Interaktive Montageplanung mit Kollisionserkennung

Dissertation

zur Erlangung des Grades des
Doktors der Naturwissenschaften
der Technischen Fakultät
der Universität des Saarlandes

von

Elmar Schömer

Saarbrücken
1994

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Simulationen benutzt man in Wissenschaft und Technik, um auf der Basis von geeigneten Modellen Prognosen für die Zukunft zu stellen oder umgekehrt Ereignisse, die in der Vergangenheit liegen, zu rekonstruieren und ihre Ursachen zu erforschen. Je genauer ein System modelliert ist, um so größer ist die Aussagekraft der Simulationsresultate. Durch den Vergleich von Simulation und Wirklichkeit kann die Güte eines Modells oder die Gültigkeit einer Theorie überprüft werden. Simulationen helfen dem Wissenschaftler, das Verhalten komplexer Systeme besser zu verstehen und damit fundiertere Entscheidungen zu treffen, um regulierend in das System einzugreifen oder ein gewünschtes Systemverhalten herbeizuführen. Oft muß man unter großem technischen Aufwand ein prototypisches System erstellen, das möglichst realen Bedingungen ausgesetzt wird, um dadurch Rückschlüsse auf das Verhalten des Originalsystems ziehen zu können. Vielfach erlauben jedoch erst die Möglichkeiten einer computerbasierten Simulation, das Verhalten eines Systems realistisch nachzubilden und zu analysieren. Eine Softwarelösung besitzt gegenüber der Erstellung von Prototypen eine sehr viel höhere Flexibilität, weil eine leichte Manipulierbarkeit von Systemparametern gewährleistet ist. Außerdem sprechen oft Gefahren–, Zeit– oder Kostengründe für eine rein computerbasierte Simulation.

Simulationsprogramme werden in vielen Bereichen eingesetzt: In der Meteorologie zur Vorhersage des Wetters oder der zukünftigen Klimaentwicklung, in der Ökologie zu Untersuchungen über Schadstoffverteilungen in der Umwelt, in der Ökonomie zur Simulation von Wirtschaftskreisläufen, in der Chemie zur gezielten Entwicklung neuer Substanzen, in der Medizin zur Planung von (mikro)chirurgischen Operationen, in der Automobil–, Luft– und Raumfahrtindustrie zur Simulation des Fahr– bzw. Flugverhaltens und – um die kleine Auswahl an Beispielen abzuschließen – in der Konstruktions– und Fertigungstechnik zur Entwicklung neuer Produkte und zur Planung ihrer Herstellung. Die vorliegende Arbeit möchte einen Beitrag zu dem zuletzt genannten Problemkreis leisten.

Gestiegene Qualitäts– und Sicherheitsanforderungen, eine Verkürzung der Produktzykluszeiten und der zunehmende Grad der Automatisierung bei der Produktion haben im Bereich der Konstruktion und Fertigung zu einem erhöhten Planungsbedarf während der Phase des Entwurfs und der *Produktionsplanung* geführt. Ein wichtiges Hilfsmittel zur Deckung dieses Bedarfs stellt das sogenannte “software prototyping” dar.
Schon während der Entwurfsphase versucht man, mit Simulationsprogrammen das physikalische Verhalten des zu entwickelnden Produktes zu analysieren, um so früh wie möglich Entwurfsfehler aufzudecken, die die Funktionsfähigkeit des Produktes beeinträchtigen könnten. Auf diese Weise erhält der Entwicklungsingenieur wertvolle Hinweise, wie einzelne Bautei-

le des Produktes auszulegen sind, damit ein optimales Zusammenspiel aller Komponenten erzielt wird. Beim Entwurf ist aber nicht nur darauf zu achten, daß Bauteile richtig zusammenpassen und ein funktionierendes Ganzes ergeben, sondern es muß auch die *Montierbarkeit* der Einzelteile garantiert sein. Zur Vereinfachung einer späteren Reparatur muß der Ingenieur zusätzlich auch die Demontage in seine Überlegungen mit einbeziehen. Aspekte der Montage und Demontage spielen eine um so größere Rolle, je stärker die Automatisierung der Produktionsprozesse voranschreitet. Soll ein Produkt zum Beispiel mit der Unterstützung von Robotern in einer Fertigungsstraße montiert werden, so ist es unbedingt erforderlich, daß seine Bauteile leicht zu handhaben und zu montieren sind. Allein schon die geometrische Auslegung der Teile und ihre Anordnung sind ausschlaggebend für die Kompliziertheit einer Montage. Aber auch der Einsatz spezieller Werkzeuge und Maschinen kann zu besonderen Formvorgaben für die Montageteile führen, damit eine maschinelle Montage beschleunigt oder überhaupt erst ermöglicht wird. Nur die Aufhebung der strikten Trennung zwischen der Entwurfsphase und der Produktionsplanung schafft die Voraussetzung für wirklich *montagegerechtes Design*, das erheblich zu einer Minderung der Produktionskosten beitragen kann.

Die Motivation für diese Arbeit war die Entwicklung eines *Systems zur Montageplanung und –simulation* für den Einsatz in der Produktionsplanung. Das System wurde am Lehrstuhl von Prof. Hotz im Rahmen der vorliegenden Arbeit konzipiert und in Fortgeschrittenenpraktika und Diplomarbeiten als Prototyp realisiert. Es dient dazu, ein Schema für den Ablauf einer Montage zu erstellen, aus dem die genaue Montagereihenfolge und die zeitliche und räumliche Koordination der Montageschritte ersichtlich ist. Es soll den Ingenieur dabei unterstützen, eine geeignete Auswahl von Montagewerkzeugen zu treffen, eingesetzte Montagevorrichtungen und –roboter günstig zu plazieren und die zur Montage benötigten Bewegungsabläufe vernünftig zu organisieren. Mit Hilfe einer Simulation kann so die prinzipielle Durchführbarkeit einer Montage zeit– und kostensparend unter Beweis gestellt werden, ohne daß eine prototypische Anlage aufgebaut werden müßte. Außerdem können zu Optimierungszwecken sehr schnell verschiedene Varianten durchgespielt werden. So können auftauchende Probleme rechtzeitig identifiziert werden, und falls notwendig kann ein Redesign kritischer Bauteile erwogen werden.

Die Intention unseres Systems zur Montageplanung und –simulation ist nicht die automatische Generierung von Montageplänen allein aus der Beschreibung der Bauteile und ihrer Anordnung im fertigen Produkt. Diese Aufgabe ist nur für sehr einfache Problemstellungen mit niedriger Komplexität und einer geringen Zahl von Freiheitsgraden algorithmisch beherrschbar. Der Ingenieur soll vielmehr voll in den Planungsprozeß integriert sein. Aufgrund seiner Erfahrung und seines Wissens über das Entwurfsziel hat er schon eine grobe Vorstellung über einen denkbaren Ablauf der Montage. In *Interaktion* mit dem Planungssystem kann er seine Ideen präzisieren und in konkrete Montagepläne umsetzen. Bei der *interaktiven Erstellung der Montagepläne* und bei der Koordination der notwendigen Bewegungssequenzen für eine robotergestützte Montage kann das System ihm eine Vielzahl von Arbeiten abnehmen: Es erlaubt ihm eine komfortable Spezifikation der Trajektorien für das Zusammenfügen der Bauteile. Bei der Roboterprogrammierung nimmt es ihm zum Beispiel die Berechnung von Gelenkparametern zum Erreichen von Zielpositionen ab und hilft ihm, die *Kollisionsfreiheit* und die Einhaltung von Sicherheitsabständen zu verifizieren.

Die *Kollisionserkennung* bildet den Kern unseres Systems zur Montageplanung und –simulation, denn ohne diese Komponente wäre der Ingenieur auf eine bloße optische Kontrolle angewiesen, um die Durchführbarkeit eines geplanten Montageablaufs zu überprüfen. Ein gängiger Ansatz zur Kollisionskontrolle besteht darin, in diskreten Zeitabständen zu testen, ob sich zwei Objekte durchdringen. (Die notwendigen Schnittberechnungen können vom CAD–System durchgeführt werden.) Bei einer zu groben Wahl des Zeitrasters besteht jedoch die Gefahr, daß Kollisionen übersehen werden. Ein zu feines Zeitraster hat hingegen den Nachteil, daß das Verfahren an Effizienz verliert. Eine Alternative besteht darin, zuerst das vom bewegten Objekt überstrichene Volumen zu konstruieren und anschließend den Schnitt zu berechnen. Aber selbst bei sehr einfachen Bewegungen wie Rotationen entstehen dadurch kompliziert zu beschreibende Volumina, die nur in approximierter Form effizient zu handhaben sind. Gerade bei einer interaktiven Arbeitsweise mit dem System ist eine effiziente Realisierung aber entscheidend für die Akzeptanz des Systems.

Deshalb ist eines der Hauptziele dieser Arbeit die Entwicklung *effizienter Algorithmen zur Kollisionserkennung*. Im Gegensatz zu den oben genannten Methoden verzichten wir auf Schnittests und suchen bei vorgegebener Bewegung der Objekte direkt nach möglichen Kontaktsituationen. Dazu verwenden wir Techniken aus dem Bereich der *algorithmischen Geometrie*. Wir können praktikable effiziente Algorithmen und interessante theoretische Resultate gewinnen, wenn wir von folgenden Modellannahmen ausgehen: "Die Objekte sind starre Körper, deren Oberfläche aus ebenen, geradlinig begrenzten Flächen besteht. Ein Objekt darf in jedem Schritt in eine beliebige Richtung verschoben oder um eine beliebige Achse gedreht werden." Diese Annahmen sind insofern gerechtfertigt, als reale Objekte durch ebenbegrenzte Körper leicht modelliert werden können und ihre Bewegungen sich durch Folgen von Translationen und Rotationen approximieren lassen. Außerdem spielen diese elementaren Bewegungsformen wegen ihrer einfachen technischen Realisierbarkeit bei Montagevorgängen eine zentrale Rolle.

Bei Bewegungsabläufen eines Montageroboters entstehen durch Überlagerung jedoch weitaus kompliziertere Bewegungen, so daß wir zur Kollisionsberechnung auf approximative, numerische Verfahren ausweichen müssen. Dabei ergibt sich die Schwierigkeit, die entstehenden Approximationsfehler abzuschätzen, um die Kollisionsfreiheit eines Bewegungsablaufs mit Sicherheit nachweisen zu können. Anstatt direkt nach Kontakten zwischen den bewegten Körpern zu suchen, kontrollieren wir eine zu große Annäherung zwischen zwei Objekten, indem wir in Abhängigkeit ihrer zurückgelegten Entfernungen immer wieder ihre momentanen Abstände berechnen. Die Tragfähigkeit dieses Konzeptes hinsichtlich der Behandlung von Approximationsfehlern demonstrieren wir am Beispiel von Bewegungssequenzen für Roboter. Die Roboter betrachten wir dabei als gelenkig verbundene starre Körper mit wohldefiniertem kinematischen Verhalten.

Effiziente Methoden zur Kollisionserkennung sind nicht nur bei der Montage– und Robotersimulation von Bedeutung, sondern ganz allgemein bei der Simulation mechanischer Vorgänge. Sie bilden die Voraussetzung für die Simulation der Dynamik eines Mechanismus. Diese Arbeit zeigt neue Wege zu einer effizienten Kollisionskontrolle und zu ihrer Anwendung im Zusammenhang mit der Montageplanung.

Gliederung und Resultate der Arbeit

Im ersten Kapitel machen wir uns zunächst anhand eines einfachen Modells mit der Problematik der Kollisionserkennung vertraut. Die Objekte betrachten wir als starre Körper und modellieren sie durch Polyeder. Als Bewegungen lassen wir nur Translationen und Rotationen zu. Es ist bekannt (vgl. [Bo79, Ca84]), daß unter diesen Bedingungen eine Kollision zwischen einem bewegten Objekt und einem stationären Hindernis in Zeit $O(n^2)$ erkannt werden kann. n mißt hierbei die Komplexität der beteiligten Polyeder in der Zahl ihrer Ecken, Kanten und Flächen. Indem wir die Invarianten der jeweiligen Bewegungsform nutzen, können wir eine Beschleunigung des einfachen quadratischen Algorithmus erreichen. Dazu stellen wir Techniken vor, die in einheitlicher Weise für Translationen und Rotationen angewendet werden können. Sie basieren auf dem Prinzip, die Dimension des Problems durch adäquate Projektionen zu reduzieren. Die Laufzeit der Kollisionstests ist sensitiv gegenüber der Analysekomplexität der projizierten Szene aus den Polyedern, und ihr asymptotisches Laufzeitverhalten ist nur noch im Worst–Case quadratisch. Für den Spezialfall von 360°–Drehungen entpuppt sich das Problem der Kollisionserkennung sogar als rein zweidimensionales Problem, das als solches eine effizientere Lösung besitzt.

In der Praxis sind nicht nur echte Kollisionen zwischen dem bewegten Objekt und den Hindernissen, sondern auch die während einer Bewegung eingehaltenen Sicherheitsabstände von Interesse. Im zweiten Kapitel beschäftigen wir uns daher näher mit den mathematischen Grundlagen zur Berechnung von Kollisionen und Abständen zwischen translatorisch oder rotatorisch bewegten Polyedern. Dazu untersuchen wir, welche Operationen notwendig sind, um die entsprechenden Fragen für die Eckpunkte, Kanten und Flächen der Polyeder zu beantworten. Das wesentliche Resultat dieses Kapitels besteht darin, daß unter den genannten Voraussetzungen sowohl Kollisionszeitpunkte als auch Sicherheitsabstände algebraisch exakt als Nullstellen von Polynomen höchstens vierten Grades bestimmt werden können. Es kann also vollständig auf numerische Approximationsverfahren verzichtet werden, was sich günstig auf die Laufzeit der Algorithmen auswirkt. Bei der Herleitung der Lösungen legen wir besonderen Wert auf eine kompakte und symmetrische Formulierung der Ergebnisse, denn dies liefert nicht nur Hinweise auf die Korrektheit sondern fördert auch die effiziente Auswertbarkeit der Formeln.

Im Mittelpunkt des dritten Kapitels stehen die theoretischen Resultate zur effizienten Berechnung von Kollisionen zwischen translatorisch bewegten Polyedern. Dobkin und Kirkpatrick haben in [DK85, DK90] dargelegt, daß sich Abstandsprobleme für konvexe Polyeder mit Hilfe einer hierarchischen Repräsentation besonders effizient behandeln lassen. Wir können zeigen, daß mit ihrer Datenstruktur Kollisionen zwischen zwei konvexen Polyedern in Zeit $O(\log^2 n)$ erkannt werden können, wenn sich diese nur translatorisch bewegen. Ist eines der Polyeder nicht konvex, so kann man noch eine Laufzeit von $O(n \log n)$ erzielen. Den zur Kollisionsberechnung verwandten Algorithmus können wir interessanterweise auch benutzen, um für das Problem des "kleinsten Aussichtsturmes" gegenüber [Sh88a] eine Laufzeitverbesserung um einen logarithmischen Faktor zu erreichen.
Konvexe Polyeder sind nicht der einzige Spezialfall, für den effiziente Algorithmen zur Kollisionsberechnung existieren. Wenn wir voraussetzen, daß die Kanten der Polyeder nur in konstant (c) vielen Richtungen verlaufen, können wir die Kollision zwischen zwei translato-

risch bewegten Polyedern in Zeit $O(c^2\, n \log^2 n)$ erkennen.
Für zwei beliebige Polyeder können wir zeigen, daß das translatorische Kollisionsproblem in Zeit $o(n^2)$ lösbar ist. Unser Beweis baut auf dem einfachen Point–Location Algorithmus von [DL76] auf und führt unsere Fragestellung auf ein spezielles Strahlverfolgungsproblem in einem höherdimensionalen Arrangement von Hyperebenen zurück. Auf diese Weise gelingt es, auf elementarem Weg die lange Zeit offene Frage nach der Existenz eines subquadratischen Algorithmus für das translatorische Problem zu beantworten. Dies läßt hoffen, daß ein analoges Ergebnis auch im Fall von Rotationen möglich ist, und daß nicht nur die asymptotische Laufzeit sondern auch die Praktikabilität der Algorithmen verbessert werden kann.

Bei der Planung von Montageprozessen und ihrer Kollisionsüberwachung hat man es im allgemeinen nicht nur mit einfachen Translationen und Rotationen zu tun, sondern mit sehr viel komplexeren Bewegungen, die sich – wie im Fall der Bewegungsabläufe von Montagerobotern – durch eine Superposition einfacher Translations– und Rotationsbewegungen ergeben. Das vierte Kapitel beschäftigt sich mit einer Methode zum Nachweis der Kollisionsfreiheit solcher Bewegungsabläufe, die auf der wiederholten Berechnung von Abständen und Trajektorienlängen beruht (vgl. [Sch92]). Um Abstandsberechnungen einzusparen, verwenden wir für alle Objekte Hüllkörper, die wir mit Hilfe numerischer Methoden bestimmen. Zur Berechnung der zurückgelegten Wegstrecken sind numerische Verfahren sogar unumgänglich. Um die Kollisionsfreiheit einer Bewegungssequenz unter diesen Umständen sicher beurteilen zu können, ist eine Analyse des Fehlerverhaltens der Verfahren unverzichtbar. Deshalb zeigen wir einen systematischen Weg zur Gewinnung von a priori Fehlerabschätzungen auf, die es gestatten, alle während der Bewegung eingehaltenen Sicherheitsabstände mit der benötigten Präzision zu bestimmen.

Im fünften Kapitel gehen wir schließlich genauer auf unser System zur Montageplanung und –simulation ein. Wir stellen die Konzeption des Systems vor und erläutern seinen modularen Aufbau und das wechselseitige Zusammenspiel der Systemkomponenten. Dabei gehen wir auch auf Fragen der Visualisierung und Animation ein. Den Schwerpunkt bildet die Beschreibung der Benutzeroberfläche zur interaktiven Spezifikation von Bewegungsabläufen und zur graphischen Erstellung von Roboterprogrammen, denn eine benutzerfreundliche Oberfläche ist für das interaktive Arbeiten mit dem System unerläßlich.
Zum Umfang des Systems gehört auch ein Modul zur automatischen Bewegungsplanung, das dem Ingenieur zumindest in einfachen Montagesituationen die Planung von Zwischenschritten abnehmen kann. Dazu verwenden wir einfache Heuristiken, die bei Vorgabe von Start– und Zielkonfiguration eine Bewegungsplanung für einzelne Objekte mit rein translatorischen Freiheitsgraden selbständig durchführen können.

Abschließend wird ein Ausblick auf Fragestellungen gegeben, die eine sinnvolle Fortführung der in dieser Arbeit behandelten Themen darstellen.

Danksagungen

Mein besonderer Dank gilt Professor Dr. Günter Hotz, der meine wissenschaftliche Ausbildung vom ersten Studiensemester an prägte. Er gab den Anstoß zu dieser Arbeit und unterstützte ihre Entstehung durch zahlreiche Ratschläge und Ermutigungen.
Weiterhin möchte ich Dr. Joachim Hartmann, Dr. Hans Georg Osthof und Dr. Gisela Pitsch für das Korrekturlesen danken. Ganz besonders möchte ich das große Engagement und den Teamgeist aller Studenten loben, die zur Verwirklichung des Projektes beigetragen haben: Thomas Chadzelek, Wolfgang Collet, Jens Eckstein, Markus Fritz und Markus Sinnwell.

Saarbrücken, im April 1994

Kapitel 1

Kollisionen und Sicherheitsabstände

Wir beginnen mit einer Hinführung auf das Problem, Kollisionen und Sicherheitsabstände zwischen bewegten Objekten im $\mathbb{R}^3$ zu berechnen. Dabei gehen wir von den Modellannahmen aus, daß die Objekte durch starre, polyederförmige Körper und die Bewegungen durch Translationen und Rotationen beschrieben werden. Der Einfachheit halber nehmen wir weiterhin an, daß sich zu jedem Zeitpunkt nur ein Körper bewegt. Eine Klassifikation der Kontaktmöglichkeiten zwischen zwei Polyedern liefert direkt einfache Algorithmen für die betrachteten Problemstellungen. Die Laufzeit dieser Algorithmen ist jedoch quadratisch bezüglich der Beschreibungskomplexität der Polyeder. Deshalb schlagen wir für die Berechnung von Kollisionszeitpunkten eine Reihe von Techniken zur Verbesserung der Laufzeit vor. Die vorgestellten Verfahren können in gleicher Weise für beide Bewegungsarten angewendet werden, wenn man für Translationen ein kartesisches und für Rotationen ein zylindrisches Koordinatensystem zugrunde legt. Durch eine geeignete Projektion der Flächen, Kanten und Eckpunkte der Polyeder entsteht im $\mathbb{R}^2$ ein Arrangement von sich schneidenden Linien(Hyperbel)segmenten, das mit Hilfe eines speziellen Sweep–Algorithmus analysiert werden kann, um den gesuchten Kollisionszeitpunkt zu bestimmen. Die Laufzeit der resultierenden Algorithmen ist davon abhängig, wie schnell Punkte in dem Arrangement lokalisiert werden können und wie effizient Schnitte zwischen den projizierten Kanten der beiden Polyeder bestimmt werden können.
Mit Hilfe der Projektionstechnik können wir zeigen, daß es in subquadratischer Zeit möglich ist zu entscheiden, ob ein Polyeder sich auf einer Gerade kollisionsfrei bewegen kann oder ob es eine Volldrehung um eine vorgegebene Achse ausführen kann, ohne mit einem Hindernis zusammenzustoßen.

1.1 Die allgemeine Problemstellung

Wir betrachten eine Punktmenge $X \subset \mathbb{R}^3$, die einer zeitlichen Veränderung unterworfen ist. Diese Veränderung beschreiben wir durch Angabe der zeitabhängigen Ortsvektoren $\mathbf{x}(t)$ für alle Punkte $\mathbf{x} \in X$. Der Zeitparameter t durchläuft dabei das Intervall $[0, T]$. Den Punkt $\mathbf{x}(0)$ identifizieren wir mit $\mathbf{x} \equiv \mathbf{x}(0)$. $X(t) = \{\mathbf{x}(t) \mid \mathbf{x} \in X\}$ gibt die Lage der Punktmenge X zum Zeitpunkt t wieder. Für zwei Punktmengen $X_1, X_2 \subset \mathbb{R}^3$ bezeichnen wir mit $\delta(X_1, X_2)$

den minimalen Abstand zwischen X_1 und X_2:

$$\delta(X_1, X_2) := \inf\{|\mathbf{x_1} - \mathbf{x_2}| \,|\, \mathbf{x_i} \in X_i\}$$

Diese Bezeichnungen genügen zur Formulierung unseres allgemeinen Problems. Gegeben seien n zeitabhängige Punktmengen $X_1, X_2, \ldots, X_n \subset \mathbb{R}^3$. Wir interessieren uns im wesentlichen für zwei Fragen:

- Wann kollidieren zwei Punktmengen erstmals?

$$t_{col}(X_1, X_2, \ldots, X_n) := \begin{cases} \infty \quad \text{falls} \quad X_i(t) \cap X_j(t) = \emptyset \quad \forall\, t \geq 0, i \neq j \\ \inf\{t \geq 0 \,|\, \exists i \neq j : X_i(t) \cap X_j(t) \neq \emptyset\} \quad \text{sonst} \end{cases}$$

- Wie nahe können sich zwei Punktmengen kommen?

$$\Delta_{min}(X_1, X_2, \ldots, X_n) := \inf\{\delta(X_i(t), X_j(t)) \,|\, i \neq j, t \in [0, T]\}$$

Wir können bereits einige einfache Eigenschaften der *symmetrischen* Abbildungen t_{col} und Δ_{min} ableiten:

$$\begin{aligned} t_{col}(X_1, X_2, \ldots, X_n) &= \min\{t_{col}(X_i, X_j) \,|\, i \neq j\} \\ \Delta_{min}(X_1, X_2, \ldots, X_n) &= \min\{\Delta_{min}(X_i, X_j) \,|\, i \neq j\} \\ t_{col}(X_1 \cup X_2, X_3) &= \min\{t_{col}(X_1, X_3), t_{col}(X_2, X_3)\} \\ \Delta_{min}(X_1 \cup X_2, X_3) &= \min\{\Delta_{min}(X_1, X_3), \Delta_{min}(X_2, X_3)\} \end{aligned}$$

Die Punktmengen sollen real vorkommende Körper modellieren. Deshalb fordern wir, daß sie zusammenhängend und beschränkt sind. Die Bewegungen der einzelnen Punkte sollen stetig verlaufen und zwar so, daß ihre relative Lage zueinander unverändert bleibt. Das heißt, daß wir nur *starre Körper* betrachten wollen und keinerlei Deformationen zulassen. Unter der zusätzlichen Voraussetzung, daß zu Beginn des Bewegungsablaufs die beteiligten Punktmengen disjunkt sind, gilt folgendes Lemma:

Lemma 1.1 *Für zwei disjunkte starre Körper X_1 und X_2 gilt:*

$$\begin{aligned} t_{col}(cl(X_1), X_2) &= t_{col}(\partial X_1, X_2) \\ \Delta_{min}(cl(X_1), X_2) &= \Delta_{min}(\partial X_1, X_2) \end{aligned}$$

Hierbei bezeichnet ∂X_1 den Rand der Punktmenge X_1 und $cl(X_1)$ ihren Abschluß. Die beiden Gleichungen besagen, daß wir uns bei der Berechnung von t_{col} und Δ_{min} auf den Rand der Punktmengen zurückziehen dürfen, wenn diese abgeschlossen sind.
Im Fall der starren Körper können wir die Trajektorien aller Punkte eines Körpers durch Angabe einer zeitabhängigen Orientierungsmatrix $\mathbf{R}(t)$ und eines Verschiebungsvektors $\mathbf{o}(t)$ charakterisieren.

$$\mathbf{x}(t) = \mathbf{R}(t)\,\mathbf{x} + \mathbf{o}(t) \tag{1.1}$$

Wenn wir dem Körper ein körperfestes Koordinatensystem zuordnen, dann beschreibt die Orientierungsmatrix $\mathbf{R} = \mathbf{R}(t)$ gerade die Lage der Einheitsvektoren dieses körperfesten Systems gegenüber dem Weltkoordinatensystem und $\mathbf{o} = \mathbf{o}(t)$ die Verschiebung seines Ursprungs. Folglich ist die Matrix $\mathbf{R}$ orthonormal. Um Spiegelungen zu verhindern, verlangen wir zusätzlich, daß stets $\det(\mathbf{R}) = 1$ gilt.
Der Abstand zwischen je zwei Punkten $\mathbf{x_1}(t)$ und $\mathbf{x_2}(t)$ eines bewegten starren Körpers bleibt zeitlich invariant, denn es gilt:

$$\begin{aligned} & (\mathbf{x_1}(t) - \mathbf{x_2}(t))^2 = (\mathbf{R}\mathbf{x_1} + \mathbf{o} - \mathbf{R}\mathbf{x_2} - \mathbf{o})^2 \\ = \; & (\mathbf{x_1} - \mathbf{x_2})^T \mathbf{R}^T \mathbf{R} (\mathbf{x_1} - \mathbf{x_2}) = (\mathbf{x_1} - \mathbf{x_2})^2, \quad \text{da} \quad \mathbf{R}^T\mathbf{R} = \mathbf{I} \end{aligned}$$

Aus (1.1) erkennen wir, daß starre Körper drei translatorische Freiheitsgrade besitzen, die durch den Vektor $\mathbf{o}$ repräsentiert werden, und drei rotatorische Freiheitsgrade, welche in der Matrix $\mathbf{R}$ verborgen sind. (Die d^2 Einträge einer d-dimensionalen orthonormalen Matrix sind wegen der Orthogonalitäts– und Normierungsbedingungen nicht unabhängig voneinander, sie läßt sich schon durch $\binom{d}{2}$ Parameter eindeutig festlegen.) Die populärste Alternative zur Festlegung der Orientierung besteht in der Angabe von drei Winkeln, die aufeinanderfolgende Rotationen um Achsen des Körper– oder Weltkoordinatensystems spezifizieren.

1.2 Modellierung der starren Körper

Die starren Körper beschreiben wir durch Polyeder. Diese spielen bei der Modellierung deshalb eine besondere Rolle, weil sie ebene Begrenzungsflächen besitzen und dadurch leicht repräsentierbar und gut zu handhaben sind. Zudem lassen sich real vorkommende Körper durch Polyeder beliebig genau approximieren.
Wir beginnen mit der Definition *konvexer* Polyeder.
Ausgehend von einer Menge $V = \{\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_{k-1}}\} \subset \mathbb{R}^3$ können wir ein konvexes Polyeder P als die kleinste konvexe Menge, die alle Punkte aus V enthält, definieren. P heißt die *konvexe Hülle* der Menge V.

$$P \;:=\; \{\mathbf{p} = \sum_{i=0}^{k-1} \lambda_i \mathbf{v_i} \,|\, \lambda_i \geq 0 \wedge \sum_{i=0}^{k-1} \lambda_i = 1\}$$

Dual hierzu können wir auch mit einer Menge von Halbräumen $\mathcal{H} = \{H_0, H_1, \ldots, H_{k-1}\}$ beginnen, wobei H_i in der Form $H_i = \{\mathbf{x} \,|\, h_{i1}x_1 + h_{i2}x_2 + h_{i3}x_3 \leq h_{i0}\}$ gegeben ist. P definieren wir als den Schnitt all dieser Halbräume.

$$P \;:=\; \bigcap_{i=0}^{k-1} H_i$$

Diese beiden impliziten Formen der Beschreibung konvexer Polyeder liefern uns aber noch keine explizite Darstellung. Eine solche erhalten wir mittels der sogenannten *Boundary Representation*, welche alle topologischen und geometrischen Informationen über den Rand ∂P eines Polyeders P enthält. Dieser Rand besteht aus konvexen Polygonen, die an ihren Randkanten miteinander verklebt sind. Die Randkanten selbst besitzen als Rand zwei

Eckpunkte, in denen mehrere Begrenzungsflächen zusammenstoßen. Konvexe Polyeder sind homöomorph zu einer Kugel. Deshalb läßt sich die topologische Struktur ihrer Oberfläche durch einen planaren Graphen repräsentieren. Die Geometrie legt man durch Angabe der kartesischen Koordinaten der Eckpunkte fest. Ergänzend kann man zu jeder Fläche noch die zugehörige Ebenengleichung angeben. Durch die Vereinigung mehrerer konvexer Polyeder können wir auch nichtkonvexe Körper modellieren. Diese besitzen unter Umständen Löcher oder Höhlungen. Die Oberflächenstruktur läßt sich in analoger Weise beschreiben, indem man sich die Inzidenzen zwischen Eckpunkten, Kanten und Flächen merkt.
Mit V bezeichnen wir in Zukunft die Menge der Eckpunkte, mit E die Menge der Kanten und mit F die Menge der Flächen des Polyeders P. Als Beschreibungskomplexität $|P|$ des Polyeders definieren wir

$$|P| \;:=\; |V| + |E| + |F|.$$

1.3 Modellierung der Bewegungen

In (1.1) sind die Position $\mathbf{o}(t)$ und die Orientierung $\mathbf{R}(t)$ beliebige Funktionen über der Zeit. Wir beschränken uns im folgenden aber darauf, daß die Körper nur *Translationen* oder *Rotationen* um eine feste Achse durchführen. Komplexere Bewegungen lassen sich stets durch eine Folge dieser elementaren Bewegungen beliebig genau approximieren. Im Fall der Translation in Richtung $\mathbf{s}$ und im Fall der Rotation um eine Achse in Richtung $\mathbf{r}$ durch den Punkt $\mathbf{o}$ ergibt sich:

$$\begin{aligned} \text{Translation:} &\quad \mathbf{x}(t) = \mathbf{x} + t\mathbf{s} \\ \text{Rotation:} &\quad \mathbf{x}(t) = \mathbf{R}(t)(\mathbf{x} - \mathbf{o}) + \mathbf{o} \end{aligned}$$

Dabei bezeichnet $\mathbf{R}(t)$ die Matrix, die eine Rotation um eine in Richtung $\mathbf{r}$ verlaufende Achse beschreibt (siehe Gleichung (A.8)).

1.4 Kollisionen und Abstände zwischen Polyedern

Wir betrachten eine einfache Situation, an der nur zwei Körper P_1 und P_2 beteiligt sind. Unser Ziel besteht in der Berechnung des frühsten Kollisionszeitpunktes $t_{col}(P_1, P_2)$ und des Sicherheitsabstandes $\Delta_{min}(P_1, P_2)$, der während der Bewegung eingehalten wird. Sowohl P_1 als auch P_2 modellieren wir durch *kompakte* Polyeder. F_1 und F_2 beschreiben die Begrenzungsflächen. Aufgrund von Lemma (1.1) gilt:

$$\begin{aligned} t_{col}(P_1, P_2) &= t_{col}(\partial P_1, \partial P_2) \;=\; t_{col}(F_1, F_2) \\ &= \min\{t_{col}(f_1, f_2) \mid f_1 \in F_1, f_2 \in F_2\} \\ \Delta_{min}(P_1, P_2) &= \Delta_{min}(\partial P_1, \partial P_2) \;=\; \Delta_{min}(F_1, F_2) \\ &= \min\{\Delta_{min}(f_1, f_2) \mid f_1 \in F_1, f_2 \in F_2\} \end{aligned}$$

Im Fall der Polyeder hat sich unser Problem darauf reduziert, t_{col} und Δ_{min} für zwei polygonale Begrenzungsflächen zu berechnen. Mit V_f bzw. E_f bezeichnen wir die Ecken bzw.

Kanten einer Fläche f. Für zwei anfangs disjunkte ebene Polygone $f_1, f_2 \subset \mathbb{R}^3$ gilt stets:

$$\begin{aligned} t_{col}(f_1, f_2) &= \min\{t_{col}(V_{f_1}, f_2), t_{col}(f_1, V_{f_2}), t_{col}(E_{f_1}, E_{f_2})\} \\ \Delta_{min}(f_1, f_2) &= \min\{\Delta_{min}(V_{f_1}, f_2), \Delta_{min}(f_1, V_{f_2}), \Delta_{min}(E_{f_1}, E_{f_2})\} \end{aligned}$$

Wir übertragen dieses Ergebnis auf die Polyeder selbst.

Lemma 1.2 *Für zwei kompakte Polyeder P_1 und P_2 mit $P_1 \cap P_2 = \emptyset$ gilt:*

$$\begin{aligned} t_{col}(P_1, P_2) &= \min\{t_{col}(V_1, F_2), t_{col}(F_1, V_2), t_{col}(E_1, E_2)\} \\ \Delta_{min}(P_1, P_2) &= \min\{\Delta_{min}(V_1, F_2), \Delta_{min}(F_1, V_2), \Delta_{min}(E_1, E_2)\} \end{aligned}$$

Die Bestimmung von t_{col} erfordert also nur das Wissen über die Berechnung, wann ein Punkt auf ein Polygon trifft und wann ein Liniensegment mit einem anderen Liniensegment kollidiert. Analog verhält es sich bei der Bestimmung des Sicherheitsabstandes Δ_{min}.

1.5 Einfache Algorithmen zur Bestimmung von Kollisionszeitpunkten und Sicherheitsabständen bei Translationen und Rotationen

Zur Berechnung von t_{col} bzw. Δ_{min} benötigt man Null– bzw. Extremstellen von entsprechenden Funktionen des Zeitparameters t. Damit diese Funktionen handhabbar bleiben, beschränken wir uns zum einen auf polyederförmige Körper und lassen zum anderen nur Translationen oder Rotationen um beliebige Achsen als Bewegungsformen der Körper zu. Selbst unter diesen Einschränkungen treten bei gleichzeitiger Bewegung von Körpern noch Funktionen auf, deren Nullstellen nur mittels numerischer Algorithmen approximativ bestimmt werden können. Deshalb fordern wir weiterhin, daß die Relativbewegung zwischen je zwei Polyedern als Translation oder als Rotation um eine Achse beschreibbar ist. Wir werden zeigen, daß in diesen Fällen stets geschlossene Lösungen für t_{col} und Δ_{min} existieren.
Unter den obigen Einschränkungen für die Bewegung der Polyeder dürfen wir bei der Betrachtung eines Polyederpaares (P_1, P_2) immer ein Polyeder als bewegt und das andere als ruhend ansehen. Im Fall der Rotation legen wir der Einfachheit halber die Rotationsachse immer in den Ursprung des Weltkoordinatensystems.
Die folgenden Notationen beziehen sich auf zwei beliebige Punktmengen $X_1 \subset P_1$ und $X_2 \subset P_2$, von denen X_1 eine Translation oder eine Rotation ausführen soll, während X_2 ihre Lage beibehält.

$$\begin{aligned} X_2(t) &:= X_2 \quad \forall\, t \\ X_1^{trans}(t) &:= \{\mathbf{x} + t\mathbf{s} \,|\, \mathbf{x} \in X_1\} \\ X_1^{rot}(t) &:= \{\mathbf{R}(t)\mathbf{x} \,|\, \mathbf{x} \in X_1\} \\ t^*_{col}(X_1, X_2) &:= \begin{cases} \infty \quad \text{falls} \quad X_1^*(t) \cap X_2 = \emptyset & \forall\, t \geq 0 \\ \inf\{t \geq 0 \,|\, X_1^*(t) \cap X_2 \neq \emptyset\} & \text{sonst} \end{cases} \\ \Delta^*_{min}(X_1, X_2) &:= \inf\{\delta(X_1^*(t), X_2) \,|\, t \in [0, T]\} \\ & \quad \text{wobei } * = trans \text{ oder } * = rot \end{aligned}$$

Wenn eine einzelne Ecke, Kante oder Fläche des Polyeders P_i die Rolle von X_i übernimmt, lassen sich $t^*_{col}(X_1, X_2)$ und $\Delta^*_{min}(X_1, X_2)$ in konstanter Zeit berechnen, wie wir später zeigen werden. Eine solche Berechnung sehen wir als eine elementare Operation an.
Nach Lemma 1.1 können wir $t^*_{col}(P_1, P_2)$ und $\Delta^*_{min}(P_1, P_2)$ somit in Zeit $O(|P_1| \cdot |P_2|)$ berechnen. Dabei ist

$$\begin{aligned} t^*_{col}(V_1, F_2) &= \min\{t^*_{col}(v_1, f_2) \mid v_1 \in V_1, f_2 \in F_2\}, \\ t^*_{col}(F_1, V_2) &= \min\{t^*_{col}(f_1, v_2) \mid f_1 \in F_1, v_2 \in V_2\}, \\ t^*_{col}(E_1, E_2) &= \min\{t^*_{col}(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2\}. \end{aligned}$$

Analoges gilt für Δ_{min}. In den folgenden Abschnitten wollen wir demonstrieren, daß sich bei der Berechnung von t^*_{col} die Zahl der elementaren Operationen mittels einfacher Techniken stark vermindern läßt. Die asymptotische Laufzeit für den Worst–Case erfährt dadurch allerdings keine Verbesserung.

1.6 Heuristiken zur Berechnung von Kollisionszeitpunkten

1.6.1 Elimination von Rückseiten

Bewegen wir ein Polyeder P_1 translatorisch in Richtung $\mathbf{s}$, so kann ein Punkt $\mathbf{p} \in P_1$ natürlich nur mit einer Fläche f eines Hindernispolyeders P_2 zusammenstoßen, deren Flächennormale $\mathbf{n}_f$ mit der Translationsrichtung $\mathbf{s}$ einen Winkel größer als 90° einschließt ($\mathbf{s}^T\mathbf{n}_f \leq 0$). Dabei soll die Konvention gelten, daß $\mathbf{n}_f$ ins Äußere von P_2 zeigt. Alle anderen Flächen von P_2 sind Rückseiten und müssen nicht betrachtet werden.

Ähnliche Überlegungen können wir auch bei einer Rotation durchführen. Dazu stellen wir uns vor, daß das Polyeder P_1 im Gegenuhrzeigersinn um eine durch den Ursprung verlaufende Achse $\mathbf{r}$ rotiert. Wenn ein Punkt $\mathbf{p} \in P_1$ mit einer Fläche f des stationären Polyeders P_2 zusammentrifft, muß zum Zeitpunkt der Kollision die momentane Bewegungsrichtung des Punktes $\mathbf{p}$ mit der Flächennormale $\mathbf{n}_f$ einen Winkel größer als 90° einschließen. Trifft $\mathbf{p}$ die Fläche f im Punkt $\mathbf{x}$, so hat die Tangente an die Kreisbahn die Richtung $\mathbf{r} \times \mathbf{x}$. Wenn es aber keinen Punkt $\mathbf{x} \in f$ mit der Eigenschaft $(\mathbf{r} \times \mathbf{x})^T\mathbf{n}_f \leq 0$ gibt, so kann f als Rückseite bezeichnet werden. Die Bedingung dafür, daß die Fläche f in Kontakt mit dem rotierenden Polyeder P_1 kommen kann, läßt sich also als die Frage formulieren, ob f den Halbraum Π^-_f schneidet, wobei $\Pi^\pm_f = \{\mathbf{x} | \pm(\mathbf{n}_f \times \mathbf{r})^T\mathbf{x} \geq 0\}$.

In der gleichen Weise können wir die Rückseiten des Polyeders P_1 bezüglich der ausgeführten Bewegung definieren. Bei der Berechnung des Kollisionszeitpunktes t^*_{col} dürfen wir dann alle diese Flächen und alle Kanten und Ecken, die nur zu rückseitigen Flächen gehören, von der Kollisionsbetrachtung ausschließen. Also ist

$$\begin{aligned} t^*_{col} &= \min\{t^*_{col}(V_1^+, F_2^-), t^*_{col}(F_1^+, V_2^-), t^*_{col}(E_1^+, E_2^-)\} \text{ mit} \\ F^\pm &= \{f \in F \mid \pm\mathbf{s}^T\mathbf{n}_f \geq 0\} \quad \text{(Translation)}, \\ F^\pm &= \{f \in F \mid f \cap \Pi^\pm_f \neq \emptyset\} \quad \text{(Rotation)}, \end{aligned}$$

$$\begin{aligned} E^{\pm} &= \{e \in E \,|\, \exists f \in F^{\pm} : e \subset f\}, \\ V^{\pm} &= \{v \in V \,|\, \exists e \in E^{\pm} : v \in e\}. \end{aligned}$$

Nach der Beseitigung der Rückseiten bietet sich zur weiteren Reduktion der elementaren Operationen, die zur Berechnung von t^*_{col} benötigt werden, folgende Strategie an:

1.6.2 Projektionsmethoden

Bei einer Translation in Richtung $\mathbf{s}$ projiziere man die Flächen der beiden Polyeder auf eine Ebene senkrecht zu $\mathbf{s}$, die von den beiden orthonormalen Vektoren $\mathbf{s_1}$ und $\mathbf{s_2}$ aufgespannt werde. Offensichtlich können zwei Flächen f_1 und f_2 nur dann kollidieren, wenn sich ihre Projektionen $\overline{f}_1$ und $\overline{f}_2$ schneiden. Denn bei einer Translation eines Punktes $\mathbf{x}$ in Richtung $\mathbf{s}$ bleiben die Komponenten von $\mathbf{x}$ in Richtung $\mathbf{s_1}$ und $\mathbf{s_2}$ unverändert.

Im Falle einer Rotation kann man eine vergleichbare Methode benutzen, wenn man mit Zylinderkoordinaten (d, φ, h) arbeitet. Die Projektion besteht dann im Weglassen der Winkelkomponente φ. Bei einer Rotation eines Punktes $\mathbf{x}$ um die $\mathbf{r}$–Achse bleiben der Radius d und die Höhe h unverändert. Es gilt

$$\begin{aligned} d(\mathbf{x}) &= |\mathbf{r} \times \mathbf{x}|, \\ \varphi(\mathbf{x}) &= arctan(\mathbf{r_2}^T\mathbf{x}, \mathbf{r_1}^T\mathbf{x}), \\ h(\mathbf{x}) &= \mathbf{r}^T\mathbf{x}. \end{aligned}$$

($\mathbf{r_1}$ und $\mathbf{r_2}$ ergänzen $\mathbf{r}$ zu einem Orthonormalsystem.)
Zur geometrischen Veranschaulichung wähle man eine Ebene, die die Rotationsachse $\mathbf{r}$ enthält und lasse die Flächen der beiden Polyeder eine 360°–Drehung durchführen. Dabei schneidet eine Fläche f beim Durchdringen dieser Ebene ein Gebiet $\overline{f}$ aus, das durch Hyperbeln begrenzt wird (Herleitung in Abschnitt 2.2.2.1). Analog zu oben gilt, daß ein Flächenpaar (f_1, f_2) bei einer Rotation nur dann zur Kollision gebracht werden kann, wenn $\overline{f}_1 \cap \overline{f}_2 \neq \emptyset$ gilt.

$$\begin{aligned} t^*_{col}(F_1, F_2) &= \min\{t^*_{col}(f_1, f_2) | f_1 \in F_1, f_2 \in F_2, \overline{f}_1 \cap \overline{f}_2 \neq \emptyset\}, \quad \text{wobei} \\ \overline{f} &= \{(\mathbf{s_1}^T\mathbf{x}, \mathbf{s_2}^T\mathbf{x}) | \mathbf{x} \in f\} \quad \text{(Translation) bzw.} \\ \overline{f} &= \{(|\mathbf{r} \times \mathbf{x}|, \mathbf{r}^T\mathbf{x}) | \mathbf{x} \in f\} \quad \text{(Rotation)} \end{aligned}$$

Das Beispiel in Abbildung 1.1 zeigt eine Szene aus einem Würfel und einem Dodekaeder. Die Rotationsachse entspricht der x_3–Achse. Daneben ist das Bild zu sehen, das die beiden Polyeder bei einer 360°–Drehung aus einer Ebene ausschneiden, die die Rotationsachse enthält. Die Rückseiten sind noch nicht eliminiert. Am Beispiel des Würfels erkennt man, daß sich eine Fläche f bei der Projektion in die dh–Ebene des Zylinderkoordinatensystems falten kann. Um solche Faltungen zu verhindern, zerteile man eine solche Fläche f einfach durch die Ebene $\Pi^{\pm}_f$ in zwei Teile. Diese werden für sich faltungsfrei abgebildet.
Nach der Projektion der Flächen der beiden Polyeder auf eine geeignete Projektionsebene stellt sich die Frage, wie man in effizienter Weise herausfinden kann, für welche Paare (f_1, f_2) ein Schnitt der Projektionen vorliegt. Mit dieser Frage beschäftigen wir uns als nächstes.

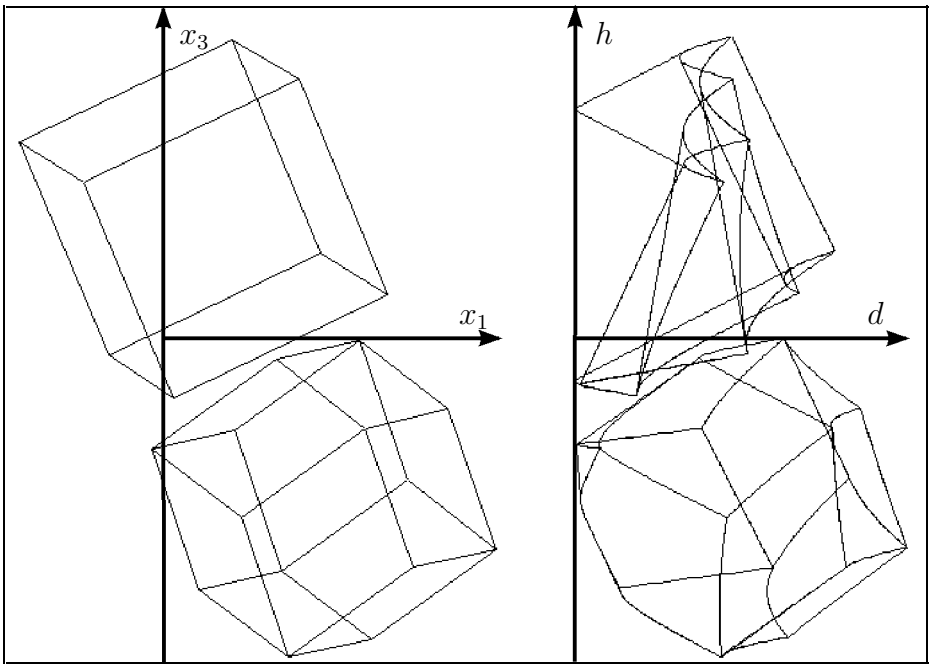


Abbildung 1.1: Projektion einer Szene in die x_1x_3–Ebene und ihre Darstellung im dh–Zylinderkoordinatensystem

1.6.3 2D–Faceboxing

Gegeben seien zwei Mengen von Gebieten im $\mathbb{R}^2$, die wir $\overline{F}_1$ und $\overline{F}_2$ nennen wollen. Unser Ziel ist die Bestimmung von

$$C_{\overline{F}_1,\overline{F}_2} \quad := \quad \{(\overline{f}_1, \overline{f}_2) \in \overline{F}_1 \times \overline{F}_2 \,|\, \overline{f}_1 \cap \overline{f}_2 \neq \emptyset\}.$$

Da $|C_{\overline{F}_1,\overline{F}_2}| \ll |\overline{F}_1| \cdot |\overline{F}_2|$ sein kann, sollte man nicht einfach alle Paare aus $\overline{F}_1 \times \overline{F}_2$ durchmustern, sondern einen Algorithmus benutzen, dessen Laufzeit von der Größe der Ausgabe bestimmt wird.

Eine gute Heuristik zur Lösung der Aufgabe besteht darin, alle Gebiete in achsenparallele Rechtecke einzuschließen und das Problem für die beiden entstehenden Rechteckmengen zu lösen. $b(\overline{f}_i)$ bezeichne das Rechteck, das $\overline{f}_i$ einschließt, dann gilt:

$$C_{\overline{F}_1,\overline{F}_2} \quad \subseteq \quad \{(\overline{f}_1, \overline{f}_2) \,|\, b(\overline{f}_1) \cap b(\overline{f}_2) \neq \emptyset\}$$

Schnitte zwischen orthogonalen Rechtecken zu finden, ist ein wohlbekanntes Problem.

> Gegeben seien zwei Mengen B_1 und B_2 von achsenparallelen Rechtecken. Die Menge $\{(b_1, b_2) \in B_1 \times B_2 \,|\, b_1 \cap b_2 \neq \emptyset\}$ kann in Zeit $O((|B_1| + |B_2|)\log(|B_1| + |B_2|) + |C_{B_1,B_2}|)$ berechnet werden.

In der Literatur findet man für dieses Problem oft den Namen "Rot–Blau Schnitt" von Rechtecken. Ein in [Ho89] (Abschnitt 3.7.) beschriebener Algorithmus führt das 2–dimensionale statische Problem auf ein dynamisches 1–dimensionales zurück, indem eine Sweep–Line über die Rechteckmengen geführt wird. Diese hält immer dann an, wenn ein Rechteck beginnt oder endet. Der Schnitt der Sweep–Line mit den Rechtecken ergibt jeweils eine Menge von roten und blauen Intervallen, die in Intervallbäumen verwaltet werden. Hält die Sweep–Line z. B. am Anfang eines blauen Rechtecks, so hat man in dem roten Intervallbaum all die aktiven Intervalle zu suchen, die das Intervall des blauen Rechtecks schneiden. Nach logarithmischer Suchzeit kann eine solche Anfrage in einer Zeit durchgeführt werden, die proportional zur Größe der gefundenen Schnitte ist. Die Intervallbäume selbst müssen nicht voll dynamisch sein, weil man im voraus bereits alle möglichen Intervallgrenzen kennt. Dies ermöglicht schließlich das Einfügen und Löschen von Intervallen in logarithmischer Zeit und führt somit auf die angegebene Gesamtlaufzeit.

Eine detaillierte Beschreibung dieses Algorithmus und Laufzeitmessungen im Vergleich zu dem naiven quadratischen Algorithmus findet man in [Fr93].

1.6.4 Sweep–Algorithmen

Ein anderer Ansatz zur Reduktion der elementaren Operationen zur Kollisionsberechnung benutzt die Beziehung

$$t^*_{col}(P_1, P_2) \quad = \quad \min\{t^*_{col}(V_1, F_2), t^*_{col}(F_1, V_2), t^*_{col}(E_1, E_2)\}.$$

Betrachten wir wieder die jeweiligen Projektionen der beiden Polyeder, so können wir die Berechnung von $t^*_{col}(E_1, E_2)$ auf die Paare (e_1, e_2) mit $\overline{e}_1 \cap \overline{e}_2 \neq \emptyset$ beschränken.
Mit Hilfe eines einfachen Sweeps über die projizierte Szene lassen sich leicht alle Kantenpaare ermitteln, deren Projektionen sich schneiden. (Der Sweep der Linien– bzw. Hyperbelsegmente kann in der gleichen Weise durchgeführt werden. Haltepunkte des Sweeps wie Extrempunkte der Kurven und Schnittpunkte zwischen zwei Kurven können jeweils in konstanter Zeit ermittelt werden.) Als Laufzeit erhalten wir $O((|E_1| + |E_2| + k)\log(|E_1| + |E_2|))$, wobei in k alle Schnitte zwischen Kurvensegmenten einschließlich der rot–rot und blau–blau Schnitte eingehen: $k = |C_{\overline{E}_1,\overline{E}_1}| + |C_{\overline{E}_2,\overline{E}_2}| + |C_{\overline{E}_1,\overline{E}_2}|$. Diese Vorgehensweise verfehlt allerdings ihr Ziel, wenn k in der Größenordnung von $|P_1| \cdot |P_2|$ liegt. Eine gewisse Verbesserung kann man erreichen, indem man Algorithmen benutzt, die unempfindlich gegenüber Schnitten gleichfarbiger Kurvensegmente sind (siehe Abschnitt 1.6.5). Die Laufzeit im Worst–Case bleibt dadurch natürlich quadratisch. In Kapitel 3.3 werden wir einen subquadratischen Algorithmus zu Berechnung von $t^{trans}_{col}(E_1, E_2)$ vorstellen.
Im Gegensatz zu den Kollisionen zwischen den Kanten der Polyeder, lassen sich Kollisionen zwischen Ecken und Flächen sehr viel leichter in subquadratischer Zeit bestimmen:

Lemma 1.3 *$t^*_{col}(V_1, F_2)$ und $t^*_{col}(F_1, V_2)$ ($* = trans$ oder $* = rot$) können in Zeit $O((|P_1| + |P_2|)^{\frac{3}{2}}\log(|P_1| + |P_2|))$ berechnet werden.*

Beweis: Zur effizienten Berechnung des Kollisionszeitpunktes zwischen einer translatorisch bzw. rotatorisch bewegten Punktmenge V_1 und einer stationären Menge von Flächen F_2 projizieren wir Punkte und Flächen zunächst in der oben beschriebenen Weise. Dadurch erhalten

wir in der Projektionsebene neben den Punkten $\overline{V}_1$ 2–dimensionale Gebiete $\overline{F}_2$, die durch Linien– bzw. Hyperbelsegmente begrenzt werden (siehe Abbildung 1.1). Analog zu der Bestimmung der Schnittpunkte zwischen den Kurvensegmenten führen wir einen Sweep durch. Zwischen zwei aufeinanderfolgenden Haltepunkten bleibt die Reihenfolge der Schnittpunkte der Kurvensegmente mit einer Sweep–Line S gleich. (Als Haltepunkte gelten Anfangs-, End-, Extrem- und Schnittpunkte von Kurvensegmenten.) Dieses Ordnungskriterium ermöglicht es uns, die aktuellen Kurvensegmente in einem balancierten Suchbaum zu speichern. Dieser dient uns aber nur als Primärstruktur zur Speicherung weiterer Daten:
Sei R eine Region, die zwischen zwei auf S benachbarten Kurvensegmenten liegt und von S zwischen zwei aufeinanderfolgenden Haltepunkten überstrichen wird. Ein Punkt $v_1 \in V_1$, dessen Projektion $\overline{v}_1$ in R liegt, kann bei seiner Bewegung nur mit solchen Flächen $f_2 \in F_2$ zusammenstoßen, für die $R \subseteq \overline{f}_2$ gilt. Deshalb merken wir uns zu jeder Region R die Menge $F_R = \{f_2 \in F_2 | R \subseteq \overline{f}_2\}$. Zu diesem Zweck ergänzen wir die Primärstrukur so, daß zu jedem Kurvensegment die Menge F_R der darüberliegenden Region R gespeichert wird.
In Abhängigkeit vom Bewegungsmodus läßt sich auf jeder Menge F_R eine Ordnung definieren. Jeder Punkt v_1, dessen Projektion $\overline{v}_1$ in R liegt, wird bei einer Volldrehung die Flächen aus F_R in der gleichen zyklischen Reihenfolge durchstoßen. Bei einer Translation lassen sich die Flächen aus F_R linear anordnen. Als Sekundärstruktur zur Speicherung der Mengen F_R benutzen wir ebenfalls balancierte Suchbäume, denn diese ermöglichen es uns, in logarithmischer Zeit die Fläche zu finden, die ein Punkt v_1 ($\overline{v}_1 \in R$) bei seiner Bewegung zuerst trifft.
Die Sweep–Line S lassen wir an allen Punkten aus $\overline{V}_1$ anhalten. Für einen Haltepunkt $\overline{v}_1$ bestimmen wir zunächst die Region R, die $\overline{v}_1$ enthält, und anschließend die Fläche $f_2 \in F_R \subseteq F_2$, mit der v_1 zuerst kollidiert. Beide Stufen entsprechen einem Suchprozess in einem balancierten Baum und können in logarithmischer Zeit durchgeführt werden.
Während des Sweeps müssen wir alle Regionen R, die die Sweep–Line schneiden, inklusive der Mengen F_R, dynamisch verwalten. Der balancierte Suchbaum der Primärstruktur erlaubt das Einfügen und Löschen von Regionbegrenzungen in logarithmischer Zeit. Diese Operationen werden immer dann benötigt, wenn die Sweep–Line S an Extrempunkten oder Schnittpunkten von Kurvensegmenten anhält. In der Sekundärstruktur können wir es uns allerdings nicht erlauben, für jede auftretende Region R die Flächenmenge F_R explizit abzuspeichern. Hier hilft uns die folgende Beobachtung weiter:
An einem Haltepunkt entstehen nur konstant viele neue Regionen. (Mehrfach vorkommende Haltepunkte bei gleicher Position der Sweep–Line werden separat behandelt.) Alle anderen werden weiterhin von den gleichen Kurvensegmenten begrenzt und die zugehörigen Flächenmengen verändern sich nicht. Wenn eine neue Region R entsteht, unterscheidet sich die Menge F_R nur um eine einzige Fläche von der Flächenmenge $F_{R'}$ einer angrenzenden Regionen R'. Deshalb genügt es, nur die Veränderung von F_R gegenüber $F_{R'}$ zu registrieren. Da $F_{R'}$ als balancierter Baum vorliegt und F_R durch Einfügen oder Löschen einer Fläche aus $F_{R'}$ hervorgeht, braucht man zur Speicherung von F_R nur den logarithmisch langen Update–Pfad mit Verweisen auf unveränderte Teilbäume. Dies beweist, daß an jedem Haltepunkt nur logarithmische Kosten verursacht werden.

Die Laufzeit des Sweep–Algorithmus beläuft sich auf $O((|V_1| + |E_2| + |C_{\overline{E}_2,\overline{E}_2}|) \log |E_2|)$. Der kritische Wert dabei ist die möglicherweise quadratische Zahl der Schnitte zwischen den

Kurvensegmenten. Deshalb ist es besser, wenn wir unser Problem in kleinere Teilprobleme aufspalten. Ohne Beschränkung der Allgemeinheit nehmen wir an, daß die Menge F_2 nur aus Dreiecken besteht. (Eine Triangulierung der Oberfläche von P_2 verändert die asymptotische Komplexität von $|P_2|$ nicht.) F_2 zerlegen wir in $\sqrt{|F_2|}$ viele Teilmengen F_2^i der Größe $\sqrt{|F_2|}$.

$$\begin{aligned} F_2 &= \bigcup_i F_2^i \\ t^*_{col}(V_1, F_2) &= \min_i t^*_{col}(V_1, F_2^i) \end{aligned}$$

Für jede Menge F_2^i führen wir den oben beschriebenen Sweep separat durch. Mit E_2^i bezeichnen wir die Begrenzungskanten von F_2^i. Da $|C_{\overline{E}_2^i, \overline{E}_2^i}| = O(|E_2|)$ ist, erhalten wir auf diese Weise eine Laufzeit von $O(\sqrt{|E_2|}(|V_1| + |E_2|) \log |E_2|)$ zur Berechnung von $t^*_{col}(V_1, F_2)$ und einen entsprechenden Term für $t^*_{col}(F_1, V_2)$. Aus $|V_1|, |E_1| \leq |P_1|$ und $|V_2|, |E_2| \leq |P_2|$ folgt die Aussage des Lemmas. ∎

Bemerkungen:
Die Idee zur effizienten Verwaltung der Mengen F_R findet man bereits bei [Nu85], der einen Algorithmus zur Berechnung der sichtbaren Flächen einer Szene aus Polyedern beschreibt. Der logarithmische Speicheraufwand für jede einzelne Menge F_R kann sogar auf eine Konstante reduziert werden, wenn man *persistente* balancierte Suchbäume benutzt (siehe [DSST89]).

1.6.5 Rot–Blau Schnitte

Wenn wir die projizierten Flächen der Polyeder in achsenparallele Rechtecke einschließen, ist es möglich, den Schnitt zwischen roten und blauen Rechtecken effizient zu berechnen, ohne Schnitte zwischen gleichfarbigen Rechtecken berücksichtigen zu müssen. Analoge Resultate gibt es auch für den Rot–Blau Schnitt von Liniensegmenten.
[Ag90] erzielt z. B. folgendes, für uns interessante Resultat:

> Rot–Blau Schnitte zwischen einer Menge von n roten und einer Menge von n blauen Liniensegmenten im $\mathbb{R}^2$ lassen sich in Zeit $O(n^{\frac{4}{3}} \log^2 n + k)$ bestimmen, wobei k der Größe der Ausgabe entspricht.

Damit können wir $t^{trans}_{col}(E_1, E_2)$ in Zeit $O((|E_1|+|E_2|)^{\frac{4}{3}} \log^2(|E_1|+|E_2|)+|C_{\overline{E}_1, \overline{E}_2}|)$ berechnen. Der Aufwand lohnt sich jedoch nur, wenn $|C_{\overline{E}_1, \overline{E}_2}| \ll |E_1| \cdot |E_2|$ ist.

1.6.6 Tests zur Kollisionsfreiheit von Volldrehungen

In diesem Abschnitt wollen wir einen effizienten Algorithmus zur Entscheidung der folgenden Frage entwerfen:

> Kann ein Polyeder P_1 in Gegenwart eines zweiten stationären Hindernispolyeders P_2 vollständig um eine gegebene Achse $\mathbf{r}$ rotieren, ohne daß es zu einer Kollision kommt?

Zur Beantwortung der obigen Frage konstruieren wir das von P_1 überstrichene Volumen $P_1^{rot} = \{\mathbf{R}(t)\mathbf{p_1}|\mathbf{p_1} \in P_1, t \in [0, 2\pi)\}$ und bilden den Schnitt mit P_2. Offensichtlich gilt $P_1^{rot} \cap P_2 \neq \emptyset \Longleftrightarrow t_{col}^{rot}(P_1, P_2) < \infty$.
Zur Beschreibung des Rotationskörpers P_1^{rot}, der bei einer Volldrehung von P_1 entsteht, genügt es aus Symmetriegründen, die 2–dimensionale Querschnittsfläche $\overline{P_1}^{rot}$ anzugeben. Auf diese Weise erweist sich das gestellte Entscheidungsproblem als 2–dimensionale Aufgabe, wie uns folgendes Lemma beweist:

Lemma 1.4 $P_1^{rot} \cap P_2 \neq \emptyset \iff P_1^{rot} \cap P_2^{rot} \neq \emptyset \iff \overline{P_1}^{rot} \cap \overline{P_2}^{rot} \neq \emptyset$

Beweis: Es genügt $P_1^{rot} \cap P_2^{rot} \neq \emptyset \Longrightarrow P_1^{rot} \cap P_2 \neq \emptyset$ zu zeigen. Alle anderen Beziehungen sind direkt einzusehen.

$$\begin{aligned} & P_1^{rot} \cap P_2^{rot} \neq \emptyset \\ \Longrightarrow\ & \exists\, t, t' \in \mathbb{R}, \mathbf{p_1} \in P_1, \mathbf{p_2} \in P_2 :\ \mathbf{R}(t)\mathbf{p_1} = \mathbf{R}(t')\mathbf{p_2} \\ \Longrightarrow\ & \exists\, \mathbf{p_1} \in P_1, \mathbf{p_2} \in P_2 :\ \mathbf{R}^{-1}(t')\mathbf{R}(t)\mathbf{p_1} = \mathbf{R}(t-t')\mathbf{p_1} = \mathbf{p_2} \\ \Longrightarrow\ & P_1^{rot} \cap P_2 \neq \emptyset \end{aligned}$$

■

Da die Oberfläche der Rotationskörper P_1^{rot} und P_2^{rot} von Hyperboloiden, Kegeln und Zylindern begrenzt werden, bestehen die Konturen der Querschnittsflächen $\overline{P_1}^{rot}$ und $\overline{P_2}^{rot}$ aus Hyperbeln und Geraden. Jetzt wollen wir Verfahren zur schnellen Berechnung dieser Konturen angeben.
Wir betrachten zunächst konvexe Polyeder, weil sich in diesem Spezialfall die entstehenden Konturen leicht beschreiben lassen:
Wenn wir ein konvexes Polyeder P_i mit einer Ebene $H(h) = \{\mathbf{x}|\mathbf{r}^T\mathbf{x} = h\}$ senkrecht zur Rotationsachse schneiden, erhalten wir im allgemeinen ein konvexes Polygon. Von diesem bestimmen wir den minimalen und maximalen Abstand zur Rotationsachse:

$$\begin{aligned} d_i(h) &= \min\{|\mathbf{r} \times \mathbf{p}|\ |\mathbf{r}^T\mathbf{p} = h, \mathbf{p} \in P_i\} \\ D_i(h) &= \max\{|\mathbf{r} \times \mathbf{p}|\ |\mathbf{r}^T\mathbf{p} = h, \mathbf{p} \in P_i\} \end{aligned}$$

Die Querschnittsfläche $\overline{P_i}^{rot}$ können wir also durch eine Funktion $C_i(h)$ beschreiben, die jedem Wert h ein Intervall $[d_i(h), D_i(h)]$ zuordnet. Der Definitionsbereich der Funktion $C_i(h)$ ist das Intervall

$$I_i = [\min_{\mathbf{p} \in P_i} \mathbf{r}^T\mathbf{p}, \max_{\mathbf{p} \in P_i} \mathbf{r}^T\mathbf{p}].$$

Die Frage nach der Kollision des rotierenden Polyeders P_1 mit dem stationären Polyeder P_2 hat sich nun auf die Untersuchung der Querschnittskonturen $C_1(h)$ und $C_2(h)$ reduziert. Wegen Lemma 1.4 gilt:

$$P_1^{rot} \cap P_2 = \emptyset \Longleftrightarrow \forall\, h \in I_1 \cap I_2 : C_1(h) \cap C_2(h) = \emptyset$$

Es bleibt also die Aufgabe, für ein konvexes Polyeder P_1 die Funktion $C_1(h)$ zu berechnen (und später mit $C_2(h)$ zu vergleichen).

Der maximale Abstand $D_1(h)$ kann offensichtlich nur von den Kanten des Polyeders bestimmt werden. Für eine Kante $e \in E_1$, die zwischen den Punkten $\mathbf{a}$ und $\mathbf{b}$ verläuft, können wir den Abstand $d_e(h)$ zur Rotationsachse leicht ermitteln. Gemäß Abschnitt 2.2.2.1 erhalten wir im allgemeinen eine Hyperbel.

$$\begin{aligned} d_e(h) &= \sqrt{\alpha_e h^2 + \beta_e h + \gamma_e} \quad \text{für} \quad h \in I_e \\ &\quad \text{wobei} \quad I_e = [\min\{\mathbf{r}^T\mathbf{a}, \mathbf{r}^T\mathbf{b}\}, \max\{\mathbf{r}^T\mathbf{a}, \mathbf{r}^T\mathbf{b}\}] \\ D_i(h) &= \max_{e \in E_i}\{d_e(h) | h \in I_e\} \end{aligned}$$

Die Funktion $D_i(h)$ ist die *obere Kontur* einer Menge von $|E_i|$ Hyperbelstücken.
Dementsprechend bestimmen wir die Funktion $d_i(h)$ als die *untere Kontur* dieser Menge von Hyperbelstücken, ergänzt um eine Menge von Geradenstücken, die wie folgt zustande kommt: Für eine Fläche $f \in F_i$ wird der minimale Abstand $d_f(h)$ entweder auf dem Rand angenommen oder auf dem Geradensegment, das man erhält, wenn man von jedem Punkt der Rotationsachse das Lot auf die Fläche fällt. Wenn $\mathbf{n}_f$ den Normalenvektor der Fläche f bezeichnet, dann ist dieses Segment der Schnitt der Ebene $\Pi_f : (\mathbf{r} \times \mathbf{n}_f)^T\mathbf{x} = 0$ mit f. Wir definieren $E_i^\perp := \{e = \Pi_f \cap f | f \in F_i\}$ und erhalten:

$$d_i(h) = \min_{e \in E_i \cup E_i^\perp}\{d_e(h) | h \in I_e\}$$

Es bleibt das Problem, die untere und obere Kontur einer Menge von partiell definierten Kurven beschränkter Ordnung zu bestimmen. Wir können hier nur deshalb von unterer und oberer Kontur reden, weil die betrachteten Polyeder konvex sind. Zumindest in diesem Fall läßt sich die Berechnung der Kontur auch effizient bewerkstelligen. Hat man die beiden Konturen $C_1(h)$ und $C_2(h)$ für die beteiligten Polyeder berechnet, so muß man nur noch testen, ob gilt $\exists h \in I_1 \cap I_2 : C_1(h) \cap C_2(h) = \emptyset$. Dies kann mittels eines herkömmlichen Sweep–Algorithmus in Zeit $O(|C_1| + |C_2|)$ geschehen, wenn die beiden Konturen in Sweeprichtung sortiert vorliegen. ($|C_i|$ mißt dabei die Zahl der Kurvensegmente, die zur Beschreibung von C_i nötig sind.)
Die Berechnung der gesuchten Konturen könnte auch mit Hilfe eines Plane–Sweep–Algorithmus geschehen, aber dann wäre die Laufzeit abhängig von der Zahl der Überkreuzungen der Kurvenstücke, die im schlechtesten Fall quadratisch sein kann. Sharir [Sh88b] schlägt folgenden einfachen Divide–And–Conquer Algorithmus vor:
Zur Berechnung der oberen Kontur $D(h)$ einer Menge von n Kurvenstücken zerlege man die gegebene Menge in zwei gleich große Teile F und G. Für jede der beiden Teilmengen berechne man die zugehörigen Konturen $D_F(h)$ und $D_G(h)$ rekursiv. Die Kontur $D(h)$ ergibt sich – ähnlich zu Mergesort – durch Mischen von $D_F(h)$ und $D_G(h)$. $D_F(h)$ sei wie folgt durch l_F Kurvensegmente gegeben (analog $D_G(h)$):

$$D_F(h) = \begin{cases} f_1(h) \text{ für } h \in [a_1, a_1'] \\ f_2(h) \text{ für } h \in [a_2, a_2'] \\ \vdots \\ f_{l_F}(h) \text{ für } h \in [a_{l_F}, a_{l_F}'] \\ \infty \text{ sonst} \end{cases} \qquad D_G(h) = \begin{cases} g_1(h) \text{ für } h \in [b_1, b_1'] \\ g_2(h) \text{ für } h \in [b_2, b_2'] \\ \vdots \\ g_{l_G}(h) \text{ für } h \in [b_{l_G}, b_{l_G}'] \\ \infty \text{ sonst} \end{cases}$$

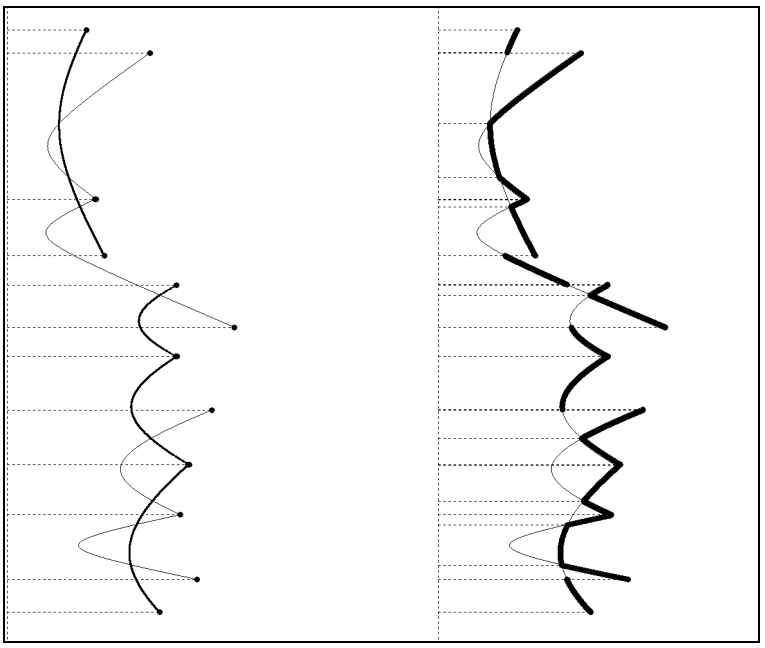


Abbildung 1.2: Illustration des Konturmischens

Dabei gelte $a_i < a'_i \leq a_{i+1}$ und $b_j < b'_j \leq b_{j+1}$. $c_1 < c_2 < \ldots < c_l$ sei die sortierte Folge (ohne Mehrfachvorkommen) der a_i, a'_i und b_j, b'_j für $i = 1, \ldots, l_F$ und $j = 1, \ldots, l_G$ d.h. $l \leq 2(l_F + l_G)$. Diese Folge stellt die gemeinsame Verfeinerung der durch $D_F(h)$ und $D_G(h)$ vorgegebenen Partitionierungen dar. Auf jedem Intervall $I_k = [c_k, c_{k+1}]$ für $k = 1, \ldots, l-1$ kann man den Verlauf der Kontur nun einfach berechnen, denn es können nur vier verschiedene Fälle eintreten:

(1) In I_k liegt kein Kurvenstück.

(2) In $I_k \subseteq [a_i, a'_i]$ liegt nur das Kurvenstück $f_i(h)$. Dann ist $D(h) = f_i(h)$ für $h \in I_k$.

(3) In $I_k \subseteq [b_j, b'_j]$ liegt nur das Kurvenstück $g_j(h)$. Dann ist $D(h) = g_j(h)$ für $h \in I_k$.

(4) In I_k liegen die Kurvenstücke $f_i(h)$ und $g_j(h)$. $I_k = [a_i, a'_i] \cap [b_j, b'_j]$. Wenn sich die beiden Kurven in höchstens m Schnittpunkten $s_1 < s_2 < \ldots < s_m$ schneiden, zerfällt I_k in maximal $m+1$ Teilintervalle $I_{k,0} = [c_k, s_1], \ldots, I_{k,m} = [s_m, c_{k+1}]$. Innerhalb jedes Intervalls $I_{k,m'}, 0 \leq m' \leq m$ ist $D(h)$ eindeutig bestimmt:

$$D(h) = \begin{cases} f_i(h) & \text{falls} \quad f_i(h) \geq g_j(h) \quad \forall h \in I_{k,m'} \\ g_j(h) & \text{falls} \quad f_i(h) \leq g_j(h) \quad \forall h \in I_{k,m'} \end{cases}$$

Aufgrund von (4) könnte man befürchten, daß die Größe der Konturen beim "Zusammenmischen" unaufhaltsam steigt. Wenn die Zahl der Schnittpunkte zwischen zwei Kurven durch eine Konstante m beschränkt ist, kann das jedoch nicht vorkommen, wie sich aus der Theorie

der *Davenport–Schinzel*–Folgen [DS65, Sz74] ableiten läßt. In [Sh88b] wird gezeigt, daß die Größe solcher Konturen stets durch $O(n \log^* n)$ beschränkt ist. Beim Mischen zweier Konturen, die aus je $n/2$ Kurvensegmenten (beschränkter Ordnung) hervorgegangen sind, benötigt man also höchstens Zeit $O(2 \cdot \frac{n}{2} \log^* \frac{n}{2}) = O(n \log^* n)$, so daß wir in den $\log n$ Rekursionsstufen insgesamt Zeit $O(n \log n \log^* n)$ verbrauchen.
Die Konvexität der beiden Polyeder spielt nur insofern eine Rolle, als daß die Konturen als Funktionen der Variable h beschreibbar sind.
[GSS89] beschreiben eine Methode, um die Kontur einer Zusammenhangskomponente eines Arrangements von n Kurvenstücken zu berechnen. Die Zusammenhangskomponente ist durch einen Punkt spezifiziert, der in ihr liegt. Unter der Voraussetzung, daß je zwei der Kurven nur konstant viele Punkte gemeinsam haben, erzielen sie eine Laufzeit in der Größenordung $O(n \log^2 n \log^* n)$. Dies ist möglich, weil auch hier die Größe der gesuchten Kontur nicht stärker als $O(n \log^* n)$ wachsen kann. Genauso wie in dem oben dargestellten Spezialfall baut ihr Algorithmus auf dem Divide–And–Conquer Prinzip auf, und der Mischschritt wird mittels eines Plane–Sweeps realisiert.
Wir können dieses Resultat benutzen, um das Kollisionsproblem für Volldrehungen auch für nicht konvexe Polyeder effizient zu lösen. Für die Polyeder P_1 und P_2 arbeiten wir mit den 2–dimensionalen Arrangements $\mathcal{A}(P_1)$ und $\mathcal{A}(P_2)$, die von den Mengen der Hyperbeln $\{d_e(h) | h \in I_e\}$ gebildet werden, wobei e die Menge $E_1 \cup E_1^\perp$ bzw. $E_2 \cup E_2^\perp$ durchläuft. Da wir von $\mathcal{A}(P_i)$ jeweils nur eine Zusammenhangskomponente berechnen müssen, erhalten wir das folgende Resultat.

Lemma 1.5 *Die Frage, ob ein Polyeder P_1 in Gegenwart eines Polyeders P_2 eine Volldrehung ausführen kann, läßt sich in Zeit $O(|P_1| \log^2 |P_1| \log^* |P_1| + |P_2| \log^2 |P_2| \log^* |P_2|)$ entscheiden. Wenn P_1 und P_2 konvex sind, benötigt man nur Zeit $O(|P_1| \log |P_1| \log^* |P_1| + |P_2| \log |P_2| \log^* |P_2|)$.*

Dieses Ergebnis läßt sich natürlich auf den Fall translatorischer Bewegungen übertragen. Statt mit Hyperbelsegmenten muß man dort nur mit Liniensegmenten operieren. Der resultierende Algorithmus kann entscheiden, ob P_1 entlang einer vorgegebenen Gerade verschoben werden kann, ohne mit P_2 zu kollidieren.

1.6.7 3D–Faceboxing bei zeitlich begrenzten Bewegungen

Statt zu fragen, wie lange man ein Polyeder P_1 verschieben oder drehen kann, ohne mit einem Hindernis P_2 zusammenzustoßen, interessiert man sich oft für die Kollisionsfreiheit einer zeitlich begrenzten Bewegung:

$$t_{col}(P_1(t), P_2(t))|_{t_1}^{t_2} := \begin{cases} \infty \quad \text{falls} \quad P_1(t) \cap P_2(t) = \emptyset \quad \forall\, t \in [t_1, t_2] \\ \min\{t \in [t_1, t_2] | P_1(t) \cap P_2(t) \neq \emptyset\} \quad \text{sonst} \end{cases}$$

In den vorangegangenen Abschnitten bestand der erste Schritt zur effizienten Berechnung des Kollisionszeitpunktes $t_{col}^*(P_1, P_2)$ stets aus einer geeigneten Projektion, wodurch sich die Dimension der Problemstellung entsprechend verminderte. Zur Berechnung von $t_{col}^*(P_1, P_2)|_{t_1}^{t_2}$ ($* = trans$ oder $* = rot$) kann man natürlich die Heuristiken, die auf der Projektionsmethode basieren, benutzen. Es ist aber überflüssig, eine Begrenzungsfläche $f_1 \in P_1$ auf Kollision mit

einer Hindernisfläche $f_2 \in P_2$ zu testen, wenn aufgrund der kurzen Bewegungsdauer f_2 für f_1 unerreichbar ist, selbst wenn sich ihre Projektionen $\overline{f}_1$ und $\overline{f}_2$ schneiden.

In dieser Situation könnte man prinzipiell so vorgehen, daß man zu jeder bewegten Fläche f_1 das überstrichene räumliche Gebiet $f_1^* = \{\mathbf{x}(t)|\mathbf{x} \in f_1, t \in [t_1, t_2]\}$ konstruiert und unter all den Hindernisflächen, die f_1^* schneiden, die zuerst getroffene bestimmt. Dazu könnte man die Begrenzungsflächen der Hindernisse so vorverarbeiten, daß man für ein beliebig vorgegebenes Gebiet f_1^* schnell die in Frage kommenden Flächen bestimmen kann. Während f_1^{trans} noch als Polyeder beschrieben werden kann, besitzt f_1^{rot} hyperbolische Begrenzungsflächen, was diese Aufgabe zusätzlich erschweren würde. Deshalb schließen wir die von den bewegten Flächen überstrichenen Gebiete und auch die Hindernisflächen in achsenparallele Quader ein. Die Quader der bewegten Flächen färben wir rot, die anderen blau. Ebenso wie im 2–dimensionalen Fall verfügt man über einen effizienten Algorithmus zur Bestimmung der Rot–Blau Schnitte dieser Quadermengen (siehe z.B. [EM81]).

> Gegeben seien zwei Mengen B_1 und B_2 von achsenparallelen Quadern. Die Menge $C_{B_1,B_2} = \{(b_1, b_2)|b_1 \in B_1, b_2 \in B_2, b_1 \cap b_2 \neq \emptyset\}$ kann in Zeit $O((|B_1| + |B_2|)\log^2(|B_1| + |B_2|) + |C_{B_1,B_2}|)$ berechnet werden.

Dieses Ergebnis kann man analog zum 2–dimensionalen Fall mit Hilfe eines Sweeps erzielen. Dabei muß man in der Sweep–Ebene Rechteckmengen dynamisch verwalten. Dazu eignen sich verschachtelte Segment– und Bereichsbäume, die Einfügen und Löschen in quadratisch–logarithmischer Zeit erlauben.

Bewegt sich ein Polyeder P_1 nur über eine kurze Entfernung in einer Hinderniswelt, die aus mehreren Polyedern $P_2, \ldots, P_n$ besteht, so wird er nur mit den Polyedern in seiner unmittelbaren Nachbarschaft zusammenstoßen können. In diesem Fall wäre es Zeitvergeudung, $t_{col}(P_1, P_i)$ mit der obigen Methode für alle $i = 2, \ldots, n$ separat zu berechnen. Statt dessen schließen wir alle Hindernisflächen der Polyeder $P_2, \ldots, P_n$ in achsenparallele Quader (B) ein und verwalten diese gemeinsam in einer geeigneten Datenstruktur. Die von den Flächen des bewegten Polyeders P_1 überstrichenen Raumbereiche approximieren wir ebenfalls durch Quader (B'). Für jeden Quader $b' \in B'$ ermitteln wir einzeln die Hindernisquader $b \in B$, die b' schneiden, indem wir die Datenstruktur für B befragen.

Auf diese Weise stoßen wir auf ein Problem, das in der Literatur unter dem Namen *Dynamic Orthogonal Intersection Query* geführt wird. Aus [Meh84] (Abschnitt 8.5.3.) ist folgendes Resultat bekannt:

> Gegeben sei eine dynamische Menge B von achsenparallelen Quadern. Für jeden beliebigen Quader b' kann man die Menge $C_{b',B} = \{b \in B|b' \cap b \neq \emptyset\}$ in Zeit $O(\log^3|B| + |C_{b',B}|)$ berechnen. Der Platzbedarf zur Speicherung von B beträgt $O(|B|\log^2|B|)$. Einfügen und Löschen von Quadern kosten amortisiert $O(\log^3|B|)$.

Es erweist sich als günstig, daß man die Menge der Hindernisquader sogar dynamisch effizient verwalten kann. Betrachten wir nämlich einen Bewegungsablauf in einer Szene von mehreren Polyedern, bei dem in jedem Schritt genau ein Polyeder bewegt wird während

alle anderen ihre Lage unverändert lassen, so ändert sich die Hinderniswelt dynamisch, weil unterschiedliche Polyeder die Rolle des bewegten Objektes übernehmen können.
Um einen solchen Bewegungsablauf auf Kollisionsfreiheit zu überprüfen, nehme man zuerst sämtliche Hüllquader aller Polyederflächen in die Menge B auf. Wenn der Polyeder P_i durch eine Bewegung seine Lage verändern möchte, werden die zu F_i gehörigen Quader aus B entfernt. Die von den Flächen F_i überstrichenen Raumbereiche werden durch Quader approximiert und bilden die Menge B'. Für jeden Quader $b' \in B'$ liefert uns die Datenstruktur für B alle von b' geschnittenen Hindernisquader ohne große zusätzliche Kosten. Die Quaderschnitte informieren uns darüber, welche Polyederflächen kollidieren können. Nur für diese führt man dann einen exakten Kollisionstest durch. Danach kann P_i seine neue Lage einnehmen. Dazu werden die neuen einhüllenden Quader für F_i berechnet und in die Menge B aufgenommen.
Bei dieser dreistufigen Vorgehensweise verteilen sich die Kosten wie folgt: Das Entfernen und das spätere Einfügen der Hüllquader der Flächen F_i kosten amortisiert jeweils $O(|F_i| \log^3 |B|)$. Der Kollisionstest selbst kostet ebenfalls $O(|F_i| \log^3 |B|)$, jedoch zuzüglich der Anzahl der Quaderschnitte.

Kapitel 2

Elementare Kollisionsberechnungen

In diesem Kapitel wollen wir die mathematischen Grundlagen zur Berechnung von Kollisionen zwischen einem translatorisch (rotatorisch) bewegten und einem stationären Polyeder erarbeiten. Neben der reinen Berechnung von Kollisionszeitpunkten t^*_{col} ($* = trans$ oder $* = rot$) interessiert uns auch die Frage nach eingehaltenen Sicherheitsabständen Δ^*_{min} und die Frage nach der Kollision der zugehörigen Hüllkörper, wie Kugeln oder Zylinder. Wir werden zeigen, daß sich t^*_{col} und Δ^*_{min} ohne numerische Approximationsmethoden direkt ermitteln lassen, weil sich die notwendigen Berechnungen auf die Nullstellenbestimmung von Polynomen höchstens vierten Grades zurückführen lassen.
Ein Teil der Resultate wurde mit Hilfe eines Algebra–Systems gefunden. Derzeit verfügbare Algebra–Systeme unterstützen jedoch symbolische Berechnungen mit Vektoren und Matrizen nur komponentenweise und sind deshalb nur zur Verifikation der Ergebnisse von wirklichem Nutzen. Wir bemühen uns um eine vektorielle Schreibweise, so daß sich Symmetrien, die bereits in der Problemstellung erkennbar sind, in der vektoriellen Formulierung der Resultate widerspiegeln. Dies führt zu wesentlich kompakteren Formeln, die sich effizienter auswerten lassen und somit zur Beschleunigung der Algorithmen beitragen.

2.1 Berechnung von Kollisionszeitpunkten

2.1.1 Kollision zwischen einem Punkt und einer Fläche

Gegeben seien ein Punkt $\mathbf{a}$ und ein konvexes Polygon f. Das Polygon f liege in der Ebene $H = \{\mathbf{x} \,|\, \mathbf{n}^T\mathbf{x} = n_0\}$. Die Eckpunkte von f in zyklischer Reihenfolge bezeichnen wir mit $\mathbf{v_0}, \mathbf{v_1}, \ldots \mathbf{v_{k-1}}, \mathbf{v_k} \equiv \mathbf{v_0}$. Das Vorzeichen des Normalenvektors $\mathbf{n}$ sei so gewählt, daß die Randpunkte im Gegenuhrzeigersinn aufgezählt sind, wenn man in Richtung $-\mathbf{n}$ auf f blickt. Eine numerisch günstige Art und Weise, die Ebenengleichung aus den Koordinaten der Eckpunkte zu bestimmen, lautet:

$$\mathbf{n} = \sum_{i=0}^{k-1} (\mathbf{v_i} \times \mathbf{v_{i+1}}) \qquad n_0 = \frac{1}{k}\mathbf{n}^T \sum_{i=0}^{k-1} \mathbf{v_i}$$

Der Normalenvektor sollte anschließend noch normiert werden. Diese Konventionen erlauben, in einfacher Weise zu testen, ob ein Punkt $\mathbf{a'}$ der Ebene H zum konvexen Polygon f gehört,

denn es gilt:

$$\mathbf{a}' \in f \iff \forall i : \det[\mathbf{n}, \mathbf{a}' - \mathbf{v_i}, \mathbf{v_{i+1}} - \mathbf{v_i}] \leq 0$$

2.1.1.1 Die Berechnung von $t_{col}^{trans}(\mathbf{a}, f)$

Wir beantworten zuerst die Frage, wann $\mathbf{a}$ bei einer Translation in Richtung $\mathbf{s}$ auf die Ebene H stößt. Dabei setzen wir voraus, daß $\mathbf{n}^T\mathbf{s} \neq 0$ ist.

$$\mathbf{n}^T(\mathbf{a} + t\mathbf{s}) = n_0 \iff t = \frac{n_0 - \mathbf{n}^T\mathbf{a}}{\mathbf{n}^T\mathbf{s}} \tag{2.1}$$

Sollte $t < 0$ sein, so bewegt sich $\mathbf{a}$ von H weg. Sei $\mathbf{a}' = \mathbf{a} + t\mathbf{s}$, so gilt:

$$t_{col}^{trans}(\mathbf{a}, f) = \begin{cases} t & \text{falls} \quad t \geq 0 \ \wedge \ \mathbf{a}' \in f \\ \infty & \text{sonst} \end{cases}$$

2.1.1.2 Die Berechnung von $t_{col}^{rot}(\mathbf{a}, f)$

Wir verfahren analog zu der Berechnung von $t_{col}^{trans}(\mathbf{a}, f)$. Die Richtung $\mathbf{r}$ der Rotationsachse sei normiert ($|\mathbf{r}| = 1$). Aus $\mathbf{n}^T\mathbf{R}(t)\mathbf{a} = n_0$ erhalten wir gemäß Formel (A.10) die Gleichung

$$\mathbf{n}^T\mathbf{r}\mathbf{r}^T\mathbf{a} + \cos t\,(\mathbf{n}^T\mathbf{a} - \mathbf{n}^T\mathbf{r}\mathbf{r}^T\mathbf{a}) + \sin t\,\mathbf{r}^T(\mathbf{a} \times \mathbf{n}) = n_0$$

Diese ist von der Form

$$\alpha \ \cos t + \beta \ \sin t + \gamma = 0$$

mit den Koeffizienten

$$\begin{aligned} \alpha &= (\mathbf{n} \times \mathbf{r})^T(\mathbf{a} \times \mathbf{r}), \\ \beta &= \mathbf{a}^T(\mathbf{n} \times \mathbf{r}), \\ \gamma &= \mathbf{a}^T\mathbf{r}\mathbf{r}^T\mathbf{n} - n_0. \end{aligned}$$

Obige Gleichung hat im allgemeinen zwei Lösungen (vgl. (A.14)). Wir bezeichnen sie mit $t_1, t_2 \in [0, 2\pi)$. Mit $\mathbf{a_i} = \mathbf{R}(t_i)\mathbf{a}$ erhalten wir:

$$t_{col}^{rot}(\mathbf{a}, f) = \begin{cases} \infty & \text{falls} \quad \mathbf{a_1} \notin f \ \wedge \ \mathbf{a_2} \notin f \\ \min\{t_i \mid \mathbf{a_i} \in f, \ i = 1, 2\} & \text{sonst} \end{cases}$$

Dies gilt natürlich nur unter der Voraussetzung, daß keine Entartung aufgetreten ist. Falls $\mathbf{n} \parallel \mathbf{r} \quad \Rightarrow \quad \mathbf{n} \times \mathbf{r} = \mathbf{0} \quad \Rightarrow \quad \alpha = \beta = 0$. Falls $\gamma \neq 0$, so kann keine Kollision auftreten, weil das Gleichungssystem keine Lösung besitzt. Im Fall $\gamma = 0$ rotiert der Punkt $\mathbf{a}$ in der gleichen Ebene, in der auch f liegt, und eine Kollision tritt dann auf, wenn $\mathbf{a}$ auf eine Begrenzungskante von f stößt.

2.1.2 Kollision zwischen zwei Liniensegmenten

Gegeben seien zwei Liniensegmente mit den Endpunkten **a** und **b** bzw. **c** und **d**:

$$\begin{aligned} l_{ab} &= \{\mathbf{a} + \mu(\mathbf{b} - \mathbf{a}) \,|\, 0 \leq \mu \leq 1\} \\ l_{cd} &= \{\mathbf{c} + \nu(\mathbf{d} - \mathbf{c}) \,|\, 0 \leq \nu \leq 1\} \end{aligned}$$

Die zugehörigen Geraden bezeichnen wir mit L_{ab} und L_{cd}. Das Liniensegment l_{ab} bewege sich translatorisch oder rotatorisch und l_{cd} sei fest. Wir möchten folgende Minima berechnen:

$$\begin{aligned} t^*_{col}(l_{ab}, l_{cd}) &= \begin{cases} \infty \quad \text{falls} \quad l^*_{ab}(t) \cap l_{cd} = \emptyset \quad \forall\, t \geq 0 \\ \min\{t \geq 0 \,|\, l^*_{ab}(t) \cap l_{cd} \neq \emptyset\} \quad \text{sonst} \end{cases} \\ &\text{wobei} \quad * = trans \text{ oder } * = rot \end{aligned}$$

2.1.2.1 Die Berechnung von $t^{trans}_{col}(l_{ab}, l_{cd})$

Zunächst wollen wir den minimalen Abstand zwischen l_{ab} und l_{cd} in Richtung **s** bestimmen. Deshalb suchen wir die Lösung der Gleichung

$$\mathbf{a} + \mu(\mathbf{b} - \mathbf{a}) + t\mathbf{s} = \mathbf{c} + \nu(\mathbf{d} - \mathbf{c}) \quad \text{mit} \quad 0 \leq t, 0 \leq \mu, \nu \leq 1$$

Dieses lineare Gleichungssystem lösen wir mit Hilfe der Kramer'schen Regel (siehe (A.2)). Dabei setzen wir voraus, daß $\det[\mathbf{s}, \mathbf{b} - \mathbf{a}, \mathbf{d} - \mathbf{c}] \neq 0$ ist. Es folgt:

$$t = \frac{\det[\mathbf{c} - \mathbf{a}, \mathbf{b} - \mathbf{a}, \mathbf{d} - \mathbf{c}]}{\det[\mathbf{s}, \mathbf{b} - \mathbf{a}, \mathbf{d} - \mathbf{c}]} \tag{2.2}$$

$$\mu = \frac{\det[\mathbf{s}, \mathbf{c} - \mathbf{a}, \mathbf{d} - \mathbf{c}]}{\det[\mathbf{s}, \mathbf{b} - \mathbf{a}, \mathbf{d} - \mathbf{c}]} \tag{2.3}$$

$$\nu = \frac{\det[\mathbf{s}, \mathbf{c} - \mathbf{a}, \mathbf{b} - \mathbf{a}]}{\det[\mathbf{s}, \mathbf{b} - \mathbf{a}, \mathbf{d} - \mathbf{c}]} \tag{2.4}$$

Daraus erhalten wir die Formel zur Berechnung des Kollisionszeitpunktes für zwei Liniensegmente. Wenn $\det[\mathbf{s}, \mathbf{b} - \mathbf{a}, \mathbf{d} - \mathbf{c}] \neq 0$ ist, gilt:

$$t^{trans}_{col}(l_{ab}, l_{cd}) = \begin{cases} t \quad \text{falls} \quad t \geq 0,\ 0 \leq \mu, \nu \leq 1 \\ \infty \quad \text{sonst} \end{cases}$$

Anderenfalls liegt eine entartete Situation vor, die sich im wesentlichen als ein 2–dimensionales Problem darstellt. Dann gilt:

$$t^{trans}_{col}(l_{ab}, l_{cd}) = \min\{t^{trans}_{col}(\{\mathbf{a}, \mathbf{b}\}, l_{cd}), t^{trans}_{col}(l_{ab}, \{\mathbf{c}, \mathbf{d}\})\}$$

Anstelle einer detaillierten Diskussion des obigen Sonderfalls betrachten wir den allgemeinen Fall genauer.
In der Formel für t sollten die Variablen $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ gleichberechtigt vorkommen. Eine symmetrische Form findet man, wenn man die Determinante im Zähler von (2.2) entwickelt:

$$\begin{aligned} -\det[\mathbf{c} - \mathbf{a}, \mathbf{b} - \mathbf{a}, \mathbf{d} - \mathbf{c}] &= \det(\mathbf{M}), \\ \text{wobei} \quad \mathbf{M} &:= \begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix} \in \mathbb{R}^{4\times 4} \end{aligned}$$

Die Matrix $\mathbf{M}$ enthält in der ersten Zeile nur Einsen. Die drei übrigen Zeilen sind aus den Vektoren $\mathbf{a}, \mathbf{b}, \mathbf{c}$ und $\mathbf{d}$ aufgebaut. $\det(\mathbf{M}) = \det[\mathbf{b}-\mathbf{a}, \mathbf{c}-\mathbf{a}, \mathbf{d}-\mathbf{a}]$ stellt ein Spatprodukt dar und ist proportional zum Volumen des Tetraeders, der von den vier Punkten aufgespannt wird. Wenn $\det(\mathbf{M}) = 0$ gilt, dann liegen die vier Punkte in einer Ebene. Dies gilt insbesondere auch dann, wenn die beiden Liniensegmente l_{ab} und l_{cd} parallel sind.

$$\det(\mathbf{M}) = 0 \iff (L_{ab} \cap L_{cd} \neq \emptyset) \vee (L_{ab} \parallel L_{cd}) \tag{2.5}$$

Wenn wir die Abkürzungen

$$\mathbf{M}_a := \begin{bmatrix} 0 & 1 & 1 & 1 \\ \mathbf{s} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix}, \quad \mathbf{M}_b := \begin{bmatrix} 1 & 0 & 1 & 1 \\ \mathbf{a} & \mathbf{s} & \mathbf{c} & \mathbf{d} \end{bmatrix},$$
$$\mathbf{M}_c := \begin{bmatrix} 1 & 1 & 0 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{s} & \mathbf{d} \end{bmatrix}, \quad \mathbf{M}_d := \begin{bmatrix} 1 & 1 & 1 & 0 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{s} \end{bmatrix}$$

benutzen, dann lassen sich t, μ und ν wie folgt ausdrücken:

$$t = \frac{\det(\mathbf{M})}{\det(\mathbf{M}_a) + \det(\mathbf{M}_b)} \quad \mu = \frac{\det(\mathbf{M}_b)}{\det(\mathbf{M}_a) + \det(\mathbf{M}_b)} \quad \nu = \frac{\det(\mathbf{M}_d)}{\det(\mathbf{M}_c) + \det(\mathbf{M}_d)}$$

Durch Ausnutzung der Beziehung $\det(\mathbf{M}_a) + \det(\mathbf{M}_b) + \det(\mathbf{M}_c) + \det(\mathbf{M}_d) = 0$ lassen sich die Bedingungen $0 \leq \mu, \nu \leq 1$ ebenfalls in eine symmetrische Form überführen:

$$\begin{aligned} &(\det(\mathbf{M}_a), \det(\mathbf{M}_b) \leq 0 \wedge \det(\mathbf{M}_c), \det(\mathbf{M}_d) \geq 0) \\ \vee\ &(\det(\mathbf{M}_a), \det(\mathbf{M}_b) \geq 0 \wedge \det(\mathbf{M}_c), \det(\mathbf{M}_d) \leq 0) \end{aligned} \tag{2.6}$$

Dieses Ungleichungssystem wird uns in Abschnitt 3.3 wiederbegegnen. Es drückt aus, unter welchen Umständen das Liniensegment l_{ab} bei einer Verschiebung in Richtung $\mathbf{s}$ mit dem Liniensegment l_{cd} zusammenstoßen kann.

Wählen wir die Verschiebungsrichtung $\mathbf{s} = (\mathbf{b}-\mathbf{a}) \times (\mathbf{d}-\mathbf{c})$ senkrecht zu den Richtungen von l_{ab} und l_{cd}, so erhalten wir aus Gleichung (2.2) den euklidischen Abstand $\delta(L_{ab}, L_{cd})$ der zugehörigen Geraden, vorausgesetzt daß $L_{ab} \nparallel L_{cd}$ ist:

$$\delta(L_{ab}, L_{cd}) = \frac{\left|\det \begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix}\right|}{|(\mathbf{b}-\mathbf{a}) \times (\mathbf{d}-\mathbf{c})|} \tag{2.7}$$

Wenn wir wissen wollen, ob die Endpunkte der kürzesten Verbindungsstrecke zwischen L_{ab} und L_{cd} auch auf den entsprechenden Liniensegmenten liegen, brauchen wir nur die Ungleichungen (2.6) zu überprüfen. Bei dieser besonderen Wahl von $\mathbf{s}$ vereinfachen sich diese zu

$$\begin{aligned} &\det[\mathbf{s}, \mathbf{d}-\mathbf{c}, \mathbf{a}] \geq \det[\mathbf{s}, \mathbf{d}, \mathbf{c}] \geq \det[\mathbf{s}, \mathbf{d}-\mathbf{c}, \mathbf{b}] \\ \wedge\ &\det[\mathbf{s}, \mathbf{b}-\mathbf{a}, \mathbf{c}] \leq \det[\mathbf{s}, \mathbf{b}, \mathbf{a}] \leq \det[\mathbf{s}, \mathbf{b}-\mathbf{a}, \mathbf{d}]. \end{aligned} \tag{2.8}$$

Für diese Bedingungen führen wir ein eigenes Prädikat $\chi(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$ ein. Dieses Prädikat ist insbesondere dann erfüllt, wenn $l_{ab} \cap l_{cd} \neq \emptyset$ oder $l_{ab} \parallel l_{cd}$ gilt.

2.1.2.2 Die Berechnung von $t_{col}^{rot}(l_{ab}, l_{cd})$

Nun wollen wir ein rotierendes Liniensegment l_{ab} betrachten und fragen, wann es mit einem stationären Liniensegment l_{cd} zusammenstößt.

$$\mathbf{R}(t)(\mathbf{a} + \mu(\mathbf{b} - \mathbf{a})) = \mathbf{c} + \nu(\mathbf{d} - \mathbf{c}) \quad \text{mit} \quad 0 \leq t, 0 \leq \mu, \nu \leq 1 \tag{2.9}$$

Dieses Gleichungssystem enthält die Unbekannten t, μ, ν, die im folgenden bestimmt werden sollen. Wir benutzen hierzu die Abkürzung $\mathbf{R} \equiv \mathbf{R}(t)$. Wenn die rotierende Gerade L_{ab} die feste Gerade L_{cd} schneidet, dann gilt nach Gleichung (2.5):

$$\begin{aligned} & \det \begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{Ra} & \mathbf{Rb} & \mathbf{c} & \mathbf{d} \end{bmatrix} = 0 \\ \iff & \det[\mathbf{R}(\mathbf{b} - \mathbf{a}), \mathbf{c}, \mathbf{d}] + \det[\mathbf{Ra}, \mathbf{Rb}, \mathbf{d} - \mathbf{c}] = 0 \\ \iff & (\mathbf{c} \times \mathbf{d})^T \mathbf{R}(\mathbf{b} - \mathbf{a}) + (\mathbf{d} - \mathbf{c})^T \mathbf{R}(\mathbf{a} \times \mathbf{b}) = 0 \end{aligned} \tag{2.10}$$

Mit Hilfe von Formel (A.10) können wir Gleichung (2.10) auflösen. Wir erhalten eine Gleichung der Form

$$\alpha \cos t + \beta \sin t + \gamma = 0 \tag{2.11}$$

mit den Koeffizienten

$$\begin{aligned} \alpha &= (\mathbf{d} - \mathbf{c})^T (\mathbf{a} \times \mathbf{b}) - \mathbf{r}^T (\mathbf{d} - \mathbf{c}) \mathbf{r}^T (\mathbf{a} \times \mathbf{b}) \\ &\quad + (\mathbf{b} - \mathbf{a})^T (\mathbf{c} \times \mathbf{d}) - \mathbf{r}^T (\mathbf{b} - \mathbf{a}) \mathbf{r}^T (\mathbf{c} \times \mathbf{d}), \\ \beta &= \mathbf{r}^T ((\mathbf{b} - \mathbf{a}) \times (\mathbf{c} \times \mathbf{d}) - (\mathbf{d} - \mathbf{c}) \times (\mathbf{a} \times \mathbf{b})), \\ \gamma &= \mathbf{r}^T (\mathbf{d} - \mathbf{c}) \mathbf{r}^T (\mathbf{a} \times \mathbf{b}) + \mathbf{r}^T (\mathbf{b} - \mathbf{a}) \mathbf{r}^T (\mathbf{c} \times \mathbf{d}). \end{aligned}$$

Man beachte die erwartete Symmetrie dieser Formeln, was die Rollen von $(\mathbf{a}, \mathbf{b})$ und $(\mathbf{c}, \mathbf{d})$ betrifft! Man erkennt, daß eine Vertauschung von $(\mathbf{a}, \mathbf{b})$ mit $(\mathbf{c}, \mathbf{d})$ gerade eine Vorzeichenumkehr von β bewirkt, was einer Negation der Lösungswinkel t gleichkommt. Wie geometrisch zu erwarten ist, sind die Formeln invariant gegenüber einer gemeinsamen Verschiebung der Liniensegmente in Richtung $\mathbf{r}$.

Entartung tritt immer dann ein, wenn $\alpha = \beta = \gamma = 0$ gilt. Das ist der Fall, wenn $(\mathbf{r} \parallel \mathbf{b} - \mathbf{a}) \wedge (\mathbf{r} \parallel \mathbf{d} - \mathbf{c})$ oder wenn $(\mathbf{r} \perp \mathbf{b} - \mathbf{a}) \wedge (\mathbf{r} \perp \mathbf{d} - \mathbf{c})$ oder wenn die beiden Geraden L_{ab} und L_{cd} einen gemeinsamen Punkt auf der Rotationsachse haben. Im letztgenannten Fall dürfen wir aufgrund der Verschiebungsinvarianz annehmen, daß sich L_{ab} und L_{cd} im Ursprung schneiden. Dann gilt $\mathbf{a} \times \mathbf{b} = \mathbf{c} \times \mathbf{d} = \mathbf{0} \Rightarrow \alpha = \beta = \gamma = 0$.

Bis jetzt haben wir so getan, als ob wir die Kollision für die beiden Geraden L_{ab} und L_{cd} durchführen wollten. Gleichung (2.11) besitzt im allgemeinen zwei Lösungen (vgl. (A.14)), die wir mit $t_1, t_2 \in [0, 2\pi)$ bezeichnen. Für jeden dieser Drehwinkel müssen wir untersuchen, ob $l_{ab}^{rot}(t_i) \cap l_{cd} \neq \emptyset$ ist. Um diese Inzidenz zu testen, ziehen wir das Prädikat χ (siehe (2.8)) zu Rate. Ebenso wie für die oben genannten entarteten Fälle muß für den Fall $l_{ab}^{rot}(t_i) \parallel l_{cd}$ eine Sonderbehandlung durchgeführt werden. Für den allgemeinen Fall erhalten wir die gesuchte Formel

$$\begin{aligned} t_{col}^{rot}(l_{ab}, l_{cd}) &= \begin{cases} \infty \quad \text{falls} \quad \chi_1 = \chi_2 = 0 \\ \min\{t_i \mid \chi_i = 1,\ i = 1, 2\} \quad \text{sonst}, \end{cases} \\ & \text{wobei} \quad \chi_i = \chi(\mathbf{R}(t_i)\mathbf{a}, \mathbf{R}(t_i)\mathbf{b}, \mathbf{c}, \mathbf{d}). \end{aligned}$$

2.1.3 Kollision zwischen Kugeln und Zylindern

Kugeln eignen sich in besonderer Weise als Hüllkörper bei der Betrachtung von Kollisionen bewegter Objekte, weil sich die Frage nach einer Kollision von Kugeln als einfaches Abstandsproblem ihrer Mittelpunkte stellt. Für zwei Kugeln $S_i = \{\mathbf{x}|\delta(\mathbf{x}, \mathbf{m_i}) \leq \rho_i\}$ $(i = 1, 2)$ gilt:

$$t_{col}(S_1(t), S_2(t)) \quad = \quad \min\{t \geq 0 | \delta(\mathbf{m_1}(t), \mathbf{m_2}(t)) = \rho_1 + \rho_2\}$$

Bewegt sich eine Kugel auf einer Geraden oder einer Kreisbahn während die andere ihre Position beibehält, so erhalten wir einfache quadratische Gleichungen zur Bestimmung des Kollisionszeitpunktes t_{col} (siehe Abschnitte 2.1.3.1 und 2.1.3.4).

Kugeln sind zwar die einfachsten Hüllkörper, aber in vielen Fällen approximieren sie die Form eines gegebenen Objektes nur unzureichend. Bei langgestreckten Objekten ist es günstiger, statt Kugeln Zylinder zu benutzen. Wenn man zusätzlich auf die Grundflächen des Zylinders Halbkugeln aufsetzt, reduziert sich das Kollisionsproblem auf ein Abstandsproblem für die Zylinderachsenabschnitte.

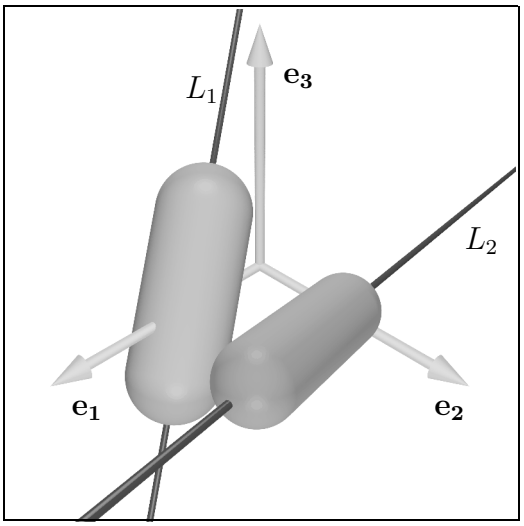


Abbildung 2.1: Kollision zwischen zwei Zylindern mit aufgesetzten Halbkugeln

An die Stelle der Bestimmung des Abstandes zwischen zwei Punkten tritt nun die Abstandsberechnung zwischen Liniensegmenten, was eine einfache Verallgemeinerung des obigen Prinzips darstellt. Betrachten wir zwei Liniensegmente l_1 und l_2 als Achsenabschnitte von zwei Zylindern mit Kugelkappen $C_i = \{\mathbf{x}|\delta(\mathbf{x}, l_i) \leq \rho_i\}$ $(i = 1, 2)$, so gilt:

$$t_{col}(C_1(t), C_2(t)) \quad = \quad \min\{t \geq 0 | \delta(l_1(t), l_2(t)) = \rho_1 + \rho_2\}$$

Der Abstand zwischen zwei windschiefen Liniensegmenten $l_1 = l_{ab}$ und $l_2 = l_{cd}$ ist gleich dem Abstand der zugehörigen Geraden L_{ab} und L_{cd}, falls die Endpunkte der kürzesten Verbindungsstrecke zwischen L_{ab} und L_{cd} auf den Liniensegmenten liegen. Diese Eigenschaft

wird durch das Prädikat $\chi(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d})$ ausgedrückt (siehe (2.8)). Ansonsten bestimmt einer der Punkte **a**, **b**, **c** oder **d** selbst den Abstand zum anderen Liniensegment. Also gilt:

$$\delta(l_{ab},l_{cd}) \;=\; \begin{cases} \delta(L_{ab},L_{cd}) \quad \text{falls} \quad \chi(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d})=1 \\ \min\{\delta(l_{ab},\mathbf{c}),\delta(l_{ab},\mathbf{d}),\delta(l_{cd},\mathbf{a}),\delta(l_{cd},\mathbf{b})\} \quad \text{sonst} \end{cases}$$

Falls die Geraden nicht parallel sind, können wir zur Berechnung von $\delta(L_{ab},L_{cd})$ die Formel (2.7) benutzen. Bei Parallelität reduziert sich das Problem auf die Abstandsbestimmung zwischen einem Punkt und einer Geraden.

Der Abstand zwischen einem Punkt **c** und einem Liniensegment l_{ab} ist gleich dem Abstand des Punktes **c** von der Geraden L_{ab}, falls der Endpunkt des Lotes von **c** auf L_{ab} auf dem Liniensegment l_{ab} liegt. Dies beschreiben wir durch das Prädikat $\psi(\mathbf{a},\mathbf{b},\mathbf{c})$. Ist dies nicht der Fall, bestimmt einer der Punkte **a** oder **b** den Abstand von l_{ab} zu **c**.

$$\delta(l_{ab},\mathbf{c}) \;=\; \begin{cases} \delta(L_{ab},\mathbf{c}) \quad \text{falls} \quad \psi(\mathbf{a},\mathbf{b},\mathbf{c})=1 \\ \min\{\delta(\mathbf{a},\mathbf{c}),\delta(\mathbf{b},\mathbf{c})\} \quad \text{sonst} \end{cases}$$

Für den Abstand des Punktes **c** zu der Geraden L_{ab} und das Prädikat $\psi(\mathbf{a},\mathbf{b},\mathbf{c})$ gilt:

$$\begin{aligned} \delta(L_{ab},\mathbf{c}) &= \frac{|\mathbf{a}\times\mathbf{b}+(\mathbf{b}-\mathbf{a})\times\mathbf{c}|}{|\mathbf{b}-\mathbf{a}|} \\ \psi(\mathbf{a},\mathbf{b},\mathbf{c}) &: (\mathbf{b}-\mathbf{a})^T\mathbf{a} \le (\mathbf{b}-\mathbf{a})^T\mathbf{c} \le (\mathbf{b}-\mathbf{a})^T\mathbf{b} \end{aligned}$$

Die Berechnung des Zeitpunktes t_{col}, zu dem die bewegten Liniensegmente $l_1(t)$ und $l_2(t)$ den Abstand $\rho_1+\rho_2=\rho$ annehmen, können wir somit auf Abstandsberechnungen zwischen Geraden und Punkten zurückführen.

$$t_{col} \;=\; \min \bigcup_{i=1}^{8} \mathcal{T}_i \qquad \text{mit}$$

$$\begin{aligned} \mathcal{T}_1 &= \{t\ge 0 | \delta(L_{ab}(t),L_{cd}(t))=\rho \wedge \chi(\mathbf{a}(t),\mathbf{b}(t),\mathbf{c}(t),\mathbf{d}(t))=1\} \\ \mathcal{T}_2 &= \{t\ge 0 | \delta(L_{ab}(t),\mathbf{c}(t))=\rho \wedge \psi(\mathbf{a}(t),\mathbf{b}(t),\mathbf{c}(t))=1\} \\ \mathcal{T}_2 &= \{t\ge 0 | \delta(L_{ab}(t),\mathbf{d}(t))=\rho \wedge \psi(\mathbf{a}(t),\mathbf{b}(t),\mathbf{d}(t))=1\} \\ \mathcal{T}_3 &= \{t\ge 0 | \delta(L_{cd}(t),\mathbf{a}(t))=\rho \wedge \psi(\mathbf{c}(t),\mathbf{d}(t),\mathbf{a}(t))=1\} \\ \mathcal{T}_4 &= \{t\ge 0 | \delta(L_{cd}(t),\mathbf{b}(t))=\rho \wedge \psi(\mathbf{c}(t),\mathbf{d}(t),\mathbf{b}(t))=1\} \\ \mathcal{T}_5 &= \{t\ge 0 | \delta(\mathbf{a}(t),\mathbf{c}(t))=\rho\} \\ \mathcal{T}_6 &= \{t\ge 0 | \delta(\mathbf{b}(t),\mathbf{c}(t))=\rho\} \\ \mathcal{T}_7 &= \{t\ge 0 | \delta(\mathbf{a}(t),\mathbf{d}(t))=\rho\} \\ \mathcal{T}_8 &= \{t\ge 0 | \delta(\mathbf{b}(t),\mathbf{d}(t))=\rho\} \end{aligned}$$

Schränken wir die Form der Bewegung so ein, daß ein Liniensegment ruht und das andere sich geradlinig bewegt oder um eine feste Achse rotiert, dann besitzen die Mengen $\mathcal{T}_i$ nur konstant viele Elemente, denn jedes $\mathcal{T}_i$ ist eine Teilmenge der Nullstellen eines Polynoms von höchstens viertem Grad.

In den folgenden Abschnitten diskutieren wir die aufgeführten Fälle genauer:

2.1.3.1 Translation Punkt – Punkt

$$|\mathbf{a} + t\mathbf{s} - \mathbf{c}| = \rho$$

führt zu einer quadratischen Gleichung der Form

$$\alpha t^2 + \beta t + \gamma = 0$$

mit den Koeffizienten

$$\begin{aligned} \alpha &= \mathbf{s}^2, \\ \beta &= 2\mathbf{s}^T(\mathbf{a} - \mathbf{c}), \\ \gamma &= (\mathbf{a} - \mathbf{c})^2 - \rho^2. \end{aligned}$$

2.1.3.2 Translation Gerade – Punkt

$$|\mathbf{a} \times \mathbf{b} + (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} + t\mathbf{s})| = \rho|\mathbf{b} - \mathbf{a}|$$

führt zu einer quadratischen Gleichung der Form

$$\alpha t^2 + \beta t + \gamma = 0$$

mit den Koeffizienten

$$\begin{aligned} \alpha &= (\mathbf{s} \times (\mathbf{b} - \mathbf{a}))^2, \\ \beta &= -2(\mathbf{s} \times (\mathbf{b} - \mathbf{a}))^T(\mathbf{a} \times \mathbf{b} + \mathbf{b} \times \mathbf{c} + \mathbf{c} \times \mathbf{a}), \\ \gamma &= (\mathbf{a} \times \mathbf{b} + \mathbf{b} \times \mathbf{c} + \mathbf{c} \times \mathbf{a})^2 - \rho^2(\mathbf{b} - \mathbf{a})^2. \end{aligned}$$

2.1.3.3 Translation Gerade – Gerade

$$\left|\det \begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{a} + t\mathbf{s} & \mathbf{b} + t\mathbf{s} & \mathbf{c} & \mathbf{d} \end{bmatrix}\right| = \rho\,|(\mathbf{b} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c})|$$

führt zu einer linearen Gleichung der Form

$$\begin{aligned} |\alpha t + \beta| &= \gamma \\ t &= \frac{-\beta \pm \gamma}{\alpha}. \end{aligned}$$

Die Koeffizienten erhält man durch Entwickeln der Determinante:

$$\begin{aligned} \alpha &= \mathbf{s}^T((\mathbf{d} - \mathbf{c}) \times (\mathbf{b} - \mathbf{a})) \\ \beta &= (\mathbf{b} - \mathbf{a})^T(\mathbf{c} \times \mathbf{d}) + (\mathbf{d} - \mathbf{c})^T(\mathbf{a} \times \mathbf{b}) \\ \gamma &= \rho|(\mathbf{b} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c})| \end{aligned}$$

2.1.3.4 Rotation Punkt – Punkt

$$|\mathbf{Ra} - \mathbf{c}| \;=\; \rho$$

ergibt eine Gleichung der Form

$$\alpha \cos t + \beta \sin t + \gamma \;=\; 0$$

mit den Koeffizienten

$$\begin{aligned} \alpha &= 2\mathbf{a}^T\mathbf{b} - 2\mathbf{r}^T\mathbf{a}\mathbf{r}^T\mathbf{b}, \\ \beta &= 2\mathbf{r}^T(\mathbf{a} \times \mathbf{b}), \\ \gamma &= \rho^2 - \mathbf{a}^2 - \mathbf{b}^2 - 2\mathbf{r}^T\mathbf{a}\mathbf{r}^T\mathbf{b}. \end{aligned}$$

Die Substitution $x = \cos t$ liefert eine quadratische Gleichung

$$\begin{aligned} f(x) &= f_2x^2 + f_1x + f_0 \quad \text{mit} \\ f_0 &= \gamma^2 - \beta^2, \\ f_1 &= 2\alpha\gamma, \\ f_2 &= \alpha^2 + \beta^2. \end{aligned}$$

2.1.3.5 Rotation Gerade – Punkt

$$|\mathbf{a} \times \mathbf{b} + (\mathbf{b} - \mathbf{a}) \times \mathbf{Rc}| \;=\; \rho|\mathbf{b} - \mathbf{a}|$$

führt zu einer Gleichung der Form

$$\alpha_1 \cos t + \beta_1 \sin t + \gamma_1 - (\alpha_2 \cos t + \beta_2 \sin t + \gamma_2)^2 + \delta = 0.$$

Mit Hilfe von Gleichung (A.10) können wir die Koeffizienten berechnen:

$$\begin{aligned} \alpha_1 &= 2\mathbf{c}^T((\mathbf{a} \times \mathbf{b}) \times (\mathbf{b} - \mathbf{a})) - \gamma_1 \\ \beta_1 &= 2\mathbf{r}^T(\mathbf{c} \times ((\mathbf{a} \times \mathbf{b}) \times (\mathbf{b} - \mathbf{a}))) \\ \gamma_1 &= 2\mathbf{r}^T\mathbf{c}\mathbf{r}^T((\mathbf{a} \times \mathbf{b}) \times (\mathbf{b} - \mathbf{a})) \\ \alpha_2 &= (\mathbf{b} - \mathbf{a})^T\mathbf{c} - \gamma_2 \\ \beta_2 &= \mathbf{r}^T(\mathbf{c} \times (\mathbf{b} - \mathbf{a})) \\ \gamma_2 &= \mathbf{r}^T\mathbf{c}\mathbf{r}^T(\mathbf{b} - \mathbf{a}) \\ \delta &= (\mathbf{a} \times \mathbf{b})^2 + (\mathbf{c}^2 - \rho^2)(\mathbf{b} - \mathbf{a})^2 \end{aligned}$$

Wir substituieren $x = \cos t$ und $y = \sin t$ und eliminieren y mittels der Beziehung $y = \pm\sqrt{1 - x^2}$. Die entstehende Wurzelgleichung können wir durch einmaliges Quadrieren in ein

Polynom vierten Grades überführen. Die reellen Nullstellen dieses Polynoms $f(x)$ entsprechen den Cosinus–Werten der Drehwinkel, bei denen der Punkt $\mathbf{c}$ von der Gerade L_{ab} gerade den Abstand ρ hat.

$$\begin{aligned}
f(x) &= f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0 \quad \text{mit} \\
f_0 &= -4\beta_1\beta_2\gamma_2 - \beta_2^4 + \beta_1^2 + 2\beta_2^2\gamma_2^2 - \gamma_1^2 + 2\gamma_1\gamma_2^2 \\
&\quad -2\gamma_1\delta + 2\gamma_2^2\delta - \delta^2 + 2\gamma_1\beta_2^2 + 2\beta_2^2\delta - \gamma_2^4 \\
f_1 &= 2\alpha_1\gamma_2^2 - 2\alpha_1\delta - 4\alpha_2\gamma_2^3 - 2\alpha_1\gamma_1 + 2\alpha_1\beta_2^2 \\
&\quad +4\alpha_2\beta_2^2\gamma_2 - 4\beta_1\alpha_2\beta_2 + 4\gamma_1\alpha_2\gamma_2 + 4\alpha_2\gamma_2\delta \\
f_2 &= 2\alpha_2^2\beta_2^2 + 2\alpha_2^2\delta + 2\gamma_1\alpha_2^2 - 6\alpha_2^2\gamma_2^2 - 2\beta_2^2\gamma_2^2 - 2\gamma_1\beta_2^2 \\
&\quad -\alpha_1^2 - \beta_1^2 + 2\beta_2^4 + 4\alpha_1\alpha_2\gamma_2 - 2\beta_2^2\delta + 4\beta_1\beta_2\gamma_2 \\
f_3 &= 2\alpha_1\alpha_2^2 - 4\alpha_2^3\gamma_2 - 2\alpha_1\beta_2^2 + 4\beta_1\alpha_2\beta_2 - 4\alpha_2\beta_2^2\gamma_2 \\
f_4 &= -\alpha_2^4 - \beta_2^4 - 2\alpha_2^2\beta_2^2
\end{aligned}$$

2.1.3.6 Rotation Gerade – Gerade

$$\left|\det\begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{Ra} & \mathbf{Rb} & \mathbf{c} & \mathbf{d} \end{bmatrix}\right| = \rho\,|\mathbf{R}(\mathbf{b}-\mathbf{a}) \times (\mathbf{d}-\mathbf{c})|$$

führt zu einer Gleichung der Form

$$(\alpha_1\cos t + \beta_1\sin t + \gamma_1)^2 + \rho^2\,(\alpha_2\cos t + \beta_2\sin t + \gamma_2)^2 - \rho^2\delta = 0.$$

Wiederum hilft uns Gleichung (A.10) bei der Berechnung der Koeffizienten:

$$\begin{aligned}
\alpha_1 &= (\mathbf{d}-\mathbf{c})^T(\mathbf{a}\times\mathbf{b}) + (\mathbf{b}-\mathbf{a})^T(\mathbf{c}\times\mathbf{d}) - \gamma_1 \\
\beta_1 &= \mathbf{r}^T((\mathbf{b}-\mathbf{a})\times(\mathbf{c}\times\mathbf{d}) - (\mathbf{d}-\mathbf{c})\times(\mathbf{a}\times\mathbf{b})) \\
\gamma_1 &= \mathbf{r}^T(\mathbf{d}-\mathbf{c})\mathbf{r}^T(\mathbf{a}\times\mathbf{b}) + \mathbf{r}^T(\mathbf{b}-\mathbf{a})\mathbf{r}^T(\mathbf{c}\times\mathbf{d}) \\
\alpha_2 &= (\mathbf{d}-\mathbf{c})^T(\mathbf{b}-\mathbf{a}) - \gamma_2 \\
\beta_2 &= \mathbf{r}^T((\mathbf{b}-\mathbf{a})\times(\mathbf{d}-\mathbf{c})) \\
\gamma_2 &= \mathbf{r}^T(\mathbf{b}-\mathbf{a})\mathbf{r}^T(\mathbf{d}-\mathbf{c}) \\
\delta &= (\mathbf{b}-\mathbf{a})^2(\mathbf{d}-\mathbf{c})^2
\end{aligned}$$

Auch hier führt uns die Substitution $x = \cos t$, $y = \sin t$ zu einer Gleichung $f(x)$ vierten Grades. Die reellen Nullstellen von $f(x)$ entsprechen den Cosinus–Werten der Drehwinkel, bei denen die Geraden L_{ab} und L_{cd} gerade den Abstand ρ haben.

$$\begin{aligned}
f(x) &= f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0 \quad \text{mit} \\
f_0 &= -2\beta_1^2\rho^2\gamma_2^2 - 2\gamma_1^2\rho^2\beta_2^2 + 2\beta_1^2\rho^2\delta + 2\rho^4\delta\beta_2^2 + 2\rho^4\beta_2^2\gamma_2^2 + 2\gamma_1^2\rho^2\delta \\
&\quad +2\beta_1^2\gamma_1^2 - \rho^4\delta^2 - \rho^4\gamma_2^4 - \beta_1^4 + 8\beta_1\gamma_1\rho^2\beta_2\gamma_2 \\
&\quad -\gamma_1^4 - \rho^4\beta_2^4 - 2\gamma_1^2\rho^2\gamma_2^2 + 2\rho^4\delta\gamma_2^2 - 2\beta_1^2\rho^2\beta_2^2 \\
f_1 &= 4\alpha_1\beta_1^2\gamma_1 - 4\alpha_1\gamma_1^3 + 8\alpha_1\beta_1\rho^2\beta_2\gamma_2 + 8\beta_1\gamma_1\rho^2\alpha_2\beta_2
\end{aligned}$$

$$
\begin{aligned}
& +4\rho^4\alpha_2\beta_2^2\gamma_2 + 4\alpha_1\gamma_1\rho^2\delta - 4\alpha_1\gamma_1\rho^2\gamma_2^2 - 4\gamma_1^2\rho^2\alpha_2\gamma_2 \\
& +4\rho^4\delta\alpha_2\gamma_2 - 4\rho^4\alpha_2\gamma_2^3 - 4\beta_1^2\rho^2\alpha_2\gamma_2 - 4\alpha_1\gamma_1\rho^2\beta_2^2 \\
f_2 \;=\; & 2\rho^4\beta_2^4 - 2\beta_1^2\gamma_1^2 + 2\alpha_1^2\beta_1^2 - 6\alpha_1^2\gamma_1^2 + 2\beta_1^4 + 8\alpha_1\beta_1\rho^2\alpha_2\beta_2 \\
& +2\rho^4\alpha_2^2\beta_2^2 + 2\alpha_1^2\rho^2\delta - 2\alpha_1^2\rho^2\gamma_2^2 - 8\alpha_1\gamma_1\rho^2\alpha_2\gamma_2 - 2\gamma_1^2\rho^2\alpha_2^2 \\
& +2\rho^4\delta\alpha_2^2 - 6\rho^4\alpha_2^2\gamma_2^2 - 2\alpha_1^2\rho^2\beta_2^2 - 2\beta_1^2\rho^2\alpha_2^2 - 8\beta_1\gamma_1\rho^2\beta_2\gamma_2 \\
& -2\rho^4\beta_2^2\gamma_2^2 - 2\beta_1^2\rho^2\delta + 2\beta_1^2\rho^2\gamma_2^2 + 2\gamma_1^2\rho^2\beta_2^2 - 2\rho^4\delta\beta_2^2 + 4\beta_1^2\rho^2\beta_2^2 \\
f_3 \;=\; & -4\alpha_1^3\gamma_1 - 4\alpha_1^2\rho^2\alpha_2\gamma_2 - 4\alpha_1\gamma_1\rho^2\alpha_2^2 - 4\rho^4\alpha_2^3\gamma_2 - 4\alpha_1\beta_1^2\gamma_1 \\
& -8\alpha_1\beta_1\rho^2\beta_2\gamma_2 - 8\beta_1\gamma_1\rho^2\alpha_2\beta_2 - 4\rho^4\alpha_2\beta_2^2\gamma_2 + 4\alpha_1\gamma_1\rho^2\beta_2^2 + 4\beta_1^2\rho^2\alpha_2\gamma_2 \\
f_4 \;=\; & -\rho^4\beta_2^4 - \alpha_1^4 - \beta_1^4 - 2\alpha_1^2\rho^2\alpha_2^2 - \rho^4\alpha_2^4 - 2\alpha_1^2\beta_1^2 - 8\alpha_1\beta_1\rho^2\alpha_2\beta_2 \\
& -2\rho^4\alpha_2^2\beta_2^2 + 2\alpha_1^2\rho^2\beta_2^2 + 2\beta_1^2\rho^2\alpha_2^2 - 2\beta_1^2\rho^2\beta_2^2
\end{aligned}
$$

2.2 Berechnung von Sicherheitsabständen

2.2.1 Sicherheitsabstand zwischen einem Punkt und einer Fläche

Wir möchten $\Delta^*_{min}(\mathbf{p}, f)$ ($* = trans$ oder $* = rot$) bestimmen. Bei der Translation eines Punktes $\mathbf{p}$ in Richtung $\mathbf{s}$ oder bei der Rotation um eine durch den Ursprung verlaufende Achse $\mathbf{r}$ bewegt sich der Punkt $\mathbf{p}$ auf einem Liniensegment l bzw. auf einem Kreissegment c. Deshalb interessiert uns der minimale Abstand von l und c zu der Fläche f. Aufgrund der Ergebnisse aus Abschnitt 2.1 stellt die Berechnung von $\delta(l, f)$ keine Schwierigkeit dar. Deshalb wollen wir uns im folgenden mit der Berechnung von $\delta(c, f)$ beschäftigen. Um die Rechnungen zu vereinfachen, nehmen wir an, daß c ein Teil eines Kreises C um den Ursprung ist, der in der x_1x_2–Ebene verläuft. Um dies zu erreichen, führen wir folgende Koordinatentransformation durch. Wir wählen einen neuen Ursprung $\mathbf{e_0}'$ und neue zueinander orthogonale und normierte Basisvektoren, die ein Rechtssystem bilden:

$$\mathbf{e_3}' := \mathbf{r}, \quad \mathbf{e_2}' := \frac{\mathbf{r}\times\mathbf{p}}{|\mathbf{r}\times\mathbf{p}|}, \quad \mathbf{e_1}' := \frac{\mathbf{r}\times(\mathbf{p}\times\mathbf{r})}{|\mathbf{r}\times\mathbf{p}|}, \quad \mathbf{e_0}' := \mathbf{r}\mathbf{r}^T\mathbf{p}.$$

Gemäß Gleichung (A.1) ergibt sich für die Koordinaten eines transformierten Punktes $\mathbf{x}$:

$$\mathbf{x}' = \frac{1}{|\mathbf{r}\times\mathbf{p}|}((\mathbf{r}\times\mathbf{p})^T(\mathbf{r}\times\mathbf{x}), \mathbf{x}^T(\mathbf{r}\times\mathbf{p}), \mathbf{r}^T(\mathbf{x}-\mathbf{p})\,|\mathbf{r}\times\mathbf{p}|)^T$$

Insbesondere gilt $\mathbf{p}' = (|\mathbf{r}\times\mathbf{p}|, 0, 0)^T$.
Die Kreisbahn C des Punktes $\mathbf{p}$ hat den Radius $\rho := |\mathbf{r}\times\mathbf{p}|$. Ab jetzt nehmen wir an, daß das Kreissegment $c \subseteq C$ als Teil des Kreises C in folgender Form vorliegt:

$$
\begin{aligned}
c \;&=\; \{\mathbf{x}(t) = \rho\,(\cos t, \sin t, 0)^T \,|\, t \in [0, T]\}, \\
C \;&=\; \{\mathbf{x} \,|\, x_1^2 + x_2^2 = \rho^2, x_3 = 0\}.
\end{aligned}
$$

Die beiden Endpunkte des Kreissegments heißen demnach $\mathbf{x}(0) \equiv \mathbf{p}'$ und $\mathbf{x}(T)$.

2.2.1.1 Distanz zwischen Kreissegment und Punkt

Der minimale Abstand $\delta(C, \mathbf{a})$ eines Punktes $\mathbf{a}$ von der Kreislinie C läßt sich direkt angeben. $\overline{\mathbf{a}} = (a_1, a_2, 0)^T$ bezeichne die Projektion des Punktes $\mathbf{a}$ in die x_1x_2–Ebene.

$$\delta^2(C, \mathbf{a}) \;=\; a_3^2 + (|\overline{\mathbf{a}}| - \rho)^2 \tag{2.12}$$

Falls $\overline{\mathbf{a}} \neq \mathbf{0}$ gilt, ist der zu $\mathbf{a}$ nächstgelegene Punkt $\tilde{\mathbf{a}}$ auf C eindeutig bestimmt. Wenn $\tilde{\mathbf{a}} = \rho\frac{\overline{\mathbf{a}}}{|\overline{\mathbf{a}}|} \notin c$, ist $\delta(c, \mathbf{a})$ bestimmt durch den minimalen Abstand der Randpunkte von c zu $\mathbf{a}$. D.h. Gleichung (2.12) spiegelt nur dann den richtigen Abstand zwischen $\mathbf{a}$ und c wieder, wenn $\arctan(a_2, a_1) < T$ erfüllt ist.

$$\delta(c, \mathbf{a}) = \begin{cases} \delta(C, \mathbf{a}) & \text{falls} \quad \arctan(a_2, a_1) < T \\ \min\{\delta(\mathbf{a}, \mathbf{x}(0)), \delta(\mathbf{a}, \mathbf{x}(T))\} & \text{sonst} \end{cases}$$

2.2.1.2 Distanz zwischen Kreissegment und Liniensegment

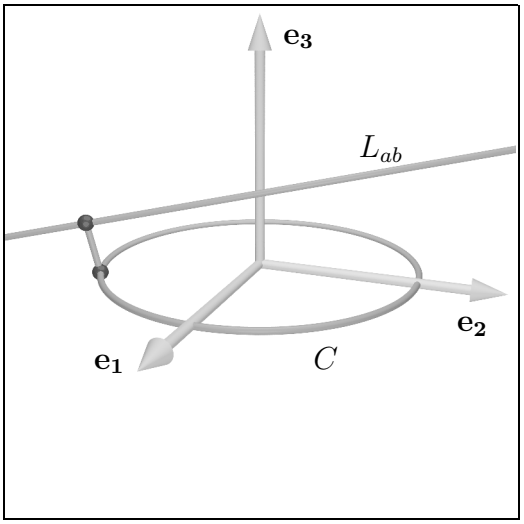


Abbildung 2.2: Distanz zwischen dem Kreis C und der Gerade L_{ab}

Zunächst betrachten wir einen Vollkreis C und eine Gerade $L_{ab} = \{\mathbf{a} + \mu(\mathbf{b} - \mathbf{a}) \,|\, \mu \in \mathbb{R}\}$. Wir berechnen die kürzeste Entfernung $\delta(C, L_{ab})$ auf der Grundlage von Formel (2.12). Dabei benutzen wir als Abkürzung $\mathbf{u} := \mathbf{b} - \mathbf{a}$.

$$\begin{aligned} \delta^2(\mu) &= (a_3 + \mu u_3)^2 + (\sqrt{(\overline{\mathbf{a}} + \mu\overline{\mathbf{u}})^2} - \rho)^2 \\ \frac{d\delta^2}{d\mu} &= 2u_3(a_3 + \mu u_3) + 2\frac{\overline{\mathbf{u}}^T(\overline{\mathbf{a}} + \mu\overline{\mathbf{u}})}{\sqrt{(\overline{\mathbf{a}} + \mu\overline{\mathbf{u}})^2}}(\sqrt{(\overline{\mathbf{a}} + \mu\overline{\mathbf{u}})^2} - \rho) \\ &= 2\mathbf{u}^T(\mathbf{a} + \mu\mathbf{u}) - 2\rho\frac{\overline{\mathbf{u}}^T(\overline{\mathbf{a}} + \mu\overline{\mathbf{u}})}{\sqrt{(\overline{\mathbf{a}} + \mu\overline{\mathbf{u}})^2}} \end{aligned}$$

Um das Minimum zu finden, setzen wir $\frac{d\delta^2}{d\mu} = 0$. Nach Quadrieren zum Beseitigen der Wurzel führt dies zu folgender Gleichung:

$$(\mathbf{u}^T(\mathbf{a} + \mu\mathbf{u}))^2(\overline{\mathbf{a}} + \mu\overline{\mathbf{u}})^2 = (\rho\overline{\mathbf{u}}^T(\overline{\mathbf{a}} + \mu\overline{\mathbf{u}}))^2$$

Nach Ausmultiplizieren erhält man eine Gleichung vierten Grades in μ:

$$\begin{aligned}
f(\mu) &= f_4\mu^4 + f_3\mu^3 + f_2\mu^2 + f_1\mu + f_0 = 0 \quad \text{mit} \\
f_4 &= \mathbf{u}^4\overline{\mathbf{u}}^2, \\
f_3 &= 2\mathbf{u}^2\{\mathbf{u}^2(\overline{\mathbf{a}}^T\overline{\mathbf{u}}) + \overline{\mathbf{u}}^2(\mathbf{a}^T\mathbf{u})\}, \\
f_2 &= \mathbf{u}^4\overline{\mathbf{a}}^2 + \overline{\mathbf{u}}^2(\mathbf{a}^T\mathbf{u})^2 + 4\mathbf{u}^2(\mathbf{a}^T\mathbf{u})(\overline{\mathbf{a}}^T\overline{\mathbf{u}}) - \rho^2\overline{\mathbf{u}}^4, \\
f_1 &= 2(\mathbf{a}^T\mathbf{u})\{\mathbf{u}^2\overline{\mathbf{a}}^2 + (\mathbf{a}^T\mathbf{u})(\overline{\mathbf{a}}^T\overline{\mathbf{u}})\} - 2\rho^2\overline{\mathbf{u}}^2(\overline{\mathbf{a}}^T\overline{\mathbf{u}}), \\
f_0 &= \overline{\mathbf{a}}^2(\mathbf{a}^T\mathbf{u})^2 - \rho^2(\overline{\mathbf{a}}^T\overline{\mathbf{u}})^2.
\end{aligned}$$

$\delta(C, L_{ab})$ erhält man aus den Werten $\delta(\mu_i)$ für die reellen Nullstellen μ_i von $f(\mu)$:

$$\delta(C, L_{ab}) = \min\{\delta(\mu_i) \,|\, f(\mu_i) = 0,\ \mu_i \in \mathbb{R}\}$$

Wir gehen nun zu den Segmenten über. Das betrachtete Liniensegment sei gegeben durch $l_{ab} = \{\mathbf{y}(\mu) = \mathbf{a} + \mu(\mathbf{b} - \mathbf{a}) \,|\, \mu \in [0, 1]\}$ und das Kreissegment wie zuvor durch $c = \{\mathbf{x}(t) = \rho(\cos t, \sin t, 0)^T \,|\, t \in [0, T]\}$. Entweder ist einer der Randpunkte von c oder von l_{ab} ein Endpunkt der kürzesten Verbindungsstrecke zwischen beiden Segmenten, oder sie verläuft zwischen zwei inneren Punkten. Letzteres ist nur möglich, falls reelle Nullstellen μ_i von $f(\mu)$ im Intervall $[0, 1]$ existieren mit $\arctan(y_2(\mu_i), y_1(\mu_i)) < T$:

$$\begin{aligned}
\delta(c, l_{ab}) = \min\big(&\{\delta(\mu_i) \,|\, f(\mu_i) = 0,\ \mu_i \in [0, 1],\ \arctan(y_2(\mu_i), y_1(\mu_i)) < T\} \\
&\cup \{\delta(\mathbf{a}, c), \delta(\mathbf{b}, c), \delta(\mathbf{x}(0), l_{ab}), \delta(\mathbf{x}(T), l_{ab})\}\big)
\end{aligned}$$

2.2.1.3 Distanz zwischen Kreissegment und Polygon

Zuerst berechnen wir den minimalen Abstand des Kreises C von einer Ebene $H: \mathbf{n}^T\mathbf{x} = n_0$ mit $|\mathbf{n}| = 1$. H enthalte das gegebene Polygon f.
Wir unterscheiden zwei Situationen, je nachdem ob die Ebene H den Kreis C schneidet oder nicht. Falls $C \cap H \neq \emptyset$, gilt $\delta(C, H) = 0$ und es gibt im allgemeinen zwei Schnittpunkte $\mathbf{q_{1,2}}$, die sich leicht durch Einsetzen der linearen Ebenengleichung in die quadratische Bestimmungsgleichung des Kreises berechnen lassen:

$$\mathbf{q_{1,2}} = \frac{n_0}{\overline{\mathbf{n}}^2}\begin{bmatrix} n_1 \\ n_2 \\ 0 \end{bmatrix} \pm \frac{\sqrt{\rho^2\overline{\mathbf{n}}^2 - n_0^2}}{\overline{\mathbf{n}}^2}\begin{bmatrix} n_2 \\ -n_1 \\ 0 \end{bmatrix}$$

Anhand dieser Formel erkennt man bereits, daß nur dann ein Schnitt vorliegen kann, wenn $|n_0| \leq \rho|\overline{\mathbf{n}}|$ gilt. Entartung tritt auf, wenn H parallel zur x_1x_2–Ebene verläuft.
Zur Analyse der zweiten möglichen Situation $C \cap H = \emptyset$ setzen wir $\mathbf{n} \neq \mathbf{e_3}$ voraus, denn dann gibt es auf dem Kreis C einen eindeutig bestimmten Punkt $\mathbf{q_0} \in C$, der der Ebene H am nächsten liegt. Zur Bestimmung dieses Punktes arbeiten wir mit der Lagrangefunktion

$\mathrm{Lag}(\mathbf{x})$, die den Abstand eines Punktes $\mathbf{x}$ von der Ebene H repräsentiert. Der Lagrangeanteil zwingt den Punkt $\mathbf{x} = (x_1, x_2, 0)^T$ auf die Kreislinie C.

$$\mathrm{Lag}(\mathbf{x}) \;=\; \mathbf{n}^T\mathbf{x} - n_0 + \lambda(x_1^2 + x_2^2 - \rho^2), \qquad x_3 = 0$$

$$\frac{\partial \mathrm{Lag}}{\partial x_1} = n_1 + 2\lambda x_1, \qquad \frac{\partial \mathrm{Lag}}{\partial x_2} = n_2 + 2\lambda x_2, \qquad \frac{\partial \mathrm{Lag}}{\partial \lambda} = x_1^2 + x_2^2 - \rho^2$$

$\frac{\partial \mathrm{Lag}}{\partial x_1} = \frac{\partial \mathrm{Lag}}{\partial x_2} = \frac{\partial \mathrm{Lag}}{\partial \lambda} = 0$ führt über eine quadratische Gleichung zu der Lösung

$$\mathbf{q_0} \;=\; \frac{\mathrm{sgn}(n_0)\rho}{|\overline{\mathbf{n}}|} \begin{bmatrix} n_1 \\ n_2 \\ 0 \end{bmatrix}, \qquad |\mathrm{Lag}(\mathbf{q_0})| \;=\; |n_0| - \rho|\overline{\mathbf{n}}|.$$

Betrachten wir anstelle des Kreises C wieder das Kreissegment c, so liegt ein Punkt $\mathbf{q} \in C$ nur dann auf c selbst, wenn $\arctan(q_2, q_1) < T$ gilt. Der zu $\mathbf{q}$ nächstgelegene Punkt $\hat{\mathbf{q}}$ auf der Ebene H muß stets darauf getestet werden, ob er auf f liegt, denn wir haben es nicht mit einer ganzen Ebene, sondern nur mit einem Polygon zu tun. In dem nun folgenden Programmfragment sind alle Sonderfälle berücksichtigt, die bei der Berechnung von $\delta \equiv \delta(c, f)$ auftreten können. Dazu gehören natürlich noch all die Fälle, in denen ein Randpunkt von c oder f ein Endpunkt der kürzesten Verbindungsstrecke zwischen c und f ist.

$\delta \leftarrow \infty$;
if $(\mathbf{n} \neq \mathbf{e_3})$ { d.h. H nicht parallel zur x_1x_2–Ebene } **then**
  if $(|n_0| > \rho|\overline{\mathbf{n}}|)$ { d.h. $H \cap C = \emptyset$ } **then**
    $\mathbf{q} \leftarrow \mathbf{q_0}$; $\hat{\mathbf{q}} \leftarrow \mathbf{q} + (n_0 - \mathbf{n}^T\mathbf{q})\mathbf{n}$;
    if $(\mathbf{q} \in c)$ **and** $(\hat{\mathbf{q}} \in f)$ **then** $\delta \leftarrow |n_0| - \rho|\overline{\mathbf{n}}|$ **fi**
  else { d.h. $H \cap C \neq \emptyset$ }
    for $i = 1$ **to** 2 **do**
      $\mathbf{q} \leftarrow \mathbf{q_i}$;
      if $(\mathbf{q} \in c)$ **and** $(\mathbf{q} \in f)$ **then** $\delta \leftarrow 0$ **fi**
    od
  fi
fi;
$\mathbf{q} \leftarrow \mathbf{x}(0)$; $\hat{\mathbf{q}} \leftarrow \mathbf{q} + (n_0 - \mathbf{n}^T\mathbf{q})\mathbf{n}$;
if $(\hat{\mathbf{q}} \in f)$ **then** $\delta \leftarrow \min\{\delta, |\mathbf{n}^T\mathbf{q} - n_0|\}$ **fi**;
$\mathbf{q} \leftarrow \mathbf{x}(T)$; $\hat{\mathbf{q}} \leftarrow \mathbf{q} + (n_0 - \mathbf{n}^T\mathbf{q})\mathbf{n}$;
if $(\hat{\mathbf{q}} \in f)$ **then** $\delta \leftarrow \min\{\delta, |\mathbf{n}^T\mathbf{q} - n_0|\}$ **fi**;
for $i = 0$ **to** k **do**
  $\mathbf{a} \leftarrow \mathbf{v_i}$; $\mathbf{b} \leftarrow \mathbf{v_{i+1}}$;
  $\delta \leftarrow \min\{\delta, \delta(c, l_{ab})\}$
od

2.2.2 Sicherheitsabstand zwischen zwei Liniensegmenten

Gegeben seien zwei Liniensegmente mit den Endpunkten **a** und **b** bzw. **c** und **d**:

$$\begin{aligned} l_{ab} &= \{\mathbf{a} + \mu(\mathbf{b} - \mathbf{a}) \,|\, 0 \leq \mu \leq 1\} \\ l_{cd} &= \{\mathbf{c} + \nu(\mathbf{d} - \mathbf{c}) \,|\, 0 \leq \nu \leq 1\} \end{aligned}$$

Die zugehörigen Geraden bezeichnen wir wieder mit L_{ab} und L_{cd}. Das Liniensegment l_{ab} bewege sich translatorisch oder rotatorisch. Wir möchten folgende Minima berechnen:

$$\begin{aligned} \Delta^*_{min}(l_{ab}, l_{cd}) &= \min\{\delta(l^*_{ab}(t), l_{cd}) \,|\, t \in [0, T]\}, \\ &\quad \text{wobei } * = trans \text{ oder } * = rot \end{aligned}$$

Wir gehen in zwei Stufen vor. Zunächst berechnen wir die Fläche f, die das bewegte Liniensegment $l_{ab}(t)$ überstreicht. Dann bestimmen wir den minimalen euklidischen Abstand dieser Fläche zu dem stationären Liniensegment l_{cd}. Bei einer Translation ist f ein Parallelogramm, und die Aufgabe reduziert sich auf die Abstandsbestimmung zwischen einem ebenen Polygon und einem Liniensegment. Ein rotierendes Liniensegment beschreibt jedoch eine gekrümmte Fläche. Um diesen Fall genauer zu untersuchen, setzen wir anstelle der Liniensegmente vorerst die zugehörigen Geraden ein und lassen die rotierende Gerade eine Volldrehung ausführen. Wir suchen also folgenden Abstand:

$$\min_{0 \leq t < 2\pi} \delta(L^{rot}_{ab}(t), L_{cd})$$

2.2.2.1 Durch rotierende Geraden erzeugte quadratische Flächen

Wir möchten die von der Geraden $L_{ab} = \{\mathbf{a} + \mu(\mathbf{b} - \mathbf{a}) \,|\, \mu \in \mathbb{R}\}$ überstrichene Fläche charakterisieren, während L_{ab} vollständig um eine durch den Ursprung verlaufende Achse **r** rotiert.

Der Abstand d eines Punktes **x** von dieser Achse bleibt während der Rotation von L_{ab} invariant, ebenso wie seine Höhe h gemessen in Richtung **r**:

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{r}^T\mathbf{x} \\ d^2(\mathbf{x}) &= (\mathbf{x} - (\mathbf{r}^T\mathbf{x})\mathbf{r})^2 \end{aligned}$$

Da $h = \mathbf{r}^T\mathbf{a} + \mu\mathbf{r}^T(\mathbf{b} - \mathbf{a})$ ist, läßt sich μ eliminieren. Im folgenden setzen wir voraus, daß $\mathbf{r}^T(\mathbf{b} - \mathbf{a}) \neq 0$ gilt, d.h. $\mathbf{r} \not\perp \mathbf{b} - \mathbf{a}$.

$$\begin{aligned} d^2(h) &= \left(\mathbf{a} + \frac{h - \mathbf{r}^T\mathbf{a}}{\mathbf{r}^T(\mathbf{b} - \mathbf{a})}(\mathbf{b} - \mathbf{a}) - h\mathbf{r}\right)^2 \\ &= \left(\frac{(\mathbf{r}^T\mathbf{b})\mathbf{a} - (\mathbf{r}^T\mathbf{a})\mathbf{b} + h(\mathbf{b} - \mathbf{a})}{\mathbf{r}^T(\mathbf{b} - \mathbf{a})} - h\mathbf{r}\right)^2 \\ &= \left(\frac{\mathbf{r} \times (\mathbf{a} \times \mathbf{b}) + h(\mathbf{b} - \mathbf{a} - (\mathbf{r}^T(\mathbf{b} - \mathbf{a}))\mathbf{r})}{\mathbf{r}^T(\mathbf{b} - \mathbf{a})}\right)^2 \end{aligned}$$

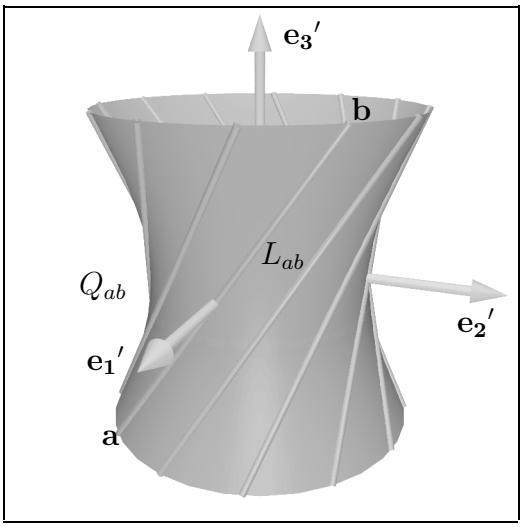


Abbildung 2.3: Das von der Gerade L_{ab} erzeugte Hyperboloid Q_{ab}

Hieraus folgt:

$$\begin{aligned} d^2(h) &= \alpha h^2 + \beta h + \gamma \quad \text{mit} \\ \alpha &= \frac{(\mathbf{r} \times (\mathbf{b}-\mathbf{a}))^2}{(\mathbf{r}^T(\mathbf{b}-\mathbf{a}))^2}, \\ \beta &= \frac{2(\mathbf{b}-\mathbf{a})^T(\mathbf{r} \times (\mathbf{a} \times \mathbf{b}))}{(\mathbf{r}^T(\mathbf{b}-\mathbf{a}))^2}, \\ \gamma &= \frac{(\mathbf{r} \times (\mathbf{a} \times \mathbf{b}))^2}{(\mathbf{r}^T(\mathbf{b}-\mathbf{a}))^2}. \end{aligned} \tag{2.13}$$

Die Fläche, die durch Gleichung (2.13) beschrieben wird, ist offensichtlich eine quadratische Form. Eine ihrer Hauptachsenrichtungen ist die Rotationsachse $\mathbf{r}$. Wir definieren ein neues Koordinatensystem E' mit den Basisvektoren $(\mathbf{e_1}', \mathbf{e_2}', \mathbf{e_3}')$ in einem geeigneten Ursprung $\mathbf{e_0}'$, so daß wir eine Hauptachsenform unserer quadratischen Fläche erhalten. Wir wählen $\mathbf{e_3}' := \mathbf{r}$ und ergänzen die Basis durch $\mathbf{e_1}'$ und $\mathbf{e_2}'$ zu einem Orthonormalsystem. Den neuen Ursprung

$$\mathbf{e_0}' = -\frac{\beta}{2\alpha}\,\mathbf{r} = \frac{(\mathbf{a}-\mathbf{b})^T(\mathbf{r} \times (\mathbf{a} \times \mathbf{b}))}{(\mathbf{r} \times (\mathbf{a}-\mathbf{b}))^2}\,\mathbf{r}$$

finden wir aus der Scheitelpunktsform der parabelförmigen Abstandsfunktion

$$d^2(h) = \alpha h^2 + \beta h + \gamma = \alpha\Big(h + \frac{\beta}{2\alpha}\Big)^2 + \gamma - \frac{\beta^2}{4\alpha}.$$

Hierbei wurde $\alpha \neq 0$ vorausgesetzt, d.h. $\mathbf{r} \nparallel \mathbf{b}-\mathbf{a}$. Mit $x_3' = h + \beta/2\alpha$ und $d^2 = x_1'^2 + x_2'^2$ ergibt sich dann:

$$q_0 = q_1 x_1'^2 + q_2 x_2'^2 + q_3 x_3'^2$$

$$
\begin{aligned}
q_1 &= q_2 = 1 \\
q_3 &= -\alpha = -\frac{(\mathbf{r} \times (\mathbf{b}-\mathbf{a}))^2}{(\mathbf{r}^T(\mathbf{b}-\mathbf{a}))^2} \\
q_0 &= \gamma - \frac{\beta^2}{4\alpha} = \frac{(\mathbf{r}^T(\mathbf{a} \times \mathbf{b}))^2}{(\mathbf{r} \times (\mathbf{b}-\mathbf{a}))^2}
\end{aligned}
$$

Da $q_0, q_1, q_2 \geq 0$ und $q_3 \leq 0$ sind, handelt es sich im allgemeinen Fall um ein Hyperboloid (siehe Abbildung 2.3). Falls $\mathbf{r}$, $\mathbf{a}$ und $\mathbf{b}$ in einer Ebene liegen, gilt $q_0 = 0$ und wir erhalten einen Kegel. Wenn zusätzlich $\mathbf{r} \parallel \mathbf{b}-\mathbf{a}$ gilt, dann ist das Ergebnis ein Zylinder, da $\alpha = \beta = 0$. In diesem Fall ergibt sich mit $x_3' = h$:

$$
\begin{aligned}
q_1 &= q_2 = 1 \\
q_3 &= 0 \\
q_0 &= \gamma = \mathbf{a}^2 - (\mathbf{r}^T\mathbf{a})^2
\end{aligned}
$$

Die quadratische Fläche, die durch die Rotation von L_{ab} erzeugt wird, nennen wir Q_{ab}. Wir haben gezeigt, daß sich Q_{ab} in dem neu gewählten Koordinatensystem $E' = (\mathbf{e_0}', \mathbf{e_1}', \mathbf{e_2}', \mathbf{e_3}')$ durch eine Gleichung der Form $Q(\mathbf{x}') = q_1 x_1'^2 + q_2 x_2'^2 + q_3 x_3'^2 - q_0 = 0$ beschreiben läßt. In Vektor–Matrixschreibweise sieht diese Hauptachsenform wie folgt aus:

$$
Q(\mathbf{x}') = \mathbf{x}'^T\,\mathbf{Q}\,\mathbf{x}' - q_0 \quad \text{mit} \quad \mathbf{Q} = \begin{bmatrix} q_1 & 0 & 0 \\ 0 & q_2 & 0 \\ 0 & 0 & q_3 \end{bmatrix}
$$

Die Berechnung des minimalen Abstandes zwischen der rotierenden Gerade $L_{ab}(t)$ und der stationären Gerade L_{cd} hat sich nun darauf reduziert, den kleinsten Abstand zwischen der quadratischen Fläche Q_{ab} und L_{cd} zu ermitteln:

$$
\min_{0 \leq t < 2\pi} \delta(L_{ab}^{rot}(t), L_{cd}) \;=\; \delta(Q_{ab}, L_{cd})
$$

Wenn $Q_{ab} \cap L_{cd} \neq \emptyset$ gilt, so ist der gesuchte Abstand $\delta(Q_{ab}, L_{cd}) = 0$. Wir betrachten diesen Sonderfall zuerst.

2.2.2.2 Schnitt einer quadratischen Fläche mit einer Gerade

Zuerst ist eine Koordinatentransformation notwendig, um die stationäre Gerade $L_{cd} = \{\mathbf{c}' + \nu(\mathbf{d}' - \mathbf{c}') \,|\, \nu \in \mathbb{R}\}$ im System E' darzustellen. Wir müssen hierzu nur die beiden Punkte $\mathbf{c}$ und $\mathbf{d}$ transformieren.

Wir setzen nun $\mathbf{x}' = \mathbf{c}' - \nu\mathbf{u}$ mit $\mathbf{u} := \mathbf{c}' - \mathbf{d}'$ in $\mathbf{x}'^T\mathbf{Q}\mathbf{x}' = q_0$ ein:

$$
\begin{aligned}
q_0 &= (\mathbf{c}' - \nu\mathbf{u})^T\,\mathbf{Q}\,(\mathbf{c}' - \nu\mathbf{u}) \\
q_0 &= (\mathbf{u}^T\mathbf{Q}\mathbf{u})\nu^2 - 2(\mathbf{c}'^T\mathbf{Q}\mathbf{u})\nu + \mathbf{c}'^T\mathbf{Q}\mathbf{c}' \\
\nu_{1,2} &= \frac{-\mathbf{c}'^T\mathbf{Q}\mathbf{u} \pm \sqrt{q_0\mathbf{u}^T\mathbf{Q}\mathbf{u} + (\mathbf{c}'^T\mathbf{Q}\mathbf{u})^2 - (\mathbf{u}^T\mathbf{Q}\mathbf{u})(\mathbf{c}'^T\mathbf{Q}\mathbf{c}')}}{\mathbf{u}^T\mathbf{Q}\mathbf{u}}
\end{aligned}
$$

Die Formel für ν gilt natürlich nur, falls $\mathbf{u}^T\mathbf{Q}\mathbf{u} \neq 0$ ist. Die Diskussion der entarteten Fälle ersparen wir uns hier. Den Ausdruck unter der Wurzel bringen wir noch in eine Form, die uns im folgenden noch begegnen wird. Als weitere Abkürzung benutzen wir $\mathbf{v} := \mathbf{c}' \times \mathbf{d}'$. *adj* bezeichnet die Adjungierte einer Matrix. Somit ist

$$\nu_{1,2} = \frac{-\mathbf{c}'^T\mathbf{Q}\mathbf{u} \pm \sqrt{q_0\mathbf{u}^T\mathbf{Q}\mathbf{u} - \mathbf{v}^T adj(\mathbf{Q})\mathbf{v}}}{\mathbf{u}^T\mathbf{Q}\mathbf{u}}. \tag{2.14}$$

Hieraus erkennen wir, daß nur dann ein Schnitt vorliegt, wenn $q_0\mathbf{u}^T\mathbf{Q}\mathbf{u} \geq \mathbf{v}^T adj(\mathbf{Q})\mathbf{v}$ gilt. Nun betrachten wir den Fall, daß $Q_{ab} \cap L_{cd} = \emptyset$ gilt.

2.2.2.3 Minimaler Abstand zwischen einer quadratischen Fläche und einer Gerade

Zur Bestimmung von $\delta(Q_{ab}, L_{cd})$ arbeiten wir mit der *Lagrange*–Funktion.

$$\mathrm{Lag}(\mathbf{x}', \nu, \lambda) = (\mathbf{x}' - \mathbf{c}' + \nu\,(\mathbf{c}' - \mathbf{d}'))^2 + \lambda\, Q(\mathbf{x}'),$$

denn unser Ziel ist es, Punkte $\mathbf{x}'$ auf der quadratischen Fläche Q_{ab} zu ermitteln, die zu Punkten $\mathbf{c}' + \nu\,(\mathbf{d}' - \mathbf{c}')$ der Geraden L_{cd} die geringste Entfernung aufweisen. Der erste Summand von f mißt diese Distanz, und der zweite Summand mit dem Lagrangemultiplikator λ sorgt für die Einhaltung der Nebenbedingung $Q(\mathbf{x}') = 0$.

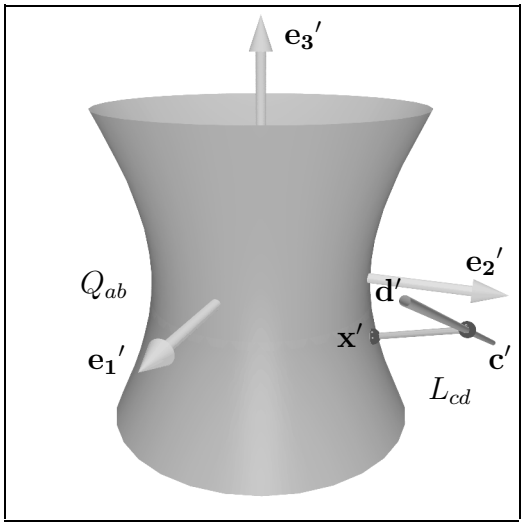


Abbildung 2.4: Distanz zwischen der quadratischen Fläche Q_{ab} und der Gerade L_{cd}

Die Extrema von *Lag* ergeben sich über die partiellen Ableitungen:

$$\frac{\partial \mathrm{Lag}}{\partial \mathbf{x}'} = 2(\mathbf{x}' - \mathbf{c}' + \nu(\mathbf{c}' - \mathbf{d}')) + \lambda grad\, Q(\mathbf{x}') \tag{2.15}$$
$$\text{mit} \quad grad\, Q(\mathbf{x}') = 2\mathbf{Q}\mathbf{x}'$$

$$\frac{\partial \text{Lag}}{\partial \nu} = 2(\mathbf{c}' - \mathbf{d}')^T(\mathbf{x}' - \mathbf{c}' + \nu(\mathbf{c}' - \mathbf{d}')) \tag{2.16}$$

$$\frac{\partial \text{Lag}}{\partial \lambda} = Q(\mathbf{x}') \tag{2.17}$$

Das Gleichungssystem

$$\frac{\partial \text{Lag}}{\partial \mathbf{x}'} = \mathbf{0} \quad \text{und} \quad \frac{\partial \text{Lag}}{\partial \nu} = \frac{\partial \text{Lag}}{\partial \lambda} = 0$$

läßt sich geometrisch leicht interpretieren: Damit die Distanz zwischen einem Punkt $\mathbf{x}'$ auf Q_{ab} (2.17) und einem Punkt $\mathbf{c}' + \nu(\mathbf{d}' - \mathbf{c}')$ auf L_{cd} ein lokales Minimum annimmt, muß die Verbindungsstrecke $\mathbf{x}' - \mathbf{c}' + \nu(\mathbf{c}' - \mathbf{d}')$ senkrecht auf Q_{ab} (2.15) und auf L_{cd} stehen (2.16). Gleichung (2.16) ist linear in ν. Wir lösen nach ν auf und setzen in (2.15) ein:

$$\nu = \frac{(\mathbf{c}' - \mathbf{d}')^T(\mathbf{c}' - \mathbf{x}')}{(\mathbf{c}' - \mathbf{d}')^2} \tag{2.18}$$

$$(\mathbf{x}' - \mathbf{c}' + \frac{(\mathbf{c}' - \mathbf{d}')^T(\mathbf{c}' - \mathbf{x}')}{(\mathbf{c}' - \mathbf{d}')^2}(\mathbf{c}' - \mathbf{d}')) + \lambda \mathbf{Q}\mathbf{x}' = 0$$

$$\iff ((\mathbf{c}' - \mathbf{d}')^2(\lambda \mathbf{Q} + \mathbf{I}) - (\mathbf{c}' - \mathbf{d}')(\mathbf{c}' - \mathbf{d}')^T)\mathbf{x}' = (\mathbf{c}' \times \mathbf{d}') \times (\mathbf{c}' - \mathbf{d}')$$

Um Übersicht zu bewahren, benutzen wir folgende Abkürzungen:

$$\mathbf{u} := \mathbf{c}' - \mathbf{d}', \qquad \mathbf{v} := \mathbf{c}' \times \mathbf{d}', \qquad \mathbf{\Lambda} := \lambda\,\mathbf{Q} + \mathbf{I}.$$

Nach dieser Substitution ergibt sich folgendes lineare Gleichungssystem für $\mathbf{x}'$:

$$(\mathbf{u}^2\,\mathbf{\Lambda} - \mathbf{u}\,\mathbf{u}^T)\,\mathbf{x}' = \mathbf{v} \times \mathbf{u}$$

Zur Bestimmung von $\mathbf{x}' = (x_1', x_2', x_3')^T = (w_1/w_0, w_2/w_0, w_3/w_0)^T$ benutzen wir die Kramer'sche Regel. Die Spalten der Matrix $\mathbf{\Lambda}$ werden mit $\mathbf{\Lambda_i}$ $(i = 1, 2, 3)$ bezeichnet.

$$\begin{aligned}
w_0 &= \det(\mathbf{u}^2\,\mathbf{\Lambda} - \mathbf{u}\,\mathbf{u}^T) \\
w_1 &= \det[\mathbf{v} \times \mathbf{u}, \mathbf{u}^2\,\mathbf{\Lambda_2} - u_2\,\mathbf{u}, \mathbf{u}^2\,\mathbf{\Lambda_3} - u_3\,\mathbf{u}] \\
w_2 &= \det[\mathbf{u}^2\,\mathbf{\Lambda_1} - u_1\,\mathbf{u}, \mathbf{v} \times \mathbf{u}, \mathbf{u}^2\,\mathbf{\Lambda_3} - u_3\,\mathbf{u}] \\
w_3 &= \det[\mathbf{u}^2\,\mathbf{\Lambda_1} - u_1\,\mathbf{u}, \mathbf{u}^2\,\mathbf{\Lambda_2} - u_2\,\mathbf{u}, \mathbf{v} \times \mathbf{u}]
\end{aligned}$$

Wir berechnen zunächst w_0:

$$\begin{aligned}
w_0 &= \det[\mathbf{u}^2\,\mathbf{\Lambda_1} - u_1\,\mathbf{u}, \mathbf{u}^2\,\mathbf{\Lambda_2} - u_2\,\mathbf{u}, \mathbf{u}^2\,\mathbf{\Lambda_3} - u_3\,\mathbf{u}] \\
&= \mathbf{u}^6\,\det(\mathbf{\Lambda}) - \mathbf{u}^4\,u_1\,\det[\mathbf{u}, \mathbf{\Lambda_2}, \mathbf{\Lambda_3}] \\
&\quad -\mathbf{u}^4\,u_2\,\det[\mathbf{\Lambda_1}, \mathbf{u}, \mathbf{\Lambda_3}] - \mathbf{u}^4\,u_3\,\det[\mathbf{\Lambda_1}, \mathbf{\Lambda_2}, \mathbf{u}] \\
&= \mathbf{u}^6\,\Lambda_{11}\Lambda_{22}\Lambda_{33} - \mathbf{u}^4\,u_1^2\Lambda_{22}\Lambda_{33} - \mathbf{u}^4\,u_2^2\Lambda_{33}\Lambda_{11} - \mathbf{u}^4\,u_3^2\Lambda_{11}\Lambda_{22} \\
&= \lambda\,\mathbf{u}^4\{q_1\,u_1^2(1 + \lambda\,q_2)(1 + \lambda\,q_3) \\
&\quad +q_2\,u_2^2(1 + \lambda\,q_3)(1 + \lambda\,q_1) + q_3\,u_3^2(1 + \lambda\,q_1)(1 + \lambda\,q_2)\}
\end{aligned}$$

Nun zu w_1 (w_2 und w_3 werden analog berechnet):

$$\begin{aligned} w_1 &= \det[\mathbf{v}\times\mathbf{u}, \mathbf{u}^2\,\mathbf{\Lambda_2} - u_2\,\mathbf{u}, \mathbf{u}^2\,\mathbf{\Lambda_3} - u_3\,\mathbf{u}] \\ &= (\mathbf{v}\times\mathbf{u})^T(\mathbf{u}^4(\mathbf{\Lambda_2}\times\mathbf{\Lambda_3}) - \mathbf{u}^2u_3(\mathbf{\Lambda_2}\times\mathbf{u}) - \mathbf{u}^2u_2(\mathbf{u}\times\mathbf{\Lambda_3})) \\ &= \mathbf{u}^4\{(\mathbf{v}^T\mathbf{\Lambda_2})(\mathbf{u}^T\mathbf{\Lambda_3}) - (\mathbf{v}^T\mathbf{\Lambda_3})(\mathbf{u}^T\mathbf{\Lambda_2})\} \\ &\quad -\mathbf{u}^2u_3\{(\mathbf{v}^T\mathbf{\Lambda_2})\mathbf{u}^2 - (\mathbf{u}^T\mathbf{v})(\mathbf{u}^T\mathbf{\Lambda_2})\} \\ &\quad -\mathbf{u}^2u_2\{(\mathbf{u}^T\mathbf{v})(\mathbf{u}^T\mathbf{\Lambda_3}) - (\mathbf{v}^T\mathbf{\Lambda_3})\mathbf{u}^2\}, \quad \text{wobei } \mathbf{u}^T\mathbf{v} = 0 \\ &= \mathbf{u}^4\{u_3v_2\Lambda_{22}\Lambda_{33} - u_3v_2\Lambda_{22} - u_2v_3\Lambda_{22}\Lambda_{33} + u_2v_3\Lambda_{33}\} \\ &= \lambda\mathbf{u}^4\{u_3v_2(1+\lambda q_2)q_3 - u_2v_3(1+\lambda q_3)q_2\} \end{aligned}$$

In Vektor–Matrixschreibweise können wir das Ergebnis eleganter formulieren. *trace* steht hierbei für die Spur einer Matrix.

$$\begin{aligned} w_0 &= \lambda\,\mathbf{u}^4\{\lambda^2\mathbf{u}^2\det(\mathbf{Q}) + \lambda(\mathbf{u}^2 trace(adj(\mathbf{Q})) - \mathbf{u}^T adj(\mathbf{Q})\mathbf{u}) + \mathbf{u}^T\mathbf{Q}\mathbf{u}\} \\ \mathbf{w} &= \lambda\mathbf{u}^4\{\lambda adj(\mathbf{Q})(\mathbf{v}\times\mathbf{u}) + \mathbf{v}\times\mathbf{Q}\mathbf{u}\} \\ \mathbf{x}' &= \frac{\lambda adj(\mathbf{Q})(\mathbf{v}\times\mathbf{u}) + \mathbf{v}\times\mathbf{Q}^T\mathbf{u}}{\lambda^2\mathbf{u}^2\det(\mathbf{Q}) + \lambda(\mathbf{u}^2 trace(adj(\mathbf{Q})) - \mathbf{u}^T adj(\mathbf{Q})\mathbf{u}) + \mathbf{u}^T\mathbf{Q}\mathbf{u}} \end{aligned} \tag{2.19}$$

$$\text{mit} \quad \mathbf{u} = \mathbf{c}' - \mathbf{d}', \quad \mathbf{v} = \mathbf{c}'\times\mathbf{d}' \quad \text{und} \quad \mathbf{Q} = \begin{bmatrix} q_1 & 0 & 0 \\ 0 & q_2 & 0 \\ 0 & 0 & q_3 \end{bmatrix}$$

Bemerkung:
Obwohl Formel (2.19) nur für Diagonalmatrizen hergeleitet wurde, gilt sie auch für beliebige Matrizen.

Von entscheidender Bedeutung in der obigen Rechnung ist, daß w_0 und $\mathbf{w}$ den gemeinsamen Faktor λ besitzen. Dieser glückliche Umstand erlaubt im folgenden eine explizite Berechnung aller gesuchten Werte.
Das Einsetzen von $\mathbf{x}'$ in $\mathbf{x}'^T\,\mathbf{Q}\,\mathbf{x}' = q_0$ ergibt eine Gleichung vierten Grades in λ:

$$\begin{aligned} f(\lambda) &= f_4\lambda^4 + f_3\lambda^3 + f_2\lambda^2 + f_1\lambda + f_0 = 0 \\ f_4 &= q_0\mathbf{u}^4\det^2(\mathbf{Q}) \\ f_3 &= 2q_0\mathbf{u}^2\det(\mathbf{Q})\{\mathbf{u}^2 trace(adj(\mathbf{Q})) - \mathbf{u}^T adj(\mathbf{Q})\mathbf{u}\} \\ f_2 &= 2q_0\mathbf{u}^2\det(\mathbf{Q})\mathbf{u}^T\mathbf{Q}\mathbf{u} + q_0\{\mathbf{u}^2 trace(adj(\mathbf{Q})) - \mathbf{u}^T adj(\mathbf{Q})\mathbf{u}\}^2 \\ &\quad -\det(\mathbf{Q})\{(\mathbf{v}^T\mathbf{Q}\mathbf{v})(\mathbf{u}^T\mathbf{Q}\mathbf{u}) - (\mathbf{v}^T\mathbf{Q}\mathbf{u})^2\} \\ f_1 &= 2\mathbf{u}^T\mathbf{Q}\mathbf{u}\{q_0\mathbf{u}^2 trace(adj(\mathbf{Q})) - q_0\mathbf{u}^T adj(\mathbf{Q})\mathbf{u} - \det(\mathbf{Q})\mathbf{v}^2\} \\ f_0 &= \mathbf{u}^T\mathbf{Q}\mathbf{u}\{q_0\mathbf{u}^T\mathbf{Q}\mathbf{u} - \mathbf{v}^T adj(\mathbf{Q})\mathbf{v}\} \end{aligned}$$

Die Nullstellen von $f(\lambda)$ lassen sich explizit berechnen. Wir nennen sie λ_i $(i = 1,\ldots,4)$. Durch Rückwärtseinsetzen der reellen Nullstellen in die Bestimmungsgleichungen (2.19) für $\mathbf{x}'(\lambda)$ und (2.18) für $\nu(\lambda)$ erhalten wir schließlich die lokalen Extrema des euklidischen Abstandes $\mathrm{Lag}(\mathbf{x}'(\lambda_i), \nu(\lambda_i))$ zwischen der Geraden L_{cd} und der quadratischen Fläche Q_{ab}.

$$\delta^2(Q_{ab}, L_{cd}) = \begin{cases} 0 & \text{falls} \quad Q_{ab}\cap L_{cd} \neq \emptyset \\ \min\limits_{\lambda_i\in\mathbb{R}}\{\mathrm{Lag}(\mathbf{x}'(\lambda_i), \nu(\lambda_i))\} & \text{sonst} \end{cases}$$

Lemma 2.1 *Es gibt eine geschlossene Formel zur Berechnung des minimalen euklidischen Abstandes zwischen einer Geraden L_{cd} und einer quadratischen Fläche Q_{ab}.*

Bemerkungen:
Bei der Vereinfachung der arithmetischen Ausdrücke für die Koeffizienten f_0, f_1, f_2 wurde nur angenommen, daß $\mathbf{Q} = \mathbf{Q}^T$ eine symmetrische Matrix ist.
Wenn $q_0\mathbf{u}^T\mathbf{Q}\mathbf{u} = \mathbf{v}^T adj(\mathbf{Q})\mathbf{v}$ gilt, dann berührt die Gerade L_{cd} die Fläche Q_{ab}, denn unter diesen Umständen ist $f_0 = 0$ und somit ist $\lambda = 0$ eine Nullstelle von f. Das aber heißt, daß $Q_{ab} \cap L_{cd} \neq \emptyset$. Diese Beobachtung ist konsistent mit der Tatsache, daß Berührung nach (2.14) dann auftritt, wenn die beiden Schnittpunkte zusammenfallen.

Nun gehen wir von den Geraden zu den Liniensegmenten über. Wir suchen

$$\min_{\substack{0\leq\mu,\nu\leq 1\\ 0\leq t\leq T}} |\mathbf{R}(t)(\mathbf{a}+\mu(\mathbf{b}-\mathbf{a})) - \mathbf{c} - \nu(\mathbf{d}-\mathbf{c})|.$$

Bisher sind die Nebenbedingungen $0 \leq \mu, \nu \leq 1$ und $0 \leq t \leq T$ außer acht geblieben. Allgemein gesehen besteht unsere Aufgabe darin, das absolute Minimum einer stetigen Funktion auf einem kompakten Definitionsgebiet zu finden. Dieses Minimum wird entweder auf dem Rand oder im Inneren des Gebietes angenommen. Wir haben bereits erkannt, daß die Funktion nur endlich viele lokale Extrema besitzt. Wir müssen also nur das Minimum der Funktion auf dem Rand bestimmen und es mit den Funktionswerten der lokalen Extrema vergleichen, die im Inneren des Definitionsgebietes liegen.
Im Koordinatensystem E' rotiert das Liniensegment $l_{ab} = \{\mathbf{a}' + \mu(\mathbf{b}' - \mathbf{a}') | 0 \leq \mu \leq 1\}$ um die $\mathbf{e_3}'$–Achse. Es überstreicht dabei aber nur den Winkelbereich $[0, T]$. Wir müssen also auch überprüfen, ob die Punkte $\mathbf{x}'(\lambda_i) \in Q_{ab}$ nicht außerhalb dieses Bereichs liegen. Die Bedingungen $0 < \mu(\lambda_i), \nu(\lambda_i) < 1$ und $0 < t(\lambda_i) < T$ lassen sich unter Beachtung der Formeln (A.13) und (2.18) äquivalent umformen:

$$\mu(\lambda_i) = \frac{x_3'(\lambda_i) - a_3'}{b_3' - a_3'}$$

$$\begin{aligned}
0 < \mu(\lambda_i) < 1 &\iff \min\{a_3', b_3'\} < x_3' < \max\{a_3', b_3'\} \\
0 < \nu(\lambda_i) < 1 &\iff (\mathbf{d}' - \mathbf{c}')^T\mathbf{c}' < (\mathbf{d}' - \mathbf{c}')^T\mathbf{x}'(\lambda_i) < (\mathbf{d}' - \mathbf{c}')^T\mathbf{d}' \\
t(\lambda_i) < T &\iff \arctan(p_1(\lambda_i)x_2'(\lambda_i) - p_2(\lambda_i)x_1'(\lambda_i), \\
&\qquad p_1(\lambda_i)x_1'(\lambda_i) + p_2(\lambda_i)x_2'(\lambda_i)) < T \\
&\quad \text{mit} \quad \mathbf{p}(\lambda_i) = \mathbf{a}' + \mu(\lambda_i)(\mathbf{b}' - \mathbf{a}')
\end{aligned}$$

Natürlich ist es auch möglich, daß das Liniensegment l_{ab} während seiner Rotation von der Ausgangsstellung bis zum Endwinkel T mit dem stationären Segment l_{cd} zusammenstößt. Diesen Fall können wir mit Gleichung (2.14) behandeln. $\mathbf{x}'(\nu_i) = \mathbf{c}' + \nu_i(\mathbf{d}' - \mathbf{c}')$. Ansonsten lassen sich die Randbedingungen analog zu oben formulieren.

Als Zusammenfassung der einzelnen Resultate dieses Kapitels erhalten wir

Satz 2.2 *Für ein translatorisch oder rotatorisch bewegtes Polyeder $P_1(t)$ und ein stationäres Polyeder P_2 lassen sich die Kollisionszeitpunkte $t^*_{col}(P_1(t), P_2)$ und die Sicherheitsabstände*

$\Delta^*_{min}(P_1(t), P_2)$ ($* = trans$ oder $* = rot$) algebraisch exakt (d.h. ohne numerische Approximation) berechnen. Die Laufzeit beträgt $O(|P_1| \cdot |P_2|)$, wenn man alle arithmetischen Operationen wie die Grundrechenarten, das Wurzelziehen und das Auswerten trigonometrischer Funktionen mit konstanten Kosten veranschlagt.

Kapitel 3

Subquadratische Kollisionsalgorithmen

Ziel dieses Kapitels ist es, Algorithmen für die Kollisionserkennung bei Polyedern zu entwickeln, die selbst im schlechtesten Fall eine subquadratische Laufzeit besitzen. Dabei entspricht die Problemgröße n der Beschreibungskomplexität der beteiligten Polyeder, gemessen in der Zahl der Ecken, Kanten und Flächen. Ohne spezielle Eigenschaften der Polyeder voraussetzen zu müssen, kann man für translatorische Bewegungen zeigen, daß ein subquadratischer Algorithmus mit der Laufzeit $O(n^{2-\varepsilon})$ ($\varepsilon > 0$) existiert, der den Abstand zweier Polyeder bezüglich einer festen Richtung berechnen kann (siehe Abschnitt 3.3). Wenn ein Polyeder konvex ist, läßt sich bereits eine Laufzeit von $O(n \log n)$ erzielen, und wenn beide konvex sind, ist sogar eine sublineare Laufzeit ($O(\log^2 n)$) möglich (siehe Abschnitt 3.1). Neben der Konvexität läßt sich ein weiteres Kriterium für Polyeder charakterisieren, das eine effiziente Abstandsberechnung ermöglicht: Für Polyeder, deren Begrenzungskanten nur in konstant vielen Richtungen verlaufen, läßt sich der Abstand bezüglich einer festen Richtung in der Zeit $O(n \log^2 n)$ bestimmen (siehe Abschnitt 3.2).

Eine Übertragung der für diese Algorithmen grundlegenden Konzepte auf den Fall rotatorischer Bewegungen scheitert. Im Fall der Rotation eines Polyeders um eine feste Achse möchte man den Drehwinkel bestimmen, der das rotierende Polyeder mit einem Hindernispolyeder kollidieren läßt. Zum gegenwärtigen Zeitpunkt ist kein subquadratischer Algorithmus zur Lösung dieses Problems bekannt, selbst dann nicht, wenn die Polyeder konvex sind. Die Schwierigkeiten liegen darin, daß das von einem rotierenden Polyeder überstrichene Volumen nicht von Flächen erster sondern zweiter Ordnung begrenzt ist und im allgemeinen nicht konvex ist.

Während die Nichtlinearität bei Rotationen zum Hauptproblem wird, hat man es bei Translationen durchweg mit linearen bzw. linearisierbaren Problemen zu tun. Dies wird bereits daran deutlich, daß sich das Kollisionsproblem für konvexe, translatorisch bewegte Polyeder als eine Fragestellung der Linearen Programmierung formulieren läßt.

3.1 Ein Kollisionstest für die Translation von konvexen Polyedern

3.1.1 Lineare Programmierung

Wir betrachten folgende einfache Situation. Ein konvexes Polyeder P_1 werde geradlinig in Richtung $\mathbf{s}$ verschoben. Ein zweites konvexes Polyeder P_2 kann als raumfestes Hindernis die Bewegung von P_1 stoppen. Wir fragen nach der größtmöglichen Verschiebung $t_{col}^{trans}(P_1, P_2)$. Unter diesen Voraussetzungen können wir sehr einfach die Existenz eines Linearzeitalgorithmus zur Lösung des Kollisionsproblems beweisen, indem wir es als ein Lineares Programm mit konstanter Variablenzahl formulieren.
O.B.d.A. sei $\mathbf{s} = \mathbf{e_3}$. Wir suchen ein Punktepaar $(\mathbf{p_1}, \mathbf{p_2}) \in P_1 \times P_2$ mit identischen x_1x_2–Koordinaten, das die Distanz in x_3–Richtung minimiert. Das Polyeder $P_i = \bigcap_{f \in F_i} {}^{-}H(f)$ fassen wir als den Schnitt der Halbräume ${}^{-}H(f) = \{\mathbf{x} | \mathbf{n}^T(f)\mathbf{x} \leq n_0(f)\}$ auf, die durch die Flächen aus F_i definiert sind. Die Variablen des Linearen Programms bezeichnen wir mit $z_1, \ldots, z_4$, wobei $\mathbf{p_1} = (z_1, z_2, z_3)^T$ und $\mathbf{p_2} = (z_1, z_2, z_4)^T$.

$$
\begin{array}{ll}
\text{Ziel:} & \text{minimiere } |z_3 - z_4| \quad \text{unter den Nebenbedingungen} \\
\forall f \in F_1: & n_1(f)z_1 + n_2(f)z_2 + n_3(f)z_3 \leq n_0(f) \quad (\text{d.h. } \mathbf{p_1} \in P_1) \\
\forall f \in F_2: & n_1(f)z_1 + n_2(f)z_2 + n_3(f)z_4 \leq n_0(f) \quad (\text{d.h. } \mathbf{p_2} \in P_2)
\end{array}
$$

Da die Dimension ($d = 4$) dieses Linearen Programms konstant ist, läßt es sich nach Megiddo [Me83] in Linearzeit lösen. Obwohl die Proportionalitätskonstante $C(d) < 2^{2^{d+2}}$ (vgl. [Me84]) riesig ist, zeigt dies zumindest prinzipiell, daß Konvexität eine wichtige Eigenschaft für effiziente Kollisionstests bei Translationen ist.
Die konvexen Polyeder P_1 und P_2 sind innerhalb des Linearen Programms nur implizit als Schnitt von Halbräumen repräsentiert. Auf der Grundlage der Randbeschreibung läßt sich für jedes konvexe Polyeder eine sukzessive Approximation konstruieren, die dann als Folge ineinandergeschachtelter konvexer Polyeder vorliegt. Diese hierarchische Art der Repräsentation ist besonders gut zur effizienten Abstandsberechnung geeignet, wie die beiden folgenden Abschnitte zeigen werden.

3.1.2 Hierarchische Repräsentation konvexer Polyeder

In diesem Abschnitt wiederholen wir kurz die Datenstruktur von [DK85] zur hierarchischen Beschreibung eines konvexen Polyeders P.
Wir können uns P als die konvexe Hülle seiner Eckpunkte V vorstellen. Einer der einfachsten Algorithmen zur Konstruktion der konvexen Hülle für eine Punktmenge $V = \{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}\}$ arbeitet inkrementell (siehe [Ed87]). Angenommen wir haben die konvexe Hülle $P^{(i)}$ für die Punktmenge $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_i}\}$ bereits konstruiert, dann ergibt sich $P^{(i+1)}$ indem wir auf $P^{(i)}$ eine "Pyramide" aufsetzen, deren Spitze von dem neuen Punkt $\mathbf{v_{i+1}}$ gebildet wird. Die Basis dieser "Pyramide" wird von all den Flächen von $P^{(i)}$ gebildet, die von $\mathbf{v_{i+1}}$ sichtbar sind. Den Übergang von $P^{(i)}$ zu $P^{(i+1)}$ nennen wir eine Verfeinerung. Die Folge $P^{(1)}, P^{(2)}, \ldots, P^{(n)} = P$ ineinandergeschachtelter Polyeder kann als eine einfache hierarchische Repräsentation des Polyeders P angesehen werden.

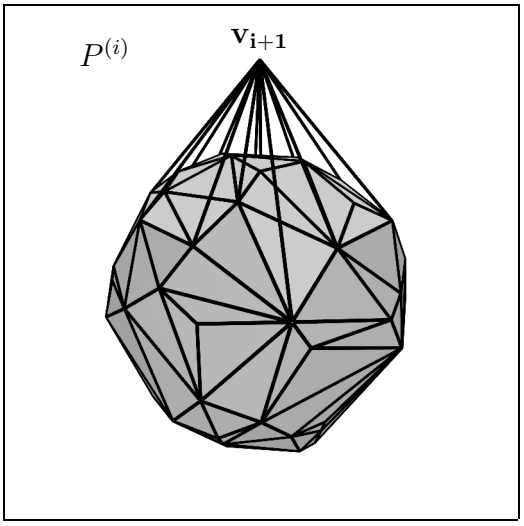


Abbildung 3.1: Verfeinerung von $P^{(i)}$ durch Aufsetzen einer "Pyramide"

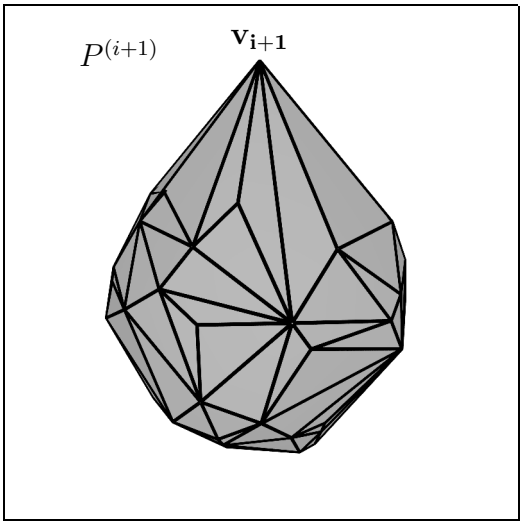


Abbildung 3.2: Das Resultat des Verfeinerungsschrittes

Sinn und Zweck einer solchen hierarchischen Beschreibung ist der Entwurf effizienter hierarchischer Algorithmen. Da die Laufzeit dieser Algorithmen in der Regel von der Tiefe der Hierarchie abhängt, ist ein vorrangiges Ziel, diese Tiefe so klein wie möglich zu halten. Andererseits darf natürlich ein Verfeinerungsschritt nicht zu komplex sein.
Das oben beschriebene Verfahren liefert leider eine große Hierarchietiefe, und es kann auch nicht garantieren, daß nur "Pyramiden" kleiner Komplexität zur Verfeinerung benutzt werden.
Wenn man während eines Verfeinerungsschrittes parallel mehrere sich nicht überdeckende "Pyramiden" aufsetzt, kann man die lineare Hierarchietiefe der oben beschriebenen Darstellung jedoch erheblich reduzieren. Dabei kann man gleichzeitig gewährleisten, daß die zur Verfeinerung benutzten "Pyramiden" nur konstante Komplexität haben. Dieses Ergebnis stammt von [DK85]:

> Für jedes konvexe Polyeder P mit $|P| = n$ gibt es eine Folge $P^{(1)} \subset P^{(2)} \subset \ldots \subset P^{(k)}$ konvexer Polyeder mit folgenden Eigenschaften:
>
> (1) $P^{(k)} = P$ und $|P^{(1)}| \leq c_1$,
>
> (2) $V^{(i)} \subset V^{(i+1)}$,
>
> (3) $V^{(i+1)} \setminus V^{(i)}$ bildet eine unabhängige Knotenmenge in dem Graphen $G^{(i+1)} = (V^{(i+1)}, E^{(i+1)})$, d.h. keine zwei Knoten aus $V^{(i+1)} \setminus V^{(i)}$ sind über eine Kante aus $E^{(i+1)}$ miteinander verbunden.
>
> (4) $\max\limits_{i} \max\limits_{v \in V^{(i+1)} \setminus V^{(i)}} deg_{G^{(i+1)}}(v) \leq c_2$
>
> Die Konstante c_1 steht für die Komplexität des innersten Polyeders der Hierarchie. Während jedes Verfeinerungsschrittes ist die Komplexität der aufgesetzten "Pyramiden" durch eine Konstante c_2 beschränkt. Gleichzeitig wird eine Hierarchietiefe von $k = O(\log n)$ erreicht.

Die Existenz einer solchen logarithmischen Hierarchie beruht auf der Tatsache, daß ein planarer Graph $G = (V, E)$ immer einen konstanten Bruchteil von Knoten mit konstantem Grad $\leq c_2$ besitzt. Unter diesen gibt es wieder einen konstanten Anteil von Knoten, die nicht über Kanten von E direkt benachbart sind. Man zeigt leicht, daß sich eine geeignete Knotenmenge mit einem Linearzeitalgorithmus finden läßt, so daß auch der Aufbau der gesamten Hierarchie nur $O(n)$ Zeit beansprucht. Deshalb dürfen wir im folgenden davon ausgehen, daß für jedes Polyeder immer auch seine hierarchische Repräsentation zur Verfügung steht.

3.1.3 Effiziente Abstandsberechnungen auf der Basis hierarchischer Repräsentationen

Wir zeigen nun, wie man mit Hilfe der hierarchischen Repräsentation Kollisionsfragen bei der Translation konvexer Polyeder effizient beantworten kann. Das folgende Lemma bildet die Grundlage:

Lemma 3.1 *Der euklidische Abstand δ und die in Translationsrichtung* **s** *größtmögliche Verschiebung t_{col}^{trans} eines Punktes a, einer Geraden L und einer Ebene H zu einem konvexen Polyeder P lassen sich in logarithmischer Zeit $O(\log |P|)$ berechnen.*

Beweis: Wir zeigen zuerst, wie man $\delta(Q, P)$ für $Q \in \{\mathbf{a}, L, H\}$ bestimmt. Dazu nehmen wir induktiv an, daß wir zwei Punkte $\mathbf{q_i} \in Q$ und $\mathbf{p_i} \in P^{(i)}$ berechnet haben, die den Abstand $\delta(Q, P^{(i)}) = \delta(\mathbf{q_i}, \mathbf{p_i})$ festlegen. Diese Invariante wollen wir aufrechterhalten, wenn wir zur nächsten Hierarchiestufe übergehen. Jeder Verfeinerungsschritt wird dabei nur konstante Kosten verursachen, weil wir die Verfeinerung lokal begrenzen können. Um das einzusehen, betrachten wir die Hilfsebene $\Pi^{(i)} := \{\mathbf{x} | (\mathbf{q_i} - \mathbf{p_i})^T(\mathbf{x} - \mathbf{p_i}) = 0\}$, die senkrecht auf der kürzesten Verbindungslinie zwischen $P^{(i)}$ und Q steht und $P^{(i)}$ berührt. ${}^{\pm}\Pi^{(i)}$ bezeichne die beiden durch $\Pi^{(i)}$ begrenzten Halbräume, wobei $P^{(i)} \subset {}^{-}\Pi^{(i)}$. Wenn es sich bei Q um eine Gerade oder eine Ebene handelt, so gilt offenbar $Q \parallel \Pi^{(i)}$.
Demzufolge können bei der Verfeinerung von $P^{(i)}$ zu $P^{(i+1)}$ nur solche "Pyramiden" den bereits gefundenen Abstand verkürzen, die in den Halbraum ${}^{+}\Pi^{(i)}$ hineinragen. Wie wir gleich sehen werden, können dafür aber nur "Pyramiden" in unmittelbarer Nähe des Punktes $\mathbf{p_i}$ in Frage kommen, wenn die Konvexität erhalten bleiben soll. Um diese "Pyramiden" schnell zu finden, schaffen wir eine Verbindung zwischen den einzelnen Hierarchiestufen, indem wir zu jeder Fläche von $P^{(i)}$ gegebenenfalls einen Zeiger auf die überdeckende "Pyramide" der nächsten Stufe abspeichern.
Anhand der Lage des Punktes $\mathbf{p_i}$ auf $P^{(i)}$ unterscheiden wir drei verschiedene Situationen, die eine entsprechende Vorgehensweise zum Auffinden des Punktes $\mathbf{p_{i+1}}$ erfordern.

(1) $\mathbf{p_i}$ liegt im Inneren einer Fläche $f \subset P^{(i)}$.

(2) $\mathbf{p_i}$ liegt auf einer Kante $e \subset P^{(i)}$.

(3) $\mathbf{p_i}$ ist identisch mit einer Ecke $v \in P^{(i)}$.

Im Fall (1) müssen wir $\mathbf{p_{i+1}}$ nur auf der f überdeckenden "Pyramide" suchen, sofern diese überhaupt existiert. Im Fall (2) können wir die lokale Verfeinerung auf die beiden "Pyramiden" beschränken, die die zur Kante e inzidenten Flächen überdecken. Für den Fall (3) schließlich müssen wir ebenfalls mit einer konstanten Anzahl von "Pyramiden" zur Verfeinerung auskommen. Deshalb dürfen wir nicht einfach alle zur Ecke v inzidenten Flächen und ihre zugehörigen "Pyramiden" betrachten. Wir gehen vielmehr folgendermaßen vor:
Wenn während des Verfeinerungsprozesses erstmals die Ecke v die Rolle des Punktes $\mathbf{p_i}$ annimmt, so war v die Spitze der "Pyramide", die im i–ten Schritt benutzt wurde, d. h. in $P^{(i)}$ gehen von v nur konstant viele Kanten aus. Diese Kanten projizieren wir auf eine Ebene ${}^{\perp}\Pi^{(i)}$ senkrecht zu $\Pi^{(i)}$ (siehe Abbildung 3.3) und merken uns die beiden Kanten, die mit der orientierten Schnittgeraden $\Pi^{(i)} \cap {}^{\perp}\Pi^{(i)}$ den kleinsten und größten Winkel einschließen. Diese beiden Kanten nennen wir in Anlehnung an [EM81] *extrem*. Welche Kanten als extrem angesehen werden, hängt zwar von der Wahl von ${}^{\perp}\Pi^{(i)}$ ab, aber entscheidend ist nur, daß eine anfangs getroffene Wahl beibehalten wird, solange die Ecke v als Punkt $\mathbf{p_i}$ fungiert. Die Bedeutung der extremen Kanten liegt bei der Verfeinerung darin, daß jede "Pyramidenspitze", die in ${}^{+}\Pi^{(i)}$ hineinragt, eine der extremen Kanten sieht, wenn die Ecke v zu den Basisflächen dieser "Pyramide" gehört. Nur die "Pyramiden", deren Spitzen eine der extremen Kanten sehen, können im nächsten Schritt zu neuen extremen Kanten beitragen. Aus diesem Grund genügt es, nur die vier Flächen zu berücksichtigen, die zu den extremen Kanten inzident sind. Falls eine der darauf aufgesetzten "Pyramiden" in den Halbraum ${}^{+}\Pi^{(i)}$ hineinragt, so ist der Punkt $\mathbf{p_{i+1}}$ auf dieser "Pyramide" zu finden. Anderenfalls setzen wir

$\mathbf{p_{i+1}} = \mathbf{p_i}$, $\Pi^{(i+1)} = \Pi^{(i)}$. Für die dabei notwendige lokale Verfeinerung kommen höchstens vier "Pyramiden" in Frage. Wir projizieren all ihre Kanten auf die Ebene ${}^{\perp}\Pi^{(i+1)} = {}^{\perp}\Pi^{(i)}$ und können so die neuen extremen Kanten in konstanter Zeit ermitteln.

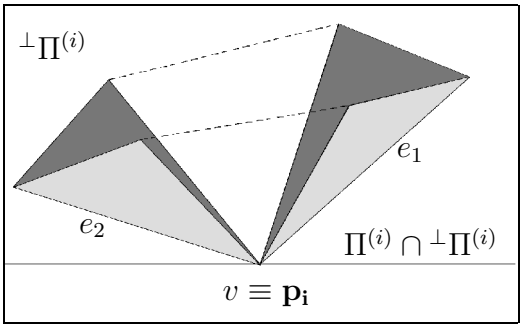


Abbildung 3.3: Projektion der extremen Kanten (e_1, e_2) und ihrer inzidenten Flächen in die Ebene ${}^{\perp}\Pi^{(i)}$

Diese Überlegungen zeigen uns, daß wir pro Schritt mit konstantem Aufwand die lokalen Verfeinerungen lenken können, um ein Punktepaar $(\mathbf{q_k}, \mathbf{p_k})$ zu bestimmen, das den euklidischen Abstand $\delta(Q, P)$ zwischen dem Polyeder $P = P^{(k)}$ und einem Punkt $Q = \mathbf{a}$, einer Geraden $Q = L$ oder einer Ebene $Q = H$ minimiert.
Wir werden jetzt darüber hinaus zeigen, daß wir auch die in Translationsrichtung $\mathbf{s}$ größtmögliche Verschiebung $t_{col}^{trans}(Q, P)$ für $Q \in \{\mathbf{a}, L, H\}$ in ähnlicher Weise effizient berechnen können.
Wir beginnen mit dem Fall einer translatorisch bewegten Ebene $Q = H$. Für die Translationsrichtung gelte: $\mathbf{s} \not\parallel H$. Mit einer kleinen Modifikation können wir genauso verfahren wie bei der Berechnung von $\delta(H, P)$. Wir benötigen nur eine neue Definition der Hilfsebene $\Pi^{(i)} := \{\mathbf{x} | \mathbf{n}_H^T(\mathbf{x} - \mathbf{p_i}) = 0\}$. $\mathbf{n}_H$ bezeichnet den Normalenvektor der Ebene H.
Bei der Translation des Punktes $\mathbf{a}$ bewegt sich dieser auf dem Strahl $\{\mathbf{a} + t\mathbf{s} | t \geq 0\}$. Zur Berechnung von $t_{col}^{trans}(\mathbf{a}, P)$ müssen wir nur diesen Strahl mit dem Polyeder P schneiden. Angenommen wir haben einen Schnittpunkt $\mathbf{p_i} \in P^{(i)}$ gefunden, der im Inneren einer Fläche f liegt, dann befindet sich $\mathbf{p_{i+1}}$ auf der "Pyramide" über f. Wenn $\mathbf{p_i}$ auf einer Kante e liegt, müssen wir nur die "Pyramiden" zu den beiden angrenzenden Flächen betrachten. Und falls $\mathbf{p_i}$ einer Ecke v entspricht, haben wir den gesuchten Schnittpunkt mit P bereits gefunden. Anfangs ist es jedoch möglich, daß kein Schnitt zwischen dem Strahl und $P^{(i)}$ vorliegt. Dann gehen wir so vor, als ob wir den minimalen euklidischen Abstand zwischen $P^{(i)}$ und der Gerade $\{\mathbf{a} + t\mathbf{s} | t \in \mathbb{R}\}$ ermitteln wollten. Auf diese Weise werden die lokalen Verfeinerungen an die richtige Stelle gelenkt, und wir finden irgendwann den ersten Schnittpunkt mit dem Strahl oder stellen fest, daß $t_{col}^{trans}(\mathbf{a}, P) = \infty$ ist.
Nun steht nur noch die Diskussion des Falles aus, an dem eine Gerade $Q = L$ und ein Polyeder P beteiligt sind. Prinzipiell könnten wir P mit der Ebene $H_L = \{\mathbf{x} + t\mathbf{s} | \mathbf{x} \in L, t \in \mathbb{R}\}$ schneiden und die Berechnung von $t_{col}^{trans}(L, P \cap H_L)$ als 2–dimensionales Problem auffassen. Wir können es uns aber nicht leisten, $P \cap H_L$ explizit zu berechnen. Statt dessen verwenden wir wieder die Methode der lokalen Verfeinerungen. Nehmen wir also an, daß wir

zwei Punkte $\mathbf{q_i} \in Q$ und $\mathbf{p_i} \in P^{(i)} \cap H_L$ kennen, die den Abstand zwischen Q und $P^{(i)}$ in Richtung $\mathbf{s}$ minimieren. Im allgemeinen wird $\mathbf{p_i}$ auf einer Kante e liegen. Daher genügt es pro Verfeinerungsschritt, die "Pyramiden" aufzusetzen, die Flächen bedecken, die zu e inzident sind. Nur dort kann sich $\mathbf{p_{i+1}}$ befinden. Um auch den Sonderfall in den Griff zu bekommen, bei dem $\mathbf{p_i}$ auf einer Ecke v liegt, merken wir uns die zu v inzidenten Flächen, deren Schnitte mit H_L den kleinsten und größten Winkel mit der orientierten Gerade L einschließen. Nur auf diese Flächen werden bei der lokalen Verfeinerung "Pyramiden" aufgesetzt. Die beschriebene Vorgehensweise kann aber erst dann einsetzen, wenn $P^{(i)} \cap H_L \neq \emptyset$ gilt. Solange das nicht der Fall ist, steuern wir die lokalen Verfeinerungen wie bei der Bestimmung des minimalen euklidischen Abstandes $\delta(H_L, P^{(i)})$.
Insgesamt haben wir damit bewiesen, daß auch bei der Berechnung von $t_{col}^{trans}(Q, P)$ zwischen dem Polyeder $P = P^{(k)}$ und einem Punkt $Q = \mathbf{a}$, einer Geraden $Q = L$ oder einer Ebene $Q = H$ lokale Verfeinerungen zum Durchqueren der Hierarchie ausreichen, und daß jeder Verfeinerungsschritt in konstanter Zeit durchgeführt werden kann. ■

Die Aussagen des Lemmas 3.1 finden sich zum Teil schon in den Arbeiten von [EM81, DK85] und später wieder in [DK90] und [DHKS90].
Mit Hilfe dieses Lemmas erhalten wir effiziente Algorithmen zur Berechnung des minimalen euklidischen Abstandes $\delta(P_1, P_2)$, der maximal möglichen Verschiebung $t_{col}^{trans}(P_1, P_2)$ in Richtung $\mathbf{s}$ und des dabei eingehaltenen Sicherheitsabstandes $\Delta_{min}^{trans}(P_1, P_2)$ für zwei Polyeder P_1 und P_2. Wir verdeutlichen die generelle Vorgehensweise zunächst am Beispiel der Berechnung von $\delta_{min} := \delta(P_1, P_2)$. Die Notationen $H(f)$ und $L(e)$ stehen für die durch f definierte Ebene und für die durch e bestimmte Gerade.

$\delta_{min} \leftarrow \infty$;
forall $f_1 \in F_1$
do bestimme einen Punkt $\mathbf{p} \in H(f_1)$ so, daß
  $\delta(H(f_1), P_2) = \delta(\mathbf{p}, P_2)$;
  if $\mathbf{p} \in f_1$ **then** $\delta_{min} \leftarrow \min\{\delta_{min}, \delta(\mathbf{p}, P_2)\}$ **fi**
od;
forall $e_1 \in E_1$
do bestimme einen Punkt $\mathbf{p} \in L(e_1)$ so, daß
  $\delta(L(e_1), P_2) = \delta(\mathbf{p}, P_2)$;
  if $\mathbf{p} \in e_1$ **then** $\delta_{min} \leftarrow \min\{\delta_{min}, \delta(\mathbf{p}, P_2)\}$ **fi**
od;
forall $v_1 \in V_1$
do bestimme $\delta(v_1, P_2)$;
  $\delta_{min} \leftarrow \min\{\delta_{min}, \delta(v_1, P_2)\}$
od;

Diese Prozedur läßt sich auf die Berechnung von $t_{col}^{trans}(P_1, P_2)$ direkt übertragen. Zur Bestimmung von $\Delta_{min} := \Delta_{min}^{trans}(P_1, P_2)$ müssen wir etwas anders vorgehen. Dabei bezeichne

$e^{trans} := \{\mathbf{x}+t\mathbf{s}|\mathbf{x} \in e, t \in [0,1]\}$ die von der Kante e während der Translation überstrichene Fläche. Für eine Ecke v sei die Strecke v^{trans} analog definiert.

$\Delta_{min} \leftarrow \min\{\delta(P_1, P_2), \delta(P_1 + \mathbf{s}, P_2)\}$;
forall $e_1 \in E_1$
do bestimme einen Punkt $\mathbf{p} \in H(e_1^{trans})$ so, daß
  $\delta(H(e_1^{trans}), P_2) = \delta(\mathbf{p}, P_2)$;
  if $\mathbf{p} \in e_1^{trans}$ **then** $\Delta_{min} \leftarrow \min\{\Delta_{min}, \delta(\mathbf{p}, P_2)\}$ **fi**
od;
forall $v_1 \in V_1$
do bestimme einen Punkt $\mathbf{p} \in L(v_1^{trans})$ so, daß
  $\delta(L(v_1^{trans}), P_2) = \delta(\mathbf{p}, P_2)$;
  if $\mathbf{p} \in v_1^{trans}$ **then** $\Delta_{min} \leftarrow \min\{\Delta_{min}, \delta(\mathbf{p}, P_2)\}$ **fi**
od;

Wenn wir voraussetzen, daß das Polyeder P_2 konvex ist, können wir Lemma 3.1 anwenden, um die Distanzberechnungen durchzuführen, die in den obigen Prozeduren vorkommen. Als Konsequenz erhalten wir folgenden Satz.

Satz 3.2 *Für zwei Polyeder P_1 und P_2, von denen P_2 konvex ist, läßt sich der minimale euklidische Abstand $\delta(P_1, P_2)$, die maximal mögliche Verschiebung $t_{col}^{trans}(P_1, P_2)$ in beliebiger Richtung* $\mathbf{s}$ *und der dabei eingehaltene Sicherheitsabstand $\Delta_{min}^{trans}(P_1, P_2)$ in Zeit $O(|P_1| \log |P_2|)$ berechnen, wenn das konvexe Polyeder P_2 hierarchisch repräsentiert ist.*

Dieses Ergebnis besitzt Ähnlichkeit zu dem Resultat von [MS85], das besagt, daß sich der Schnitt zweier Polyeder in Zeit $O(n \log n)$ konstruieren läßt, wenn eines der beiden Polyeder konvex ist.

Interessanterweise läßt sich Satz 3.2 anwenden, um für das Optimierungsproblem des "Kleinsten Aussichtsturmes" eine Verbesserung der Laufzeit gegenüber [Sh88a] zu erzielen. Deshalb schieben wir Abschnitt 3.1.4 ein.

3.1.4 Das Problem des "Kleinsten Aussichtsturmes"

Sharir beschreibt in [Sh88a] die Problemstellung wie folgt:

> Auf einem Gelände soll ein Aussichtsturm gebaut werden, von dessen Spitze aus man das ganze Gelände überblicken kann. Gesucht ist ein Bauplatz, auf dem ein Turm kleinst möglicher Höhe errichtet werden kann.

Zur Modellierung des Geländes legen wir einen triangulierten planaren Graphen in die x_1x_2–Ebene und ordnen jedem Knoten des Graphen eine Höhe in Form der x_3–Koordinate zu. Die Flächen der so entstandenen polygonalen Oberfläche bezeichnen wir mit $F_1 = \{f_1, f_2, \ldots f_n\}$,

das Polytop selbst mit P_1. Es ist leicht einzusehen, daß die Spitze eines solchen Aussichtturms oberhalb jeder durch f_i definierten Ebene $H(f_i)$ liegen muß, d.h. in dem Halbraum ${}^{+}H(f_i)$. Liegt umgekehrt die Spitze oberhalb $H(f_i)$ für alle $1 \leq i \leq n$, so hat man von dort überall im Gelände Einblick. Folglich muß sich die Turmspitze in dem konvexen Polytop P_2 befinden, das durch den Schnitt $P_2 = \bigcap_{i=1}^{n} {}^{+}H(f_i)$ entsteht. Es bleibt also die Aufgabe, die minimale vertikale Distanz zwischen dem nicht notwendigerweise konvexen Polytop P_1 und dem konvexen Polytop P_2 zu bestimmen. Das entspricht gerade dem Wert von $t_{col}^{trans}(P_1, P_2)$ für $\mathbf{s} = \mathbf{e_3}$. Unter Umständen ist es aber gar nicht notwendig, einen Aussichtsturm zu bauen, denn wenn $P_1 \cap P_2 \neq \emptyset$, dann gibt es auf dem Gelände selbst einen Punkt, von dem aus alles überschaubar ist.

Sharir [Sh88a] gelingt es nur, einen Algorithmus mit der Zeitkomplexität $O(n \log^2 n)$ zur Lösung des Problems des "Kleinsten Aussichtsturmes" anzugeben, er äußert aber die Vermutung, daß eine Laufzeit in der Größenordnung von $O(n \log n)$ möglich sein sollte. Satz 3.2 liefert uns den Beweis dafür: P_2 kann als Schnitt von n Halbräumen in Zeit $O(n \log n)$ konstruiert werden. Anschließend kann die Datenstruktur für die hierarchische Repräsentation für P_2 in Zeit $O(n)$ aufgebaut werden. Die minimale vertikale Distanz zwischen P_1 und P_2 finden wir gemäß unseres Satzes in Zeit $O(n \log n)$ und auch den Sonderfall $P_1 \cap P_2 \neq \emptyset$ können wir in dieser Zeit abhandeln. ∎

Wir verschärfen nun die Voraussetzungen von Satz 3.2 und fordern die Konvexität beider Polyeder $P_1 = Q$ und $P_2 = P$. In [DK90] wird skizziert, daß $\delta(Q, P)$ in diesem Fall sogar in Zeit $O(\log |P| \log |Q|)$ berechenbar ist. Wir übertragen dieses Resultat jetzt auf die Berechnung von $t_{col}^{trans}(Q, P)$. Dabei benutzen wir die gleichen Methoden wie im Beweis zu Lemma 3.1. Wir nehmen wieder induktiv an, daß wir bereits ein Punktepaar $(\mathbf{q_i}, \mathbf{p_i}) \in Q \times P^{(i)}$ kennen, für das $t_{col}^{trans}(Q, P^{(i)}) = t_{col}^{trans}(\mathbf{q_i}, \mathbf{p_i}) < \infty$ gilt. Die Hilfsebene $\Pi^{(i)} = \{\mathbf{x} | \mathbf{n}^T(\mathbf{x} - \mathbf{p_i}) = 0\}$ sei so gewählt, daß $P^{(i)} \subset {}^{-}\Pi^{(i)}$ gilt, daß sie tangential zu $P^{(i)}$ verläuft und daß die dazu parallele Ebene $\{\mathbf{x} | \mathbf{n}^T(\mathbf{x} - \mathbf{q_i}) = 0\}$ das konvexe Polyeder Q berührt.

Unser Ziel ist es, durch eine geeignete lokale Verfeinerung von $P^{(i)}$ das neue Punktepaar $(\mathbf{q_{i+1}}, \mathbf{p_{i+1}}) \in Q \times P^{(i+1)}$ zu finden. Da auch hier $\mathbf{p_{i+1}}$ in ${}^{+}\Pi^{(i)}$ liegt, müssen wir nur Verfeinerungen in unmittelbarer Nähe des Punktes $\mathbf{p_i}$ vornehmen. Diese ergeben sich entsprechend seiner Lage – er kann im Inneren einer Fläche f, auf einer Kante e oder auf einer Ecke v des Polyeders $P^{(i)}$ liegen. In allen drei Fällen können wir uns im Grunde wie in Lemma 3.1 verhalten.

Pro Verfeinerungschritt werden zur Bestimmung von $\mathbf{p_{i+1}}$ höchstens vier "Pyramiden" mit konstanter Komplexität untersucht. Für jede Fläche, Kante und Ecke dieser "Pyramiden" läßt sich $t_{col}^{trans}(Q, .)$ mittels der hierarchischen Repräsentation von Q in Zeit $O(\log |Q|)$ berechnen und der neue Punkt $\mathbf{q_{i+1}}$ ermitteln. Da die Polyederhierarchie von P nur logarithmische Tiefe besitzt, kommen wir also mit $O(\log |P| \log |Q|)$ Zeit aus. Die Voraussetzung war jedoch, daß wir zu Anfang zwei Punkte $\mathbf{q_i}$ und $\mathbf{p_i}$ kennen, die den Abstand zwischen Q und $P^{(i)}$ in Richtung $\mathbf{s}$ minimieren.

Sei i der kleinste Index, für den $t_{col}^{trans}(Q, P^{(i)}) < \infty$ gilt. Um unsere Argumentation zu vervollständigen, müssen wir noch zeigen, wie wir für $1 \leq j < i$ die lokalen Verfeinerungen in geeigneter Weise lenken. Dazu projizieren wir $P^{(j)}$ und Q auf eine Ebene ${}^{\perp}\Pi$ senkrecht

zur Translationsrichtung $\mathbf{s}$. Die Projektionen bezeichnen wir mit $\overline{P}^{(j)}$ und $\overline{Q}$. Es gilt:

$$\forall 1 \leq j < i : \ \overline{P}^{(j)} \cap \overline{Q} = \emptyset, \quad \overline{P}^{(i)} \cap \overline{Q} \neq \emptyset$$

Als Richtlinie für die ersten Verfeinerungen dient uns der minimale euklidische Abstand $\delta(\overline{Q}, \overline{P}^{(j)})$ der projizierten Polyeder. Das Punktepaar $(\mathbf{q_j}, \mathbf{p_j})$ werde stets so berechnet, daß $\delta(\overline{\mathbf{q_j}}, \overline{\mathbf{p_j}}) = \delta(\overline{Q}, \overline{P}^{(j)})$ gilt. Die Tangentialebene $\Pi^{(j)}$, die $P^{(j)}$ im Punkt $\mathbf{p_j}$ berührt, ist bestimmt durch $\{\mathbf{x} | \mathbf{n}^T(\mathbf{x} - \mathbf{p_j}) = 0\}$, wobei der Normalenvektor $\mathbf{n} = \overline{\mathbf{q_j}} - \overline{\mathbf{p_j}}$ senkrecht zu $\mathbf{s}$ steht. Nur die "Pyramide", die beim Verfeinern von $P^{(j)}$ in den Halbraum ${}^{+}\Pi^{(j)}$ hineinragt, kann den euklidischen Abstand zwischen den projizierten Polyedern vermindern. Im Fall $j = i - 1$ kommt es erstmals zum Schnitt zwischen $\overline{Q}$ und $\overline{P}^{(i)}$ aufgrund dieser zuletzt aufgesetzten "Pyramide". Eben diese "Pyramide" ist es dann auch, die bei der Translation von Q in Richtung $\mathbf{s}$ eine Kollision mit $P^{(i)}$ verursacht – immer unter der Voraussetzung, daß $t_{col}^{trans}(Q, P^{(i)}) < \infty$. (Wenn keine Projektion $\overline{P}^{(j)}$ einen Schnitt mit $\overline{Q}$ aufweist oder wenn $t_{col}^{trans}(Q, P^{(i)}) = \infty$ ist, können wir schließen, daß auch $t_{col}^{trans}(Q, P) = \infty$ ist.) Ein Verfeinerungsschritt selbst, einschließlich der Behandlung extremer Kanten in der Projektionsebene ${}^{\perp}\Pi$, verläuft analog zu oben, so daß jeder Übergang von $P^{(j)}$ zu $P^{(j+1)}$ nur logarithmischen Aufwand $O(\log |Q|)$ bedeutet. Über beide Phasen des Algorithmus summiert erhält man eine Laufzeit von $O(\log |P| \log |Q|)$. Diese Ausführungen beweisen den folgenden Satz.

Satz 3.3 *Mit Hilfe hierarchischer Repräsentationen für zwei konvexe Polyeder P_1 und P_2 läßt sich die größtmögliche Verschiebung $t_{col}^{trans}(P_1, P_2)$ von P_1 zu P_2 in Zeit $O(\log |P_1| \log |P_2|)$ berechnen.*

3.2 Ein Kollisionstest für die Translation von iso–orientierten Polyedern

Als Spezialfall betrachten wir nun Polyeder, bei denen die Kanten nur in konstant vielen Richtungen verlaufen können, sogenannte c–orientierte Polyeder. Es gibt zum Beispiel nur drei verschiedene Kantenorientierungen für Polyeder, die aus achsenparallelen Quadern zusammengebaut sind. Im folgenden betrachten wir zwei Polyeder P_1 und P_2 mit obiger Eigenschaft. Die Konstanten c_1 und c_2 sollen die Zahl der verschiedenen Kantenorientierungen von P_1 und P_2 messen. Unter diesen Voraussetzungen wollen wir zeigen, daß sich das translatorische Kollisionsproblem effizient lösen läßt. Unser Ziel wird es sein, eine Laufzeit von $O((|P_1| + |P_2|) \log^2(|P_1| + |P_2|))$ zu erreichen.

Der Grund dafür, daß sich die Kollisionsberechnungen im Fall von c–orientierten Polyedern effizient durchführen lassen, liegt darin, daß sich die auftretenden Teilprobleme durch geeignete Transformationen auf Schnittprobleme von orthogonalen Objekten wie Punkten, horizontalen und vertikalen Liniensegmenten und achsenparallelen Rechtecken im $\mathbb{R}^2$ reduzieren lassen. Eines der Teilprobleme bei der Berechnung von $t_{col}^{trans}(P_1, P_2)$ ist zum Beispiel die Bestimmung des Kollisionszeitpunktes $t_{col}^{trans}(E_1, E_2)$ für die zugehörigen Polyederkanten (vgl. Abschnitt 3.2.1). Da in E_i nur c_i verschiedene Richtungen vorkommen, können wir für jede der $c_1 c_2$ Richtungskombinationen die Aufgabenstellung separat lösen. Durch den Übergang

zu schiefwinkligen Koordinaten lassen sich die Kanten, die zu einem Paar von Richtungen gehören, in horizontale und vertikale Kanten verwandeln. Dadurch erhält man konstant viele Abstandsprobleme zwischen orthogonalen Objekten. (Dafür stehen eine Reihe effizienter Algorithmen zur Verfügung, die oftmals auf der Sweep–Methode beruhen. Ihr Einsatz erfordert dynamische Datenstrukturen wie balancierte Suchbäume und Segmentbäume (siehe [Meh84]), die entsprechend geschachtelt werden.) Eine ähnliche Technik kann man auch bei der Berechnung von $t_{col}^{trans}(V_1, F_2)$ anwenden (vgl. Abschnitt 3.2.2).

3.2.1 Bestimmung von $t_{col}^{trans}(E_1, E_2)$

Wir wollen den minimalen Abstand $t_{col}^{trans}(P_1, P_2)$ von P_1 und P_2 in Translationsrichtung $\mathbf{s}$ bestimmen. Dazu berechnen wir zuerst den minimalen Abstand in Richtung $\mathbf{s}$ zwischen den beiden Kantenmengen E_1 und E_2. Diese lassen sich anhand der Kantenrichtungen partitionieren, d.h.

$$E_1 = \bigcup_{i=1}^{c_1} E_1^i \quad \text{und} \quad E_2 = \bigcup_{j=1}^{c_2} E_2^j.$$

Es gilt:

$$t_{col}^{trans}(E_1, E_2) = \min_{\substack{1 \le i \le c_1 \\ 1 \le j \le c_2}} t_{col}^{trans}(E_1^i, E_2^j).$$

Wir müssen also nur zeigen, wie man alle Kanten einer festen Richtung $\mathbf{s_1}$ von P_1 gegen alle Kanten einer festen Richtung $\mathbf{s_2}$ von P_2 testet.
Wir dürfen annehmen, daß die drei ausgezeichneten Richtungen $\mathbf{s_1}$, $\mathbf{s_2}$ und $\mathbf{s}$ linear unabhängig voneinander sind. (Andernfalls kann man davon ausgehen, daß $t_{col}^{trans}(P_1, P_2)$ durch ein Ecke–Flächenpaar bestimmt wird.) Um die Situation noch weiter zu vereinfachen, transformieren wir die Basis $(\mathbf{s_1}, \mathbf{s_2}, \mathbf{s})$ in die Standardbasis $(\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3})$. Die Transformationsmatrix für den Basiswechsel ist durch $\mathbf{S} = [\mathbf{s_1}, \mathbf{s_2}, \mathbf{s}]^{-1}$ gegeben. Wie man aus den Gleichungen (2.2)-(2.4) ablesen kann, ist $t_{col}^{trans}(l_{ab}, l_{cd})$ invariant unter dieser Transformation. (l_{ab} und l_{cd} bezeichnen zwei Kanten, die zwischen den Punkten a, b und c, d verlaufen.) Aus Gleichung (2.2) folgt mit Hilfe des Determinantenmultiplikationssatzes: (analog für (2.3) und (2.4))

$$t = \frac{\det(\mathbf{S}) \cdot \det[\mathbf{c}-\mathbf{a}, \mathbf{b}-\mathbf{a}, \mathbf{d}-\mathbf{c}]}{\det(\mathbf{S}) \cdot \det[\mathbf{s}, \mathbf{b}-\mathbf{a}, \mathbf{d}-\mathbf{c}]} = \frac{\det[\mathbf{S}(\mathbf{c}-\mathbf{a}), \mathbf{S}(\mathbf{b}-\mathbf{a}), \mathbf{S}(\mathbf{d}-\mathbf{c})]}{\det[\mathbf{Ss}, \mathbf{S}(\mathbf{b}-\mathbf{a}), \mathbf{S}(\mathbf{d}-\mathbf{c})]}$$

Danach stellt sich unser Problem wie folgt dar:

> **Gegeben:** Eine Menge E_H von horizontalen (x_1–Richtung) und eine Menge E_V von vertikalen (x_2–Richtung) Liniensegmenten im $\mathbb{R}^3$.
> **Gesucht:** Der minimale Abstand $t_{col}^{trans}(E_H, E_V)$ zwischen E_H und E_V in positiver x_3–Richtung.

Eine effiziente Lösung dieses geometrischen Problems könnte wie folgt aussehen:
Wir projizieren alle Liniensegmente in die x_1x_2–Ebene. Die resultierende Szene besteht aus sich kreuzenden horizontalen und vertikalen Liniensegmenten mit bis zu $|E_H| \cdot |E_V|$ vielen Kreuzungspunkten. Wir wollen den Kreuzungspunkt der beiden Liniensegmente ermitteln, die in positiver x_3–Richtung gesehen am dichtesten zusammenliegen. Dazu führen wir

einen Plane–Sweep in x_1–Richtung durch. Die horizontalen Liniensegmente, die die vertikale Sweep–Line momentan schneiden, werden in einem balancierten binären Suchbaum T_1 verwaltet. Ihre x_2–Koordinate dient als Ordnungskriterium. Die Sweep–Line muß zu Beginn und am Ende eines horizontalen Liniensegmentes anhalten, um den Suchbaum T_1 auf dem jeweils aktuellen Stand zu halten. Wenn sie auf ein vertikales Liniensegment l_v trifft, soll die x_2–Struktur in der Lage sein, das zu l_v in positiver x_3–Richtung nächstgelegene horizontale Liniensegment l_h zu benennen.

Mit Hilfe des Suchbaumes T_1 ist es leicht möglich, zu einem vertikalen Segment $l_v = ((x_1, a_2), (x_1, b_2))$ die Menge aller schneidenden horizontalen Segmente zu bestimmen. Diese Menge befindet sich im Suchbaum T_1 im Bereich zwischen den Suchpfaden für a_2 und b_2. Diese Suchpfade besitzen im allgemeinen ein gemeinsames Stück S, das sich aufspaltet in S_l und S_r. Setzen wir $a_2 < b_2$ voraus, so liegt die gesuchte Menge in den rechten Teilbäumen von S_l und den linken Teilbäumen von S_r. Sortieren wir alle in einem Teilbaum $T_1(u)$ mit Wurzel u liegenden horizontalen Segmente zusätzlich nach ihrer x_3–Koordinate, so können wir das vertikale Segment l_v durch binäre Suche schnell nach seinem x_3–Wert einordnen und somit das nächstgelegene horizontale Liniensegment in $T_1(u)$ bestimmen. Aus diesem Grund fügen wir dem Suchbaum T_1 eine entsprechende Sekundärstruktur T_2 bei. Zu jedem Teilbaum $T_1(u)$ existiert dann ein weiterer balancierter binärer Suchbaum $T_2(u)$, der alle in $T_1(u)$ enthaltenen horizontalen Liniensegmente nach x_3–Koordinate sortiert enthält. Der Platzbedarf für n Elemente steigt dadurch nur auf $O(n \log n)$. Die Suchzeit beträgt $\log^2 n$, weil wir mittels der Primärstruktur T_1 zunächst $O(\log n)$ viele Teilbäume $T_2(u_i)$ ermitteln müssen, in denen die Suchzeit nochmals $O(\log n)$ beträgt. Von all den Antworten, die wir von den $T_2(u_i)$'s bekommen, muß dann nur noch das Minimum gebildet werden.

Da die eben beschriebene Datenstruktur während des Plane–Sweeps eingesetzt wird, muß sie dynamisch sein. Einfügen und Löschen von Elementen ist in Zeit $O(\log^2 n)$ möglich: Sei $l_h = ((a_1, x_2), (b_1, x_2))$ ein horizontales Segment, das bei Position a_1 eingefügt bzw. bei Position b_1 gelöscht wird. Bei Updates treten Änderungen in der Primärstruktur T_1 entlang des Suchpfades S für x_2 auf. Also sind auch alle Teilbäume $T_2(u_i)$ der Sekundärstruktur betroffen, deren Wurzel u_i auf S liegt. Ein einzelnes Update bewirkt also logarithmisch viele Baumrebalancierungen mit Kosten $O(\log n)$.

Der gesamte Plane–Sweep Algorithmus hat also eine Laufzeit von $O(n \log^2 n)$ mit $n = |E_H| + |E_V|$. Wir erhalten das folgende

Lemma 3.4 *Der minimale Abstand $t_{col}^{trans}(E_1, E_2)$ zwischen den Kantenmengen zweier c–orientierter Polyeder P_1 und P_2 läßt sich in Zeit $O(c_1 c_2 \cdot (|E_1| + |E_2|) \log^2(|E_1| + |E_2|))$ bestimmen.*

3.2.2 Bestimmung von $t_{col}^{trans}(V_1, F_2)$

Nun können wir dazu übergehen, den minimalen Abstand in Translationsrichtung $\mathbf{s}$ zwischen der Menge der Eckpunkte V_1 und der Menge der Flächen F_2 zu bestimmen. Da die Kanten von P_2 nur c_2 verschiedene Orientierungen annehmen können, ist natürlich auch die Zahl der möglichen Flächennormalen durch eine Konstante $c_2' \leq \binom{c_2}{2}$ beschränkt. Dementsprechend

partitionieren wir auch die Menge F_2, als

$$F_2 = \bigcup_{i=1}^{c_2'} F_2^i.$$

Es gilt:

$$t_{col}^{trans}(V_1, F_2) = \min_{1 \le i \le c_2'} t_{col}^{trans}(V_1, F_2^i)$$

Betrachten wir also eine Teilmenge F_2^i mit fester Normalenrichtung $\mathbf{n}$. Durch eine einfache Transformation unserer Szene können wir $\mathbf{n} = \mathbf{s} = \mathbf{e_3}$ erreichen. Dazu drehen wir die Szene zunächst so, daß $\mathbf{s}$ mit der x_3–Richtung übereinstimmt. Daran schließen wir die Transformation $\mathbf{x} \mapsto (x_1, x_2, \frac{\mathbf{n}^T\mathbf{x}}{n_3})^T$ an. Durch diese Transformation bleiben die Abstände in x_3–Richtung zwischen den Flächen und den Punkten erhalten (siehe Gleichung (2.1)). Sei $\overline{\mathbf{x}} = (x_1, x_2)^T$ und $\overline{f} = \{\overline{\mathbf{x}} \,|\, \mathbf{x} \in f\}$, dann gilt:

$$\begin{aligned} f = \{\mathbf{x} \,|\, \mathbf{n}^T\mathbf{x} = n_0,\ \overline{\mathbf{x}} \in \overline{f}\} &\mapsto \{\mathbf{x} \,|\, x_3 = \frac{n_0}{n_3},\ \overline{\mathbf{x}} \in \overline{f}\} \\ \mathbf{p} &\mapsto (p_1, p_2, \frac{\mathbf{n}^T\mathbf{p}}{n_3})^T \\ t &= \frac{n_0}{n_3} - \frac{\mathbf{n}^T\mathbf{p}}{n_3} \end{aligned}$$

Auf diese Weise wird aus F_2^i eine Menge von Flächen, die parallel zur x_1x_2–Ebene liegen und deren Begrenzungskanten nur in $c_2^i \le c_2$ viele Richtungen verlaufen. Wir suchen den minimalen x_3–Abstand eines Punktes aus V_1 zu dieser Menge. Dazu führen wir einen Space–Sweep in negativer x_3–Richtung durch mit einer zur x_1x_2–Ebene parallelen Sweep–Plane. Als Invariante während des Sweeps soll gelten, daß sich alle Flächen oberhalb der Sweep–Plane in einer geeigneten Datenstruktur befinden. Wenn die Sweep–Plane an einem Punkt $\mathbf{p} \in V_1$ hält, ist diese Datenstruktur zu befragen, um die Fläche herauszufinden, auf die der Strahl $\mathbf{p} + \zeta\mathbf{e_3}$ ($\zeta > 0$) zuerst trifft. Obwohl die betrachteten Flächen c_2^i–orientiert sind, sind sie einer effizienten Verwaltung nicht unmittelbar zugänglich. Um dies zu verbessern, zerlegen wir die Flächen in Trapeze und Dreiecke wie folgt:
Von jedem Eckpunkt aus zerschneiden wir eine Fläche entlang der x_2–Richtung bis zur gegenüberliegenden Kante. Die Dreiecke sind als entartete Trapeze aufzufassen. Die entstehenden Trapeze lassen sich nach dem Verlauf von oberer und unterer Seite klassifizieren. Zwei Trapeze gehören zur gleichen Klasse genau dann, wenn ihre oberen Seiten zueinander parallel sind und auch ihre unteren Seiten. Auf diese Art und Weise können nur $(c_2^i)^2$ viele verschiedene Klassen entstehen. Jede einzelne Klasse wird separat behandelt.
Wir haben also noch folgende Aufgabe zu lösen:

> **Gegeben:** Eine Menge von Punkten V und eine Menge von Trapezen F im $\mathbb{R}^3$, die parallel zur x_1x_2–Ebene liegen und deren linke und rechte Seiten parallel zur x_2–Achse sind und deren untere und obere Seiten in festen Richtungen verlaufen.
> **Gesucht:** Der minimale Abstand zwischen V und F in positiver x_3–Richtung.

Zur Vereinfachung transformieren wir die Szene erst noch so, daß die unteren Trapezseiten in x_1–Richtung verlaufen. Dies können wir durch einen Basiswechsel erreichen.
Als zur Sweep–Plane adjungierte Datenstruktur benutzen wir primär einen Segmentbaum T_1 für die vorkommenden x_1–Intervalle. Diese sind vor Beginn des Sweeps bekannt, so daß das Gerüst von T_1 statisch vorberechnet werden kann. Das Einfügen der Trapeze in T_1 bewirkt deren Zerstückelung in logarithmisch viele Streifen. Jeden Streifen eines so zerschnittenen Trapezes unterteilen wir in ein achsenparalleles Rechteck und ein aufgesetztes Dreieck. Das erleichtert uns die Arbeit, weil alle in einem x_1–Bereich vorkommenden Dreiecke kongruent sind. Dies folgt aus der Parallelität der oberen Seiten der betrachteten Klasse von Trapezen.

Die Behandlung der Rechtecke: Für die x_2–Intervalle der Rechtecke, die in einem Knoten u von T_1 liegen, wird ein sekundärer Segmentbaum $T_2(u)$ angelegt, dessen Gerüst ebenfalls im voraus bestimmt werden kann. Ein Knoten von $T_2(u)$ muß lediglich den minimalen x_3–Wert der Rechtecke enthalten, die dort abgespeichert würden. Diese Konstruktion ermöglicht es, während des Sweeps das von dem Query–Strahl $\mathbf{p} + \zeta\mathbf{e_3}$ zuerst getroffene Rechteck zu bestimmen. Dazu bestimmt man in T_1 den Suchpfad S_1 für p_1 und für jeden Knoten u auf S_1 berechnet man in $T_2(u)$ das Minimum der x_3–Werte auf dem Suchpfad $S_2(u)$ für p_2. Suchen und dynamisches Einfügen in der Sekundärstruktur $T_2(u)$ erfordern nur $O(\log|F|)$ Zeit.

Die Behandlung der Dreiecke: Alle Dreiecke, die einem Knoten u mit dem x_1–Bereich $[l(u), r(u)]$ im Segmentbaum T_1 zugeordnet werden, sind kongruent. Ihre unteren Seiten verlaufen parallel zur x_1–Achse und ihre oberen Seiten seien parallel zur Gerade $x_2 = a_1x_1$. Die definierende Gerade für die obere Seite des i–ten Dreiecks sei $x_2 = a_1x_1 + a_{i,2}$, und es befinde sich in der Höhe $x_3 = a_{i,3}$. Dann können wir das i–te Dreieck wie folgt charakterisieren: $\{\mathbf{x}|l(u) \le x_1 \le r(u), a_1l(u) \le x_2 - a_{i,2} \le a_1x_1, x_3 = a_{i,3}\}$. Um das vom Query–Strahl $\mathbf{p}+\zeta\mathbf{e_3}$ erstgetroffene Dreieck zu bestimmen, haben wir für einen Knoten u mit $p_1 \in [l(u), r(u)]$ das folgende Minimum zu berechnen, wobei während des Space–Sweeps stets $p_3 \le a_{i,3}\ \forall i$ gilt:

$$\min_i\{a_{i,3} - p_3 | a_1l(u) \le p_2 - a_{i,2} \le a_1p_1\} = \min_i\{a_{i,3} - p_3 | p_2 - a_1p_1 \le a_{i,2} \le p_2 - a_1l(u)\}$$

Als 2–dimensionales Problem in der x_2x_3–Ebene formuliert, sucht man also unter all den Punkten $(a_{i,2}, a_{i,3})$ denjenigen mit minimaler x_3–Komponente, der im x_2–Bereich zwischen $l = p_2 - a_1p_1$ und $r = p_2 - a_1l(u)$ liegt. Zur Lösung dieses Problems eignet sich ein binärer balancierter Suchbaum $T_3(u)$ über den $a_{i,2}$'s, dessen Knoten den minimalen x_3–Wert der Elemente des zugehörigen Unterbaumes als Zusatzinformation enthalten. Für den vorgegebenen Bereich $[l, r]$ erhält man den minimalen x_3–Wert wie folgt: Man bestimmt die Suchpfade für l und r. Diese spalten sich irgendwo in S_l und S_r auf. Der gesuchte x_3–Wert ergibt sich als Minimum der x_3–Werte der rechten Söhne von S_l und der linken Söhne von S_r. Die Laufzeit für diese Suche in der Sekundärstruktur $T_3(u)$ läßt sich durch $O(\log|F|)$ abschätzen. Das gilt auch für das dynamische Einfügen neuer Elemente. ∎

In der Primärstruktur des Segmentbaumes T_1 gehört zu jedem Knoten u ein weiterer Segmentbaum $T_2(u)$ und ein binärer balancierter Suchbaum $T_3(u)$. Sie dienen zur Verwaltung der rechteckigen bzw. dreieckigen Anteile der Trapeze. Diese Kombination gewährleistet eine Antwortzeit von $O(\log^2|F|)$ auf eine Anfrage nach dem ersten Schnittpunkt eines Strahls $\mathbf{p} + \zeta\mathbf{e_3}$ mit der Menge der Trapeze. Die Primärstruktur T_1 leitet eine solche Anfrage an

höchstens $O(\log |F|)$ viele Suchstrukturen der zweiten Stufe weiter, die ihrerseits logarithmisches Suchen gestatten. Ähnlich verhält es sich mit dem dynamischen Einfügen weiterer Trapeze beim Fortschreiten der Sweep–Plane.
Berücksichtigen wir noch die vorbereitenden Berechnungen wie Sortieren der Haltepunkte der Sweep–Plane und Aufbau der Gerüste für die Segmentbäume, so erhalten wir insgesamt:

Lemma 3.5 *Der minimale Abstand $t_{col}^{trans}(V_1, F_2)$ zwischen der Eckenmenge und der Flächenmenge zweier c–orientierter Polyeder P_1 und P_2 läßt sich in Zeit $O(c_2^2\,(|V_1| + |F_2|)\log^2(|V_1| + |F_2|))$ bestimmen.*

Beweis: Noch unbewiesen ist die Abhängigkeit der Laufzeit von dem Parameter c_2. Zu Beginn mußten wir eine Klasseneinteilung der Flächen F_2 gemäß ihrer Normalen und der Richtungen ihrer Begrenzungskanten vornehmen. Es bleibt zu zeigen, daß nur $O((c_2)^2)$ verschiedene Klassen entstehen können. Die Zahl der Orientierungen für die Normalen ist durch $c_2' \leq \binom{c_2}{2}$ beschränkt. c_2^i bezeichnete die Zahl der Kantenrichtungen, die auf der i-ten Normalenrichtung senkrecht stehen. Die Anzahl der Klassen ist beschränkt durch die Größe $\sum_{i=1}^{c_2'}(c_2^i)^2$, weil für die i-te Normalenrichtung jedes Paar der c_2^i Kantenrichtungen eine Klasse hervorbringen kann. Eine Aufzählung aller Richtungspaare ergibt: $\sum_{i=1}^{c_2'}\binom{c_2^i}{2} = \binom{c_2}{2}$. Weil $c_2^i \geq 2$ gilt, folgt daraus:

$$\sum_{i=1}^{c_2'}(c_2^i)^2 \leq \sum_{i=0}^{c_2'} 2c_2^i(c_2^i - 1) \leq 2(c_2)^2$$

■

Zusammenfassend erhalten wir:

Satz 3.6 *Der minimale Abstand $t_{col}^{trans}(P_1, P_2)$ zweier c–orientierter Polyeder P_1 und P_2 läßt sich für eine feste Richtung* **s** *in Zeit $O((c_1 + c_2)^2 \cdot (|P_1| + |P_2|)\log^2(|P_1| + |P_2|))$ berechnen.*

Bemerkung:
Das translatorische Kollisionsproblem ist eng verwandt mit dem Problem, die sichtbaren Flächen einer Szene für eine vorgegebene Blickrichtung zu berechnen. Beschränkt man sich auf c–orientierte Szenen, so kann man auch hier Laufzeitgewinne gegenüber dem allgemeinen Fall erzielen (siehe [Be92]).

3.3 Ein subquadratischer Kollisionstest für die Translation beliebiger Polyeder

In diesem Abschnitt gehen wir von zwei beliebigen Polyedern P_1 und P_2 aus und treffen keine Annahmen über spezielle Eigenschaften wie Konvexität (siehe Abschnitt 3.1) oder isoorientierte Begrenzungskanten (siehe Abschnitt 3.2). Die Gesamtkomplexität der beiden Polyeder sei $n := |P_1| + |P_2|$. P_1 führe eine Verschiebung in eine feste Richtung oder eine Rotation um eine feste Achse aus, P_2 ruhe währenddessen. Es stellt sich die Frage, ob es prinzipiell möglich ist, einen Algorithmus für die Kollisionserkennung zu entwerfen, dessen Laufzeit sich asymptotisch besser verhält als die des naiven Algorithmus, der $O(n^2)$ Zeit

benötigt. Für den Fall der Translation werden wir zeigen, daß n^2 keine untere Schranke für dieses Problem darstellt. Für Rotationen bleibt diese Frage weiterhin offen.
Bereits in Lemma 1.3 haben wir gesehen, daß die Erkennung einer Kollision zwischen den Ecken des einen Polyeders und den Flächen des anderen Polyeders weniger Schwierigkeiten bereitet als die Erkennung einer Kollision zwischen den Kanten der Polyeder. Betrachten wir zu letzterem Fall eine einzelne translatorisch bewegte Kante e und ignorieren alle stationären Kanten, mit denen e bei der vorgegebenen Translationsrichtung überhaupt nicht kollidieren kann, so macht es keinen Unterschied, ob wir die Kanten als Liniensegmente oder als Geraden ansehen. Das Problem, eine Kollision zwischen einer translatorisch bewegten Gerade und einer Menge von stationären Geraden zu erkennen, läßt sich mit Hilfe von Plückerkoordinaten auf ein Strahlverfolgungsproblem in einem höherdimensionalen Arrangement von Hyperebenen reduzieren. Durch eine geeignete Vorverarbeitung dieses Arrangements ist es möglich, die vom Strahl zuerst getroffene Hyperebene in sublinearer Zeit zu bestimmen und damit die Gerade zu finden, mit der die bewegte Gerade zuerst zusammenstößt. Diesen Test führt man für jede bewegte Kante durch und erhält so einen Algorithmus mit subquadratischer Laufzeit für die Kollisionserkennung bei translatorisch bewegten Polyedern.

Anstelle der Polyeder können wir auch einfach zwei Mengen von Dreiecken benutzen um die Problemstellung zu formulieren.

> **Gegeben:** Eine Menge F_1 von roten Dreiecken, die in x_3–Richtung verschoben werden, und eine Menge F_2 von blauen stationären Dreiecken im $\mathbb{R}^3$.
> **Gesucht:** Die maximale Verschiebung $t_{col}^{trans}(F_1, F_2)$ von F_1 in x_3–Richtung bis zur ersten Berührung mit F_2.

Der naive Algorithmus zur Lösung dieses Problems hat eine Laufzeit von $O(|F_1| \cdot |F_2|)$, wenn man alle rot–blauen Dreieckpaare betrachtet. Es stellt sich jedoch die Frage, ob es einen Algorithmus gibt, dessen Komplexität asymptotisch besser ist als das Quadrat der Größe der Eingabe. Wir werden nun folgenden Satz beweisen:

Satz 3.7 *$t_{col}^{trans}(F_1, F_2)$ für zwei Mengen F_1 und F_2 von roten und blauen Dreiecken im $\mathbb{R}^3$ kann in Zeit $o((|F_1| + |F_2|)^2)$ berechnet werden.*

Mit den Bezeichnungen E_1 (E_2) und V_1 (V_2) für die roten (blauen) Begrenzungskanten und die roten (blauen) Eckpunkte gilt:

$$t_{col}(F_1, F_2) = \min\{\min_{v \in V_1} t_{col}(v, F_2), \min_{v \in V_2} t_{col}(F_1, v), \min_{e \in E_1} t_{col}(e, E_2)\}$$

Die generelle Vorgehensweise sieht so aus: F_1 und F_2 werden separat vorverarbeitet, so daß man für einen beliebigen Punkt v die gesuchte Distanz zu F_1 bzw. F_2 schnell erfragen kann. Ebenso verfahren wir mit E_2, damit man für eine beliebige Kante e die Distanz zu E_2 effizient berechnen kann.

3.3.1 Laufzeitersparnis durch Vorverarbeitung

Wir betrachten die vorliegende Problemstellung in einer allgemeineren Formulierung:

Gegeben sei eine Funktion $f: U_A \times U_B \to H$, die jedem Paar (a, b) aus einem Universum $U_A \times U_B$ ein Element einer Halbgruppe H zuordnet. Mit Hilfe der assoziativen Verknüpfung $\odot$ der Halbgruppe sei die Abbildung f auf Teilmengen $A \subseteq U_A$, $B \subseteq U_B$ fortgesetzt:

$$f(A, B) \quad = \quad \bigodot_{a \in A, b \in B} f(a, b)$$

(Für unser konkretes Problem gilt: $f = t_{col}^{trans}$, $\odot = \min$ und $A \times B = V_1 \times F_2$ oder $F_1 \times V_2$ oder $E_1 \times E_2$) Wenn wir annehmen, daß eine Funktionsauswertung $f(a, b)$ und die Operation $\odot$ in konstanter Zeit berechnet werden können, so belaufen sich die Kosten zur Bestimmung von $f(A, B)$ auf $O(|A| \cdot |B|) = O((|A| + |B|)^2)$, welche im schlechtesten Fall quadratisch in der Problemgröße $|A| + |B|$ wachsen. Unter zusätzlichen Bedingungen ist aber eine kostengünstigere Auswertung des Funktionswertes $f(A, B)$ denkbar. Das ist zum Beispiel der Fall, wenn sich die Menge B effizient so vorverarbeiten läßt, daß $f(a, B)$ für ein beliebiges $a \in U_A$ schnell berechenbar ist. Das Preprocessing für eine n–elementige Teilmenge $B \subseteq U_B$ koste $O(n^p)$ und die Zeit für eine Anfrage eines Elementes $a \in U_A$ an diese vorverarbeitete Menge zur Berechnung von $f(a, B)$ sei durch $O(\log^q n)$ beschränkt.
Da wir zur Berechnung von $f(A, B)$ mit subquadratischer Zeit auskommen wollen, können wir es uns nicht erlauben, die Menge B als Ganzes zu bearbeiten, falls $p \geq 2$ ist. Wir müssen sie vielmehr in kleinere Teile zerlegen und anschließend das Ergebnis einer Anfrage aus den Teilantworten kombinieren. Wir unterteilen B in k disjunkte Mengen gleicher Mächtigkeit:

$$B = \bigcup_{i=1}^{k} B_i \quad \text{mit} \quad |B_i| \approx \frac{1}{k}|B|$$

$$\text{Es gilt:} \qquad f(A, B) \quad = \quad \bigodot_{a \in A} \bigodot_{i=1}^{k} f(a, B_i).$$

Bei einer gleichmäßigen Aufteilung von B in k Teilmengen verteilen sich die Kosten für Preprocessing und Query–Zeiten wie folgt:

$$\text{Preprocessing:} \quad O\left(k \left(\frac{|B|}{k}\right)^p\right) \qquad \text{Queries:} \quad O\left(|A|\, k \log^q \left(\frac{|B|}{k}\right)\right)$$

Um diese beiden Anteile zu balancieren, wählen wir $k = |B|\,|A|^{-1/p} \log^{-q/p} |B| \geq 1$ und erhalten so Gesamtkosten in Höhe von

$$O\left(|B|\,|A|^{1-1/p} \log^{q(1-1/p)} |B|\right) = O\left((|A| + |B|)^{2-\varepsilon}\right).$$

Falls $k < 1$ ist, dann ist $|B| < |A|^{1/p} \log^{q/p} |B|$ und der naive Algorithmus leistet bereits das Gewünschte.

3.3.2 Bestimmung von $t_{col}^{trans}(v, F)$

Widmen wir uns zunächst der Aufgabe, für eine Menge F von Dreiecken eine Datenstruktur aufzubauen, mit deren Hilfe für jeden beliebigen Punkt $\mathbf{v}$ die maximale Verschiebung $t_{col}^{trans}(v, F)$ in logarithmischer Zeit berechnet werden kann (vgl. Abschnitt 1.6.4).

Die Frage nach der Bestimmung von $t_{col}^{trans}(v, F)$ für einen beliebigen Punkt $\mathbf{v}$ führt uns zu der Problemstellung, das erste Dreieck zu finden, das der Strahl $r_v = \{(v_1, v_2, v_3+\zeta)^T \mid \zeta \geq 0\}$ trifft. Dazu projizieren wir alle Dreiecke aus F in die x_1x_2–Ebene. Dort bilden die projizierten Begrenzungskanten $\overline{E}$ ein Arrangement von Liniensegmenten im $\mathbb{R}^2$. Dieses Arrangement besitzt $O(|F|^2)$ elementare Regionen. Für alle Punkte $\overline{\mathbf{v}} = (v_1, v_2)^T$ einer solchen Region $\overline{R}$ gilt, daß alle Geraden $L_v = \{(v_1, v_2, \zeta) \mid \zeta \in \mathbb{R}\}$ die Dreiecke aus F in der gleichen Reihenfolge schneiden. Zu jeder Region können wir uns deshalb eine nach der x_3–Koordinate sortierte Liste der Dreiecke speichern. Durch Binärsuche nach v_3 können wir dann für den Punkt $\mathbf{v}$ das in x_3–Richtung nächstgelegene Dreieck finden, sobald wir die Region $\overline{R}$ kennen, die die Projektion $\overline{\mathbf{v}}$ des Punktes $\mathbf{v}$ enthält.

Bei der Lösung des verbleibenden 2–dimensionalen Point–Location–Problems legen wir keinen großen Wert auf die Optimierung der Preprocessing–Kosten, sondern wir benutzen das klassische Verfahren von Dobkin und Lipton [DL76]. Es wird uns später noch einmal in einem Raum höherer Dimension begegnen. Wegen seiner zentralen Bedeutung für den Gesamtalgorithmus skizzieren wir es kurz für den 2–dimensionalen Fall:

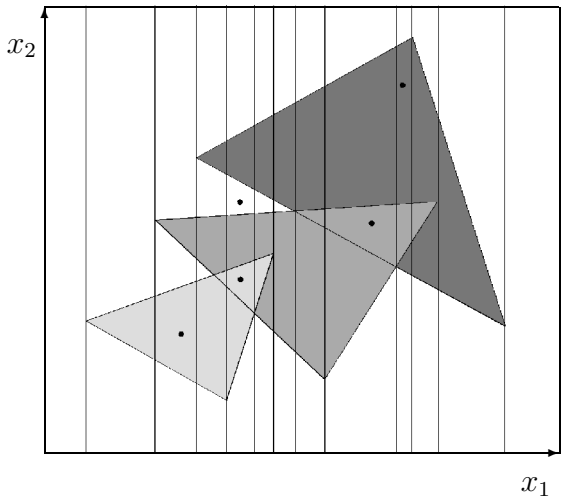


Abbildung 3.4: Point–Location im $\mathbb{R}^2$

Unser Arrangement wird von $|\overline{E}|$ Liniensegmenten in der Ebene gebildet. Durch alle Randpunkte und durch ihre möglicherweise $\binom{|\overline{E}|}{2}$ Schnittpunkte legen wir Geraden parallel zur x_2–Achse und erhalten dadurch eine Streifeneinteilung. Diese impliziert auch eine Verfeinerung der elementaren Regionen in Trapeze und Dreiecke. Innerhalb jedes Streifens gilt, daß jede zur x_2–Achse parallele Gerade die Liniensegmente in der gleichen Reihenfolge schneidet. Diese Eigenschaft erlaubt pro Streifen eine Sortierung der Segmente nach x_2. Die Streifen selbst können natürlich nach x_1 sortiert werden. Zur Lokalisierung eines Punktes $\overline{\mathbf{v}} = (v_1, v_2)^T$ bestimmt man zuerst durch Binärsuche nach v_1 in der Streifenliste den entsprechenden Streifen. Zu jedem Streifen gehört eine nach x_2 sortierte Liste der kreuzenden Liniensegmente, so daß eine Binärsuche nach v_2 das Trapez/Dreieck und somit die Region liefert, die $\overline{v}$ enthält.

Die Vorverarbeitungskosten sind sicherlich polynomiell und lassen sich grob durch $O(|F|^4$

$\log |F|$) abschätzen. (Es gibt $O(|F|^3)$ Trapeze/Dreiecke, wofür eine nach x_3–Koordinate sortierte Liste der $|F|$ Flächen zu bestimmen ist.) Die Suchzeit beträgt $O(\log |F|)$.

3.3.3 Bestimmung von $t_{col}^{trans}(e, E)$

Gegeben sei eine Menge E von Liniensegmenten im $\mathbb{R}^3$. Wir möchten diese Menge so vorverarbeiten, daß wir für ein beliebig vorgegebenes Segment e das Segment aus E schnell bestimmen können, mit dem e zuerst kollidiert, wenn es in x_3–Richtung verschoben wird. Um eine logarithmische Antwortzeit zu erzielen, nehmen wir polynomielle Vorverarbeitungskosten in Kauf.

Unsere Vorgehensweise ist zweistufig: Bei vorgegebenem e ermitteln wir zunächst die Menge $E(e) \subseteq E$ all der Liniensegmente, mit denen e während einer Translation in x_3–Richtung zusammenstoßen kann. Anschließend bestimmen wir unter diesen dasjenige, das zur ersten Kollision führt.

1. Stufe

Fragen wir uns zunächst einmal, unter welchen Bedingungen ein translatorisch bewegtes Liniensegment l_{ab} mit einem Liniensegment l_{cd} zusammenstoßen kann. Wir werden nur *echte* Zusammenstöße betrachten, d.h:

- Die beiden Liniensegmente l_{ab} und l_{cd} sind nicht parallel.
- Keiner der Endpunkte $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ ist ein Kollisionspunkt.

Eine Antwort auf diese Frage haben wir bereits mit Formel (2.6) gegeben. Für $\mathbf{s} = \mathbf{e_3}$ ergibt die Expansion der Determinanten die beiden folgenden Ungleichungssysteme.

$$\begin{aligned}
(\mathbf{a} \times \mathbf{b})_3 + (\mathbf{b} \times \mathbf{c})_3 + (\mathbf{c} \times \mathbf{a})_3 &< 0 \; (> 0) \\
(\mathbf{b} \times \mathbf{a})_3 + (\mathbf{a} \times \mathbf{d})_3 + (\mathbf{d} \times \mathbf{b})_3 &< 0 \; (> 0) \\
(\mathbf{c} \times \mathbf{d})_3 + (\mathbf{d} \times \mathbf{b})_3 + (\mathbf{b} \times \mathbf{c})_3 &< 0 \; (> 0) \\
(\mathbf{d} \times \mathbf{c})_3 + (\mathbf{c} \times \mathbf{a})_3 + (\mathbf{a} \times \mathbf{d})_3 &< 0 \; (> 0)
\end{aligned}$$

Wenn die Endpunkte der beiden Linensegmente l_{ab} und l_{cd} eines der beiden Ungleichungssysteme erfüllen, ist gewährleistet, daß sich ihre Projektionen in der x_1x_2–Ebene schneiden, d.h: $\overline{l}_{ab} \cap \overline{l}_{cd} \neq \emptyset$. Dies ist gleichbedeutend damit, daß eine Translation eines Liniensegments entweder in positiver oder negativer x_3–Richtung zu einer Kollision führt. Wegen des Terms $(\mathbf{a} \times \mathbf{b})_3$ ist dieses System jedoch nicht linear in den kartesischen Koordinaten von $\mathbf{a}$ und $\mathbf{b}$. Deshalb führen wir neue Koordinaten ein:

$$x_1 = a_1, \quad x_2 = a_2, \quad x_3 = b_1, \quad x_4 = b_2, \quad x_5 = a_1b_2 - a_2b_1$$

Da $x_5 = x_1x_4 - x_2x_3$ gilt, sind diese Koordinaten nicht unabhängig voneinander. Außerdem wird durch die Einführung von x_5 die Dimension des Problems erhöht. Der entscheidende

Vorteil ist jedoch die resultierende Linearisierung der Ungleichungen.

$$\begin{aligned} -c_2x_1 + c_1x_2 + c_2x_3 - c_1x_4 + x_5 &< 0\ (>0) \\ (c_2 - d_2)x_3 - (c_1 - d_1)x_4 + c_1d_2 - c_2d_1 &< 0\ (>0) \\ d_2x_1 - d_1x_2 - d_2x_3 + d_1x_4 - x_5 &< 0\ (>0) \\ -(c_2 - d_2)x_1 + (c_1 - d_1)x_2 - c_1d_2 + c_2d_1 &< 0\ (>0) \end{aligned}$$

Wenn wir das Liniensegment l_{ab} als einen Punkt $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)^T$ im 5-dimensionalen Raum auffassen, dann beschreiben die beiden Ungleichungssysteme gerade zwei zusammenhängende Raumgebiete $R_<(l_{cd})$ und $R_>(l_{cd})$, in denen $\mathbf{x}$ liegen muß, damit l_{ab} bei einer Translation in x_3–Richtung mit dem Liniensegment l_{cd} zusammenstößt. Die Begrenzungen von $R_<(l_{cd})$ und $R_>(l_{cd})$ sind vier Hyperebenen, deren Koeffizienten nur von $\mathbf{c}$ und $\mathbf{d}$ abhängen.

Nun untersuchen wir die Situation, daß wir nicht nur ein einzelnes sondern mehrere stationäre Liniensegmente haben. Für ein beliebig vorgegebenes, in x_3–Richtung verschiebbares Segment l_{ab} möchten wir die Menge

$$E(l_{ab}) = \{e \in E \mid \overline{e} \cap \overline{l}_{ab} \neq \emptyset\}$$

bestimmen. Der Querstrich steht für die Projektion in die x_1x_2–Ebene. Mit Erreichen dieses Zwischenziels kennen wir alle Elemente aus E, die für eine Kollision überhaupt in Frage kommen. Im 5-dimensionalen Raum liefert uns jedes Liniensegment $e \in E$ vier Hyperebenen als Begrenzungsflächen der Kollisionsbereiche $R_<(e)$ und $R_>(e)$. In dem Arrangement $\mathcal{K}(E)$, das von diesen $4|E|$ Hyperebenen gebildet wird, können wir zu jeder 5-dimensionalen Zelle C all die Liniensegmente $E(C)$ bestimmen, deren Kollisionsbereiche C überdecken.

$$E(C) = \{e \in E \mid C \subset R_<(e) \cup R_>(e)\}$$

Die gesuchte Menge $E(l_{ab}) = E(C_{\mathbf{x}})$ erhalten wir, indem wir die Zelle $C_{\mathbf{x}}$ des Arrangements $\mathcal{K}(E)$ bestimmen, die den Punkt $\mathbf{x} = (a_1, a_2, b_1, b_2, a_1b_2 - a_2b_1)^T$ enthält. Sollte der Punkt $\mathbf{x}$ auf dem Rand einer Zelle $C_{\mathbf{x}}$ liegen, so können wir sicher sein, daß kein echter Zusammenstoß vorliegt.

Das Arrangement $\mathcal{K}(E)$ kann in polynomieller Zeit so vorverarbeitet werden, daß die Lokalisierung eines Punktes $\mathbf{x}$ und damit die Bestimmung der gesuchten Menge $E(l_{ab})$ in logarithmischer Zeit möglich ist. Auf diesen Punkt werden wir später noch genauer eingehen.

2. Stufe

Nach Abschluß der 1. Stufe haben wir alle Liniensegmente $E(l_{ab})$ zur Verfügung, die mit l_{ab} bei einer Translation in x_3–Richtung kollidieren können. Wir möchten das Segment $e \in E(l_{ab})$ bestimmen, mit dem l_{ab} zuerst zusammenstößt. Dazu können wir alle beteiligten Liniensegmente zu Geraden ausdehnen, ohne das Endergebnis zu verändern.

Betrachten wir also zwei Geraden L_{ab} und L_{cd} durch die Punkte $\mathbf{a}$, $\mathbf{b}$ und $\mathbf{c}$, $\mathbf{d}$. Uns interessiert der Moment, in dem $L_{ab} \cap L_{cd} \neq \emptyset$ gilt. Dann sind alle vier Punkte komplanar. In homogenen

Koordinaten ausgedrückt bedeutet dies (siehe auch Gleichung (2.5)):

$$\det \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{bmatrix} = 0$$

Eine Entwicklung dieser 4×4–Determinante nach den 2×2–Unterdeterminanten der ersten beiden und letzten beiden Zeilen ergibt folgende homogene Gleichung:

$$\gamma_{23}\alpha_{01} + \gamma_{31}\alpha_{02} + \gamma_{12}\alpha_{03} + \gamma_{03}\alpha_{12} + \gamma_{01}\alpha_{23} + \gamma_{02}\alpha_{31} = 0$$
$$\text{mit } \alpha_{ij} = a_i b_j - a_j b_i \text{ und } \gamma_{ij} = c_i d_j - c_j d_i$$

Mit dieser bilinearen Gleichung kann man die Kollision der beiden Geraden L_{ab} und L_{cd} im $\mathbb{R}^3$ als die Kollision eines Punktes $\mathbf{p}_{ab}$ mit einer Hyperebene H_{cd} im $\mathbb{R}^6$ interpretieren, wobei $\mathbf{p}_{ab}$ und H_{cd} gegeben sind durch:

$$H_{cd} : \gamma_{23}x_1 + \gamma_{31}x_2 + \gamma_{12}x_3 + \gamma_{03}x_4 + \gamma_{01}x_5 + \gamma_{02}x_6 = 0$$
$$\mathbf{p}_{ab} : (x_1, x_2, x_3, x_4, x_5, x_6)^T = (\alpha_{01}, \alpha_{02}, \alpha_{03}, \alpha_{12}, \alpha_{23}, \alpha_{31})^T$$

Die Parameter α_{ij} und γ_{ij} sind unter dem Namen *Plückerkoordinaten* bzw. *Plückerkoeffizienten* bekannt. (Die Rollen von $\mathbf{p}_{ab}$ und H_{cd} können auch vertauscht werden.)

Wenn die Gerade L_{ab} in Richtung $\mathbf{e_3}$ bewegt wird, stellt sich die Frage, wie sich dabei ihre Plückerkoordinaten verändern. Aus den 2×2 Unterdeterminanten der Matrix

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 + \zeta\, a_0 \\ b_0 & b_1 & b_2 & b_3 + \zeta\, b_0 \end{bmatrix}$$

ergeben sich die Plückerkoordinaten $\alpha_{ij}(\zeta)$ als

$$\mathbf{p}_{ab}(\zeta) \;=\; (\alpha_{01}, \alpha_{02}, \alpha_{03}, \alpha_{12}, \alpha_{23} - \zeta\,\alpha_{02}, \alpha_{31} + \zeta\,\alpha_{01})^T. \tag{3.1}$$

Das heißt, daß sich der Plückerpunkt $\mathbf{p}_{ab}$ geradlinig im $\mathbb{R}^6$ bewegt, wenn die zugehörige Gerade L_{ab} translatorisch bewegt wird.

Nach dieser Transformation des Problems in den 6–dimensionalen Plückerraum stehen wir vor folgender Aufgabe:

Die Menge $E(l_{ab})$ der ortsfesten Liniensegmente werde abgebildet auf eine Menge von Hyperebenen, die im Plückerraum ein Arrangement $\mathcal{P}(E(l_{ab}))$ bilden. Der zum bewegten Liniensegment l_{ab} gehörige Plückerpunkt bewegt sich auf einem Strahl $\mathbf{p}_{ab}(\zeta)$. Gesucht ist die erste Hyperebene H_{cd}, die dieser Strahl durchdringt. Die Antwort muß in logarithmischer Zeit gefunden werden, und die Vorverarbeitungskosten für $\mathcal{P}(E(l_{ab}))$ dürfen nur polynomiell sein.

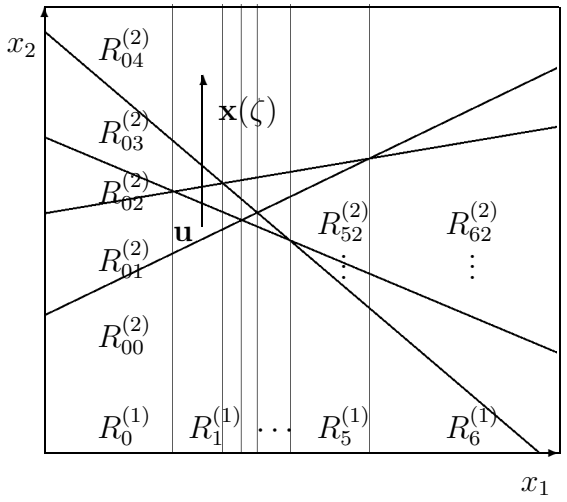


Abbildung 3.5: Regionenaufteilung in einem 2–dimensionalen Arrangement

3.3.4 Point–Location im $\mathbb{R}^d$

Wir betrachten zunächst das einfachere Problem, die Zelle eines Arrangements zu bestimmen, die einen vorgegebenen Punkt enthält. Das klassische Verfahren dafür stammt von Dobkin und Lipton [DL76]. Es bildet die Basis für die Lösung unseres Problems. Bei der Diskussion des 2–dimensionalen Falls haben wir bereits eine Einsicht in die prinzipielle Vorgehensweise erhalten (vgl. Abbildung 3.5). Die benutzte Methode der Dimensionsreduktion durch Projektion läßt sich in verallgemeinerter Form auch im d–dimensionalen Raum anwenden:
Gegeben sei ein Arrangement $\mathcal{P}^{(d)}$ von $n = n_d$ Hyperebenen $\mathcal{H}^{(d)}$ im $\mathbb{R}^d$. Der paarweise Schnitt dieser $(d–1)$–dimensionalen Hyperebenen liefert uns $n_{d-1} = \binom{n_d}{2}$ $(d–2)$–dimensionale Teilräume, die wir in den $x_1x_2\ldots x_{d-1}$–Raum projizieren. In diesem $(d-1)$–dimensionalen Raum erhalten wir so ein Arrangement $\mathcal{P}^{(d-1)}$ von n_{d-1} $(d-2)$–dimensionalen Hyperebenen. Alle Geraden $L_u = \{(u_1, u_2, \ldots, u_{d-1}, \zeta)^T | \zeta \in \mathbb{R}\}$ parallel zur x_d–Achse schneiden die ursprünglichen Hyperebenen in der gleichen Reihenfolge, solange sich die Projektion $\mathbf{u}^{(d-1)} = (u_1, u_2, \ldots, u_{d-1})^T$ von $\mathbf{u}$ in ein und derselben Zelle des Arrangements $\mathcal{P}^{(d-1)}$ befindet. Diese Invariante gestattet pro Zelle eine Sortierung der Hyperebenen nach x_d. Dadurch ist die Dimension des Ausgangsproblems um 1 vermindert, und wir können dieses Verfahren für $\mathcal{P}^{(d-1)}$ rekursiv fortsetzen. Zur Lokalisierung eines Punktes $\mathbf{u} = (u_1, u_2, \ldots, u_d)^T$ ermitteln wir rekursiv die Zelle von $\mathcal{P}^{(d-1)}$, die $\mathbf{u}^{(d-1)}$ enthält, und können dann in der zu dieser Zelle adjungierten Liste der Hyperebenen nach u_d binär suchen.
Wir starten also mit einer Menge $\mathcal{H}^{(d)} = \{H_1^{(d)}, H_2^{(d)}, \ldots H_{n_d}^{(d)}\}$ von Hyperebenen im $\mathbb{R}^d$. Für jedes Paar von Hyperebenen bilden wir den Schnitt und projizieren das Ergebnis in den $x_1x_2\ldots x_{d-1}$–Raum. Diesen Vorgang setzen wir rekursiv fort bis der resultierende Raum 1–dimensional geworden ist.
Für $k = d-1, \ldots, 1$ bestimmen wir

$$\begin{aligned}\mathcal{H}^{(k)} &= \{H_1^{(k)}, H_2^{(k)}, \ldots H_{n_k}^{(k)}\} \\ &= \{\overline{H_i \cap H_j}^{x_1x_2\ldots x_k} \mid 1 \le i < j \le n_{k+1},\ H_i, H_j \in \mathcal{H}^{(k+1)}\},\end{aligned}$$

$$H_i^{(k)} \quad : \quad \sum_{j=1}^{k} h_{ij}^{(k)} x_j = h_{i0}^{(k)}, \qquad\qquad n_k = \binom{n_{k+1}}{2}.$$

Jede Hyperebene $H_i^{(k)}$ unterteilt den Raum in zwei Halbräume ${}^{+}H_i^{(k)}$ und ${}^{-}H_i^{(k)}$. Um diese zu beschreiben, fassen wir die Hyperebene als eine Abbildung $f_i^{(k)} : \mathbb{R}^d \to \mathbb{R}$ auf. Wenn wir die Gleichung für $H_i^{(k)}$ nach x_k auflösen, können wir für $h_{ik}^{(k)} \neq 0$ und $\mathbf{x} \in \mathbb{R}^d$ folgendes definieren:

$$\begin{aligned} f_i^{(k)}(\mathbf{x}) \quad &:= \quad \frac{1}{h_{ik}^{(k)}} \Big(h_{i0}^{(k)} - \sum_{j=1}^{k-1} h_{ij}^{(k)} x_j \Big) \\ {}^{+}H_i^{(k)} := \{\mathbf{x} \,|\, x_k \geq f_i^{(k)}(\mathbf{x})\} \qquad & {}^{-}H_i^{(k)} := \{\mathbf{x} \,|\, x_k \leq f_i^{(k)}(\mathbf{x})\}. \end{aligned}$$

Wenn die Rekursion bei $k = 1$ stoppt, läßt sich die Menge der Hyperebenen $\mathcal{H}^{(1)}$ nach der x_1–Koordinate sortieren und zwar so, daß für alle $\mathbf{x} \in \mathbb{R}^d$ gilt:

$$-\infty < f_{\sigma(1)}^{(1)}(\mathbf{x}) \leq f_{\sigma(2)}^{(1)}(\mathbf{x}) \leq \ldots \leq f_{\sigma(n_1)}^{(1)}(\mathbf{x}) < \infty$$

σ beschreibt die Permutation der Hyperebenen nach dem Sortieren. Auf diese Weise erhalten wir eine Zerlegung des $\mathbb{R}^d$ in Regionen

$$R_{i_1}^{(1)} \quad = \quad \begin{cases} {}^{-}H_{\sigma(i_1+1)}^{(1)} & \text{für} \quad i_1 = 0, \\ {}^{+}H_{\sigma(i_1)}^{(1)} \cap {}^{-}H_{\sigma(i_1+1)}^{(1)} & \text{für} \quad 0 < i_1 < n_1, \\ {}^{+}H_{\sigma(i_1)}^{(1)} & \text{für} \quad i_1 = n_1. \end{cases}$$

Innerhalb jeder Region $R_{i_1}^{(1)}$ können wir die Hyperebenen aus $\mathcal{H}^{(2)}$ nach der x_2–Koordinate sortieren, wodurch wir eine erneute Zerlegung erhalten. Jede entstehende Region kann nach dem gleichen Verfahren rekursiv verfeinert werden. Die Sortierung der Hyperebenen aus $\mathcal{H}^{(k)}$ nach der x_k–Koordinate für eine Region $R_{i_1,i_2,\ldots,i_{k-1}}^{(k-1)}$ erfassen wir durch $\sigma(i_1, i_2, \ldots, i_k)$ für $1 \leq i_k \leq n_k$. Dann gilt für alle $\mathbf{x} \in R_{i_1,\ldots,i_{k-1}}^{(k-1)} \subset \mathbb{R}^d$:

$$-\infty < f_{\sigma(i_1,\ldots,i_{k-1},1)}^{(k)}(\mathbf{x}) \leq f_{\sigma(i_1,\ldots,i_{k-1},2)}^{(k)}(\mathbf{x}) \leq \ldots \leq f_{\sigma(i_1,\ldots,i_{k-1},n_k)}^{(k)}(\mathbf{x}) < \infty$$

Für $0 \leq i_j \leq n_j$, $j = 1, \ldots, k-1$, ist die Partitionierung der Region $R_{i_1,\ldots,i_{k-1}}^{(k-1)}$ gegeben durch

$$R_{i_1,\ldots,i_k}^{(k)} \quad = \quad \begin{cases} R_{i_1,\ldots,i_{k-1}}^{(k-1)} \cap {}^{-}H_{\sigma(i_1,\ldots,i_k+1)}^{(k)} & \text{für } i_k = 0, \\ R_{i_1,\ldots,i_{k-1}}^{(k-1)} \cap {}^{+}H_{\sigma(i_1,\ldots,i_k)}^{(k)} \cap {}^{-}H_{\sigma(i_1,\ldots,i_k+1)}^{(k)} & \text{für } 0 < i_k < n_k, \\ R_{i_1,\ldots,i_{k-1}}^{(k-1)} \cap {}^{+}H_{\sigma(i_1,\ldots,i_k)}^{(k)} & \text{für } i_k = n_k. \end{cases}$$

Für einen beliebig vorgegebenen Punkt $\mathbf{u} \in \mathbb{R}^d$ können wir durch d–malige binäre Suche die Region $R_{i_1,\ldots,i_d}^{(d)}$ bestimmen, in der $\mathbf{u}$ liegt.

Unter der Annahme, daß $\mathbf{u} = (u_1, \ldots, u_d)^T \in R^{(k-1)}_{i_1,\ldots,i_{k-1}}$ ist, bestimme i_k sukzessive für $k = 1, \ldots, d$ so, daß gilt:

$$f^{(k)}_{\sigma(i_1,\ldots,i_{k-1},i_k)}(\mathbf{u}) \leq u_k \leq f^{(k)}_{\sigma(i_1,\ldots,i_{k-1},i_k+1)}(\mathbf{u})$$

Wenn wir die Region $R^{(d)}_{i_1,\ldots,i_d}$, die $\mathbf{u}$ enthält, gefunden haben, kennen wir damit auch die Hyperebene, die von dem Strahl $x(\zeta) = (u_1, \ldots, u_d + \zeta)^T$, $\zeta \geq 0$, zuerst getroffen wird.

Korrektheitsbeweis

Seien $H_a : \sum_{i=1}^{d} a_i x_i = a_0$ und $H_b : \sum_{i=1}^{d} b_i x_i = b_0$ zwei Hyperebenen im $\mathbb{R}^d$. Die Projektion des Schnittes $H_a \cap H_b$ in die $x_1 x_2 \ldots x_{d-1}$–Ebene ist gemäß (A.15) gegeben durch:

$$H_{ab} : \sum_{i=1}^{d-1} \det \begin{bmatrix} a_i & a_d \\ b_i & b_d \end{bmatrix} x_i \;=\; \det \begin{bmatrix} a_0 & a_d \\ b_0 & b_d \end{bmatrix}$$

Zur Korrektheit genügt es zu zeigen, daß zwei beliebige zur x_d–Achse parallele Geraden $L_u = \{(u_1, u_2, \ldots, u_{d-1}, \zeta)^T | \zeta \in \mathbb{R}\}$ und $L_v = \{(v_1, v_2, \ldots, v_{d-1}, \zeta)^T | \zeta \in \mathbb{R}\}$ die Hyperebenen in der gleichen Reihenfolge schneiden, wenn $\mathbf{u}^{(d-1)}$ und $\mathbf{v}^{(d-1)}$ in der gleichen Zelle des Arrangements $\mathcal{P}^{(d-1)}$ liegen. Wir führen den Beweis indirekt, indem wir annehmen, daß es zwei Hyperebenen H_a und H_b gibt, die von den beiden Geraden in unterschiedlicher Reihenfolge geschnitten werden. Vergleichen wir die x_d–Koordinaten der Schnitte, so erhalten wir o.B.d.A.

$$\begin{aligned} \frac{1}{a_d}(a_0 - \sum_{i=1}^{d-1} a_i u_i) &< \frac{1}{b_d}(b_0 - \sum_{i=1}^{d-1} b_i u_i) \\ \frac{1}{a_d}(a_0 - \sum_{i=1}^{d-1} a_i v_i) &> \frac{1}{b_d}(b_0 - \sum_{i=1}^{d-1} b_i v_i) \end{aligned}$$

Daraus aber folgt unmittelbar, daß $\mathbf{u}^{(d-1)}$ und $\mathbf{v}^{(d-1)}$ auf verschiedenen Seiten der Hyperebene H_{ab} liegen und somit nicht in der gleichen Zelle des Arrangements $\mathcal{P}^{(d-1)}$ liegen können. ∎

Komplexitätsanalyse

Für die Größe $s(d, n)$ der beschriebenen Datenstruktur im $\mathbb{R}^d$ für n Hyperebenen findet man sofort die Rekursionsgleichung

$$\begin{aligned} s(1, n) &\leq cn \\ s(d, n) &\leq cn \cdot s(d-1, n^2) \end{aligned}$$

mit der Lösung $s(d, n) = O(n^{2^d - 1})$. Die Zeit $p(d, n)$ zum Erstellen dieser Datenstruktur läßt sich in analoger Weise abschätzen durch $p(d, n) = O(n^{2^d - 1} \log n)$. Für die Zeit zur Lokalisierung eines Punktes erhält man

$$\begin{aligned} t(1, n) &\leq c \log n \\ t(d, n) &\leq c \log n + t(d-1, n^2) \end{aligned}$$

mit der Lösung $t(d,n) = O(\log n)$.
Bemerkung:
Leider erzeugt dieser Algorithmus durch die wiederholten Projektionen bis zu $O(n^{2^d-1})$ Regionen, die eine wesentlich feinere Partitionierung des $\mathbb{R}^d$ darstellen, als sie durch die Zelleneinteilung von $\mathcal{P}^{(d)}$ gegeben ist, die nur $O(n^d)$ Zellen besitzt.

3.3.5 Spezielle Strahlverfolgungsprobleme im $\mathbb{R}^d$

Wie wir eben gesehen haben, ist die Datenstruktur von [DL76] insbesondere auch dazu geeignet, die Hyperebene im Arrangement $\mathcal{P}^{(d)}$ zu bestimmen, die von einem Strahl der Form $\mathbf{x}(\zeta) = (u_1, u_2, \ldots, u_{d-1}, \zeta)^T$, $\zeta \geq 0$ zuerst getroffen wird. Wir interessieren uns aber für Strahlen der Form

$$\mathbf{x}(\zeta) = (u_1, u_2 \ldots, u_{d-1} + \zeta w_{d-1}, u_d + \zeta w_d)^T, \quad \zeta \geq 0.$$

Diese besitzen einen weiteren Freiheitsgrad. Um diesen in den Griff zu bekommen, werden wir den $x_1 x_2 \ldots x_{d-2}$–Raum so partitionieren, daß für jede dadurch entstehende Region $\hat{R}^{(d-2)}$ gilt:

> Die kombinatorische Struktur des Arrangements $\mathcal{A}(\mathbf{u})$, das in dem 2–dimensionalen Raum $S(\mathbf{u}) = \{(u_1, \ldots, u_{d-2}, x_{d-1}, x_d)^T | x_{d-1}, x_d \in \mathbb{R}\}$ durch den Schnitt mit den Hyperebenen aus $\mathcal{H}^{(d)}$ entsteht, ist gleich für alle $\mathbf{u}^{(d-2)} = (u_1, \ldots, u_{d-2})^T \in \hat{R}^{(d-2)}$, d.h. die relative Lage der Geraden von $\mathcal{A}(\mathbf{u})$ und ihrer Schnittpunkte bleibt invariant, solange sich $\mathbf{u}^{(d-2)}$ nur in einer Region bewegt.

Diese Eigenschaft ist in Analogie zu der von [DL76] zu sehen, deren Partitionierung des $x_1 x_2 \ldots x_{d-1}$–Raum garantiert, daß die Reihenfolge der Hyperebenenschnitte mit dem 1–dimensionalen Raum $\{(u_1, \ldots, u_{d-1}, x_d)^T | x_d \in \mathbb{R}\}$ invariant bleibt für alle $\mathbf{u}^{(d-1)} = (u_1, \ldots, u_{d-1})^T$ einer entsprechenden Region.
Behauptung:
Die gewünschte Partitionierung läßt sich wie folgt gewinnen: Wir bringen alle Tripel der $(d-1)$–dimensionalen Hyperebenen zum Schnitt und projizieren diese $\binom{n}{3}$ $(d-3)$–dimensionalen Teilräume in den $x_1 x_2 \ldots x_{d-2}$–Raum. Die Zellen des Arrangements $\hat{\mathcal{P}}^{(d-2)}$ in diesem $(d-2)$–dimensionalen Raum, gebildet aus den $\binom{n}{3}$ neuen Hyperebenen, sind die gesuchten Regionen.

Beweis: Zum Beweis dieser Aussage betrachten wir zwei verschiedene 2–dimensionale Räume $S(\mathbf{u}) = \{(u_1, u_2, \ldots, u_{d-2}, x_{d-1}, x_d)^T | x_{d-1}, x_d \in \mathbb{R}\}$ und $S(\mathbf{v}) = \{(v_1, v_2, \ldots, v_{d-2}, x_{d-1}, x_d)^T | x_{d-1}, x_d \in \mathbb{R}\}$. Wir möchten zeigen, daß die kombinatorische Struktur von $\mathcal{A}(\mathbf{u})$ und $\mathcal{A}(\mathbf{v})$ gleich ist, wenn $\mathbf{u}^{(d-2)}$ und $\mathbf{v}^{(d-2)}$ in der gleichen Zelle $\hat{R}^{(d-2)}$ des zugehörigen Arrangements $\hat{\mathcal{P}}^{(d-2)}$ liegen. Wir führen den Beweis wieder indirekt:

Nehmen wir an, daß sich die kombinatorische Struktur von $\mathcal{A}(\mathbf{u})$ von der von $\mathcal{A}(\mathbf{v})$ unterscheidet. Dann gibt es drei Hyperebenen H_a, H_b und H_c, so daß der Schnittpunkt $H_a \cap H_b \cap S(\mathbf{u})$ links/rechts der Geraden $H_c \cap S(\mathbf{u})$ liegt, aber $H_a \cap H_b \cap S(\mathbf{v})$ rechts/links der Geraden $H_c \cap S(\mathbf{v})$ liegt. Daraus schließen wir, daß sich $\mathbf{u}^{(d-2)}$ und $\mathbf{v}^{(d-2)}$ auf verschiedenen Seiten

der Hyperebene H_{abc} befinden. H_{abc} bezeichnet dabei die Projektion von $H_a \cap H_b \cap H_c$ in den $x_1 x_2 \ldots x_{d-2}$–Raum und ist gegeben durch (vgl. Gleichung (A.15)):

$$H_{abc}: \sum_{i=1}^{d-2} \det \begin{bmatrix} a_i & a_{d-1} & a_d \\ b_i & b_{d-1} & b_d \\ c_i & c_{d-1} & c_d \end{bmatrix} x_i = \det \begin{bmatrix} a_0 & a_{d-1} & a_d \\ b_0 & b_{d-1} & b_d \\ c_0 & c_{d-1} & c_d \end{bmatrix}$$

Die Schnittgeraden in $S(\mathbf{u})$ lauten:

$$\begin{aligned} a_{d-1}x_{d-1} + a_d x_d &= a_0 - \sum_{i=1}^{d-2} a_i u_i \\ b_{d-1}x_{d-1} + b_d x_d &= b_0 - \sum_{i=1}^{d-2} b_i u_i \\ c_{d-1}x_{d-1} + c_d x_d &= c_0 - \sum_{i=1}^{d-2} c_i u_i \end{aligned}$$

Die x_{d-1}, x_d–Koordinaten des Schnittes $H_a \cap H_b \cap S(\mathbf{u})$ sind bestimmt durch:

$$\begin{aligned} \det \begin{bmatrix} a_{d-1} & a_d \\ b_{d-1} & b_d \end{bmatrix} x_{d-1} &= \det \begin{bmatrix} a_0 - \sum a_i u_i & a_d \\ b_0 - \sum b_i u_i & b_d \end{bmatrix} \\ \det \begin{bmatrix} a_{d-1} & a_d \\ b_{d-1} & b_d \end{bmatrix} x_d &= \det \begin{bmatrix} a_{d-1} & a_0 - \sum a_i u_i \\ b_{d-1} & b_0 - \sum b_i u_i \end{bmatrix} \end{aligned}$$

Eingesetzt in die Geradengleichung von $H_c \cap S(\mathbf{u})$ ergibt dies:

$$\begin{aligned} & \pm c_{d-1} \det \begin{bmatrix} a_0 - \sum a_i u_i & a_d \\ b_0 - \sum b_i u_i & b_d \end{bmatrix} \pm c_d \det \begin{bmatrix} a_{d-1} & a_0 - \sum a_i u_i \\ b_{d-1} & b_0 - \sum b_i u_i \end{bmatrix} \\ < \; & \pm (c_0 - \sum c_i u_i) \det \begin{bmatrix} a_{d-1} & a_d \\ b_{d-1} & b_d \end{bmatrix} \end{aligned}$$

Dies führt schließlich zu:

$$\pm \sum_{i=1}^{d-2} \det \begin{bmatrix} a_i & a_{d-1} & a_d \\ b_i & b_{d-1} & b_d \\ c_i & c_{d-1} & c_d \end{bmatrix} u_i > \pm \det \begin{bmatrix} a_0 & a_{d-1} & a_d \\ b_0 & b_{d-1} & b_d \\ c_0 & c_{d-1} & c_d \end{bmatrix}$$

Ersetzen wir $\mathbf{u}$ durch $\mathbf{v}$, so kehren sich die Vorzeichen um, und wir erhalten den Widerspruch, daß $\mathbf{u}^{(d-2)}$ und $\mathbf{v}^{(d-2)}$ auf verschiedenen Seiten von H_{abc} liegen. ∎

Wir möchten einen Strahl $\mathbf{x}(\zeta) = (u_1, u_2, \ldots, u_{d-1} + \zeta w_{d-1}, u_d + \zeta w_d)^T$, $\zeta \geq 0$ untersuchen, der orthogonal zum $x_1 x_2 \ldots x_{d-2}$–Raum verläuft. Diesen Raum haben wir so partitioniert, daß alle 2–dimensionalen Arrangements $\mathcal{A}(\mathbf{u})$ die gleiche kombinatorische Struktur aufweisen, sofern $\mathbf{u}^{(d-2)}$ sich nur in einer Region $\hat{R}^{(d-2)}$ dieser Partitionierung bewegt. Diese Invarianz ermöglicht es uns, die begrenzenden Hyperebenen einer Zelle von $\mathcal{A}(\mathbf{u})$ zyklisch zu sortieren.

Eine solche Sortierung ist von Vorteil, denn mit ihrer Hilfe können wir die erste von unserem Strahl $\mathbf{x}(\zeta)$ getroffene Hyperebene in logarithmischer Zeit mittels Binärsuche bestimmen. Es bleibt also nur noch zu beschreiben, wie wir die Zelle von $\mathcal{A}(\mathbf{u})$ für den Startpunkt $\mathbf{u}$ des Strahls finden können.

Die Zelleinteilung des Arrangements $\mathcal{P}^{(d-2)}$ stellt eine Verfeinerung der Zelleinteilung des Arrangements $\hat{\mathcal{P}}^{(d-2)}$ im $x_1x_2\ldots x_{d-2}$–Raum dar, denn die Menge der Hyperebenen von $\hat{\mathcal{P}}^{(d-2)}$ ist eine Teilmenge der Hyperebenen, aus denen $\mathcal{P}^{(d-2)}$ besteht. Da jede Region $R^{(d-2)}_{i_1,\ldots,i_{d-2}}$ eine Teilmenge einer $(d-2)$–dimensionalen Zelle von $\mathcal{P}^{(d-2)}$ ist, ist die kombinatorische Struktur von $\mathcal{A}(\mathbf{u})$ invariant für alle $\mathbf{u}^{(d-2)} \in R^{(d-2)}_{i_1,\ldots,i_{d-2}}$. Eine Konsequenz davon ist:

> Für alle $\mathbf{u} \in R^{(d)}_{i_1,\ldots,i_d}$ wird die Zelle von $\mathcal{A}(\mathbf{u})$, die $\mathbf{u}$ enthält, von den gleichen Hyperebenen in der gleichen zyklischen Reihenfolge begrenzt.

Wir müssen die Datenstruktur von [DL76] also nur etwas erweitern, indem wir zu jeder Region $R^{(d)}_{i_1,\ldots,i_d}$ diese eindeutig bestimmte zyklische Reihenfolge der Hyperebenen abspeichern. Jede Zelle von $\mathcal{A}(\mathbf{u})$ ist ein konvexes Polygon, das wir als den Schnitt von n Halbebenen der Form $a_{d-1}x_{d-1} + a_dx_d < a_0 - \sum_{i=1}^{d-2} a_iu_i$ bestimmen können. Dies geschieht für jede Region, indem wir einen beliebigen Probepunkt $\mathbf{u} \in R^{(d)}_{i_1,\ldots,i_d}$ wählen.

Diese Argumente zeigen uns, daß wir unser spezielles Strahlverfolgungsproblem in logarithmischer Zeit lösen können, wenn wir die Datenstuktur von [DL76] entsprechend erweitern. Die notwendigen Modifikationen vergrößern den Platzbedarf um den Faktor n auf $O(n^{2^d})$ und die Preprocessing–Zeit ist durch $O(n^{2^d}\log n)$ beschränkt.

Kehren wir zu unserem Ausgangspunkt zurück. Im 6–dimensionalen Plückerraum können wir nun in logarithmischer Zeit die erste Hyperebene H_{cd} bestimmen, mit welcher der Punkt $\mathbf{p}_{ab}$ auf seiner Bahn $\mathbf{p}_{ab}(\zeta) = (\alpha_{01}, \alpha_{02}, \alpha_{03}, \alpha_{12}, \alpha_{23} - \zeta\,\alpha_{02}, \alpha_{31} + \zeta\,\alpha_{01})^T$ zusammenstößt. Auf diesem Weg haben wir somit die erste Gerade L_{cd} und natürlich auch das Liniensegment l_{cd} bestimmt, mit dem unser bewegtes Liniensegment l_{ab} zuerst kollidiert.

3.3.6 Zusammenfassung

Bei der Berechnung von $t^{trans}_{col}(l_{ab}, E)$ können wir sowohl in Stufe 1 als auch in Stufe 2 logarithmische Suchzeiten garantieren. Stufe 1 erfordert ein Point–Location im 5–dimensionalen Arrangement $\mathcal{K}(E)$ zur impliziten Ermittlung der Menge $E(C)$ der potentiellen Kollisionskandidaten. Unter Benutzung des Verfahrens von [DL76] beläuft sich die dafür notwendige Preprocessing–Zeit auf $p_1 = O(|E|^{2^5})$. In Stufe 2 benutzen wir erneut diese Technik in einer modifizierten Form zum Ray–Tracing im 6–dimensionalen Arrangement $\mathcal{P}(E(C))$. Auch in diesem Fall ist die Suchzeit logarithmisch. Das Preprocessing kostet $O(|E(C)|^{2^6}\log|E(C)|)$ für jede Zelle C des Arrangements $\mathcal{K}(E)$. Wir ersparen es uns, diese Zellen explizit zu berechnen. Statt dessen benutzen wir einfach die verfeinerte Partitionierung, die der Algorithmus von [DL76] ohnehin liefert. Diese besteht aus $O(|E|^{2^5-1})$ Regionen, deshalb ist $p_2 = O(|E|^{2^5-1} \cdot |E|^{2^6+1}\log|E|)$.

Wir haben also erreicht, die Vorverarbeitungsdauer polynomiell zu beschränken, wenn auch der Exponent fast dreistellig geworden ist. Damit erhalten wir die Aussage, daß $t^{trans}_{col}(E_1, E_2)$

für zwei Mengen von roten und blauen Liniensegmenten im $\mathbb{R}^3$ in Zeit $O((|E_1| + |E_2|)^{2-\varepsilon}$ ($\varepsilon > \frac{1}{100}$) berechnet werden kann. Zusammen mit den Aussagen, die wir zur Berechnung von $t_{col}^{trans}(V, F)$ für eine Menge V von Punkten und einer Menge F von Dreiecken gewonnen haben, folgt unser Satz.

Bemerkungen:

Mit den probabilistischen Methoden des Random Sampling ist es möglich, eine deutlich bessere Preprocessing–Zeit für die auftretenden Point–Location Probleme zu erzielen. Außerdem kann man mit ihrer Hilfe auch das Strahlverfolgungsproblem für vertikal verlaufende Strahlen in einem Arrangement von Hyperebenen effizient lösen (vgl. [Ma93]). Für unsere Problemstellung sind die vertikalen Strahlen jedoch nicht ausreichend. In [AM92] wird folgender Satz gezeigt:

> Für eine Menge von n Hyperebenen im $\mathbb{R}^d$ kann man in Zeit $O(n^{d+\delta})$, ($\delta > 0$ beliebig) eine Datenstruktur aufbauen, so daß man die von einem Strahl (mit beliebiger Richtung) zuerst getroffende Hyperebene in Zeit $O(\log^4 n)$ bestimmen kann.

Dieses oder ähnliche Resultate lassen sich benutzen, um unsere Konstante ε zu verbessern. Alle diese theoretischen Resultate stehen in grassem Widerspruch zu der Tatsache, daß bisher kein praktikabler subquadratischer Algorithmus zur Berechnung von $t_{col}^{trans}(E_1, E_2)$ bekannt ist.

Beim Versuch, den Algorithmus zur Berechnung von $t_{col}^{trans}(E_1, E_2)$ auf Rotationen zu übertragen, stößt man auf das Problem, daß die rotatorische Bewegung einer Geraden L_{ab} im $\mathbb{R}^3$ eine Bewegung des zugehörigen Plückerpunktes $\mathbf{p}_{ab}$ nach sich zieht, die nicht mehr geradlinig im Plückerraum verläuft, wie folgende Berechnung zeigt:
Analog zu Gleichung (3.1) ergeben sich die Plückerkoordinaten $\mathbf{p}_{ab}(\zeta)$ aus den 2×2 Unterdeterminanten der folgenden Matrix:

$$\begin{bmatrix} a_0 & \cos\zeta\, a_1 + \sin\zeta\, a_2 & -\sin\zeta\, a_1 + \cos\zeta\, a_2 & a_3 \\ b_0 & \cos\zeta\, b_1 + \sin\zeta\, b_2 & -\sin\zeta\, b_1 + \cos\zeta\, b_2 & b_3 \end{bmatrix}, \quad \mathbf{p}_{ab}(\zeta) = \begin{bmatrix} \cos\zeta\, \alpha_{01} + \sin\zeta\, \alpha_{02} \\ -\sin\zeta\, \alpha_{01} + \cos\zeta\, \alpha_{02} \\ \alpha_{03} \\ \alpha_{12} \\ \cos\zeta\, \alpha_{23} + \sin\zeta\, \alpha_{31} \\ -\sin\zeta\, \alpha_{23} + \cos\zeta\, \alpha_{31} \end{bmatrix}$$

Kapitel 4

Approximative Algorithmen zur Kollisionserkennung

In dem vorangegangenen Kapitel standen die theoretischen Resultate für effiziente Abstands– und Kollisionsberechnungen im Mittelpunkt. Die beteiligten Körper wurden durch Polyeder und die Bewegungen durch Translationen bzw. Rotationen modelliert. Unter der Einschränkung, daß jeweils nur ein einzelner Körper bewegt werden durfte, konnten die notwendigen Berechnungen auf die Nullstellenbestimmung von Polynomen kleinen Grades zurückgeführt werden. Vom praktischen Standpunkt aus betrachtet, ist aber eine effiziente Kollisionserkennung für sehr viel komplexere Bewegungen notwendig. Für eine realistische Simulation von Bewegungsabläufen in der Robotik muß man beispielsweise Bewegungen von gelenkig verbundenen Körpern betrachten, deren Beweglichkeit von der Art ihrer mechanischen Kopplung abhängt. Da ein Roboter in der Regel aus mehreren Gliedern besteht, die gleichzeitig bewegt werden können, entstehen komplizierte überlagerte Bewegungen. Wenn ein Körper eine gleichförmige, geradlinige Bewegung ausführt, während sich ein zweiter mit konstanter Winkelgeschwindigkeit um eine feste Achse dreht, entsteht als Relativbewegung bereits eine Spirale. Im Fall von Polyedern muß man dann zur exakten Ermittlung des ersten Kollisionszeitpunktes unter anderem den Schnitt von spiralförmigen Bahnkurven mit Ebenen berechnen, was nur mittels numerischer Approximation möglich ist.

Das Ziel dieses Kapitels ist es, approximative Algorithmen zur Kollisionserkennung für kompliziertere Bewegungen als Translationen und Rotationen zu entwickeln. Sie beruhen auf statischen Abstandsberechnungen und der Bestimmung von Bahnkurvenlängen (siehe Abschnitt 4.2). Um mit Hilfe dieser Algorithmen die Kollisionsfreiheit eines Bewegungsablaufes zu verifizieren, sind Fehlerabschätzungen für die Güte der verwendeten Approximationen erforderlich. Indem wir die Bahnkurven der bewegten Objekte durch Interpolationspolynome annähern, sind wir in der Lage, a priori Fehlerabschätzungen zu erhalten und die Toleranzwerte der Algorithmen festzulegen. Die erzielten Resultate werden wir in Abschnitt 4.3 anwenden, um Kollisionen bei Bewegungsabläufen von Robotern zu erkennen. Um die Effizienz und damit die praktische Anwendbarkeit der Algorithmen zu steigern, benutzen wir neben der Polyederbeschreibung einfachere Objekte, die den eigentlichen Körper einhüllen. Dank dieser Hüllkörper lassen sich viele Abstands– und Bahnkurvenberechnungen einsparen. In Abschnitt 4.1 beschreiben wir, wie man in einfacher Weise optimale Hüllkörper für

Polyeder generieren kann.

4.1 Numerische Methoden zur Bestimmung von Hüllkörpern

Komplizierte Objekte durch einfache Formen zu approximieren ist eine unverzichtbare Maßnahme, um die Detailfülle komplexer geometrischer Szenenbeschreibungen zu bewältigen. Für viele Algorithmen erweist es sich als günstig, jedes Objekt in einfachere Körper einzuhüllen.

Für das Auftreten einer Kollision zwischen bewegten Objekten ist eine Kollision der zugehörigen Hüllkörper eine notwendige Bedingung. Eine vorgeschaltete Überprüfung dieser Bedingung hilft, Berechnungen für die eingehüllten Objekte einzusparen, und trägt damit zur Effizienzsteigerung der Algorithmen bei. Je einfacher die Form der Hüllkörper ist, desto effizienter verlaufen die Vortests, aber desto gröber wird die ursprüngliche Form approximiert und desto geringer ist die Aussagekraft des Vortests. Diesem Trade–off zwischen Einfachheit der Form und Güte der Approximation kann man begegnen, indem man mit einer immer exakter werdenden Folge von Hüllkörpern arbeitet.

Als Hüllkörper für Polyeder bieten sich zum Beispiel ihre konvexen Hüllen, Quader, Kugeln oder Zylinder an. In den folgenden Abschnitten wollen wir zeigen, wie man für Polyeder auf einfachem Weg kleinste einschließende Kugeln, Zylinder und Quader bestimmen kann. Wir benutzen dazu keine exakten kombinatorischen Algorithmen, sondern ein leicht implementierbares, numerisches Verfahren, das in allen drei Fällen in der gleichen Weise angewendet werden kann.

4.1.1 Kleinste einschließende Kugel

Bei der Berechnung eines konvexen Hüllkörpers für ein Polyeder kann man sich darauf beschränken, die Eckpunkte P der konvexen Hülle des Polyeders einzuhüllen.

Um eine Punktmenge $P = \{\mathbf{p_1}, \mathbf{p_2}, \ldots \mathbf{p_n}\} \subset \mathbb{R}^3$ in eine möglichst kleine Kugel S zu verpacken, legt man den Kugelmittelpunkt $\mathbf{c}$ so, daß der größte Abstand eines Punktes aus P zu $\mathbf{c}$ minimiert wird. $r_i(\mathbf{c}) = |\mathbf{p_i} - \mathbf{c}|$ bezeichne den Abstand des i-ten Punktes von $\mathbf{c}$. Der gesuchte Kugelradius r_{min} berechnet sich wie folgt:

$$r_{min} = \min_{\mathbf{c}\in\mathbb{R}^3} r(\mathbf{c}), \quad \text{wobei} \quad r(\mathbf{c}) \;=\; \max_{i=1,\ldots,n} r_i(\mathbf{c})$$

Die Funktion $r(\mathbf{c}) : \mathbb{R}^3 \to \mathbb{R}$ ist konvex, denn für $\lambda \in [0,1]$ gilt:

$$r((1-\lambda)\mathbf{c_1} + \lambda\mathbf{c_2}) \;\leq\; (1-\lambda)\, r(\mathbf{c_1}) + \lambda\, r(\mathbf{c_2})$$

Beweis: Wegen der Dreiecksungleichung gilt für alle $i = 1, \ldots, n$:

$$\begin{aligned} |\mathbf{p_i} - (1-\lambda)\mathbf{c_1} - \lambda\mathbf{c_2}| &= |(1-\lambda)(\mathbf{p_i} - \mathbf{c_1}) + \lambda(\mathbf{p_i} - \mathbf{c_2})| \\ &\leq (1-\lambda)|\mathbf{p_i} - \mathbf{c_1}| + \lambda|\mathbf{p_i} - \mathbf{c_2}| \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad & \max_i |\mathbf{p_i} - (1-\lambda)\mathbf{c_1} - \lambda\mathbf{c_2}| \\ \leq \; & \max_i \{(1-\lambda)|\mathbf{p_i} - \mathbf{c_1}| + \lambda|\mathbf{p_i} - \mathbf{c_2}|\} \\ \leq \; & (1-\lambda)\max_i |\mathbf{p_i} - \mathbf{c_1}| + \lambda \max_i |\mathbf{p_i} - \mathbf{c_2}| \end{aligned}$$

■

Zur Optimierung von $\mathbf{c}$ wollen wir ein numerisches Verfahren einsetzen, das mit einem vorgegebenen Wert startet und in iterativer Weise bessere Lösungen konstruiert. Solche Verfahren können jedoch nur lokale Minima finden. In unserem Fall ist wegen der Konvexität der Zielfunktion $r(\mathbf{c})$ aber jedes lokale Minimum auch ein globales Minimum. Folglich werden wir von jedem beliebigen Startpunkt aus immer ins absolute Minimum geführt.
Wir benutzen den Downhill Simplex Algorithmus von Nelder und Mead [NM65], weil er ohne die Berechnung von Gradienten auskommt ($r(\mathbf{c})$ ist nicht differenzierbar!) und besonders einfach implementiert werden kann.

4.1.2 Kleinster einschließender Zylinder

Analog zur numerischen Berechnung der kleinsten einschließenden Kugel wollen wir nun ein Verfahren entwickeln, das es uns erlaubt, eine vorgegebene Punktmenge $P = \{\mathbf{p_1}, \mathbf{p_2}, \ldots \mathbf{p_n}\}$ in einen möglichst kleinen Zylinder einzuschließen.
Die Zylinderachse sei durch zwei Punkte $\mathbf{a}, \mathbf{b}$ beschrieben. Den Abstand des i-ten Punktes $\mathbf{p_i}$ von dieser Achse bezeichnen wir mit $r_i(\mathbf{a}, \mathbf{b})$ und seine Höhe bezüglich der Achsrichtung mit $h_i(\mathbf{a}, \mathbf{b})$. Es gilt:

$$\begin{aligned} r_i(\mathbf{a}, \mathbf{b}) &= \frac{|\mathbf{a} \times \mathbf{b} + \mathbf{b} \times \mathbf{p_i} + \mathbf{p_i} \times \mathbf{a}|}{|\mathbf{b} - \mathbf{a}|} \\ h_i(\mathbf{a}, \mathbf{b}) &= \frac{\mathbf{p_i}^T(\mathbf{b} - \mathbf{a})}{|\mathbf{b} - \mathbf{a}|} \end{aligned}$$

Auf der Suche nach einem Zylinder mit kleinstmöglichem Volumen V_{min} haben wir die Zylinderachse mit Hilfe der Punkte $\mathbf{a}$ und $\mathbf{b}$ so zu wählen, daß folgender Ausdruck minimiert wird:

$$\begin{aligned} V_{min} &= \min_{\mathbf{a},\mathbf{b} \in \mathbb{R}^3} V(\mathbf{a}, \mathbf{b}), \quad \text{wobei} \\ V(\mathbf{a}, \mathbf{b}) &= \min_{\mathbf{a},\mathbf{b} \in \mathbb{R}^3} r^2(\mathbf{a}, \mathbf{b})\, h(\mathbf{a}, \mathbf{b}) \quad \text{mit} \\ h(\mathbf{a}, \mathbf{b}) &= \max_i h_i(\mathbf{a}, \mathbf{b}) - \min_i h_i(\mathbf{a}, \mathbf{b}) \\ r(\mathbf{a}, \mathbf{b}) &= \max_i r_i(\mathbf{a}, \mathbf{b}) \end{aligned}$$

Die zu minimierende Funktion ist hier leider nicht konvex. In Abhängigkeit vom Startwert wird uns eine iterativ arbeitende numerische Minimumsuche zu unterschiedlichen lokalen Minima führen. Deshalb beginnen wir die Suche nach dem absoluten Minimum von vielen unterschiedlichen Startpunkten aus, in der Hoffnung, auch einmal zum absoluten Minimum zu gelangen. Hier stellt sich natürlich sofort die Frage nach geeigneten Startwerten.
Zunächst einmal stellt man fest, daß man bei der Spezifikation der Zylinderachse mit weniger Parametern auskommt als durch die kartesischen Koordinaten der Punkte **a** und **b** gegeben sind, denn eine Gerade im $\mathbb{R}^3$ besitzt nur 4 Freiheitsgrade. Wir legen sie dadurch fest, daß wir die Punkte **a** und **b** auf der Oberfläche einer Kugel S wählen, die P umschließt. Das ist natürlich nur dann gerechtfertigt, wenn wir sicher sein können, daß die Achse eines kleinsten einschließenden Zylinders diese Kugel schneidet.

Lemma 4.1 *Für eine Punktmenge P sei C_{min} ein Zylinder kleinsten Volumens mit der Eigenschaft $P \subset C_{min}$. Gleichzeitig sei $S \supset P$ eine einhüllende Kugel für P. Es gilt: Die Gerade, die der Zylinderachse von C_{min} entspricht, schneidet S.*

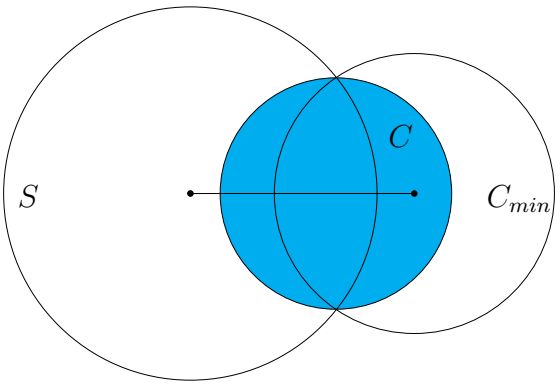


Abbildung 4.1: Zylinderachse von C_{min} schneidet S

Beweis: Der Radius $r_{C_{min}}$ des Zylinders ist nicht größer als der Radius r_S der Kugel. Angenommen die Zylinderachse schneidet S nicht, so entsteht bei einer Projektion der Szene auf eine Ebene senkrecht zur Achsrichtung die in Abbildung 4.1 dargestellte Situation ($S \cap C_{min} \neq \emptyset$ und $r_{C_{min}} \leq r_S$). Man erkennt, daß sich in einfacher Weise ein Zylinder C mit kleinerem Radius und damit mit kleinerem Volumen konstruieren läßt, der P enthält ($P \subset (S \cap C_{min}) \subset C$) und dessen Achse parallel zur Achse von C_{min} verläuft. Das ist ein Widerspruch zur Wahl von C_{min}. ■

Zur Spezifikation der Punkte **a** und **b** auf der Kugeloberfläche von S benutzen wir Kugelkoordinaten. **c** sei der Kugelmittelpunkt und r_S der Kugelradius. Dann ist

$$\mathbf{a} = \mathbf{c} + r_S \begin{bmatrix} \sin\vartheta_a \cos\varphi_a \\ \sin\vartheta_a \sin\varphi_a \\ \cos\vartheta_a \end{bmatrix}, \qquad \mathbf{b} = \mathbf{c} + r_S \begin{bmatrix} \sin\vartheta_b \cos\varphi_b \\ \sin\vartheta_b \sin\varphi_b \\ \cos\vartheta_b \end{bmatrix}.$$

Damit ist die Funktion $V(\mathbf{a}, \mathbf{b}) = V(\vartheta_a, \varphi_a, \vartheta_b, \varphi_b)$ nur noch abhängig von 4 Parametern, und es gilt:

$$V_{min} = \min\{V(\vartheta_a, \varphi_a, \vartheta_b, \varphi_b) \mid \vartheta_a, \vartheta_b \in [0, \pi], \varphi_a, \varphi_b \in [0, 2\pi]\}$$

Wählt man S als die kleinste einschließende Kugel für P, so ist der Suchraum für die Zylinderachse in sinnvoller Weise eingeschränkt. Bei der numerischen Minimumsuche kann man zum Beispiel mit zufälligen Quadrupeln von Winkeln als Startwerten arbeiten und unter den gefundenen Minima dasjenige mit kleinstem Funktionswert als Approximation für das absolute Minimum benutzen.

Bei der obigen Formulierung des Problems ist der Suchraum für die Zylinderachse 4–dimensional, entsprechend der Zahl der Freiheitsgrade einer Gerade im $\mathbb{R}^3$. Die nun folgende Formulierung unserer Minimierungsaufgabe reduziert die Dimension des Suchraums auf zwei:
Sei $\mathbf{d}$ eine zunächst feste Raumrichtung. Mit $V(\mathbf{d})$ wollen wir das Volumen des minimalen, die Punktmenge P umschließenden Zylinders bezeichnen, dessen Achsrichtung parallel zu $\mathbf{d}$ liegt. Projizieren wir P auf eine Ebene senkrecht zu $\mathbf{d}$, so verläuft die Achse des Zylinders mit kleinstem Radius gerade durch den Mittelpunkt des kleinsten einschließenden Kreises der projizierten Punktmenge. Dies ist dann auch der Zylinder mit kleinstem Volumen, da die Zylinderhöhe $h(\mathbf{d})$ bei fester Achsrichtung invariant ist. Es gilt:

$$\begin{aligned} V(\mathbf{d}) &= r^2(\mathbf{d})\, h(\mathbf{d}) \quad \text{mit} \\ h(\mathbf{d}) &= \max_i \mathbf{d}^T \mathbf{p_i} - \min_i \mathbf{d}^T \mathbf{p_i} \\ r^2(\mathbf{d}) &= \min_{\mathbf{c} \in \mathbb{R}^2} \max_i (c_1 - \mathbf{d_1}^T \mathbf{p_i})^2 + (c_2 - \mathbf{d_2}^T \mathbf{p_i})^2 \ , \text{ wobei} \\ & \quad \mathbf{d}, \mathbf{d_1}, \mathbf{d_2} \text{ ein Orthonormalsystem bilden} \end{aligned}$$

Den kleinsten einschließenden Kreis können wir leicht mit der in Abschnitt 4.1.1 vorgestellten Methode berechnen. Es bleibt also nur noch die optimale Wahl der Achsrichtung $\mathbf{d}$. Wie oben benutzen wir den Polarwinkel ϑ_d und den Azimut φ_d zur Richtungsangabe und erhalten $\mathbf{d} = (\sin\vartheta_d \cos\varphi_d, \sin\vartheta_d \sin\varphi_d, \cos\vartheta_d)^T$. Die Funktion $V(\mathbf{d}) = V(\vartheta_d, \varphi_d)$ ist somit nur von 2 Parametern abhängig, und es gilt:

$$V_{min} = \min\{V(\vartheta_d, \varphi_d) \mid \vartheta_d \in [0, \frac{\pi}{2}], \varphi_d \in [0, 2\pi]\}$$

Auch hier müssen wir wegen der Existenz lokaler Minima der Funktion $V(\vartheta_d, \varphi_d)$ von hinreichend vielen Startwerten ausgehen, um das absolute Minimum zu finden. Die Startwerte (ϑ_d, φ_d) kann man im Bereich $[0, \frac{\pi}{2}] \times [0, 2\pi]$ systematisch wählen oder auch würfeln.

Bemerkungen:
Bei einer Implementierung schnitt die Variante der Minimumsuche im 4–dimensionalen Raum gegenüber der zuletzt vorgestellten bezüglich der Laufzeit besser ab. Dies ist darauf zurückzuführen, daß im letzteren Fall eine Funktionsauswertung, die der Berechnung des kleinsten einschließenden Kreises entspricht, relativ teuer ist, wenn man den numerischen Algorithmus von [NM65] benutzt. Besser wäre es, einen kombinatorischen Algorithmus zu diesem Zweck einzusetzen.

Zur Berechnung der kleinsten einschließenden Kugel bzw. des kleinsten einschließenden Ellipsoides existieren effiziente kombinatorische Algorithmen: In [We91] wird ein randomisierter Linearzeitalgorithmus für diese Problemklasse beschrieben, dessen inkrementelle Vorgehensweise sich aber nicht auf unser Problem des kleinsten einschließenden Zylinders anwenden läßt.

Anstatt kleinste einschließende Körper für ein Polyeder zu suchen, kann man auch größte eingeschlossene Körper berechnen. Sie sind dann besonders nützlich, wenn man einfache hinreichende Kriterien für das Auftreten von Kollisionen benötigt. Für ein konvexes Polyeder kann man zum Beispiel eine größte eingeschlossene Kugel wie folgt finden:

$$\begin{aligned} r_{max} &= \max_{\mathbf{c}\in\mathbb{R}^3} \min_i (n_{i0} - \mathbf{n_i}^T\mathbf{c}) \\ &= -\min_{\mathbf{c}\in\mathbb{R}^3} \max_i (\mathbf{n_i}^T\mathbf{c} - n_{i0}) \end{aligned}$$

Dabei sei das Polyeder als Schnitt von Halbräumen gegeben: $\bigcap_i \{\mathbf{x} | \mathbf{n_i}^T\mathbf{x} \leq n_{i0}\}$. Ebenso wie bei der Berechnung der kleinsten einschließenden Kugel ist auch hier die zu minimierende Funktion konvex. Daher kommt eine Minimumsuche auf numerischem Weg mit einem Startwert aus. Aufgrund der Linearität läßt sich dieses Problem aber auch als einfaches Lineares Programm fester Dimension formulieren:

$$\max\ r, \quad \text{so daß} \quad \forall i: \quad n_{i0} - \mathbf{n_i}^T\mathbf{c} \geq r$$

Dafür sind sowohl deterministische [Me83] als auch randomisierte [Se91] Linearzeitalgorithmen bekannt.

4.1.3 Kleinster einschließender Quader

Als nächstes wollen wir die Punktmenge $P = \{\mathbf{p_1}, \mathbf{p_2}, \ldots \mathbf{p_n}\}$ in einen Quader B kleinsten Volumens einschließen. Wenn wir die Orientierung der Achsrichtungen fixieren, können wir die Position des gesuchten Quaders leicht bestimmen. $\mathbf{d_1}, \mathbf{d_2}$ und $\mathbf{d_3}$ bezeichnen im folgenden die paarweise senkrechten Richtungen der Achsen:

$$V(\mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3}) = \prod_{j=1}^{3} (\max_i \mathbf{d_j}^T\mathbf{p_i} - \min_i \mathbf{d_j}^T\mathbf{p_i})$$

Die Achsrichtungen können wir mittels dreier Winkel spezifizieren. $\mathbf{d_j}$ erhält man durch drei sukzessive Rotationen des j–ten Einheitsvektors $\mathbf{e_j}$ um die Koordinatenachsen. Dabei dreht man zunächst um den Winkel α um die $\mathbf{e_1}$-Achse, dann um den Winkel β um die $\mathbf{e_2}$-Achse und schließlich um den Winkel γ um die $\mathbf{e_3}$-Achse. Die Spalten der zugehörigen Rotationsmatrix $\mathbf{R}(\alpha, \beta, \gamma)$ entsprechen den Achsrichtungen $\mathbf{d_j}$:

$$\begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} =$$

$$\begin{bmatrix} \cos\gamma\cos\beta & \cos\gamma\sin\beta\sin\alpha - \sin\gamma\cos\alpha & \cos\gamma\sin\beta\cos\alpha + \sin\gamma\sin\alpha \\ \sin\gamma\cos\beta & \sin\gamma\sin\beta\sin\alpha + \cos\gamma\cos\alpha & \sin\gamma\sin\beta\cos\alpha - \cos\gamma\sin\alpha \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha \end{bmatrix}$$

Da $[\mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3}] = \mathbf{R}(\alpha, \beta, \gamma)$ ist, hängt das Volumen des Quaders nur noch von den drei Eulerwinkeln ab, und es gilt:

$$V_{min} = \min\{V(\alpha, \beta, \gamma) | \alpha, \beta, \gamma \in [0, 2\pi]\}$$

Um das absolute Minimum der Funktion $V(\alpha, \beta, \gamma)$ zu finden, müssen wir, wegen der Existenz lokaler Minima, mit mehreren Startwerten bei der numerischen Minimumsuche beginnen.

4.2 Kollisionserkennung für komplexe Bewegungsabläufe

Bisher haben wir nur Kollisionserkennung für translatorische und rotatorische Bewegungen betrachtet. Dies ist insofern gerechtfertigt, als daß sich jede Bewegung eines starren Körpers durch eine geeignete Folge von Translationen und Rotationen beliebig genau approximieren läßt, weil diese beiden elementaren Bewegungsformen gerade die 6 benötigten Freiheitsgrade zur Verfügung stellen. Um eine hinreichend genaue Approximation einer gegebenen Bewegung zu gewährleisten, muß man in der Regel jedoch sehr kleine Schrittweiten wählen. In einer solchen Situation würde die Kollisionserkennung sehr ineffizient arbeiten, wenn sie für jeden Teilschritt einen kompletten Kollisionstest für eine Translation bzw. Rotation benutzen würde. Hier sind Einsparungen erforderlich.
Wenn das bewegte Objekt noch weit von den Hindernissen entfernt ist, kann ein kleiner, räumlich begrenzter Bewegungsschritt nicht die Gefahr einer Kollision heraufbeschwören. Erst in der Summe können die einzelnen Bewegungsschritte das Objekt so nah an ein Hindernis heranbringen, daß für den weiteren Verlauf der Bewegung eine Kollision nicht mehr auszuschließen ist. Aus dieser Sicht heraus erscheint folgende Vorgehensweise vernünftig:

> Man berechne die kürzeste Entfernung r eines bewegten Objektes P zu einem Hindernis Q. Solange sich kein Punkt $\mathbf{p} \in P$ von seinem Ausgangspunkt um mehr als r entfernt hat, besteht keine Möglichkeit des Zusammenstoßes. Es kann erst dann zu einem kritischen Moment kommen, wenn es einen Punkt $\mathbf{p} \in P$ gibt, dessen Abstand vom Ausgangspunkt den Wert r erreicht. Erst dann muß die entstandene Situation neu eingeschätzt werden.

Im folgenden diskutieren wir dieses einfache Prinzip genauer. Wir beginnen mit einer Demonstration des Verfahrens an einem 2–dimensionalen Beispiel.
Ein einzelner Punkt $\mathbf{p}$ beginne seine Bewegung zum Zeitpunkt t_0 und durchlaufe die Bahnkurve $\mathbf{p}(t)$. Wir fragen uns, ob er seine Bewegung ausführen kann, ohne mit einem stationären Hindernis Q zusammenzustoßen.
Anhand von Abbildung 4.2 erläutern wir nun die oben skizzierte Methode:

> Zu Beginn berechnet man den Abstand r des Punktes $\mathbf{p}(t_0)$ zum Hindernis Q und schlägt um den Startpunkt $\mathbf{p}(t_0)$ einen Kreis mit Radius r. Erst wenn die Bahnkurve $\mathbf{p}(t)$ diesen Kreis verläßt, berechnet man den Abstand zum Hindernis neu und setzt das Verfahren in dieser Weise fort.

Beobachtung 4.2 *Je mehr sich der Punkt* $\mathbf{p}$ *dem Hindernis* Q *nähert, desto öfter müssen Abstandsberechnungen durchgeführt werden. Dies steht im Einklang mit der Intuition, daß man mit geringer werdendem Abstand zum Hindernis auch "vorsichtiger" agieren muß. Solange die Bahnkurve* $\mathbf{p}(t)$ *dem Hindernis nicht allzu nahe kommt, ist die beschriebene Methode bestens geeignet, die Kollisionsfreiheit einer Bewegung zu verifizieren. Wenn aber ein Schnitt der Bahnkurve mit dem Hindernis existiert, werden kurz vor der Kollision die Abstände in immer kürzeren Zeitintervallen bestimmt. Auf diese Weise wird der Kollisionszeitpunkt beliebig genau approximiert. In diesem Fall benutzen wir als Abbruchkriterium, daß die Bewegung stoppt, wenn der momentan berechnete Abstand einen Toleranzwert* EPS *unterschreitet.*

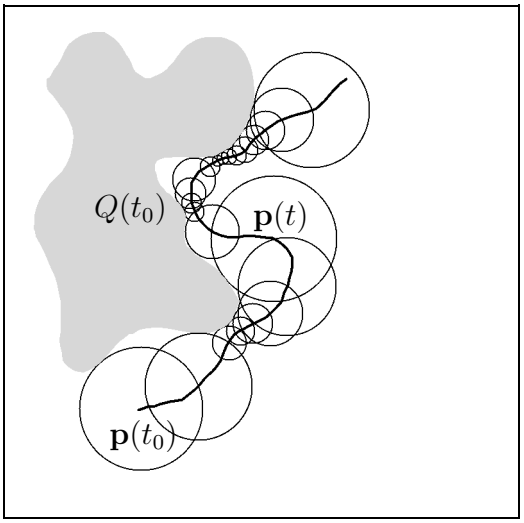


Abbildung 4.2: Approximativer Kollisionstest für die Bewegung eines Punktes

Der folgende Algorithmus präzisiert das oben vorgestellte Verfahren:

$r \leftarrow \delta(\mathbf{p}(t_0), Q(t_0));$
while $r > EPS$
do
  $t_0 \leftarrow \min\{t_1 \geq t_0 \,|\, |\mathbf{p}(t_1) - \mathbf{p}(t_0)| \geq r\};$
  $r \leftarrow \delta(\mathbf{p}(t_0), Q(t_0));$
od
stop

Der zentrale Punkt dieses Algorithmus ist die Berechnung des Minimums $\min\{t_1 \geq t_0 \,|\, |\mathbf{p}(t_1) - \mathbf{p}(t_0)| \geq r\}$. Je nach Form der Bahnkurve $\mathbf{p}(t)$ kann dies erhebliche numerische Probleme aufwerfen, denn man hat es hier im wesentlichen mit einer Nullstellenbestimmung zu tun. Ein Ausweg aus dieser Lage bietet folgende Alternative: Anstatt die Distanz zwischen $\mathbf{p}(t_1)$

und $\mathbf{p}(t_0)$ "Luftlinie" zu messen, kann man auch die Länge des tatsächlich zurückgelegten Weges benutzen, auf dem sich der Punkt $\mathbf{p}$ im Zeitintervall $[t_0, t_1]$ bewegt hat. Diese Länge ist gleich dem Integral über den Betrag der Bahngeschwindigkeit von $\mathbf{p}$. Es gilt:

$$\int_{t_0}^{t_1} |\dot{\mathbf{p}}(t)|\, dt \geq \left| \int_{t_0}^{t_1} \dot{\mathbf{p}}(t) dt \right| = |\mathbf{p}(t_1) - \mathbf{p}(t_0)|$$

Aufgrund dieser Ungleichung darf man im obigen Algorithmus das Minimum auch durch $\min\{t_1 \geq t_0 \mid \int_{t_0}^{t_1} |\dot{\mathbf{p}}(t)|\, dt \geq r\}$ ersetzen, ohne die Korrektheit zu verletzen. Die Berechnung dieses Integrals ist im allgemeinen numerisch gutartiger als die Bestimmung von Nullstellen. Im allgemeinen müssen jedoch die Abstände häufiger neu berechnet werden.

Diese generelle Methode läßt sich nicht nur bei der Bewegung eines punktförmigen Objektes in Anwesenheit eines stationären Hindernisses anwenden. Sie ist auch auf die Situation übertragbar, in der sich zwei starre Körper gleichzeitig bewegen. Grundlage dafür ist folgendes Lemma:

Lemma 4.3 *Für zwei starre Körper P und Q, die sich zum Zeitpunkt t_0 nicht schneiden, gilt:*

$$\begin{aligned} & \max_{p \in P} \int_{t_0}^{t_1} |\dot{\mathbf{p}}(t)|\, dt + \max_{q \in Q} \int_{t_0}^{t_1} |\dot{\mathbf{q}}(t)|\, dt < \delta(P(t_0), Q(t_0)) \\ \Longrightarrow \quad & \max_{p \in P} |\mathbf{p}(t_1) - \mathbf{p}(t_0)| + \max_{q \in Q} |\mathbf{q}(t_1) - \mathbf{q}(t_0)| < \delta(P(t_0), Q(t_0)) \\ \Longrightarrow \quad & P(t_1) \cap Q(t_1) = \emptyset \end{aligned}$$

Beweis: Die erste Folgerung ist klar. Zum Beweis der zweiten zeigen wir, daß $\forall \mathbf{p} \in P, \mathbf{q} \in Q : |\mathbf{p}(t_1) - \mathbf{q}(t_1)| > 0$.

$$\begin{aligned} |\mathbf{p}(t_1) - \mathbf{q}(t_1)| &= |\mathbf{p}(t_1) - \mathbf{p}(t_0) + \mathbf{p}(t_0) - \mathbf{q}(t_0) + \mathbf{q}(t_0) - \mathbf{q}(t_1)| \\ &\geq |\mathbf{p}(t_0) - \mathbf{q}(t_0)| - |\mathbf{p}(t_1) - \mathbf{p}(t_0) + \mathbf{q}(t_0) - \mathbf{q}(t_1)| \\ &\geq \delta(P(t_0), Q(t_0)) - (|\mathbf{p}(t_1) - \mathbf{p}(t_0)| + |\mathbf{q}(t_1) - \mathbf{q}(t_0)|) > 0 \\ &\quad \text{da} \quad |\mathbf{p}(t_1) - \mathbf{p}(t_0)| \leq \max_{p \in P} |\mathbf{p}(t_1) - \mathbf{p}(t_0)| \\ &\quad \text{und} \quad |\mathbf{q}(t_1) - \mathbf{q}(t_0)| \leq \max_{q \in Q} |\mathbf{q}(t_1) - \mathbf{q}(t_0)| \end{aligned}$$

■

Die Verwendbarkeit des Lemmas hängt davon ab, wie leicht das Maximum $\max_{p \in P} d(\mathbf{p}, t_0, t_1)$ bestimmt werden kann. Als Funktion $d(\mathbf{p}, t_0, t_1)$ wählen wir entweder

$$|\mathbf{p}(t_1) - \mathbf{p}(t_0)| \quad \text{oder} \quad \int_{t_0}^{t_1} |\dot{\mathbf{p}}(t)|\, dt.$$

Wenn die betrachteten Körper Polyeder sind, brauchen wir die Maxima nicht über die gesamte Punktmenge zu bilden. Wir dürfen uns auf die Menge ihrer Eckpunkte beschränken, weil die Eckpunkte bei einer Bewegung den größten Weg zurücklegen, wie folgendes Lemma zeigt:

Lemma 4.4 *Für jeden Punkt* $\mathbf{p}$ *eines Polyeders* P *mit der Eckenmenge* $V_P = \{\mathbf{p_1}, \ldots, \mathbf{p_i}, \ldots\}$ *gilt:*

$$d(\mathbf{p}, t_0, t_1) \leq \max_i d(\mathbf{p_i}, t_0, t_1)$$

Beweis: $\forall \mathbf{p} \in P \; \exists \lambda_i \geq 0$ mit $\sum_i \lambda_i = 1$, so daß $\mathbf{p} = \sum_i \lambda_i \mathbf{p_i}$. Daraus folgt:

$$\begin{aligned} d(\mathbf{p}, t_0, t_1) &= d\left(\sum_i \lambda_i \mathbf{p_i}, t_0, t_1\right) \\ &\leq \sum_i \lambda_i \, d(\mathbf{p_i}, t_0, t_1) \\ &\leq \max_i d(\mathbf{p_i}, t_0, t_1) \cdot \sum_i \lambda_i \end{aligned}$$

■

In Abbildung 4.3 ist dieser Sachverhalt im zweidimensionalen für ein bewegtes Polygon $P(t)$ und ein stationäres Hindernis Q skizziert. Solange sich kein Eckpunkt von P von seinem Ausgangspunkt um $r = \delta(P(t_0), Q(t_0))$ entfernt, kann es zu keiner Kollision mit dem Hindernis Q kommen.

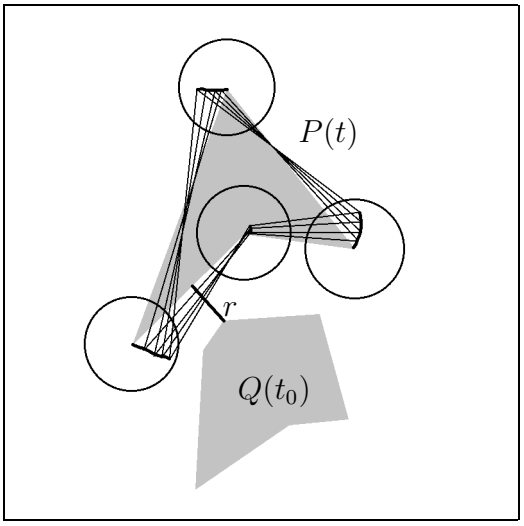


Abbildung 4.3: Approximativer Kollisionstest für die Bewegung eines Polygons

Für die gleichzeitige Bewegung zweier Polyeder $P(t)$ und $Q(t)$ erhalten wir somit folgenden Algorithmus:

$r \leftarrow \delta(P(t_0), Q(t_0))$;
while $r > EPS$
do
  $t_0 \leftarrow \min\{t_1 \geq t_0 \mid \max_{p \in V_P} d(\mathbf{p}, t_0, t_1) + \max_{q \in V_Q} d(\mathbf{q}, t_0, t_1) \geq r\}$;
  $r \leftarrow \delta(P(t_0), Q(t_0))$
od;
stop

Wie bereits erwähnt, läßt sich das Minimum $\min\{t_1 \geq t_0 \,|\, \max_{p \in V_P} d(\mathbf{p}, t_0, t_1) + \max_{q \in V_Q} d(\mathbf{q}, t_0, t_1) \geq r\}$ nur bei sehr einfachen Bahnkurven exakt ermitteln. Wenn P und Q nur Translationen oder Rotationen ausführen – wie in den vorangegangenen Kapiteln angenommen – läßt sich das gesuchte Minimum als Lösung einer linearen Gleichung ermitteln, vorausgesetzt man benutzt die Bahnkurvenlänge $\int_{t_0}^{t_1} |\dot{\mathbf{p}}(t)|\, dt$ für die Funktion $d(\mathbf{p}, t_0, t_1)$. Die Verwendung von $|\mathbf{p}(t_1) - \mathbf{p}(t_0)|$ kann bereits zu Gleichungen führen, die nur numerisch gelöst werden können. Deshalb müssen wir in der Regel mit Näherungen arbeiten. Dann ist es besonders wichtig, die Güte der Näherung zu kennen. Ein systematischer Weg zur Approximation besteht darin, die vorliegende Bahnkurve $\mathbf{v}(t)$ stückweise durch Interpolationspolynome $\mathbf{v}^*(t)$ anzunähern.

Beginnend vom Zeitpunkt t_0 betrachten wir nacheinander kleine Zeitintervalle I_j der Länge Δt. Während jedes solchen Intervalls interpolieren wir die i-te Koordinatenfunktion $v_i(t)$ der Bahnkurve mit einem Polynom $v_i^*(t)$ vom Grad $k-1$. Dazu unterteilen wir jedes Zeitintervall I_j äquidistant durch k Stützstellen. Bei Verwendung der Newton'schen Interpolationsmethode gilt im Intervall I_j für den Fehler der n–ten Ableitung der Bahnkurve (vgl. [Ra86]):

$$\forall t \in I_j : \quad \left| \frac{d^n}{dt^n} v_i(t) - \frac{d^n}{dt^n} v_i^*(t) \right| \;\leq\; \frac{1}{(k-n)!} \max_{\tau \in I_j} \left| \frac{d^k}{dt^k} v_i(\tau) \right| \Delta t^{k-n} \tag{4.1}$$

Diese Abschätzung des Interpolationsfehlers ist sehr hilfreich für die weiteren Betrachtungen. Zuerst beschäftigen wir uns mit der Approximation von

$$\min\{t_1 \geq t_0 | \max_{p \in V_P} |\mathbf{p}(t_1) - \mathbf{p}(t_0)| + \max_{q \in V_Q} |\mathbf{q}(t_1) - \mathbf{q}(t_0)| \geq r\}.$$

Dazu untersuchen wir die Bahnkurven der Polyedereckpunkte in aufeinanderfolgenden Zeitintervallen I_j der Länge Δt. Während jeder Zeitspanne I_j gelte für alle Eckpunkte $\mathbf{v} \in V_P \cup V_Q$, daß $\max_{t \in I_j} |\mathbf{v}^*(t) - \mathbf{v}(t)| \leq \epsilon_1(\Delta t)$ ist. Wir möchten die Bewegung stoppen, bevor die Möglichkeit einer Kollision gegeben ist. Dabei dürfen wir den Interpolationsfehler ϵ_1 nicht vernachlässigen.

$$
\begin{aligned}
&j \leftarrow 0;\\
&\textbf{repeat}\\
&\quad j \leftarrow j+1;\\
&\quad I_j \leftarrow [t_0+(j-1)\Delta t, t_0+j\Delta t];\\
&\quad \textbf{forall } \mathbf{v} \in V_P \cup V_Q\\
&\quad \textbf{do } d(\mathbf{v}) \leftarrow \max_{t\in I_j} |\mathbf{v}^*(t)-\mathbf{v}(t_0)| \textbf{ od}\\
&\textbf{until } \max_{p\in V_P} d(\mathbf{p}) + \max_{q\in V_Q} d(\mathbf{q}) \geq r-2\epsilon_1;\\
&t_1 \leftarrow t_0+(j-1)\Delta t;
\end{aligned}
\tag{4.2}
$$

Um eine geeignete Schrittweite Δt zu wählen, schätzen wir den Interpolationsfehler $\epsilon_1(\Delta t)$ mit Hilfe von Ungleichung (4.1) ab:

$$
\begin{aligned}
|\mathbf{v}^*(t)-\mathbf{v}(t)| \;&\leq\; \sqrt{3}\max_{i=1\ldots3}|v_i^*(t)-v_i(t)|\\
\leq\; \frac{\sqrt{3}}{k!}\max_{\tau\in[t_0,t_1]}&\left|\frac{d^k}{dt^k}v_i(t)\right|\Delta t^k
\end{aligned}
\tag{4.3}
$$

Wenn wir die Bahnkurven linear interpolieren, also $k=2$ setzen, wird $\epsilon_1(\Delta t)$ durch die maximal auftretenden Bahnbeschleunigungen $|\ddot{\mathbf{v}}|$ bestimmt und fällt quadratisch mit abnehmender Schrittweite Δt. Im Fall der linearen Interpolation ist die Bestimmung von $\max_{t\in I_j}|\mathbf{v}^*(t)-\mathbf{v}(t_0)|$ trivial und der obige Approximationsalgorithmus führt zu einer einfachen Abtastung der Bahnkurven in Zeitabständen Δt. Das Kriterium für den Abbruch lautet: Je ein Eckpunkt von P und Q haben sich zusammen um mehr als $r-2\epsilon_1$ von ihren Ausgangspunkten entfernt. Bei quadratischer Interpolation ($k=3$) muß man zur Berechnung von $\max_{t\in I_j}|\mathbf{v}^*(t)-\mathbf{v}(t_0)|$ bereits eine quadratische Gleichung lösen.

Als Alternative betrachten wir nun die Approximation von

$$
\min\{t_1 \geq t_0 | \max_{p\in V_P}\int_{t_0}^{t_1}|\dot{\mathbf{p}}(t)|\,dt + \max_{q\in V_Q}\int_{t_0}^{t_1}|\dot{\mathbf{q}}(t)|\,dt \geq r\}.
$$

Bei komplizierteren Bewegungen werden wir die Bogenlänge $\int_{t_0}^{t_1}|\dot{\mathbf{v}}(t)|\,dt$ der Bahnkurven nicht analytisch berechnen können. – Die Bogenlänge einer Ellipse führt bereits auf ein elliptisches Integral, das sich nicht durch elementare Funktionen ausdrücken läßt. – Deshalb sind wir auch hier auf numerische Methoden angewiesen. An dieser Stelle könnten wir auf die Standardmethoden der numerischen Quadratur zurückgreifen, um die Funktion $|\dot{\mathbf{v}}(t)|$ zu integrieren. Wir könnten zum Beispiel einfach die Trapez– oder die Kepler'sche Regel verwenden. Dagegen spricht zum einen die Notwendigkeit einer schnellen Auswertung des Integranden $|\dot{\mathbf{v}}(\mathbf{t})|$ an den Stützstellen und zum anderen die Schwierigkeit, eine vernünftige Fehlerabschätzung zu gewinnen. Um den Fehler durch eine geeignete Wahl der Schrittweite a priori beschränken zu können, muß man höhere Ableitungen des Integranden kennen, deren Abschätzung in der Regel nicht praktikabel ist. Deshalb schlagen wir den selben Weg wie oben ein: $\mathbf{v}^*(t)$ bezeichne wieder das stückweise definierte Interpolationspolynom für

die Bahnkurve $\mathbf{v}(t)$. Während jedes Zeitintervalls I_j der Länge Δt sei der Approximationsfehler für die Bogenlänge der Bahnkurven aller Eckpunkte $\mathbf{v} \in V_P \cup V_Q$ beschränkt durch $\left| \int_{I_j} |\dot{\mathbf{v}}^*(t)|\, dt - \int_{I_j} |\dot{\mathbf{v}}(t)|\, dt \right| \leq \epsilon_2(\Delta t)$. Dieser Fehler summiert sich fortlaufend und muß deshalb für die Approximation des gesuchten Zeitpunkts t_1 berücksichtigt werden.

$$
\begin{array}{l}
\textbf{forall } \mathbf{v} \in V_P \cup V_Q \\
\textbf{do } d(\mathbf{v}) \leftarrow 0 \textbf{ od}; \\
j \leftarrow 0; \\
\textbf{repeat} \\
\quad j \leftarrow j + 1; \\
\quad I_j \leftarrow [t_0 + (j-1)\Delta t, t_0 + j\Delta t]; \\
\quad \textbf{forall } \mathbf{v} \in V_P \cup V_Q \\
\quad \textbf{do } d(\mathbf{v}) \leftarrow d(\mathbf{v}) + \displaystyle\int_{I_j} |\dot{\mathbf{v}}^*(t)|\, dt \textbf{ od}; \\
\textbf{until } \max\limits_{p \in V_P} d(\mathbf{p}) + \max\limits_{q \in V_Q} d(\mathbf{q}) \geq r - 2j\epsilon_2; \\
t_1 \leftarrow t_0 + (j-1)\Delta t;
\end{array}
\tag{4.4}
$$

Wir wollen wieder den durch die Interpolation bedingten Fehler $\epsilon_2(\Delta t)$ abschätzen:

$$
\begin{aligned}
& \left| \int_{t_0}^{t_1} |\dot{\mathbf{v}}^*(t)|\, dt - \int_{t_0}^{t_1} |\dot{\mathbf{v}}(t)|\, dt \right| \;=\; \left| \int_{t_0}^{t_1} \left(|\dot{\mathbf{v}}^*(t)| - |\dot{\mathbf{v}}(t)| \right) dt \right| \\
\leq \; & \int_{t_0}^{t_1} \left| \, |\dot{\mathbf{v}}^*(t)| - |\dot{\mathbf{v}}(t)| \, \right| dt \;\leq\; \int_{t_0}^{t_1} |\dot{\mathbf{v}}^*(t) - \dot{\mathbf{v}}(t)|\, dt
\end{aligned}
$$

Aufgrund von Gleichung (4.1) ist der Fehler $|\dot{v}_i^*(t) - \dot{v}_i(t)|$ während jedes Zeitintervalls I_j beschränkt durch

$$
\frac{1}{(k-1)!} \max_{\tau \in I_j} \left| \frac{d^k}{dt^k} v_i(\tau) \right| \Delta t^{k-1} \;\leq\; \frac{1}{(k-1)!} \max_{\tau \in [t_0, t_1]} \left| \frac{d^k}{dt^k} \mathbf{v}(\tau) \right| \Delta t^{k-1}.
$$

Daraus folgt:

$$
\begin{aligned}
\int_{t_0}^{t_1} |\dot{\mathbf{v}}^*(t) - \dot{\mathbf{v}}(t)|\, dt \;&=\; \sum_j \int_{I_j} |\dot{\mathbf{v}}^*(t) - \dot{\mathbf{v}}(t)|\, dt \\
&\leq\; \sum_j \int_{I_j} \sqrt{3} \max_{i=1\ldots3} |\dot{v}_i^*(t) - \dot{v}_i(t)|\, dt \\
&\leq\; \sqrt{3} \sum_j \int_{I_j} \frac{1}{(k-1)!} \max_{\tau \in [t_0, t_1]} \left| \frac{d^k}{dt^k} \mathbf{v}(\tau) \right| \Delta t^{k-1} dt \\
&\leq\; \frac{\sqrt{3}}{(k-1)!} \max_{\tau \in [t_0, t_1]} \left| \frac{d^k}{dt^k} \mathbf{v}(\tau) \right| \Delta t^{k-1} (t_1 - t_0)
\end{aligned}
\tag{4.5}
$$

Dieser Fehlerterm enthält zwar auch höhere Ableitungen, aber dies sind die Ableitungen der Bahnkurve $\mathbf{v}(t)$ selbst und nicht die der Funktion $|\dot{\mathbf{v}}(t)|$, was eine große Vereinfachung

darstellt, wenn man a priori Fehlerabschätzungen sucht. Außerdem wird die Berechnung der Bahngeschwindigkeit $\dot{\mathbf{v}}(t)$ überflüssig.
Leider ist die Anwendbarkeit unserer Methode auf die Fälle linearer und quadratischer Interpolation der Bahnkurve beschränkt, weil sich bereits die Bogenlänge kubischer Polynome im allgemeinen nicht explizit berechnen läßt. Im Fall der linearen Interpolation ($k = 2$) wird die Bahnkurve durch einen Polygonzug approximiert. Benutzt man pro Zeitintervall $k = 3$ Stützstellen, so besteht die Approximation aus einer Menge von Parabelstücken. Um die Bogenlänge dieser Stücke zu bestimmen, betrachten wir drei aufeinanderfolgende Punkte: $\mathbf{v}(-\frac{\Delta t}{2}) = \mathbf{a}$, $\mathbf{v}(0) = \mathbf{b}$ und $\mathbf{v}(\frac{\Delta t}{2}) = \mathbf{c}$. Die interpolierende Parabel und ihre Ableitung haben dann die folgende Form:

$$\begin{aligned}\mathbf{v}^*(t) &= \mathbf{b} + (\mathbf{c}-\mathbf{a})\frac{t}{\Delta t} + (2\mathbf{a}-4\mathbf{b}+2\mathbf{c})\Big(\frac{t}{\Delta t}\Big)^2 \\ \dot{\mathbf{v}}^*(t) &= \frac{1}{\Delta t}\Big(4(\mathbf{a}-2\mathbf{b}+\mathbf{c})\frac{t}{\Delta t} + \mathbf{c} - \mathbf{a}\Big)\end{aligned}$$

Für die Bogenlänge dieser Parabel gilt:

$$\begin{aligned}\int_{-\frac{\Delta t}{2}}^{\frac{\Delta t}{2}} |\dot{\mathbf{v}}^*(t)|dt &= \int_{-1}^{1} \Big|(\mathbf{a}-2\mathbf{b}+\mathbf{c})x + \frac{1}{2}(\mathbf{c}-\mathbf{a})\Big|dx \\ &= \frac{1}{\sqrt{\alpha}}\int_{-1}^{1}\sqrt{(\alpha x+\beta)^2+\gamma^2}dx = \frac{\gamma^2}{\alpha\sqrt{\alpha}}\int_{\frac{\beta-\alpha}{\gamma}}^{\frac{\beta+\alpha}{\gamma}}\sqrt{1+z^2}dz \\ &= \frac{\gamma^2}{2\alpha\sqrt{\alpha}}\Big[z\sqrt{1+z^2}+\ln(z+\sqrt{1+z^2})\Big]_{\frac{\beta-\alpha}{\gamma}}^{\frac{\beta+\alpha}{\gamma}} \\ \text{mit}\quad \alpha &= (\mathbf{a}-2\mathbf{b}+\mathbf{c})^2 \neq 0, \\ \beta &= \frac{1}{2}(\mathbf{c}-\mathbf{a})^T(\mathbf{a}-2\mathbf{b}+\mathbf{c}), \\ \gamma &= \frac{1}{2}|(\mathbf{c}-\mathbf{a})\times(\mathbf{a}-2\mathbf{b}+\mathbf{c})|.\end{aligned}$$

Falls $\alpha = 0$ ist, sind $\mathbf{a}$, $\mathbf{b}$ und $\mathbf{c}$ kollinear, und die gesuchte Bogenlänge ist einfach $|\mathbf{c}-\mathbf{a}|$.

4.3 Kollisionserkennung für Roboterbewegungen

Ohne zu tief in die Kinematik von Robotern einzusteigen, wollen wir die Tragfähigkeit unserer Methode aus Abschnitt 4.2 zur Approximation von Bahnkurven exemplarisch für einfache Bewegungen eines Roboters mit n Gelenken untersuchen. Dadurch wird deutlich werden, wie sie eingesetzt werden kann.

4.3.1 Kinematik eines Roboters

Im Gegensatz zur freien Bewegung starrer Körper ist die Beweglichkeit der einzelnen Glieder eines Roboters eingeschränkt. Sie können sich nicht unabhängig voneinander bewegen, sondern sie sind durch Gelenke bzw. Lager miteinander verbunden. Diese Verbindungen können

translatorischer oder rotatorischer Natur sein und besitzen in der Regel genau einen Freiheitsgrad. Die einfachste Form der Anordnung von Gliedern und Gelenken besteht aus einer offenen Kette: Für $i = 1, \ldots, n$ ist Glied $(i-1)$ durch ein Gelenk mit Glied i verbunden. n bezeichnet die Zahl der Gelenke. Das 0-te Glied der Kette sei fest montiert und das n-te Glied trage den Effektor, z.B: eine Greifhand. Damit ein solcher Roboter seine Hand im Rahmen seiner Reichweite überall in jeder gewünschten Orientierung positionieren kann, braucht er mindestens 6 Freiheitsgrade. Der 6–Gelenk–Roboter aus Abbildung 4.4 besitzt diese Eigenschaft. All seine Gelenke sind einfache Drehgelenke.

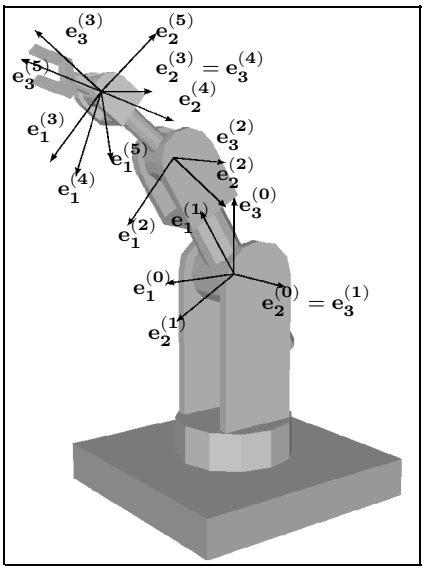


Abbildung 4.4: Koordinatensysteme eines 6–Gelenk–Roboters

Um Positionen und Orientierungen der Roboterglieder (und davon abgeleitete Größen, wie Geschwindigkeiten) elegant beschreiben zu können, ordnet man jedem Glied ein eigenes Koordinatensystem zu und bestimmt Transformationsmatrizen $\mathbf{T_i}$, die den Übergang vom i-ten zum (i–1)-ten Koordinatensystem beschreiben. (Üblicherweise entsprechen die Gelenkachsen den $\mathbf{e_3}$–Achsen dieser Koordinatensysteme.) Die Zuordnung der Koordinatensysteme zu den einzelnen Gliedern kann anhand der Lage der Gelenkachsen so geschehen, daß vier Parameter zur Spezifikation einer Matrix $\mathbf{T_i}$ genügen: die Länge l_i, die Verdrehung α_i, die Verschiebung d_i des Gliedes i, sowie der Gelenkwinkel φ_i (siehe Abbildung 4.5). Diese Größen sind unter dem Namen *Denavit–Hartenberg–Parameter* bekannt. Bei einem translatorischen Gelenk ist die Verschiebung d_i die variable Größe, bei einem Drehgelenk der Gelenkwinkel φ_i.
Die Tranformationsmatrix $\mathbf{T_i}$ ergibt sich durch Hintereinanderausführung von Drehungen und Verschiebungen, die das $(i-1)$–te Koordinatensystem in das i–te überführen (siehe [Cr89], Kapitel 3).

- Drehung um die Achse $\mathbf{e_1^{(i-1)}}$ um den Winkel α_i
- Verschiebung entlang der Richung $\mathbf{e_1^{(i-1)}}$ um l_i

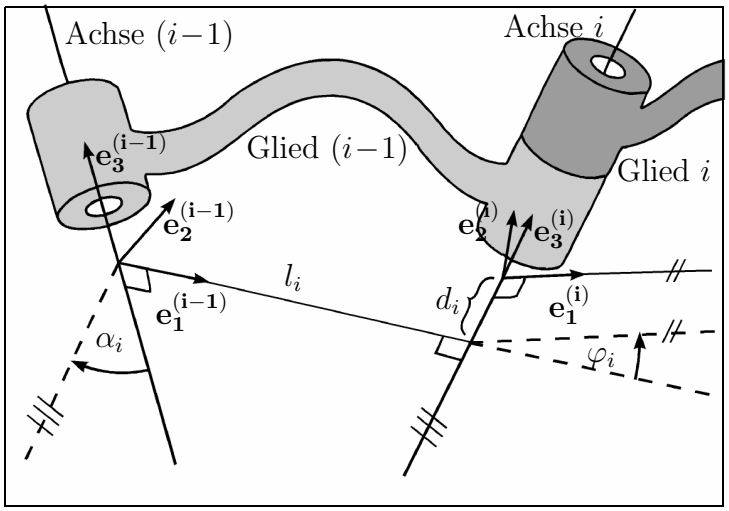


Abbildung 4.5: Denavit–Hartenberg–Parameter

- Drehung um die Achse $\mathbf{e_3^{(i)}}$ um den Winkel φ_i
- Verschiebung entlang der Richung $\mathbf{e_3^{(i)}}$ um d_i

Als Ergebnis erhält man:

$$\mathbf{T_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_i & \cos\varphi_i & -\sin\varphi_i & 0 \\ -\sin\alpha_i d_i & \cos\alpha_i \sin\varphi_i & \cos\alpha_i \cos\varphi_i & -\sin\alpha_i \\ \cos\alpha_i d_i & \sin\alpha_i \sin\varphi_i & \sin\alpha_i \cos\varphi_i & \cos\alpha_i \end{bmatrix}$$

Für die homogenen Koordinaten eines Punktes bezüglich des $(i-1)$–ten und des i–ten Koordinatensystems gilt

$$\begin{aligned} \hat{\mathbf{p}}^{(i-1)} &= \mathbf{T_i}\,\hat{\mathbf{p}}^{(i)} \quad \text{für} \quad i = 1, \ldots, n \\ \text{d.h.} \quad \hat{\mathbf{p}}^{(0)} &= \prod_{i=1}^{n} \mathbf{T_i}\,\hat{\mathbf{p}}^{(n)}. \end{aligned}$$

Dabei entspricht das 0–te Koordinatensystem dem Weltkoordinatensystem.

4.3.2 Fehlerabschätzungen bei der Approximation von Bahnkurven

Für die weiteren Betrachtungen ist es günstiger, von den homogenen zu den kartesischen Koordinaten überzugehen. Die Denavit–Hartenberg–Matrix $\mathbf{T_i}$ ist von der Form

$$\left[\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline \mathbf{s_i} & & \mathbf{R_i} & \end{array}\right] \quad \text{mit} \quad \mathbf{R_i}^T \mathbf{R_i} = \mathbf{I}.$$

Deshalb gilt für die kartesischen Koordinaten:

$$\begin{aligned} \mathbf{p}^{(i-1)} &= \mathbf{R_i}\,\mathbf{p}^{(i)} + \mathbf{s_i} \quad \text{für} \quad i = 1, \ldots, n, \\ \text{d.h.} \quad \mathbf{p} \equiv \mathbf{p}^{(0)} &= \sum_{i=0}^{n} \prod_{j=1}^{i} \mathbf{R_j}\,\mathbf{s_{i+1}} \quad \text{mit} \quad \mathbf{s_{n+1}} = \mathbf{p}^{(n)}. \end{aligned} \tag{4.6}$$

Um unser Beispiel weiter zu konkretisieren, nehmen wir an, daß unser Roboter nur rotatorische Gelenke besitzt. In diesem Fall sind nur die Gelenkwinkel φ_i variabel, die übrigen Parameter l_i,α_i und d_i sind konstant. Die Gelenke sollen sich unabhängig voneinander, aber gleichförmig bewegen. Das heißt: $\varphi_i(t) = \omega_i\, t + \phi_i$. Die dabei auftretenden Winkelgeschwindigkeiten ω_i sollen einen Maximalwert ω_{max} nicht überschreiten.
Unser Ziel ist es, den Approximationsfehler für die Bogenlänge der Bahnkurven abzuschätzen, die bei dieser Form der Roboterbewegung entstehen. Dazu greifen wir uns einen Punkt $\mathbf{p}^{(n)}$ heraus, der auf dem n–ten Glied des Roboters liegt. Gleichung (4.6) beschreibt seine Bahnkurve in Weltkoordinaten. Gemäß Gleichung (4.5) sind die höheren Ableitungen der Bahnkurve wesentlich für das Fehlerverhalten unserer Approximation.

Lemma 4.5

$$\left|\frac{d^k}{dt^k}\mathbf{p}(t)\right| \;\leq\; \omega_{max}^k \sum_{i=1}^{n} i^k |\mathbf{s_{i+1}}|$$

Beweis: Aus Gleichung (4.6) folgt:

$$\left|\frac{d^k}{dt^k}\mathbf{p}(t)\right| \;\leq\; \sum_{i=0}^{n} \left\|\frac{d^k}{dt^k} \prod_{j=1}^{i} \mathbf{R_j}(t)\right\| |\mathbf{s_{i+1}}|$$

Dabei steht $\|.\|$ für die zur euklidischen Vektornorm $|.|$ gehörige natürliche Matrizennorm. Diese Norm angewendet auf eine orthonormale Matrix ergibt den Wert 1. $\mathbf{R_j}$ ist ein Produkt zweier orthonormaler Matrizen, und zwar ist

$$\mathbf{R_j}(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_j & -\sin\alpha_j \\ 0 & \sin\alpha_j & \cos\alpha_j \end{bmatrix} \cdot \begin{bmatrix} \cos(\omega_j t + \phi_j) & -\sin(\omega_j t + \phi_j) & 0 \\ \sin(\omega_j t + \phi_j) & \cos(\omega_j t + \phi_j) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\Longrightarrow \left\|\frac{d^l}{dt^l}\mathbf{R_j}(t)\right\| = |\omega_j|^l \qquad \forall\, t$$

$\frac{d^k}{dt^k}\prod_{j=1}^{i}\mathbf{R_j}(t)$ besteht aus i^k Summanden der Form $\prod_{j=1}^{i}\frac{d^{k_j}}{dt^{k_j}}\mathbf{R_j}(t)$ mit $\sum_j k_j = k$. Deshalb gilt:

$$\left\|\frac{d^k}{dt^k}\prod_{j=1}^{i}\mathbf{R_j}(t)\right\| \;\leq\; (i\omega_{max})^k$$

■

4.3.3 Ein Beispiel

Wir betrachten den 6–Gelenk Roboter aus Abbildung 4.4 mit den folgenden Denavit–Hartenberg–Parametern:

i	α_i	l_i	ϕ_i	d_i
1	$0°$	0.00	$0°$	0.00
2	$-90°$	0.00	$300°$	0.00
3	$0°$	0.70	$110°$	0.00
4	$90°$	0.00	$0°$	0.49
5	$-90°$	0.00	$20°$	0.00
6	$90°$	0.00	$40°$	0.00

Wir wollen die Bahnkurve des Greifpunktes $\mathbf{p}$ verfolgen. Bezüglich des 6–ten Koordinatensystems lauten seine Koordinaten $\mathbf{p}^{(6)} = (0.01, 0, 0)^T$.
Die im vorigen Abschnitt vorgestellten Algorithmen zur Kollisionserkennung können kompliziertere Bewegungsgleichungen wie (4.6) nicht direkt benutzen. Statt dessen interpolieren sie die Bahnkurve stückweise mit Polynomen $\mathbf{p}^*(t)$ (kleinen Grades). Für unsere Algorithmen war zum einen der Interpolationsfehler $\epsilon_1 = \max_{t\in[t_0,t_1]} |\mathbf{p}^*(t) - \mathbf{p}(t)|$ von Bedeutung, zum anderen der Fehler für die approximierte Bogenlänge $\epsilon_2 = \left| \int_{t_0}^{t_1} |\dot{\mathbf{p}}^*(t)|\, dt - \int_{t_0}^{t_1} |\dot{\mathbf{p}}(t)|\, dt \right|$. Wir wollen ϵ_1 exemplarisch für die lineare Interpolation abschätzen und ϵ_2 für die quadratische, wobei wir eine Schrittweite von Δt annehmen.
$\Delta\varphi_{max} = \omega_{max}\Delta t$ bezeichne den größten Winkelvorschub während eines Zeitintervalls der Länge Δt. Mit Hilfe von Lemma 4.5 und den Gleichungen (4.3) und (4.5) erhalten wir:

$$\begin{aligned}
\epsilon_1 &= \frac{\sqrt{3}}{2}\Delta t^2\omega_{max}^2 \sum_{i=1}^{6} i^2 |\mathbf{s_{i+1}}| \\
&= \frac{\sqrt{3}}{2}\Delta\varphi_{max}^2 \left(2^2 l_3 + 3^2 d_4 + 6^2 |\mathbf{p}^{(6)}|\right) \\
&\approx 6.6\Delta\varphi_{max}^2 \\
\epsilon_2 &= \frac{\sqrt{3}}{2}\Delta t^3\omega_{max}^3 \sum_{i=1}^{6} i^3 |\mathbf{s_{i+1}}| \\
&= \frac{\sqrt{3}}{2}\Delta\varphi_{max}^3 \left(2^3 l_3 + 3^3 d_4 + 6^3 |\mathbf{p}^{(6)}|\right) \\
&\approx 18.2\Delta\varphi_{max}^3
\end{aligned}$$

■

Dieses Ergebnis zeigt, daß sich die Interpolationsfehler gutartig verhalten. Wir haben dabei vorausgesetzt, daß die Bewegungen der Roboterglieder kontinuierlich verlaufen und im Konfigurationsraum (der Gelenkwinkel) stückweise linear sind. Außerdem haben wir angenommen, daß eine Obergrenze für die Winkelgeschwindigkeiten der Gelenke ω_{max} existiert. ϵ_1 stellt eine a priori Abschätzung für den Interpolationsfehler dar, der ensteht wenn man die exakte Bahnkurve mit Hilfe von Geradenstücken interpoliert. Wenn durch die Diskretisierung die Zeitintervalle so klein gewählt werden, daß die Gelenkwinkeländerungen pro Zeitintervall

höchstens $\Delta\varphi_{max} = \omega_{max}\,\Delta t$ betragen, so fällt ϵ_1 quadratisch in $\Delta\varphi_{max}$. Der Approximationsfehler ϵ_2 für die Bahnlängenbestimmung mit quadratischer Interpolation sinkt sogar in der dritten Potenz in $\Delta\varphi_{max}$.
Mit Hilfe dieser Methode können die Interpolationsfehler ϵ_1 und ϵ_2, die in den Algorithmen (4.2) und (4.4) zur Kollisionserkennung zu berücksichtigen sind, für jedes Glied direkt bestimmt werden. Wie das Beispiel und Lemma 4.5 zeigen benötigt man dazu nur die Denavit–Hartenberg–Parameter und den größten Abstand, den ein Punkt des jeweiligen Gliedes zum Gliedkoordinatensystem hat.

Kapitel 5

Systembeschreibung

Unser System zur *Interaktiven Montagesimulation* ist als Teil eines komplexen CAE–Systems (computer–aided engineering) konzipiert. In einem CAE–System vollzieht sich die Herstellung eines Produktes in mehreren aufeinanderfolgenden Stufen: dem Entwurf (CAD), der Produktionsplanung (CAP), der Fertigung (CAM) und der Qualitätsprüfung (CAQ). In diesem Schema ist unser System unmittelbar nach dem Entwurf anzusiedeln und als Vorbereitung für die Produktionsplanung gedacht. Es dient dem Zweck, Bewegungsabläufe bei Montageprozessen zu simulieren, um dadurch die grundsätzliche Durchführbarkeit der Montage eines Produktes unter Beweis zu stellen und dabei Montagepläne zu erstellen oder nachträglich zu optimieren. Die Einbeziehung von Robotern und sonstigen Montagewerkzeugen in die Montagesimulation stellt bereits einen Schritt in Richtung Produktionsplanung dar. Ziel unseres Systems ist es, bereits sehr früh im Herstellungsprozeß eines Produktes Probleme zu erkennen, die eine einfache Montage erschweren oder sogar verhindern. Der Entwicklungsingenieur versucht in Interaktion mit dem System einen Montageprozeß zu planen und in einer Simulation die kollisionsfreie Durchführbarkeit zu verifizieren. Die Erkenntnisse, die er dabei gewinnt, können im Idealfall direkt in die Produktionsplanung einfließen. Unter Umständen ist aber auch ein montagegerechterer Entwurf der Bauteile des Produktes notwendig.

5.1 Systemübersicht

Im folgenden wollen wir unser System zur *Interaktiven Montagesimulation* kurz in einer Übersicht beschreiben und dabei die Rolle der verschiedenen Komponenten und ihr Zusammenspiel anhand von Abbildung 5.1 erläutern. In den einzelnen Abschnitten des Kapitels gehen wir dann näher auf die Teilkomponenten ein.
Zur Konzeption eines Produktes gehört ein aufeinander abgestimmter Entwurf seiner Einzelteile, um die Montierbarkeit aller Teile zu gewährleisten. Während des *Objektentwurfs* werden nicht nur Daten zur Modellierung der Einzelteile selbst erzeugt, sondern auch Informationen über ihre räumliche Anordnung im Endprodukt. Der Konstrukteur muß während des Entwurfs zumindest eine ungefähre Vorstellung davon haben, wie die Einzelteile zusammengefügt werden können. Unser Simulationssystem soll ihm dabei helfen, seine Vorstellung zu präzisieren und vor allem auf ihre kollisionsfreie Durchführbarkeit hin zu überprüfen.

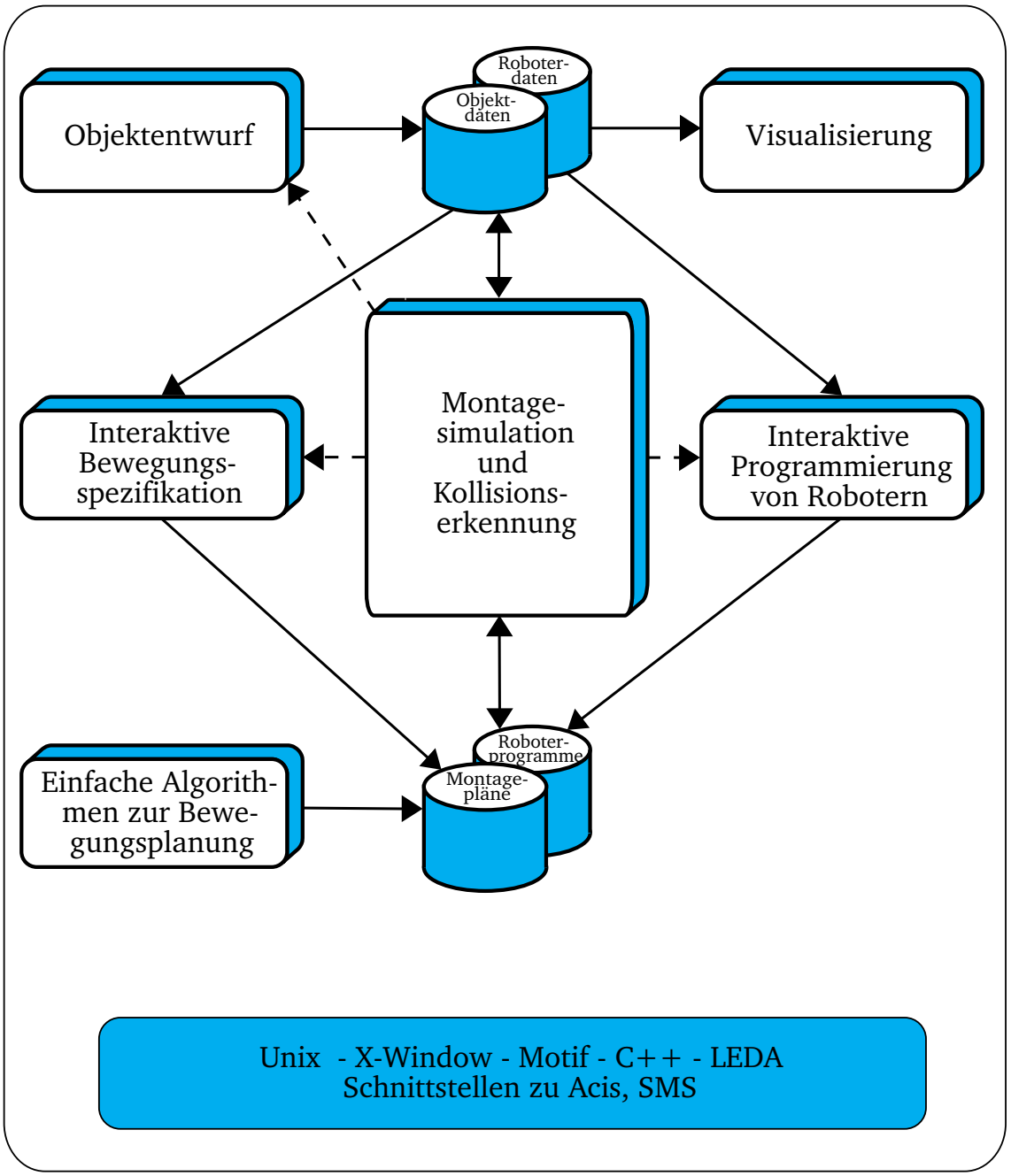


Abbildung 5.1: Zusammenspiel der Systemkomponenten

Diese Form der Montageplanung kann er zunächst auf der alleinigen Basis der *Objektdaten* ausführen, um zum Beispiel eine günstige Montagereihenfolge für die Einzelteile festzulegen und einen einfachen *Montageplan* zu erstellen. Dazu kann er *Bewegungen interaktiv spezifizieren* und gleichzeitig ihre *Kollisionsfreiheit kontrollieren*. In einfachen Situationen kann er auch automatische *Verfahren zur Bewegungsplanung* benutzen. Vielleicht stellt er bereits in diesem Stadium fest, daß Bauteile des Produktes modifiziert werden müssen, um eine einfachere Montage zu ermöglichen.

Für die Planung einer maschinellen Produktion eines Produktes sind die Resultate einer solchen Montagesimulation nicht aussagekräftig genug, weil sie nur die prinzipiell durchzuführenden Montageschritte beinhalten. Vollkommen ohne Berücksichtigung bleiben dabei der Einsatz von Montagewerkzeugen wie Robotern und die daraus resultierenden Nebenbedingungen, die die Beweglichkeit von Einzelteilen stark einschränken und eine Montage verhindern können. Deshalb sind in unserem Simulationssystem auch die Produktionswerkzeuge (Roboter) integriert. Zu ihrer Modellierung müssen sowohl die geometrischen als auch die kinematischen Verhältnisse bekannt sein. Für eine spätere robotergestützte Fertigung sind auch dynamische Aspekte wichtig, aber aus Effizienzgründen lassen wir sie hier unberücksichtigt, um eine interaktive Arbeitsweise mit dem System nicht zu beeinträchtigen. Unser System stellt dem Ingenieur eine graphische Oberfläche zur Verfügung, mit der er interaktiv die zur Montage eines Produktes notwendigen Roboterprogramme erstellen kann. Bereits bei der Programmierung der einzelnen Aktionen macht das System den Monteur auf *erkannte Kollisionen* aufmerksam. Montageanweisungen in Form von *Roboterprogrammen* sind das Ergebnis der Montagesimulation und können zur Produktionsplanung direkt verwendet werden.

Die Teilkomponenten unseres Systems sind:

- **Objektentwurf**

  Die Bestandteile des zu montierenden Produktes und der Roboter wollen wir der Einfachheit halber als starre Körper betrachten und durch Polyeder modellieren. Das heißt, wir beschreiben ihre Oberfläche durch ebene, geradlinig begrenzte Flächenstücke. Diese Randbeschreibung repräsentiert die *Objektdaten*, die dem Montagesystem zur Verfügung stehen. Neben der reinen geometrischen Form spielt im Zusammenhang von gelenkig verbundenen Objekten auch der Gelenktyp und die Lage der Gelenkachsen eine Rolle. Diese Informationen werden bei der Robotersimulation benötigt und sind hier unter dem Begriff *Roboterdaten* zusammengefaßt.

  Unser System selbst verfügt nur über sehr einfache Möglichkeiten zum Entwurf von Objekten (siehe [Co94]). Es existieren aber Schnittstellen zu kommerziellen CAD–Systemen (dem Robotersimulationssystem *SMS* der Firma Siemens/Nixdorf und dem geometrischen Modellierer *ACIS* der Firma Spatial Technology), um Objekt– und Roboterdaten zu importieren.

- **Visualisierung**

  Zur graphischen Darstellung einer Szene aus Einzelteilen, die eine Folge von Montageschritten durchlaufen sollen, dient das Modul *Visualisierung*. Es bietet unter anderem die Möglichkeit zur Festlegung der Perspektive und des Zeichenmodus, sei es eine

einfache Darstellung der Polyeder durch Drahtmodelle oder eine realistischere Darstellung, die verdeckte Flächen und Lichtverhältnisse berücksichtigt oder Stereosehen ermöglicht. Außer der Visualisierung statischer Szenen muß natürlich auch eine Animation der Bewegungsabläufe möglich sein.

Bei der Implementierung wurden nur Graphikroutinen aus dem 2D–Bereich verwendet, weil die Standardsoftware (z.B. PHIGS) zur Visualisierung animierter dreidimensionaler Szenen auf den zur Verfügung stehenden Workstations nicht leistungsfähig genug war. Die benutzten Verfahren werden in Abschnitt 5.2 genauer beschrieben.

- **Interaktive Bewegungsspezifikation**

  Ein Teil der Benutzeroberfläche ist die *Interaktive Bewegungsspezifikation*. Sie gestattet einem Monteur nicht nur die Festlegung von Positionen und Orientierungen für einzelne Körper im Raum, sondern dient in erster Linie der Definition von Bewegungsabläufen. Die Bewegungen der Einzelteile modellieren wir durch Folgen von Translationen und Rotationen, wodurch alle Freiheitsgrade eines starren Körpers abgedeckt sind. Bei der Erstellung von *Montageplänen* werden alle wesentlichen Zwischenstadien einer Montage dokumentiert und damit der exakte Bewegungsablauf aller an der Montage beteiligten Objekte beschrieben. Details hierzu finden sich in Abschnitt 5.3.

- **Interaktive Programmierung von Robotern**

  Zur Spezifikation von Bewegungsabläufen bei Robotern können wir neben einer textuellen auch eine graphisch orientierte Form der Roboterprogrammierung verwenden (vgl. Abschnitt 5.5). Die Möglichkeit zur Programmierung der Roboter setzt natürlich die Kenntnis eines entsprechenden kinematischen Modells des Roboters voraus, d.h. wie man aus den Gelenkwinkeln Position und Orientierung des Endeffektors berechnet und umgekehrt sowie einfache Trajektorien für Punkt–zu–Punkt–Bewegungen bestimmt. Die erstellten Roboterprogramme bestehen aus einer Folge von Kommandos zur Positionierung und Orientierung des Endeffektors, zum Greifen von Objekten und zur Auswahl von Bewegungsmodi.

- **Montagesimulation und Kollisionserkennung**

  Herzstück des Systems zur *Interaktiven Montagesimulation* 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 algorithmischen Grundlagen für eine effiziente Umsetzung wurden in den vorangegangenen Kapiteln erarbeitet.

- **Einfache Algorithmen zur Bewegungsplanung**

  Dieses Modul stellt heuristische Werkzeuge zur automatischen Generierung von Wegen zur Verfügung, so daß nach Vorgabe von Start– und Zielkonfiguration eine einfache Trajektorienplanung stattfinden kann. Ihr Einsatz beschränkt sich jedoch auf die Spezialfälle, in denen das bewegte Objekt nur über wenige Freiheitsgrade verfügt. In diesem Zusammenhang können die Resultate der Kollisionsberechnungen aus den vorherigen Kapiteln genutzt werden. Die Einzelheiten sind in Abschnitt 5.4 beschrieben.

Die Implementierung des Gesamtsystems basiert auf dem Betriebssystem *Unix* und der graphischen Oberfläche *X–Window* des MIT. Zur Gestaltung der Benutzeroberfläche wurde *Motif* benutzt. Alle Teilkomponenten wurden in der objektorientierten Programmiersprache *C++* realisiert. Außerdem wurde das Softwarepaket *Leda* vom MPI verwendet. Auf diese Weise ist die Portabilität des Systems auf die meisten graphischen Workstations gewährleistet.

5.2 Visualisierung

5.2.1 Parallel– und Zentralprojektion

Zu einer allgemeinen Sichtspezifikation gehört neben der Angabe von Projektionsebene und Projektionszentrum auch eine Abgrenzung des sichtbaren Raumgebietes. Wir benutzen zur Festlegung der Sicht der Einfachheit halber nur einen Vektor $\mathbf{r}$. Dieser gibt sowohl die Lage des Blickpunktes (Projektionszentrums) an, als auch den Normalenvektor $\mathbf{r}^\circ$ der Projektionsebene, die durch den Ursprung verlaufen soll. In der Projektionsebene führen wir ein eigenes Koordinatensystem ein, das dem Bildschirmkoordinatensystem entspricht (vgl. Abbildung 5.2). Dazu wählen wir zwei orthonormale Vektoren $\mathbf{e'_1}$ und $\mathbf{e'_2}$, die zusammen mit $\mathbf{e'_3} = \mathbf{r}^\circ$ ein Rechtssystem, das Sichtkoordinatensystem, bilden. Um eine natürliche Sicht zu gewährleisten, wollen wir die $\mathbf{e_3}$–Richtung des Weltkoordinatensystems auf die $\mathbf{e'_2}$–Richtung des Sichtkoordinatensystems projizieren. Deshalb definieren wir

$$\begin{aligned} \mathbf{e'_3} &:= \mathbf{r}^\circ, \\ \mathbf{e'_1} &:= (\mathbf{e_3} \times \mathbf{r}^\circ)^\circ, \\ \mathbf{e'_2} &:= \mathbf{r}^\circ \times \mathbf{e'_1}, \end{aligned}$$

falls $\mathbf{r}^\circ \neq \mathbf{e_3}$, ansonsten setzen wir $\mathbf{e'_i} = \mathbf{e_i}$ für $i = 1, 2, 3$. Die Weltkoordinaten lassen sich mittels der Transformation

$$\mathbf{x'} \;=\; \mathbf{V}\,\mathbf{x} \quad \text{mit} \quad \mathbf{V} = [\mathbf{e'_1}, \mathbf{e'_2}, \mathbf{e'_3}]^T$$

ins Sichtkoordinatensystem übertragen.
Zur Spezifikation des Projektionszentrums $\mathbf{r}$ verwenden wir Kugelkoordinaten (r, ϑ, φ). r gibt den Abstand des Blickpunktes vom Ursprung an, ϑ den Polarwinkel (den Winkel zwischen $\mathbf{r}$ und der x_3–Achse) und φ den Azimut (den Winkel zwischen der Projektion von $\mathbf{r}$ auf die x_1x_2–Ebene und der x_1–Achse).

$$\mathbf{r} \;=\; r \begin{bmatrix} \sin\vartheta\cos\varphi \\ \sin\vartheta\sin\varphi \\ \cos\vartheta \end{bmatrix} \quad \text{wobei} \quad \begin{array}{l} -180^\circ < \varphi \leq 180^\circ \\ -90^\circ < \vartheta \leq 90^\circ \end{array}$$

Die Matrix $\mathbf{V}$, die die Transformation vom Weltkoordinatensystem ins Sichtkoordinatensystem beschreibt, hat folgende Form:

$$\mathbf{V} \;=\; \begin{bmatrix} -\sin\varphi & \cos\varphi & 0 \\ -\cos\vartheta\cos\varphi & -\cos\vartheta\sin\varphi & \sin\vartheta \\ \sin\vartheta\cos\varphi & \sin\vartheta\sin\varphi & \cos\vartheta \end{bmatrix}$$

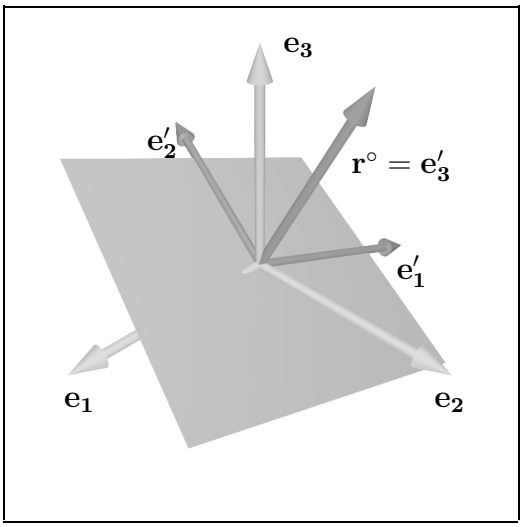


Abbildung 5.2: Sichtkoordinatensystem

Durch Weglassen der dritten Vektorkomponente im Sichtkoordinatensystem erhält man die Parallelprojektion in Richtung $\mathbf{r}^\circ$:

$$x''_1 = x'_1, \qquad x''_2 = x'_2, \qquad x''_3 = 0$$

Bei der Zentralprojektion ermitteln wir die Koordinaten eines projizierten Punktes $\mathbf{x}''$ mit Hilfe des Strahlensatzes (vgl. Abbildung 5.3). Das Projektionszentrum hat im Sichtkoordinatensystem die Koordinaten $(0, 0, |\mathbf{r}|)^T$, und es gilt:

$$x''_i = \frac{|\mathbf{r}|}{|\mathbf{r}| - x'_3} x'_i \quad \text{für } i = 1, 2 \quad \text{und } x''_3 = 0$$

Dabei muß natürlich gelten, daß der Punkt $\mathbf{x}'$ außerhalb der Brennebene liegt ($x'_3 \neq |\mathbf{r}|$). Als Grenzwert für $|\mathbf{r}| \to \infty$ geht die Zentralprojektion in die Parallelprojektion über.
Beim Technischen Zeichnen verwendet man oft die durch eine DIN–Norm festgelegte isometrische bzw. dimetrische orthogonale Projektion. Beide sind Parallelprojektionen auf eine Projektionsebene senkrecht zur Projektionsrichtung. Sie zeichnen sich dadurch aus, daß die Länge von Linien parallel zur x_1–, x_2– und x_3–Achse im Verhältnis $a : 1 : 1$ abgebildet werden ($a = 1$ für isometrische Projektion und $a = \frac{1}{2}$ für dimetrische Projektion). Für die Projektionen der drei Einheitsvektoren des Weltkoordinatensystems muß somit gelten:

$$\frac{1}{a}|\mathbf{PVe_1}| = |\mathbf{PVe_2}| = |\mathbf{PVe_3}| \qquad \text{mit} \quad \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\Longrightarrow \quad \frac{1}{a^2}(\sin^2\varphi + \cos^2\vartheta\cos^2\varphi) = \cos^2\varphi + \cos^2\vartheta\sin^2\varphi = \sin^2\vartheta$$

$$\Longrightarrow \quad \sin\varphi = \pm\frac{a}{\sqrt{2}}, \quad \sin\vartheta = \pm\sqrt{\frac{2}{2+a^2}}$$

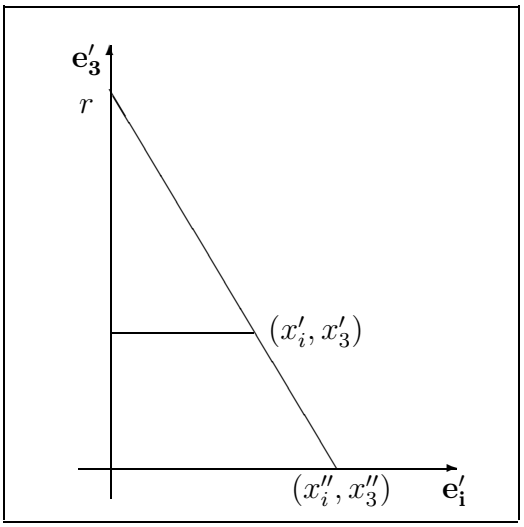


Abbildung 5.3: Zentralprojektion

Um eine isometrische Projektion ($a = 1$) zu erreichen, wählen wir $\varphi = \pm 45° + z\,90°$ und $\vartheta \approx \pm 54.74°$ für $z \in \{-1, 0, 1\}$. Bei der dimetrischen Projektion ($a = \frac{1}{2}$) wählen wir $\varphi \approx \pm 20.7° + z\,90°$ und $\vartheta \approx \pm 70.53°$ für $z \in \{-1, 0, 1\}$.

Bemerkung:

Schiefwinklige Projektionen wie z.B. die Kavalliersperspektive können wir durch unsere Spezifikation der Sicht nicht erfassen, denn hier steht die Projektionsrichtung nicht mehr senkrecht auf der Projektionsebene.

Um eine benutzerfreundliche Möglichkeit zur Sichtspezifikation zu schaffen, verwenden wir numerische Eingabefelder für die Kugelkoordinaten (r, φ, ϑ) mit De- und Inkrementierungsmöglichkeiten, kombiniert mit der Abbildung einer Kugel ("Trackball"). Zur Unterstützung der Intuition unterteilen wir die Kugel durch Längen– und Breitengrade und projizieren ihre Vorderseite immer entsprechend der momentanen Perspektive. Durch Angabe eines Punktes $\mathbf{x}''$ auf der projizierten Kugel kann der Benutzer sehr leicht den Polarwinkel ϑ und den Azimut φ festlegen (siehe Abbildung 5.4).

Wenn wir voraussetzen, daß $\mathbf{x}''$ durch Parallelprojektion eines Punktes $\mathbf{x}'$ auf der Kugeloberfläche hervorgeht und daß die Kugel den Radius 1 ($|\mathbf{x}'| = 1$) hat, erhalten wir die gesuchten Kugelkoordinaten wie folgt:

$$
\begin{aligned}
& x'_1 = x''_1,\ x'_2 = x''_2,\ x'_3 = \sqrt{1 - x''^2_1 - x''^2_2}, \\
& \mathbf{x} = \mathbf{V}^T\,\mathbf{x}', \\
& \varphi = \arctan(x_2, x_1), \qquad \vartheta = \arccos(x_3).
\end{aligned}
$$

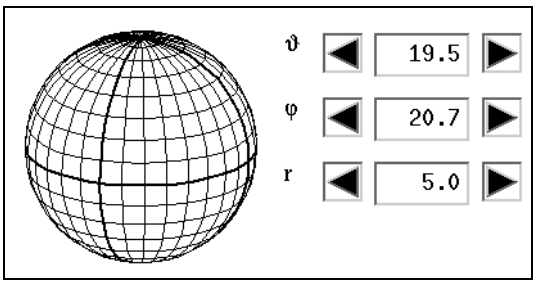


Abbildung 5.4: Sichtspezifikation

5.2.2 Darstellungsmodi

Die einfachste Möglichkeit zur Visualisierung einer Szene von Polyedern besteht darin, nur die Kanten der Polyeder auf die Bildebene zu projizieren. Diese Form der Darstellung ist zwar sehr schnell, liefert aber nur Drahtmodelle der Polyeder, die unter Umständen keine eindeutige Interpretation der Lage der Polyederflächen zulassen. Eine kleine Verbesserung ergibt sich bereits, wenn nur die Kanten der Flächen des Polyeders gezeichnet werden, die dem Betrachter zugewandt sind. Dies sind die Flächen, deren Flächennormalen mit dem Ortsvektor des Projektionszentrums einen Winkel $< 90°$ einschließen, wenn wir uns an die Konvention halten, daß die Flächennormalen ins Äußere der Polyeder gerichtet sind. Aber auch diese Technik kann nicht verhindern, daß Begrenzungskanten von Flächen gezeichnet werden, die in Wirklichkeit von anderen Flächen verdeckt werden. Diesen Schwachpunkt kann man erst mit Hidden–Line– bzw. Hidden–Surface–Algorithmen beseitigen. Die Verfahren, die sich auf die Elimination verdeckter Teile der Begrenzungskanten beschränken, haben den Nachteil, daß sie zur flächenfüllenden, farbigen Darstellung nur bedingt geeignet sind.

5.2.3 Binary Space Partitioning Trees

Für eine flächenorientierte Vorgehensweise bietet sich folgende Methode an (Painter–Algorithmus):

> Zeichne alle Flächen der Szene in einer solchen Reihenfolge, daß die vom Blickpunkt am weitesten entfernten Flächen zuerst dargestellt werden und später von den näherliegenden überdeckt werden. (Als Ordnungskriterium kann man beispielsweise den Abstand zwischen Blickpunkt und Flächenschwerpunkt wählen.)

Bei zyklischer Überlappung von Flächen scheitert dieses Verfahren natürlich, weil eine solche Reihenfolge nicht existiert. Um dem Abhilfe zu schaffen, kann man die betroffenen Flächenstücke durch Zerschneiden in mehrere Teile zerlegen. In [FKN80] und später in [PY89] wird ein rekursives Bisektionsverfahren diskutiert, das mittels ebener Schnitte den Raum und die darin enthaltenen Flächen der Szene so partitioniert, daß man für jeden beliebigen Blickpunkt die Reihenfolge zum Zeichnen der Teilflächen sofort ablesen kann. Wir wollen die grundlegende Datenstruktur und Algorithmen des Verfahrens kurz erläutern. Zur Un-

terstützung der Intuition soll das 2–dimensionale Beispiel in den Abbildungen 5.5 und 5.6 dienen.

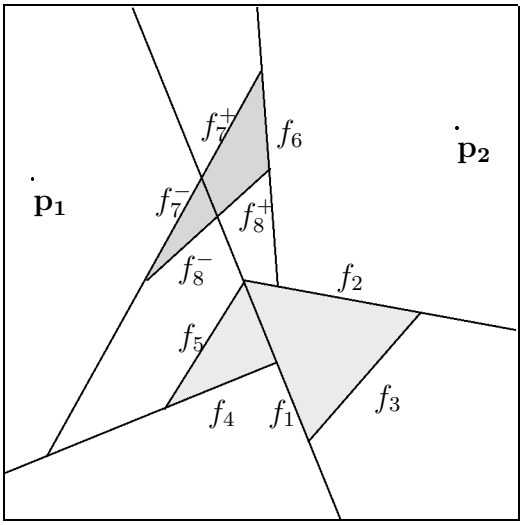


Abbildung 5.5: Ein 2D–Binary Space Partitioning Tree

Gegeben sei eine räumliche Szene bestehend aus polygonalen Flächen. Die gewünschte Partitionierung wird wie folgt konstruiert: Durch eine orientierte Ebene H unterteilt man den zu betrachtenden Raum S in zwei Teilräume $S^{\pm} = S \cap H^{\pm}$. Jede Fläche $f \subset S$ klassifiziert man bezüglich ihrer Mitgliedschaft zu H^- und H^+. Wird f von H zerschnitten, so ersetzen wir f durch die entstandenen Teile $f^{\pm} = f \cap H^{\pm}$. In den beiden Teilräumen $S^{\pm}$ setzen wir das Verfahren mit den darin enthaltenen Flächen rekursiv fort. Die Rekursion stoppt, wenn pro Teilraum nicht mehr als ein Flächenstück vorkommt. Der Algorithmus zur graphischen Darstellung entscheidet, ob der Blickpunkt $\mathbf{p}$ in H^- oder H^+ liegt. Bevor die Flächen des Raumes dargestellt werden, in dem $\mathbf{p}$ lokalisiert wurde, werden zuerst die des anderen Raumes gezeichnet. Dies geschieht jeweils rekursiv.
Während der Partitionierung hat man eine gewisse Freiheit bei der Wahl der Ebene H. Je weniger Flächen sie schneidet, um so günstiger wirkt sich dies auf die Größe der Datenstruktur aus. Da eine Analyse der Szene zur bestmöglichen Wahl von H zu aufwendig wäre, beschränkt man sich auf solche Partitionierungsebenen H, die eine Fläche f der Szene enthalten. Das stellt zwar eine große Einschränkung des Suchraums für H dar, hat sich aber als sehr einfache und effiziente Möglichkeit erwiesen.
Die gerade beschriebene Autopartitionierung läßt auch folgende inkrementelle Strategie zur Partitionierung zu: Dazu nehmen wir an, daß für eine Reihe von Flächen schon ein Partitionierungsbaum aufgebaut worden ist und eine weitere Fläche f eingefügt werden soll. In jedem Knoten ν des Baumes speichern wir neben der Ebene H_ν zwei Listen $L_\nu^{\pm}$ ab, die zur Aufnahme der Flächen dienen, die vollständig in der Ebene H_ν liegen. (Wenn wir orientierte Flächen betrachten, legen wir in L^+ die Flächen ab, die die gleiche Orientierung wie H_ν haben, in L^- die anderen.) Das Einfügen einer Fläche f startet in der Wurzel des Baumes. An einem Knoten ν wird anhand der Ebene H_ν entschieden, ob f dort abgelegt

werden kann oder ob f im linken oder/und rechten Teilbaum des Knotens ν eingefügt werden muß. Der genaue Ablauf zum Aufbau des Partitionierungsbaumes ist in der im folgenden aufgeführten Prozedur $insert(Node\ \nu, Face\ f)$ dokumentiert, das Zeichnen der Szene vom Blickpunkt $\mathbf{p}$ aus geschieht mit der Prozedur $render(Node\ \nu, Point\ \mathbf{p})$. Beide Prozeduren werden an der Wurzel des Partitionierungsbaumes gestartet. In dem 2–dimensionalen Beispiel aus Abbildung 5.5 wurden die Flächen in der Reihenfolge $f_1, f_2, \ldots, f_8$ eingefügt. Dabei mußte sowohl f_7 als auch f_8 gesplittet werden. In Abbildung 5.6 wurden im zugehörigen Partitionierungsbaum zusätzliche Blätter ergänzt, die die verschiedenen Gebiete der Szene wiedergeben. Zum Zeichnen der Flächen vom Blickpunkt $\mathbf{p_1}$ erhält man die Reihenfolge $(f_3), f_2, (f_6), (f_8^+), f_7^+, f_1, (f_4), f_5, (f_8^-), f_7^-$ und von $\mathbf{p_2}$ aus $(f_4), (f_7^-), f_8^-, (f_5), (f_1), (f_3), f_2, (f_7^+), f_8^+, f_6$. Die in Klammern erscheinenden Flächen sind vom Blickpunkt abgewandt.

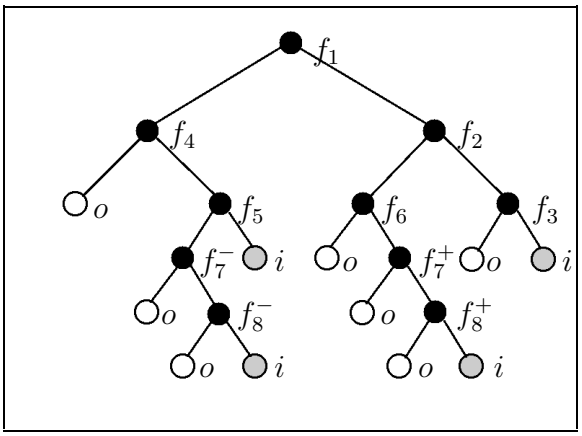


Abbildung 5.6: Der zugehörige Partitionierungsbaum

proc $insert(Node\ \nu, Face\ f)$
  if $(f \subset H_\nu)$ **then**
    if $(\mathbf{n}_\nu{}^T \mathbf{n}_f > 0)$ **then** $L_\nu^+ \leftarrow L_\nu^+ \cup \{f\}$
    else $L_\nu^- \leftarrow L_\nu^- \cup \{f\}$ **fi**;
    return
  fi;
  $f^- \leftarrow f \cap H_\nu^-$;
  $f^+ \leftarrow f \cap H_\nu^+$;
  if $(f^- \neq \emptyset)$ **then**
    if $(\nu^- =$ **nil**$)$ **then** $\nu^- \leftarrow$ **new** $Node(f^-)$ **fi**;
    $insert(\nu^-, f^-)$
  fi;
  if $(f^+ \neq \emptyset)$ **then**

```
      if (ν⁺ = nil) then ν⁺ ← new Node(f⁺) fi;
      insert(ν⁺, f⁺)
    fi
  corp


  proc render(Node ν, Point p)
    if (ν = nil) then return fi;
    if (p ∈ H_ν⁺) then
      render(ν⁻, p);
      forall f ∈ L_ν⁺ do draw(f) od;
      render(ν⁺, p)
    else
      render(ν⁺, p);
      forall f ∈ L_ν⁻ do draw(f) od;
      render(ν⁻, p)
    fi
  corp
```

Die Reihenfolge, in der die Flächen eingefügt werden, kann die Größe des resultierenden Baumes entscheidend beeinflussen. Bei der Komplexitätsbetrachtung nehmen wir an, daß die Szene aus n Dreiecken besteht. Im schlechtesten Fall verbraucht man soviel Platz wie zur Speicherung eines Arrangements aus n Ebenen nötig wäre, also $O(n^3)$. Wählt man eine zufällige Reihenfolge, so wird man im Mittel mit sehr viel weniger Platz und Zeit auskommen. In [PY89] wird gezeigt, daß bei dieser probabilistischen Strategie die Größe der Partitionierung im Mittel bei $O(n^2)$ liegt, und zwar unabhängig von der Lage der n Dreiecke.
Daß die Beschränkung auf Autopartitionierungen nicht so kritisch ist, verdeutlicht das Beispiel in Abbildung 5.7, das ebenfalls aus [PY89] stammt. Es zeigt, daß man auch bei Partitionierungen, die sich nicht ausschließlich an den Flächen der Szene orientieren, mit einem Aufwand in der Größenordnung von $n^{1.5}$ rechnen muß. In dem Beispiel sind $n = 3k^2$ Stäbe entlang der Achsen eines 3–dimensionalen Gitters angeordnet. Dadurch entstehen $k^3 = \Theta(n^{1.5})$ lokale Situationen, in denen sich je drei Stäbe zyklisch überlappen, was je eine Zerschneidung erfordert.
Für die Familie der Stabanordnungen $(n = 1, \ldots, 20)$ haben wir die Laufzeit des implementierten Algorithmus experimentell überprüft. Das Diagramm in Abbildung 5.8 zeigt den Logarithmus der Laufzeit t, gemessen in Sekunden, aufgetragen gegenüber dem Logarithmus der Flächenzahl.
Die Zahl f der Flächen der Szenen reichte von 18 bis 7200. Der zu erwartende lineare Zusammenhang zwischen $\log t$ und $\log f$ ist in Abbildung 5.8 gut erkennbar. Für den funktionalen Zusammenhang zwischen Laufzeit und Flächenzahl darf somit die Form $t(f) = cf^d$ angenommen werden. Ermittelt man die Gauß'sche Ausgleichsgerade, so ergibt sich für den

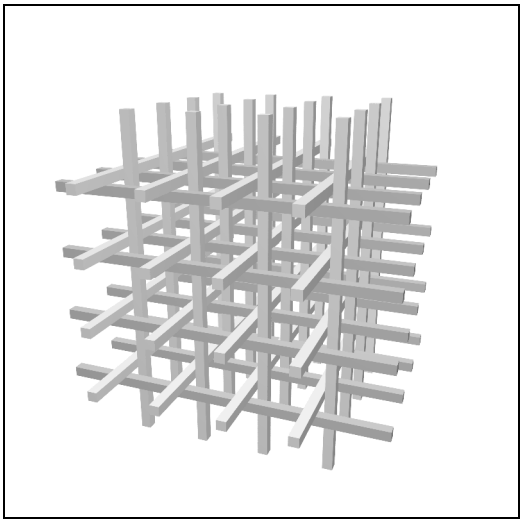


Abbildung 5.7: Eine Anordnung von n Stäben, die $\Theta(n^{1.5})$ Partitionierungsschnitte erfordert

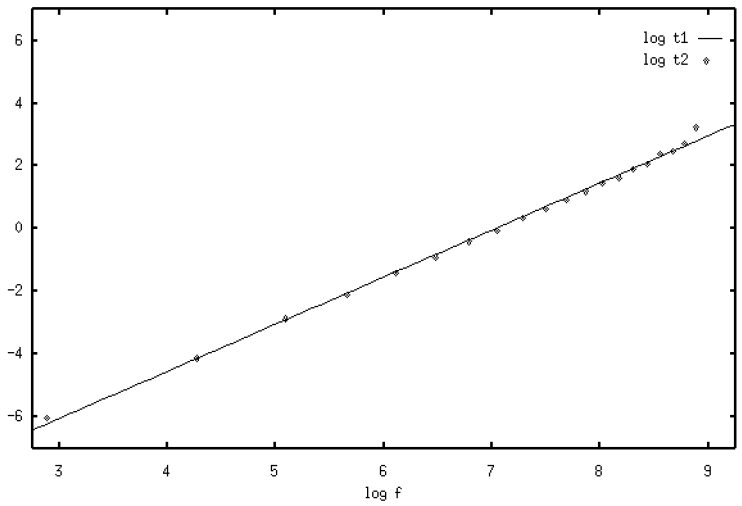


Abbildung 5.8: Experimentelle Laufzeitanalyse für BSP am Beispiel der obigen Stabanordnungen

Exponenten d ein Wert von 1.502, was der unteren Schranke für dieses Beispiel sehr nahe kommt.
Dieses Beispiel repräsentiert in gewissem Sinne den Worst–Case – zumindest ist keine Familie von Anordnungen disjunkter Polyeder bekannt, für die die Größe jedes Partitionierungsbaumes asymptotisch schneller als $n^{1.5}$ wächst. Für allgemeinere Szenen wird der Exponent d in der Regel kleiner sein. Dies bestätigt folgende Beispielsituation, die für die Laufzeitmessungen repräsentativen Charakter trägt.
Eine variierende Zahl von Polyedern aus einem Modellgehirn wurde mit Hilfe des Binary Space Partitioning visualisiert. Die anatomischen Strukturen wurden aus einer Serie 2–dimensionaler Schnitte eines idealisierten Gehirnatlas rekonstruiert. Durch die Modellierung wurden circa 50 Polyeder gewonnen, die die verschiedenen Gehirnbereiche approximieren. Die betrachteten Szenen setzten sich aus bis zu 20000 Dreiecken zusammen (vgl. Abbildung 5.9).

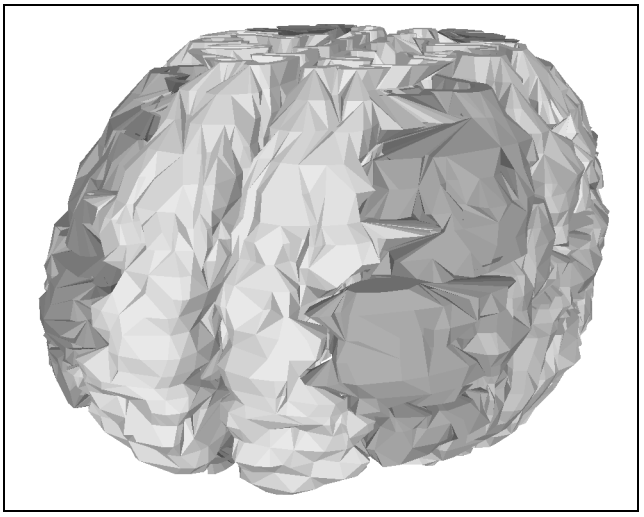


Abbildung 5.9: 3D–Rekonstruktion eines idealisierten Modellgehirns

Bei der experimentellen Laufzeitanalyse ergab sich ein Wert von $d = 1.30$ für den Exponenten.

5.3 Interaktive Bewegungsspezifikation

Der Entwerfer der zu montierenden Objekte hat in der Regel eine klare Vorstellung über den groben Ablauf der Montage. Eine interaktive Spezifikation der verschiedenen Zwischenstadien bei der Montage wird ihm nicht schwer fallen. In den beiden folgenden Abschnitten beschreiben wir daher kurz die Möglichkeiten des Systems, Positionen, Orientierungen und dadurch einfache Bewegungen von Körpern interaktiv zu spezifizieren.

5.3.1 Position und Orientierung

Die Position eines Körpers wird durch die x-, y- und z-Werte des Ursprungs seines körpereigenen Koordinatensystems angezeigt. Seine Orientierung wird in Form von zyx–Euler Winkeln ausgegeben. Diese numerischen Werte und damit auch die Position und die Orientierung des Körpers können auf vielfache Weise direkt manipuliert werden.

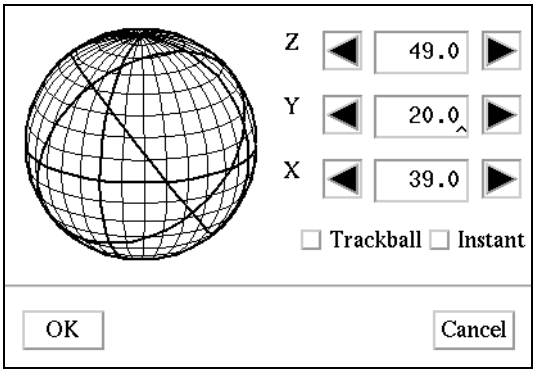


Abbildung 5.10: Orientierungsspezifikation

1. Per Tastatur können die Werte in den dafür vorgesehenen Eingabefeldern editiert werden.

2. Über zusätzliche Knöpfe können die einzelnen Werte bequem schrittweise erhöht oder erniedrigt werden.

3. Im Modus der Dreiseitenansicht stehen für die Positionsangabe drei gekoppelte Cursor in der xy-, yz- und der xz-Ebene zur Verfügung (vgl. Abbildung 5.11).

4. Der Trackball ist sowohl zur Positions– als auch zur Orientierungsspezifikation sehr nützlich. Da Positions– und Orientierungsänderungen oft bezüglich der Weltachsen oder der körpereigenen Achsen angegeben werden, sind diese auf dem Trackball hervorgehoben (vgl. Abbildung 5.10). Einen Einheitsvektor, den man mittels des Trackballs festlegt, kann man einerseits als Richtung für eine Translation des Körpers interpretieren und andererseits als Rotationsachse, um die das Körperkoordinatensystem gedreht werden soll.

5.3.2 Translation und Rotation

Die Mechanismen zur Positions– und Orientierungsspezifikation dienen gleichzeitig zur Festlegung einer Bewegung. Eine einfache Positionsänderung ruft eine Translation hervor. Eine Orientierungsänderung kann man stets als eine Rotation um eine geeignete Achse auffassen. Diese Achse und ein geeigneter Rotationswinkel lassen sich aus Anfangs– und Endorientierung leicht berechnen (siehe A.3). Um Eindeutigkeit zu erreichen, wird die Rotationsachse

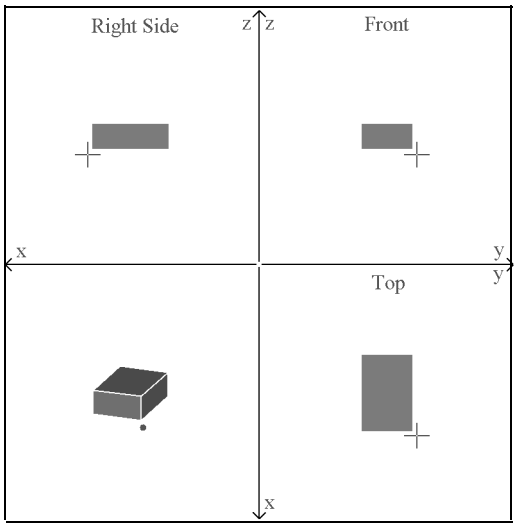


Abbildung 5.11: Gekoppelte Cursor zur Positionsspezifikation

durch den Ursprung des Körperkoordinatensystems gelegt, und vom Rotationswinkel wird angenommen, daß er kleiner als 180° ist.
Neben diesen Möglichkeiten, einfache Translationen und Rotationen von Körpern durchzuführen, benötigt man oft Operationen, die es ermöglichen, Körper relativ zueinander auszurichten. Folgende Operationen sind zum Beispiel sehr nützlich:

1. Bei Polyedern kann man durch zwei Eckpunkte leicht einen Translationsvektor bestimmen.

2. Durch das Kreuzprodukt zweier Vektoren ist in einfacher Weise die Richtung einer Rotationsachse definiert, um die man einen Körper drehen kann, um die eine Richtung in die andere zu überführen. Bei Polyedern kann man die Richtungen aus Kanten oder Flächennormalen erhalten.

5.3.3 Repräsentation von Bewegungsabläufen

Bewegungsabläufe werden vom System nicht als wirkliche Bewegungen abgespeichert sondern als eine Folge von Momentaufnahmen der Positionen und Orientierungen der einzelnen Körper. Jede Aufnahme zeigt eine legale Konfiguration der Körper, d.h. alle Körper sind paarweise disjunkt. Zur Platzersparnis werden bei einem Übergang von einem Zeitpunkt zum nächsten jeweils nur die Positionen und Orientierungen der Körper notiert, deren Lage sich verändert hat.
Da ein Schritt eines Bewegungsablaufs nur aus einer Translation oder einer Rotation eines Körpers oder einer Gruppe von Körpern bestehen darf, läßt sich aus zwei aufeinanderfolgenden Positions– und Orientierungsangaben die durchgeführte Bewegung der betrachteten Körper rekonstruieren. Bei Rotationen wird vorausgesetzt, daß der Rotationswinkel kleiner als 180° ist.

Bei der Durchführung einer Montage kann man sich zunächst Ausgangs– und Endkonfiguration der Montageteile vorgeben und anschließend durch Einfügen von Zwischenstadien den Montageplan so vervollständigen, daß jeder Einzelschritt nur aus einer Translation oder einer Rotation besteht. Wenn man nach Wegen sucht, um von einer Konfiguration kollisionsfrei zur nächsten zu kommen, kann man die implementierten Heuristiken zur Bewegungsplanung benutzen.

5.4 Heuristiken zur Bewegungsplanung

Zur Erleichterung der interaktiven Bewegungsspezifikation verfügt das System über Komponenten zur automatischen Bewegungsplanung, die in einfachen Situationen anwendbar sind.

Bei der automatischen Generierung eines Bewegungsplanes gibt man Start– und Endkonfiguration der beteiligten Körper vor und klassifiziert alle möglichen Zwischenkonfigurationen danach, ob sie kollisionsfrei sind oder nicht. Dadurch wird der Konfigurationsraum, dessen Dimension der Summe der Freiheitsgrade der bewegten Körper entspricht, in Freiräume und Konfigurationsraumhindernisse unterteilt. Innerhalb der Freiräume versucht man dann, Start– und Endkonfiguration durch einen Weg zu verbinden. Auf diese Weise ist gewährleistet, daß der korrespondierende Bewegungsablauf ohne Kollisionen durchführbar ist.
Da ein einzelner starrer Körper bereits 6 Freiheitsgrade besitzt, kann die Dimension des Konfigurationsraumes sehr hoch werden, wenn man die Bewegungsmöglichkeiten der Körper nicht einschränkt. Zudem erschweren die rotatorischen Freiheitsgrade eine einfache Repräsentation der Konfigurationsraumhindernisse, weil diese im allgemeinen von Flächen höherer Ordnung begrenzt werden. Deshalb haben wir uns bei der Implementierung der Algorithmen zur Bewegungsplanung auf Problemstellungen beschränkt, die keine Orientierungsänderungen der bewegten Objekte zulassen.
Konfigurationsraumhindernisse für translatorisch bewegte Polyeder in einer Hinderniswelt aus Polyedern sind selbst als polygonal begrenzte Bereiche beschreibbar. Betrachten wir einen Polyeder P_1, der in Gegenwart eines Hindernispolyeders P_2 verschoben wird. Der zugehörige Konfigurationsraum hat Dimension 3 und enthält ein Konfigurationsraumhindernis C_{12}.

$$C_{12} \quad = \quad \{\mathbf{s} \mid (P_1 + \mathbf{s}) \cap P_2 \neq \emptyset\}$$

C_{12} kann man als Minkowski-Differenz $P_2 \ominus P_1$ berechnen:

$$P_2 \ominus P_1 \quad = \quad \{\mathbf{p_2} - \mathbf{p_1} \mid \mathbf{p_2} \in P_2, \mathbf{p_1} \in P_1\}$$

Die Minkowski-Differenz zweier Polyeder und damit die Konfigurationsraumhindernisse können in Zeit $O(|P_1|^3|P_2|^3 \log(|P_1||P_2|))$ berechnet werden, bzw. für den Spezialfall konvexer Polyeder in Zeit $O(|P_1||P_2|)$ (siehe [La91], Kapitel 3). Bereits für das entsprechende 2–dimensionale Problem muß man mit einer Komplexität von $O(|P_1|^2|P_2|^2)$ zur Repräsentation des kompletten Konfigurationsraumes rechnen.
In den beiden nachfolgenden Abschnitten werden zwei einfache Heuristiken zur Wegesuche beschrieben, die die Berechnung des gesamten Konfigurationsraumes bewußt vermeiden. Sie versuchen vielmehr mit partiellem Wissen über das Aussehen der Konfigurationsraumhindernisse auszukommen: Es wird nur ihr Schnitt mit einer Ebene oder einer Geraden berechnet.

5.4.1 Bewegungsplanung für ein Polyeder mit zwei translatorischen Freiheitsgraden

In diesem Abschnitt wollen wir die Bewegungsmöglichkeiten der Polyeder auf Verschiebungen beschränken, die sich als Linearkombination von zwei ausgezeichneten Vektoren $\mathbf{s_1}$ und $\mathbf{s_2}$ beschreiben lassen. Zur Planung einer Bewegung eines einzelnen Polyeders P_1 verbleiben uns also nur zwei Freiheitsgrade. Als Beispiel betrachten wir Abbildung 5.12. Hier soll ein Piano aus einem Zimmer bewegt werden, indem es auf dem Fußboden verschoben wird und dabei seine Orientierung beibehält.

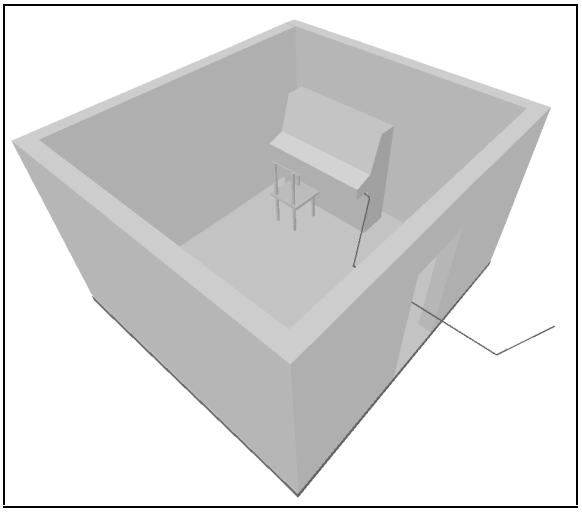


Abbildung 5.12: Bewegungsplanung für ein Piano in einem Zimmer

Die Konfigurationsraumhindernisse des zugehörigen 2–dimensionalen Konfigurationsraumes lassen sich als Polygone (mit Löchern) in der x_1x_2–Ebene darstellen. Das Hindernispolygon $\overline{C}_{12}$, das durch den Polyeder P_2 hervorgerufen wird, läßt sich als Schnitt der Minkowski–Differenz $P_2 \ominus P_1$ mit der Translationsebene $H_{\mathbf{s_1 s_2}}$ berechnen:

$$\begin{aligned} \overline{C}_{12} &= (P_2 \ominus P_1) \cap H_{\mathbf{s_1 s_2}} \quad \text{mit} \\ H_{\mathbf{s_1 s_2}} &= \{x_1\mathbf{s_1} + x_2\mathbf{s_2} \mid x_1, x_2 \in \mathbb{R}\} \end{aligned}$$

Das Konfigurationsraumhindernis $\overline{C}_{12}$ läßt sich aber auch anders berechnen, ohne die 3–dimensionale Konstruktion von $P_2 \ominus P_1$: Wenn eine Verschiebung $\mathbf{s} = x_1\mathbf{s_1} + x_2\mathbf{s_2}$ den Polyeder P_1 mit P_2 in Kontakt bringt, dann gilt:

$$\begin{aligned} & \exists v_1 \in V_1, f_2 \in F_2 : \quad (v_1 + \mathbf{s}) \cap f_2 \neq \emptyset \\ \vee \quad & \exists f_1 \in F_1, v_2 \in V_2 : \quad (f_1 + \mathbf{s}) \cap v_2 \neq \emptyset \\ \vee \quad & \exists e_1 \in E_1, e_2 \in E_2 : \quad (e_1 + \mathbf{s}) \cap e_2 \neq \emptyset \end{aligned}$$

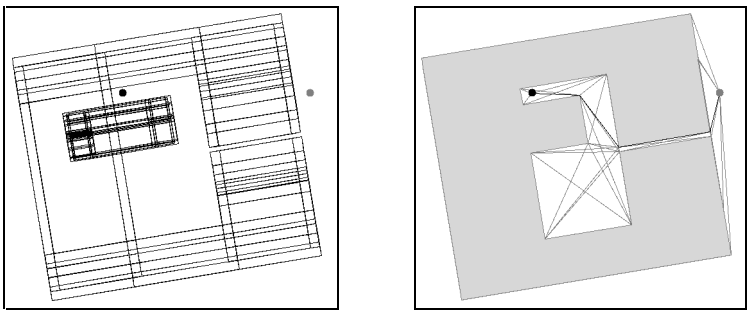


Abbildung 5.13: 2–dimensionaler Konfigurationsraum für die Bewegung eines Pianos aus einem Zimmer mit zugehörigem Sichtbarkeitsgraph

Jede dieser drei Bedingungen induziert im allgemeinen im Konfigurationsraum (x_1x_2–Ebene) genau ein Liniensegment, wenn man von konvexen Begrenzungsflächen ausgeht (siehe A.5). Betrachtet man alle Ecke–Fläche und alle Kante–Kante Paare, so erhält man ein Arrangement aus $O(|P_1|\,|P_2|)$ vielen Liniensegmenten, was uns ein Bild von dem Konfigurationsraumhindernis $\overline{C}_{12}$ vermittelt (siehe Abbildung 5.13, links). Wir gehen davon aus, daß die Startkonfiguration, die dem Ursprung in der x_1x_2–Ebene entspricht, kollisionsfrei ist. Wir können die Zielkonfiguration für P_1 nur dann erreichen, wenn sie in der gleichen Zusammenhangskomponente wie die Startkonfiguration liegt. Deshalb genügt es, die Begrenzung der Zusammenhangskomponente zu berechnen, die den Ursprung enthält. Falls auch die Zielkonfiguration in dieser Zusammenhangskomponente liegt, können wir im Anschluß an die Berechnung des relevanten Teils des Konfigurationsraumes auch einen Weg konstruieren, der Start– und Zielkonfiguration verbindet. Aufgrund der Überschneidungen der Liniensegmente im Konfigurationsraum wäre zu befürchten, daß die Komplexität der Freiraumbegrenzung quadratisch in der Zahl der Liniensegmente wächst. In [GSS89] wird aber folgender Satz gezeigt:

> Die kombinatorische Komplexität einer Zusammenhangskomponente in einem Arrangement von n Liniensegmenten ist $O(n\alpha(n))$. Zu einem vorgegebenen Punkt kann die Zusammenhangskomponente, die diesen Punkt enthält, in Zeit $O(n\alpha(n)\log^2 n)$ berechnet werden. ($\alpha(n)$ bezeichnet die inverse Ackermannfunktion.)

Dieser Satz gewährleistet, daß zur Beschreibung des für uns relevanten Teils des Konfigurationsfreiraumes nur unwesentlich mehr als $O(|P_1|\,|P_2|)$ Platz benötigt wird. Gleichzeitig gibt er uns einen effizienten Algorithmus zur Berechnung dieses Freiraumes an die Hand.
Wenn wir außer dem Polyeder P_2 noch weitere Hindernispolyeder berücksichtigen, erhalten wir in der x_1x_2–Ebene eine Menge von Hindernispolygonen (ohne Löcher) und eventuell eine äußere polygonale Begrenzung. Wir suchen einen Weg vom Ursprung zum Zielpunkt. Schon Lozano-Pérez und Wesley (siehe [LW79]) haben bewiesen, daß der kürzeste Weg auf den Kanten des Sichtbarkeitsgraphen verläuft. Der Sichtbarkeitsgraph (siehe Abbildung 5.13, rechts) für n Liniensegmente läßt sich laut [Ed87] in Zeit $O(n^2)$ berechnen. n entspricht hier der Komplexität der relevanten Zusammenhangskomponente des Konfigurationsraumes. Diese

kann höchstens $O(|P_1|\,|P_2|\,\alpha(|P_1|\,|P_2|))$ betragen, wenn wir mit P die Menge aller Hindernispolyeder bezeichnen. Abschließend muß nur noch ein kürzester Weg auf diesem Graphen bestimmt werden. Die Gesamtlaufzeit ist somit durch $O(|P_1|^2\,|P|^2\alpha^2(|P_1|\,|P|))$ beschränkt. Details zur Implementierung findet man in [ES93].

Bemerkung:
Als Alternative zur Wegesuche bietet sich die Methode von [OY82] an, die das Voronoi–Diagramm der Polygonmenge benutzt, um Start– und Zielpunkt miteinander zu verbinden. Das Voronoi–Diagramm für eine Menge von Polygonen, bestehend aus n Liniensegmenten, kann in der Zeit $O(n \log n)$ berechnet werden (siehe [Fo86]). Die Motivation für die Benutzung von Voronoi–Diagrammen zur Wegesuche liegt darin, möglichst große Sicherheitsabstände zu den Hindernissen zu wahren und weniger darin, einen möglichst kurzen Weg zu bestimmen. Durch die Verwendung von Voronoi–Diagrammen ließe sich die Gesamtlaufzeit des oben skizzierten Algorithmus auf $O(|P_1|\,|P|\log(|P_1|\,|P_2|)\alpha(|P_1|\,|P|))$ senken.

5.4.2 Bewegung eines Polyeders im Raum bei fester Orientierung

Im Gegensatz zum vorherigen Abschnitt lassen wir nun alle drei translatorischen Freiheitsgrade für die Bewegung des Polyeders P_1 zu. Zur Bewegungsplanung benutzen wir allerdings nur eine sehr einfache Heuristik, die auch von [Sch92] verwendet wurde. Diese Heuristik ist zwar nicht in der Lage, immer einen Weg zu konstruieren, wenn ein solcher existiert, oder zu entscheiden, daß es keine Lösung gibt, aber sie liefert für viele praktische Beispiele sehr schnell akzeptable Wege. Anhand von Abbildung 5.14 erläutern wir kurz diese Methode der Wegesuche für zwei Freiheitsgrade.

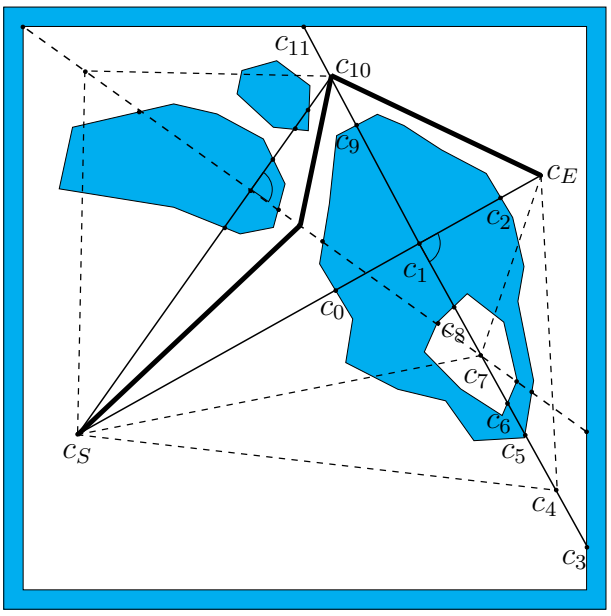


Abbildung 5.14: Eine einfache Strategie zur Navigation im Konfigurationsraum

Gesucht sei ein Weg von der Startkonfiguration c_S zur Endkonfiguration c_E. Die geradlinige Verbindung zwischen c_S und c_E im Konfigurationsraum wäre der einfachste Weg, aber dieser ist nicht zulässig, weil Konfigurationsraumhindernisse gekreuzt werden. Die Strecke $c_S c_E$ zerfällt in erlaubte ($c_S c_0$, $c_2 c_E$) und verbotene Streckenabschnitte ($c_0 c_2$). Um das erste Hindernis auf $c_S c_E$ zu umgehen, sollte man seitlich dazu ausweichen. Zu diesem Zweck errichtet man auf dem ersten verbotenen Streckenabschnitt die Mittelsenkrechte und sucht auf ihr ebenfalls die erlaubten ($c_3 c_5$, $c_6 c_8$, $c_9 c_{11}$) und verbotenen ($c_5 c_6$, $c_8 c_9$) Bereiche. Als mögliche Umwegpunkte c' bestimmt man die Mittelpunkte (c_4, c_7, c_{10}) der erlaubten Bereiche auf der Mittelsenkrechten. Anschließend versucht man in rekursiver Weise, Wege von c_S nach c' und von c' nach c_E zu konstruieren. Wenn man als Zwischenkonfiguration $c' = c_4$ wählt, hat man bereits einen kollisionsfreien Weg gefunden, denn die Strecken $c_S c'$ und $c' c_E$ werden von keinem Hindernis geschnitten. Entscheidet man sich allerdings für $c' = c_7$, so gerät das Suchverfahren in eine ausweglose Situation, weil c' in einer anderen Zusammenhangskomponente liegt als Start– und Endkonfiguration. Fällt die Wahl auf $c' = c_{10}$, so ist man insofern einer Lösung näher gekommen, als daß man "nur noch" einen Weg von c_S zu c' finden muß, denn die Strecke $c' c_E$ ist kollisionsfrei.

Wie im Beispiel hat man auf jeder Rekursionsstufe im allgemeinen mehrere Zwischenkonfigurationen zur Auswahl. Um sich nicht von vorneherein auf eine Wahl zu fixieren, kann man mittels Backtracking alle Kombinationen bis zu einer vorgegebenen Rekursionstiefe durchmustern. Die Reihenfolge, in der Umwegpunkte durchprobiert werden, kann man von ihrem Abstand zur Gerade zwischen momentanem Start– und Endpunkt abhängig machen. Wählt man zuerst die Umwegpunkte, die die kleinste seitliche Abweichung aufweisen, so tendiert das Verfahren zwar zu einer zielgerichteten Wegesuche, aber in vielen Fällen lassen sich die gefundenen Wege noch weiter verkürzen. Ein gefundener Weg besteht aus einer Folge von geradlinig verbundenen Konfigurationen. Eine nachträgliche Optimierung kann versuchen, überflüssige Zwischenkonfigurationen zu entfernen. Dazu muß man – wie im Algorithmus selbst – nur die Kollisionsfreiheit von einfachen Strecken testen.

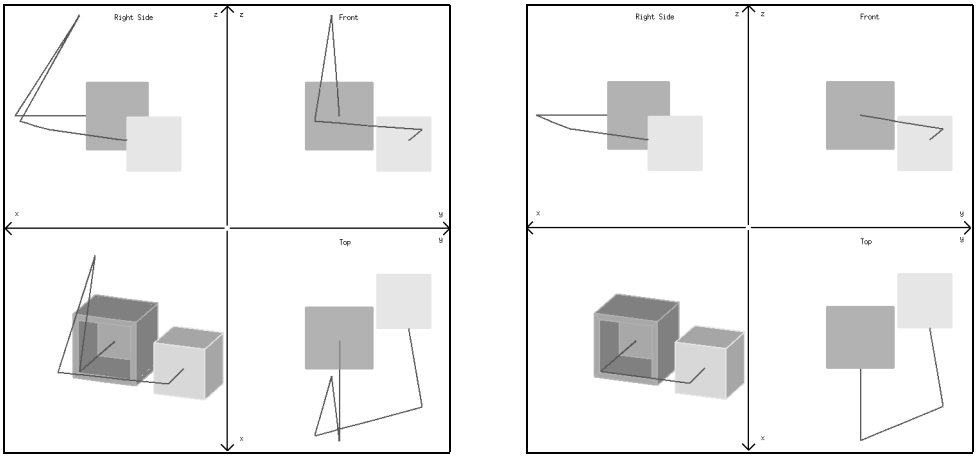


Abbildung 5.15: Heuristische Wegeplanung für Körper mit drei translatorischen Freiheitsgraden und anschließender Optimierung des gefundenen Weges

Abbildung 5.15 zeigt ein einfaches Bewegungsplanungsproblem mit *drei* translatorischen Freiheitsgraden, dessen Lösungsweg mit Hilfe der eben skizzierten Heuristik gefunden und anschließend verkürzt wurde. Zur Übertragung der geschilderten Heuristik in höherdimensionale Konfigurationsräume berechnet man ebenfalls den ersten verbotenen Streckenabschnitt c_0c_2 auf der Verbindungsstrecke zwischen Start– und Endkonfiguration. c_1 sei der Mittelpunkt der Strecke c_0c_2. In Analogie zum 2–dimensionalen Fall sucht man erlaubte Umwegkonfigurationen auf der Hyperebene $H = \{\mathbf{x} \,|\, (\mathbf{c_2} - \mathbf{c_0})^T(\mathbf{x} - \mathbf{c_1}) = 0\}$, die im 2–dimensionalen der Mittelsenkrechten entspricht. Dazu berechnet man aber nicht den kompletten Schnitt des Konfigurationsraumes mit H, sondern nur mit Geraden durch c_1, die in H verlaufen. Die Richtung dieser Geraden bestimmt man zufällig. Ansonsten kann man genau wie im 2–dimensionalen Fall verfahren.

Zur Realisierung dieser heuristischen Wegesuche benötigt man also nur eine Komponente, die den Schnitt einer Geraden im Konfigurationsraum mit den Konfigurationsraumhindernissen berechnen kann. Für den Fall eines translatorisch bewegten Polyeders in einer Hinderniswelt aus Polyedern gestaltet sich die Berechnung der erlaubten und verbotenen Abschnitte auf einer Geraden im Konfigurationsraum besonders einfach, denn wir können sie auf die elementaren Kollisionsberechnungen aus Abschnitt 2.1 zurückführen:
Betrachten wir zwei disjunkte Polyeder P_1 und P_2 und eine beliebige, aber feste Translationsrichtung $\mathbf{s}$. Gesucht ist

$$\mathcal{T} \;=\; \{t \in [t_1, t_2] \,|\, (P_1 + t\,\mathbf{s}) \cap P_2 \neq \emptyset\}.$$

Zur Berechnung von $\mathcal{T}$ gehen wir in drei Schritten vor:

1. Berechne $\mathcal{T}_1 = \{t \in [t_1, t_2] \,|\, (\partial P_1 + t\,\mathbf{s}) \cap \partial P_2 \neq \emptyset\}$.
2. Für einen beliebigen Punkt $\mathbf{p_1} \in P_1$ berechne $\mathcal{T}_2 = \{t \in [t_1, t_2] \,|\, \mathbf{p_1} + t\,\mathbf{s} \in P_2\}$.
3. Vereinige die Kollisionsintervalle $\mathcal{T}_1$ und $\mathcal{T}_2$ zu $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$.

Im ersten Schritt berechnen wir alle Zeitpunkte $\mathcal{T}_1$, zu denen ein Schnitt der Begrenzungsflächen vorliegt. Damit erfassen wir aber noch nicht die Zeitpunkte $\mathcal{T}'$, zu denen der bewegte Polyeder P_1 vollständig in dem Hindernis P_2 enthalten ist. Im zweiten Schritt bestimmen wir eine Obermenge für $\mathcal{T}' \subseteq \mathcal{T}_2 \subseteq \mathcal{T}$, indem wir die Menge aller Zeitpunkte $\mathcal{T}_2$ berechnen, zu denen ein Referenzpunkt von P_1 sich im Inneren von P_2 befindet. Anschließend brauchen wir nur noch die Vereinigung von $\mathcal{T}_1$ und $\mathcal{T}_2$ zu bestimmen.
Zu 1:

$$\partial P_1 \cap \partial P_2 \neq \emptyset \iff \exists f_1 \in F_1, f_2 \in F_2 : f_1 \cap f_2 \neq \emptyset$$

Wenn f_1 und f_2 konvexe Flächen sind, ist $\{t \,|\, (f_1 + t\,\mathbf{s}) \cap f_2 \neq \emptyset\}$ ein Intervall: $[t_{min}(f_1, f_2), t_{max}(f_1, f_2)]$

$$\begin{aligned} t_{min}(f_1, f_2) &= \min\{\min_{v \in V_{f_1}} t_{col}(v, f_2), \min_{v \in V_{f_2}} t_{col}(f_1, v), \min_{\substack{e_1 \in E_{f_1} \\ e_2 \in E_{f_2}}} t_{col}(e_1, e_2)\} \\ t_{max}(f_1, f_2) &= \max\{\max_{v \in V_{f_1}} t_{col}(v, f_2), \max_{v \in V_{f_2}} t_{col}(f_1, v), \max_{\substack{e_1 \in E_{f_1} \\ e_2 \in E_{f_2}}} t_{col}(e_1, e_2)\} \end{aligned}$$

$$\mathcal{T}_1 \;=\; \bigcup_{\substack{f_1\in F_1\\ f_2\in F_2}} [t_{min}(f_1,f_2), t_{max}(f_1,f_2)]$$

Zu 2:
Den Schnitt der Gerade $\mathbf{p_1} + t\,\mathbf{s}$ mit dem Polyeder P_2 erhält man am einfachsten durch Sortieren alle Kollisionszeitpunkte $\{t_{col}(\mathbf{p_1}, f_2) \mid f_2 \in F_2\}$.

$$\mathcal{T}_2 \;=\; \bigcup_{i=0}^{k} [t_{2i}, t_{2i+1}]$$

Zu 3:
Bei der Berechnung von $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ entstehen in Schritt 1 höchstens $|F_1|\,|F_2|$ viele Intervalle und in Schritt 2 höchstens $|F_2|$. Die Darstellung ihrer Vereinigung in Form disjunkter Intervalle kann in der Zeit $O(|F_1|\,|F_2|\log(|F_1|\,|F_2|))$ mittels Sortieren berechnet werden. ■

Eine Implementierung des Algorithmus ([Ch93]) hat gezeigt, daß sich die Techniken zur effizienten Kollisionserkennung hier sehr gut einsetzen lassen, so daß der Algorithmus zur Wegeplanung davon in erheblichem Maße profitiert.

5.5 Robotersimulation

Bei Montagevorgängen spielt nicht nur die Geometrie der Montageteile eine Rolle sondern auch die der Montagewerkzeuge. Zu den komplexeren Werkzeugen sind alle Arten von Robotern zu zählen. Montagepläne in Form von Roboterprogrammen bedürfen einer eingehenden Kontrolle, denn auch nur die kleinste ungewollte Kollision eines Roboters mit seiner Umgebung kann große Schäden verursachen. Ein zentraler Punkt bei der Simulation von Roboterbewegungen wird daher stets die Verifikation der Kollisionsfreiheit des Bewegungsablaufes sein.

5.5.1 Kinematik

Im Abschnitt 4.3 haben wir uns bereits mit der Kinematik von Robotern beschäftigt, soweit dies für das Verständnis der Algorithmen zur Kollisionserkennung notwendig war. Möchte man auch dynamische Aspekte berücksichtigen, so müssen auch Massen und Trägheitsmomente der Glieder bekannt sein, um die notwendigen Kräfte und Drehmomente zu berechnen. Wir konzentrieren uns hier aber auf das kinematische Verhalten des Roboters. Für die Kinematik spielen die Gelenktypen und die Lage der Gelenkachsen, die sich allein mit Hilfe der Denavit–Hartenberg–Parameter charakterisieren lassen, die entscheidende Rolle. Hinsichtlich der Wechselwirkung des Roboters mit seiner Umgebung interessieren wir uns hauptsächlich für die Erkennung von Kollisionen der Glieder untereinander bzw. mit ihrer Umgebung. Dazu ist eine genaue geometrische Modellierung der Form der Glieder des Roboters erforderlich. Deshalb benutzen wir zur Spezifikation eines Roboters, der aus einer offenen kinematischen Kette besteht, neben einer Tabelle der Denavit–Hartenberg–Parameter auch eine möglichst genaue geometrische Repräsentation der einzelnen Glieder. Bei Kenntnis des jeweiligen Gelenktyps (translatorisch oder rotatorisch) und der aktuellen Belegung der freien Parameter

κ_i ($= d_i$ oder $= \varphi_i$) lassen sich durch Aufmultiplizieren der Transformationsmatrizen $\mathbf{T_i}(\kappa_i)$ Position und Orientierung jedes Gliedes und damit auch des Endeffektors berechnen (siehe Gleichung (4.6)). Diesen Prozeß nennt man Vorwärtstransformation:

$$\begin{aligned}(\kappa_1, \ldots, \kappa_n) &\mapsto (\mathbf{o}, \mathbf{R}) \\ n &= \text{Zahl der Gelenke} \\ \mathbf{o} &= \text{Position des Endeffektors} \\ \mathbf{R} &= \text{Orientierung des Endeffektors}\end{aligned}$$

Um den Endeffektor in eine bestimmte Position $\mathbf{o}$ und Orientierung $\mathbf{R}$ zu bringen, kann es durchaus mehrere n–Tupel $(\kappa_1, \ldots, \kappa_n)$ als Urbild geben, sofern überhaupt eine Lösung existiert. In Abbildung 5.16 sehen wir einen Roboter mit drei rotatorischen und einem translatorischen Gelenk in zwei unterschiedlichen Konfigurationen aber gleicher Position und Orientierung des Endeffektors.

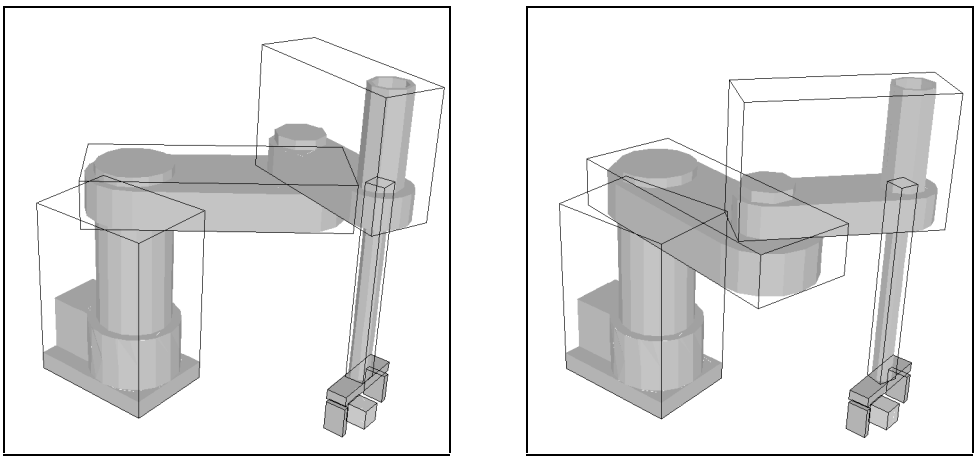


Abbildung 5.16: 4–Gelenk–Roboter in unterschiedlichen Konfigurationen mit gleicher Position und Orientierung des Endeffektors

Im Gegensatz zur Vorwärtstransformation ist die Rückwärtstransformation nicht eindeutig bestimmt (siehe [Cr89], Kapitel 4!). Wenn sich, wie in Abbildung 4.4 (oder auch Abbildung 5.16), drei aufeinanderfolgende Gelenkachsen ($\mathbf{e_3}^{(3)}$, $\mathbf{e_3}^{(4)}$, $\mathbf{e_3}^{(5)}$) schneiden, gibt es für $n \leq 6$ eine geschlossene Formel zur Berechnung aller Konfigurationen $(\kappa_1, \ldots, \kappa_n)$, die zu der gewünschten Position $\mathbf{o}$ und der Orientierung $\mathbf{R}$ führt (siehe [PR69]). Wir beschränken uns auf diese Form von Robotern, weil unter allgemeineren Voraussetzungen nur iterative, numerische Lösungen für die inverse Transformation existieren. In der vorliegenden Implementierung benutzt [Si93] ausschließlich algebraische Methoden zur Herleitung der inversen Tranformationen, die dem Simulationssystem in Form von Prozeduren zur Verfügung gestellt werden müssen.

5.5.2 Trajektorien

Um einen n–Gelenkroboter von einer Startkonfiguration $\kappa(t_0)$ in eine Zielkonfiguration $\kappa(t_1)$ zu bewegen, hat man viele Freiheiten bei der Wahl von $\kappa(t) = (\kappa_1(t), \ldots, \kappa_n(t))$. $\kappa_i(t)$ beschreibt die zeitliche Entwicklung des i–ten Gelenkparameters. Die einfachste Methode zur Generierung einer Bewegung der Glieder von der Start– in die Zielkonfiguration besteht darin, die Gelenkparameter $\kappa_i(t)$ mittels linearer Interpolation zu bestimmen. Man kann die einzelnen Gelenkparameter κ_i für $i = 1, \ldots, n$ hintereinander oder auch gleichzeitig verändern. Man kann dazu für alle κ_i's eine einheitliche Geschwindigkeit wählen oder die Geschwindigkeiten so koordinieren, daß alle Gelenke zum Zeitpunkt t_1 gleichzeitig ihr Ziel erreichen. Aus Gründen der Einfachheit beschränken wir uns bei der Punkt–zu–Punkt Bewegung auf die genannten Formen von Geschwindigkeitsprofilen, obwohl dies zu unstetigen Geschwindigkeits– und Beschleunigungsverläufen für $\kappa_i(t)$ führt, die in der Praxis (siehe [Cr89], Kapitel 7!) unerwünscht sind.

Die oben diskutierten Bewegungsschemata haben gemeinsam, daß sie im Konfigurationsraum der Gelenkparameter einen stückweise linearen Weg beschreiben. Die zugehörige Bahnkurve des Endeffektors im kartesischen Raum ist im allgemeinen sehr viel komplizierter, so daß man zum Beispiel im Rahmen der Kollisionsberechnungen auf Approximationen zurückgreifen muß. Die stückweise Linearität der Bahnkurve im Konfigurationsraum garantiert aber (siehe Abschnitt 4.3.2) eine einfache Abschätzung der Güte der Approximation. Oft möchte man den Endeffektor aber auf einer geradlinigen Bahn führen und seine Orientierung währenddessen konstant halten. Eine solche kartesische Bewegung läßt sich dadurch approximieren, daß man in genügend dichten Abständen Zwischenpunkte auf der geradlinigen Verbindung zwischen Start– und Zielpunkt generiert und diese Zwischenkonfigurationen in dem Punkt–zu–Punkt Bewegungsmodus ansteuert. Dazu muß man für jeden Zwischenpunkt die inverse Transformation berechnen, um im Raum der Gelenkparameter eine der beschriebenen Interpolationsmethoden benutzen zu können. Probleme ergeben sich dadurch, daß eine stetige Bewegung im kartesischen Raum nicht notwendigerweise ein stetiges Urbild im Raum der Gelenkparameter besitzt. Die Rückwärtstransformation liefert uns für jede Zwischenkonfiguration eventuell keine oder mehrere Lösungen, die man soweit möglich zu einem Weg kombinieren muß, der die gewünschte kartesische Bewegung approximiert (siehe [Si93]). In Abbildung 5.17 und 5.18 sind die Trajektorien für zwei interaktiv erstellte Bewegungsabläufe skizziert. Sie setzen sich aus den verschiedensten Bewegungstypen zusammen. Sie enthalten Punkt–zu–Punkt Bewegungen, bei denen die Gelenkparameter aufeinanderfolgend oder gleichzeitig variiert werden, und geradlinige Bewegungen des Endeffektors, bei denen seine Orientierung unverändert bleibt.

5.5.3 Roboterprogrammierung

Zur komfortablen Erstellung von Roboterprogrammen besteht sowohl die Möglichkeit zur textuellen als auch zur graphischen Programmierung. Um bereits bei der Spezifikation einer Aktion Kollisionskontrollen durchführen zu können, bietet sich eine interaktive Vorgehensweise an. Die nachfolgenden Abbildungen 5.18-5.20 zeigen die verschiedenen Bestandteile der graphischen Oberfläche zur Programmierung von Bewegungsabläufen bei Robotern.

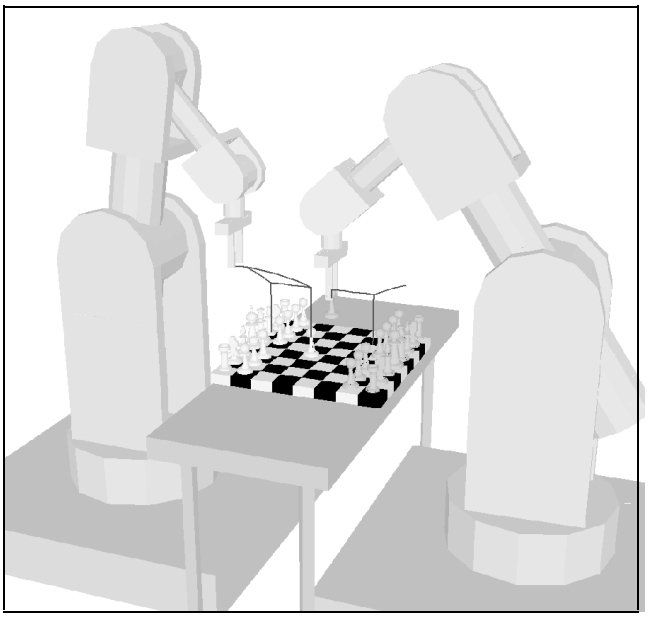


Abbildung 5.17: Trajektorien der Greifpunkte zweier Roboter während eines Bewegungsablaufs

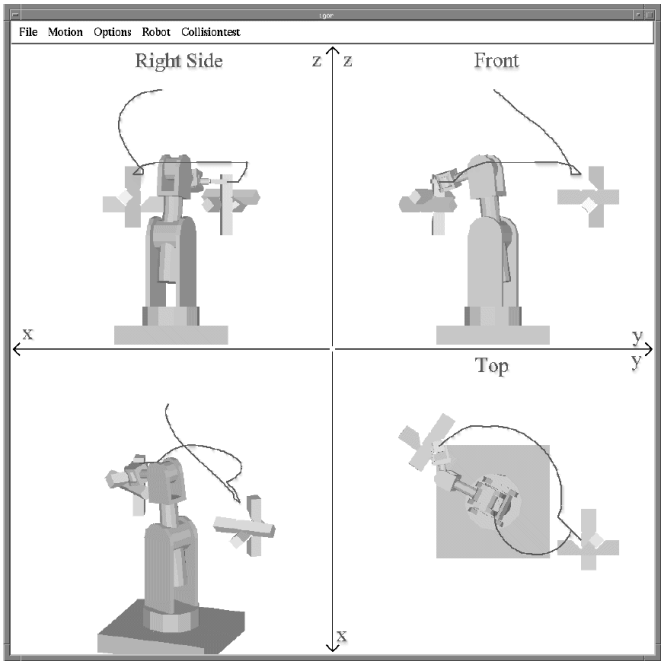


Abbildung 5.18: Oberfläche zur interaktiven graphischen Roboterprogrammierung

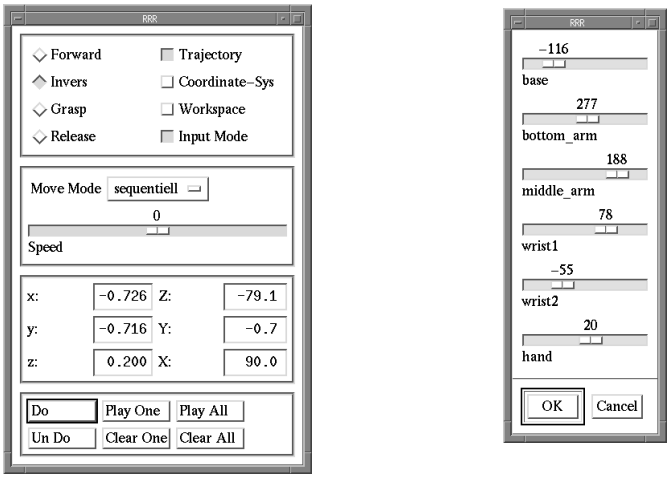


Abbildung 5.19: Spezifikation der Gelenkparameter

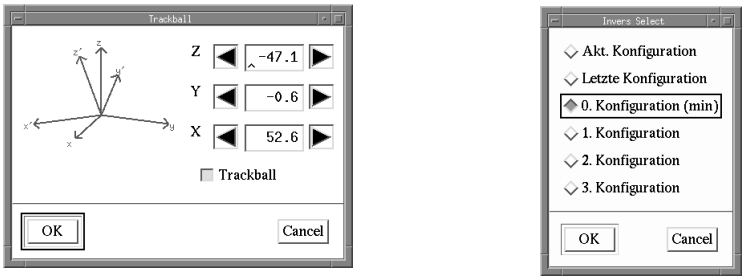


Abbildung 5.20: Spezifikation der Orientierung des Endeffektors für eine Rückwärtstransformation und Auswahl einer errechneten Gelenkwinkel–Konfiguration

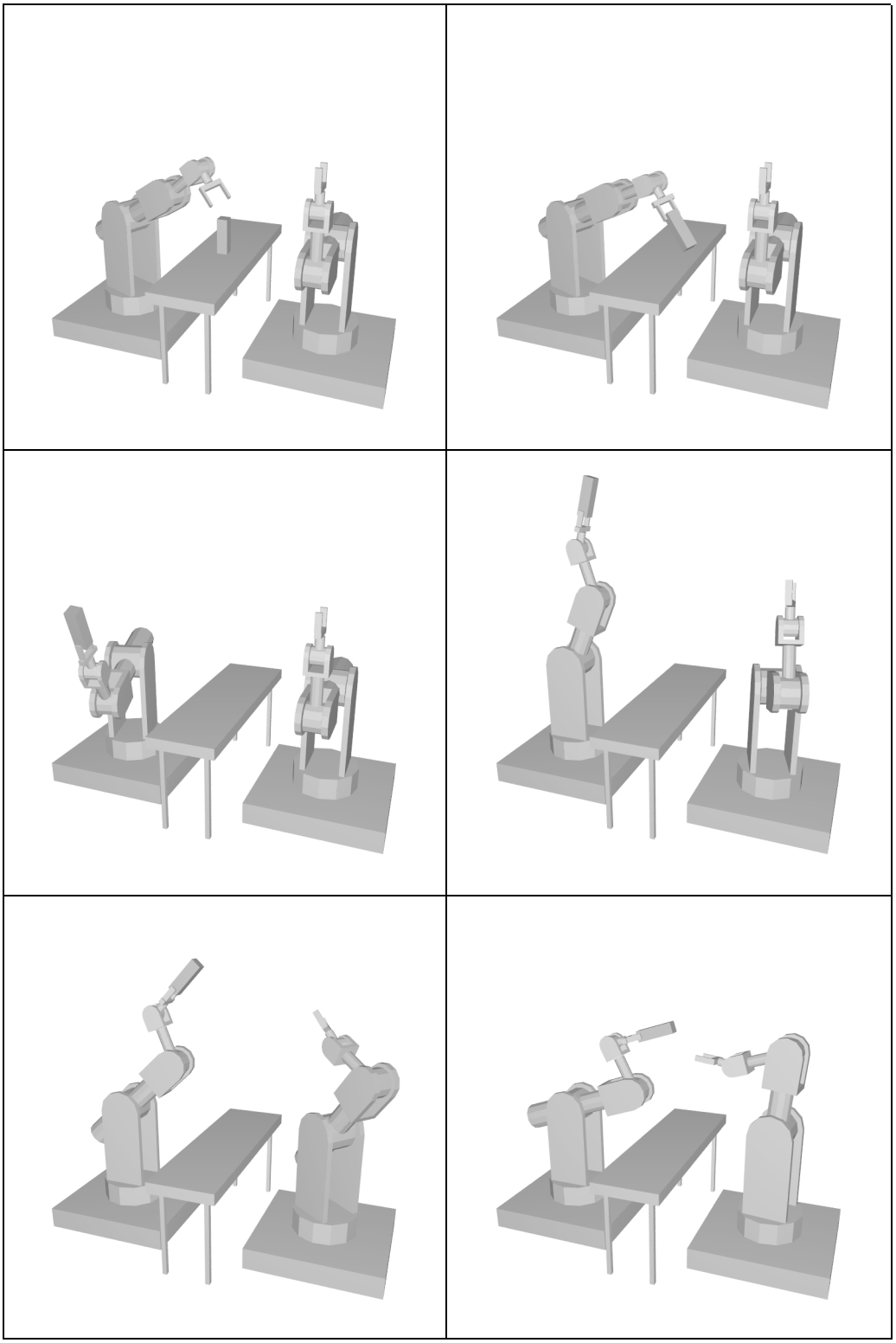
Zur eigentlichen Steuerung aller Aktionen eines oder mehrerer Roboter dient das Kontrollmenü in Abbildung 5.19 (links). Hier kann der Typ einer Aktion (Vorwärts– oder Rückwärtstransformation, Greifen und Loslassen von Objekten) bestimmt werden. Außerdem bestehen die Optionen, den (approximierten) Arbeitsraum des Roboters, die Koordinatensysteme der Glieder und die zurückgelegten Bahnkurven des Endeffektors zu visualisieren. Um die Form von Trajektorien zu beeinflussen, kann man bei der Punkt–zu–Punkt Bewegung die verschiedenen Interpolationsmethoden auswählen oder sich für eine geradlinige kartesische Bewegung entscheiden. Desweiteren läßt sich die Geschwindigkeit einer Bewegung über einen Schieberegler steuern.
Die Gelenkparameter für eine Vorwärtstransformation lassen sich unter Berücksichtigung der für die einzelnen Gelenke gültigen Unter– und Obergrenzen ebenfalls über Schieberegler einstellen (vgl. Abbildung 5.19, rechts). Zur Eingabe von Position und Orientierung des Endeffektors können die gekoppelten Cursor im Hauptfenster (siehe Abbildung 5.18) und der Trackball benutzt werden, so daß jederzeit eine numerische und graphische Spezifikation aller Werte möglich ist. Außerdem ist eine Darstellung des Endeffektorkoordinatensystems in Bezug zum Weltkoordinatensystem (siehe Abbildung 5.20, links) vorgesehen. Durch Auswahl einer Achse dieser beiden Koordinatensysteme läßt sich der Endeffektor um diese Achse drehen. Diese Möglichkeit bietet eine sehr intuitive Orientierungsspezifikation. Nach der Vorgabe seiner Position und Orientierung können alle Gelenkparameter–Konfigurationen durch Rückwärtstransformation bestimmt werden. Da dabei Mehrfachlösungen auftauchen können, hat man nach einer Rückwärtstransformation die Möglichkeit, zwischen bestimmten Konfigurationen zu wählen. Die Zielkonfiguration, die von der aktuellen Konfiguration mit geringstem Aufwand erreichbar ist, wird besonders markiert (vgl. Abbildung 5.20, rechts).

In Szenen mit mehreren kooperierenden Robotern können auch gleichzeitig ablaufende Aktionen definiert werden, die von unterschiedlichen Robotern ausgeführt werden sollen (siehe Abbildung 5.21). Diese Möglichkeit der Spezifikation paralleler Roboterprogramme ist für praxisorientierte Anwendungen unerläßlich, erschwert aber die Kollisionserkennung beträchtlich.

5.5.4 Kollisionserkennung

Bewegt sich nur ein einzelner Roboter und gibt der Bewegungsmodus an, daß die Gelenke nacheinander bewegt werden sollen, so können wir die Kollisionsberechnungen für einfache Translationen und Rotationen einsetzen. Sobald mehrere Gelenke oder mehrere Roboter gleichzeitig bewegt werden, beschreiben die Glieder komplexere Bewegungen, die sich aus einer Überlagerung von einzelnen Translationen und Rotationen ergeben. In diesem Fall benutzen wir die Algorithmen aus Abschnitt 4.2 zur Kollisionserkennung. Diese können den exakten Zeitpunkt einer Kollision zwar nur approximieren, aber sie sind in der Lage, die Kollisionsfreiheit eines Bewegungsablaufs schnell zu verifizieren, wenn die bewegten Objekte sich nicht zu nahe kommen. Dies ist ohnehin nur in den seltensten Fällen erwünscht. Zum Beispiel beim Greifen, Ablegen und Einpassen von Objekten. In diesen besonderen Situationen werden die Objekte, zwischen denen kritische Abstände herrschen, meistens in sehr einfacher Weise bewegt, so daß ihre Bewegung entweder als eine reine Translation oder Rotation beschreibbar ist.

Damit die Detailfülle komplizierter Formen für Roboterglieder und Hindernisse die Laufzeit der Kollisionsalgorithmen nicht unnötig beeinträchtigt, arbeiten alle Kollisionsalgorithmen nach einem hierarchischen Prinzip. Alle Polyeder, ob sie die Glieder eines Roboters modellieren oder sonstige Körper, werden in möglichst kleine Kugeln und Quader eingeschlossen (siehe Abschnitt 4.1 und Abbildung 5.16). Die Kollisionserkennung basiert auf Abstandsbestimmungen oder auf der expliziten Berechnung von Kollisionszeitpunkten. In beiden Fällen führen wir die entsprechenden Berechnungen immer zuerst für die Hüllkugeln durch. Nur wenn dadurch die Kollision zweier Körper nicht ausgeschlossen werden kann, wiederholen wir die Berechnungen für die Hüllquader. Wenn selbst das nicht das gewünschte Ergebnis liefert, rechnen wir mit dem genauen Polyedermodell.



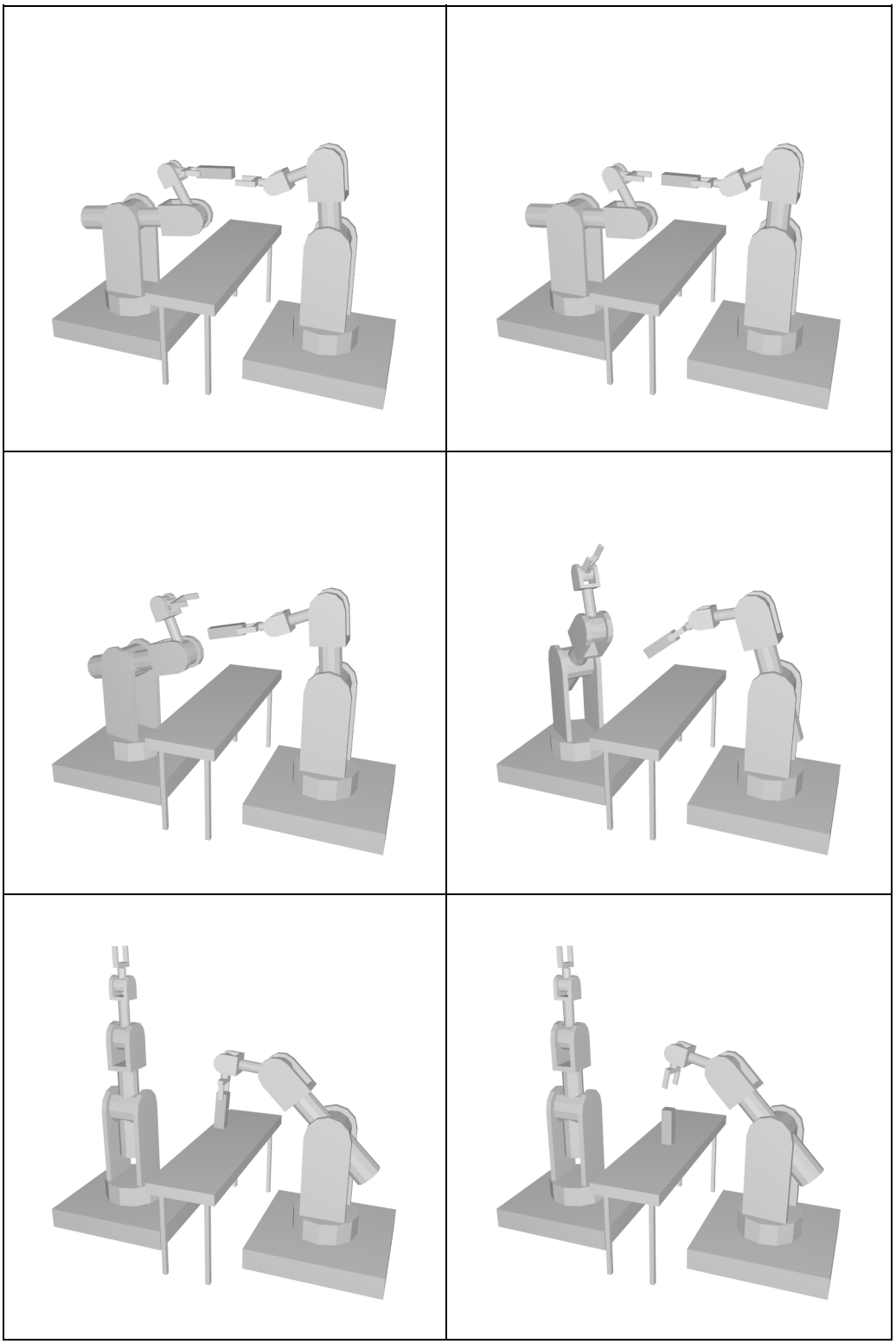


Abbildung 5.21: Zwei kooperierende Roboter bei der parallelen Ausführung eines interaktiv erstellten Roboterprogramms

Ausblick

Bei allen Berechnungen in dieser Arbeit wurde implizit angenommen, daß eine exakte reelle Arithmetik zur Verfügung steht. Bei der Implementierung mußte jedoch der Umstand berücksichtigt werden, daß durch die Verwendung der Fließkomma–Arithmetik Rundungsfehler zu Inkonsistenzen in den Berechnungen führen können. Eine kritische Situation für die Algorithmen zur Kollisionserkennung liegt zum Beispiel dann vor, wenn Objekte sich quasi unter Berührung aneinander vorbeibewegen. Die ausführliche Behandlung solcher entarteter Fälle ist für die *Robustheit* der Algorithmen ausschlaggebend. Die jetzige Implementierung basiert auf dem Konzept der "Epsilon–Umgebungen", d.h. zwei Punkte werden als identisch angesehen, wenn ihr Abstand kleiner als eine festgelegte Toleranz ist.
Wenn wir Objekte durch Polyeder und Bewegungen durch Translationen und Rotationen modellieren, könnte man dem Problem der Rundungsfehler vollständig aus dem Weg gehen, indem man alle Operationen symbolisch mit Hilfe eines Algebrasystems durchführt. Dazu würde man annehmen, daß sich die Koordinaten der Polyedereckpunkte als rationale (algebraische) Zahlen beschreiben lassen. Auf dieser Basis ließen sich alle elementaren Kollisionstests exakt realisieren. Der entscheidende Nachteil wäre jedoch ein starker Effizienzverlust. Statt dessen könnte man zu vorgegebener Präzision der Eingabedaten jede einzelne Operation in Hinblick auf die für exakte Berechnungen erforderliche Rechengenauigkeit analysieren und einfach mit entsprechend erhöhter Stellenzahl rechnen. Dabei könnte man zur Beschleunigung die Tatsache ausnutzen, daß für viele Eingabewerte die meisten zu berechnenden Prädikate auch mit stark verminderter Genauigkeit entscheidbar wären.

In Abschnitt 3.3 wird ein subquadratischer Algorithmus zur Berechnung des Kollisionszeitpunktes zwischen einem translatorisch bewegten und einem stationären Polyeder vorgestellt. Dieses Resulat liefert jedoch für die Praxis keinen brauchbaren Algorithmus zur Lösung der Problemstellung. Deshalb sollte das nächste Ziel die Entwicklung eines praktikableren Algorithmus sein. Von besonderem theoretischen Interesse wäre die Klärung der Frage, ob ein subquadratischer Kollisionstest auch dann existiert, wenn das bewegte Polyeder anstelle einer Translation eine Rotation um eine feste Achse ausführt. Ein solcher Algorithmus ist zur Zeit selbst für konvexe Polyeder nicht bekannt.

In Abschnitt 5.4.2 wird eine Heuristik zur Bewegungsplanung vorgeschlagen, die auf den Algorithmen zur effizienten Kollisionserkennung basiert. Die vorliegende Implementierung erlaubt aber nur die Planung einer Bewegung mit konstanter Orientierung. Die Einbeziehung rotatorischer Freiheitsgrade stellt für diese Vorgehensweise kein prinzipielles Problem dar, zumal die entsprechenden Algorithmen zur Kollisionserkennung vorhanden sind.

Unser System zur interaktiven Montageplanung und –simulation trägt dazu bei, bereits

während der ersten Schritte der Produktionsplanung die grundsätzliche Durchführbarkeit einer Montage zu prüfen und den dafür benötigten Aufwand zu beurteilen. Der Entwicklungsingenieur kann in Interaktion mit dem System einen schematischer Ablauf für die Montage erstellen und anschließend den Einsatz geeigneter Montagevorrichtungen und Roboter planen. Dazu muß er Plazierungsfragen klären, Erreichbarkeitsstudien durchführen und Kollisionsgefahren erkennen. Um diesen Anforderungen gerecht zu werden und gleichzeitig eine effiziente interaktive Arbeitsweise mit dem System zu gewährleisten, müssen Objekte und Bewegungen in idealisierter Weise modelliert werden. Wir betrachten Objekte als starre, masselose, polyederförmige Körper, die als Roboter auch gelenkig verbunden sein können. Die geometrische Form der Objekte und ihr kinematisches Verhalten genügen zwar zur Klärung der grundsätzlichen Fragen bei der Montageplanung, aber darüber hinaus können auch dynamische Aspekte von Bedeutung sein, die gegenwärtig nicht von unserem Modell erfaßt werden.

Wünschenswert wäre eine Montagesimulation unter Berücksichtigung aller wirkenden Kräfte und Drehmomente. Dazu müssen als weitere Modellparameter zumindest die Massen und Trägheitsmomente der verschiedenen Körper bekannt sein. Eine Simulation des dynamischen Verhaltens eines mechanischen Systems erfordert die Lösung von Differentialgleichungssystemen, um die Bewegung der einzelnen Körper vorhersagen zu können. Hier können effiziente Kollisionstests nützliche Dienste leisten, denn jede auftretende Kollision zwischen zwei Körpern verändert das Systemverhalten nachhaltig.

Ein weiterer Schritt in Richtung einer Verallgemeinerung des Modells bestände darin, nicht nur starre Körper zu betrachten, sondern auch Deformationen von Körpern zu erlauben. Dadurch ließe sich ein noch höheres Maß an Realitätstreue erreichen. Man muß sich aber stets des Trade-Off zwischen der Güte des Modells und der Effizienz der Simulation bewußt sein.

Anhang A

Mathematische Grundlagen

A.1 Koordinatentransformationen

Gegeben seien ein Weltkoordinatensystem $E = (\mathbf{e_0}, \mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3})$ und ein weiteres Koordinatensystem $E' = (\mathbf{e_0}', \mathbf{e_1}', \mathbf{e_2}', \mathbf{e_3}')$, das bezüglich E beschrieben ist. $\mathbf{e_0}'$ bezeichne den Ursprung von E', $\mathbf{e_1}', \mathbf{e_2}'$ und $\mathbf{e_3}'$ die Basisvektoren. Die Weltkoordinaten des Punktes $\mathbf{x}$ hängen mit den Koordinaten $\mathbf{x}'$ im System E' wie folgt zusammen:

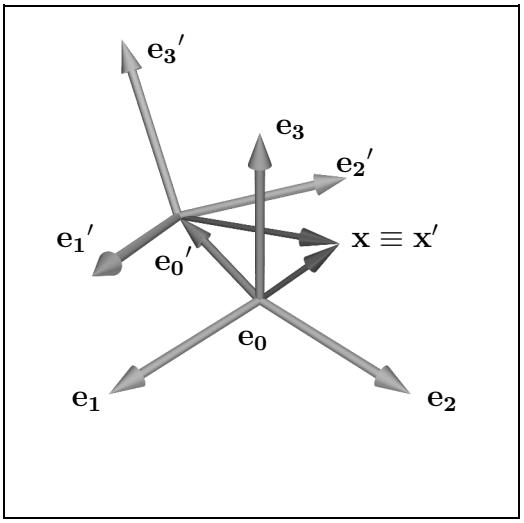


Abbildung A.1: Koordinatentransformation

$$\begin{aligned} \mathbf{x} &= [\mathbf{e_1}', \mathbf{e_2}', \mathbf{e_3}'] \cdot \mathbf{x}' + \mathbf{e_0}' \\ \mathbf{x}' &= [\mathbf{e_1}', \mathbf{e_2}', \mathbf{e_3}']^{-1} \cdot (\mathbf{x} - \mathbf{e_0}') \end{aligned} \tag{A.1}$$

A.2 Rechenregeln für Vektor–Matrixoperationen

Seien $\alpha, \beta, \gamma \in \mathbb{R}$ und $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in \mathbb{R}^3$ und $\mathbf{Q} \in \mathbb{R}^{3\times 3}$.
Die Kramer'sche Regel zur Lösung von 3×3 Gleichungssystemen lautet:

$$\begin{aligned} \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} &= \mathbf{d} \\ \alpha = \frac{\det[\mathbf{d},\mathbf{b},\mathbf{c}]}{\det[\mathbf{a},\mathbf{b},\mathbf{c}]}, \quad \beta &= \frac{\det[\mathbf{a},\mathbf{d},\mathbf{c}]}{\det[\mathbf{a},\mathbf{b},\mathbf{c}]}, \quad \gamma = \frac{\det[\mathbf{a},\mathbf{b},\mathbf{d}]}{\det[\mathbf{a},\mathbf{b},\mathbf{c}]} \end{aligned} \tag{A.2}$$

Doppelte Vektorprodukte lassen sich wie folgt entwickeln:

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

Mit der Lagrangeschen Identität bezeichnet man die Gleichung

$$(\mathbf{a} \times \mathbf{b})^T(\mathbf{c} \times \mathbf{d}) = (\mathbf{a}^T\mathbf{c})(\mathbf{b}^T\mathbf{d}) - (\mathbf{a}^T\mathbf{d})(\mathbf{b}^T\mathbf{c}). \tag{A.4}$$

Zwei weitere nützliche Regeln sind

$$(\mathbf{Q}\mathbf{a}) \times (\mathbf{Q}\mathbf{b}) = adj(\mathbf{Q}^T)(\mathbf{a} \times \mathbf{b}), \tag{A.5}$$

$$\mathbf{Q}(\mathbf{a} \times (\mathbf{Q}^T\mathbf{b})) = (adj(\mathbf{Q}^T)\mathbf{a}) \times \mathbf{b}. \tag{A.6}$$

A.3 Beschreibung von Rotationen

Bezeichnet $\mathbf{r}$ mit $|\mathbf{r}| = 1$ eine Rotationsachse durch den Ursprung und t den Rotationswinkel, gemessen im Gegenuhrzeigersinn, dann erhält man den Punkt $\mathbf{x}$, der durch Rotation des Punktes $\mathbf{p}$ hervorgeht, wie folgt:

$$\begin{aligned} \mathbf{x} &= (\mathbf{r}^T\mathbf{p})\mathbf{r} + \cos t\, \mathbf{r} \times (\mathbf{p} \times \mathbf{r}) + \sin t\, \mathbf{r} \times \mathbf{p} \\ &= (1 - \cos t)(\mathbf{r}^T\mathbf{p})\, \mathbf{r} + \cos t\, \mathbf{p} + \sin t\, \mathbf{r} \times \mathbf{p} \end{aligned} \tag{A.7}$$

In Matrixschreibweise ergibt sich: $\mathbf{x} = \mathbf{R}(\mathbf{r}, t)\, \mathbf{p}$ mit

$$\mathbf{R}(\mathbf{r}, t) = (1 - \cos t)\, \mathbf{r}\mathbf{r}^T + \cos t\, \mathbf{I} + \sin t \begin{bmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{bmatrix}. \tag{A.8}$$

Für die Matrix $\mathbf{R} = \mathbf{R}(\mathbf{r}, t)$ gilt:

$$\mathbf{R}^{-1} = \mathbf{R}^T, \quad \det(\mathbf{R}) = 1 \quad \text{und} \quad \mathbf{R}\,\mathbf{r} = \mathbf{r} \tag{A.9}$$

$\mathbf{R}$ ist also eine orthonormale Matrix mit Determinante 1. Sie besitzt einen Eigenwert 1 und $\mathbf{r}$ ist ein zugehöriger Eigenvektor. Aus Gleichung (A.8) folgt für $\mathbf{p}, \mathbf{q} \in \mathbb{R}^3$:

$$\begin{aligned} \mathbf{r} \times \mathbf{R}\mathbf{p} &= \cos t\, \mathbf{r} \times \mathbf{p} + \sin t\, \mathbf{r} \times (\mathbf{r} \times \mathbf{p}) \\ \mathbf{q}^T\mathbf{R}\mathbf{p} &= \mathbf{q}^T\mathbf{r}\mathbf{r}^T\mathbf{p} + \cos t\, (\mathbf{q}^T\mathbf{p} - \mathbf{q}^T\mathbf{r}\mathbf{r}^T\mathbf{p}) + \sin t\, \mathbf{r}^T(\mathbf{p} \times \mathbf{q}) \end{aligned} \tag{A.10}$$

Starten wir nun umgekehrt mit einer beliebigen Matrix $\mathbf{R}$, die die Bedingungen in (A.9) erfüllt, so können wir uns fragen, ob $\mathbf{R} \in \mathbb{R}^{3\times 3}$ als eine Abbildung im $\mathbb{R}^3$ interpretiert werden kann, die einer Rotation um eine Achse entspricht. Die Antwort gibt ein Satz von Euler:

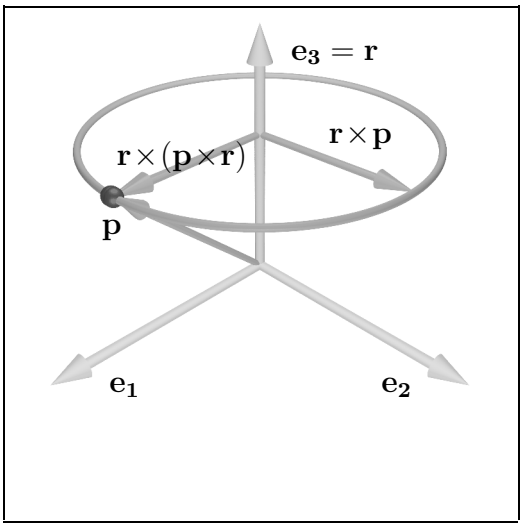


Abbildung A.2: Rotation eines Punktes $\mathbf{p}$ um eine Achse $\mathbf{r}$

Den Übergang eines starren Körpers von einer Orientierung in eine andere kann man als eine Rotation um eine geeignete Achse beschreiben.

Um diesen Zusammenhang zu verstehen, berechnen wir zunächst das charakteristische Polynom von $\mathbf{R}$. Allgemein gilt für 3×3–Matrizen:

$$\chi_{\mathbf{R}}(\lambda) = \det(\mathbf{R} - \lambda\mathbf{I}) = -\lambda^3 + trace(\mathbf{R})\lambda^2 - trace(adj(\mathbf{R}))\lambda + \det(\mathbf{R})$$

Da in unserem Fall $adj(\mathbf{R}) = \mathbf{R}^T$, gilt

$$\begin{aligned} \chi_{\mathbf{R}}(\lambda) &= -\lambda^3 + trace(\mathbf{R})\lambda^2 - trace(\mathbf{R})\lambda + 1 \\ &= -(\lambda - 1)(\lambda^2 - (trace(\mathbf{R}) - 1)\lambda + 1) \end{aligned}$$

Das zeigt uns nicht nur, daß 1 ein Eigenwert ist, sondern auch, daß der Spur $trace(\mathbf{R})$ eine besondere Bedeutung zukommt. Um dies noch deutlicher zu sehen, führen wir eine Koordinatentransformation durch. Als einen neuen Basisvektor wählen wir einen normierten Eigenvektor zum Eigenwert 1, den wir $\mathbf{r}$ nennen, und ergänzen ihn durch $\mathbf{u}$ und $\mathbf{v}$ zu einem Orthonormalsystem: $\mathbf{r}^T\mathbf{u} = 0$ und $\mathbf{v} = \mathbf{r} \times \mathbf{u}$. In diesem neuen System stellt sich die durch $\mathbf{R}$ induzierte Abbildung $\mathbf{R}' = [\mathbf{u}, \mathbf{v}, \mathbf{r}]^{-1}\ \mathbf{R}\ [\mathbf{u}, \mathbf{v}, \mathbf{r}]$ folgendermaßen dar:

$$R' = \begin{bmatrix} \mathbf{u}^T\mathbf{R}\mathbf{u} & \mathbf{u}^T\mathbf{R}\mathbf{v} & \mathbf{u}^T\mathbf{R}\mathbf{r} \\ \mathbf{v}^T\mathbf{R}\mathbf{u} & \mathbf{v}^T\mathbf{R}\mathbf{v} & \mathbf{v}^T\mathbf{R}\mathbf{r} \\ \mathbf{r}^T\mathbf{R}\mathbf{u} & \mathbf{r}^T\mathbf{R}\mathbf{v} & \mathbf{r}^T\mathbf{R}\mathbf{r} \end{bmatrix} = \begin{bmatrix} \mathbf{u}^T\mathbf{R}\mathbf{u} & -\mathbf{v}^T\mathbf{R}\mathbf{u} & 0 \\ \mathbf{v}^T\mathbf{R}\mathbf{u} & \mathbf{u}^T\mathbf{R}\mathbf{u} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Dies gilt, weil $\mathbf{R}\mathbf{r} = \mathbf{r}$ und $\mathbf{u}^T\mathbf{R}\mathbf{v} = -\mathbf{v}^T\mathbf{R}\mathbf{u}$ und $\mathbf{v}^T\mathbf{R}\mathbf{v} = (\mathbf{r} \times \mathbf{u})^T\mathbf{R}(\mathbf{r} \times \mathbf{u}) = (\mathbf{r} \times \mathbf{u})^T(\mathbf{r} \times \mathbf{R}\mathbf{u}) = \mathbf{u}^T\mathbf{R}\mathbf{u}$. Außerdem gilt wegen des Determinantenmultiplikationssatzes:

$$\det(\mathbf{R}) = \det(\mathbf{R}') = (\mathbf{u}^T\mathbf{R}\mathbf{u})^2 + (\mathbf{v}^T\mathbf{R}\mathbf{u})^2 = 1.$$

Deshalb dürfen wir $\cos t := \mathbf{u}^T\mathbf{R}\mathbf{u}$ und $\sin t := \mathbf{v}^T\mathbf{R}\mathbf{u}$ setzen. Es folgt:

$$\mathbf{R}' = \begin{bmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Dies läßt uns nun erkennen, daß $\mathbf{R}'$ und somit $\mathbf{R}$ eine Abbildung darstellt, die einer Rotation um die Achse $\mathbf{r}$ um einen Winkel t entspricht. Da die Spur $trace(\mathbf{R})$ als einer der Koeffizienten des charakteristischen Polynoms $\chi_{\mathbf{R}}(\lambda)$ eine Invariante unter Basiswechseln darstellt, können wir den Winkel t aus $\mathbf{R}$ leicht berechnen:

$$trace(\mathbf{R}) = trace(\mathbf{R}') = 2\cos t + 1 \tag{A.11}$$

Der Eigenvektor $\mathbf{r}$ zum Eigenwert 1 gibt die Rotationsachse an. Letztlich haben wir also gezeigt, daß sich jede Matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$, die (A.9) erfüllt, in der Form (A.8) darstellen läßt. Anhand dieser Form kann man aus $\mathbf{R}$ auch leicht die Achse $\mathbf{r}$ rekonstruieren:

$$\mathbf{r} \;=\; \frac{1}{2\sin t}(R_{32} - R_{23}, R_{13} - R_{31}, R_{21} - R_{12})^T \tag{A.12}$$

Die Lage eines starren Körpers läßt sich durch einen Positionsvektor im $\mathbb{R}^3$ und durch eine Orientierungsangabe in Form einer orthonormalen Matrix $\mathbb{R}^{3\times3}$ spezifizieren. Wir betrachten zwei verschiedene Lagen eines Körpers mit den Positionsvektoren $\mathbf{a}$ bzw. $\mathbf{b}$ und den Orientierungen $\mathbf{A}$ bzw. $\mathbf{B}$. Wenn $\mathbf{A} = \mathbf{B}$ ist, dann können wir den Körper durch eine einfache Translation um den Vektor $\mathbf{b} - \mathbf{a}$ von der Ausgangslage in die Endlage überführen. Anderenfalls haben wir eine Orientierungsänderung der Form $\mathbf{R} = \mathbf{B} \cdot \mathbf{A}^T$. Diese können wir als eine Rotation um eine geeignete Achse beschreiben. Im allgemeinen kann eine Lageänderung also als eine Komposition einer Translation und einer Rotation um eine feste Achse dargestellt werden. Die Translation kann entfallen, wenn der Vektor $\mathbf{b} - \mathbf{a}$ senkrecht auf der Achsrichtung $\mathbf{r}$ steht. Aus $\mathbf{R}$ können wir mittels (A.11) und (A.12) den Rotationswinkel t und die Richtung $\mathbf{r}$ der gesuchten Rotationsachse berechnen. Wir setzen

$$t \;=\; \arccos\left(\frac{1}{2}(trace(\mathbf{R}) - 1)\right)$$

und haben uns damit für eine Rotation im Gegenuhrzeigersinn um einen Winkel $0 \le t \le \pi$ entschieden, da $\arccos : [-1, 1] \to [0, \pi]$. Im Fall der reinen Rotation legen wir die Rotationsachse vollständig fest, indem wir zusätzlich einen Punkt $\mathbf{o}$ auf dieser Achse angeben. Wir wählen $\mathbf{o}$ als Mittelpunkt des Kreises, den die Punkte $\mathbf{a}$ und $\mathbf{b}$ bei einer Volldrehung um die gesuchte Achse beschreiben würden. Dieser Kreis liegt in einer Ebene mit dem Normalenvektor $\mathbf{r}$. $\mathbf{o}$ liegt auf der Gerade

$$\mathbf{x} = \frac{\mathbf{a} + \mathbf{b}}{2} + \mu\frac{\mathbf{r} \times (\mathbf{b} - \mathbf{a})}{|\mathbf{b} - \mathbf{a}|}.$$

Da die Vektoren $\mathbf{a} - \mathbf{o}$ und $\mathbf{b} - \mathbf{o}$ einen Winkel von t einschließen, erhält man den Punkt $\mathbf{o}$ für $\mu = \pm\frac{|\mathbf{b}-\mathbf{a}|/2}{\tan(t/2)}$. Den negativen Wert dürfen wir unberücksichtigt lassen, weil er zu einer Rotation mit falschem Drehsinn führen würde. Für $\mathbf{o}$ erhalten wir dann:

$$\mathbf{o} \;=\; \frac{1}{2}\left(\mathbf{a} + \mathbf{b} + \frac{1}{\tan(t/2)}\mathbf{r} \times (\mathbf{b} - \mathbf{a})\right)$$

■

Oft ist es wichtig, Gleichung (A.7) nach t aufzulösen:

$$\begin{aligned} t &= \arctan(\det[\mathbf{r},\mathbf{p},\mathbf{x}], \mathbf{x}^T\mathbf{p} - \mathbf{x}^T\mathbf{r}\mathbf{r}^T\mathbf{p}) \quad \text{bzw.} \\ t &= \arctan(p_1x_2 - p_2x_1, p_1x_1 + p_2x_2) \quad \text{für} \quad \mathbf{r} = \mathbf{e_3} = (0,0,1)^T \end{aligned} \tag{A.13}$$

Die Funktion arctan : $\mathbb{R}^2 \to [0, 2\pi)$ berechnet dabei die Phase bei der Umwandlung von kartesischen Koordinaten in Polarkoordinaten:

$$x_1 + ix_2 = \sqrt{x_1^2 + x_2^2}\, e^{i \arctan(x_2, x_1)}$$

Folgendes Gleichungssystem, das man als den Schnitt einer Geraden mit dem Einheitskreis auffassen kann, taucht bei Rotationsproblemen immer wieder auf:

$$\begin{aligned} \alpha\, \cos t + \beta\, \sin t + \gamma &= 0 \\ \cos^2 t + \sin^2 t &= 1 \end{aligned}$$

$\sin t$ und $\cos t$ ergeben sich durch Lösung einer quadratischen Gleichung zu:

$$\begin{aligned} \sin t_{1,2} &= \frac{-\beta\gamma \pm \alpha\sqrt{\alpha^2 + \beta^2 - \gamma^2}}{\alpha^2 + \beta^2} \\ \cos t_{1,2} &= \frac{-\alpha\gamma \mp \beta\sqrt{\alpha^2 + \beta^2 - \gamma^2}}{\alpha^2 + \beta^2} \end{aligned} \tag{A.14}$$

A.4 Schnitt von Hyperebenen im $\mathbb{R}^d$

Gegeben seien k $(d-1)$–dimensionale Hyperebenen im $\mathbb{R}^d$ in der Form

$$H_j : \sum_{i=1}^{d} a_{j,i}x_i = a_{j,0} \qquad \text{für} \quad j = 1, \ldots, k.$$

Gesucht ist die Projektion des Schnittes $\bigcap_{j=1}^{k} H_j$ in den $x_1x_2 \ldots x_{d-k+1}$–Raum. In diesem $(d-k+1)$–dimensionalen Raum stellt die Projektion des Schnittes selbst eine $(d-k)$–dimensionale Hyperebene dar. Die Lösung des Problems ergibt sich aus der Lösung des folgenden linearen Gleichungssystems:

$$\begin{bmatrix} a_{1,d-k+1} & \ldots & a_{1,d} \\ \vdots & & \vdots \\ a_{k,d-k+1} & \ldots & a_{k,d} \end{bmatrix} \cdot \begin{bmatrix} x_{d-k+1} \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_k \end{bmatrix} \quad \text{mit} \quad b_j = a_{j,0} - \sum_{i=1}^{d-k} a_{j,i}x_i$$

Wir bestimmen x_{d-k+1} mit Hilfe der Kramer'schen Regel:

$$\det \begin{bmatrix} a_{1,d-k+1} & \ldots & a_{1,d} \\ \vdots & & \vdots \\ a_{k,d-k+1} & \ldots & a_{k,d} \end{bmatrix} x_{d-k+1} = \det \begin{bmatrix} b_1 & a_{1,d-k+2} & \ldots & a_{1,d} \\ \vdots & \vdots & & \vdots \\ b_k & a_{k,d-k+2} & \ldots & a_{k,d} \end{bmatrix}$$

Die Expansion der Determinante ergibt die gesuchte Bestimmungsgleichung:

$$\sum_{i=1}^{d-k+1} \det \begin{bmatrix} a_{1,i} & a_{1,d-k+2} & \ldots & a_{1,d} \\ \vdots & \vdots & & \vdots \\ a_{k,i} & a_{k,d-k+2} & \ldots & a_{k,d} \end{bmatrix} x_i = \det \begin{bmatrix} a_{1,0} & a_{1,d-k+2} & \ldots & a_{1,d} \\ \vdots & \vdots & & \vdots \\ a_{k,0} & a_{k,d-k+2} & \ldots & a_{k,d} \end{bmatrix} \tag{A.15}$$

A.5 Berechnung potentieller Kontaktsituationen für die Bewegung von Polyedern mit zwei translatorischen Freiheitsgraden

Gegeben seien ein bewegtes Polyeder P_1 und ein Hindernispolyeder P_2. P_1 besitze zwei translatorische Freiheitsgrade, die durch die Richtungsvektoren $\mathbf{s_1}$ und $\mathbf{s_2}$ beschrieben werden sollen. In Abwesenheit von P_2 kann P_1 jede Position der Form

$$P_1 + x_1\mathbf{s_1} + x_2\mathbf{s_2} \qquad (x_1, x_2) \in \mathbb{R}^2$$

einnehmen. Zu einem Kontakt zwischen P_1 und P_2 kann es nur dann kommen, wenn ein Eckpunkt des einen Polyeders auf eine Fläche des anderen Polyeders trifft oder wenn sich zwei Kanten von unterschiedlichen Polyedern berühren. Wir betrachten diese beiden Fälle nun genauer:

A.5.1 Kontakt zwischen Punkt und Fläche

Gegeben seien ein bewegter Punkt $\mathbf{p}$ und eine stationäre, konvexe Fläche

$$f \;=\; \{\mathbf{x} \mid \mathbf{x} = \sum_{j=0}^{k-1} \lambda_j \mathbf{v_j},\ \sum_{j=0}^{k-1} \lambda_j = 1,\ \lambda_j \geq 0\},$$

die in der Ebene $\mathbf{n}^T\mathbf{x} = n_0$ liege und die Eckpunkte $\mathbf{v_0}, \ldots, \mathbf{v_{k-1}}$ besitze. Es gilt:

$$\mathbf{p} \cap f \neq \emptyset \;\Leftrightarrow\; \mathbf{n}^T\mathbf{p} = n_0 \wedge \det \begin{bmatrix} 0 & 1 & 1 & 1 \\ \mathbf{n} & \mathbf{p} & \mathbf{v_j} & \mathbf{v_{j+1}} \end{bmatrix} \leq 0 \quad \forall j$$

Substitution von $\mathbf{p}$ durch $\mathbf{p} + \sum x_i\mathbf{s_i}$ ergibt folgende Bedingungen:

$$\sum_{i=1}^{2} x_i \mathbf{n}^T\mathbf{s_i} = n_0 - \mathbf{n}^T\mathbf{p}$$

$$\sum_{i=1}^{2} x_i \det \begin{bmatrix} 0 & 0 & 1 & 1 \\ \mathbf{n} & \mathbf{s_i} & \mathbf{v_j} & \mathbf{v_{j+1}} \end{bmatrix} + \det \begin{bmatrix} 0 & 1 & 1 & 1 \\ \mathbf{n} & \mathbf{p} & \mathbf{v_j} & \mathbf{v_{j+1}} \end{bmatrix} \leq 0$$

Auf diese Art erhalten wir eine lineare Gleichung, die eine Gerade im Konfigurationsraum ($\mathbb{R}^2$) repräsentiert und k lineare Ungleichungen, die diese Gerade beschneiden, so daß im allgemeinen die Lösungsmenge ein Liniensegment darstellt.

A.5.2 Kontakt zwischen zwei Liniensegmenten

Gegeben seien ein bewegtes Liniensegment l_{ab} und ein stationäres Liniensegment l_{cd}:

$$\begin{aligned} l_{ab} &= \{\mathbf{x} \mid \mathbf{x} = \mathbf{a} + \mu(\mathbf{b} - \mathbf{a}), 0 \leq \mu \leq 1\} \\ l_{cd} &= \{\mathbf{x} \mid \mathbf{x} = \mathbf{c} + \nu(\mathbf{d} - \mathbf{c}), 0 \leq \nu \leq 1\} \\ \mathbf{n} &= (\mathbf{b} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c}) \end{aligned}$$

Falls $\mathbf{n} \neq \mathbf{0}$ gilt:

$$\begin{aligned} l_{ab} \cap l_{cd} \neq \emptyset \quad \Leftrightarrow \quad & \det \begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix} = 0 \wedge \\ & \det \begin{bmatrix} 0 & 1 & 1 & 1 \\ \mathbf{n} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix} \geq 0 \wedge \det \begin{bmatrix} 1 & 0 & 1 & 1 \\ \mathbf{a} & \mathbf{n} & \mathbf{c} & \mathbf{d} \end{bmatrix} \geq 0 \wedge \\ & \det \begin{bmatrix} 1 & 1 & 0 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{n} & \mathbf{d} \end{bmatrix} \leq 0 \wedge \det \begin{bmatrix} 1 & 1 & 1 & 0 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{n} \end{bmatrix} \leq 0 \end{aligned}$$

Substitution von $\mathbf{a}$ und $\mathbf{b}$ durch $\mathbf{a} + \sum x_i \mathbf{s_i}$ und $\mathbf{b} + \sum x_i \mathbf{s_i}$ ergibt folgende Bedingungen:

$$\begin{aligned} & \sum_{i=1}^{2} x_i \det \begin{bmatrix} 0 & 1 & 1 & 1 \\ \mathbf{s_i} & \mathbf{b}-\mathbf{a} & \mathbf{c} & \mathbf{d} \end{bmatrix} + \det \begin{bmatrix} 1 & 1 & 1 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix} = 0 \\ & \det \begin{bmatrix} 1 & 0 & 1 & 1 \\ \mathbf{a} & \mathbf{n} & \mathbf{c} & \mathbf{d} \end{bmatrix} \leq \sum_{i=1}^{2} x_i \det \begin{bmatrix} 0 & 0 & 1 & 1 \\ \mathbf{s_i} & \mathbf{n} & \mathbf{c} & \mathbf{d} \end{bmatrix} \leq \det \begin{bmatrix} 0 & 1 & 1 & 1 \\ \mathbf{n} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{bmatrix} \\ & \det \begin{bmatrix} 1 & 1 & 0 & 1 \\ \mathbf{a} & \mathbf{b} & \mathbf{n} & \mathbf{d} \end{bmatrix} \leq \sum_{i=1}^{2} x_i \det \begin{bmatrix} 0 & 0 & 1 & 1 \\ \mathbf{s_i} & \mathbf{n} & \mathbf{a} & \mathbf{b} \end{bmatrix} \leq \det \begin{bmatrix} 1 & 1 & 1 & 0 \\ \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{n} \end{bmatrix} \end{aligned}$$

Auch hier erhalten wir eine lineare Gleichung und vier lineare Ungleichungen. Die Lösungsmenge ist im allgemeinen wieder ein Liniensegment.

Literaturverzeichnis

[Ag90] P. Agarwal:
Partitioning Arrangements of Lines II: Applications, Discrete Comp. Geometry 5 (1990), S. 533-573

[AM92] P. Agarwal, J. Matousek:
Ray shooting and Parametric Search, 24th ACM STOC (1992), S. 517-526

[AS90] P. Agarwal, M. Sharir:
Red-Blue Intersection Detection Algorithms, with Applications to Motion Planning and Collision Detection, SIAM J. Comp. Vol. 19(2) (1990), S. 297-321

[Be92] M. de Berg:
Dynamic output-sensitive hidden surface removal for c–oriented polyhedra, Comp. Geom. Theory and Applications 2 (1992), S. 119-140

[Bo79] J.W. Boyse:
Interference Detection Among Solids and Surfaces, CACM Vol. 22(1) (1979), S. 3-9

[Ca84] J. Canny:
On Detecting Collisions between Polyhedra, Proc. ECAI (1984), S. 533-542

[Ca87] J. Canny:
The Complexity of Robot Motion Planning, MIT Press (1987)

[Ch93] T. Chadzelek:
Eine Heuristik zur Planung von translatorischen Bewegungen, Fortgeschrittenenpraktikum an der Universität des Saarlandes (1993)

[Co94] W. Collet:
Ein geometrischer Modellierer für Polyeder auf der Basis eines Bisektionsverfahrens, Diplomarbeit an der Universität des Saarlandes (1994)

[Cr89] J. Craig:
Introduction to Robotics, Mechanics and Control, Addison-Wesley Publishing (1989)

[DHKS90] D. Dobkin, J. Hershberger, D. Kirkpatrick, S. Suri:
Implicitly Searching Convolutions and Computing Depth of Collision, Lecture Notes in Computer Science 450 (1990), S. 165-180

[DK85] D.P. Dobkin, D.G. Kirkpatrick:
A Linear Algorithm for Determining the Separation of Convex Polyhedra, J. Algor. 6 (1985), S. 381-392

[DK90] D.P. Dobkin, D.G. Kirkpatrick:
Determining the Separation of Preprocessed Polyhedra – a Unified Approach, Lecture Notes in Computer Science 443 (1990), S. 400-413

[DL76] D.P. Dobkin, R.J. Lipton:
Multidimensional Searching Problems, SIAM J. Comp. 5 (1976), S. 181-186

[DS65] H. Davenport, A. Schinzel:
A Combinatorial Problem Connected With Differential Equations, Amer. J. Math. 87 (1965), S. 684-694

[DSST89] J.R. Driscoll, N. Sarnak, D.D. Sleator, R.E. Tarjan:
Making Data Structures Persistent, J. Computer System Sciences 38 (1989), S. 86-124

[Ed87] H. Edelsbrunner:
Algorithms in Combinatorial Geometry, Springer Verlag, Berlin (1987)

[EM81] H. Edelsbrunner, H.A. Maurer:
On the Intersection of Orthogonal Objects, Information Processing Letters 13 (1981), S. 177-181

[ES93] J. Eckstein, A. Schacke:
Bewegungsplanung mit zwei Freiheitsgraden, Fortgeschrittenenpraktikum an der Universität des Saarlandes (1993)

[FKN80] H. Fuchs, Z. Kedem, B. Naylor:
On visible surface generation by a priori tree structures, Computer Graphics (SIGGRAPH'80 Conference Proceedings), S. 124-133

[Fo86] S. Fortune:
A Sweepline Algorithm For Voronoi Diagrams, Proceedings of the Second ACM Symposium on Computational Geometry (1986), S. 124-133

[Fr93] M. Fritz:
Kollisionstests in der Objektmontage, Diplomarbeit an der Universität des Saarlandes (1993)

[GSS89] L.J. Guibas, M. Sharir, S. Sifrony:
On the General Motion-Planning Problem with Two Degrees of Freedom, Discrete Computational Geometry 4 (1989), S. 491-521

[Ho89] C.M. Hoffmann:
Geometric and Solid Modeling. An Introduction, Morgan Kaufmann, San Mateo, California (1989), S. 96-109

[HS86] S. Hart, M. Sharir:
Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes, Combinatorica 6 (1986), S. 151-177

[La91] J.C. Latombe:
Robot Motion Planning, Kluwer Academic Publishers (1991)

[Lo83] T. Lozano-Pérez:
Spatial Planning: A Configuration Space Approach, IEEE Transactions on Computers, C-32(2) (1983), S. 108-120

[LW79] T. Lozano-Pérez, M.A. Wesley:
An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles, CACM 22(10) (1979), S. 560-570

[Ma93] J. Matousek:
On Vertical ray shooting in arrangements Comp. Geom. Theory and Applications 2 (1993), S. 279-285

[Me83] N. Megiddo:
Linear-Time Algorithms for Linear Programming in $\mathbf{R}^3$ *and Related Problems*, SIAM J. Comp. Vol. 12, No. 4 (Nov 1983), S. 759-776

[Me84] N. Megiddo:
Linear Programming in Linear Time when the Dimension is Fixed, JACM, Vol. 31 No. 1 (Jan 1984), S. 114-127

[Meh84] K. Mehlhorn:
Multi-dimensional Searching and Computational Geometry, Springer-Verlag (1984)

[MS85] K. Mehlhorn, K. Simon:
Intersecting two Polyhedra one of which is Convex, Lecture Notes in Computer Science 199 (1985), S. 534-542

[MS88] R.P. Martin, P.C. Stephenson:
Putting objects into boxes, CAD 6 (1988), S. 506-514

[NM65] J.A. Nelder, R. Mead:
Computer Journal, Vol. 7 (1965), S. 305-309

[Nu85] O. Nurmi:
A fast line-sweep algorithm for hidden-line elimination, BIT, Vol. 25 (1985), S. 466-472

[OY82] C. Ó'Dúnlaing, C.K. Yap:
A Retraction Method for Planning the Motion of a Disc, Journal of Algorithms 6 (1982), S. 104-111

[Po84] M.J. Post:
Minimum spanning ellipsoids, Proc. 16th Symposium on Theory of Computing (1984), S. 108-116

[PR69] D. Pieper, B. Roth:
The Kinematics of Manipulators Under Computer Control, Proc. 2nd International Congress on Theory of Machines and Mechanisms Vol. 2 (1969), S. 159-169

[PY89] M. Paterson, F. Yao:
Binary Partitions with Applications to Hidden–Surface Removal and Solid Modelling, Proc. 5th Symposium on Computational Geometry (1989), S. 23-32

[Ra86] R. Rannacher:
Vorlesung zur praktischen Mathematik, Saarbrücken (1986)

[Sa89] H. Samet:
The Design and Analysis of Spatial Data Structures, Addison-Wesley Publishing (1989)

[Sch92] A. Schweikard:
A Simple Path Search Strategy Based on Calculation of Free Sections of Motion, Engng. Applic. Artif. Intell. Vol. 5 (1992), S. 1-10

[Se91] R. Seidel:
Lower-dimensional linear programming and convex hulls made easy, Discrete Comp. Geom. 6 (1991), S. 423-434

[Sh88a] M. Sharir:
The Shortest Watchtower and Related Problems for Polyhedral Terrains, Information Processing Letters 29 (1988), S. 265-270

[Sh88b] M. Sharir:
Davenport-Schinzel Sequences and their Geometric Applications, Theor. Found. Comp. Graphics and CAD, NATO ASI Series, Vol F-40 (1988), S. 253-278

[Si93] M. Sinnwell:
Interaktive Robotersimulation mit Kollisionserkennung, Diplomarbeit an der Universität des Saarlandes (1993)

[Sz74] E. Szemeredi:
On a Problem by Davenport and Schinzel, Acta Arithmetica 25 (1974), S. 213-224

[We91] E. Welzl:
Smallest enclosing disks (balls and ellipsoids), Lecture Notes in Computer Science (1991), S. 359-370