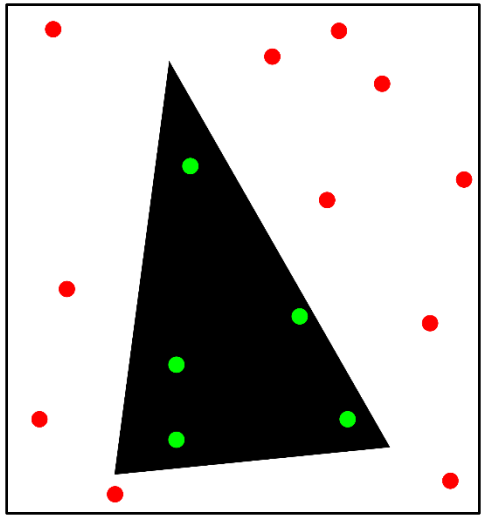
Technische Universität Braunschweig

Einführung in algorithmische Geometrie

Arne Schmidt

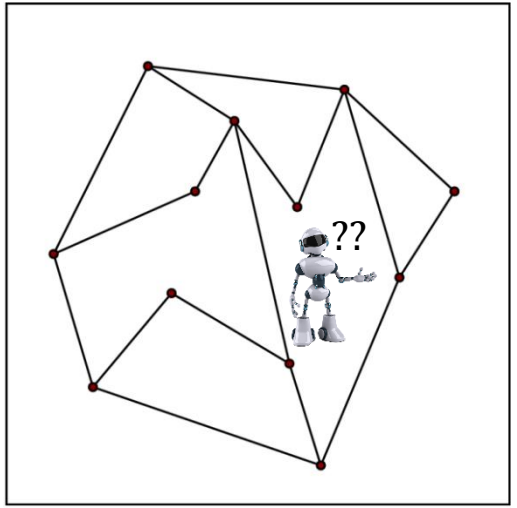
Zusammenfassung

Probleme



Welche Punkte liegen in dem Bereich?

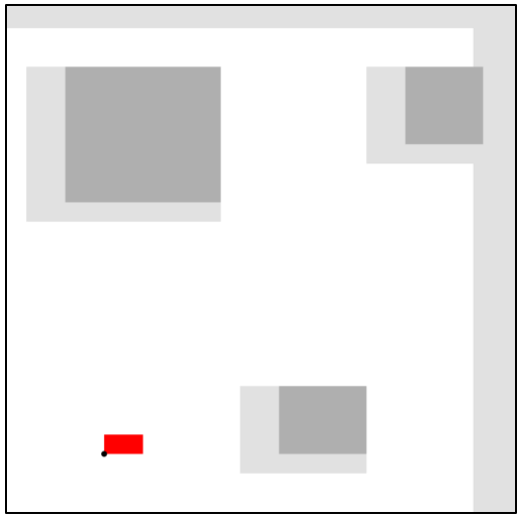
→ Range Query



In welchem Bereich liegt ein Punkt?

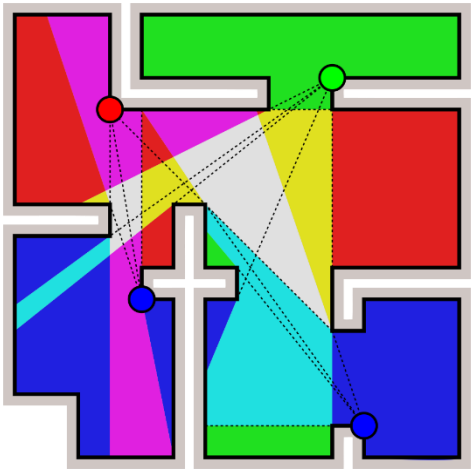
→ Punktlokalisierung

Probleme



Wohin und wie kann das rote Objekt bewegt werden?

→ Robot Motion Planning



Wie viele Wächter werden zur Abdeckung benötigt?

→ Art Gallery Problem

Punkte und Segmente

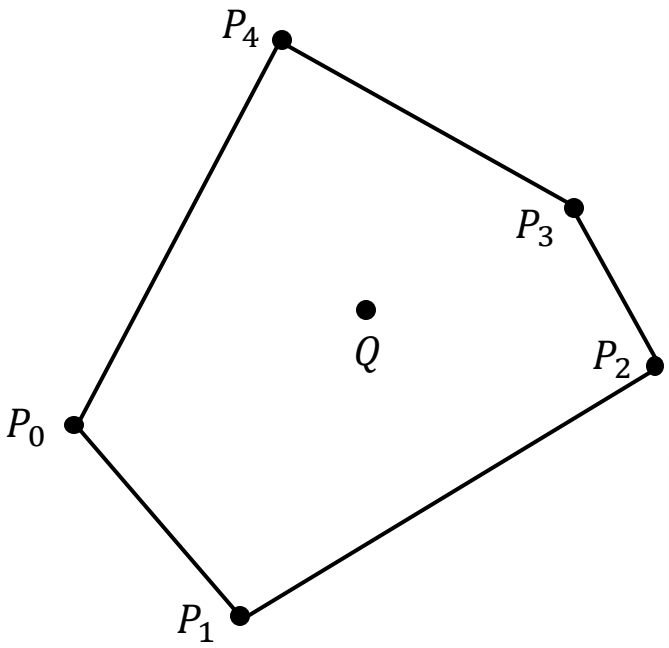
Definition 1.5:
Seien $P, S, T \in \mathbb{R}^2$. Wir definieren

$$ccw(S,T,P) \coloneqq \det\begin{pmatrix} 1 & S.x & S.y \\ 1 & T.x & T.y \\ 1 & P.x & P.y \end{pmatrix}$$

Theorem 1.6:
Seien $P, S, T \in \mathbb{R}^2$. Dann ist STP

- ein links-Knick, wenn $ccw(S,T,P) > 0$
- ein rechts-Knick, wenn $ccw(S,T,P) < 0$
- kolinear, wenn $ccw(S,T,P) = 0$

Konvexe Polygone



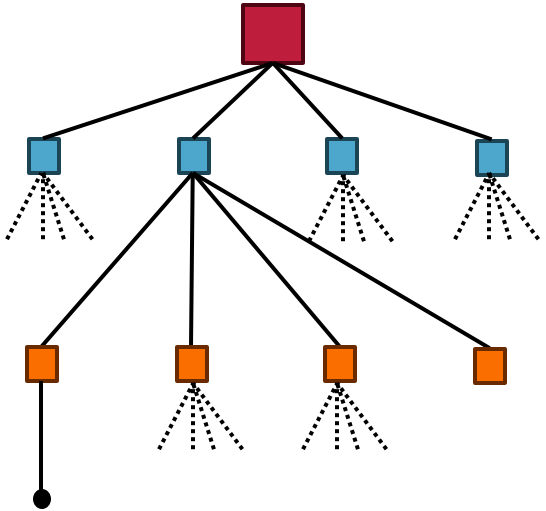
Definition 1.9 (Polygone):
Sei $P := (P_0, \dots, P_{n-1})$ eine Folge von n Punkten. Dann heißt P *einfaches Polygon*, wenn für jedes Paar P_i, P_j mit $j \notin \{i-1,\ i,\ i+1\}$ die Segmente P_iP_{i+1} und P_jP_{j+1} sich nicht schneiden.

P heißt *konvexes* Polygon, wenn jedes Tripel P_i, P_{i+1}, P_{i+2} ein links-Knick bildet (jeder Innenwinkel ist als maximal 180°).

Achtung: Die Indizes sind immer mod n zu verstehen.

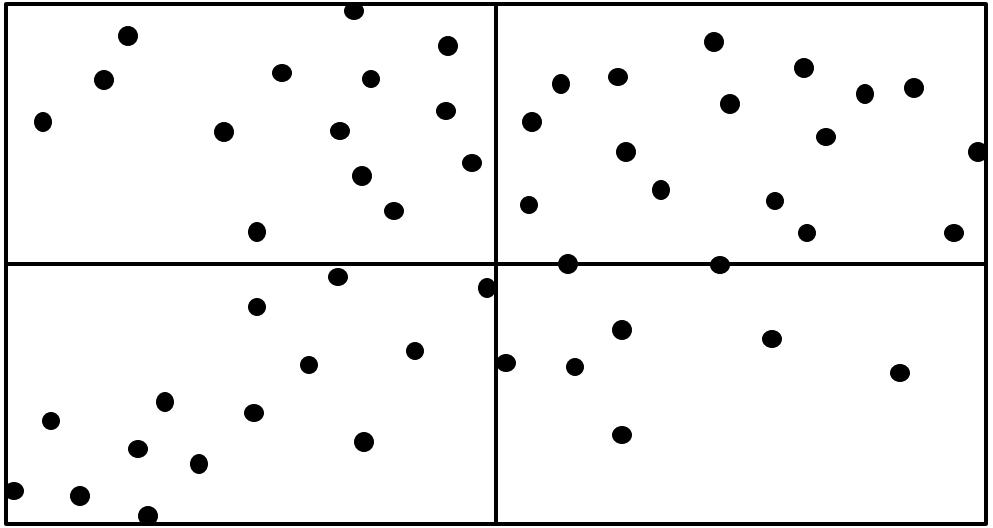
Lemma 1.10:
Sei $P := (P_0, \dots, P_{n-1})$ ein konvexes Polygon und Q ein Punkt. Q liegt genau dann innerhalb von P, wenn für alle Paare P_i, P_{i+1} gilt: $ccw(P_i, P_{i+1}, Q) \geq 0$

Kapitel 2.1.1 – Quadtrees



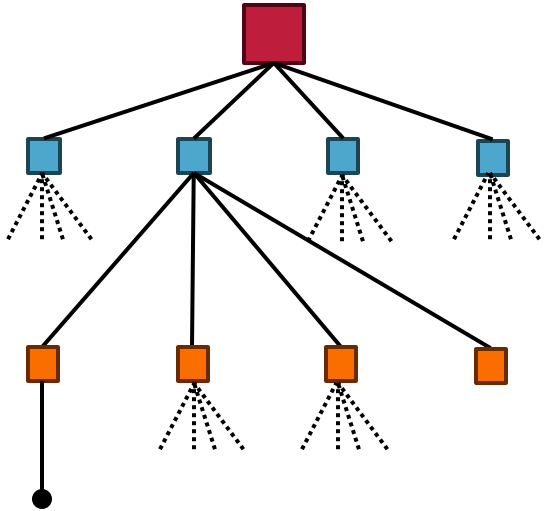
Rekursives Teilen!

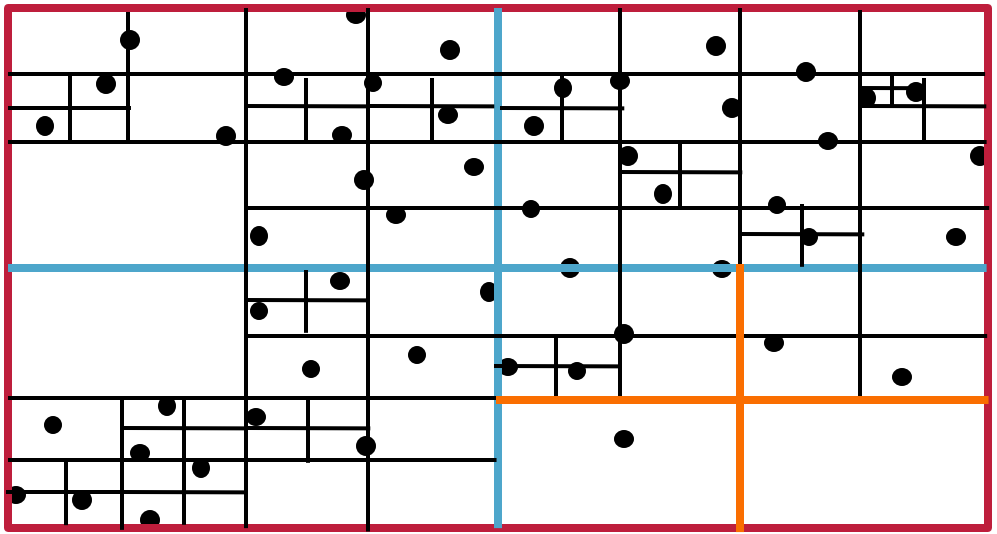
Da wir in zwei Dimensionen sind, können wir gleichzeitig die Breite und Höhe in zwei Teile scheiden!



Hierarchy von Räumen

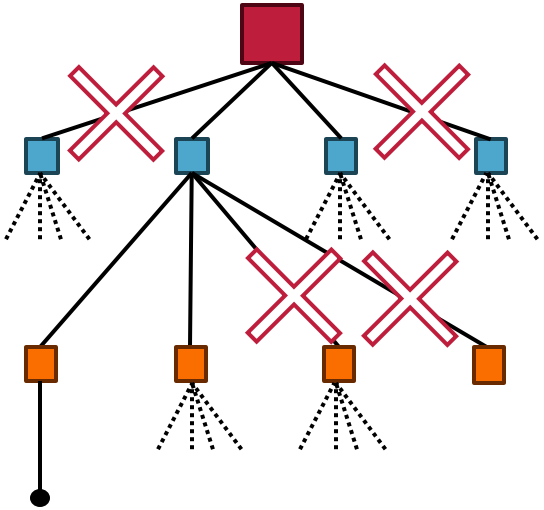
Stelle die unterteilten Räume als Knoten dar!

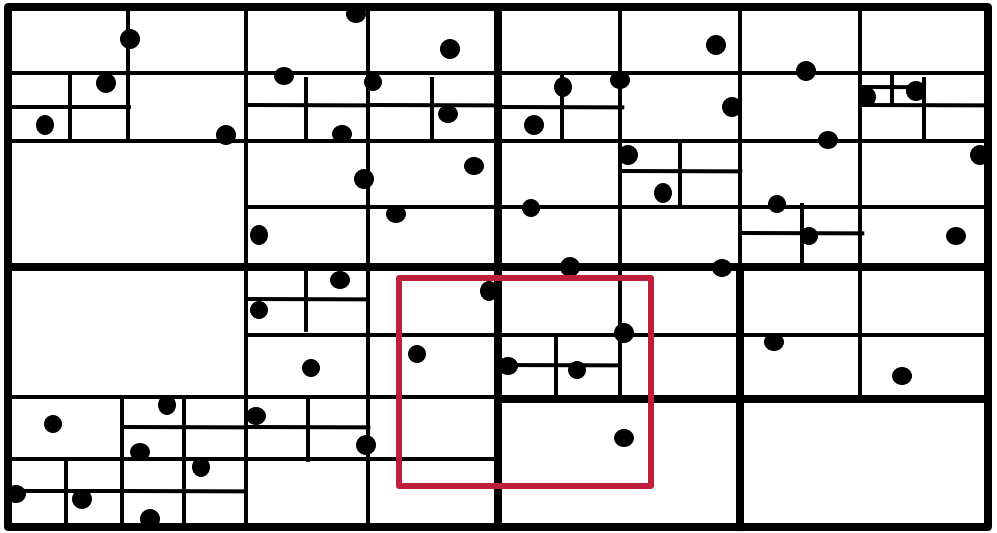




Jeder Punkt ist ein Blatt. Ist ein Raum leer, besitzt er keine Kinder (und ist auch ein Blatt).

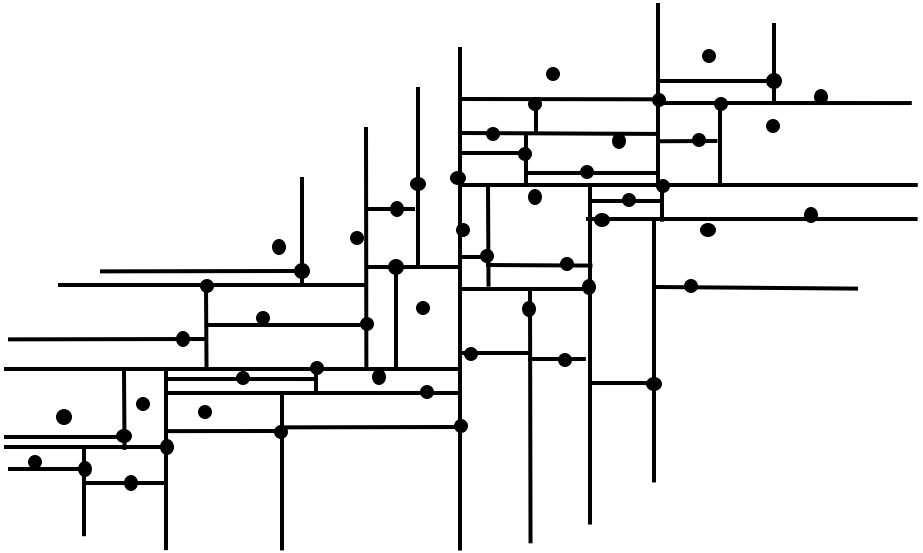
Range Query mit Quadtrees



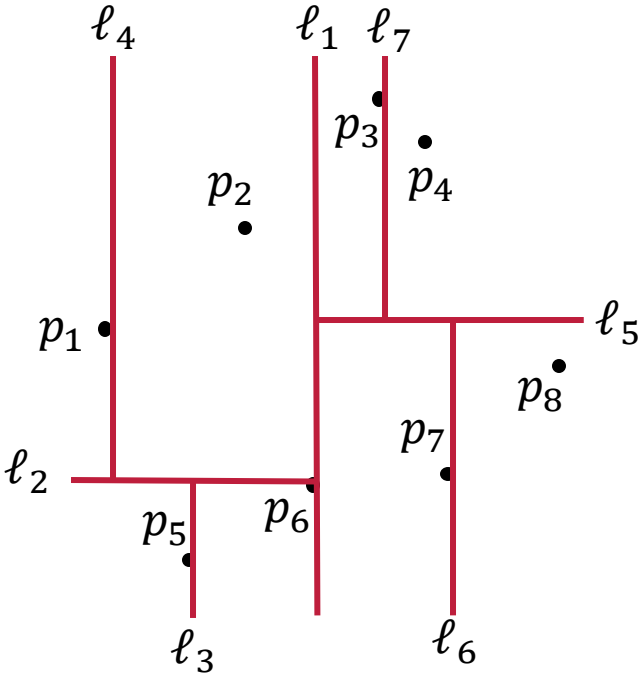


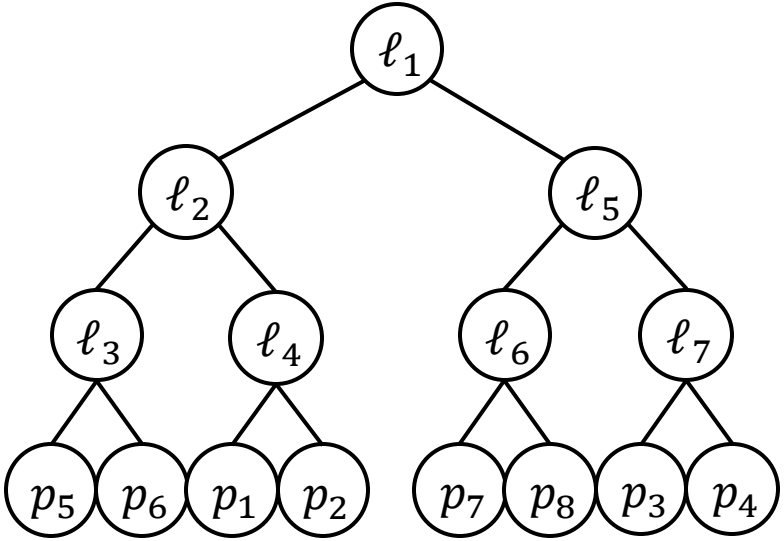
Welche Kinder überlappen sich mit dem Range-Query-Rechteck?
Suche auf diese rekursiv weiter.

Kapitel 2.1.2 – KD-Trees

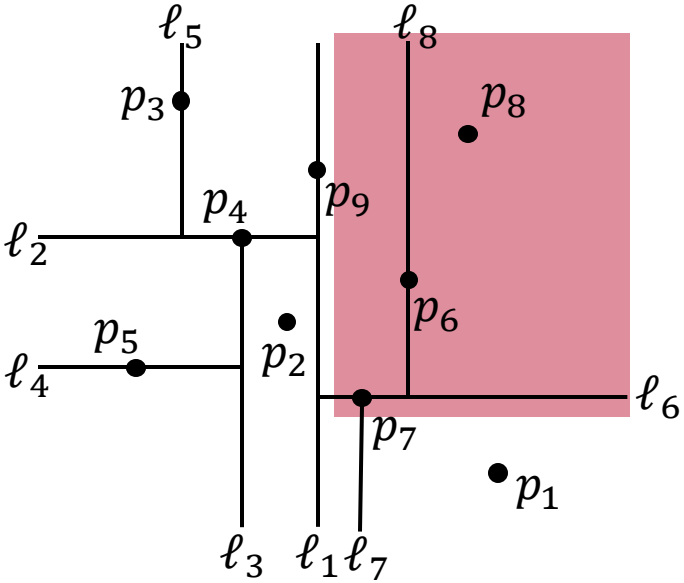


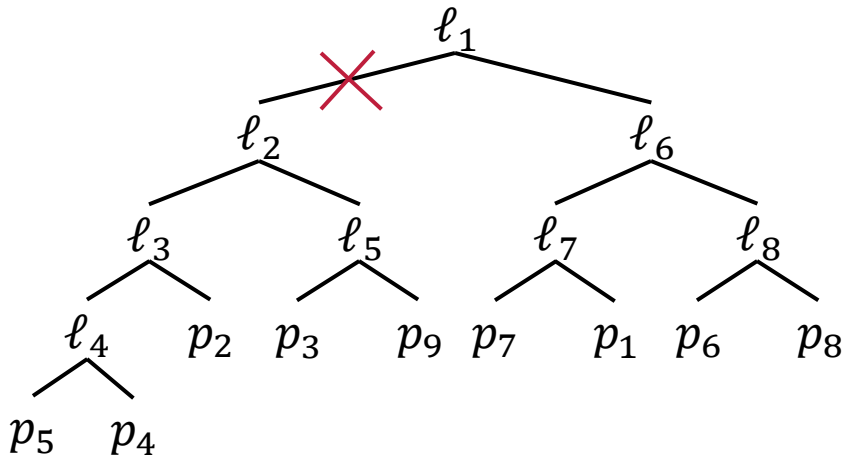
2D-Baum


ℓ_4
ℓ_1
ℓ_7
p_3
p_2
p_4
p_1
ℓ_5
p_8
p_7
ℓ_2
p_5
p_6
ℓ_3
ℓ_6


ℓ_1
ℓ_2
ℓ_5
ℓ_3
ℓ_4
ℓ_6
ℓ_7
p_5
p_6
p_1
p_2
p_7
p_8
p_3
p_4

2D-Bäume – Range Query





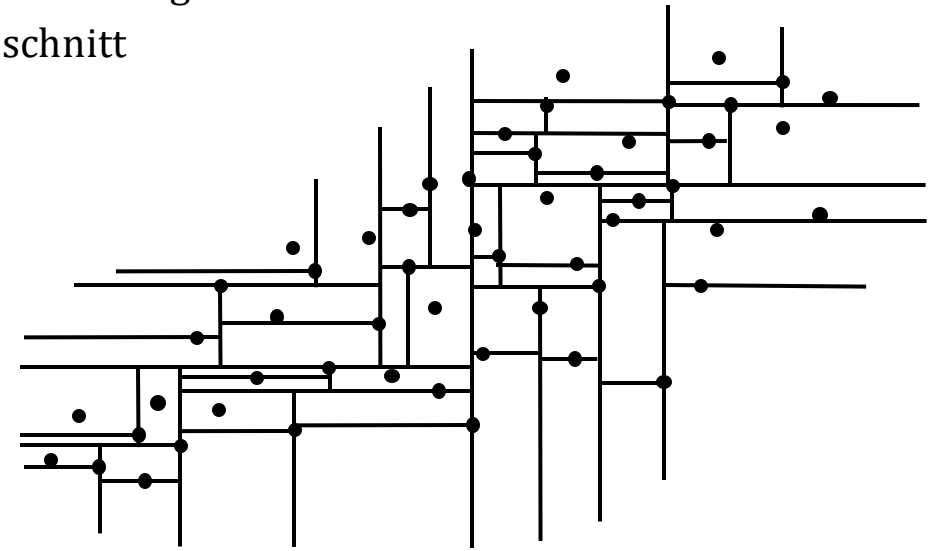
Welche Punkte liegen im Rechteck?

Antwort: p_7 p_6 p_8

kd-Bäume – Höhere Dimensionen

Bisher nur 1D und 2D betrachtet. Laufzeiten für $k \geq 2$ Dimensionen:

- BuildKDTree: $O(n \log n)$
- SearchKDTree: $O(n^{1-\frac{1}{k}} + x)$, um x Elemente auszugeben
- FindNearestNeighbor: $O(\log n)$ im Durchschnitt

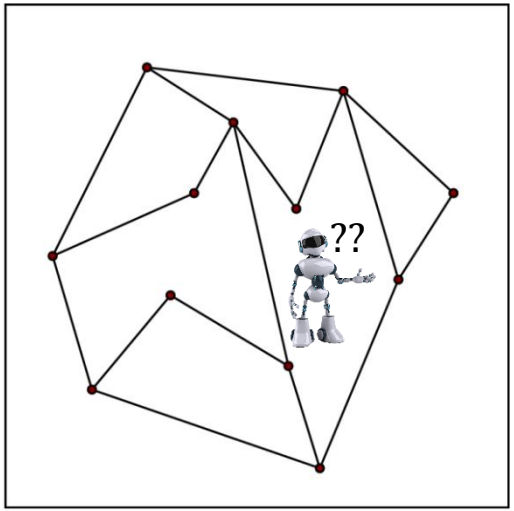


Randbemerkungen

Datenstruktur	Erstellen	Query-Zeit	Speicherbedarf
Quadtree	$O(n \log n)$	$O(n + h)$	$O(n \log N)$
Kd-Tree	$O(n \log n)$	$O\left(n^{1-\frac{1}{d}} + k\right)$	$O(n)$
Range-Trees	$O\left(n \log^{d-1} n\right)$	$O\left(\log^{d} n + k\right)$	$O\left(n \log^{d-1} n\right)$
Range-Trees mit Fractional Cascading	$O\left(n \log^{d-1} n\right)$	$O\left(\log^{d-1} n + k\right)$	$O\left(n \log^{d-1} n\right)$

Erlaubt man ein dynamisches Setting (einfügen und löschen von Punkten), benötigt man für n Einfüge-, Lösch-, und Queryoperationen $\Omega\left(n \log^{d} n\right)$ Zeit.

Kapitel 2.2 – Punktlokalisierung

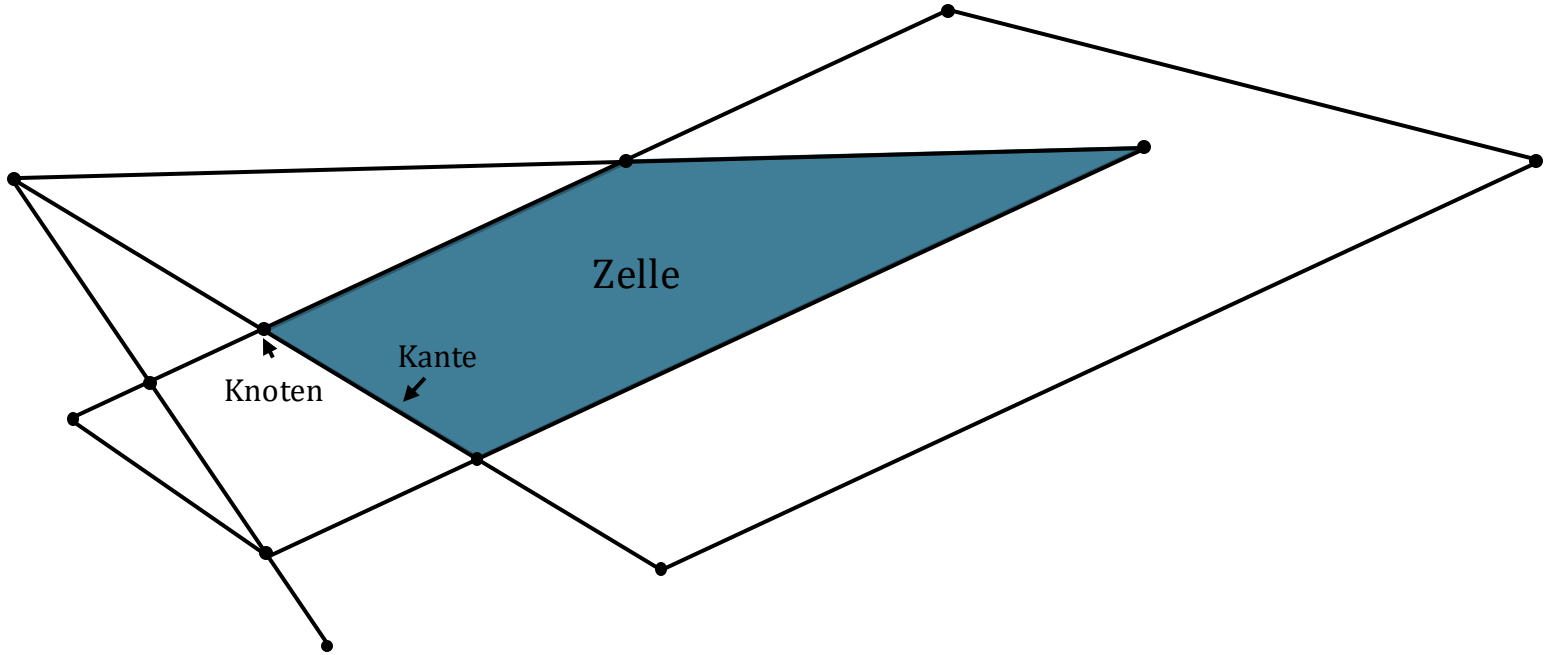


Arrangement – Definition

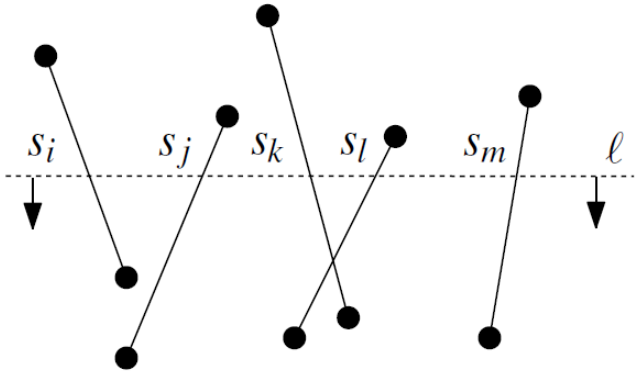
Definition 2.17 (2D-Arrangement)
Für eine Menge von Segmenten S, sei $\mathcal{A}(S)$ das dazugehörige Arrangement. $\mathcal{A}(S)$ besteht dann aus:

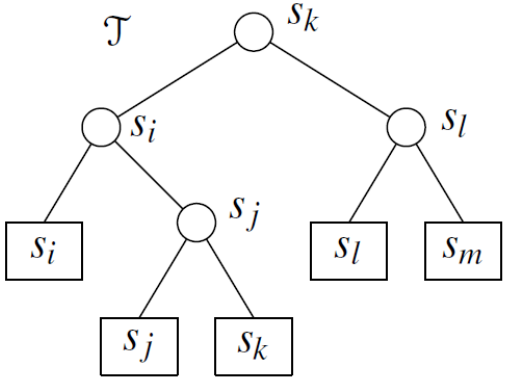
- Zellen (**faces**): Komponenten von $\mathbb{R}^2 \setminus S$. Eine unbeschränkt große Zelle wird auch äußere Zelle (**outer face**) bezeichnet.
- Jede Zelle F wird von (Teil-)Segmenten beschränkt. Diese werden auch **Kanten** genannt.
- Endpunkten der Kanten, auch **Knoten** genannt.

Unterteilung der Ebene



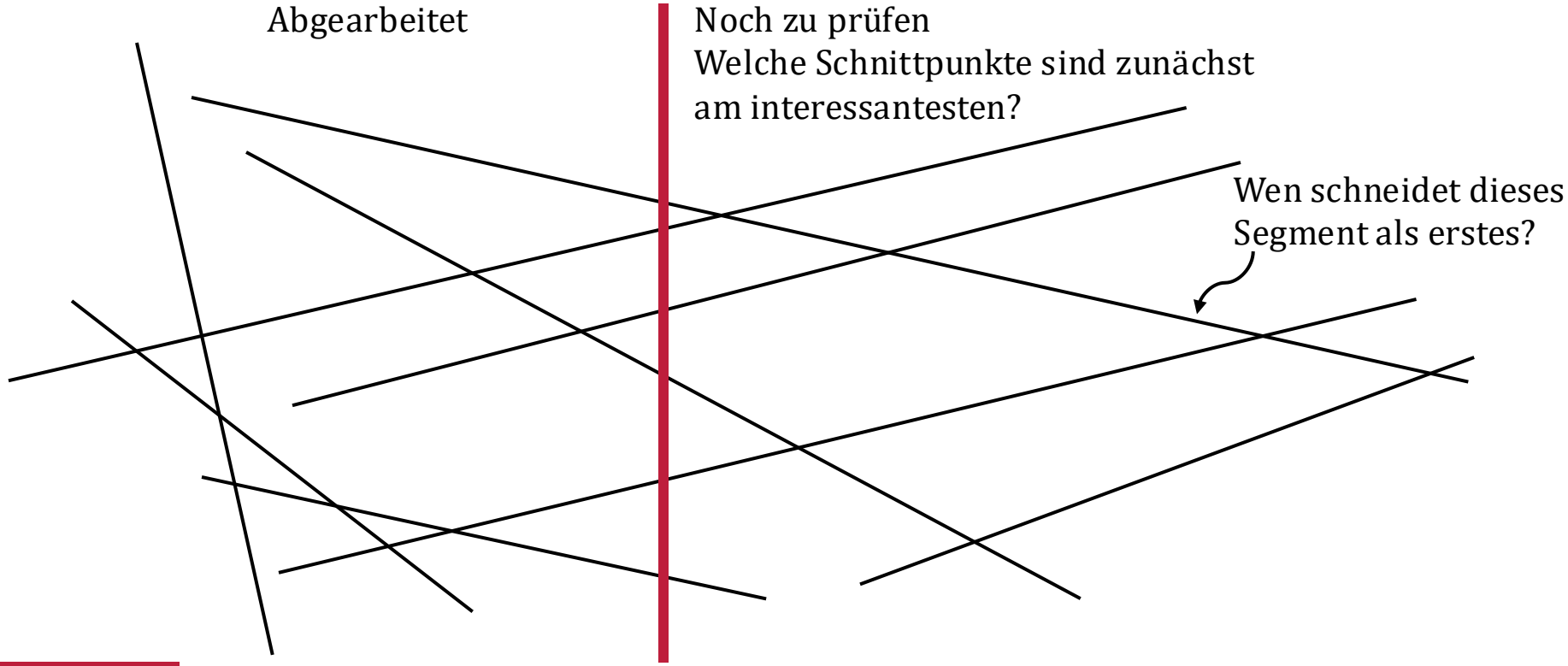
Kapitel 2.2.2 – Sweep-Line-Algorithmen



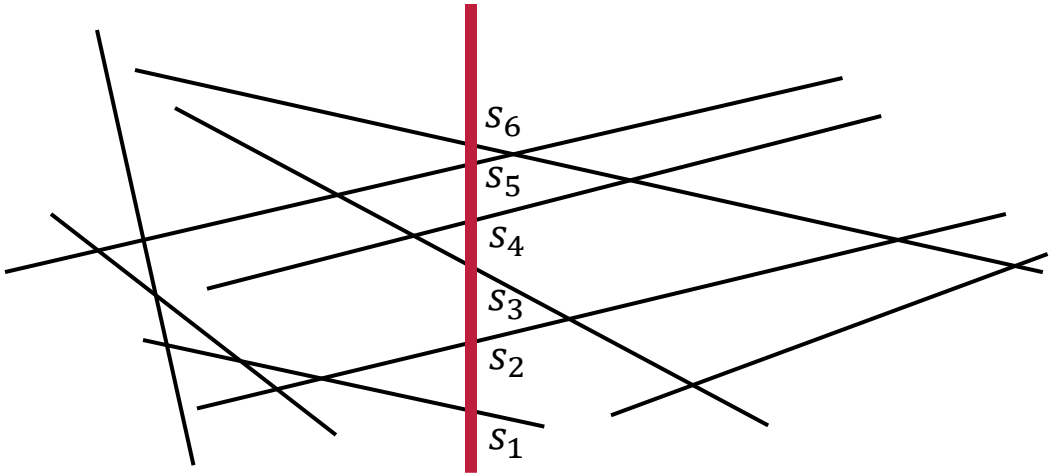


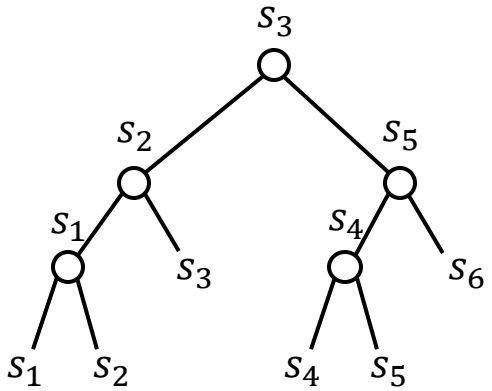
de Berg et al.: Computational Geometry – Algorithms and Applications

Kreuzungen finden: Algorithmus-Idee



Datenstruktur – Balancierte Suchbäume!





Einfügen / Löschen von Segmenten im Suchbaum benötigt $O(\log n)$ Zeit.

Sweep Line – Allgemein

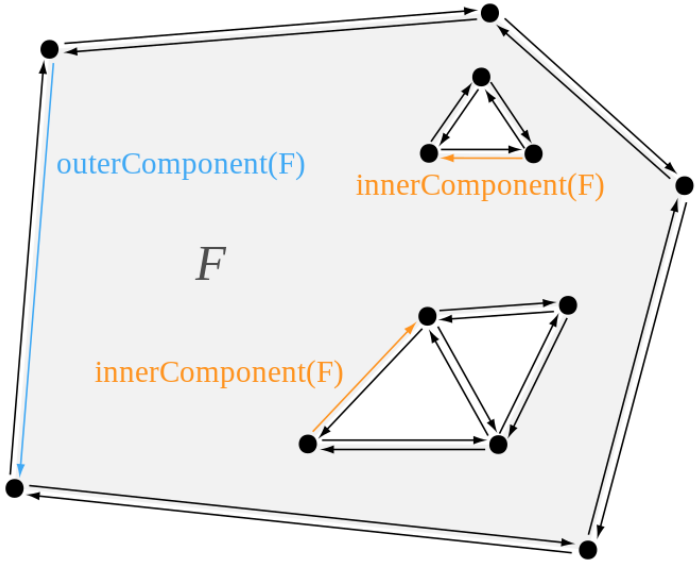
Wie soll **gesweept** werden (entlang einer Achse, um einen Punkt herum, ...)?

Was sind die **Event-Points**? Gibt es verschiedene Event-Typen?

Wie werden Event-Points **behandelt**?

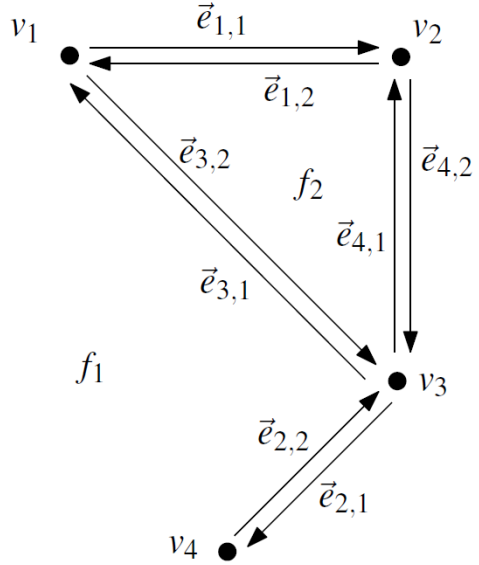
Welche **Datenstrukturen** werden benutzt?

Kapitel 2.2.3 – DCEL



https://de.wikipedia.org/wiki/Half-Edge-Datenstruktur#/media/Datei:Dcel-components.svg

DCEL Beispiel

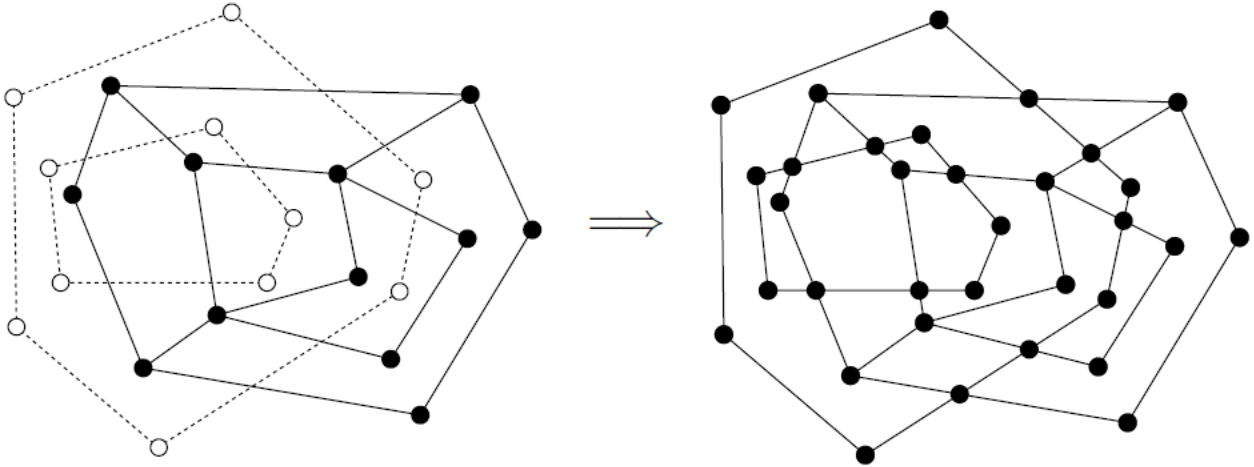


Vertex	Coordinates	IncidentEdge
v_1	$(0,4)$	$\vec{e}_{1,1}$
v_2	$(2,4)$	$\vec{e}_{4,2}$
v_3	$(2,2)$	$\vec{e}_{2,1}$
v_4	$(1,1)$	$\vec{e}_{2,2}$

Face	OuterComponent	InnerComponents
f_1	**nil**	$\vec{e}_{1,1}$
f_2	$\vec{e}_{4,1}$	**nil**

Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1	$\vec{e}_{3,1}$	$\vec{e}_{2,1}$
$\vec{e}_{3,1}$	v_3	$\vec{e}_{3,2}$	f_1	$\vec{e}_{1,1}$	$\vec{e}_{2,2}$
$\vec{e}_{3,2}$	v_1	$\vec{e}_{3,1}$	f_2	$\vec{e}_{4,1}$	$\vec{e}_{1,2}$
$\vec{e}_{4,1}$	v_3	$\vec{e}_{4,2}$	f_2	$\vec{e}_{1,2}$	$\vec{e}_{3,2}$
$\vec{e}_{4,2}$	v_2	$\vec{e}_{4,1}$	f_1	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

Kapitel 2.2.4 – Overlays



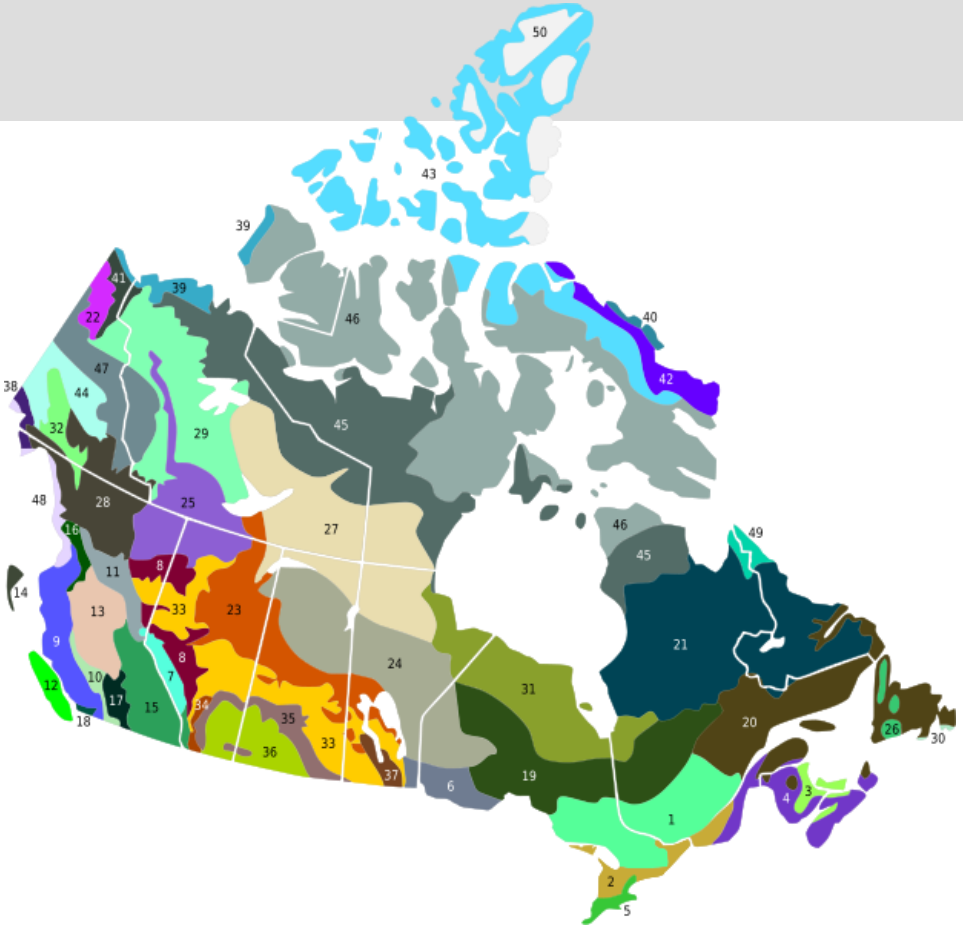
De Berg et al.: Computational Geometry – Algorithms and Applications

Overlay

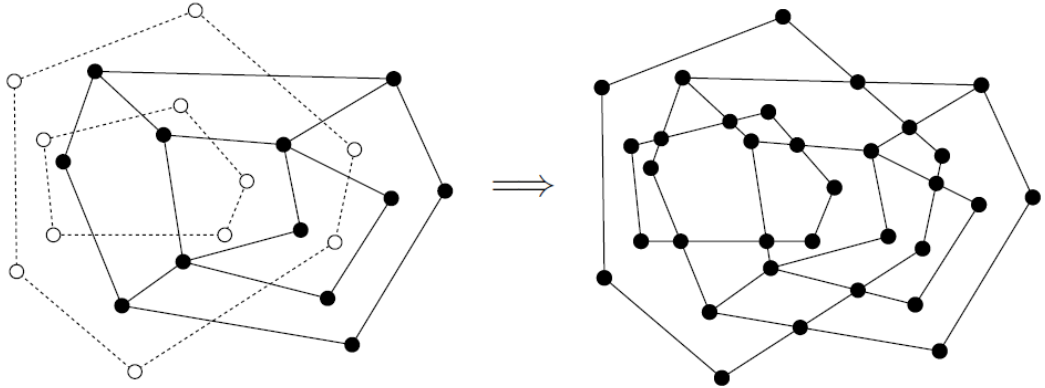
Im Fall von Kanada möchte man nicht nur wissen, in welcher Ökoregion man ist, sondern auch in welcher Provinz.

Man kann für beide Arrangements separat eine Lokalisierungsanfrage durchführen.

Aber was passiert, wenn wir k viele Arrangements besitzen?
→ Können wir alle Arrangements zusammenlegen?



Ideen für einen Algorithmus



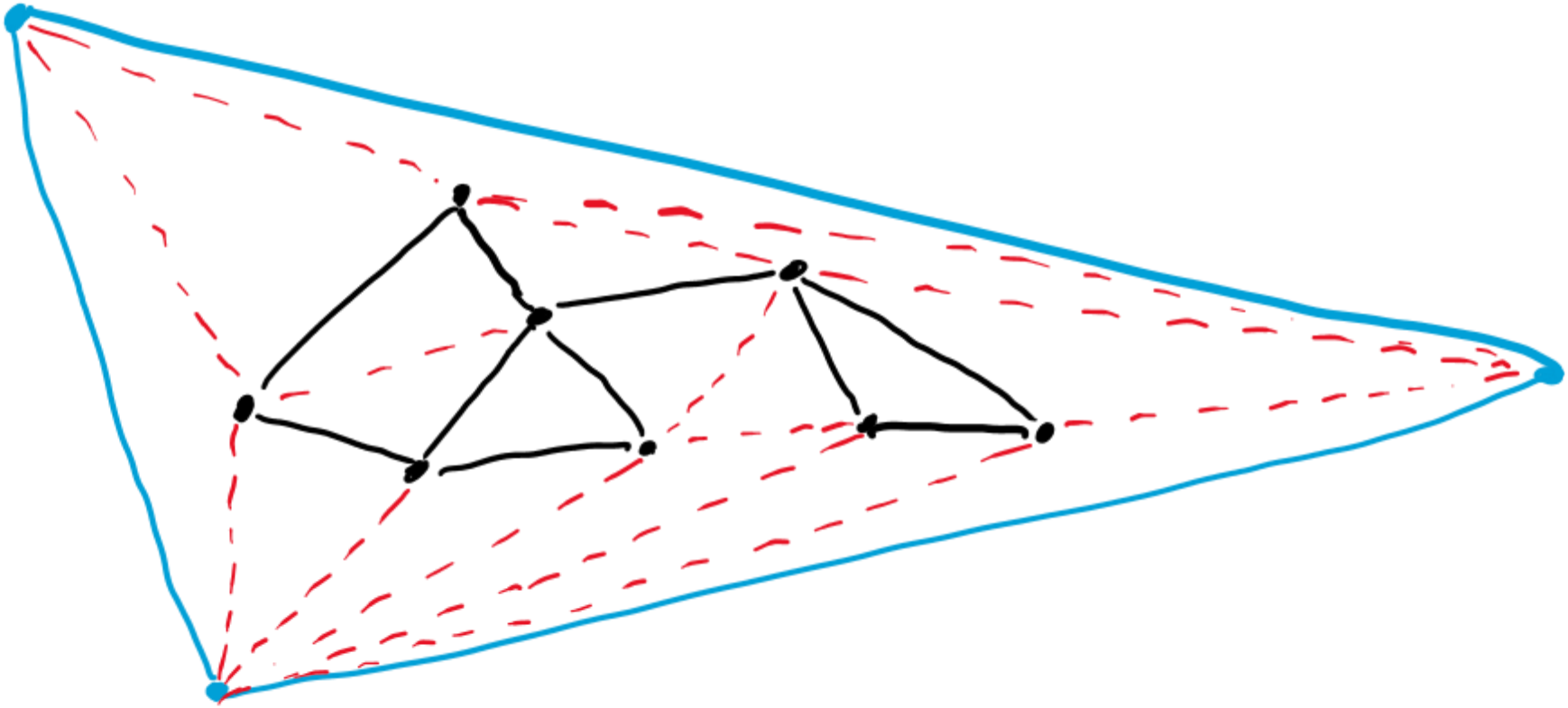
Das Ergebnis sieht ähnlich zu dem Intersection-Problem aus. Können wir den Sweep-Line-Algorithmus wiederverwenden?

Wichtig:

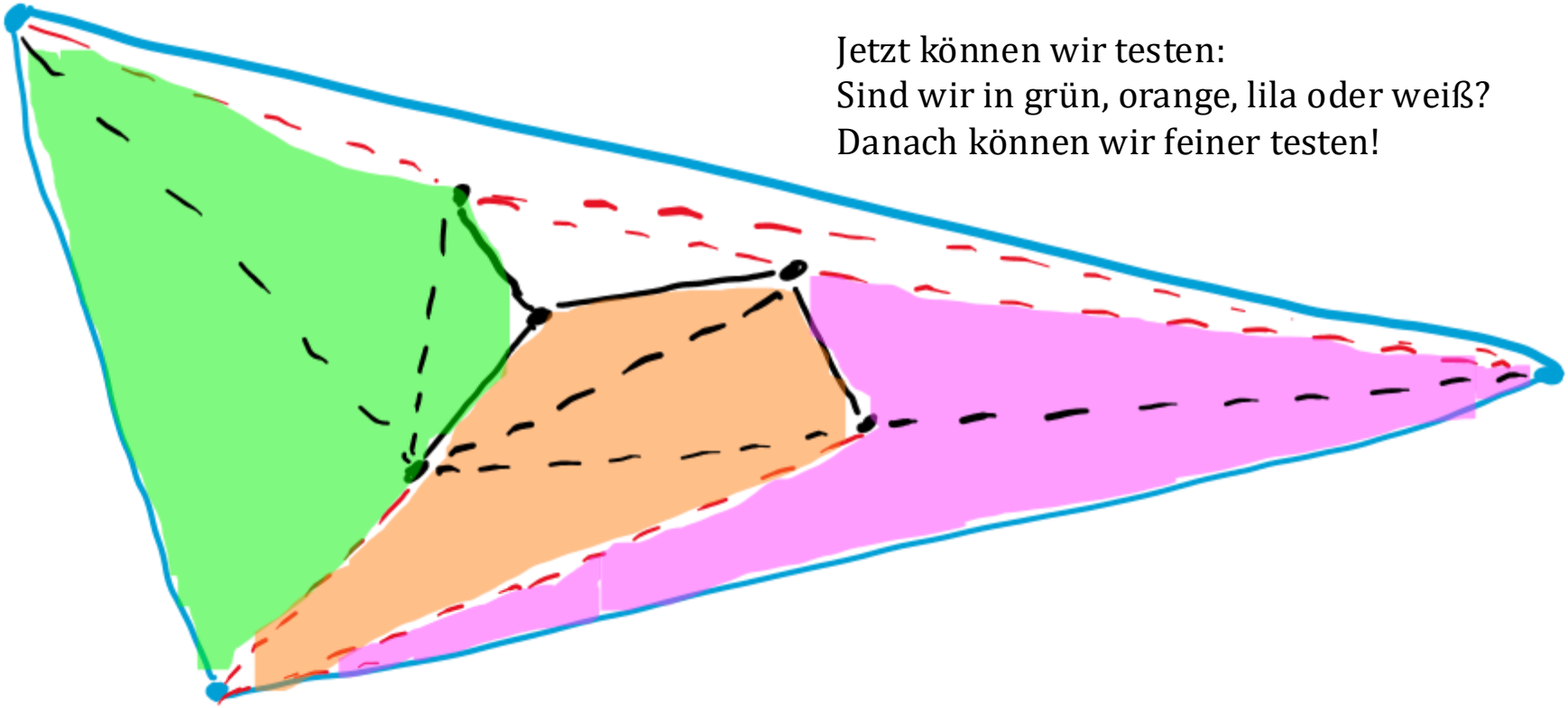
- Jeder vorhandene Knoten bleibt bestehen.
- Half-Edges spalten sich auf und erzeugen neue Knoten und Half-Edges. Nur bei einer Intersection!
- Bestehende Faces werden weiter unterteilt. Wie erkennt man das?

Kapitel 2.2.5 – Arrangement Triangulationen

Triangulations-Beispiel

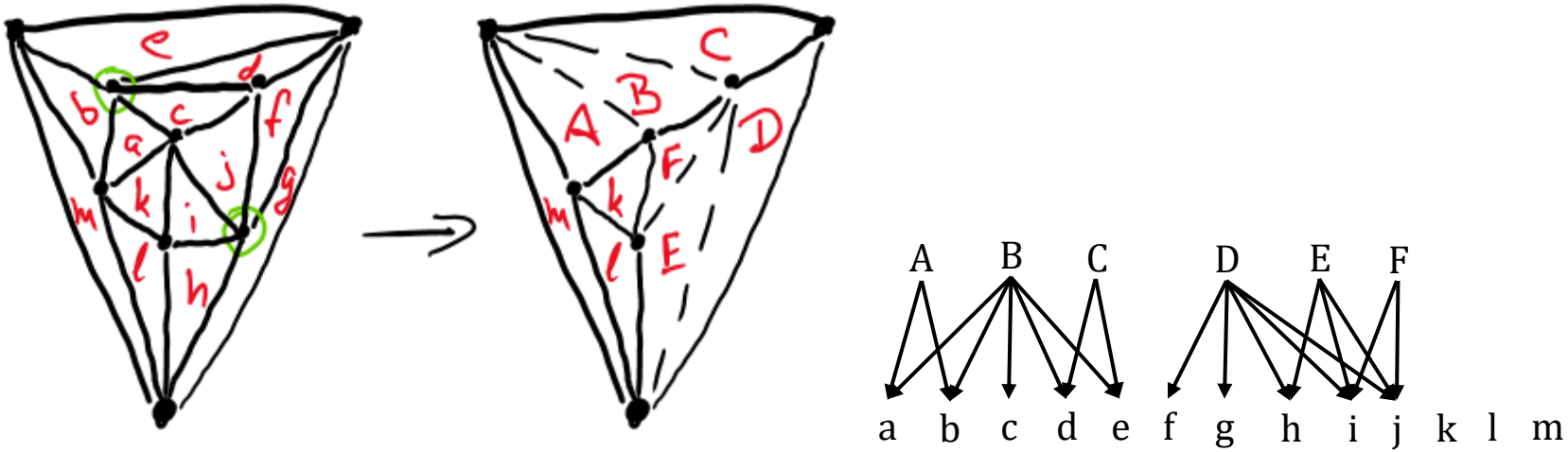


Simplifizieren

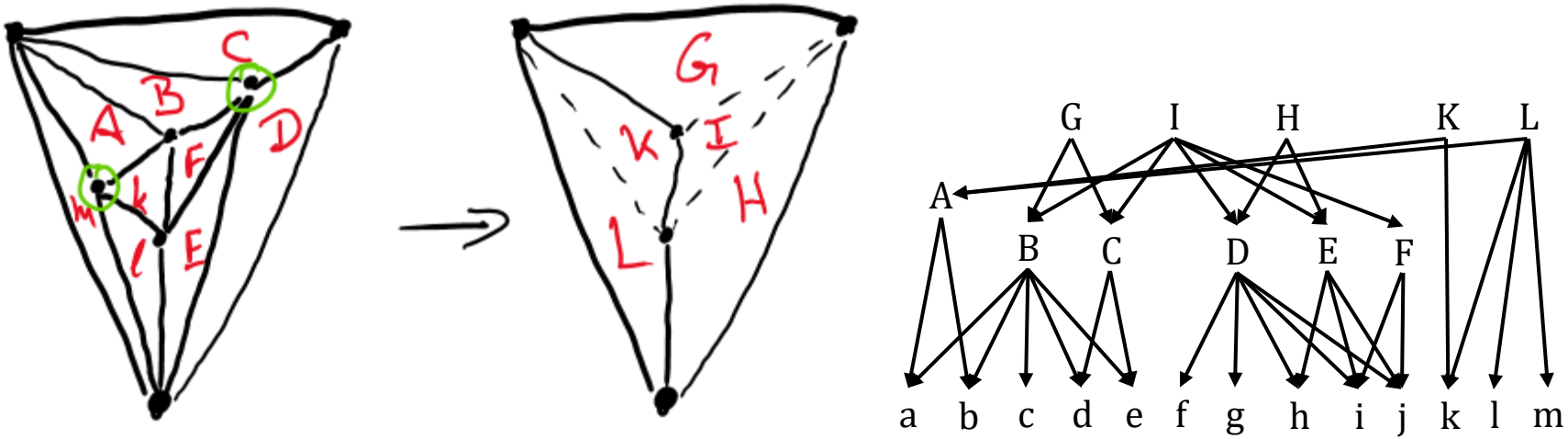


Jetzt können wir testen:
Sind wir in grün, orange, lila oder weiß?
Danach können wir feiner testen!

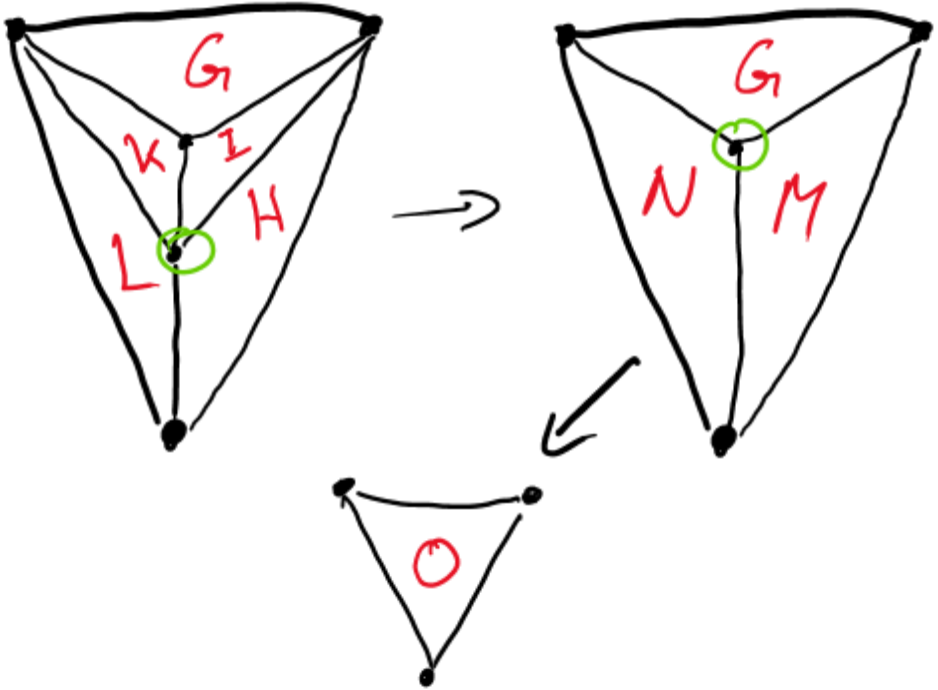
Hierarchie von Simplifizierungen

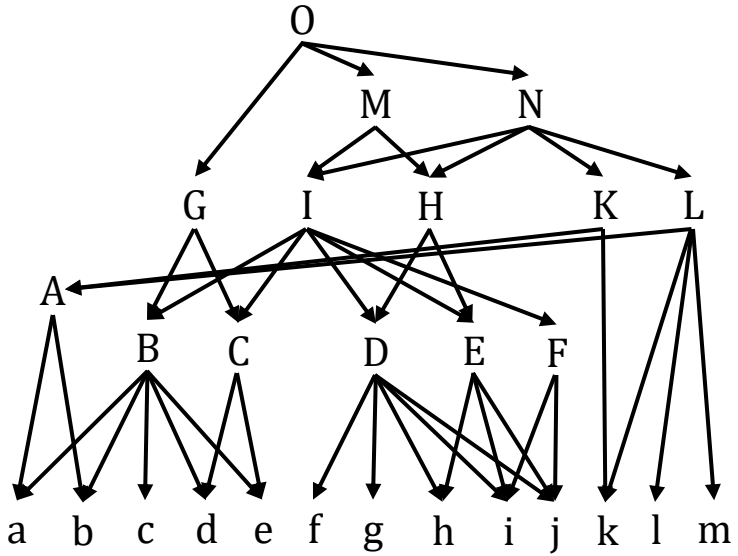


Hierarchie von Simplifizierungen



Hierarchie von Simplifizierungen





Datenstruktur der Hierarchie

Definition 2.32 (Kirkpatricks Hierarchie)
Sei $\mathcal{A}$ ein Arrangement, $T(\mathcal{A})$ dessen Triangulation und $T_0, T_1, \dots, T_k$ eine Folge von Triangulationen mit folgenden Eigenschaften.

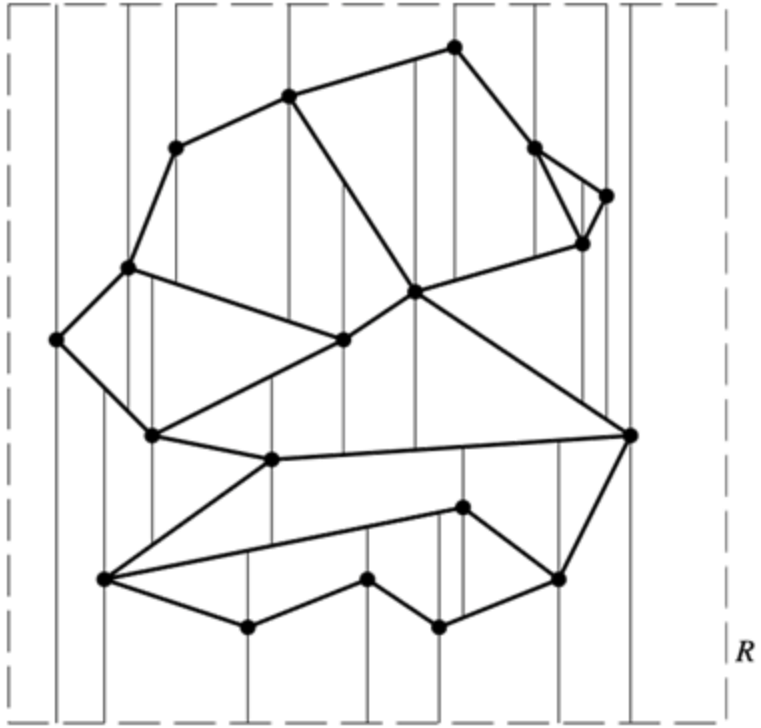
- $T_0 \coloneqq T(\mathcal{A})$
- T_{i+1} entsteht aus T_i durch Löschen unabhängiger Knoten und Triangulieren freier Flächen.
- T_k ist ein Dreieck

Gespeichert wird die Hierarchie mit einem gerichteten, azyklischen Graphen (kurz **DAG**):

- Jedes Dreieck aus T_i entspricht einem Knoten.
- Es existiert eine Kante von einem Dreieck $\Delta_1 \in T_{i+1}$ zu einem Dreieck $\Delta_2 \in T_i$, wenn $\Delta_1 \cap \Delta_2 \neq \emptyset$

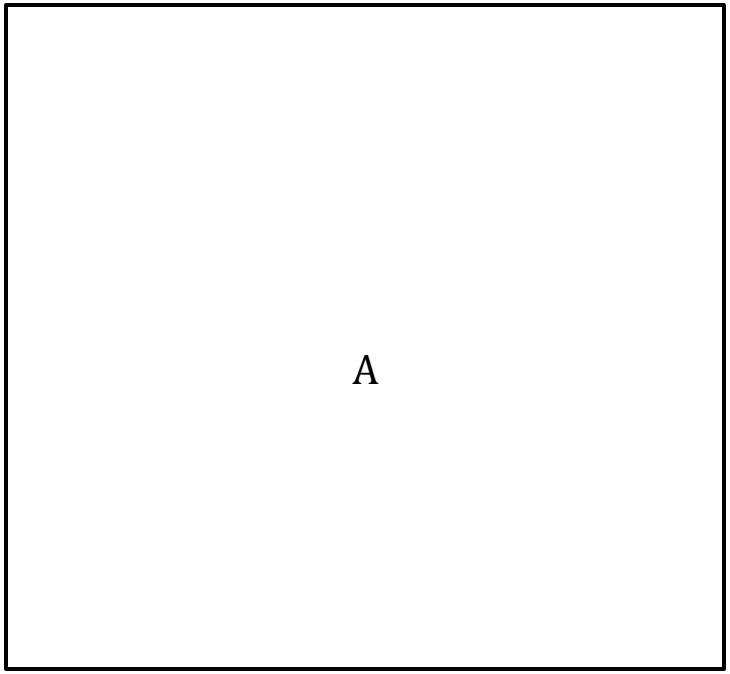
Kapitel 2.2.6 – Trapezoidal Maps

Beispiel



Trapezoidal Map

Insertion-Methode

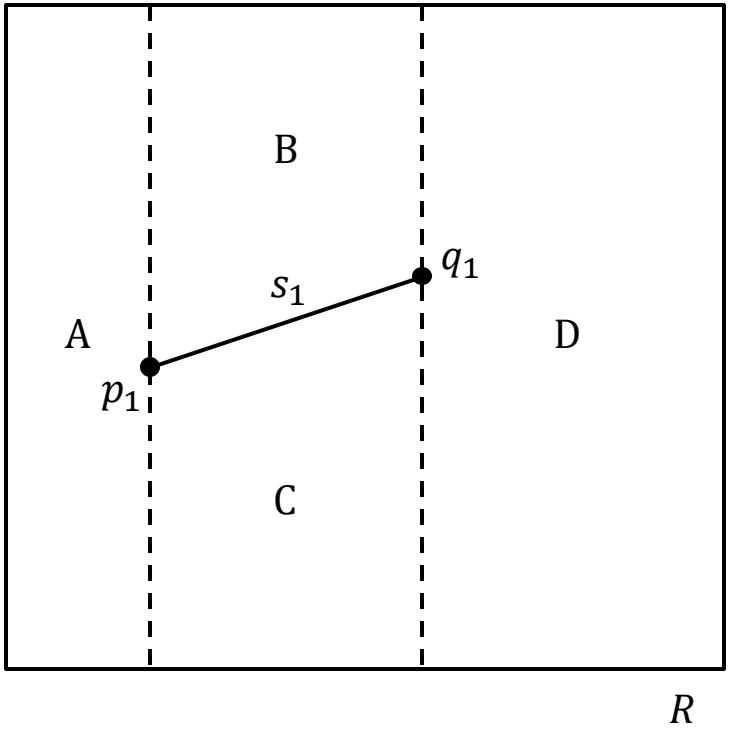


Suchstruktur:

A

Wenn wir ein Segment hinzufügen, was passiert?

Insertion-Methode



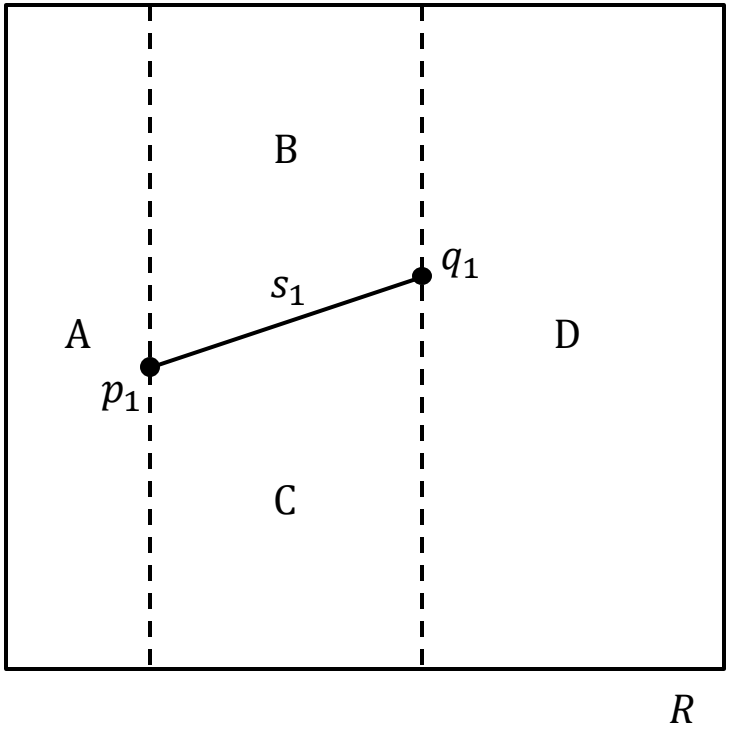
Suchstruktur:

A

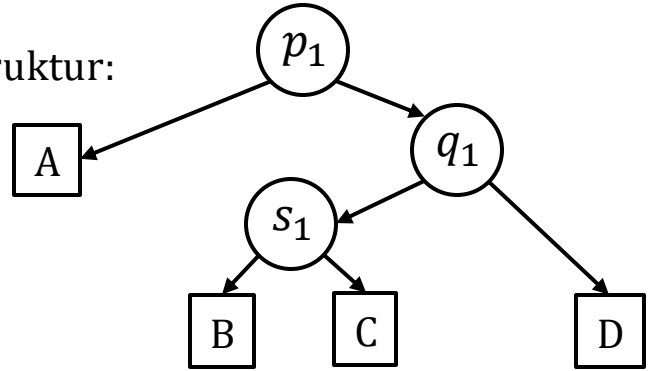
Wenn wir ein Segment hinzufügen, was passiert?
Es entstehen Flächen:

- Links von p_1
- Rechts von q_1
- Zwischen p_1 und q_1, aber unter s_1
- Zwischen p_1 und q_1, aber über s_1

Insertion-Methode



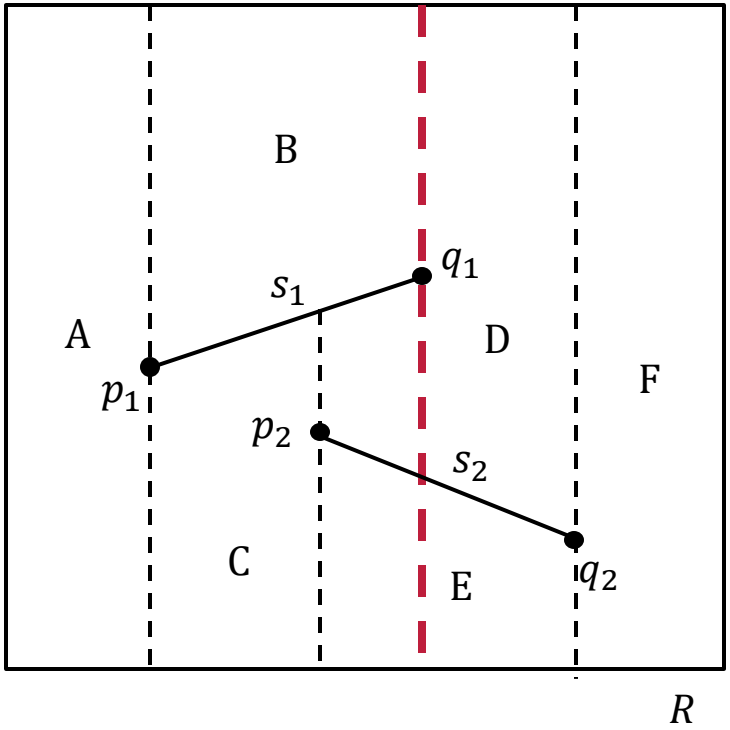
Suchstruktur:



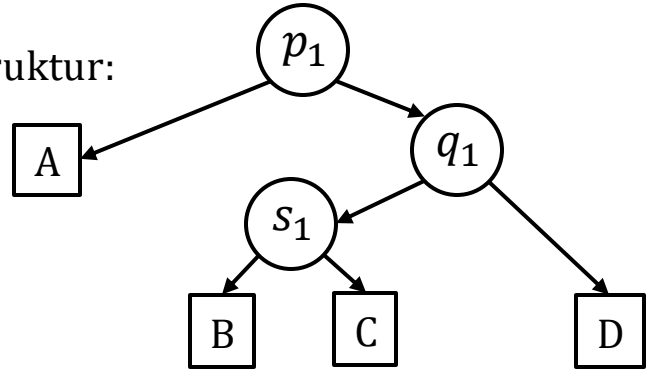
Wenn wir ein Segment hinzufügen, was passiert?
Es entstehen Flächen:

- Links von p_1
- Rechts von q_1
- Zwischen p_1 und q_1, aber unter s_1
- Zwischen p_1 und q_1, aber über s_1

Insertion-Methode

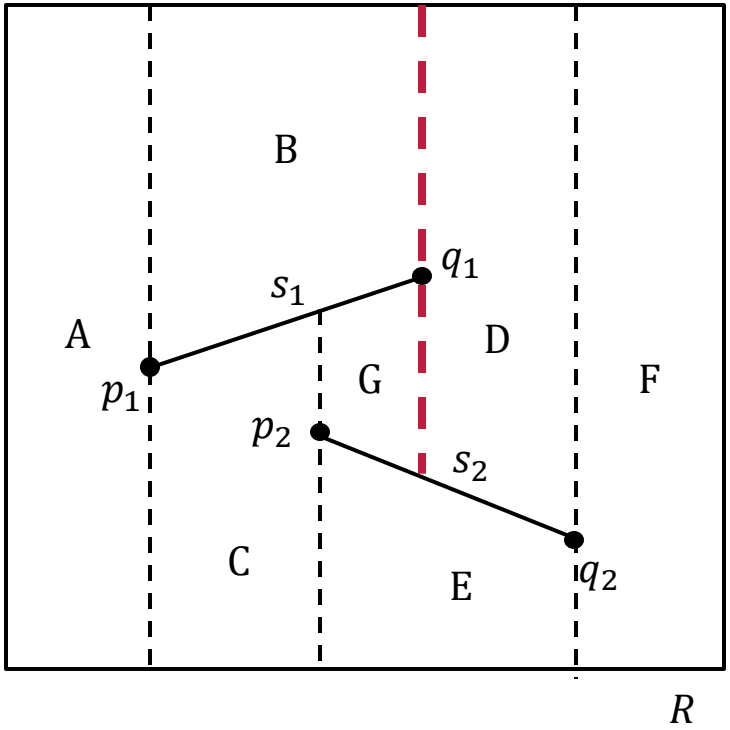


Suchstruktur:

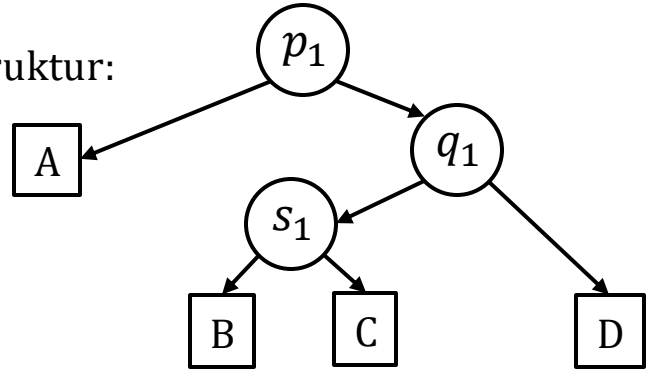


Und wenn wir ein weiteres Segment hinzufügen?
Vertikale Erweiterungen können geschnitten werden!

Insertion-Methode

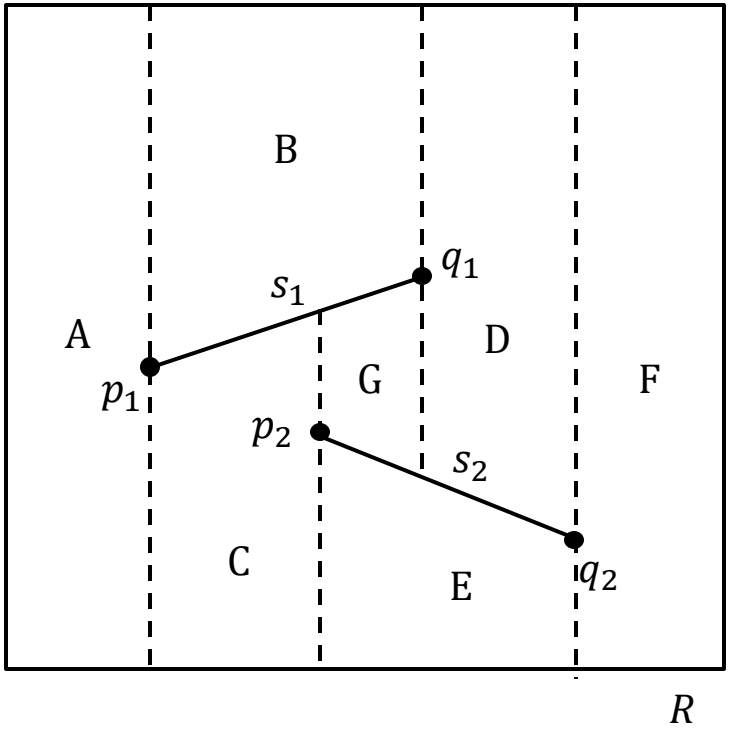


Suchstruktur:

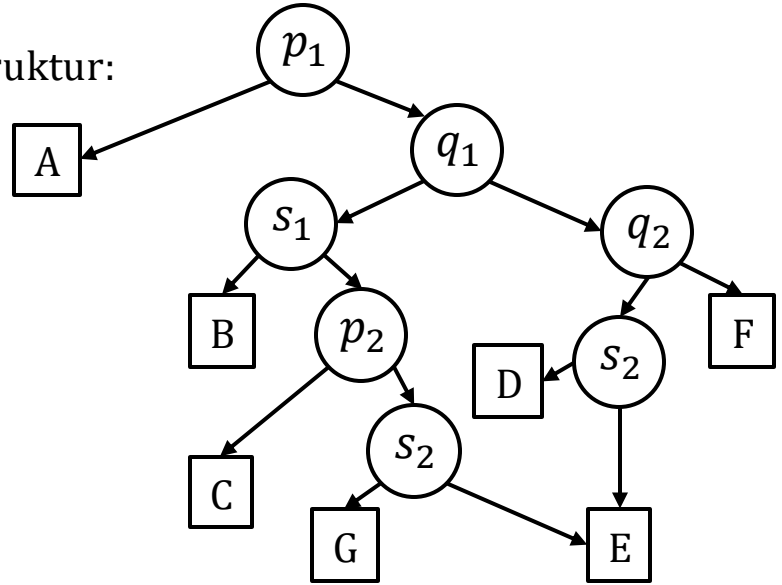


Und wenn wir ein weiteres Segment hinzufügen?
Vertikale Erweiterungen können geschnitten werden!

Insertion-Methode

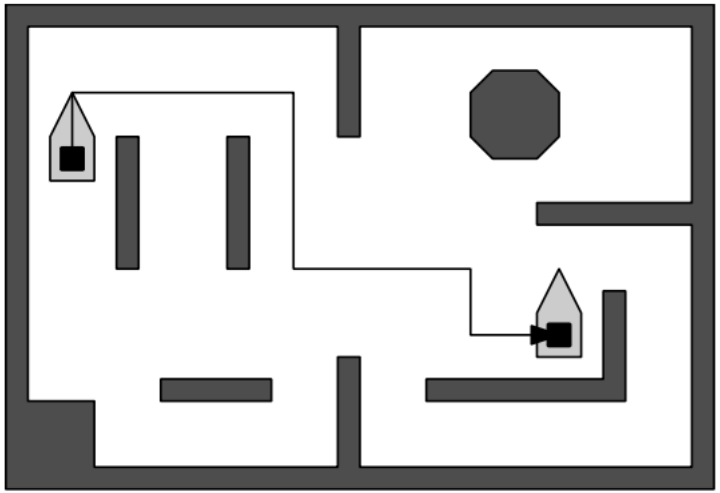


Suchstruktur:



Und wenn wir ein weiteres Segment hinzufügen?
Vertikale Erweiterungen können geschnitten werden!

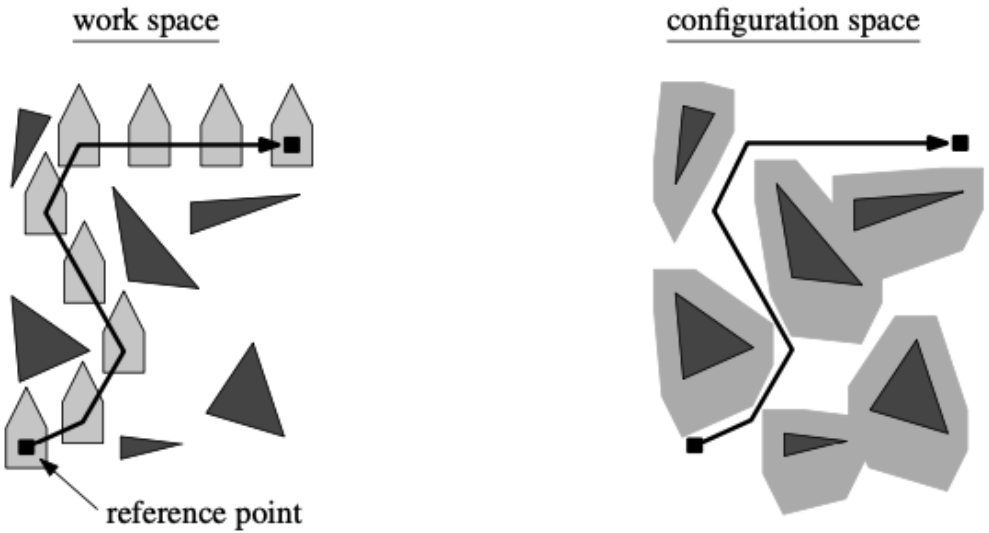
Kapitel 3 – Robot Motion Planning



Kapitel 3.1 – Work- und Configurationspace

Bewegungen

Lemma 3.3
Ein Pfad im Workspace entspricht einer Kurve im Konfigurationsraum.
Ein kollisionsfreier Pfad entspricht einer Kurve im freien Konfigurationsraum.



Kapitel 3.2 – Roadmap

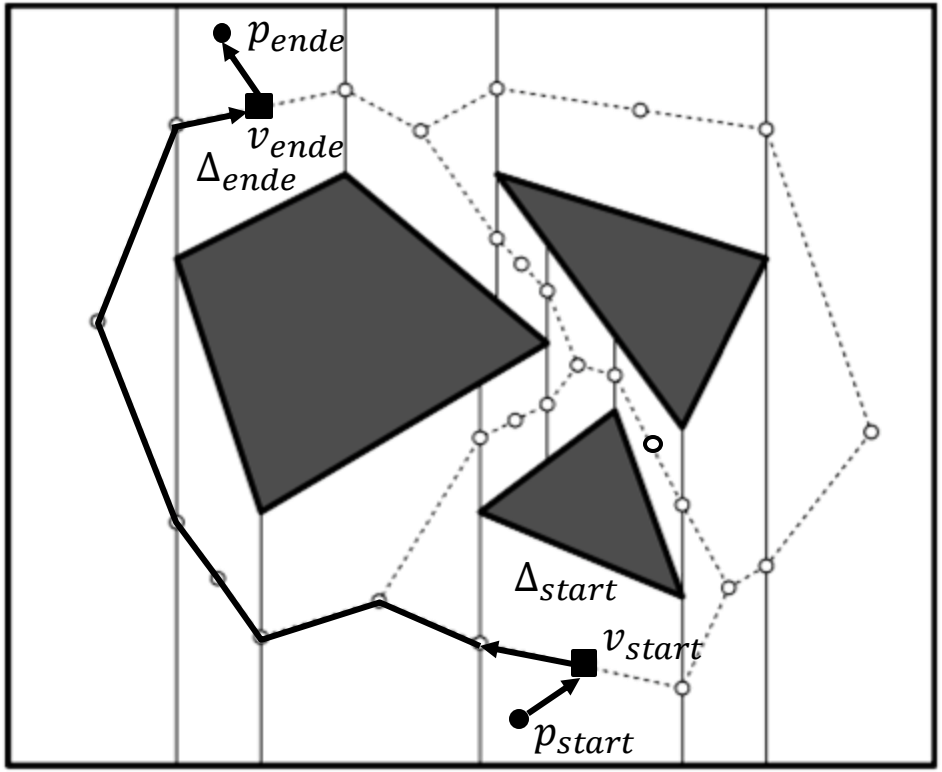
Beispiel

Schritt 1:
Bestimme Δ_{start} und Δ_{ende}, sowie v_{start} und v_{ende} der Road Map

Schritt 2:
Bestimme Pfad von v_{start} nach v_{ende} über die Road Map.

Das funktioniert bspw. mit Breitensuche

Schritt 3:
Verbinde p_{start} mit v_{start} sowie v_{ende} mit p_{ende}



Kapitel 3.3 – Minkowski Summen

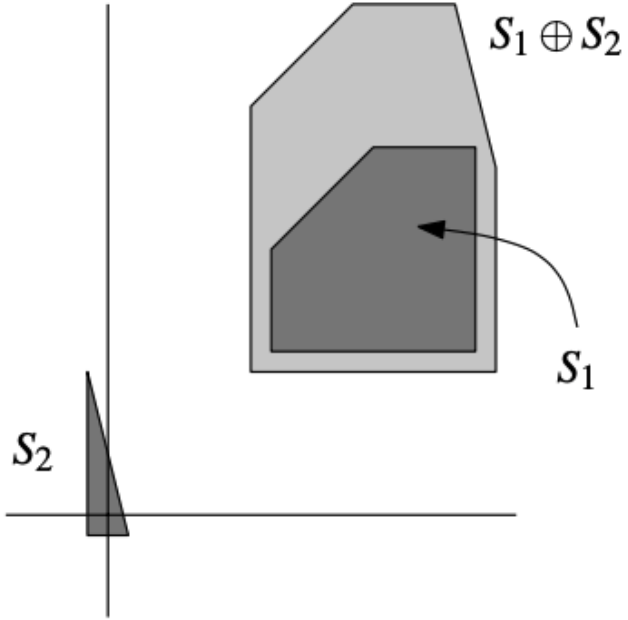
Minkowski Summe

Definition 3.10
Die **Minkowski Summe** von zwei mengen $S_1, S_2 \subset \mathbb{R}^2$ ist definiert als

$$S_1 \oplus S_2 := \{ p + q \mid p \in S_1, q \in S_2 \},$$

wobei $p + q$ eine Vektoraddition ist.

Die Hindernisse und den Roboter können wir als solche planaren Mengen beschreiben!

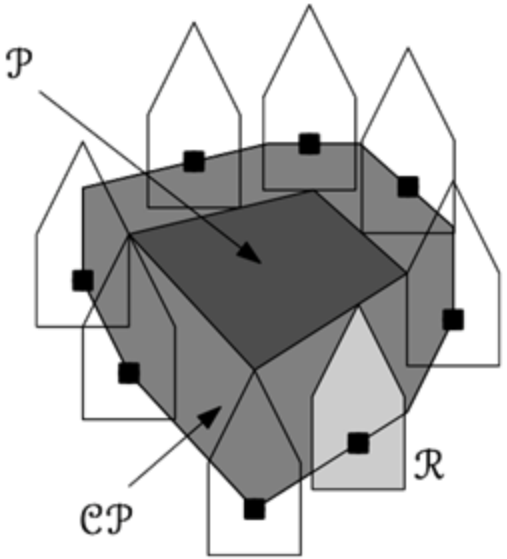


Minkowski Summe

Wie berechnen wir den Configuration Space?

Polygon $\oplus$ negativer Roboter!

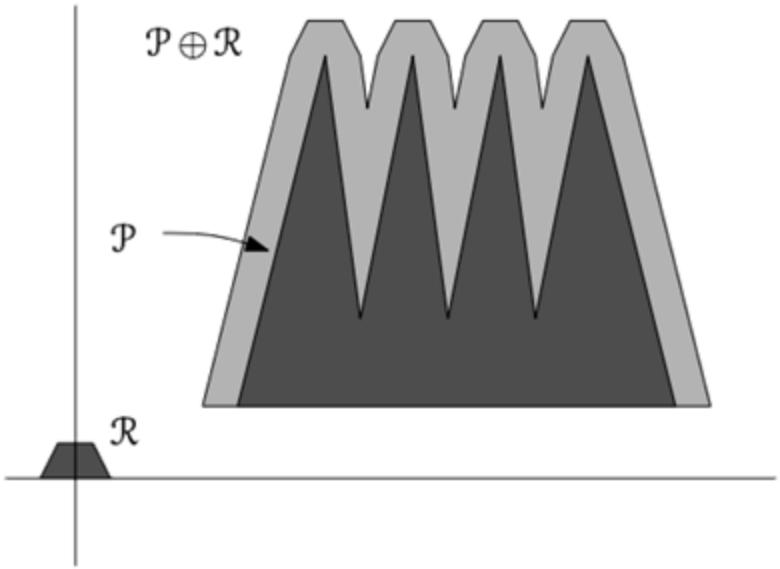
Wir müssen am Ende nur den Rand der Summe bestimmen, auf dem sich der Referenzpunkt gerade noch bewegen darf.

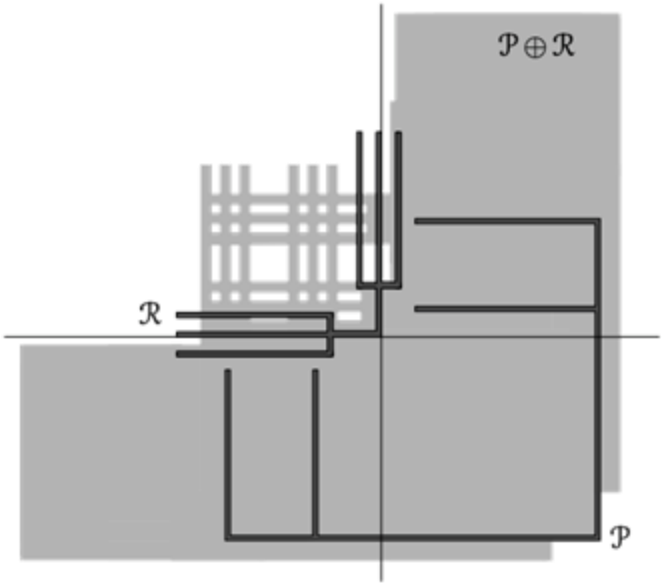


Beobachtungen, wenn $\mathcal{R}$ und P konvex sind:

- Entweder laufen wir entlang einer Kante von $\mathcal{R}$ oder P.
- Das machen wir nur einmal!
- Wir können uns auf die Ränder von $\mathcal{R}$ und P konzentrieren.

Worst-Cases


$\mathcal{P} \oplus \mathcal{R}$
$\mathcal{P}$
$\mathcal{R}$


$\mathcal{P} \oplus \mathcal{R}$
$\mathcal{R}$
$\mathcal{P}$

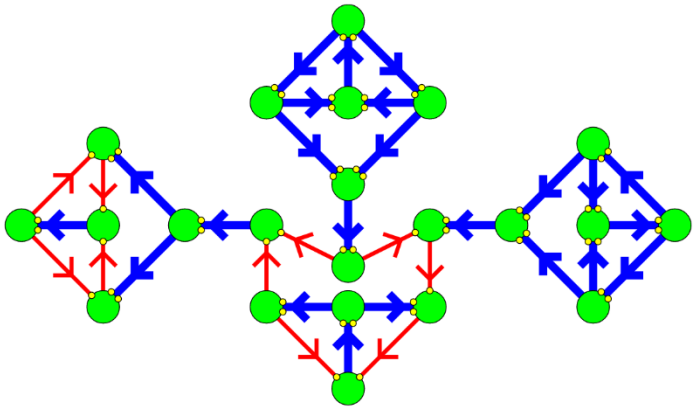
Kapitel 3.5 – Multi-Robot-Motionplanning

PSPACE

Theorem:
Das genannte Problem ist PSPACE-vollständig

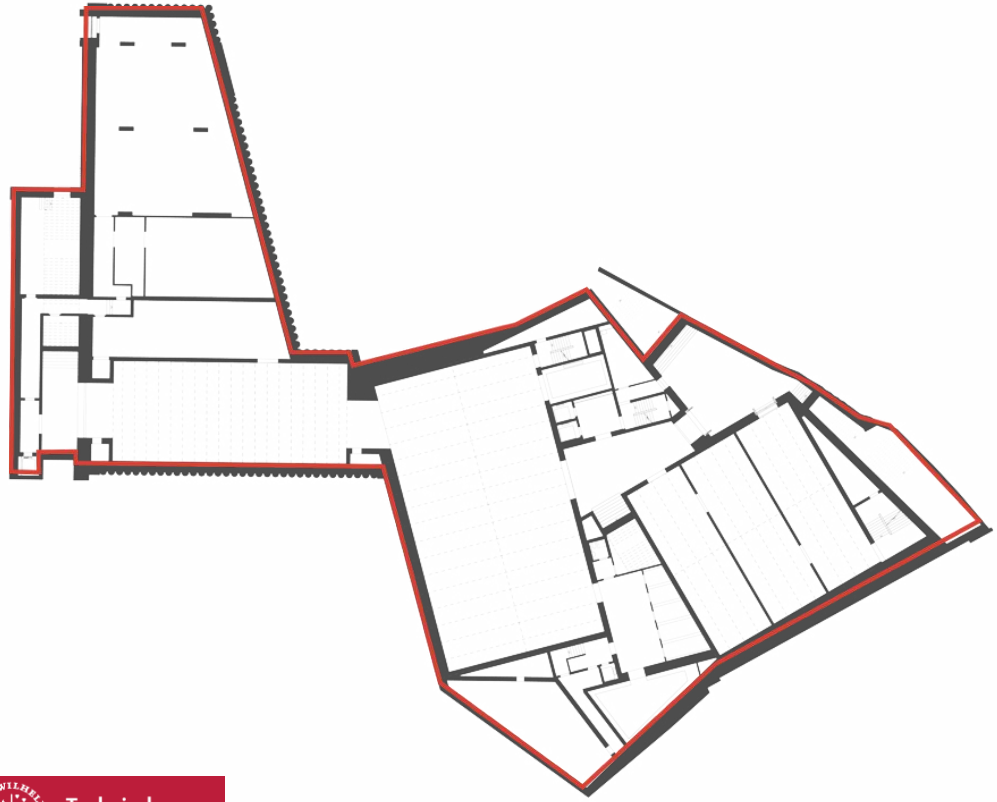
On the hardness of unlabeled multi-robot motion planning*

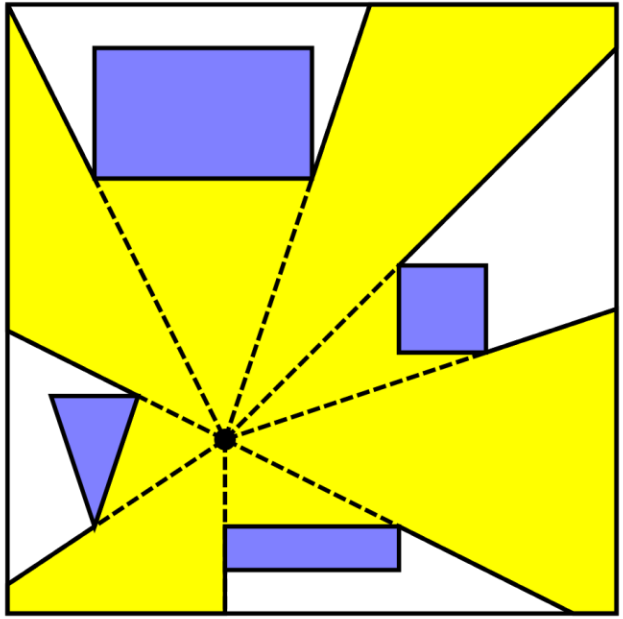
Kiril Solovey and Dan Halperin
Blavatnic School of Computer Science
Tel Aviv University, Israel
email: {kirilsol,danha}@post.tau.ac.il



Kapitel 4 – Art Gallery

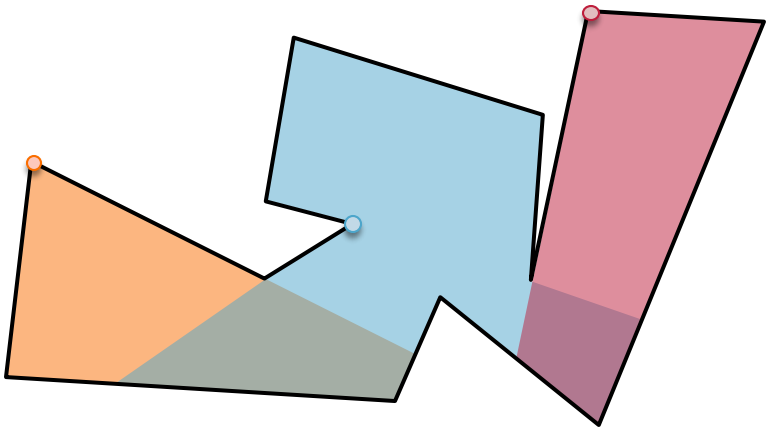
Art Gallery und Guards



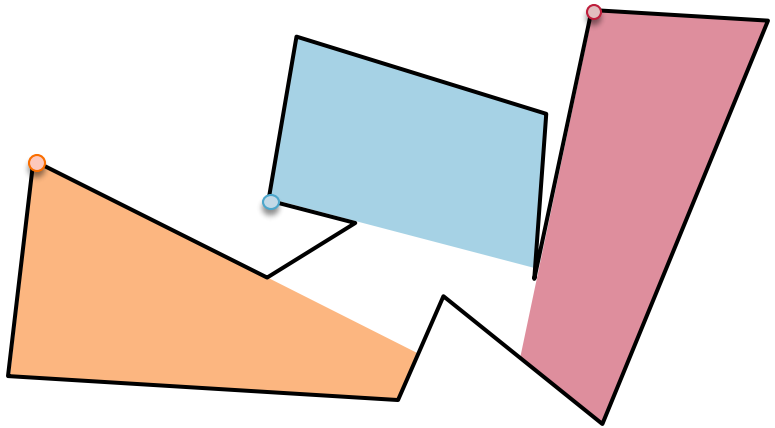


Guard- und Witness-Sets

Guard-Set:
Vereinigung muss das Polygon ergeben.



Disjunktes Witness-Set:
Visibility muss disjunkt sein.

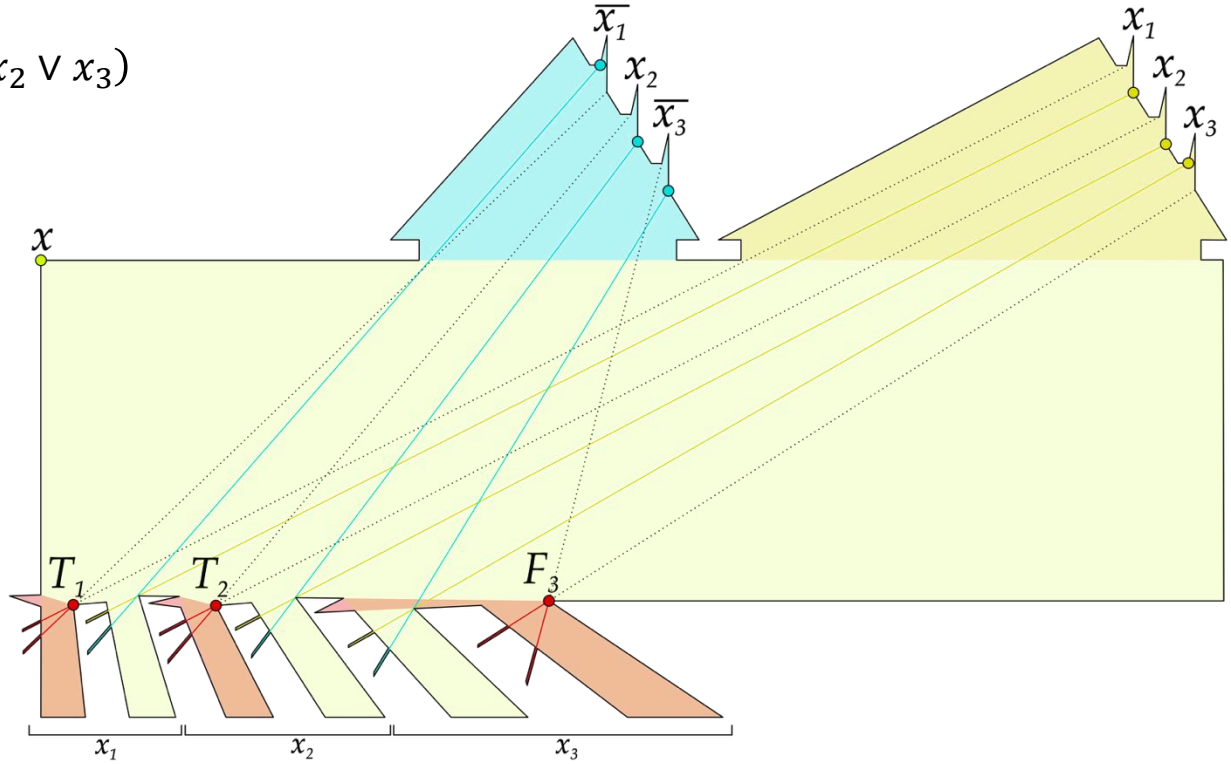


3SAT auf AGP – Eine erfüllende Belegung

Beispiel:

$F = (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3)$

$x_1 = true$
$x_2 = true$
$x_3 = false$



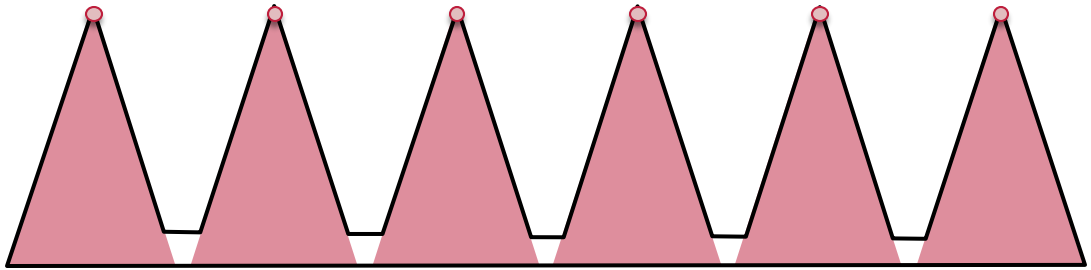
Theorem 4.5
Das AGP mit Vertex-Guards ist NP-vollständig.

Kapitel 4.1 – Schranken

Manchmal notwendig

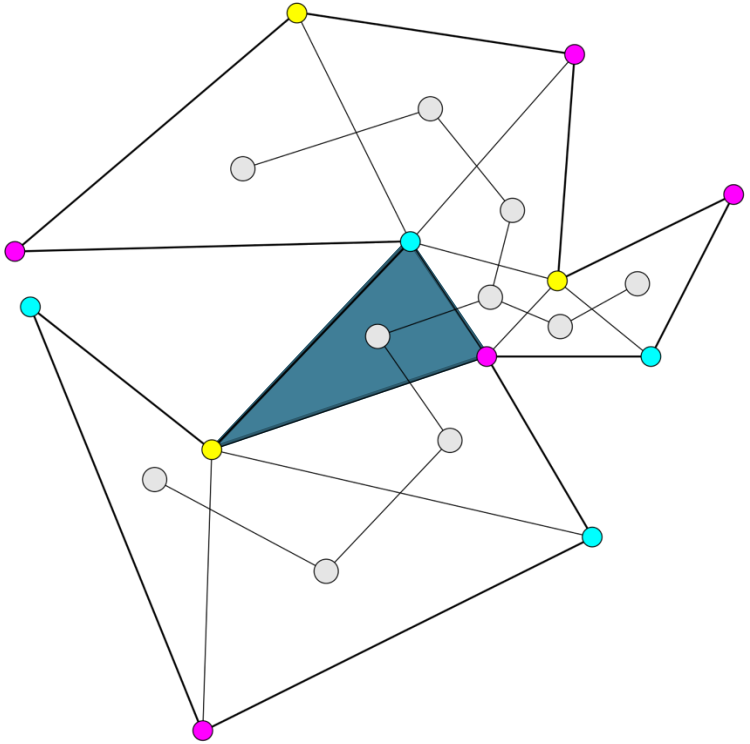
Lemma 4.6

Für ein einfaches Polygon mit n Knoten sind manchmal $\left\lfloor \frac{n}{3} \right\rfloor$ (Vertex-)Guards notwendig.



Das Polygon hat $3m$ Knoten und ein Witness-Set der Größe m.

Fisks Beweis



Wähle ein Dreieck und färbe die Knoten beliebg.
Fahre rekursiv auf den Nachbarn fort:

- Zwei Knoten haben bereits eine Farbe.
- Gib dem dritten die fehlende Farbe.
- Fahre mit unbearbeiteten Nachbarn fort.

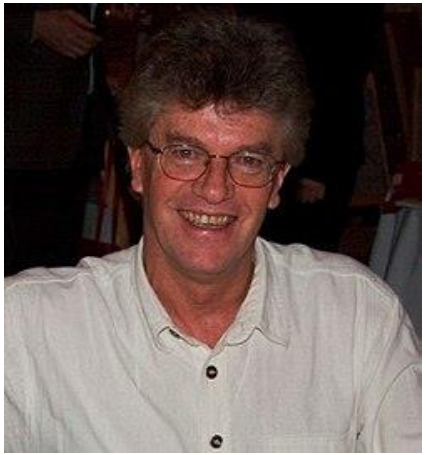
Achtung: Das funktioniert nur, weil P einfach ist!

Lemma 4.7

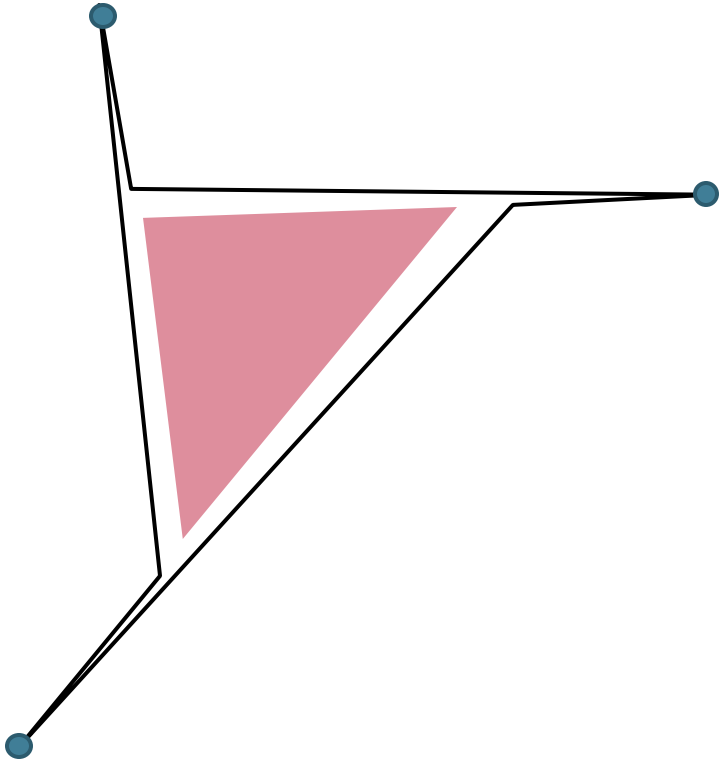
Für ein einfaches Polygon mit n Knoten sind $\left\lfloor \frac{n}{3} \right\rfloor$ Vertex-Guards immer ausreichend.

Godfried‘s Favorite Polygon

Die Guards auf den Spitzen decke jede Kante ab, aber nicht das gesamte Polygon!



Godfried Toussaint
(1944 – 2019)



Kapitel 4.3 – Sichtbarkeitspolygone

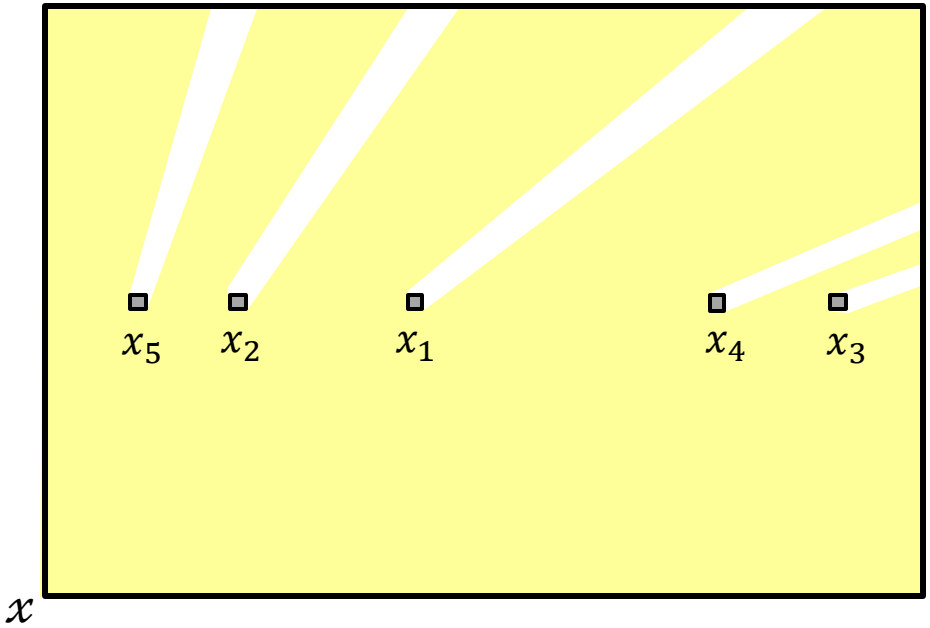
Reduktion: Sortieren auf Visibility

Sei $\{x_1, \dots, x_n\} \subset \mathbb{N}$.

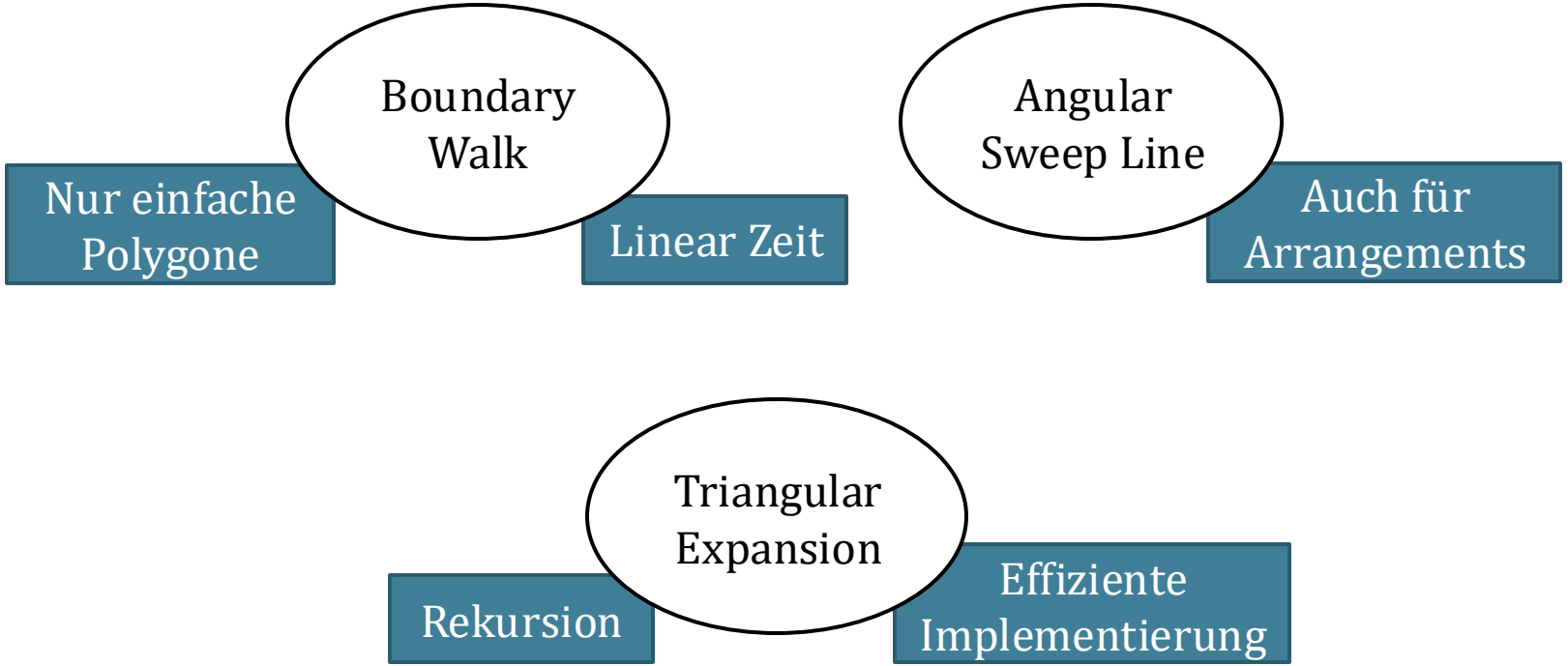
Betrachte das Polygon mit

- Äußerer Hülle
  $\left(-1, -\frac{\Delta}{2}\right), \left(\Delta, -\frac{\Delta}{2}\right), \left(\Delta, \frac{\Delta}{2}\right), \left(-1, \frac{\Delta}{2}\right)$
- Löchern
  $(x_i, 0) \pm (\varepsilon, \varepsilon)$

Berechne nun $\mathcal{V}\left(x = \left(-1, -\frac{\Delta}{2}\right)\right)$

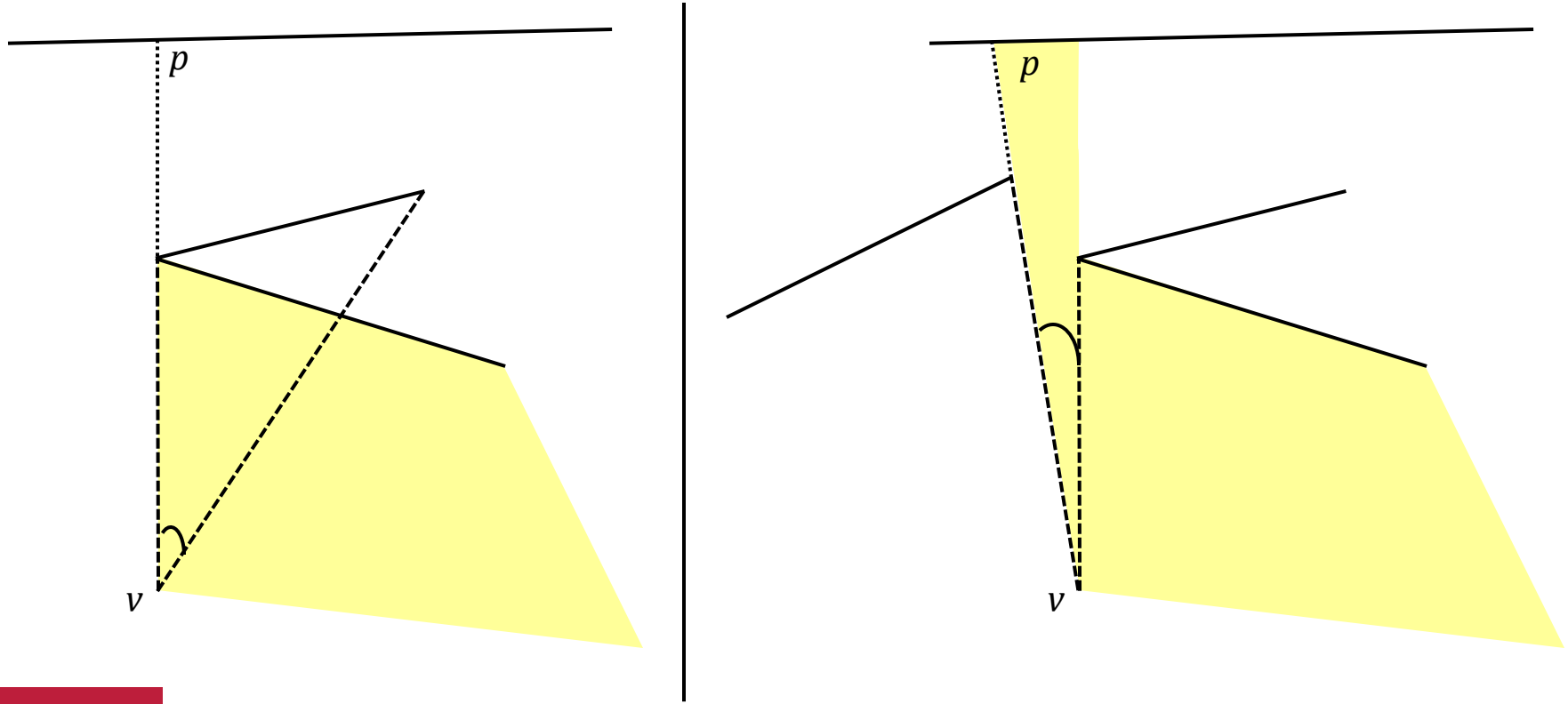


Drei Algorithmen



Kapitel 4.3.2 – Angular Sweep Line

Segmente enden



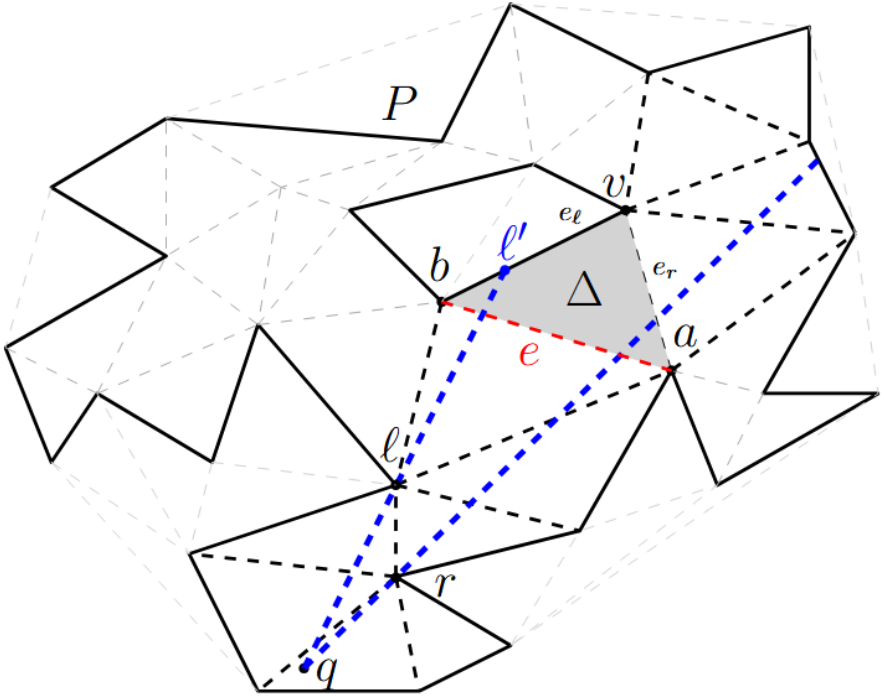
Kapitel 4.3.3 – Triangular Expansion

Idee – Triangulation

Nutze eine Triangulation, um das Visibilitypolygon nach und nach aufzubauen.

Dafür eignet sich Rekursion:

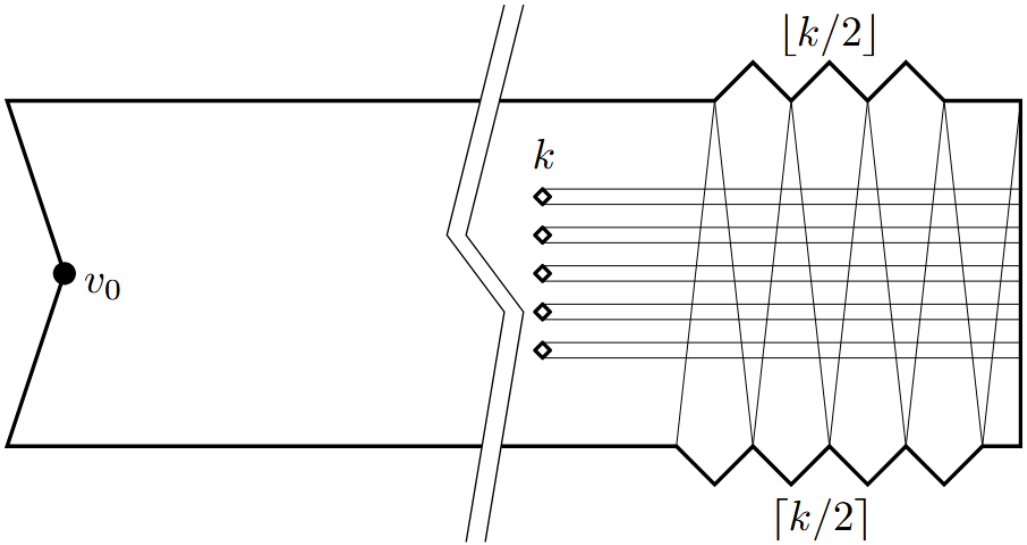
- Ein Dreieck wird mit einem Sichtbereich über die Kante betreten.
- Bestimme, bei welchen benachbarten Dreiecken wir mit welchem Sichtbereich weiter machen



Worst-Case-Beispiel

Dreiecke können im Worst-Case sehr oft rekursiv aufgerufen werden.

→ Worst-Case-Lauftzeit ist in $\Omega(n^2)$



Veranstaltungen im Winter

Veranstaltung	Empfehlung	Inhalte
Mathematische Methoden der Algorithmik (Wahlpflicht **Masterbereich** Informatik / Wirtschaftsinformatik)	Lineare Algebra bestanden Interesse an linearer Algebra	(Ganzzahlige) lineare Optimierung Simplex-Algorithmus Dualität
Parametrisierte Algorithmen (Neu! Wahlpflicht Informatik)	AuD 1 + 2 bereits bestanden (Theo Inf 2 bereits bestanden)	Kernelisation FPT Komplexitätstheorie “Was macht Probleme schwer zu lösen?”