Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik I


Elmar Langetepe

Methoden der Offline Bewegungsplanung

BA-INF 124

Wintersemester 2016/2017
Dozent: Elmar Langetepe
Skript: Tom Kamphans/Elmar Langetepe

Einleitung

Die Vorstellung von autonomen Maschinen, die den Menschen schwierige und gefährliche Arbeiten abnehmen, hat immer wieder die Phantasie vieler Science–Fiction Autoren[1] beflügelt und viele Wissenschaftler unterschiedlicher Disziplinen beschäftigt. Das Feld "Robotics" involviert daher verschiedenste Bereiche, darunter:

- Elektrotechnik, Elektronik, Mechanik
- Kontroll– und Regelungstechnik
- Systemtheorie
- Softwareengineering
- Künstliche Intelligenz (Neuronale Netze, Fuzzy Logic)
- Algorithmik

Im Rahmen dieser Vorlesung werden wir uns auf den letzten Punkt konzentrieren und Probleme aus der Algorithmischen Geometrie behandeln, die im Zusammenhang mit der Steuerung von Robotern interessant sind.

Wir benötigen zur Behandlung dieser Fragestellungen einige Begriffe und Verfahren der Algorithmischen Geometrie, die hier nicht in aller Ausführlichkeit vorgestellt werden können, sondern nur kurz angerissen oder gar nur als "black–box" verwendet werden. Für ein tieferes Verständnis dieser Themen sei auf die Vorlesung Algorithmische Geometrie oder auf die einschlägige Literatur in diesem Bereich verwiesen.

Die diskutierten Themen sind Varianten einer Grundaufgabe: Gegeben ist ein Robotersystem mit k Freiheitsgraden, das sich in einer Umgebung U mit oder ohne (stationären) Hindernissen bewegt, ein Startpunkt s und ein Zielpunkt t. Stelle fest, ob es eine kollisionsfreie und kostengünstige Bewegung von s nach t gibt und falls ja, bestimme eine solche. Wichtige Kriterien für die vorgestellten Lösungen sind dabei:

- Korrektheit, d.h. es soll genau dann eine kollisionsfreie Bewegung gefunden werden, wenn eine solche existiert.
- Effizienz der Berechnung, denn natürlich soll die Bewegung schnell berechnet werden, der Roboter soll nicht vor lauter Rechenaufwand auf der Stelle stehen bleiben.
- Qualität der Lösung, d.h. die gefundene Bahn ist auch günstig bzgl. der festgelegten Kostenmaße.

Zunächst wollen wir die Betrachtung von Kollisionsproblemen zurückstellen, uns auf punktförmige Roboter beschränken und die Länge der vom Roboter zurückgelegten Bahnen als Qualitätskriterium betrachten.

Literatur

Begleitend zu diesem Skript sei folgende Literatur empfohlen:

- Mark de Berg, Marc van Kreveld, Mark Overmars
  und Otfried Schwarzkopf:
  Computational Geometry: Algorithms and Applications.
  Springer-Verlag, Berlin 1997, [dBvKOS97]

[1] Der Begriff "Roboter" wurde von dem tschechischen Autor Karel Čapek (1890-1938) geprägt. [ČČ61]

- Rolf Klein:
  Algorithmische Geometrie.
  Addison–Welsey, Bonn 1997, [Kle97]

- Micha Sharir und Pankaj K. Agarwal:
  Davenport-Schinzel Sequences and Their Geometric Applications.
  Cambridge University Press, New York 1995, [SA95]

- Micha Sharir:
  Algorithmic Motion Planning.
  Kapitel 40 in: Jacob E. Goodman und Joseph O'Rourke: *Handbook of Discrete and Computational Geometry.*
  CRC Press LLC, Boca Raton, FL, 1997, [Sha97]

- Joseph S. B. Mitchell:
  Geometric Shortest Paths and Network Optimization.
  Kapitel 15 in: Jörg-Rüdiger Sack and Jorge Urrutia: *Handbook of Computational Geometry.*
  Elsevier Science Publishers B.V. North-Holland, 2000, [Mit00]

- Jean-Claude Latombe:
  Robot Motion Planning.
  Kluwer Academic Publishers, Boston, 1991, [Lat91]

- Jakob T. Schwartz und Chee-Keen Yap (Editor):
  Advances in Robotics.
  Lawrence Earlbaum, 1987, [Yap87]

- Sowie viele in Journalen und Konferenzbänden erschienene Arbeiten und auch Dissertationen wie:
  A. Frank van der Stappen:
  Motion Planning amidst Fat Obstacles.
  Dissertation, Dept. Comput. Sci., Utrecht Univ., NL, 1994, [vdS94]

An dieser Stelle sei auch auf das Geometrie–Labor

`http://www.geometrylab.de/`

mit Java–Applets aus dem Bereich Algorithmische Geometrie und Bewegungsplanung für Roboter hingewiesen.

Danksagungen

Sehr herzlich danken wir Annette Ebbers–Baumann, Dorthe Lübbert, Andrea Eubeler, Stefan Kamphans und allen Teilnehmern der Vorlesung für viele Korrekturhinweise und Verbesserungsvorschläge.

Korrekturhinweise

Trotz aller Sorgfalt ist dieses Skript bestimmt nicht fehlerfrei. Bitte teilt uns Fehler aller Art oder unverständlich aufgeschriebenes per Mail an
`Elmar.Langetepe@cs.uni-bonn.de` mit.

Kapitel 1

Kürzeste Pfade

Die Untersuchung kürzester Wege beschäftigt die Wissenschaft bereits seit dem Altertum. In der Mathematik befaßt man sich in der Theorie der metrischen Räume und in der Differentialgeometrie damit. So zeigten bereits Joh. Bernoulli (1698) und L. Euler (1728), daß für Kurven C von a nach b auf einer glatten Fläche S gilt: C ist Kürzeste $\implies C$ ist Geodätische, d. h. $\forall p \in C$: Schmiegeebene$(C, p) \perp$ Tangentialebene(S, p).

Dabei ist die Schmiegeebene die Ebene, an die C sich lokal anschmiegt,[1] und die Tangentialebene die in p tangential zu S liegende Ebene. Geodätische Kurven sind nur lokal nicht verkürzbar, im Gegensatz zu kürzesten Wegen, die auch global unverkürzbar sind. Anschaulich kann man sich geodätische Kurven als straff gespannte Gummibänder vorstellen. Die Umkehrung obigen Satzes gilt nicht: In Abbildung 1.1 ist die Helix von a' nach b' zwar eine Geodätische, aber keine Kürzeste.

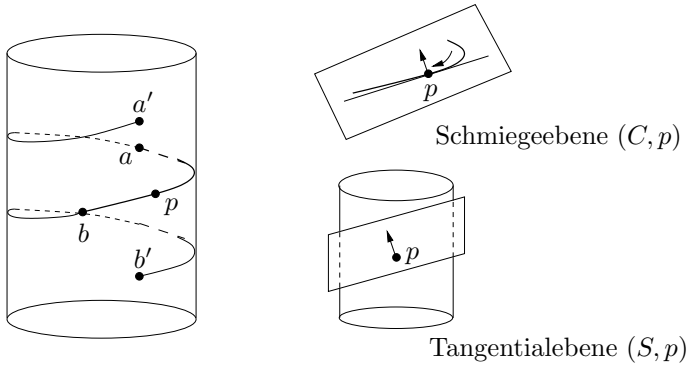


Abbildung 1.1: Kürzeste von a nach b, Geodätische von a' nach b'.

Wir interessieren uns hier für diskrete Umgebungen.

1.1 Kürzeste Pfade in der Ebene mit polygonalen Hindernissen

Die Umgebung, in der sich der Roboter bewegt, sei durch h Hindernisse in Form von Polygonen P_i mit insgesamt n Ecken gegeben. Dabei sind auch Polygone mit $\overset{\circ}{P}_i = \emptyset$ erlaubt,[2] also zu einem Linienzug degenerierte Polygone. Weiterhin seien ein Startpunkt s und Zielpunkt t gegeben, die nicht im Inneren eines Polygons liegen dürfen; es muß gelten $s, t \notin \bigcup \overset{\circ}{P}_i$. Die Aufgabe ist nun, einen Pfad π minimaler Länge von s nach t zu finden, der keine Hinderniskante kreuzt.

[1] Genauer: die Grenzlage einer durch drei benachbarte Kurvenpunkte p', p, p'' gegebene Ebene für die Grenzwerte $p' \longrightarrow p$ und $p'' \longrightarrow p$ [Stö99].

[2] $\overset{\circ}{P}$ bezeichnet das Innere eines Polygons, also alle Punkte aus P, deren ε–Umgebung vollständig in P liegt.

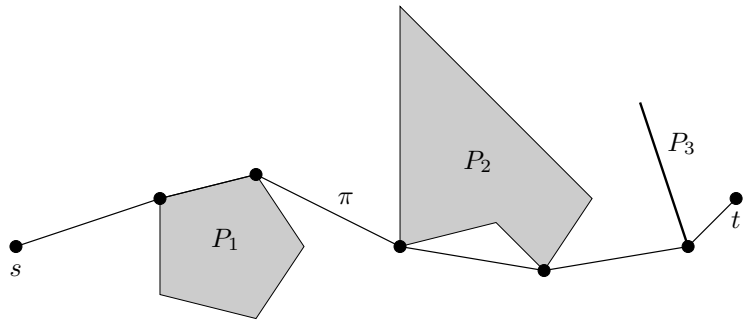


Abbildung 1.2: Umgebung mit polygonalen Hindernissen.

Der kürzeste Pfad ist nicht immer eindeutig, je nach Umgebung kann es exponentiell in h viele kürzeste Wege geben, Abbildung 1.3 zeigt eine Umgebung mit $h = 4m + 1$ Hindernissen (Liniensegmente) und 2^{m+1} kürzesten Wegen (Akman, 1987 [Akm87b]).

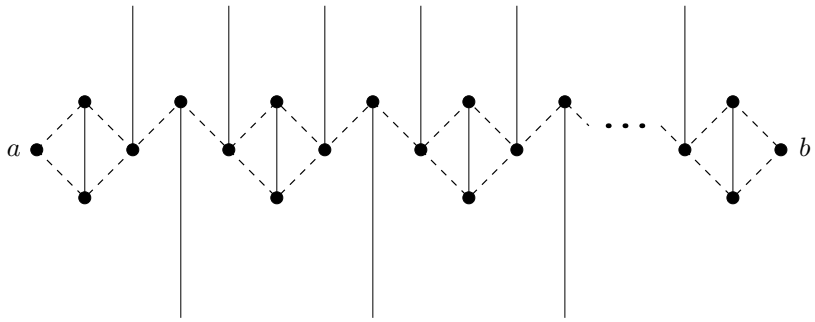
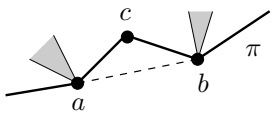


Abbildung 1.3: Es kann exponentiell viele kürzeste Wege geben.

Kürzeste Pfade haben die Gestalt einer polygonalen Kette, die nur an konvexen Polygonecken[3] abknicken kann, da sich die Kette andernfalls verkürzen ließe. Dies wird klar, wenn man sich den kürzesten Pfad als ein Gummiband vorstellt, das von s nach t gespannt ist: die Knicke im Gummiband müssen sich an einer Polygonecke abstützen, ohne eine Polygonecke würde sich das Gummiband strammziehen. In nebenstehender Abbildung ist π also kein kürzester Pfad, da sich der Knick in Punkt c nicht auf eine Polygonecke abstützt. Der kürzeste Pfad zwischen a und b wäre die gestrichelte Linie.



Genauer bestehen kürzeste Pfade aus einer Startkante von s aus, einer Zielkante zu t hin und ansonsten nur aus Kanten, die *gegenseitig sichtbare*, konvexe Hindernisecken verbinden. Diese Beobachtung führt zu einem ersten Ansatz zur Lösung des Problems, bei der wir auf eine aus der Algorithmischen Geometrie bekannte Struktur zurückgreifen können:

Definition 1.1 Sei $\mathcal{L}$ eine Menge sich nicht kreuzender Liniensegmente in der Ebene. Dann ist der **Sichtbarkeitsgraph** von $\mathcal{L}$ ein Graph $\mathrm{VisG}(\mathcal{L}) = (V, E)$ mit

$V = \{$ alle Endpunkte der Liniensegmente in $\mathcal{L}\,\}$

$E = \{\, (p, q) \mid p, q \in V,\ \overline{pq}$ kreuzt kein Liniensegment aus $\mathcal{L}\,\}$.

Der Sichtbarkeitsgraph einer Menge von Polygonen P_i entsteht aus $\mathrm{VisG}(\mathcal{L})$ mit $\mathcal{L} = \{$ Kanten der $P_i\,\}$ durch Entfernen aller im Inneren eines Polygons liegenden Sichtbarkeitskanten.

Der Sichtbarkeitsgraph von n Liniensegementen hat die Komplexität $O(n^2)$. Abbildung 1.4(i) zeigt ein Beispiel eines Arrangements von n Liniensegmenten, dessen Sichtbarkeitsgraph $\Omega(n^2)$ viele Kanten enthält. Jedoch kann der Sichtbarkeitsgraph auch deutlich weniger Kanten enthalten,

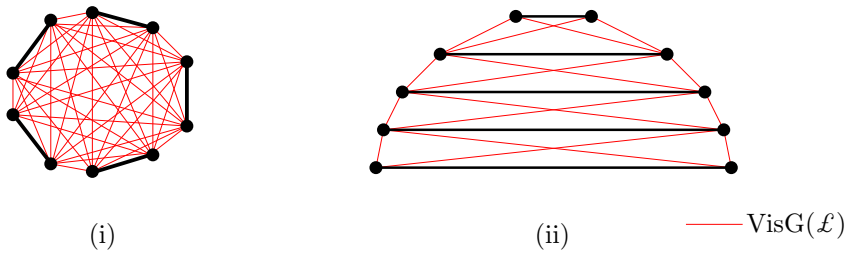


Abbildung 1.4: Der Sichtbarkeitsgraph kann (i) $\Omega(n^2)$ viele Kanten enthalten, aber auch (ii) nur $O(n)$ viele.

wie das Beispiel in Abbildung 1.4(ii) zeigt.

Algorithmus 1.1 Berechnung des kürzesten Weges mit Hilfe des Sichtbarkeitsgraphen

1. Berechne den Sichtbarkeitsgraph VisG(P).
2. Füge alle Kanten (s, v) mit $v \in V$ und s sieht v ein:
   $\forall v \in V$: Teste, ob das Segment $\overline{sv}$ von einer Randkante eines der Hindernisse geschnitten wird. Ist dies nicht der Fall, dann füge das Segment $\overline{sv}$ zum Sichtbarkeitsgraphen hinzu.
3. Füge alle Kanten (v, t) mit $v \in V$ und t sieht v ein (analog).
4. Berechne mit dem Algo. von Dijkstra in dem modifizierten Graph mit Distanzen als Kantengewichten alle kürzesten Wege von s.

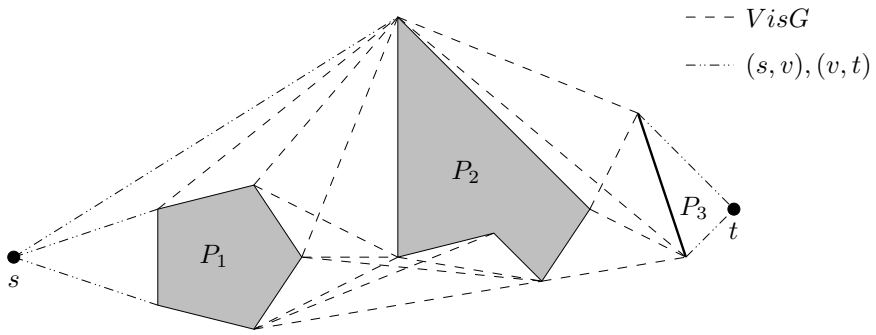
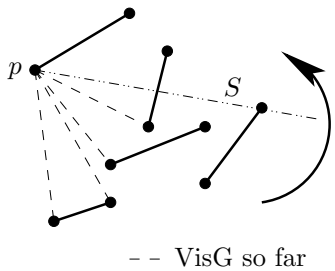


Abbildung 1.5: VisG und Kanten von s und zu t.

Damit läßt sich ein kürzester Weg wie in Algorithmus 1.1 beschrieben berechnen. Abbildung 1.5 zeigt das Ergebnis der Schritte 1 bis 3 auf der Umgebung aus Abbildung 1.2. Die zusätzlichen Kanten lassen sich offensichtlich in Zeit $O(n^2)$ einfügen, die Komplexität der anderen beiden Schritte bedarf einer genaueren Analyse. Beginnen wir mit der Berechnung des Sichtbarkeitsgraphen: Dieser läßt sich naiv in Zeit $O(n^3)$ berechnen, indem für jedes Paar (p, q) von Ecken das Segment $\overline{pq}$ auf Kreuzung mit den n Polygonrandsegmenten getestet wird. Ein besseres Laufzeitverhalten hat der in Algorithmus 1.2 skizzierte radiale Sweep, der für ein festes p Zeit $O(n \log n)$ benötigt, insgesamt also $O(n^2 \log n)$.



Dies kann noch verbessert werden, indem nicht jeder Punkt für sich einen radialen Sweep durchführt, sondern die Bearbeitung so synchronisiert wird, daß Sichtinformationen weitergereicht

[3] Polygonecke, deren Außenwinkel $> \pi$ ist.

Algorithmus 1.2 Sweep zur Berechnung des Sichtbarkeitsgraphen

Für jeden Endpunkt p:

- Sortiere alle Endpunkte rechts von p nach Winkeln von p.
- Drehe einen Strahl S von p gegen den Uhrzeigersinn von Süd nach Nord:
  - Führe in balanciertem Baum sortiert nach Abstand von p Buch darüber, welche Segmente von S gerade geschnitten werden.
  - Sei q der Schnittpunkt von S mit dem zu p nächsten Liniensegment auf S. Falls $q \in V$, berichte Sichtbarkeitskante $\overline{pq}$.

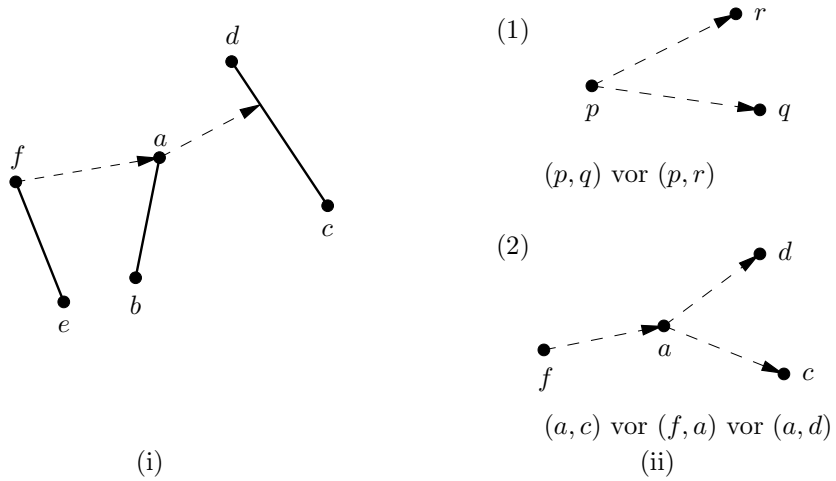


Abbildung 1.6: (i) Sichtbarkeitsinformationen können von a an f weitergereicht werden, (ii) Typen von Bedingungen an die Bearbeitungsreihenfolge.

werden können. Wenn z. B. in Abbildung 1.6(i) der Punkt f beim Sweep den Punkt a entdeckt, kann a die Information "$\overline{cd}$ ist sichtbar" an f weiterreichen.

Die Wahl der Bearbeitungsreihenfolge R hängt dabei von der Steigung $\text{slope}(p, q)$ des Segmentes zwischen p und q ab. Wir müssen dabei zwei Typen von Bedingungen erfüllen, siehe Abbildung 1.6(ii):

(1) $\text{slope}(p, q) < \text{slope}(p, r) \Rightarrow (p, q)$ vor (p, r) bearbeiten

(2) $\text{slope}(a, c) < \text{slope}(f, a) < \text{slope}(a, d) \Rightarrow (a, c)$ vor (f, a) vor (a, d)

Man beachte, dass in Bedingung (2) weder die erste noch die zweite Ungleichung aus Bedingung (1) folgt.

Die Wahl der Bearbeitungsreihenfolge werden wir später betrachten. Nehmen wir zunächst an, wir hätten eine solche Bearbeitungsreihenfolge R bereits festgelegt. Nun wird zur Initialisierung zu jedem Punkt p ein Zeiger $\wp$ auf das erste unterhalb von p liegende Segment berechnet; also das Liniensegment, das ein Strahl von p in negativer Y–Richtung als erstes treffen würde. Liegt unter p kein Liniensegment, wird der Zeiger auf NIL gesetzt.

Bei der Bearbeitung der Paare (p_i, q_i) aus der Liste R soll folgende Invariante erhalten bleiben:

> Sei q der zuletzt bearbeitete Partner von p. Dann gibt es einen Zeiger $\wp$ von p auf das in Sweeprichtung kurz hinter q sichtbare Segment σ.

Nach der Initialisierung ist die Invariante erfüllt, $\wp$ zeigt auf das von p in Richtung Süden liegende Segment. Vor der Bearbeitung eines Paares (p, r) sei q der zuletzt bearbeitete Partner

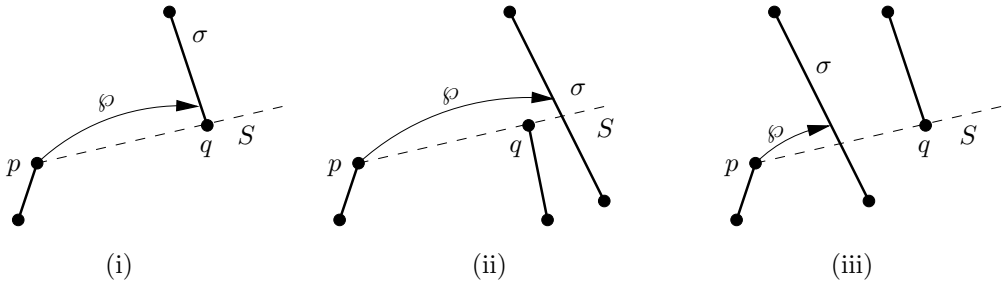


Abbildung 1.7: Es gibt drei mögliche Ausgangssituationen eines Punktes p mit zuletzt bearbeitetem Partner q und sichtbarem Segment σ.

von p, und der Zeiger von p zeige auf σ. Es gibt drei mögliche Ausgangssituationen von p, q und σ, siehe Abbildung 1.7:

(i) q ist von p aus sichtbar und Anfangspunkt eines Segments σ.
(ii) q ist von p aus sichtbar und Endpunkt eines Segments. σ ist das auf dem Strahl S von p durch q nächste Segment hinter q.
(iii) q ist von p aus nicht sichtbar. σ ist das zu p nächste Segment auf dem Strahl S von p durch q.

Bei der Bearbeitung des Paares (p, r) können nun folgende Fälle eintreten, siehe Abbildung 1.8:

1. r liegt hinter σ. In allen drei Ausgangssituationen zeigt $\wp$ nach wie vor auf σ.

2. r liegt vor σ. Dann ist r Anfangspunkt eines Segments τ und $\wp$ zeigt nun auf τ.

3. r ist Endpunkt von σ. Hier muss die Sichtbarkeitsinformation für r benutzt werden. Je nach Ausgangssituation gibt es in diesem Fall folgende drei Möglichkeiten, die prinzipiell analog behandelt werden können:

   (i) $\wp$ wird auf das erste Segment auf dem Strahl zwischen p und r hinter r gesetzt. Dies entspricht der Ausgangssituation (ii).
   (ii) In diesem Fall kann der Winkelbereich W zwischen den Strahlen von p durch q und r keinen Punkt enthalten, ansonsten wäre r nicht der nächste Partner von p. Außerdem muß r von p aus sichtbar sein. Andernfalls müßte sich ein Endpunkt in W befinden, oder σ wäre nicht das nach q sichtbare Segment.
   Falls ein Segment τ existiert auf das r zeigt, ist es durch die Bearbeitung des letzten Paares (r, v) bezüglich r entstanden, dass vor (p, r) dran war. Der Punkt v liegt unterhalb von W wegen Teil 2) der Bearbeitungsreihenfolge. Das Segment $\tau = \overline{uw}$ muss vollständig den obigen Winkelbereich W durchqueren und ist somit sichtbar von p aus. Der *untere* Endpunkt u kann nicht oberhalb vom Strahl durch r und v liegen (u könnte allerdings mit v identisch sein), da τ ja die Sichtbarkeitsbedingung erfüllen muss. Der *obere* Endpunkt w kann nicht zwischen dem Strahl durch r und v und dem Bereich W liegen, da dann (r, w) noch nach (r, v) und vor (p, r) betrachtet worden wäre. Der Endpunkt w kann wie oben bereits erwähnt nicht im Bereich W liegen. Folglich sieht p einen Punkt r' auf τ und $\wp$ wird also auf τ gesetzt.
   (iii) Analog zu (ii).

Damit können wir den Sichtbarkeitsgraphen nach der Berechnung der Bearbeitungsreihenfolge R in Zeit $O(n^2)$ bestimmen. Um R festzulegen, müssen wir die Liste der Punktpaare (p_i, q_i) so sortieren, daß die Steigungen der Liniensegmente $\overline{p_i q_i}$ die geforderten Eigenschaften (1) und (2) haben. Dabei können wir eine aus der Algorithmischen Geometrie bekannte Eigenschaft

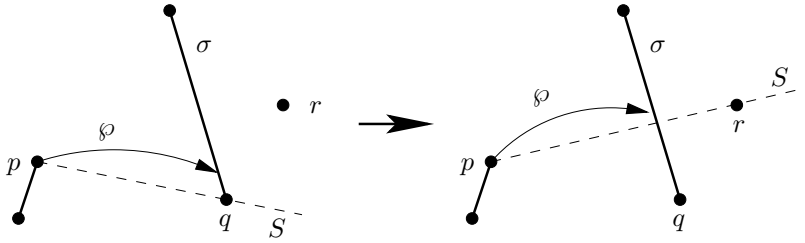
ausnutzen, die Dualität zwischen Punkten und Geraden, die gegeben ist durch die Abbildung[4]

$$\text{Punkt } p = (p_x, p_y) \longmapsto \text{ Gerade } p^* = \{\, Y = p_x X - p_y \,\}.$$

Seien $p = (p_x, p_y)$ und $q = (q_x, q_y)$ o. E. mit $p_x < q_x$ gegeben. Dann gilt für die Steigung des Segments

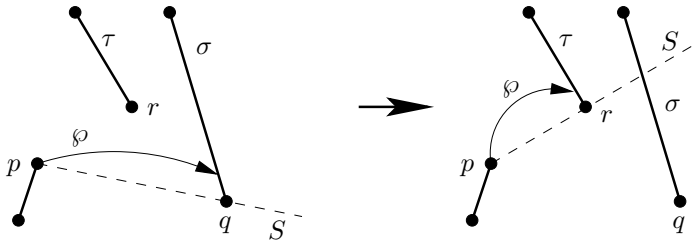
$$\text{slope}(p, q) = \frac{q_y - p_y}{q_x - p_x} \,.$$

Fall 1: r liegt hinter σ (Situation (i))



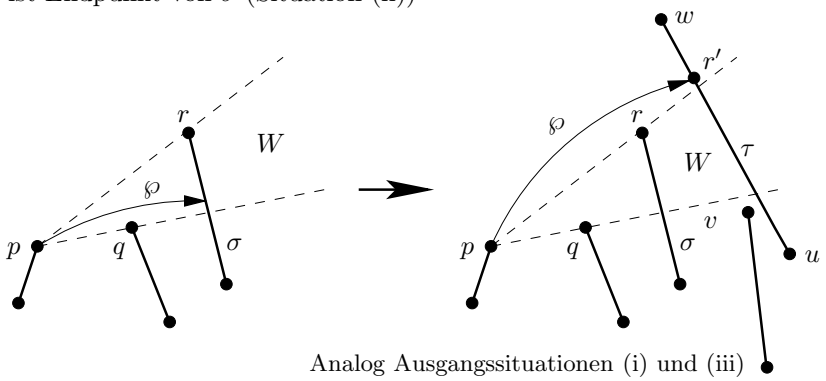
Analog Ausgangssituationen (ii) und (iii)

Fall 2: r liegt vor σ (Situation (i))



Analog Ausgangssituationen (ii) und (iii)

Fall 3: r ist Endpunkt von σ (Situation (ii))



Analog Ausgangssituationen (i) und (iii)

Abbildung 1.8: Drei Fälle bei der Bearbeitung eines Paares (p, r).

[4]Diese Dualisierung unterscheidet sich nur im Vorzeichen von der in [Kle05] betrachteten Dualisierung $p = (p_x, p_y) \longmapsto p^* = \{Y = p_x X + p_y\}$.

Dem entspricht die X–Koordinate des Schnittpunktes (x, y) der zu p und q dualen Geraden p^* und q^*:

$$\begin{aligned} Y = &\quad p_x X - p_y = q_x X - q_y \\ \Rightarrow \quad X = &\quad \frac{q_y - p_y}{q_x - p_x} = \text{slope}(p, q) \ . \end{aligned}$$

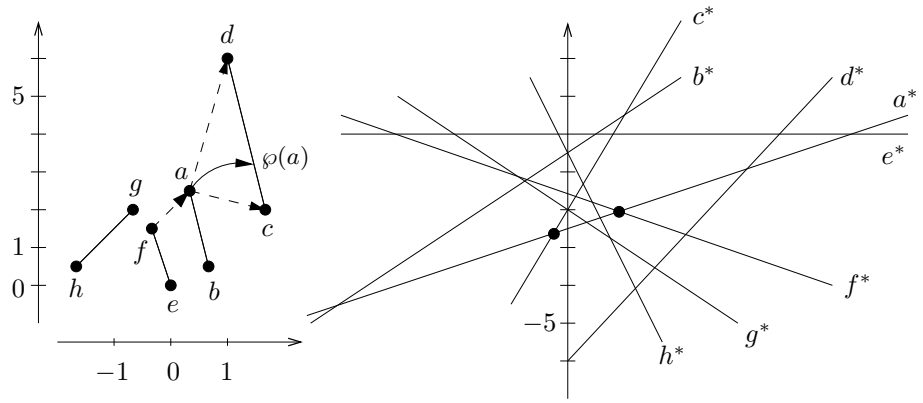


Abbildung 1.9: Dualisierung $p = (p_x, p_y) \longmapsto p^* = \{\, Y = p_x X - p_y \,\}$.

Wir können also die Bedingungen aus Abbildung 1.6(ii) so übersetzen:

(1) $\text{slope}(p, q) < \text{slope}(p, r) \Leftrightarrow p^* \cap\ q^*$ links von $p^* \cap\ r^*$ auf Gerade p^*

(2a) $\text{slope}(a, c) < \text{slope}(f, a) \Leftrightarrow a^* \cap\ c^*$ links von $f^* \cap\ a^*$ auf Gerade a^*

(2b) $\text{slope}(f, a) < \text{slope}(a, d) \Leftrightarrow f^* \cap\ a^*$ links von $a^* \cap\ d^*$ auf Gerade a^*

Also müssen wir die Schnittpunkte im Arrangement der Geraden p_i^* so bearbeiten, daß längs jeder Geraden die richtige Reihenfolge eingehalten wird. Abbildung 1.9 zeigt ein Beispiel eines Arrangements mit den dazu dualen Geraden. Da $\text{slope}(a, c) < \text{slope}(f, a)$ gilt, muß der Schnittpunkt von a^* mit c^* links des Schnittpunktes von f^* mit a^* auf a^* liegen.

Zu lösen ist nun folgendes Problem: Gegeben seien n Geraden im $\mathbb{R}^2$ in allgemeiner Lage, die eine Zerlegung der Ebene bewirken. Gesucht ist eine Reihenfolge der Schnittpunkte, die mit der X–Ordnung längs jeder Geraden verträglich ist. Dazu sind folgende Verfahren bekannt:

	Rechenzeit	Speicherplatz
Inkrementelle Konstruktion des Arrangements der Geraden (Edelsbrunner, O'Rourke, Seidel, 1986, [EOS86])	$O(n^2)$	$O(n^2)$
Sweep mit einer Geraden	$O(n^2 \log n)$	$O(n)$
Sweep mit einer Kurve (topologischer Sweep, Edelsbrunner, Guibas, 1986, [EG86, EG89])	$O(n^2)$	$O(n)$

Im folgenden konzentrieren wir uns auf den topologischen Sweep. Die wesentliche Idee dabei ist, den Sweep in der Ebene nicht mit einer Geraden durchzuführen, sondern mit einer Kurve, die topologisch einer Geraden entspricht:

Definition 1.2 In einem Arrangement $\mathcal{A}$ von Geraden heißt eine Kurve S **Pseudogerade**, wenn sie jede Gerade aus $\mathcal{A}$ in höchstens einem Punkt schneidet.

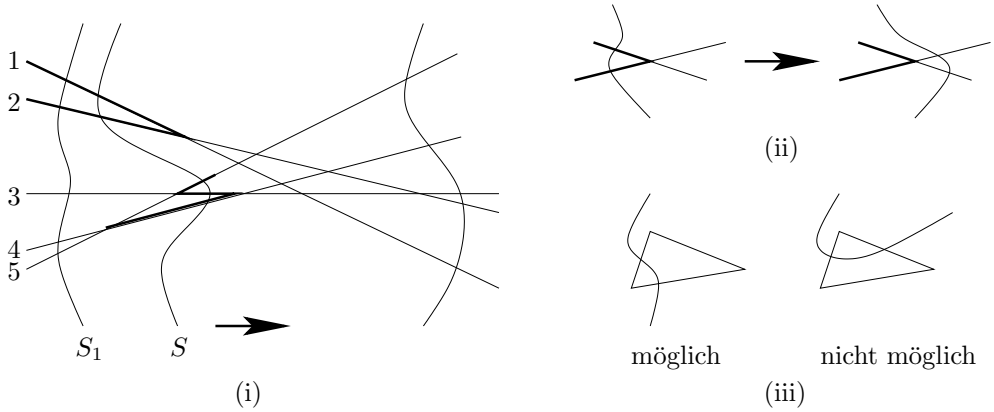


Abbildung 1.10: (i) Topologischer Sweep, (ii) Elementarschritt, (iii) Die Sweepkurve kann eine Zelle nur von oben betreten und nach unten verlassen.

Beim Sweep wird also eine Sweepkurve S (Pseudogerade) über das Arrangement $\mathcal{A}$ bewegt (Abbildung 1.10(i)). Die von S geschnittenen Geraden werden längs S sortiert in einem Verzeichnis mitgeführt. In einem Elementarschritt des Sweeps wird die Sweepkurve über genau einen Schnittpunkt hinweg geführt (Abbildung 1.10(ii)). Das Problem dabei ist, zwei konsekutive Geraden längs S mit Schnittpunkt rechts von S für das nächste Elementarereignis zu finden, wenn im Mittel dafür Zeit $O(1)$ zur Verfügung steht.

Die Existenz solcher zwei Geraden ist jedenfalls sichergestellt: Seien G_1 und G_2 die Geraden, deren Schnittpunkt rechts von S am weitesten links liegt, dann müssen G_1 und G_2 entlang S benachbart sein. Ein Elementarschritt ist also stets durchführbar, wenn es auch im allgemeinen mehrere Kandidaten dafür gibt.

Ein Ansatz zur Lösung dieses Problems ist, in einer Liste N entlang S sortiert für jede Gerade G festzuhalten, von welcher anderen Geraden G' das von S durchkreuzte Segment von G rechts begrenzt wird. Für das Arrangement in Abbildung 1.10(i) ergibt sich also entlang der ersten eingezeichneten Sweepkurve S_1 die Liste

$$N = \{\,(1,2),(2,1),(3,5),(4,5),(5,4)\,\}.$$

Aus dieser Liste läßt sich ablesen, daß die Geraden 1 und 2 sowie die Geraden 4 und 5 benachbart und somit Kandidaten für den nächsten Elementarschritt sind. Für die zweite Sweepkurve S ergibt sich die Liste

$$N = \{\,(1,2),(2,1),(5,1),(3,4),(4,3)\,\},$$

und Kandidaten für den nächsten Elementarschritt sind die Paare $(1,2)$ und $(3,4)$.

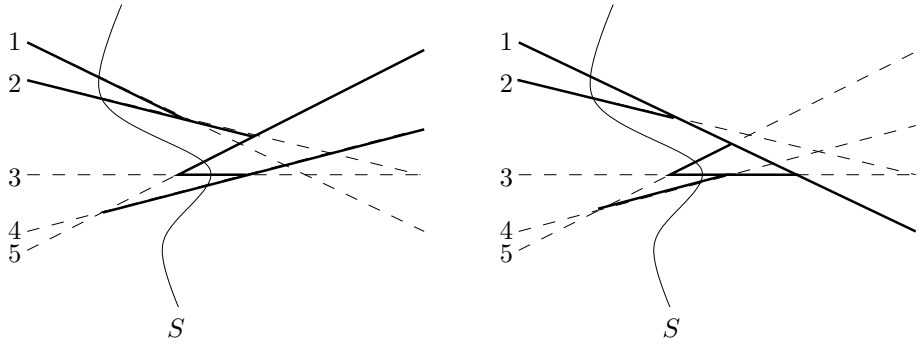
Nach einem Elementarschritt ist die Liste N zu aktualisieren. Dabei helfen uns obere und untere Horizontbäume, die wie folgt definiert sind, vgl. Abbildung 1.11:

Oberer Horizontbaum T^O:

- Markiere jede von S geschnittene Kante.
- Setze Markierung längs der Geraden nach rechts fort.
- Wo sich zwei markierte Kanten treffen, setze diejenige mit größerer Steigung nach rechts fort.

Unterer Horizontbaum T_U:

- Markiere jede von S geschnittene Kante.
- Setze Markierung längs der Geraden nach rechts fort.
- Wo sich zwei markierte Kanten treffen, setze diejenige mit kleinerer Steigung nach rechts fort.



Oberer Horizontbaum T^O Unterer Horizontbaum T_U

Abbildung 1.11: Oberer und unterer Horizontbaum.

Eine Eigenschaft der Horizontbäume ist folgende: Sei e eine von S geschnittene Kante von $\mathcal{A}$, und seien e^O und e_U ihre Verlängerungen in T^O, T_U. Dann ist $e = \min(e^O, e_U)$, d. h. aus T^O und T_U kann schnell bestimmt werden, von welchen Geraden die Kante e begrenzt wird; die kürzere Kante liefert den Schnittpunkt mit der begrenzenden Geraden.

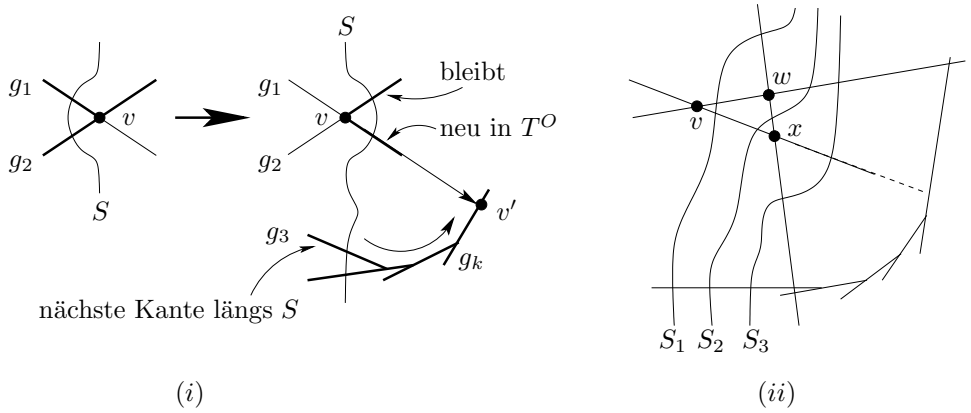


Abbildung 1.12: Aktualisierung der Horizontbäume am Beispiel von T^O.

Abbildung 1.12(i) zeigt die Aktualisierung der Horizontbäume am Beispiel von T^O: Nachdem die Sweepkurve S den Punkt v überschritten hat, wird die Gerade g_1, die bisher nur bis v markiert war, hinter v von S geschnitten. Somit wird das Segment $\overline{vv'}$ der Geraden rechts von v Teil des Horizontbaumes. v' ist dabei der zu v nächstgelegene Schnittpunkt von g_1 mit einer markierten Geraden g_k mit größerer Steigung. Um diese zu finden, suchen wir längs S die nächste Schnittkante g_3 und folgen der konvexen Kantenfolge bis zum Schnittpunkt mit g_1. Die Suche nach einem Schnittpunkt kann dabei $O(n)$ viele Schritte erfordern. Es kann auch vorkommen, daß dieselben Kanten mehrfach abgesucht werden müssen; Abbildung 1.12(ii) zeigt ein Arrangement, in dem bei den Knoten v und x dieselben Kanten durchsucht werden.

Um eine Übersicht über die Kosten zu bekommen, wollen wir die Kosten des Besuchs einer Kante e dem verursachenden Knoten v in Rechnung stellen. Ideal wäre, wenn Bilanzen nur zwischen Knoten und Kanten derselben Zelle im Arrangement aufgestellt werden müßten. Abbildung 1.12(ii) zeigt allerdings, daß dies nicht ohne weiteres möglich ist: die Kanten, die bei der Bearbeitung von v besucht werden, liegen nicht alle in derselben Zelle wie v. Dies wird erst bei der Bearbeitung von x festgestellt, und x liegt in derselben Zelle wie die Kanten, also transferieren wir die Kosten dieser Kanten auf x. Zur Berechnung dieser Kosten brauchen wir Folgendes:

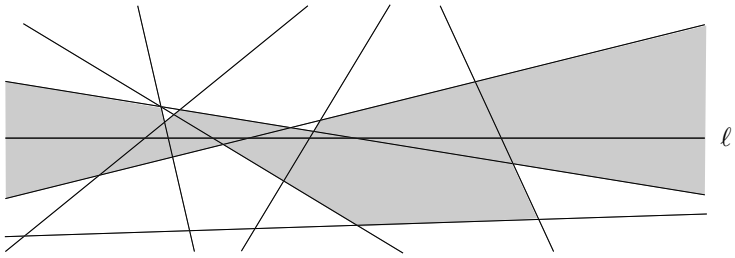


Abbildung 1.13: Zone einer Geraden ℓ in einem Arrangement $\mathcal{A}$.

Definition 1.3 Sei $\mathcal{A}$ ein Arrangement von n nicht senkrechten Geraden, und sei ℓ eine beliebige Gerade. Die **Zone** von ℓ in $\mathcal{A}$ besteht aus allen Zellen, deren Abschluß von ℓ geschnitten wird:

$$\text{Zone}(\ell) := \{ \text{ Zellen } Z \text{ in } \mathcal{A} \mid \ell \cap \overline{Z} \neq \emptyset \}$$

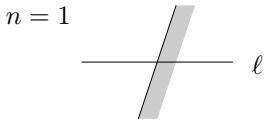
Theorem 1.4 *(Zonentheorem)*
Sei $\mathcal{A}$ ein Arrangement von n nicht senkrechten Geraden und sei ℓ eine beliebige Gerade. Die Zone von ℓ hat die Komplexität $O(n)$.

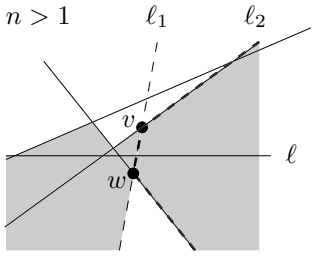
Beweis.
In jeder Zelle gibt es einen rechten, obersten Punkt und einen linken, untersten Punkt. Diese Punkte teilen den Rand einer Zelle eindeutig in einen linken und einen rechten Rand.

Wir zeigen durch Induktion über n, daß es höchstens $5n$ linke Kanten auf den Rändern der Zonenzellen geben kann. Aus Symmetriegründen gibt es auch höchstens $5n$ rechte Kanten.

Für $n = 1$ haben wir eine linke Kante.

Für $n > 1$ sei ℓ_1 die Gerade im Arrangement, deren Schnitt mit ℓ am weitesten rechts liegt (wir nehmen an, diese sei eindeutig).
Sei v der niedrigste Schnitt von ℓ_1 mit einer Geraden aus $\mathcal{A}$ oberhalb von ℓ und w der höchste Schnitt von ℓ_1 mit einer Geraden aus $\mathcal{A}$ unterhalb von ℓ.
Sei $\mathcal{A}'$ das Arrangement ohne ℓ_1. Lt. Induktionsvoraussetzung hat die Zone von ℓ in $\mathcal{A}'$ höchstens $5(n-1)$ linke Kanten.



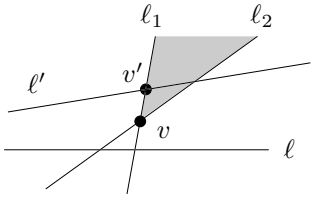


Wieviele linke Kanten werden nun beim Einfügen von ℓ_1 höchstens hinzugefügt?

(i) Die Kante $\overline{vw}$.
(ii) Sei ℓ_2 die Gerade, die ℓ_1 in v schneidet. ℓ_2 hat vor dem Einfügen von ℓ_1 möglicherweise mehrere linke Kanten gestellt. Höchstens eine davon kann durch Einfügen von ℓ_1 in v geteilt werden.
(iii) Entsprechend für w.

Es kommen also höchstens drei linke Kanten hinzu, also hat Zone(ℓ) in $\mathcal{A}$ höchstens $5(n-1)+3 \leq 5n$ linke Kanten.

Zu zeigen ist noch, warum beim Einfügen von ℓ_1 keine weiteren linken Kanten geteilt werden können. Nehmen wir dazu an, die Gerade ℓ' würde ℓ_1 oberhalb von ℓ schneiden
$\Rightarrow \exists$ Schnittpunkt v' oberhalb von v.
$\Rightarrow$ kurz rechts von ℓ_1 kann kein Stück von ℓ' zu Zone(ℓ) gehören, weil ℓ' dort in einer Region liegt, die nach unten



hin von ℓ_1 und ℓ_2 begrenzt wird.
$\Rightarrow$ es kann keine Teilung stattgefunden haben.

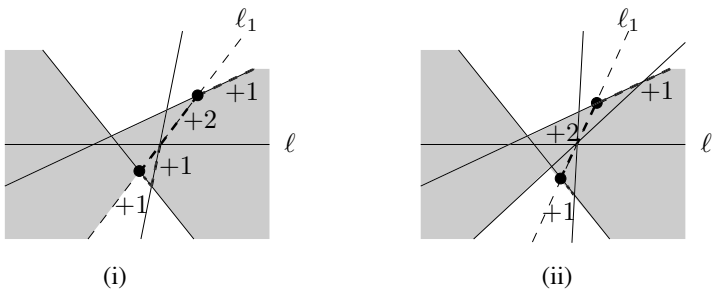


Abbildung 1.14: Im rechtesten Schnittpunkt schneiden sich zwei oder mehrere Geraden. Die gestrichelten Kanten werden neu eingefügt.

Ist die Gerade ℓ_1 nicht eindeutig, d. h. in dem weitesten rechts liegenden Schnittpunkt mit ℓ schneiden sich mehrere Geraden, so wählen wir eine dieser Geraden als ℓ_1. Mit der gleichen Zählweise wie oben stellen wir fest, daß wir für zwei schneidende Geraden 5 neue linke Kanten bekommen können, siehe Abbildung 1.14(i), für drei oder mehrere Geraden 4 linke Kanten (Abbildung 1.14(ii)). □

Wir können die Kosten für die Aktualisierung der Horizontbäume wie folgt abschätzen:

$$
\begin{aligned}
& \text{Kosten der Aktualisierung von } T^O, T_U \\
\approx\ & \text{Summe aller Kosteneinheiten Kante–Knoten aus je einer Zelle} \\
\leq\ & \sum_{\text{Zellen } Z} (\#\text{Kanten von } Z) \cdot (\#\text{Knoten von } Z) \\
=\ & \sum_{\text{Zellen } Z} (\#\text{Kanten von } Z)^2 \\
\leq\ & \sum_{\text{Geraden } G} \underbrace{\sum_{\substack{\text{Zellen } Z \text{ mit} \\ \text{Kante auf } G}} (\#\text{Kanten von } Z)}_{O(n) \text{ nach Theorem 1.4}} \\
\in\ & O(n^2)
\end{aligned}
$$

Bemerkung: Die Kosten der Aktualisierung der Horizontbäume, nachdem die Sweep Pseudogerade über einen Schnittpunkt hinweggehoben wurde, entstehen durch entlang laufen des Randes der aktuellen Zelle z. Der Rand einer Zelle z kann also insgesamt maximale Anzahl Knoten von z mal durchlaufen werden. Die Gesamtkosten ergeben sich durch Summation über alle Zellen z.

Damit haben wir folgende Laufzeiten gezeigt:

Aktualisierung der Horizontbäume	$O(n^2)$
$\Rightarrow$ Topologischer Sweep	$O(n^2)$
$\Rightarrow$ Wahl der Bearbeitungsreihenfolge	$O(n^2)$
$\Rightarrow$ Konstruktion des Sichtbarkeitsgraphen	$O(n^2)$

Also gilt:

Theorem 1.5 *(Welzl, 1985)*
Der Sichtbarkeitsgraph von n Liniensegmenten kann in Zeit $O(n^2)$ und in Speicherplatz $O(n)$ konstruiert werden. *[Wel85]*

Dabei werden sichtbare Paare von Endpunkten nur berichtet, nicht gespeichert. Außerdem setzen wir voraus, daß sich die Punkte in allgemeiner Lage befinden, insbesondere keine drei Punkte kollinear sind.[5]

Den Sichtbarkeitsgraphen in Algorithmus 1.1 können wir also in Zeit $O(n^2)$ konstruieren, auch das Hinzufügen der Kanten zu Start- und Zielpunkt ist in $O(n^2)$ möglich. Damit bleibt aus Algorithmus 1.1 das Problem der Suche nach einem kürzesten Weg in ungerichteten Graphen mit nicht-negativen Kantengewichten; dies ist ein diskretes Problem.

Dieses Problem läßt sich mit dem Algorithmus von Dijkstra (Algorithmus 1.3, [Dij59]) lösen. Dieser verwaltet eine Welle W mit einem Ausläufer A um a und aktualisiert in jedem Schritt die Knotenmarkierung

$$\mathrm{d}(p) = \begin{cases} \text{min. Länge eines Pfads von } a \text{ nach } p & \text{für } p \in W \\ \text{min. Länge eines Pfads von } a \text{ nach } p, \text{ der bis auf } p \text{ in } W \text{ verläuft} & \text{für } p \in A \\ \infty & \text{sonst} \end{cases}$$

Algorithmus 1.3 Kürzeste Pfade in gewichtetem Graph (Dijkstra, 1959)

Gegeben: Graph $G = (V, E)$ zusammenhängend, $|V| = n$, $|E| = m$
Kantengewichtung $g : E \longrightarrow \mathbb{R}^{\geq 0}$, Knoten $a, b \in V$.

Gesucht: Kantenfolge von a nach b mit minimalem Gesamtgewicht.

Solange $A \neq \emptyset$:

- Entnehme A ein p mit minimalem $\mathrm{d}(p)$.
- $W := W \cup \{p\}$.
- Für alle direkten Nachbarn q von p in W^C:
  - $\mathrm{d}(q) := \min\left\{ \mathrm{d}(q),\ \mathrm{d}(p) + g(p,q) \right\}$
  - Wenn $q \notin A$ dann $A := A \cup \{q\}$.

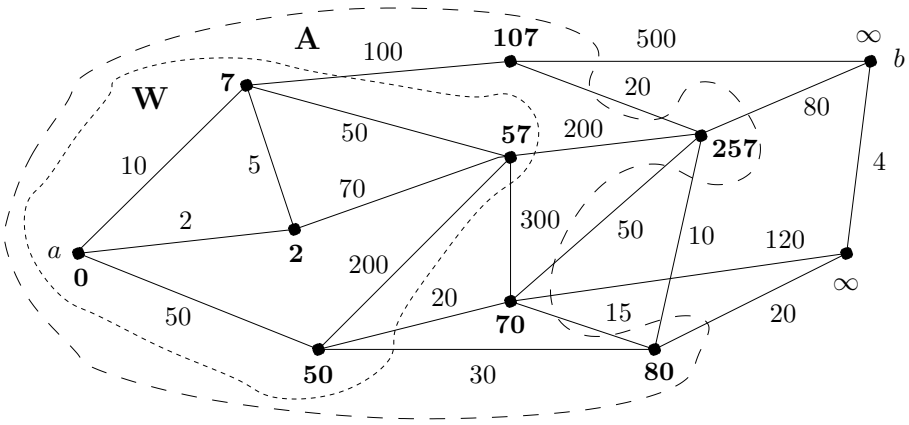


Abbildung 1.15: Welle, Ausläufer und Knotenmarkierungen während des Algorithmus von Dijkstra.

[5] Ansonsten wären noch einige Spezialfälle zu beachten, die aber an Laufzeit und Speicherplatz nichts ändern würden.

Abbildung 1.15 zeigt einen Zwischenstand bei der Berechnung des kürzesten Weges mit der aktuellen Welle W, dem aktuellen Ausläufer A und den aktuellen Knotenmarkierungen (fett).

Zur Implementierung des Algorithmus von Dijkstra wird eine Prioritätswarteschlange für den Ausläufer A benötigt. Diese soll eine Menge von Paaren $(p, \mathrm{d}(p))$ verwalten. Die Operationen Einfügen eines Elementes, Löschen des Minimums werden insgesamt $O(n)$ mal durchgeführt. Vermindern des Wertes $\mathrm{d}(p)$ kann insgesamt $O(m)$ mal durchgeführt werden. Die Elemente befinden sich bei diesen Operationen in bekannter Position, d. h. es kann ohne vorherige Suche darauf zugegriffen werden. Zur Realisierung dieser Warteschlange gibt es folgende Möglichkeiten:

Führen wir i mal einfügen, j mal vermindern und k mal löschen des Minimums in beliebiger Reihenfolge aus.

- Lineare Liste, unsortiert: Einfügen, Vermindern in $O(1)$, Löschen–Min in $O(i)$. Gesamtlaufzeit in $O(i^2 + j)$
- Fibonacci Heap (Fredman, Tarjan, 1987; beschrieben u.a. im Buch von Ottmann und Widmayer, [OW93]): Einfügen, Vermindern in $O(1)$ und Löschen–Min $O(\log i)$. Gesamtlaufzeit in $O(k \log i + i + j)$. [FT87]
- Relaxed Heap (Driscoll, Gabow, Shrairaman, Tarjan, 1988): Einfügen, Vermindern in $O(1)$, Löschen–Min in $O(\log i)$.
  Gesamtlaufzeit in $O(k \log i + j)$ [DGST88]

Wird der Algorithmus von Dijkstra mit Fibonacci–Heaps implementiert, kann man in einem Graph G mit n Knoten und m Kanten die kürzesten Wege von einem Knoten a zu allen übrigen Knoten in Zeit $O(n \log n + m)$ bestimmen.

Damit ist folgendes gezeigt:

Theorem 1.6
Gegeben seien h Polygone mit n Ecken, ein Startpunkt s und ein Zielpunkt t. Ein kürzester Weg von s nach t kann in Zeit $O(n^2)$ berechnet werden, die kürzesten Wege von s zu allen Ecken ebenfalls in Zeit $O(n^2)$.

Der Engpaß bei der bisherigen Berechnung kürzester Wege ist die Konstruktion des Sichtbarkeitsgraphen. Wie bereits erwähnt, kann dieser $\Omega(n^2)$ viele Kanten enthalten, jedoch auch viel weniger, vgl. Abbildung 1.4. Beim obigen Algorithmus brauchen wir stets eine Laufzeit von $\Omega(n^2)$ wegen der Konstruktion des Arrangements der dualen Geraden. Die Frage ist nun, ob sich der Sichtbarkeitsgraph auch output–sensitiv bzgl. seiner Größe berechnen läßt. Tatsächlich ist dies möglich: Zu einem Arrangement aus n Liniensegmenten, dessen Sichtbarkeitsgraph aus m Kanten besteht, sind folgende output–sensitiven Algorithmen bekannt:
Overmars, Welzl, 1988: Zeit $O(m \log n)$ und Platz $O(n)$ [OW88],
Ghosh, Mount, 1987: Zeit $O(n \log n + m)$ und Platz $O(m)$ [GM87, GM91],
Pocchiola, Vegter, 1995: Zeit $O(n \log n + m)$ und Platz $O(n)$ [PV95].

Damit gilt:

Theorem 1.7 *Mit der Fibonacci–Heap Implementierung des Algorithmus von Dijkstra und der output–sensitiven Berechnung des Sichtbarkeitsgraphen lassen sich kürzeste Wege in Zeit $O(n \log n + m)$ berechnen.*

Bei der Laufzeit ist jedoch der Anteil von $O(n \log n)$ unvermeidlich:

Theorem 1.8 *(untere Schranke) In einer Umgebung von Polygonen mit insgesamt n Ecken braucht die Berechnung eines kürzesten Pfades Zeit*
$\Omega(n \log n)$.

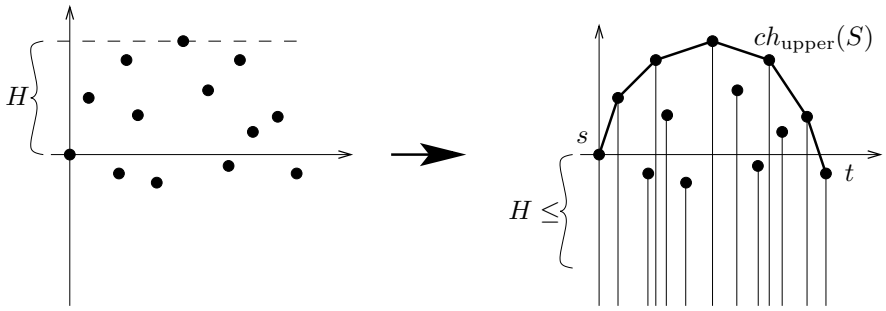


Abbildung 1.16: Reduktion: Konstruktion der konvexen Hülle auf Bestimmung des kürzesten Pfades.

Beweis.
Wir zeigen, daß sich die Konstruktion der konvexen Hülle auf die Berechnung kürzester Pfade reduzieren läßt. Da sich Sortieren auf die Konstruktion der konvexen Hülle reduzieren läßt, erhalten wir eine Laufzeit von $\Omega(n \log n)$.

Sei eine Menge von Punkten gegeben. Sei s der linkeste Punkt, t der rechteste Punkt und H die Höhe der vertikale Abstand zwischen niedrigsten und höchsten Punkt. Lege an jeden Punkt ein senkrecht nach unten zeigendes Liniensegment einer Länge $> 2H$. Dann entspricht der kürzeste Weg von s nach t der oberen konvexen Hülle, siehe Abbildung 1.16. □

Die nächste Frage ist, ob man wirklich den Sichtbarkeitsgraphen konstruieren muß. In der Tat ist dies nicht nötig, folgender Ansatz ist als Locus Approach bekannt und sei hier nur kurz umrissen:

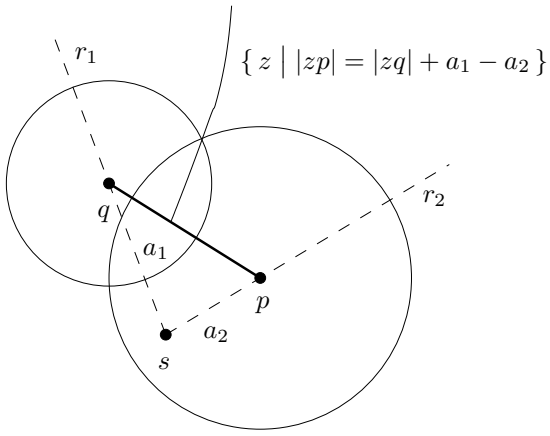


Abbildung 1.17: Berechnung der Shortest Path Map.

Wir zerlegen den freien Teil der Ebene in Regionen, so daß alle Punkte einer Region kombinatorisch gleiche[6] Kürzeste zum Startpunkt s haben (Shortest Path Map). In Abbildung 1.17 sind genau für die Punkte auf der Hyperbel $H = \{ z \mid |zp| = |zq| + a_1 - a_2 \}$ die Wege von s über p oder über q gleich lang. Dies entspricht einem Voronoi–Diagramm mit additiven Gewichten, siehe Abschnitt A.3.4. Die Shortest Path Map läßt sich mit der Continuous Dijkstra Methode berechnen: Wir lassen um s einen Kreis wachsen. Wenn dieser eine Hindernisecke berührt, entsteht ein neuer Kreis, der mit der gleichen Geschwindigkeit wie die bisherigen Kreise um die Hindernisecke wächst (Wellenfront). Es wird festgehalten, von welchem Kreis ein Punkt der Ebene zuerst erreicht wird. Auf die Continuous Dijkstra Methode werden wir in Abschnitt 1.4 noch zurückkommen, wir halten hier nur fest:

[6] D. h. sie berühren dieselben Ecken in derselben Reihenfolge.

Theorem 1.9 *(Hershberger, Suri, 1997)*
Gegeben seien polygonale Hindernisse mit insgesamt n Ecken und ein fester Startpunkt s. Dann läßt sich in Zeit $O(n \log n)$ und Platz $O(n \log n)$ eine Karte der kürzesten Wege berechnen. Diese braucht $O(n)$ Platz und ermöglicht für einen beliebigen Zielpunkt t

(i) in Zeit $O(\log n)$ die Entfernung von s zu bestimmen und

(ii) in Zeit $O(\log n + k)$ den kürzesten Weg von s nach t zu berichten, wobei k die Anzahl der Kanten auf dem Weg ist. *[HS99]*

1.2 Kürzeste Pfade im Inneren eines einfachen Polygons

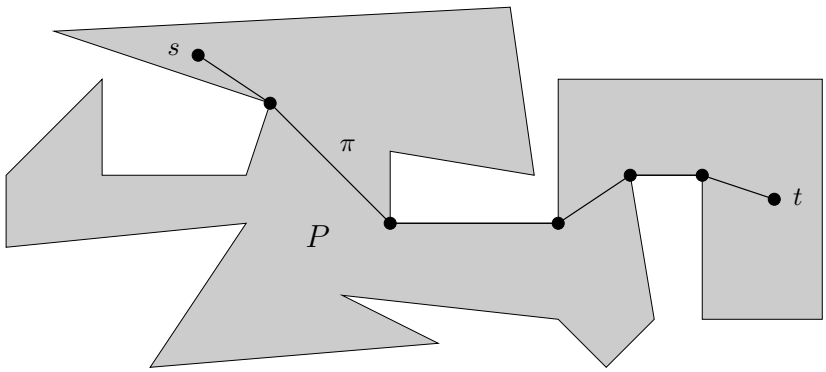


Abbildung 1.18: Kürzester Pfad im Inneren eines einfachen Polygons.

Betrachten wir nun das Problem, zu einem gegebenem einfachen Polygon mit n Ecken und zwei beliebigen Punkten s und t im Inneren des Polygons den kürzesten Pfad π zu bestimmen, der im Inneren des Polygons verläuft. Im Gegensatz zu dem Problem in Abschnitt 1.1 ist der kürzeste Pfad hier eindeutig bestimmt. Man kann sich den Pfad auch in diesem Fall als straff gespanntes Gummiband zwischen s und t vorstellen, das wie in Abschnitt 1.1 nur an den Ecken des Polygons abknicken kann.

1.2.1 Der Algorithmus von Lee und Preparata

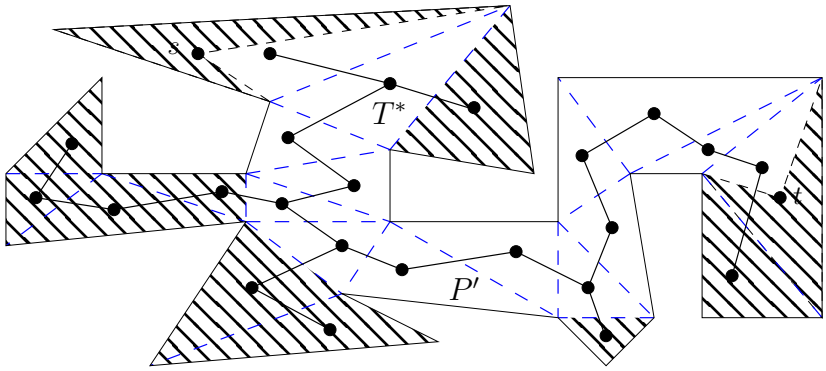


Abbildung 1.19: Triangulation T und dualer Graph T^*.

Wie in Abschnitt 1.1 ist unser Ansatz auch hier, das gegebene Problem in ein diskretes Problem zu verwandeln; dabei hilft uns die aus der Algorithmischen Geometrie bekannte Triangulation des Polygons:

Definition 1.10 Zu einem einfachen Polygon P ist die **Triangulation** T eine maximale Menge von kreuzungsfreien Liniensegmenten im Innern von P, deren Endpunkte auf den Ecken von P liegen. Der zur Triangulation T duale Graph T^* enthält die Dreiecke der Triangulation als Knoten und Kanten zwischen den Knoten von Dreiecken mit einer gemeinsamen Seite.

Aus der Algorithmischen Geometrie [Kle97] kennen wir ein randomisiertes Verfahren zur Berechnung der Triangulation in Zeit $O(n \log n)$, ein lineares Verfahren wurde 1991 von Chazelle vorgestellt [Cha91]; Abbildung 1.19 zeigt die Triangulation und den dualen Graphen zu dem Beispiel aus Abbildung 1.18.

Der duale Graph T^* ist ein Baum mit Knotengrad ≤ 3, also entspricht dem kürzesten Pfad π von s nach t ein eindeutiger Pfad π_{T^*} im Baum T^* von dem Dreieck, das s enthält, zu dem

Dreieck, das t enthält. Nach Lokalisation der Dreiecke, die s bzw. t enthalten, läßt sich π_{T^*} durch Tiefensuche in T^* in linearer Zeit bestimmen.

Betrachten wir das Teilpolygon P', das die Dreiecke längs π_{T^*} sowie zwei angesetzte Dreiecke mit Ecken s bzw. t enthält und sich in Zeit $O(n)$ bestimmen läßt, so muß der kürzeste Pfad π in P' verlaufen. Der nicht zu P' gehörende Teil ist in Abbildung 1.19 schraffiert dargestellt.

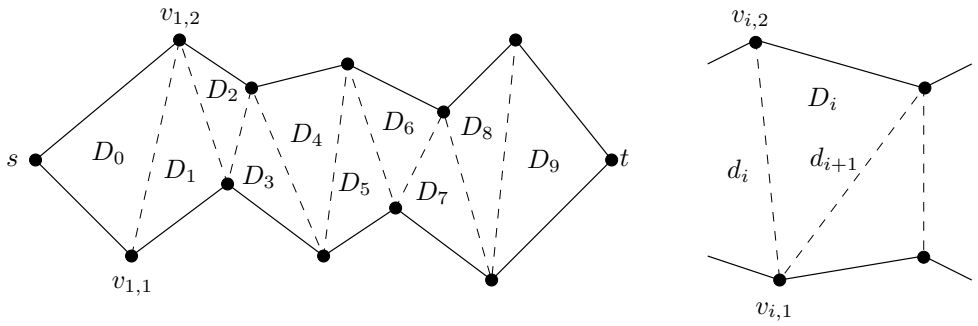


Abbildung 1.20: Kette von Dreiecken.

Damit reduziert sich das anfängliche Problem auf die Bestimmung des kürzesten Pfades von s nach t in P'. Dieses Polygon hat jedoch eine spezielle Eigenschaft: es besteht aus einer Kette von Dreiecken, bei der zwei konsekutive Diagonalen d_i, d_{i+1} immer eine gemeinsame Ecke haben, vgl. Abbildung 1.20. Das erlaubt uns, induktiv die Kürzesten $\pi(s, v_{i,1})$ und $\pi(s, v_{i,2})$, $1 \leq i \leq n$ von s zu den Endpunkten $v_{i,1}$ und $v_{i,2}$ der Diagonalen d_i zu bestimmen: der Fall $i = 1$ ist trivial, der kürzeste Weg besteht aus den beiden Kanten des Polygons mit Ecke s. Nehmen wir also per Induktion an, der kürzeste Pfad von s zu $v_{i,1}$ und $v_{i,2}$ sei bekannt. Die Gestalt der Pfade $\pi(s, v_{i,1})$ und $\pi(s, v_{i,2})$ ist in Abbildung 1.21 dargestellt: nach einem eventuellen gemeinsamen Teil vom Startpunkt s bilden sie einen Trichter mit der Diagonalen d_i als Oberseite.

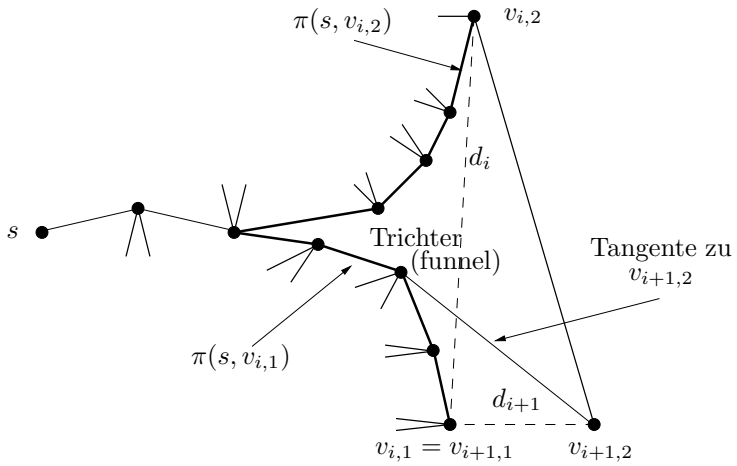


Abbildung 1.21: Trichter.

Sei o. B. d. A. $v_{i+1,1} = v_{i,1}$. Zu berechnen ist also der kürzeste Weg von s nach $v_{i+1,2}$. Dazu starte in $v_{i,2}$ und suche die Wand des alten Trichters gegen den Uhrzeigersinn nach einem Tangentenpunkt von $v_{i+1,2}$ ab. Diese Tangente gehört zum kürzesten Weg von s nach $v_{i+1,2}$. Dabei durchlaufene Kanten fallen entweder weg oder gehören zu einem gemeinsamen Stück des neuen Trichters und werden nie wieder besucht, so daß sich eine Laufzeit von $O(n)$ für die Bestimmung des kürzesten Weges von s nach t in P' ergibt. Damit haben wir folgendes Theorem gezeigt, vgl. Algorithmus 1.4:

Theorem 1.11 *(Lee, Preparata, 1984)*
Der kürzeste Pfad zwischen zwei Punkten in einem einfachen Polygon kann in Zeit $O(n)$ und Platz $O(n)$ bestimmt werden. *[LP84]*

Algorithmus 1.4 Kürzester Pfad in einem Polgyon (Lee, Preparata)

- Berechne Triangulation T und dualen Graphen T^* des Polygons P. $O(n)$
- Bestimme die Dreiecke D_s und D_t die s bzw. t enthalten. $O(n)$
- Bestimme das Teilpolygon P', das nur Dreiecke längs des Pfades von D_s nach D_t in T^* enthält (Tiefensuche). $O(n)$
- Bestimme induktiv die Kürzesten $\pi(s, v_{i,1})$ und $\pi(s, v_{i,2})$, $1 \leq i \leq n$ von s zu den Endpunkten $v_{i,1}$ und $v_{i,2}$ der Diagonale d_i. $O(n)$

1.2.2 Der Algorithmus von Guibas und Hershberger

Das Ergebnis von Lee und Preparata läßt sich verbessern; wir benötigen dazu folgendes Theorem:

Theorem 1.12 *(Cutting–Theorem, Chazelle, 1982)*
Sei T eine Triangulation eines einfachen Polygons P, dann existiert eine Diagonale $d \in T$, so daß auf jeder Seite von d mindestens $\frac{n}{3} - 1$ Dreiecke liegen. *[Cha82]*

Beweis. Übungsaufgabe. Hinweis: Man nutze die Tatsache aus, daß der duale Graph von T ein Baum ist. □

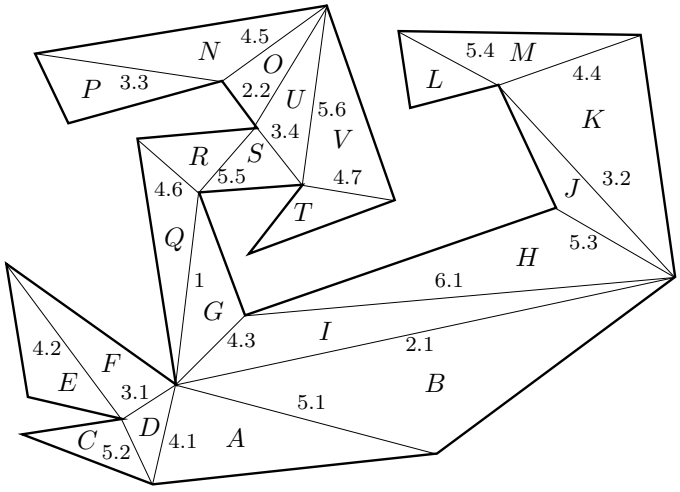


Abbildung 1.22: Balancierte hierarchische Triangulation.

Durch iterierte Anwendung von Theorem 1.12 auf die Teilpolygone zu beiden Seiten von d entsteht eine balancierte, hierarchische Zerlegung von P in Dreiecke, vgl. Abbildung 1.22. Diese Zerlegung läßt sich in einem Binärbaum $\hat{T}$ darstellen, dessen Knoten den Diagonalen in T entsprechen, vgl. Abbildung 1.23. Die Wurzel von $\hat{T}$ entspricht also der Diagonalen, die P in P_1 und P_2 unterteilt, die Kinder der Wurzel sind die Diagonalen der Zerlegungen von P_1 und P_2 etc. Bezeichnet $P(d)$ das Subpolygon, das von der Diagonalen d unterteilt wird, so entspricht jedem Knoten in $\hat{T}$ ein Subpolygon $P(d)$ von P; die Wurzel von $\hat{T}$ entspricht P, die Blätter von

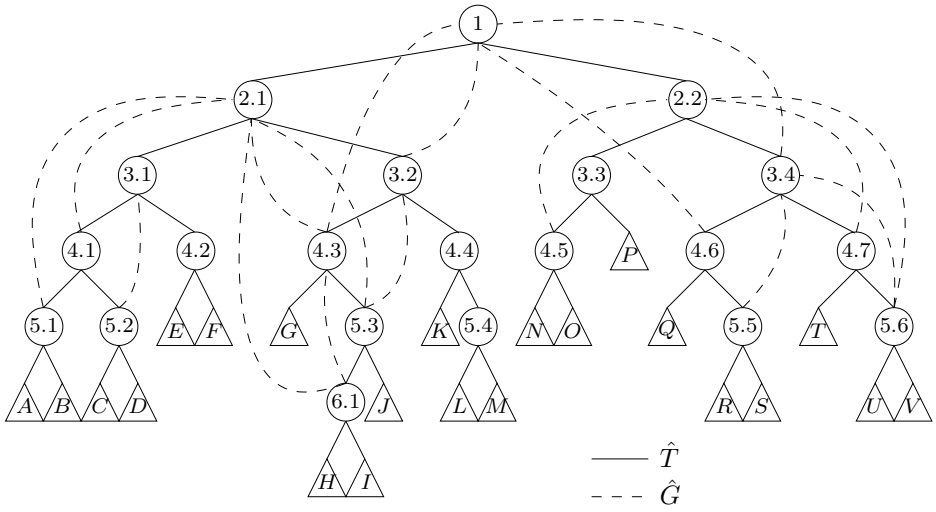


Abbildung 1.23: Baum der balancierten hierarchischen Triangulation.

$\hat{T}$ den Dreiecken der Triangulation von P. Diese Subpolygone werden von Kanten von P und Diagonalen berandet.

Zur Illustration des Zusammenhangs zwischen den Pfaden in der Triangulation T und den Pfaden im Baum $\hat{T}$ betrachten wir den Pfad von Dreieck P nach Dreieck M in Abbildung 1.22 und vergleichen die in T überquerten Diagonalen mit den in $\hat{T}$ durchlaufenen Knoten:

T	3.3	4.5	2.2	3.4	5.5	4.6	1	4.3	6.1	5.3	3.2	4.4	
$\hat{T}$	3.3		2.2				1		*2.1*		3.2	4.4	*5.4*

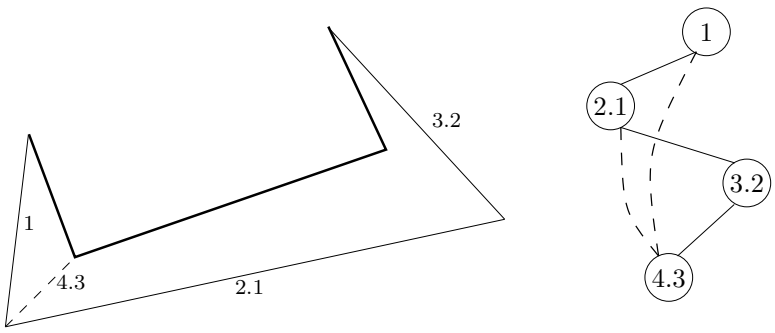
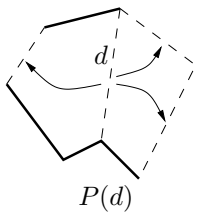


Abbildung 1.24: Subpolygon $P(d_{4.3})$ und Abkürzungen in $\hat{G}$.



Zu beobachten ist, daß der Pfad in $\hat{T}$ im allgemeinen kürzer ist — da der Baum balanciert ist, ist die Länge des Pfades $O(\log n)$ —, jedoch enthält er Knoten, die Diagonalen entsprechen, die vom Pfad im Polygon nicht überquert werden, wie die Knoten 2.1 und 5.4 im obigen Beispiel. 5.4 ist dabei nur ein "falsches" Endstück, Probleme bereitet 2.1. Abhilfe schafft die Bereitstellung von Abkürzungen in $\hat{T}$: kürzeste Pfade, die ein Subpolygon $P(d)$ passieren, passieren ein Paar von Diagonalen d_1, d_2, die auf dem Rand von $P(d)$ liegen. Um diese Paare systematisch zu erfassen, werden beim Einfügen der Diagonalen d in $\hat{T}$ zusätzliche Zeiger von d auf alle älteren Diagonalen, die das von d geteilte Polygon $P(d)$ beranden, hinzugefügt, siehe obige Abbildung. Wird im Beispiel von oben der Knoten 4.3 eingefügt, so werden die zusätzlichen Zeiger von 4.3 auf die Knoten 1 und 2.1 aufgenommen; mit dem Knoten 3.2 ist 4.3 schon in $\hat{T}$ verbunden, siehe Abbildung 1.24. So

entsteht aus dem Hierarchiebaum[7] $\hat{T}$ der Triangulation T der Schichtengraph $\hat{G}$, mit folgenden Eigenschaften:

Lemma 1.13 *Eigenschaften von $\hat{G}$:*
(i) Der Weg zwischen zwei beliebigen Dreiecken in T ergibt sich durch Konkatenation von $O(\log n)$ vielen Kanten im Graphen $\hat{G}$.

(ii) Diese Kanten lassen sich in Zeit $O(\log n)$ in $\hat{G}$ bestimmen.

(iii) Die Größe von $\hat{G}$ ist $O(n)$.

Beweis.
(i) Sei π^* der Weg im dualen Graphen T^*. Betrachte die von π^* überquerten Diagonalen der Triangulation und ihre Tiefe (Schicht) in $\hat{T}$:

Wir konstruieren einen Pfad $\pi' = d_1, d_2, \ldots, d_l, d_{l+1}, \ldots, d_{\text{first}}, \ldots, d_k$ aus π^*. π' ensteht aus π^* durch Streichen von Diagonalen. Es soll gelten, dass zwischen zwei Diagonalen d_l und d_{l+1} in π^* keine Diagonale exestiert mit Tiefe kleiner als d_l. Jedoch soll d_{l+1} eine kleinere Tiefe haben als d_l. Bis d_{first} wird so die Tiefe immer geringer und ab d_{first} entsprechend wieder größer. π' entsteht aus einem Durchlauf von d_1 nach d_{first}, bei dem die minimale Tiefe festgehalten wird und alle Diagonalen verworfen werden, die tiefer liegen als das bisherige Minimum, entsprechend für den Weg von d_k nach d_{first}.

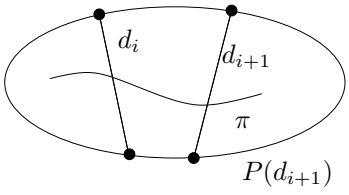
$$\pi^* = d_1 \ \ldots \xrightarrow[\text{tiefer als}]{} d_2 \ \ldots \xrightarrow[\text{tiefer als}]{} d_\ell \ \ldots \xrightarrow[\text{tiefer als}]{} d_{\ell+1} \ \ldots \xrightarrow[\text{tiefer als}]{} \underbrace{d_{\text{first}}}_{\substack{\text{kleinste}\\ \text{Tiefe}}} \ \ldots \ d_{l+2} \ldots \xleftarrow[\text{tiefer als}]{} d_k$$

Im Beispiel ergibt sich also für den Weg von Dreieck P nach Dreieck M der Pfad $\pi' = 3.3$ 2.2 1 3.2 4.4.

π' besteht bis d_{first} aus Kanten mit strikt abnehmender Tiefe, ab d_{first} aus Kanten mit strikt zunehmender Tiefe, also können nur $O(\log n)$ viele Kanten in π' liegen.

Beh.: Im Baum $\hat{T}$ ist d_{i+1} Vorfahre von d_i, und im Graphen $\hat{G}$ sind d_i, d_{i+1} mit einer Kante verbunden.

Bew.:
Unmittelbar nach dem Einfügen von d_i in die hierarchische Triangulation sind d_i und d_{i+1} längs π' direkt benachbart, denn die Diagonalen auf π' zwischen d_i und d_{i+1} sind noch nicht eingefügt
$\Rightarrow d_i \subseteq P(d_{i+1})$
$\Rightarrow$ in $\hat{T}$ ist d_i im Teilbaum mit Wurzel d_{i+1} enthalten, d. h. d_{i+1} ist Vorgänger von d_i.



Außerdem ist $d_{i+1} \subset \partial P(d_i)$, also sind d_i, d_{i+1} in $\hat{G}$ verbunden. $\diamond$

(ii) Seien d_1 und d_m die erste bzw. letzte Diagonale auf dem Weg vom Start zum Ziel in T. Sei d_{first} der tiefste gemeinsame Vorfahre in $\hat{T}$ von d_1 und d_m. Dieser läßt sich in Zeit $O(\log n)$ bestimmen, da $\hat{T}$ balanciert ist.

Laufe in $\hat{T}$ von d_1 aufwärts nach d_{first} und teste für jeden Knoten mit Diagonale d auf dem Weg in $\hat{T}$, ob d in der Triangulation T zwischen d_1 und d_{first} liegt. Falls ja, nimm die letzte Kante zu d in $\hat{G}$ in den Weg auf (d. h. die Kante vom aktuellen Knoten zu dem Knoten, in dem zuletzt eine Kante aufgenommen wurde). Verfahre ebenso für d_m bis d_{first} rückwärts.

Um diesen Test durchführen zu können, betrachte die Kanten des dualen Graphen T^* und zeichne einen inneren Knoten als Wurzel aus.[8] Offenbar liegt die Diagonale d genau dann auf

[7] Nicht zu verwechseln mit dem dualen Graph T^*!

[8] Zur Erinnerung: In T^* entsprechen die Kanten den Diagonalen der Triangulation; im Gegensatz zu $\hat{T}$, wo die Knoten den Diagonalen entsprechen!

dem Pfad von d_1 nach d_{first}, wenn die Kante d in T^* mit Wurzel w Vorgänger von *genau einer* der beiden Kanten d_1 oder d_{first} ist. Vergleiche hierzu $d \in \{j, k, l\}$ in Abbildung 1.25. Die Diagonale j ist weder Vorgänger von d_1 noch von d_{first}. Die Diagonale k ist Vorgänger von d_1 und l ist Vorgänger von beiden Diagonalen d_1, d_{first}.

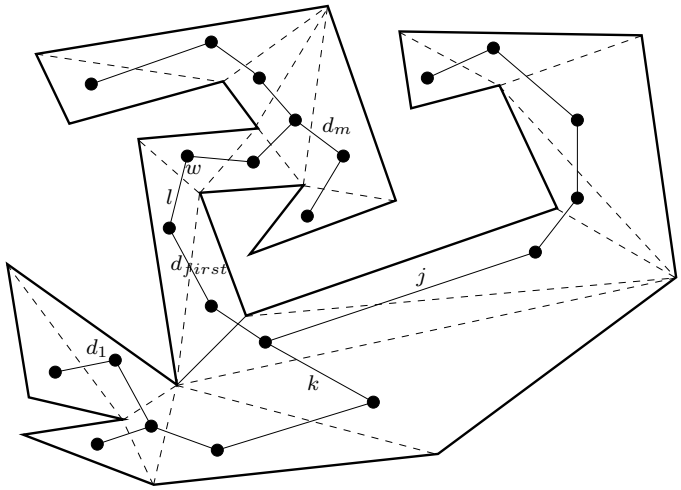


Abbildung 1.25: Diagonalen j, k, l, d_1, d_{first} im dualen Baum T^* mit Wurzel w.

Identifiziere nun jede Kante d mit ihrem "unteren", d. h. von der Wurzel w weiter entfernten Knoten $\delta(d)$.

$$\begin{array}{lrcl} \delta: & \text{Kanten} & \underset{\text{bijektiv}}{\longrightarrow} & \text{Knoten } \setminus \{ w \} \\ \delta: & \text{Kante } d \text{ von } T^* & \longmapsto & \text{Knoten } \delta(d) \text{ der weiter von w entfernt ist.} \end{array}$$

Es gilt:

$$\forall \text{ Kanten } d, d' : d \text{ ist Vorgänger von } d' \iff \delta(d) \text{ ist Vorgänger von } \delta(d').$$

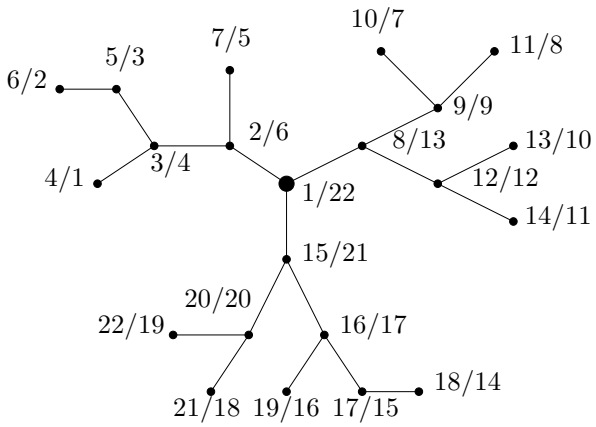


Abbildung 1.26: Prä– und Postorderdurchlauf.

Damit reduziert sich das Problem auf folgende Frage: Wie entscheidet man schnell für zwei Knoten a, b eines Baumes T^* mit Wurzel w, ob a Vorgänger von b ist. Dies kann wie in Algorithmus 1.5 beschrieben nach einem Preprocessing in Zeit $O(n)$ für jedes Knotenpaar in Zeit $O(1)$ entschieden werden.

(iii) Beim Einfügen berandet jede Diagonale zu jedem Zeitpunkt genau zwei Teilpolygone. Daraus folgt, daß von jedem Knoten in $\hat{G}$ maximal zwei Kanten zu jeder tieferen Schicht führen. Weiter

Algorithmus 1.5 Schneller Test auf Vorgängerrelation im Baum

Preprocessing:

Durchlaufe T^* einmal in Präorder und einmal in Postorder und beschrifte jeden Knoten a mit seinen Werten $\text{pre}(a)$ und $\text{post}(a)$.

$O(n)$

Nun gilt:

$$a \text{ ist Vorgänger von } b \Longleftrightarrow \text{pre}(a) < \text{pre}(b) \wedge \text{post}(a) > \text{post}(b)$$

ist die Anzahl der Schichten unterhalb von d gleich der Höhe h von d in $\hat{T}$. Der Teilbaum von d in $\hat{T}$ enthält wegen der Balancierung mindestens $\left(\frac{3}{2}\right)^h$ Blätter (Übungsaufgabe). Da die Teilbäume disjunkt sind, kann es höchstens $\frac{n}{\left(\frac{3}{2}\right)^h} = \left(\frac{2}{3}\right)^h n$ solcher Knoten der Höhe h geben. Also gilt:

$$|\hat{G}| \leq 2 \sum_{d} \text{ Höhe von } d \leq 2 \sum_{h=1}^{\log_{\frac{3}{2}} n} h \left(\frac{2}{3}\right)^h n \in O(n),$$

da die Reihe $\sum_h hX^h$ für $X < 1$ den Konvergenzradius 1 hat. □

Wir wissen nun, daß jeder Pfad in P zwischen zwei Punkten nur eine logarithmische Anzahl von Subpolygonen passiert. Anstatt Wege der Länge $O(n)$ im dualen Graphen der Triangulation zu suchen, können wir also Wege der Länge $O(\log n)$ in $\hat{G}$ betrachten und daraus mit Hilfe vorab berechneter Informationen den kürzesten Weg ableiten.

Dazu benutzen wir "Sanduhren", eine Verallgemeinerung der Trichter (Abbildung 1.21) ohne festen Startpunkt, als Struktur zur Repräsentation aller kürzesten Wege zwischen zwei Diagonalen. Mit jeder Kante in $\hat{G}$ ist eine Sanduhr verbunden. Durch Konkatenation der Sanduhren aller Diagonalen entlang eines Weges π' in $\hat{G}$ erhält man eine Repräsentation aller kürzesten Wege zwischen den gewünschten Diagonalen.

Eine Sanduhr zwischen zwei Diagonalen $d_1 = \overline{ab}$ und $d_2 = \overline{cd}$ — a, c, d, b seien Eckpunkte von P im Uhrzeigersinn — wird von den beiden Diagonalen und den kürzesten Pfaden $\pi(a, c)$ und $\pi(b, d)$ berandet, vgl. Abbildung 1.27. Wir nennen eine Sanduhr *geschlossen*, falls $\pi(a, c)$ und $\pi(b, d)$ gemeinsame Segmente haben. In diesem Fall besteht die Sanduhr aus zwei Trichtern mit einer polygonalen Kette zwischen den Trichteröffnungen e und f. Ansonsten nennen wir die Sanduhr *offen*. Offene Sanduhren werden von zwei konvexen, polygonalen Ketten berandet, und die Diagonalen d_1 und d_2 sind gegenseitig sichtbar, d. h. es existiert ein Liniensegment $\overline{gh}$ mit $g \in d_1$ und $h \in d_2$.

Über die Komplexität der Sanduhren gibt folgendes Lemma Auskunft:

Lemma 1.14 *Die Gesamtkomplexität aller Sanduhren liegt in $O(n)$.*

Beweis. Die Komplexität einer Sanduhr $S(d_i, d_j)$ hängt von der Höhe h der in $\hat{T}$ tieferen Diagonalen ab. Sei d_j die tiefere Diagonale, dann läßt sich die Komplexität der Sanduhr $S(d_i, d_j)$ durch die Anzahl der Knoten unterhalb d_j abschätzen, also durch (Höhe von d_j)^2. Da wir beim Einfügen der Diagonalen d_j den Knoten d_j in $\hat{G}$ mit den benachbarten Diagonalen verbinden, gibt es 2 Sanduhren, in denen d_j der tiefere Knoten ist. Analog zu Lemma 1.13(iii) gibt es $\left(\frac{2}{3}\right)^h n$ Knoten der Höhe h. Insgesamt erhalten wir:

$$2 \sum_{\text{Diagonale } d} (\text{Höhe von } d)^2 \quad \leq \quad 2 \sum_{h=1}^{\log_{\frac{3}{2}} n} (\#\text{Knoten der Höhe } h) \cdot h^2$$

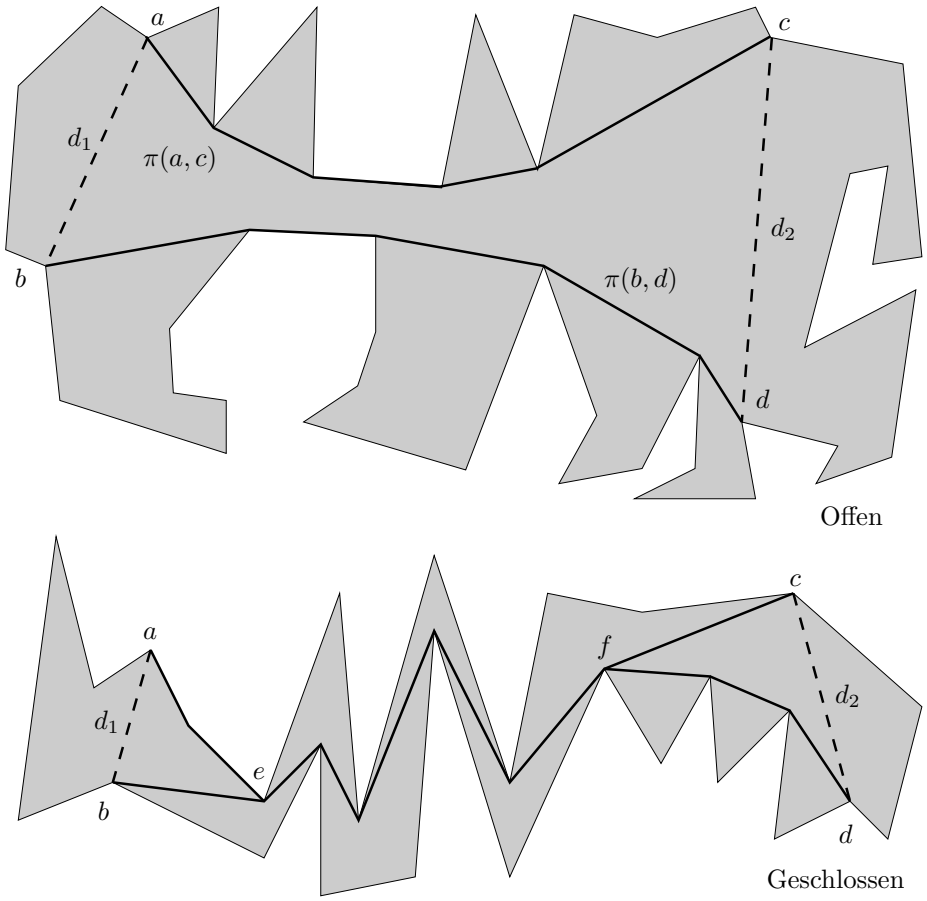


Abbildung 1.27: Offene und geschlossene Sanduhr zwischen $d_1 = \overline{ab}$ und $d_2 = \overline{cd}$.

$$\leq \quad 2n \sum_{h=1}^{\log_{\frac{3}{2}} n} \left(\frac{2}{3}\right)^h \cdot h^2 = 2nC \in O(n)$$

□

Für den kürzesten Pfad zwischen zwei gegebenen Anfragepunkten $p \in d_1$ und $q \in d_2$ sind folgende Fälle zu betrachten:

- Im geschlossenen Fall besteht $\pi(p,q)$ aus der Konkatenation der Kürzesten $\pi(p,e)$, der polygonalen Kette $\pi(e,f)$ und $\pi(f,q)$.
- Ist die Sanduhr offen, sind folgende Fälle zu betrachten:
  - Sind p und q gegenseitig sichtbar, so ist $\pi(p,q) = \overline{pq}$.
  - Andernfalls betrachte $\mathcal{L} = \{\, \overline{rs} \mid r \in d_1, s \in d_2 \,\}$. Wenn ein $\ell \in \mathcal{L}$ existiert, so daß p und q auf derselben Seite von ℓ liegen (o. E. auf der Seite von b und d), so besteht $\pi(p,q)$ aus der Tangente von p an $\pi(b,d)$, der Tangente von q an $\pi(b,d)$ und dem Stück von $\pi(b,d)$, das beide Tangentenendpunkte verbindet. Die Menge $\mathcal{L}$ dient dabei nur der Anschauung, natürlich können nicht alle Liniensegmente explizit berechnet werden. Die Lage von p und q läßt sich auch durch sukzessive Tangentenbildung bestimmen.
  - Wenn jedes $\ell \in \mathcal{L}$ zwischen p und q liegt — o. E. liege p auf der Seite von a, q auf der Seite von d — setzt sich $\pi(p,q)$ zusammen aus der Tangente von p an $\pi(a,c)$, der Tangente von q an $\pi(b,d)$, der gemeinsamen Tangente zwischen $\pi(a,c)$ und $\pi(b,d)$

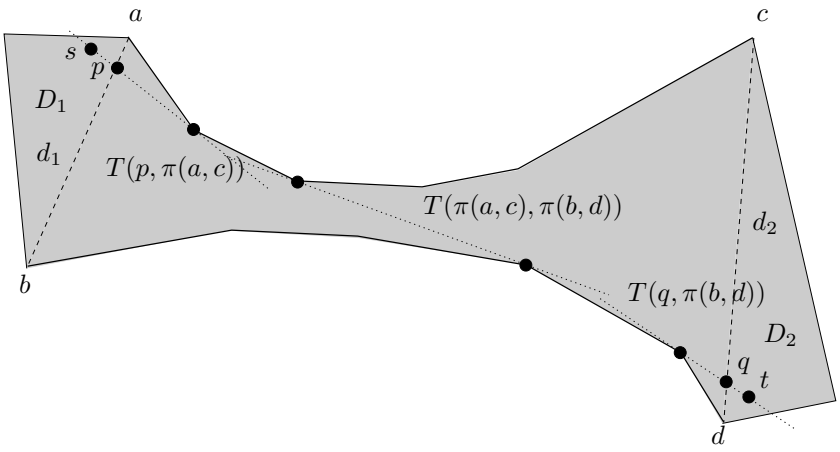
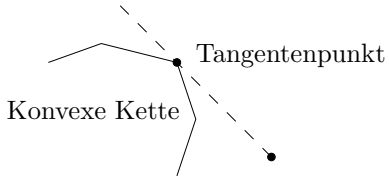


Abbildung 1.28: Kürzester Pfad in offener Sanduhr.

und den Teilen von $\pi(a, c)$ und $\pi(b, d)$, die die Tangentenendpunkte verbinden, siehe Abbildung 1.28.

Analog lassen sich kürzeste Wege zwischen zwei Punkten $s \in D_1$ und $t \in D_2$ finden, die in den d_1 und d_2 aufgesetzten Dreiecken D_1 und D_2 liegen.

Damit reduziert sich die Bestimmung kürzester Wege zwischen Punkten in d_1 und d_2 oder in den angrenzenden Dreiecken auf die Berechnung eines Tangentenpunktes auf konvexen Ketten. Dies läßt sich mit binärer Suche in Zeit $O(\log n)$ lösen; ein Verfahren, das auch bei der Konstruktion der konvexen Hülle benutzt wird, siehe Abbildung 1.29.



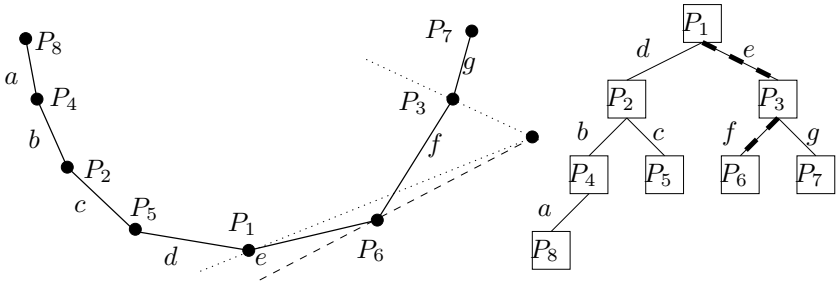


Abbildung 1.29: Speicherung einer Sanduhr in einem binären Baum.

Eine entscheidende Eigenschaft der Sanduhren ist, daß sie sich in logarithmischer Zeit konkatenieren lassen. Dabei sind einige Fallunterscheidungen zu treffen und Details zu beachten, z. B. kann die Konkatenation zweier offener Sanduhren geschlossen sein. Für Details sei hier auf die Arbeit von Guibas und Hershberger [GH87] verwiesen. Es gilt:

Lemma 1.15 *Eine Sanduhr zwischen zwei Diagonalen d_i und d_j werde von m_{ij} Dreiecken der Triangulation überdeckt. Es existiert eine Datenstruktur zur Speicherung der Sanduhren $S(d_i, d_j)$, die folgende Operationen unterstützt:*

(i) Entfernung zwischen zwei Punkten in D_i und D_j in Zeit $O(\log m_{ij})$.

(ii) Kürzeste zwischen zwei Punkten in D_i und D_j in Zeit $O(\log m_{ij} + k)$.

(iii) Zusammensetzen zweier Sanduhren $S(d_i, d_j)$ und $S(d_j, d_\ell)$ zu einer Sanduhr $S(d_i, d_\ell)$ in Zeit und zusätzlichem Platz $O(\log m_{ij} + \log m_{j\ell})$.

Zusammenfassend erhalten wir Algorithmus 1.6 und folgendes Theorem:

Theorem 1.16 *(Guibas, Hershberger, 1987)*
Zu einem Polygon P mit n Ecken kann nach Triangulation und Vorbereitungszeit $O(n)$ zwischen zwei beliebigen Punkten a und b im Inneren von P die Entfernung in Zeit $O(\log n)$ und ein kürzester Pfad in Zeit $O(\log n + k)$ bestimmt werden, wobei k die Anzahl der Segmente auf dem kürzesten Pfad ist. Der Speicherbedarf liegt in $O(n)$. *[GH87, GH89]*

Eine einfachere Datenstruktur für Sanduhren findet sich in [Her91]. Offen ist die Frage, ob sich kürzeste Wege im Polygon in Zeit $O(n)$ ohne Triangulation berechnen lassen.

Algorithmus 1.6 Kürzeste Pfade in einem Polygon (Guibas, Hershberger)

Vorbereitungsphase:

- Berechne Triangulation T, dualen Graphen T^*, hierarchische Triangulation $\hat{T}$ und Schichtengraph $\hat{G}$.
- Bestimme Prä– und Postorderbeschriftungen der Knoten in T^*.
- Berechne für jede Kante (d_i, d_j) in $\hat{G}$ die Sanduhr $S(d_i, d_j)$.
- Baue eine Suchstruktur auf, um Punkte schnell in den Dreiecken lokalisieren zu können (Point-Location).
- Bereite den Graphen $\hat{G}$ für Pfadsuche vor.

Zeit und Platz $O(n)$

Anfrage für Punkte p, q:

- Bestimme die Dreiecke D_p und D_q. $O(\log n)$
- Bestimme in $\hat{G}$ den Pfad zwischen den Diagonalen d_p und d_q:
  - Finde tiefsten gemeinsamen Vorfahren d_{first}
  - Betrachte Pfad $\pi_{\hat{T}}$ von D_p nach D_q in $\hat{T}$:
    Teste für jede Diagonale d in $\pi_{\hat{T}}$, ob d zwischen D_p und D_q in T liegt. Dies ist dann der Fall, wenn d Vorgänger von *entweder D_p oder D_q* in T^* ist.

  Liefert Pfad $\pi_{\hat{G}}$ mit Länge $O(\log n)$ in Zeit $O(\log n)$
- Konstruiere $S(d_p, d_q)$ aus den Sanduhren längs des Pfades $\pi_{\hat{G}}$ von D_p nach D_q im Graphen $\hat{G}$. $O(\log^2 n)$
  Durch Berechnung zusätzlicher Sanduhren im voraus in Zeit und Platz $O(n)$ auch dies auch möglich in Zeit $O(\log n)$
- Bestimme die Kürzeste von p nach q mit Hilfe von $S(d_p, d_q)$.
  $O(\log n + k)$

1.2.3 Anwendung: Berechnung des geodätischen Durchmessers

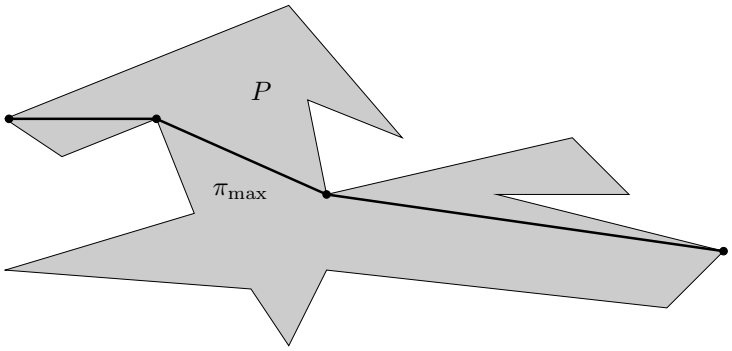


Abbildung 1.30: Ein Polygon und sein geodätischer Durchmesser.

Eine Anwendung von kürzesten Pfaden im Inneren eines Polygons ist die Bestimmung des geodätischen Durchmessers, der im Bereich der Mustererkennung eine Rolle spielt; Abbildung 1.30 zeigt ein Polygon mit seinem geodätischem Durchmesser.

Definition 1.17 Der geodätische Durchmesser $\pi_{\max}$ eines Polygons P ist der längste kürzeste Pfad zwischen zwei Ecken im Inneren von P:

$$\pi_{\max} := \max_{p_i,p_j \text{Ecken von } P} \mathrm{d}(p_i, p_j).$$

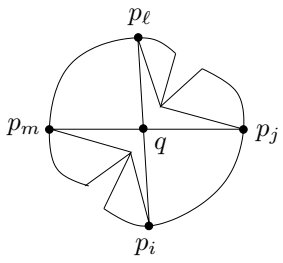
Dabei bezeichnet $\mathrm{d}(p_i, p_j)$ den geodätischen Abstand von p_i und p_j: die Länge des kürzesten Pfades von p_i nach p_j.

Ein naiver Ansatz wäre, für jedes Paar (p_i, p_j) den geodätischen Abstand $\mathrm{d}(p_i, p_j)$ nach Theorem 1.16 in Zeit $O(\log n)$ und Vorbereitungszeit $O(n)$ zu berechnen und dann das Maximum dieser Abstände zu bestimmen; damit könnten wir den geodätischen Durchmesser in Zeit $O(n^2 \log n)$ berechnen.

Auch hier läßt sich mehr erreichen, die folgenden beiden Ideen gehen auf Aggarwal, Klawe, Moran, Shor und Wilber (1986) zurück. [AKM$^+$87a]

1. Idee:
Nutze aus, daß die paarweisen Abstände $\mathrm{d}(p_i, p_j)$ nicht voneinander unabhängig sind. Konkret gilt für alle Ecken p_i, p_j, p_ℓ, p_m, die in dieser Reihenfolge auf dem Rand des Polygons liegen:

$$\mathrm{d}(p_i, p_m) + \mathrm{d}(p_j, p_\ell) < \mathrm{d}(p_i, p_\ell) + \mathrm{d}(p_j, p_m).$$



Dies folgt aus den Dreiecksungleichungen
$\mathrm{d}(p_i, p_m) \le \mathrm{d}(p_i, q) + \mathrm{d}(q, p_m)$ und
$\mathrm{d}(p_j, p_\ell) \le \mathrm{d}(p_j, q) + \mathrm{d}(q, p_\ell)$ mit
$\mathrm{d}(p_j, p_m) = \mathrm{d}(p_j, q) + \mathrm{d}(q, p_m)$ und
$\mathrm{d}(p_i, p_\ell) = \mathrm{d}(p_i, q) + \mathrm{d}(q, p_\ell)$.

Seien $p_1, p_2, \ldots, p_n$ die Ecken des Polygons gegen den Uhrzeigersinn. Definiere eine $n \times n$ Matrix $A = (a_{i\ell})$ durch:

$$a_{i\ell} = \begin{cases} \ell - n & \text{für } 1 \le \ell \le i \\ \mathrm{d}(i, \ell) & \text{für } i+1 \le \ell \le n \end{cases}$$

Dabei sei $\mathrm{d}(i,j) := \mathrm{d}(p_i, p_j)$. Die Matrix A besteht im oberen Dreieck aus Pfadlängen, im unteren Dreieck aus negativem Füllwerk.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ n-1 \\ n \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & & n-1 & n \end{array} \\ \begin{pmatrix} 1-n & \mathrm{d}(1,2) & \mathrm{d}(1,3) & \dots & \mathrm{d}(1,n-1) & \mathrm{d}(1,n) \\ 1-n & 2-n & \mathrm{d}(2,3) & \dots & \mathrm{d}(2,n-1) & \mathrm{d}(2,n) \\ 1-n & 2-n & 3-n & \dots & \mathrm{d}(3,n-1) & \mathrm{d}(3,n) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1-n & 2-n & 3-n & \dots & \dots & \mathrm{d}(n-1,n) \\ 1-n & 2-n & 3-n & \dots & \dots & 0 \end{pmatrix} \end{array}$$

Definition 1.18 Eine $n \times m$-Matrix $A = (a_{i\ell})$ heißt **monoton**, falls

$$\forall 1 \le i < j \le n, 1 \le k < \ell \le m : (a_{ik} < a_{i\ell} \Rightarrow a_{jk} < a_{j\ell}).$$

Anschaulich:

$$\begin{array}{c} \\ i \\ \\ j \end{array} \begin{array}{c} \begin{array}{ccc} k & & \ell \end{array} \\ \begin{pmatrix} a_{ik} & < & a_{i\ell} \\ & \Downarrow & \\ a_{jk} & < & a_{j\ell} \end{pmatrix} \end{array}$$

Bemerkung. Sei A eine monotone Matrix. Dann gilt: je tiefer die Zeile, desto weiter rechts liegt das linkeste Zeilenmaximum.

So kann in folgender monotonen Matrix das Maximum der dritten Zeile nicht links von der vierten Spalte stehen — dem Maximum der zweiten Zeile.

$$\begin{pmatrix} 5 & 4 & 3 & 2 & 1 & 0 \\ 1 & 7 & 3 & \boxed{9} & 4 & 8 \\ 4 & 6 & 5 & \boxed{8} & 8 & 9 \end{pmatrix}$$

Lemma 1.19 *Die oben definierte Matrix A ist monoton.*

Beweis.

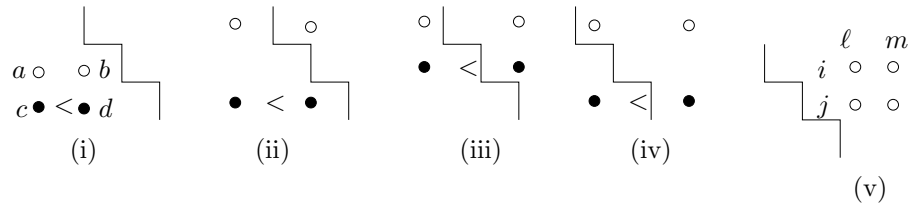


Abbildung 1.31: Fallunterscheidung nach Lage von A in T.

Sei $T = \left(\begin{smallmatrix} a_{i\ell} & a_{im} \\ a_{j\ell} & a_{jm} \end{smallmatrix}\right) = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ mit $i < j, \ell < m$ eine Untermatrix von A. Je nach Lage von T in A unterscheiden wir folgende Fälle, siehe Abbildung 1.31: In Fall (i) liegt T unterhalb der Treppe, in diesem Fall—und ebenso in den Fällen (ii) bis (iv) gilt $c < d$.

Als einzig interessanter Fall bleibt (v). In diesem Fall folgt aus der Definition der Matrix A: $i < j < \ell < m$. Die Ungleichung $j < l$ gilt, da sich a_{il} oberhalb der Diagonalen der Matrix befindet. Insgesamt folgt aus den Annahmen $i < j, l < m$, dass $1 \le i < j < l < m \le n$ gilt und sich somit die Punkte i, j, l, m in dieser Reihenfolge auf dem Rand des Polygons befinden.

Für die Knoten auf dem Rand des Polygons gilt deshalb $\mathrm{d}(i,m)+\mathrm{d}(j,\ell) \leq \mathrm{d}(i,\ell)+\mathrm{d}(j,m)$, aus $\mathrm{d}(i,\ell) < \mathrm{d}(i,m)$ folgt also $\mathrm{d}(j,\ell) < \mathrm{d}(j,m)$.

□

2. Idee:

Theorem 1.20 *Sei A eine monotone $n \times m$-Matrix, $n \leq m$. Dann läßt sich das maximale Element in Zeit $O(m)$.*

Beweis. Berechne mit Algorithmus 1.8 für jede Zeile i das am weitesten links stehende Maximum $\max(i)$. Die Anzahl der Vergleiche in Algorithmus 1.7 ist dabei beschränkt durch das Maximum der Anzahl Zeilen (Anzahl der Elemente auf der Diagonalen) und dem der Anzahl der Spalten (maximale Anzahl der Spaltenstreichungen), also m.

Damit ist nur noch die Laufzeit von Algorithmus 1.8 zu bestimmen: Sei $T(n)$ die Anzahl der Zugriffe auf Matrixelemente der Prozedur Zeilenmaxima. Sei $n = 2^k$:

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{2}\right) + C \cdot n + O(m) \\ &\leq T\left(\frac{n}{4}\right) + C \cdot \frac{n}{2} + C \cdot n + O(m) \\ &\leq \dots \\ &\leq T(1) + C\left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \ldots + \frac{1}{2} + 1\right) \cdot n + O(m) \\ &\in O(n+m) \\ &\in O(m) \end{aligned}$$

□

Bemerkung: Im ersten Rekursionsschritt entsehen $O(m)$ Kosten für die Spaltenreduktion zu einer quadratischen Matrix. In den weiteren Rekursionsschritten sind die Kosten der Spaltenreduktion in der Konstante C enthalten. Analog entstehen am Ende einmalig zusätzlich O(m) Kosten für die Berechnung der Maxima der ungeraden Zeilen.
Da wir nach Theorem 1.20 die Zeilenmaxima in Zeit $O(m)$ bestimmen können, folgt direkt, dass wir nur $O(m)$ Matrixelemente miteinander vergleichen müssen.
Mit Theorem 1.20 können wir zu einer *gegebenen* Matrix schnell das maximale Element ausrechnen. Allerdings würde es uns schon Zeit $O(n^2)$ kosten, die Matrix A aufzustellen! Hier hilft uns die Tatsache, daß wir gar nicht alle $O(n^2)$ Einträge der Matrix brauchen, sondern mit $O(n)$ Vergleichen auskommen. Wir berechnen die Matrix A also nicht explizit, sondern die benötigten Matrixelemente erst dann, wenn sie gebraucht werden. Damit gilt:

Theorem 1.21 *Der geodätische Durchmesser eines einfachen Polygons mit n Ecken läßt sich in Zeit $O(n \log n)$ bestimmen. Es läßt sich sogar für jede Ecke der am weitesten entfernte Partner in dieser Zeit feststellen.* *[AKM+87a]*

Durch bessere Kombination von Matrixsuche und Kürzestenbestimmung ist dies sogar in Zeit $O(n)$ möglich (Hershberger, Suri, 1993, [HS93, HS97]).

Algorithmus 1.7 Spaltenreduktion

Input: monotone $n \times m$-Matrix A, $n \leq m$.

Output: monotone $n \times n$-Matrix A' mit denselben Zeilenmaxima wie A; aus der Position von $\max(i)$ in A' ist $\max(i)$ in A rekonstruierbar.

Beginnend mit $a_{1,1}$ teste, ob $a_{i,i} < a_{i,i+1}$ ist. Ist dies der Fall, streiche die i-te Spalte, da die Elemente der i-ten Spalte wegen der Monotonieeigenschaft keine Zeilenmaxima mehr sein können. Vergleiche dann $a_{i-1,i-1}$ mit $a'_{i-1,i} = a_{i-1,i+1}$ usw. Ist $a_{i,i} \geq a_{i,i+1}$, inkrementiere i; vergleiche also als nächstes $a_{i+1,i+1}$ mit $a_{i+1,i+2}$.

Lautet die Antwort für alle Elemente auf der Diagonalen "nein", so sind alle Elemente der n-ten Spalte größer gleich den Elementen der Spalte $(n+1)$, also können wir die $(n+1)$-ten Spalte streichen und mit dem Test $a_{n,n} \stackrel{?}{<} a_{n,n+2}$ fortfahren.

Als Output erhalten wir eine quadratische Matrix mit der Eigenschaft, dass die Elemente rechts von der Diagonalen in jeder Zeile eine monoton fallende Folge bilden, d.h. $a_{i,j} \geq a_{i,j+1}\ \forall j \geq i$.

Algorithmus 1.8 Zeilenmaxima

Input: monotone $n \times m$-Matrix B, $n \leq m$

Output: alle linkesten Zeilenmaxima $\max(i), 1 \leq i \leq n$

- $C :=$ alle Zeilen von B mit geradem Zeilenindex; $O(n)$
- $C' :=$ `Spaltenreduktion`(C); $O(m)$
- `Zeilenmaxima`(C'); `/*` C' ist $\frac{n}{2} \times \frac{n}{2}$-Matrix `*/` $T(\frac{n}{2})$
- Rekonstruiere Zeilenmaxima von C; `/*` = gerade Zeilen von B `*/` $O(n)$
- Berechne die Maxima der ungeraden Zeilen von B:
  Wegen Bemerkung nach Definition 1.18 liegt $\max(2i+1)$ zwischen den Spalten, in denen $\max(2i)$ und $\max(2i+2)$ liegen. Da sich die Intervalle, in denen nach den Maxima der ungeraden Zeilen gesucht werden muß, nicht überlappen, ist dies möglich in Zeit $O(m)$

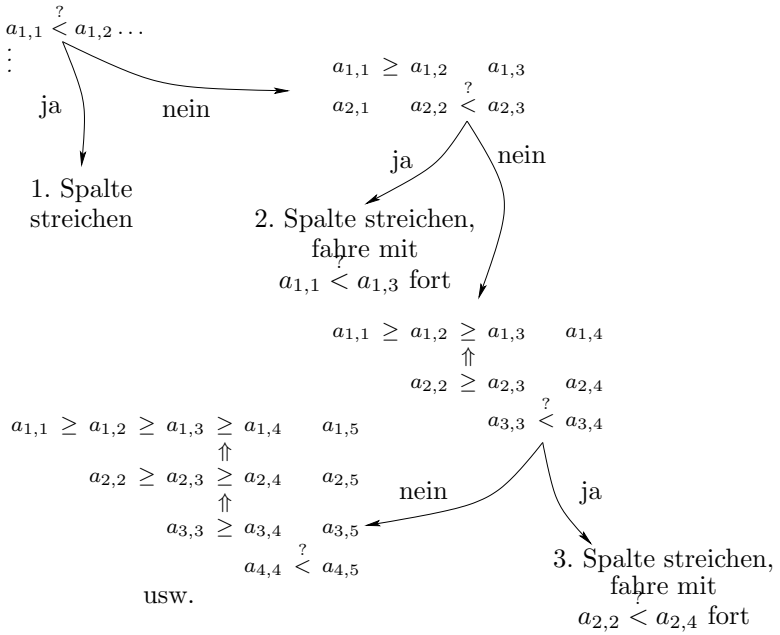


Abbildung 1.32: Spaltenreduktion.

1.2.4 Shortest Watchman Routes

Eine andere Anwendung von kürzesten Pfaden innerhalb eines Polygons ist die Berechnung der kürzesten Wächterroute oder Shortest Watchman Route (SWR). Gesucht ist der kürzeste Weg in einem Polygon, den ein Wachmann oder Roboter mit unbeschränkter Sicht gehen kann und auf dem Weg jeden Punkt in dem Polygon mindestens einmal sieht.[9] Eine Route ist stets ein geschlossener Weg, d. h. der Roboter kehrt zum Startpunkt zurück. Bei der einfacheren Variante des Shortest Watchman Problems ist der Startpunkt des Roboters und damit der Route gegeben, bei der schwierigeren Variante sucht man die kürzeste Route, die in einem beliebigen Punkt des Polygons startet. Ebenfalls schwieriger zu berechnen als eine SWR ist eine Shortest Watchman Tour, also ein Pfad, der jeden Punkt im Polygon einmal sieht, jedoch nicht zum Startpunkt zurückkehrt. Das Shortest Watchman Problem wurde 1986 von Chin und Ntafos vorgestellt [CN86]. Seitdem wurden zahlreiche Arbeiten zur Berechnung der SWR veröffentlicht, von denen sich jedoch viele als fehlerhaft herausgestellt haben oder bzgl. der Laufzeit verbessert wurden, siehe z. B. [CN91, THI93, TH93, HN97, THI99, CJN99, CJ95].

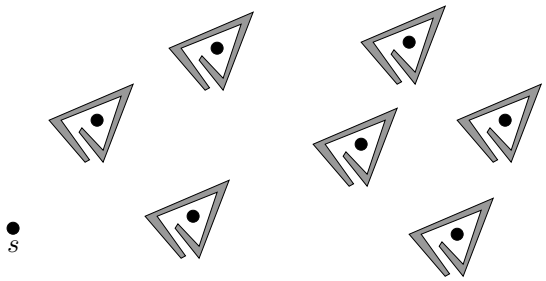


Abbildung 1.33: Die Berechnung einer SWR liefert eine Besuchsreihenfolge der Punkte.

Im allgemeinen ist die Berechnung einer SWR schwierig:[10]

Theorem 1.22 *(Chin, Ntafos, 1986)*
Die Berechnung einer Shortest Watchman Route in einer Szene mit polygonalen Hindernissen (Polygon mit Löchern) ist NP-hard. *[CN86, CN88]*

Beweis. In Polygonen mit Löchern läßt sich das Traveling Salesman Problem auf das Shortest Watchman Problem reduzieren: die Bestimmung der kürzesten Besuchsreihenfolge der polygonalen Löcher entspricht dem TSP. Wir umgeben alle Punkte aus der Eingabe für das TSP mit einem Hindernis, das die Sicht auf alle anderen Punkte abschneidet, siehe Abbildung 1.33. Jedes dieser Hindernisse hat $O(1)$ Kanten und die Konstruktion der gesamten Szene ist in Zeit $O(n)$ möglich. Eine Lösung für das Shortest Watchman Problem in dieser Szene liefert eine kürzeste Besuchsreihenfolge der Punkte und damit eine Lösung für das TSP. □

Wir beschränken uns daher auf einfache Polygone, d. h. Polygone ohne Selbstschnitte und Löcher, und auch in dieser Klasse von Polygonen zunächst nur auf zwei Spezialfälle, siehe Abbildung 1.34.

Definition 1.23 Ein Polygon P heißt **monoton**, wenn eine Gerade ℓ existiert, so daß für jede zu ℓ orthogonale Gerade ℓ' der Schnitt von P mit ℓ' zusammenhängend ist, d. h. der Schnitt ist *ein* Liniensegment, *ein* Punkt oder leer. Ist die Gerade ℓ eine Parallele zur Y–Achse, so heißt P **y–monoton**.
P heißt **rechtwinklig**, wenn jeder innere Winkel 90° oder 270° beträgt.

Den einfachsten Fall stellen rechtwinklige, monotone Polygone dar:

[9] Ein Weg π sieht das gesamte Polygon P :⇔ zu jedem Punkt $p \in P$ existiert ein Punkt $p' \in \pi$, so daß das Liniensegment $\overline{pp'}$ vollständig in P liegt.

[10] Zu den Begriffen NP-hard und NP-vollständig siehe Anhang.

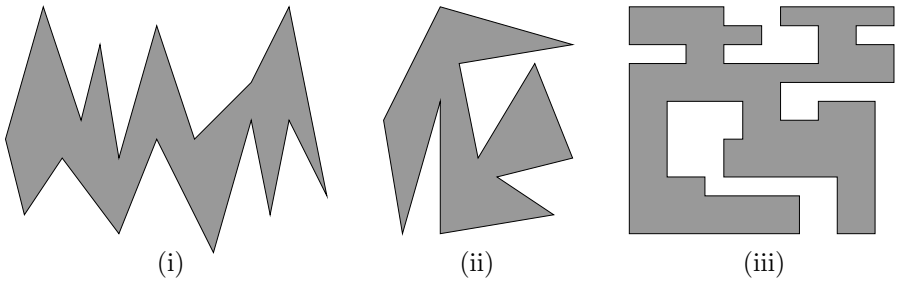


Abbildung 1.34: (i) X–monotones Polygon, (ii) nicht–monotones Polygon, (iii) rechtwinkliges Polygon.

Theorem 1.24 *(Chin, Ntafos, 1986)*
In einem rechtwinkligen, monotonen Polygon kann die Shortest Watchman Route in Zeit $O(n)$ berechnet werden. *[CN86, CN88]*

Beweis. Übungsaufgabe. □

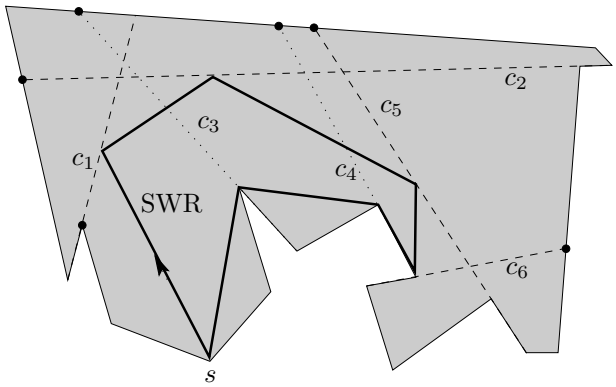


Abbildung 1.35: Polygon mit notwendigen Cuts (gepunktet), wesentlichen Cuts (gestrichelt) und Shortest Watchman Route.

Auch in rechtwinkligen, nicht–monotonen Polygonen kann die SWR recht einfach berechnet werden. Wir brauchen dazu einige Begriffe, die wir gleich für allgemeine — d. h. nicht–rechtwinklige — Polygone einführen.

Definition 1.25 Die Verlängerungen der Kanten von reflexen Ecken[11] in das Polygon hinein sind die **Cuts** des Polygons. Von den beiden Cuts einer reflexen Ecke ist nur einer interessant, nämlich derjenige, der bzgl. des Startpunktes s die Sicht blockiert.[12] Wir nennen diese Cuts **notwendige Cuts**. Ein Cut c_1 **dominiert** einen Cut c_2, wenn jeder Weg von s zu c_1 den Cut c_2 schneidet. Notwendige Cuts, die von keinem anderen notwendigen Cut dominiert werden, heißen **wesentliche Cuts**.

Ein Cut c_i teilt das Polygon in zwei Teile. Die zu c_i gehörende **Tasche** P_{c_i} ist der Teil, der den Startpunkt s nicht enthält. Damit können wir definieren — und man kann leicht sehen, daß beide Definitionen äquivalent sind —, daß ein Cut c_1 einen anderen Cut c_2 dominiert, wenn $P_{c_1} \subset P_{c_2}$ gilt.

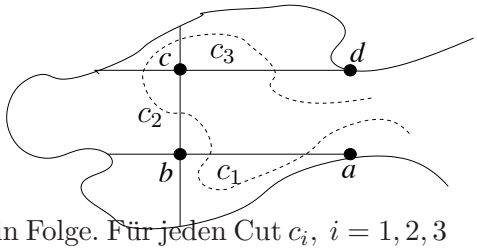
Cuts, die von anderen dominiert werden, werden sozusagen nebenbei exploriert und müssen nicht weiter berücksichtigt werden. Abbildung 1.35 zeigt ein Beispiel für ein Polygon mit seinen notwendigen und wesentlichen Cuts. c_3 und c_4 sind nicht wesentlich, da jeder Weg zu c_5 über diese Cuts führt. c_5 dominiert c_3 und c_4. Die SWR (und jede andere Explorationstour) berührt

[11] Polygonecke, deren Innenwinkel $> \pi$ ist.
[12] Anschaulich der Cut, den wir überschreiten müssen, um hinter die Ecke schauen zu können.

alle wesentlichen Cuts und die Menge der wesentlichen Cuts ist die kleinste Menge von Cuts, die besucht werden müssen, um das ganze Polygon zu sehen.

In einem Polygon mit gegebenem Startpunkt s können wir die Cuts entsprechend dem im Uhrzeigersinn ersten Schnittpunkt zwischen Cut und Polygonrand ordnen, siehe Abbildung 1.35. Es gilt:

Lemma 1.26 *In einem rechtwinkligen Polygon existiert stets eine SWR, die die wesentlichen Cuts gemäß der Ordnung auf dem Rand besucht.*



Beweis. Wir zeigen zunächst, dass eine SWR nie in eine Tasche hineinläuft bzw. es wird kein wesentlicher Cut überschritten. Kritisch sind nur Situationen, wo sich wesentliche Cuts schneiden. Dies können bei rechtwinkeligen Polygonen maximal vier in Folge sein. Wir betrachten die Situation mit drei Cuts in Folge. Für jeden Cut $c_i,\ i = 1, 2, 3$ gilt: Wenn man den Weg in der Tasche von c_i durch die orthogonale Projektion auf c_i ersetzt, werden die gleichen Cuts besucht wie vorher. Der Weg wird dabei nur kürzer. Daher reicht es, die Cuts $c'_1 = ab$, $c'_2 = bc$, $c'_3 = cd$ zu besuchen, ohne in die Taschen zu laufen.
Eine kürzeste Tour darf keine Selbstschnitte haben, sonst könnte man die Tour lokal abkürzen. Es folgt somit, dass nur durch Besuchen der Cuts entlang der Ordnung auf dem Rand eine schnittfreie und somit kürzeste Tour entsteht. □

Algorithmus 1.9 Shortest Watchman Route in einfachen Polygonen

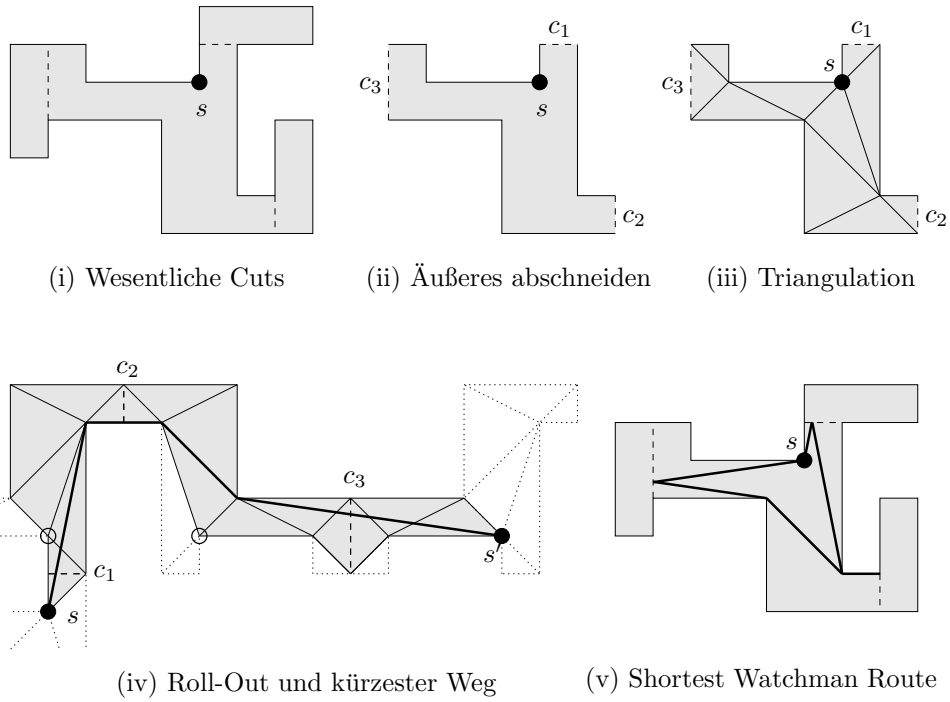
- Bestimme die wesentlichen Cuts $c_1, \ldots, c_k$. $O(n)$
- Schneide die Teile des Polygons ab, die bzgl. des Startpunktes hinter wesentlichen Cuts liegen. $O(n)$
- Trianguliere das resultierende Polygon P'. $O(n)$
- Konstruiere P'' durch "Roll–Out" von P': $O(n)$
  - $P^{(1)}$ seien die relevanten Dreiecke von P' auf dem Weg im dualen Graphen T^* von s nach c_1.
  - Für jeden Cut c_i, $i = 2, \ldots, k$: Erweitere $P^{(i-1)}$ zu $P^{(i)}$ durch alle relevanten Dreiecke entlang des Randes von P' auf dem Weg von c_{i-1} zu c_i, gespiegelt an den Cuts $c_1, \ldots, c_{i-1}$ und angesetzt an c_{i-1}.
  - Erweitere $P^{(k)}$ zu P'' wie oben durch alle relevanten Dreiecke auf dem Weg von c_k nach s. Dabei entsteht eine gespiegelte Kopie s' des Zielpunktes.
- Berechne in P'' den kürzesten Weg π von s nach s'. $O(n)$
- Die SWR entsteht durch "zurückfalten" der Liniensegmente in π.

Lemma 1.26 liefert uns den Algorithmus für die Berechnung einer SWR:

Theorem 1.27 *(Chin, Ntafos, 1986)*
In einem rechtwinkligen Polygon kann die Shortest Watchman Route in Zeit $O(n)$ berechnet werden. *[CN86, CN88]*

Beweis. Algorithmus 1.9 berechnet die SWR in einem rechtwinkligen Polygon. Abbildung 1.36 zeigt ein Beispiel.

Die wesentlichen Cuts lassen sich in Zeit $O(n)$ berechnen (ohne Beweis). Damit bleibt zu zeigen, daß P'' nicht mehr als $O(n)$ Kanten enthält. Dazu betrachten wir den zur Triangulation dualen Graphen T^*. Dem sukzessiven Besuch der Cuts $c_1, \ldots, c_n$ entspricht ein Depth-First Durchlauf von T^*. Jede Kante von T^* wird dabei zweimal durchlaufen. Damit wird auch jedes Dreieck der Triangulation maximal zweimal in P'' übernommen und die Anzahl der Kanten von P'' liegt in $O(n)$. □



(i) Wesentliche Cuts (ii) Äußeres abschneiden (iii) Triangulation

(iv) Roll-Out und kürzester Weg (v) Shortest Watchman Route

Abbildung 1.36: Berechnung der SWR in einem rechtwinkligen Polygon.

Algorithmus 1.9 wäre mit leichten Modifikationen bei gleicher Laufzeit in Polygonen anwendbar, in denen sich je maximal zwei wesentliche Cuts gegenseitig schneiden, da auch Polygone dieser Art Lemma 1.26 erfüllen. In allgemeinen Polygonen können sich allerdings mehr als zwei Cuts gegenseitig schneiden. Wir nennen solche Mengen von Cuts eine "Corner". In solchen Situationen ist nicht von vornherein klar, in welcher Reihenfolge die Cuts besucht werden.

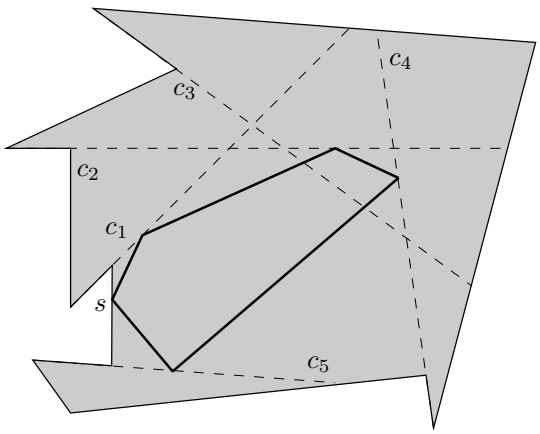


Abbildung 1.37: Eine "Corner": die Cuts c_1, c_2, c_3 und c_4 schneiden sich gegenseitig.

Definition 1.28 Eine "Corner" ist eine geordnete Menge von Cuts $c_i, c_{i+1}, \ldots c_j$, für die gilt

(i) c_k schneidet c_{k+1} für $k \in \{i, \ldots, j-1\}$,
(ii) c_i schneidet c_{i-1} *nicht* und
(iii) c_j schneidet c_{j+1} *nicht.*

Betrachte Abbildung 1.37: die Cuts c_1, c_2, c_3 und c_4 bilden eine Corner und die Reihenfolge der Cuts wird von der SWR nicht eingehalten: c_3 wird überschritten, bevor c_2 erreicht wird. Die Besuchsreihenfolge der Corners ist jedoch fest:

Lemma 1.29 *Die Shortest Watchman Route besucht die Corners eines Polygons gemäß der Ordnung entlang des Randes.*

Die von der SWR berührten Abschnitte der Cuts zwischen zwei Schnittpunkten werden aktive Segmente genannt. Sind die aktiven Segmente gegeben, kann man die SWR wie oben berechnen, das große Problem ist jedoch, die Menge der aktiven Segmente zu berechnen. Erste Ansätze waren, mit einer Menge von aktiven Segmenten eine Watchman Route R' zu berechnen und diese durch lokale Anpassungen, d. h. durch Austausch der aktiven Segmente, zu verkürzen. Einen anderen Ansatz verfolgen Dror et. al. die die folgende Verallgemeinerung des Shortest Watchman Problems und ähnlicher Probleme lösen.

Definition 1.30
(i) Beim einfachen **Touring Polygon Problem** (TPP) ist eine Sequenz von einfachen, konvexen, disjunkten Polygonen $P_1, P_2, \ldots, P_k$ mit insgesamt n Kanten, ein Startpunkt s und ein Zielpunkt t gegeben. Gesucht ist der kürzeste Pfad π, der in s startet, in t endet und alle Polygone P_i in der gegebenen Reihenfolge besucht.
(ii) Beim allgemeinen TPP verlangen wir zusätzlich, daß die Pfade zwischen zwei Polygonen P_i und P_{i+1} ($i = 0, \ldots, k$; $P_0 := s; P_{k+1} := t$) innerhalb des **Zaunes** F_i verlaufen. Ein Zaun F_i ist gegeben durch ein einfaches Polygon, das P_i und P_{i+1} enthält. Die Polygone dürfen sich überlappen und nur die **Fassade** eines Polygons P_i muß konvex sein. Die Fassade eines Polygons P_i ist der Teil des Randes von P_i, der nicht auf dem Rand des Zaunes F_{i-1} liegt: Fassade$(P_i) := \partial P_i \setminus \partial F_{i-1}$.

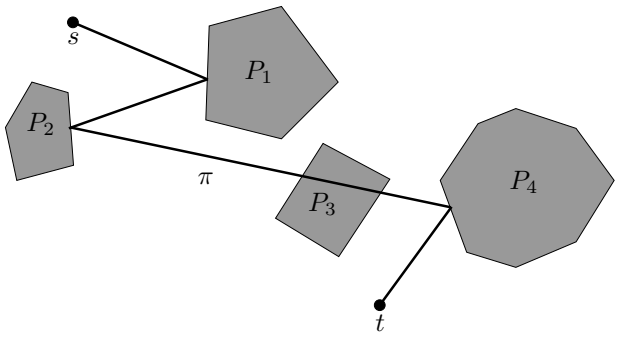


Abbildung 1.38: Ein Beispiel für das einfache Touring Polygon Problem.

Beim TPP wird nicht ausgeschlossen, daß ein Polygon P_i vor einem Polygon P_j mit $j < i$ besucht wird, allerdings muß das Polygon P_i dann ein weiteres mal besucht werden. Der Besuch eines Polygons P_i ist erst dann gültig, wenn vorher alle anderen Polygone $P_1, \ldots P_{i-1}$ besucht wurden, andernfalls gilt das Polygon nach wie vor als unbesucht. Abbildung 1.38 zeigt ein Beispiel für das einfache TPP, Abbildung 1.39 für das allgemeine TPP. Der gestrichelte Teil des Randes von P_4 ist die Fassade von P_4. Beachte, daß P_5 vor P_4 besucht wird, aber "gewertet" wird erst der Besuch von P_5 nach dem Besuch von P_4.

Beschränken wir uns zunächst auf das einfache TPP. Klar ist, daß der kürzeste Weg eine polygonale Kette sein muß, andernfalls ließe sich der Weg verkürzen. Die erste wichtige Idee zur

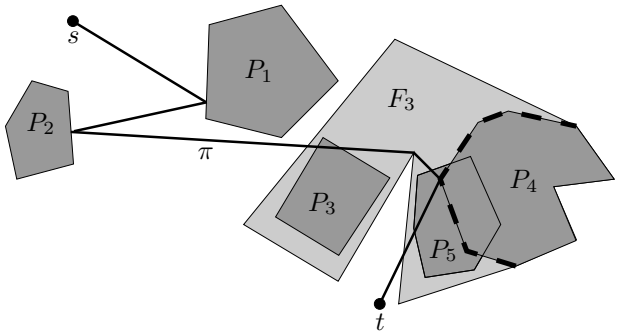


Abbildung 1.39: Ein Beispiel für das allgemeine Touring Polygon Problem.

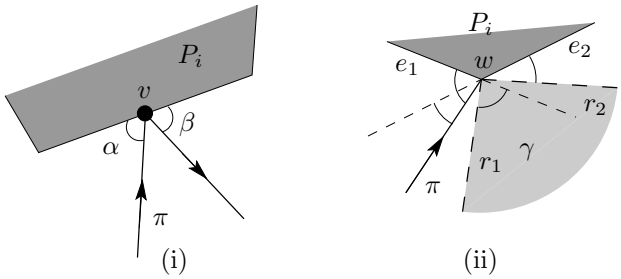


Abbildung 1.40: Lokale Optimalität: (i) bei Reflektion an einer Kante gilt $\alpha = \beta$, (ii) bei Reflektion an einer Ecke liegt die Ausgangskante im Winkelbereich γ.

Berechnung des Pfades π ist, lokale Optimalität auszunutzen; eine Idee, die wir schon bei der Berechnung der kürzesten Pfade in Abschnitt 1.2 benutzt haben:

Lemma 1.31

(i) Ein Pfad π ist lokal optimal, wenn die Ecken von π auf dem Rand eines der Polygone P_i liegen. Liegt die Ecke v von π auf einer Polygonkante, so sind die Winkel zwischen der Polygonkante und den beiden zu v inzidenten Kanten gleich (Einfallswinkel = Ausfallswinkel, siehe Abbildung 1.40(i)). Liegt die Ecke v auf einer Polygonecke w, so liegt die ausgehende Kante von π in dem Winkelbereich γ, der durch die Reflektionen r_1 und r_2 der eingehenden Kante an den beiden zu w inzidenten Polygonkanten e_1 und e_2 begrenzt wird, siehe dazu auch Abbildung 1.40(ii)).

(ii) Für jedes $p \in \mathbb{R}^2$ und jedes $i \in \{1, \ldots, k\}$ gibt es einen eindeutigen, lokal optimalen Pfad $\pi_i(p)$, der in s startet, die Polygone P_1 bis P_i in der gegebenen Reihenfolge besucht und in p endet. Einen solchen Pfad nennen wir im folgenden den **kürzesten i–Pfad** *zu p.*

(iii) Ein lokal optimaler i–Pfad ist global optimal.

Beweis.

(i) Ist die Bedingung verletzt, wäre der Pfad lokal verkürzbar.

(ii) Dies folgt später aus der Konstruktion der lokal optimalen Pfade.

(iii) Folgt aus (ii).

□

Die zweite Idee ist, die Ebene in Regionen mit kombinatorisch gleichen Pfaden einzuteilen, d. h. wir zerlegen die Ebene $\mathbb{R}^2$ disjunkt in Regionen $\mathcal{R}_j$, so daß für alle Punkte $p \in \mathcal{R}_j$ der kürzeste k–Pfad von p zu s über die gleichen Polygonecken/Polygonkanten führt bzw. an den gleichen Polygonecken/Polygonkanten reflektiert wird. Auch diese Idee haben wir bereits in Form der Shortest Path Map benutzt. Abbildung 1.41 zeigt ein Beispiel für eine vollständige kombinatorische Shortest Path Map für unser Problem. Die Beschriftungen der Regionen geben

an, an welchen Kanten bzw. Ecken der Szene der kürzeste Weg zu s reflektiert wird. Das Problem ist, daß eine vollständige kombinatorische Shortest Path Map in einer Szene mit k Polygonen und insgesamt n Ecken eine Komplexität von $\Omega((n-k)2^k)$ haben kann (Übungsaufgabe). Andererseits brauchen wir gar nicht die gesamte Karte zu speichern. Es reicht aus, wenn wir für jedes Hindernis P_i eine Karte speichern in der festgehalten wird, über welche Kante bzw. Ecke von P_i das letzte Segment eines kürzesten i–Pfades von s zu p verläuft. Die Fortsetzung des Weges von P_i zu P_{i-1} erhalten wir dann durch Nachschauen in der Karte zu P_{i-1} usw.

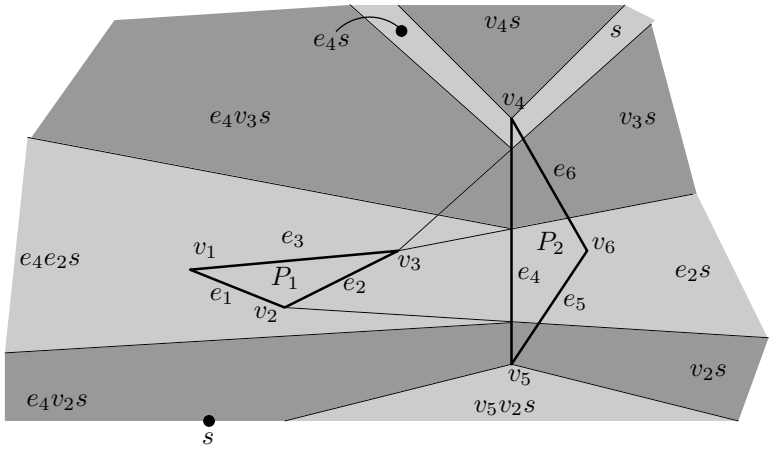


Abbildung 1.41: Vollständige kombinatorische Shortest Path Map.

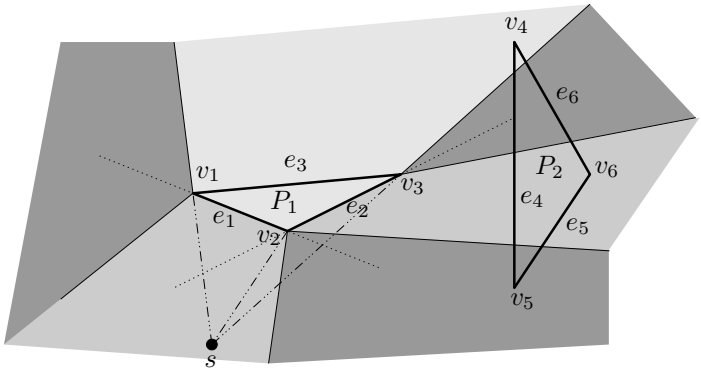


Abbildung 1.42: Last Step Shortest Path Map zu P_1.

In einer solchen Karte gibt es zwei Klassen von Zellen: Zellen, bei denen der kürzeste Weg über P_i zu s an einer Kante oder Ecke von P_i gemäß den Bedingungen in Lemma 1.31(i) reflektiert wird nennen wir **Reflektionszellen**. Zellen, bei denen der kürzeste Weg über die Kante/Ecke hinweg führt nennen wir **Durchgangszellen**. Wir können alle Durchgangszellen zur **Durchgangsregion** zusammenfassen.

Die Reflektionszellen lassen sich wiederum in zwei Klassen einteilen, abhängig davon, ob der kürzeste Pfad an einer Kante oder an einer Ecke reflektiert wird. Unsere Karte, die **Last Step Shortest Path Map** $\mathcal{S}_i$ zu P_i, besteht dann aus Zellen, die von den an Polygonecken reflektierten Eingangskanten begrenzt werden. Die Eingangskante zu einem Knoten v von P_i ist das letzte Segment des kürzesten $i-1$–Pfades zu v. Damit lassen sich die $\mathcal{S}_i$ sukzessive aufbauen. Abbildung 1.42 und Abbildung 1.43 zeigen Beispiele für die Last Step Shortest Path Maps $\mathcal{S}_1$ und $\mathcal{S}_2$. Abbildung 1.44 zeigt ein weiteres Beispiel. Die hell gefärbten Bereiche stellen die Durchgangsregionen dar.

Wir bezeichnen den Teil von P_i, in dem eingehende Pfadsegmente gemäß Lemma 1.31(i) reflektiert werden, den **Reflektionsbereich** T_i von P_i. Gleichzeitig ist T_i der Bereich, in dem

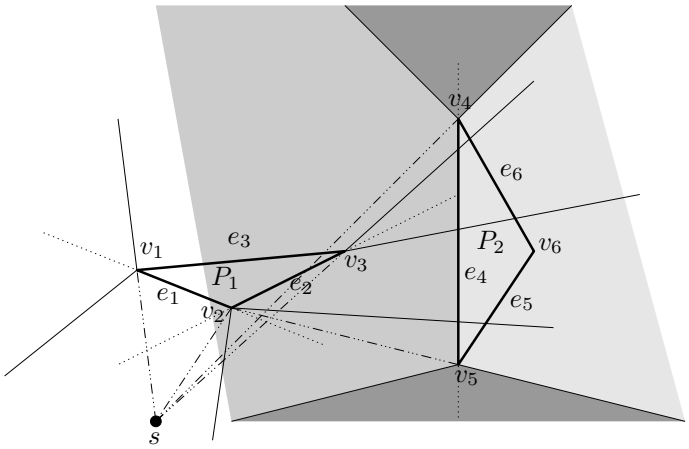


Abbildung 1.43: Last Step Shortest Path Map zu P_2.

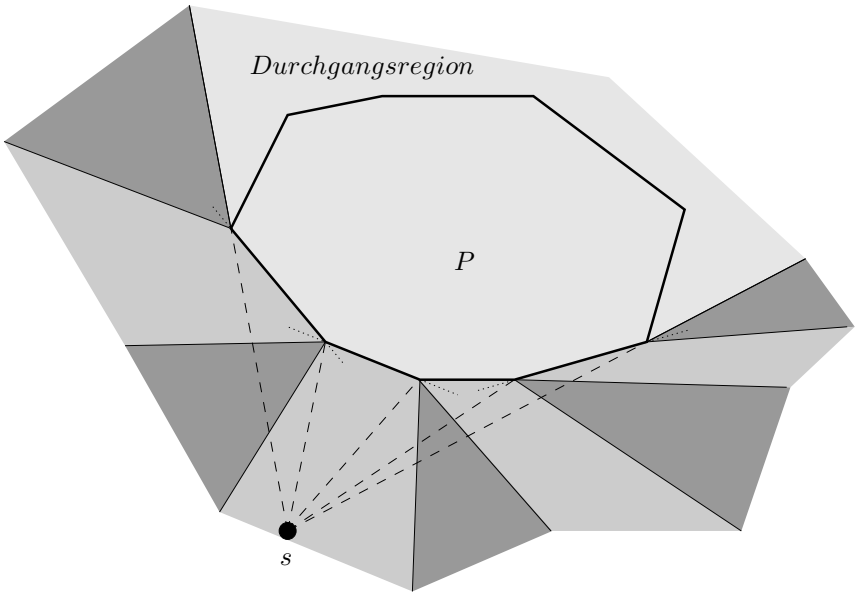


Abbildung 1.44: Last Step Shortest Path Map zu P.

ein kürzester i–Pfad das Polygon P_i nach dem Besuch von $P_1, \ldots, P_{i-1}$ zum ersten Mal besuchen kann. Die von T_i ausgehenden, reflektierten Strahlen bezeichnen wir als den **Stern** R_i von T_i.

Im Gegensatz zur vollständigen kombinatorischen Shortest Path Map kann eine Last Step Shortest Path Map keine hohe Komplexität haben, wie folgendes Lemma zeigt:

Lemma 1.32

(i) T_i ist zusammenhängend.

(ii) Die Strahlen R_i sind disjunkt und jeder Punkt $p \in \mathbb{R}^2$ außerhalb der Durchgangsregion wird von genau einem Strahl von R_i getroffen.

(iii) Die Komplexität einer $\mathcal{S}_i$ liegt in $O(|P_i|)$.

Beweis.

(i) Übungsaufgabe

(ii) Wir zeigen induktiv, daß die Strahlen R_i disjunkt sind. Für $\mathcal{S}_1$ gilt dies offensichtlich. Nehmen wir an, die Aussage gilt für $\mathcal{S}_1, \ldots, \mathcal{S}_{i-1}$. Zu jedem Knoten v von P_i läßt sich eindeutig die Region von $\mathcal{S}_{i-1}$ bestimmen, in der v liegt. Damit ist auch der kürzeste

$(i-1)$–Pfad zu v eindeutig, insbesondere die letzte Kante dieses Pfades. Damit sind alle letzten Kanten zu Knoten aus P_i disjunkt bis auf einen gemeinsamen Startpunkt. Es ist bekannt, daß die Reflektionen disjunkter Liniensegmente an einem konvexen Polygon ebenfalls disjunkt sind.

(iii) Die Kanten in $\mathcal{S}_i$ lassen sich in zwei Klassen unterteilen: die Kanten von T_i und die Strahlen, die von den Ecken von T_i ausgehen. T_i ist zusammenhängend und — da die P_j disjunkt sind — Teil des Randes von P_i. Daher besteht T_i aus maximal $|P_i|$ Kanten. Die von den Ecken von T_i ausgehenden Strahlen sind Teile von R_i und damit disjunkt. Ihr Beitrag zur Komplexität von $\mathcal{S}_i$ liegt damit ebenfalls in $O(|P_i|)$.

□

Damit haben wir eine Vorstellung, wie eine Karte aussehen kann, mit Hilfe derer wir unsere Pfade berechnen können. Es stellen sich nun die Fragen, wie wir die $\mathcal{S}_i$ effizient berechnen und wie wir aus den $\mathcal{S}_i$ einen kürzesten Pfad konstruieren können. Die letzte Frage beantwortet folgendes Lemma:

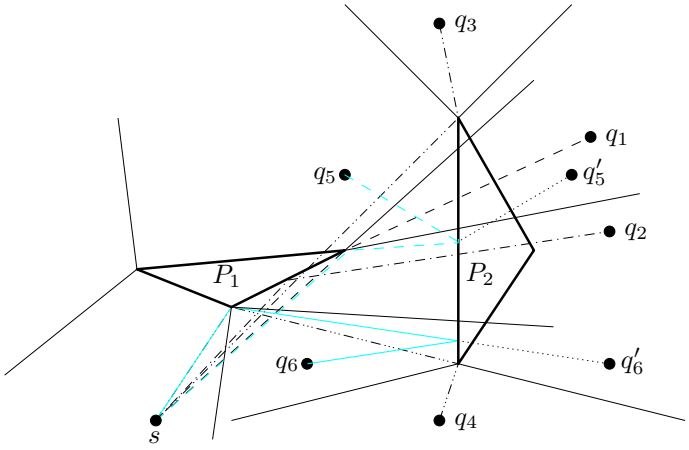
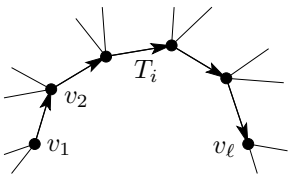


Abbildung 1.45: Queries beim einfachen TPP: q_1 und q_2 liegen in der Durchgangsregion, q_3 und q_4 in einer Zelle zu einer Ecke, q_5 und q_6 in der Zelle einer Kante.

Lemma 1.33 *Sind für das einfache TPP die Last Step Shortest Path Maps $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_i$ gegeben, so kann zu allen $q \in \mathbb{R}^2$ der kürzeste i–Pfad $\pi_i(q)$ in Zeit $O(k \cdot \log \frac{n}{k})$ berechnet werden.*

Beweis. Algorithmus 1.10 liefert zu einem Anfragepunkt $q \in \mathbb{R}^2$ den Pfad $\pi_i(q)$, Abbildung 1.45 zeigt einige Beispiele für kürzeste i–Pfade.



Wir wissen aus Lemma 1.32, daß die Komplexität einer LSSPM $\mathcal{S}_i$ in $O(|P_i|)$ liegt. Die Karte kann daher in einer Datenstruktur gespeichert werden, die in Zeit $O(\log |P_i|)$ die Zelle liefert, in der ein Anfragepunkt liegt. Wir könnten dazu eine Point-Location Datenstruktur benutzen; es reicht jedoch, die Knoten der konvexen Kette T_i in einem Suchbaum oder Array zu speichern. Zu jedem Knoten v halten wir die beiden von v ausgehenden Strahlen ebenfalls fest.

Durch die rekursiven Anfragen liegt die Zeit, um einen kürzesten i–Pfad zu bestimmen, in $O(\sum_{j=1}^{k} \log |P_j|)$. Diese Summe nimmt bei $|P_j| = \frac{n}{k}$ ihr Maximum an. □

Mit Hilfe der Queries lassen sich die Last Step Shortest Path Maps recht einfach sukzessiv konstruieren, siehe Algorithmus 1.11. Die Zeit zur Konstruktion einer einzelnen $\mathcal{S}_i$ liegt in

Algorithmus 1.10 Einfaches TPP: Query

Bestimmt zu gegebenen Last Step Shortest Path Maps $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_i$ und Anfragepunkt $q \in \mathbb{R}^2$ den kürzesten i–Pfad von s zu q.

- Bestimme die Region von q in $\mathcal{S}_i$:
  Durch prüfen der Lage (rechts oder links) von q bzgl. der konvexen Kette aus T_i und den beiden Strahlen am Rand der Durchgangsregion wird bestimmt, ob q in der Durchgangsregion oder in einer Reflektionszelle liegt. Liegt q in einer Reflektionszelle, so kann diese durch binäre Suche mit Rechts/Linkstests in den Strahlen, die von den Knoten von T_i ausgehen, bestimmt werden.
- Falls q in der Durchgangsregion liegt: es gilt $\pi_i(q) = \pi_{i-1}(q)$, bestimme rekursiv $\pi_{i-1}(q)$.
- Falls q in der Zelle zu einer Ecke v liegt: das letzte Segment des kürzesten i–Pfades von s zu q ist $\overline{vq}$. Bestimme rekursiv $\pi_{i-1}(v)$.
- Falls q in der Zelle zu einer Kante e liegt: sei q' die Spiegelung von q an e. Berechne rekursiv $\pi_{i-1}(q')$ und ersetze in diesem Pfad das Segment von e zu q' durch das Segment von e zu q.

Algorithmus 1.11 Einfaches TPP: Konstruktion der $\mathcal{S}_i$

- Berechne $\mathcal{S}_1$ durch Reflektionen an den Knoten von P_1.
- Berechne sukzessiv $\mathcal{S}_i$ aus $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{i-1}\}$ wie folgt: Berechne zu jedem Knoten $v \in P_i$ den kürzesten $(i-1)$–Pfad $\pi_{i-1}(v)$. Schneidet das letzte Stück dieses Pfades das Innere von P_i, so liegt v in der Durchgangsregion von $\mathcal{S}_i$. Andernfalls liegt v in T_i und die beiden Reflektionen des letzten Segmentes von $\pi_{i-1}(v)$ an den zu v inzidenten Kanten bestimmen die Strahlen, die die Zelle von v begrenzen.

$O(|P_i|k \log \frac{n}{k})$, insgesamt also

$$O(\sum_{i=1}^{k} |P_i|k \log \frac{n}{k}) = O(nk \log \frac{n}{k}).$$

Mit Lemma 1.32 und Lemma 1.33 folgt:

Theorem 1.34 *(Dror, Efrat, Lubiw, Mitchell, 2003)*
Für das einfache TPP mit k Polygonen mit insgesamt n Kanten läßt sich in Zeit $O(kn \log \frac{n}{k})$ eine Suchstruktur der Größe $O(n)$ aufbauen, mit Hilfe derer der kürzeste i–Pfad zu einem Punkt $q \in \mathbb{R}^2$ in Zeit $O(k \log \frac{n}{k})$ berechnet werden kann. *[DELM03]*

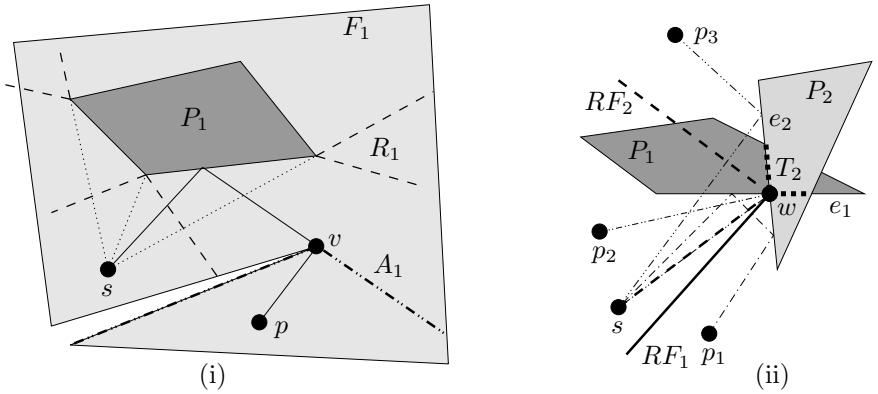


Abbildung 1.46: (i) Kürzeste Wege können an reflexen Ecken der Zäune abknicken, (ii) mögliche Kombinationen bei überlappenden Polygonen.

Im allgemeinen TPP müssen wir die Zäune F_i berücksichtigen, d. h. der kürzeste i–Pfad zu einem Punkt p muß zwischen P_j und P_{j+1} in F_j liegen. Kürzeste Pfade können nun zusätzlich an reflexen Ecken eines Zaunes abknicken, wie z. B. der Weg von s nach p in Abbildung 1.46(i). Dadurch gibt es eine weitere Klasse von Zellen in der Zerlegung der Ebene: Zellen, die einer reflexen Ecke eines Zaunes zugeordnet sind. Z. B. führen kürzeste i–Pfade aus Punkten der Zelle unterhalb der strichpunktierten Linien in Abbildung 1.46(i) über den Knoten v.

Die zweite Verallgemeinerung ist, daß sich die Polygone überlappen dürfen, wodurch weitere Arten von kombinatorisch kürzesten Wegen ins Spiel kommen. Betrachte Abbildung 1.46(ii): der Strahl RF_1 entsteht durch Punktspiegelung von $\overline{sw}$ an w und anschließender Spiegelung an den Kanten e_1 und e_2, also den Kanten von P_1 und P_2, auf denen der Schnittpunkt w liegt. RF_2 entsteht durch Reflektion von $\overline{sw}$ an e_2. Der Punkt p_1 liegt unterhalb des Strahles RF_1 und der kürzeste Weg von s nach p_1 wird erst an P_1, danach an P_2 reflektiert. p_2 liegt zwischen RF_1 und RF_2 und sein kürzester Weg verläuft direkt über w. Schließlich liegt p_3 überhalb von RF_2 und in der Durchgangsregion von P_1 und der kürzeste Weg wird lediglich durch eine Reflektion an P_2 gebildet.

Wie bereits im einfachen Fall sei T_i der Reflektionsbereich, also der Bereich von P_i, in dem die letzten Kanten eingehender i–Pfade reflektiert werden. Da sich die P_j überlappen können, ist T_i hier nicht mehr unbedingt ein Teil des Randes von P_i sondern kann auch im Inneren von P_i liegen, siehe z. B. T_2 (gepunktet) in Abbildung 1.46(ii). R_i bezeichne wie oben die Menge der Strahlen, die von T_i ausgehen. Berücksichtigen wir auch den Zaun Z_j, so müssen wir folgendes beachten: Die Strahlen R_j, die an einer Ecke des Zaunes abknicken, um eine sonst unerreichbare Region zu durchqueren nennen wir A_j, siehe Abbildung 1.46. Wir können folgendes für das allgemeine TPP festhalten (ohne Beweis):

Lemma 1.35
(i) Für ein Polygon P_i ist der Reflektionsbereich T_i ein Baum.
(ii) Der Stern R_i besteht aus disjunkten Strahlen außerhalb der Durchgangsregion (ohne Berücksichtigung der F_i) und jeder Punkt $p \in \mathbb{R}^2$ wird von einem eindeutigen Strahl getroffen.
(iii) Ein lokal optimaler Pfad ist eindeutig und lokale Optimalität garantiert globale Optimalität.

Wie beim einfachen TPP können wir nun sukzessive die T_i, R_i und A_i sowie die zugehörigen Last Step Shortest Path Maps berechnen. Wir wollen hier nur das Ergebnis festhalten:

Theorem 1.36 *(Dror, Efrat, Lubiw, Mitchell, 2003)*
Für das allgemeine TPP mit k Polygonen und $k+1$ Zäunen mit insgesamt n Kanten können wir in Zeit $O(k^2 n \log n)$ eine Suchstruktur der Größe $O(kn)$ aufbauen, mit Hilfe derer der kürzeste i–Pfad zu einem Punkt $q \in \mathbb{R}^2$ in Zeit $O(i \log n + m)$ berechnet werden kann, wobei m die Anzahl der Segmente des Pfades angibt. *[DELM03]*

Kommen wir nun zurück zu unserem eigentlichen Problem, die Shortest Watchman Route zu berechnen.

Theorem 1.37 *(Dror, Efrat, Lubiw, Mitchell, 2003)*
In einem einfachen Polygon mit n Ecken und Startpunkt s läßt sich die SWR in Zeit $O(n^3 \log n)$ berechnen. *[DELM03]*

Beweis. Aus der Eingabe (P, s) zum SWR können wir wie folgt eine Eingabe gegeben durch $(P_1, \ldots, P_k, F_1, \ldots, F_k, s', t)$ für das allgemeine TPP berechnen: Start– und Zielpunkt des TPP–Pfades ist der Startpunkt s der SWR. Als Polygone P_i wählen wir die Taschen P_{c_i} von P, siehe Definition 1.25, als Zäune legen wir P fest ($F_i := P$). Für die Laufzeit des TPP ist nur die Komplexität der Fassade der P_i entscheidend, und diese besteht in unserem Fall nur aus 2 Ecken. Somit liegt die Komplexität aller Fassaden in $O(n)$. Die Komplexität aller Zäune liegt in $O(n^2)$ und wir erhalten eine Laufzeit von $O(n^3 \log n)$. □

1.3 Kürzeste Pfade im Raum

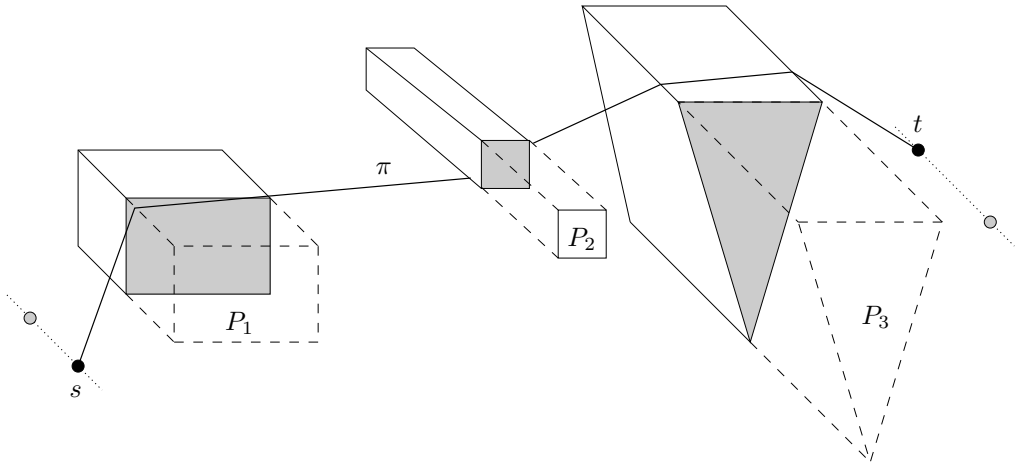


Abbildung 1.47: Kürzester Pfad im 3D.

Verlassen wir nun die Ebene und betrachten kürzeste Pfade im dreidimensionalem Raum: Gegeben seien h Polyeder im Raum mit insgesamt n Ecken, ein Startpunkt s und ein Endpunkt t. Gesucht ist der kürzeste Pfad π von s nach t, der außerhalb der Polyeder verläuft.

Eine erste (ernüchternde) Erkenntnis ist, daß dieses Problem nicht mehr diskret ist, da kürzeste Pfade über innere Punkte von Kanten verlaufen können. Wir haben zwei Teilprobleme zu lösen:

1. Bestimme die Folge der Kanten bzw. Ecken, die von einer Kürzesten berührt werden. (kombinatorisches Problem)

2. Bestimme die Berührpunkte auf den Kanten. (algebraisch-numerisches Problem)

Leider ist bereits das erste Teilproblem NP-hard:

Theorem 1.38 *(Canny, Reif, 1987)*
Die Bestimmung der Kantenfolge einer Kürzesten im $\mathbb{R}^3$ ist NP-hard. Bereits die Frage, ob es zu gegebenem k eine Kantenfolge gibt, die einen Weg π mit $|\pi| \leq k$ liefert, ist NP-hard. [CR87]

Beweis.

Zeige: 3–SAT ist auf die Kantenbestimmung reduzierbar. Dazu bilden wir in polynomieller Zeit den booleschen Ausdruck $\alpha = \bigwedge_{i=1}^{m} \left(\Pi_{i1} \vee \Pi_{i2} \vee \Pi_{i3}\right)$ mit $\Pi_{ij} \in \{X_k, \overline{X_k}\}, 1 \leq k \leq n$, auf einen Parcours P_α im Raum mit Startpunkt s und Endpunkt t ab. Die Geodätischen von s nach t in P_α zerfallen dabei in Klassen G_w, die den 2^n möglichen Wahrheitswertbelegungen w der Variablen X_i entsprechen. Die Klasse einer Geodätischen läßt sich aus der Folge der von ihr besuchten Kanten in P_α ablesen. P_α wird so konstruiert, daß für eine Kürzeste π von s nach t, die in einer Klasse G_w liegt, gilt: entweder die Belegung w erfüllt den Ausdruck α, oder α ist unerfüllbar.

Abbildung 1.48 zeigt die wesentliche Idee: die vom Startpunkt s ausgehende Geodätische wird n mal verdoppelt, so daß für jede der 2^n möglichen Variablenbelegungen ein Pfad existiert. Diejenigen Pfade, die die Klausel nicht erfüllen, werden verlängert — in der Abbildung durch $\sim\sim\sim$ angedeutet — und schließlich wieder zu einer Geodätischen zusammengefaßt, wobei darauf zu achten ist, daß die Konstruktion des Parcours in polynomieller Zeit möglich sein soll!

Der Parcours besteht im wesentlichen aus Platten der Dicke ε mit Schlitzen, durch die die Geodätischen verlaufen können. Legt man mehrere Platten in geeigneter Weise mit einem

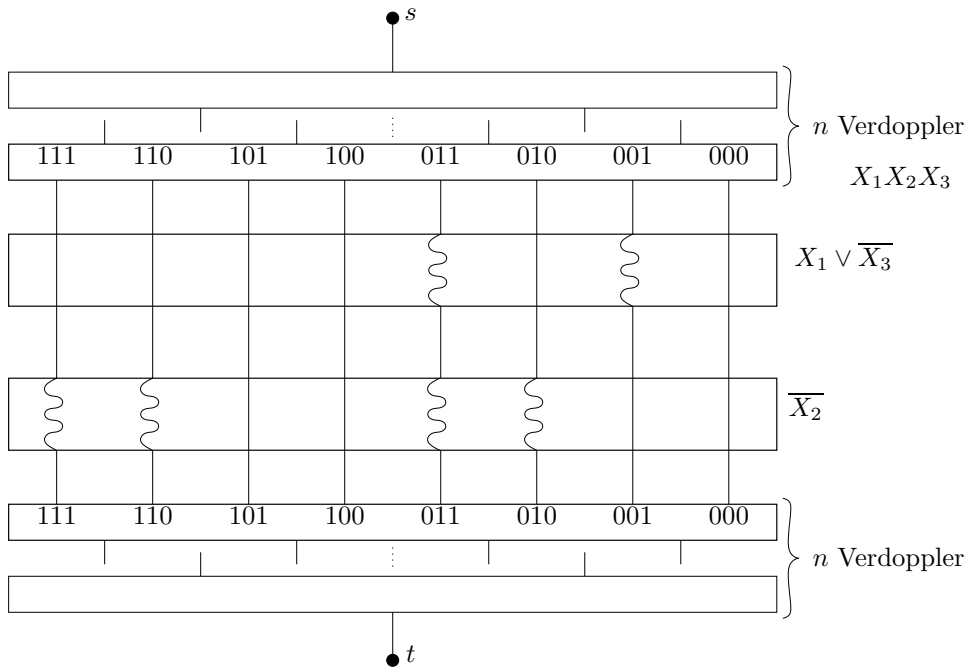


Abbildung 1.48: Beispiel: Filter für Klausel $(X_1 \vee X_3) \wedge \overline{X_2}$.

Abstand ε übereinander, läßt sich der Verlauf der Geodätischen beeinflussen. Zur Konstruktion eines solchen Parcours benötigen wir die im folgenden erklärten Bausteine. In jedem Baustein — bis auf die Barrieren im Literalfilter — werden alle Klassen von Geodätischen um den gleichen Betrag verlängert.

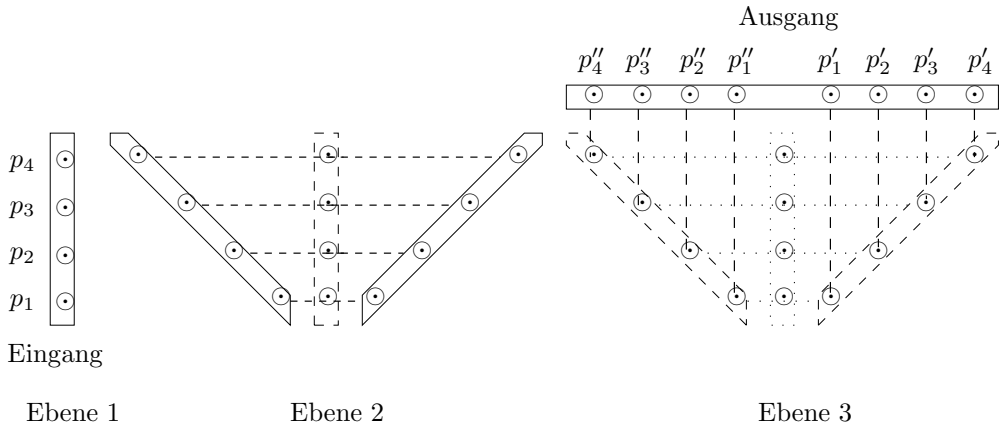


Abbildung 1.49: Die drei Ebenen eines Verdopplers.

Verdoppler

Ein Verdoppler dient dazu, seine aus n Klassen von Geodätischen bestehende Eingabe in $2n$ Klassen aufzuspalten. Er besteht aus drei Ebenen: die unterste Ebene enthält einen vertikalen Schlitz, durch den die Eingabe in den Verdoppler geführt wird. Die Eingabe verzweigt nach rechts und links und wird über zwei diagonale Schlitze in der zweiten Ebene zu einem horizontalen Schlitz in der dritten Ebene geführt, siehe Abbildung 1.49. Das Zeichen $\odot$ symbolisiert in den Abbildungen eine Geodätische, die von einer unteren Ebene in eine obere Ebene läuft.

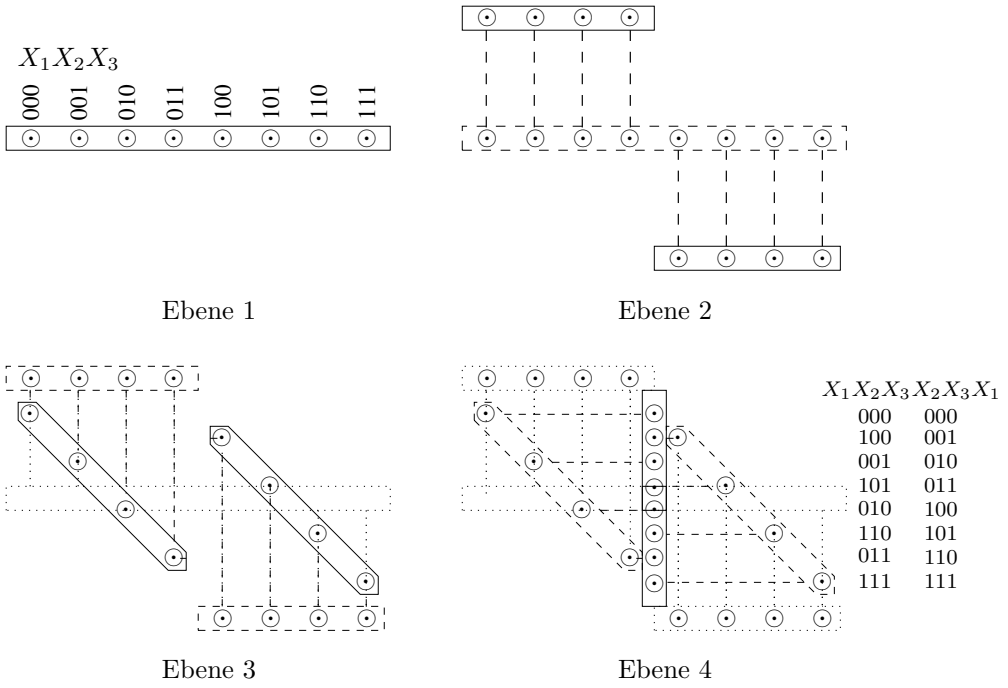


Abbildung 1.50: Die vier Ebenen eines Mischers.

Mischer

Ein Mischer besteht aus vier Ebenen. Die Eingabe wird durch einen horizontalen Schlitz in der untersten Ebene geführt, in zwei Hälften geteilt und durch zwei horizontale Schlitze in der zweiten Ebene geführt, vgl. Abbildung 1.50. In der dritten Ebene liegen zwei Reihen mit diagonalen Schlitzen, die die gesplittete Eingabe in die Vertikale umlenken, wo sie durch eine Reihe vertikaler Schlitze in der vierten Ebene ausgegeben wird. Die Schlitze in den Diagonalen sind dabei so versetzt, daß sich jeweils eine Klasse der ersten Hälfte und eine Klasse der zweiten Hälfte abwechseln. Die Eingabe $X_iX_jX_k\ldots$ wird derart permutiert, daß die Klassen mit $X_j = 1$ in einer Hälfte liegen, die Klassen mit $X_j = 0$ in der anderen Hälfte. Anders ausgedrückt ergibt sich eine Linksrotation der Variablen, X_j wird zum signifikantesten Bit.

Literalfilter

Eine Klasse von Geodätischen kodiert eine Binärzahl $w = (w_1, \ldots, w_n) \in \{0,1\}^n$, die einer Belegung der Variablen entspricht. Der Literalfilter dient nun dazu, die Klassen von Geodätischen zu verlängern, deren Belegung das Literal nicht erfüllt. Um also alle Pfade herauszufiltern, bei denen $w_i = 0$ den Ausdruck erfüllt, müssen alle Pfade mit $w_i = 1$ mit einer Barriere verlängert werden. Dabei dürfen wir die Barrieren allerdings nicht in die ursprüngliche Kodierung einfügen, wie dies in Abbildung 1.48 gezeigt ist, da dies evtl. zu exponentiell vielen Barrieren führen würde; um z. B. w_n zu filtern, wäre eine Barriere in jedem zweiten Pfad nötig! Also setzen wir Mischer ein, um die zu filternde Variable zur signifikantesten Variable zu machen, und können dann mit *einer* Barriere auf der rechten bzw. linken Seite alle Pfade mit $w_i = 0$ bzw. $w_i = 1$ verlängern, siehe Abbildung 1.51. Da ein Mischer eine Linksrotation der Variablen bewirkt, ist w_i nach $i-1$ Mischerdurchläufen das signifikanteste Bit. Anschließend benötigen wir noch $n-i+1$ Mischer, um wieder die ursprüngliche Kodierung zu erhalten.

Klauselfilter

Der Klauselfilter dient schließlich zur Umsetzung einer Klausel der Form $C_i = \Pi_{i1} \vee \Pi_{i2} \vee \Pi_{i3}$ mit $\Pi_{ij} \in \{X_k, \overline{X_k}\}, 1 \leq k \leq n$. Dazu benötigen wir drei "parallel geschaltete" Literalfilter:

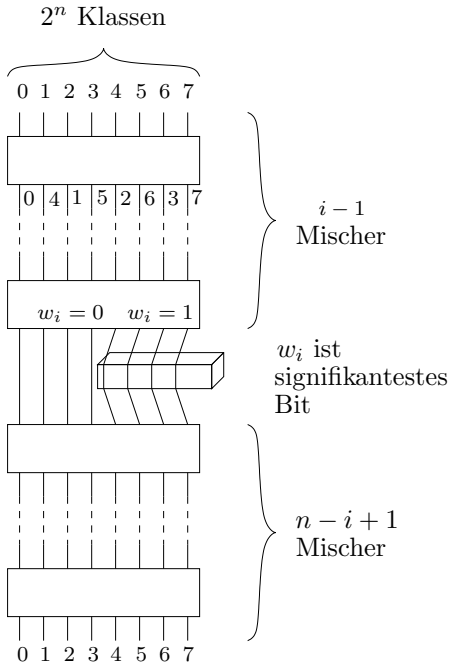


Abbildung 1.51: Literalfilter.

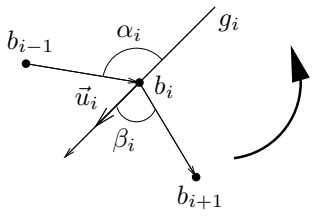
die Eingabe von 2^n Klassen von Geodätischen wird allen Literalfiltern zugeführt. In den Literalfiltern werden die Geodätischen verlängert, die die Klausel nicht erfüllen. Durch die Parallelschaltung ergibt sich eine Disjunktion der Literale, siehe Abbildung 1.52.

Zur Konstruktion eines Parcours aus einer 3–SAT Klauselmenge

$$\alpha = \bigwedge_{i=1}^{m} C_i$$

über n Variablen benötigen wir zunächst einen Block aus n Verdopplern, um aus der vom Startpunkt ausgehenden Geodätischen 2^n Klassen für jede mögliche Variablenbelegung zu erzeugen. Diese Klassen werden durch m hintereinander geschaltete Klauselfilter und schließlich durch n invers benutzte Verdoppler geführt, die die gefilterten Klassen von Geodätischen wieder zu einer zusammenfassen, die zum Zielpunkt t führt. □

Damit bleibt das zweite Teilproblem: Wie bestimmt man zu gegebener Kantenfolge die richtigen Berührpunkte, also den kürzesten Pfad, der die Kanten in dieser Reihenfolge besucht?



Schon das einfachere Problem, zu Geraden g_i im Raum und fest vorgegebener Besuchsreihenfolge die Berührpunkte des Pfades so zu verschieben, daß seine Länge minimal wird, ist schwierig. Ein lokales Optimalitätskriterium erhält man durch Hochklappen der durch g_i und b_{i+1} gegebenen Ebene in die durch g_i und b_{i-1} gegebene Ebene: liegen b_{i-1}, g_i und b_{i+1} in einer Ebene, muß der kürzeste Pfad zwischen b_{i-1} und b_{i+1} ein Liniensegment durch b_i sein. Dies gilt genau dann, wenn die beiden Winkel α_i und β_i gleich sind. Wären beide Winkel nicht gleich, so ließe sich der Pfad zwischen b_{i-1} und b_{i+1} verkürzen.

Sei $\vec{u}_i$ ein Vektor in Richtung g_i mit $|\vec{u}_i| = 1$, dann gilt:[13]

$$\alpha_i = \beta_i \iff \frac{\overrightarrow{b_{i-1}b_i} \cdot \vec{u}_i}{|\overrightarrow{b_{i-1}b_i}|} = \frac{\overrightarrow{b_ib_{i+1}} \cdot \vec{u}_i}{|\overrightarrow{b_ib_{i+1}}|} \quad (*).$$

[13] $\vec{a} \cdot \vec{b}$ bezeichnet dabei das Skalarprodukt zweier Vektoren mit eingeschlossenem Winkel γ: $\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos\gamma$.

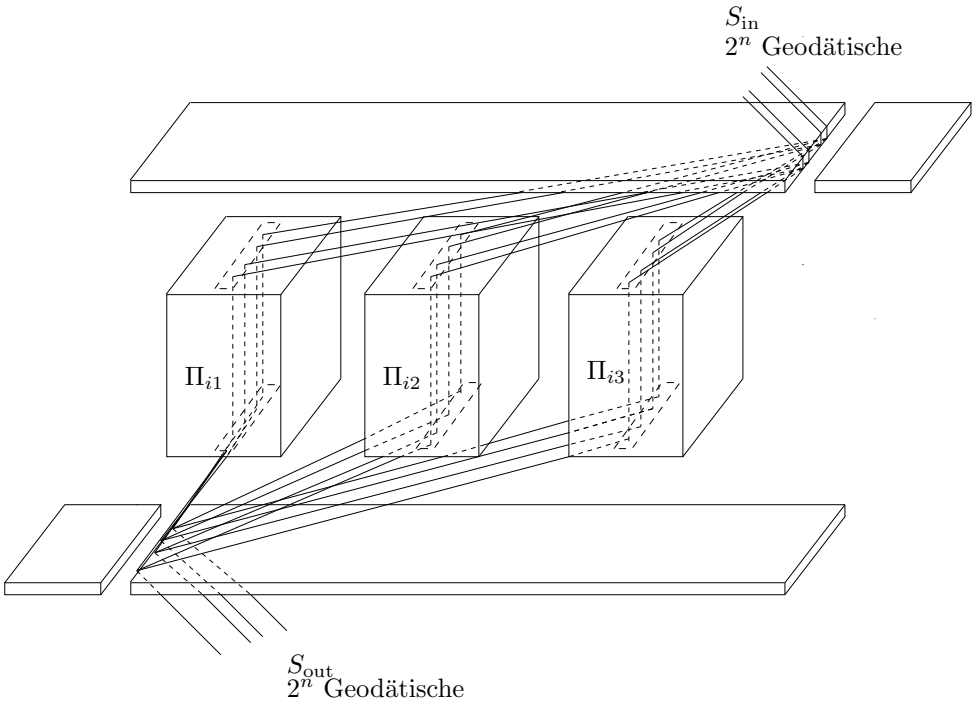
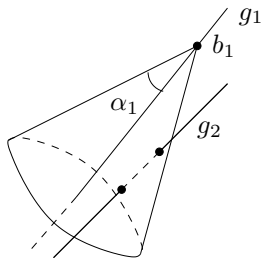


Abbildung 1.52: Klauselfilter.

Die Gleichung $(*)$ muß für jede Gerade g_i mit $1 \leq i \leq n$ gelten. Ein naiver Ansatz wäre nun, den ersten Berührpunkt b_1 hinter s zu raten und mit der Winkelbedingung die Fortsetzung des Pfades zu bestimmen. Das Problem dabei ist, daß g_2 den α_1–Kegel um g_1 zweimal schneiden kann (siehe Abbildung), die Fortsetzung des Pfades ist also nicht eindeutig, es bleiben 2^{n-1} Möglichkeiten.

Der Versuch, die Gleichungen $(*)$ mit algebraischen Methoden exakt zu lösen, führt zu Weglängen, die durch algebraische Zahlen[14] mit exponentiell in n wachsendem Grad dargestellt werden. [Baj85]



Das Problem zu entscheiden, ob ein Pfad einer Länge $\leq k$ in einer Umgebung mit n Ecken existiert, kann in Zeit $2^{n^{O(1)}}$ und Platz $n^{O(\log n)}$ Platz gelöst werden. [RS94]

Es gibt polynomiale $(1+\varepsilon)$–Approximationsschemata[15] in Zeit $O(n^2 \log^c n \cdot \frac{1}{\varepsilon^4})$. [Cla87]

[14] Eine algebraische Zahl ist eine komplexe Zahl, die Nullstelle eines Polynoms $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ mit rationalen oder ganzzahligen Koeffizienten ist. Nicht algebraisch sind z. B. π oder e.

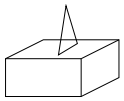
[15] D. h. der kürzeste Weg läßt sich beliebig genau approximieren, je kleiner das ε, desto genauer die Approximation, aber desto höher die Rechenzeit.

1.4 Kürzeste Pfade auf der Oberfläche eines Polyeders

Da sich die Bestimmung kürzester Pfade im Raum im allgemeinen als sehr schwierig erwiesen hat, betrachten wir den Spezialfall, bei dem sowohl s als auch t auf der Oberfläche desselben Polyeders mit n Kanten im Raum liegen. Auch der kürzeste Weg soll auf der Oberfläche des Polyeders verlaufen und nicht frei durch den Raum, auch wenn die "Luftlinie" kürzer wäre. Wir interessieren uns also z. B. für den Weg, den ein Wanderer vom Gipfel eines Berges zum nächsten Gipfel nehmen würde: ohne weitere Hilfsmittel muß er durch das Tal zwischen beiden Gipfeln wandern.

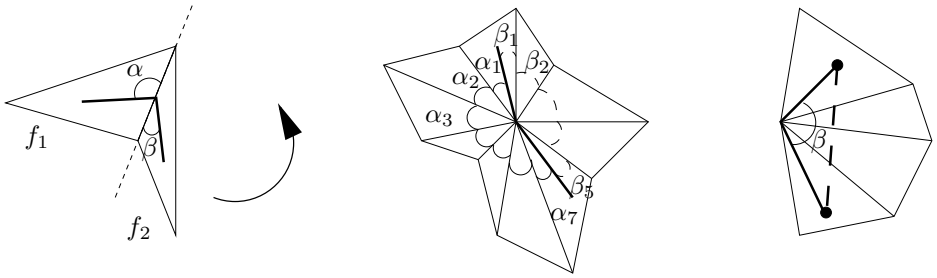
Ein Polyeder habe dabei folgende Eigenschaften:

- Es ist eine kompakte, zusammenhängende Teilmenge des $\mathbb{R}^3$.



- Der Rand besteht aus Polygonen.
- In jeder Umgebung eines Randpunktes liegen sowohl innere als auch äußere Punkte. Obige Abbildung zeigt also *keinen* Polyeder in unserem Sinn.

Weiterhin nehmen wir an, die Oberfläche des Polyeders sei bereits trianguliert;[16] aus der Euler–Formel ergibt sich, daß die Anzahl der Dreiecke in $O(n)$ liegt.



optimal $\Leftrightarrow \alpha = \beta$ (i) optimal $\Leftrightarrow \alpha, \beta \geq \pi$ (ii) (iii)

Abbildung 1.53: Eigenschaften von Kürzesten auf Polyedern.

Halten wir zunächst einige Eigenschaften von kürzesten Pfaden auf Polyedern fest:

Eine lokale Eigenschaft ist, daß in Abbildung 1.53(i) der Pfad genau dann optimal ist, wenn $\alpha = \beta$ gilt. Dreht man nämlich eine der Flächen um ihre gemeinsame Kante in die von der anderen Fläche aufgespannte Ebene, so daß die Flächen rechts und links von der Kante liegen und nicht übereinander, so muß der kürzeste Pfad eine Gerade in der Ebene sein. Dies ist mit der Bedingung $\alpha = \beta$ äquivalent.

In Abbildung 1.53(ii) verläuft die Kürzeste (fett gezeichnet) durch einen Eckpunkt. Dies ist dann und nur dann möglich, wenn sowohl die Summe der Winkel $\alpha = \sum \alpha_i$ als auch die Summe der Winkel $\beta = \sum \beta_i$ größer gleich[17] π ist. Wäre z. B. $\beta < \pi$, wie in der Abwicklung in Abbildung 1.53(iii) gezeigt, kann die fett gezeichnete Linie nicht die Kürzeste sein, da die gestrichelte Linie kürzer ist. Da zudem gilt: p konvex[18] $\Rightarrow \alpha + \beta < 2\pi$ haben wir folgendes gezeigt:

[16] Ansonsten ließe sich das Polyeder in Zeit $O(n)$ triangulieren.

[17] An dieser Stelle ist natürlich die Zahl $\pi = 3,1415\ldots$ gemeint, nicht der Pfad π.

[18] Eine Ecke v im $\mathbb{R}^3$ ist konvex, wenn eine Ebene existiert an der das Polyeder 'durchgeschnitten' werden kann, so daß der v enthaltende Teil des Polyeders ein konvexes Polyeder ist.

Lemma 1.39 *Eine Geodätische π auf einem Polyeder hat folgende lokale Eigenschaften:*

(i) An jeder von π überschrittenen Polyederkante müssen die Scheitelwinkel gleich sein.
(ii) π kann nicht über eine konvexe Ecke von P führen.

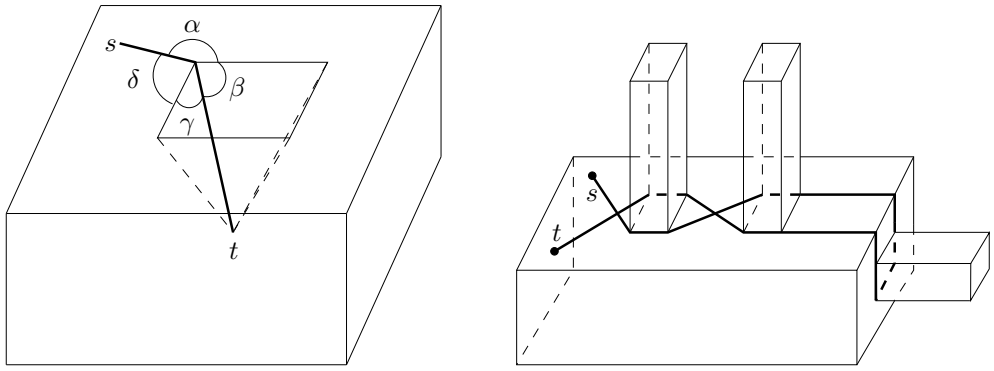


Abbildung 1.54: (i) Kürzeste über nichtkonvexe Ecke, (ii) Geodätische von s nach t, aber keine Kürzeste.

Wie das Beispiel in Abbildung 1.54(i) zeigt, kann eine Kürzeste sehr wohl über nichtkonvexe Ecken laufen.

Neben den lokalen Eigenschaften können wir auch einige globale Eigenschaften festhalten, die jedoch *nicht* für Geodätische gelten. Abbildung 1.54(ii) zeigt eine Geodätische, für die alle drei Eigenschaften nicht gelten![19]

Lemma 1.40 *Kürzeste Wege π_i, π_j von a nach b_i, b_j auf einem Polyeder haben folgende globale Eigenschaften:*

(i) π_i ist einfach, d. h. π_i enthält keine Selbstschnitte.
(ii) π_i schneidet jede Fläche des Polyeders höchstens einmal.
(iii) π_i und π_j können sich nicht im Inneren einer Fläche kreuzen.

Beweis.(Skizze) Bei allen Eigenschaften kann man sich leicht klar machen, daß π_i verkürzbar wäre, wenn eine Eigenschaft verletzt wird. □

Die erste Arbeit, die sich mit diesem Problem befaßte, lieferte dieses Ergebnis:

Theorem 1.41 *(Sharir, Schorr, 1984)*
Sei P ein konvexes Polyeder, $a \in P$ fest (o. E. eine Ecke von P) und $b \in P$ gegeben. Die Entfernung bzw. die Kürzeste von a nach b in P läßt sich nach Vorbereitungszeit $O(n^3 \log n)$ in Zeit $O(n)$ und Platz $O(n^2)$ berechnen. *[SS84, SS86]*

Im folgenden werden wir uns mit einem anderen Ansatz befassen, der das Resultat von Theorem 1.41 verbessert, insbesondere auf beliebige Polyeder erweitert hat. Dazu untersuchen wir zunächst, wie die Kandidaten für Kürzeste von a nach b aussehen: Es sind Kantenzüge π auf P, die durch nichtkonvexe Ecken von P laufen und Folgen von Kanten von P schneiden, vgl. Abbildung 1.55:

$$\pi = \underbrace{v_0}_{=a}, \underbrace{e_{1,1}, \ldots, e_{1,m_1}}_{\text{Kanten}}, \underbrace{v_1}_{\text{2. Ecke}}, \underbrace{e_{2,1}, \ldots, e_{2,m_2}}_{\text{Kanten}}, v_2, \ldots, \underbrace{v_{k-1}}_{\text{letzte Ecke}}, e_{k,1}, \ldots, e_{k,m_k}, \underbrace{v_k}_{=b}$$

Aus Lemma 1.40 folgt, daß π einfach ist und jede Kante e_{ij} nur einmal kreuzt.

[19] Zur Erinnerung: eine Geodätische ist lokal unverkürzbar, eine Kürzeste auch global.

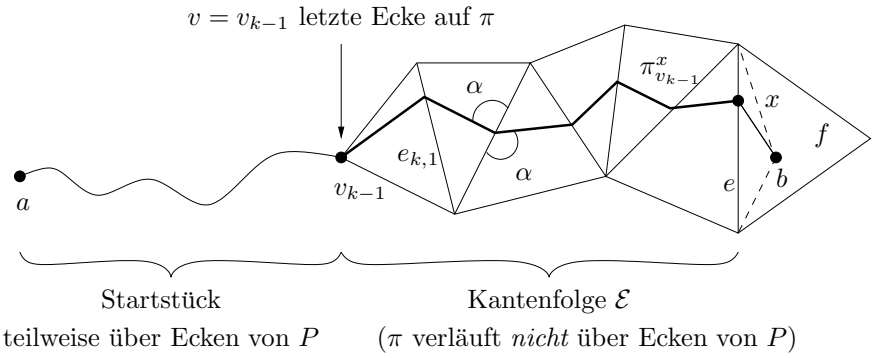


Abbildung 1.55: Pfad π von a nach b über $x \in e$.

Eine erste Idee ist nun, zunächst nur die Punkte auf den Rändern der Dreiecke des Polyeders zu betrachten und zu einer Kante e eines Dreiecks f für alle $x \in e$ den kürzesten Weg π_a^x von a nach x mit $\pi_a^x \cap \overset{\circ}{f} = \emptyset$ (d. h. π_a^x verläuft außerhalb von f) zu berechnen. Hier wären jedoch überabzählbar viele Wege zu berechnen! Abhilfe schafft die Idee, Regionen von Punkten mit kombinatorisch gleichen Kürzesten zum Punkt x zu bilden. Es genügt, die Kantenfolge $\mathcal{E} = e_{k,1}, \ldots, e_{k,m_k}$ die der Weg ab der "letzten Ecke" $v = v_{k-1}$ durchlaufen hat zu betrachten, da wir nur *eine* Kürzeste berechnen wollen. Die verschiedenen Kürzesten von a nach v brauchen nicht unterschieden zu werden, wir müssen uns nur eine solche merken.

Wegen der Winkelbedingung in Lemma 1.39(i) lassen sich v und alle Dreiecke ab v, die der Weg π_v^x durchquert, in die f–Ebene aufklappen.[20] Dabei wird π_v^x zu einem Liniensegment zwischen a und x. Im folgenden bezeichne $\bar{v}$ das in die f–Ebene aufgeklappte Bild des Punktes v.

Fassen wir also alle $x \in e$ zusammen, für die ein kürzester Weg von a über v und $\mathcal{E}$ den Punkt x erreicht:

Definition 1.42 Sei e eine fest gewählte Kante des Dreiecks f, v eine beliebige Ecke von P und $\mathcal{E} = (e_1, \ldots, e_m)$ eine beliebige Folge von Kanten von P. Das Optimalitätsintervall von v und $\mathcal{E}$ bzgl. e und f ist definiert als

$$I(v, \mathcal{E}) := \Big\{ x \in e \mid \exists \text{ Kürzeste } \delta \text{ von } a \text{ nach } x \text{ mit}$$
$$\bullet\ \delta \cap \overset{\circ}{f} = \emptyset$$
$$\bullet\ \delta \text{ endet mit } v, e_1, \ldots, e_m, x \Big\} .$$

Lemma 1.43 *Eigenschaften von $I(v, \mathcal{E})$:*
(i) Jede solche Menge I ist ein Intervall auf e (evtl. leer).
(ii) Zwei verschiedene Intervalle können sich nicht überlappen.
(iii) e wird von Intervallen $I(v, \mathcal{E})$ ganz überdeckt.

Beweis.

(i) Zu zeigen: $I(v, \mathcal{E})$ ist zusammenhängend. Seien $x_1, x_2 \in e$ in $I(v, \mathcal{E})$ enthalten, und ein Punkt x' liege zwischen x_1 und x_2 auf e, vgl. Abbildung 1.56. Zeige: $x' \in I(v, \mathcal{E})$.
Angenommen, $x' \notin I(v, \mathcal{E})$, dann klappe die Dreiecke entlang der Kantenfolge $\mathcal{E}$ in die Fläche von f auf
$\Rightarrow v' \neq v$, sogar liegt v' außerhalb des Dreiecks $\triangle(v, x_1, x_2)$
$\Rightarrow$ die beiden Kürzesten von a nach x_1/x_2 und x' kreuzen sich im Inneren einer Fläche $\lightning$.

[20] D. h. eines von je zwei Dreiecken wird solange um die gemeinsame Kante gedreht, bis beide in einer Ebene, eins rechts und eins links der Kante liegen.

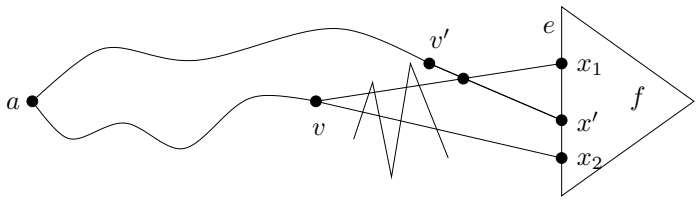


Abbildung 1.56: $I(v, \mathcal{E})$ muß zusammenhängend sein.

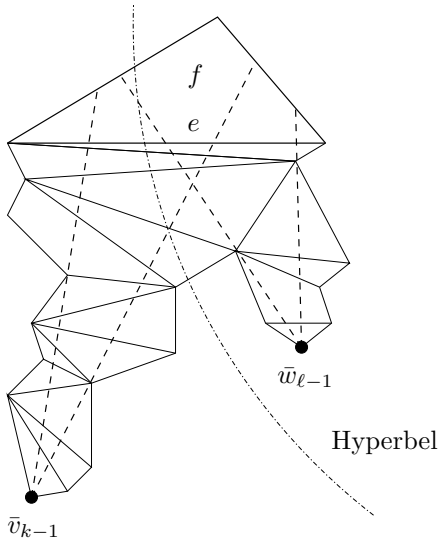


Abbildung 1.57: Der Bisektor von $\bar{v}_{k-1}$ und $\bar{w}_{\ell-1}$ ist eine Hyperbel.

(ii) Angenommen, zwei Intervalle überlappen sich. Sei x im Durchschnitt der Intervalle, dann ergibt sich beim Weg von a nach x über $\bar{v}_{k-1}$ eine Weglänge von $|\pi_a^{v_{k-1}}| + |\bar{v}_{k-1} - x|$, beim Weg über $\bar{w}_{\ell-1}$ eine von $|\pi_a^{w_{\ell-1}}| + |\bar{w}_{\ell-1} - x|$.
Für jeden Punkt x im Durchschnitt der beiden Intervalle müssen beide Weglängen gleich sein, also muß gelten:

$$|\pi_a^{v_{k-1}}| + |\bar{v}_{k-1} - x| = |\pi_a^{w_{\ell-1}}| + |\bar{w}_{\ell-1} - x|.$$

Durch Umformen dieser Gleichung erhält man die Gleichung einer Hyperbel, die den Bisektor zwischen $\bar{v}_{k-1}$ und $\bar{w}_{\ell-1}$ beschreibt, also die Menge der Punkte, für die beide Alternativen gleich sind. Diese Hyperbel hat mit e jedoch nur einen Schnittpunkt, nämlich den Punkt, an dem sich beide Intervalle berühren. Zwei Intervalle können sich also nicht überlappen.

(iii) Dies ist klar, da es für jedes $x \in e$ eine Kürzeste von a gibt.

□

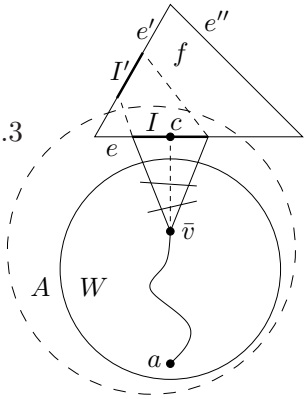
Achtung: Es kann mehrere Intervalle $I(v_i, \mathcal{E}_j)$ mit derselben "letzten" Ecke v_i geben, d. h. v_i kann mehrere unterschiedlich abgewickelte Kopien $\bar{v}_{i,k}$ haben.

Algorithmus 1.12 Continuous Dijkstra zur Berechnung der $I(v, \mathcal{E})$

Datenstruktur: Prioritätswarteschlange W von Intervallen mit bisher bekannten Kürzesten von a.

Iterationsschritt:

- Wähle das Intervall $I = I(v, \mathcal{E})$ aus W, für das $\mathrm{d}(I) = \mathrm{d}(a, v) + |\bar{v} - c|$ minimal ist, wobei c der zu $\bar{v}$ nächstgelegene Punkt auf I ist.
- Propagiere dieses Intervall I durch das Dreieck auf die beiden gegenüberliegenden Kanten e' und e'': Berechne das Intervall I' auf Kante e' mit $I' = \{x \in e' \mid$ der Weg von a über v und I durch das Dreieck f ist kürzer als der kürzeste bisher bekannte Weg zu dem Intervall $I'(v', \mathcal{E}')$ auf Kante e', das x enthält $\}$ und füge dies in die Intervalliste von e' ein; analog mit e''.



Im ersten Schritt werden also zu jeder Fläche f die Intervalle $I(v, \mathcal{E})$ aller drei Kanten von f berechnet. Dazu benutzen wir eine Methode, die als Continuous Dijkstra bekannt ist. Allgemein wird bei dieser Methode wie in Algorithmus 1.3 (Dijkstra) auf Seite 14 eine Welle mit einem Ausläufer verwaltet, die sich vom Startpunkt ausbreitet. Erreicht die Welle einen Punkt x zum ersten Mal, wird x beschriftet, und die Welle breitet sich von x weiter aus und propagiert dabei den Wert von x. In unserem Fall genügen eine endliche Zahl solcher Wellenausbreitungen. Die Welle enthält dabei alle Intervalle mit bekannten Kürzesten, der Ausläufer die Intervalle mit vorläufigen Kürzesten. Propagiert wird das "nächstgelegene" Intervall auf die anderen Dreiecksseiten, dort konkurriert es mit den vorhandenen vorläufigen Intervallen, siehe Algorithmus 1.12, I' ist dabei wirklich nur *ein* Intervall!
Für jede Kante e werden die Intervalle I in einem balancierten Blattsuchbaum gespeichert. Dies sind maximal $O(n)$ Intervalle je Kante. Um ein neues Intervall I' der Kante e' hinzuzufügen, benötigen wir zunächst $O(\log n)$ Zeit um ein Intervall J zu finden, für das es Punkte in J gibt, so dass der kürzeste Weg über $I(v, \mathcal{E})$ kürzer ist als der bisherige Weg zu diesen Punkten. Von diesem Intervall aus suchen wir alle Nachbarintervalle, die sich ändern bzw. verschwinden. Das können $O(k)$ viele sein. Weil wir einen Blattsuchbaum verwenden benötigen wir dafür $O(k)$ Schritte und für das Entfernen jeweils $O(\log n)$ Zeit. Am Ende wird ein neues Intervall eingefügt.

Der Dijkstra-Algorithmus wählt stets das Intervall mit dem momentan kürzesten Weg zu einem Intervall aus, und dieses Intervall wird nie mehr verschwinden. Pro Schritt werden nun maximal 2 neue Intervalle eingefügt. Insgesamt kann es nur $O(n^2)$ Intervalle geben, also kann der Dijkstra insgesamt nicht mehr als $O(n^2)$ Intervalle erzeugen. Das Einfügen und Entfernen von Intervallen wie oben beschrieben benötigt also insgesamt $O(n^2 \log n)$ Zeit. Das Ablaufen der Intervalllisten der Blattsuchbäume benötigt nicht mehr als $O(n^2)$ Zeit. Alle Blattsuchbäume zusammen haben einen Speicherplatzbedarf von $O(n^2)$.

Im zweiten Schritt werden die durch diese Intervalle vorhandenen Weginformationen in das Innere der Dreiecke fortgesetzt. Die Intervalle auf den drei Kanten von f — genauer: ihre "Quellen" $\bar{v}_i$ — teilen sich das Innere von f nach einer gewichteten Distanzfunktion auf: Setze $a_i := \mathrm{d}(a, v_i)$. Ein Punkt x auf der Dreiecksfläche wird dem Punkt $\bar{v}_i$ zugeordnet, über den der Weg von a nach x minimal wird, für den also gilt:

$$a_i + |\bar{v}_{i,k} - x| < \min_{\substack{j \neq i \\ \vee \ell \neq k}} \left\{ a_j + |\bar{v}_{j,\ell} - x| \right\} .$$

Die $\bar{v}_{i,k}$ teilen also die Fläche f untereinander auf. Die Grenzen der von den $\bar{v}_{i,k}$ definierten

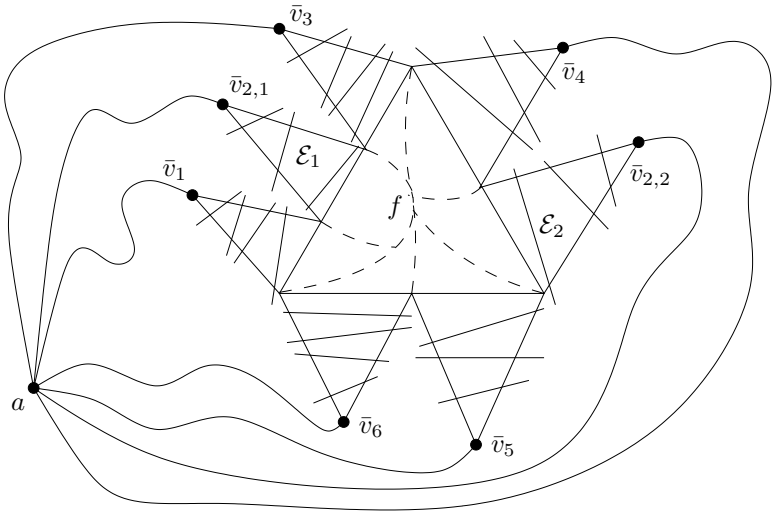


Abbildung 1.58: Dreiecksfläche f mit Intervallen $I(v, \mathcal{E})$.

Regionen von f, die Bisektoren, sind Hyperbelbögen, da die Punkte auf dem Bisektor zwischen $\bar{v}_{i,k}$ und $\bar{v}_{j,\ell}$ gleiche Weglängen haben müssen; durch die Gleichung $a_i + |\bar{v}_{i,k} - x| = a_j + |\bar{v}_{j,\ell} - x|$ wird eine Hyperbel beschrieben.

Diese Unterteilung der Fläche f entspricht einem Voronoi–Diagramm der Ecken $\bar{v}_{i,k}$ mit additiven Gewichten a_i, siehe Anhang. Dies kann für ℓ Intervalle auf den Kanten von f in Zeit $O(\ell \log \ell)$ und Speicherplatz $O(\ell)$ konstruiert werden. Doch wieviele nicht–leere Intervalle kann es auf einer Kante geben?

Lemma 1.44 *Die Anzahl ℓ der nicht–leeren Intervalle auf einer Kante e ist in $O(n)$.*

Beweis.

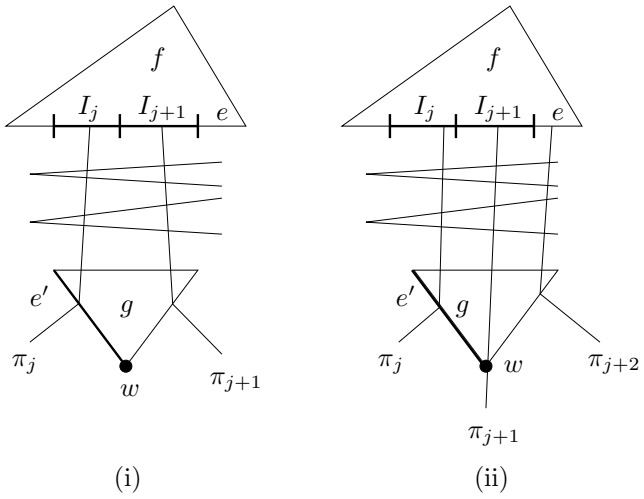


Abbildung 1.59: Ein Dreieck g mit Ecke w fungiert höchstens zweimal als Trenner für zwei Intervalle.

Seien $I_j = I_j(v_j, \mathcal{E}_j)$ und $I_{j+1} = I_{j+1}(v_{j+1}, \mathcal{E}_{j+1})$ zwei benachbarte Intervalle auf e. Wähle je einen inneren Punkt pro Intervall und verfolge die definierenden Kürzesten π_j und π_{j+1}, die zu I_j und I_{j+1} gehören, rückwärts. π_j und π_{j+1} können zunächst dieselben Kanten im Inneren kreuzen, d. h. $\mathcal{E}_j$ und $\mathcal{E}_{j+1}$ können gemeinsame Endstücke haben, müssen sich jedoch wegen $(v_j, \mathcal{E}_j) \neq (v_{j+1}, \mathcal{E}_{j+1})$ einmal trennen. Betrachte das Dreieck g und die Ecke w, an der sich die Trennung vollzieht: das Paar (g, w) fungiert höchstens zweimal als Trenner, siehe Abbildung 1.59(ii), und

jedem Paar läßt sich eindeutig eine Kante e' zuordnen, die g berandet und im Uhrzeigersinn w verläßt. Wir haben also eine Abbildung $(I_j, I_{j+1}) \longmapsto (g, w)$, die "fast injektiv" ist — jedes (g, w) hat höchstens 2 Urbilder — und eine injektive Abbildung $(g, w) \longmapsto e'$. Also gilt $\ell \leq 2n \in O(n)$. □

Der Beweis zu Lemma 1.44 benutzt die häufig angewandte Technik, die Kosten für eine Berechnung den geometrischen Objekten, hier den Kanten, zuzuordnen.

Algorithmus 1.13 Berechnung kürzester Wege auf Polyedern

Input: Polyeder P mit n Ecken, Oberfläche sei trianguliert, Startecke a.

Output: Für beliebigen Anfragepunkt $b \in \partial P$: Entfernung und kürzesten Weg von a nach b.

Vorbereitungsphase:

- Berechne für jede Dreieckskante die Intervalle $I(v, \mathcal{E})$ mit Continuous Dijkstra. $O(n^2 \log n)$
- Berechne für jede Dreiecksfläche das durch die Intervalle $I(v, \mathcal{E})$ der drei Kanten bestimmte Voronoi–Diagramm mit additiven Gewichten (siehe Anhang A.3.4). $O(n^2 \log n)$
- Bereite die Voronoi–Diagramme auf den Flächen für Lokalisierung von Punkten vor (z. B.: Trapezzerlegung, [Sei91]). $O(n^2)$, Platz $O(n^2)$

Query für Zielpunkt b in Dreieck f:

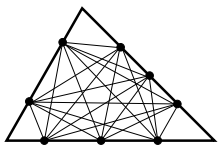
- Bestimme die Voronoi–Region in f, die den Punkt b enthält
  $\Longrightarrow$ Abstand zu a ist bekannt. $O(\log n)$
- Setze die Kürzeste von a nach b zusammen aus:
  - einer Kürzesten von a nach v (Nebenprodukt des Continuous Dijkstra: in v verankerte, verkettete Liste)
  - dem "geraden" Weg durch die Kantenfolge e zum Punkt $x \in I$
  - dem Liniensegment $\overline{xb}$.

  Bei k Segmenten auf dem kürzesten Weg: $O(k)$

Zusammenfassend erhalten wir Algorithmus 1.13 und folgendes Theorem:

Theorem 1.45 *(Mitchell, Mount, Papadimitriou, 1986)*
Sei $s \in P$ fest, o. E. eine Ecke von P, gegeben $b \in P$. Die Entfernung bzw. die Kürzeste von a nach b in P läßt sich nach Vorbereitungszeit $O(n^2 \log n)$ in Zeit $O(\log n + k)$ und Platz $O(n^2)$ berechnen; dabei ist k die Anzahl der Segmente auf dem kürzesten Pfad. *[MMP87]*

Ein praxistauglicher Ansatz zur $(1+\varepsilon)$–Approximation kürzester Wege auf Polyedern stammt von J. Sack.[21] Hierbei werden in die Kanten der Dreiecke geeignet viele und adäquat verteilte Stützpunkte eingefügt und daraus ein vollständiger Graph gebildet, auf dem der Algorithmus von Dijkstra angewendet wird. Dies liefert gute Approximationsergebnisse und erlaubt die Einhaltung von Nebenbedingungen wie maximale Steigung, maximaler Böschungswinkel, z. B. bei der Berechnung von Wegen für Fahrzeuge. [ALMS98]



[21] `http://www.scs.carleton.ca/~sack/`

1.5 Kürzeste Pfade für Liniensegmente

Bisher hatten wir nur kürzeste Pfade für Punkte betrachtet. Zum Abschluß des ersten Kapitels wollen wir kurz auf die Probleme eingehen, die bei der Berechnung kürzester Pfade für ausgedehnte Objekte entstehen. Schon für Liniensegmente im $\mathbb{R}^2$ ohne Hindernisse ist das Problem nicht trivial. Die Bewegung von Liniensegmenten im $\mathbb{R}^2$ kann man als Modell für einen Balken oder eine Leiter ansehen, die von zwei Trägern an je einem Ende getragen wird.

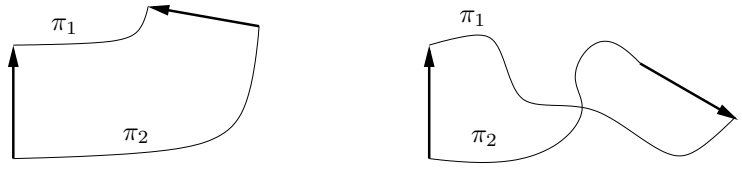


Abbildung 1.60: Bewegung eines Liniensegmentes.

Bei ausgedehnten Objekten ist zunächst einmal festzulegen, was eine Kürzeste ist bzw. wie sich die Kosten für eine Bewegung berechnen. In der Anschauung mit dem Balken sind sicherlich die Wege interessant, die die beiden Träger zurücklegen.[22] Eine Möglichkeit ist also, die Längen der von den Endpunkten zurückgelegten Bahnen zu summieren und diese Summe zu minimieren. Für Liniensegmente gilt damit für eine Bewegung B:

$$\text{Kosten}(B) := \text{ Länge}(\pi_1) + \text{ Länge}(\pi_2).$$

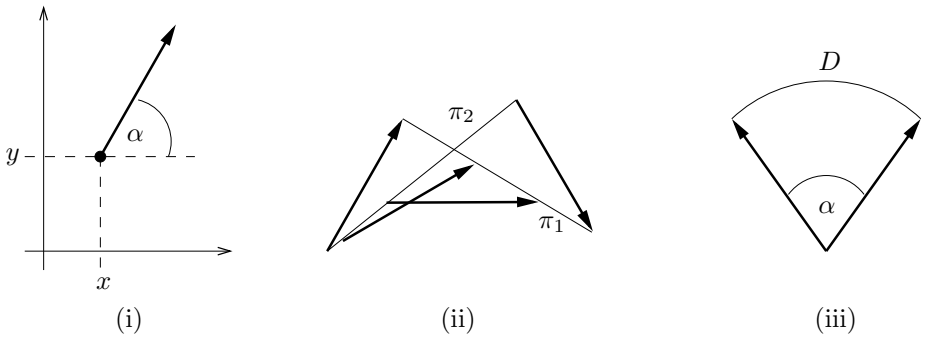


Abbildung 1.61: (i) Freiheitsgrade eines Liniensegmentes, (ii) geradlinige Bewegung (iii) Rotation.

Ein Liniensegment hat drei Freiheitsgrade, siehe Abbildung 1.61(i): die Koordinaten seines Startpunktes und sein Drehwinkel; seine Position läßt sich also durch das Tripel $A = (x, y, \alpha) \in \mathcal{C}$ mit $\mathcal{C} = \mathbb{R} \times \mathbb{R} \times [0, 2\pi)$ beschreiben.

Unser Problem läßt sich damit folgendermaßen beschreiben: gegeben zwei Positionen $A, A' \in \mathcal{C}$, bestimme eine Bewegung von A nach A' mit minimalen Kosten.

Doch welche Bewegungen sind minimal? Die geradlinige Bewegung in Abbildung 1.61(ii) ist sicherlich optimal, da π_1 und π_2 Geraden sind und nicht kürzer sein können. Auch die Rotation in Abbildung 1.61(iii) ist optimal, zum Beweis dieser vermeintlich einfachen Tatsache brauchen wir das folgende Theorem:

Theorem 1.46 *(Surface–Area Formel von Cauchy)*
Sei C eine geschlossene konvexe Kurve in der Ebene. Mit

$$h_C(\alpha) := \sup\{\, x\cos\alpha + y\sin\alpha \mid (x,y) \in C \,\},$$

$$\text{dia}_C(\alpha) := h_C(\alpha) + h_C(\alpha + \pi), \quad \alpha \in [0, \pi)$$

[22] Je schwerer der Balken ist, umso größer dürfte das Interesse der beiden Träger an kürzesten Wegen sein.

gilt für die Länge von C:

$$\textit{Länge}(C) = \int_0^\pi \mathrm{dia}_C(\alpha)\, d\alpha.$$

Dabei ist $\mathrm{dia}_C(\alpha)$ anschaulich die Messung von C mit einer um den Winkel α zur X–Achse gedrehten Schieblehre, siehe Abbildung 1.62(i).

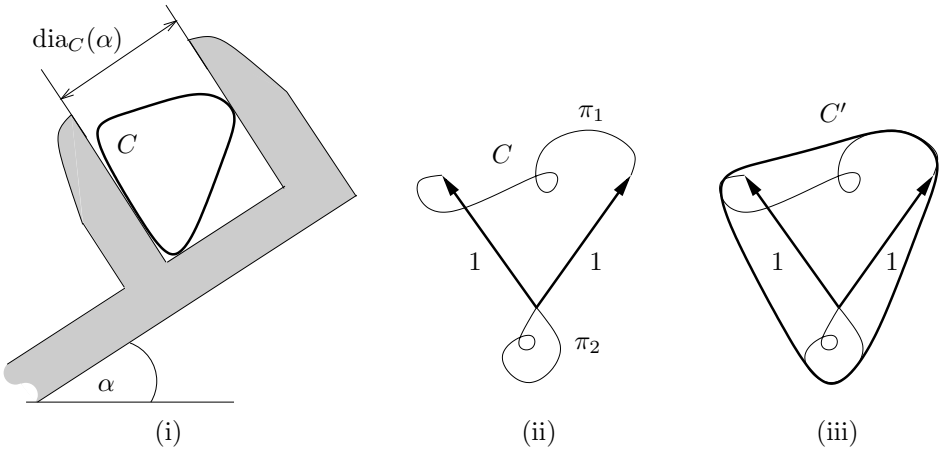


Abbildung 1.62: (i) $\mathrm{dia}_C(\alpha)$, (ii) Bewegung B, (iii) konvexe Hülle von C.

Nehmen wir nun an, eine beliebige andere Bewegung B mit Pfaden π_1 und π_2 sei optimal, vgl. Abbildung 1.62(ii). Sei C die Verknüpfung der beiden Kurven π_1 und π_2 und der beiden Liniensegmente (o. E. sei die Länge der Liniensegmente 1). C' sei die konvexe Hülle von C, siehe Abbildung 1.62(iii). D sei die Kurve der Rotation (Abbildung 1.61(iii)). Dann gilt:

$$\begin{aligned}
2 + \mathrm{Kosten}(B) &= \mathrm{Länge}(C) \\
&\geq \mathrm{Länge}(C') \\
&= \int_0^\pi \mathrm{dia}_{C'}(\alpha)\, d\alpha \quad \text{nach Theorem 1.46} \\
&\geq \int_0^\pi \mathrm{dia}_D(\alpha)\, d\alpha \quad \text{wegen } \mathrm{dia}_{C'}(\alpha) \geq \mathrm{dia}_D(\alpha)\ (*) \\
&= \mathrm{Länge}(D) \\
&= 2 + \mathrm{Kosten}(R)
\end{aligned}$$

Begründung $(*)$: Im *kritischen* Winkelbereich für α gilt $dia_D(\alpha) = 1$. Da auch bei beliebiger Bewegung C der Winkel α (oder $-\alpha$) angenommen werden muss (irgendwo muss das Segment ja liegen), gilt für alle Winkel in diesem Bereich $dia_{C'}(\alpha) \geq 1$, siehe Abbildung 1.63.

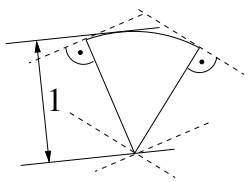


Abbildung 1.63: dia_D im *kritischen* Winkelbereich.

Also ist eine Rotation um einen Winkel $\leq \pi$ optimal.

Für sich betrachtet sind also Rotation um einen Punkt auf dem Liniensegment und Translation optimal, doch wie verhält es sich mit Kombinationen davon? Es gilt:

Theorem 1.47 *(Icking, Rote, Welzl, Yap, 1989)*
Zwischen je zwei Positionen von Liniensegmenten gibt es eine optimale Bewegung von einem der folgenden Typen:
(i) maximal drei Rotationen,
(ii) maximal zwei Rotationen und eine geradlinige Bewegung,
(iii) eine Rotation zwischen zwei geradlinigen Bewegungen.
Die Teilbewegungen lassen sich effektiv berechnen. *[IRWY93]*

Das Problem der kürzesten Pfade von Liniensegmenten wurde von Icking et. al. wiederentdeckt. Erwähnt wurde es bereits von Dubovitskij [Dub85] und Gurevich [Gur75].

Fassen wir abschließend die Ergebnisse aus Kapitel 1 zusammen:

- Kürzeste Pfade für Punkte im $\mathbb{R}^2$ lassen sich effizient berechnen.
- Kürzeste Pfade für Punkte im $\mathbb{R}^d, d > 2$ zu finden ist NP-hard.
- Kürzeste Pfade für ausgedehnte Objekte zu finden ist schwierig.

Deshalb werden wir uns im folgenden Kapitel auf Kollisionsfreiheit konzentrieren.

Kapitel 2

Kollisionsfreie Bahnen für polygonale Roboter

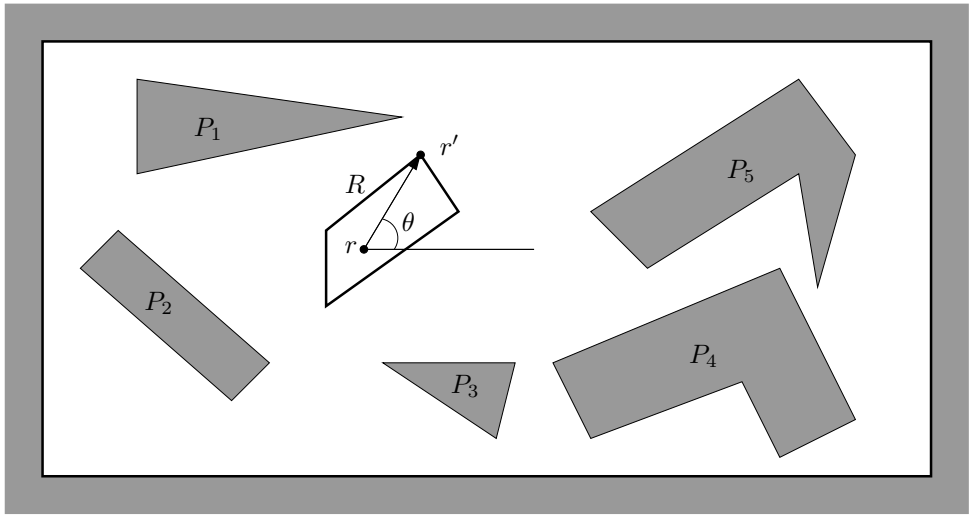


Abbildung 2.1: Roboter R in einem Arbeitsraum.

Wir betrachten nun nicht länger punktförmige Roboter, sondern ausgedehnte Objekte. Wir modellieren den Roboter R durch ein abgeschlossenes Polygon — konvex oder allgemein — mit m Kanten. Der Arbeitsraum des Roboters sei eine beschränkte Umgebung mit h polygonalen Hindernissen P_i mit insgesamt n Ecken, siehe Abbildung 2.1.

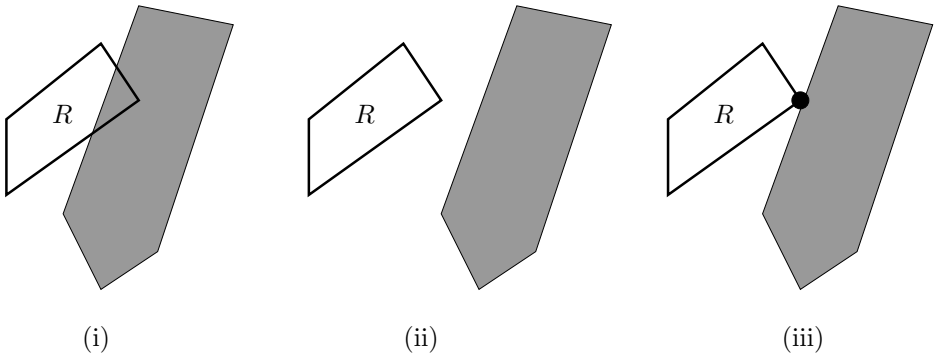


Abbildung 2.2: (i) Verbotene Konfiguration, (ii) freie Plazierung (erlaubt), (iii) halbfreie Plazierung (erlaubt).

Zur Diskussion kollisionsfreier Bahnen wollen wir einige Begriffe definieren:

Definition 2.1

(i) Eine **Plazierung** oder **Konfiguration** ist eine Position des Roboters im Arbeitsraum. Zur eindeutigen Beschreibung der Position legen wir einen Referenzpunkt $r = (x, y)$ fest, sowie einen zweiten Punkt r' als Winkelreferenz. Die Plazierung läßt sich dann durch die Angabe des Referenzpunktes r und des Winkels θ zwischen der X–Achse und dem Vektor $\overrightarrow{rr'}$ als $R(x, y, \theta)$ mit $(x, y, \theta) \in \mathbb{R} \times \mathbb{R} \times [0, 2\pi)$ angeben. Der Roboter hat damit drei Freiheitsgrade (Degree of Freedom, DOF).

(ii) Der **Konfigurationsraum** ist der Raum der möglichen Plazierungen des Roboters, hier $\mathcal{C} = \mathbb{R} \times \mathbb{R} \times [0, 2\pi)$.

(iii) Die **verbotenen Konfigurationen** sind solche, bei denen der Roboter in ein Hindernis eindringt. Das Berühren einer Hinderniswand dagegen ist erlaubt (**halbfreie Plazierung**), vgl. Abbildung 2.2.[1]

$$\mathcal{C}_{\text{verb}} := \left\{ c \in \mathcal{C} \mid R(c) \cap \bigcup \overset{\circ}{P}_i \neq \emptyset \right\}$$

(iv) Der **Raum der freien Plazierungen** enthält alle erlaubten Konfigurationen

$$\mathcal{C}_{\text{frei}} := \mathcal{C} \setminus \mathcal{C}_{\text{verb}}$$

Damit haben wir folgendes Bahnplanungsproblem: Gegeben $s, t \in \mathcal{C}_{\text{frei}}$. Gibt es einen Weg von s nach t, der ganz in $\mathcal{C}_{\text{frei}}$ verläuft? Wenn ja, bestimme einen.

Die Frage nach der Existenz eines Weges ist äquivalent zu der Frage, ob s und t in derselben Zusammenhangskomponente von $\mathcal{C}_{\text{frei}}$ liegen. Zur Erinnerung:

Definition 2.2 Eine Menge Z (Teilmenge eines topologischen Raumes T) heißt genau dann **weg–zusammenhängend**, wenn

$$\forall a, b \in Z : \exists \text{ Pfad } \pi \text{ von } a \text{ nach } b \text{ in } Z.$$

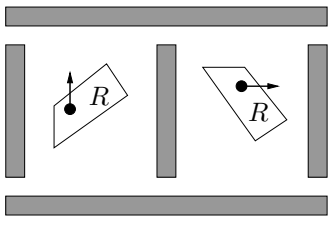
Der Pfad π ist dabei gegeben durch eine stetige Abbildung $\pi : [0, 1] \longrightarrow Z$ mit $\pi(0) = a$ und $\pi(1) = b$.

Lemma 2.3 *Sei $A \subset T$ beliebig, dann wird für alle $a, b \in A$ durch*

$$a \sim b :\Leftrightarrow \exists \textit{ Pfad } \pi \subset A \textit{ von } a \textit{ nach } b \textit{ in } A \textit{ mit } \pi(0) = a \textit{ und } \pi(1) = b$$

eine Äquivalenzrelation definiert. Die Äquivalenzklassen Z_i von A bzgl. $\sim$ heißen **Zusammenhangskomponenten** *von A. Sie sind die größten zusammenhängenden Teilmengen und überdecken A: $A = \dot{\bigcup} Z_i$.*[2]

Natürlich kann $\mathcal{C}_{\text{frei}}$ aus mehreren Zusammenhangskomponenten bestehen. In nebenstehender Abbildung kann R sich in einem der beiden Räume bewegen, $\mathcal{C}_{\text{frei}}$ hat zwei Zusammenhangskomponenten. Das Bahnplanungsproblem ist unlösbar, wenn $s \in Z_i$ und $t \in Z_j$ mit $i \neq j$.



Somit stellen sich uns folgende algorithmische Probleme:

- Bestimme die Zusammenhangskomponenten von $\mathcal{C}_{\text{frei}}$ bzw. die Komponente Z_s, die s enthält.
- Bestimme, ob t in Z_s liegt. Falls ja, berechne einen Weg $\pi \subseteq Z_s$ mit $\pi(0) = a$ und $\pi(1) = b$.

[1] Zur Erinnerung: $\overset{\circ}{P}$ bezeichnet das Innere des Polygons.

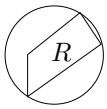
[2] $A \dot{\cup} B$ bezeichne die **disjunkte Vereinigung** $A \dot{\cup} B = A \cup B$ mit $A \cap B = \emptyset$.

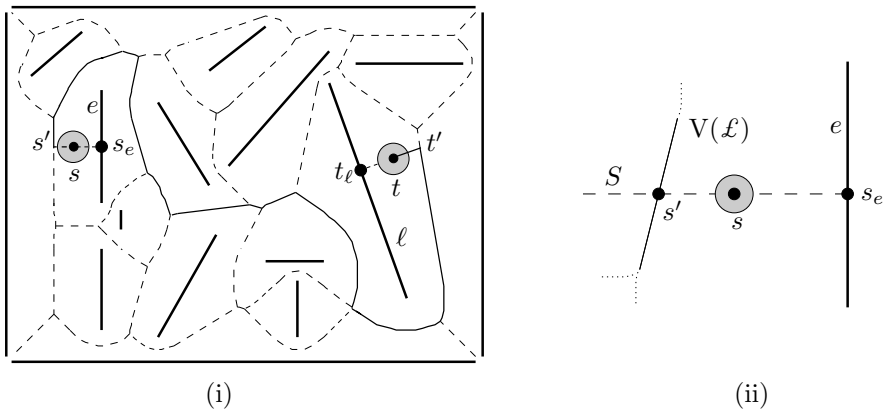
2.1 Reine Translationsbewegungen für konvexe Roboter

Beschränken wir uns zunächst auf den Spezialfall eines *konvexen* Roboters mit m Kanten, der sich in einer Umgebung aus h Polygonen P_i mit insgesamt n Kanten und $\forall i \neq j : \overset{\circ}{P}_i \cap \overset{\circ}{P}_j = \emptyset$ bewegt. Außerdem beschränken wir uns auf reine Translationsbewegungen, der Roboter darf also nur verschoben, aber nicht gedreht werden. Die Ausrichtung des Roboters ist stets gleich, damit hat der Roboter hier nur zwei Freiheitsgrade, und seine Plazierung wird durch $R(x, y) \in \mathcal{C} = \mathbb{R} \times \mathbb{R}$ eindeutig beschrieben.

2.1.1 Voronoi–Diagramme, Translationsbewegungen mit Sicherheitsabstand

Sei zunächst der Roboter R durch einen Kreis gegeben. Kreisförmige Roboter haben die Vorteile, daß man keine Rotationsbewegungen zu betrachten hat, wenn als Referenzpunkt der Kreismittelpunkt gewählt wird, und sie außerdem als Approximation für beliebige Roboter dienen können, siehe nebenstehende Abbildung. Dabei werden allerdings nicht alle Wege gefunden; man kann sich leicht Umgebungen vorstellen, in denen der Roboter R eine Lücke zwischen zwei Hindernissen passieren kann, während der kreisförmige Roboter zu groß dafür ist.





(i) (ii)

Abbildung 2.3: (i) Bahnplanung im Voronoi–Diagramm für Liniensegmente (ii) Konstruktion von s'.

Das Problem läßt sich lösen, indem wir, wie in Abschnitt A.3.3 beschrieben, das Voronoi–Diagramm von Liniensegmenten bzgl. der L_2–Metrik mit den Kanten der Hindernisse als Liniensegmente konstruieren, siehe Abbildung 2.3. Wie in Algorithmus 2.1 beschrieben reduziert sich damit die Suche nach einem kollisionsfreien Weg von s nach t auf die Suche in dem planaren Graphen $\mathrm{V}(\pounds)$ von s' nach t'. Wegen $s, t \in \mathcal{C}_{\text{frei}}$ existiert eine Bewegung von s nach s' und von t' nach t.

Mit dieser Methode erhält man Pfade mit maximalem Sicherheitsabstand. Außerdem läßt sie sich auf die Translation konvexer Roboter verallgemeinern, indem man mit dem Spiegelbild von R eine konvexe Distanzfunktion definiert und das Voronoi–Diagramm von Liniensegmenten bzgl. dieser konvexen Distanzfunktion berechnet. Nach einer Vorbereitungszeit von $O(mn \log n)$ ermöglicht dies, eine Anfrage in Zeit $O(mn)$ zu beantworten, siehe [Ma00]. Auf nicht–konvexe Roboter, Roboter mit mehreren Freiheitsgraden oder höhere Dimensionen läßt sich diese Methode nicht effektiv verallgemeinern. Das Prinzip der Reduktion auf einen eindimensionalen Graphen läßt sich jedoch verallgemeinern, wir werden später darauf zurückkommen. Der Ansatz, in einer erzeugten Karte einen Pfad zu suchen, ist als Roadmap–Approach bekannt.

Algorithmus 2.1 Bahnplanung mit Sicherheitsabstand für kreisförmige Roboter

Vorbereitung:

- Konstruiere das Voronoi–Diagramm $\mathrm{V}(\mathcal{L})$ mit den Kanten der Hindernisse als Liniensegmente. $O(n \log n)$
- Bereite das Voronoi–Diagramm für Point-Location vor. mit Trapezzerlegung $O(n \log n)$

Query für zwei Punkte s, t:

- Lokalisiere s, t im Voronoi–Diagramm. $O(\log n)$
- Sei $s \in \mathrm{VR}(e, \mathcal{L})$, $t \in \mathrm{VR}(\ell, \mathcal{L})$:
- Bestimme den zu s nächstgelegenen Punkt s_e auf e.
- Bestimme Strahl S von s_e durch s.
- s' sei Schnittpunkt von S mit $\mathrm{V}(\mathcal{L})$; analog t'.
- Berechne mit Breitensuche in $\mathrm{V}(\mathcal{L})$ einen Pfad von s' nach t' mit zulässigem Mindestabstand oder berichte, daß es keinen solchen gibt. $O(n)$

2.1.2 Berechnung von $\mathcal{C}_{\text{frei}}$

Ein anderer Ansatz zur Berechnung reiner Translationsbewegungen konvexer Roboter ist, den Raum der freien Konfigurationen $\mathcal{C}_{\text{frei}}$ explizit zu berechnen. Nehmen wir zunächst an, die Hindernisse seien konvex. Betrachten wir ein Hindernis P_i und untersuchen, welche Positionen $R(x, y)$ aufgrund der P_i verboten sind. Von Lozano-Pérez stammt die Idee, die Hindernisse so zu expandieren, daß in der entstehenden Szene das Bahnplanungsproblem für einen Punkt zu lösen ist [LP83].

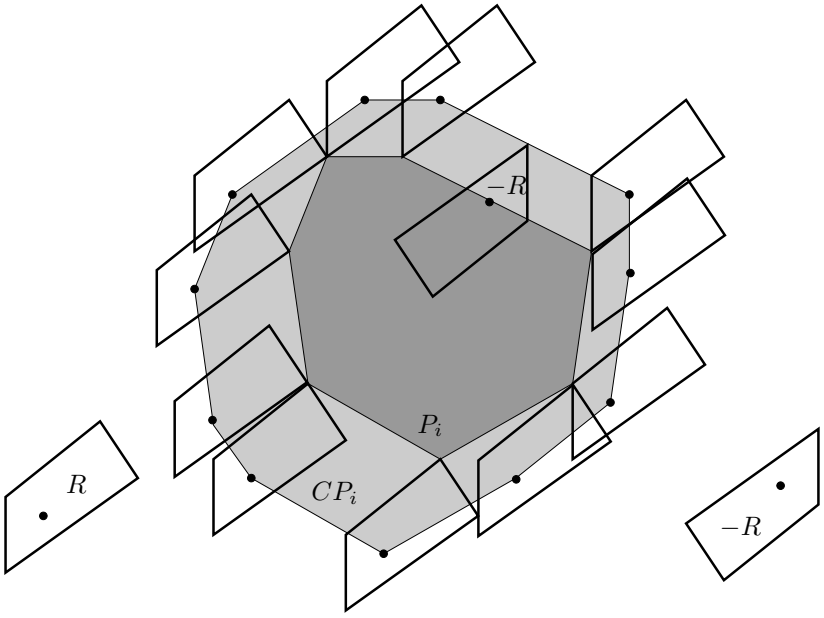


Abbildung 2.4: Hindernis, Roboter und Konfigurationsraum–Hindernis.

Das zu P_i gehörende expandierte Hindernis CP_i (Konfigurationsraum–Hindernis), ist definiert als

$$CP_i := \{c \in \mathcal{C} \mid R(c) \cap P_i \neq \emptyset\}.$$

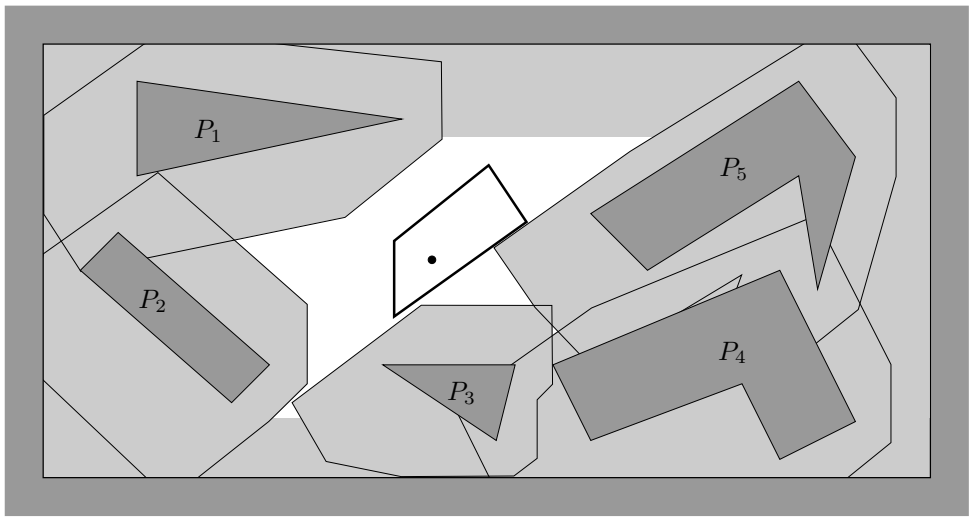


Abbildung 2.5: Konfigurationsraum–Hindernisse zu Abbildung 2.1.

Anschaulich ist der Rand von CP_i die Bahn des Referenzpunktes die entsteht, wenn der Roboter R um P_i wandert, vgl. Abbildung 2.4.[3] Abbildung 2.5 zeigt die Konfigurationsraum–Hindernisse zu Abbildung 2.1. Man beachte, daß $\mathcal{C}_{\text{frei}}$ hier aus zwei Komponenten besteht! Anstatt den Rand von R um den Rand von P_i herumzuschieben, kann man auch den Referenzpunkt des am Ursprung gespiegelten Roboters $-R = -R(0,0)$ um den Rand von P_i schieben. Das Konfigurationaraum–Hindernis zu P_i ergibt sich dann aus der Vereinigung von P_i und allen Positionen von $-R$, bei denen der Referenzpunkt auf dem Rand von P_i liegt. Dies entspricht genau der Minkowski–Summe von P_i und $-R$.

Definition 2.4 Die **Minkowski–Summe** zweier Mengen $A, B \subset \mathbb{R}^2$ ist

$$A \oplus B := \{a + b \mid a \in A, b \in B\}.$$

Bemerkung 2.5 *Minkowski–Summen sind*
(i) kommutativ: $A \oplus B = B \oplus A$,
(ii) assoziativ: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$,
(iii) distributiv bzgl. der Vereinigung: $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$.

Lemma 2.6 *Das zu P_i gehörende Konfigurationsraum–Hindernis bzgl. R ist*

$$CP_i = P_i \oplus (-R(0,0)).$$

Beweis.
Zeige: $\forall x, y : R(x,y) \cap P_i \neq \emptyset \iff (x,y) \in P_i \oplus (-R(0,0))$.
"$\Longrightarrow$" Sei $q = (q_x, q_y) \in R(x,y) \cap P_i$

$$\begin{aligned} &\Rightarrow \quad (q_x, q_y) \in R(x,y) \\ &\Rightarrow \quad (q_x - x, q_y - y) \in R(0,0) \\ &\Rightarrow \quad (-q_x + x, -q_y + y) \in -R(0,0). \end{aligned}$$

Außerdem gilt $(q_x, q_y) \in P_i$, insgesamt also

$$(x,y) = (-q_x + x, -q_y + y) + (q_x, q_y) \in -R(0,0) \oplus P_i.$$

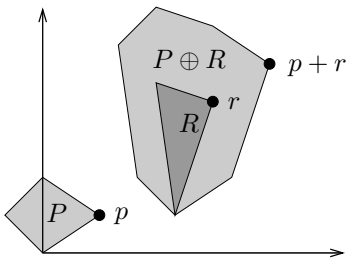
[3] In Anhang B findet sich eine Seite mit Robotern zum Ausschneiden und Nachvollziehen der in diesem Kapitel vorgestellten Beispiele.

"$\Longleftarrow$" Sei $(x,y) \in P_i \oplus -R(0,0)$
$\Rightarrow \exists (q_x,q_y) \in P_i, (r_x,r_y) \in R(0,0)$:
$(x,y) = (q_x,q_y) - (r_x,r_y) = (q_x - r_x, q_y - r_y)$
$\Rightarrow (q_x,q_y) = (x,y) + \underbrace{(r_x,r_y)}_{\in R(0,0)} \in R(x,y) \cap P_i.$ □

Minkowski–Summen treten also bei der Bewegungsplanung in natürlicher Weise auf. Betrachten wir deshalb einige Eigenschaften von Minkowski–Summen.

Definition 2.7 Gegeben sei ein Richtungsvektor d mit $|d| = 1$. Ein Punkt $p \in P$ ist **Extremalpunkt** von P in Richtung d, wenn folgendes gilt: Die zu d orthogonale Gerade g durch p teilt die Ebene in zwei Halbebenen, von denen die eine zusammen mit g ganz P enthält und die Andere den Punkt $p + d$.

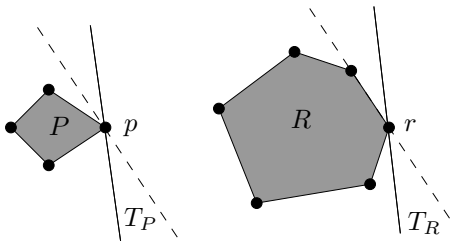
Bemerkung 2.8 *Extremalpunkte der Minkowski–Summe $P \oplus R$ sind Summen von Extremalpunkten von P und R.*



Lemma 2.9 *Seien P und R konvexe Polygone mit n bzw. m Kanten. Dann ist $P \oplus R$ konvex, hat maximal $n + m$ Kanten und kann in Zeit $O(n + m)$ berechnet werden.*

Beweis.
Die Konvexität folgt aus der Definition der Minkowski–Summe.

Rotiere simultan zwei parallele Tangenten T_P, T_R um P und R. Wenn T_P eine Ecke p von P berührt und gleichzeitig T_R eine Ecke r von R, dann ist $p + r$ eine Ecke von $P \oplus R$. Alle Ecken von $P \oplus R$ entstehen auf diese Weise. Da alle Ecken von P und R genau einmal betrachtet werden, braucht die Rotation Zeit $O(n + m)$.



□

Definition 2.10 Zwei durch Jordankurven[4] berandete Mengen $A, B \subseteq \mathbb{R}^2$ heißen ein Paar **Pseudokreise** (pseudo disks), wenn ihre Ränder entweder höchstens zwei echte Kreuzungen oder höchstens einen Berührpunkt aufweisen.

Abbildung 2.6(i) und (ii) veranschaulicht die Definition. Beispiele für Pseudokreise sind Kreise, Quadrate und achsenparallele Rechtecke mit festem Seitenverhältnis (Abbildung 2.7(i)–(iii)). Beliebige Rechtecke sind im allgemeinen keine Pseudokreise (Abbildung 2.7(iv)). Folgendes Lemma gibt eine äquivalente Charakterisierung von Pseudokreisen; die Konvexität ist dabei eine notwendige Forderung, wie Abbildung 2.6(iii) zeigt.

Lemma 2.11 *Seien A, B konvex. A, B sind genau dann Pseudokreise, wenn $A \backslash B$ und $B \backslash A$ zusammenhängend sind.*

Beweis. Übungsaufgabe. □

[4]Eine geschlossene Kurve, die die Ebene in zwei Gebiete unterteilt.

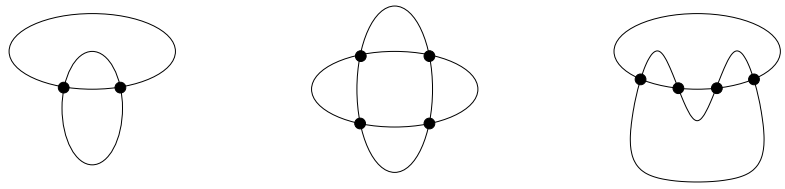


Abbildung 2.6: (i) Pseudokreise, (ii) keine Pseudokreise, (iii) die Konvexität ist eine notwendige Forderung.

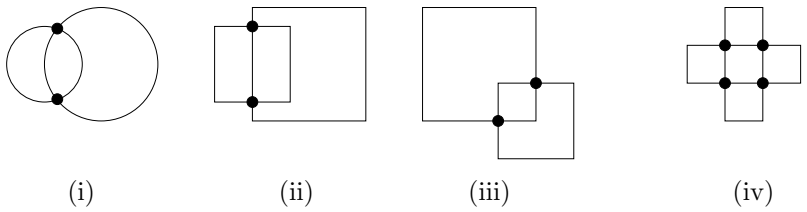


Abbildung 2.7: (i)–(iii) Beispiele für Pseudokreise, (iv) Rechtecke sind im allgemeinen keine Pseudokreise.

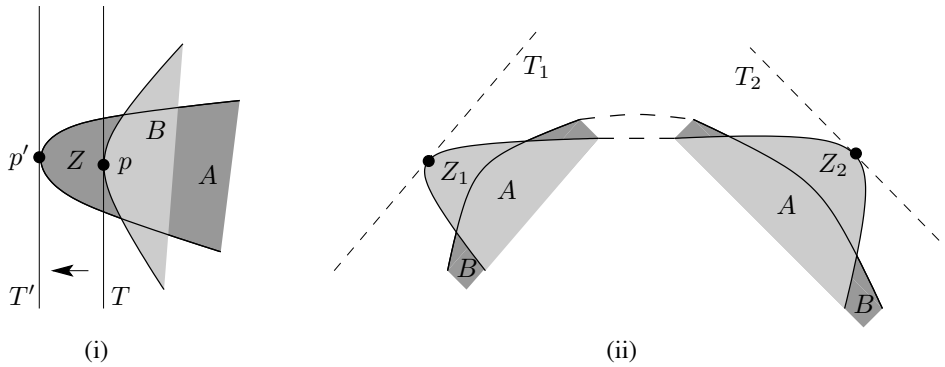


Abbildung 2.8: (i) p muß Element von Z sein, (ii) $A \cup B$ kann nicht aus zwei Zusammenhangskomponenten bestehen.

Lemma 2.12 *Seien P_1, P_2, R konvexe Polygone mit $\overset{\circ}{P}_1 \cap \overset{\circ}{P}_2 = \emptyset$. Dann sind $A = P_1 \oplus R$ und $B = P_2 \oplus R$ ein Paar konvexer Pseudokreise.* *[KLPS86]*

Beweis.
Die Konvexität folgt aus der Definition von $\oplus$. Wegen der Symmetrie genügt es zu zeigen: $A \backslash B$ ist zusammenhängend. Jede Zusammenhangskomponente Z von $A \backslash B$ trägt zum Rand der konvexen Hülle $\mathrm{ch}(A \cup B)$ bei: Falls $Z = A$ ist, also $A \cap B = \emptyset$, müssen auch Punkte von A auf $\partial\,\mathrm{ch}(A \cup B)$ liegen; sonst wäre $A \subset \mathrm{ch}(A \cup B) = \mathrm{ch}(B) = B$. Ist Z eine *echte* Teilmenge von A, so muß ∂Z einen Punkt $p \in \partial B$ enthalten. Lege die Tangente T durch p an B, dann liegt B vollständig auf einer Seite von T; siehe Abbildung 2.8(i). Verschiebe jetzt T parallel in die B entgegengesetzte Richtung, bis der letzte Randpunkt p' von A erreicht wird. Die Tangente T' durch p' an A ist auch Tangente an $A \cup B$, also gilt $p' \in \partial A \cap \partial\,\mathrm{ch}(A \cup B)$.

Nehmen wir nun an, es gäbe zwei Zusammenhangskomponenten Z_1, Z_2 in $A \backslash B$.

$\Rightarrow$ Es gibt zwei Richtungen $\vec{d_1}, \vec{d_2}$, in denen A extremer als B ist, d. h. die Tangente an $\mathrm{ch}(A \cup B)$ berührt A, siehe Abbildung 2.8(ii).

$\overset{\Rightarrow}{\text{Bem. 2.8}}$ Es gibt zwei Richtungen $\vec{d_1}, \vec{d_2}$, in denen P_1 extremer als P_2 ist.

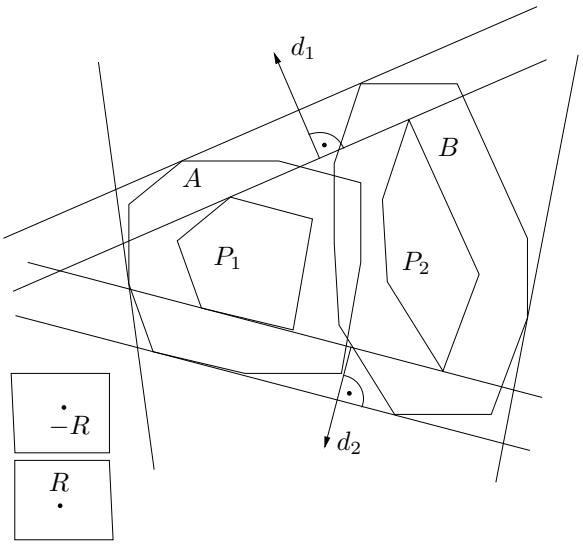


Abbildung 2.9: Um $A \cup B$ rotierende Tangenten wechseln zweimal zwischen A und B, weil P_1 und P_2 disjunkt sind.

$\Rightarrow$ Da P_1, P_2 konvex und nicht leer sind, und $\overset{\circ}{P}_1 \cap \overset{\circ}{P}_2 = \emptyset$ gilt, ist P_1 entweder in allen Richtungen zwischen $\vec{d_1}$ und $\vec{d_2}$ oder in allen Richtungen zwischen $\vec{d_2}$ und $\vec{d_1}$ extremer als P_2, siehe Abbildung 2.9.

$\overset{\Rightarrow}{\text{Bem. 2.8}}$ A ist entweder in allen Richtungen zwischen $\vec{d_1}$ und $\vec{d_2}$ oder in allen Richtungen zwischen $\vec{d_2}$ und $\vec{d_1}$ extremer als B.

$\Rightarrow$ Eine um $A \cup B$ rotierende Tangente berührt erst A, dann $A \cup B, B, A \cup B$ und schließlich wieder A, siehe Abbildung 2.9.

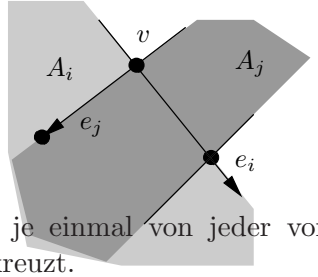
$\Rightarrow$ Es gibt nur eine Zusammenhangskomponente $\lightning$.

□

Theorem 2.13 *(Kedem, Livne, Pach, Sharir, 1986)*
Sei $A_1, \ldots, A_k$ eine Familie von polygonalen Pseudokreisen mit insgesamt n Kanten. Dann hat $\partial \bigcup A_i$ die Komplexität $O(n)$. *[KLPS86]*

Beweis.
Auf $\partial \bigcup A_i$ gibt es zwei Typen von Ecken: Ecken von Pseudokreisen — davon gibt es maximal n Stück — und Schnittpunkte von Kanten von Pseudokreisen. Sei v ein Schnittpunkt zweier Pseudokreis–Kanten e_i und e_j. Folge der Kante e_i ins Innere von A_j, dann gibt es wiederum zwei Fälle:
1. Fall: e_i endet in A_j, dann stelle v dem Endpunkt von e_i in A_j in Rechnung.

2. Fall: e_i geht durch A_j hindurch. Auf e_i liegen dann zwei Schnittpunkte von A_i und A_j. Da A_i und A_j Pseudokreise sind, kann es keinen weiteren Schnittpunkt geben. Also muß e_j im Inneren von A_i enden, und wir können v dem Endpunkt von e_j in A_i in Rechnung stellen.



Auf diese Weise wird jede Ecke w höchstens zweimal belastet, je einmal von jeder von w ausgehenden Kante, falls diese sich mit den Rändern anderer A_ℓ kreuzt. □

Theorem 2.13 gilt auch allgemeiner, ist jedoch in dieser Form viel schwieriger zu beweisen:

Theorem 2.14 *Sei $A_1, \ldots, A_k$ eine Familie von Pseudokreisen. Dann hat $\partial \bigcup A_i$ die Komplexität $O(n)$.* *[KLPS86]*

Welche Komplexität hat nun $P \oplus R$, wenn nicht beide Polygone konvex sind?

Lemma 2.15 *Seien P_1, P_2 Polygone mit n bzw. m Ecken.*
(i) Sind P_1, P_2 konvex, so ist $P_1 \oplus P_2$ konvex und hat die Komplexität $O(m+n)$.
(ii) Ist nur P_2 konvex, so hat $P_1 \oplus P_2$ die Komplexität $O(mn)$.
(iii) Ist kein P_i konvex, so hat $P_1 \oplus P_2$ die Komplexität $O(m^2n^2)$.
Alle Schranken sind scharf.

Beweis.
(i) Bereits gezeigt.
(ii) Trianguliert man P_1, so entstehen $n-2$ Dreiecke D_i mit paarweise disjunktem Inneren und $P_1 = \bigcup_{i=1}^{n-2} D_i$.

$$P_1 \oplus P_2 = \left(\bigcup_{i=1}^{n-2} D_i\right) \oplus P_2 = \bigcup_{i=1}^{n-2} \underbrace{(D_i \oplus P_2)}_{O(m) \text{ Kanten}} .$$

Dies ist nach Lemma 2.12 eine Familie von Pseudokreisen mit $O(mn)$ Kanten, also hat $P_1 \oplus P_2$ die Komplexität $O(mn)$ nach Theorem 2.13.
(iii) Trianguliere P_1 und P_2, $P_1 = \bigcup_{i=1}^{n-2} D_i$ und $P_2 = \bigcup_{j=1}^{m-2} D'_j$.

$$P_1 \oplus P_2 = \bigcup_{i=1}^{n-2} \bigcup_{j=1}^{m-2} \underbrace{D_i \oplus D'_j}_{O(1) \text{ Kanten}} .$$

Insgesamt erhalten wir $O(mn)$ Kanten von denen sich jedes Paar kreuzen kann, also $O(m^2n^2)$ viele Schnittpunkte. Theorem 2.13 ist hier nicht anwendbar, da $\bigcup D_i \oplus D'_j$, $1 \le i \le n-2, 1 \le j \le m-2$ keine Familie von Pseudokreisen ist (Übungsaufgabe).

Abbildung 2.10 zeigt Beispiele, in denen die Schranken scharf sind. □

Damit können wir folgende Aussage beweisen:

Theorem 2.16 *Sei R ein polygonaler, konvexer Roboter mit m Ecken, und seien P_i polygonale Hindernisse mit ingesamt n Ecken; dann hat $\mathcal{C}_{\text{frei}}$ die Komplexität $O(mn)$ und kann in Zeit $O(mn \log^2(mn))$ berechnet werden.*

Beweis.
Es genügt, $\mathcal{C}_{\text{verb}}$ zu berechnen. P_i habe n_i Ecken. Triangulieren wir die P_i, so entsteht eine Familie von Pseudokreisen, für die gilt:

$$\mathcal{C}_{\text{verb}} = \underbrace{\bigcup_i \underbrace{P_i \oplus -R(0,0)}_{\text{Komplexität } O(m \cdot n_i)}}_{\text{Komplexität } O(m \cdot \sum n_i) = O(mn)} .$$

$\mathcal{C}_{\text{verb}}$ kann mit dem Divide–and–Conquer Algorithmus 2.2 berechnet werden. Die Lösung der dabei entstehenden Rekursion $T(k) \le 2T(\frac{k}{2}) + Ck \log k$ mit $k = mn$ liegt in $O(k \log^2 k)$:

$$\begin{aligned}
T(k) &\le 2T(\frac{k}{2}) + Ck \log k \\
&\le 4T(\frac{k}{4}) + 2C\frac{k}{2} \log \frac{k}{2} + Ck \log k \\
&\le \cdots \\
&\le 2^{\log k} T(1) + \Sigma_{i=0}^{\log k - 1} Ck \log \frac{k}{2^i} \\
&\le Ck + \Sigma_{i=0}^{\log k - 1} Ck \log k \\
&= Ck + Ck \log^2 k \in O(k \log^2 k)
\end{aligned}$$

□

Algorithmus 2.2 Berechnung von $\mathcal{C}_{\text{verb}}$

- Unterteile die Hindernisse beliebig in gleichgroße Teilmengen $\mathcal{P}_1$ und $\mathcal{P}_2$.
- Berechne rekursiv $\mathcal{C}_{\text{verb}}(\mathcal{P}_1)$ und $\mathcal{C}_{\text{verb}}(\mathcal{P}_2)$ mit je $O(mn)$ Kanten.
- Berechne mit Sweep die Vereinigung $\mathcal{C}_{\text{verb}}(\mathcal{P}_1) \cup \mathcal{C}_{\text{verb}}(\mathcal{P}_2)$ in Zeit $O(mn \log(mn))$.

Damit haben wir das Bahnplanungsproblem für konvexe Roboter und Translationen zwischen polygonalen Hindernissen auf das Finden von Pfaden für Punkte im zweidimensionalem Raum $\mathcal{C}_{\text{frei}}$ — d. h. zwischen polygonalen Hindernissen mit Gesamtgröße $O(mn)$ — reduziert. Dies läßt sich mit Algorithmen zur Berechnung kürzester Pfade oder mit dem in Algorithmus 2.3 beschriebenen Roadmap–Verfahren lösen.

Algorithmus 2.3 Roadmap–Verfahren

Vorbereitung:

- Berechne die Trapezzerlegung von $\mathcal{C}_{\text{frei}}$ und die zugehörige Suchstruktur für Point-Location. randomisiert $O(mn \log(mn))$
- Entferne die Trapeze, die im Inneren eines Hindernisses liegen.
- Plaziere je einen Knoten im Inneren eines jeden Trapez und auf jeder vertikalen Verlängerung. $O(mn)$
- Verbinde adjazente Knoten (siehe Abbildung 2.11)
  $\rightsquigarrow$ Zusammenhangsgraph von $\mathcal{C}_{\text{frei}}$ (Roadmap).

Query:

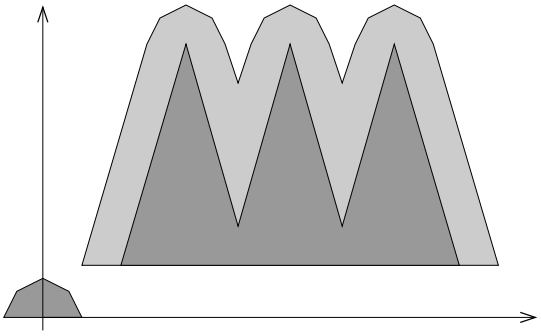
- Lokalisiere s und t in den Trapezen. $O(\log(mn))$
- Falls s und t im selben Trapez liegen: verbinde s und t.
- Ansonsten verbinde s und t mit den in den Trapezen liegenden Knoten und führe Breitensuche in der Roadmap durch. $O(mn)$

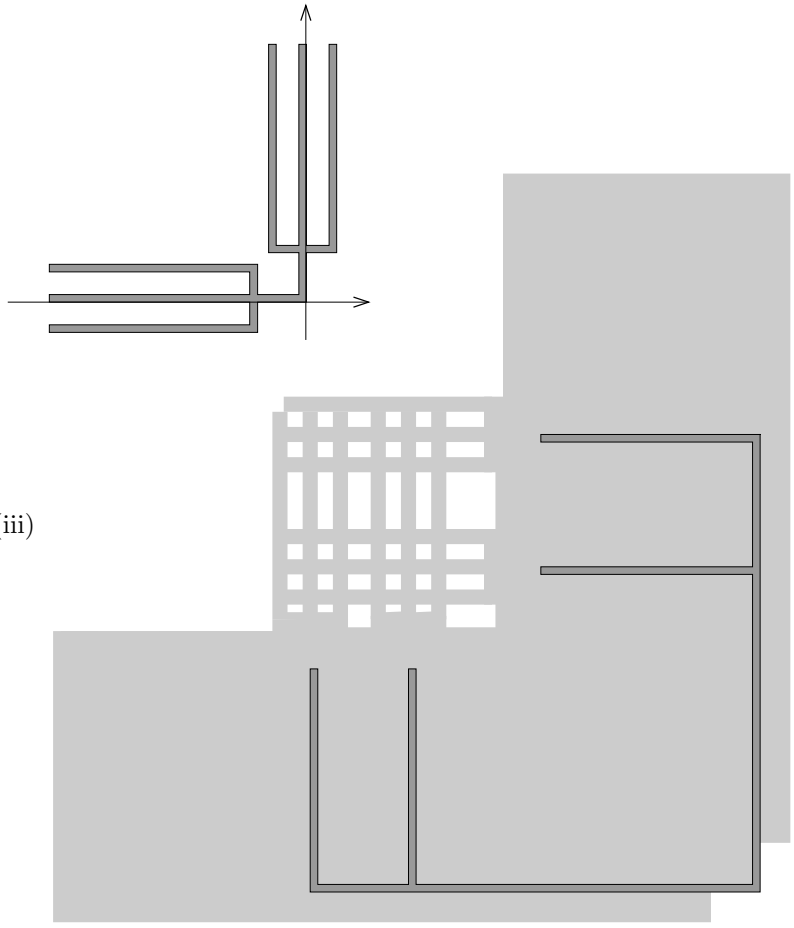
Insgesamt erhalten wir folgendes Theorem:

Theorem 2.17 *Translationsbewegungen eines konvexen, polygonalen Roboters mit m Ecken in einer Umgebung mit polygonalen Hindernissen mit insgesamt n Ecken können nach $O(mn \log^2(mn))$ (randomisiert) Vorbereitungszeit in Zeit $O(mn)$ geplant werden.*

Algorithmus 2.3 ist vergleichsweise einfach, jedoch nicht optimal.

(ii)





(iii)

Abbildung 2.10: (ii) $P_1 \oplus P_2$ hat $\Theta(mn)$ Ecken, wenn nur eines der Polygone konvex ist, (iii) $P_1 \oplus P_2$ hat $\Theta(m^2n^2)$ Ecken, wenn keines der Polygone konvex ist.

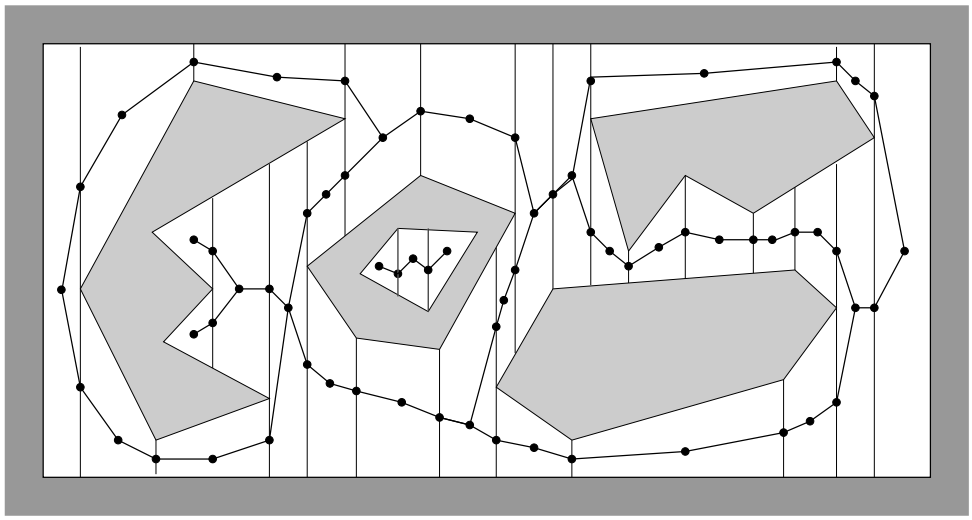


Abbildung 2.11: Konfigurationsraum–Hindernisse und Roadmap.

2.2 Reine Translationsbewegungen für beliebige Roboter

Betrachten wir nun einen allgemeinen, polygonalen Roboter mit m Ecken, der sich wie gehabt in einer Umgebung mit polygonalen Hindernissen P_i mit insgesamt n Ecken bewegt.

Eine mögliche Vorgehensweise wäre, die konvexe Hülle des Roboters zu betrachten und wie in Abschnitt 2.1 vorzugehen. Das Problem dabei ist uns schon bei der Approximation durch kreisförmige Roboter begegnet: es werden evtl. keine Bahnen gefunden, obwohl welche existieren.

Da der Roboter R nicht mehr konvex ist, hat $\mathcal{C}_{\text{frei}}$ hier nach Lemma 2.15(iii) die Komplexität $O(m^2n^2)$. Die Berechnung des gesamten Raumes $\mathcal{C}_{\text{frei}}$ wäre also sehr aufwendig. Die Idee ist nun, nur noch diejenige Zusammenhangskomponente Z_s von $\mathcal{C}_{\text{frei}}$ zu berechnen, die die Startposition s enthält. Dies genügt, denn wenn die Zielposition t nicht in Z_s liegt, ist das Bahnplanungsproblem nicht lösbar, andernfalls ist ein Pfad von s nach t in Z_s zu berechnen.

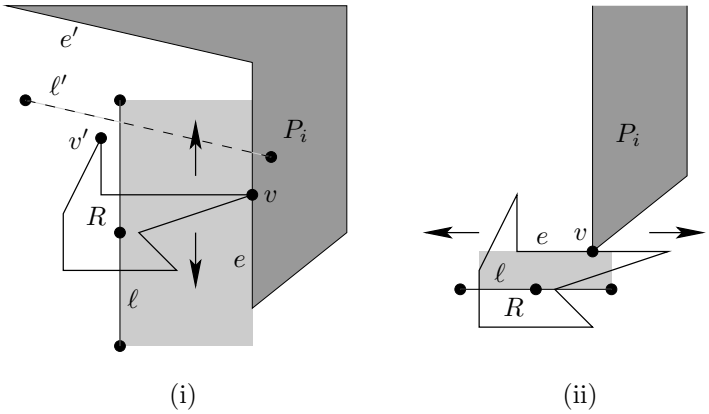


Abbildung 2.12: Paare von Ecken und Kanten von P_i und R erzeugen im Konfigurationsraum ein Liniensegment.

Untersuchen wir einmal genauer, wodurch $\mathcal{C}_{\text{frei}}$ eingeschränkt wird: Für jede Ecke v des Roboters und jede Kante e eines Hindernisses erzeugt das Ecke/Kante–Paar (v, e) im Konfigurationsraum ein Liniensegment ℓ, über das der Referenzpunkt des Roboters nicht hinweg bewegt werden darf. Dieses Liniensegment entsteht durch Bewegung der Roboterecke v entlang der Hinderniskante e. Andere Ecke/Kante–Paare werden dabei außer acht gelassen. In Abbildung 2.12(i) erzeugen die Ecke/Kante–Paare (v, e) und (v', e') auf diese Weise die Liniensegmente ℓ und ℓ'. Weiterhin erzeugt jedes Kante/Ecke–Paar (e, v) einer Kante e des Roboters und einer Hindernisecke v ein Liniensegment ℓ, siehe Abbildung 2.12(ii).

Wir brauchen daher nicht zu jedem Hindernis das Konfigurationsraumhindernis zu berechnen, sondern können jedes Ecke/Kante– und Kante/Ecke–Paar einzeln betrachten. Für jedes dieser Paare erhalten wir ein Liniensegment. Z_s ist dann eine Zelle in einem Arrangement von $2mn$ Liniensegmenten.

2.2.1 Komplexität und Berechnung einzelner Zellen

Wie kompliziert kann eine solche Zelle sein? Das folgende Theorem beantwortet die Frage etwas allgemeiner, nämlich für die Komplexität einer Zelle in einem Arrangement von Kurvenstücken ohne Selbstschnitte:

Theorem 2.18 *(Guibas, Sharir, Sifrony, 1989)*
Sei $\mathcal{A}$ ein Arrangement von n Kurvenstücken, von denen sich je zwei höchstens s mal schneiden. Dann hat jede Zelle in $\mathcal{A}$ die Komplexität $O(\lambda_{s+2}(n)) \subseteq O(n \log^* n)$. *[GSS89]*

$\lambda_s(n)$ bezeichnet die maximale Länge einer Davenport–Schinzel–Sequenz der Ordnung s über einem Alphabet mit n Buchstaben, siehe Abschnitt A.4.

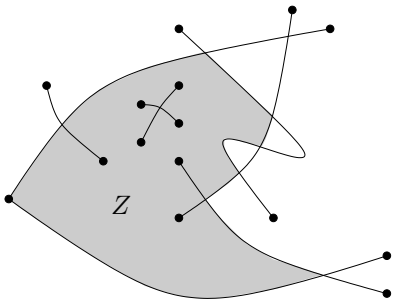


Abbildung 2.13: Zelle in Arrangement von Kurvenstücken.

Für den Beweis von Theorem 2.18 betrachten wir eine Zelle. Der Rand ∂Z von Z zerfällt in Zykel. Wir betrachten nur den äußeren Zyklus, C, und nehmen an, daß C entgegen den Uhrzeigersinn orientiert ist. Wie man sich leicht überlegen kann, reicht für die Komplexitätsbetrachtung der Zelle die Betrachtung des äußeren Zykels aus (Übungsaufgabe).

Nun orientieren wir jeden Bogen, z. B. von links nach rechts, wodurch ein positiver und negativer Durchlaufsinn festgelegt wird. Den Zyklus C aus Abbildung 2.14 können wir so rein kombinatorisch durch die zyklische Folge

$$S =< \gamma_4^+ \; \gamma_2^- \; \gamma_1^- \; \gamma_1^+ \; \gamma_2^- \; \gamma_2^+ \; \gamma_1^+ \; \gamma_7^- \; \gamma_3^- \; \gamma_6^+ \; \gamma_6^- \; \gamma_3^- \; \gamma_5^+ \; \gamma_5^- \; \gamma_3^- >$$

beschreiben.

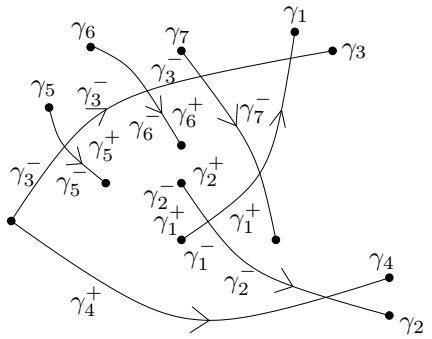


Abbildung 2.14: Die orientierten Segmente von C.

Lemma 2.19 *(Konsistenzlemma)*
Die Segmente eines orientierten Bogens γ_i^+ (γ_i^-) erscheinen in C in derselben Reihenfolge wie längs γ_i^+ (γ_i^-).

Beweis. Zur Repräsentation von γ_i^+ und γ_i^- erweitern wir γ_i zu einem Schlauch ohne die Topologie von C zu verändern. Diesen Schlauch nennen wir nun ebenfalls γ_i.

Seien ζ und η zwei Segmente auf einer Seite von γ_i, die in C konsekutiv sind, das heißt, kein Stück von γ_i taucht zwischen ζ und η entlang C auf. Unterscheiden wir je nach Verlauf der Kurve C, so können wir einige Fälle ausschließen. In Abbildung 2.15 liegt die Zelle Z auf der linken Seite von C. Da sie in η auf der Außenseite des Schlauches liegen muss, ist dieser Fall nicht möglich. Die Zelle Z läge hier auf beiden Seiten von C.

Dadurch ergeben sich zwei Fälle, siehe Abbildung 2.16.

In beiden Fällen können wir innerhalb des Schlauches eine Kurve α von $x \in \zeta$ nach $y \in \eta$ legen, die ζ und η verbindet. Das Kurvenstück C_x^y nennen wir im folgenden β. Die geschlossene Kurve $\alpha \cup \beta$ liegt im Komplement des Inneren von Z.

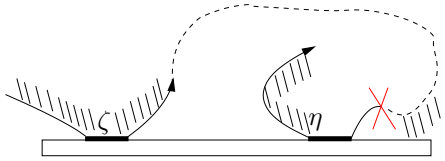


Abbildung 2.15: Die Zelle Z läge hier auf beiden Seiten von C.

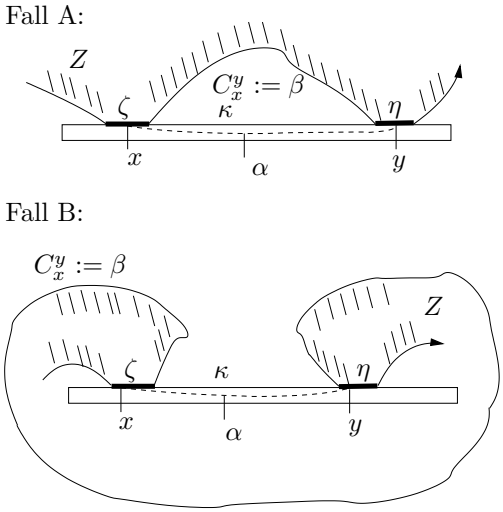


Abbildung 2.16: Kein Stück von γ_i taucht zwischen ζ und η entlang C auf.

In beiden Fällen liegt Z links von der gegen den Uhrzeigersinn orientierten, geschlossenen Kurve C. Das bedeutet, daß $\alpha \cup \beta$ die Kurve C vom Segment κ zwischen ζ und η separiert. Kein Punkt $z \in \kappa$ liegt somit auf C. □

Um die Komplexitäten der im Anhang beschriebenen Davenport–Schinzel–Sequenzen anwenden zu können, müssen wir uns von der zyklischen Betrachtung lösen und eine lineare Ordnung der betrachteten Segmente erschaffen.

Dabei kann der Fall auftreten, daß in der Sequenz s bezüglich eines Startpunktes p einige orientierte Bogensegmente nicht mehr in der Reihenfolge entlang der Segmente besucht werden, siehe zum Beispiel γ_3^+ in Abbildung 2.17, daher verdoppeln wir diese Segmente einfach. Aus γ_3^- wird $\gamma_{3.1}^-$ und $\gamma_{3.2}^-$. Es entsteht dabei eine lineare Sequenz s^* mit $4n$ Segmenten.

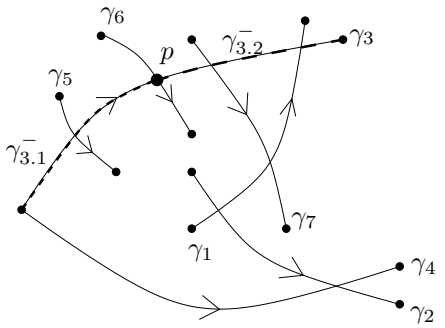


Abbildung 2.17: Das Segment γ_3^- wird in der nicht-zyklischen Sequenz nicht in der Reihenfolge gemäß der Orientierung besucht und wird deshalb am Startpunkt p verdoppelt.

Bemerkung: Es kann tatsächlich vorkommen, dass wir jedes Segement teilen müssen, siehe Abbildung 2.18.

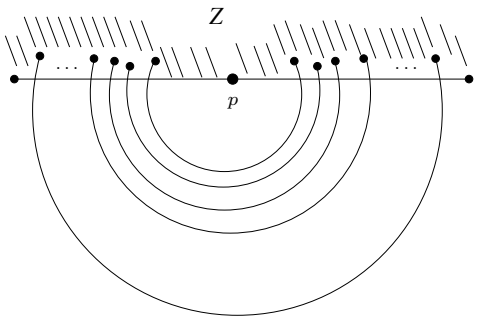


Abbildung 2.18: Wird der Startpunkt p gewählt, so müssen alle Segmente geteilt werden.

Mit diesen Vorbereitungen zeigen wir die folgende Behauptung, mit der Theorem 2.18 bewiesen wird.

Lemma 2.20 *Die Sequenz s^* ist eine $(4n, s+2)$ Davenport–Schinzel–Sequenz.*

Beweis. Es ist klar, daß nie zwei Segemente $\gamma^+_{i.j}$ ($\gamma^-_{i.j}$) hintereinander stehen, da ein Schnittpunkt dazwischen sein muß. Es bleibt zu zeigen, daß sich keine zwei verschiedenen Bogensegmente ζ und η in der Sequenz s^* mehr als $s+2$ mal abwechseln. Dazu nehmen wir an, es gäbe eine solche (Teil-)Sequenz s^* mit $s+3$ Wechseln und führen dies zu einem Widerspruch, indem wir zeigen, daß sich ζ und η dann mehr als s mal schneiden.
Für eine gegebene Sequenz

$$s^* = (\cdots \zeta_1 \cdots \eta_1 \cdots \zeta_2 \cdots \eta_2 \cdots\cdots \zeta_j \cdots \eta_j \cdots \zeta_k \cdots (\eta_k \cdots))$$

mit den signifikanten ζ_i und η_i fassen wir je 4 sukzessive Segmente zu einem Quadrupel wie folgt zusammen.

$$\begin{array}{llll} (\zeta_1 & ,\eta_1 & ,\zeta_2 & ,\eta_2) \\ & (\eta_1 & ,\zeta_2 & ,\eta_2, \quad \zeta_3) \\ & & (\zeta_2 & ,\eta_2, \quad \zeta_3, \quad \eta_3) \\ & & & \vdots \end{array}$$

Der Index k und die Existenz von η_k hängt von s ab: wenn $s+3$ ungerade ist, dann existiert η_k und es gilt $k = \frac{s+2}{2} + 1$. Wenn $s+3$ gerade ist, entfällt das letzte η_k und der Wert von k ist eine leichte Übungsaufgabe.

Sei $s+3$ ungerade. In diesem Fall erhalten wir insgesamt $s+1$ Quadrupel:

$$(\zeta_1, \eta_1, \zeta_2, \eta_2), (\eta_1, \zeta_2, \eta_2, \zeta_3), (\zeta_2, \eta_2, \zeta_3, \eta_3), \ldots, (\zeta_{\frac{s+2}{2}}, \eta_{\frac{s+2}{2}}, \zeta_{\frac{s+2}{2}+1}, \eta_{\frac{s+2}{2}+1}).$$

Die Anzahl der Quadrupel ergibt sich aus $2\cdot(\frac{s+2}{2})-1$ Quadrupeln angeführt von je $\frac{s+2}{2}$ Elementen aus ζ und $\frac{s+2}{2} - 1$ Elementen aus η.

Wir zeigen, daß für jedes Quadrupel der Form $(\zeta_i, \eta_i, \zeta_{i+1}, \eta_{i+1})$ (bzw. $(\eta_i, \zeta_{i+1}, \eta_{i+1}, \zeta_{i+2})$) genau ein Schnittpunkt zwischen ζ_i und η_i (bzw. zwischen η_i und ζ_{i+1}) liegen muß. Damit hätten wir insgesamt $s+1$ verschiedene Schnittpunkte im Widerspruch zu der Annahme, daß sich ζ und η nicht mehr als s mal schneiden. Ergo existieren höchstens $s+2$ Wechsel.

Es bleibt zu zeigen, daß für ein beliebiges Quadrupel, nennen wir es o. B. d. A. $(\zeta_i, \eta_i, \zeta_{i+1}, \eta_{i+1})$ der besagte Schnittpunkt existiert. Dazu wählen wir $x \in \zeta_i$, $y \in \zeta_{i+1}$, $z \in \eta_i$ und $w \in \eta_{i+1}$ so, daß C diese Punkte in der Reihenfolge x, z, y, w besucht. Die Teilbögen $\beta^z_x, \beta^y_z, \beta^w_y$ und β^x_w aus C schneiden sich nur in den Endpunkten, weil C eine geschlossene, einfache Kurve ist. Wir betrachten weiterhin die Bögen $\beta^y_x \in \zeta$ und $\beta^w_z \in \eta$, siehe Abbildung 2.19.

Wir wollen zeigen, daß sich β^y_x und β^w_z, wie Abbildung 2.19 durch den Bogen $\beta^{w'}_{z'}$ angedeutet wird.

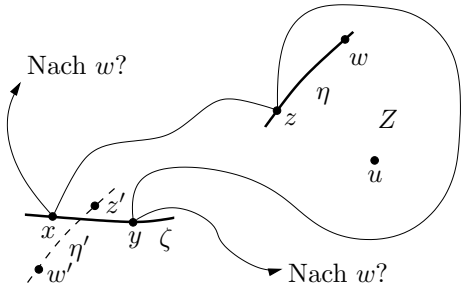


Abbildung 2.19: Die paarweise disjunkten Bögen $\beta_x^z, \beta_z^y, \beta_y^w, \beta_w^x, \beta_x^y$ und β_z^w.

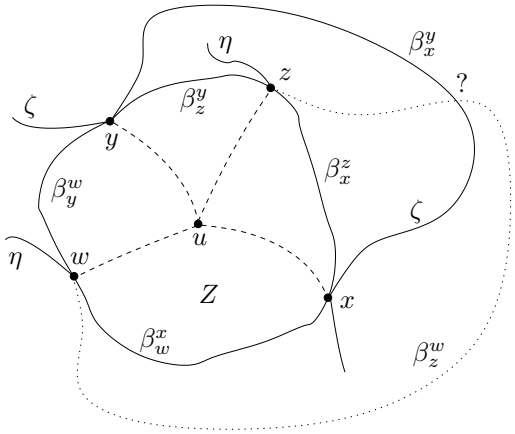
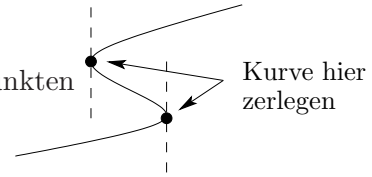


Abbildung 2.20: Eine kreuzungsfreie Einbettung, wie sie von der angenommenen Lage der Bögen induziert wird, existiert nicht.

Wir wollen einen formalen Widerspruchsbeweis führen. Dazu nehmen wir an, daß sich β_x^y und β_z^w nicht schneiden. Wählen wir einen Punkt u im Inneren von Z, dann können wir x, y, z und w kreuzungsfrei mit u verbinden. Da die Bögen $\beta_x^z, \beta_z^y, \beta_y^w, \beta_w^x, \beta_x^y$ und β_z^w allesamt kreuzungsfrei sind, hätten wir den vollständigen Graphen K_5 kreuzungsfrei in die Ebene eingebettet. Das ist aber nicht möglich: wie in Abbildung 2.20 zeigt, kann man nicht, jeden der Punkte u, w, x, y, z mit jedem anderen verbinden, ohne daß sich zwei Bögen kreuzen. Diese Tatsache läßt sich auch mit Hilfe der Eulerformel und einer weiteren Abschätzung formal beweisen, siehe z. B. [Kle97].

□

Wie können wir nun eine einzelne Zelle Z im Arrangement von n Kurvenstücken berechnen, die durch einen in Z enthaltenen Punkt — den Referenzpunkt des Roboters — bestimmt ist? Zur Berechnung gehen wir von folgenden Annahmen aus:[5]
(i) Je zwei Bögen schneiden sich höchstens s mal.
(ii) Jeder Bogen hat nur konstant viele vertikale Tangenten. Damit können wir ohne Einschränkung annehmen, daß alle Bögen X–monoton sind, nicht X–monotone Bögen können an den Tangentenpunkten der vertikalen Tangenten in X–monotone Teilbögen zerlegt werden, vgl. Abbildung.



Ein erster Ansatz zur Lösung dieses Problems ist der Divide–and–Conquer Algorithmus 2.4. Das Problem dabei ist, daß $Z_R \cap Z_B$ sehr komplex werden kann und die explizite Berechnung zu hohe Kosten verursacht. Abbildung 2.21 zeigt ein Beispiel, in dem $Z_R \cap Z_B$ recht komplex ist, Z selbst jedoch recht einfach. Wir brauchen also einen effizienteren Ansatz für den Merge–Schritt; die Frage ist nun, wie man die Zelle Z berechnen kann, ohne alle Zusammenhangskomponenten von $Z_R \cap Z_B$ zu berechnen.

[5]Diese Forderungen sind erfüllt, wenn die Bögen durch algebraische Gleichungen vom Grad $O(1)$ beschrieben werden.

Algorithmus 2.4 Berechnung einer Zelle im Arrangement von n Bögen

Gegeben: Punkt x, n Bögen.

Gesucht: Die Zelle Z im Arrangement der Bögen, die x enthält.

- Zerlege die Menge der Bögen in gleichgroße Teilmengen R und B.
- Berechne rekursiv im Arrangement $\mathcal{A}(R)$ die Zelle Z_R, die x enthält.
- Berechne rekursiv im Arrangement $\mathcal{A}(B)$ die Zelle Z_B, die x enthält.
- Berechne die Zusammenhangskomponente Z von $Z_R \cap Z_B$, die x enthält und berichte diese (Red–Blue–Merge).

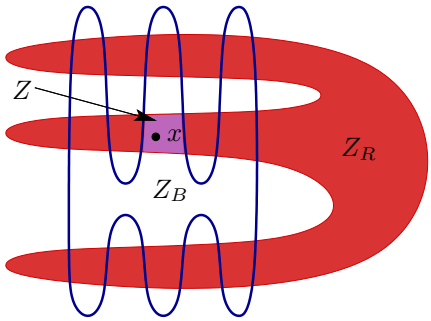


Abbildung 2.21: $Z_R \cap Z_B$ kann sehr komplex sein, obwohl Z recht einfach ist.

Betrachten wir einen etwas allgemeineren Fall; gegeben sind:

- Ein "rotes" Arrangement $\mathcal{R}$ mit ausgezeichneten Zellen $R_1, \ldots, R_{m_R}$ mit insgesamt r Ecken (Bogensegmenten).
- Ein "blaues" Arrangement $\mathcal{B}$ mit ausgezeichneten Zellen $B_1, \ldots, B_{m_B}$ mit ingesamt b Ecken (Bogensegmenten). [6]
- Eine Menge von Punkten $p_i \in R_{\mu_i} \cap B_{\nu_i}$, $1 \leq i \leq k$ — d. h. jeder Punkt ist sowohl in einer roten als auch in einer blauen Zelle enthalten —, wobei nicht jede Menge $R_\mu \cap B_\nu$ ein p_i enthalten muß und verschiedene p_i in derselben Menge $R_\mu \cap B_\nu$ liegen können.

Betrachte nun das "violette" Arrangement $\mathcal{V}$, das durch Überlagerung des roten Arrangements $\mathcal{R}$ und des blauen Arrangements $\mathcal{B}$ entsteht. In $\mathcal{V}$ liegt jeder Punkt p_i in einer Zelle Z_i. Wir wollen nun diese Z_i berechnen, jedoch jede nur einmal, auch wenn sie mehrere Punkte enthält. Über die Komplexität dieser Zellen, d.h. die Anzahl Segmente auf dem Rand, gibt das folgende Lemma Auskunft:

Lemma 2.21 *(Kombinationslemma; Guibas, Sharir, Sifrony, 1989)*
Für die Komplexität der Zellen $Z_1, \ldots, Z_\ell$, $\ell \leq k$ gilt:[7] *[GSS89]*

$$|Z_1| + |Z_2| + \cdots + |Z_\ell| \in O(r + b + k)$$

Eine stärkere Version dieses Lemmas stammt von Edelsbrunner, Guibas und Sharir [EGS90], zu finden auch in [SA95]. Die Aussage von Lemma 2.21 ist keineswegs trivial: ein Bogen kann zu vielen violetten Zellen, ja sogar mehrere Stücke zum Rand einer Zelle beitragen, siehe Abbildung 2.23. Wir geben hier keinen Beweis für Lemma 2.21.

[6] Wir bezeichnen im Folgenden die Segmente der Zellen R_i und B_j als Bögen, um sie von den neuen Segmenten zu unterscheiden, die beim Schnitt der Zellen entstehen.

[7] $\ell \leq k$, da mehrere Punkte in derselben Zelle liegen können.

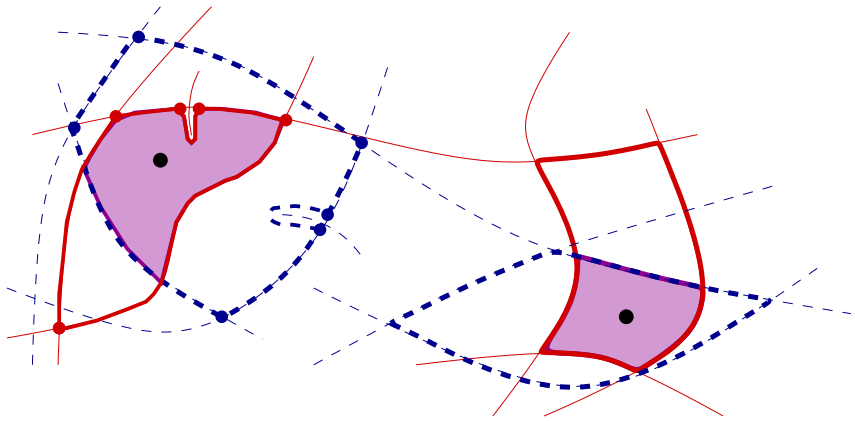


Abbildung 2.22: Arrangement von roten und blauen Zellen und Überlagerung. Komplexität der linken blauen (linken roten) Zelle: 6 (5) Segmente.

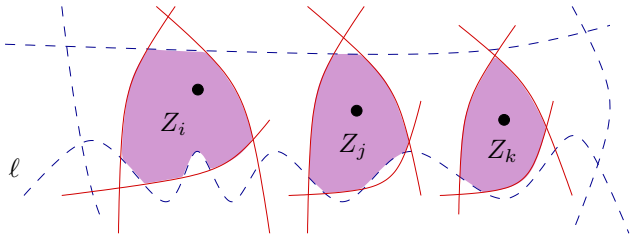


Abbildung 2.23: Der Bogen ℓ trägt zu vielen violetten Zellen bei.

Wie lassen sich nun die Zellen Z_i berechnen?

Theorem 2.22 *Seien $Z_1, \ldots, Z_\ell$ diejenigen Zellen, welche die Punkte $p_1, \ldots, p_k$ enthalten. Dann lassen sich die Zellen $Z_1, \ldots, Z_\ell$ in Zeit $O((r+b+k)\log(r+b+k))$ berechnen.*

Beweis. Algorithmus 2.5 berechnet etwas mehr als verlangt, nämlich alle violetten Zellen, die einen Punkt $q \in Q$ im Abschluß enthalten. Wie bereits erwähnt, liegt die Schwierigkeit darin, *nur* diese Zellen zu berechnen, nicht alle Zusammenhangskomponenten von $R_\mu \cap B_\nu$. Die beiden Sweeps sind notwendig, da z. B. in Abbildung 2.25(i) während des ersten Sweeps nicht erkannt werden kann, daß in a eine violette Region beginnt, da noch kein $q \in Q$ entdeckt wurde. a liegt zwar auf dem Rand einer Zelle, ist jedoch kein Endpunkt eines Bogens. Erst wenn die Sweepline q_1 erreicht, wird eine violette Teilregion begonnen. Analog kann während des zweiten Sweeps der Beginn der violetten Region noch nicht im Punkt b sondern erst bei q_3 erkannt werden.

Die Idee bei dem Sweep (Algorithmus 2.6) ist die Verwendung von je einem oberen und unteren "Scout" je violetter Teilregion. Diese Scouts bewegen sich mit der Sweepline weiter und helfen bei der Erkennung und Behandlung von Ereignissen, die Einfluß auf den Verlauf des Randes der violetten Region haben. Für die Abschätzung des Arbeitsaufwands bei der Ereignisbehandlung ist wichtig, daß jeder Bogen von höchstens einem oberen Scout und höchstens einem unteren Scout bewacht wird, der sich auf einem Bogen der anderen Farbe bewegt. Es treten zwei Arten von Aktualisierungen der Ereignisstruktur auf:
A: berechne den in X–Richtung nächsten Schnittpunkt zwischen Scout und beobachtetem Bogen.
B: berechne den in X–Richtung nächsten Schnittpkt. zwischen beiden Scouts.

[8] Die Schnittpunkte gleichfarbiger Bögen gehören zwar zur Ereignisstruktur, jedoch nicht zu Q. Ansonsten würde in Abbildung 2.25(i) bereits in Punkt a eine Region beginnen! Diese Schnittpunkte sind aus dem letzten Rekursionsschritt bekannt, müssen also nicht explizit berechnet werden.

Algorithmus 2.5 Red–Blue–Merge

- Sei $Q := \left\{ p_1, \ldots, p_k \right\} \cup \left\{ \text{ Endpunkte von auf dem Rand der roten oder blauen Zellen liegenden Bögen } \right\}$. $n := r + b + k$, $|Q| \in O(n)$.
- Für jedes $q \in Q$: berechne mit gewöhnlichem Sweep in den Arrangements $\mathcal{R}$ und $\mathcal{B}$ die vertikalen Segmente von q nach oben und unten zum nächsten roten und blauen Bogen. Diese werden wie zusätzliche Bögen behandelt (Abbildung 2.24(i)). $O(n \log n)$
- Sortiere die $q \in Q$ und alle Ecken[8] der Zellen R_μ und B_ν nach auf– und absteigenden X–Koordinaten (Ereignisstrukturen). $O(n \log n)$
- Sweep von links nach rechts:
  Berechne die Teile der violetten Teilzellen, die rechts vom linkesten darin enthaltenen $q \in Q$ liegen.
- Sweep von rechts nach links:
  Berechne die Teile der violetten Teilzellen, die links vom rechtesten darin enthaltenen $q \in Q$ liegen.
- Setze die Teilregionen zusammen. $O(n)$

Folgende Ereignisse können auftreten:

1. Eine neue Region startet. Zeit $O(1)$ für das Anlegen einer neuen Region und zwei Scouts, Aktualisierungen **A** und **B**.
2. Schnittpunkt von Bögen gleicher Farbe oder Endpunkt eines Bogens. Evtl. muss ein beobachtender Scout einen neuen Bogen beobachten, Aktualisierung **A**.
3. Der Scoutbogen und der beobachtete Bogen treffen aufeinander. Der Scout wechselt auf den beobachteten Bogen und beobachtet den ehemaligen Scoutbogen, Aktualisierung **A** und **B**.
4. Die Scouts treffen aufeinander. Eine violette Teilzelle endet, die Scouts verschwinden, Zeit $O(1)$.
5. Ein Punkt $q \in Q$ wird in einer aktiven Region angetroffen. Die aktive Region endet und es entstehen ein oder zwei neue Teilzellen. Wir brauchen $O(1)$ Zeit um festzustellen, daß q zwischen den Scouts liegt und $O(1)$ Zeit für den Abschluß der alten Teilzelle und den Beginn der neuen Teilzelle(n) wie unter 1.

Pro Ereignis tragen wir konstant viele neue Schnittpunkte in die Ereignisstruktur ein, das Einfügen und die Aktualisierung der Sweep–Status–Struktur kostet uns jeweils $O(\log n)$. Wir haben nach Lemma 2.21 $O(r + b + k) = O(n)$ viele Ereignisse. Damit können alle violetten Regionen in Zeit $O(n \log n)$ berechnet werden. □

Abbildung 2.27 zeigt zwei Beispiele für die Aufgabenteilung der Scouts:

(i) In Punkt a wechselt der Scout O_1 von einem roten auf einen blauen Bogen und bewacht jetzt den roten Bogen, auf dem er vorher gelaufen ist. O_2 muß aktiv werden und den blauen Rand bewachen.

(ii) In Punkt b ist das Ende einer violetten Teilregion erreicht. Der obere und untere Scout der Teilregion treffen zusammen und verschwinden, der Scout O_2 wird aktiv und bewacht den blauen Rand.

In beiden Fällen wird der aktiv werdende Scout O_2 durch binäre Suche in der Sweep–Status–Struktur gefunden.
Insgesamt können wir festhalten:

Theorem 2.23 *Gegeben seien n Bögen mit $O(1)$ senkrechten Tangenten; je zwei Bögen schneiden sich höchstens s mal. Die Zelle Z im Arrangement, die einen gegebenen Punkt x enthält, kann in Zeit $O(\lambda_{s+2}(n)\log^2 n)$ berechnet werden.*[9]

Beweis. Die Laufzeit ergibt sich aus der Rekursionsgleichung $T(n) = 2T(\frac{n}{2}) +$ Merge(n) für den Divide–and–Conquer Ansatz. Die Kosten für einen Merge betragen nach Theorem 2.22 $O((r+b+k)\log(r+b+k))$. In unserem Fall gilt $k = 1$ und mit Theorem 2.18 $r + b \in O(\lambda_{s+2}(n))$. Insgesamt also Merge$(n) \in O(\lambda_{s+2}(n)\log(\lambda_{s+2}(n)) \subseteq O(\lambda_{s+2}(n)\log n)$. Damit ergibt sich

$$T(n) = 2^\ell T(1) + C \cdot \sum_{i=1}^{\ell} \lambda_{s+2}(n)\log n \in O(\lambda_{s+2}(n)\log^2 n)\,.$$

□

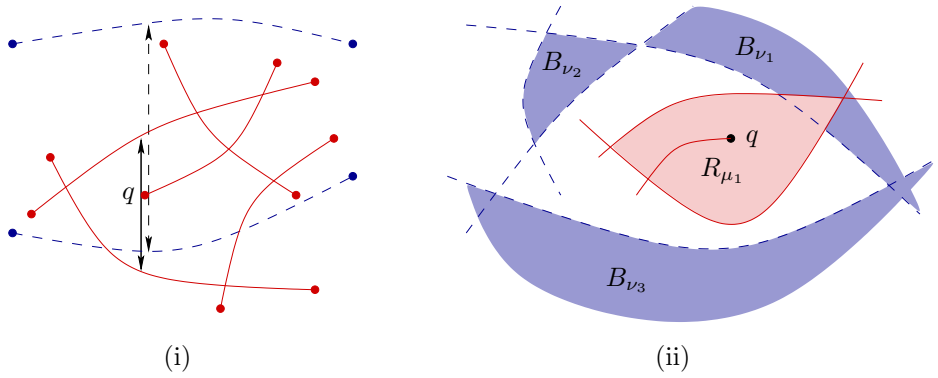


Abbildung 2.24: (i) Zusätzliche vertikale Segmente, (ii) bei q beginnt keine violette Region.

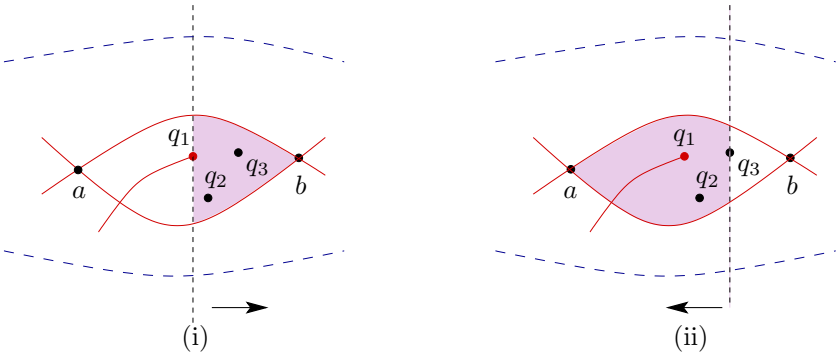


Abbildung 2.25: (i) Sweep von links nach rechts, (ii) Sweep von rechts nach links.

[9] $\log^2(n) = \log(n) \cdot \log(n)$.

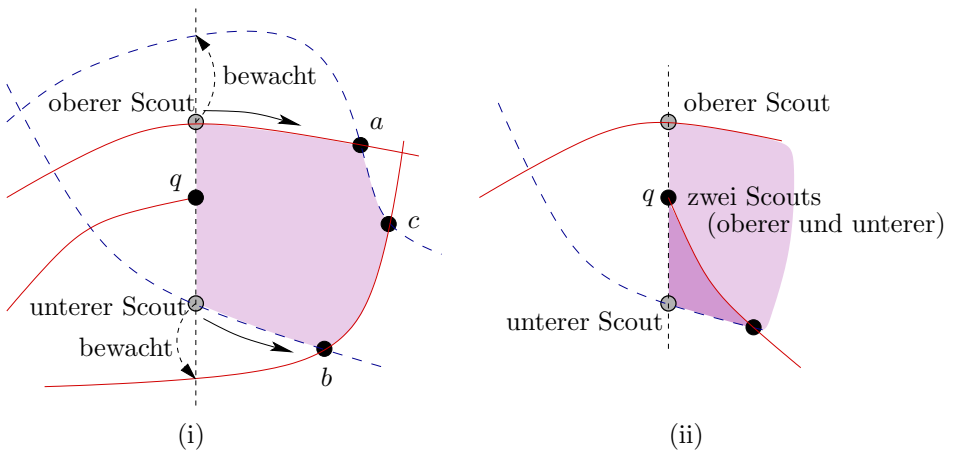


Abbildung 2.26: In q starten (i) eine oder (ii) zwei violette Teilregionen mit Scouts.

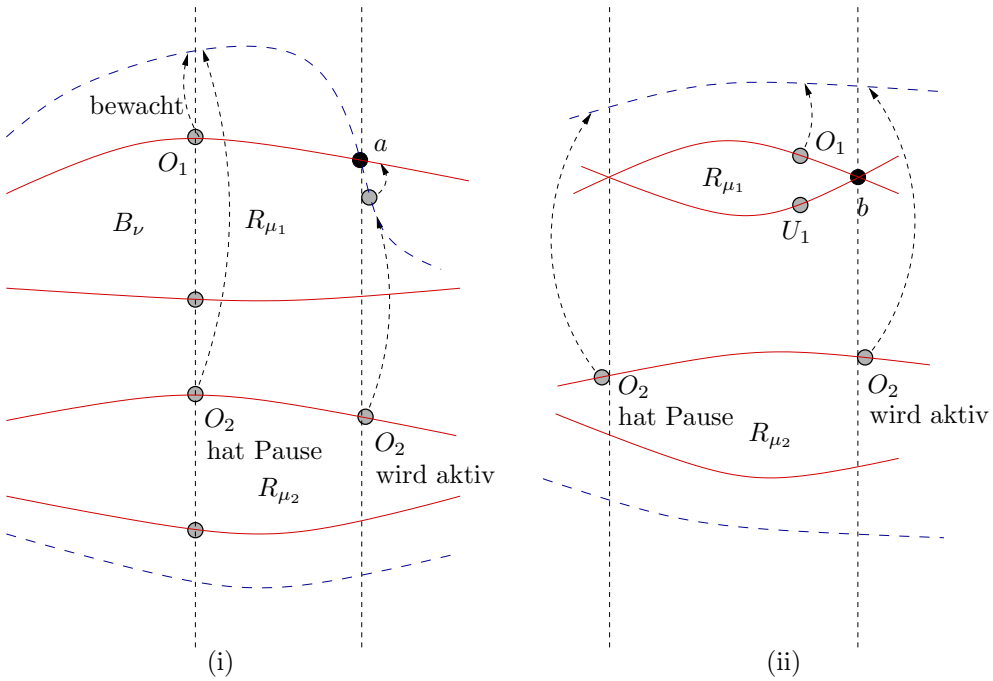


Abbildung 2.27: Beispiele für die Aufgaben der Scouts.

Algorithmus 2.6 Sweep zur Berechnung der Teilzellen

(hier: Sweep von links nach rechts, rechts nach links analog)

Ereignisstruktur: Priority–Queue W sortiert nach X–Koordinaten, initialisiert mit den Punkten aus Q und allen Knoten auf dem Rand der blauen und roten Regionen.

Sweep–Status–Struktur: enthält die Bögen entlang der Sweepline, nach Y–Koordinaten sortiert. Für jeden Bogen wird ein Zeiger auf den beobachtenden Scout gespeichert und pro Scout ein Zeiger auf den Bogen, auf dem sich der Scout bewegt und den Bogen, den der Scout beobachtet.

Ereignis: Wenn ein Punkt $q \in Q$ angetroffen wird, benutze die vertikalen Segmente von q um festzustellen, ob bei q eine violette Teilregion beginnt. Dies ist genau dann der Fall, wenn q sowohl in einem R_μ als auch in einem B_ν enthalten ist. In Abbildung 2.24(ii) beginnt in q *keine* violette Region, da $q \notin \bigcup B_{\nu_j}$. Falls ja, beginne bei q eine oder zwei violette Teilregionen, und starte am oberen und unteren Rand jeder Teilregion je einen Scout, siehe Abbildung 2.26.

Aufgaben der Scouts:

Hier die Aufgaben des oberen Scouts O, der untere Scout verhält sich symmetrisch.

- Der Scout startet auf dem zu q nächsthöheren Bogen. Hier starte O o. B. d. A. auf einem roten Bogen ϱ.
- Der Scout bewegt sich mit der Sweepline.
- Der Scout bewacht den nächsthöheren blauen Bogen β:
  - Berechne den in X–Richtung nächsten Schnittpunkt x zwischen ϱ und β und trage ihn in W ein (Abbildung 2.26(i), Punkte a und b).
  - Wenn die Sweepline den Punkt x erreicht, wechselt O auf β und bewacht ϱ.
  - Wenn β endet oder einen anderen blauen Bogen schneidet, beobachte analog den nächsthöheren Bogen β'.
- "Arbeitsteilung": falls zwei Scouts denselben Bogen bewachen, ist nur der oberste aktiv. Längs der Sweepline gilt also: der Rand einer blauen Region wird nur von dem obersten darunter befindlichen Scout auf einem roten Bogen bewacht. Diese Invariante muß bei jedem Ereignis aktualisiert werden.
- O beobachtet den unteren Scout, d. h. ihr Schnittpunkt wird berechnet und falls existent in W eingetragen. Wenn beide sich treffen, ist das Ende der violetten Region erreicht und beide Scouts verschwinden (Abbildung 2.26(i), Punkt c). Die violette Region endet evtl. schon eher, falls nämlich ein weiterer Punkt aus Q zwischen den Scouts erreicht wird.

2.2.2 Anwendung: Translation eines beliebigen Polygons

Mit den Ergebnissen des vorherigen Abschnittes können wir nun unser Bahnplanungsproblem lösen:

Theorem 2.24 *Sei R ein polygonaler Roboter mit m Ecken, der sich in einer Umgebung mit polygonalen Hindernissen P_i mit insgesamt n Ecken bewegt. Gegeben seien Start– und Zielposition s, t. Dann läßt sich in Zeit $O(mn\,\alpha(mn)\log^2(mn))$ eine kollisionsfreie Translation von s nach t bestimmen oder feststellen, daß keine solche existiert.*

Beweis. Die Laufzeit von Algorithmus 2.7 folgt aus Theorem 2.23 mit $\lambda_3(mn) \in O(mn\,\alpha(mn))$ (Theorem A.9 auf Seite 154). □

Wenn man bedenkt, daß $\alpha(mn)$ fast konstant ist und auch $\log^2(mn)$ nur sehr langsam wächst, so bleibt im wesentlichen eine Laufzeit von $O(mn)$. Angesichts dieses doch recht komplexen Problems ein bemerkenswertes Ergebnis!

Algorithmus 2.7 Translation eines beliebigen Roboters

Vorbereitung:

- Betrachte das Arrangement $\mathcal{A}$ der $2mn$ Liniensegmente, die sich aus den Bedingungen der Ecke/Kante–Paare ergeben.
- Konstruiere in $\mathcal{A}$ die Zelle Z, die s enthält.
  Komplexität $O(mn\,\alpha(mn))$, Laufzeit $O(mn\,\alpha(mn)\log^2(mn))$
- Führe Trapezzerlegung in Z aus, und konstruiere den Zusammenhangsgraph (Sweep). $O(mn\,\alpha(mn)\log(mn))$

Query: für gegebenes t:

- Bestimme das Trapez, das t enthält. $O(\log(mn))$
- Finde einen Pfad von s nach t im Zusammenhangsgraph. $O(mn\,\alpha(mn))$

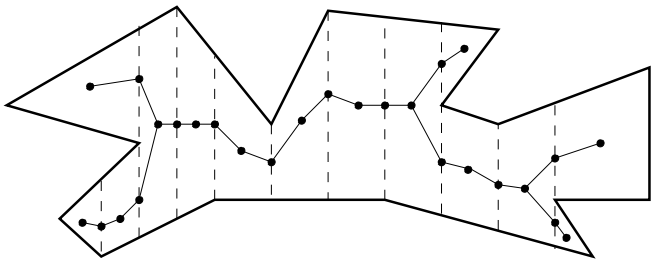


Abbildung 2.28: Trapezzerlegung und Zusammenhangsgraph einer Zelle.

2.2.3 Anwendung: Allgemeine Systeme mit zwei Freiheitsgraden

Bisher haben wir nur die Translation von Polygonen betrachtet, die eindeutig durch die Position ihres Referenzpunktes bestimmt sind, also zwei Freiheitsgrade (DOF) haben. Der Ansatz des vorherigen Abschnittes ist jedoch allgemein genug, um andere Systeme mit zwei Freiheitsgraden zu behandeln! Abbildung 2.29 zeigt einen Roboter mit zwei Armen, dessen Position durch die beiden Winkel α und γ eindeutig beschrieben wird. Sein Konfigurationsraum ist also $\mathcal{C} = [0, 2\pi) \times [0, 2\pi)$.

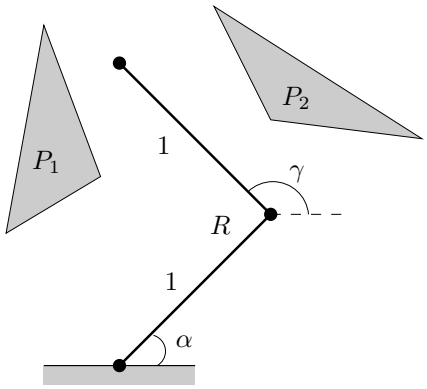


Abbildung 2.29: Roboter mit zwei Armen und Freiheitsgraden α und γ.

Wie sieht hier der Rand des Raumes der freien Konfigurationen $\mathcal{C}_{\text{frei}}$ aus?

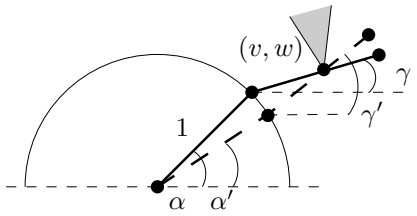
Ein Segment auf dem Rand von $\mathcal{C}_{\text{frei}}$ entsteht z. B. durch den Kontakt einer Hindernisecke mit dem vorderen Arm (siehe Abbildung). Da die Hindernisecke nicht den Endpunkt von R berührt, muß ein $\lambda \in [0,1]$ existieren, so daß für den Kontaktpunkt (v,w) gilt:

$$\begin{aligned} \begin{pmatrix} v \\ w \end{pmatrix} &= \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} + \lambda \begin{pmatrix} \cos\gamma \\ \sin\gamma \end{pmatrix} \\ \Rightarrow \lambda &= \frac{v-\cos\alpha}{\cos\gamma} = \frac{w-\sin\alpha}{\sin\gamma} \end{aligned}$$

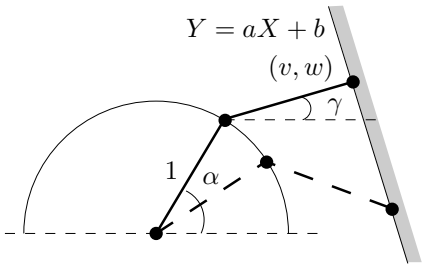
mit $x := \cos\alpha, y := \cos\gamma$:

$$\begin{aligned} (v-x)\sqrt{1-y^2} &= (w-\sqrt{1-x^2}\,)y \\ \Leftrightarrow (v-x)^2(1-y^2) &= (w^2 - 2w\sqrt{1-x^2} + 1 - x^2)y^2. \end{aligned}$$

Aus der letzten Gleichung hebt sich x^2y^2 fort. Löst man nach der Wurzel auf und quadriert die Gleichung, so ergibt sich eine Gleichung vom Grad[10] 6 in x, y. Wegen $0 \le \lambda \le 1$ beschreibt diese ein Bogenstück vom Grad 6. Es handelt sich dabei um Bögen im Konfigurationsraum des Roboters, die Bögen sind daher nicht direkt in der Anschauung ablesbar.



Ein anderer Typ von Segmenten auf dem Rand von $\mathcal{C}_{\text{frei}}$ entsteht durch den Kontakt des Roboterkopfes mit einer Wand. Die Wand läßt sich durch eine Gerade $Y = aX + b$ beschreiben, und für den Kontaktpunkt (v,w) gilt:



$$\begin{aligned} (v,w) &= (\cos\alpha + \cos\gamma, \sin\alpha + \sin\gamma) \in \{\, Y = aX + b \,\} \\ &\Leftrightarrow a(\cos\alpha + \cos\gamma) + b = \sin\alpha + \sin\gamma \\ &\Leftrightarrow a(x+y) + b = \sqrt{1-x^2} + \sqrt{1-y^2} \\ &\Leftrightarrow a(x+y)^2 + b - 2 + x^2 + y^2 = \sqrt{1-x^2}\sqrt{1-y^2} \end{aligned}$$

Durch Quadrieren der letzten Gleichung ergibt sich ein Bogenstück vom Grad 4.

Alle Konfliktbögen haben einen Grad kleiner gleich 6, mit Bezouts Theorem[11] folgt, daß je zwei

[10] Der Grad eines Polynoms mit mehreren Variablen ist die Summe der Grade (d. h. der höchsten Potenzen) der einzelnen Variablen.

[11] Zwei algebraische Kurven von Grad m und n schneiden sich in höchstens $m \cdot n$ Punkten.

Bögen höchstens 6^2 Schnitte haben. Mit Theorem 2.23 auf Seite 81 folgt, daß für dieses Problem kollisionsfreie Bewegungen in Zeit $O(\lambda_{38}(n)\log^2 n) \subseteq O(n\log^{2+\varepsilon} n)$ geplant werden können.

Literaturverzeichnis

[AAAS97] Pankaj K. Agarwal, Nina Amenta, Boris Aronov, and Micha Sharir. Largest placements and motion planning of a convex polygon. In Jean-Paul Laumond and Mark Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 143–154, Wellesley, MA, 1997. A. K. Peters.

[Ack28] W. Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematical Annals*, 99:118–133, 1928.

[AK00] Franz Aurenhammer and Rolf Klein. Voronoi diagrams. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[AKM+87a] A. Aggarwal, M. M. Klawe, S. Moran, P. W. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[Akm87b] V. Akman. *Unobstructed Shortest Paths in Polyhedral Environments*, volume 251 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1987.

[ALMS98] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack. An ε-approximation algorithm for weighted shortest paths on polyhedral surfaces. In *Proc. 6th Scand. Workshop Algorithm Theory*, volume 1432 of *Lecture Notes Comput. Sci.*, pages 11–22. Springer-Verlag, 1998.

[AS00] Pankaj K. Agarwal and Micha Sharir. Davenport-Schinzel sequences and their geometric applications. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 1–47. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[Baj85] C. Bajaj. The algebraic complexity of shortest paths in polyhedral spaces. In *Proc. 23rd Allerton Conf. Commun. Control Comput.*, pages 510–517, 1985.

[BGG96] E. Börger, E. Grädel, and Y. Gurevic. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer Verlag, Berlin, 1996.

[ČČ61] Josef Čapek and Karel Čapek. *R.U.R. and the Insect Play.* Oxford Paperbacks, 1961.

[Cha82] Bernard Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 339–349, 1982.

[Cha91] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[CJ95] Svante Carlsson and Hakan Jonsson. Computing a shortest watchman path in a simple polygon in polynomial time. In *Proc. 4th Workshop Algorithms Data Struct.*, volume 955 of *Lecture Notes Comput. Sci.*, pages 122–134. Springer-Verlag, 1995.

[CJN99] S. Carlsson, H. Jonsson, and B. J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete Comput. Geom.*, 22(3):377–402, 1999.

[Cla87] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 56–65, 1987.

[CN86] W. Chin and S. Ntafos. Optimum watchman routes. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 24–33, 1986.

[CN88] W. Chin and S. Ntafos. Optimum watchman routes. *Inform. Process. Lett.*, 28:39–44, 1988.

[CN91] W.-P. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete Comput. Geom.*, 6(1):9–31, 1991.

[Col75] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes Comput. Sci.*, pages 134–183. Springer-Verlag, 1975.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158. Shaker Heights, 1971.

[CR87] J. Canny and J. H. Reif. New lower bound techniques for robot motion planning problems. In *Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 49–60, 1987.

[dBvKOS97] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[DELM03] Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *Proc. 35th Annu. ACM Sympos. Theory Comput.*, pages 473–482, 2003.

[DGST88] J. R. Driscoll, H. N. Gabow, R. Shrairaman, and R. E. Tarjan. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. *Commun. ACM*, 31:1343–1354, 1988.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DS65] H. Davenport and A. Schinzel. A combinatorial problem connected with differential equations. *Amer. J. Math.*, 87:684–689, 1965.

[Dub85] V. A. Dubovitskij. The Ulam problem of optimal motion of line segments. In *Optimization Software*. ??, New York, NY, 1985.

[EG86] H. Edelsbrunner and Leonidas J. Guibas. Topologically sweeping an arrangement. In *Proc. 18th Annu. ACM Sympos. Theory Comput.*, pages 389–403, 1986.

[EG89] H. Edelsbrunner and Leonidas J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 38:165–194, 1989. Corrigendum in 42 (1991), 249–251.

[EGS90] H. Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. The complexity and construction of many faces in arrangements of lines and of segments. *Discrete Comput. Geom.*, 5:161–196, 1990.

[EOS86] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, 15:341–363, 1986.

[FT87] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615, 1987.

[Gar83] Martin Gardner. The game of life, parts I–III. In *Wheels, Life, and other Mathematical Amusements*. W. H. Freeman, New York, 1983.

[GH87] Leonidas J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 50–63, 1987.

[GH89] Leonidas J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, October 1989.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[GM87] S. K. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. In *Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 11–19, 1987.

[GM91] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, 20:888–910, 1991.

[Gol93] Kenneth Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.

[GSS89] Leonidas J. Guibas, Micha Sharir, and S. Sifrony. On the general motion planning problem with two degrees of freedom. *Discrete Comput. Geom.*, 4:491–521, 1989.

[Gur75] A. B. Gurevich. The 'most economical' displacement of a segment. *Differential Equations*, 11:1583–1589, 1975.

[Her91] J. Hershberger. A new data structure for shortest path queries in a simple polygon. *Inform. Process. Lett.*, 38:231–235, 1991.

[HN97] M. Hammar and Bengt J. Nilsson. Concerning the time bounds of existing shortest watchman route algorithms. In *Proc. 11th International Symposium on Fundamentals of Computation Theory*, volume 1279 of *Lecture Notes Comput. Sci.*, pages 210–221. Springer-Verlag, September 1997.

[HS86] S. Hart and Micha Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6:151–177, 1986.

[HS93] J. Hershberger and Subhash Suri. Matrix searching with the shortest path metric. In *Proc. 25th Annu. ACM Sympos. Theory Comput.*, pages 485–494, 1993.

[HS96] D. Halperin and Micha Sharir. A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. *Discrete Comput. Geom.*, 16:121–134, 1996.

[HS97] J. Hershberger and Subhash Suri. Matrix searching with the shortest path metric. *SIAM J. Comput.*, 26(6):1612–1634, December 1997.

[HS99] John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999.

[HSS84] J. E. Hopcroft, J. T. Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects: P-space hardness of the "Warehouseman's Problem". *Internat. J. Robot. Res.*, 3(4):76–88, 1984.

[IKKM97] Christian Icking, Rolf Klein, Peter Köllner, and Lihong Ma. A Java applet for the dynamic visualization of Voronoi diagrams. In *Abstracts 13th European Workshop Comput. Geom.*, pages 46–47. Universität Würzburg, 1997.

[IRWY93] Christian Icking, Günter Rote, Emo Welzl, and Chee Yap. Shortest paths for line segments. *Algorithmica*, 10:182–200, 1993.

[Kle89] Rolf Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1989.

[Kle97] Rolf Klein. *Algorithmische Geometrie.* Addison-Wesley, Bonn, 1997.

[Kle05] Rolf Klein. *Algorithmische Geometrie - Grundlagen, Methoden, Anwendungen.* Springer, Heidelberg, 2nd edition, 2005.

[KLPS86] K. Kedem, R. Livne, J. Pach, and Micha Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.

[KO88] Y. Ke and J. O'Rourke. Lower bounds on moving a ladder in two and three dimensions. *Discrete Comput. Geom.*, 3:197–217, 1988.

[KS90] K. Kedem and Micha Sharir. An efficient motion planning algorithm for a convex rigid polygonal object in 2-dimensional polygonal space. *Discrete Comput. Geom.*, 5:43–75, 1990.

[KST97] K. Kedem, Micha Sharir, and S. Toledo. On critical orientations in the Kedem-Sharir motion planning algorithm for a convex polygon in the plane. *Discrete Comput. Geom.*, 17:227–240, 1997.

[Lat91] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, 1991.

[LP83] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.

[LP84] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14:393–410, 1984.

[LS87] D. Leven and Micha Sharir. On the number of critical free contacts of a convex polygonal object moving in two-dimensional polygonal space. *Discrete Comput. Geom.*, 2:255–270, 1987.

[Ma00] Lihong Ma. Bisectors and voronoi diagrams for convex distance functions, 2000.

[Mit00] Joseph S. B. Mitchell. Geometric shortest paths and network optimization. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[MMP87] Joseph S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16:647–668, 1987.

[OW88] M. H. Overmars and Emo Welzl. New methods for computing visibility graphs. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 1988.

[OW93] Th. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen*, volume 70 of *Reihe Informatik*. BI Wissenschaftsverlag, Mannheim, 1993.

[PV95] M. Pocchiola and G. Vegter. Computing the visibility graph via pseudo-triangulations. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 248–257, 1995.

[RS94] J. H. Reif and J. A. Storer. A single-exponential upper bound for finding shortest paths in three dimensions. *J. ACM*, 41(5):1013–1019, 1994.

[SA95] Micha Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[Sei91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1(1):51–64, 1991.

[Sha97] Micha Sharir. Algorithmic motion planning. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 40, pages 733–754. CRC Press LLC, Boca Raton, FL, 1997.

[SS83a] J. T. Schwartz and Micha Sharir. On the "piano movers" problem I: The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Commun. Pure Appl. Math.*, 36:345–398, 1983.

[SS83b] J. T. Schwartz and Micha Sharir. On the "piano movers" problem II: General techniques for computing topological properties of real algebraic manifolds. *Adv. Appl. Math.*, 4:298–351, 1983.

[SS84] Micha Sharir and A. Schorr. On shortest paths in polyhedral spaces. In *Proc. 16th Annu. ACM Sympos. Theory Comput.*, pages 144–153, 1984.

[SS86] Micha Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, 15:193–215, 1986.

[Stö99] H. Stöcker. *Taschenbuch mathematischer Formeln und moderner Verfahren*. Verlag Harri Deutsch, Thun und Frankfurt am Main, 1999.

[Tar51] A. Tarski. *A decision method for elementary algebra and geometry*. Univ. of California Press, Berkeley, CA, 1951.

[TH93] Xuehou Tan and Tomio Hirata. Constructing shortest watchman routes by divide-and-conquer. In *Proc. 4th Annu. Internat. Sympos. Algorithms Comput.*, volume 762 of *Lecture Notes Comput. Sci.*, pages 68–77. Springer-Verlag, 1993.

[THI93] X. H. Tan, T. Hirata, and Y. Inagaki. An incremental algorithm for constructing shortest watchman routes. *Internat. J. Comput. Geom. Appl.*, 3(4):351–365, 1993.

[THI99] X. Tan, T. Hirata, and Y. Inagaki. Corrigendum to "an incremental algorithm for constructing shortest watchman routes". *Internat. J. Comput. Geom. Appl.*, 9(3):319–323, 1999.

[vdS94] A. F. van der Stappen. *Motion Planning amidst Fat Obstacles*. Ph.D. dissertation, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, 1994.

[Wel85] Emo Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Inform. Process. Lett.*, 20:167–171, 1985.

[Yap87] C.-K. Yap. Algorithmic motion planning. In J. T. Schwartz and C.-K. Yap, editors, *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, pages 95–143. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Algorithmen