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.

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

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]

- 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.Langetep@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.

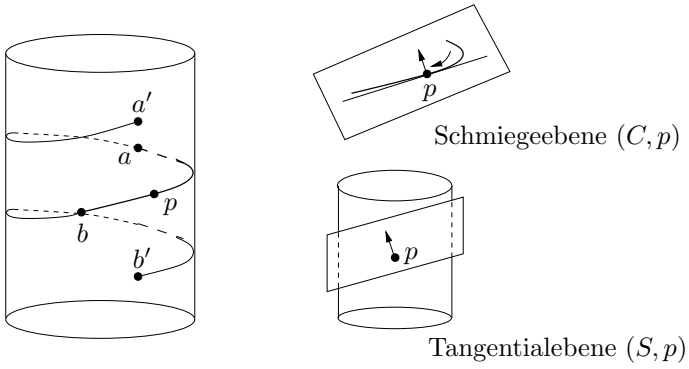


Abbildung 1.1: Kürzeste von a nach b, Geodätische von a' nach b'.

Wir interessieren uns hier für diskrete Umgebungen.

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

1.1 Kürzeste Pfade in der Ebene mit polygonalen Hindernissen

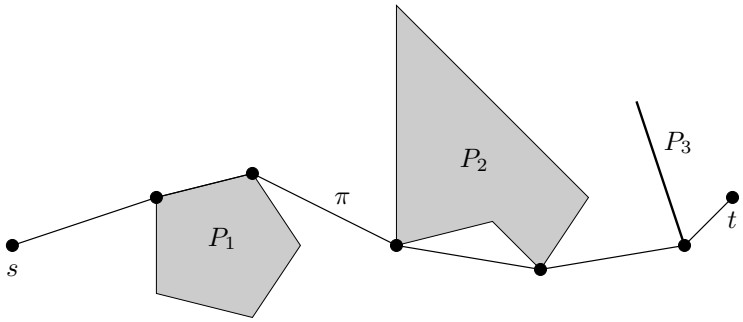


Abbildung 1.2: Umgebung mit polygonalen Hindernissen.

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.

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]).

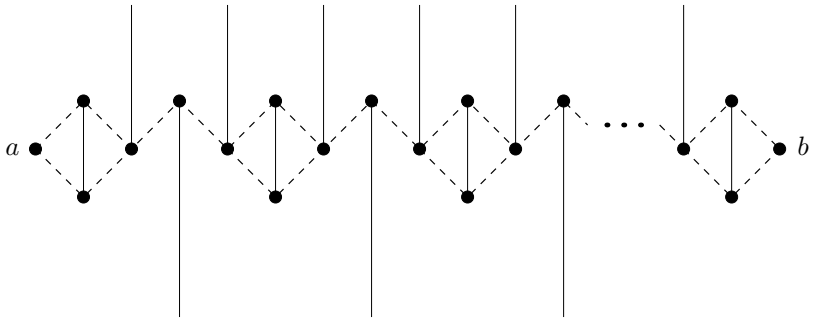
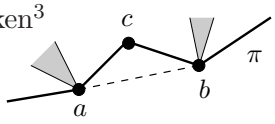


Abbildung 1.3: Es kann exponentiell viele kürzeste Wege geben.

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

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, wie das Beispiel in Abbildung 1.4(ii) zeigt.

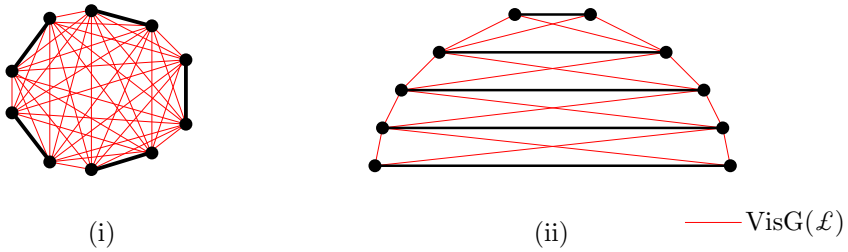


Abbildung 1.4: Der Sichtbarkeitsgraph kann (i) $\Omega(n^2)$ viele Kanten enthalten, aber auch (ii) nur $O(n)$ viele.

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

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.

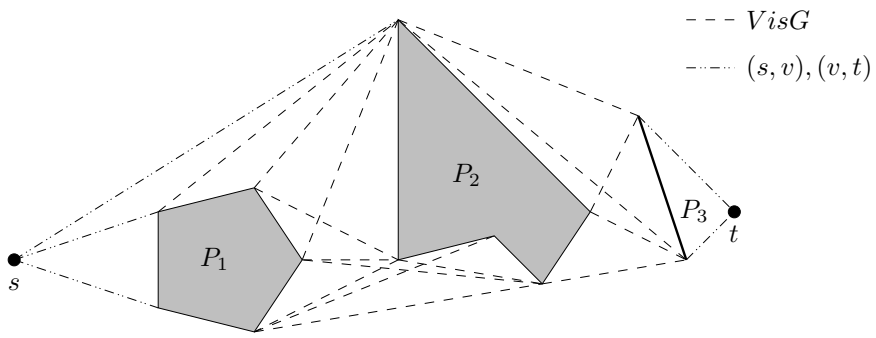
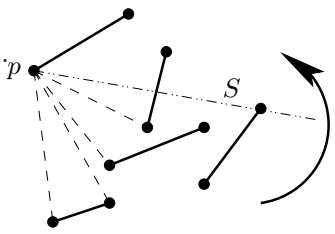


Abbildung 1.5: VisG und Kanten von s und zu t.



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

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 $\overline{pq}$.

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

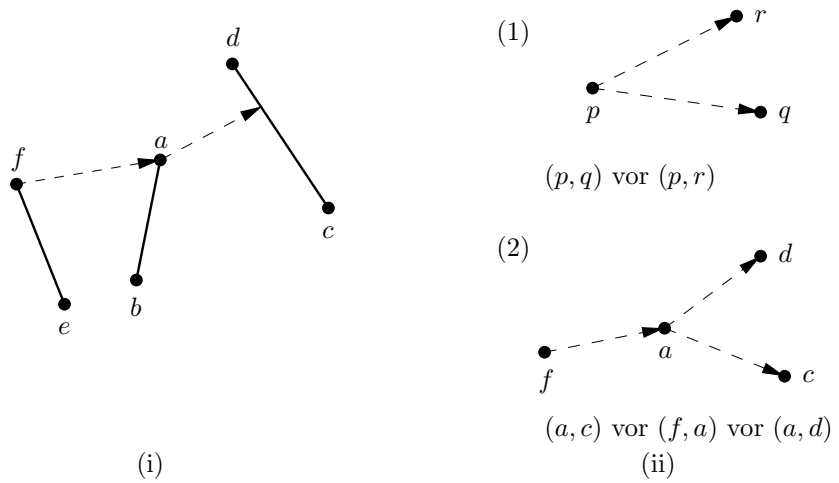


Abbildung 1.6: (i) Sichtbarkeitsinformationen können von a an f weitergereicht werden, (ii) Typen von Bedingungen an die Bearbeitungsreihenfolge.

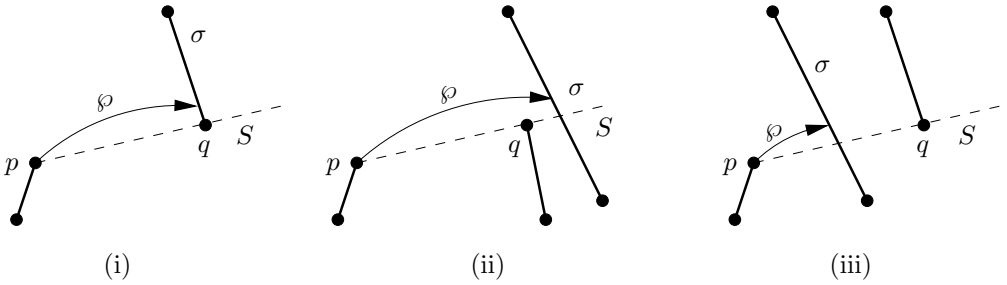


Abbildung 1.7: Es gibt drei mögliche Ausgangssituationen eines Punktes p mit zuletzt bearbeitetem Partner q und sichtbarem Segment σ.

(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]

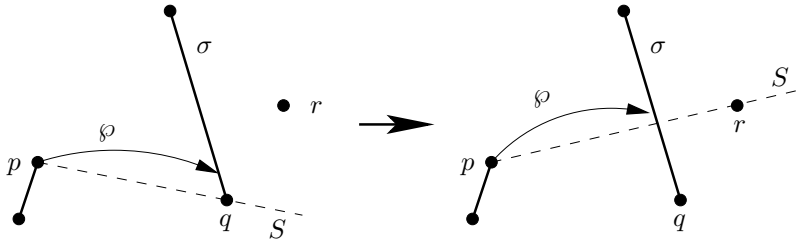
$$\text{Punkt } p = (p_x, p_y) \longmapsto \text{ Gerade } p^* = \{\, Y = p_x X - p_y \,\}.$$

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

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

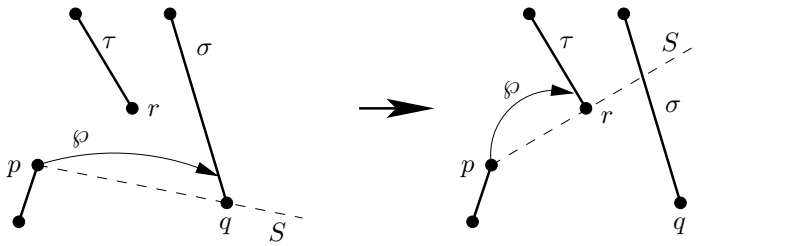
[4]Diese Dualisierung unterscheidet sich nur im Vorzeichen von der in [Kle05] betrachteten Dualisierung $p = (p_x, p_y) \longmapsto p^* = \{Y = p_x X + p_y\}$.

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



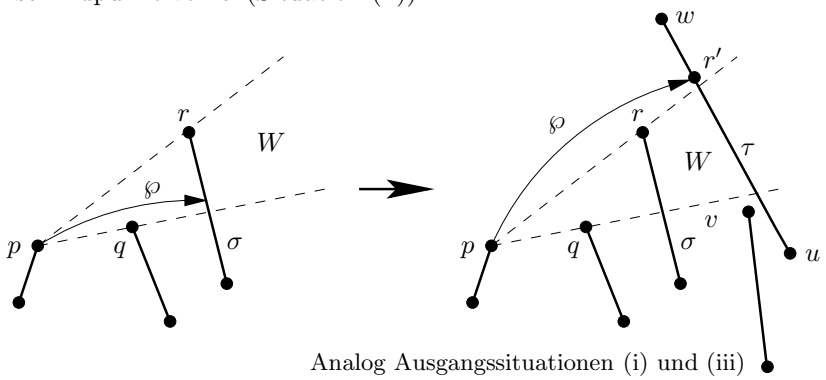
Analog Ausgangssituationen (ii) und (iii)

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).

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

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

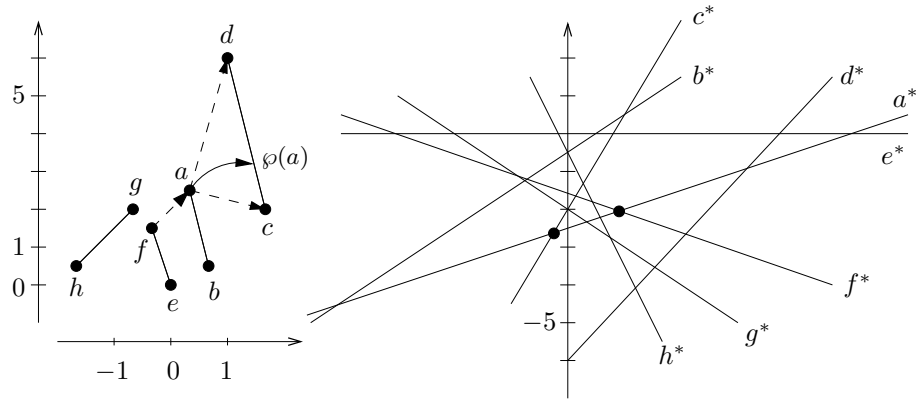


Abbildung 1.9: Dualisierung $p = (p_x, p_y) \longmapsto p^* = \{\, Y = p_x X - p_y \,\}$.

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:

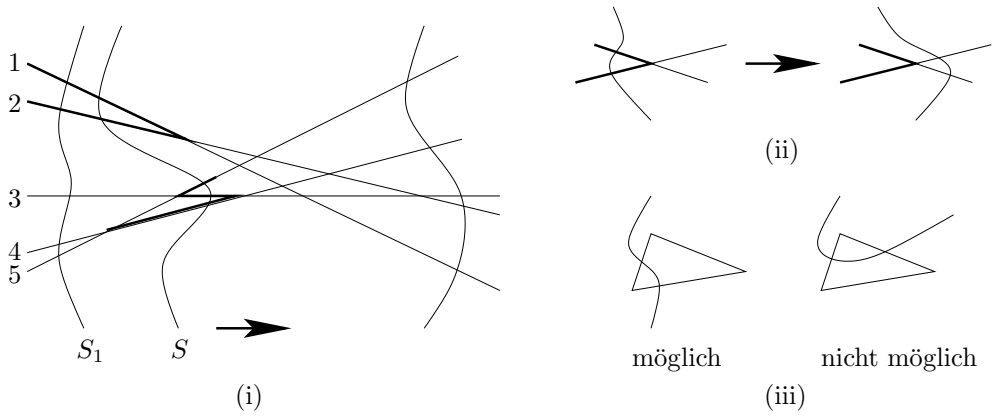


Abbildung 1.10: (i) Topologischer Sweep, (ii) Elementarschritt, (iii) Die Sweepkurve kann eine Zelle nur von oben betreten und nach unten verlassen.

	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.

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.

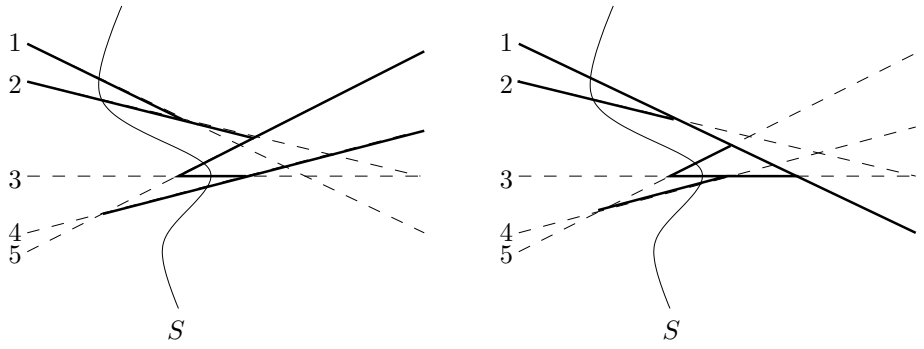
Ein Ansatz zur Lösung dieses Problems ist, in einer Liste N entlang S sortiert für jede Gerade G festzuhalten, von welcher anderen Geraden G' das von S durchkreuzte Segment von G rechts begrenzt wird. Für das Arrangement in Abbildung 1.10(i) ergibt sich also entlang der ersten eingezeichneten Sweepkurve 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 $(3,4)$.



Oberer Horizontbaum T^O Unterer Horizontbaum T_U

Abbildung 1.11: Oberer und unterer Horizontbaum.

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.

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.

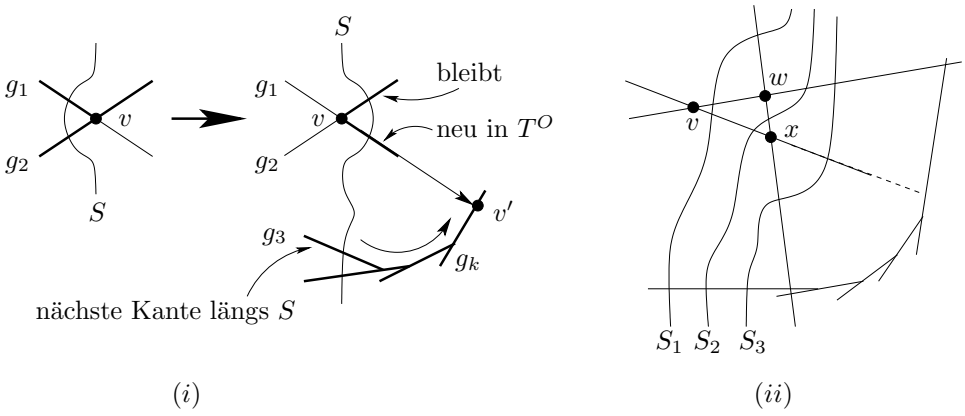


Abbildung 1.12: Aktualisierung der Horizontbäume am Beispiel von T^O.

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:

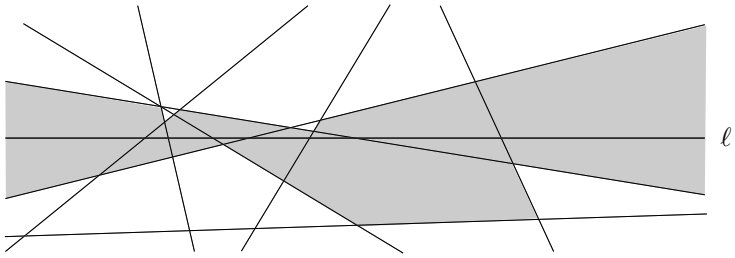


Abbildung 1.13: Zone einer Geraden ℓ in einem Arrangement $\mathcal{A}$.

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:

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

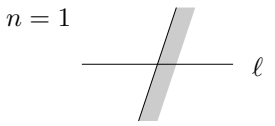
Theorem 1.4 *(Zonentheorem)*
Sei $\mathcal{A}$ ein Arrangement von n nicht senkrechten Geraden und sei ℓ eine beliebige Gerade. Die Zone von ℓ hat die Komplexität $O(n)$.

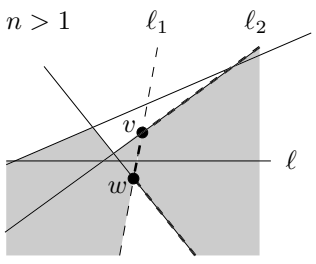
Beweis.
In jeder Zelle gibt es einen rechten, obersten Punkt und einen linken, untersten Punkt. Diese Punkte teilen den Rand einer Zelle eindeutig in einen linken und einen rechten Rand.

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?

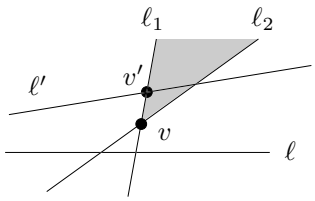
(i) Die Kante $\overline{vw}$.

(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.

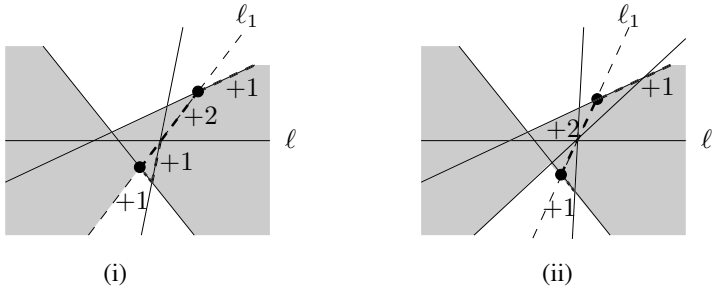


Abbildung 1.14: Im rechtesten Schnittpunkt schneiden sich zwei oder mehrere Geraden. Die gestrichelten Kanten werden neu eingefügt.

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:

$$
\begin{aligned}
& \text{Kosten der Aktualisierung von } T^O, T_U \\
\approx\ & \text{Summe aller Kosteneinheiten Kante–Knoten aus je einer Zelle} \\
\leq\ & \sum_{\text{Zellen } Z} (\#\text{Kanten von } Z) \cdot (\#\text{Knoten von } Z) \\
=\ & \sum_{\text{Zellen } Z} (\#\text{Kanten von } Z)^2
\end{aligned}
$$

$$\leq \sum_{\text{Geraden } G} \underbrace{\sum_{\substack{\text{Zellen } Z \text{ mit} \\ \text{Kante auf } G}} (\#\text{Kanten von } Z)}_{O(n) \text{ nach Theorem 1.4}}$$

$$\in O(n^2)$$

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

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

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

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 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

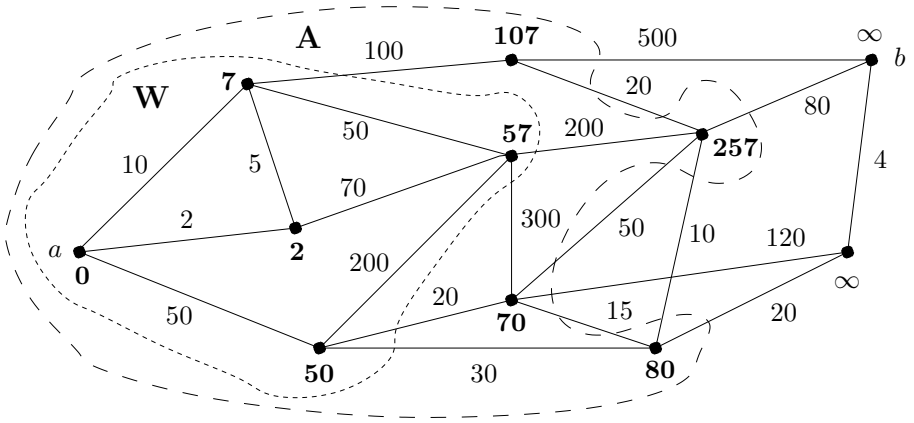


Abbildung 1.15: Welle, Ausläufer und Knotenmarkierungen während des Algorithmus von Dijkstra.

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 $\Omega(n \log n)$.*

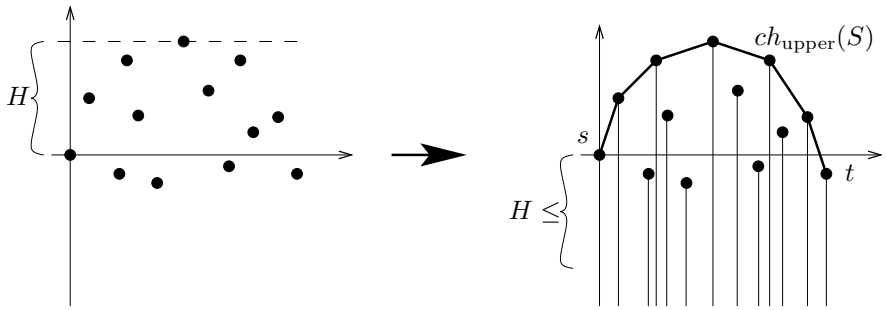


Abbildung 1.16: Reduktion: Konstruktion der konvexen Hülle auf Bestimmung des kürzesten Pfades.

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:

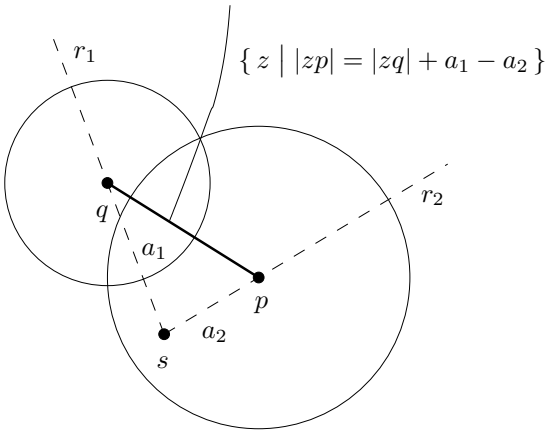


Abbildung 1.17: Berechnung der Shortest Path Map.

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:

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]*

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