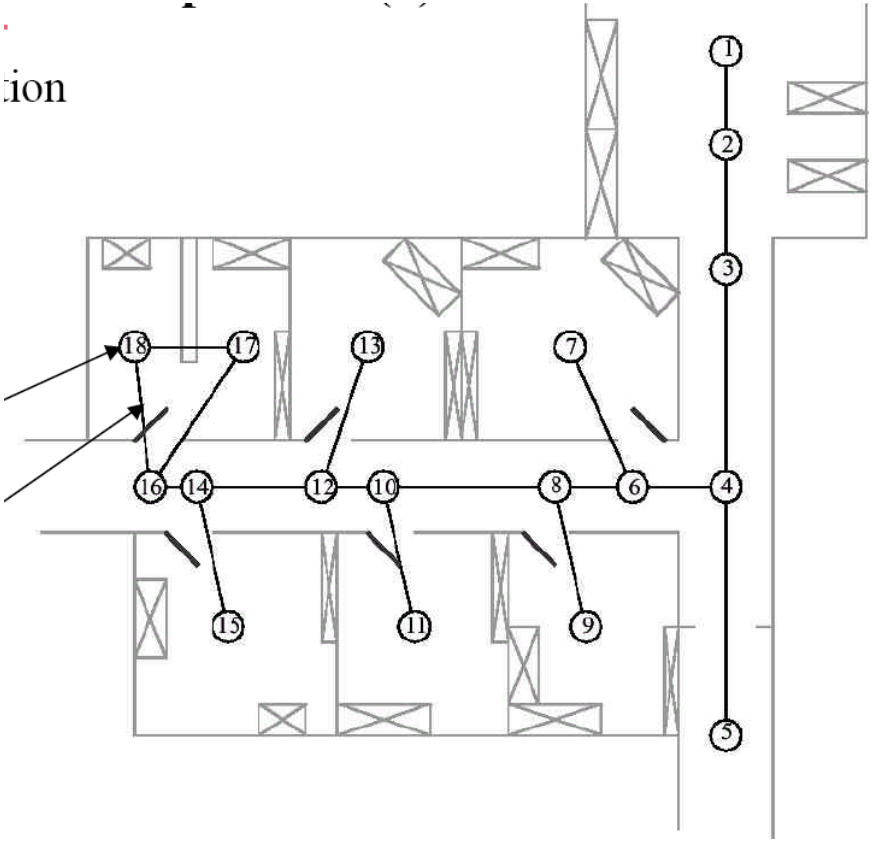
Wegeplanung: Wegekartenverfahren

- Idee
- Sichtbarkeitsgraph
- Voronoi-Diagramm
- Probabilistische Wegekarten
- Rapidly-Exploring Random Tree

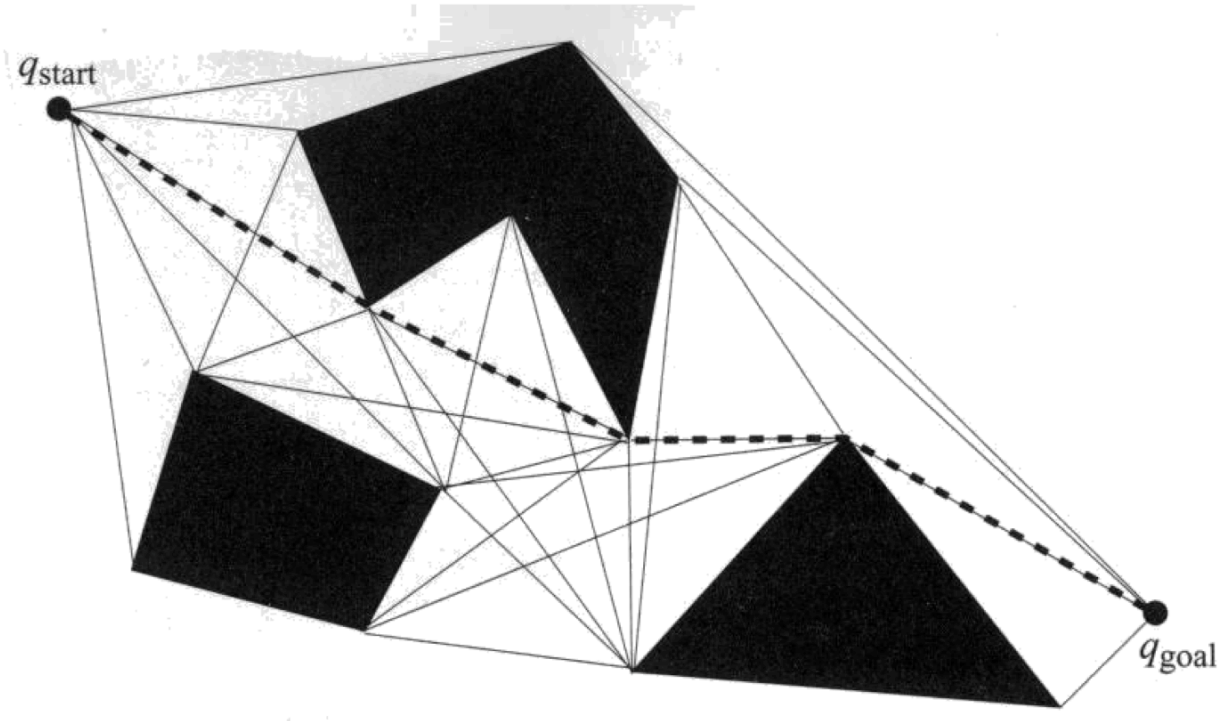
Idee

- Spanne hindernisfreien Raum durch Graphen auf.
- Knoten liegen im hindernisfreien Raum.
- Benachbarte Knoten (d.h. durch Kanten verbunden) sind kollisionsfrei und einfach zu erreichen
- Graph sollte Arbeitsraum „gut“ aufspannen.
- Wegeplanung reduziert sich damit auf Verbinden von Start- und Endpunkt zu einem nächstgelegenen Knoten im Graphen und anschließender Wegesuche im Graphen.



Sichtbarkeitsgraph

- Für eine polygonale Hinderniswelt P ist der Sichtbarkeitsgraph S definiert durch:
  - alle Hindernis-Ecken und Start- und Zielpunkt bilden die Knoten;
  - für Knoten u und v ist (u,v) eine Kante des Sichtbarkeitsgraphen, falls $\overline{uv}$ keine Hindernisse schneidet.



Berechnung des Sichtbarkeitsgraphen

Algorithmus VisibiltyGraph(P, q_{Start}, q_{Ziel})

```
V = { Menge aller Ecken aus P } ∪ {q_Start, q_Ziel };
for all v ∈ V do
    W = VisibleVertices(v, P);
    for all w ∈ W do
        füge (v,w) zu E hinzu;
    endfor
endfor
return (V,E);
```

Eingabe:

Polygonale Hinderniswelt P, gegeben durch Menge von Kanten.

Start- und Zielpunkt.

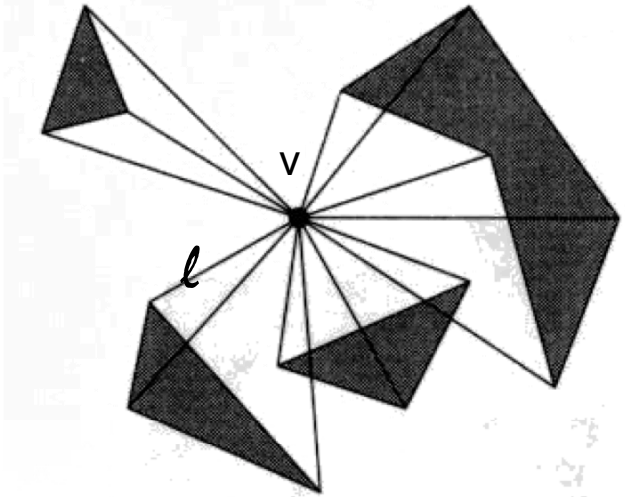
VisibleVertices(v, P) berechnet alle Knoten, die von v aus sichtbar sind.

Ausgabe:

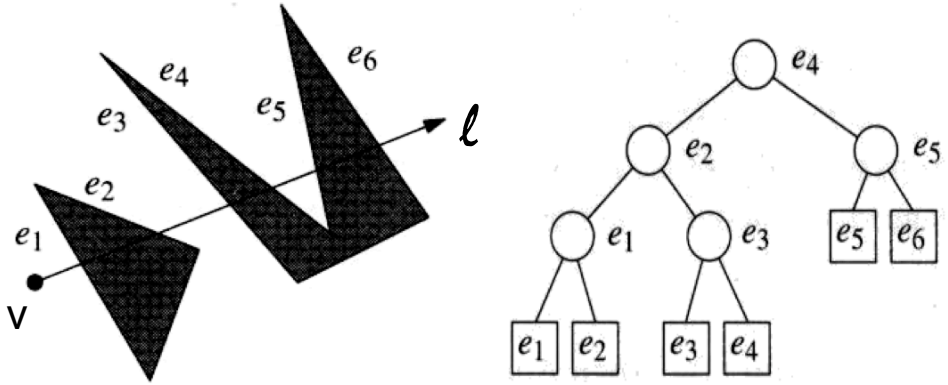
Sichtbarkeitsgraph G = (V,E)

Rotierendes Sweep-Line-Verfahren (1)

- VisibleVertices(v, P) berechnet alle Knoten, die von v aus sichtbar sind.
- Algorithmus ist ein rotierendes Sweep-Line-Verfahren:
  Rotiere Line ℓ um v jeweils zur nächsten Ecke.
- Dabei werden alle Hinderniskanten, die ℓ schneiden, in einer effizienten Suchstruktur (z.B. balanzierter Suchbaum) gespeichert.



Line ℓ, die um v rotiert



Line ℓ und alle Hinderniskanten,
die ℓ schneiden, in einem Suchbaum.

Rotierendes Sweep-Line-Verfahren (2)

Algorithmus VisibleVertices(v, P)

(1) sortiere alle Ecken aus P nach ihren Polarkoordinaten bzgl. Knoten v (zuerst nach Winkel und dann nach Abstand); sortierte Folge sei $w_1, w_2, ..., w_n$;

(2) Sei l die Linie von v in Richtung x-Achse (0 Grad); Speichere alle Hinderniskanten, die l schneiden, in einem Suchbaum T; (Reihenfolge ergibt sich durch Entfernung zwischen v und Schnittpunkt der Hinderniskanten mit l)

(3) W = { };

(4) for i = 1 to n do

(5) drehe Linie l, so dass sie durch w_i geht;

(6) if w_i ist von v aus sichtbar then

(7) W = W ∪ {(v,w_i)};

(8) if w_i ist der Beginn einer oder zweier Hinderniskanten $e_{1,2} \notin T$ then

(9) füge $e_{1,2}$ zu T dazu;

(10) if w_i ist das Ende einer oder zwei Hinderniskanten $e_{1,2} \in T$ then

(11) lösche $e_{1,2}$ in T;

(12) endfor

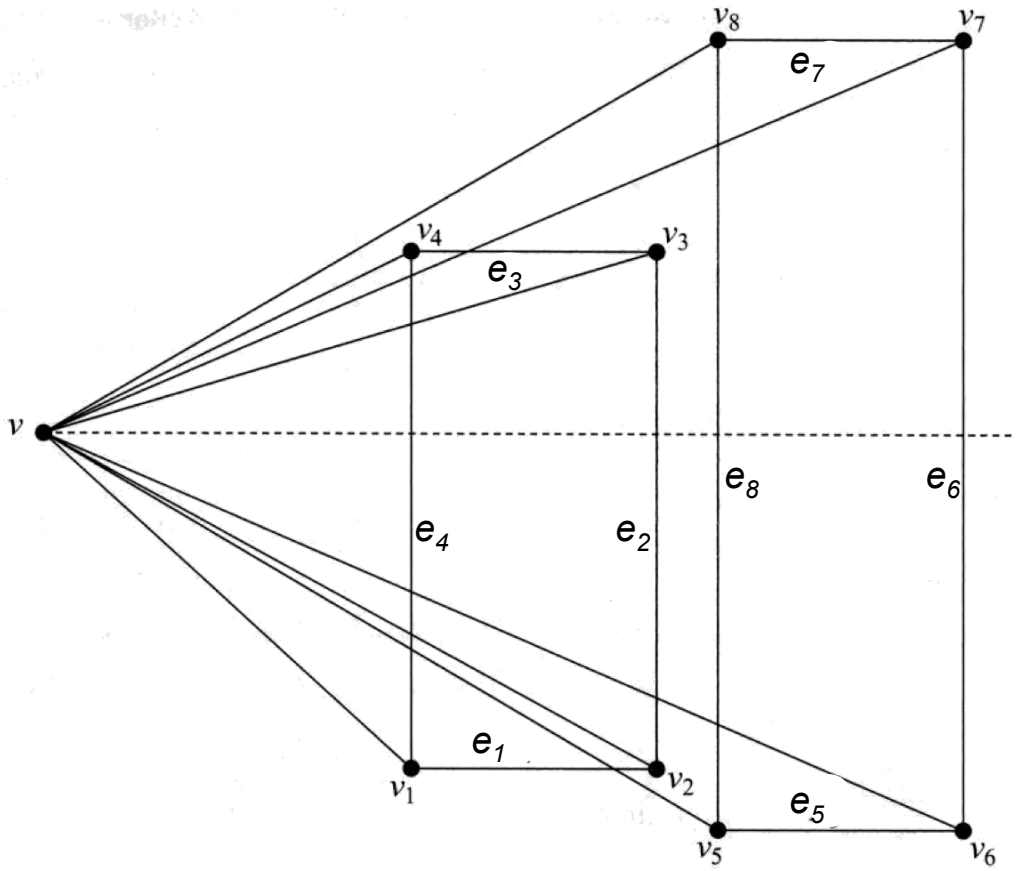
(13) return W;

Eingabe:
Polygonale Hindernis-welt P und Knoten v.

Linienbezogene Sichtbarkeits-prüfung (später)

Ausgabe:
Menge W alle Knoten, die von v aus sichtbar sind.

Beispiel zu Sweep-Line-Verfahren



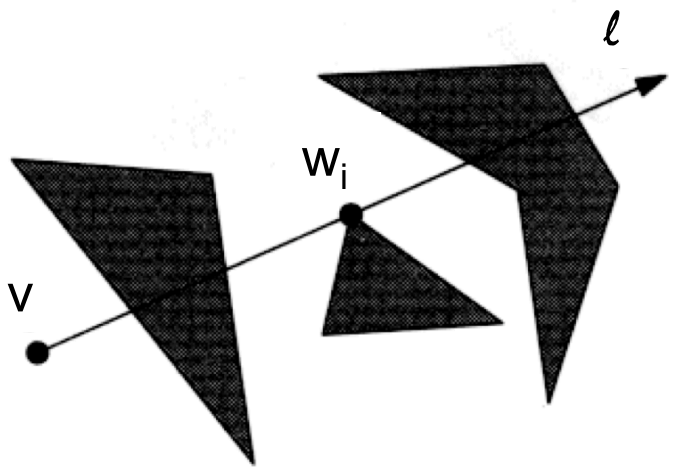
Sweep-Line ℓ durch Ecke	Suchbaum T mit Kanten	Ecke sichtbar
Initialisierung: Schritt (2)	$\{e_4,e_2,e_8,e_6\}$	
v_3	$\{e_4,e_3,e_8,e_6\}$	
v_7	$\{e_4,e_3,e_8,e_7\}$	
v_4	$\{e_8, e_7\}$	ja
v_8	$\{\ \}$	ja
v_1	$\{e_1,e_4\}$	ja
v_5	$\{e_4,e_1,e_8,e_5\}$	
v_2	$\{e_4,e_2,e_8,e_5\}$	
v_6	$\{e_4,e_2,e_8,e_6\}$	

Ecken v_i sind nach dem Winkel zur x-Achse sortiert.
ℓ rotiert gegen den Uhrzeigersinn.

Linienbezogene Sichtbarkeitsprüfung (1)

w_i ist nicht sichtbar, falls

(1) auf der Sweep-Line ℓ liegt links von w_i keine Ecke (d.h. w_{i-1} liegt nicht auf ℓ), aber es liegt eine Hinderniskante $e \in T$ links von w_i.

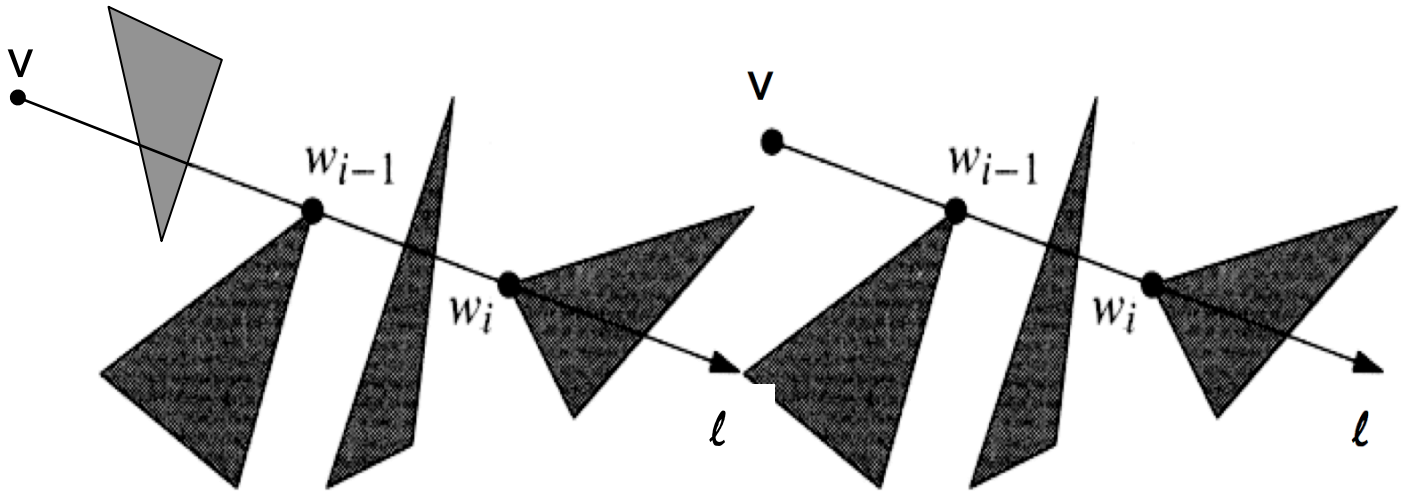


Linienbezogene Sichtbarkeitsprüfung (2)

w_i ist nicht sichtbar, falls

(2) w_{i-1} liegt auch auf der Sweep-Line ℓ und ist nicht sichtbar **oder**

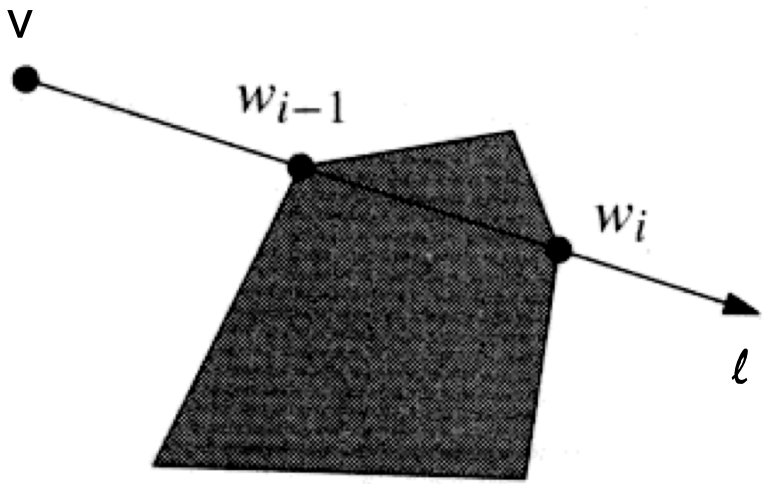
(3) w_{i-1} liegt auch auf der Sweep-Line ℓ und ist sichtbar, aber zwischen w_{i-1} und w_i liegt eine Hinderniskante.



Linienbezogene Sichtbarkeitsprüfung (3)

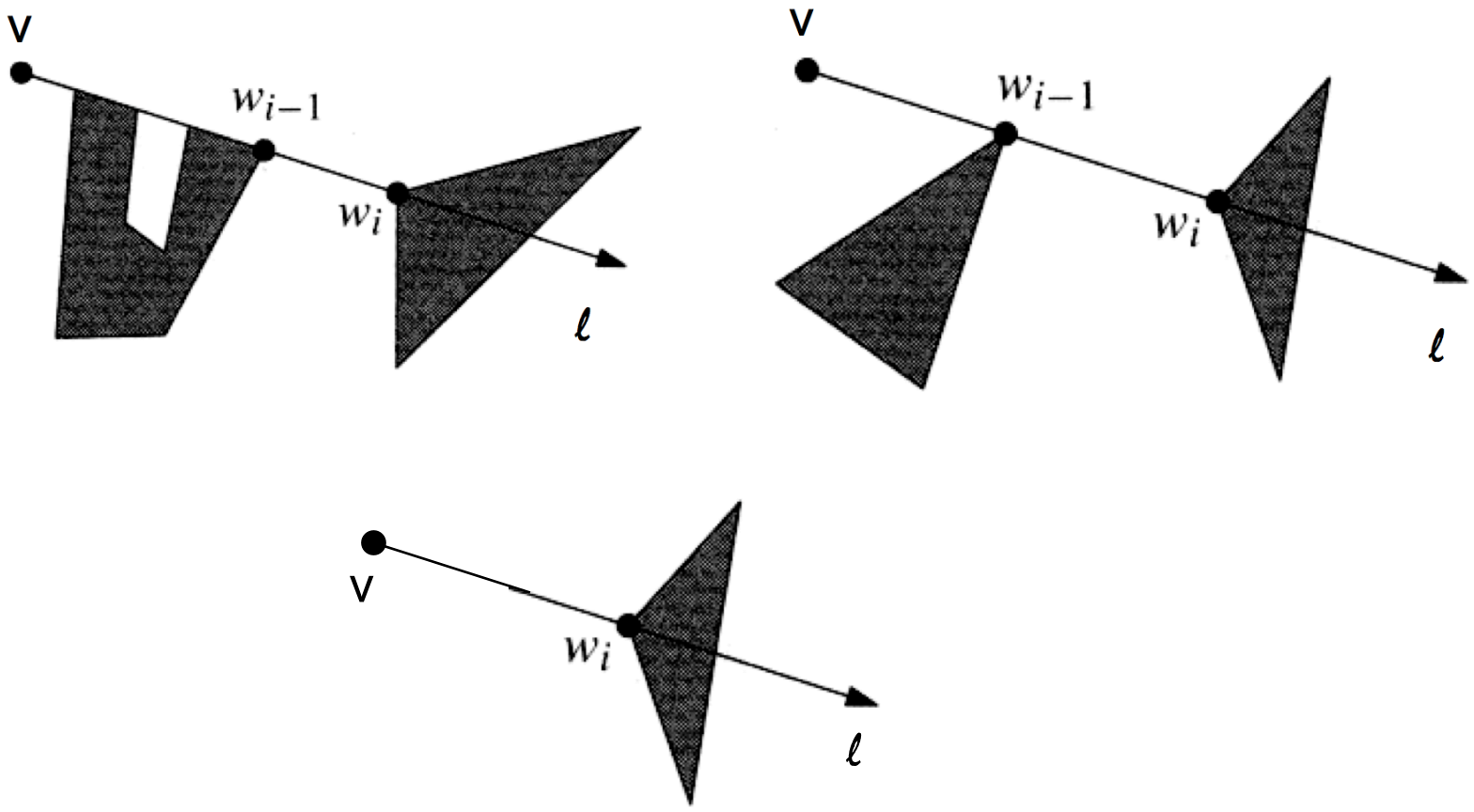
w_i ist nicht sichtbar, falls

(4) w_{i-1} liegt auch auf der Sweep-Line ℓ und w_i und w_{i-1} gehören zum selben Hindernis (Polygonzug) und die Kante (w_{i-1},w_i) schneidet das Hindernis.



Linienbezogene Sichtbarkeitsprüfung (4)

w_i ist sichtbar in allen anderen Fällen:



Laufzeit des Sweep-Line-Verfahrens

Algorithmus VisibleVertices(v, P):

- Zeile (1): Sortieren aller n Ecken in O(n log n)
- Zeile (2): Einfügen in Suchbaum T kostet O(log n); also insgesamt O(n log n)
- for-Schleife (4): n-mal Zugriff auf Suchbaum T; also T(n log n)

Insgesamt: T(n) = O(n log n).

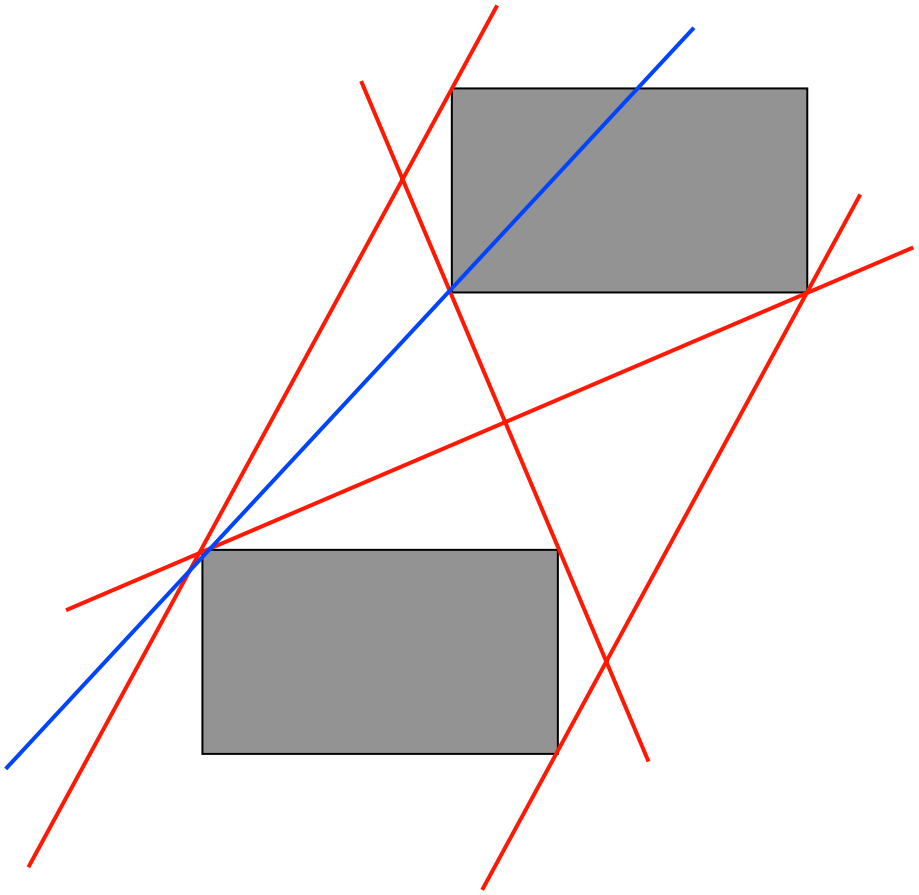
Algorithmus VisibiltyGraph(P, q_{Start}, q_{Ziel})

Insgesamt: T(n) = $O(n^2 \log n)$.

Reduzierter Sichtbarkeitsgraph (1)

Reduzierung unnötiger Kanten im Sichtbarkeitsgraph

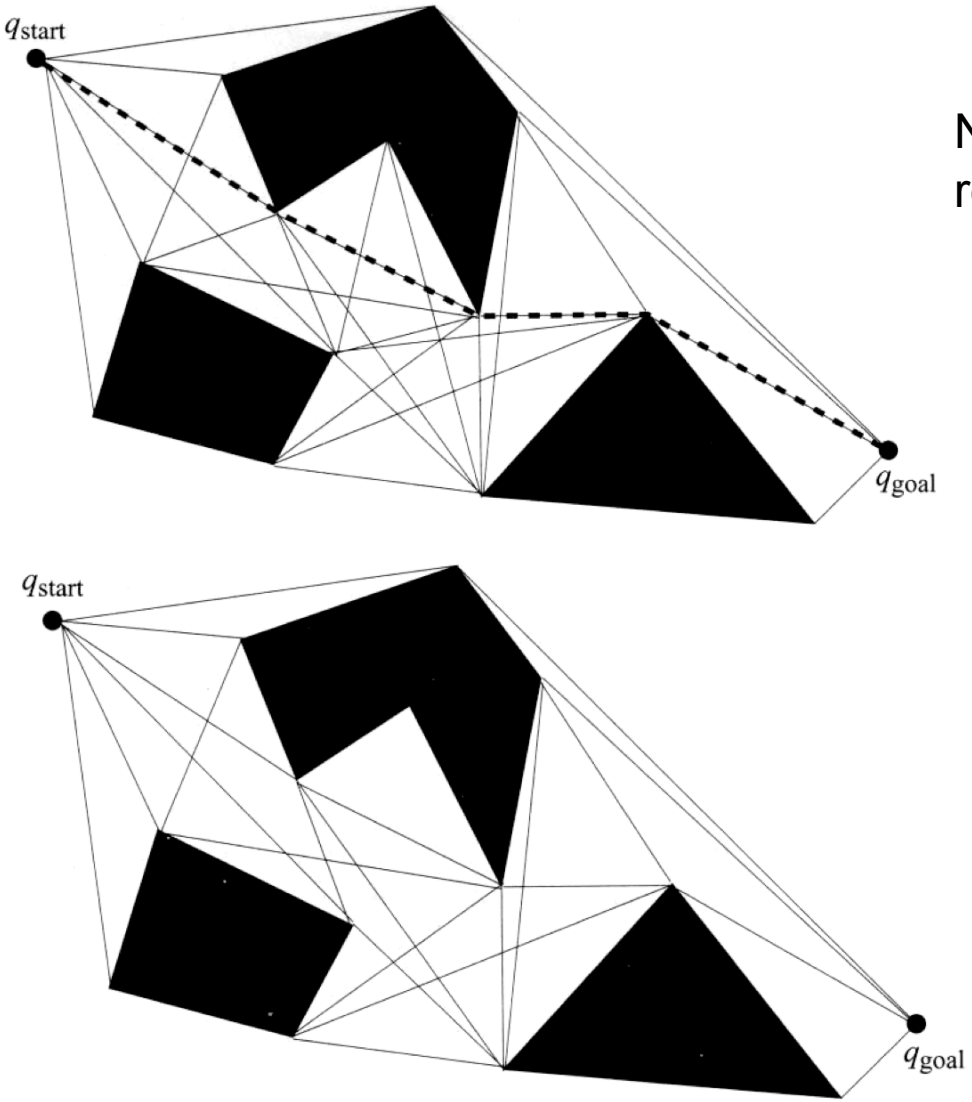
Alle Kanten, die nicht auf einer „2-Punkt-Tangente“ liegen werden entfernt



Rot: alle 2-Punkt-Tangenten

Blau: Beispiel für Nicht-Tangente

Reduzierter Sichtbarkeitsgraph (2)



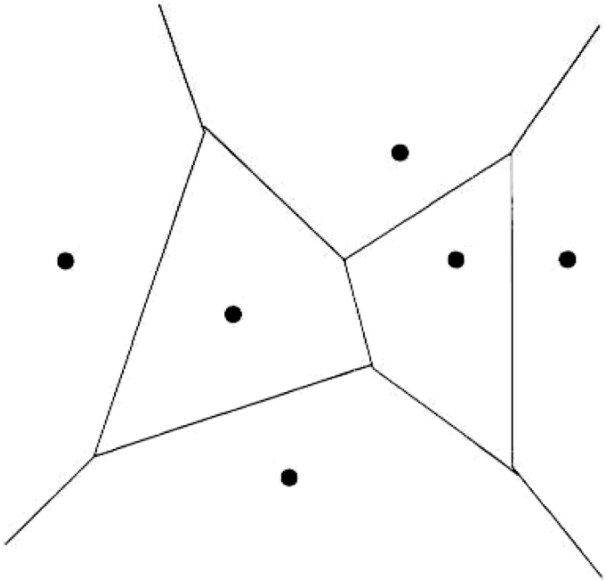
Nicht-reduzierter und
reduzierter Sichtbarkeitsgraph

Wegeplanung: Wegekartenverfahren

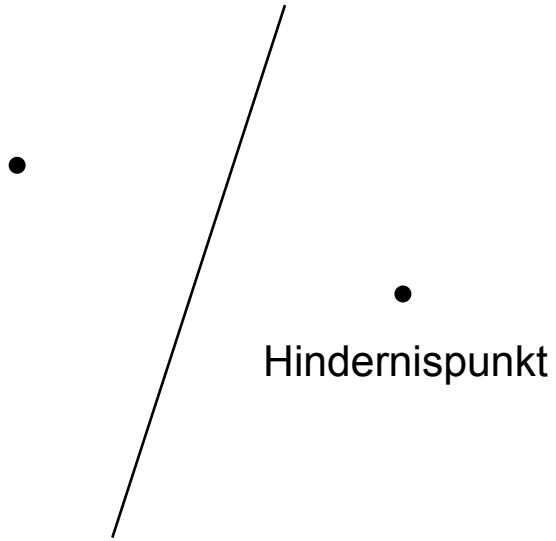
- Idee
- Sichtbarkeitsgraph
- Voronoi-Diagramm
- Probabilistische Wegekarten
- Rapidly-Exploring Random Tree

Voronoi-Diagramm

Ein Voronoi-Diagramm VD(P) für eine Menge von Hindernispunkte P teilt die Ebene in (Voronoi-)Regionen auf, die jeweils alle Punkte enthalten, die zum gleichen Hindernispunkt am nächsten gelegen sind.



VD mit 6 Regionen.



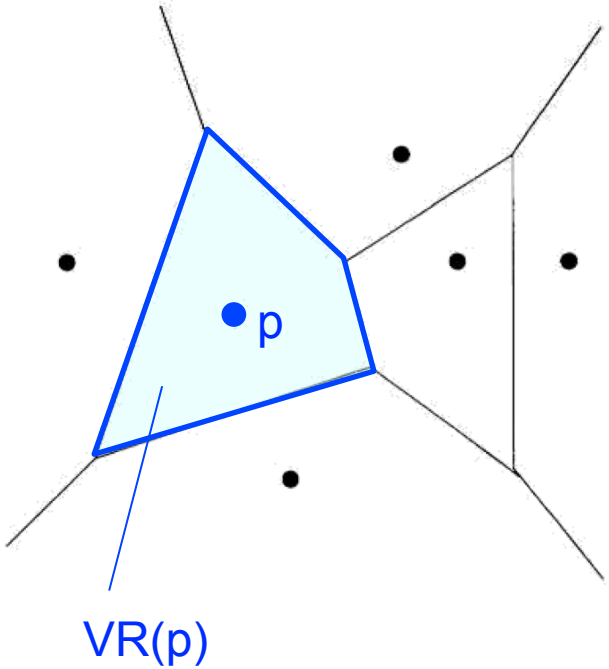
VD mit 2 Regionen.

Definition der Voronoi-Region

- Menge von Hindernispunkte $P = \{p_1, p_2, ..., p_n\}$
- Voronoi-Region zu einem Hindernispunkt $p \in P$:

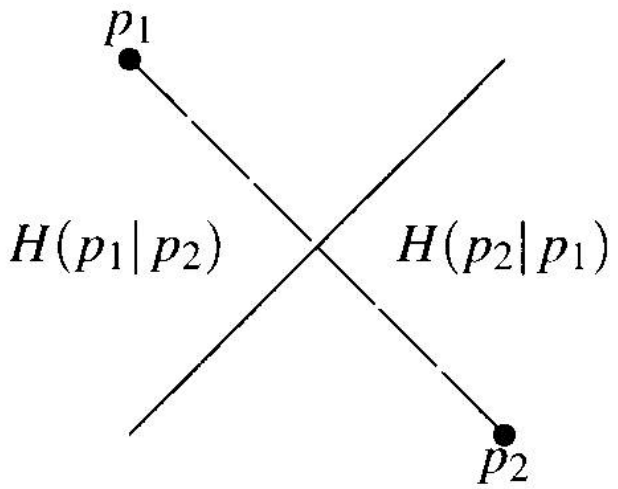
$$VR(p) = \{x \in R^2 \mid d_p(x) \leq d_q(x) \text{ für alle } q \in P \text{ mit } q \neq p\}$$

wobei $d_p(x)$ der Euklidische Abstand vom Punkt x zu Punkt p ist.



Charakterisierung einer Voronoi-Region als Polygon (1)

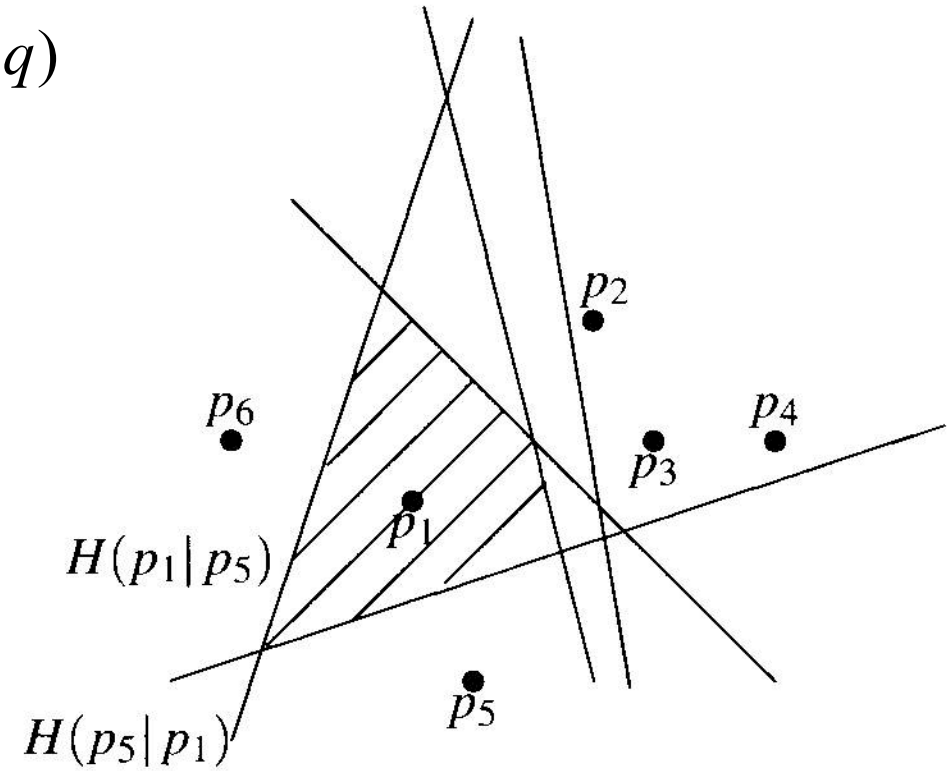
- $H(p_1|p_2)$ = Halbebene von p_1 mit p_2.
  Besteht aus allen Punkten, die näher bei p_1 als bei p_2 liegen.
- $H(p_2|p_1)$ analog definiert.



Charakterisierung einer Voronoi-Region als Polygon (2)

Für p ∈ P ist die Voronoi-Region VR(p) der Durchschnitt aller Halbebenen von p mit den anderen Punkten aus P:

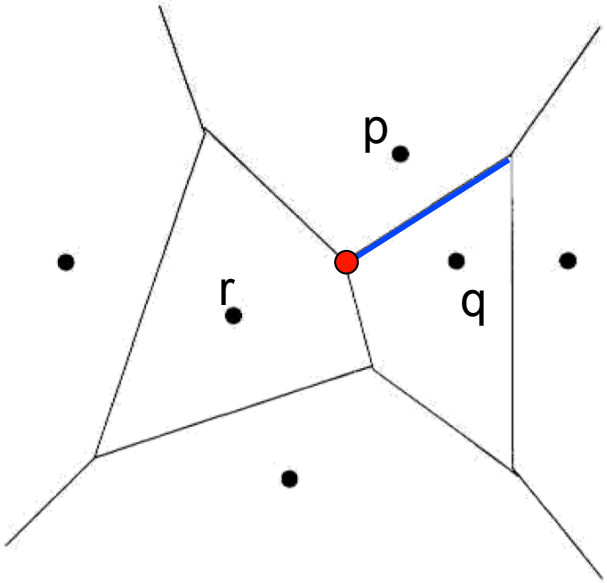
$$VR(p) = \bigcap_{q \in P \setminus \{p\}} H(p \mid q)$$



$VR(p_{1)}$ ist schaffriert.

Voronoi-Kanten und -Ecken

- Grenzlinien zwischen Voronoi-Regionen sind Kanten und werden auch Voronoi-Kanten genannt. Die Enden der Voronoi-Kanten werden auch Vornoi-Ecken genannt.
- Kanten können Strecken, Strahlen oder Geraden sein
- Oft wird unter einem VD auch die Menge alle Voronoi-Kanten verstanden.
- Die Voronoi-Ecken sind von 3 Hindernispunkte gleichweit entfernt;
  die Punkte einer Voronoi-Kante sind von 2 Hindernispunkten gleichweit entfernt.

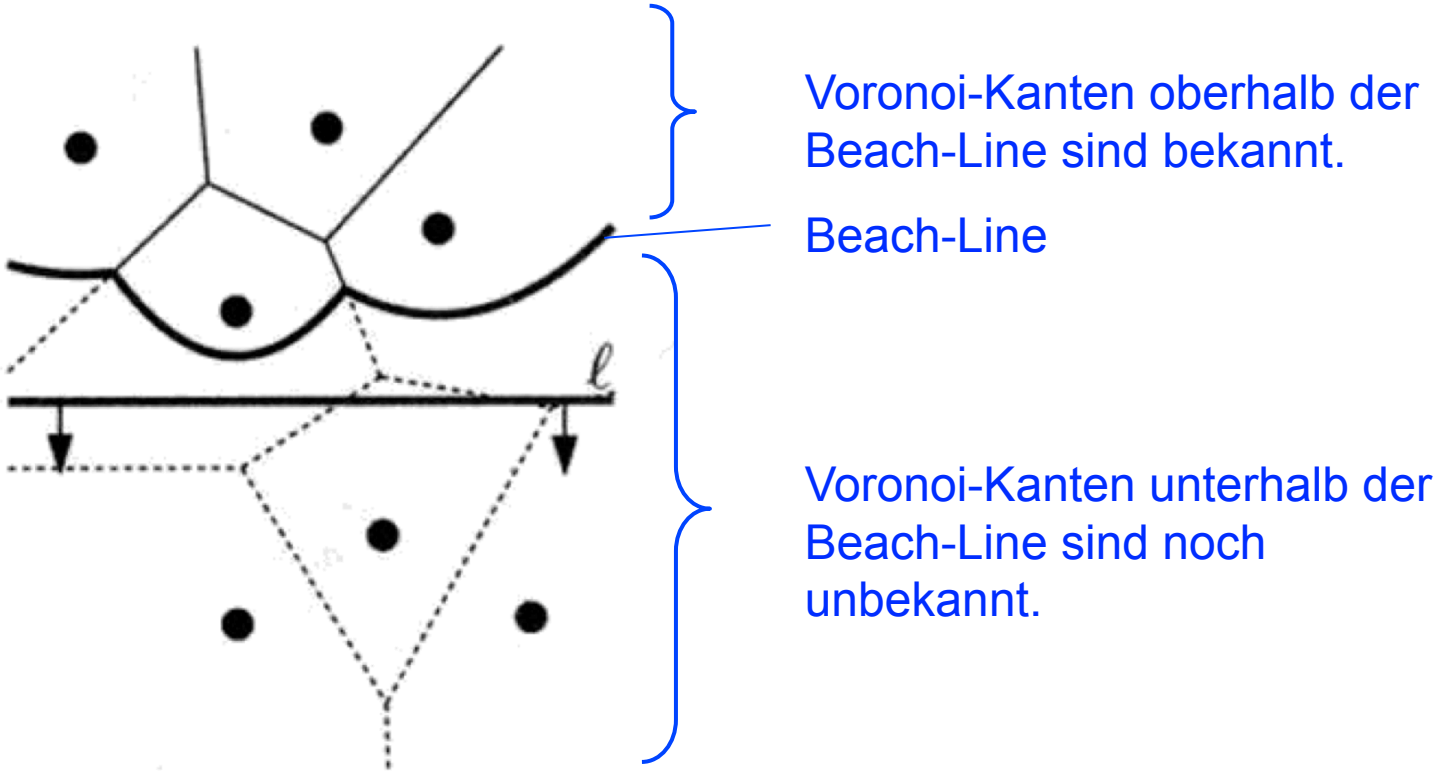


Voronoi-Kante;
alle Punkte der Kante sind von p und q gleichweit entfernt.

Voronoi-Ecke ist von
p, q und r gleichweit entfernt.

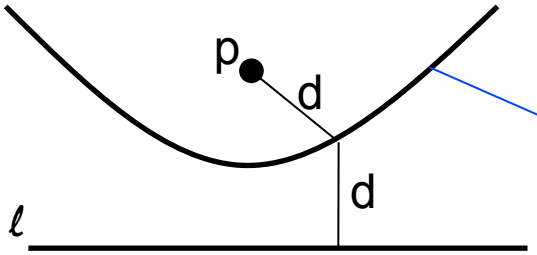
Sweep-Line-Verfahren

- Bewege horizontale Sweep-Line ℓ von oben nach unten.
- Dabei wird das VD für die oberhalb von ℓ liegende Halbebene bis zur sogenannten „Beach-Line“ konstruiert.
- Die Beach-Line ist die Menge aller Punkte, die von ℓ und dem nächsten Punkt aus P gleichweit entfernt sind.

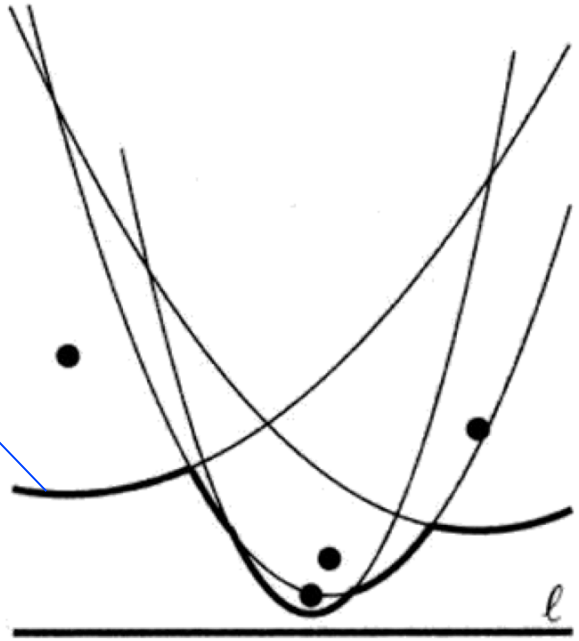


Beach-Line

- Die Beach-Line ist die Menge aller Punkte, die von ℓ und dem nächsten Punkt aus P gleichweit entfernt sind.
- Die Beach-Line besteht aus einer Folge von Parabelstücken. Für ein Hindernispunkt ist die Beach-Line eine Parabel.



Beach Line



Die Menge alle Punkte, die von ℓ und p gleicht weit entfernt sind, bilden eine Parabel.

Punkt- und Kreis-Ereignisse

Beim Bewegen der horizontalen Sweep-Line l
gibt es zwei Ereignisse:

- Punkt-Ereignis:

  $\Rightarrow$ es wird ein neues Parabelstück zur Beach-Line dazugefügt.

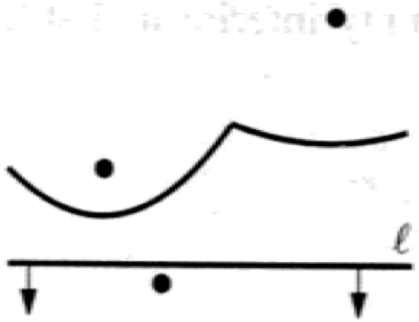
- Kreis-Ereignis:

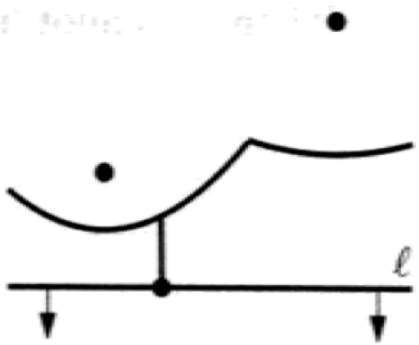
  $\Rightarrow$ es wird ein Parabelstück von der Beach-Line entfernt.

Punkt-Ereignis

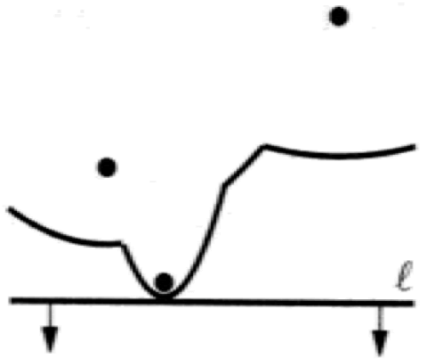
Sweep-Line ℓ trifft auf ein Punkt p∈P:

- es wird ein neues Parabelstück zur Beach-Line dazugefügt.





Eine neue Parabel mit Breite = 0 wird dazugefügt.

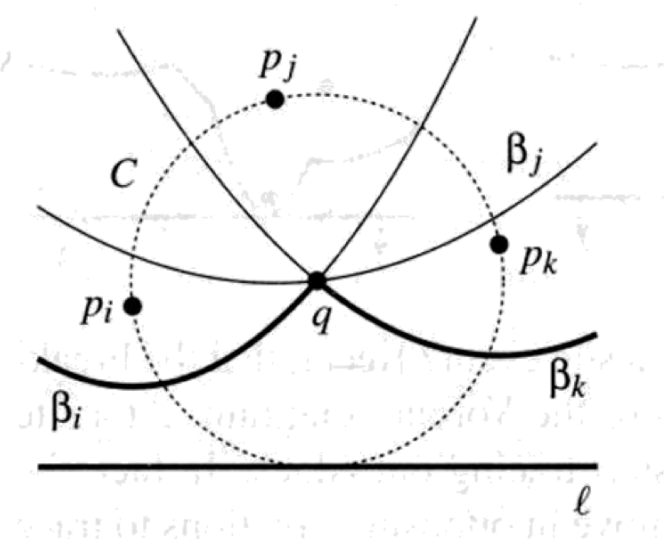


Parabel wird breiter.

- Beachte: die Verbindungsstellen der Parabelstücke bewegen sich auf den Voronoi-Kanten:

Kreis-Ereignis (1)

Sweep-Line ℓ trifft auf den unteren Punkt eines Kreises, der auf 3 Punkte aus P liegt:

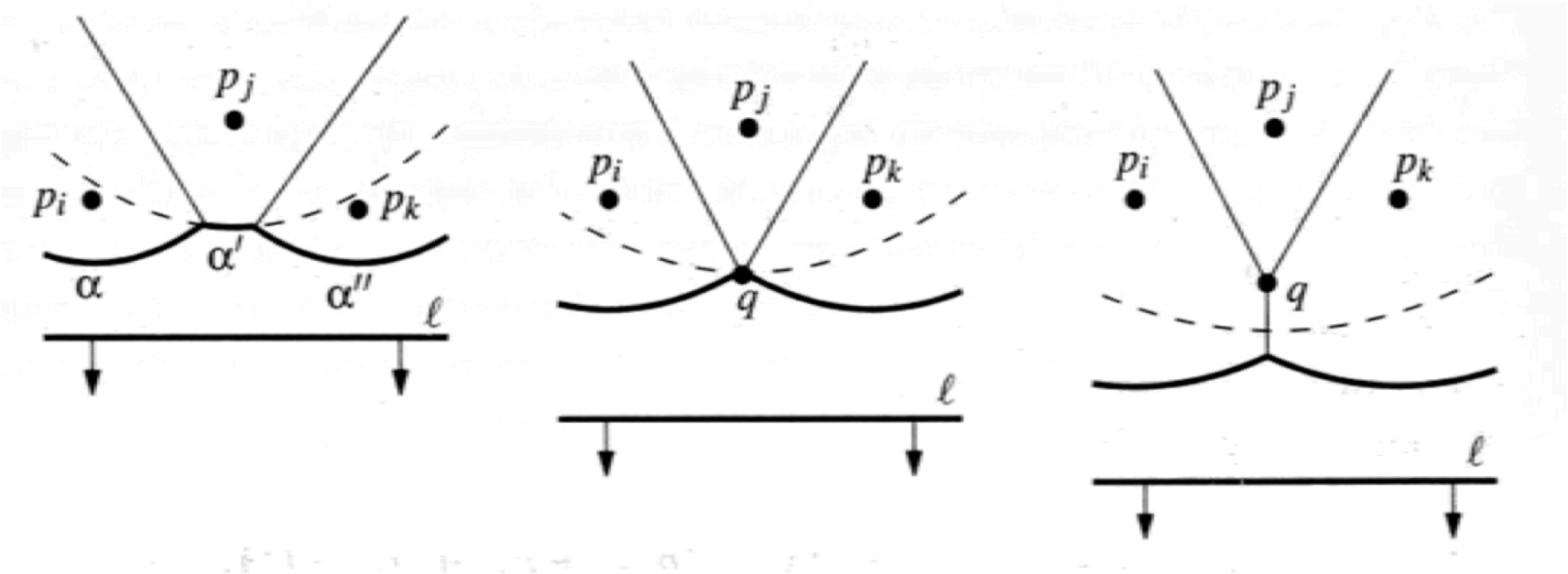


Sweep-Line ℓ trifft unteren Punkt eines Kreises, der auf den 3 Punkten p_i, p_j und p_k liegt.

- Damit ist der Mittelpunkt q des Kreises eine Voronoi-Ecke.

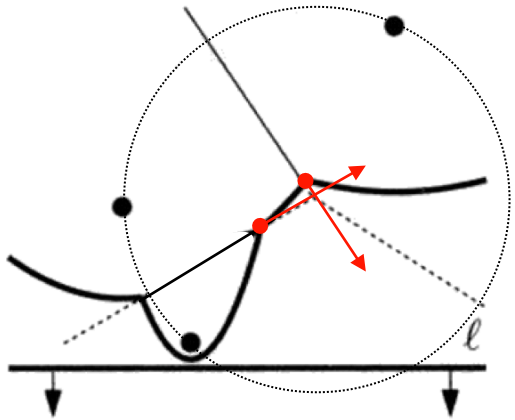
Kreis-Ereignis (2)

- Es wird ein Parabelstück von der Beach-Line entfernt (Bildmitte):



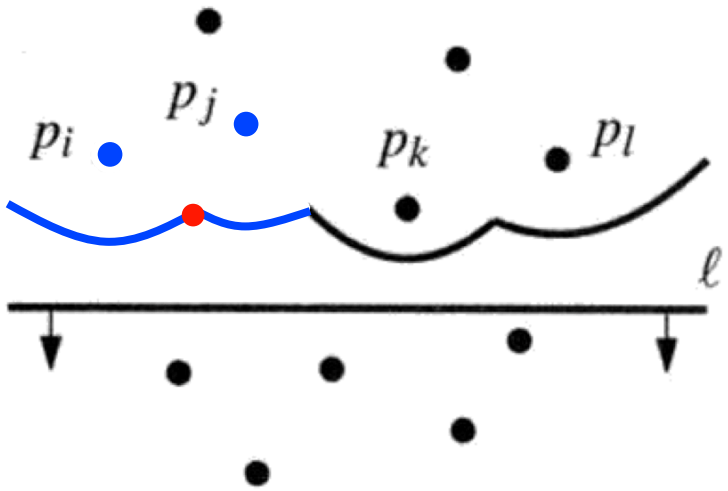
Steuerung der Sweep-Line-Bewegung

- Es wird eine Ereignis-Schlange Q mit allen Punkten aus P initialisiert (nach y-Koordinaten sortiert).
- Damit sind die Punkt-Ereignisse generiert.
- Sobald ein neues Parabelstück eingefügt oder gelöscht wird, wird geprüft, ob ein neues Kreis-Ereignis möglich wird. Falls ja, wird es in Q eingefügt.
- Prüfe dazu, ob sich die zwei Verbindungspunkte 3 aufeinanderfolgender Parabelstücke aufeinander zubewegen.
- Beachte: ein in Q eingefügtes Kreis-Ereignis muss evtl. später ignoriert werden.

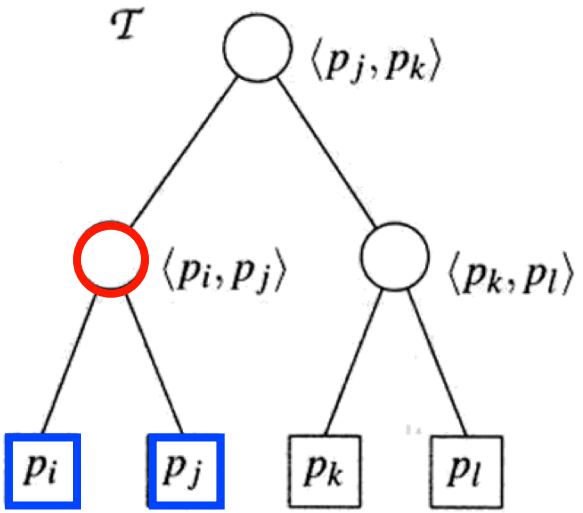


Die 3 gezeigten Punkte werden (eventuell) ein Kreis-Ereignis auslösen, da sich die zwei rot dargestellten Verbindungspunkte aufeinander zubewegen

Beach-Line als balanzierter Baum



Beach-Line mit Folge von Parabelstücke



Parabelstücke der Beach-Line in einem balanziertem Suchbaum

Laufzeit des Sweep-Line-Verfahrens

für n Hindernispunkte und Beach-Line als balanzierter Suchbaum:

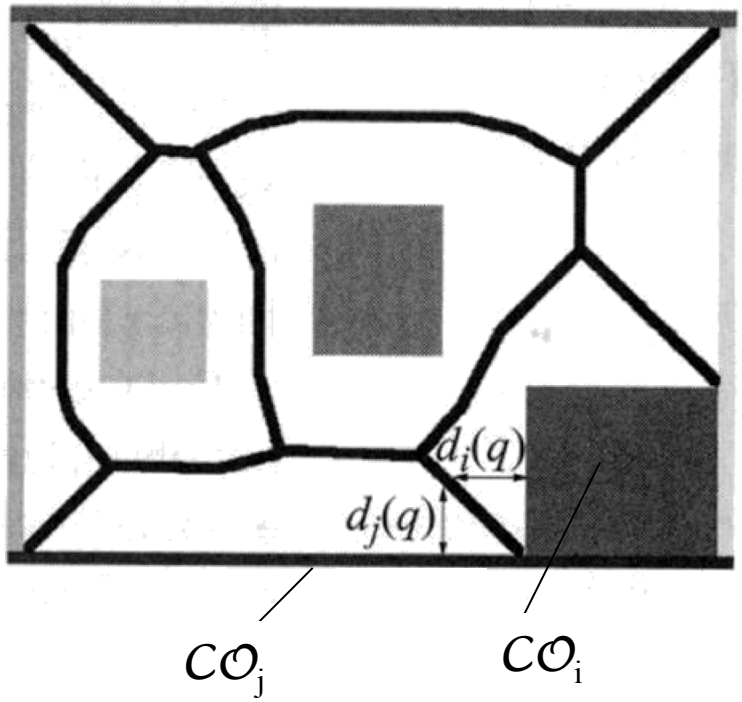
$T(n) = O(n \log n)$

Verallgemeinertes Voronoi-Diagramm

- Statt Punkte besteht die Umgebung nun aus beliebigen Hindernissen CO_i.
- Voronoi-Region zu einem Hindernis CO_i:

  VR(i) = {x∈C_{free} | $d_i(x) \leq d_j(x)$ für alle Hindernisse CO_j mit $i \neq j$ }

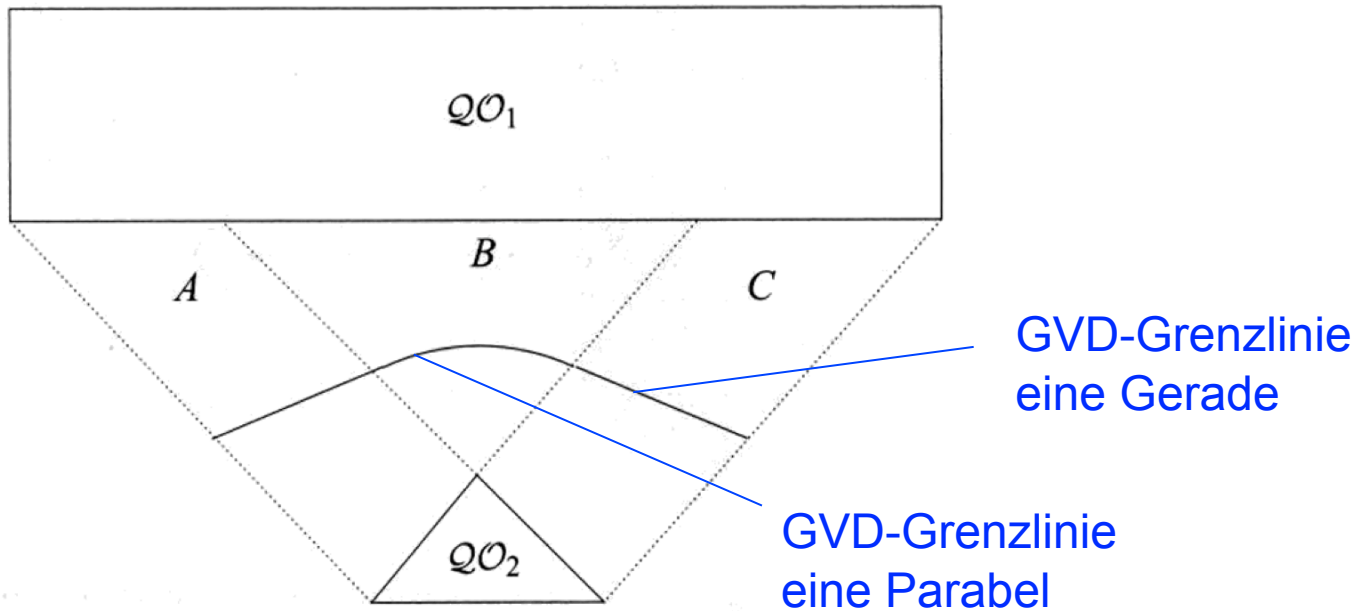
  wobei $d_i(x)$ der kürzeste Abstand vom Punkt x zum Hindernis CO_i ist.
- Ein Verallgemeinertes Voronoi-Diagramm GVD ist eine Zerlegung von C_{free} in Voronoi-Regionen.
- Oft versteht man unter GVD auch die Grenzlinien zwischen den Voronoi-Regionen.



GVD für polygonale Hindernisse

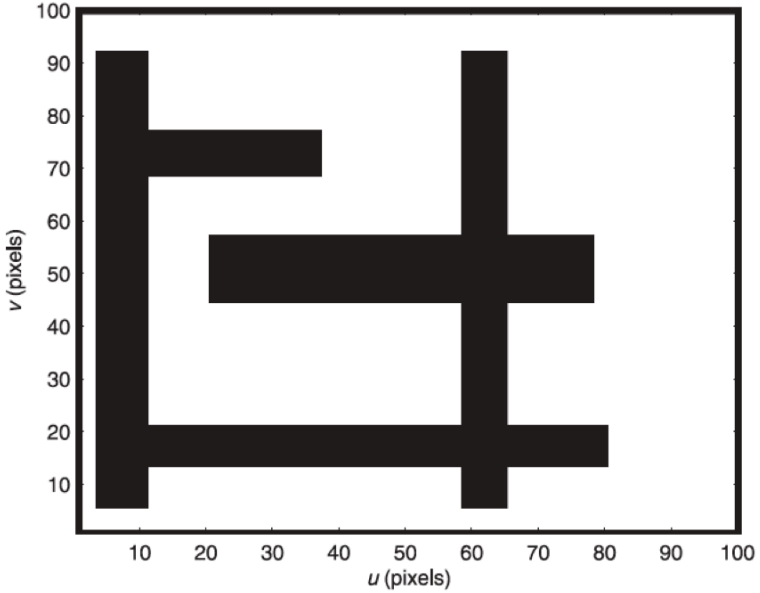
Freier Raum lässt sich in 2 Arten von Regionen zerlegen:

- die nächsten Hindernisteile sind Kanten (z.B. Region A und C)
  → GVD-Grenzlinie ist eine Gerade
- die nächsten Hindernisteile sind Kanten und Ecken sind (z.B: Region B)
  → GVD-Grenzlinie ist eine Parabel

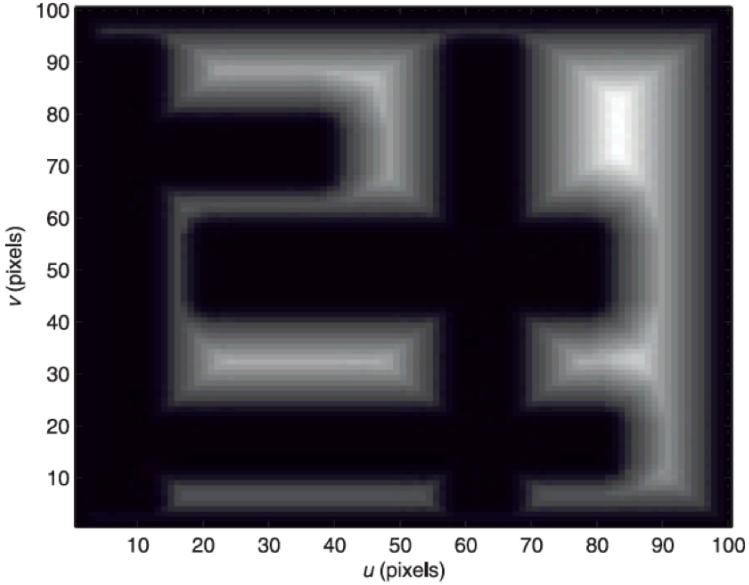


Approximative Berechnung der Voronoi-Diagramme (1)

- Erstelle zur Umgebung ein Belegtheitsgitter.
- Ermittle alle Randzellen. Randzellen sind Zellen, die zu einem Hindernisrand gehören.
- Starte nun das Dijkstra-Verfahren, wobei Open List mit allen Randzellen initialisiert wird. Der d-Wert der Randzellen wird auf 0 gesetzt.
- Das Dijkstra-Verfahren berechnet nun für alle Zellen den Abstand zu den nächstgelegenen Hindernissen.



Belegtheitsgitter

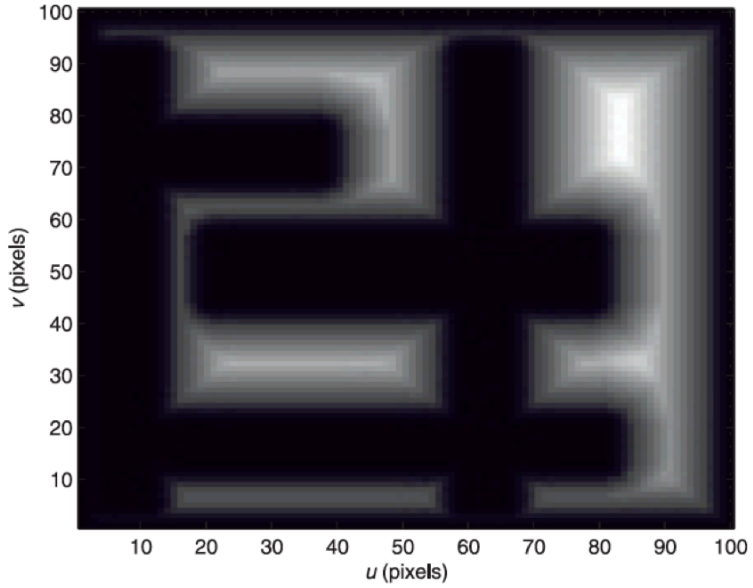


Gitter mit Entfernungswerten zu den nächstgelegenen Hindernissen

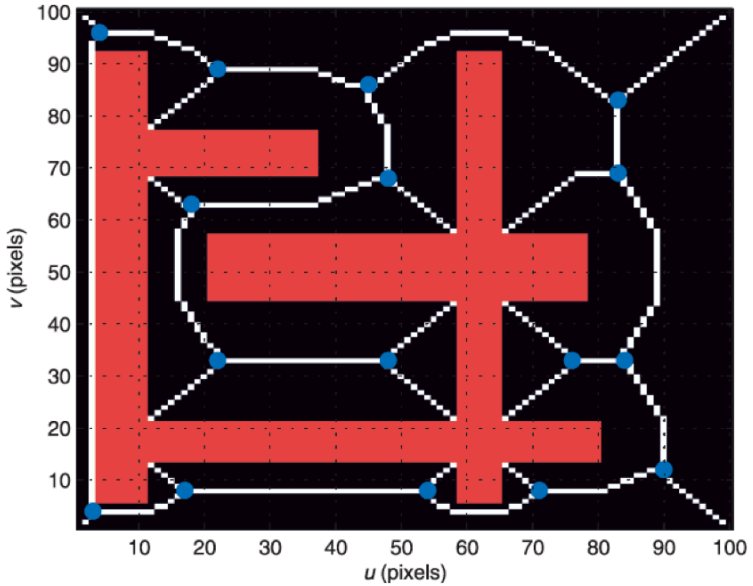
Bilder aus [Corke 2011]

Approximative Berechnung der Voronoi-Diagramme (2)

- Das Voronoi-Diagramm besteht aus allen Zellen, deren d-Wert ein lokales Maximum bilden.
- Voronoi-Diagramm kann zu Graph abstrahiert werden. Nehme Voronoi-Ecken als Knoten und Verbindungen als Kanten. Führe evtl. Zwischenknoten ein.



Gitter mit Entfernungswerten zu den nächstgelegenen Hindernissen



Voronoi-Diagramm. Voronoi-Ecken sind blau dargestellt. Bei Überführung in ein Graph können noch Zwischenknoten eingebaut werden.

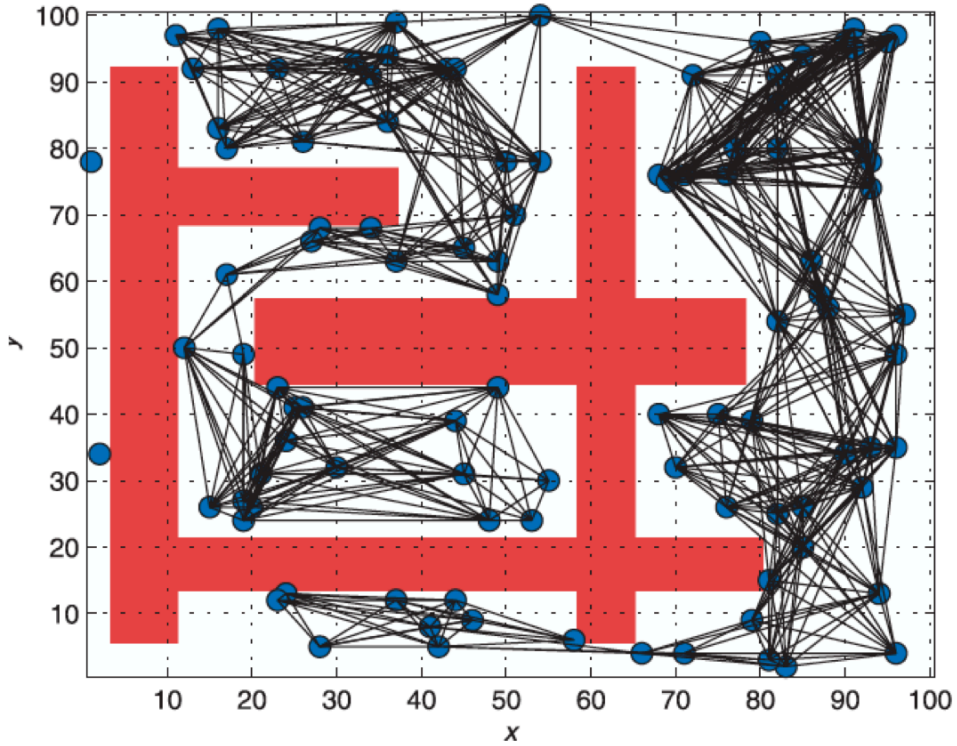
Bilder aus [Corke 2011]

Wegeplanung: Wegekartenverfahren

- Idee
- Sichtbarkeitsgraph
- Voronoi-Diagramm
- Probabilistische Wegekarten
- Rapidly-Exploring Random Tree

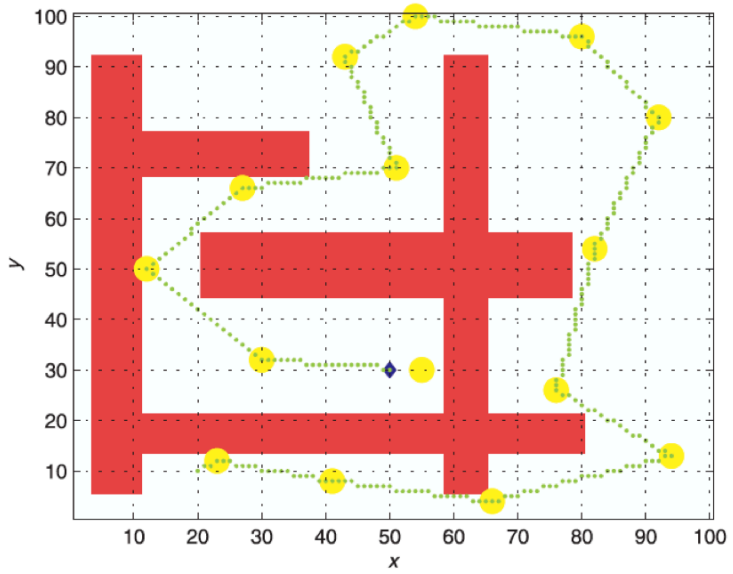
Probabilistische Wegekarten

- Beim PRM-Verfahren (Probabilistic Roadmap Method) werden zufällig Punkte auf den Freiflächen generiert.
- Punkte werden mit allen sichtbaren Nachbarknoten, die innerhalb einer bestimmten Entfernung liegen, mit einer Kante verbunden.
- In den PRM-Karten lassen sich mit den üblichen Graphenalgorithmen Wege finden.

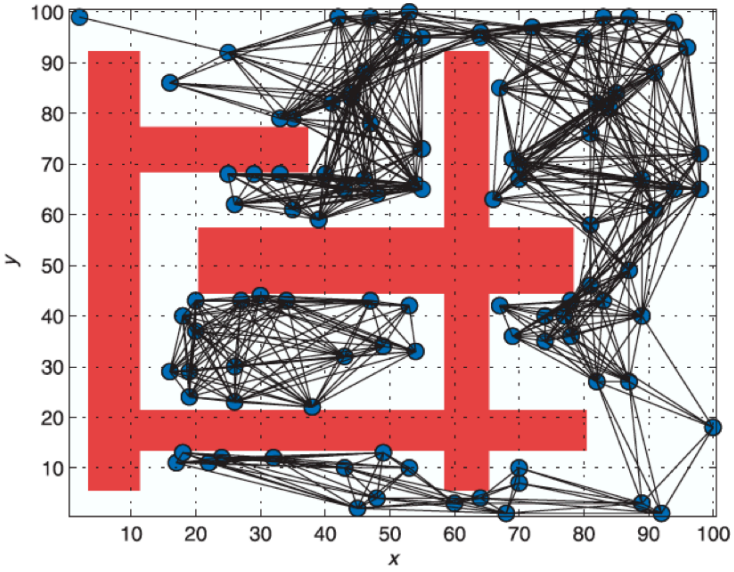


Bemerkungen

- Voronoi-Diagramme haben den Nachteil, dass große Freiflächen nur durch wenige Ecken und Verbindungen aufgespannt werden. Die Planung kann daher sehr ineffiziente Wege generieren.
- Das PRM-Verfahren führt bei großen Freiflächen zu einer wesentlich besseren Abdeckung.
- Jedoch werden möglicherweise kleine Freiflächen nicht abgedeckt oder ein nicht zusammenhängender Graph wird generiert. Das Verfahren kann dadurch unvollständig werden oder suboptimale Wege generieren.



Suboptimaler Weg aufgrund zu weniger Knoten in der Engstelle ganz links.



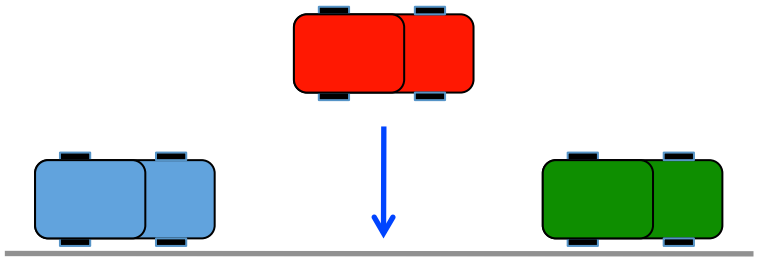
Nicht zusammenhängender Graph.

Wegeplanung: Wegekartenverfahren

- Idee
- Sichtbarkeitsgraph
- Voronoi-Diagramm
- Probabilistische Wegekarten
- Rapidly-Exploring Random Tree

Probleme bei 2-dimensionalen Planungen

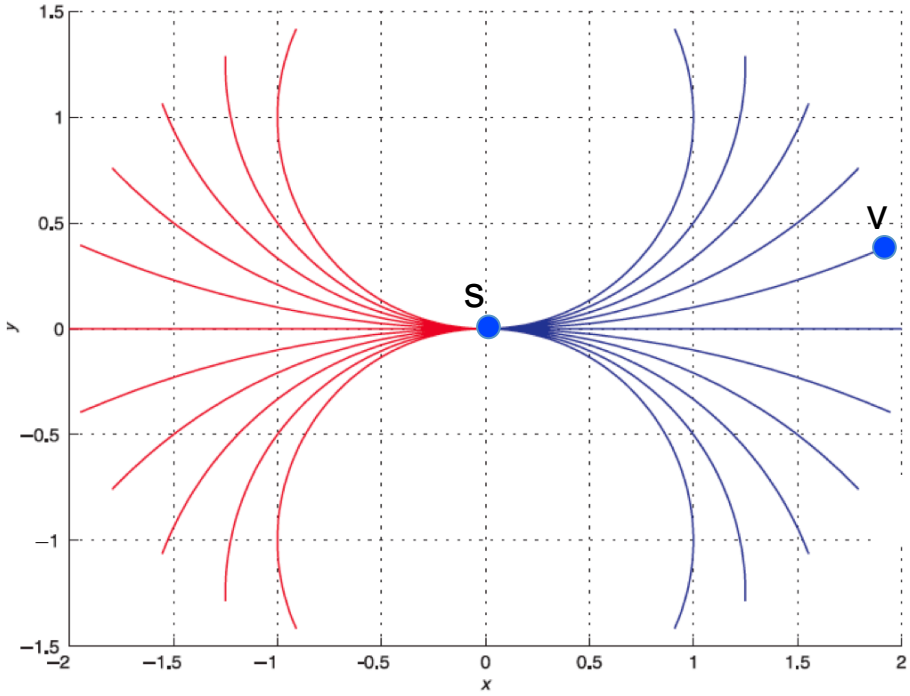
- Hat ein mobiler Roboter kinematische Einschränkungen, dann kann oft ein in der x-y-Ebene geplanter Weg nicht direkt vom Roboter abgefahren werden.
- Beispiel: Auto-Roboter, der seitwärts einparken soll.



- Eine einfache 2-dimensionale Planung (z.B. A* in einem Belegtheitsgitter) würde einen direkten Weg seitwärts in die Parklücke finden (blauer Pfeil). Dieser Weg könnte zwar von einem Roboter mit omnidirektionalem Antrieb aber nicht von einem Auto-Roboter abgefahren werden.

Planung in höher-dimensionalen Räumen

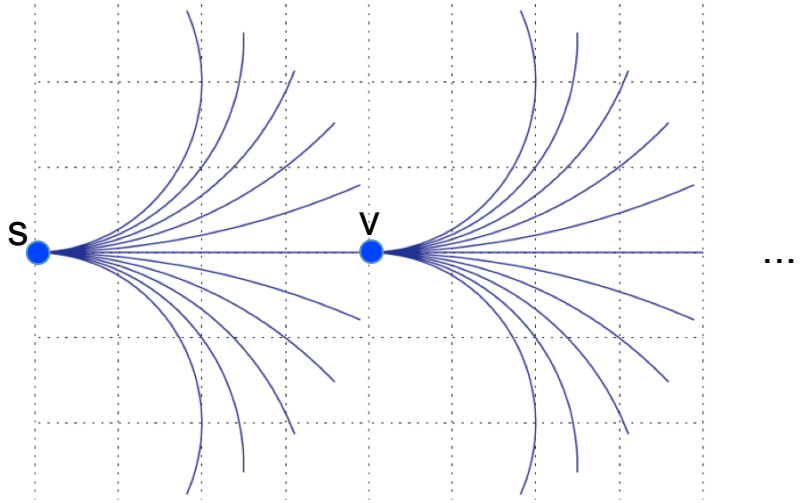
- Planung der kompletten Pose (x,y,θ).
  Kinematische Einschränkung des Roboters wird berücksichtigt.
- Planung in einem Graphen mit Posen als Knoten.
  Zwei Posen sind durch eine Kanten verbunden, falls die eine Pose aus der anderen Pose durch einen einfachen Steuerbefehl erreichbar ist.
- Das folgende Bild zeigt für die Start-Pose s=(0,0,0°) eines Auto-Roboters verschiedene mögliche Folge-Posen v, die durch einfache Steuerbefehle erreichbar sind.



- Verschiedene Pfade für ein Auto-Roboter, der in der Pose s = (0,0,0°) startet.
- Pfade für eine Periode von T = 2 sec und verschiedenen Lenkwinkeln $\alpha \in [-1, 1]$.
- v = -1 m/sec (rot) und v = +1 m/sec (blau).
- Bild aus [Corke 2011].

Planung in höher-dimensionalen Räumen (2)

- Problem: Anzahl der Knoten.
  Würde auf eine Folge-Pose v wiederum die verschiedenen einfachen Steuerbefehle angewendet, würde sich ein kombinatorisch explodierender Graph ergeben.
- Beim RRT-Verfahren wird dieses Problem umgangen, indem vergleichsweise wenige Posen zufällig generiert werden.



Rapidly-Exploring Random Tree (RRT-Verfahren)

Algorithmus RRT(x_Start, x_Target)

initialisiere Baum RT mit Start-Pose x_Start;

for i = t **to** K **do**

(1) erzeuge zufällige Pose x_rand;

(2) bestimme in RT diejenige Pose x_near, die am nächsten an x_rand liegt;

(3) bestimme einfachen Steuerbefehl u, der x_near möglichst nahe an x_rand führt;

(4) x_new ergibt sich durch Anwendung von Steuerbefehl u auf x_near;

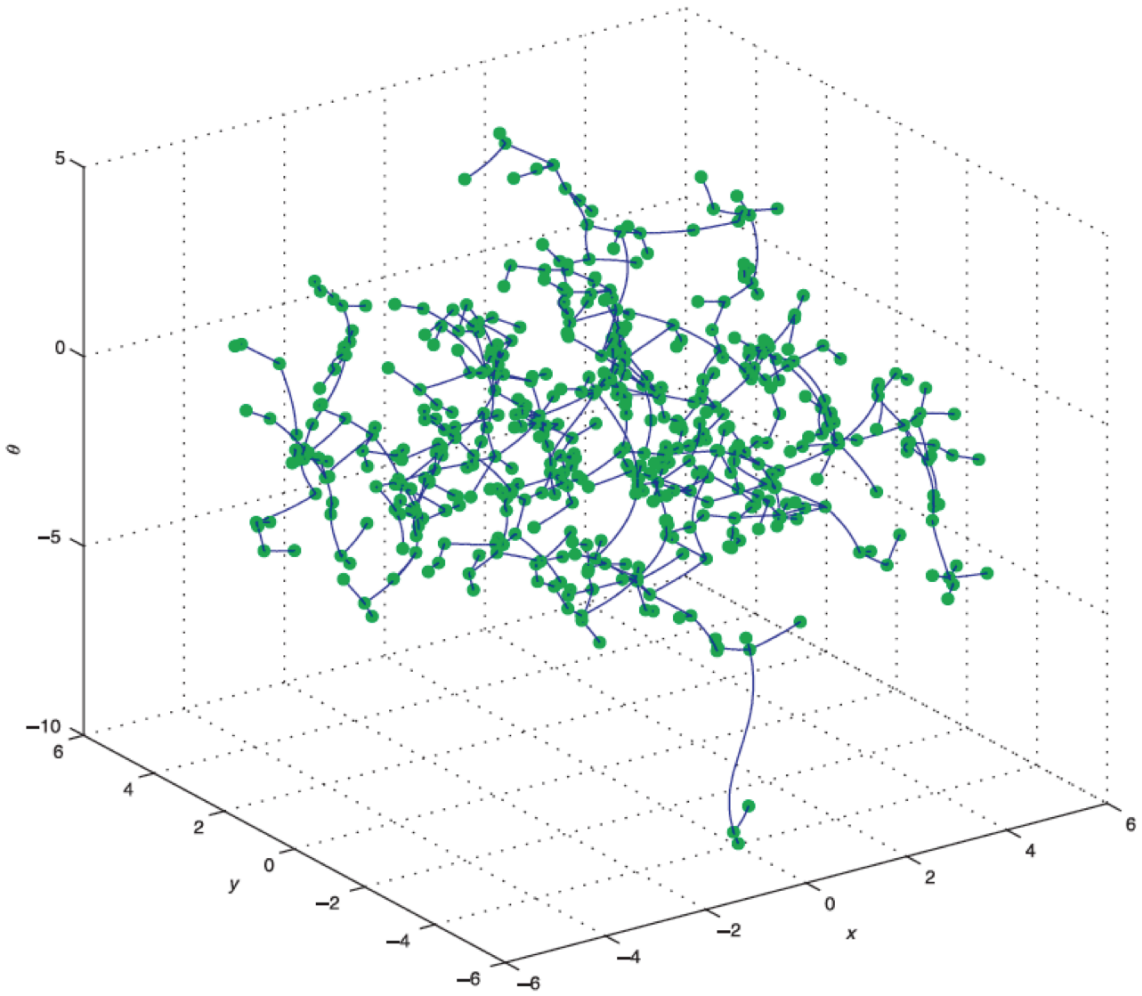
(5) erweitere RT um x_new;

(6) breche ab, falls sich von x_new die Ziele-Pose x_Target ungefähr erreichen lässt;

Steuerbefehl u würde beim Auto-Roboter aus v und Lenkwinkel α bestehen.

Der Steuerbefehl wird über eine feste Zeitperiode T durchgeführt.

Beispiel für RRT-Verfahren (1)



- Grüne Punkte:
  im RRT-Verfahren
  erzeugte Posen (x,y,θ)
- Start-Pose = (0, 0, 0°)
- Ziel-Pose = (0, 2, 0°)
- K = 500 Knoten
- v = ± 1 m/sec.
- T = 1 sec
- Lenkwinkel $\alpha \in [-1.2, 1.2]$.
- aus [Corke 2011].

Beispiel für RRT-Verfahren (2)

- Aus dem Baum gewonnenen Weg für Start-Pose = (0, 0, 0°) und Ziel-Pose = (0, 2, 0°).
- Beachte: Fahrzeug fährt rückwärts.
- Ziel-Pose wird aufgrund des probabilistischen Ansatzes nur ungefähr erreicht.