{o_1}(t)\right) - \left(\frac{\mathbf{R'_2}(t)}{N_2(t)}\mathbf{b} + \mathbf{o_2}(t)\right) \\
\Longleftrightarrow & \mathbf{f}(t)N_1(t)N_2(t) & = & (\mathbf{R'_1}(t)\mathbf{a})N_2(t) - (\mathbf{R'_2}(t)\mathbf{b})N_1(t) + (\mathbf{o_1}(t) - \mathbf{o_2}(t))N_1(t)N_2(t)
\end{array}
$$

Als Komponenten des ganzzahligen, polynomiellen Vektors $\mathbf{f}(t)N_1(t)N_2(t)$ ergeben sich bei linearer Interpolation zwischen den verschiedenen Konfigurationen Polynome vom

Grad 5, so daß die endgültige polynomielle Bedingung

$$f'(t) \quad = \quad (f(t)N_1(t)N_2(t))_x^2 + (f(t)N_1(t)N_2(t))_y^2 + (f(t)N_1(t)N_2(t))_z^2$$

den Grad 10 erreicht. Bedenkt man, daß bei den bisherigen Fällen die polynomiellen Bedingungen maximal den Grad 5 erreichten, erscheint der Grad 10 zu hoch.

Deshalb greifen wir an dieser Stelle wieder auf Satz 2.3 zurück, der es uns gestattet die Bedingung $\mathbf{f}(t) = \vec{0}$ gemäß

$$\exists\, t_0 \in \mathbb{C} \text{ mit } \mathbf{f}(t_0) = \vec{0} \iff ggt(f_x(t), f_y(t), f_z(t)) \neq 1$$

zu überprüfen. Für den ggT der 3 polynomiellen Vektorkomponenten gilt offensichtlich:

$$0 \leq grad(ggt(f_x(t), f_y(t), f_z(t))) \leq 5$$

Zur Beschreibung der Kollisionsbedingungen im Fall der Punkt-Punkt-Kollision genügt – im Gegensatz zu den bisher behandelten Fällen – lediglich eine polynomielle Hauptbedingung der Form $f(t) = 0$.

2.4 Zusammenfassung

In den vorigen Abschnitten haben wir für die vier möglichen Kollisionsarten

- Kante-Kante-Kollision
- Punkt-Kante-Kollision
- Punkt-Fläche-Kollision
- Punkt-Punkt-Kollision

zweier gleichzeitig bewegter Polyeder $\mathcal{P}_1$ und $\mathcal{P}_2$ Kollisionsbedingungen hergeleitet. Parametrisieren wir die Bewegung der Polyeder über dem Zeitintervall $(0,1)$, ergeben sich in allen vier Fällen mehrere, durch "$\vee$" verknüpfte, polynomielle Gleichungssysteme der Art

$$\begin{aligned} f(t) &= 0 \\ \wedge\, g_1(t) &> 0 \\ &\vdots \\ \wedge\, g_l(t) &> 0 \end{aligned}$$

von denen es zu entscheiden gilt, ob sie für ein $t \in (0,1)$ eine Lösung besitzen. Je nach betrachtetem Kollisionstyp variiert hierbei lediglich die Anzahl der Nebenbedingungen. Bei linearer Interpolation zwischen den vorgegebenen Konfigurationen der einzelnen Objekte erreichen die Polynome maximal den Grad 5. Die Länge der Koeffizienten wächst gegenüber der Eingabe höchstens (Kante-Kante-Kollision, Punkt-Fläche-Kollision) um den Faktor 7.

Wir haben unser Ausgangsproblem auf diese Weise auf ein rein algebraisches Problem, nämlich die Vorzeichenbestimmung mehrerer Polynome $g_i(t)$ an den Nullstellen eines Polynoms $f(t)$ im Intervall $(0,1)$, reduziert:

$$\#\{t \in (0,1) \mid f(t) = 0 \ \wedge \ g_i(t) > 0 \ \forall 1 \leq i \leq l\} \quad > \quad 0 \ ?$$

Da jedoch selbst unter einfachsten Modellannahmen die polynomiellen Kollisionsbedingungen schon den Grad 5 erreichen, läßt sich obige Frage nur mit speziellen Verfahren *exakt* beantworten. Es ist z.B. nicht möglich, die Nullstellen des Polynoms $f(t)$ durch eine geschlossene Lösungsformel zu bestimmen und die Vorzeichen der Nebenbedingungen $g_i(t)$ an solchen Nullstellen aus dem Intervall $(0,1)$ zu berechnen. Wegen unseres Ziels der *exakten* Lösung des Entscheidungsproblems 2.1 entfällt auch ein numerischer Lösungsansatz, d.h. numerische Nullstellenbestimmung des Polynoms $f(t)$ und Vorzeichenauswertung der Polynome $g_i(t)$ an den so gewonnenen Nullstellen im Intervall $(0,1)$.

Im folgenden Kapitel werden wir deshalb einige spezielle Verfahren vorstellen, die es uns erlauben, obiges Problem *exakt* zu entscheiden.

Kapitel 3

Existenz von Nullstellen unter polynomiellen Nebenbedingungen

Im vorigen Kapitel haben wir gesehen, daß uns die Frage, ob zwei polyhedrale Objekte während ihrer Bewegung zusammenstoßen, auf das Problem führt, von polynomiellen Gleichungssystemen der Art

$$\begin{array}{rcl} f(t) & = & 0 \\ \wedge\, g_1(t) & > & 0 \\ & \vdots & \\ \wedge\;\, g_l(t) & > & 0 \end{array} \tag{3.1}$$

entscheiden zu müssen, ob sie in einem vorgegebenen Intervall (bei uns $(0,1)$) eine Lösung besitzen. Die entstandenen Polynome erreichten bereits unter einfachsten Modellannahmen (lineare Interpolation zwischen den Konfigurationen aufeinanderfolgender Zeitpunkte) den Grad 5, was uns das Leben zusätzlich erschwert und spezielle Verfahren zur Entscheidungsfindung erfordert. Ein Satz aus der Algebra von *Abel-Ruffini* besagt nämlich:

Satz 3.1 :
Die allgemeine Gleichung n-ten Grades

$$f(t) = a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t^1 + a_0 = 0$$

ist für $n \geq 5$ nicht auflösbar.

Dabei versteht man unter *Auflösung einer Gleichung f(t) = 0*, daß die Nullstellen des Polynoms f durch endlichfache Anwendung der Körperoperationen und des Prozesses des *Wurzelziehens* aus den Elementen des Koeffizientenkörpers gewonnen werden.

Es ist demnach nicht möglich, die Nullstellen des Polynoms $f(t)$ durch eine geschlossene Lösungsformel zu bestimmen und die Vorzeichen der Nebenbedingungen $g_i(t)$ an solchen Nullstellen aus dem Intervall $(0,1)$ zu berechnen. Wegen unseres Ziels der

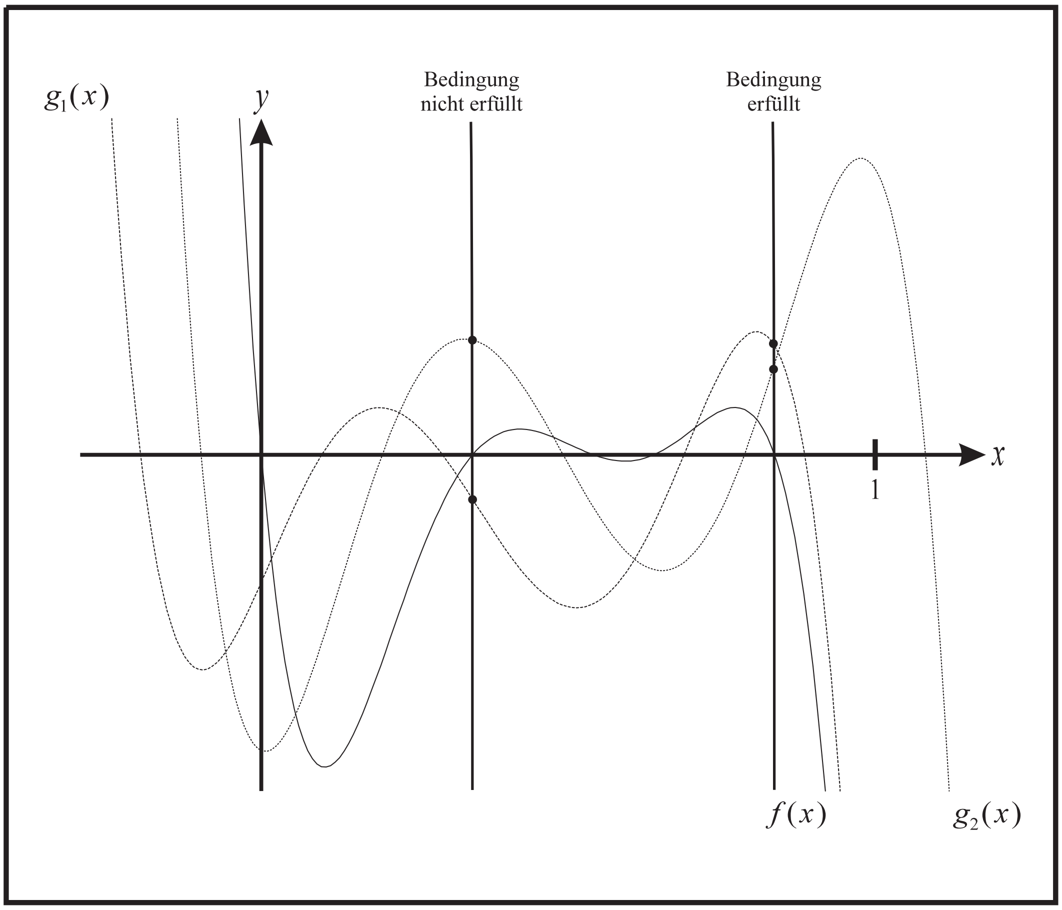

Abbildung 3.1: POLYNOMIELLE KURVEN

exakten Lösung des Entscheidungsproblems 2.1 entfällt auch ein numerischer Lösungsansatz, d.h. numerische Nullstellenbestimmung des Polynoms $f(t)$ und Vorzeichenauswertung der Polynome $g_i(t)$ an den so gewonnenen Nullstellen im Intervall $(0, 1)$.

In den folgenden Abschnitten werden wir deshalb zwei spezielle Verfahren vorstellen, die es uns erlauben obiges Problem *exakt* zu entscheiden. Daß wir eine solche *exakte* Lösung teurer bezahlen müssen als eine numerische, ist verständlich: die Laufzeit der hier vorgestellten Verfahren wird im Wesentlichen von folgenden Faktoren beeinflußt:

- Anzahl der Nebenbedingungen

- Grad der Polynome

- Länge der Koeffizienten

3.1 Das Verfahren von Pedersen

In diesem Abschnitt werden wir ein Verfahren zur Lösung des Entscheidungsproblems 2.1 kennenlernen, das in seinem Ablauf absolut statisch ist: bevor Aussagen über die Existenz von Lösungen gemacht werden können, muß ein starres Schema gänzlich durchlaufen werden. Bei unserem Alternativverfahren, das wir im nächsten Abschnitt vorstellen werden, kann bereits während der Abarbeitung des algorithmischen Schemas erkannt werden, daß das zu untersuchende Gleichungssystem keine Lösung besitzt, was einen vorzeitigen Ausstieg – und somit eine Laufzeiteinsparung – ermöglicht.

Zunächst werden wir die Frage, ob ein polynomielles Gleichungssystem der Art 3.1 eine Lösung besitzt, gemäß [Ped91] so vereinfachen, daß wir nur noch eine Nebenbedingung betrachten müssen. Für das verbleibende Gleichungssystem der Art

$$\begin{aligned} f(t) &= 0 \\ \wedge\, g(t) &> 0 \end{aligned} \tag{3.2}$$

hilft uns die in [US92] beschriebene Methode von Hermite zur Bestimmung der Anzahl seiner Lösungen.

3.1.1 Reduktion der Anzahl der Nebenbedingungen

Die Menge der Lösungen des Gleichungssystems 3.1 läßt sich beschreiben als

$$\{t \in \mathbb{R} \mid f(t) = 0 \wedge g_1(t) > 0 \wedge \ldots \wedge g_l(t) > 0\}.$$

Mit

$$Z_f \;=\; \{t \in \mathbb{R} \mid f(t) = 0\}$$

bezeichnen wir die Menge der reellen Nullstellen des Polynoms f, mit

$$\chi_g(t) \;=\; \begin{cases} 1 & , \text{ falls } g_1(t) > 0 \wedge \ldots \wedge g_l(t) > 0 \\ 0 & , \; sonst \end{cases}$$

die charakteristische Funktion des Prädikates $[g_1(t) > 0 \wedge \ldots \wedge g_l(t) > 0]$. Für die Anzahl der Lösungen des Gleichungssystems 3.1 gilt dann:

$$\#\{t \in \mathbb{R} \mid f(t) = 0 \wedge g_1(t) > 0 \wedge \ldots \wedge g_l(t) > 0\} = \sum_{t \in Z_f} \chi_g(t) \tag{3.3}$$

Wenn wir zu Beginn sicherstellen, daß keines der Polynome g_i, $1 \le i \le n$, mit dem Polynom f eine gemeinsame Nullstelle besitzt – diese können das Gleichungssystem 3.1 nicht erfüllen –, so läßt sich, wie man leicht überprüft, $\chi_g(t)$ berechnen durch:

$$\begin{aligned} \chi_g(t) &= 2^{-l} \prod_{i=1}^{l} (1 + sgn(g_i(t))) && (3.4) \\ &= 2^{-l} \sum_{I \in 2^{\mathbb{N}_l}} \prod_{i \in I} sgn(g_i(t)) && (3.5) \end{aligned}$$

Hierbei bezeichnet $2^{\mathbb{N}_l}$ die Potenzmenge der Menge $\mathbb{N}_l = \{1, \ldots, l\}$. Gleichung 3.5 ergibt sich durch Ausmultiplizieren von Gleichung 3.4 und entsprechende Summation der entstehenden Terme. Ist das Prädikat erfüllt, bewirkt der Faktor 2^{-l} die Kürzung des Produktergebnisses 2^l auf 1, wie von der charakteristischen Funktion $\chi_g(t)$ verlangt. In dem Fall, daß mindestens eine Nebenbedingung $g_i(t) > 0$ nicht erfüllt ist, ihr Vorzeichen an der betrachteten Nullstelle von f also negativ ist, wird der entsprechende Faktor $(1 + sgn(g_i(t)))$ und somit das gesamte Produkt 0. Sollte eine Nebenbedingung konstant den Wert 0 besitzen, was – wie man sich für die einzelnen Kollisionstests überlegt – einer entarteten Situation entspricht, kann die Berechnung abgebrochen werden; die Kontaktsituation wird dann von einem anderen Kollisionstest erkannt, in dem die Nebenbedingungen nicht entarten (z.B. Kante-Kante → Punkt-Kante).

Setzen wir nun 3.5 in 3.3 ein, so ergibt sich für die Anzahl der reellen Wurzeln des Polynoms f, die die obigen Nebenbedingungen erfüllen:

$$\begin{aligned}
\sum_{t \in Z_f} \chi_g(t) &= 2^{-l} \sum_{I \in 2^{\mathbb{N}_l}} \sum_{t \in Z_f} \prod_{i \in I} sgn(g_i(t)) \\
&= 2^{-l} \sum_{I \in 2^{\mathbb{N}_l}} \Bigg(\#\{\underbrace{t \in Z_f}_{f(t)=0} \mid \underbrace{\prod_{i \in I} g_i(t) > 0}_{g(t)>0}\} - \\
&\qquad \#\{\underbrace{t \in Z_f}_{f(t)=0} \mid \underbrace{\prod_{i \in I} g_i(t) < 0}_{g(t)<0}\} \Bigg) \qquad (3.6)
\end{aligned}$$

Formel 3.6 gibt den Rahmen des Algorithmus zur Bestimmung der Anzahl der Lösungen von Gleichungssystem 3.1 vor. Durch die Umformung von Gleichung 3.3 zu Gleichung 3.6 haben wir eine Reduktion der Komplexität in der Anzahl der zu betrachtenden Nebenbedingungen erreicht. Die Reduktion in diesem Bereich müssen wir allerdings mit einer größeren Anzahl der zu untersuchenden Gleichungssysteme – jetzt nämlich 2^l der Art 3.2 – bezahlen.

An diesem Punkt angelangt, benötigen wir nun ein Verfahren zur Berechnung der Anzahl der reellen Nullstellen eines Polynoms f, die zusätzlich einer polynomiellen Nebenbedingung g gehorchen.

3.1.2 Die Methode von Hermite für eine polynomielle Nebenbedingung

Im nun folgenden Abschnitt werden wir ein Verfahren vorstellen, das es uns ermöglicht, die Anzahl der reellen Nullstellen eines polynomiellen Gleichungssystems der Art 3.2 *exakt* zu berechnen; es basiert lediglich auf *algebraischen* Methoden: zur Berechnung genügen endlich viele algebraische Operationen auf den Koeffizienten der Polynome f und g:

$$f(t) = \sum_{i=0}^{n} a_i t^i$$

$$g(t) = \sum_{i=0}^{m} b_i t^i$$

Um im späteren Verlauf des Verfahrens auf Fallunterscheidungen verzichten zu können und frei von Entartungen zu sein, eliminieren wir im Polynom f

- mehrfache Nullstellen und

- etwaige mit g gemeinsame Nullstellen

Mehrfachnullstellen von f können zu einfachen Nullstellen reduziert werden, da dieser Vorgang auf die Existenz von Lösungen des Gleichungssystems 3.2 keine Auswirkungen hat. Damit dies auch im Fall der Eliminierung von mit g gemeinsamen Nullstellen so bleibt, müssen diese im Polynom f (und nicht im Polynom g) entfernt werden. Eine Eliminierung solcher Nullstellen im Polynom g könnte Auswirkungen auf die Vorzeichen von g an weiteren Nullstellen von f – und somit auf die Anzahl der Lösungen von 3.2 – nach sich ziehen.

Mit $\lambda_1, \cdots, \lambda_n$ bezeichnen wir die Nullstellen von $f(t)$, mit s_k die Newtonsche Summe $\sum_{i=1}^{n} \lambda_i^k$. Zusätzlich definieren wir $h_k = \sum_{i=1}^{n} g(\lambda_i)\lambda_i^k$. Da s_k und h_k symmetrische Polynome in $\lambda_1, \cdots, \lambda_n$ sind, kann man sie laut [US92] als rationale Funktionen in den Koeffizienten von f und g wie folgt ausdrücken:

$$s_k = \begin{cases} n & \text{für} \quad k = 0 \\ -\frac{1}{a_n}\, a_{n-1} & \text{für} \quad k = 1 \\ -\frac{1}{a_n}\,(a_{n-1}s_{k-1} + \cdots + a_{n-k+1}s_1 + k a_{n-k}) & \text{für} \quad 2 \le k \le n \\ -\frac{1}{a_n}\,(a_{n-1}s_{k-1} + a_{n-2}s_{k-2} + \cdots + a_0 s_{k-n}) & \text{für} \quad k > n \end{cases} \tag{3.7}$$

$$h_k = b_m s_{k+m} + b_{m-1}s_{k+m-1} + \cdots + b_0 s_k \tag{3.8}$$

Eine wichtige Rolle bei diesem Verfahren spielen die beiden Hankelmatrizen $\mathbf{S}$ und $\mathbf{H}$, die aus den Werten s_k und h_k wie folgt aufgebaut werden:

$$\mathbf{S} = [s_{i+j}]_{i,j=0}^{n-1} = \begin{bmatrix} s_0 & s_1 & s_2 & \ldots & s_{n-1} \\ s_1 & s_2 & s_3 & \ldots & s_n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ s_{n-1} & s_n & s_{n+1} & \ldots & s_{2n-2} \end{bmatrix}$$

$$\mathbf{H} = [h_{i+j}]_{i,j=0}^{n-1} = \begin{bmatrix} h_0 & h_1 & h_2 & \ldots & h_{n-1} \\ h_1 & h_2 & h_3 & \ldots & h_n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ h_{n-1} & h_n & h_{n+1} & \ldots & h_{2n-2} \end{bmatrix}$$

Mit S_i bezeichnen wir folgende Determinante einer Teilmatrix von $\mathbf{S}$:

$$S_i = det \begin{bmatrix} s_0 & s_1 & \dots & s_{i-1} \\ s_1 & s_2 & \dots & s_i \\ \vdots & \vdots & \vdots & \vdots \\ s_{i-1} & s_i & \dots & s_{2i-2} \end{bmatrix}$$

Analog dazu bezeichnen wir mit H_i folgende Determinante einer Teilmatrix von $\mathbf{H}$:

$$H_i = det \begin{bmatrix} h_0 & h_1 & \dots & h_{i-1} \\ h_1 & h_2 & \dots & h_i \\ \vdots & \vdots & \vdots & \vdots \\ h_{i-1} & h_i & \dots & h_{2i-2} \end{bmatrix}$$

Die beiden nun folgenden Sätze von Jacobi und Sylvester/Hermite erlauben die Berechnung der Anzahl der reellen Nullstellen des polynomiellen Gleichungssystems 3.2 mittels dieser Determinanten. Zunächst jedoch eine Definition, von der wir im folgenden des öfteren Gebrauch machen werden:

Definition 3.2:
Sei $u = (u_1, \dots, u_s)$ eine Folge reeller Zahlen und $u' = (u'_1, \dots, u'_t)$ die Teilfolge von u aller Zahlen $u_i \neq 0$. Dann definieren wir $var(u_1, \dots, u_s)$ als die Anzahl der Vorzeichenwechsel in der Folge u':

$$var(u_1, \dots, u_s) = \#\{1 \leq i < t \mid u'_i \cdot u'_{i+1} < 0\}$$

Satz 3.3 (Jacobi)
Die Zahl der verschiedenen Nullstellen des Polynoms $f(t)$ ist gleich dem Rang r der Matrix $\mathbf{S}$. Die Zahl der verschiedenen reellen Nullstellen des Polynoms $f(t)$ ist gleich $r - 2\,var(1, S_1, S_2, \dots, S_r)$.

Satz 3.4 (Sylvester/Hermite)
Die Zahl der verschiedenen reellen Nullstellen von f(t), die die Bedingung $g(t) > 0$ erfüllen, ist gleich $r - var(1, S_1, S_2, \dots, S_r) - var(1, H_1, H_2, \dots, H_r)$.

Der Rang der Matrizen $\mathbf{S}$ und $\mathbf{H}$ ist abhängig von der Art der Nullstellen der Polynome f und g: besitzen diese Mehrfachnullstellen, ergeben sich lineare Abhängigkeiten in den Zeilen bzw. Spalten der beiden Matrizen, die dann auch keinen vollen Rang mehr besitzen. Dies führt dazu, daß gewisse Hauptminoren den Wert null annehmen. Wegen der komplexen Struktur der Matrizen im allgemeinen Fall werden wir diesen Sachverhalt an einem allgemein gehaltenen Beispiel verdeutlichen:

Beispiel:
Sei das Polynom $f(t) = (ct+d)^2 \cdot (at+b)$ gegeben. Wie sich ablesen läßt besitzt es eine einfache und eine doppelte Nullstelle. Eine Expansion ergibt für seine Koeffizienten:

$$\begin{aligned} a_0 &= bd^2 \\ a_1 &= 2bcd + ad^2 \\ a_2 &= bc^2 + 2acd \\ a_3 &= ac^2 \end{aligned}$$

Die Newton-Summen zum Aufbau der Matrix $\mathbf{S}$ berechnen sich zu:

$$\begin{aligned} s_0 &= 3 \\ s_1 &= -\frac{bc+2ad}{ac} \\ s_2 &= \frac{b^2c^2+2a^2d^2}{a^2c^2} \\ s_3 &= -\frac{b^3c^3+2a^3d^3}{a^3c^3} \\ s_4 &= \frac{b^4c^4+2a^4d^4}{a^4c^4} \end{aligned}$$

Für die Hankelmatrix

$$\mathbf{S} = \begin{pmatrix} 3 & -\frac{bc+2ad}{ac} & \frac{b^2c^2+2a^2d^2}{a^2c^2} \\ -\frac{bc+2ad}{ac} & \frac{b^2c^2+2a^2d^2}{a^2c^2} & -\frac{b^3c^3+2a^3d^3}{a^3c^3} \\ \frac{b^2c^2+2a^2d^2}{a^2c^2} & -\frac{b^3c^3+2a^3d^3}{a^3c^3} & \frac{b^4c^4+2a^4d^4}{a^4c^4} \end{pmatrix}$$

gilt $Rang(\mathbf{S}) = 2$, da die drei Spaltenvektoren linear abhängig sind.

Da die betrachteten Polynome unseren Voraussetzungen entsprechend keine gemeinsamen und keine Mehrfachnullstellen besitzen dürfen, werden die Matrizen $\mathbf{S}$ und $\mathbf{H}$ immer vollen Rang besitzen. Sollte dennoch eine der Folgen $S_1, \ldots, S_r$ bzw. $H_1, \ldots, H_r$ Nullen enthalten, so kann man die Regel von *Frobenius* anwenden (siehe [Gan59]).

Um den Term $r - var(1, S_1, S_2, \ldots, S_r) - var(1, H_1, H_2, \ldots, H_r)$ *exakt* berechnen zu können, ist es zunächst erforderlich, die verschiedenen Determinanten S_i und H_i exakt zu berechnen. Da die Einträge der zugehörigen Matrizen rational sind, wir aber mit Ganzzahlarithmetik operieren, ist eine Modifikation der Definition der s_k und h_k notwendig. Dazu überlegt man sich, daß die Newtonsche Summe s_k gemäß 3.8 den Nenner a_n^k besitzt. Wenn wir vor Beginn unserer Betrachtungen sicherstellen, daß der Leitkoeffizient a_n des Polynoms f ein positives Vorzeichen besitzt, verändern wir durch Multiplikation der s_k mit dem Nenner a_n^k das Vorzeichen der Terme nicht. Die modifizierten Summen s'_k lauten dann:

$$s'_k = a_n^k s_k$$

$$= \begin{cases} n & \text{für} \quad k=0 \\ -a_{n-1} & \text{für} \quad k=1 \\ -(a_{n-1}s_{k-1} + \ldots + a_{n-k+1}s_1 + ka_{n-k})a_n^{k-1} & \text{für} \quad 2 \le k \le n \\ -(a_{n-1}s_{k-1} + a_{n-2}s_{k-2} + \ldots + a_0 s_{k-n})a_n^{k-1} & \text{für} \quad k > n \end{cases}$$

$$= \begin{cases} n & \text{für} \quad k=0 \\ -a_{n-1} & \text{für} \quad k=1 \\ -(a_{n-1}s'_{k-1}a_n^0 + \ldots + a_{n-k+1}s'_1 a_n^{k-2} + ka_{n-k}a_n^{k-1}) & \text{für} \quad 2 \le k \le n \\ -(a_{n-1}s'_{k-1}a_n^0 + \ldots + a_0 s'_{k-n}a_n^{n-1}) & \text{für} \quad k > n \end{cases} \tag{3.9}$$

Um die Größe der Zahlen für die algorithmische Weiterverarbeitung so gering wie möglich zu halten, werden wir die s'_k rechnerintern als Polynome in a_n repräsentieren. Die aus diesen modifizierten Newtonschen Summen s'_k bestehende Matrix bezeichnen wir mit $\mathbf{S}'$. In Anlehnung an die zuvor eingeführte Notation bezeichnen wir die Determinanten der Hauptminoren dieser Matrix $\mathbf{S}'$ mit S'_i:

$$S'_i \;=\; det \begin{bmatrix} s'_0 & s'_1 & \ldots & s'_{i-1} \\ s'_1 & s'_2 & \ldots & s'_i \\ \vdots & \vdots & \vdots & \vdots \\ s'_{i-1} & s'_i & \ldots & s'_{2i-2} \end{bmatrix} \tag{3.10}$$

Erst die Vorzeichenberechnung einer solchen Determinante erfordert eine Auswertung des entstehenden Polynoms. Die Effizienz des Verfahrens ist demnach vor allem von der Güte der Implementierung der Operationen auf großen Integern und auf Polynomen abhängig - doch dazu mehr in Kapitel 4.

Daß die Korrektheit des Verfahrens auch nach der Modifikation der s_k erhalten bleibt, zeigt der folgende Satz:

Satz 3.5:
Besitzt der Leitkoeffizient a_n des Polynoms f ein positives Vorzeichen, dann gilt:

$$sign(S_i) = sign(S'_i)$$

Beweis:
Es gilt: $s'_k = a_n^k s_k$.

$$\begin{aligned} S'_i &= \sum_{\pi \in \sigma_i} (-1)^{\sigma(\pi)} \cdot s'_{0\pi(0)} \cdot s'_{1\pi(1)} \cdot \ldots \cdot s'_{i\pi(i)} \\ &= \sum_{\pi \in \sigma_i} (-1)^{\sigma(\pi)} \cdot a_n^{0+\pi(0)} s_{0\pi(0)} \cdot \ldots \cdot a_n^{i-1+\pi(i-1)} s_{i-1\pi(i-1)} \\ &= a_n^{2 \cdot \sum_{j=0}^{i-1} j} \cdot \sum_{j=0}^{i-1} (-1)^{\sigma(\pi)} \cdot s_{j\pi(j)} \\ &= a_n^{i(i-1)} \cdot S_i \end{aligned}$$

Stellen wir nun vor Beginn unserer Betrachtungen sicher, daß der Leitkoeffizient a_n des Polynoms f ein positives Vorzeichen besitzt, so gilt:

$$\begin{aligned} sign(S_i') &= sign(a_n^{i(i-1)} \cdot S_i) \\ &= sign(a_n^{i(i-1)}) \cdot sign(S_i) \\ &= sign(S_i) \end{aligned}$$

■

Analog zu dieser Vorgehensweise zur Eliminierung der Nenner in den s_k, beseitigen wir auch die Nenner in den h_k:

$$\begin{aligned} h_k' &= a_n^{k+m} h_k \\ &= b_m s_{k+m} a_n^{k+m} + b_{m-1} s_{k+m-1} a_n^{k+m} + \ldots + b_0 s_k a_n^{k+m} \\ &= b_m s_{k+m}' + b_{m-1} s_{k+m-1}' a_n^1 + \ldots + b_0 s_k' a_n^m \end{aligned} \tag{3.11}$$

Definiert man analog zur modifizierten Hankelmatrix $\mathbf{S}'$ die Determinanten der Teilmatrizen von $\mathbf{H}'$ als

$$H_i' = det \begin{bmatrix} h_0' & h_1' & \ldots & h_{i-1}' \\ h_1' & h_2' & \ldots & h_i' \\ \vdots & \vdots & \vdots & \vdots \\ h_{i-1}' & h_i' & \ldots & h_{2i-2}' \end{bmatrix} \tag{3.12}$$

so gilt:

$$\begin{aligned} H_i' &= \sum_{\pi \in \sigma_i} (-1)^{\sigma(\pi)} \cdot h_{0\pi(0)}' \cdot h_{1\pi(1)}' \cdot \ldots \cdot h_{i-1\pi(i-1)}' \\ &= \sum_{\pi \in \sigma_i} (-1)^{\sigma(\pi)} \cdot a_n^{0+\pi(0)+m} h_0 \cdot \ldots \cdot a_n^{i-1+\pi(i-1)+m} h_{i-1} \\ &= a_n^{\sum_{j=0}^{i-1} 2j+m} \cdot \sum_{j=0}^{i-1} (-1)^{\sigma(\pi)} \cdot h_{j\pi(j)} \\ &= a_n^{im+i(i-1)} \cdot H_i \end{aligned}$$

Mit positivem Leitkoeffizient a_n des Polynoms f gilt auch hier:

$$sign(H_i') = sign(H_i)$$

Da die Vorzeichen der Determinanten S_i und H_i nach einer Modifikation der s_k und h_k – sofern der Leitkoeffizient a_n des Polynoms f positiv ist – erhalten bleiben, können die Sätze von Jacobi und Sylvester/Hermite auch auf die Folge der modifizierten Determinanten S_i' und H_i' angewendet werden.

Kehren wir nun zu unserem Ausgangsproblem, von einem Gleichungssystem der Art 3.1 entscheiden zu müssen, ob es im Intervall $(0, 1)$ eine Lösung besitzt, zurück: 3.6

gibt uns hierbei die Möglichkeit, dieses komplexe Problem dahingehend zu vereinfachen, daß wir die Anzahl seiner Lösungen schrittweise durch Betrachtung reduzierter Gleichungssysteme mit nur einer Nebenbedingung berechnen können. Wie dies geschieht, haben wir in diesem Abschnitt erfahren. Da für unsere Zwecke lediglich Lösungen der Gleichungssysteme im Intervall $(0,1)$ von Interesse sind, die Sätze von Jacobi und Sylvester/Hermite jedoch Aussagen über die Anzahl aller reellen Nullstellen eines solchen reduzierten Gleichungssystems machen, müssen wir das betrachtete Intervall noch auf $(0,1)$ einschränken: das erreichen wir durch Hinzufügen der zusätzlichen Nebenbedingung $g_{l+1}(t) = -t^2 + t$ zum Gleichungssystem 3.1. Dieses Polynom besitzt lediglich im Intervall $(0,1)$ positive Funktionswerte, so daß in Gleichung 3.6 nur dort die Anzahl der Nullstellen des erweiterten Systems erfaßt werden.

Gehen wir nun davon aus, daß im Gleichungssystem 3.1 bereits eine polynomielle Nebenbedingung enthalten ist, die das betrachtete Intervall – wie oben beschrieben – begrenzt, so lautet die algorithmische Umsetzung von 3.6 zur Bestimmung der Anzahl seiner Lösungen im Intervall $(0,1)$ wie folgt:

```
int Pedersen(Polynom f,Polynom g[m])
begin
    lk = Leitkoeffizient(f);
    n = Grad(f);
    nullstellen = 0;
    S' = Hankelmatrix(f);
    for (i = 1 to n) do
        S'_i = Determinante(Hauptminor(i, S'));
    od
    P = Potenzmenge(IN_l);
    forall (M ∈ P) do
        nebenbedingung = 1;
        forall (i ∈ M) do
            nebenbedingung = nebenbedingung * g[i];
        od
        H' = Hankelmatrix(nebenbedingung);
        for (i = 1 to n) do
            H'_i = Determinante(Hauptminor(i, H'));
        od
        for (i = 1 to n) do
            f_Hauptminor[i] = signum(S'_i(lk));
            g_Hauptminor[i] = signum(H'_i(lk));
        od
        nullstellen = nullstellen + Rang(S')
                      − var(f_Hauptminor) − var(g_Hauptminor);
        for (i = 1 to Rang(S') step 2) do
            g_Hauptminor[i] = − g_Hauptminor[i];
        od
```

nullstellen = nullstellen − (Rang($\mathbf{S}'$)
− var(f_Hauptminor) − var(g_Hauptminor));
od
return (2^{-l} · nullstellen);
end

Die Funktion Hankelmatrix(...) baut die zu dem übergebenen Polynom gehörige Hankelmatrix auf. Die Funktion var(...) berechnet die Anzahl der Vorzeichenwechsel einer vorgegebenen Zahlenfolge.

Anhand der Definition der h'_k in Gleichung 3.11 überlegt man sich leicht, daß die zu einer Nebenbedingung $g(t) < 0 \iff -g(t) > 0$ gehörige Hankelmatrix **H**' aus der zur Nebenbedingung $g(t) > 0$ gehörigen durch Negation sämtlicher Matrixeinträge hervorgeht. Daraus läßt sich zwischen den Hauptminoren beider Nebenbedingungen eine Korrespondenz herstellen, die zu Laufzeitoptimierungen genutzt werden kann. Sei n der Grad der polynomiellen Hauptbedingung f, $H'_i(g)$, $1 \leq i \leq n$, der gemäß 3.12 definierte, zur polynomiellen Nebenbedingung $g(t) > 0$ gehörige, i-te Hauptminor; analog: $H'_i(-g)$. Dann gilt:

$$H'_i(-g) = (-1)^i \cdot H'_i(g)$$

Hat man die Folge $H'_i(g)$, $1 \leq i \leq n$, berechnet, ergibt sich die Folge $H'_i(-g)$ aus dieser durch Negation an den ungeraden Positionen.

3.1.3 Komplexitätsanalyse

Wie wir zu Beginn dieses Kapitels bereits angedeutet haben, wird die Laufzeit des Verfahrens von mehreren Faktoren beeinflußt:

- der Komplexität des zu untersuchenden Gleichungssystems, d.h. der Anzahl der zu erfüllenden Nebenbedingungen, die zwischen den einzelnen Kollisionstests variiert
- dem Grad der Polynome, von denen die Größe der zu berechnenden Matrizen und somit die Anzahl der zu berechnenden Hauptminoren abhängig ist
- den während der Berechnung sich ergebenden Zahlenlängen, da die Multiplikation zweier L-Bit Zahlen bestenfalls in Zeit $\Omega(L \, log \; L)$ geschehen kann

3.1.3.1 Anzahl der Nebenbedingungen

Bis auf den Punkt-Fläche-Test lassen sich alle übrigen Kollisionstests durch eine konstante Anzahl von Nebenbedingungen beschreiben; um Aussagen über die Komplexität des Verfahrens machen zu können, gehen wir davon aus, daß die konvexen Flächen unserer Objekte aus maximal 4 Kanten bestehen (ansonsten Triangulierung). Diese Annahme gewährleistet, daß in den einzelnen Teiltests maximal 4 polynomielle Nebenbedingungen auftreten können. Da die Sätze von Jacobi und Sylvester/Hermite

Aussagen über alle reellen Nullstellen eines Gleichungssystems der Art 3.2 machen, bedarf es beim Pedersen-Verfahren einer zusätzlichen polynomiellen Nebenbedingung vom Grad 2, die das betrachtete Intervall auf $(0, 1)$ beschränkt. Somit erreichen wir im Kante-Kante-Test die Maximalzahl von 5 Nebenbedingungen.

3.1.3.2 Grad der Polynome

Wie bereits in einem früheren Kapitel erwähnt, begnügen wir uns mit linearer Interpolation zwischen zeitlich aufeinanderfolgenden Konfigurationen $\mathcal{C}_i(k)$ und $\mathcal{C}_i(k+1)$ um die Konfiguration zum Zeitpunkt $\mathcal{C}_i(k+t)$ zu berechnen (siehe Modellbeschreibung). Hierdurch erreichen die zu lösenden Gleichungssysteme einen niedrigeren Grad:

- Kante-Kante-Test:
  eine Hauptbedingung vom Grad 5, vier Nebenbedingungen vom Grad 5 und eine Nebenbedingung vom Grad 2
- Punkt-Kante-Test:
  eine Hauptbedingung vom Grad 5, zwei Nebenbedingungen vom Grad 5 und eine Nebenbedingung vom Grad 2
- Punkt-Fläche-Test:
  eine Hauptbedingung vom Grad 5, drei oder vier Nebenbedingungen vom Grad 5 und eine Nebenbedingung vom Grad 2
- Punkt-Punkt-Test:
  eine Hauptbedingung vom Grad ≤ 5 und eine Nebenbedingung vom Grad 2

Im Verlauf des Verfahrens werden über der Potenzmenge der Menge der Nebenbedingungen die entsprechenden Produkte gebildet, was im Fall des Kante-Kante-Tests zu einer polynomiellen Nebenbedingung vom Grad 22 führt.

3.1.3.3 Zahlenlängen während Zwischenberechnungen

Entscheidend für die praktische Anwendbarkeit dieses Verfahrens ist die Größe der entstehenden Zwischenergebnisse, die bei einer Implementierung die Laufzeit des Verfahrens stark beeinflußt. In Abhängigkeit von der Größe der Eingabe – der Bitlänge der Koeffizienten der Polynome f und g also – werden wir die Länge der zu berechnenden Determinanten abschätzen. Mit L_a bezeichnen wir hierbei die Länge der Koeffizienten des Polynoms f, mit L_b analog die Länge der Koeffizienten des Polynoms g.

Berechnung der Determinanten der Hankelmatrix S' Durch die im vorigen Abschnitt beschriebene Modifikation der Newtonschen Summen s_k verlieren diese ihren Nenner und lassen sich als Polynome über dem Leitkoeffizienten a_n auffassen. Desweiteren möchten wir bemerken, daß eine Anwendung des Verfahrens erst sinnvoll wird, wenn für den Grad n der polynomiellen Hauptbedingung f gilt: $n \geq 2$.

Satz 3.6 :
Sei $f(t) = \sum_{i=0}^{n} a_i t^i = 0,\ n \geq 2$, die polynomielle Hauptbedingung eines Gleichungssystems der Art 3.2. Die Koeffizienten a_i seien ganzzahlig und von der maximalen Länge L_a. Für die Länge der gemäß 3.9 modifizierten Newtonschen Summen s'_k gilt:

$$|s'_k| \quad \leq \quad k \cdot L_a + k \cdot log\, n \tag{3.13}$$

Beweis:
Beweis durch vollständige Induktion über k

- $\underline{k = 2:}$
  $s'_2 = -(a_{n-1}s'_1 + 2a_{n-2}a_n^1) = -2a_{n-2}a_n^1 + a_{n-1}^2 a_n^0$
  $|s'_2| \leq 2L_a + log\, 3 \leq 2L_a + 2 \leq 2L_a + 2 \cdot log\, n$

- $\underline{k \rightarrow k+1:}$
  An der Definition 3.9 der s'_k läßt sich erkennen, daß diese für $k \leq n$ aus k Summanden, für $k > n$ aus n Summanden bestehen. Zur Abschätzung ihrer Länge ist es daher ausreichend, den Fall $k > n$ zu betrachten:

  $$s'_{k+1} \quad = \quad -(a_{n-1}s'_k + a_{n-2}s'_{k-1}a_n^1 + \ldots + a_0 s'_{k-n+1}a_n^{n-1})$$

  Jeder einzelne Summand wiederum wird durch ein Produkt aus einem Koeffizienten a_{n-j}, einem s'_{k-j+1} und einer j-ten Potenz des Leitkoeffizienten a_n gebildet. Für die Länge eines jeden solchen Summanden gilt:

  $$\begin{aligned} |a_{n-j}s'_{k-j+1}a_n^{j-1}| &= |a_{n-j}| + |s'_{k-j+1}| + |a_n^{j-1}| \\ &\leq L_a + (k-j+1) \cdot L_a + (k-j+1) \cdot log\, n + (j-1) \cdot L_a \\ &= (k+1) \cdot L_a + (k-j+1) \cdot log\, n \\ &\leq (k+1) \cdot L_a + k \cdot log\, n \end{aligned}$$

  Für die Länge von s'_{k+1} gilt dann:

  $$\begin{aligned} |s'_{k+1}| &\leq |a_{n-j}s'_{k-j+1}a_n^{j-1}| + log\, n \\ &\leq (k+1) \cdot L_a + k \cdot log\, n + log\, n \\ &= (k+1) \cdot L_a + (k+1) \cdot log\, n \end{aligned}$$

■

Nachdem wir die Länge der Newtonschen Summen s'_k abgeschätzt haben, können wir die Länge der Determinante der gesamten Hankelmatrix $\mathbf{S}'$ berechnen:

Satz 3.7 :
Sei $f(t) = \sum_{i=0}^{n} a_i t^i$ ein Polynom mit ganzzahligen Koeffizienten a_i, deren maximale Länge mit L_a bezeichnet werde. Für die Länge der gemäß 3.10 modifizierten Determinante der Hankelmatrix $\mathbf{S}'$ gilt:

$$|S'_n| \quad = \quad O(n^2 \cdot (L_a + log\, n)) \tag{3.14}$$

Beweis:

$$\begin{aligned} S'_n &= \sum_{\pi \in \sigma_n} (-1)^{\sigma(\pi)} \prod_{i=0}^{n-1} s'_{i\pi(i)} \\ &= \sum_{\pi \in \sigma_n} (-1)^{\sigma(\pi)} \prod_{i=0}^{n-1} s'_{i+\pi(i)} \end{aligned}$$

Die äußere Summe, die über alle Permutationen der Gruppe σ_n summiert, besitzt $n!$ Summanden. Jeder Summand besteht hierbei aus einem Produkt von n Faktoren. Für die Länge der Determinante gilt also:

$$\begin{aligned} |S'_n| &\leq \sum_{i=0}^{n-1} |s'_{i+\pi(i)}| + log\,(n!) \\ &= \left(\sum_{i=0}^{n-1} (i+\pi(i)) \cdot L_a + (i+\pi(i)) \cdot log\, n \right) + log\,(n!) \\ &= \left(\sum_{i=0}^{n-1} 2 \cdot i \cdot (L_a + log\, n) \right) + log\,(n!) \\ &= (n^2 - n)(L_a + log\, n) + log\,(n!) \\ &= O(n^2 \cdot (L_a + log\, n)) \end{aligned}$$

■

Berechnung der Determinanten der Hankelmatrix H' Um die Länge der Determinante der Hankelmatrix **H**' zu berechnen, müssen wir zuerst die Länge ihrer Einträge abschätzen:

Satz 3.8:
Sei $g(t) = \sum_{i=0}^{m} b_i t^i > 0$ die polynomielle Nebenbedingung eines Gleichungssystems der Art 3.2. Die Koeffizienten b_i seien ganzzahlig und von der maximalen Länge L_b. Für die Länge der gemäß 3.11 modifizierten Newtonschen Summen h'_k gilt:

$$|h'_k| \leq L_b + (k+m) \cdot (L_a + log\, n) + log\,(m+1) \tag{3.15}$$

Beweis:
Ein Term $h'_k = b_m s'_{k+m} + b_{m-1} s'_{k+m-1} a_n^1 + \ldots + b_0 s'_k a_n^m$ besteht aus $(m+1)$ Summanden. Die Länge eines solchen Summanden läßt sich wie folgt bestimmen ($0 \leq j \leq m$):

$$\begin{aligned} |b_{m-j} s'_{k+m-j} a_n^j| &= |b_{m-j}| + |s'_{k+m-j}| + |a_n^j| \\ &\leq L_b + (k+m-j) \cdot (L_a + log\, n) + j \cdot L_a \\ &= L_b + (k+m) \cdot L_a + (k+m-j) \cdot log\, n \\ &\leq L_b + (k+m) \cdot (L_a + log\, n) \end{aligned}$$

Für die Länge des gesamten Terms h'_k gilt dann:

$$\begin{aligned} |h'_k| &\leq |b_{m-j} s'_{k+m-j} a_n^j| + log\,(m+1) \\ &\leq L_b + (k+m) \cdot (L_a + log\, n) + log\,(m+1) \end{aligned}$$

■

Durch die Längenabschätzung sind wir nun in der Lage, die Länge der Determinante der Hankelmatrix **H**' abzuschätzen:

Satz 3.9:
Sei $g(t) = b_m t^m + b_{m-1} t^{m-1} + \ldots + b_1 t^1 + b_0$ ein Polynom mit ganzzahligen Koeffizienten b_i, deren maximale Länge mit L_b bezeichnet werde. Für die Länge der gemäß 3.12 modifizierten Determinante der Hankelmatrix H'_n gilt:

$$|H'_n| = O(n(n+m)(L_a + L_b + log\,(n+m))) \tag{3.16}$$

Beweis:

$$\begin{aligned} H'_n &= \sum_{\pi \in \sigma_n} (-1)^{\sigma(\pi)} \prod_{i=0}^{n-1} h'_{i\pi(i)} \\ &= \sum_{\pi \in \sigma_n} (-1)^{\sigma(\pi)} \prod_{i=0}^{n-1} h'_{i+\pi(i)} \end{aligned}$$

Die äußere Summe, die über alle Permutationen der Gruppe σ_n summiert, besitzt $n!$ Summanden. Jeder Summand besteht hierbei aus einem Produkt von n Faktoren. Für die Länge der Determinante gilt also:

$$\begin{aligned} |H'_n| &\leq \sum_{i=0}^{n-1} |h'_{i+\pi(i)}| + log\,(n!) \\ &= \Big(\sum_{i=0}^{n-1} (L_b + (i + \pi(i) + m) \cdot (L_a + log\, n) + log\,(m+1) \Big) + log\,(n!) \\ &= \Big(\sum_{i=0}^{n-1} L_b + \sum_{i=0}^{n-1} (2i + m) \cdot (L_a + log\, n) + \sum_{i=0}^{n-1} log\,(m+1) \Big) + log\,(n!) \\ &= n \cdot L_b + (n^2 - n + nm) \cdot (L_a + log\, n) + n \cdot log\,(m+1) + log\,(n!) \\ &= O(n(n+m)(L_a + log\,(n+m)) + nL_b) \\ &= O(n(n+m)(L_a + L_b + log\,(n+m))) \end{aligned}$$

■

3.1.3.4 Gesamtlaufzeit

Das Pedersen-Verfahren beruht zunächst auf der Reduktion der Anzahl der Nebenbedingungen gemäß Gleichung 3.6. Anstatt von einem Gleichungssystem der Form 3.1 die Anzahl seiner Lösungen zu bestimmen, wird hierbei von 2^l Systemen der Form 3.2 die Anzahl ihrer Lösungen berechnet. Um hierfür den Satz von Sylvester/Hermite verwenden zu können, müssen die Vorzeichen sämtlicher Hauptminoren der beiden $(n \times n)$-Hankelmatrizen $\mathbf{S}'$ und $\mathbf{H}'$ berechnet werden (n ist hierbei der Grad der polynomiellen Hauptbedingung f). Die Berechnung der Determinante einer $(n \times n)$-Matrix nach dem *Gaußschen* Eliminationsverfahren benötigt $O(n^3)$ arithmetische Operationen. Wegen der Ganzzahligkeit unserer Berechnungen hat sich die Verwendung dieses Verfahrens jedoch nicht bewährt: eine Multiplikation der einzelnen Zeilen mit den zugehörigen Hauptnennern, die man sich entsprechend merkt, läßt die Größe der Zwischenergebnisse erheblich anwachsen. Diese Multiplikationen akkumulieren sich von Zeile zu Zeile und erlangen ihr Maximum beim letzten Matrizenelement. In Kapitel 4 beschreiben wir daher eine alternative Vorgehensweise, die ohne Division auskommt und für unsere speziellen Determinanten von (5×5)-Matrizen in etwa so viele Operationen benötigt wie das Gaußsche Eliminationsverfahren.

Bei der Laufzeitanalyse des Verfahrens legen wir den in [Yap94] beschriebenen, auf *Bareiss* zurückgehenden, modifizierten Gaußschen Algorithmus zur ganzzahligen Determinantenberechnung zugrunde. Dessen Komplexität liegt für eine $(n \times n)$-Matrix bei

$$\mathit{Cost(Bareiss)} \;=\; O\Big(n^3 \cdot \mu(n(\log n + L))\Big)$$

wobei $\mu(L)$ die für die Multiplikation zweier L-Bit Zahlen benötigte Zeit berechnet. Eine obere Schranke hierfür ist $O(L \cdot \log L \cdot \log\log L)$. Da die Matrix $\mathbf{S}'$ für alle 2^l Durchläufe nur einmal berechnet und ausgewertet werden muß, die Matrizen $\mathbf{H}'$ jedoch variieren, liegt der Laufzeitschwerpunkt in der Berechnung dieser Hauptminoren. Unter den Annahmen

- $m \leq l \cdot n + 2$
- $L_b = O(l \cdot (L_a + \log l + \log n)) = O(l \cdot (L + \log l + \log n))$
- $\log n \leq L$ (L bezeichnet die Länge der Eingabedaten)

und der Bezeichnung L_w für die im *worst-case* auftretenden Bitlängen der Matrixeinträge von $\mathbf{H}'$ gilt:

$$\begin{aligned}
L_w &= |h'_{2n-2}| \\
&= O\Big(L_b + (2 \cdot n - 2 + m) \cdot (L_a + \log n) + \log(m+1)\Big) \\
&= O\Big(l \cdot (L + \log l + \log n) + (n \cdot (l+2)) \cdot (L + \log n) + \log(l \cdot n + 3)\Big) \\
&= O\Big(l \cdot L \cdot n + l \cdot \log l\Big) \\
&= O(l \cdot (L \cdot n + \log l))
\end{aligned}$$

Die Laufzeit des Pedersen-Verfahrens berechnet sich damit wie folgt:

$$\begin{aligned} T_{Ped} &= \#(Iterationen) \cdot Cost(Bareiss) \\ &= O\Big(2^l \cdot n^3 \cdot \mu(n \cdot (log\, n + L_w))\Big) \\ &= O\Big(2^l \cdot n^3 \cdot \mu(n \cdot (log\, n + l \cdot L \cdot n + l \cdot log\, l))\Big) \\ &= O\Big(2^l \cdot n^3 \cdot \mu(l \cdot L \cdot n^2 + l \cdot log\, l \cdot n)\Big) \end{aligned}$$

Auf eine weitere Expansion dieses Ausdrucks durch Auswertung der Funktion μ möchten wir aus Gründen der Übersichtlichkeit verzichten und werden ihn deshalb auch im folgenden in der obigen, kompakten Darstellung verwenden.

3.2 Das Verfahren von Uspensky

In diesem Abschnitt werden wir eine Vorgehensweise zur *exakten* Lösung des Entscheidungsproblems 2.1 angeben, die auf der *exakten* Separierung der positiven Nullstellen eines Polynoms $A(x)$ basiert. Diese Separierung geschieht durch ein rekursives Verfahren, das wir im folgenden näher untersuchen werden und dessen Flußdiagramm in Abbildung 3.3 zu sehen ist.

3.2.1 Ein Verfahren zur Nullstellenseparierung

Der Ursprung des hier vorgestellten rekursiven Verfahrens zur Nullstellenseparierung eines Polynoms mit ganzzahligen Koeffizienten, das in [CL82] zum ersten Mal erschienen ist, geht auf *Uspensky* zurück; eine Laufzeitverbesserung erfolgte durch *Collins/Akritas*.

Gegeben sei im folgenden ein Polynom $A(x) = \sum_{i=0}^{n} a_i x^i$ mit $a_i \in \mathbb{Z}$ (oder $a_i \in \mathbb{Q}$), von der Art also, wie die Polynome unserer polynomiellen Gleichungssysteme des vorigen Kapitels. Unser Ziel ist es, die Nullstellen dieses Polynoms im Intervall $(0, 1)$, über dem die Bewegungen unserer Objekte zeitlich parametrisiert sind, zu separieren, d.h. eine vollständige Liste $L_{sep}^{A((0,1))}$ disjunkter Teilintervalle mit rationalen Intervallgrenzen zu berechnen, von denen jedes Teilintervall genau eine reelle Nullstelle des Polynoms $A(x)$ enthält. Eine solche Separierung ist jedoch nur dann möglich, wenn $A(x)$ keine mehrfachen Nullstellen besitzt; dies ist gleichbedeutend mit der Tatsache, daß der minimale Abstand zweier Nullstellen des Polynoms $A(x)$, den man mit $sep(A)$ bezeichnet, größer Null sein muß: dieser minimale Abstand läßt sich laut [CL82] abschätzen durch

$$sep(A) \;>\; \frac{1}{2}(e^{1/2}\, n^{3/2}\, d)^{-n}\ , \tag{3.17}$$

wobei mit

$$\begin{aligned} d &= d_A = \Big(\sum_{i=0}^{n} a_i^2\Big)^{1/2} \\ &= O(L_a + log\, n) \end{aligned}$$

die Semi-Norm des Polynoms $A(x)$ bezeichnet sei. L_a ist hierbei die Länge der Polynomialkoeffizienten a_i. Die zur Darstellung der Semi-Norm benötigte Stellenzahl, die wir im Verlauf der Arbeit noch mehrfach benötigen werden, berechnet sich zu:

$$\begin{aligned} L_d &= log\, d \\ &= O(log(L_a + log\, n)) \end{aligned}$$

Entscheidend für die Berechnung ist ein Satz von Descartes und ein hieraus resultierendes Korollar, das die Abbruchbedingungen für das rekursive Verfahren bereitstellt.

Satz 3.10 (Descartes)
Sei $A(x) = \sum_{i=0}^{n} a_i x^i$ *ein Polynom mit ganzzahligen Koeffizienten und* $p(A) = \#\{x \in \mathbb{R}^+ \mid A(x) = 0\}$ *die Anzahl der positiven Nullstellen von* $A(x)$*. Dann gilt:*

$$\exists k \in \mathbb{N}_0 \; mit \; var(a_n, a_{n-1}, \ldots, a_0) \;=\; p(A) + 2k$$

Korollar 3.11:
(1) $var(A) = 0 \Rightarrow p(A) = 0$
(2) $var(A) = 1 \Rightarrow p(A) = 1$

Während der Satz von Descartes und das daraus resultierende Korollar Aussagen über *alle positiven* (reellen) Nullstellen eines Polynoms $A(x)$ machen, zur Lösung des Entscheidungsproblems jedoch nur solche aus dem Intervall $(0, 1)$ interessant sind, müssen wir nach einem geeigneten Weg suchen, um uns diese mathematischen Hilfsmittel zu Nutze machen zu können. Dabei ist uns eine Transformation behilflich, die den für uns interessanten Definitionsbereich $(0, 1)$ des Polynoms $A(x)$ auf den Bereich $(0, \infty)$ streckt und die somit am Anfang unseres rekursiven Verfahrens steht.

Lemma 3.12:
Sei $A(x) = \sum_{i=0}^{n} a_i x^i$ *ein Polynom mit ganzzahligen Koeffizienten und* $A^*(y) := (y+1)^n \cdot A(\frac{1}{y+1})$*. Dann gilt:*

1. *die Koeffizienten des Polynoms* $A^*(y)$ *sind wieder ganzzahlig*

2. *reelle Nullstellen des Polynoms* $A(x)$ *aus dem Intervall* $(0, 1)$ *werden zu positiven Nullstellen des Polynoms* $A^*(y)$

3. *reelle Nullstellen des Polynoms* $A(x)$ *aus dem Intervall* $[1, \infty)$ *werden zu Nullstellen des Polynoms* $A^*(y)$ *im Intervall* $(-1, 0]$

4. *reelle Nullstellen des Polynoms* $A(x)$ *aus dem Intervall* $(-\infty, 0)$ *werden zu Nullstellen des Polynoms* $A^*(y)$ *im Intervall* $(-\infty, -1)$

Beweis:
Für das transformierte Polynom $A^*(y)$ gilt:

$$\begin{aligned}
A^*(y) &= (y+1)^n \cdot A(\frac{1}{y+1}) \\
&= (y+1)^n \cdot \sum_{k=0}^{n} a_k \Big(\frac{1}{y+1}\Big)^k \\
&= \sum_{k=0}^{n} a_k (y+1)^{n-k} \\
&= \sum_{k=0}^{n} a_{n-k} (y+1)^k \\
&= \sum_{k=0}^{n} a_{n-k} \sum_{i=0}^{k} \binom{k}{i} y^{k-i} \\
&= \sum_{k=0}^{n} \underbrace{\left(\sum_{i=k}^{n} a_{n-i} \binom{i}{k} \right)}_{\in \mathbb{Z}} y^k
\end{aligned}$$

Damit ist gezeigt, daß die Koeffizienten von $A^*(y)$ ganzzahlig sind. Die Verschiebung der Nullstellen des Polynoms $A(x)$ zu Nullstellen des Polynoms $A^*(y)$ vollzieht sich gemäß folgender Substitution:

$$x = \frac{1}{y+1} \iff y = \frac{1}{x} - 1$$

Anhand dieser Substitution lassen sich folgende Korrespondenzen der Definitionsbereiche nachvollziehen:

- $x \in (0,1) \rightarrow y \in (0,\infty)$
- $x \in [1,\infty) \rightarrow y \in (-1,0]$
- $x \in (-\infty,0) \rightarrow y \in (-\infty,-1)$

Damit ist obiges Lemma bewiesen. ∎

Ergänzend sei noch erwähnt, daß die Anwendung der Transformation die komplexen Nullstellen des Polynoms $A(x)$ in analoger Weise bzgl. des Einheitskreises verschiebt. Diese Tatsache wird im späteren Verlauf Verwendung finden, um die Terminierung des Verfahrens zu zeigen. Abbildung 3.2 zeigt die Wirkungsweise dieser Transformation an einem einfachen Beispiel.

Die Transformation $A^*(y)$ und die Folgerungen aus dem Satz von Descartes ermöglichen es uns, in den beiden Fällen $var(A^*) = 0$ und $var(A^*) = 1$ Aussagen über das Vorhandensein von Nullstellen des Ausgangspolynoms $A(x)$ im uns interessierenden Intervall $(0,1)$ zu machen: im ersten Fall ergibt sich als Resultat die leere Liste, im zweiten

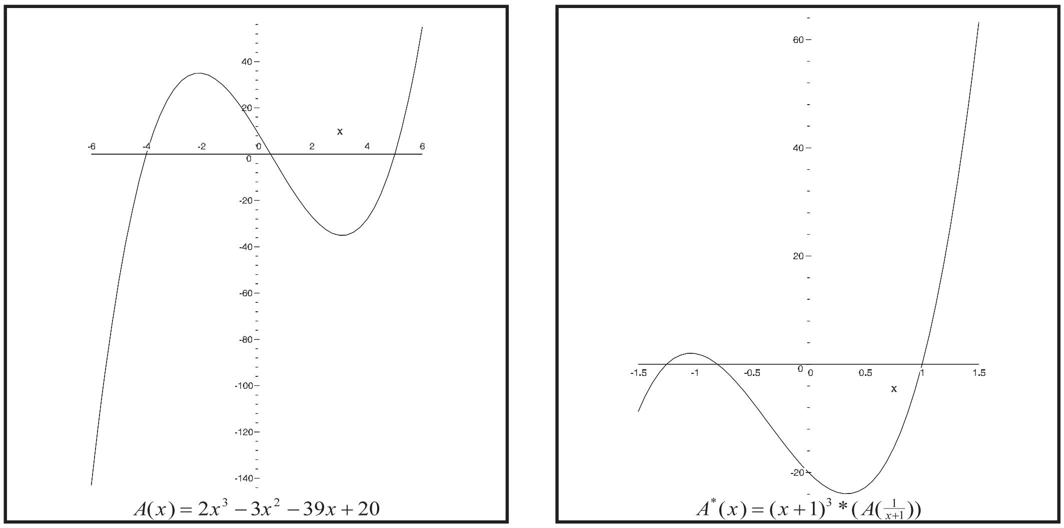

Abbildung 3.2: Nullstellenverschiebung durch Anwendung der Transformation A^*

Fall die Liste, die das Intervall $(0, 1)$ enthält. In den übrigen Fällen $(var(A^*) > 1)$ kann allein durch Vorzeichenbetrachtung der Koeffizienten von $A^*(y)$ keine Aussage über die Anzahl der im Intervall vorhandenen Nullstellen gemacht werden: die Sichtweise hierfür ist sozusagen noch zu grob. Aus diesem Grund greifen wir auf das *Divide-and-Conquer-Prinzip* zurück: wir verfeinern den betrachteten Definitionsbereich $(0, 1)$ des Polynoms $A(x)$ durch rekursiv fortgesetzte Intervallhalbierung solange, bis wir mittels Korollar 3.11 entscheiden können, ob er eine Nullstelle enthält oder nicht. Um unsere Berechnungen auch nach einer Intervallhalbierung weiterhin ausschließlich mit offenen Intervallen fortführen zu können, untersuchen wir das Verhalten des Polynoms $A(x)$ im Mittelpunkt des Ausgangsintervalles gesondert. Da ein Intervallhalbierungsschritt eine Einschränkung des uns interessierenden Definitionsbereiches vom Intervall $(0, 1)$ auf die Teilintervalle $(0, \frac{1}{2})$ und $(\frac{1}{2}, 1)$ bedeutet, ist es erforderlich, die Voraussetzungen für einen rekursiven Funktionsaufruf zu schaffen: die Nullstellen des Polynoms $A(x)$ aus der linken und rechten Intervallhälfte müssen jeweils zu Nullstellen im Intervall $(0, 1)$ transformiert werden. Wir erreichen dies, indem wir das Eingabepolynom $A(x)$ in zwei Polynome $A_{left}(x)$ und $A_{right}(x)$ transformieren, für die gilt:

- Nullstellen von $A(x)$ aus dem Teilintervall $(0, \frac{1}{2})$ werden zu Nullstellen von $A_{left}(x)$ im Intervall $(0, 1)$.
- Nullstellen von $A(x)$ aus dem Teilintervall $(\frac{1}{2}, 1)$ werden zu Nullstellen von $A_{right}(x)$ im Intervall $(0, 1)$.

Wie die Transformationen $A_{left}(x)$ und $A_{right}(x)$ geartet sein müssen, können wir dem folgenden Lemma entnehmen:

Lemma 3.13:
Sei $A(x) = \sum_{i=0}^{n} a_i x^i$ *ein Polynom mit ganzzahligen Koeffizienten.*

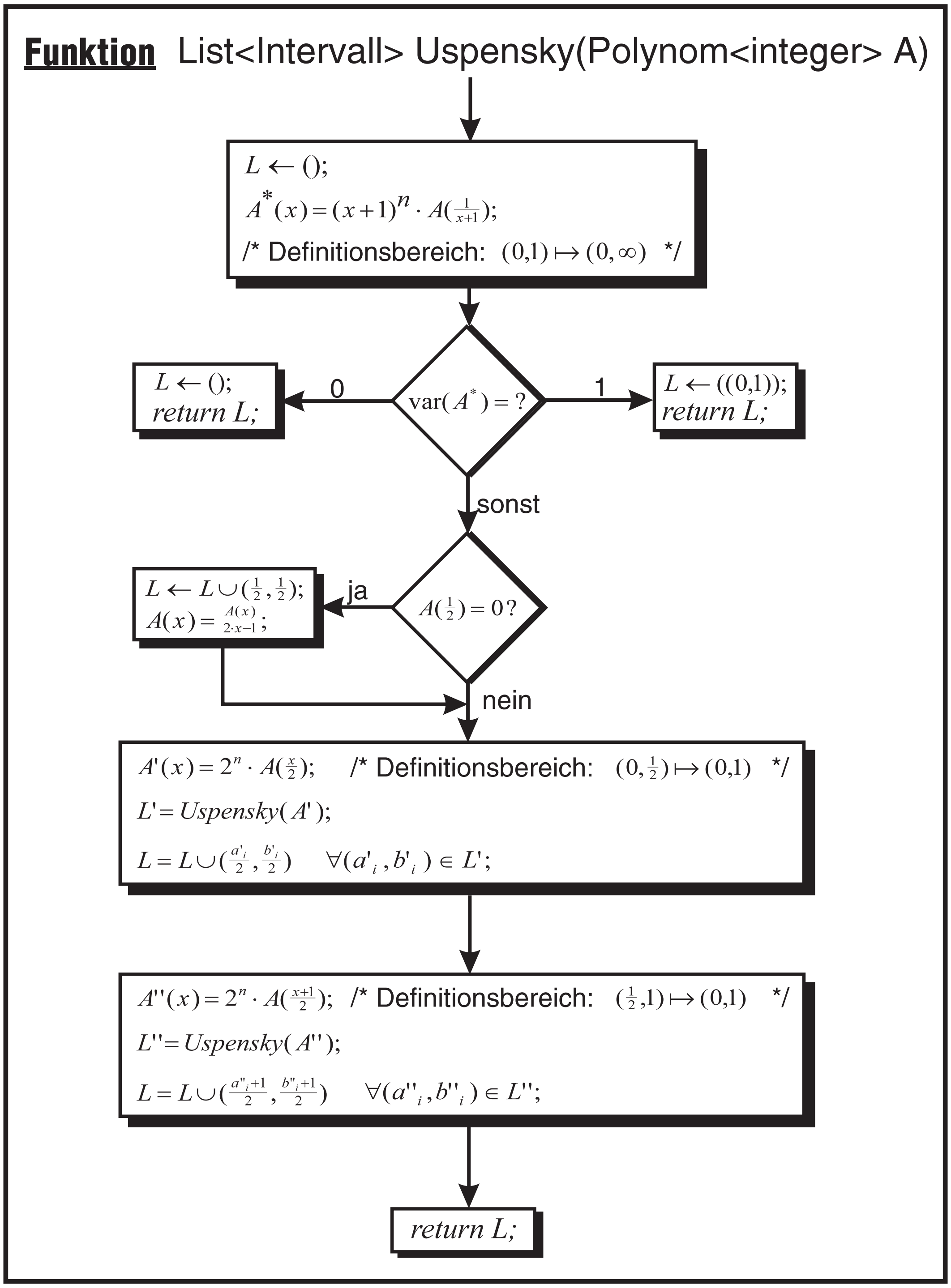

Abbildung 3.3: Flussdiagramm des modifizierten Uspensky-Verfahrens

1. *Das Polynom $A_{left}(x) = 2^n A(\frac{x}{2})$ besitzt nur ganzzahlige Koeffizienten; Nullstellen von $A(x)$ aus dem Intervall $(0, \frac{1}{2})$ werden auf Nullstellen von $A_{left}(x)$ im Intervall*

$(0,1)$ transformiert.

2. *Das Polynom $A_{right}(x) = 2^n A(\frac{x+1}{2})$ besitzt nur ganzzahlige Koeffizienten; Nullstellen von $A(x)$ aus dem Intervall $(\frac{1}{2},1)$ werden auf Nullstellen von $A_{right}(x)$ im Intervall $(0,1)$ transformiert.*

Aus den bisher vorgestellten Sätzen ergibt sich die Korrektheit des vorgestellten rekursiven Verfahrens im Falle der Terminierung: der Satz von Descartes, der unter gewissen Umständen Aussagen über die Anzahl der positiven reellen Nullstellen eines Polynoms macht, bildet hierbei das Herzstück. Damit dieses Verfahren nach einer gewissen (endlichen) Anzahl von Rekursionsschritten wirklich abbricht, muß irgendwann einer der Fälle $var(A^*) = 0$ oder $var(A^*) = 1$ eintreten. Um einzusehen, daß dies auch tatsächlich der Fall ist, betrachten wir die folgenden, in [CL82] bewiesenen Sätze:

Satz 3.14:
Sei $A(x) = \sum_{i=0}^{n} a_i x^i$ ein von Null verschiedenes Polynom vom Grad $n \geq 2$. Dann gelten folgende Aussagen:

1. *Besitzt $A(x)$ keine Nullstellen in der rechten Hälfte der komplexen Ebene, dann gilt $var(A) = 0$.*

2. *Besitzt $A(x)$ eine Nullstelle im Intervall $(0,1)$ und keine weitere in einem Kreis um den Ursprung mit Radius $r_n = O(n^2)$, dann gilt $var(A^*) = 1$.*

Führen wir uns die Wirkungsweise der Hintereinanderausführung $(A_{side} \circ A^*)$ der in den Lemmata 3.12 und 3.13 beschriebenen Transformationen vor Augen, so erkennen wir, daß diese einen solchen in obigem Satz beschriebenen Zustand herbeiführen. Daß die Anzahl der hierfür benötigten Rekursionen beschränkt ist, ist die Aussage des folgenden Satzes:

Satz 3.15:
Sei $A(x) = \sum_{i=0}^{n} a_i x^i$ ein von Null verschiedenes Polynom vom Grad $n \geq 2$. Mit $T(A)$ werde eine der Transformationen $A_{side}(x)$ mit $side \in \{left, right\}$ bezeichnet. Dann besitzt das Polynom $T^k(A)$ mit $k = O(nL_d)$ höchstens eine Nullstelle λ_n im Intervall $(0,1)$. Alle anderen Nullstellen $\lambda_1, \ldots, \lambda_{n-1}$, ob reell oder komplex, liegen außerhalb des Kreises um den Ursprung mit Radius $r_n = O(n^2)$.

Wie man aus diesem Satz erschließen kann – Anzahl der durchzuführenden Transformationen ist linear in der Anzahl der Nullstellen – werden nur solche Pfade im vollständigen Rekursionsbaum gänzlich durchlaufen, deren Blätter Nullstellen des Polynoms enthalten. Gerade dieser Tatsache ist die Effizienz des Verfahrens zuzuschreiben. Da die Transformationen $A_{side}(x)$ ($side \in \{left, right\}$) einer Verkleinerung des betrachteten Definitionsbereichs des Ausgangspolynoms $A(x)$ entsprechen, bedeutet der obige Satz anschaulich, daß sich die Nullstellen des Polynoms $A(x)$ durch endlichfache Intervallhalbierung in Intervallen geeigneter Länge separieren lassen.

Um die Arbeitsweise des nicht so leicht überschaubaren Algorithmus zu verdeutlichen, möchten wir ein einfaches Beispiel angeben:

Beispiel:
Gegeben sei ein Polynom $A(x) = 10000x^3 - 25000x^2 + 10621x - 1242$, das die Nullstellen 0.23, 0.27 und 2 besitzt. Starten wir den vorgestellten Algorithmus mit diesem Polynom, so ergeben sich folgende Funktionsaufrufe:

<table>
<tr><td colspan="3">Intervall: $(0,1)$</td></tr>
<tr><td colspan="3">$L \leftarrow \emptyset$
$A(x) = 10000x^3 - 25000x^2 + 10621x - 1242$
$A^*(y) = -1242y^3 + 6895y^2 - 7484y - 5621$
$\rightarrow var(A^*) = 2$
$\rightarrow$ rekursive Intervallhalbierung:
$A_{left}(x) = 10000x^3 - 50000x^2 + 42484x - 9936$
$A_{right}(x) = 10000x^3 - 20000x^2 - 27516x - 7452$
$L' = Uspensky(A_{left}(x))$
$L' = \{[0, \frac{1}{2}], [\frac{1}{2}, 1]\}$
$L = L \cup \{[0, \frac{1}{4}], [\frac{1}{4}, \frac{1}{2}]\}$
$L'' = Uspensky(A_{right}(x))$
$L'' = \emptyset$
$L = L \cup \emptyset$
$return\ L$</td></tr>
<tr><td colspan="2">Intervall: $(0, \frac{1}{2})$</td><td>Intervall: $(\frac{1}{2}, 1)$</td></tr>
<tr><td colspan="2">$L \leftarrow \emptyset$
$A(x) = 10000x^3 - 50000x^2 + 42484x - 9936$
$A^*(y) = -9936y^3 + 12676y^2 + 5610y - 7452$
$\rightarrow var(A^*) = 2$
$\rightarrow$ rekursive Intervallhalbierung:
$A_{left}(x) = 10000x^3 - 100000x^2 + 169936x - 79512$
$A_{right}(x) = 10000x^3 - 70000x^2 - 62x + 426$
$L' = Uspensky(A_{left}(x))$
$L' = \{(0,1)\}$
$L = L \cup (0, \frac{1}{2})$
$L'' = Uspensky(A_{right}(x))$
$L'' = \{(0,1)\}$
$L = L \cup (\frac{1}{2}, 1)$
$L = \{[0, \frac{1}{2}], [\frac{1}{2}, 1]\}$
$return\ L$</td><td>$A(x) = 10000x^3 - 20000x^2 - 27516x - 7452$
$A^*(y) = -7452y^3 - 49872y^2 - 97388y - 44968$
$\rightarrow var(A^*) = 0$
$\rightarrow$ Intervall enthält keine Nullstelle
$return(L \leftarrow \emptyset)$</td></tr>
<tr><td>Intervall: $(0, \frac{1}{4})$</td><td>Intervall: $(\frac{1}{4}, \frac{1}{2})$</td><td></td></tr>
<tr><td>$A(x) = 10000x^3 - 100000x^2 + 169936x - 79512$
$A^*(y) = -79512y^3 - 68000y^2 + 1336y + 424$
$\rightarrow var(A^*) = 1$
$\rightarrow$ Intervall enthält genau eine Nullstelle
$return(L \leftarrow (0,1))$</td><td>$A(x) = 10000x^3 - 70000x^2 - 62x + 426$
$A^*(y) = 426y^3 + 1216y^2 - 68846y - 59636$
$\rightarrow var(A^*) = 1$
$\rightarrow$ Intervall enthält genau eine Nullstelle
$return(L \leftarrow (0,1))$</td><td></td></tr>
</table>

3.2.2 Lösung des Entscheidungsproblems mittels Nullstellenseparierung

Nachdem wir im vorigen Abschnitt ein Verfahren kennengelernt haben, mit dem wir die Nullstellen eines vorgegebenen Polynoms $A(x)$ im Intervall $(0,1)$ *exakt* separieren können, steht einer *exakten* Lösung des Entscheidungsproblems 2.1 nichts mehr im Wege. Betrachten wir Abbildung 3.1 genauer, so kann man sich überlegen, daß die grobe Strategie wie folgt aussehen wird:

1. Separierung der Nullstellen der polynomiellen Hauptbedingung $f(t)$. Ergebnis ist eine Liste $L^{sep}_{f(t)}$ von Intervallen $I^{f(t)}_{(a_j,b_j)}$ mit rationalen Endpunkten. Jedes dieser Intervalle enthält genau eine Nullstelle von $f(t)$ aus dem Intervall $(0,1)$.

2. Bestimmung des Verhaltens der polynomiellen Nebenbedingungen $g_i(t)$ in allen Intervallen $I^{f(t)}_{(a_j,b_j)}$ der Liste $L^{sep}_{f(t)}$ durch Auswertung an den *rationalen* Intervallgrenzen.

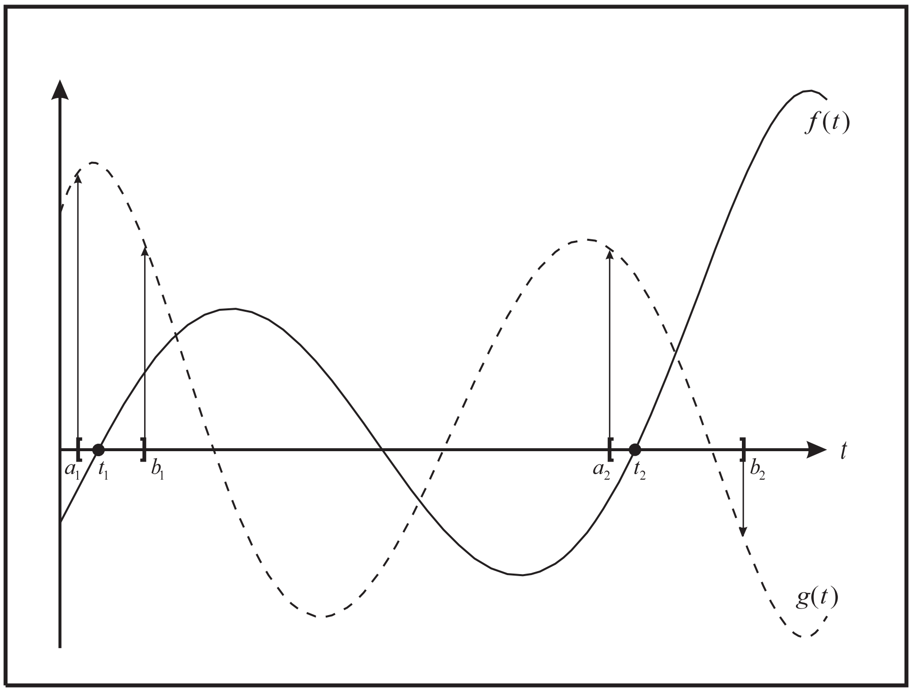

Abbildung 3.4: möglicher Verlauf eines Polynoms in einem Intervall

Damit wir jedoch das Verhalten der $g_i(t)$ in einem Intervall lediglich durch deren Auswertung an den *rationalen* Intervallgrenzen bestimmen können, ist es von entscheidender Bedeutung, daß in einem solchen separierenden Intervall $I^{f(t)}_{(a_j,b_j)}$ keine der Nebenbedingungen einen Vorzeichenwechsel vornimmt. Zur Verdeutlichung der damit verbundenen Problematik sei auf Abbildung 3.4 verwiesen:

- Besitzt $g_i(t)$ in einem separierenden Intervall eine ungerade Anzahl von Vorzeichenwechseln, ergibt die Auswertung an den Intervallgrenzen einen positiven und einen negativen Wert.

- Besitzt $g_i(t)$ in einem separierenden Intervall eine gerade Anzahl von Vorzeichenwechseln, so ergibt eine Auswertung an den Intervallgrenzen entweder zwei positive oder zwei negative Wert.

Über das Vorzeichen der Nebenbedingung an der eigentlichen Nullstelle der Hauptbedingung können wir anhand dieser Informationen in beiden Fällen keine Entscheidung treffen.

Um den Fall, daß eine Nebenbedingung in einem separierenden Intervall ihr Vorzeichen ändert, auszuschließen, separieren wir nicht die Nullstellen von $f(t)$ alleine, sondern jeweils von dem Produkt $f(t)g_i(t)\ \forall 1 \leq i \leq l$. Wir erhalten jeweils eine Liste $L_{sep}^{f(t)g_i(t)}$ von separierenden Intervallen. Durch Vorzeichenbetrachtung an den Intervallgrenzen können wir nun leicht entscheiden, ob das Intervall eine Nullstelle von $f(t)$ oder $g_i(t)$ enthält. Sei $I_{(a_j,b_j)}^{f(t)g_i(t)}$ ein solches separierendes Intervall:

- $(f(a_j) > 0 \wedge f(b_j) < 0) \vee (f(a_j) < 0 \wedge f(b_j) > 0) \Rightarrow$ Nullstelle von $f(t)$
- $(g_i(a_j) > 0 \wedge g_i(b_j) < 0) \vee (g_i(a_j) < 0 \wedge g_i(b_j) > 0) \Rightarrow$ Nullstelle von $g_i(t)$

Erinnern wir uns an die Voraussetzungen für das Verfahren zur Nullstellenseparierung aus vorigem Abschnitt: dort forderten wir, daß das Polynom $A(t)$, dessen Nullstellen im Intervall $(0, 1)$ separiert werden sollten, keine mehrfachen Nullstellen besitzt. Auf unsere neue Situation mit $A(t) = f(t)g_i(t)$ angewendet, bedeutet dies, daß $f(t)$ und $g_i(t)$ keine mehrfachen und keine gemeinsamen Nullstellen besitzen dürfen. Wurden diese Einschränkungen dort getroffen, um die Terminierung des rekursiven Verfahrens zu garantieren, können wir sie uns an dieser Stelle zur Bestimmung des Verlaufs der $g_i(t)$ in einem Intervall $I_{(a_j,b_j)}^{f(t)g_i(t)}$ zu nutze machen. Wissen wir von diesem Intervall bereits, daß es eine Nullstelle des Polynoms $f(t)$ enthält, läßt sich der Verlauf des Polynoms $g_i(t)$ innerhalb des Intervalls wie folgt bestimmen:

- $g_i(a_j) > 0 \vee g_i(b_j) > 0 \Rightarrow g_i(t) > 0\ \forall t \in I_{(a_j,b_j)}^{f(t)g_i(t)}$
- $g_i(a_j) < 0 \vee g_i(b_j) < 0 \Rightarrow g_i(t) < 0\ \forall t \in I_{(a_j,b_j)}^{f(t)g_i(t)}$

Durch Anwendung der Nullstellenseparierung auf die l Polynome $f(t)g_i(t)\ \forall 1 \leq i \leq l$ erhalten wir l Listen $L_{sep}^{f(t)g_i(t)}$ mit separierenden Intervallen. Aus diesen Listen $L_{sep}^{f(t)g_i(t)}$ löschen wir zunächst solche Intervalle, die Nullstellen von $g_i(t)$ enthalten, oder in denen $g_i(t) < 0$ gilt, da diese Intervalle die Bedingung nicht erfüllen können. Die endgültige Ergebnisliste geht durch Schnitt aus den verbleibenden Einzellisten $L_{sep}^{f(t)g_i(t)}$ hervor. Ist sie leer, gibt es kein separierendes Intervall, also keine Nullstelle im Bereich $(0, 1)$, die das Gleichungssystem 3.1 erfüllt. Im folgenden nun die Grobstruktur des implementierten Algorithmus. Hierin werden neben der bereits bekannten Funktion *Uspensky(...)* zwei weitere Funktionen benutzt, die einer kurzen Beschreibung bedürfen:

- Intervallreduktion(Polynom<integer> f,Polynom<integer> g,
  List<Intervall> L)
  Diese Funktion streicht aus der Liste L der separierenden Intervalle des Polynoms $A(t) = f(t) \cdot g(t)$ all diejenigen Intervalle, in denen gilt:
  - $f(t) \neq 0$. Dieses Intervall enthält also eine Nullstelle von g.

  - $g(t) < 0$, so daß die Nullstelle dieses Intervalls keine Lösung des gesamten Gleichungssystems 3.1 sein kann.

- List<Intervall> Intervallistenschnitt(List<Intervall> $L1$, List<Intervall> $L2$)
  Diese Funktion berechnet den Schnitt aller Intervallpaare der Listen $L1$ und $L2$.

```
int Lösungsanzahl(Polynom<integer> f,Polynom<integer> g[anzahl])
begin
    for(i = 1 to anzahl) do
        Elimination-gemeinsamer-Nullstellen(f,g[i]);
    od
    A = f * g[1];
    L = Uspensky(A);
    Intervallreduktion(f,g[0],L);
    for(i = 2 to anzahl) do
        if(L = leere Liste) then return 0;
        A = f * g[i];
        L1 = Uspensky(A);
        Intervallreduktion(f,g[i],L1);
        if(L1 = leere Liste) then return 0;
        L2 = Intervallistenschnitt(L,L1);
        Lösche L;
        Lösche L1;
        L = L2;
    od
    return Größe(L);
end
```

3.2.3 Komplexitätsanalyse

Bevor wir die Gesamtlaufzeit des Uspensky-Verfahrens angeben, werden wir – wie bei der Untersuchung der Komplexität des Pedersen-Verfahrens in Abschnitt 3.1.3 – zunächst die Faktoren

- Anzahl der Nebenbedingungen

- Grad der Polynome

- Zahlenlängen während der Berechnungen

betrachten, da die Laufzeit im Wesentlichen von diesen abhängig ist.

3.2.3.1 Anzahl der Nebenbedingungen

Im Gegensatz zum Pedersen-Verfahren ist keine polynomielle Nebenbedingung erforderlich, die das betrachtete Intervall auf $(0,1)$ beschränkt; diese Einschränkung des Definitionsbereiches wird beim modifizierten Uspensky-Verfahren durch Transformation des Eingabepolynoms erreicht. Gehen wir wiederum davon aus, daß keine Fläche von mehr als 4 Kanten begrenzt wird, erhalten wir gemäß den Betrachtungen aus Kapitel 2 maximal vier polynomielle Nebenbedingungen (Kante-Kante-Test).

3.2.3.2 Grad der Polynome

Bei linearer Interpolation zwischen den gegebenen Objektkonfigurationen ergeben sich zur Beschreibung der einzelnen Kollisionsbedingungen Polynome vom maximalen Grad 5. Bei der in diesem Abschnitt beschriebenen Lösung des Entscheidungsproblems 2.1 mittels Nullstellenseparierung nach Uspensky wird der rekursiven Funktion ein polynomielles Produkt der Art $f(t)g_i(t)$ übergeben, das somit den Grad 10 besitzt.

3.2.3.3 Zahlenlängen während Zwischenberechnungen

Im folgenden sei ein Polynom

$$A(t) \quad = \quad \sum_{i=0}^{n} a_i t^i$$

gegeben, dessen Nullstellen mittels des in diesem Kapitel beschriebenen rekursiven Uspensky-Verfahrens separiert werden sollen. Im Verlauf des Verfahrens können diverse Transformationen auf das Ausgangspolynom $A(t)$ angewendet werden, die Einfluß auf die Länge der Koeffizienten haben. Deren Veränderung unter den drei verschiedenen Transformationen

$$A^*(t) \quad = \quad (t+1)^n \cdot A(\frac{1}{t+1}) \tag{3.18}$$

$$A_{left}(t) \quad = \quad 2^n \cdot A(\frac{t}{2}) \tag{3.19}$$

$$A_{right}(t) \quad = \quad 2^n \cdot A(\frac{t+1}{2}) \tag{3.20}$$

werden wir im folgenden betrachten, bevor wir Aussagen für den *worst-case* machen. Sei $L_{alt} = max\{|a_0|, \ldots, |a_n|\}$.

Veränderung der Koeffizientenlängen durch Transformation 3.18:

$$A^*(t) \quad = \quad (t+1)^n \cdot \sum_{i=0}^{n} a_i (\frac{1}{t+1})^i$$

$$
\begin{aligned}
&= \sum_{i=0}^{n} a_i (t+1)^{n-i} \\
&= \sum_{i=0}^{n} a_i \sum_{j=0}^{n-i} \binom{n-i}{j} t^j
\end{aligned}
$$

Für die Länge $L_{Trafo\ 1}$ der Koeffizienten des transformierten Polynoms gilt dann:

$$
\begin{aligned}
L_{Trafo\ 1} &\leq L_{alt} + |\sum_{j=0}^{n} \binom{n}{j}| \\
&= L_{alt} + |2^n| \\
&= L_{alt} + n
\end{aligned}
$$

Veränderung der Koeffizientenlängen durch Transformation 3.19:

$$
\begin{aligned}
A_{left}(t) &= 2^n \cdot \sum_{i=0}^{n} a_i (\frac{x}{2})^i \\
&= \sum_{i=0}^{n} 2^{n-i} \cdot a_i \cdot x^i
\end{aligned}
$$

Für die Länge $L_{Trafo\ 2}$ der Koeffizienten des transformierten Polynoms gilt dann:

$$
\begin{aligned}
L_{Trafo\ 2} &\leq L_{alt} + |2^n| \\
&= L_{alt} + n
\end{aligned}
$$

Veränderung der Koeffizientenlängen durch Transformation 3.20:

$$
\begin{aligned}
A_{right}(t) &= 2^n \sum_{i=0}^{n} a_i \cdot (\frac{t+1}{2})^i \\
&= \sum_{i=0}^{n} 2^{n-i} \cdot a_i \cdot (t+1)^i \\
&= \sum_{i=0}^{n} 2^{n-i} \cdot a_i \cdot \left(\sum_{j=0}^{i} \binom{i}{j} t^j \right)
\end{aligned}
$$

Für die Länge $L_{Trafo\ 3}$ der Koeffizienten des transformierten Polynoms gilt dann:

$$
L_{Trafo\ 3} \leq L_{alt} + (n+1)
$$

Koeffizientenlängen im worst-case:
Aus Satz 3.15 wissen wir, daß bis zum Abbruch des rekursiven Verfahrens höchstens $k = O(n\, L_d)$ Transformationen der Art 3.19 bzw. 3.20 notwendig sind. Wie dem Flußdiagramm 3.3 des modifizierten Uspensky-Verfahrens zu entnehmen ist, wird einer solchen Transformation zunächst eine Transformation der Art 3.18 vorausgeschickt. Für

die Koeffizientenlängen $L_{Trafo\ k}$ eines solchen in k Phasen transformierten Polynoms gilt dann:

$$\begin{aligned} L_{Trafo\ k} &\leq 2 \cdot (n + L_{alt}) \cdot k \\ &= 2 \cdot (n + L_{alt}) \cdot (c \cdot n \cdot L_d) \\ &= O(n^2 \cdot L_d) \end{aligned}$$

3.2.3.4 Gesamtlaufzeit

Die Laufzeit des Verfahrens ist zunächst abhängig von der zur Separierung der Nullstellen des Polynoms $A(t)$ benötigten Anzahl von Intervallhalbierungen; diese wird durch den Abstand der Wurzeln des jeweiligen Polynoms bestimmt. Eine untere Schranke hierfür ist in Formel 3.17 gegeben. Das zur Separierung benutzte rekursive Uspensky-Verfahren bricht ab, wenn $var(A) = 0 \vee var(A) = 1$ gilt, was gleichbedeutend mit der Tatsache ist, daß im Intervall $(0, 1)$ höchstens eine Nullstelle liegt. Daß dieser Zustand nach Anwendung von $O(n \cdot L_d)$ Transformationen der Art $A_{side}(t) \circ A^*$ mit $side \in \{left, right\}$ auf das Ausgangspolynom $A(t)$ erreicht wird, wissen wir bereits aus Satz 3.15. Die Kosten einer solchen Transformation betragen $O(n^2)$. Zur Abschätzung der Gesamtlaufzeit gehen wir von der Annahme $log\, n \leq L$ aus. Dies ist für unsere Anwendungen durchaus berechtigt, da der maximale Grad der Eingabepolynome bei linearer Interpolation gerade den Grad 10 erreicht, die Länge der Binärdarstellung der Eingabewerte $log\, 10$ in der Regel jedoch um ein Vielfaches übertrifft. Bezeichnen wir mit $L_w = O(n^2 \cdot L_d)$ die Zahlenlängen im *worst-case* und nehmen als obere Schranke für die Laufzeit der Multiplikation zweier L_w-Bit Zahlen gemäß [Yap94] Zeit $\mu(L_w) = O(L_w \cdot log\, L_w \cdot log\, log\, L_w)$ an, dann berechnet sich die Laufzeit des Verfahrens wie folgt:

$$\begin{aligned} T_{Usp} &= \#(\mathit{Transformationen}) \cdot \#(\mathit{arith.\ Operationen}) \cdot \mu(L_w) \\ &= O\Big(n \cdot L_d \cdot n^2 \cdot \mu(n^2 \cdot L_d)\Big) \\ &= O\Big(n^3 \cdot log(L + log\, n) \cdot \mu(n^2 \cdot log(L + log\, n))\Big) \\ &= O\Big(n^3 \cdot log\, L \cdot \mu(n^2 \cdot log\, L)\Big) \end{aligned}$$

Auf eine weitere Expansion obigen Ausdrucks durch Auswertung der Funktion μ möchten wir aus Gründen der Überschaubarkeit verzichten.

3.3 Zusammenfassung

In diesem Kapitel haben wir zwei Verfahren zur Lösung des Entscheidungsproblems 2.1 kennengelernt:

- das in seinem Ablauf absolut statische Pedersen-Verfahren. Hierin werden für die polynomielle Hauptbedingung f und für die 2^l möglichen Produktkombinationen der l polynomiellen Nebenbedingungen g_i die Hankelmatrizen aufgebaut, von denen dann jeweils die Folge der Hauptminoren zu berechnen ist. Von jeder dieser Folgen muß schließlich die Zahl der Vorzeichenwechsel bestimmt werden. Anhand der Anzahl dieser Vorzeichenwechsel läßt sich die Anzahl der Lösungen eines Gleichungssystems der Art 3.2 in einfacher Weise angeben. Durch Kombination der Ergebnisse gemäß Formel 3.6 erhält man schließlich die Zahl der Lösungen des Gleichungssystems 3.1. Um die Zahl der Lösungen im Intervall $(0,1)$ zu berechnen, wird zu Beginn eine zusätzliche polynomielle Nebenbedingung der Art $t-t^2 > 0$ zum existierenden Gleichungssystem hinzugefügt: diese besitzt lediglich im Intervall $(0,1)$ positive Funktionswerte. Die Laufzeit des Verfahrens beträgt:

$$O\Big(2^l \cdot n^3 \cdot \mu(l \cdot L \cdot n^2 + l \cdot log\, l \cdot n)\Big)$$

wobei mit $\mu(L)$ die zur Multiplikation zweier L-Bit Zahlen benötigte Zeit bezeichnet sei.

- ein Verfahren auf der Basis der Nullstellenseparierung nach dem modifizierten Uspensky-Verfahren. Hierbei werden von den l Polynomen $A(t) = f(t) \cdot g_i(t)$ die separierenden Intervalle bestimmt und entschieden, ob die Intervalle eine Nullstelle von f oder g_i enthalten. Handelt es sich um eine Nullstelle von f muß an einem Intervallpunkt die Nebenbedingung g_i ausgewertet werden, um deren Vorzeichen im gesamten Intervall zu bestimmen. Enthält das Intervall eine Nullstelle von g_i, ist es für unsere weiteren Betrachtungen nicht von Interesse. Bei dieser Vorgehensweise kann ein dynamischer Abbruch erfolgen, wenn:

  - von der polynomiellen Hauptbedingung f festgestellt wird, daß sie im Intervall $(0,1)$ keine Nullstelle besitzt
  - von einer polynomiellen Nebenbedingung g_i festgestellt wird, daß sie an allen Nullstellen von f im Intervall $(0,1)$ nur negative Funktionswerte besitzt

  Da im *worst-case* kein vorzeitiger Abbruch erfolgen kann, muß das Verfahren zur Lösung des Entscheidungsproblems 2.1 die Nullstellen aller l Polynome $A(t) = f(t) \cdot g_i(t)$, $1 \le i \le l$ separieren. Die Laufzeit dieses Verfahrens liegt dann bei:

  $$O\Big(l \cdot log\, L \cdot n^3 \cdot \mu(n^2 \cdot log\, L)\Big)$$

Kapitel 4

Ergebnisse aus der Implementierung

Die bisher vorgestellten Verfahren zur *exakten* Lösung des Problems der geometrischen Kollisionserkennung wurden alle im Rahmen dieser Arbeit in das $\mathbb{ICOR}$ -System, einem System zur interaktiven Montagesimulation, integriert. Es besteht aus folgenden Teilkomponenten, die im Laufe der Zeit im Rahmen von Diplomarbeiten und Fortgeschrittenenpraktikas implementiert und weiterentwickelt wurden:

- Objektentwurf (starre Körper, Polyeder)
- Visualisierung (Festlegung der Perspektive, des Darstellungsmodus, Animationen)
- Interaktive Bewegungsspezifikation (Definition von Bewegungsabläufen durch eine Folge von Translationen und Rotationen; auch eine Überlagerung beider Bewegungsarten ist möglich)
- Interaktive Programmierung von Robotern (textuelle und graphisch orientierte Form der Roboterprogrammierung)
- **Montagesimulation und Kollisionserkennung**
  Herzstück des Systems ist die Kollisionserkennung. Sie beinhaltet die Simulation von Bewegungsabläufen für einfache starre Körper und für Roboter unter ständiger Kontrolle, daß zu keinem Zeitpunkt eine Durchdringung von Objekten stattfindet. Die Kollisionsberechnungen wurden bisher auf numerischer Basis durchgeführt, was – wie wir im Verlauf dieser Arbeit bereits gesehen haben – durchaus zu Fehlentscheidungen in der Kollisionserkennung führen kann. Somit stellt die Integrierung der in Kapitel 3 vorgestellten Verfahren zur *exakten* Kollisionserkennung in diesen Teil des Gesamtsystems eine wichtige und wesentliche Erweiterung auf dem Weg zu einem realitätsnahen Simulationssystem dar.
- Algorithmen zur Bewegungsplanung (automatische Generierung von Wegen nach Vorgabe von Start- und Zielkonfiguration)

Im Vordergrund der Implementierung der Verfahren zur *exakten geometrischen Kollisionserkennung* standen die Objektorientiertheit und die Austauschbarkeit der benutzten Datentypen, um von der speziellen Implementierung eines Zahlenpaketes zur Darstellung *großer* Zahlen unabhängig zu werden. Dieses Vorgehen trägt sowohl zur Verständlichkeit als auch zur Wiederverwendbarkeit der entwickelten Software wesentlich bei. Durch die Parametrisierung der einzelnen Klassen können wir uns – wie wir noch sehen werden – die Vorteile einzelner Zahlenpakete hinsichtlich der Effizienz der zur Verfügung stehenden Operationen zu Nutze machen, während wir ihre Nachteile auszusparen versuchen. Außerdem ermöglicht dieses Konzept die einfache Verwendung verschachtelter Datentypen: z.B. Matrizen, deren Einträge Polynome sind.

4.1 Die verwendeteten Klassen

Die objektorientierte Implementierung der beschriebenen Verfahren erfordert zunächst die Bereitstellung verschiedener Klassen algebraischer Ausdrücke. Die Klassen der Polynome und Matrizen besitzen hierbei den größten Funktionsumfang.

- **Vektoren:**
  Vektoren bilden die Basis der analytischen Geometrie. Man benutzt sie, um Punkte zu repräsentieren und diese zu verschieben. Neben den Standardoperationen wurden im Rahmen der Implementierung noch das Skalar- und das Vektorprodukt benötigt.

- **Quaternionen:**
  Quaternionen benutzen wir, um die Orientierung unserer Objekte im Raum zu beschreiben. Auf ihnen operieren wir mittels Addition, Subtraktion und der eigenen Multiplikation.

- **Matrizen:**
  Matrizen werden im Verlauf sämtlicher Rechnungen früher oder später benutzt. Neben den Standardoperationen auf Matrizen wird im Pedersen-Verfahren die Folge der Determinanten der Hauptminoren zweier Hankelmatrizen berechnet, um Aussagen über die Anzahl der Nullstellen, die sowohl eine polynomielle Haupt- als auch Nebenbedingung erfüllen, zu machen. Da Standardverfahren zur Determinantenberechnung wie das Gaußsche Eliminationsverfahren z.B. Division benutzen, haben wir ein alternatives Verfahren implementiert, das lediglich auf Addition und Multiplikation zurückgreift. Am Beispiel einer 4×4-Matrix möchten wir die Vorgehensweise verdeutlichen: sei die Matrix

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

gegeben. Entwickelt man zur Berechnung von $det(A)$ nach der vierten Zeile, so ergibt sich:

$$det(\mathbf{A}) \quad = \quad a_{44}\ det(1,2,3) - a_{43}\ det(1,2,4) + a_{42}\ det(1,3,4) - a_{41}\ det(2,3,4)$$

$det(i_1, \ldots, i_j), 1 \leq j \leq 4$, bezeichnet hierbei die Determinante der Matrix, die entstanden ist durch:

- Streichung der Spalten der Matrix **A** mit den Nummern, die nicht in der Menge $\{i_1, \ldots, i_j\}$ enthalten sind
- Streichung der letzten $n - j$ Zeilen der Matrix **A**

Die fortgesetzte Berechnung der hier auftretenden Determinanten durch Entwicklung nach der jeweils letzten Zeile der verbleibenden Streichungsmatrix ergibt dann:

$$\begin{aligned}
det(1,2,3) &= a_{33}\ det(1,2) - a_{32}\ det(1,3) + a_{31}\ det(2,3) \\
det(1,2,4) &= a_{34}\ det(1,2) - a_{32}\ det(1,4) + a_{31}\ det(2,4) \\
det(1,3,4) &= a_{34}\ det(1,3) - a_{33}\ det(1,4) + a_{31}\ det(3,4) \\
det(2,3,4) &= a_{34}\ det(2,3) - a_{33}\ det(2,4) + a_{32}\ det(3,4)
\end{aligned}$$

$$\begin{aligned}
det(1,2) &= a_{22}\ det(1) - a_{21}\ det(2) \\
det(1,3) &= a_{23}\ det(1) - a_{21}\ det(3) \\
det(1,4) &= a_{24}\ det(1) - a_{21}\ det(4) \\
det(2,3) &= a_{23}\ det(2) - a_{22}\ det(3) \\
det(2,4) &= a_{24}\ det(2) - a_{22}\ det(4) \\
det(3,4) &= a_{24}\ det(3) - a_{23}\ det(4)
\end{aligned}$$

$$\begin{aligned}
det(1) &= a_{11} \\
det(2) &= a_{12} \\
det(3) &= a_{13} \\
det(4) &= a_{14}
\end{aligned}$$

In diesem Schema lassen sich die gesuchten Determinanten der Hauptminoren der Matrix **A** ablesen als $det(1), det(1,2), det(1,2,3), det(1,2,3,4)$. Die Anzahl der benötigten arithmetischen Operationen AOP bei dieser Berechnungsweise berechnet sich wie folgt:

$$\begin{aligned}
AOP &= \sum_{j=2}^{n} \binom{n}{j} \cdot (2 \cdot j - 1) \\
&= 2 \cdot \sum_{j=2}^{n} \binom{n}{j} \cdot j - \sum_{j=2}^{n} \binom{n}{j}
\end{aligned}$$

$$\begin{aligned}
&= 2 \cdot \sum_{j=1}^{n} \binom{n}{j} \cdot j - 2 \cdot n - \sum_{j=0}^{n} \binom{n}{j} + n + 1 \\
&= 2 \cdot n \cdot \sum_{j=1}^{n} \binom{n-1}{j-1} - 2 \cdot n - 2^n + n + 1 \\
&= 2 \cdot n \cdot 2^{n-1} - 2 \cdot n - 2^n + n + 1 \\
&= (n-1) \cdot 2^n - n + 1
\end{aligned}$$

Obwohl die Anzahl der hier benötigten Operationen im Vergleich zum Gaußschen Eliminationsverfahren asymptotisch gesehen wesentlich höher liegt, läßt sich der Einsatz dieser Berechnungsweise dennoch rechtfertigen:

- Für die bei uns auftretenden (5×5)-Matrizen ist die für beide Verfahren benötigte Anzahl von Operationen in etwa gleich.
- Obiges Verfahren kommt – im Gegensatz zum Gaußschen Eliminationsverfahren – ohne Division aus, was die Länge der Zahlen bei ganzzahliger Repräsentation während der Zwischenberechnungen geringer hält.
- Da für die Berechnung lediglich Addition und Multiplikation verwendet werden, erleichtert dies die Austauschbarkeit der verwendeten Datentypen erheblich.
- Im Gegensatz zu anderen Verfahren ist seine Implementierung recht unkompliziert.

Für den Fall, daß wir Determinanten größerer Matrizen berechnen müssen – wenn wir z.B. eine höhergradige polynomielle Interpolation zwischen den Objektkonfigurationen einsetzen – ist der Einsatz eines alternativen Determinantenberechnungsverfahrens möglich, das ebenfalls ohne Division auskommt und asymptotisch gesehen eine kubische Laufzeit besitzt: es ist dies das Verfahren von *Bareiss*, das in [Yap94] ausführlich beschrieben wird.

- **Polynome:**
  Polynome stellen im Rahmen unserer Anwendung die wichtigste Klasse von arithmetischen Objekten dar. Wie wir in Kapitel 2 gesehen haben, lassen sich die Kollisionsbedingungen der verschiedenen Kollisionsarten in unserem Modell durch polynomielle Gleichungen und Ungleichungen beschreiben, deren Grad von der verwendeten polynomiellen Interpolationsfunktion abhängig ist. Wie bereits erwähnt nehmen wir im Rahmen dieser Arbeit eine lineare Interpolation zwischen den vorgegebenen Konfigurationen an; die Kollisionsbedingungen erreichen dadurch maximal den Grad 5. Im Verlauf des Pedersen-Verfahrens können dann Polynome vom Grad 22 entstehen. Dieser immer noch relativ geringe Grad und die Tatsache, daß aus Erfahrung heraus in der Regel alle Koeffizienten ungleich 0 sind, legitimiert die folgende prinzipielle Darstellung der Klasse Polynom als Feld von Koeffizienten.

Polynome höheren Grades mit relativ wenig besetzten Potenzen wird man i.a. als Liste realisieren, was in unserem recht speziellen Fall jedoch zur Effizienzminderung führen würde. Als Koeffizienten unserer Polynome kommen zunächst nur ganzzahlige Datentypen (Big-Integer) in Frage, da die Verfahren zur Nullstellenseparierung auf solchen basieren. Zur Laufzeitverbesserung dieser Verfahren werden wir später bei der Verwendung von Heuristiken auf andere Datentypen zurückgreifen.

Neben den Standard-Arithmetikoperationen benötigen wir zusätzliche Funktionen, wie die Mehrfachnullstelleneliminierung eines Polynoms mit ganzzahligen Koeffizienten. Besitzt ein Polynom, das dem Uspensky-Algorithmus als Eingabe übergeben wird, nämlich mehrfache Nullstellen, wird das Abbruchkriterium für den rekursiven Algorithmus nicht erfüllbar. Ob ein Polynom mit ganzzahligen Koeffizienten mehrfache Nullstellen besitzt, läßt sich folgendermaßen überprüfen:

$$ggT(f, f') \neq 1 \Longrightarrow \text{Mehrfachnullstellen vorhanden}$$

Die Berechnung des ggT geschieht durch den *Euklidischen Algorithmus*.

Desweiteren müssen wir garantieren können, daß zwei verschiedene Polynome keine gemeinsamen Nullstellen besitzen. Bei der Lösung des Entscheidungsproblems 2.1 unter Verwendung des Uspensky-Algorithmus haben wir die Nullstellen von Polynomen der Art $f(t) \cdot g_i(t)$ zu separieren. Besitzen $f(t)$ und $g_i(t)$ hierbei eine gemeinsame Nullstellen, muß diese im Polynom $f(t)$ eliminiert werden. Die Berechnung etwaiger gemeinsamer Nullstellen geschieht hierbei wieder mittels des *Euklidischen Algorithmus*; zur Eliminierung dieser Nullstellen wird jedoch eine Division benötigt, die den Ring der ganzen Zahlen $\mathbb{Z}$ nicht verläßt, sozusagen eine *Pseudo-Polynomdivision*. Hierbei werden die Koeffizienten vor einer Division so erweitert, daß diese im Ring der ganzen Zahlen $\mathbb{Z}$ möglich ist. Diese Manipulation ändert nichts an der Lage der Nullstellen des Polynoms.

- **Intervalle:**
  Bei der Berechnung der isolierenden Intervalle eines Polynoms muß mit Listen von Intervallen mit rationalen Intervallgrenzen operiert werden.

4.2 Methoden zur Effizienzsteigerung

In diesem Abschnitt werden wir uns mit der Fragestellung beschäftigen, wie wir die zur Verfügung stehenden Ressourcen effizienter einsetzen können: im Vordergrund stehen dabei Techniken zur Speicherplatzeinsparung und zur Laufzeitverbesserung. Laufzeiteinsparungen werden wir durch den Einsatz geeigneter Heuristiken und Datentypen zu erreichen versuchen. Ansatzpunkt für die Heuristiken stellen die polynomiellen Ungleichungssysteme dar, von denen es zu entscheiden gilt, ob sie im Intervall $(0, 1)$ eine Lösung besitzen. Wir werden einfach zu berechnende Kriterien angeben, die eine ausführliche und somit auch teure Untersuchung der Ungleichungssysteme mittels der in Kapitel 3 beschriebenen Verfahren von vornherein verhindern.

4.2.1 Reference-Counting

Ein Ansatz, die vorhandenen Ressourcen effektiver zu nutzen, ist das sogenannte *Reference-Counting*. Diese Technik ist vor allem in der objektorientierten Programmierung anzutreffen; sie zielt darauf ab, gemeinsame Daten verschiedener Objekte nur einmal anzulegen und die darauf verwendeten Referenzen zu zählen. Der angelegte Speicherplatz wird erst dann wieder freigegeben, wenn von den Objekten, die ihn ursprünglich benutzt haben, keines mehr existiert. Die Vorteile dieser Strategie liegen auf der Hand:

- Einsparung von Hauptspeicherplatz
- Reduzierung der Aufrufe von teuren Speicherverwaltungsfunktionen (Anlegen, Freigeben)

Die Implementierung dieser Technik sieht so aus, daß man dem für Objekte angelegten Speicherplatz eine Struktur zuordnet, die zusätzlich einen Zähler besitzt, der die Anzahl der auf diesen Speicherplatz benutzten Referenzen registriert. Dieser Zähler wird verändert beim:

- Aufruf des entsprechenden Konstruktors, der ein neues Objekt anlegt. In diesem Fall wird der Zähler mit 1 initialisiert.
- Aufruf des Copy-Konstruktors, der ein neues Objekt aus einem bereits existierenden anlegt. Der Zähler wird um 1 inkrementiert.
- Aufruf des Zuweisungsoperators, der ein Objekt einem anderen zuweist. Auch hier wird der Zähler um 1 inkrementiert.
- Aufruf des Destruktors. Der Zähler wird um 1 dekrementiert, falls noch mindestens zwei Objekte existieren, die eine Referenz auf den zugehörigen Speicherplatz benutzen. Gibt es nur noch ein solches Objekt, kann der Speicherplatz wieder freigegeben werden.

Da in manchen Situationen tatsächlich eine *echte* Kopie eines Objektes benötigt wird, muß zusätzlich eine spezielle *Hard-Copy*-Funktion zur Verfügung gestellt werden, die eine reale Kopie des entsprechenden Speicherplatzes anlegt.

4.2.2 Heuristiken zur Laufzeitverbesserung

Die Tatsache, daß zur *exakten* Entscheidung, ob ein Gleichungssystem der Art 3.2 im Intervall $(0, 1)$ eine Lösung besitzt, unter Umständen ein l-facher Aufruf des Uspensky-Algorithmus mit den Polynomen $A(t) = f(t) \cdot g_i(t)$, $1 \leq i \leq l$, notwendig ist, ist nicht zufriedenstellend. Die neue Frage, die sich an dieser Stelle auftut, lautet, ob man einem solchen Gleichungssystem in vielen Fällen nicht einfach *ansehen* kann, daß es im

Intervall $(0,1)$ keine Lösung besitzt. An erster Stelle wird hierbei die Untersuchung des Polynoms

$$f(t) \;=\; \sum_{i=0}^{n} a_i t^i$$

stehen: besitzt dieses keine Nullstelle im Intervall $(0,1)$, ist eine weitere Untersuchung der Nebenbedingungen hinfällig. Ebenso braucht das Ungleichungssystem nicht weiter betrachtet zu werden, wenn von einer polynomiellen Nebenbedingung festgestellt werden kann, daß sie im Intervall $(0,1)$ nur negative Werte annimmt. Im folgenden nun einige Strategien zur Ausdünnung der zu untersuchenden Gleichungssysteme:

- Gilt für alle Koeffizienten des Polynoms $a_i > 0$, sind dessen Funktionswerte im Intervall $(0,1)$ ebenfalls größer null.
- Gilt umgekehrt für alle Koeffizienten des Polynoms $a_i < 0$, sind dessen Funktionswerte im Intervall $(0,1)$ ebenfalls kleiner null.
- Für ein Polynom $f(t)$ gelten folgende Abschätzungen, falls $t \in [0,1]$:

$$\begin{aligned}
f(t) &= \sum_{i=0}^{n} a_i t^i \\
&= a_0 + \sum_{a_i>0, i\neq 0} a_i t^i + \sum_{a_i<0, i\neq 0} a_i t^i \\
&\leq a_0 + \sum_{a_i>0, i\neq 0} a_i \qquad (4.1) \\
&= S_{oben} \\
&\text{und} \\
f(t) &= \sum_{i=0}^{n} a_i t^i \\
&= a_0 + \sum_{a_i>0, i\neq 0} a_i t^i + \sum_{a_i<0, i\neq 0} a_i t^i \\
&\geq a_0 + \sum_{a_i<0, i\neq 0} a_i \qquad (4.2) \\
&= S_{unten} \qquad (4.3)
\end{aligned}$$

Diese beiden Schranken für die Funktionswerte eines Polynoms $f(t)$ im Intervall $[0,1]$ lassen sich effizient berechnen und bieten weitere Möglichkeiten die Zahl der genau zu untersuchenden polynomiellen Ungleichungssysteme einzuschränken:

- Gilt für die polynomielle Hauptbedingung

  $$S_{oben} < 0 \quad \text{oder } S_{unten} > 0$$

  kann sie im Intervall $(0,1)$ keine Nullstelle besitzen.

- Gilt für eine polynomielle Nebenbedingung

$$S_{oben} < 0$$

  kann das gesamte Ungleichungssystem 3.1 keine Lösung besitzen, da die betrachtete Nebenbedingung im Intervall $(0, 1)$ nur negative Funktionswerte besitzt.

Wie sich der Einsatz dieser einfachen Heuristiken in Laufzeiteinsparungen niederschlägt, werden wir in Abschnitt 4.3 an praktisch durchgeführten Laufzeitmessungen ablesen können.

4.2.3 Der verwendete Datentyp

Im folgenden Abschnitt werden wir uns mit den Problemen auseinandersetzen, die im Verlauf geometrischer Berechnungen auftreten können; Auslöser für diese ist die Verwendung von Datentypen begrenzter Präzision, wie sie hardwaremäßig implementiert sind. Will man geometrisch *exakt* rechnen, müssen softwaremäßig implementierte Datentypen verwendet werden, die eine nahezu unbegrenzte Genauigkeit zulassen. Wir werden einige ausgefeilte Techniken kennenlernen, die diesen ein gewisses Maß an *Intelligenz* verleihen, was zu Laufzeiteinsparungen führen kann.

4.2.3.1 Allgemeines zu exakten geometrischen Berechnungen

Geometrische Berechnungen lassen sich in *numerische* und *kombinatorische* Teile gliedern. Numerisches Rechnen wird z.B. bei der Lösung linearer Gleichungssysteme angewendet. Die Graphsuche ist ein typisches Beispiel für kombinatorisches Rechnen. Die Berechnung der konvexen Hülle einer Menge von Punkten ist ein typisch geometrisches Problem: die Punkte werden hierbei durch Zahlen spezifiziert, die konvexe Hülle ist dann im Wesentlichen eine kombinatorische Struktur – nämlich ein beschrifteter Graph. Damit aber in solchen geometrischen Berechnungen die berechneten Zahlenwerte auch mit der zugrundeliegenden kombinatorischen Struktur konsistent bleiben, ist ein hoher Aufwand erforderlich. Die Implementierung geometrischer Algorithmen gerät damit i.a. zu einer schwierigen Aufgabe. Die meisten hierbei entstehenden Probleme können auf die Tatsache zurückgeführt werden, daß der Großteil dieser Algorithmen unter Annahme exakter Real-Arithmetik formuliert ist; d.h. von den zugrundeliegenden arithmetischen Operationen wird eine fehlerfreie Ausführung erwartet. Die Standardimplementierung der reellen Zahlen – die Floating-Point-Arithmetik – erfüllt diese Annahme jedoch nicht, so daß wir bei deren Verwendung schon in einfachen numerischen Berechnungen dem *Problem unexakter Arithmetik* gegenüber stehen.

Wenn wir nun im Rahmen dieser Arbeit von *exaktem Rechnen* reden, fassen wir damit speziell zwei Punkte zusammen:

- wir repräsentieren alle numerischen Werte in *exakter* Weise (nicht zu verwechseln mit *exakter* Berechnung der numerischen Werte)

- wir treffen alle Entscheidungen *fehlerfrei*

Das *Problem unexakter Arithmetik* kann überall dort auftreten, wo Zahlenmodelle begrenzter Genauigkeit benutzt werden. Insbesondere birgt das Rechnen mit *algebraischen Zahlen* (komplexe Wurzel eines Polynoms mit ganzzahligen Koeffizienten) Schwierigkeiten in sich: es ist nämlich nicht möglich, jede algebraische Zahl in einem Zahlenmodell begrenzter Genauigkeit zu berechnen. Solche Zahlen lassen sich lediglich endlich beschreiben: die algebraische Zahl $\sqrt{2}$ ist so z.B. eindeutig beschreibbar als die Wurzel des Polynoms $x^2 - 2$ im Intervall $[1, 2]$: kurz $(x^2 - 2, [1, 2])$. Diese Darstellungsweise bezeichnet man als die *isolierende Intervallrepräsentation*. Im obigen Fall ist $[1, 2]$ isolierendes Intervall von $x^2 - 2$, da dieses Polynom dort nur eine Nullstelle besitzt. Die Tatsache, daß wir $\sqrt{2}$ als $(x^2 - 2, [1, 2])$ exakt beschreiben können, suggeriert eine unbegrenzte Präzision, die für sich alleine allerdings noch kein exaktes Rechnen zur Folge hat.

Wenn wir nun im Verlauf unserer exakten Berechnungen mit algebraischen Zahlen operieren und nur fehlerfreie Entscheidungen treffen, impliziert dies, daß wir von jedem arithmetischen Ausdruck, der algebraische Zahlen enthält, sein Vorzeichen exakt bestimmen können. Wir müssen also in der Lage sein, die isolierende Intervallrepräsentation von $\sqrt{5} - \sqrt{3}$ anzugeben und das Vorzeichen von $2\sqrt{5} - 2\sqrt{3} - 1$ zu bestimmen.

Um das Problem der geometrischen Kollisionserkennung *exakt* lösen zu können, ist es – wie wir in Kapitel 2 gesehen haben – notwendig, effizient mit solchen algebraischen Zahlen umgehen zu können.

Grundlage für das exakte Rechnen ist ein Zahlenmodell, das es uns erlaubt, mit beliebig[1] großen Zahlen zu hantieren. Da die Standardbreite der Arithmetikeinheiten bei heutigen Rechnern lediglich bei 32 bzw. 64 Bit liegt, wir jedoch in der Lage sein müssen, in Abhängigkeit von der Eingabegenauigkeit Zahlen von mehreren 100 Bits zu verarbeiten, sind wir auf Softwarepakete angewiesen, die Funktionen auf solch großen Zahlen zur Verfügung stellen; daß eine solche Softwarelösung einer Hardwareimplementierung in punkto Effizienz weit unterlegen ist, liegt auf der Hand. Hier deutet sich ein *Trade-Off* zwischen Rechengeschwindigkeit und Rechengenauigkeit an. Deshalb wird man darum bemüht sein, in möglichen Fällen – also dann, wenn die benötigte Präzision nicht allzu groß ist – auf die hardwaremäßig implementierte *floating-point-Arithmetik* zurückzugreifen.

4.2.3.2 Effizienzsteigerung durch Datentypaustausch

Eine weitere Möglichkeit zur Verbesserung der Laufzeit eröffnet sich aus der Beobachtung, daß in nur sehr wenigen Fällen eine hohe Präzision in den Zwischenrechnungen benötigt wird, um beispielsweise das Vorzeichen eines Ausdrucks zu berechnen. Für solche Szenen, in denen die Objekte von einer gemeinsamen Kollision weit entfernt sind und sich auch nur in geringem Maße fortbewegen, wäre es wünschenswert, durch

[1] Die Länge der darstellbaren Zahlen ist natürlich durch den vorhandenen Speicherplatz beschränkt; für unsere Anwendungen kann davon ausgegangen werden, daß dies jedoch nicht der limitierende Faktor sein wird.

die Verwendung eines *intelligenteren* Datentyps ein gewisses Maß an Präzision und somit auch an Laufzeit einzusparen. Aufgrund der Datentyp-Parametrisierung der implementierten Klassen ist es ohne weiteres möglich, den für den Aufbau der Polynome standardmäßig benutzten LEDA-integer-Datentyp durch die schnelleren *real* oder *floatf* zu ersetzen. Man kann nun hoffen, daß durch die bei der Implementierung dieser Datentypen verwendeten Techniken des *Floating-Point-Filters* und der *Lazy-Evaluation* die Laufzeit der Algorithmen im Mittel günstiger ausfällt (siehe hierzu: [MN94]):

- *Floating-Point-Filter* werten einen arithmetischen Ausdruck E zu seiner Vorzeichenbestimmung zunächst auf der Basis der schnellen, weil hardwaremäßig implementierten Floating-Point-Arithmetik aus; dieser Vorgang ist gleichbedeutend mit der Berechnung einer Approximation $\tilde{E}$ von E. Zusätzlich wird eine Fehleranalyse durchgeführt, um eine Fehlerschranke δ für $|E - \tilde{E}|$ zu erhalten. Ergibt sich nun:
  - $\delta = 0$ oder $|\tilde{E}| > \delta$, so gilt $sign(\tilde{E}) = sign(E)$
  - $\delta > 0$ und $|\tilde{E}| < \delta$, so kann anhand der Approximation $\tilde{E}$ keine Aussage über das Vorzeichen von E getroffen werden. In diesem Fall hat der Floating-Point-Filter versagt; das Vorzeichen des Ausdrucks E muß exakt berechnet werden

- die Idee der *Lazy-Evaluation*-Technik besteht darin, jede Zahl zweimal zu repräsentieren: einmal durch ihre Definition (integer oder Operator mit Zeigern auf seine Argumente) und einmal durch eine Approximation (floating-point-Zahl beliebiger Präzision mit Fehlerschranke). Jedesmal wenn nun das Vorzeichen dieser Zahl bestimmt werden muß, wird die Güte der Approximation so lange verbessert (z.B. wiederholte Verdoppelung der Präzision), bis eine exakte Entscheidung getroffen werden kann. Diese Technik ist beispielsweise im Datentyp *real* der LEDA-Bibliothek realisiert.

4.3 Laufzeitvergleiche

In diesem Abschnitt werden wir

- die Laufzeiten der beiden in Kapitel 3 vorgestellten Verfahren zur Lösung des Entscheidungsproblems 2.1 an mehreren Szenen

- die Auswirkungen des Einsatzes verschiedener Datentypen auf die Laufzeit der Algorithmen

- die Wirkungsweise von verschiedenen Heuristiken, wie sie in Abschnitt 4.2.2 beschrieben werden, zur Reduzierung der Anzahl der exakt zu lösenden Gleichungssysteme im Rahmen eines vollständigen Kollisionstests

- die Laufzeiten einer numerischen und einer exakten Lösung des Entscheidungsproblems 2.1

miteinander vergleichen. Die von uns hierzu konstruierten Szenen enthalten jeweils zwei polyhedrale Objekte: einen Quader und einen Tetraeder, deren statische Daten in Anhang B ausführlich gegeben sind und die – um die volle Anzahl an Kollisionstests durchführen zu müssen um eine exakte Aussage bzgl. des Vorhandenseins einer etwaigen Kollision machen zu können – in den zugrundegelegten Szenen während ihren Bewegungen nicht zusammenstoßen. Die Komplexität dieser Objekte ergibt sich als:

	Quader	Tetraeder
Ecken	8	4
Kanten	12	6
Flächen	6	4

Die Anzahl der einzeln durchzuführenden Kollisionstests, nach denen die gemessenen Laufzeiten in den nachfolgenden Grafiken aufgeschlüsselt sind, berechnen sich wie folgt:

- Kante-Kante: $12 \cdot 6 = 72$
- Punkt-Fläche 1: $8 \cdot 4 = 32$ (Quader - Tetraeder)
- Punkt-Fläche 2: $6 \cdot 4 = 24$ (Tetraeder - Quader)
- Punkt-Kante 1: $8 \cdot 6 = 48$ (Quader - Tetraeder)
- Punkt-Kante 2: $12 \cdot 4 = 48$ (Tetraeder - Quader)
- Punkt-Punkt: $8 \cdot 4 = 32$

- Gesamt: 256

Da die Anzahl der gleichzeitig bewegten Objekte auf den Grad der entstehenden polynomiellen Ungleichungssysteme, wie sie in Kapitel 2 hergeleitet werden, wesentliche Auswirkungen hat, machen wir bei den Messungen Unterschiede zwischen Szenen mit einem und Szenen mit zwei gleichzeitig bewegten Objekten.

Desweiteren betrachten wir in jeder Gegenüberstellung die Unterschiede, die sich durch den Einsatz einfacher Heuristiken, die als Filter für die exakt zu lösenden Ungleichungssysteme fungieren, gegenüber einer *vollständig exakten* Lösung in punkto Laufzeit ergeben.

Im Vergleich der beiden unterschiedlichen Lösungsverfahren, die in Kapitel 3 ausführlich geschildert wurden, konnte in der Szene mit zwei gleichzeitig bewegten Objekten eine Entscheidung bzgl. des Vorhandenseins einer Kollision mittels des Pedersen-Verfahrens aufgrund zu geringer Hauptspeicherkapazität des verwendeten Rechners nicht getroffen werden.

Zur Verdeutlichung der Abhängigkeit der Laufzeit der einzelnen Verfahren von den jeweilig verwendeteten Datentypen, stellen wir zwei Zahlenpakete zur Darstellung *beliebig* großer, ganzer Zahlen gegenüber:

- den Datentyp *integer* aus der LEDA-Bibliothek (Library of Efficient Data types and Algorithms), die am Max-Planck-Institut für Informatik an der Universität des Saarlandes entwickelt wurde.

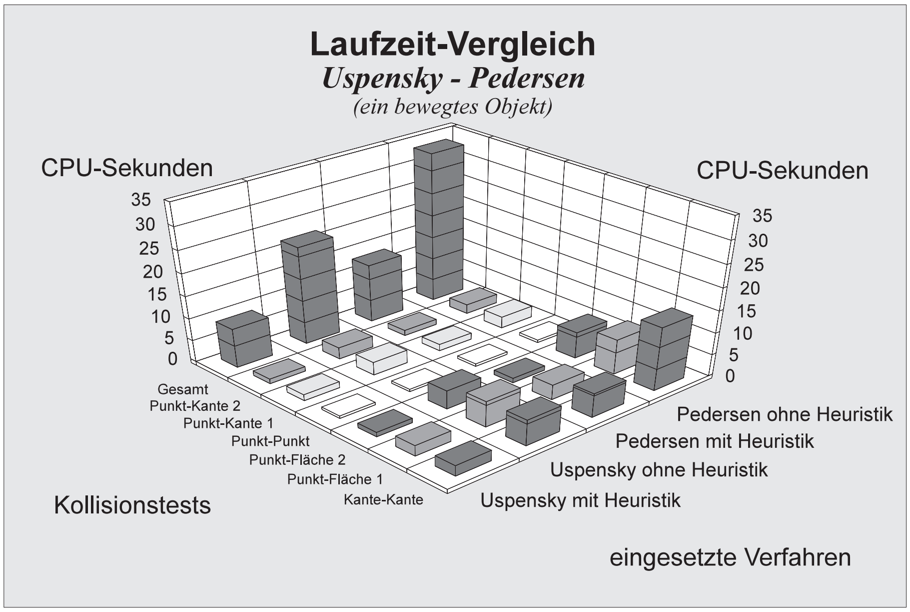

Abbildung 4.1: Laufzeitvergleich: Uspensky-Pedersen bei einem bewegten Objekt

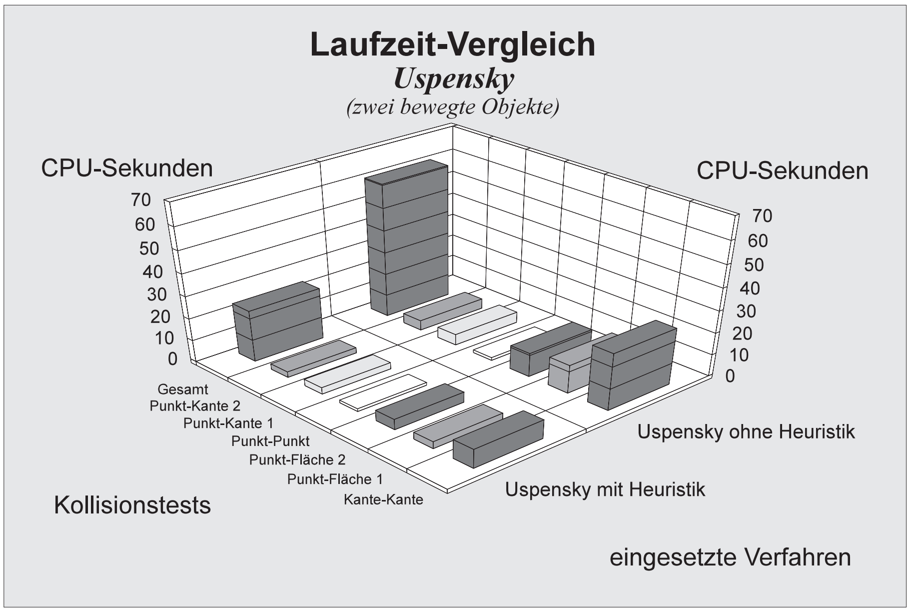

Abbildung 4.2: Laufzeitvergleich: Uspensky bei zwei bewegten Objekten

- den Datentyp *bigint* aus der LiDIA-Bibliothek, die von der LiDIA-Gruppe am Lehrstuhl von Prof. Buchmann an der Universität des Saarlandes entwickelt wurde.

Wie sich in mehrfachen Messreihen herausgestellt hat, nimmt die Performance der Operationen auf den *LiDIA-bigints* im Verlauf der komplexen Kollisionsberechnungen gegenüber den *LEDA-integern* merklich ab, was wir auf ein günstigeres Speichermanagement innerhalb der LEDA-Bibliothek zurückführen (z.B. *reference counting*).

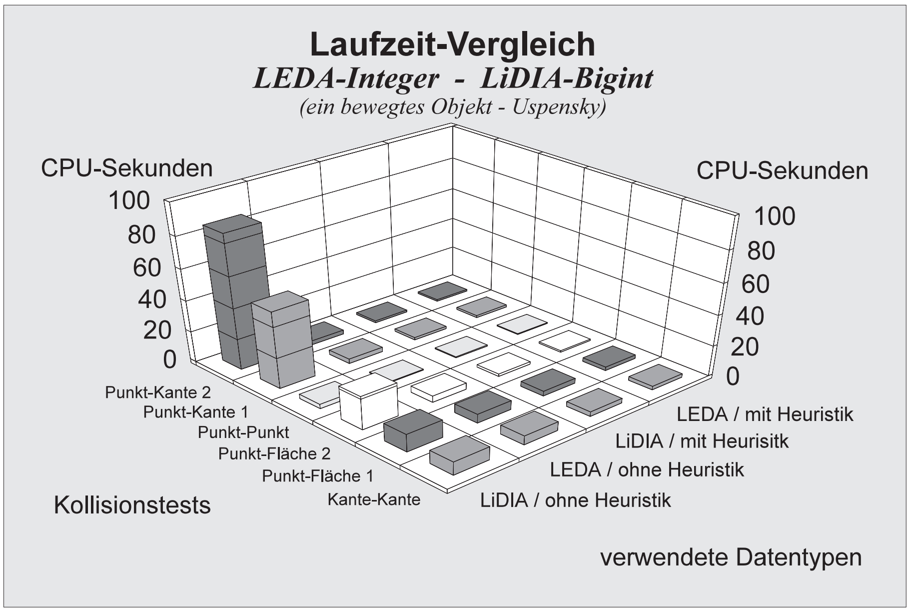

Abbildung 4.3: Laufzeitvergleich: LEDA-Integer – LiDIA-Bigint bei einem bewegten Objekt

Eine sehr interessante Laufzeitgegenüberstellung erfolgt in Abbildung 4.5: dort können wir die Berechnungszeiten einer exakten Lösung mit denen einer numerischen vergleichen. Bei der exakten Kollisionsberechnung wurde das in Kapitel 3 beschriebene Uspensky-Verfahren unter Verwendung des *integer*-Datentyps der LEDA-Bibliothek eingesetzt, da sich diese Konfiguration im Verlauf unserer Tests als die effizienteste erwiesen hat. Die numerische Lösung läuft unter Verwendung des hardwaremäßig implementierten Gleitkommadatentyps *double* wie folgt ab:

1. Aufbau der dem jeweiligen Kollisionstest entsprechenden, polynomiellen Ungleichungssysteme.

2. numerische Berechnung der komplexen Nullstellen der polynomiellen Hauptbedingung nach dem Verfahren von Laguerre, das in [FPT88] ausführlich beschrieben ist. Da als Lösung des Ungleichungssystems jedoch nur reelle Nullstellen der polynomiellen Hauptbedingung in Frage kommen, gilt es, diese zu selektieren. Zur Feststellung, ob der Imaginärteil der komplexen Wurzel tatsächlich null ist, verwenden wir, aufgrund der in Zahlendarstellungen begrenzter Präzision hinsichtlich Nullvergleichen herrschenden Problematik, die Technik des ϵ-Tweaking. Hierbei wird eine Zahl als null angesehen, wenn ihr Abstand zu null kleiner einer vorgegebenen Schranke ϵ ist. Daß diese Vorgehensweise *unexakt* ist und somit

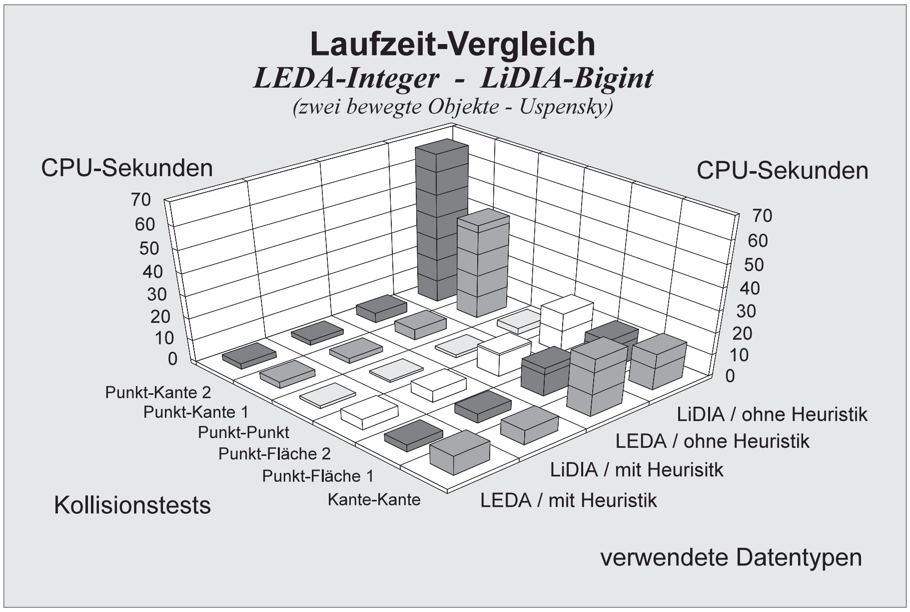

Abbildung 4.4: Laufzeitvergleich: LEDA-Integer – LiDIA-Bigint bei zwei bewegten Objekten

durchaus zu Fehlentscheidungen hinsichtlich der korrekten Nullstellenselektion führen kann, ist unmittelbar verständlich.

3. Auswertung der polynomiellen Nebenbedingungen an den Nullstellen der Hauptbedingung aus dem Intervall $(0, 1)$. Hierbei verstärken sich die von der numerischen Nullstellenberechnung stammenden Rundungsfehler weiter.

Aufgrund der Problematik mit Nullvergleichen bei Benutzung der floating-point-Arithmetik unter Verwendung der ϵ-Tweaking-Technik werden bei einer numerischen Lösung die Kollisionstests

- Punkt-Kante
- Punkt-Punkt

bereits von den beiden verbleibenden (Kante-Kante, Punkt-Fläche) abgedeckt.

Es ist unmittelbar erkennbar, daß eine steif schematisch ablaufende exakte Lösung der numerischen in punkto Geschwindigkeit – erwarteterweise – um ein Vielfaches unterlegen ist. Im Gegenzug erhält man allerdings die Sicherheit einer exakten Aussage bzgl. des Vorliegens einer Kollision, was im Fall der numerischen Berechnung nicht gewährleistet ist. Der Einsatz einfacher Heuristiken, wie sie in Abschnitt 4.2.2 beschrieben werden, ist jedoch durchaus dazu geeignet, die Diskrepanz in der Laufzeit beider Vorgehensweisen deutlich zu verringern.

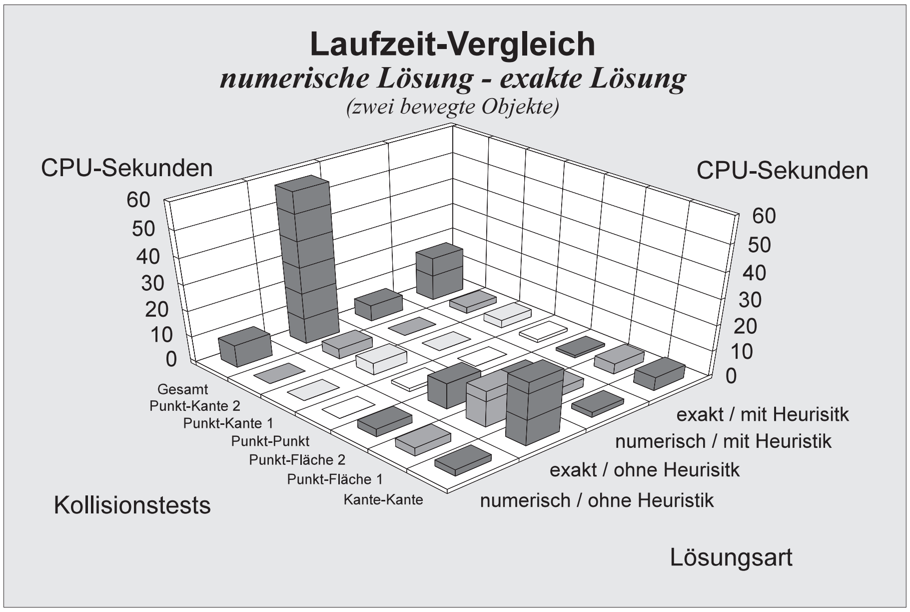

Abbildung 4.5: Laufzeitvergleich: numerische-exakte Lösung

4.4 Zusammenfassung

Zur praktischen Umsetzung unseres Ansatzes zur exakten geometrischen Kollisionserkennung war zunächst eine gewisse Vorarbeit zu leisten, um überhaupt mit den entstehenden algebraischen Ausdrücken und Formeln hantieren zu können. Das Hauptaugenmerk ruhte hierbei auf einer modularen Implementierung, die zudem eine einfache Austauschbarkeit der zugrundeliegenden Datentypen ermöglichen sollte.

Besondere Aufmerksamkeit haben wir im weiteren Verlauf der Effizienzsteigerung der implementierten exakten Verfahren gewidmet, um sie gegenüber numerischen in punkto Laufzeit konkurrenzfähig zu machen. Ansatzpunkte bildeten hierfür die Speicherplatzverwaltung, die zu untersuchenden polynomiellen Ungleichungssysteme sowie die zur Darstellung *beliebig* großer Zahlen verwendeten Datentypen.

Abgerundet wurde dieses Kapitel von Laufzeitgegenüberstellungen verschiedenartiger Verfahren und Verfahrenskonfigurationen. Bei den Meßreihen hat sich gezeigt, daß der *exakte* Ansatz zur Kollisionsberechnung gegenüber einem numerischen durch Verwendung ausgefeilter Filter durchaus konkurrenzfähig werden kann. Die Laufzeiten beider Ansätze unterscheiden sich nach unseren Erfahrungen in der Regel um einen Faktor 2 - 4.

Anhang A

Weitere mathematische Grundlagen

A.1 Rechenregeln für Vektor-Matrixoperationen

Seien im folgenden $\lambda, \mu, \nu \in \mathbb{R}$ und $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^3$.
Ein Vektorprodukt $\mathbf{c} = \mathbf{a} \times \mathbf{b}$ läßt sich, wie man leicht nachvollziehen kann, folgendermaßen als Matrix-Vektor-Multiplikation schreiben:

$$\begin{aligned}
\mathbf{c} &= \mathbf{a} \times \mathbf{b} \\
&= \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \\
&= \begin{pmatrix} b_3 a_2 - b_2 a_3 \\ b_1 a_3 - b_3 a_1 \\ b_2 a_1 - b_1 a_2 \end{pmatrix} \\
&= \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \qquad (A.1) \\
&= \mathbf{A}^{\times} \cdot \mathbf{b}
\end{aligned}$$

Für doppelte Vektorprodukte gilt folgende Expansionsregel:

$$\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) \;=\; (\mathbf{a}^T \mathbf{c})\mathbf{b} - (\mathbf{a}^T \mathbf{b})\mathbf{c} \qquad (A.2)$$

Das Skalarprodukt zweier Vektorprodukte läßt sich gemäß der *Lagrangeschen Identität* vereinfachen:

$$(\mathbf{a} \times \mathbf{b})^T (\mathbf{c} \times \mathbf{d}) \;=\; (\mathbf{a}^T \mathbf{c})(\mathbf{b}^T \mathbf{d}) - (\mathbf{a}^T \mathbf{d})(\mathbf{b}^T \mathbf{c}) \qquad (A.3)$$

Die Lösungen λ, μ, ν eines 3×3-Gleichungssystems der Form

$$\lambda \mathbf{a} + \mu \mathbf{b} + \nu \mathbf{c} \;=\; \mathbf{d} \qquad (A.4)$$

lassen sich mit der Regel von *Cramer* wie folgt berechnen:

$$\lambda = \frac{det[\mathbf{d},\mathbf{b},\mathbf{c}]}{det[\mathbf{a},\mathbf{b},\mathbf{c}]}, \qquad \mu = \frac{det[\mathbf{a},\mathbf{d},\mathbf{c}]}{det[\mathbf{a},\mathbf{b},\mathbf{c}]}, \qquad \nu = \frac{det[\mathbf{a},\mathbf{b},\mathbf{d}]}{det[\mathbf{a},\mathbf{b},\mathbf{c}]} \tag{A.5}$$

Satz A.1:
Vier Punkte $\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d} \in \mathbb{R}^3$ *liegen gemeinsam in einer Ebene, wenn gilt:*

$$det\begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix} = 0 \tag{A.6}$$

Beweis:
Liegen die drei Punkte $\mathbf{a},\mathbf{b}$ und $\mathbf{c}$ bereits in einer Ebene, so läßt sich gemäß der Drei-Punkte-Ebenengleichung folgendermaßen überprüfen, ob ein weiterer Punkt $\mathbf{d}$ ebenfalls in dieser Ebene liegt:

$$\begin{aligned} & \mathbf{a} + \lambda(\mathbf{b}-\mathbf{a}) + \mu(\mathbf{c}-\mathbf{a}) &=& \ \mathbf{d} \\ \Longleftrightarrow \quad & (1-\lambda-\mu)\mathbf{a} + \lambda\mathbf{b} + \mu\mathbf{c} &=& \ \mathbf{d} \end{aligned}$$

Die drei reellen Parameter $(1-\lambda-\mu), \lambda, \mu$ lassen sich durch die Cramersche Regel A.5 berechnen. Soll obiges Gleichungssystem eine Lösung besitzen, so muß gelten:

$$\begin{aligned} & 1-\lambda-\mu &=& \ \tfrac{det[\mathbf{d},\mathbf{b},\mathbf{c}]}{det[\mathbf{a},\mathbf{b},\mathbf{c}]} \\ \Longleftrightarrow \quad & 1 - \tfrac{det[\mathbf{a},\mathbf{d},\mathbf{c}]}{det[\mathbf{a},\mathbf{b},\mathbf{c}]} - \tfrac{det[\mathbf{a},\mathbf{b},\mathbf{d}]}{det[\mathbf{a},\mathbf{b},\mathbf{c}]} &=& \ \tfrac{det[\mathbf{d},\mathbf{b},\mathbf{c}]}{det[\mathbf{a},\mathbf{b},\mathbf{c}]} \\ \Longleftrightarrow \quad & det[\mathbf{a},\mathbf{b},\mathbf{c}] - det[\mathbf{a},\mathbf{d},\mathbf{c}] - det[\mathbf{a},\mathbf{b},\mathbf{d}] &=& \ det[\mathbf{d},\mathbf{b},\mathbf{c}] \\ \Longleftrightarrow \quad & -det[\mathbf{b},\mathbf{c},\mathbf{d}] + det[\mathbf{a},\mathbf{c},\mathbf{d}] - det[\mathbf{a},\mathbf{b},\mathbf{d}] + det[\mathbf{a},\mathbf{b},\mathbf{c}] &=& \ 0 \\ \Longleftrightarrow \quad & -det\begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix} &=& \ 0 \\ \Longleftrightarrow \quad & det\begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix} &=& \ 0 \end{aligned}$$

■

A.2 Beschreibung von Rotationen im $\mathbb{R}^3$

In dieser Arbeit betrachten wir nur Rotationen um eine feste, normierte Achse, die durch den Ursprung des körpereigenen Koordinatensystems verläuft: wir bezeichnen sie mit $\mathbf{r}$. Der Rotationswinkel α bestimmt den Drehwinkel des Objektes um diese Achse $\mathbf{r}$, gemessen im Gegenuhrzeigersinn. Da wir unsere Objekte durch starre Polyeder modellieren, können wir uns bei der Betrachtung von Drehungen solcher Objekte auf die Rotation von Punkten im $\mathbb{R}^3$ zurückziehen. Wollen wir nun einen beliebigen

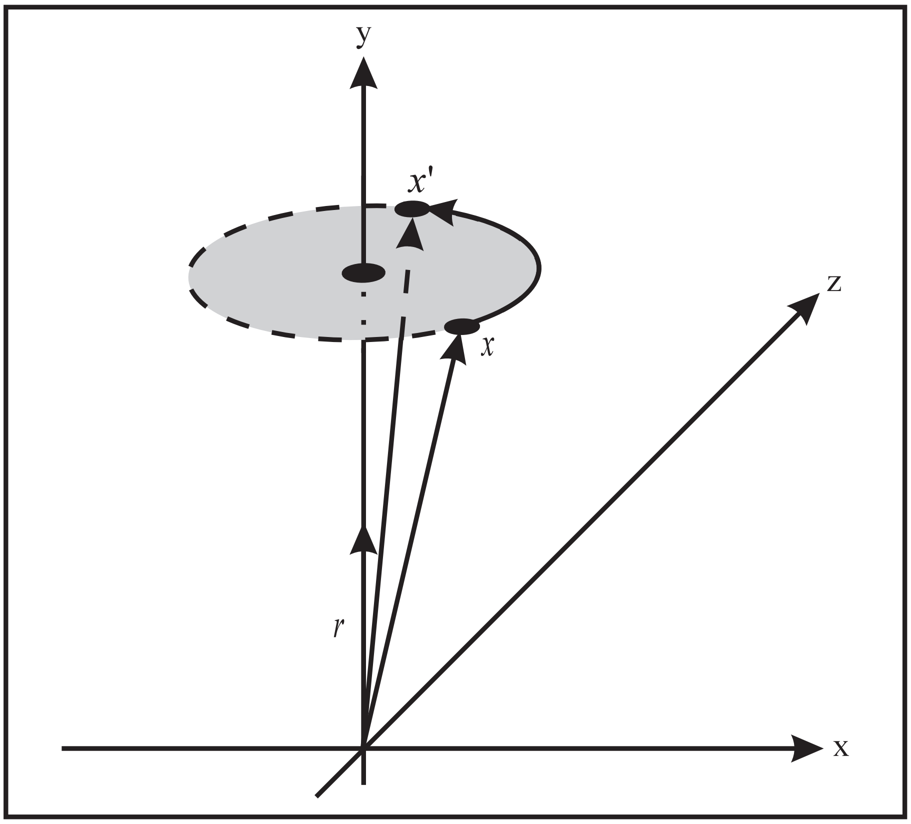

Abbildung A.1: Rotation eines Punktes $\mathbf{x}$ im $\mathbb{R}^3$

Vektor $\mathbf{x}$ um eine Achse $\mathbf{r}$ und den Winkel α drehen, so berechnet sich der daraus hervorgehende Vektor $\mathbf{x}'$, wie man anhand der Abbildungen A.1 und A.2 nachvollziehen kann, folgendermaßen:

$$\begin{aligned} \mathbf{x}' &= (\mathbf{r}^T\mathbf{x})\mathbf{r} + \underbrace{(\mathbf{r}\times\mathbf{x})\times\mathbf{r}\ cos\alpha + (\mathbf{r}\times\mathbf{x})\ sin\alpha}_{\text{Drehung im } \mathbb{R}^2} \\ &= (1-cos\alpha)(\mathbf{r}^T\mathbf{x})\mathbf{r} + \mathbf{x}\ cos\alpha + (\mathbf{r}\times\mathbf{x})\ sin\alpha \qquad \text{(A.7)} \end{aligned}$$

A.3 Bewegte Flächen im $\mathbb{R}^3$

Sei eine Fläche $f_{\mathbf{v_1}\cdots\mathbf{v_s}}$, $s \geq 3$, im $\mathbb{R}^3$ gegeben. Für drei linear unabhängige Vektoren $\mathbf{v_i}$, $\mathbf{v_j}$, $\mathbf{v_k}$, $i, j, k \leq s$, berechnet sich der Normalenvektor der zugehörigen Ebene E_f, wie man sich schnell überlegt, als:

$$\mathbf{n} = \mathbf{v_i}\times\mathbf{v_j} + \mathbf{v_j}\times\mathbf{v_k} + \mathbf{v_k}\times\mathbf{v_i}$$

Der Abstand n_0 der Ebene ergibt sich als

$$n_0 = \mathbf{n}^T \cdot \mathbf{a}$$

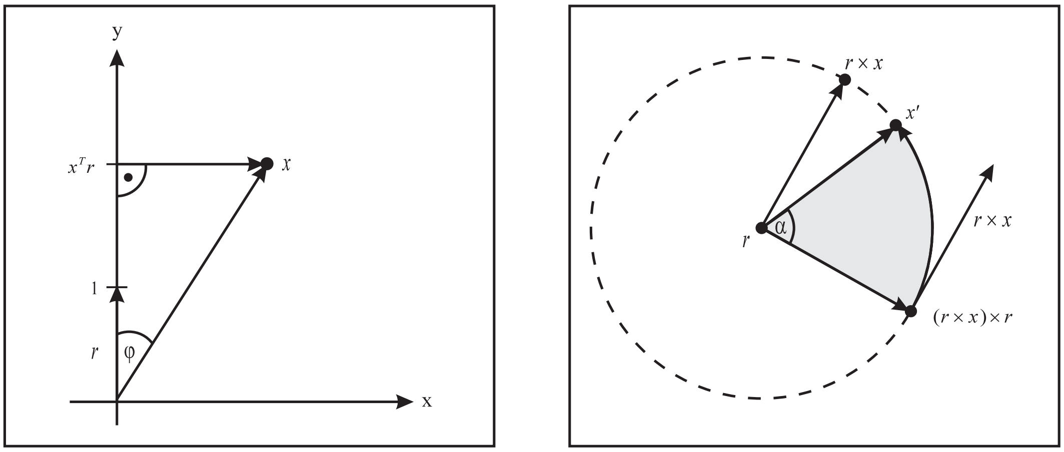

Abbildung A.2: FRONT- UND DRAUFSICHT

für einen Punkt $\mathbf{a} = \lambda \cdot (\mathbf{v_j} - \mathbf{v_i}) + \beta \cdot (\mathbf{v_k} - \mathbf{v_j}) \in E_f$. Die vollständige Ebenengleichung lautet dann:

$$\mathbf{n}^T \cdot (\mathbf{x} - \mathbf{a}) \quad = \quad 0,$$

soweit nichts Neues.

Bewegt sich nun die Fläche $f_{\mathbf{v_1} \cdots \mathbf{v_s}}$ aus ihrer Ausgangsposition in eine Zielposition, so läßt sich gemäß 1.9 die zeitabhängige Position und Orientierung ihrer Eckpunkte beschreiben durch:

$$\mathbf{v_i}(t) \quad = \quad \mathbf{R}(t)\,\mathbf{v_i} + \mathbf{o}(t) \text{ für } 1 \leq i \leq s$$

Der zeitabhängige Normalenvektor $\mathbf{n}(t)$ der bewegten Fläche $f_{\mathbf{v_1} \cdots \mathbf{v_s}}(t)$ berechnet sich dann wie folgt:

$$\begin{aligned}
\mathbf{n}(t) &= \mathbf{v_i}(t) \times \mathbf{v_j}(t) + \mathbf{v_j}(t) \times \mathbf{v_k}(t) + \mathbf{v_k}(t) \times \mathbf{v_i}(t) \\
&= \Big(\mathbf{R}(t)\,\mathbf{v_i} + \mathbf{o}(t)\Big) \times \Big(\mathbf{R}(t)\,\mathbf{v_j} + \mathbf{o}(t)\Big) + \\
&\quad \Big(\mathbf{R}(t)\,\mathbf{v_j} + \mathbf{o}(t)\Big) \times \Big(\mathbf{R}(t)\,\mathbf{v_k} + \mathbf{o}(t)\Big) + \\
&\quad \Big(\mathbf{R}(t)\,\mathbf{v_k} + \mathbf{o}(t)\Big) \times \Big(\mathbf{R}(t)\,\mathbf{v_i} + \mathbf{o}(t)\Big) \\
&= \Big(\mathbf{R}(t)\,\mathbf{v_i}\Big) \times \Big(\mathbf{R}(t)\,\mathbf{v_j}\Big) + \Big(\mathbf{R}(t)\,\mathbf{v_i}\Big) \times \mathbf{o}(t) + \mathbf{o}(t) \times \Big(\mathbf{R}(t)\,\mathbf{v_j}\Big) + \mathbf{o}(t) \times \mathbf{o}(t) + \\
&\quad \Big(\mathbf{R}(t)\,\mathbf{v_j}\Big) \times \Big(\mathbf{R}(t)\,\mathbf{v_k}\Big) + \Big(\mathbf{R}(t)\,\mathbf{v_j}\Big) \times \mathbf{o}(t) + \mathbf{o}(t) \times \Big(\mathbf{R}(t)\,\mathbf{v_j}\Big) + \mathbf{o}(t) \times \mathbf{o}(t) + \\
&\quad \Big(\mathbf{R}(t)\,\mathbf{v_k}\Big) \times \Big(\mathbf{R}(t)\,\mathbf{v_i}\Big) + \Big(\mathbf{R}(t)\,\mathbf{v_k}\Big) \times \mathbf{o}(t) + \mathbf{o}(t) \times \Big(\mathbf{R}(t)\,\mathbf{v_j}\Big) + \mathbf{o}(t) \times \mathbf{o}(t) \\
&= \Big(\mathbf{R}(t)\,\mathbf{v_i}\Big) \times \Big(\mathbf{R}(t)\,\mathbf{v_j}\Big) + \Big(\mathbf{R}(t)\,\mathbf{v_j}\Big) \times \Big(\mathbf{R}(t)\,\mathbf{v_k}\Big) + \Big(\mathbf{R}(t)\,\mathbf{v_k}\Big) \times \Big(\mathbf{R}(t)\,\mathbf{v_i}\Big) \\
&= \mathbf{R}(t) \cdot \Big(\mathbf{v_i} \times \mathbf{v_j} + \mathbf{v_j} \times \mathbf{v_k} + \mathbf{v_k} \times \mathbf{v_i}\Big) \\
&= \mathbf{R}(t) \cdot \mathbf{n}
\end{aligned}$$

Für den zeitlichen Abstand $n_0(t)$ der Ebene E_f gilt:

$$\begin{aligned} n_0(t) &= \mathbf{n}(t)^T \cdot \Big(\mathbf{R}(t)\,\mathbf{a} + \mathbf{o}(t)\Big) \\ &= \Big(\mathbf{R}(t)\,\mathbf{n}\Big)^T \cdot \Big(\mathbf{R}(t)\,\mathbf{a} + \mathbf{o}(t)\Big) \\ &= \mathbf{n}^T\,\mathbf{R}(t)^T\,\mathbf{R}(t)\,\mathbf{a} + \Big(\mathbf{R}(t)\,\mathbf{n}\Big)^T\,\mathbf{o}(t) \\ &= \mathbf{n}^T\,\mathbf{E}\,\mathbf{a} + \mathbf{o}(t)^T\,\Big(\mathbf{R}(t)\,\mathbf{n}\Big) \\ &= n_0 + \mathbf{o}(t)^T\,\Big(\mathbf{R}(t)\,\mathbf{n}\Big) \\ &= n_0 + \mathbf{o}(t)^T \cdot \mathbf{n}(t) \end{aligned}$$

Die vollständige zeitabhängige Beschreibung der zur bewegten Fläche $f_{\mathbf{v_1}\cdots\mathbf{v_S}}(t)$ zugehörigen bewegten Ebene $E_f(t)$ lautet also:

$$\mathbf{n}(t)^T \cdot \mathbf{x} = n_0 + \mathbf{o}(t)^T \cdot \mathbf{n}(t) \tag{A.8}$$

A.4 Sonstiges

Bei der Umformung von Termen mit trigonometrischen Funktionen hat sich die Anwendung folgender Beziehungen als hilfreich erwiesen:

$$cos(\alpha - \beta) = cos\alpha\; cos\beta + sin\alpha\; sin\beta \tag{A.9}$$

$$cos(2\alpha) = cos^2\alpha - sin^2\alpha \tag{A.10}$$

$$sin(2\alpha) = 2\; sin\alpha\; cos\alpha \tag{A.11}$$

$$2\; sin^2\alpha = 1 - cos(2\alpha) \tag{A.12}$$

Anhang B

Ein ausführliches Beispiel

Um dem Leser einen Eindruck von der Art der zu behandelnden polynomiellen Ungleichungssysteme zu vermitteln, werden wir in diesem Abschnitt die verschiedenen Stufen eines Kante-Kante-Kollisionstests einer einfachen Szene mit zwei polyhedral modellierten Objekten angeben: einem Quader und einem Tetraeder wie sie in Abbildung 2.5 zu sehen sind. Die Eingabedaten seien dabei wie folgt gegeben:

- **Quaderdaten:**
  Der Ursprung des körpereigenen Koordinatensystems ist $(0, 0, 0)$, die Eckpunkte des Quaders sind gegeben als:

$$\begin{aligned}
\mathbf{v_0} &= (-3000, -2000, -1000)\\
\mathbf{v_1} &= (-3000, 2000, -1000)\\
\mathbf{v_2} &= (-3000, 2000, 1000)\\
\mathbf{v_3} &= (-3000, -2000, 1000)\\
\mathbf{v_4} &= (3000, -2000, -1000)\\
\mathbf{v_5} &= (3000, 2000, -1000)\\
\mathbf{v_6} &= (3000, 2000, 1000)\\
\mathbf{v_7} &= (3000, -2000, 1000)
\end{aligned}$$

  Anfangs- und Zielkonfiguration seien gegeben durch:

Anfangsposition:	$(0, 0, 1000)$
Zielposition:	$(-1001, 2345, 2876)$
Anfangsorientierung:	$(1000, 0, 0, 0)$
Zielorientierung:	$(1000, 143, -325, 188)$.

- **Tetraeder-Daten:**
  Der Ursprung des körpereigenen Koordinatensystems ist $(0, 0, 0)$, die Eckpunkte des Tetraeders sind gegeben als:

$$\mathbf{v_0} = (1, 1, 1)$$

$$
\begin{aligned}
\mathbf{v_1} &= (-1500, 1299, -2598) \\
\mathbf{v_2} &= (1500, 1299, -2598) \\
\mathbf{v_3} &= (1, -1299, -2598)
\end{aligned}
$$

Anfangs- und Zielkonfiguration seien gegeben durch:

Anfangsposition:	$(3000, 2000, -2000)$
Zielposition:	$(2345, 594, -1432)$
Anfangsorientierung:	$(1000, 0, 0, 0)$
Zielorientierung:	$(1000, -45, 551, -266)$.

Im folgenden werden wir den Kante-Kante-Kollisionstest für zwei bewegte Liniensegmente $l_{\mathbf{ab}}(t)$ (Quader) und $l_{\mathbf{cd}}(t)$ (Tetraeder) untersuchen; diese seien gegeben durch:

$$
\begin{aligned}
\text{Punkt } \mathbf{a} &: (-3000, -2000, -1000) \\
\text{Punkt } \mathbf{b} &: (-3000, 2000, -1000) \\
\text{Punkt } \mathbf{c} &: (1, 1, 1) \\
\text{Punkt } \mathbf{d} &: (-1500, 1299, -2598)
\end{aligned}
$$

Damit berechnet sich die zeitabhängige Orientierungsmatrix des Quaders zu:

$$
\mathbf{R'_1}(t) = \begin{pmatrix} -120520t^2 + 1000000 & -92950t^2 - 376000t & 53768t^2 - 650000t \\ -92950t^2 + 376000t & 49832t^2 + 1000000 & -122200t^2 - 286000t \\ 53768t^2 + 650000t & -122200t^2 + 286000t & -90730t^2 + 1000000 \end{pmatrix}
$$

Ihr Nenner lautet:

$$
N_1(t) = 161418t^2 + 1000000
$$

Für die Orientierungsmatrix des Tetraeders ergibt sich:

$$
\mathbf{R'_2}(t) = \begin{pmatrix} -372332t^2 + 1000000 & -49590t^2 + 532000t & 23940t^2 + 1102000t \\ -49590t^2 - 532000t & 230820t^2 + 1000000 & -293132t^2 + 90000t \\ 23940t^2 - 1102000t & -293132t^2 - 90000t & -234870t^2 + 1000000 \end{pmatrix}
$$

Der dazugehörige Nenner berechnet sich als:

$$
N_2(t) = 376382t^2 + 1000000
$$

Der zeitabhängige Positionsvektor des Quaders lautet

$$
\mathbf{o_1}(t) = \begin{pmatrix} -1001t \\ 2345t \\ 1876t + 1000 \end{pmatrix},
$$

der des Tetraeders

$$
\mathbf{o_2}(t) = \begin{pmatrix} -655t + 3000 \\ -1406t + 2000 \\ 568t - 2000 \end{pmatrix}
$$

Das aus diesen Daten resultierende polynomielle Ungleichungssystem, das die Kollisionsbedingungen für die beiden bewegten Objekte im Zeitintervall $(0,1)$ beschreibt, lautet dann:

$$\begin{aligned}
f(t) &= -139260899875094760000 \cdot t^5 + 536433959856711561600 \cdot t^4 + \\
&\quad 217990984653939200000 \cdot t^3 - 331555497744480000000 \cdot t^2 - \\
&\quad 1024569240000000000000 \cdot t + 743883920000000000000 \\
g_1(t) &= 788222881976768000 \cdot t^5 - 1333540895880064000 \cdot t^4 + \\
&\quad 1989148496688000000 \cdot t^3 - 531585528000000000 \cdot t^2 - \\
&\quad 274088000000000000 \cdot t + 799600000000000000 \\
g_2(t) &= -788222881976768000 \cdot t^5 + 1927434645400576000 \cdot t^4 - \\
&\quad 2706194953184000000 \cdot t^3 + 778098304000000000 \cdot t^2 + \\
&\quad 4938304000000000000 \cdot t - 1839200000000000000 \\
g_3(t) &= 122357971504419420 \cdot t^5 - 59676966099217608 \cdot t^4 + \\
&\quad 309487818679012000 \cdot t^3 - 1310321213902000000 \cdot t^2 + \\
&\quad 13266759634000000000 \cdot t - 780389700000000000 \\
g_4(t) &= -122357971504419420 \cdot t^5 + 653570715619729608 \cdot t^4 - \\
&\quad 1026534275175012000 \cdot t^3 + 1556833989902000000 \cdot t^2 - \\
&\quad 8602543634000000000 \cdot t - 259210300000000000
\end{aligned}$$

Das Polynom $A(t) = f(t) \cdot g_1(t)$, dessen Nullstellen mittels des rekursiven Uspensky-Verfahrens separiert werden, ergibt sich dann zu:

$$\begin{aligned}
A(t) &= 10^{36} \cdot \\
&\quad (-1295919222529015187823285159408551540839822910501683200 \cdot t^{10} + \\
&\quad 7184367845212827376792103134804535747800127683178790912 \cdot t^9 - \\
&\quad 9687247155293285975823183466789347269336860379920203776 \cdot t^8 - \\
&\quad 20814070037615915914961624488047965986567901381918720000 \cdot t^7 + \\
&\quad 44867970003959329874539522816934894737510241402880000000 \cdot t^6 - \\
&\quad 8757676614065500917201314699432016352580206592000000000 \cdot t^5 - \\
&\quad 70433368626367298184086631352048496721002496000000000000 \cdot t^4 + \\
&\quad 212428711261811662976876524711223490510848000000000000000 \cdot t^3 - \\
&\quad 356358448953844424190002454773907849216000000000000000000 \cdot t^2 - \\
&\quad 120790486779233793199561480710651904000000000000000000000 \cdot t^1 + \\
&\quad 702227208941641537661626291847168000000000000000000000000)
\end{aligned}$$

Um mittels des Satzes von Descartes Aussagen über die Existenz von Nullstellen im Intervall $(0,1)$ zu machen, ist zunächst eine Transformation gemäß Lemma 3.12 erforderlich, die die Nullstellen des Polynoms im Intervall $(0,1)$ zu positiven reellen Nullstellen transformiert:

$$
\begin{aligned}
A^*(y) \quad = \quad & 10^{36} \cdot \\
& (70222720894164153766162629184716800000000000000000000000000 \cdot y^{10} + \\
& 690148160263718158341670143776102809600000000000000000000000 \cdot y^{9} + \\
& 3015675157240692063178712735195278501478400000000000000000000 \cdot y^{8} + \\
& 7728036866857562423366779860259665215278284800000000000000000 \cdot y^{7} + \\
& 12875984402778775474551233721616761622280693350400000000000000 \cdot y^{6} + \\
& 145815227225820697425437627424729326843953212817408000000000 \cdot y^{5} + \\
& 1136826050684936858292145147281214601809192517684428800000000 \cdot y^{4} + \\
& 602622096075996867709119822322637841908013010783096012800000 \cdot y^{3} + \\
& 207877343113313041483765245755009934875349893779716036362240 \cdot y^{2} + \\
& 4209742044214322018626218219258484963958483659429102223360 \cdot y + \\
& 37857104066367612432037446785984478990983108182171818393 6)
\end{aligned}
$$

Die Anzahl der Vorzeichenwechsel in der Folge der Koeffizienten des transformierten Polynoms ist gleich null, woraus gemäß Descartes gefolgert werden kann, daß dieses Polynom im Intervall $(0, 1)$ keine Nullstellen besitzt. Da nun schon für das erste untersuchte Polynom festegestellt werden konnte, daß es keine Nullstellen im Intervall $(0, 1)$ besitzt, kann das gesamte Gleichungssystem folgerichtig dort ebenfalls keine Lösung besitzen. Aufgrund dieser Tatsache brauchen die symmetrischen Fälle $(A(t) = -f(t) \cdot g_i(t))$ nicht betrachtet zu werden, so daß wir gleich zum nächsten Kante-Kante-Test übergehen können.

Die Länge der Koeffizienten der im Verlauf der rekursiven Anwendung des Uspensky-Verfahrens zu bearbeitenden Polynome ist wesentlich abhängig von der Anzahl der gemäß 3.13 durchzuführenden Intervallhalbierungen. In obigem Beispiel brauchte keine solche Transformation durchgeführt zu werden. Aus Platzgründen möchten wir an dieser Stelle auf die Angabe eines aus der Mehrfachanwendung von Transformation 3.13 hervorgegangenen Polynoms verzichten.

Literaturverzeichnis

[BK95] Bohne / Klix
Geometrie - Grundlagen für Anwendungen
Fachbuchverlag Leipzig-Köln, (1995)

[Can87] John F. Canny
The Complexity of Robot Motion Planning
MIT-Press, (1987)

[Cha95] Thomas Chadzelek
Heuristische Bewegungsplanung mit vielen Freiheitsgraden
Diplomarbeit an der Universität des Saarlandes, (1995)

[CL82] G.E. Collins / R. Loos
Real Zeros of Polynomials
Computing Supplementum 4, pp. 83-94, (1982)

[CLR90] T.H. Cormen / C.E Leiserson / R.L. Rivest
Introduction to Algorithms
MIT-Press, (1990)

[Cra89] John J. Craig
Introduction to Robotics
Addison-Wesley Publishing Company, (1989)

[Eck95] Jens Eckstein
Heuristische Bewegungsplanungsstrategien im $\mathbb{R}^3$
Diplomarbeit an der Universität des Saarlandes, (1995)

[FPT88] B.P. Flannery / W.H. Press / S.A. Teukolsky / W.T Vetterling
Numerical Recipes in C
Cambridge University Press, (1988)

[Gan59] F.R. Gantmacher
The Theory of Matrices
Chelsea, New York, (1959)

[Lat91] Jean Claude Latombe
Robot Motion Planning
Kluwer Academic Publishers, (1991)

[MN94] Kurt Mehlhorn, Stefan Näher
The Implementation of Geometric Algorithms
Proceedings of the IFIP 13th World Computer Congress. Volume 1 : Technology and Foundations, (1994)

[Nee91] Francis Neelamkavil
Computer Simulation and Modelling
John Wiley & Sons, (1991)

[Ped91] P. Pedersen
Counting Real Zeros
New York University, NYU Technical Report 545-R243, (1991)

[RSV83] H.-J. Reiffen / G. Scheja / U. Vetter
Algebra
BI-Hochschultaschenbücher, (1983)

[Sch94] Elmar Schömer
Interaktive Montageplanung mit Kollisionserkennung
Dissertation im Fachbereich Informatik, Universität des Saarlandes, (1994)

[SSW95] Elmar Schömer / Jürgen Sellen / Markus Welsch
Exact Geometric Collision Detection
Proceedings der 7th Canadian Conference on Computational Geometry, pp. 211-216, (1995)

[US92] A.Y. Uteshev / S.G. Shulyak
Hermite's Method of Separation of Solutions of Systems of Algebraic Equations and its Applications
Linear Algebra and its Applications 177, pp. 49-88, (1992)

[Yap93] Chee-Keng Yap
Towards Exact Geometric Computation
Proceedings der 5th Canadian Conference on Computational Geometry, pp. 405-419, (1993)

[YD93] Chee-Keng Yap, Thomas Dubé
The Exact Computation Paradigm
In Computing in Euclidean Geometry (eds. Du and Hwang, World Scientific Press, 2nd Edition), (1995)

[YD93a] Chee-Keng Yap, Thomas Dubé
A Basis for Implementing Exact Geometric Algorithms
Extended Abstract, (1993)

[Yap94] Chee-Keng Yap
Fundamental Problems in Algorithmic Algebra
Princeton University Press, (1994) (soll erscheinen)