

Visualisierung

Vorlesung im Wintersemester 2016/17
3) Interpolation und Filterung

Dozent: Prof. Dr. Boris Neubert

Folien: Prof. Dr. Carsten Dachsbacher

Lehrstuhl für Computergrafik

Karlsruher Institut für Technologie



Literaturhinweise

► **Scientific Visualization:**

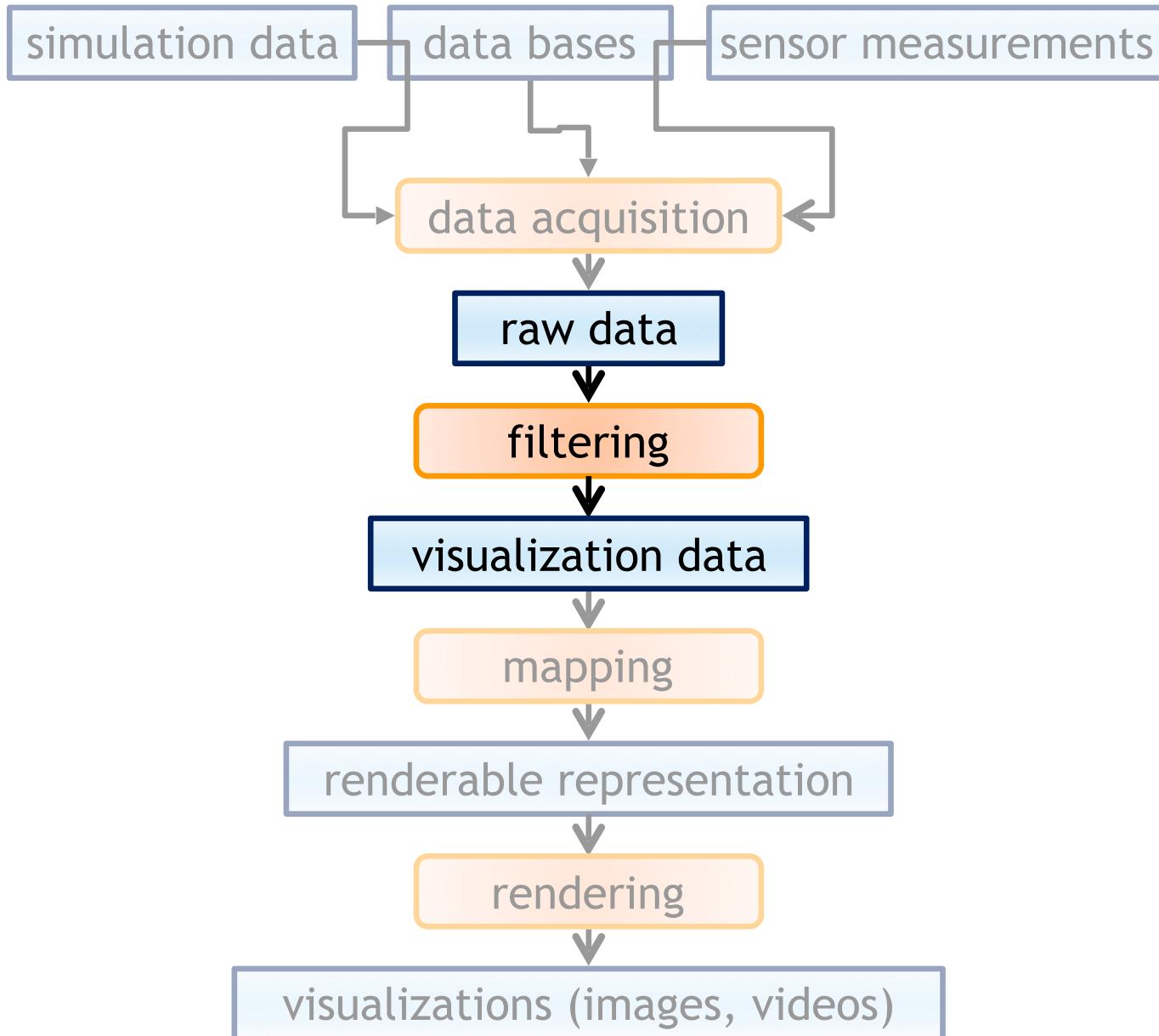
- Kapitel 11: Controlled Interpolation for Scientific Visualization
- Kapitel 20: Tools for Triangulation and Tetrahedrizations
- Hagit Shatkay: *The Fourier Transform - A Primer*, Department of Computer Science, Brown University, CS-95-37, November 1995

Zusätzliche Literatur / Hintergrundinformation

- G. Farin: Curves and Surfaces for Computer Aided Geometric Design, Morgan-Kaufmann, San Francisco. 5th Edition, 2001

Interpolation und Filtering

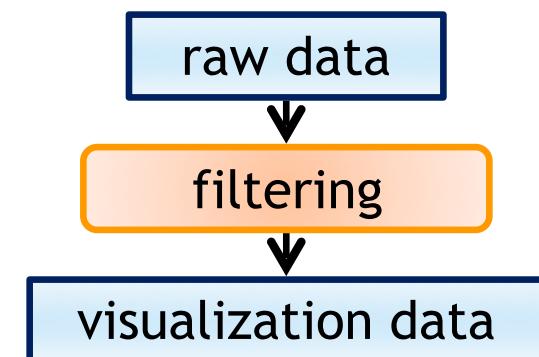
► wir befassen uns mit...



Visualisierungspipeline

Filtering / Datenaufbereitung

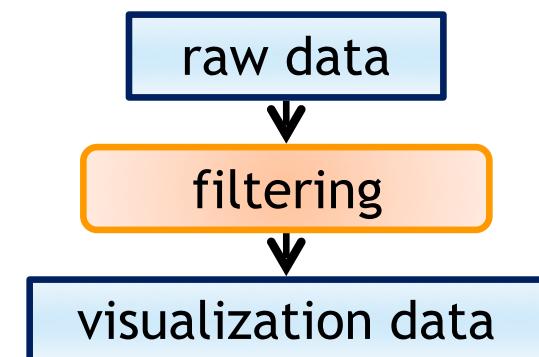
- ▶ Abbildung von (Roh-)Daten zu (Visualisierungs-)Daten
- ▶ Beispiele für Aufbereitungsoperationen
 - ▶ Konversion von Datenformaten
 - ▶ Vervollständigung der Datenmenge, z.B. durch Interpolation
 - ▶ Reduzierung der Datenmenge, z.B. Clipping, Cropping
 - ▶ Resampling oder Rauschentfernung
 - ▶ Schwellwertoperationen (Outlier-Removal)
 - ▶ Approximation, Slicing, Klassifikation, Segmentierung, ...



Visualisierungspipeline

Filtering / Datenaufbereitung

- ▶ Abbildung von (Roh-)Daten zu (Visualisierungs-)Daten
- ▶ Beispiele für Aufbereitungsoperationen
 - ▶ Konversion von Datenformaten
 - ▶ Vervollständigung der Datenmenge, z.B. durch Interpolation
 - ▶ Reduzierung der Datenmenge, z.B. Clipping, Cropping
 - ▶ Resampling oder Rauschentfernung
 - ▶ Schwellwertoperationen (Outlier-Removal)
 - ▶ Approximation, Slicing, Klassifikation, Segmentierung, ...



Motivation

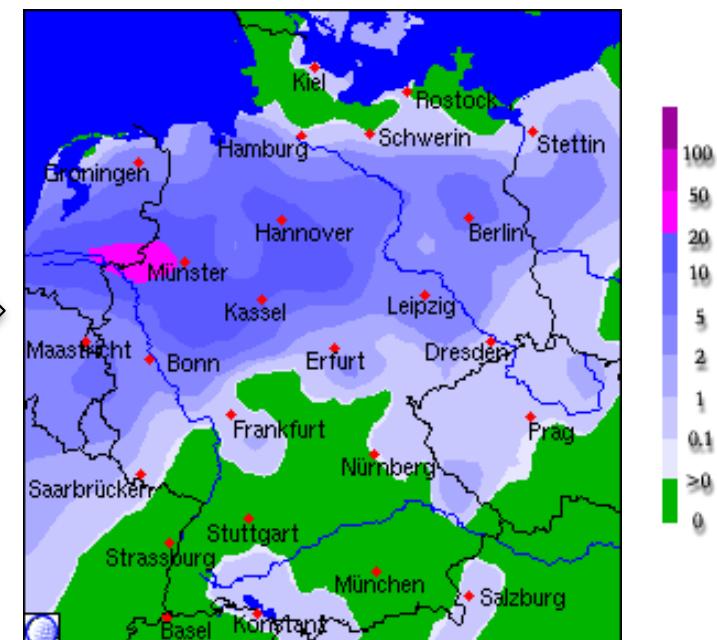
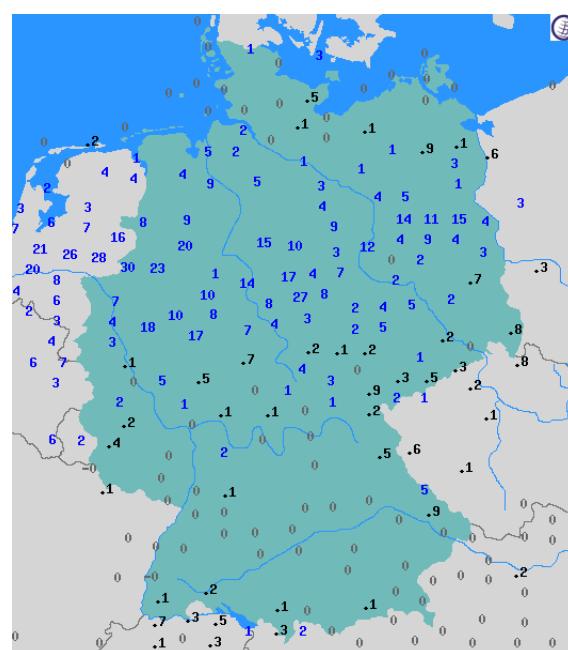
- ▶ oft sind Daten nur für diskrete Zeitpunkte oder Orte gegeben
- ▶ allgemein: wir kennen nur eine endliche Zahl an Messwerten
 - ▶ das (u.U.) kontinuierliche Signal ist nur an den Datenpunkten bekannt
 - ▶ für die Visualisierung benötigen wir Daten dazwischen
- ▶ durch Interpolation erhalten wir eine Darstellung die mit der tatsächlichen Funktion zumindest an den Datenpunkten übereinstimmt
 - ▶ erfordert Annahmen über die Funktion (meist: Glattheit)
 - ▶ eine Auswertung/Rekonstruktion des Signals an beliebigen Stellen ist dann möglich

Interpolation und Filtering

Motivation

► Bsp. Niederschlag innerhalb von 6 Stunden

<u>Aachen (205 m)</u>	6.5
<u>Ahaus (46 m)</u>	30.0
<u>Angermünde (55 m)</u>	1.0
<u>Artern (166 m)</u>	2.5
<u>Aue (397 m)</u>	0.3
<u>Augsburg/Mühlhausen (477 m)</u>	0.0
<u>Bad Hersfeld (273 m)</u>	0.7
<u>Balingen (266 m)</u>	0.1
<u>Brieselang (158 m)</u>	10.0
<u>Wittenberg (106 m)</u>	0.0
<u>Würzburg (272 m)</u>	0.0
<u>Zinnwald (882 m)</u>	0.3
<u>Zugspitze (2962 m)</u>	1.9
<u>Zwiesel (613 m)</u>	0.9
<u>Öhringen (277 m)</u>	0.1

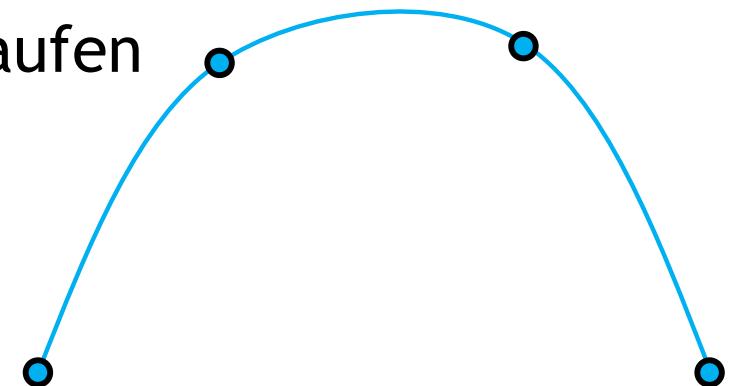


Quelle: www.wetteronline.de, 07.05.2007 20:00

Interpolation und Approximation

► Interpolation:

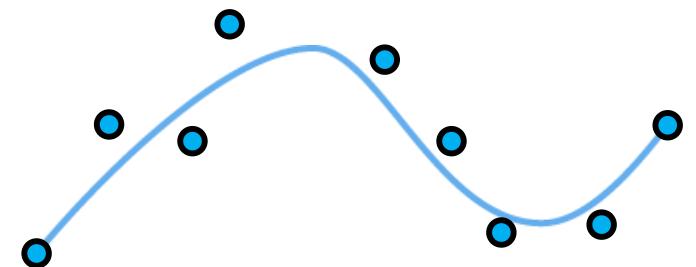
alle vorgegebenen Punkte werden durchlaufen



► Approximation:

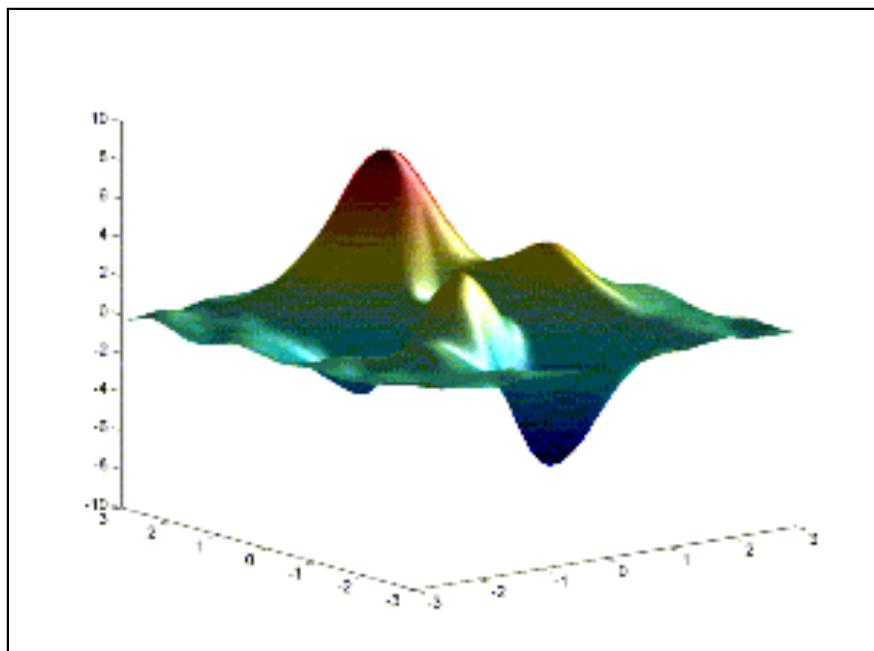
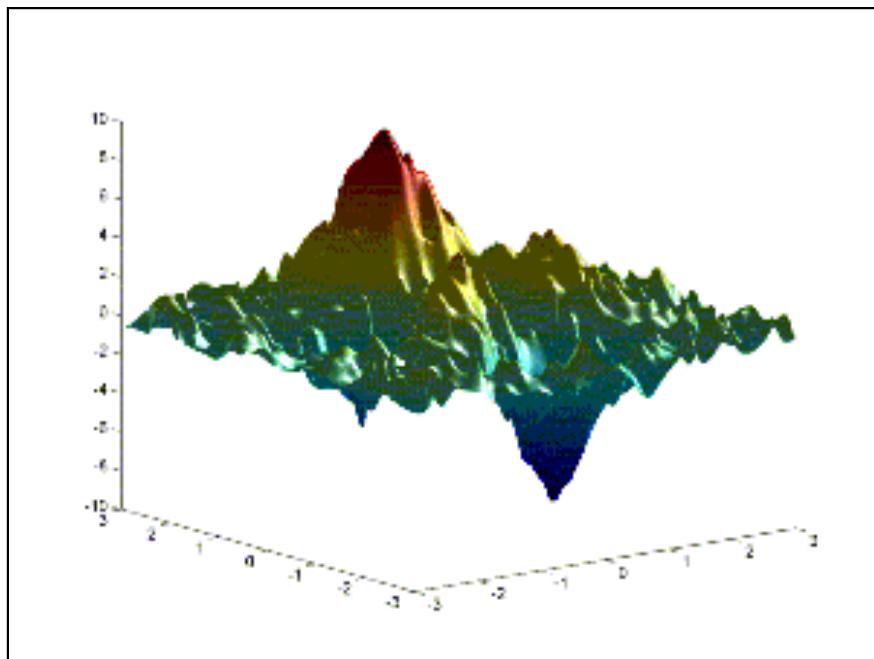
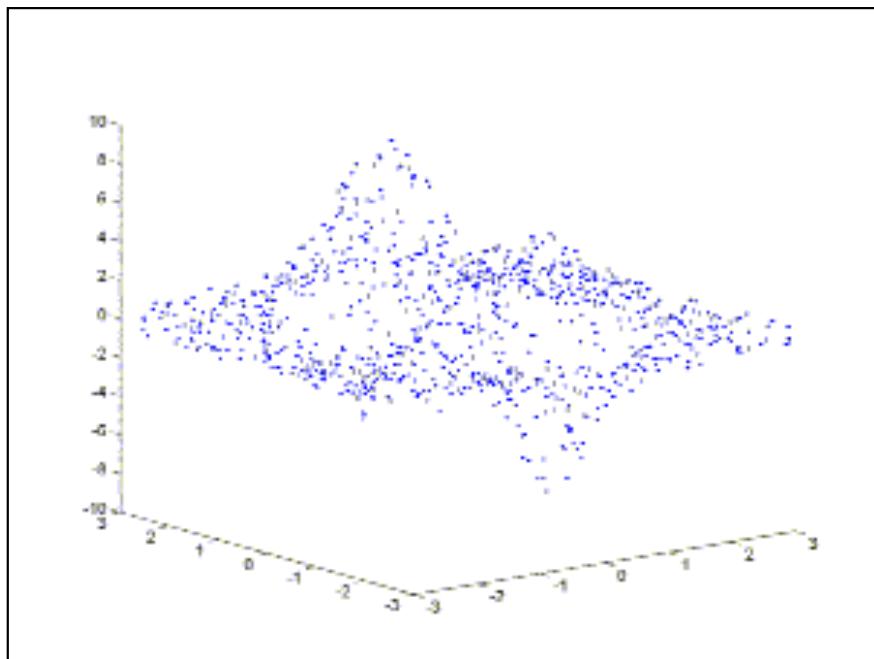
die Kurve geht nicht durch alle Punkte

► u.U. stabiler bei vielen Datenpunkten



Interpolation und Approximation

Motivation



Interpolation

Grundlegendes

- ▶ geg. eine Funktion $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ von der wir nur einige diskrete Funktionswerte $f(x_i)$, $x_i \in \mathbb{R}^n$, kennen
- ▶ ges. eine Funktion („Interpolant“) $\phi(x)$, $x \in \mathbb{R}^n$, die sich möglichst wenig von f unterscheidet und leicht auszuwerten ist
- ▶ bei der Wahl des Interpolanten $\phi(x)$ treffen wir Annahmen über die nicht vollständig bekannte Funktion $f(x)$

Interpolationsmethoden

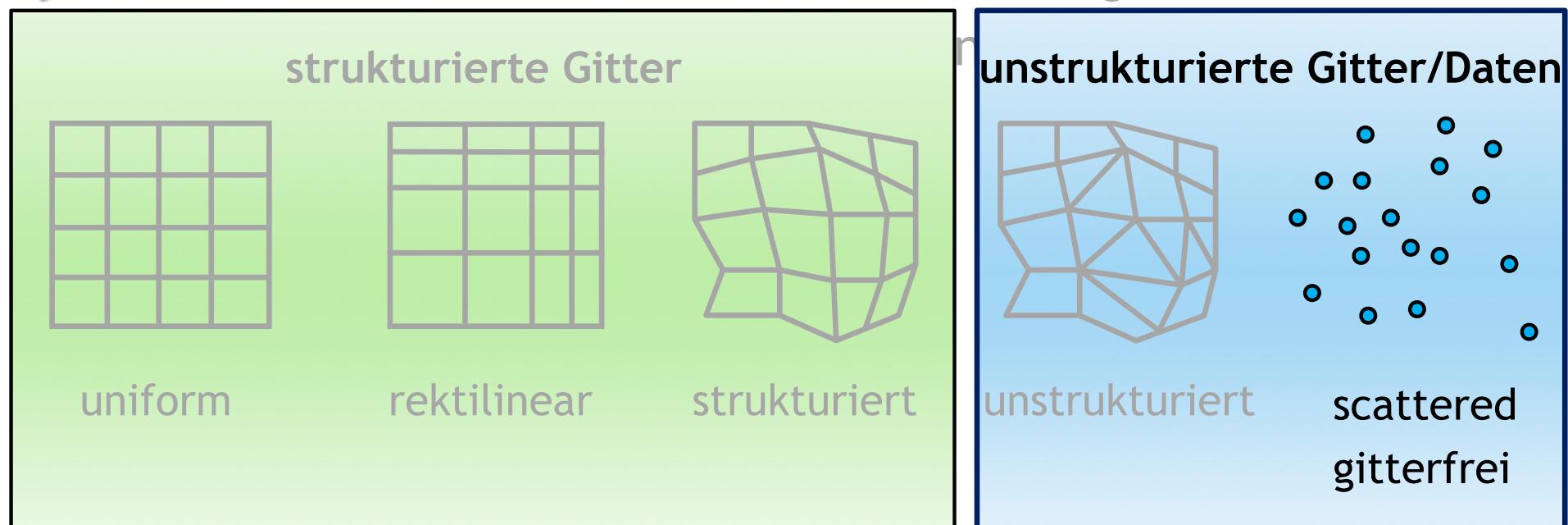
Klassifikation

- lokale Interpolationsmethoden (zellenweise)
 - verwendet die Information der direkt anliegenden Vertizes, z.B.
 - Interpolation in einem Dreieck eines Netzes
 - Interpolation in einem Hexaeder eines Gitters
 - ...
- globale Interpolationsmethoden
 - verwendet Informationen aller Vertizes/Datenpunkte oder zumindest einer größeren Nachbarschaft, z.B.
 - Shepard-Interpolation, Radiale Basisfunktionen
 - Splines und darauf basierende Tensorprodukte

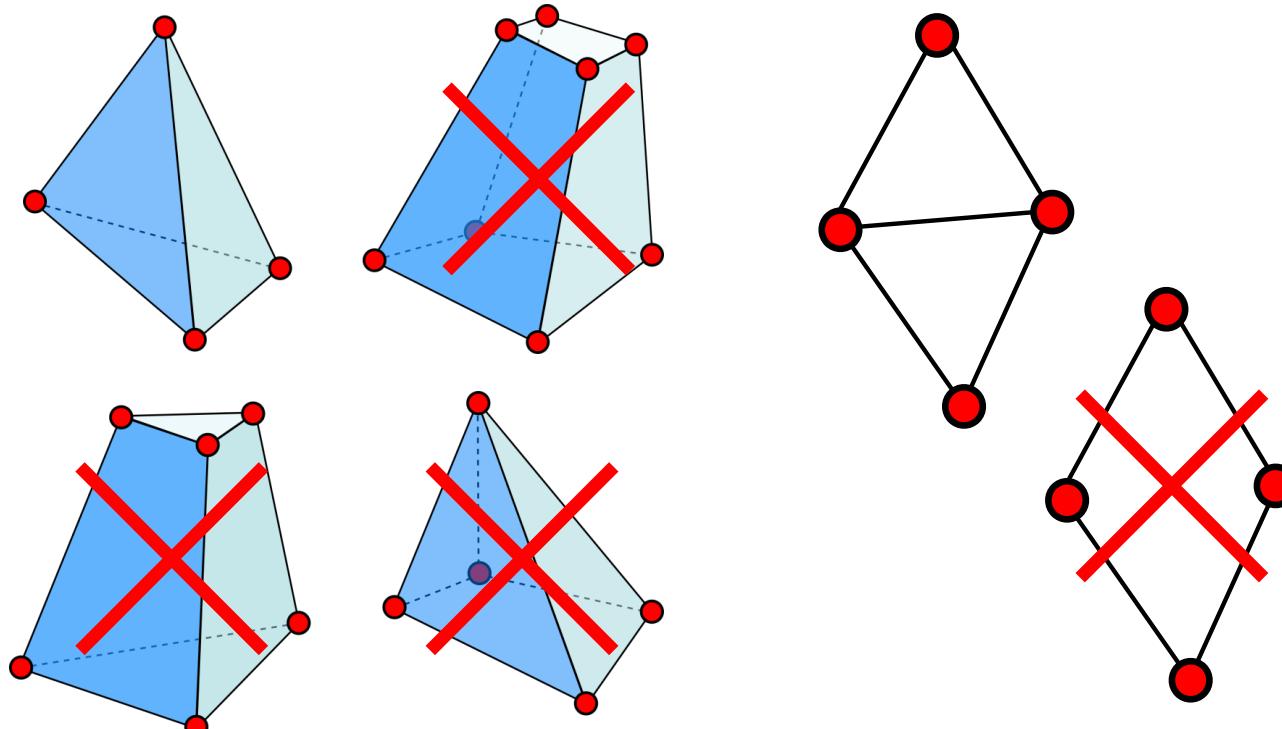
Datenstrukturen

Gitterfreie Daten und Gittertypen

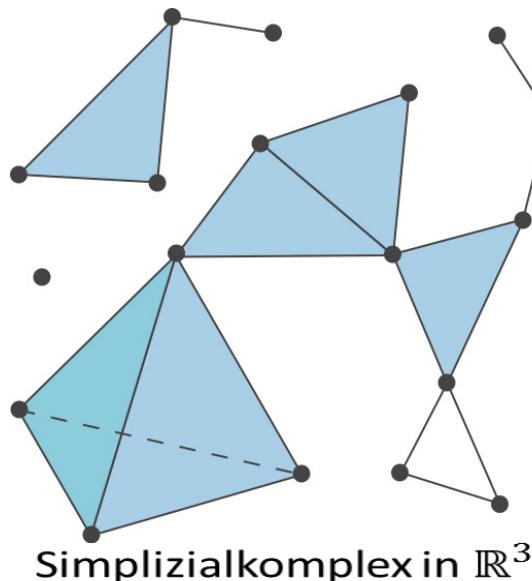
- gitterfreie Daten: irregulär verteilte Positionen, keine Konnektivität
- Konnektivität herstellen und dann (in Simplizes) interpolieren, oder
- direkte Interpolation anhand aller Datenpunkte
- Gitter unterscheiden sich in
- Art und Form der Zellen aus denen sie aufgebaut sind



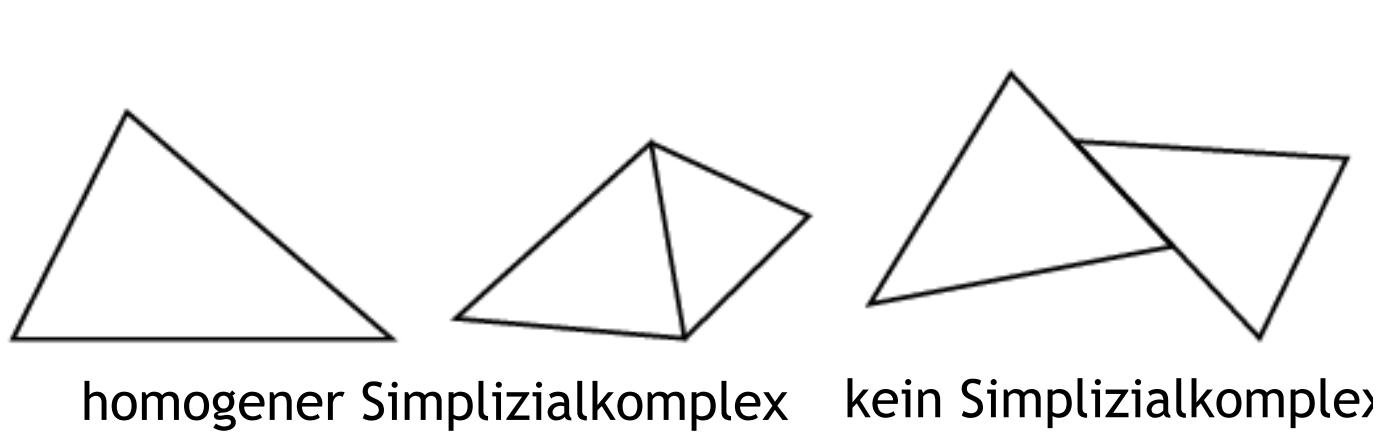
- ▶ oft erzeugt man ein Gitter aus/für gitterfreien Daten: kein wdh.
Berechnen von Voronoi-Zerlegungen, schnelle Interpolation zur Laufzeit
- ▶ ein n -Simplex ist
 - ▶ die konvexe Hülle von $n + 1$ affin-unabhängigen Punkten
 - ▶ das kann es nur im \mathbb{R}^m mit $n \leq m$ geben
 - ▶ $n = 0$ Punkte, $n = 1$ Linien, $n = 2$ Dreiecke, $n = 3$ Tetraeder



- ▶ die Unterteilung eines Raums mit Simplizes nennt man Triangulierung
- ▶ ein Simplizialkomplex C ist eine Menge von Simplizes mit Eigenschaften:
 - ▶ jedes „Face“ (Facette) eines Elementes in C ist ebenfalls in C
(Achtung: eine Facette kann jeder Typ von Simplex sein)
 - ▶ Schnitt zweier Elemente aus C ist leer, oder ein Face beider Elemente
- ▶ ein homogener simplizialer k -Komplex ist eine Triangulierung, die nur aus k -dimensionalen Simplizes besteht
- ▶ wenn Sie mehr wissen wollen → algebraische Topologie

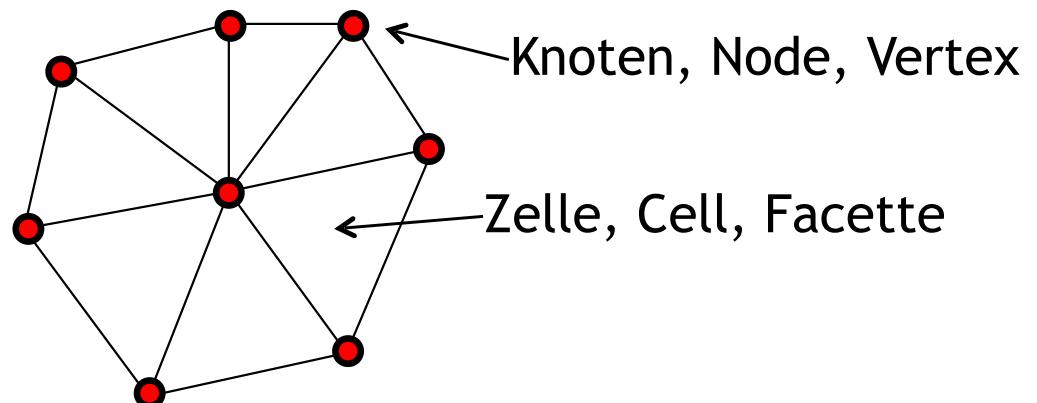
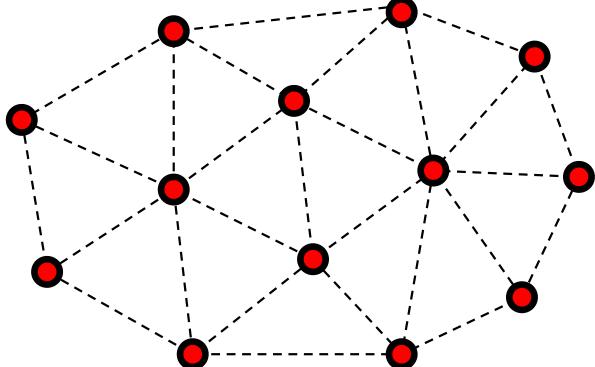
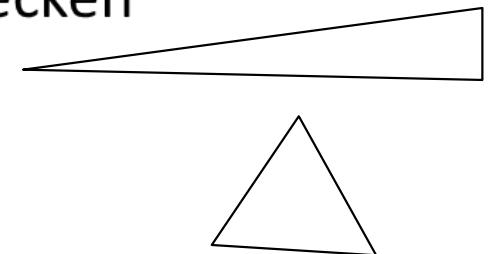


Simplizialkomplex in \mathbb{R}^3



Triangulation einer Punktmenge

- ▶ geg. beliebig verteilt Punkte ohne Konnektivitätsinformation
- ▶ gibt es immer eine Triangulation? Ja!
(es sei denn alle Punkte liegen auf einer Geraden in \mathbb{R}^2 bzw. \mathbb{R}^3 , ...)
- ▶ ges. finde Konnektivität einer „guten“ Triangulation
- ▶ es gibt viele mögliche Triangulationen – welche ist gut?
 - ▶ ein Qualitätsmerkmal ist das Aspect Ratio von Dreiecken
 - ▶ vermeide lange, dünne Dreiecke



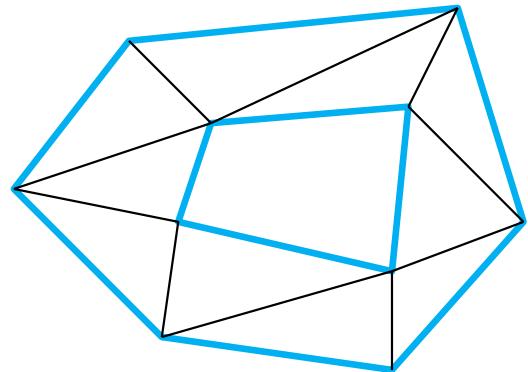
Triangulation einer Punktmenge in \mathbb{R}^2

- ▶ eine Triangulation von Punkten $S = \{s_1, \dots, s_n\} \in \mathbb{R}^2$ besteht aus
 - ▶ Knoten/Vertizes (0D) = S
 - ▶ Kanten (1D), die je 2 Vertizes verbinden
 - ▶ Zellen/Faces (2D), die je 3 Vertizes verbinden
- ▶ eine Triangulation muss folgende Anforderungen erfüllen
 - ▶ (1) die Vereinigung aller Dreiecke (inkl. dem Rand) entspricht der konvexen Hülle der Vertizes S
 - ▶ (2) die Schnittmenge zweier Dreiecke ist entweder leer, oder
 - ▶ ein gemeinsamer Knoten, oder
 - ▶ eine gemeinsame Kante, oder
 - ▶ bei Triangulation im \mathbb{R}^3 : eine gemeinsame Fläche zweier Tetraeder
 - ▶ (3) die Triangulation ist eine Partitionierung der Ebene bzw. des Raums (keine Überschneidungen)

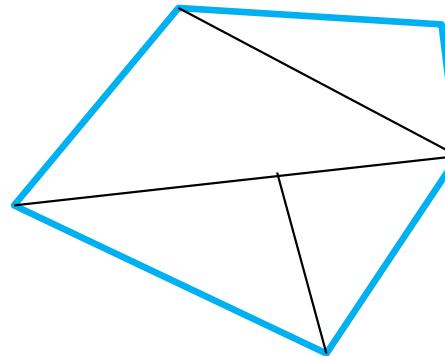
Fehlerhafte Triangulationen (in \mathbb{R}^2)

- ... haben beispielsweise ...

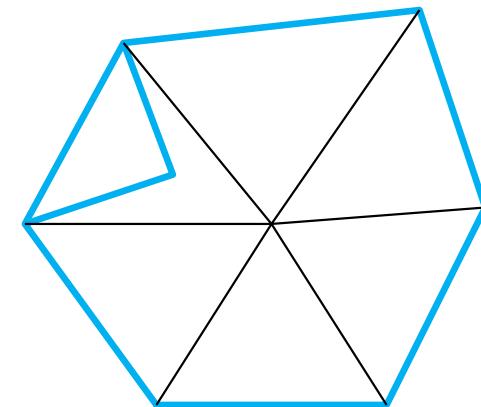
Löcher (1)



T-Vertizes (2)



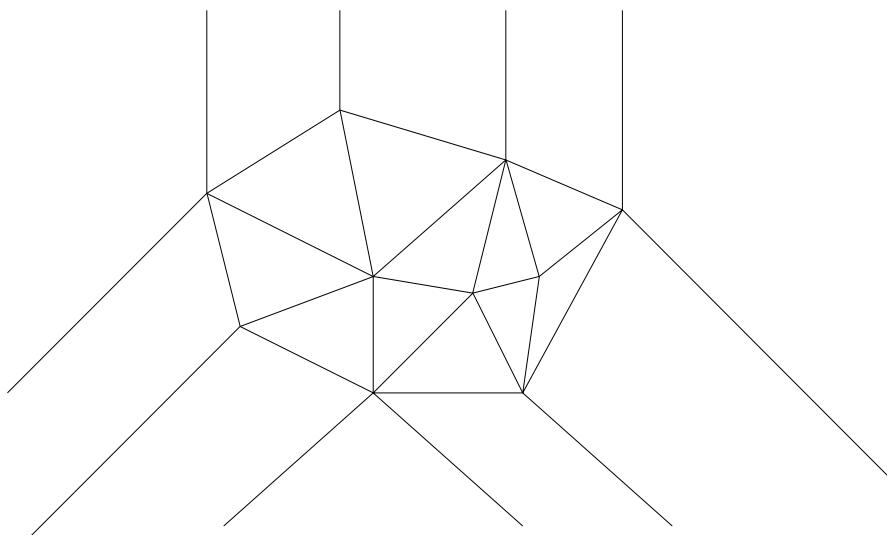
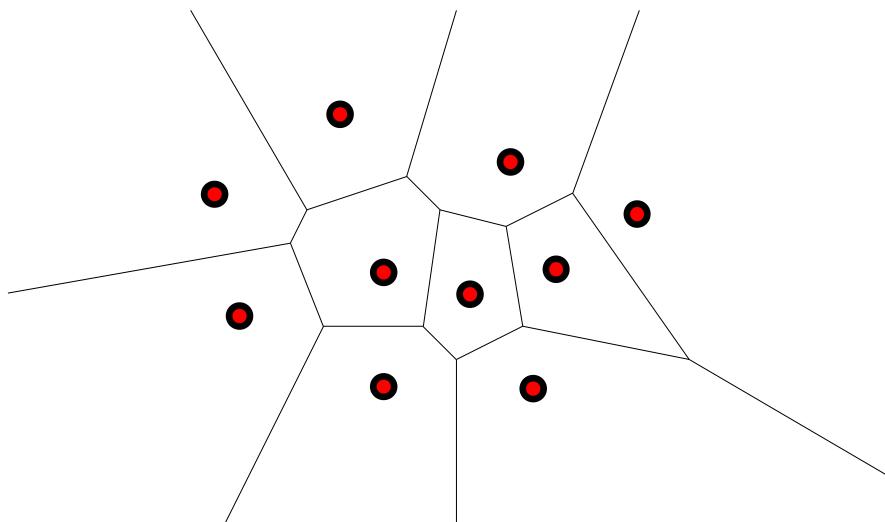
überlappendende Dreiecke (3)



- blaue Kanten sind nur Teil eines Dreiecks
- T-Vertizes (Mitte): der Schnitt zweier Dreiecke ist hier nur eine Teilkante

Triangulation einer Punktmenge in \mathbb{R}^2

- wie bekommt man nun also eine Triangulation oder Konnektivitätsinformation?
- zwei Strukturen sind hier wichtig:
 - ▶ Voronoi Diagramme und
 - ▶ Delaunay Triangulation
 - ▶ beide stehen in engem Zusammenhang („haben Sie eine berechnet, dann kennen Sie auch die andere“)



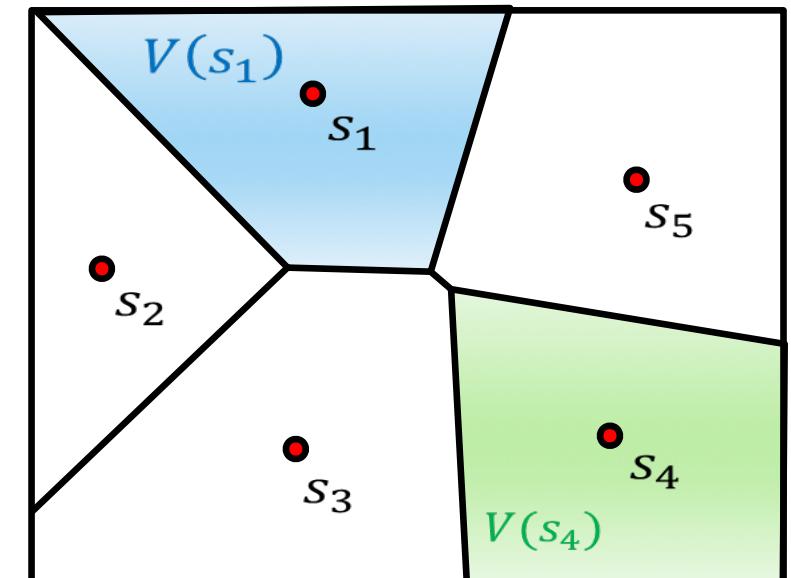
Voronoi Diagramme

Definition

- ▶ alle Punkte in der Voronoi-Region eines Datenpunktes sind näher an diesem Datenpunkt als an allen anderen
- ▶ für eine Menge von Datenpunkten $S = \{s_1, \dots, s_n\} \in \mathbb{R}^d$ und eine Abstandsfunktion/Metrik $d(x, y): \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$
- ▶ enthält das Voronoi-Diagramm $Vor(S)$ für jeden Datenpunkt s_i eine Zelle $V(s_i)$ mit

$$V(s_i) = \{x | dist(x, s_i) < dist(x, s_j) \forall j \neq i\}$$

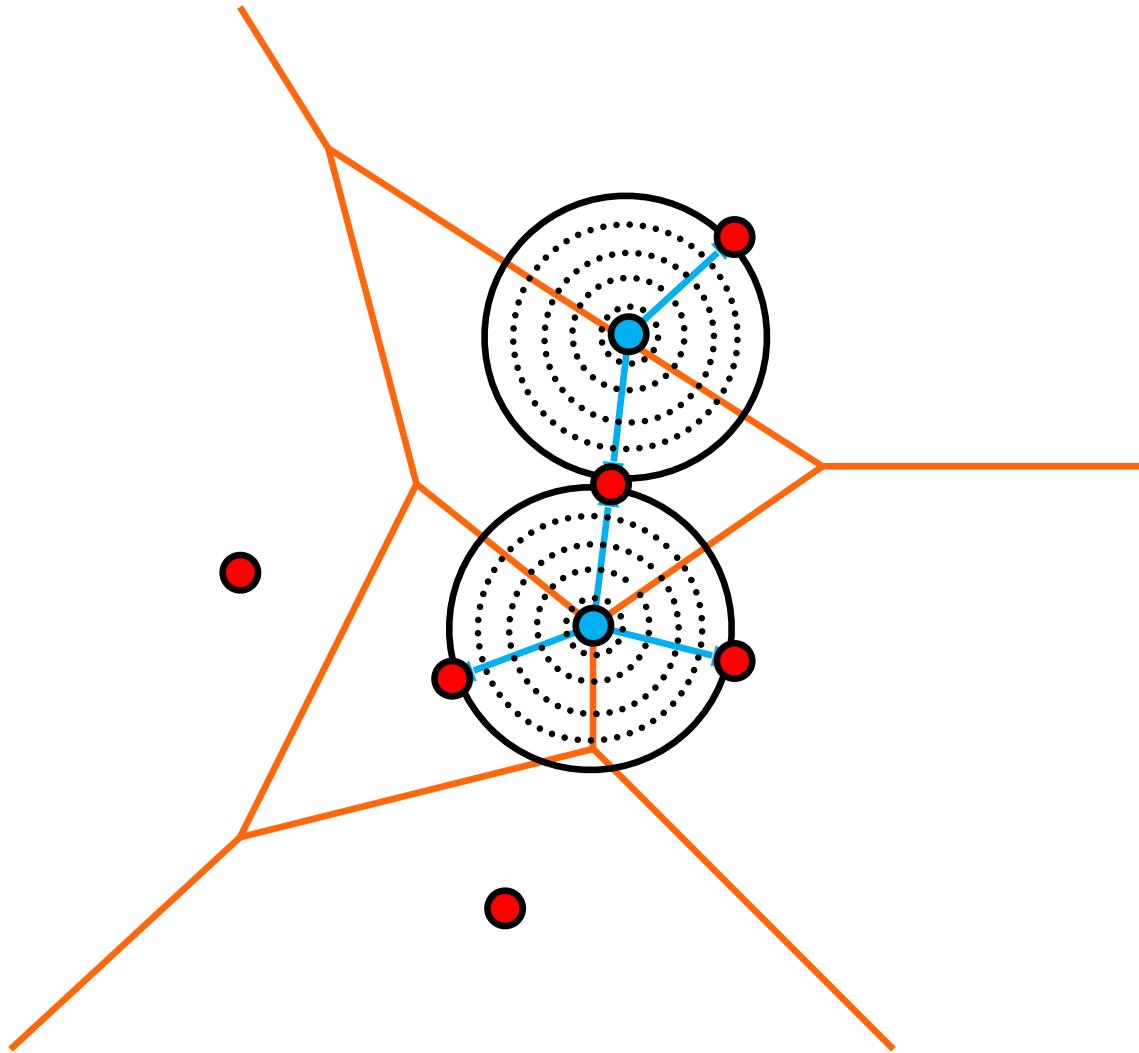
- ▶ Anwendung: der Wert eines Datenpunkts wird für die gesamte Zelle angenommen
 - ▶ dazu muss man nicht die Form der Voronoi-Zelle bestimmen: für einen Punkt x sucht man s_i mit $\min_i dist(x, s_i)$



Voronoi Diagramme

Beispiel

Datenpunkte s_i ●



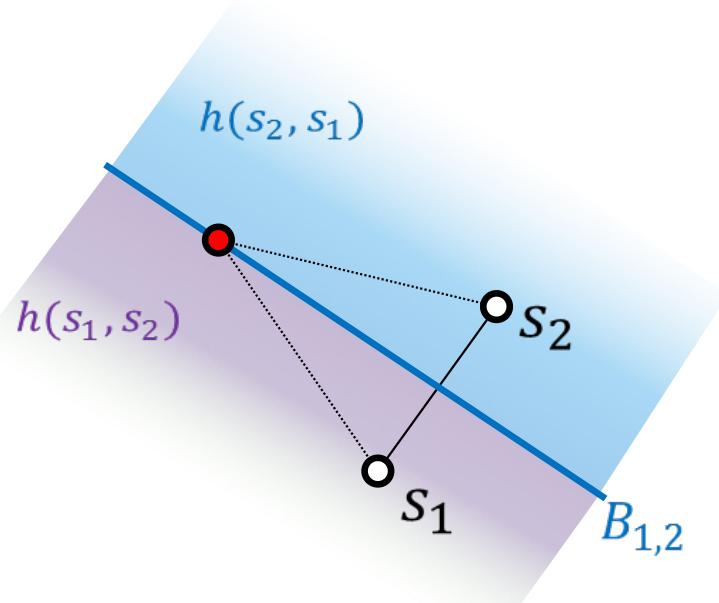
Voronoi Diagramme

Voronoi-Zellen

- ▶ der Halbraum $h(s_i, s_j)$ enthält den Punkt s_i und wird begrenzt durch die Mittelsenkrechte der Strecke zwischen s_i und s_j
- ▶ die Voronoi-Zelle erhält man durch Schnitt aller Halbräume

$$V(s_i) = \cap_{j \neq i} h(s_i, s_j)$$

⇒ Voronoi-Zellen sind immer konvex



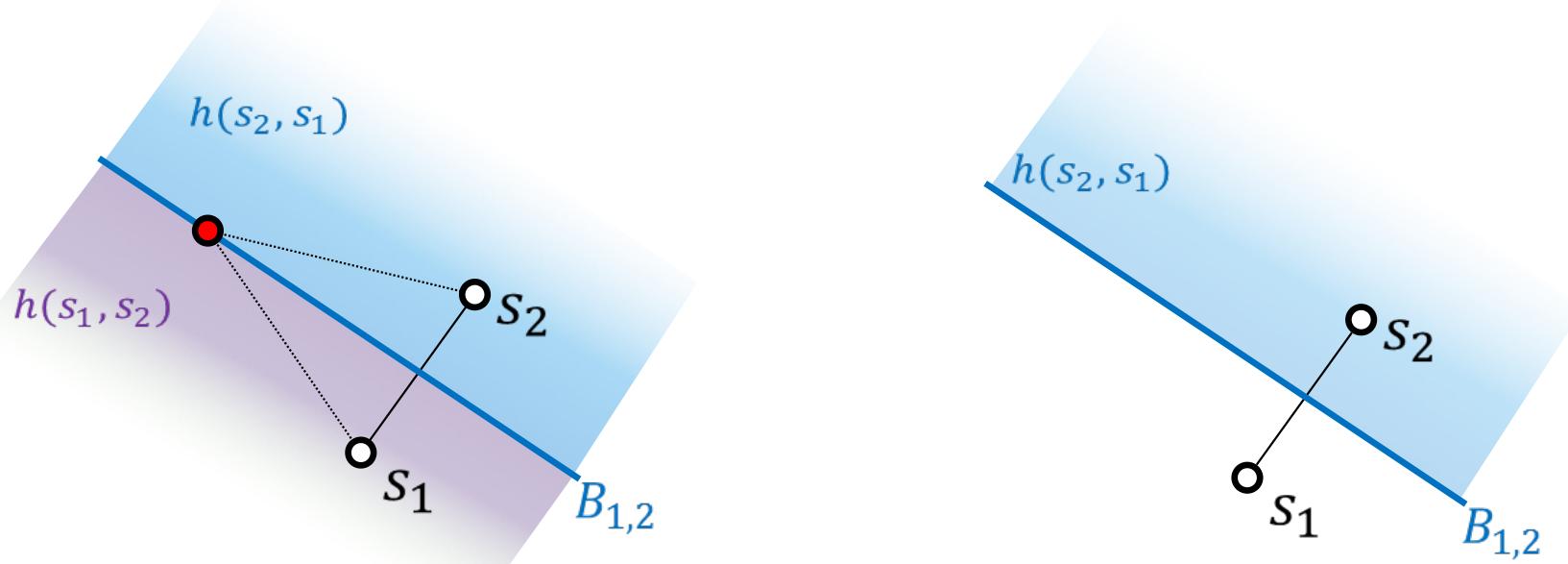
Voronoi Diagramme

Voronoi-Zellen

- ▶ der Halbraum $h(s_i, s_j)$ enthält den Punkt s_i und wird begrenzt durch die Mittelsenkrechte der Strecke zwischen s_i und s_j
- ▶ die Voronoi-Zelle erhält man durch Schnitt aller Halbräume

$$V(s_i) = \cap_{j \neq i} h(s_i, s_j)$$

⇒ Voronoi-Zellen sind immer konvex



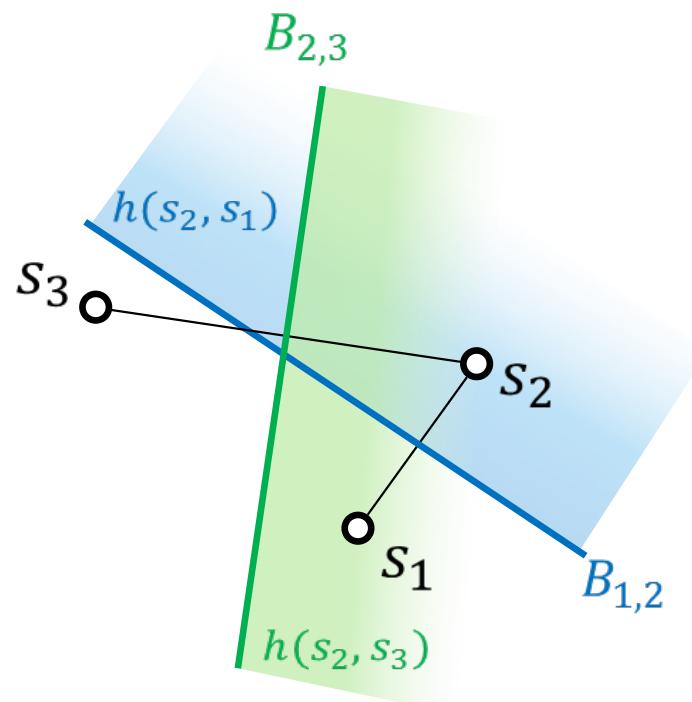
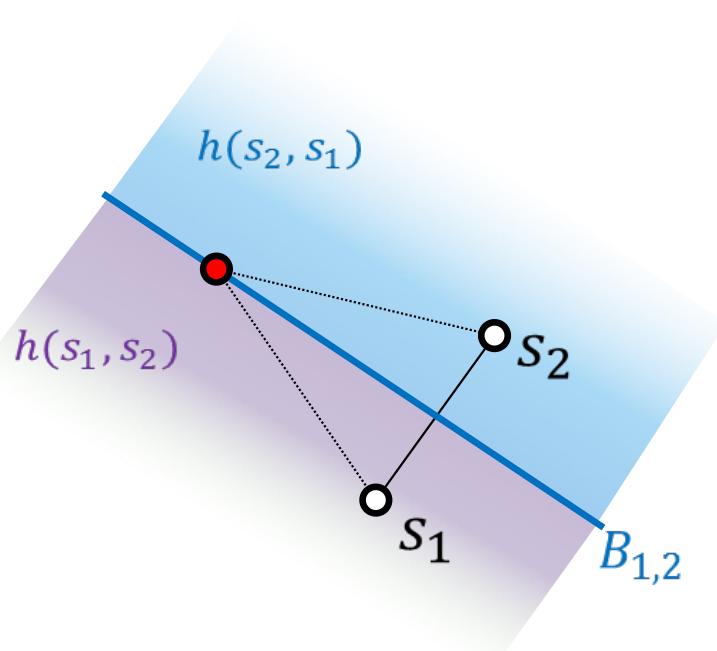
Voronoi Diagramme

Voronoi-Zellen

- ▶ der Halbraum $h(s_i, s_j)$ enthält den Punkt s_i und wird begrenzt durch die Mittelsenkrechte der Strecke zwischen s_i und s_j
- ▶ die Voronoi-Zelle erhält man durch Schnitt aller Halbräume

$$V(s_i) = \cap_{j \neq i} h(s_i, s_j)$$

⇒ Voronoi-Zellen sind immer konvex



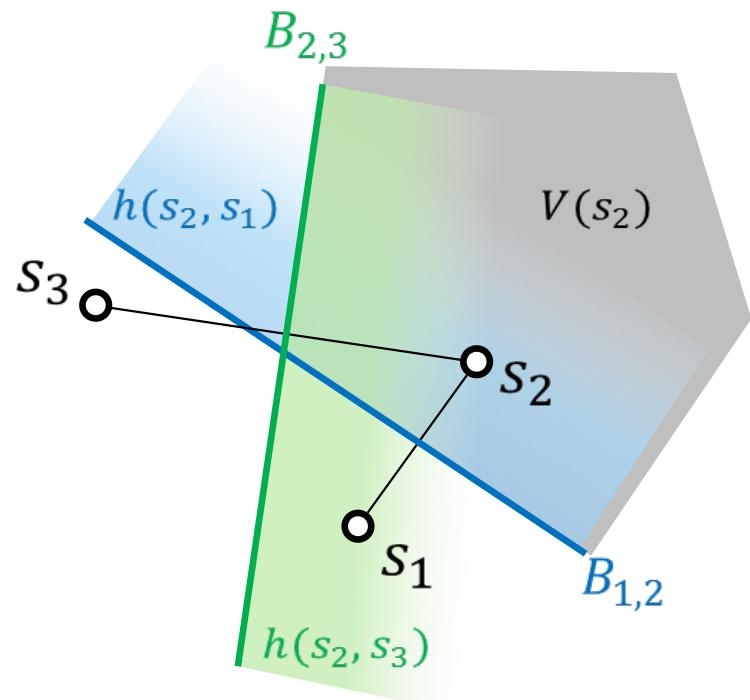
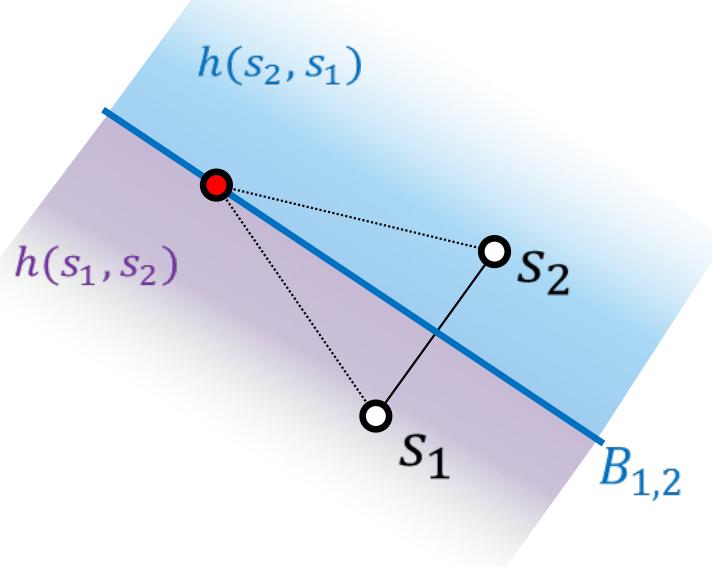
Voronoi Diagramme

Voronoi-Zellen

- ▶ der Halbraum $h(s_i, s_j)$ enthält den Punkt s_i und wird begrenzt durch die Mittelsenkrechte der Strecke zwischen s_i und s_j
- ▶ die Voronoi-Zelle erhält man durch Schnitt aller Halbräume

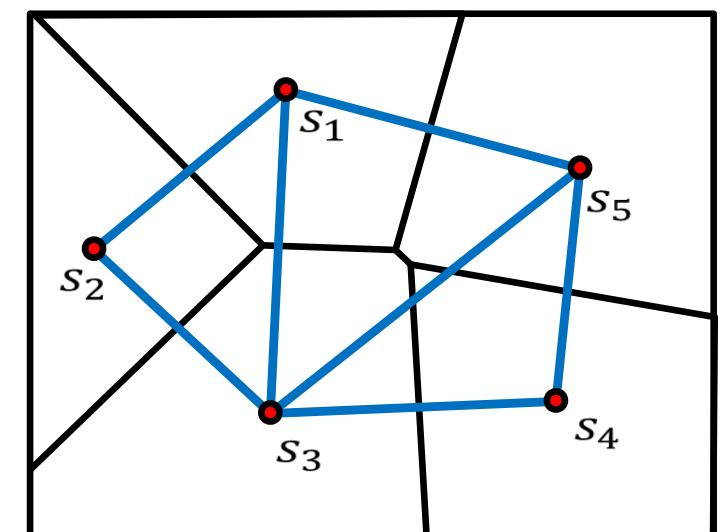
$$V(s_i) = \cap_{j \neq i} h(s_i, s_j)$$

⇒ Voronoi-Zellen sind immer konvex



Voronoi Diagramme und Delaunay Triangulation

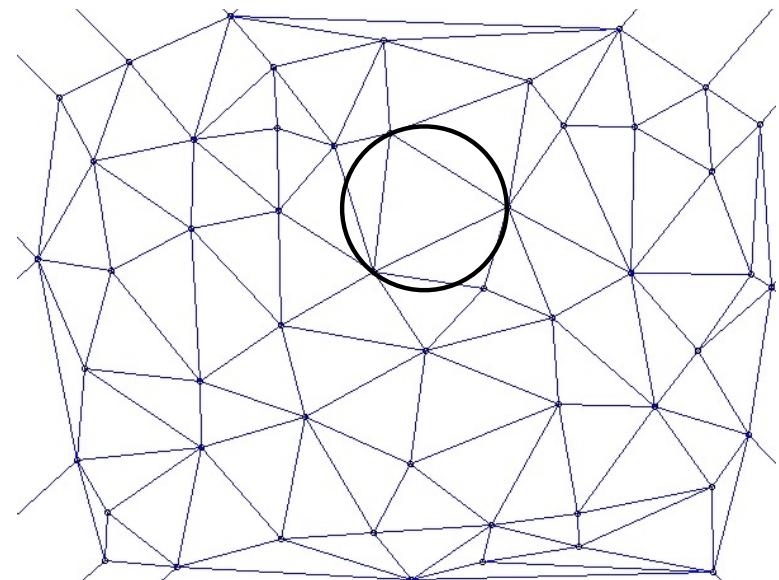
- ▶ die Delaunay Triangulation $D(S)$ ist der duale Graph des Voronoi-Diagramms, d.h.
 - ▶ die Datenpunkte s_i sind die Knoten der Triangulation
 - ▶ zwei Knoten s_i und s_j sind genau dann verbunden, wenn sich die Voronoi-Zellen $V(s_i)$ und $V(s_j)$ eine Kante teilen
 - ▶ die Ecken der Voronoi-Zellen sind die Umkreismittelpunkte der Dreiecke
- ▶ wenn wir die Konnektivität einer der beiden Repräsentationen berechnet haben, können wir daraus die jeweils andere erhalten
- ▶ wir betrachten die Delaunay-Triangulation (die Algorithmen zur Berechnung sind „ähnlich“)



Delaunay Triangulation

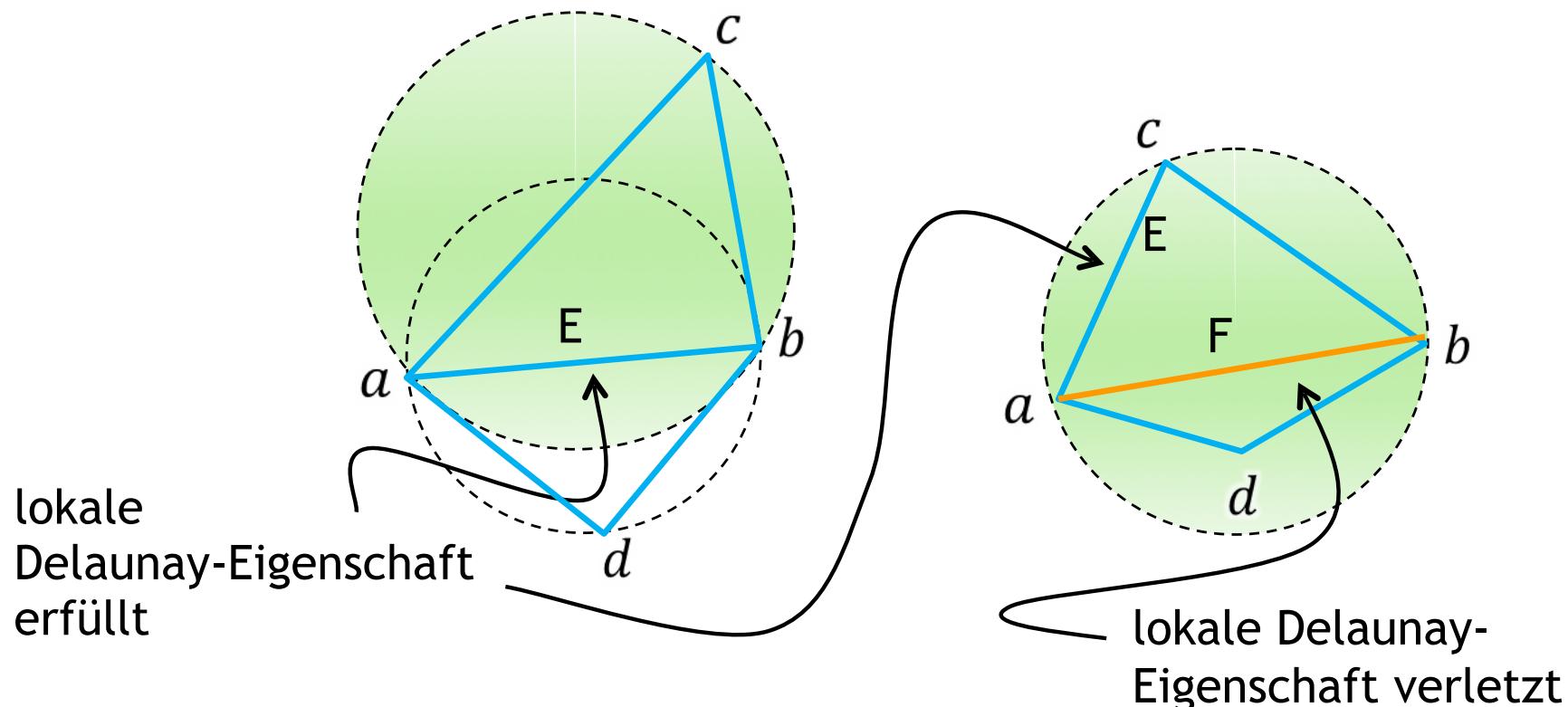
Delaunay Triangulation in \mathbb{R}^2

- wir müssen Knoten so verbinden, dass die folgende **Umkreisbedingung** erfüllt wird:
3 Punkte $s_i, s_j, s_k \in S$ bilden ein Dreieck aus $D(S)$ gdw. kein weiterer Punkt innerhalb des Kreises mit s_i, s_j, s_k
- Eigenschaften:
 - ▶ Maximierung des kleinsten Winkels der Triangulation
 - ▶ Maximierung der Verhältnisse: Radius Inkreis zu Radius Umkreis
 - ▶ abgesehen von einem Spezialfall ist die Triangulation eindeutig (d.h. unabh. von der Reihenfolge der Punkte)
 - ▶ Spezialfall: mehr als 3 Punkte liegen auf einem Kreis



Lokale Umkreisbedingung (für Konstruktionsalgorithmen)

- ▶ eine Kante zw. Punkten a und b erfüllt die lokale Bedingung, wenn
 - ▶ sie nur Teil eines Dreiecks ist (Randkante der konvexen Hülle), oder
 - ▶ sie zu 2 Dreiecken abc und abd gehört und d außerhalb des Umkreises von abc liegt und c außerhalb des Umkreises von abd



Delaunay Triangulation

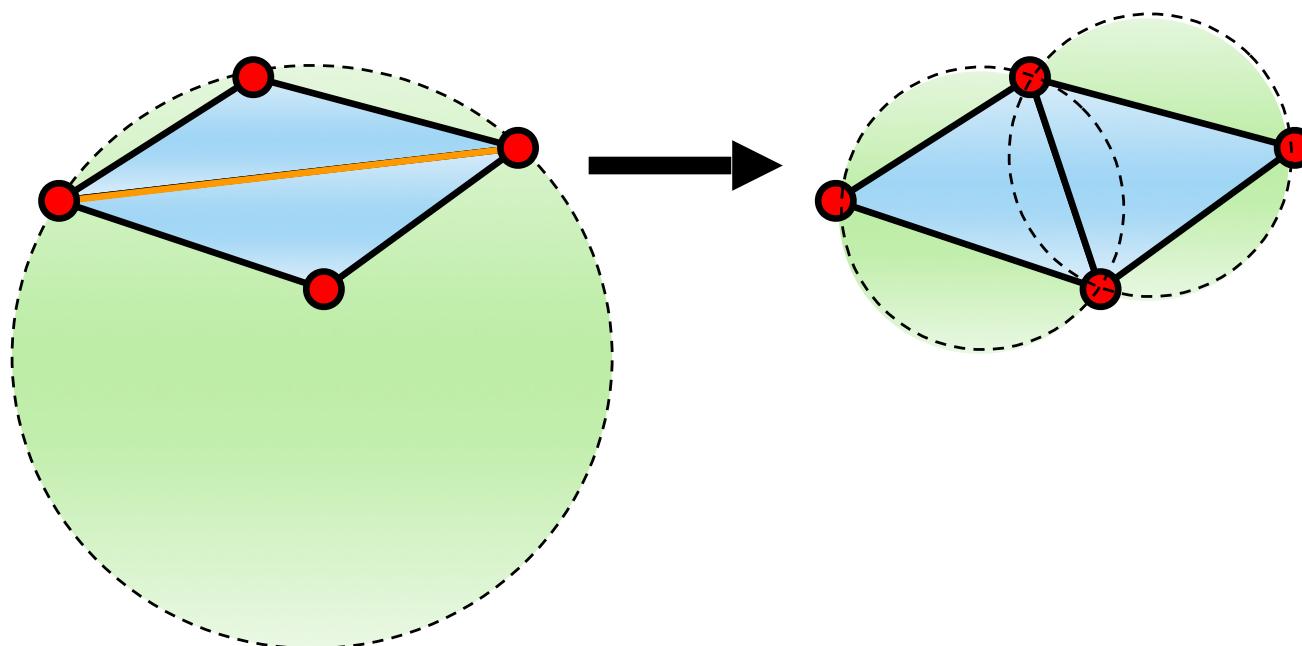
Edge Flip Algorithmus (nur für \mathbb{R}^2)

- ... direkte Umsetzung des lokalen Kriteriums:
ausgehend von einer beliebigen – aber zulässigen – Triangulation
(gleich!) wird die Delaunay-Eigenschaft hergestellt
- Algorithmus:
 - Erzeuge eine initiale zulässige Triangulation
 - Finde alle Kanten, an denen das lokale Kriterium verletzt wird
(d.h. betrachte die angrenzenden Dreiecke und deren Umkreise)
 - Markiere diese Kanten und lege sie auf einen Stack
 - Pseudocode:
 - while (Stack nicht leer)
 - nehme Kante vom Stack
 - „flippe“ diese Kante
 - überprüfe, ob dadurch das Kriterium in den Nachbardreiecken
verletzt wird (und aktualisiere Stack)
-

Delaunay Triangulation

Edge Flip Algorithmus (nur für \mathbb{R}^2)

- ▶ Prinzip: ausgehend von einer beliebigen zulässigen Triangulation wird lokal „repariert“, bis die Delaunay-Eigenschaft global erfüllt ist
 - ▶ durch Umdrehen (Flip) der Kante

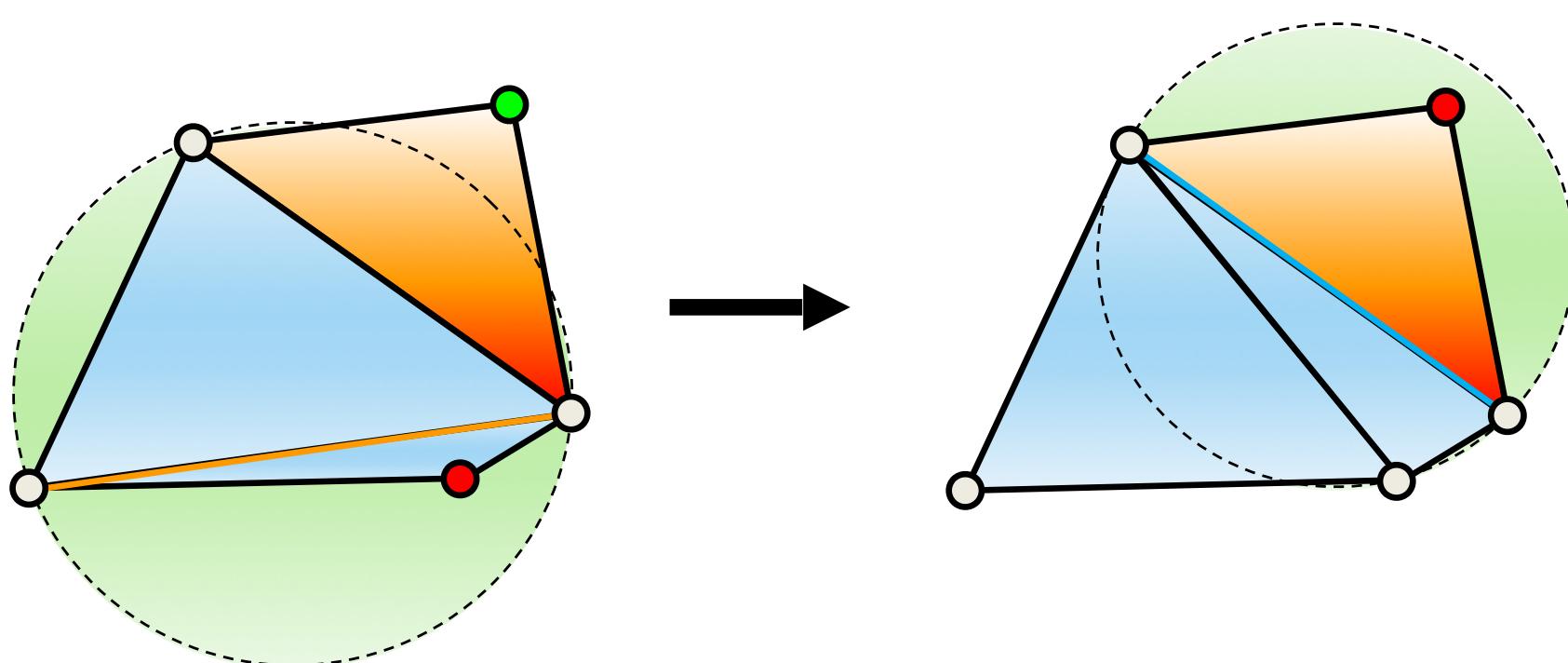


- ▶ Berechnung der Umkreise: der Mittelpunkt des Umkreises ist der Schnitt der Mittelsenkrechten (siehe auch Halbräume bei Voronoi-Diagrammen)

Delaunay Triangulation

Edge Flip Algorithmus (nur für \mathbb{R}^2)

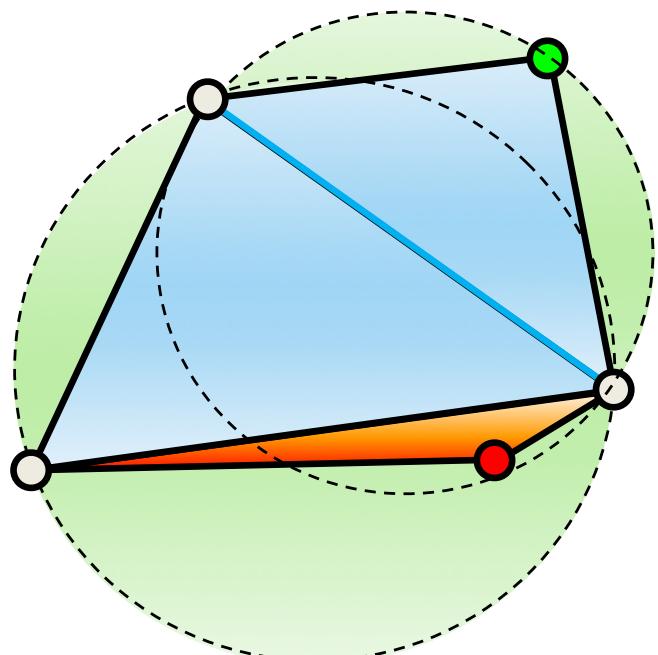
- Prinzip: ausgehend von einer beliebigen zulässigen Triangulation wird lokal „repariert“, bis die Delaunay-Eigenschaft global erfüllt ist
- Flipping kann die Delaunay-Eigenschaft benachbarter Dreiecke verletzen
 - links: der rote Punkt verletzt die lokale Bedingung → Flip
 - rechts: die blaue Kante muss jetzt anschließend geflipped werden!



Delaunay Triangulation

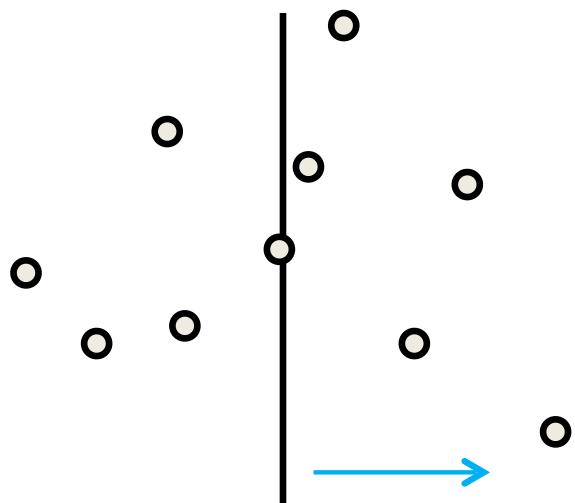
Edge Flip Algorithmus (nur für \mathbb{R}^2)

- ▶ Prinzip: ausgehend von einer beliebigen zulässigen Triangulation wird lokal „repariert“, bis die Delaunay-Eigenschaft global erfüllt ist
- ▶ Flipping kann die Delaunay-Eigenschaft benachbarter Dreiecke verletzen
- ▶ vor dem Flip auf der vorherigen Folie ist die **lokale Bedingung** nicht verletzt: der rote Punkt hat nichts mit der blauen Kante zu tun: die Dreiecke waren da noch nicht benachbart



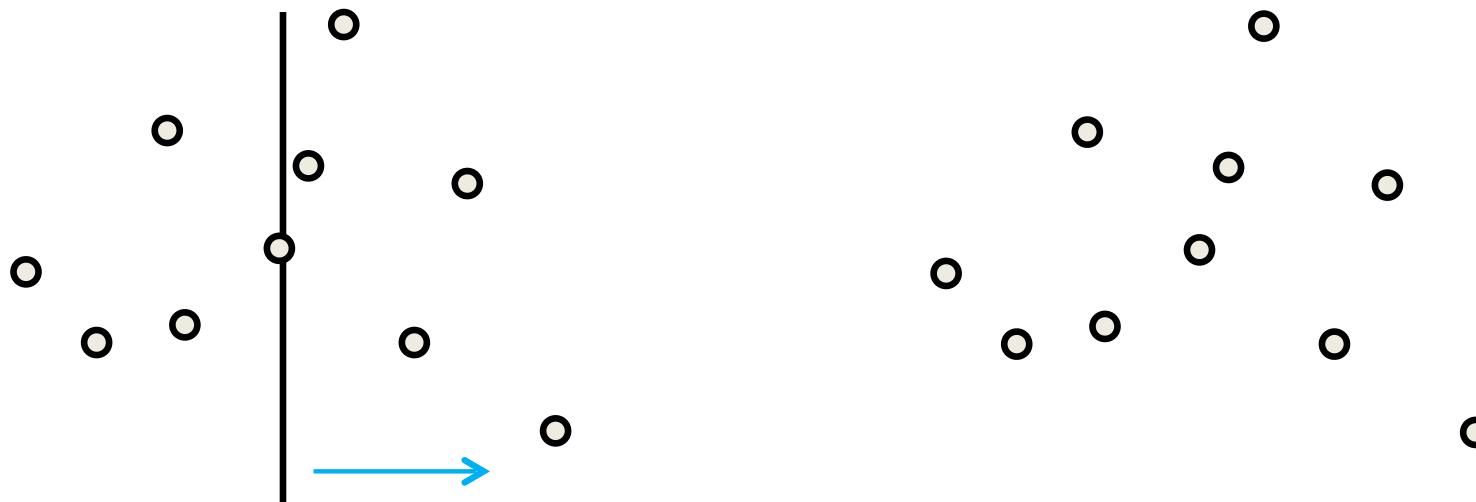
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



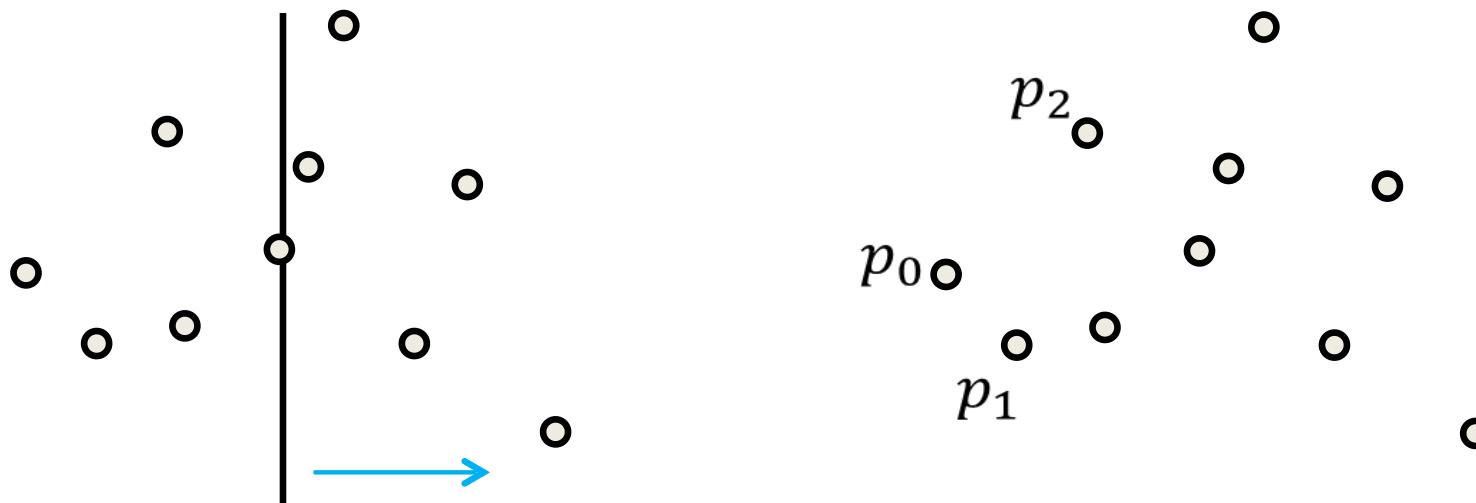
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



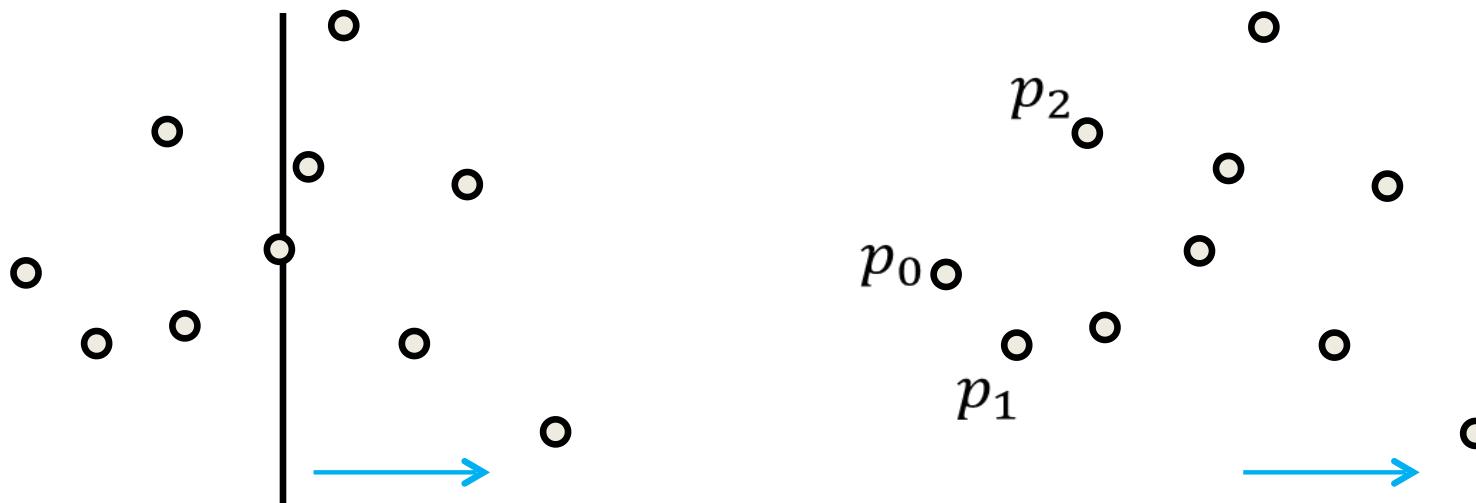
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



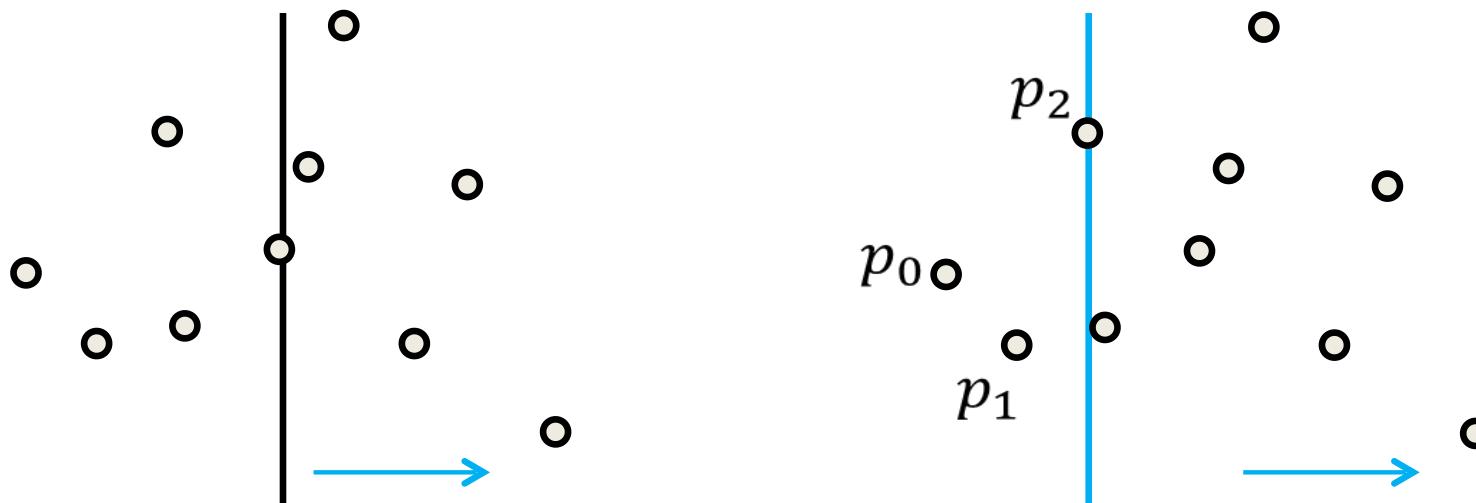
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



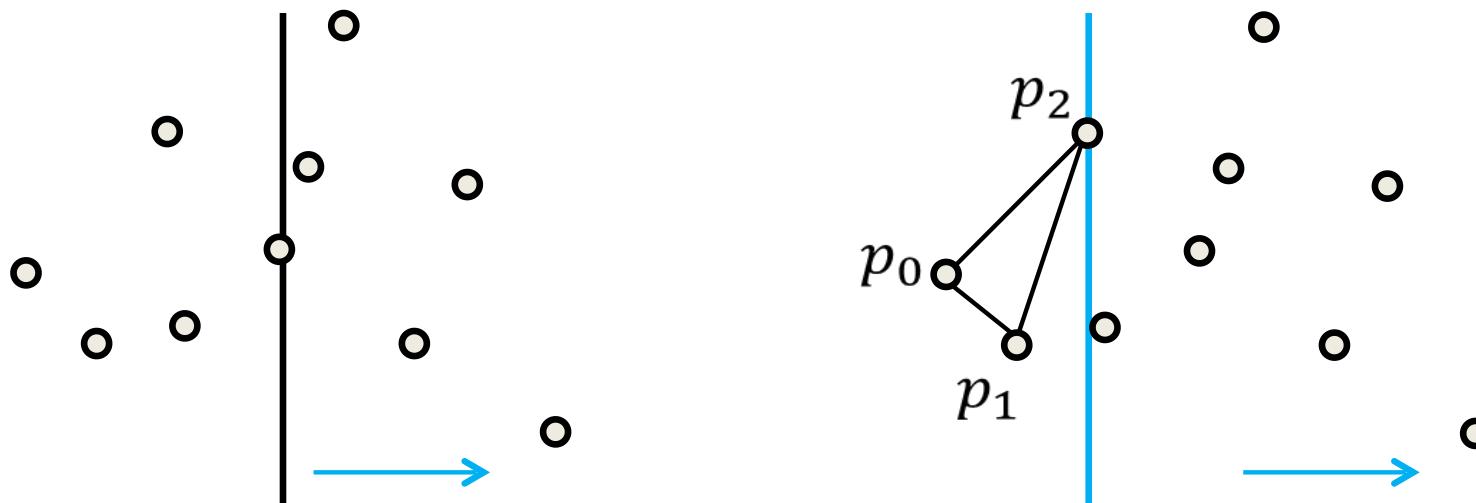
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



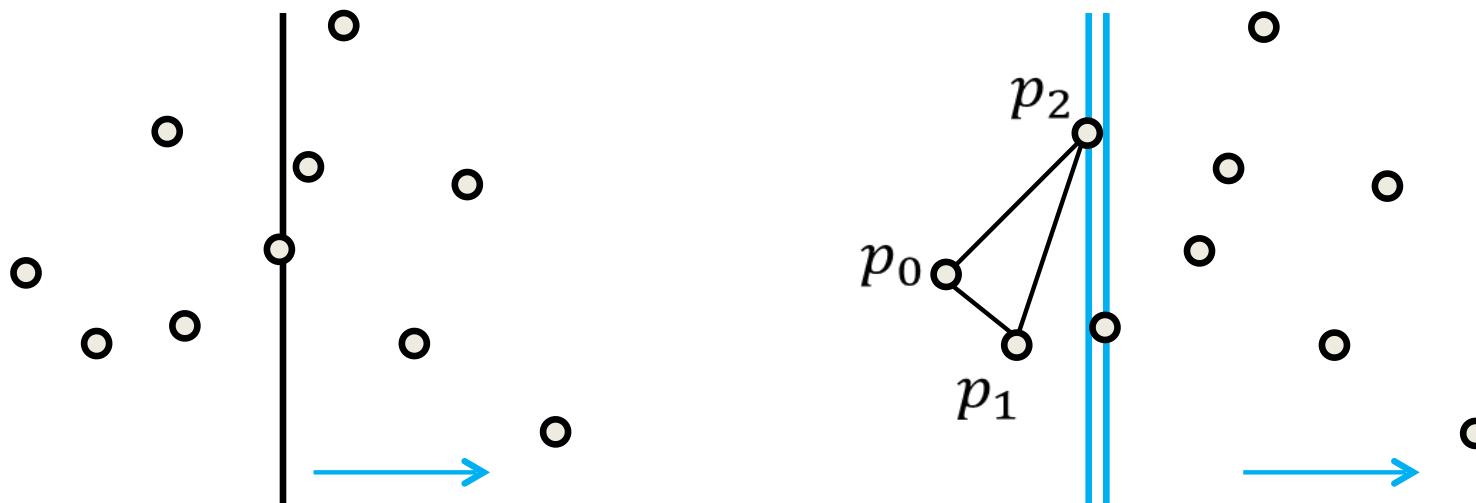
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



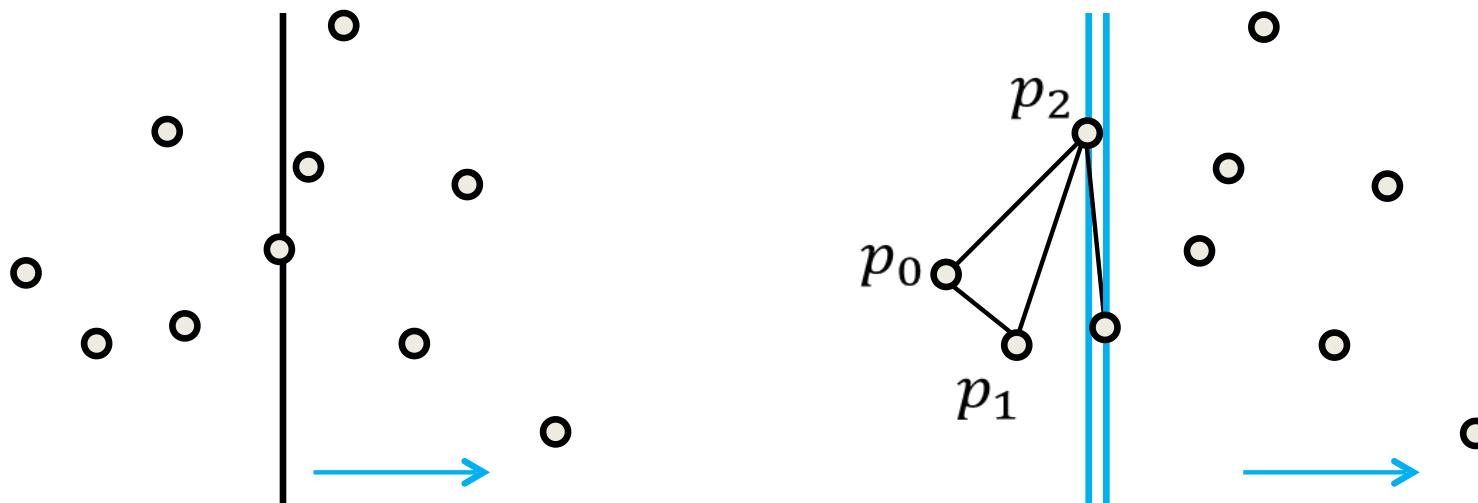
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



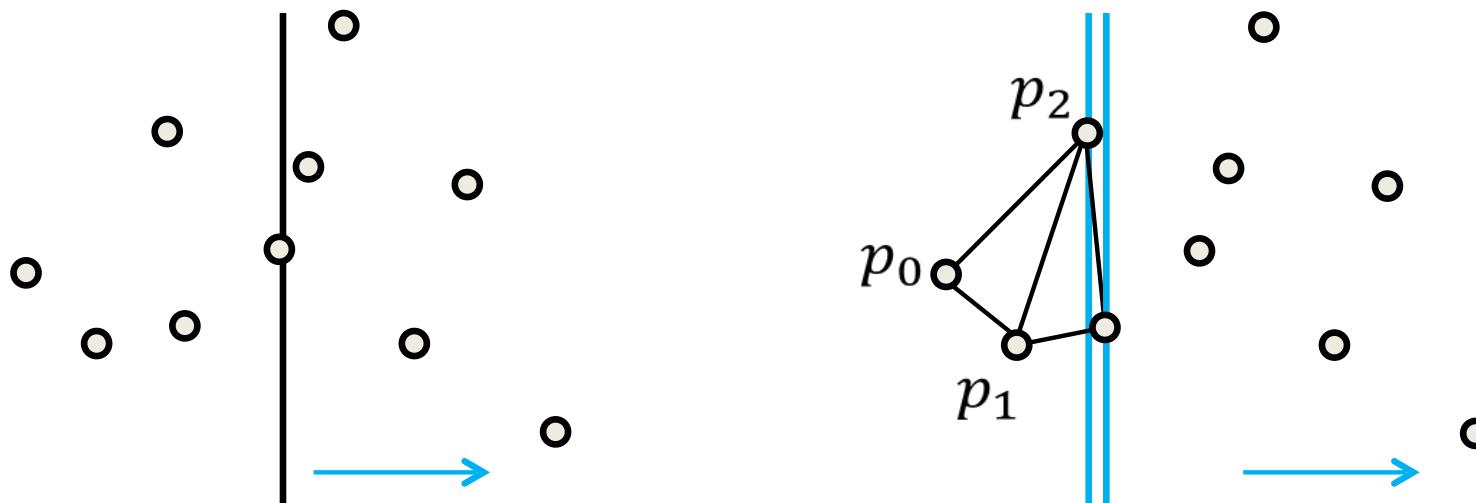
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



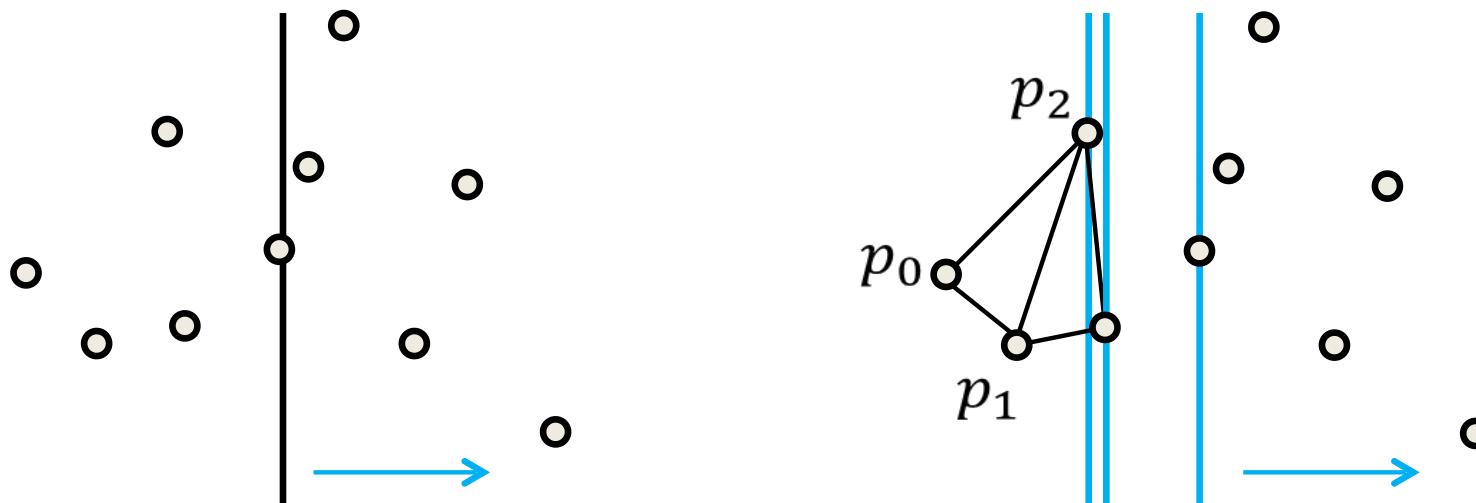
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



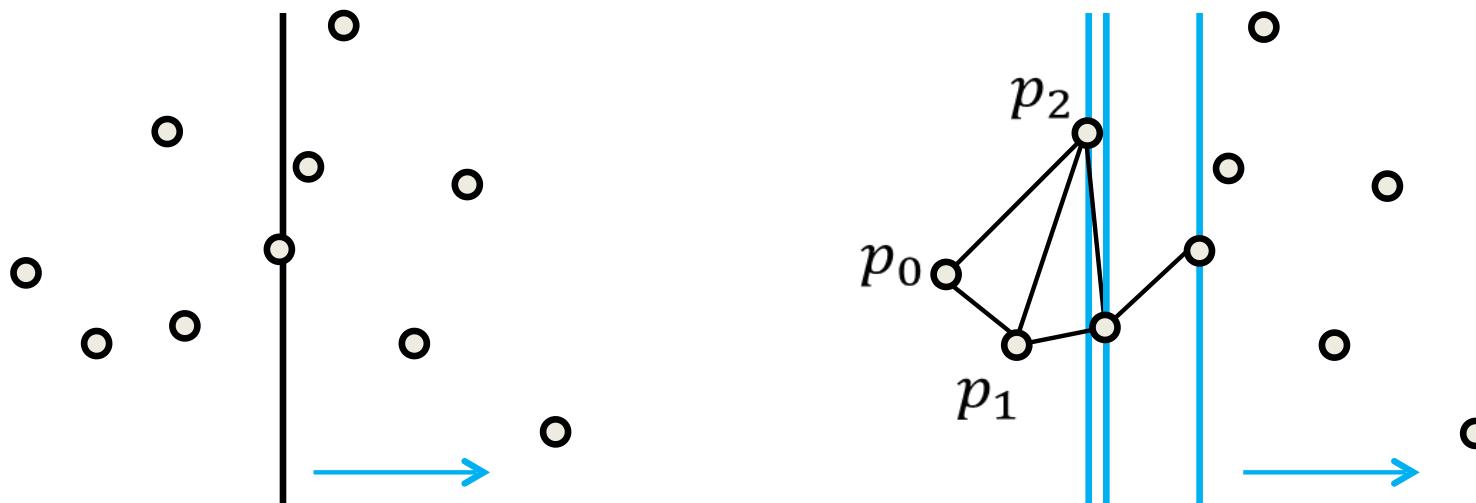
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



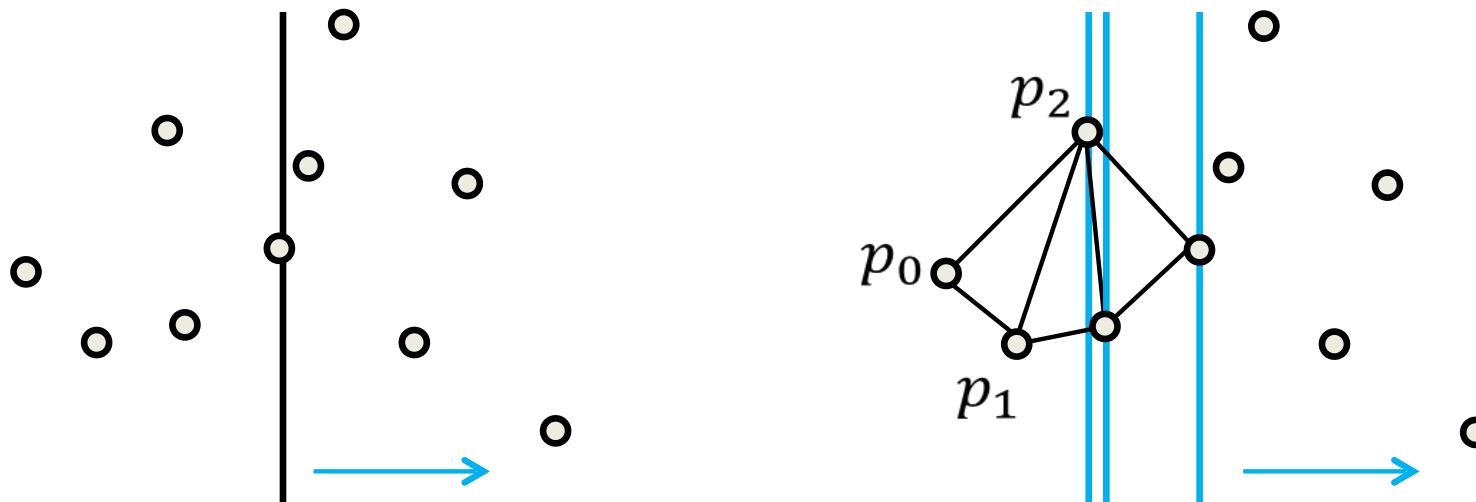
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



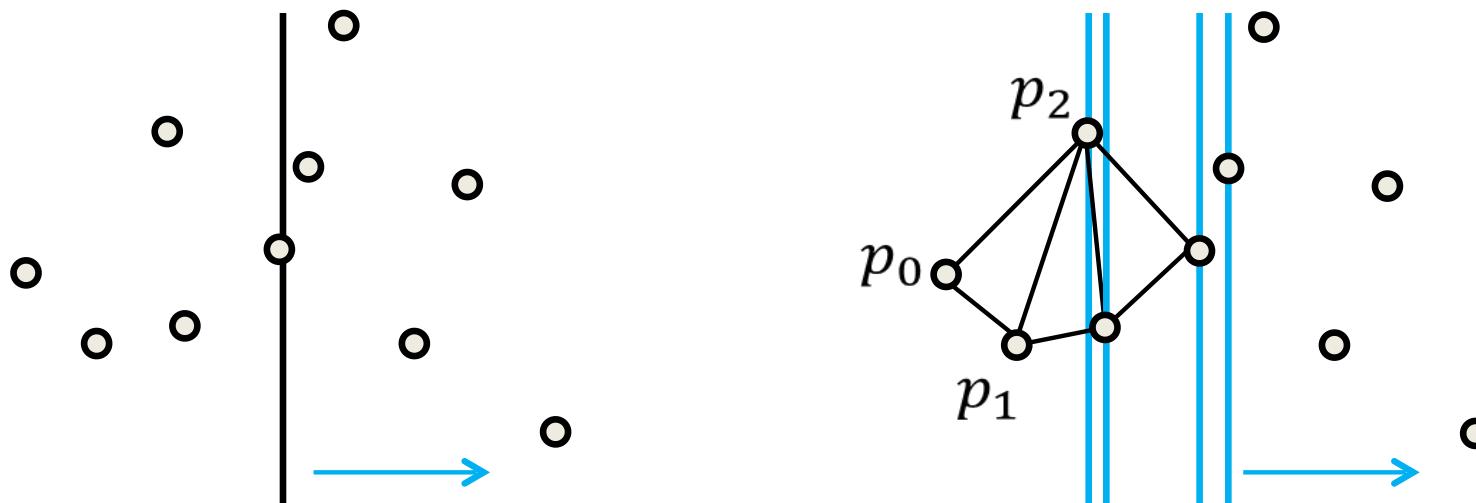
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



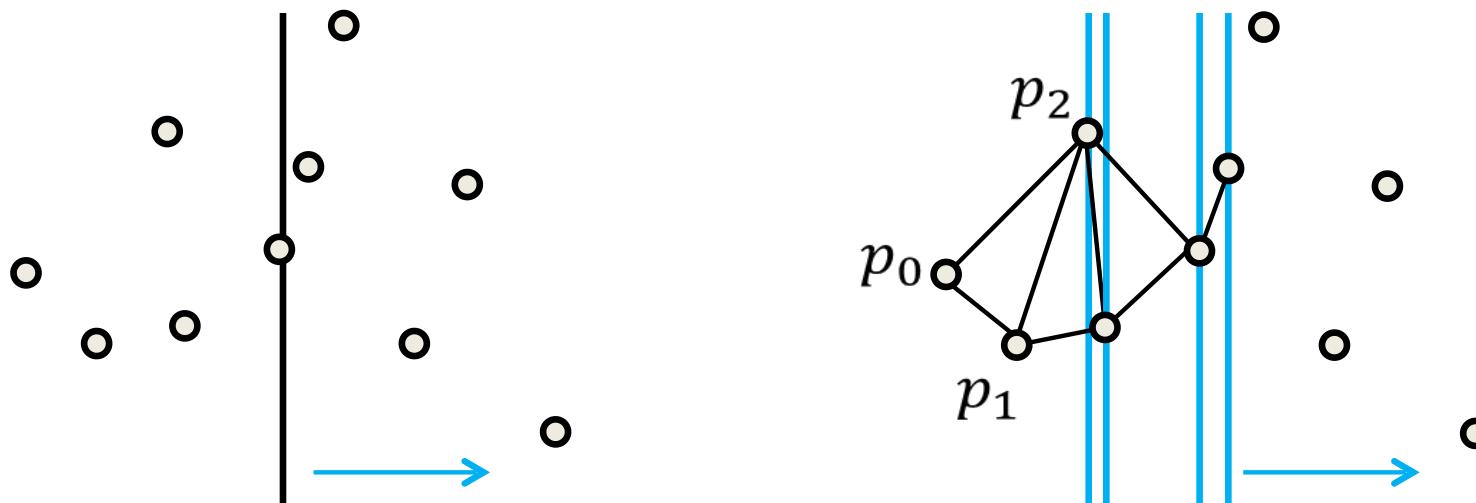
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



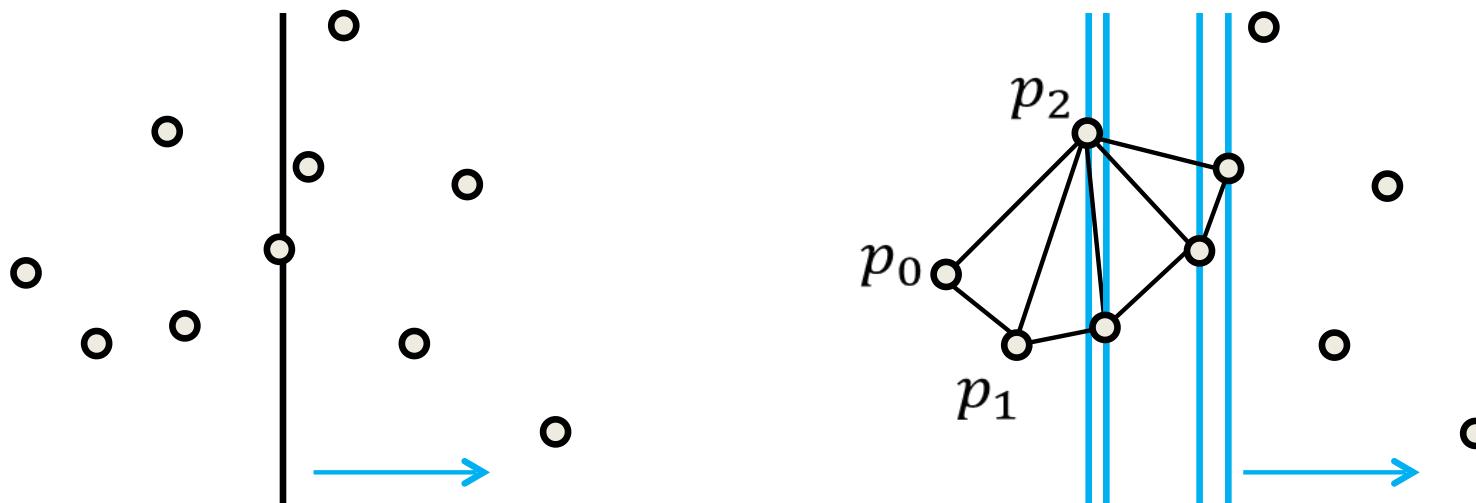
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



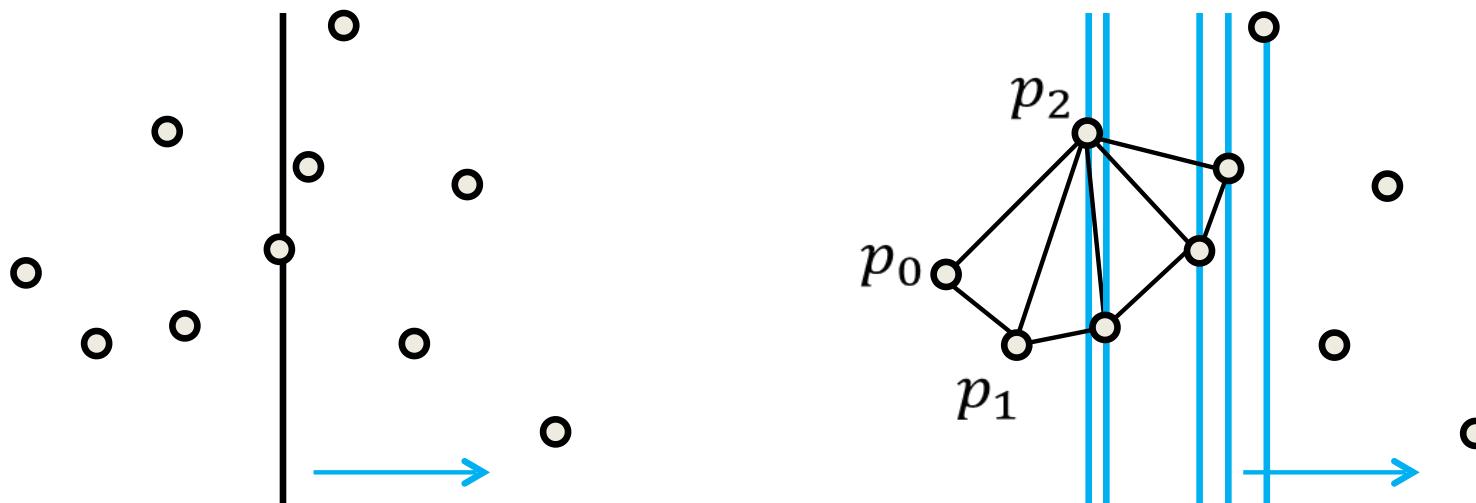
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



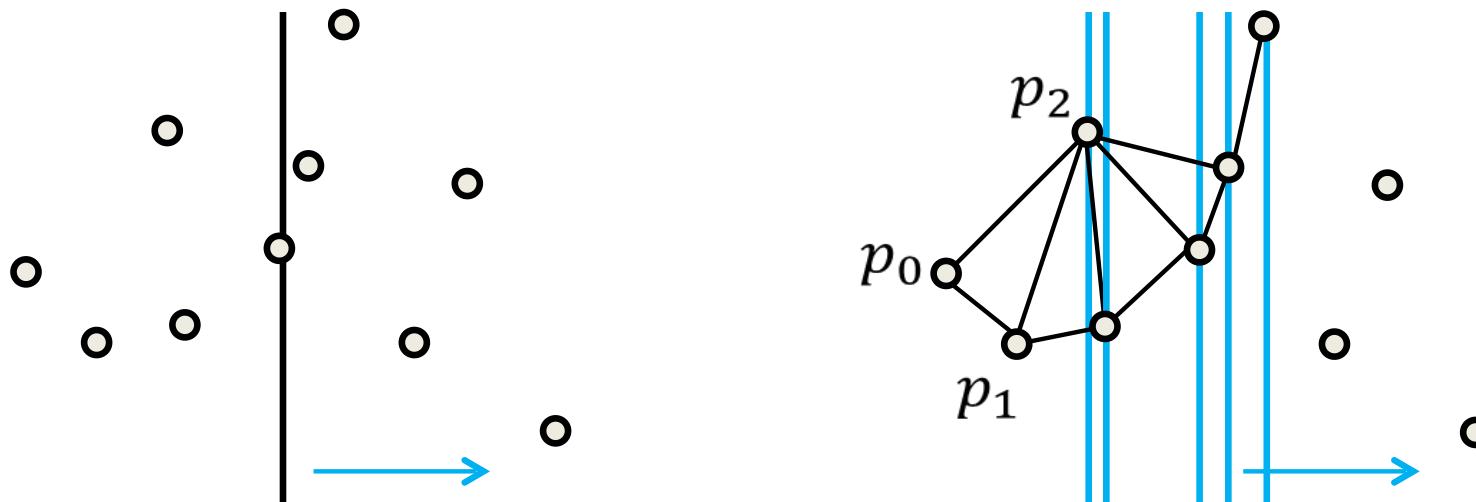
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



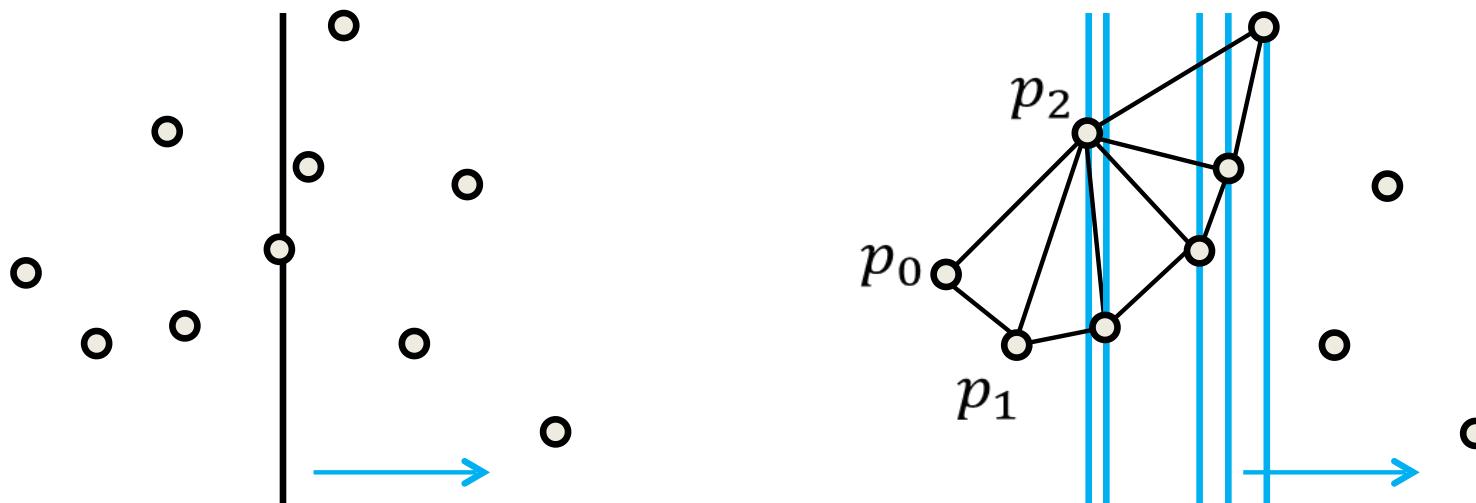
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



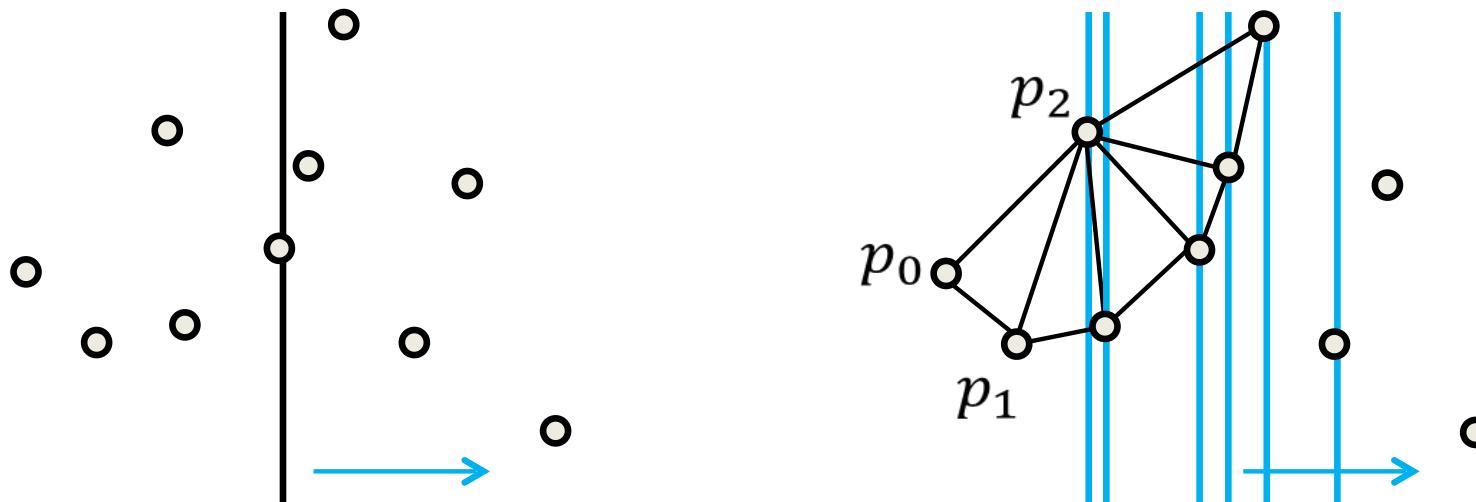
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



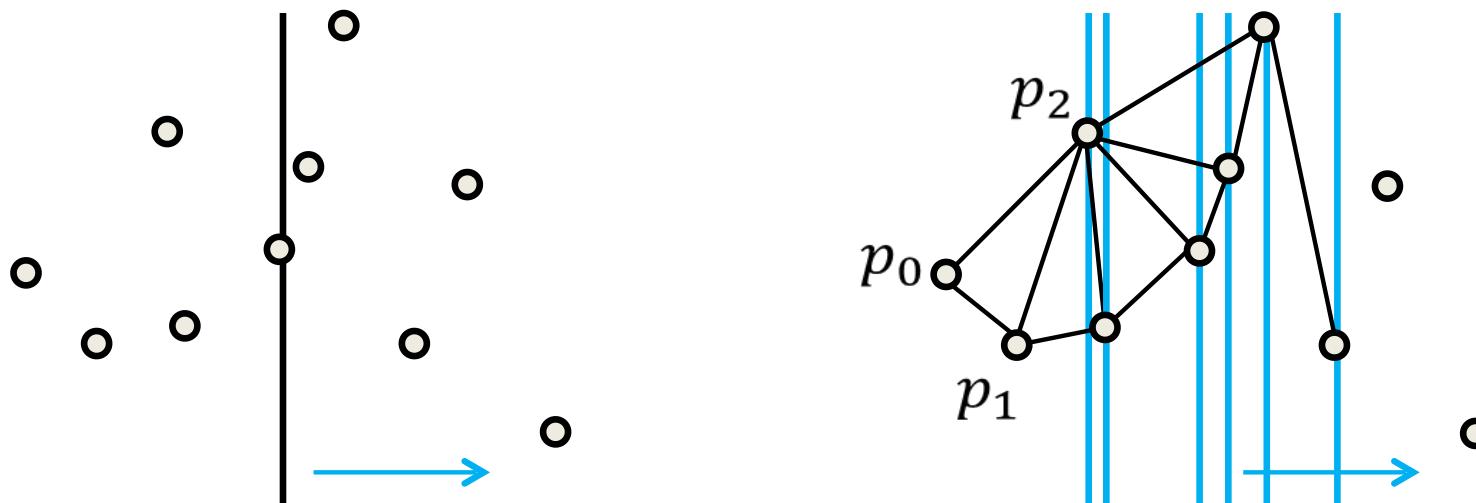
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



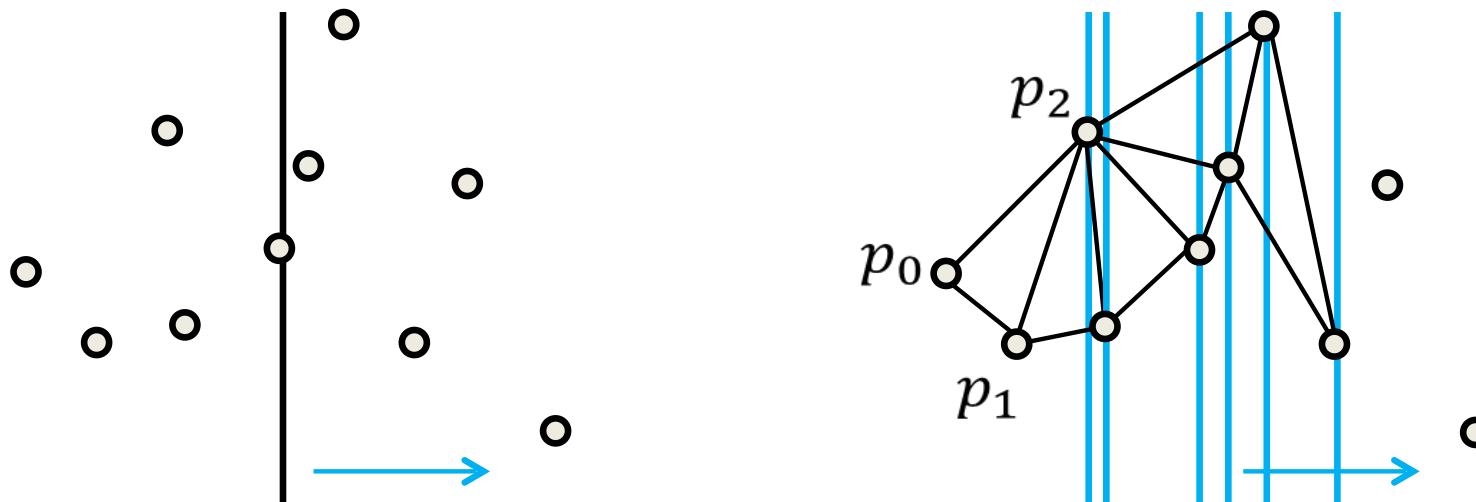
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



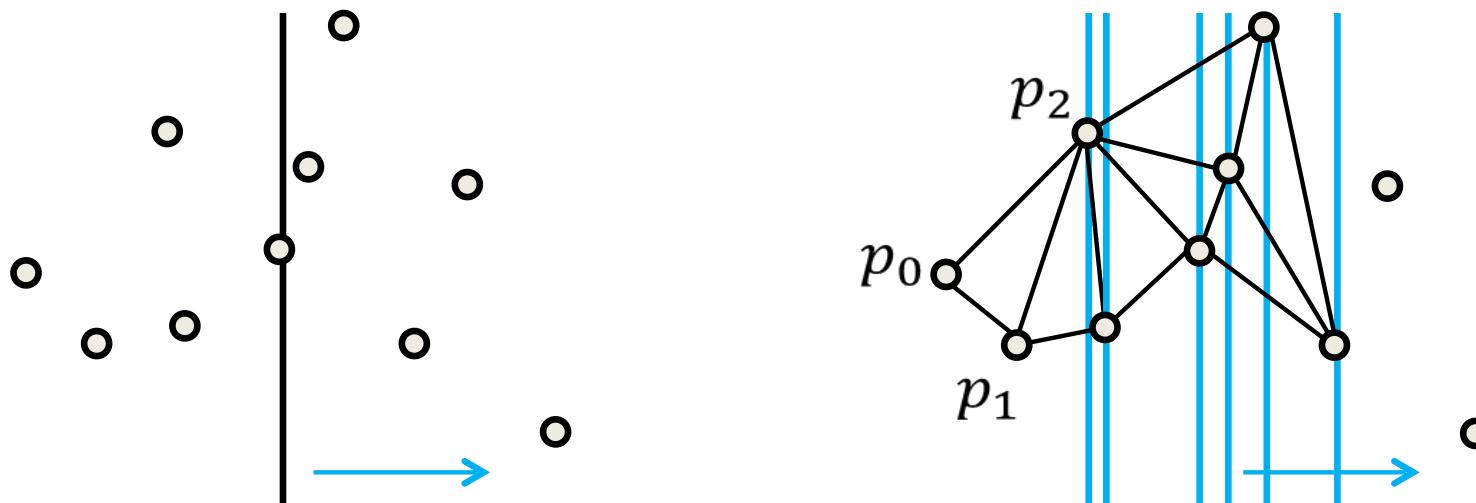
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



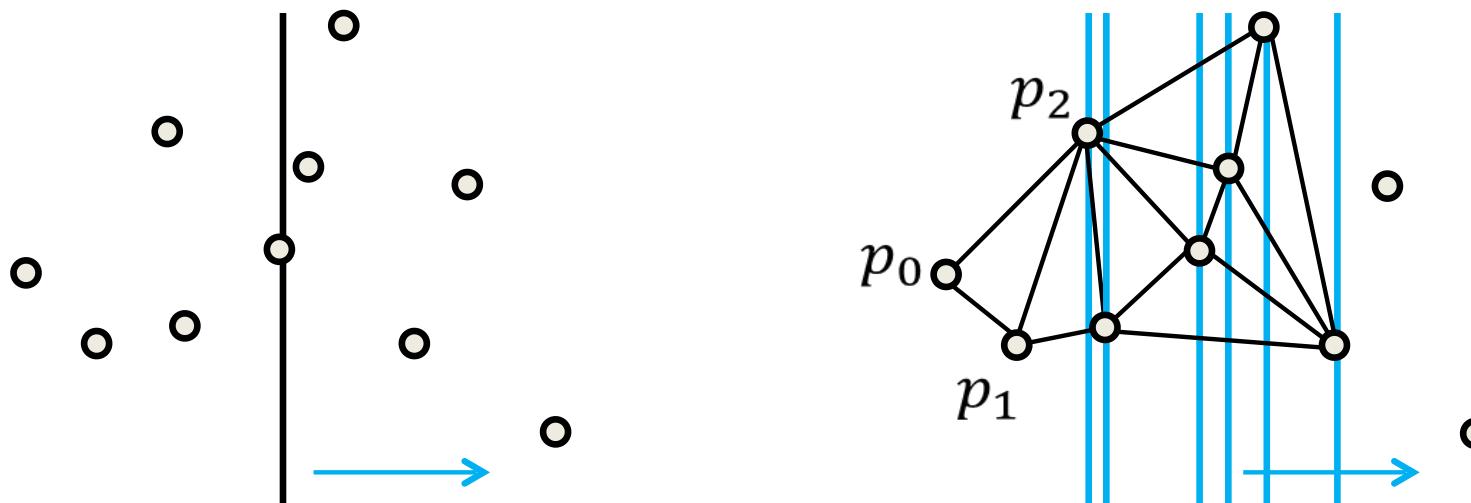
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



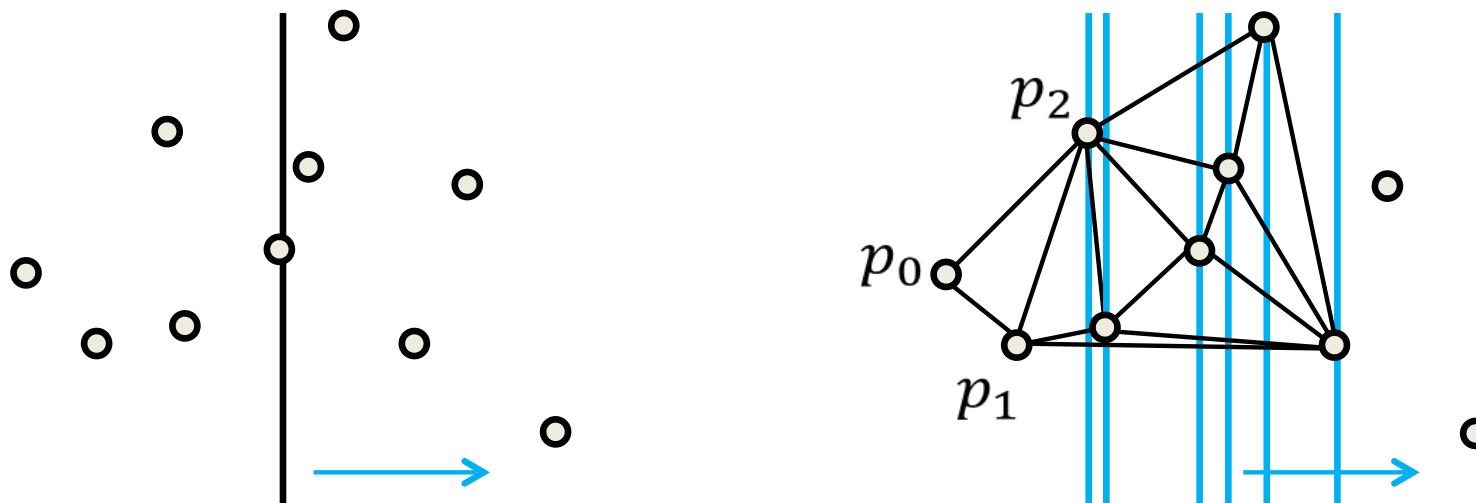
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



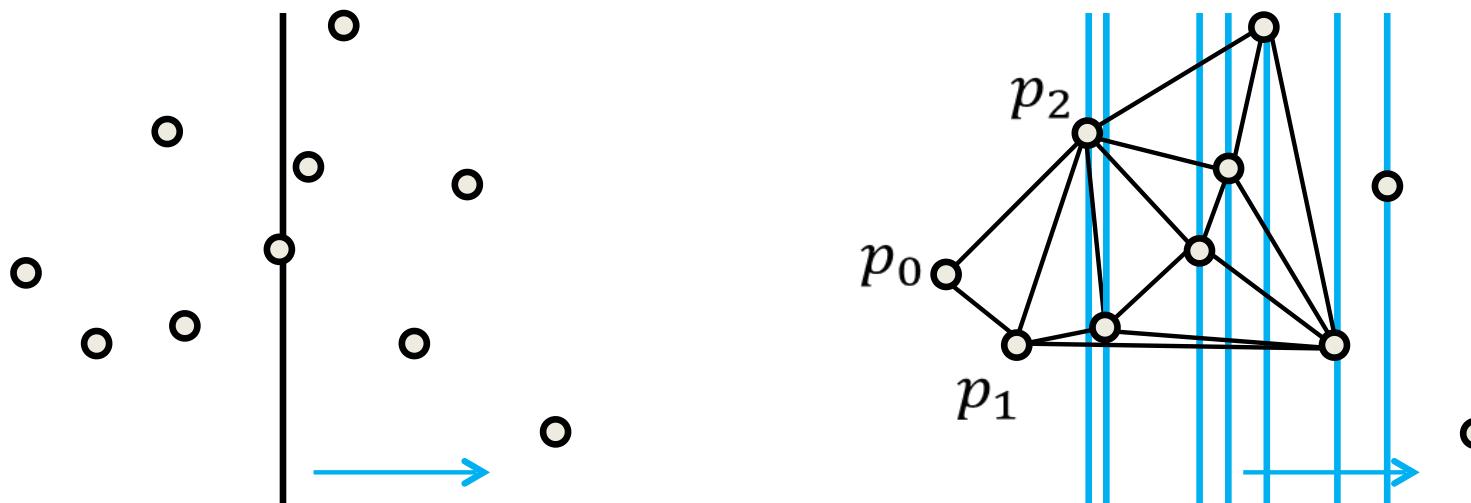
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



Plane-Sweep Algorithmus

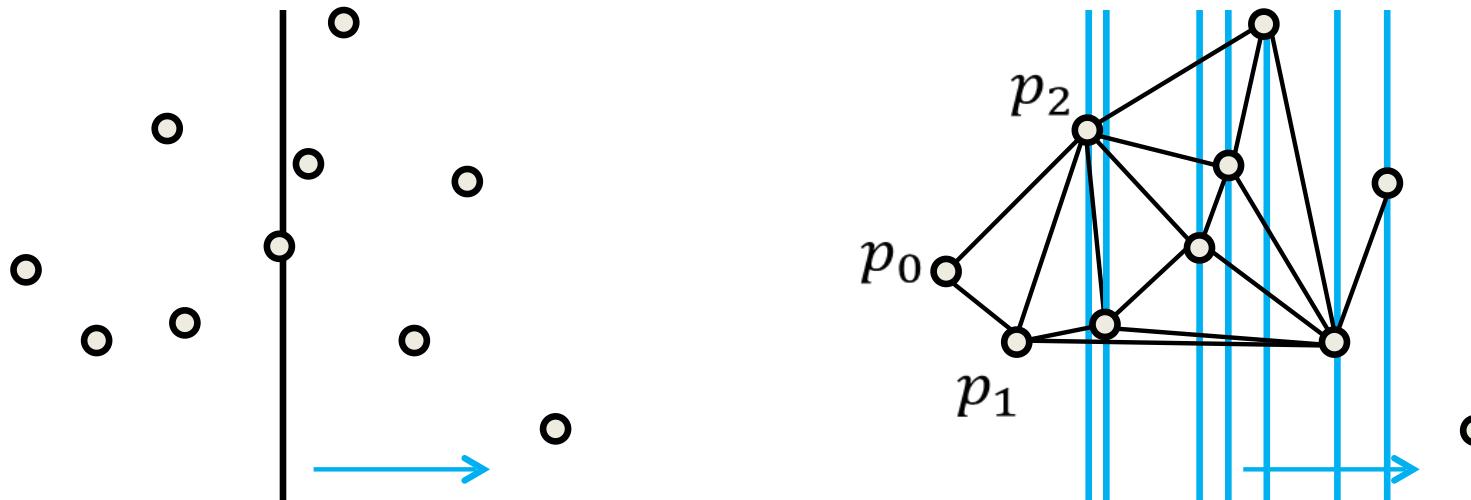
- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



Delaunay Triangulation

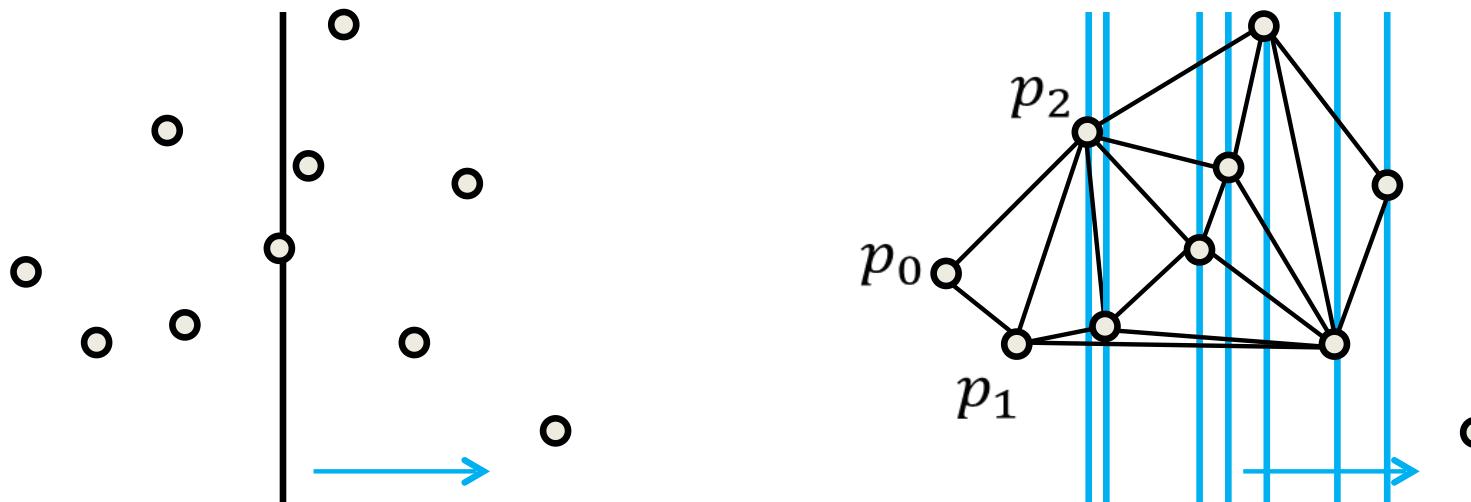
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



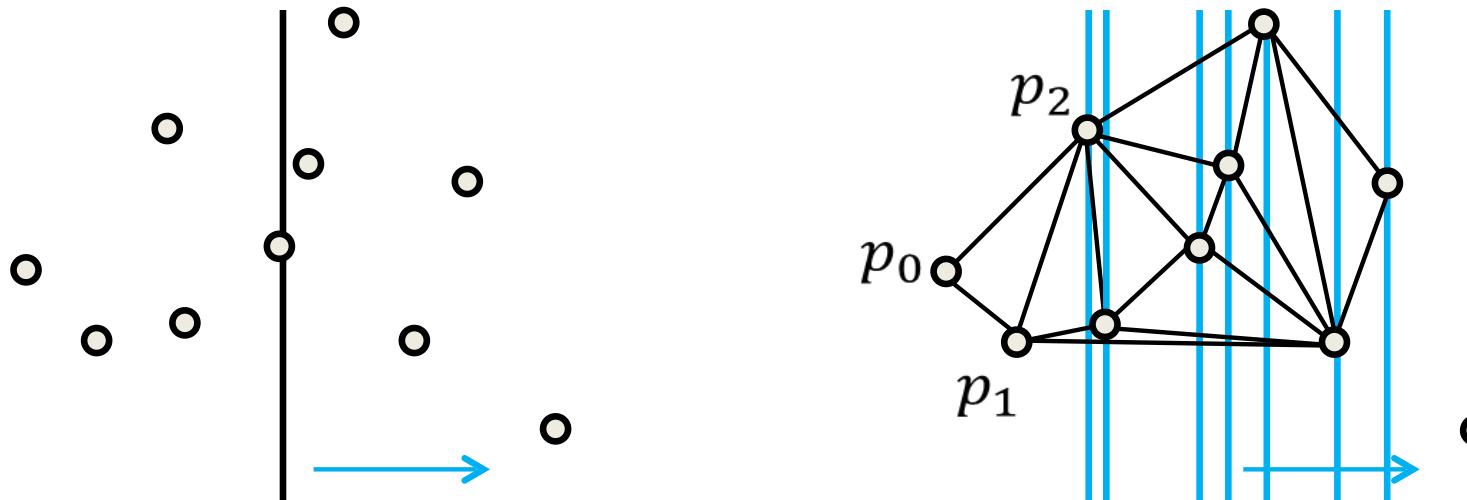
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



Plane-Sweep Algorithmus

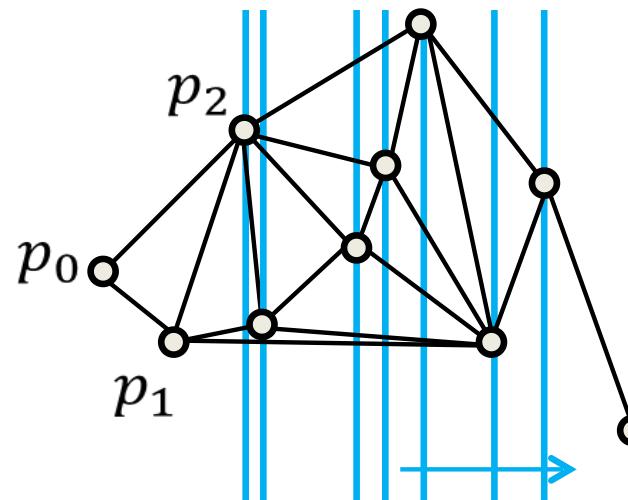
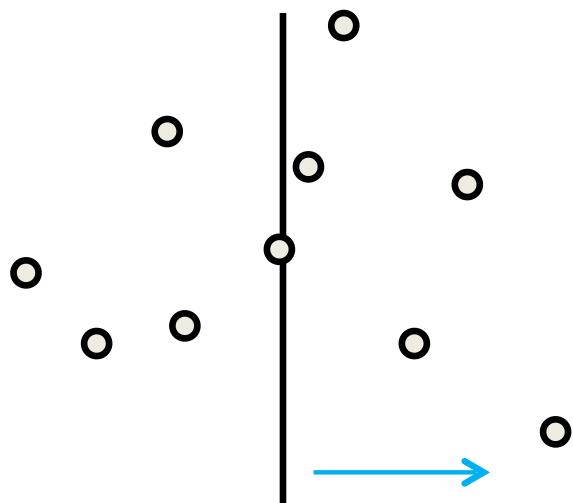
- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



Delaunay Triangulation

Plane-Sweep Algorithmus

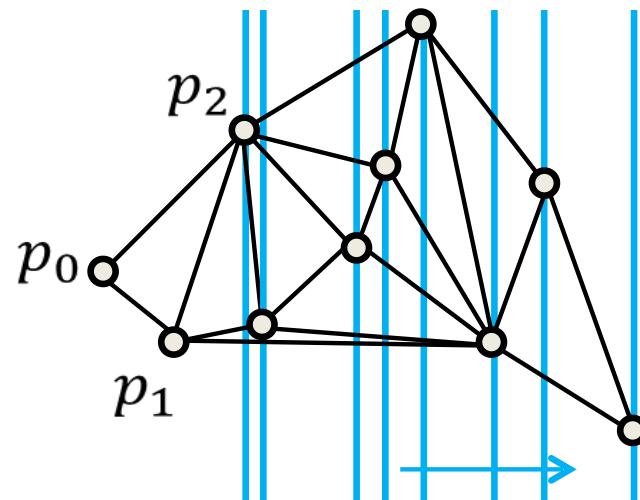
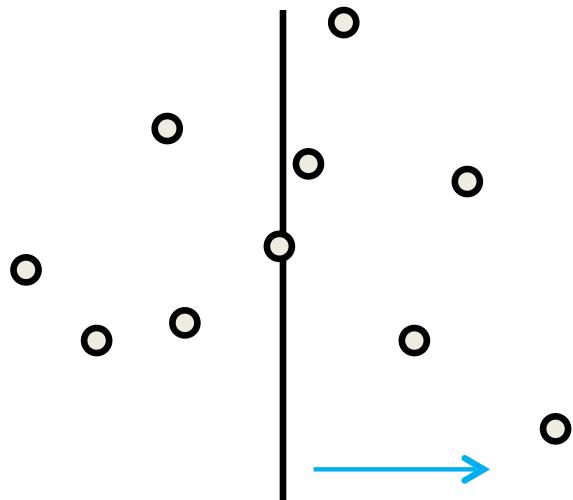
- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



Delaunay Triangulation

Plane-Sweep Algorithmus

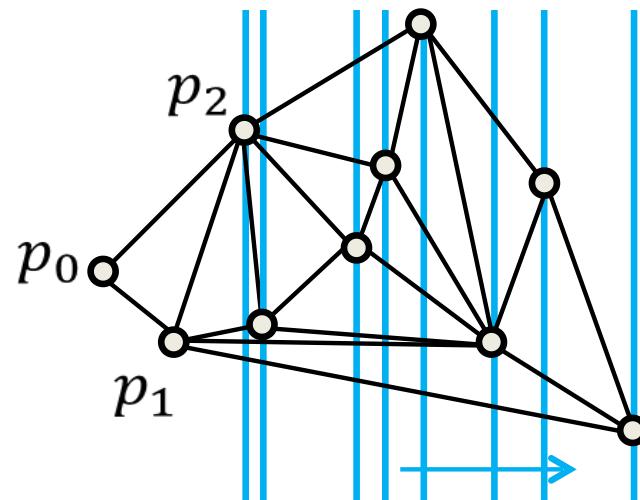
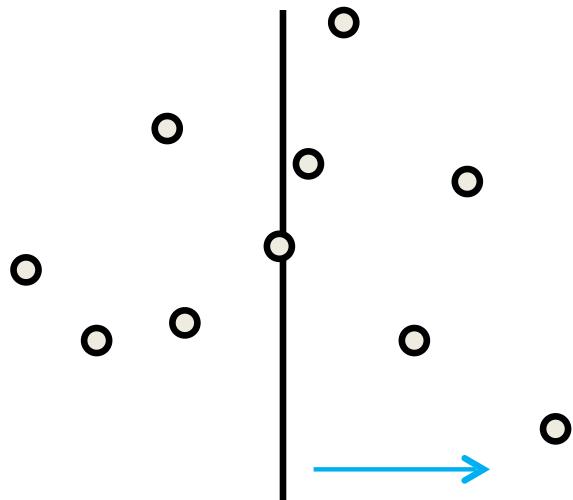
- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



Delaunay Triangulation

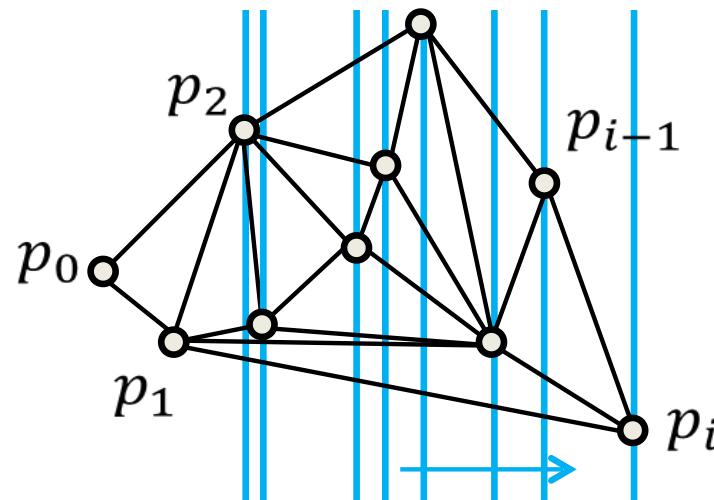
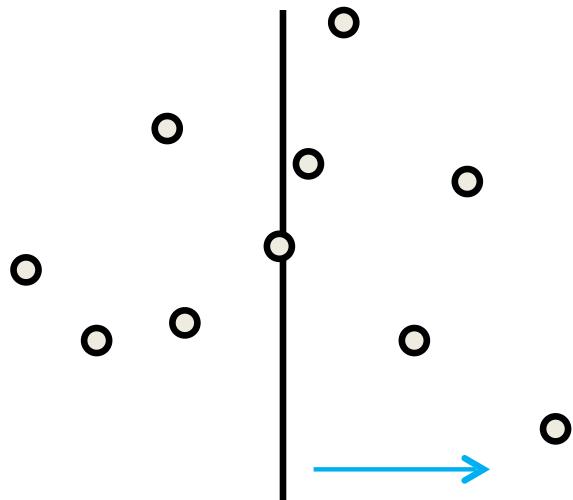
Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme



Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Idee: eine gedachte vertikale Linie verläuft von links nach rechts über \mathbb{R}^2
 - ▶ das Problem wurde für alle Punkte links der Sweep Line gelöst
 - ▶ es wird gerade für Punkte auf oder nahe der Sweep Line bearbeitet
 - ▶ später kümmern wir uns um die Punkte weiter rechts
- ▶ damit reduzieren wir das 2D Problem auf eine Reihe 1D Probleme

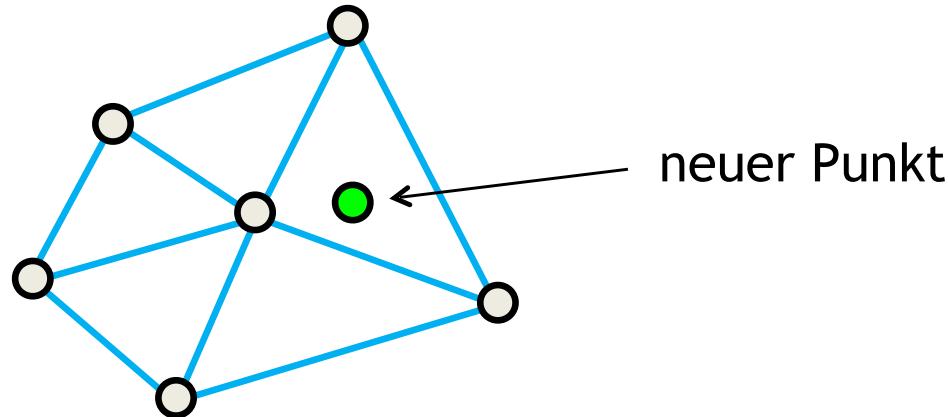


Plane-Sweep Algorithmus

- ▶ ... zur Berechnung einer initialen Triangulation
- ▶ Algorithmus
 - ▶ sortiere Punkte von links nach rechts
 - ▶ erzeuge erstes Dreieck aus den ersten 3 Vertizes
 - ▶ for i = 4 to n
 - ▶ verwende den zuletzt eingefügten Punkt p_{i-1} als Startpunkt
 - ▶ laufe gegen Uhrzeigersinn um die konvexe Hülle der Triangulation bis zum Tangentenpunkt und füge dabei Kanten zu p_i ein
 - ▶ laufe im Uhrzeigersinn um die konvexe Hülle der Triangulation bis zum Tangentenpunkt und füge dabei Kanten zu p_i ein
 - ▶ aktualisiere konvexe Hülle

Bowyer-Watson Algorithmus

- Idee: inkrementeller Aufbau der Triangulation durch Einfügen der Punkte nacheinander
- Vorteil: die Triangulation kann nachträglich um Punkte erweitert werden



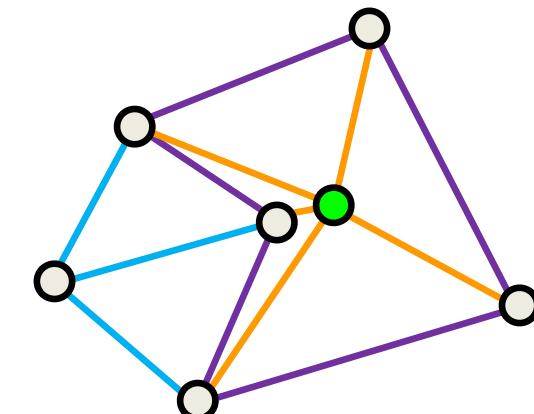
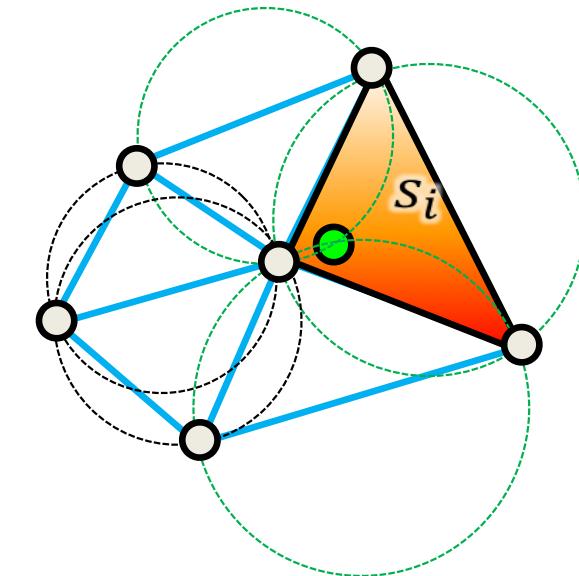
[D.F. Watson. Computing the n-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes. The Computer Journal, 24(2):167-172, 1981]

[A. Bowyer. Computing Dirichlet Tessellations. The Computer Journal, 24(2):162-166, 1981]

Voronoi Diagramme und Delaunay Triangulation

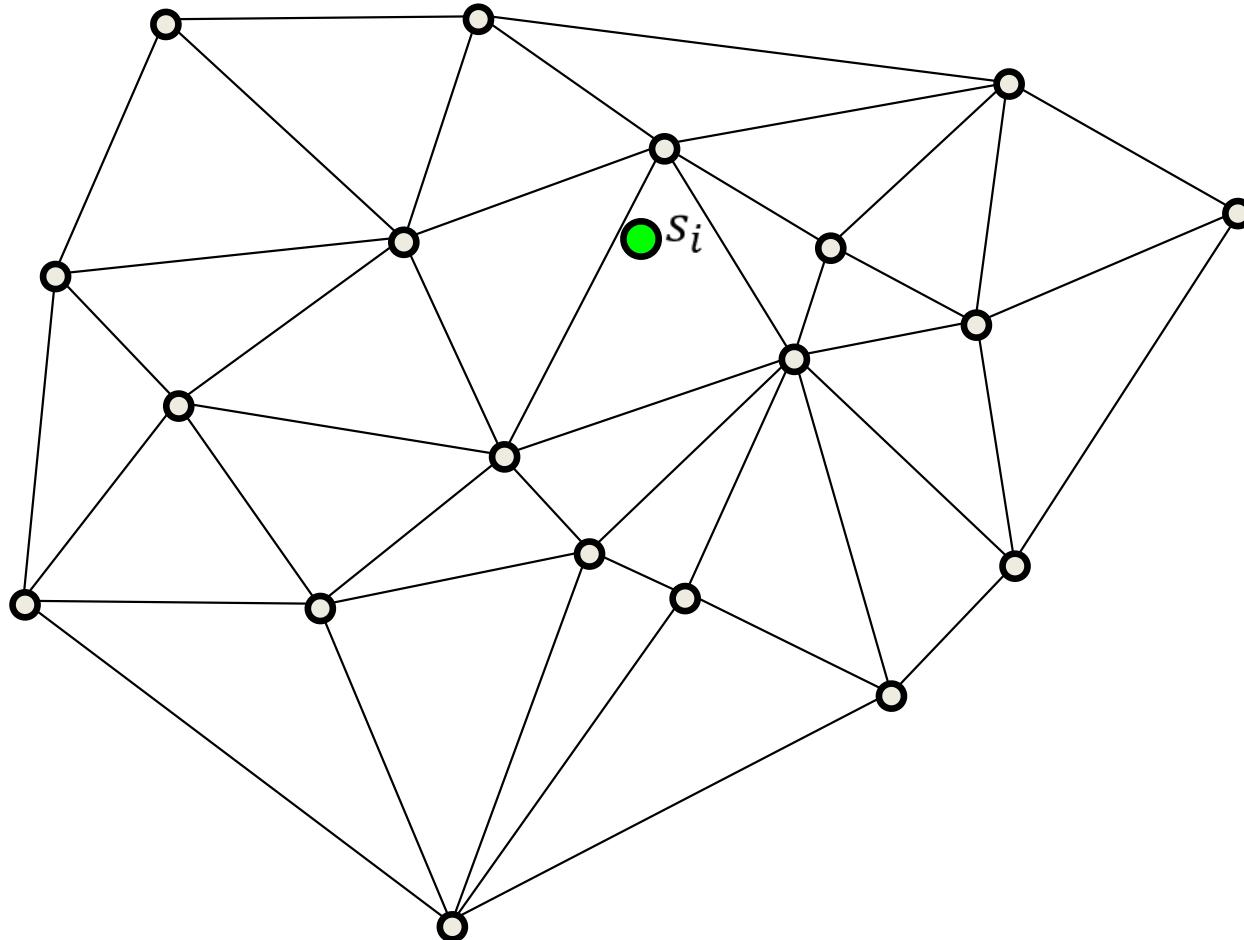
Bowyer-Watson Algorithmus

- ▶ beginne mit einer initialen (Delaunay-) Triangulation, die alle Punkte enthält (z.B. 2 Dreiecke bestimmt durch die AABB)
- ▶ füge Punkte s_i nacheinander ein
 - ▶ entferne alle Dreiecke, deren Umkreis s_i enthält
(siehe Algorithmus bzgl. „Super Triangle“)
 - ▶ dadurch entsteht ein Loch in der Triangulation, das man sternförmig triangulieren kann
 - ▶ trianguliere das entstandene Loch neu:
verbinde s_i mit allen Vertizes am Rand



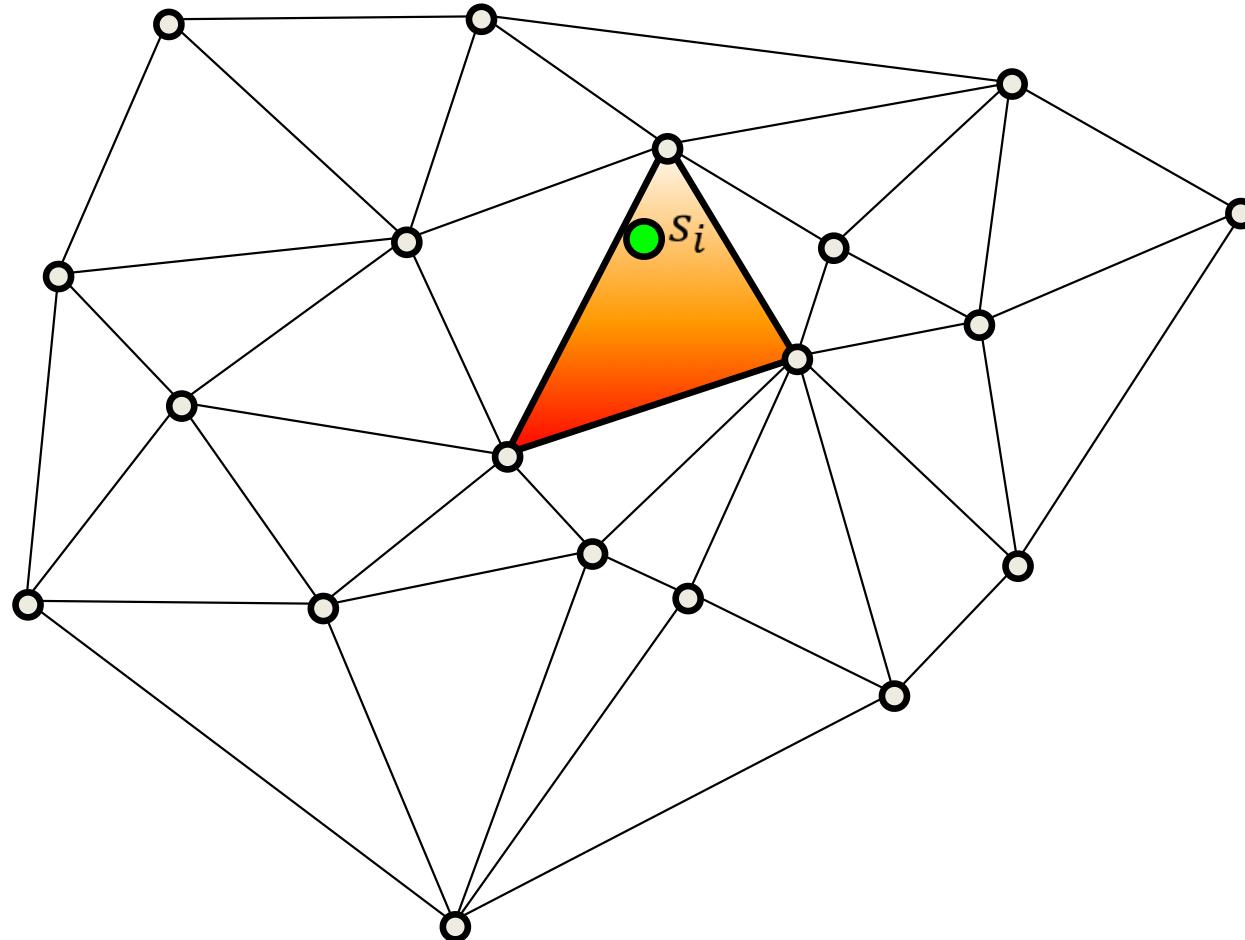
Bowyer-Watson Algorithmus

Beispiel



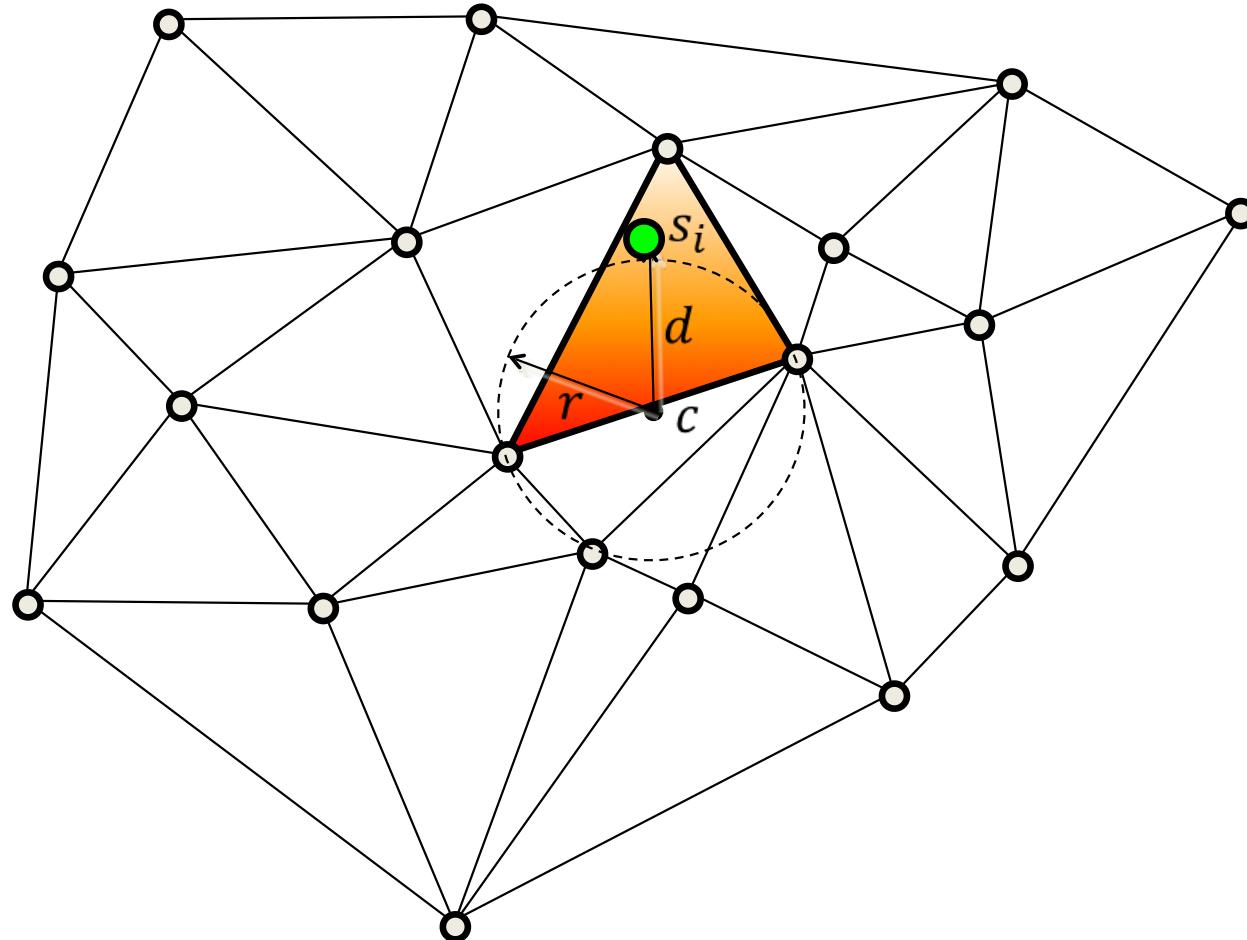
Bowyer-Watson Algorithmus

Beispiel



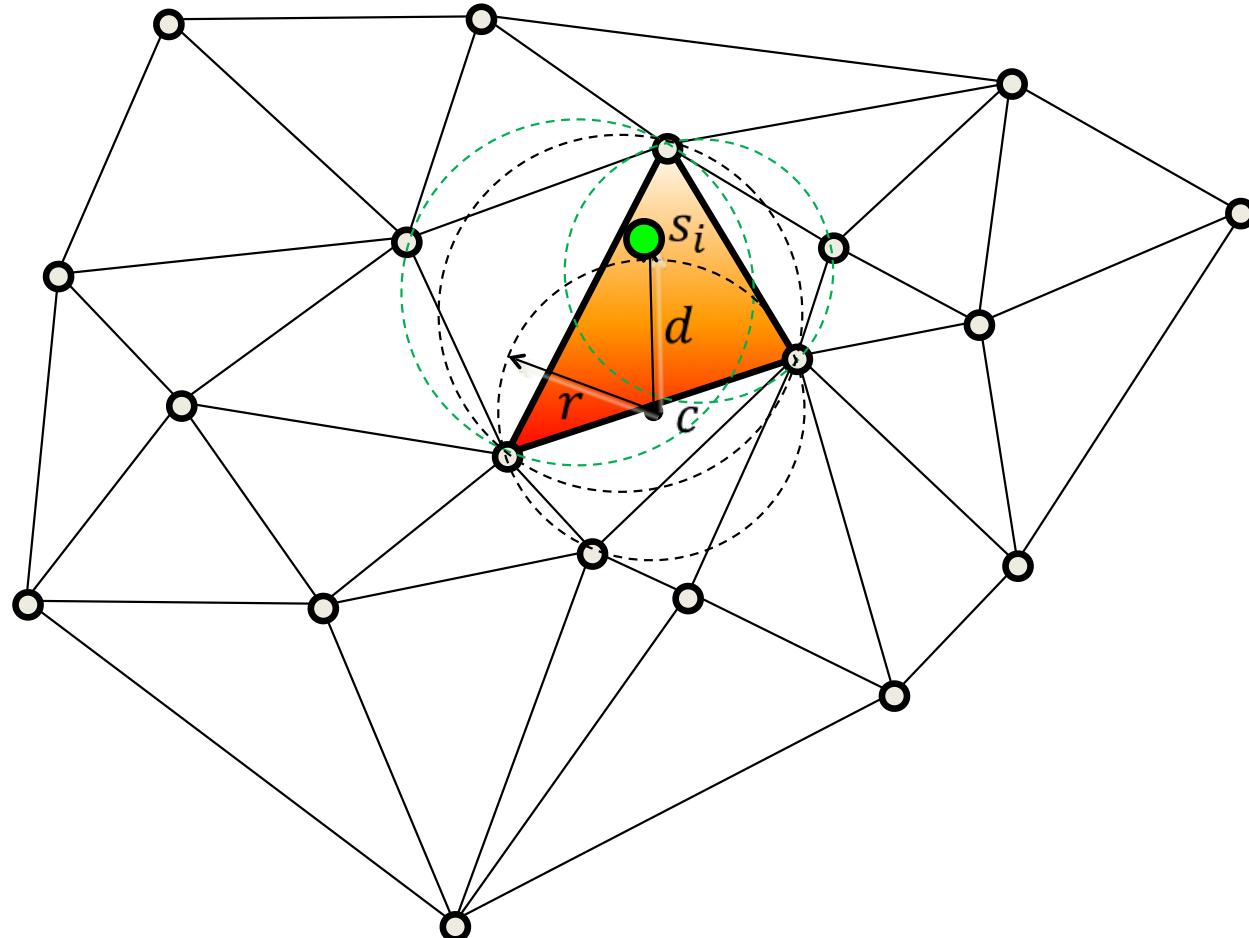
Bowyer-Watson Algorithmus

Beispiel



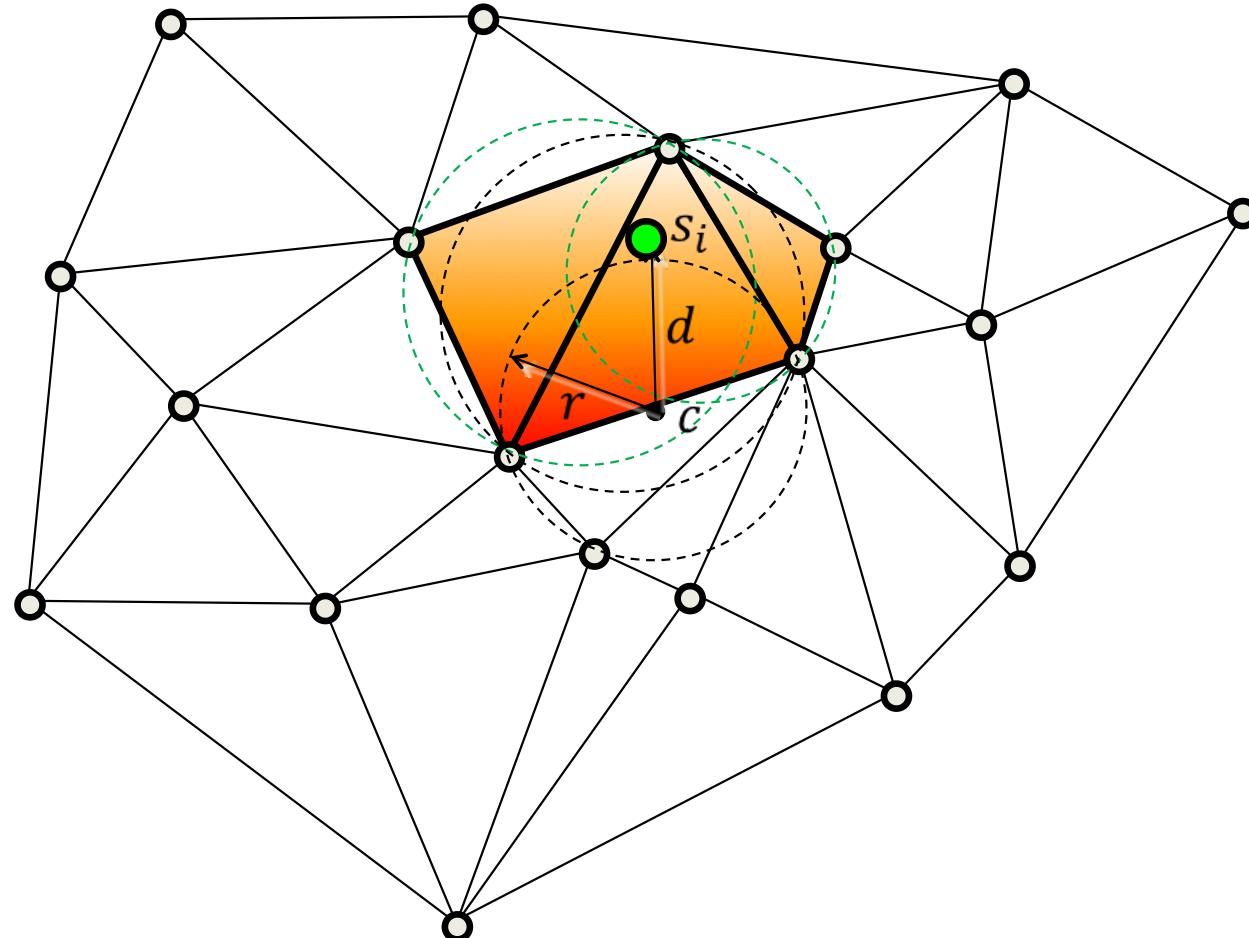
Bowyer-Watson Algorithmus

Beispiel



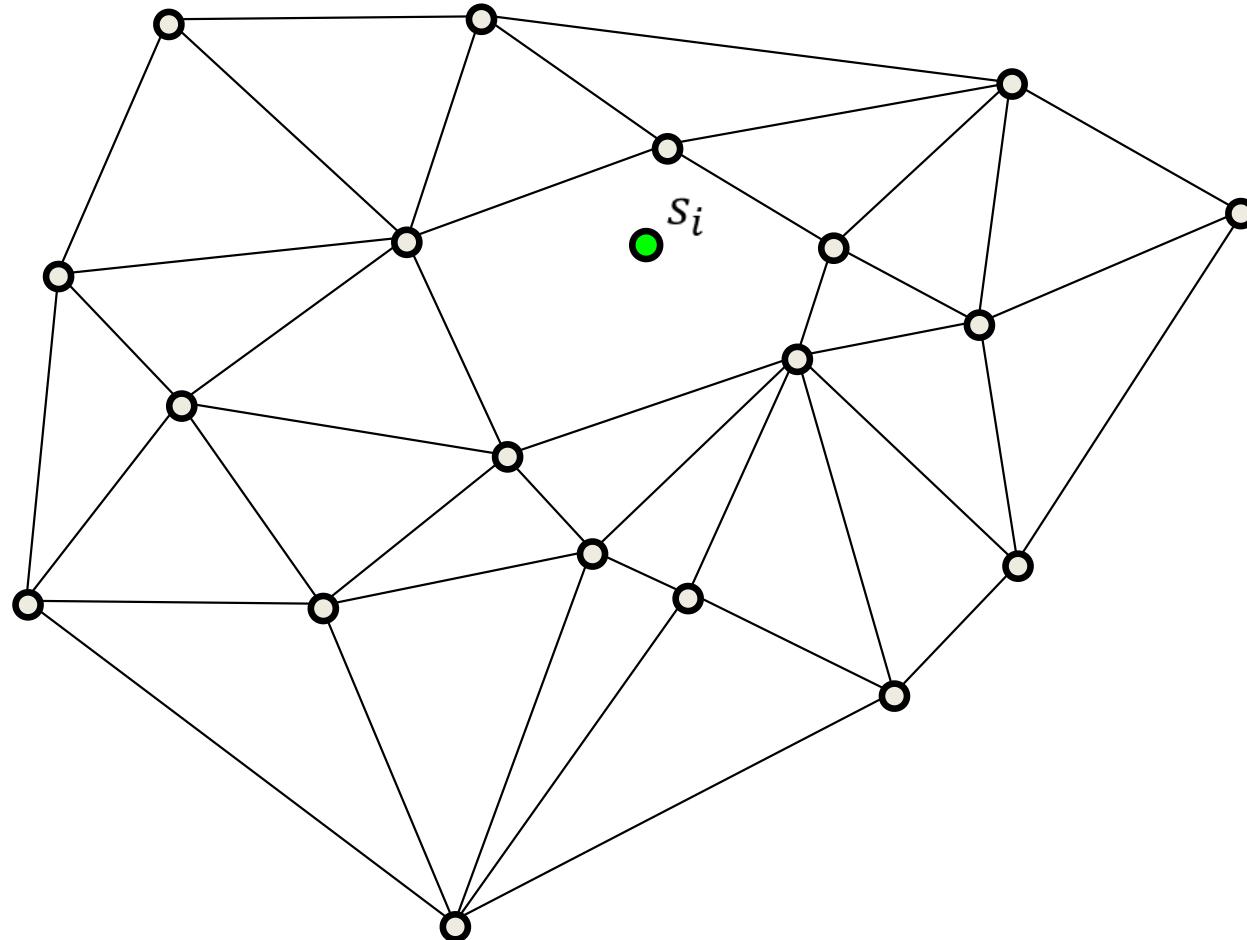
Bowyer-Watson Algorithmus

Beispiel



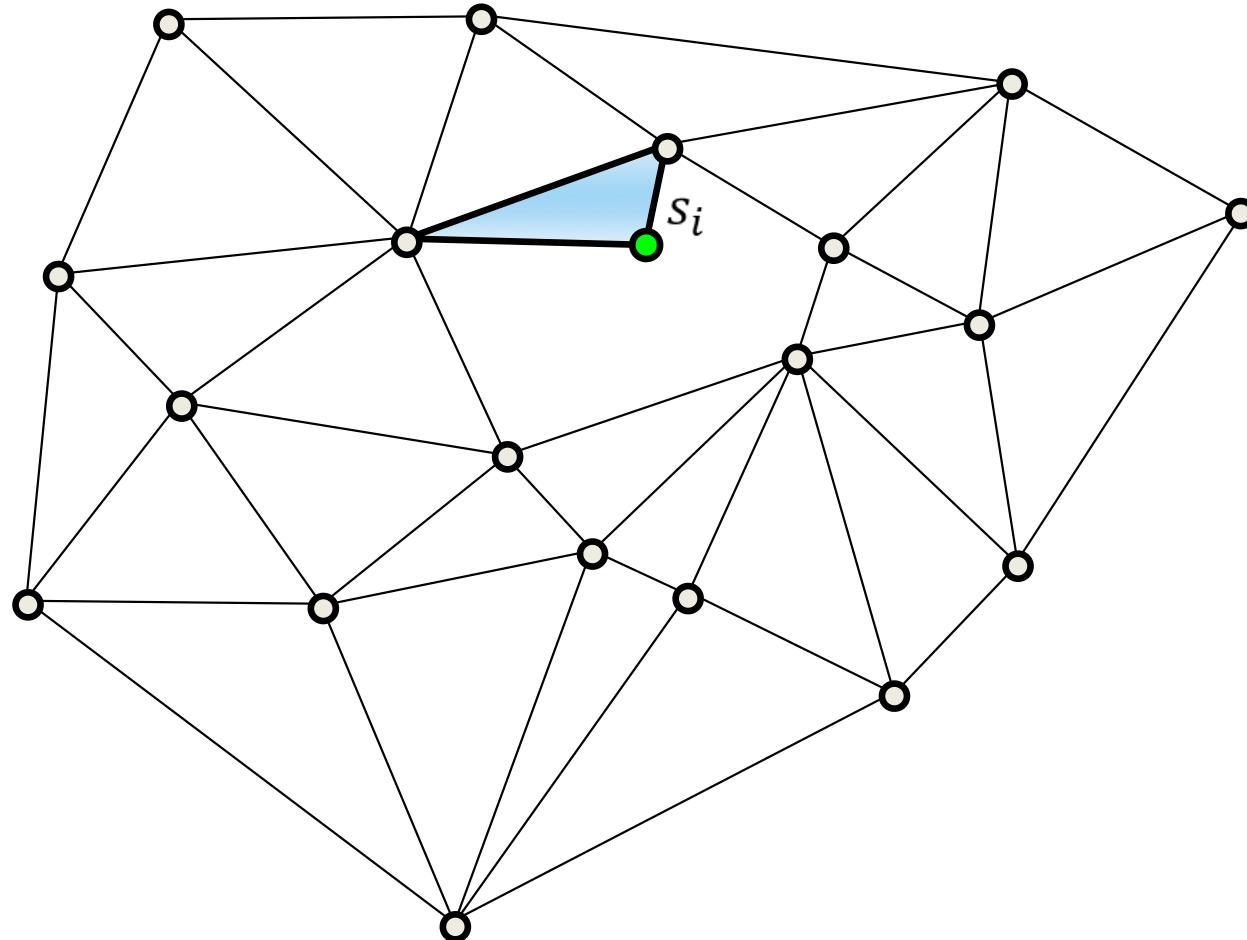
Bowyer-Watson Algorithmus

Beispiel



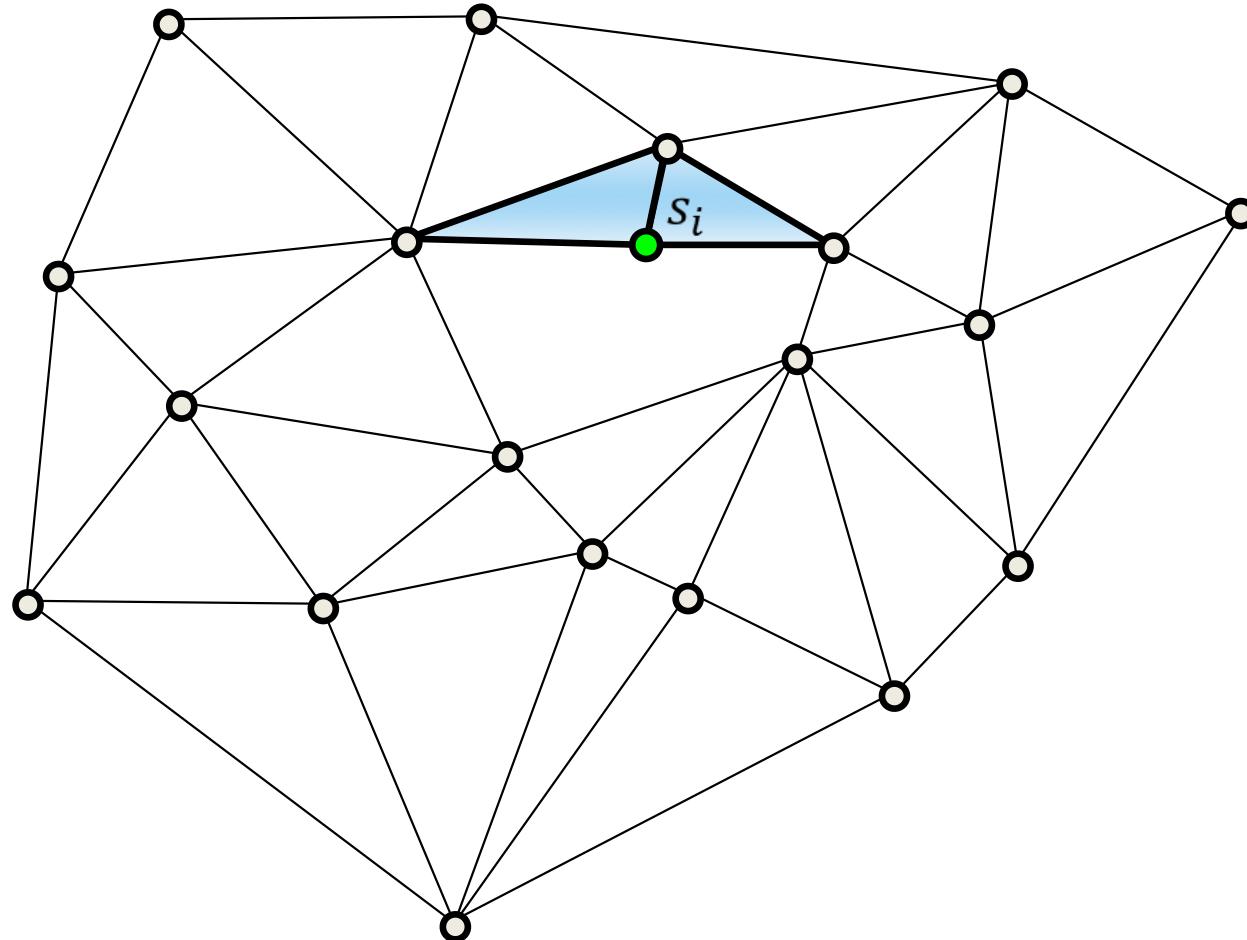
Bowyer-Watson Algorithmus

Beispiel



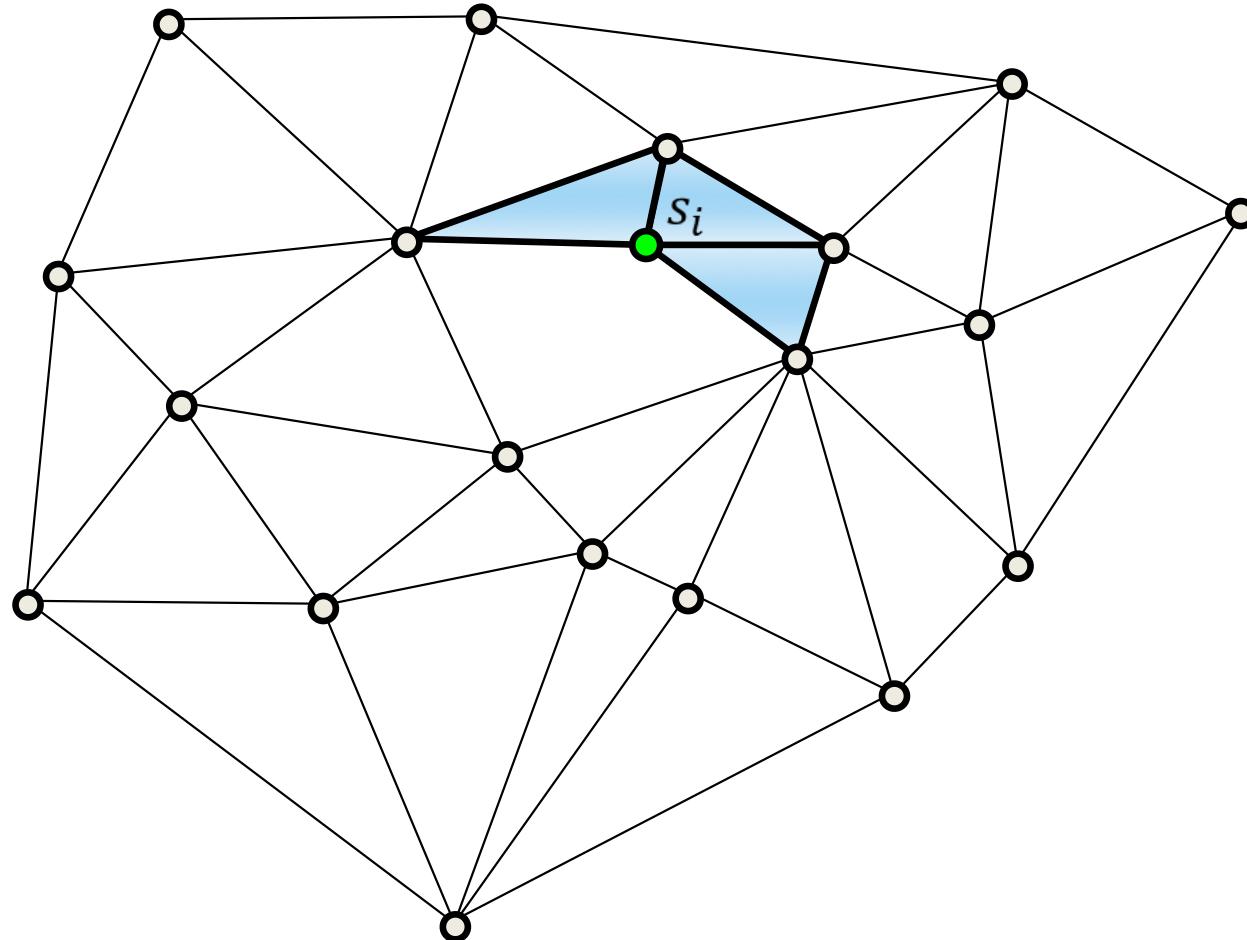
Bowyer-Watson Algorithmus

Beispiel



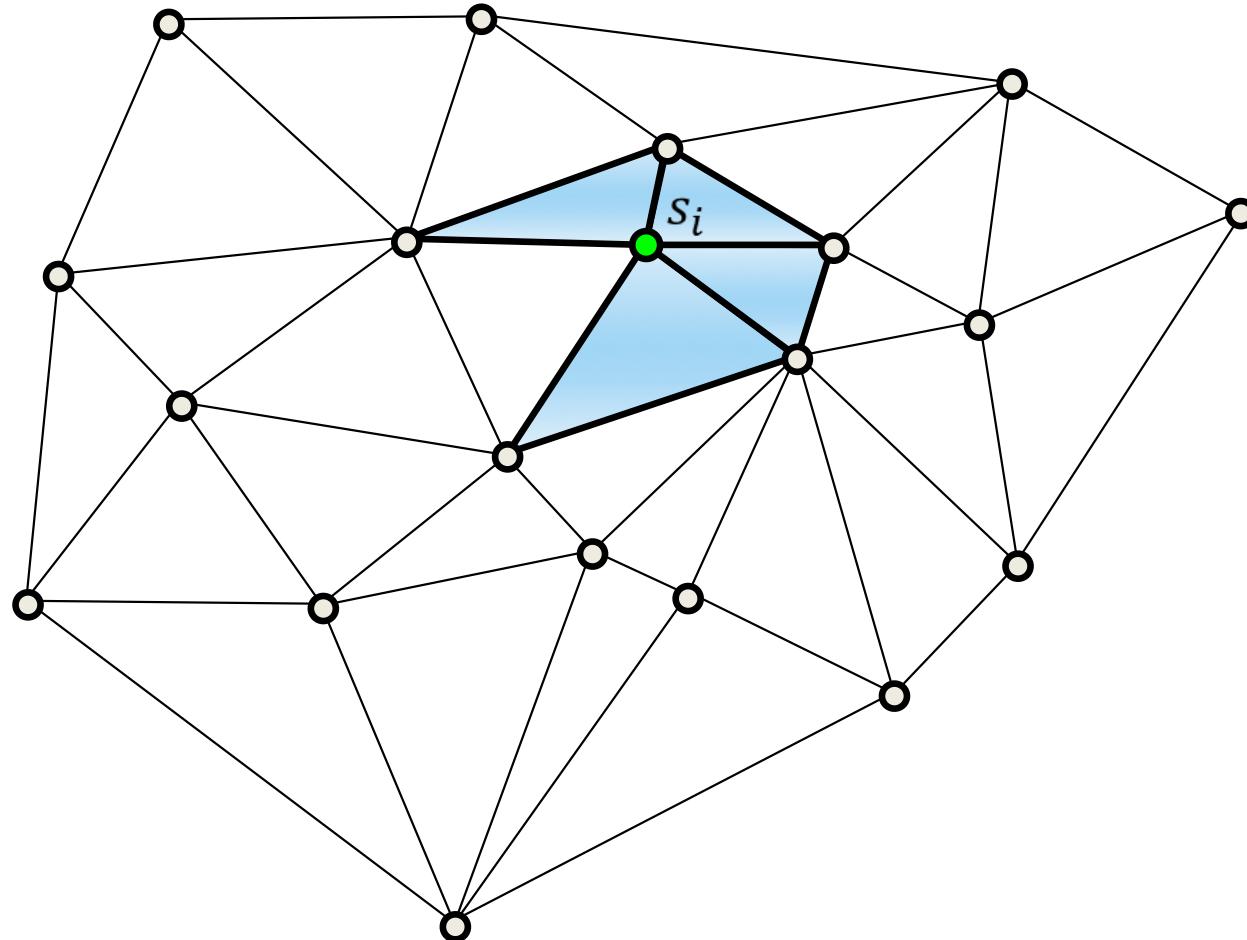
Bowyer-Watson Algorithmus

Beispiel



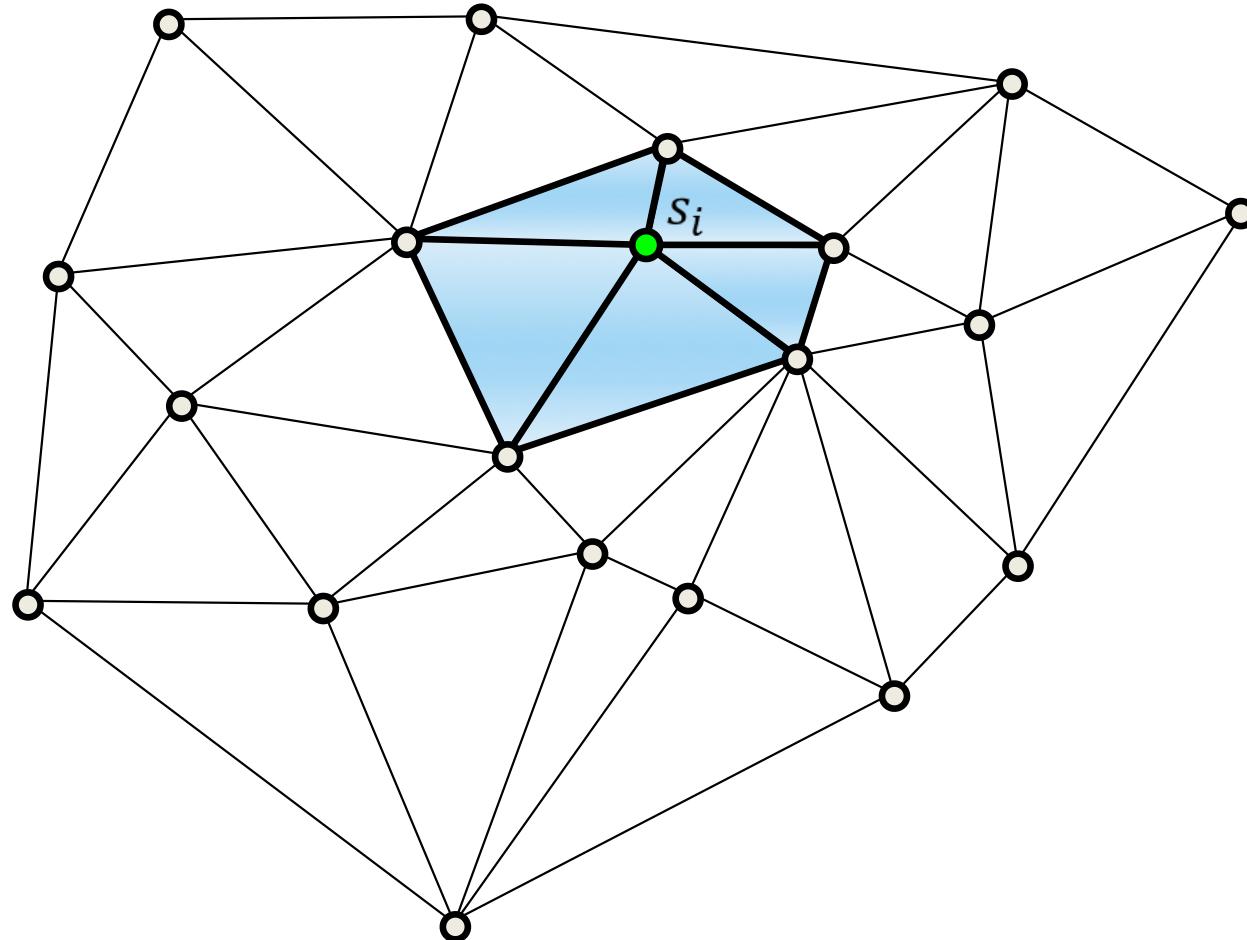
Bowyer-Watson Algorithmus

Beispiel



Bowyer-Watson Algorithmus

Beispiel



Bowyer-Watson Algorithmus

- ▶ bestimme ein Super Triangle (oder mehrere), das alle Punkte enthält
- ▶ erzeuge Dreiecksliste T , die zunächst nur aus dem Super Triangle besteht
- ▶ für jeden Punkt s_i
 - ▶ erzeuge eine (zunächst leere) Kantenliste K
 - ▶ für jedes Dreieck $t_j \in T$
 - ▶ wenn s_i im Umkreis von t_j liegt
 - ▶ füge die 3 Kanten von t_j in K ein und entferne t_j aus T
 - ▶ entferne mehrfach enthaltene Kanten in K
(danach enthält K nur den Rand des Lochs)
 - ▶ füge neue Dreiecke in T ein, die durch s_i und den Knoten des Lochs gebildet werden
 - ▶ entferne alle Dreiecke aus T , die Vertizes des Super Triangle enthalten

- es gibt noch eine Reihe anderer Algorithmen, z.B.
 - Radial Sweep
 - Divide and Conquer Ansätze (merge-based, split-based)
 - Intersecting Half Spaces

- interessant sind natürlich die Laufzeiten solcher Algorithmen
(hier: worst case)

Algorithmus	Dimension	Aufwand (Zeit)
Flipping	2	
Plane-sweep (wie gezeigt)	2	
Randomized Incremental (BW)	2	
Improved Plane-Sweep	2	
Divide and Conquer	2	
Randomized Incremental (BW)	≥ 3	

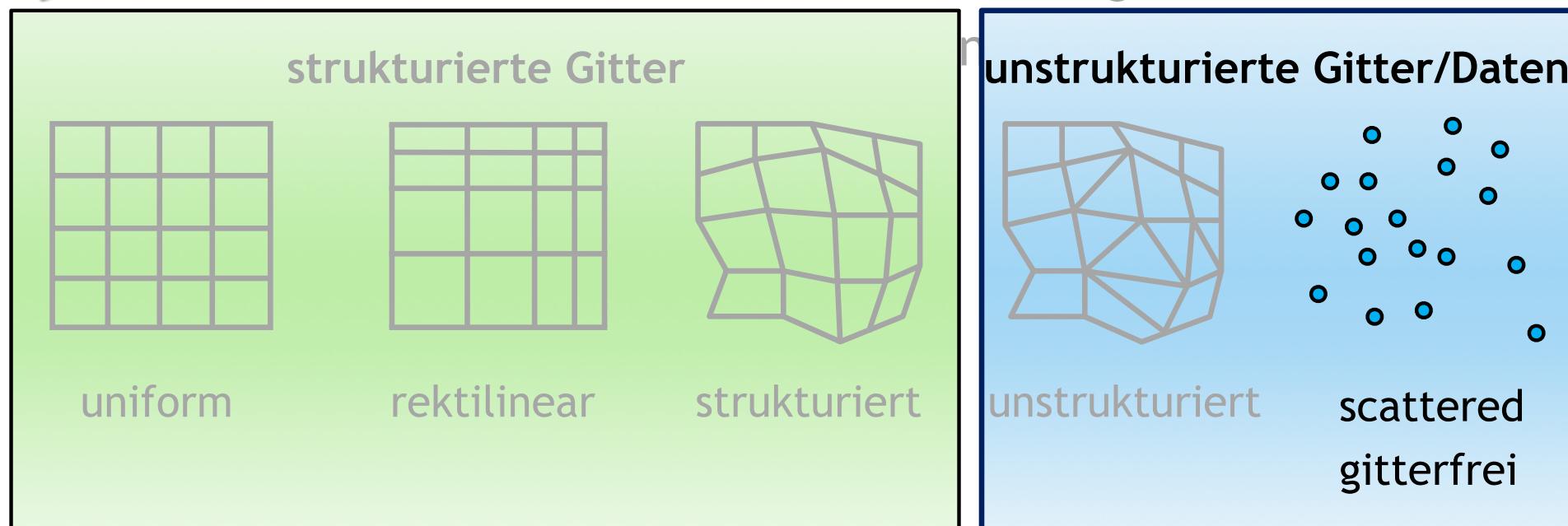
- es gibt noch eine Reihe anderer Algorithmen, z.B.
 - Radial Sweep
 - Divide and Conquer Ansätze (merge-based, split-based)
 - Intersecting Half Spaces

- interessant sind natürlich die Laufzeiten solcher Algorithmen
(hier: worst case)

Algorithmus	Dimension	Aufwand (Zeit)
Flipping	2	$O(n^2)$
Plane-sweep (wie gezeigt)	2	$O(n^2)$
Randomized Incremental (BW)	2	$O(n \log n)$
Improved Plane-Sweep	2	$O(n \log n)$
Divide and Conquer	2	$O(n \log n)$
Randomized Incremental (BW)	≥ 3	$O(n^{\lceil d/2 \rceil})$

Gitterfreie Daten und Gittertypen

- gitterfreie Daten: irregulär verteilte Positionen, keine Konnektivität
- Konnektivität herstellen und dann (in Simplizes) interpolieren, oder
- **direkte Interpolation anhand aller Datenpunkte**
- Gitter unterscheiden sich in
- Art und Form der Zellen aus denen sie aufgebaut sind



Grundidee

- bisher haben wir die Punktmenge trianguliert, dadurch können wir
 - ▶ Datenwerte innerhalb der Dreiecke einfach interpolieren
 - ▶ eine Umgebung/Wirkungskreis für Daten bestimmen (Voronoi-Zellen)
- ohne Gitter: **Inverse Distance Weighting** – fehlende Datenwerte durch eine gewichtete Summe der Werte an den Datenpunkten berechnen
 - ▶ Datenwerte v_i an den Punkten $s_i, x \in \mathbb{R}^d$

$$v(x) = \sum_{i=1}^N \frac{w_i(x)v_i}{\sum_{j=1}^N w_j(x)}$$

- ▶ Basisfunktion für einen Datenpunkt: $\frac{w_i(x)}{\sum_{j=1}^N w_j(x)}$
- ▶ als Gewicht könnte man z.B. $w_i(x) = \frac{1}{\|x-s_i\|^2}$ verwenden
- interessanter Überblick:
<http://scribblethink.org/Courses/ScatteredInterpolation/scatteredinterpcoursenotes.pdf>

Interpolation ohne Gitter

Inverse Distance Weighting: Shepard Interpolation

- Datenwerte v_i an den Punkten $s_i, \mathbf{x} \in \mathbb{R}^d$

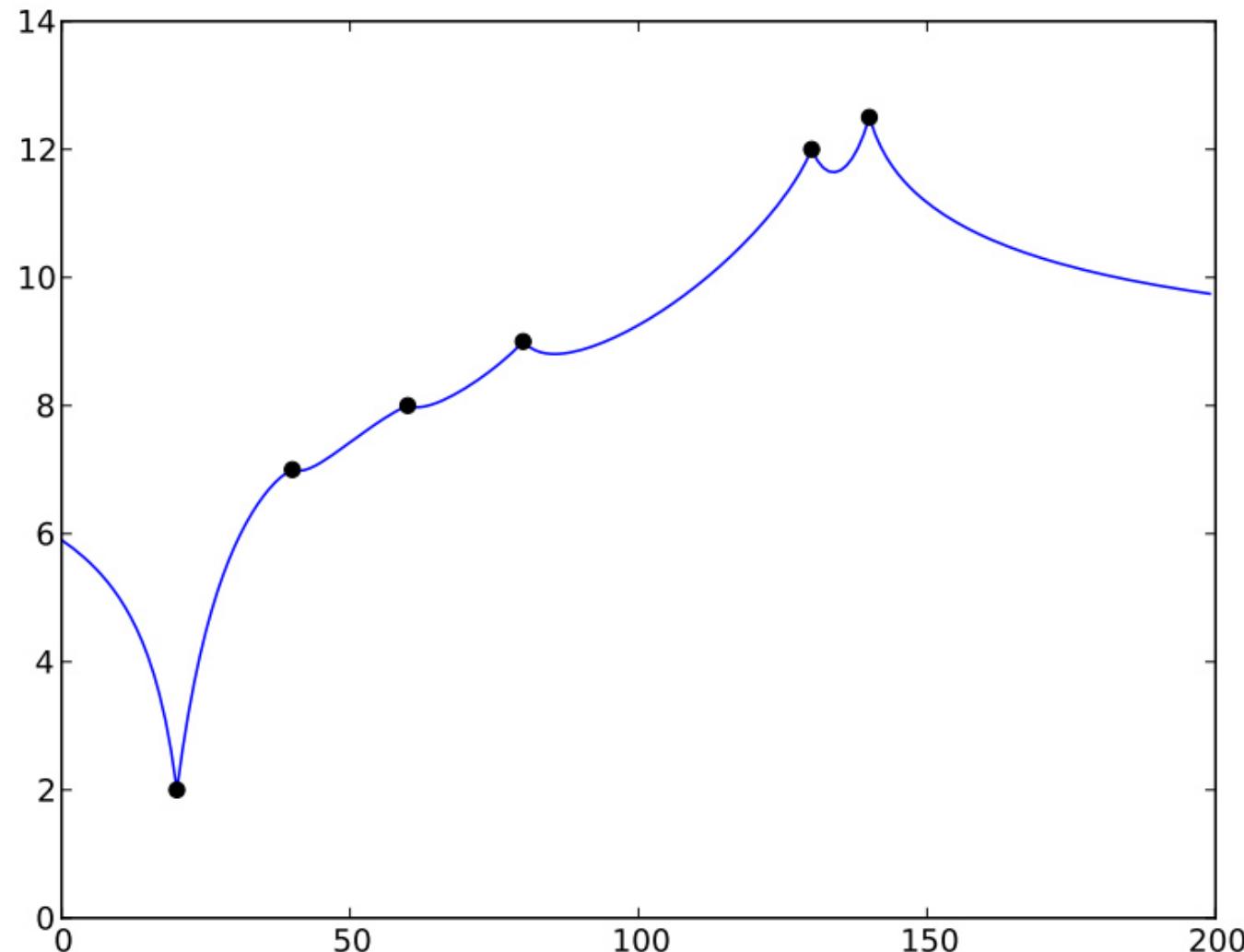
$$v(\mathbf{x}) = \sum_{i=1}^N \frac{w_i(\mathbf{x}) v_i}{\sum_{j=1}^N w_j(\mathbf{x})}$$

- $w(\mathbf{x}) = \frac{1}{d(\mathbf{x}, s_i)^p}$ mit einer beliebigen Metrik $d(\dots)$
 - meist euklidische Metrik
 - größere Werte für p erhöhen den lokalen Einfluss eines Datenpunktes
 - scharfe Peaks für $0 < p \leq 1$, „weiche“ Interpolation für $p > 1$
 - man kann zeigen: $p < d$ (Dimension) \Leftrightarrow interpolierte Werte dominiert durch entfernte Punkte
- optional: Interpolation berücksichtigt nur Datenwerte in der Nähe, z.B. die n -nahsten Werte, oder Werte mit einem maximalen Abstand
- andere Metriken sind natürlich möglich (und gebräuchlich)

Interpolation ohne Gitter

Inverse Distance Weighting: Shepard Interpolation

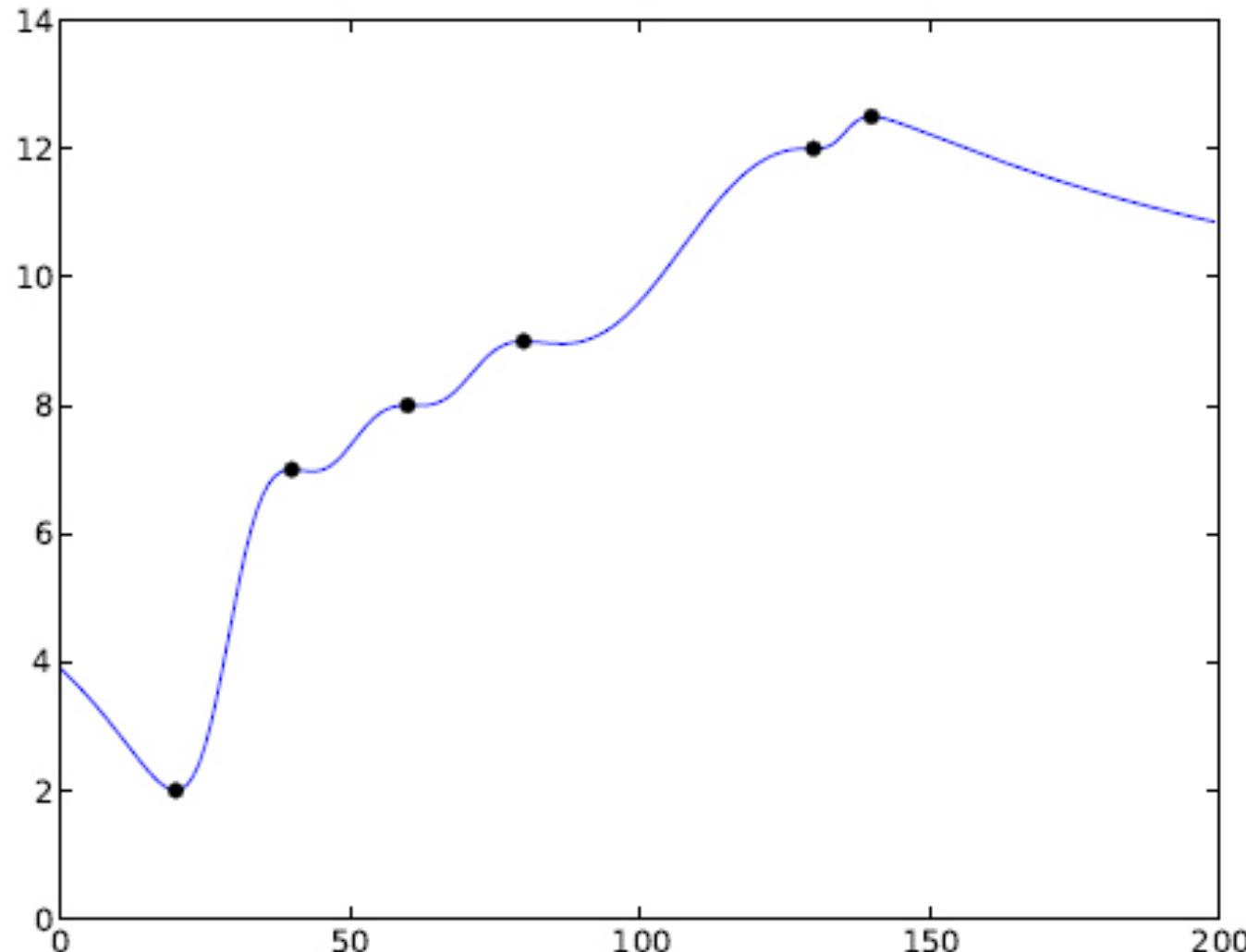
- Beispiel mit $p = 1$
- beachte: Peaks



Interpolation ohne Gitter

Inverse Distance Weighting: Shepard Interpolation

- Beispiel mit $p = 2$
- beachte: waagrechter Verlauf bei den Datenpunkten



Interpolation ohne Gitter

Inverse Distance Weighting: Shepard Interpolation

- Beispiel: zufällige Punkte auf der Fläche $z = \exp(-x^2 - y^2)$

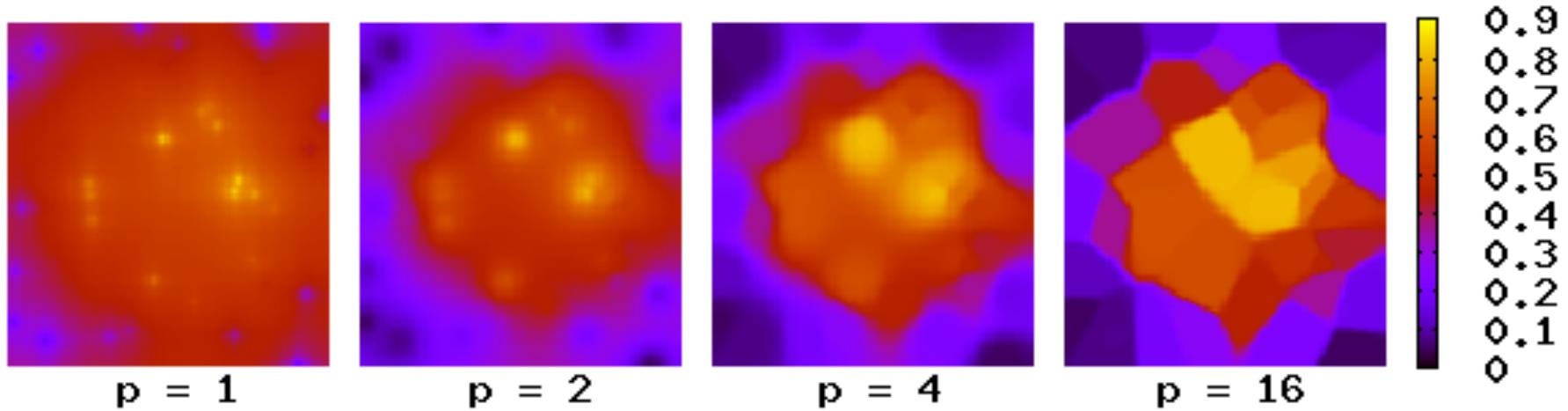


Bild: Wikipedia

- Shepard Interpolation ist kein idealer Interpolator
 - besser: nur Abtastpunkte in einer bestimmten Umgebung
 - dadurch bessere lokale Anpassung und weniger Aufwand

Radiale Basis Funktionen (RBF)

- ▶ ... sind ein ähnliches Konzept: eine Funktion wird wieder durch eine gewichtete Summe dargestellt, aber: es gibt nur eine Basisfunktion
- ▶ diese Gewichtsfunktion („Kernel“) hängt nur vom Abstand $\|\mathbf{x} - \mathbf{s}_i\|$ ab, Gewichte λ_i müssen bestimmt werden, dann ist der Interpolant:

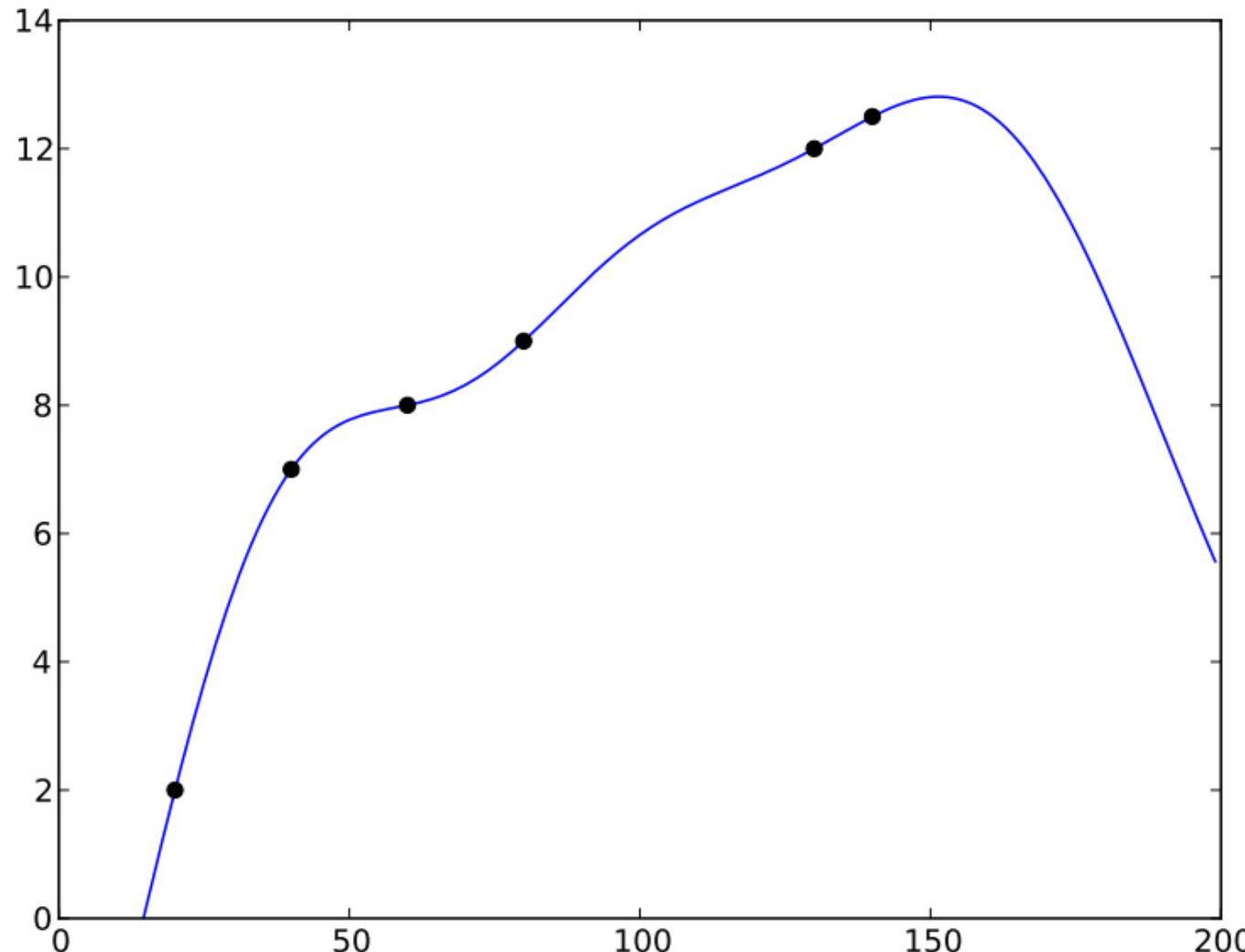
$$v(\mathbf{x}) = \sum_{i=1}^n \lambda_i \phi(\|\mathbf{x} - \mathbf{s}_i\|) + \sum_{m=0}^k c_m p_m(\mathbf{x})$$

- ▶ mit **univariater radialer Basisfunktion $\phi(r)$**
 - ▶ z.B. $\phi(r) = e^{-(r/c)^2}$
 - ▶ **polynomieller Basis p_m** für einen $(k+1)$ -dimensionalen Vektorraum
 - ▶ warum? polynomieller Teil ist **optional**, wird aber oft hinzugefügt, weil lineare Funktionen sonst sehr schlecht interpoliert werden!
 - ▶ außerdem: Extrapolation, da Kernel im Unendlichen verschwindet
 - ▶ daher haben die Polynome einen sehr niedrigen Grad (i.d.R. ≤ 2)

Interpolation ohne Gitter

Radiale Basis Funktionen (RBF)

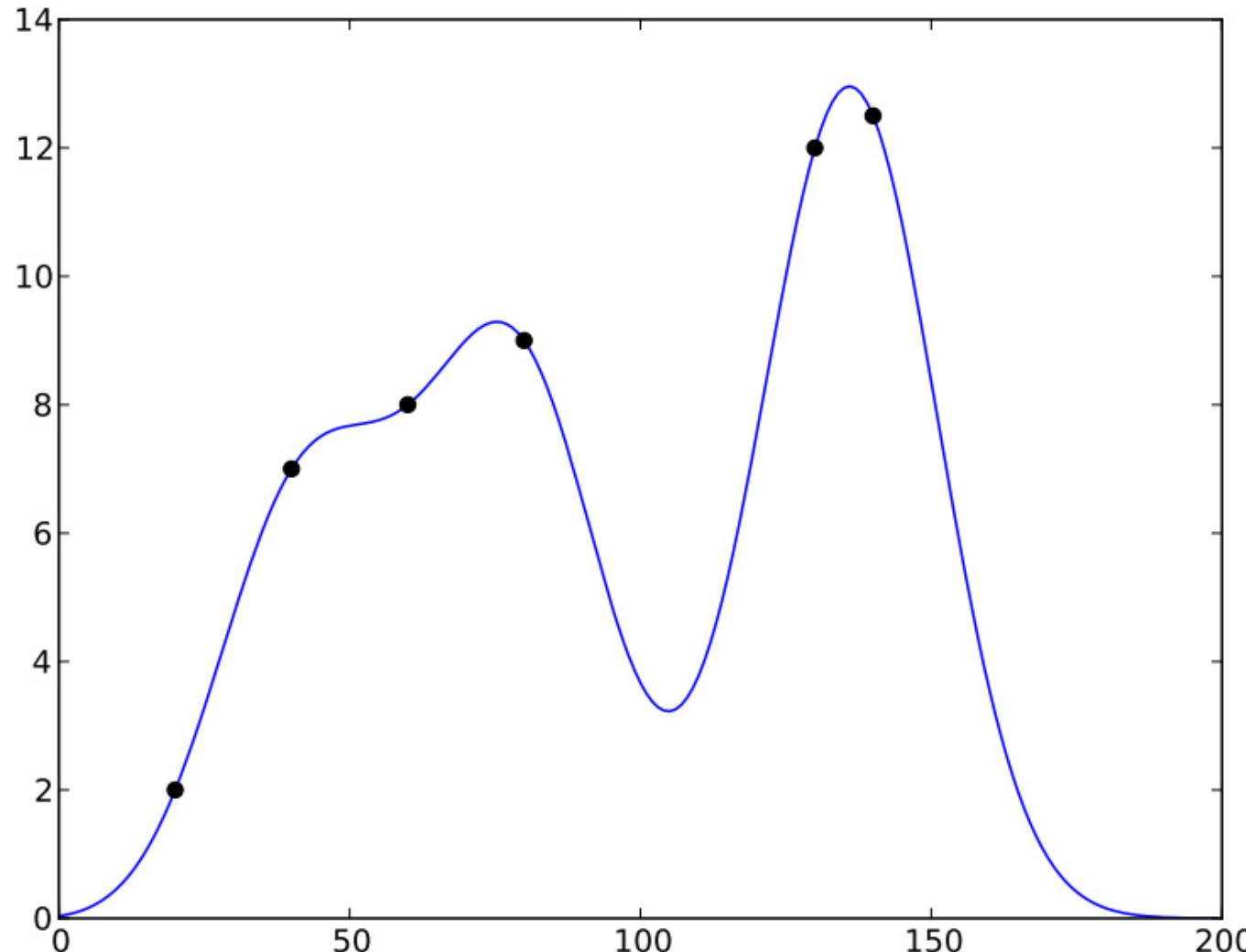
► Beispiel mit $\phi(r) = e^{-(r/c)^2}$



Interpolation ohne Gitter

Radiale Basis Funktionen (RBF)

► Beispiel mit $\phi(r) = e^{-(r/c)^2}$ (kleinerer Kernel als auf vorheriger Folie)



Radiale Basis Funktionen (RBF)

- wir haben ein unterbestimmtes System: wir haben n Gleichungen (die n Datenpunkte sollen interpoliert werden) aber $n + (k + 1)$ Unbekannte
- Gleichungssystem: alle Datenwerte sollen interpoliert werden

$$\sum_{i=1}^n \lambda_i \phi(\|s_j - s_i\|) + \sum_{m=0}^k c_m p_m(s_j) = f_j$$

- wenn der polynomielle Teil verwendet wird benötigen wir zusätzliche Orthogonalitäts-Randbedingungen
- wenn alle Punkte durch ein Polynom darstellbar sind, verschwinden die λ_i (siehe auch SIGGRAPH Course)

$$\sum_{i=1}^n \lambda_i p_m(s_i) = 0 \quad \forall m = 0 \dots k$$