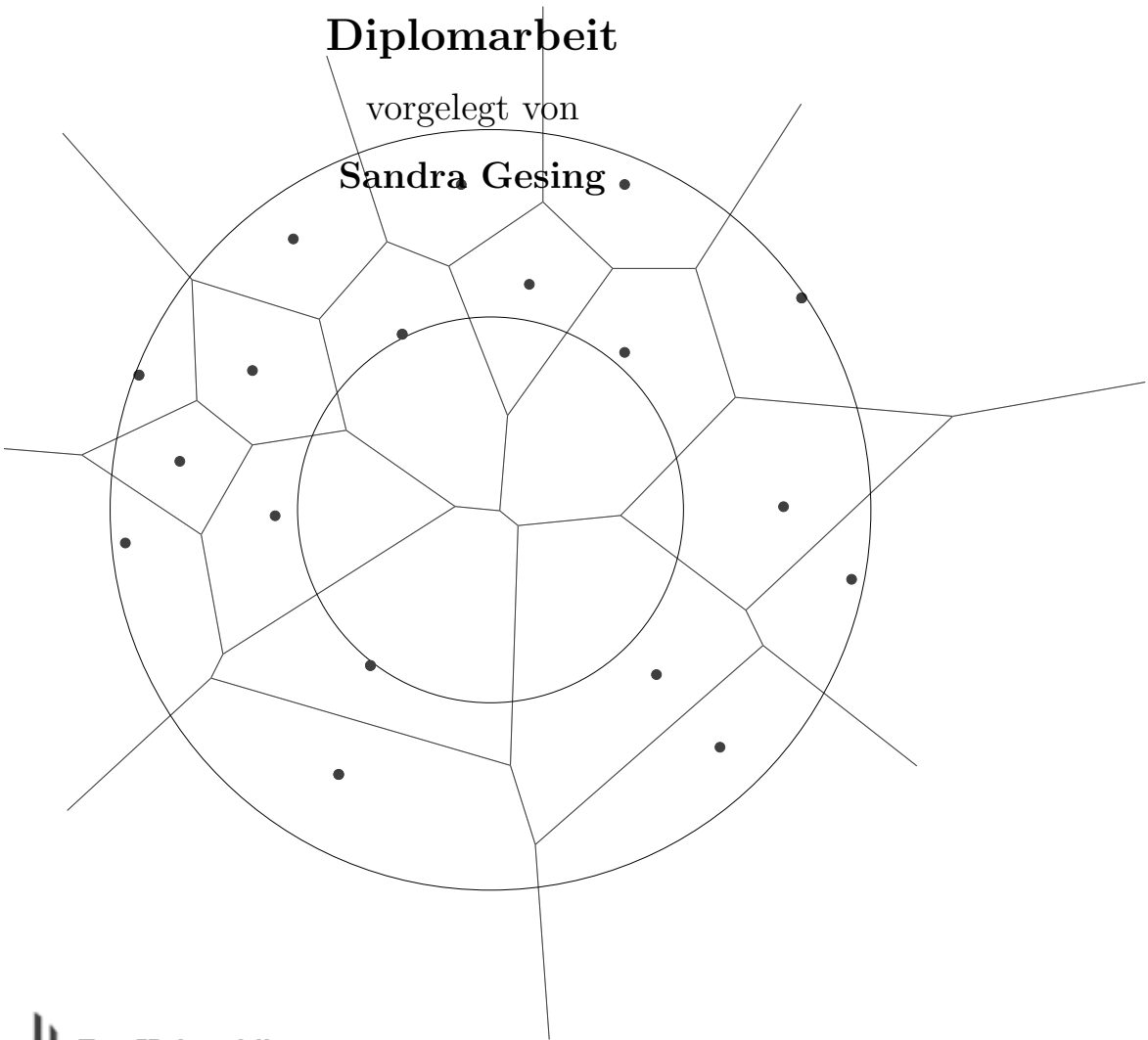
Approximation von Punktmengen durch Kreise

Diplomarbeit

vorgelegt von

Sandra Gesing



FernUniversität
Gesamthochschule
Hagen

Betreuer	Dr. Christian Icking
	Dr. Lihong Ma
Fachbereich	Praktische Informatik VI
Leiter	Prof. Dr. Jörg M. Haake

März 2005

Das Titelbild zeigt einen Ring mit minimaler Breite und das klassische Voronoi-Diagramm zu einer Punktmenge in der euklidischen Ebene.

Approximation von Punktmengen durch Kreise

Diplomarbeit

vorgelegt von

Sandra Gesing

März 2005

Inhaltsverzeichnis

Although this may seem a paradox, all science is dominated by the idea of approximation.
Bertrand Russell (1872 - 1970)

1

Einleitung

Der Anreiz, sich mit geometrischen Objekten wie Punkten und Kreisen zu beschäftigen, reicht bis zu den alten Ägyptern zurück. Er wird unter anderem dadurch motiviert, dass reale Objekte anhand ihrer geometrischen Eigenschaften wissenschaftlich untersucht werden können. Werden gegebene geometrische Objekte durch andere geometrische Objekte in einem mathematischen Kontext beschrieben, liegt eine Approximation im geometrischen Sinn vor. In welcher Art eine Annäherung vollzogen wird, hängt von der Zielsetzung ab. Die Qualität der Annäherung wird durch ein definiertes Maß, das den Abstand der Objekte zueinander wiedergibt, festgelegt.

Betrachtet man eine Punktmenge, die durch einen Kreis approximiert werden soll, kann die Zielsetzung sein, den maximalen Abstand der Punkte zu dem Kreis zu minimieren. Diese Approximation ist unter anderem in der Metrologie (*Messtechnik*) von Interesse. Da durch eine maschinelle Fertigung nicht immer exakte Kopien einer Vorlage gewährleistet werden können, ist bei der Qualitätssicherung zu überprüfen, ob die Abmessungen der Werkstücke in einer vorgegebenen Toleranz liegen. Ist die Vorlage kreisförmig und wird die Überprüfung anhand von Stichproben vollzogen, stellen diese die Punktmenge dar, deren maximaler Abstand zu einem Kreis bestimmt wird.

Approximationsaufgaben gehören zu der Klasse der Optimierungsaufgaben, welche dadurch charakterisiert sind, dass aus einer Menge von möglichen Lösungen für ein Problem die möglichst „beste“ Lösung ermittelt werden soll. Die Güte einer Lösung wird anhand einer reellen Zahl beurteilt. Die Aufgabe beispielsweise, einen Kreis zu finden, der eine gegebene Punktmenge in der Ebene umschließt, hat unendlich viele Lösungen. Wird sie allerdings um die Bedingung erweitert, dass der Kreis minimalen Radius haben soll, führt dies zu einer eindeutigen Lösung, die Teilmenge der Lösungsmenge der erstgestellten Aufgabe ist. Die Aufgabe, einen

Kreis mit minimalem Radius zu ermitteln, wurde bereits im Jahr 1857 von Sylvester gestellt [Syl57] und findet Anwendung in einer Teildisziplin der Unternehmensforschung (Operations Research), der Standortoptimierung (facility location). Wird zum Beispiel die Anforderung gestellt, mit einem Sender ein bestimmtes Gebiet von Haushalten zu versorgen, wird der kleinste umfassende Kreis um diese Haushalte gesucht. Der Sender wird auf den Mittelpunkt positioniert, so dass seine Leistung optimal ausgenutzt werden kann (siehe [PS85]). Strebt man im Gegensatz dazu an, seinen Wohnsitz soweit wie möglich von Störquellen wie Flugplätzen, Kraftwerken etc. in einem bestimmten Gebiet zu wählen, wird ein Kreis mit maximalem Radius, der keine Störquellen enthält, in diesem Gebiet gesucht. Der Mittelpunkt des Kreises stellt in diesem Fall die „beste“ Wahl für den Wohnsitz dar (siehe [Kle97]).

Anwendungen werden nicht nur durch die Frage „Was soll gelöst werden?“ spezifiziert, sondern ebenfalls durch die Fragen „Wie soll das Problem gelöst werden?“ und „Unter Einsatz welcher Mittel soll das Problem gelöst werden?“. Die Frage nach dem „Wie“ wird durch einen Algorithmus beantwortet, bei dem neben der Korrektheit die Laufzeit- und Speicherkomplexität (in Abhängigkeit von der Problemgröße) von besonderem Interesse sind. Je geringer der Bedarf an Laufzeit und Speicherplatz ist, desto effizienter kann die Implementation des Algorithmus auf einem Computer umgesetzt werden. Trotz der steigenden Leistungsfähigkeit von Computern ist es nach dem heutigen Stand der Forschung nur annähernd möglich, Probleme exponentieller Größe mit maschineller Hilfe zu lösen. Auch Algorithmen für einfache Probleme werden auf ihre Laufzeit im *worst-case* untersucht und wenn möglich optimiert, um spezielle Anforderungen an eine Anwendung erfüllen zu können.

Bei Animationen zum Beispiel, die Algorithmen für Voronoi-Diagramme visualisieren und dem Benutzer erlauben zu interagieren (siehe [urld]), spielt die Antwortzeit für die Bedienerfreundlichkeit eine entscheidende Rolle. Diese Antwortzeit hängt unter anderem von dem Laufzeitverhalten des implementierten Algorithmus, den benutzten Datenstrukturen, der Programmiersprache, in der der Algorithmus implementiert wurde, und dem System, auf dem das Programm gestartet wurde, ab. Während Datenstrukturen eng mit dem Algorithmus verknüpft sind und seinen Ablauf bedingen, geben die Programmiersprache und der ausführende Computer die Antwort auf die gestellte Frage, welche Mittel eingesetzt werden sollen.

Die Algorithmische Geometrie, der sich die vorliegende Arbeit zuordnen läßt, beschäftigt sich mit der Entwicklung und Analyse von effizienten Algorithmen zur Lösung geometrischer Probleme. Die geometrischen Probleme, die in dieser Arbeit vorgestellt werden, betreffen Punktmengen in der euklidischen Ebene. Es werden vier verschiedene Optimierungen bzw. Approximationen zu einer Punktmenge analysiert. Das sind der größte leere Kreis, der kleinste umfassende Kreis, der Ring mit minimaler Breite und der am besten angepasste Kreis. Die Aufbereitung dieser Themen gliedert sich in sechs Kapitel.

Das zweite Kapitel behandelt sowohl die Grundlagen für die Analyse der geometrischen Probleme, als auch einige Grundlagen für effiziente Algorithmen und ihrer Implementierung. Insbesondere wird dort auf Voronoi-Diagramme eingegan-

gen, anhand deren strukturellen Eigenschaften die Mittelpunkte und die Radien der gesuchten Kreise ermittelt werden können.

Der Zusammenhang zwischen den Kreisen und den Voronoi-Diagrammen wird im dritten Kapitel analysiert. Diese Analyse basiert teilweise auf bestehende Veröffentlichungen, die bei dem jeweiligen geometrischen Problem angegeben werden. Der größte leere Kreis kann anhand des klassischen Voronoi-Diagramms und der konvexen Hülle bestimmt werden. Für die Berechnung des kleinsten umfassenden Kreises dient das furthest-point Voronoi-Diagramm und der Durchmesser einer Punktmenge. Aus den Schnittpunkten der beiden genannten Voronoi-Diagramme und der Breite einer Punktmenge lässt sich ein Ring minimaler Breite ermitteln. Die Ermittlung des Rings stellt die Grundlage für den am besten angepassten Kreis dar. Für alle vier geometrischen Probleme werden im dritten Kapitel Beispiele für Anwendungen in der Praxis gegeben. Mit diesem dritten Kapitel sind die mathematischen Voraussetzungen für das vierte Kapitel geschaffen.

Dort werden effiziente Algorithmen zur Lösung der geometrischen Probleme vorgestellt. Sowohl der Algorithmus für den größten leeren Kreis als auch der Algorithmus für den Ring minimaler Breite ist in der Art noch nicht veröffentlicht worden. Die Algorithmen sind als Java-Applet *FitCircle* umgesetzt und im fünften Kapitel werden Aspekte dieser Implementierung konkretisiert. Eine explizite Hilfe für die Bedienung des Applets enthält der Anhang.

Abschließend wird das Thema „Approximation von Punktmengen durch Kreise“ im sechsten Kapitel diskutiert. Die Erweiterung der strukturellen Zusammenhänge zwischen Voronoi-Diagrammen und den vorgestellten Kreisen auf höherdimensionale Räume ist problemlos möglich. Allerdings erfordern zugehörige Algorithmen zum Teil andere Vorgehensweisen, als die, die in der Arbeit aufgezeigt werden.

Danksagung und Widmung

Ich danke besonders den Betreuern Dr. Christian Icking und Dr. Lihong Ma für das interessante Thema, die umfangreichen Hilfestellungen und die motivierende Unterstützung.

Sören Perrey, Frank Joest und Dr. Hagen Barlag haben durch konstruktive Diskussionen erheblich zum Gelingen der Arbeit beigetragen. Vielen Dank.

Auch meinen Arbeitskollegen und Freunden gebührt viel Dank für ihre kreativen Anregungen und ihre Geduld.

Zum guten Schluss bedanke ich mich bei meinen Eltern für ihre verständisvolle Unterstützung. Insbesondere widme ich meiner Mutter diese Arbeit, die ihre schwere Krankheit mit viel Kraft und Humor meistert.

Die beste Methode, um Informationen zu bekommen,
ist die, selbst welche zu geben.
Niccolò Machiavelli (1469 - 1527)

2

Grundlagen

In diesem Kapitel werden die Grundlagen und die Begriffe vorgestellt, die für die Diskussion von Approximationen im geometrischen Sinn unerlässlich sind. Neben allgemeinen Aussagen aus der Topologie, Geometrie und Graphentheorie wenden wir uns Aspekten der Implementation und speziellen Themen der Algorithmischen Geometrie zu.

Die Algorithmische Geometrie entstand in den 70er Jahren als Teildisziplin der theoretischen Informatik. Laut der Antrittsvorlesung von Smid *Algorithmische Geometrie: Reine Theorie?* [Smi97] wird die Dokorarbeit von Shamos *Computational Geometry* [Sha78] als Anfang dieses jungen Fachgebietes angesehen, obwohl es einige Veröffentlichungen davor gab (zum Beispiel *Graham's Scan* [Gra72] zur Berechnung der konvexen Hülle). Die Dissertation von Shamos enthält einige effiziente Algorithmen zur Lösung von grundlegenden geometrischen Aufgabenstellungen. Sie wurde von ihm selbst und Preparata zu dem mittlerweile als Standardwerk geltenden Buch *Computational Geometry: An Introduction* [PS85] erweitert. In den letzten Jahren haben sich die Veröffentlichungen des Fachgebiets immer mehr den praxisrelevanten Aspekten zugewandt. Smid hat diesen Wandel in einigen Punkten wie der Entwicklung von Softwarebibliotheken und robusten Algorithmen aufgezeigt.

Die folgenden Inhalte orientieren sich im Wesentlichen an [Kle97], können aber auch in einer Vielzahl von anderen Lehrbüchern nachgelesen werden.

2.1 Topologie und Geometrie

Die euklidische Geometrie ist Grundlage für die folgende theoretische Betrachtung von geometrischen Objekten in einem kartesischen Koordinatensystem.

Definition 2.1 (Distanzfunktion)
Sei M eine nichtleere Menge und $dist : M \times M \to \mathbb{R}$ eine Funktion mit den Eigenschaften

$dist(p,q) \geq 0$ *und* $dist(p,q) = 0 \Leftrightarrow p = q$ *(Definitheit)*
$dist(p,q) \leq dist(p,r) + dist(r,q)$ *(Dreiecksungleichung)*

für alle p, q, r aus M, dann heißt $dist$ Distanzfunktion oder Halbmetrik.

Definition 2.2 (Metrik)
Sei M eine nichtleere Menge und $dist : M \times M \to \mathbb{R}$ eine Distanzfunktion mit der zusätzlichen Eigenschaft

$dist(p,q) = dist(q,p)$ *(Symmetrie)*

für alle p, q aus M, dann heißt $dist$ Metrik und $(M, dist)$ heißt metrischer Raum.

Punkte sind elementare geometrische Objekte und lassen sich im d-dimensionalen Raum $\mathbb{R}^d$ als geordnete d-Tupel in der Form $p = (p_1, ..., p_d)$ mit $p_i \in \mathbb{R}$ darstellen. Die euklidische Norm ordnet jedem Punkt p den Abstand vom Ursprung zu (bestimmt also die Länge des Ortsvektors) und ist gegeben durch

$$\| p \|_2 := \sqrt{\sum_{i=1}^{d} p_i^2}.$$

Darauf aufbauend wird der euklidische Abstand bzw. die euklidische Metrik zwischen zwei Punkten p und q als

$$|pq| := \| p - q \|_2 = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2}$$

definiert. Die euklidische Metrik ist ein Spezialfall der Minkowski-Metriken L_r, die

für $1 \leq r < \infty$ als $L_r(p,q) := (\sum_{i=1}^{d} (p_i - q_i)^r)^{\frac{1}{r}}$ und
für $r = \infty$ als $L_\infty(p,q) := \max_{i=1}^{d} |x_i - y_i|$ festgelegt sind.

$\mathbb{R}^d$ zusammen mit der euklidischen Metrik wird als der d-dimensionale euklidische Raum bzw. im Fall $d = 2$ als die euklidische Ebene bezeichnet.

Die euklidische Ebene stellt in der vorliegenden Arbeit den zugrundeliegenden Raum für die Betrachtung von Approximationen durch Kreise dar. Aus diesem Grund werden wir uns im Weiteren auf die euklidische Ebene beschränken und $\mathbb{R}^2$ in diesem Sinne verwenden. Das Java-Programm *FitCircle* (siehe Kapitel 5) stellt den $\mathbb{R}^2$ anhand des kartesischen Koordinatensystems graphisch dar.

Der Umgebungsbegriff eines Punktes p im $\mathbb{R}^2$ wird über die euklidische Metrik festgelegt.

Definition 2.3 (ε-Umgebung)
Eine ε-Umgebung von $p \in \mathbb{R}^2$ mit $\varepsilon > 0$ und $\varepsilon \in \mathbb{R}$ ist eine Menge

$$U_\varepsilon(p) = \{q \in \mathbb{R}^2; |pq| < \varepsilon\}.$$

Eine Punktmenge $M \subset \mathbb{R}^2$ heißt *offen*, wenn es zu jedem Punkt eine ε-Umgebung gibt, die ganz in M liegt. Die Punkte mit dieser Eigenschaft werden als *innere* Punkte bezeichnet. Unter den *Randpunkten* von M versteht man die Punkte, für die jede ε-Umgebung sowohl Elemente aus M als auch aus dem Komplement $M^c := \mathbb{R}^2 \setminus M$ beinhalten. Die Menge der Randpunkte von M wird mit ∂M bezeichnet. Ist M offen und damit der Schnitt von M mit ihrem Rand ∂M leer, so heißt M^c *abgeschlossen*.

Der Kreisbegriff entstammt dem Umgebungsbegriff bei euklidischer Normierung.

Definition 2.4 (Kreis)
Ein Kreis $\mathcal{C}(m) \subset \mathbb{R}^2$ mit Mittelpunkt $m \in \mathbb{R}^2$ und Radius $r \in \mathbb{R}$ ist definiert als

$$\mathcal{C}(m) := \{p \in \mathbb{R}^2 : |mp| \leq r\}.$$

Der Rand eines Kreises beinhaltet alle Punkte p mit $\partial\mathcal{C}(m) = \{p \in \mathbb{R}^2 : |mp| = r\}$.

Der Abstand eines Punktes p zu einem Kreisrand $\partial C(m)$ mit Radius r ist definiert als $|pC(m)| := \sqrt{(r - |mp|)^2}$. Wird der Rand eines Kreises verbreitert, erhält man zwei konzentrische Kreise. Der innere Radius r_1 und der äußere Radius r_0 definieren einen *Ring* $\mathcal{A}$ um m durch $\mathcal{A}(m, r_1, r_0) = \{x : r_1 \leq |x, m| \leq r_0\}$ mit der Breite $r_0 - r_1$.

Definition 2.5 (Weg)
Sei $M \subset \mathbb{R}^2$. Unter einem Weg w von $p \in M$ nach $q \in M$ verstehen wir das Bild einer stetigen Abbildung

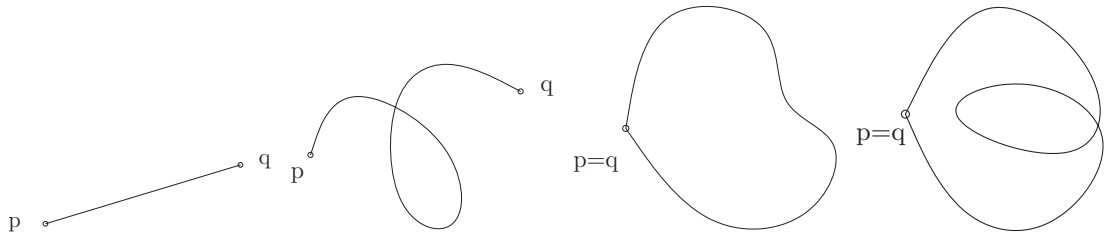
$$\tilde{w} : [0, 1] \to M \text{ mit } \tilde{w}(0) = p \text{ und } \tilde{w}(1) = q$$

und nennen $\tilde{w}$ eine Parametrisierung des Wegs w.

Ein Weg w heißt *geschlossen*, wenn $\tilde{w}(0) = \tilde{w}(1)$ gilt. Ist $\tilde{w}$ auf $[0, 1[$ injektiv, heißt w *einfach*.

Ein *Liniensegment* ist ein Spezialfall eines einfachen Wegs (siehe zum Beispiel die Abbildung 2.1(a) auf der nächsten Seite). Liniensegmente $\overline{pq}$ sind wie Punkte ebenfalls elementare geometrische Objekte, die durch die Menge $\overline{pq} = \{p + a(q - p); a \in \mathbb{R} \text{ mit } 0 \leq a \leq 1\}$ definiert werden. Ersetzen wir die Bedingung $0 \leq a \leq 1$ in der Menge durch $0 \leq a \leq \infty$, sprechen wir von einem *Strahl*, für $a \in \mathbb{R}$ ohne weitere Einschränkung von einer *Geraden*.

Ein *Gebiet* ist eine offene Teilmenge des $\mathbb{R}^2$, in denen je zwei Punkte miteinander verbunden werden können, so dass ein Weg existiert, der ganz in dem Gebiet liegt. Diese Eigenschaft nennt man *wegzusammenhängend* bzw. *zusammenhängend*.



(a) w_1 (b) w_2 (c) w_3 (d) w_4

Abbildung 2.1: Die Wege w_1 und w_3 sind einfach, w_3 und w_4 geschlossen.

Liegt für beliebige zwei Elemente p, q einer abgeschlossenen Menge $M \subset \mathbb{R}^2$ jeder Punkt des Liniensegments $\overline{pq}$ in M, heißt M *konvex*. Kreise beispielsweise sind offenbar konvex. Trifft zusätzlich für eine konvexe Menge $M \subset \mathbb{R}^2$ zu, dass für jeden Punkt $p \in M$ kein Strahl existiert, der in p startet und ganz in M enthalten ist, heißt M *beschränkt*. Die Eigenschaft der Beschränktheit einer Menge $M \subset \mathbb{R}^2$ lässt sich ohne die Voraussetzung der Konvexität folgendermaßen formulieren. M heißt beschränkt, wenn ein $D \in \mathbb{R}$ existiert, so dass für alle Punkte p, q aus M gilt, dass $|pq| \leq D$. Das Infimum aller $D \geq 0$ wird als *Durchmesser* von M bezeichnet. Eine abgeschlossene und beschränkte Menge $M \subset \mathbb{R}^2$ wird auch *kompakt* genannt.

Geraden teilen den $\mathbb{R}^2$ in zwei Halbebenen auf und können zur Charakterisierung dieser Halbebenen dienen wie beispielsweise der Bisektor zwischen zwei Punkten.

Definition 2.6 (Bisektor im $\mathbb{R}^2$)
Seien p, q verschiedene Punkte im $\mathbb{R}^2$. Dann heißt die Menge

$$B(p, q) := \{r \in \mathbb{R}^2 : |pr| = |qr|\}$$

Bisektor von p und q.

Der Bisektor teilt den $\mathbb{R}^2$ in zwei offene Halbebenen, die von p bzw. q dominiert werden. Das heißt, in der von p dominierten Halbebene $D(p, q)$ liegen alle Elemente aus $\mathbb{R}^2$ näher an p als an q. Für die von q dominierte Halbebene $D(q, p)$ gilt dies analog. Der Bisektor zwischen zwei Punkten im $\mathbb{R}^2$ entspricht der Mittelsenkrechten zwischen diesen Punkten.

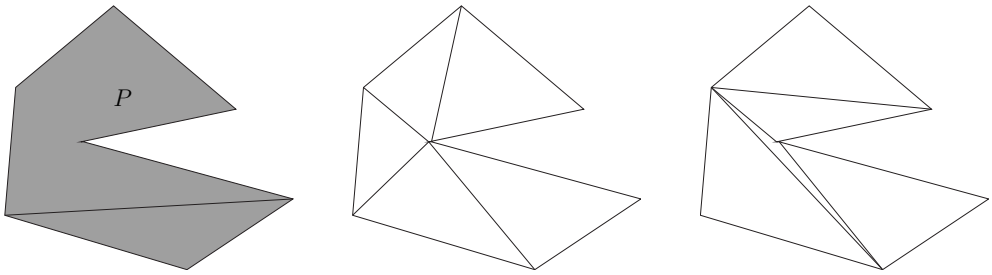
Eine *Stützgerade* zu einer Menge $M \subset \mathbb{R}^2$ ist eine ausgezeichnete Gerade, die einen Randpunkt von M enthält und $\mathbb{R}^2$ in zwei offene Halbebenen teilt, von der eine disjunkt von M ist. Gibt es für jeden Randpunkt einer Menge M eine Stützgerade, ist M konvex. Die Halbebene, die alle Punkte aus M enthält, wird mit h^+, die disjunkte Halbebene mit h^- bezeichnet.

Durch eine Folge von aneinander anschließenden Liniensegmenten wird eine *polygonale Kette* definiert. Falls der Anfangs- und Endpunkt in der Ebene übereinstimmen, heißt das umschlossene, innere Gebiet zusammen mit der polygonalen Kette *Polygon* P. Die definierenden Punkte der Liniensegmente werden auch mit *Ecken* bezeichnet. Der Rand von P, also die polygonale Kette, wird ∂P, das innere Gebiet $int(P)$ genannt. Schneiden sich die Liniensegmente nur an ihren Endpunkten,

sprechen wir von einem *einfachen* Polygon. Eine *Diagonale* eines einfachen Polygons P ist ein Liniensegment $l \in P$, das zwei Ecken von P verbindet und nur in diesen Endpunkten ∂P schneidet. Durch eine Diagonale wird P in zwei Teilpolygone zerlegt.

Definition 2.7 (Triangulation eines einfachen Polygons)
Sei P ein einfaches Polygon. Dann heißt eine maximale Menge sich nicht kreuzender Diagonalen von P zusammen mit ∂P eine Triangulation von P.

Eine Triangulation ist eine Zerlegung eines einfachen Polygons in Dreiecke. Jedes einfache Polygon P kann trianguliert werden und es gibt endlich viele verschiedene Triangulationen von P. Diese unterscheiden sich anhand der Lage der Dreiecke und Diagonalen, jedoch per definitionem nicht anhand deren Anzahl. Durch eine einfache Induktion über die Anzahl der Ecken kann gezeigt werden, dass jede Triangulation eines einfachen Polygons mit n Ecken genau $n-2$ Dreiecke und $n-3$ Diagonale besitzt.



(a) P mit einer Diagonalen (b) (c)

Abbildung 2.2: Ein einfaches Polygon P mit möglichen Triangulationen

Ein konvexes Polygon ist ein spezielles einfaches Polygon.

Definition 2.8 (Konvexes Polygon)
Sei $H := \{h_1^+, ..., h_n^+\}$ eine endliche Menge abgeschlossener Halbebenen des $\mathbb{R}^2$ für $n \geq 3$, deren Schnitt nichtleer und beschränkt ist. Dann bezeichnen wir

$$P := \bigcap_{i=1}^{n} h_i^+$$

als konvexes Polygon.

Die Polygonkanten eines konvexen Polygons induzieren jeweils eine Stützgerade, die eine Halbebene h_i^+ für $i \in \{1, ..., n\}$ begrenzt. Allgemein haben Geraden höchstens zwei „echte“ Schnittpunkte mit dem Rand eines konvexen Polygons. Als „echt“ bezeichnen wir die Schnittpunkte zwischen Geraden und Kanten, wenn eine Kante

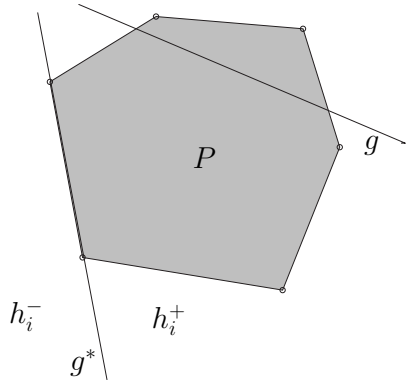


Abbildung 2.3: Ein konvexes Polygon

nicht in einer Geraden enthalten ist. Treten genau zwei Schnittpunkte auf, schneidet eine Gerade zwei verschiedene Kanten des konvexen Polygons.

Abbildung 2.3 stellt exemplarisch eine Stützgerade g^* eines konvexen Polygons P und eine Gerade g, die zwei Kanten von P schneidet, dar. Der *Kern* $ker(P)$ eines Polygons P beinhaltet die Menge der Punkte, die mit allen Punkten aus P durch ein Liniensegment im Innern von P verbunden werden können, ist also immer konvex.

Konvexe Polygone in der Ebene können als kreuzungsfreie geometrische Graphen interpretiert werden.

2.2 Graphentheorie

Ein *Graph* G ist ein Paar (V, E) bestehend aus einer endlichen Knotenmenge V (*vertices*) und einer Kantenmenge E (*edges*) mit $E \subseteq V \times V$. In *gerichteten* Graphen wird für eine Kante $e = (p, q)$ als Anfangsecke p und als Endecke q und damit die Orientierung festgelegt. Bei *ungerichteten* Graphen wird nicht zwischen der Kante (p, q) und der Kante (q, p) unterschieden. Die Anzahl der Kanten, die von einem Knoten ausgehen, nennt man den *Grad* dieses Knotens. Die Flächen F (faces) eines Graphen in der Ebene sind Gebiete $\mathbb{R}^2 \setminus G$.

Die Knoten p und q heißen *inzident* zur Kante $e = (p, q)$. Jede Kante ist höchstens zu zwei Knoten inzident. Sind zwei Knoten p und q benachbart, werden sie also über eine Kante verbunden, sind sie zueinander *adjazent*. Dementsprechend werden Kanten als adjazent bezeichnet, wenn sie einen Knoten gemeinsam haben. Wir sprechen von einem *schlichten* Graphen, wenn es keine Kanten gibt, die nur zu einem Knoten inzident sind (Schlingen, $e = (p, p)$) und keine parallelen Kanten zwischen zwei Knoten auftreten. Ein Graph $G = (V, E)$ wird *zusammenhängend* genannt, wenn ein Weg w zwischen zwei beliebigen Knoten $p, q \in V$ existiert, so dass w einer endlichen Folge von Kanten $e_i \in E$ mit $i \in \{1, ..., n\}$ entspricht. Ein schlichter Graph, der zusammenhängt und keine beschränkten Flächen hat, heißt *Baum*.

Eine geometrische Realisierung eines Graphen G bildet die Knoten auf paarweise

verschiedene Punkte in einem metrischen Raum ab. Die Kanten wiederum verbinden jeweils zwei Punkte ohne andere Punkte zu berühren. Diese Darstellung heißt der zu G gehörige *geometrische* Graph. Hat dieser zusätzlich die Eigenschaft, dass sich keine Kanten kreuzen, nennt man den geometrischen Graphen *kreuzungsfrei*.

Für einen zusammenhängenden kreuzungsfreien geometrischen Graphen in der Ebene existieren folgende zentrale Beziehungen zwischen der Anzahl der Knoten, der Anzahl der Kanten und der Anzahl der Flächen (Beweis siehe [Kle97]).

Satz 2.1 (Eigenschaften eines geometrischen Graphen)
Sei $G = (V, E)$ ein zusammenhängender kreuzungsfreier geometrischer Graph in der Ebene. Bezeichne $|v|$ die Anzahl seiner Knoten, $|e|$ die Anzahl seiner Kanten und $|f|$ die Anzahl seiner Flächen.

1. $|v| - |e| + |f| = 2$ *(Eulersche Formel)*
2. $|v| \leq \frac{2}{3}|e|$, *falls alle Knoten mindestens den Grad 3 haben.*

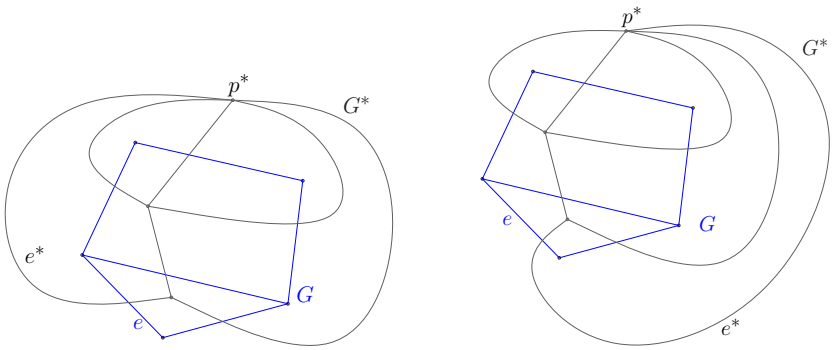


Abbildung 2.4: Ein Graph G mit möglichen dualen Graphen G^* in der Ebene

Ein *dualer* Graph G^* eines kreuzungsfreien, nichtleeren, zusammenhängenden Graphen G in der Ebene lässt sich über zwei Bedingungen herleiten. Als Knoten wird jeweils ein Punkt p^* aus jeder Fläche von G für G^* gewählt. Diese werden mit einer Kante e^* in G^* verbunden, falls ihre Flächen benachbart sind und e^* nur die Kante e aus G kreuzt, die zwischen diesen beiden Flächen herführt. Die duale Darstellung ist nicht eindeutig, wie Abbildung 2.4 zeigt. Die Lage der Kante e^* ist bei den beiden dualen Graphen in der Ebene unterschiedlich.

Stellt man sich jedoch vor, dass der duale Graph auf einer Kugeloberfläche konstruiert wird, ist er, bis auf Verformungen, eindeutig bestimmt. Aufgrund dieser Eigenschaft wird G^* als dualer Graph von G bezeichnet. Die nochmalige Anwendung der Konstruktionsvorschrift auf einen dualen Graphen führt zu einem Graphen $(G^*)^*$, der äquivalent zum Ausgangsgraphen G ist.

2.3 Komplexität von Algorithmen

Im Allgemeinen wird in der Informatik unter Komplexität von Algorithmen die *rechnerische Komplexität* verstanden, die den abgeschätzten Bedarf an Rechenzeit und Speicherplatz angibt. Bei der Untersuchung der Komplexität von Algorithmen unterscheiden wir zwischen dem experimentellen Ansatz auf einem konkreten Prozessor und dem analytischen Ansatz. Für Letzteres wird ein abstraktes Modell herangezogen, um die Komplexität bezüglich einer idealisierten Maschine näherungsweise zu bestimmen.

Allgemein verwendete Rechenmodelle sind in diesem Zusammenhang die REAL RAM, die Turing- und die Registermaschine, wobei die erste oft in der Algorithmischen Geometrie eingesetzt wird. Sie lässt in einem unendlichen Adressraum Operationen auf reellen Zahlen in konstanter Zeit zu. Speicherzellen können dabei jeweils direkt oder indirekt adressiert werden. Zur Analyse der Komplexität von Algorithmen werden wir von dem Berechnungsmodell der REAL RAM ausgehen.

Die klassische Komplexitätstheorie (siehe zum Beispiel [Ver99]) sowie die O-, Ω- und Θ-Notationen werden als bekannt vorausgesetzt. In [Kle97] wird gezeigt, dass das *Sortierproblem* von n Objekten die Komplexität $\Theta(n \log n)$ aufweist, wenn auf den Objekten eine vollständige Ordnung $<$ definiert ist. Diese Aussage werden wir bei weiteren Betrachtungen voraussetzen. Kann ein Problem Π_1 in Linearzeit auf ein Problem Π_2 reduziert werden, geben wir dies mit $\Pi_1 \propto \Pi_2$ an.

In der Praxis spielt jedoch nicht nur die Komplexität eines Algorithmus eine entscheidende Rolle für die Implementierung, sondern auch der Programmieraufwand, numerische Stabilität oder sein Zeitverhalten in Abhängigkeit von Eingabegrößen. Das Ziel ist einen praktikablen und optimalen Algorithmus zu finden.

Eine eher empirische Betrachtung der Komplexität von Algorithmen stellt die *psychologische Komplexität* dar, die sich auf den Aufwand zum Verständnis eines Algorithmus und seiner Implementation bezieht (siehe [Dum99]). Eine allgemeine Bewertung der psychologischen Komplexität wird über definierte Komplexitätsmaße ermöglicht und für diese werden auf Grundlage von empirischen Untersuchungen konkrete Werte vorgeschlagen. Ob ein Programm verstanden wird, ist eine subjektive Einschätzung, jedoch unterstützt laut Dumke beispielsweise die Kommentierung und Strukturierung des Programmtextes nachweisbar die Lesbarkeit. Neben diesen Maßnahmen und aussagekräftigen Variablen- und Methodennamen sind bei der Programmierung von *FitCircle* folgende von Dumke aufgeführte Empfehlungen für objektorientierte Komplexitätsmaße weitestgehend beachtet worden.

- *„die Vererbungstiefe in der Klassenhierarchie, die nach Lorenz nicht tiefer als 5 sein sollte,*

- *die Anzahl der Methoden pro Klasse, die nach Chidamber nicht mehr als 20 sein sollte,*

- *die Anzahl Anweisungen pro Methode, die ebenfalls nach Lorenz bei Java (in Anlehnung an C++) nicht mehr als 18 betragen sollte.“*

2.4 Konvexe Hülle

Eine wichtige Struktur in der Algorithmischen Geometrie ist die konvexe Hülle. Schon frühe Veröffentlichungen des Fachgebiets wie die von Graham [Gra72] widmen sich der konvexen Hülle. Sie wird nicht nur in zahlreichen Lehrbüchern der Algorithmischen Geometrie analysiert, sondern spielt ebenfalls eine grundlegende Rolle in der reinen Mathematik (siehe [BSMM93]).

Definition 2.9 (Konvexe Hülle)
Für eine beliebige Menge M des $\mathbb{R}^2$ heißt

$$ch(M) := \bigcap_{\substack{K \supseteq M \\ K konvex}} K$$

die konvexe Hülle von M.

$ch(M)$ zeichnet sich unter anderem dadurch aus, dass sie die kleinste konvexe Menge ist, die M enthält. In der vorliegenden Arbeit ist die konvexe Hülle ebener Punktmengen und ihre Eigenschaften von besonderem Interesse.

Lemma 2.1 (Eigenschaften der konvexen Hülle endlicher Punktmengen im $\mathbb{R}^2$)
Sei M eine Menge von n Punkten im $\mathbb{R}^2$.

1. *Sei $n = 1$. Dann besteht $ch(M)$ aus einem Punkt.*
2. *Sei $n = 2$ oder seien alle Punkte aus M kollinear[1] angeordnet. Dann besteht $ch(M)$ aus einem Liniensegment.*
3. *Sei $n \geq 3$ und nicht alle Punkte aus M kollinear angeordnet. Dann ist $ch(M)$ ein konvexes Polygon, dessen Ecken Punkte aus M sind.*
4. *Sei w ein einfacher, geschlossener Weg, der $ch(M)$ umschließt. Dann ist der Rand von $ch(M)$ höchstens so lang wie w.*

Für Beweise der Aussagen sei an dieser Stelle auf [Kle97] verwiesen. Die ersten drei Aussagen implizieren, dass die konvexe Hülle aus $O(n)$ vielen Knoten und Kanten besteht. Nimmt man vereinfachend an, dass ein Punkt oder ein Liniensegment ein konvexes Polygon darstellt, sind Aussage 1 und 2 in Aussage 3 enthalten. Für die Ermittlung der konvexen Hülle einer ebenen Punktmenge haben wir also das konvexe Polygon zu bestimmen, das M enthält und dessen Ecken Punkte aus M

[1] Punkte einer endlichen Teilmenge des $\mathbb{R}^2$ heißen kollinear angeordnet, wenn mindestens 3 Punkte auf einer Geraden liegen.

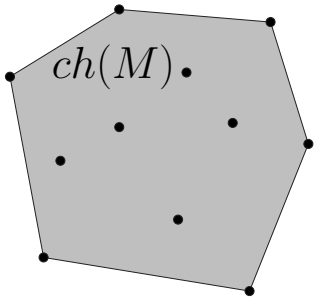


Abbildung 2.5: Konvexe Hülle einer ebenen Punktmenge M

sind. Die Eigenschaft, dass der Rand von $ch(M)$ der kürzeste Weg um die Menge M ist, lässt sich nach [Kle97] folgendermaßen veranschaulichen: „Stellt man sich die Punkte als Nägel vor, die aus der Ebene herausragen, wirkt der Rand der konvexen Hülle wie ein Gummiband, das sich um die Nägel herumspannt."

2.5 Voronoi-Diagramme

Das Konzept von Voronoi-Diagrammen basiert auf der Idee, dass ein gegebener Raum durch den Einfluss ausgezeichneter Objekte in Regionen aufgeteilt wird. Diese allgemeine Beschreibung lässt großen Spielraum für Interpretationen und spiegelt die Vielseitigkeit dieses Konzeptes wider. Das Spektrum reicht von Gedanken im Mittelalter über den begrenzten Einfluss von Göttern, den sie durch übermenschliche Kräfte auf ihr Umfeld nehmen[2], bis zu mathematischen Definitionen in der Neuzeit, die Voronoi-Regionen anhand von Metriken festlegen.

Im Jahr 1644 veröffentlichte Descartes [Des44] seine Theorie über das Sonnensystem, dessen graphische Darstellung der Materiewirbel um Fixsterne einem Voronoi-Diagramm sehr nahe kommt. Die exakte mathematische Darstellung wurde allerdings erst von Dirichlet 1850 [Dir50] genutzt, wobei der etablierte Name im Bereich der Informatik auf den russischen Mathematiker Voronoj[3] und seine Arbeit [Vor07] von 1907 zurückzuführen ist.

Anwendungen von Voronoi-Diagrammen werden in einer Vielzahl von Wissenschaften wie beispielsweise Archäologie, Biologie, Kartographie, Kristallographie, Meterologie und Operations Research eingesetzt und wurden unabhängig voneinander entwickelt. Ähnlich vielfältig wie ihre Einsatzgebiete sind die Synonyme, die für Voronoi-Diagramme existieren und in den verschiedenen Wissenschaften verwandt werden: Dirichlet Tesselationen, Wigner-Seitz-Zellen, Thiessen-Polygone. Neben der konvexen Hülle ist das Voronoi-Diagramm eine der fundamentalen Strukturen der

[2] In der Diplomarbeit [Sau95] wird vermutet, dass diese Gedanken schon in der Antike auftraten, während sich [Tin98]) an mittelalterliche Schriften hält.

[3] Es gibt wenige Arbeiten, die die Schreibweise „Voronoj-Diagramme" bevorzugen. In der Fachliteratur hat sich die englische Transliteration „Voronoi" durchgesetzt.

Algorithmischen Geometrie. Es dient unter anderem zur Lösung von sogenannten *Point-Location-Problems*, der Bewegungsplanung in der Robotik und zur Cluster-Analyse. Gerade in den letzten zwei Jahrzehnten ist in der Algorithmischen Geometrie ein wachsendes Interesse an Voronoi-Diagrammen zu beobachten, was sich in der Menge der Veröffentlichungen zu diesem Thema niederschlägt. Für einen Überblick zu den verschiedenen Anwendungen werden hier nur einige Quellen aufgeführt wie das Buch von Okabe et al. [OBS92] und die Übersichtsartikel von Aurenhammer [Aur91], Aurenhammer und Klein [AK96], Leven und Sharir[LS87] und Fortune [For95].

Neben der Wissenschaft findet diese geometrische Struktur ebenfalls ihren Platz in der Kunst, wie die Artikel [Nee88] von Nees und [Ste99] von Stewart exemplarisch aufzeigen. Stewart bezeichnet sie dort als „klassisches Beispiel für die Einheit von Mathematik, Kunst und Wissenschaft“ und schließt seine Ausführungen mit dem Satz „So umfasst diese einfache Idee interessante Kunstwerke, elegante Mathematik und tiefliegende Ideen über die Struktur des Universums“. In dem Artikel von Stewart wird das Voronoi-Diagramm betrachtet, bei dem disjunkte Regionen über jeweils ausgezeichnete Objekte bezüglich der euklidischen Metrik definiert sind. Die Regionen enthalten die Elemente, die zu dem zugehörigen ausgezeichnetem Objekt näher liegen als zu irgendeinem anderen ausgezeichetem Objekt. Befindet sich außerdem das Voronoi-Diagramm in der euklidischen Ebene, liegt ein sogenanntes *klassisches* Voronoi-Diagramm vor. Grundsätzlich ist dieses gemeint, wenn über Voronoi-Diagramme ohne Angabe weiterer Attribute diskutiert wird.

2.5.1 Voronoi-Diagramm $\mathcal{VD}(\mathcal{S})$

Bei der Definition von $\mathcal{VD}(\mathcal{S})$ legen wir endliche Punktmengen zugrunde, deren Elemente als *Orte* (*sites*) bezeichnet werden. Die Bisektoren $B(p_i, p_j)$ für $p_i, p_j \in \mathbb{R}^2, j \in \{1, ..., n\}, j \neq i$ teilen den $\mathbb{R}^2$ in offene Halbebenen $D(p_i, p_j)$ und $D(p_j, p_i)$ und bestehen aus den Punkten, die zu p_i und p_j denselben Abstand haben. $D(p_i, p_j)$ wird von p_i und $D(p_j, p_i)$ von p_j dominiert.

Definition 2.10 (Voronoi-Diagramm $\mathcal{VD}(\mathcal{S})$ im $\mathbb{R}^2$)
Sei S eine n-elementige Menge von Orten p_i mit $i \in \{1, ..., n\}$ im $\mathbb{R}^2$.
Als Voronoi-Region $VR(p_i, S)$ des Ortes p_i bezüglich S bezeichnen wir folgende Menge

$$VR(p_i, S) = \{r \in \mathbb{R}^2; |rp_i| < |rp_j| \forall j \in \{1, ..., n\}, j \neq i\}$$
$$= \bigcap_{p_j \in S \setminus \{p_i\}} D(p_i, p_j)$$

mit

$$D(p_i, p_j) = \{r \in \mathbb{R}^2; |rp_i| < |rp_j|\}.$$

Als Durchschnitt von offenen Halbebenen ist $VR(p_i, S)$ offen und konvex, aber nicht notwendig beschränkt.

Als Voronoi-Diagramm $\mathcal{VD}(\mathcal{S})$ bezeichnet man die Menge

$$\mathcal{VD}(\mathcal{S}) = \mathbb{R}^2 \setminus \bigcup_{p_i \in S} VR(p_i, S).$$

$\mathcal{VD}(\mathcal{S})$ enthält also alle Punkte des $\mathbb{R}^2$, die zu mehr als einem Ort exakt denselben Abstand haben, welcher kleiner ist, als die Abstände zu allen anderen Orten. Es besteht aus Teilmengen der Bisektoren $B(p_i, p_j)$, den *Voronoi-Kanten*.

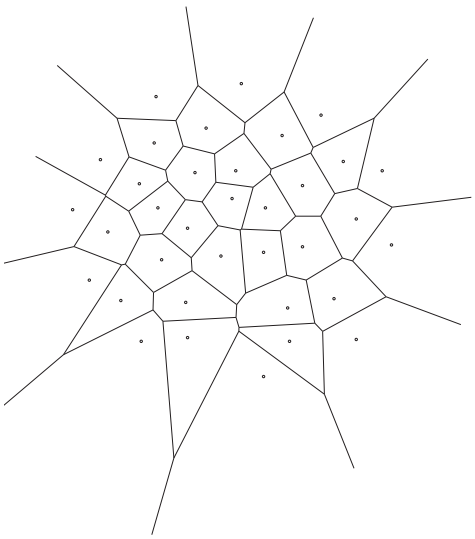
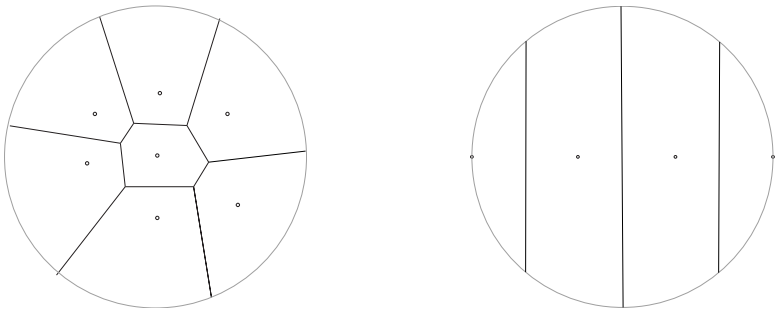


Abbildung 2.6: Voronoi-Diagramm $\mathcal{VD}(\mathcal{S})$

Sei $\partial(VR(p_i, S))$ der Rand der Voronoi-Region des Ortes p_i, dann heißt die Menge $VR(p_i, S) \bigcup \partial(VR(p_i, S))$ die *abgeschlossene* Voronoi-Region von p_i. *Voronoi-Knoten* entstehen aus dem Schnitt von mindestens drei abgeschlossenen Voronoi-Regionen und Voronoi-Kanten aus dem Schnitt von zwei abgeschlossenen Voronoi-Regionen.

Das Voronoi-Diagramm in der Ebene hat graphähnlichen Charakter und unter der Voraussetzung, dass die Orte p_i nicht kollinear angeordnet sind, ist es zusammenhängend.

Sind alle Orte p_i kollinear angeordnet, besteht es aus genau $n - 1$ Bisektoren. Um nicht zusammenhängende Voronoi-Diagramme zu vermeiden, legt man um das Diagramm einen Kreis mit hinreichend großem, endlichen Radius, der nur die unbeschränkten Regionen schneidet und deshalb alle Voronoi-Knoten umschließt. Dadurch werden „künstliche“ Voronoi-Knoten und -Kanten erzeugt und die Kanten außerhalb des Kreises werden „abgeschnitten“. Zusammenhängende Voronoi-Diagramme können als kreuzungsfreie geometrische Graphen aufgefasst werden, falls sie auch im Fall der Nicht-Kollinearität um diesen Kreis ergänzt werden und besitzen deren Charakteristika.



(a) nicht-kollineare Orte (b) kollineare Orte

Abbildung 2.7: $\mathcal{VD}(\mathcal{S})$ mit umschließenden Kreis

Satz 2.2 (Anzahl der Knoten und Kanten des $\mathcal{VD}(\mathcal{S})$)
Sei S eine n-elementige Menge im $\mathbb{R}^2$ und $\mathcal{VD}(\mathcal{S})$ zusammenhängend. Dann hat $\mathcal{VD}(\mathcal{S})$ $O(n)$ viele Knoten und Kanten.

Beweis:
Wir können $\mathcal{VD}(\mathcal{S})$ als zusammenhängenden kreuzungsfreien geometrischen Graphen mit genau $n+1$ Flächen auffassen. n Flächen entsprechen den n Voronoi-Regionen, eine Fläche entsteht durch das zusätzliche Gebiet außerhalb des Kreises. Jeder Voronoi-Knoten hat mindestens den Grad 3, da er durch den Schnitt von mindestens drei abgeschlossenen Voronoi-Regionen definiert ist oder auf dem Kreis mit dem Schnittpunkt einer „ursprünglichen“ Kante liegt. Nach der *Eulerschen Formel* gilt $|v| - |e| + |f| = 2$. Da $|f| = n+1 \Rightarrow |v| = |e| - n + 1$. Nach Theorem 2.1 auf Seite 10 gilt zudem $|v| \leq \frac{2}{3}|e|$. Also ist $|e| - n + 1 \leq \frac{2}{3}|e|$, woraus folgt $|e| \leq 3n - 3$ und $|v| \leq 2n - 2$. Damit ist gezeigt, dass die Anzahl der Knoten und Kanten in der Größenordnung $O(n)$ liegt. □

Wir gehen auf weitere allgemeine Eigenschaften des $\mathcal{VD}(\mathcal{S})$ ein, ohne die zugehörigen Beweise anzuführen. Diese können in [Kle97] nachvollzogen werden.

Satz 2.3 (Eigenschaften des $\mathcal{VD}(\mathcal{S})$)
Sei S eine n-elementige Menge von Orten p_i mit $i \in \{1, ..., n\}$ im $\mathbb{R}^2$ und $\mathcal{VD}(\mathcal{S})$ gegeben. Seien p_j Orte des $\mathcal{VD}(\mathcal{S})$ für $j \in \{1, ...n\}$ mit $j \neq i$.

1. *$p_i \in VR(p_i, S)$*
2. *$\forall p_i$ gilt $VR(p_i, S) \neq \emptyset$. Im Fall der Kollinearität sind alle $VR(p_i, S)$ unbeschränkt.*
3. *Sei $x \in \mathbb{R}^2, C(x)$ ein sich um x ausbreitender Kreis. Dann gilt:*
   (a) *$C(x)$ trifft zuerst genau auf einen Ort $p_i \Leftrightarrow$ $x \in VR(p_i, S)$ mit $x \neq p_i$.*

   (b) *$C(x)$ trifft zuerst genau auf zwei Orte $p_i, p_j \Leftrightarrow$ x liegt auf der Voronoi-Kante zwischen $VR(p_i, S)$ und $VR(p_j, S)$.*

   (c) *$C(x)$ trifft zuerst genau auf $p_1, ..., p_i$ mit $i \geq 3 \Leftrightarrow$ x ist ein Voronoi-Knoten mit den angrenzenden Voronoi-Regionen von $p_1, ..., p_i$.*

4. *$VR(p_i, S)$ ist unbeschränkt $\Leftrightarrow p_i \in \partial(ch(S))$*

5. *$VR(p_j, S)$ jedes nächsten Nachbarorts p_j von p_i hat eine gemeinsame Kante mit $VR(p_i, S)$.*

Die erste und die zweite Aussage ergeben sich aus der Definition von Voronoi-Regionen. Die Bestimmung der nächsten Nachbarn sowohl der Orte untereinander als auch beliebiger Punkte im $\mathbb{R}^2$ zu den Orten kann Mithilfe der strukturellen Gegebenheiten vorgenommen werden (siehe dritte und fünfte Aussage). Voronoi-Knoten sind eindeutig durch einen Kreis, dessen Rand die drei benachbarten Orte enthält, festgelegt. Befinden sich mehrere Orte in kozirkularer Lage[4], dann kann der Kreisrand mehr als drei Orte enthalten. Der festgelegte Voronoi-Knoten entspricht dem Mittelpunkt des Kreises und innerhalb des Kreises befinden sich keine anderen Orte. Auf den Zusammenhang zwischen der konvexen Hülle und dem $\mathcal{VD}(\mathcal{S})$ geht die vierte Eigenschaft ein und impliziert, dass sich die konvexe Hülle einer Punktmenge S von $\mathcal{VD}(\mathcal{S})$ ableiten lässt.

Der Nutzen der Eigenschaften des $\mathcal{VD}(\mathcal{S})$ zeigt sich insbesondere bei der Lösung von Distanzproblemen. Der Mittelpunkt des größten leeren Kreises zu einer Menge von Orten befindet sich immer entweder auf dem Voronoi-Diagramm oder in einer unbeschränkten Voronoi-Region abhängig von den gegebenen Voraussetzungen. Dieser Sachverhalt wird in Kapitel 3.1 vertieft.

Verallgemeinerungen des $\mathcal{VD}(\mathcal{S})$ treten in vielen verschiedenen Formen auf. Da wäre die Erweiterung auf höherdimensionale Räume, das Voronoi-Diagramm von Liniensegmenten, Voronoi-Diagramme zu verschiedenen Metriken, abstrakte Voronoi-Diagramme mit Bisektorkurven, Power-Diagramme, etc. Für weiterführende Informationen sei an dieser Stelle auf [Kle89] und [AK96] hingewiesen.

Eine noch nicht erwähnte Verallgemeinerung beinhaltet die Definition der Voronoi-Regionen nach den k nächsten Nachbarn aus S für $1 \leq k \leq n-1$, die sogenannten Voronoi-Diagramme höherer Ordnung. Das $\mathcal{VD}(\mathcal{S})$ stellt mit seinen Merkmalen das Voronoi-Diagramm erster Ordnung dar. Das Voronoi-Diagramm der Ordnung $n-1$ wird als *furthest-point* Voronoi-Diagramm $\mathcal{FVD}(\mathcal{S})$ bezeichnet und dessen Voronoi-Regionen enthalten die Punkte, die den gleichen entferntesten Nachbarn in S besitzen.

[4] Punkte einer endlichen Teilmenge des $\mathbb{R}^2$ sind kozirkular angeordnet, wenn mindestens 4 Punkte auf einem Kreis liegen.

2.5.2 furthest-point Voronoi-Diagramm $\mathcal{FVD}(\mathcal{S})$

Als Spezialfälle der Voronoi-Diagramme k-ter Ordnung besitzen das $\mathcal{FVD}(\mathcal{S})$ und das $\mathcal{VD}(\mathcal{S})$ eine analoge Definition, die auf der Unterteilung des $\mathbb{R}^2$ durch die Menge der Bisektoren zwischen allen p_i aufbaut. Im Fall des $\mathcal{VD}(\mathcal{S})$ wird die offene Halbebene $D(p_i, p_j)$ von p_i dominiert und enthält alle Punkte, die näher zu p_i als zu p_j liegen. Auch bei der Definition von $\mathcal{FVD}(\mathcal{S})$ werden offene Halbebenen $D_F(p_i, p_j)$ von p_i dominiert, doch sie enthalten alle Punkte, die sich weiter entfernt von p_i als von p_j befinden. $D_F(p_i, p_j)$ entspricht also $D(p_j, p_i)$. Aufgrund dieser Gegebenheit können die Charakteristika des $\mathcal{VD}(\mathcal{S})$ mit entsprechenden Anpassungen auf $\mathcal{FVD}(\mathcal{S})$ übertragen werden.

Definition 2.11 (furthest-point Voronoi-Diagramm $\mathcal{FVD}(\mathcal{S})$ im $\mathbb{R}^2$)
Sei S eine n-elementige Menge von Orten p_i mit $i \in \{1, ..., n\}$ im $\mathbb{R}^2$.
Als Voronoi-Region $VR_F(p_i, S)$ des Ortes p_i bezüglich S bezeichnen wir folgende Menge

$$VR_F(p_i, S) = \{r \in \mathbb{R}^2; |rp_i| > |rp_j| \ \forall j \in \{1, ..., n\}, j \neq i\}$$

$$= \bigcap_{p_j \in S \setminus \{p_i\}} D_F(p_i, p_j)$$

mit

$$D_F(p_i, p_j) = \{r \in \mathbb{R}^2; |rp_i| > |rp_j|\}.$$

Als Durchschnitt von offenen Halbebenen ist $VR_F(p_i, S)$ offen und konvex und falls $VR_F(p_i, S) \neq \emptyset$, ist $VR_F(p_i, S)$ unbeschränkt.
Das furthest-point Voronoi-Diagramm $\mathcal{FVD}(\mathcal{S})$ wird durch

$$\mathcal{FVD}(\mathcal{S}) = \mathbb{R}^2 \setminus \bigcup_{p_i \in S} VR_F(p_i, S)$$

festgelegt.

Obwohl die Definition des furthest-point Voronoi-Diagramms der Definition des klassischen Voronoi-Diagramms sehr ähnelt, ist die Darstellung und Struktur komplett verschieden.

Das Voronoi-Diagramm $\mathcal{FVD}(\mathcal{S})$ enthält alle Punkte des $\mathbb{R}^2$, die mehr als einen Ort als entferntesten Nachbarn haben. Auch hier entstehen die Voronoi-Kanten und -Knoten aus dem Schnitt von zwei bzw. drei abgeschlossenen Voronoi-Regionen $VR_F(p_i, S) \bigcup \partial(VR_F(p_i, S))$ als Teilmengen der Bisektoren $B(p_i, p_j)$. Das $\mathcal{FVD}(\mathcal{S})$ hat in der euklidischen Ebene graphähnlichen Charakter und erfüllt zusätzlich die Bedingung, ausschließlich unbeschränkte Flächen zu separieren und ist somit ein Baum. Im Fall der Kollinearität besteht es nur aus dem Bisektor zwischen den beiden Punkten mit der größten Distanz zueinander. Damit kann die Anzahl der Knoten und Kanten des $\mathcal{FVD}(\mathcal{S})$ analog zu der Anzahl der Knoten und Kanten des $\mathcal{VD}(\mathcal{S})$ abgeschätzt werden.

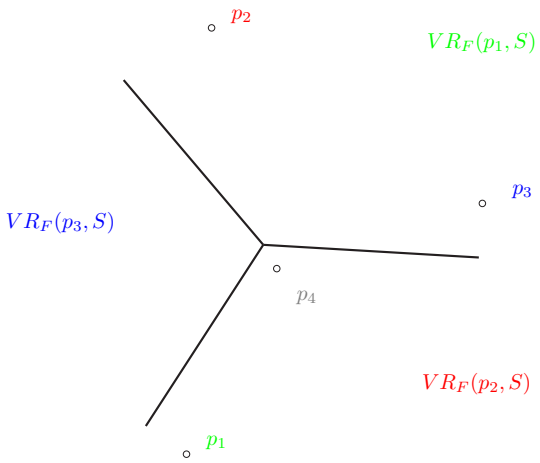


Abbildung 2.8: furthest-point Voronoi-Diagramm $\mathcal{FVD}(\mathcal{S})$

Satz 2.4 (Anzahl der Knoten und Kanten des $\mathcal{FVD}(\mathcal{S})$)
Sei S eine n-elementige Menge im $\mathbb{R}^2$. Dann hat $\mathcal{FVD}(\mathcal{S})$ $O(n)$ viele Knoten und Kanten.

Beweis:
Für den Fall, dass alle Orte $p_1, ..., p_n \in S$ kollinear in der Ebene liegen, kann das zugrundeliegende Koordinatensystem so gedreht werden, dass die Gerade durch die Orte der x-Achse entspricht. Dann gilt o.B.d.A. $p_1 < p_2 < ... < p_n$. Für $m > k$ folgt dadurch $D_F(p_1, p_k) \supset D_F(p_1, p_m)$ und $D_F(p_n, p_k) \subset D_F(p_n, p_m)$. Also entspricht $VR_F(p_1, S)$ dem Halbraum $D_F(p_1, p_n)$ und $VR_F(p_n, S)$ dem Halbraum $D_F(p_n, p_1)$. Da für jedes p_k $D_F(p_k, p_1) \cap D_F(p_k, p_n) = \emptyset$ zutrifft, ist $VR_F(p_k, S) = \emptyset$. $\mathcal{FVD}(\mathcal{S})$ besteht nur aus einer Geraden und zwar dem Bisektor $B(p_1, p_n)$ und weist keine Knoten auf.

Im allgemeinen Fall setzen wir einen Kreis um das $\mathcal{FVD}(\mathcal{S})$ ein, der alle Voronoi-Knoten enthält und nur die Strahlen aber kein Liniensegment schneidet. Dadurch erhält man einen Graphen, dessen Knoten mindestens den Grad 3 haben aufgrund ihrer Festlegung als Durchschnitt von mindestens drei Voronoi-Regionen VR_F oder als Schnittpunkt des Kreises mit den „ursprünglichen“ Voronoi-Kanten. Da die Anzahl der Voronoi-Regionen höchstens n entspricht, können wir hier analog zum $\mathcal{VD}(\mathcal{S})$ die *Eulersche Formel* auf das $\mathcal{FVD}(\mathcal{S})$ anwenden. □

An dieser Stelle kommen wir zu den allgemeinen Eigenschaften des $\mathcal{FVD}(\mathcal{S})$. Die Reihenfolge der Aufzählung entspricht der Reihenfolge, in der die analogen Eigenschaften des $\mathcal{VD}(\mathcal{S})$ angegeben wurden.

Satz 2.5 (Eigenschaften des $\mathcal{FVD}(\mathcal{S})$)
Sei S eine n-elementige Menge von Orten p_i mit $i \in \{1, ..., n\}$ im $\mathbb{R}^2$ und $\mathcal{FVD}(\mathcal{S})$ gegeben. Seien p_j Orte des $\mathcal{FVD}(\mathcal{S})$ für $j \in \{1, ...n\}$ mit $j \neq i$.

1. *$p_i \notin VR_F(p_i, S)$ und im Fall, dass $VR_F(p_i, S) \neq \emptyset$ liegt p_i seiner zugehörigen Voronoi-Region diametral*[5] *gegenüber.*

2. *Es können p_i mit $VR_F(p_i, S) = \emptyset$ existieren und es gilt, $VR_F(p_i, S) \neq \emptyset \Leftrightarrow VR_F(p_i, S)$ ist unbeschränkt. Daraus folgt, dass $\mathcal{FVD}(\mathcal{S})$ eine Baumstrukur ist.*

3. *Sei $x \in \mathbb{R}^2$, $C(x)$ ein sich um x ausbreitender Kreis. Dann gilt:*

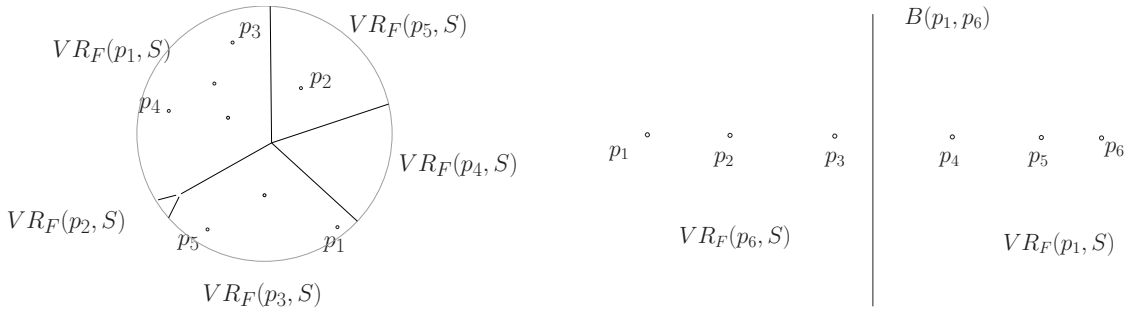
   (a) *$C(x)$ trifft zuletzt genau auf einen Ort $p_i \Leftrightarrow$ $x \in VR_F(p_i, S)$.*

   (b) *$C(x)$ trifft zuletzt genau auf zwei Orte $p_i, p_j \Leftrightarrow$ x liegt auf der Voronoi-Kante zwischen $VR_F(p_i, S)$ und $VR_F(p_j, S)$.*

   (c) *$C(x)$ trifft zuletzt genau auf $p_1, ..., p_i$ mit $i \geq 3 \Leftrightarrow$ x ist ein Voronoi-Knoten mit den angrenzenden Voronoi-Regionen von $p_1, ..., p_i$.*

4. *Unter der Annahme, dass keine drei Punkte auf dem Rand der konvexen Hülle kollinear sind, gilt: $VR_F(p_i, S) \neq \emptyset \Leftrightarrow p_i \in \partial(ch(S))$*

5. *Für jeden Nachbarort $p_j \in \partial(ch(S))$ von $p_i \in \partial(ch(S))$ haben die beiden zugehörigen Voronoi-Regionen eine gemeinsame Kante.*



(a) nicht-kollineare Orte (b) kollineare Orte

Abbildung 2.9: furthest-point Voronoi-Diagramme $\mathcal{FVD}(\mathcal{S})$

[5] „diametral gelegen“ bedeutet hier, dass die Punkte aus $VR_F(p_i, S)$ maximalen Abstand zu p_i haben.

Beweis:

1. Dass p_i nicht in $VR_F(p_i, S)$ enthalten sein kann und die Punkte innerhalb der zugehörigen Voronoi-Region maximalen Abstand zu p_i haben, ergibt sich direkt aus der Definition von $VR_F(p_i, S)$.

2. Für den Beweis der Existenz von Orten, deren Voronoi-Regionen der leeren Menge entsprechen, reicht das Angeben eines Beispiels. Im Fall der Kollinearität der Orte wurde bereits gezeigt, dass das $\mathcal{FVD}(\mathcal{S})$ nur aus dem Bisektor der Orte mit dem maximalen Abstand besteht und dass die Voronoi-Regionen der anderen Orte gleich der leeren Menge sind. Ein weiteres Beispiel stellt das $\mathcal{FVD}(\mathcal{S})$ in der Abbildung 2.8 auf Seite 19 dar, das zu vier Orten nur drei Regionen berandet.

   Die Äquivalenz zwischen der Existenz von Voronoi-Regionen und ihrer Unbeschränktheit zeigen wir in zwei Schritten.
   „$\Leftarrow$“: Wenn $VR_F(p_i, S)$ unbeschränkt ist, gibt es mindestens einen Strahl $g \in VR_F(p_i, S)$ mit einem Startpunkt $x \in VR_F(p_i, S)$. Damit ist $VR_F(p_i, S) \neq \emptyset$.
   „$\Rightarrow$“: Sei $VR_F(p_i, S) \neq \emptyset$ und $x \in VR_F(p_i, S)$. Dann ist $|xp_i|$ das Maximum aller Abstände $|xp_j|$ für $j \in \{1, ..., n\}$. Sei g ein Strahl, der in x als Startpunkt beginnt und in Richtung von p_i nach x läuft. Jeder beliebige Punkt $y \in g$ hat wiederum die maximale Distanz zu p_i, da $|p_iy| = |p_ix| + |xy|$ und für $j \neq i$ gilt $|p_ix| + |xy| > |p_jx| + |xy| \geq |p_jy|$. Also gilt $y \in VR_F(p_i, S)$, g liegt komplett in $VR_F(p_i, S)$ und $VR_F(p_i, S)$ ist unbeschränkt. Da nur unbeschränkte Flächen auftreten, haben wir definitionsgemäß eine Baumstruktur gegeben.

3. Die Kriterien für die Lage der $x \in \mathbb{R}^2$ ergeben sich aus der Definition der Voronoi-Regionen und der Festlegung des $\mathcal{FVD}(\mathcal{S})$ anhand des Durchschnitts der $D_F(p_i, p_j)$.

4. Da $VR_F(p_i, S) \neq \emptyset$ äquivalent zu der Unbeschränktheit von $VR_F(p_i, s)$ gilt, sei bei dieser Aussage auf den Beweis zum Zusammenhang zwischen der konvexen Hülle und den unbeschränkten $VR(p_i, S)$ von $\mathcal{VD}(\mathcal{S})$ in [Kle97] verwiesen.

5. Da p_i und p_j auf dem Rand der konvexen Hülle liegen, existiert zu beiden Orten jeweils eine Voronoi-Region. Es bleibt zu zeigen, dass $VR_F(p_i, S)$ und $VR_F(p_j, S)$ eine gemeinsame Kante besitzen.
   $p_j \in \partial(ch(S))$ ist laut Voraussetzung ein Nachbarort von $p_i \in \partial(ch(S))$. Dann haben p_i und p_j eine gemeinsame Kante auf dem Rand der konvexen Hülle und induzieren eine Stützgerade von $ch(S)$. Sei $x \in B(p_i, p_j)$ und wandere in h^+ in Richtung unendlich. Der Abstand eines beliebigen Ortes p_l berechnet sich aus seinem Lotpunkt z_l auf dem Bisektor zu $\sqrt{|p_lz_l|^2 + |z_lx|^2}$. Sobald x den Rand der konvexen Hülle verlassen hat, kann der Abstand zwischen p_i (bzw. p_j) und x durch $\sqrt{|p_iz_i|^2 + (|z_iz_l| + |z_lx|)^2}$ für $l \neq i, j$ berechnet werden.

Vergleicht man die Quadrate der beiden Abstände $|p_i x|$ und $|p_l x|$, ergibt sich die folgende Gleichung.

$$\begin{aligned} g(p_l, x) &:= |p_i z_i|^2 + (|z_i z_l| + |z_l x|)^2 - (|p_l z_l|^2 + |z_l x|^2) \\ &= (|p_i z_i|^2 + |z_i z_l|^2 - |p_l z_l|^2) + 2|z_i z_l||z_l x| \end{aligned}$$

Da die Menge S endlich ist und alle Summanden bis auf $2|z_i z_l||z_l x|$ konstant sind, gibt es einen Abstand $|z_l x_l|$, so dass $g(p_l, x_l) > 0$. Also gibt es ein x_0 für $p_l \in S \notin \{p_i, p_j\}$, so dass $g(p_l, x) > 0$ für alle $|p_l x| \geq |p_l x_0|$. Das heißt, jeder Kreis um ein Zentrum $x \in B(p_i, p_j)$ für $|p_l x| \geq |p_l x_0|$, dessen Kreisrand durch p_i und p_j führt, enthält alle Orte aus S. Damit liegt x auf der Voronoi-Kante $\partial(VR_F(p_i, S)) \cap \partial(VR_F(p_j, S))$ in $\mathcal{FVD}(\mathcal{S})$.

□

Die eindeutige Festlegung eines Voronoi-Knoten durch Orte auf einem Kreis gilt sowohl für $\mathcal{FVD}(\mathcal{S})$ als auch für $\mathcal{VD}(\mathcal{S})$. Im Fall von $\mathcal{FVD}(\mathcal{S})$ liegen alle Orte aus S in diesem Kreis, wogegen im Fall von $\mathcal{VD}(\mathcal{S})$ kein Ort aus S in dem Inneren des Kreises liegt.

Die allgemeinen Eigenschaften des $\mathcal{FVD}(\mathcal{S})$ belegen, dass auch dieses Voronoi-Diagramm für die Lösung von Distanzproblemen geeignet ist. Ein Beispiel ist das *Diameter-Problem*, bei dem der Durchmesser einer n-elementigen Punktmenge im $\mathbb{R}^d$ gesucht wird, (siehe [Bes01] für den dreidimensionalen Fall). Die damit verwandte Aufgabenstellung der Ermittlung des kleinsten umfassenden Kreises ist Thema des Kapitels 3.2.

2.6 Delaunay-Triangulationen

Der russische Mathematiker Delone gab mit seiner Veröffentlichung im Jahr 1934 [Del34] der klassischen Delaunay-Triangulation[6] ihren Namen. Sie stellt in der euklidischen Ebene den dualen Graphen des Voronoi-Diagramms $\mathcal{VD}(\mathcal{S})$ dar und teilt die konvexe Hülle einer endlichen Punktmenge in Dreiecke auf. Die Dualität lässt sich auf höhere Dimensionen übertragen und impliziert, dass sich eine Delaunay-Triangulation eindeutig aus dem Voronoi-Diagramm ableiten lässt und umgekehrt. Unter anderen Metriken als der euklidischen ist weder diese Eindeutigkeit noch die Existenz der Delaunay-Triangulation zwangsläufig gegeben.

Allgemein gesehen sind Triangulationen einer Punktmenge S Zerlegungen von S in Dreiecke.

[6] „Delaunay“ als französische Transliteration seines russischen Namens hat sich im Gegensatz zu englischen Übersetzung „Delone“ durchgesetzt, da seine Arbeit in Französisch veröffentlicht wurde. Dies war neben Deutsch die zu jener Zeit übliche Wissenschaftssprache.

Definition 2.12 (Triangulation einer Punktmenge im $\mathbb{R}^2$)
Sei $S \subset \mathbb{R}^2$ eine n-elementige Menge, deren Punkte nicht alle kollinear liegen. Unter einer Zerlegung $T(S)$ von S verstehen wir eine Menge von konvexen Polygonen $P_1, ..., P_m$ aus definierenden Punkten aus S, für die gilt:

1. *$int(P_i) \cap int(P_j) = \emptyset$ für alle $i \neq j$*
2. *$P_i \cap P_j$ ist entweder ein Liniensegment, ein Punkt oder die leere Menge für alle $i \neq j$*
3. *$ch(S) = \bigcup_{1 \leq i \leq m} P_i$.*

Sind alle P_i Dreiecke, heißt die Zerlegung $T(S)$ Triangulation von S.

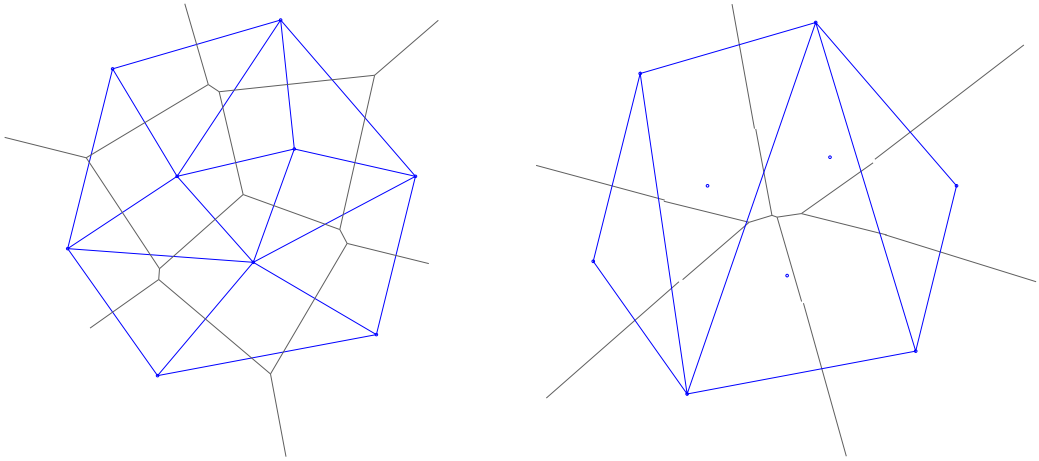
Es gibt endlich viele verschiedene Triangulationen einer endlichen Menge S im $\mathbb{R}^2$. Diese unterscheiden sich nicht anhand der Anzahl der Dreiecke, jedoch anhand der Lage der Dreiecke. Der Rand der konvexen Hülle ist immer in den Kanten der Dreiecke enthalten. Liegen alle Punkte aus S auf $\partial(ch(S))$, entspricht eine Triangulation der Punktmenge einer Triangulation des konvexen Polygons $ch(S)$.

Satz 2.6 (Anzahl der Dreiecke einer Triangulation im R^2)
Sei S eine n-elementige Punktmenge im $\mathbb{R}^2$, die nicht alle kollinear liegen. Dann ist die Anzahl der Dreiecke aller Triangulationen von S in $O(n)$.

Neben der klassischen Delaunay-Triangulation $\mathcal{DT}(\mathcal{S})$ führen wir an dieser Stelle auch die *furthest-point* Delaunay-Triangulation $\mathcal{FDT}(\mathcal{S})$ im $\mathbb{R}^2$ ein. Wir nehmen an, dass die Orte sich in *allgemeiner Lage* befinden, also dass sie weder kollinear noch kozirkular angeordnet sind.

Definition 2.13 (Delaunay-Triangulation im $\mathbb{R}^2$)
Sei S eine endliche Teilmenge des $\mathbb{R}^2$ in allgemeiner Lage.
Dann ist eine Triangulation von S genau dann eine Delaunay-Triangulation $\mathcal{DT}(\mathcal{S})$, wenn die definierenden Punkte der Dreiecke jeweils in dem Rand eines Kreises enthalten sind und kein Punkt aus S innerhalb dieses Kreises liegt.
Sei $M = \partial(ch(S)) \bigcap S$. Eine Triangulation von M ist genau dann eine furthest-point Delaunay-Triangulation $\mathcal{FDT}(\mathcal{S})$, wenn die definierenden Punkte der Dreiecke jeweils in dem Rand eines Kreises enthalten sind und kein Punkt aus S außerhalb dieses Kreises liegt.

Sowohl $\mathcal{DT}(\mathcal{S})$ als auch $\mathcal{FDT}(\mathcal{S})$ im $\mathbb{R}^2$ lassen sich also eindeutig anhand von Kreisen definieren. Wenn sich mehr als drei Punkte aus S in kozirkularer Lage befinden, entsteht eine Delaunay-Zerlegung in Polygone, die sogenannte Delaunay-Vortriangulation. Die entstehenden Polygone können wiederum in Dreiecke aufgeteilt werden und führen damit zu einer Delaunay-Triangulation. $\mathcal{DT}(\mathcal{S})$ stellt eine Triangulation der Punktmenge S, $\mathcal{FDT}(\mathcal{S})$ eine Triangulation des konvexen Polygons $ch(S)$ dar.



(a) $\mathcal{DT}(\mathcal{S})$ (b) $\mathcal{FDT}(\mathcal{S})$

Abbildung 2.10: Delaunay-Triangulationen mit ihren dualen Voronoi-Diagrammen

Für die Beweise der folgenden Eigenschaften für $\mathcal{DT}(\mathcal{S})$ sei auf [Kle97] hingewiesen. Für $\mathcal{FDT}(\mathcal{S})$ können diese analog geführt werden.

Satz 2.7 (Eigenschaften der Delaunay-Triangulation im $\mathbb{R}^2$)
Sei S eine endliche Menge im $\mathbb{R}^2$ in allgemeiner Lage und $M = \partial(ch(S)) \bigcap S$. $\mathcal{DT}(\mathcal{S})$ und $\mathcal{FDT}(\mathcal{S})$ seien gegebene Delaunay-Triangulationen.

1. *$\mathcal{DT}(\mathcal{S})$ bzw. $\mathcal{FDT}(\mathcal{S})$ ist die duale Struktur zum $\mathcal{VD}(\mathcal{S})$ bzw. $\mathcal{FVD}(\mathcal{S})$ und eindeutig bestimmt.*

2. *$\partial(ch(S))$ ist in den Kanten von $\mathcal{DT}(\mathcal{S})$ bzw. $\mathcal{FDT}(\mathcal{S})$ enthalten.*

3. *Sei $S = M$ und die Anzahl von $S > 3$. Dann enthält $\mathcal{FDT}(\mathcal{S})$ kein Dreieck aus $\mathcal{DT}(\mathcal{S})$ und umgekehrt.*

Die erste Aussage betrifft den Zusammenhang der Delaunay-Triangulationen zu Voronoi-Diagrammen. $\mathcal{DT}(\mathcal{S})$ ist zudem eine kreuzungsfreie geometrische Realisierung des dualen Graphen des Voronoi-Diagramms $\mathcal{VD}(\mathcal{S})$. Es werden je zwei Orte $p_i, p_j \in S$ mit einer Kante verbunden, deren Voronoi-Regionen eine gemeinsame Kante besitzen. Diese Vorschrift lässt sich auch auf $\mathcal{FDT}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ übertragen, allerdings entsteht dadurch kein dualer Graph und nur die Orte aus M werden trianguliert. Da in den Voronoi-Regionen kein Ort oder auch mehrere Orte auftreten können, werden bei der Konstruktion nicht Knoten aus verschiedenen Flächen mit einer Kante verbunden, sondern durchaus aus gleichen Flächen. Dies widerspricht der Festlegung von dualen Graphen und so wird an dieser Stelle von dualen Strukturen gesprochen. Bei der Dualität nehmen die Voronoi-Knoten in beiden Fällen eine interessante Rolle ein. Sie entsprechen den Mittelpunkten der Kreise, die laut Definition der Delaunay-Triangulation durch jeweils drei Punkte eines Dreiecks führen. Für $\mathcal{DT}(\mathcal{S})$ enthalten sie keine Orte und für $\mathcal{FDT}(\mathcal{S})$ alle Orte.

Die Aussage über den Zusammenhang zur konvexen Hülle impliziert, dass keine Kante der Triangulation außerhalb der konvexen Hülle verläuft. Dies führt dazu, dass bei der Konstruktion innere Kanten immer anders als die feststehenden Randkanten behandelt werden müssen. Während einige Dreiecke von $\mathcal{DT}(\mathcal{S})$ durchaus in Triangulationen vorkommen können, die nicht identisch mit $\mathcal{DT}(\mathcal{S})$ sind, trifft dies für $\mathcal{FDT}(\mathcal{S})$ niemals zu, was sich aus der Definition von beiden ergibt. Nur für den Fall, dass S ausschließlich aus kozirkularen Orten besteht, kann das Erscheinungsbild von $\mathcal{DT}(\mathcal{S})$ und $\mathcal{FDT}(\mathcal{S})$ übereinstimmen, da in diesem Fall beide Kriterien für die Kreise erfüllt sind. Alle Orte liegen auf dem Rand des Kreises, der die Dreiecke umschließt, und keiner innerhalb oder außerhalb.

Delaunay-Triangulationen nehmen einen wichtigen Platz bei der Approximation von räumlichen Flächen ein, wie die Beispiele in [AM91] und [Klö97] belegen. Neben diesen Anwendungen werden sie ebenso in vielen Bereichen eingesetzt, die aus der Verwandtschaft zu Voronoi-Diagrammen resultieren. So benutzen viele Implementationen, die die Konstruktion von Voronoi-Diagrammen animieren, die Delaunay-Triangulation als grundlegende Datenstruktur, aus der Voronoi-Diagramme in Linearzeit ermittelt werden können.

2.7 Exakte Arithmetik und Sonderfälle

Im Gegensatz zu der theoretischen Betrachtung von Algorithmen und ihrer Ausführung auf der REAL RAM stehen in der Praxis keine reellen Zahlen zur Verfügung, sondern Gleitkommazahlen mit endlicher Genauigkeit. Exakte Arithmetik kann auf einem Rechner nur bis zu der Genauigkeit rationaler Zahlen erreicht werden, wobei gerade bei geometrischen Problemstellungen mit irrationalen Zahlen gerechnet wird, wie zum Beispiel bei der Berechnung von Abständen. Werden Abstände wiederum nur verglichen, können unnötige Berechnungen vermieden werden, in dem nicht die exakten Ergebnisse, sondern die quadratischen Abstände herangezogen werden. Jede vermeidbare Berechnung sollte bei der Implementation weggelassen werden, um die Effizienz zu steigern und Fehlern vorzubeugen.

Wird ein sogenannter *Incircle-Test* benötigt, der entscheidet, ob ein Punkt innerhalb eines Kreises, auf dem Rand des Kreises oder außerhalb liegt, können Rundungsfehler zu einem verfälschten Ergebnis führen, wenn der Abstand dieses Punktes zum Kreisrand sehr klein ist. Bei der Konstruktion einer Delaunay-Triangulation zum Beispiel könnte dies zu einer inkorrekten Struktur führen. Ein so verfälschtes Ergebnis kann einen Abbruch eines Programms, das rein nach theoretischen Vorgaben implementiert wurde, oder eine Endlosschleife zur Folge haben oder liefert eine inkorrekte Ausgabe. Abhilfe wird zum Teil über Berechnungen von Determinanten von Matrizen geschaffen, über die Längen ermittelt werden (siehe [urlb] mit Verweisen auf dementsprechenden Programmcode) oder durch Einführung eigener Zahlenformate mit beliebiger, selbstanpassender Präzision wie in [She96], [She97]. Eine andere Strategie zur Behandlung von Rundungsfehlern wird in [dBvKOS97]

vorgestellt, bei dem abhängig von der Problemstellung entschieden wird, ob inexakte Ausgaben für die Anwendung genügen. Zu dieser Thematik im Zusammenhang mit robusten Algorithmen sind eine Vielzahl von Veröffentlichungen erschienen, die Beispiele exakter Arithmetik vorstellen und zum Teil auch den dadurch verursachten Performanceverlust. Während auf der REAL RAM auf reelle Zahlen in konstanter Zeit zugegriffen werden kann, wird in der Praxis schon beim Lesen einer m-Bit-Zahl $\Omega(\log m)$ Zeit verbraucht. Dies kann neben Umformatierungen in nicht konstanter Zeit bei theoretisch optimalen Algorithmen zu suboptimalen Implementationen führen.

Gehen wir davon aus, dass die Implementation mit hinreichend exakter Arithmetik für das zu lösende Problem rechnet, ist über die Handhabung von Sonderfällen bei der Eingabe zu entscheiden. Es gibt dafür mehrere Möglichkeiten, wobei die eleganteste die wäre, dass degenerierte[7] Eingaben schon implizit im Algorithmus verarbeitet werden können. Ist dieses nicht gegeben, kann jeder Sonderfall explizit abgefragt werden, was zu einer ansteigenden Komplexität der Implementation führt. Eine andere Alternative sind Pertubationstechniken, wie sie in [EM90] vorgestellt werden, die aus degenerierten Eingaben durch infinitesimal kleine Änderungen bei der Berechnung nicht-degenerierte Eingaben erzeugen und trotzdem zu der richtigen Lösung kommen.

Mit den Programmbibliotheken *LEDA* und *CGAL* in C++ (siehe [urla]) werden robuste Module zur Berechnung von Aufgaben aus der algorithmischen Geometrie unter Beachtung der vorgestellten Problematik angeboten. Können bestehende Module nicht genutzt werden, sind die anzuwendenden Maßnahmen für einen robusten und möglichst exakten Algorithmus von der Anwendung abhängig und werden dementsprechend vorgenommen. Darauf kommen wir bei der Beschreibung der Implementation von *FitCircle* zurück.

[7] Ein Beispiel für degenerierte Eingaben sind Punktmengen, die nicht in allgemeiner Lage angeordnet sind.

Das Weltall ist ein Kreis, dessen Mittelpunkt überall, dessen Umfang nirgends ist.
Blaise Pascal (1623 - 1662)

3

Verschiedene Optimierungen und Approximationen

Optimierungsaufgaben werden allgemein über einer Menge M aller möglichen Lösungen eines Problems definiert. Ziel ist es, eine reellwertige Funktion $f : M \to \mathbb{R}$ zu minimieren bzw. zu maximieren. Bei der geometrischen Optimierung handelt es sich bei den Elementen von M um geometrische Objekte, die bestimmte Bedingungen erfüllen. Betrachten wir beispielsweise den größten leeren Kreis einer Punktmenge $S \subset \mathbb{R}^2$, dessen Mittelpunkt in der konvexen Hülle von S enthalten sein soll. Dann entspricht die zu maximierende Funktion f dem Radius des Kreises, wenn M alle möglichen leeren Kreise, deren Mittelpunkte innerhalb der konvexen Hülle liegen, beinhaltet.

Approximationsaufgaben gehören zur Klasse der Optimierungsaufgaben und werden über einem metrischen Raum O festgelegt. $M \subset O$ sei die Menge der Elemente, mit denen approximiert wird. Für $x \in M$ soll die entsprechende Distanzfunktion $dist(x, y)$ mit $y \in O$ minimiert werden, wobei y das zu approximierende Element ist. Als Gütekriterium der Approximation wird im Allgemeinen die Distanz zwischen x und y genutzt. Die damit verwandte geometrische Approximation nähert geometrische Objekte an andere, meist einfachere, an und mit Hilfe der Kombinatorik können Aussagen über die Eigenschaften jener Objekte für bestimmte Problemstellungen getroffen werden.

Die Annäherung einer Punktmenge durch andere geometrische Objekte wie Geraden, Kreise, Ellipsen, Rechtecken gehört zu dem fundamentalen Bereich des *Shape-Fitting* der algorithmischen Geometrie. In der vorliegenden Arbeit werden wir uns auf Kreise spezialisieren und den $\mathbb{R}^2$ zugrundelegen. Intuitiv ist nachvollziehbar, dass Punktmengen existieren, die sich besser durch eine Gerade als einen Kreis annähern

lassen (zum Beispiel Punkte in kollinearer Lage). Dieser Fall kann als Sonderfall der Approximation durch einen Kreis aufgefasst werden, bei dem der Mittelpunkt des Kreises gegen unendlich verschoben wird.

Auf den Zusammenhang zwischen den strukturellen Eigenschaften von Voronoi-Diagrammen und Kreisen sind wir bei der Vorstellung des $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ bereits eingegangen. Dies werden wir im Folgenden vertiefen und Charakteristika beschreiben, die nützlich für die Erstellung entsprechender Algorithmen sind, die im nächsten Kapitel eingeführt werden. Der größte leere Kreis einer Punktmenge lässt sich über $\mathcal{VD}(\mathcal{S})$ und der kleinste umfassende Kreis über $\mathcal{FVD}(\mathcal{S})$ ermitteln. Der am besten angepasste Kreis als Annäherung einer Punktmenge kann unter verschiedenen Gesichtspunkten betrachtet werden. Da wären als Beispiele die Minimierung des maximalen Abstands der Punkte zu einem Kreis als auch die Minimierung der Summe aller Abstände zu einem Kreis. Die beiden Voronoi-Diagramme $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ können zur Ermittlung des am besten angepassten Kreises in dem erstgenannten Sinn dienen. Diese Problemstellung entspricht der Ermittlung des Rings mit minimaler Breite.

3.1 Größter leerer Kreis

Die Ermittlung des größten leeren Kreises oder auch *largest empty circle* $\mathcal{LEC}(\mathcal{S})$ zu einer Punktmenge S wird in der Literatur unter verschiedenen Nebenbedingungen untersucht. In [Kle97] wird als Nebenbedingung ein konvexes Polygon als unabhängiges Gebiet festgelegt, in dem sich der Mittelpunkt des Kreises befinden muss. Es wird keine Aussage über die Lage der Punkte aus S bezüglich des Polygons getroffen. Eine natürliche Grenze für die Festlegung des $\mathcal{LEC}(\mathcal{S})$ stellt die konvexe Hülle von S dar, wie es in [PS85] vorgestellt wird. Für die weitere Untersuchung nehmen wir ebenfalls an, dass sich der Kreismittelpunkt in der konvexen Hülle der Punktmenge befindet. Die Festlegung einer Grenze ist notwendig, da ansonsten unter der Maximierungsbedingung der Radius des Kreises gegen ∞ geht und der Kreis selbst sich einer Ebene annähert.

Definition 3.1 (Größter leerer Kreis $\mathcal{LEC}(\mathcal{S})$)
Sei S eine endliche Menge von Punkten im $\mathbb{R}^2$. Dann verstehen wir unter $\mathcal{LEC}(\mathcal{S})$ den größten Kreis, der keine Punkte aus S in seinem Inneren enthält und dessen Mittelpunkt in der konvexen Hülle von S liegt.

Aus der Definition ergeben sich zwei verschiedene Alternativen für die Lage des Mittelpunktes des $\mathcal{LEC}(\mathcal{S})$. Er liegt entweder auf dem Rand der konvexen Hülle von S oder im Innern dieser konvexen Hülle. Für die genaue Bestimmung können die strukturellen Eigenschaften eines $\mathcal{VD}(\mathcal{S})$ und seine Schnittpunkte mit dem Rand der konvexen Hülle genutzt werden. Auf die Existenz und Anzahl solcher Schnittpunkte gehen wir in dem nächsten Lemma ein.

Lemma 3.1 (Schnittpunkte des $\mathcal{VD}(\mathcal{S})$ mit $\partial(ch(S))$)
Sei S eine n-elementige Menge von Orten im $\mathbb{R}^2$. Gegeben seien $\mathcal{VD}(\mathcal{S})$ und $ch(S)$. Dann gilt:
Jede Kante der konvexen Hülle von S schneidet mindestens eine Voronoi-Kante. Jede Voronoi-Kante wiederum schneidet höchstens zwei Kanten der konvexen Hülle.

Beweis:
Die erste Aussage kann anhand der Definition des $\mathcal{VD}(\mathcal{S})$ nachgewiesen werden. Jede Voronoi-Region wird durch $\mathcal{VD}(\mathcal{S})$ von einer anderen Voronoi-Region „abgegrenzt“. Liegen die Orte p_i und p_j aus S benachbart auf dem Rand der konvexen Hülle, werden sie durch eine Kante e des konvexen Polygons $ch(S)$ verbunden. Da p_i in $VR(p_i, S)$ und p_j in $VR(p_j, S)$ enthalten ist, liegt in e mindestens ein Punkt aus $VR(p_i, S)$ und ein Punkt aus $VR(p_j, S)$. Dementsprechend besteht mindestens ein Schnittpunkt von e mit einer Voronoi-Kante.
Für den Beweis der zweiten Aussage nutzen wir die Eigenschaft, dass $ch(S)$ einem konvexen Polygon entspricht. Da jede Gerade höchstens zwei Kanten eines konvexen Polygons schneidet (siehe Kapitel 2.3 auf Seite 9), kann eine Voronoi-Kante als Teilmenge eines Bisektors höchstens zwei Kanten der konvexen Hülle schneiden. □

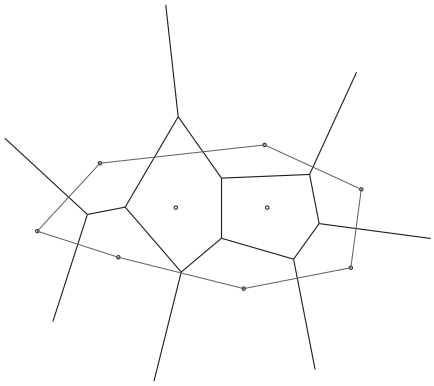


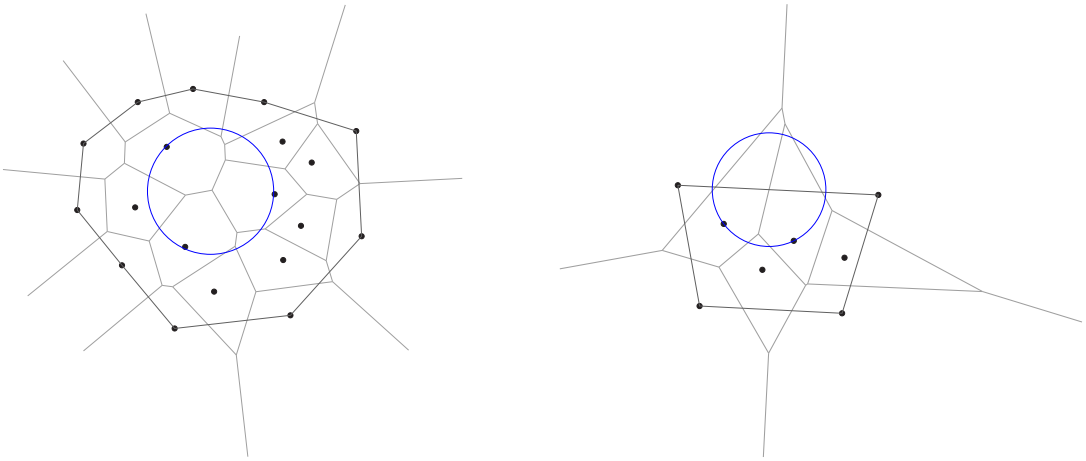
Abbildung 3.1: Voronoi-Diagramm $\mathcal{VD}(\mathcal{S})$ und konvexe Hülle $ch(S)$

Da für jede Kante der konvexen Hülle mindestens ein Schnittpunkt mit einer Voronoi-Kante existiert, können wir die Lage des Mittelpunktes genauer festlegen.

Lemma 3.2 (Lage des Mittelpunktes des $\mathcal{LEC}(\mathcal{S})$)
Sei S eine n-elementige Menge von Orten p_i mit $i \in \{1, ..., n\}$ und $\mathcal{VD}(\mathcal{S})$ gegeben im $\mathbb{R}^2$. Dann liegt der Mittelpunkt des $\mathcal{LEC}(\mathcal{S})$ entweder auf einem Voronoi-Knoten oder auf dem Schnittpunkt einer Voronoi-Kante mit einer Kante des Randes der konvexen Hülle von S.

Beweis:
Bei [Kle97] wird die Lage des Mittelpunktes im Zusammenhang mit einem unabhängigen Polygon bewiesen. Dies übertragen wir analog auf die zwei möglichen Positionen im Zusammenhang mit der konvexen Hülle.

Sei $x \in ch(S)$ beliebig und $C(x)$ ein Kreis um x, der keinen inneren Punkt $p_i \in S$ enthält. Zu beweisen ist, dass $C(x)$ vergrößert und gegebenenfalls verschoben werden kann, bis der Mittelpunkt von $C(x)$ entweder auf einem Voronoi-Knoten oder auf dem Schnittpunkt einer Voronoi-Kante mit einer Kante des Randes der konvexen Hülle liegt. Trifft dies für einen beliebigen „leeren“ Kreis zu, so gilt dies auch für den $\mathcal{LEC}(\mathcal{S})$.
$C(x)$ wird solange bei einem festen x vergrößert, bis $\partial(C(x))$ mindestens einen Ort aus S enthält. Sind mehr als zwei Orte betroffen, ist x ein Voronoi-Knoten (siehe Lemma 2.3 auf Seite 16). Werden zwei Orte p_i und p_j mit $i, j \in \{1, ..., n\}$ berührt, liegt x auf einer Voronoi-Kante, die Teil vom Bisektor $B(p_i, p_j)$ ist. In diesem Fall wird x so lange entlang $B(p_i, p_j)$ verschoben, wobei $C(x)$ weiterhin p_i und p_j enthält und durch die Verschiebung vergrößert wird, bis $C(x)$ auf einen weiteren Ort aus S oder x auf $\partial(ch(S))$ stößt. Wenn $\partial(C(x))$ am Anfang nur einen Ort p_i enthält, wird x unter Vergrößerung von $C(x)$ und $p_i \in C(x)$ so lange verschoben, bis $\partial(C(x))$ einen weiteren Ort aus S enthält oder x $\partial(ch(S))$ erreicht. Trifft das Letztgenannte zu, wird die Verschiebung von x unter Vergrößerung des Radius von $C(x)$ auf $\partial(ch(S))$ fortgesetzt, bis $C(x)$ mindestens einen weiteren Ort aus S berührt. Dann liegt x auf einen Schnittpunkt einer Kante der konvexen Hülle mit einer Voronoi-Kante. □



(a) Mittelpunkt auf einem Voronoi-Knoten

(b) Mittelpunkt auf einer Voronoi-Kante

Abbildung 3.2: Der größte leere Kreis einer Punktmenge

Die Aussage von Lemma 3.2 impliziert unter der Voraussetzung der allgemeinen Lage, dass der Kreis durch zwei oder drei Punkte aus S führt, die die maximal mögliche Distanz zu dem Mittelpunkt des $\mathcal{LEC}(\mathcal{S})$ haben.

Da wir die die konvexe Hülle als Begrenzung für die Lage des Mittelpunktes des $\mathcal{LEC}(\mathcal{S})$ gewählt haben, lässt sich bezüglich der Schnittpunkte zwischen der konvexen Hülle und des $\mathcal{VD}(\mathcal{S})$ noch eine weitere Aussage treffen.

Lemma 3.3 (Eigenschaften des Mittelpunktes des $\mathcal{LEC}(\mathcal{S})$)
Sei S eine n-elementige Menge mit $n \geq 3$ und $\mathcal{VD}(\mathcal{S})$ gegeben im $\mathbb{R}^2$. Der Schnittpunkt einer unbeschränkten Kante mit dem Rand der konvexen Hülle kann nur dem Mittelpunkt von $\mathcal{LEC}(\mathcal{S})$ entsprechen, wenn er identisch zu einem Voronoi-Knoten ist.

Beweis:
Sei q der Schnittpunkt zwischen der Kante der konvexen Hülle mit den Endpunkten $p_i, p_j \in S$ und einer unbeschränkten Voronoi-Kante k. Nach der Definition von $\mathcal{VD}(\mathcal{S})$ liegt $p_i \in VR(p_i, S)$ und $p_j \in VR(p_j, S)$. k ist also eine Teilmenge der Mittelsenkrechten zwischen p_i und p_j. Da die Kardinalität von S größer oder gleich 3 ist, existiert ein Voronoi-Knoten $v \in k$, um den ein leerer Kreis mit dem Radius $|vp_i|$ bzw. $|vp_j|$ gezogen werden kann. Für den Abstand $|vp_i|$ bzw. $|vp_j|$ gilt, dass $|vp_i|^2 = |vq|^2 + |qp_i|^2 \geq |qp_i|^2$. Also ist der Radius eines leeren Kreises um q kleiner als der Radius eines leeren Kreises um v, falls diese verschieden sind. □

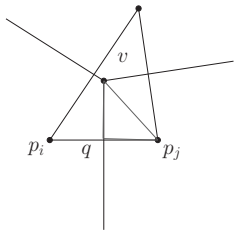


Abbildung 3.3: Schnittpunkte von $\mathcal{VD}(\mathcal{S})$ mit $ch(S)$

Die Anwendungen des größten leeren Kreises sind weitverbreitet bei Problemstellungen der Standortoptimierung. Neben dem schon in der Einleitung erwähnten Beispiel wird hier ein weiteres aus dem Bereich der Geodäsie angeführt. Unter [HSB02] wird vom Bundesamt für Kartographie und Geodäsie eine transportable Station (TIGO) zur Messung von entsprechenden Daten aus der Geodäsie, Meteorologie etc. vorgestellt. Um eine neue Station möglichst effektiv einzusetzen, wird der Mittelpunkt des größten leeren Kreises innerhalb der weltweit verteilten Stationen als Standort gewählt.

Anwendung findet der größte leere Kreis auch in anderen Bereichen wie bei der effizienten Informationsbeschaffung aus dem Internet (siehe [BFS02]) oder bei der Perturbation von Degenerationen für geometrische Algorithmen (siehe [hL03]).

3.2 Kleinster umfassender Kreis

Der kleinste umfassende Kreis (*smallest enclosing circle*) $\mathcal{SEC}(\mathcal{S})$ zu einer Punktmenge ist eindeutig und stellt ein Spezialfall des euklidischen *k-center-Problems* im $\mathbb{R}^d$ dar. Im allgemeinen Fall werden k Kugeln zu einer endlichen Menge S gesucht, die alle Punkte aus S enthalten und den maximalen aller k Radien minimieren.

Definition 3.2 (Kleinster umfassender Kreis $\mathcal{SEC}(\mathcal{S})$)
Sei S eine endliche Menge von Punkten im $\mathbb{R}^2$. Dann verstehen wir unter $\mathcal{SEC}(\mathcal{S})$ den kleinsten Kreis, der alle Punkte aus S enthält.

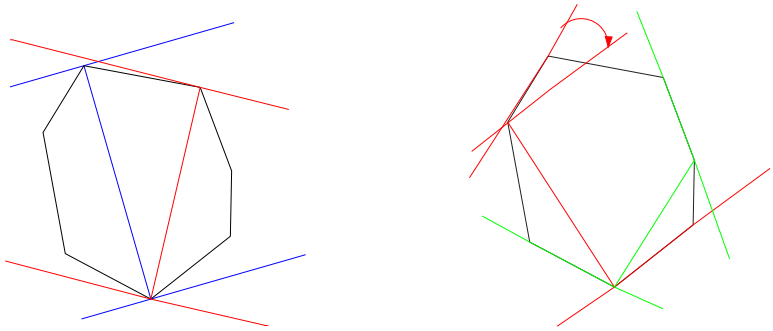
Für die Bestimmung des $\mathcal{SEC}(\mathcal{S})$ ist der Durchmesser der Punktmenge S von entscheidender Bedeutung. Alle Punkte aus S sind in $\mathcal{SEC}(\mathcal{S})$ enthalten, wodurch die maximale Distanz zwischen zwei Punkten dieser Menge der minimal mögliche Durchmesser des $\mathcal{SEC}(\mathcal{S})$ ist. In den beiden nächsten Lemmata gehen wir auf Teilmengen von S ein, die ausschlaggebend für den Durchmesser von S sind. Bei den Eigenschaften des Voronoi-Diagramms $\mathcal{FVD}(\mathcal{S})$ wurde bereits vorgestellt, dass nur die Punkte auf dem Rand der konvexen Hülle die Bildung des Voronoi-Diagramms und damit der Voronoi-Regionen beeinflussen. Diese Aussage kann in der Art auf die Bestimmung des Durchmesser einer Punktmenge übertragen werden, dass nur die Punkte auf dem Rand der konvexen Hülle für diesen Durchmesser verantwortlich sind.

Lemma 3.4 (Durchmesser einer Punktmenge)
Sei S eine endliche Punktmenge im $\mathbb{R}^2$. Dann ist der Durchmesser von S gleich der maximalen Distanz zwischen zwei Punkten aus S auf dem Rand der konvexen Hülle von S.

Beweis:
Wir werden zeigen, dass ein Punktepaar aus S, das maximalen Abstand zueinander hat und den Durchmesser $d = max\{|p_ip_j|; p_i, p_j \in S\}$ bestimmt, auf dem Rand der konvexen Hülle von S liegt. Für Punktmengen in allgemeiner Lage gilt, dass $VR_F(p_i, S) \neq \emptyset \Leftrightarrow p_i \in \partial(ch(S))$.
O.B.d.A. gehen wir davon aus, dass nur ein Punktepaar $p_i, p_j \in S$ mit maximalen Abstand zueinander existiert und S sich in allgemeiner Lage befindet. Alle $x \in \mathbb{R}^2$, die einen größeren Abstand zu p_i als zu irgendeinem anderen Ort aus S haben, liegen nach der Definition von $\mathcal{FVD}(\mathcal{S})$ in $VR_F(p_i, S)$. Also liegt p_i in $VR_F(p_j, S)$ und p_j in $VR_F(p_i, S)$. Daraus folgt, dass $VR_F(p_i, S), VR_F(p_j, S) \neq \emptyset$ und damit $p_i, p_j \in \partial(ch(S))$. □

Der Durchmesser von S ist also gleich dem Durchmesser von $ch(S)$. Liegen alle Punkte aus S auf dem Rand der konvexen Hülle von S, kann anhand dieser Aussage die zu untersuchende Punktmenge nicht eingeschränkt werden. Diese Möglichkeit eröffnet ein Satz von Yaglom-Bolyanski [YB61], der besagt, dass der Durchmesser einer konvexen Abbildung der größte Abstand zwischen zwei parallelen Stützgeraden dieser Abbildung ist. Diesen Satz kann man auf das konvexe Polygon $ch(S)$ anwenden. Parallele Stützgeraden wiederum können nicht jedes Paar von Ecken des konvexen Polygons $ch(S)$ tangieren. Die Paare von Ecken, für die parallele Stützgeraden gefunden werden können, nennt man *entgegengesetzt*. Abbildung 3.4 auf der nächsten Seite veranschaulicht entgegengesetzte Ecken und zeigt Beispiele für nicht-entgegengesetzte Ecken. Würden die Geraden um die nicht-entgegengesetzten Ecken

so gedreht werden, dass sie parallel zueinander liegen, wären sie keine Stützgeraden mehr, sondern würden $ch(S)$ schneiden.



(a) (b)

Abbildung 3.4: Entgegengesetzte (a) und nicht-entgegengesetzte Ecken (b)

Lemma 3.5 (Durchmesser der konvexen Hülle einer Punktmenge)
Sei S eine endliche Punktmenge im $\mathbb{R}^2$. Der Durchmesser von $ch(S)$ ist der größte Abstand zwischen zwei entgegengesetzten Punkten.

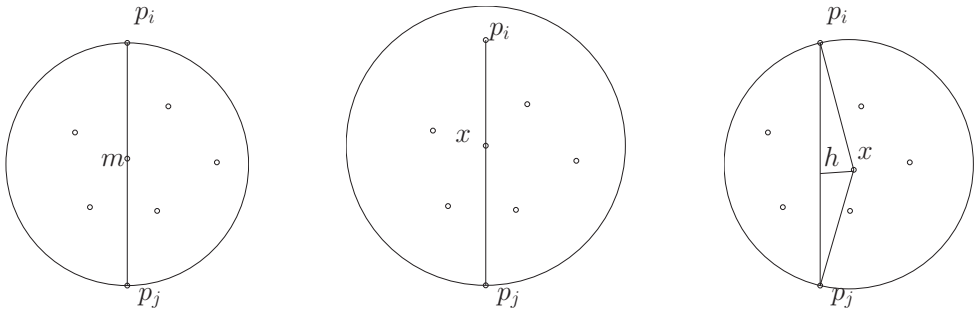
Beweis:
Zu zeigen ist, dass die Distanz zwischen zwei parallelen Stützgeraden von $ch(S)$ kleiner oder gleich dem Durchmesser d von $ch(S)$ ist und zwei parallele Stützgeraden existieren, die im Abstand d zueinander liegen.
Angenommen, es gäbe zwei parallele Stützgeraden g_1, g_2 von $ch(S)$ mit einem größerem Abstand zueinander als der Durchmesser $d = max\{|p_ip_j|; p_i, p_j \in S\}$. Dann gibt es zwei Punkte $p_1, p_2 \in S$ mit $p_1 \in g_1,\ p_2 \in g_2$ und es gilt $|p_1p_2| \geq |g_1g_2|$[1]. Für den Abstand zwischen g_1 und g_2 gilt wiederum laut Annahme $|g_1g_2| > d$. Daraus folgt $|p_1p_2| \geq |g_1g_2| > max\{|p_ip_j|; p_i, p_j \in S\}$. Widerspruch. Also trifft für g_1 und g_2 zu, dass $|g_1g_2| \leq d$.
Sei $d = |p_1p_2|$ und g_1, g_2 parallele Geraden mit $p_1 \in g_1,\ p_2 \in g_2$, in der Art, dass $|p_1p_2| = |g_1g_2|$. Zu zeigen bleibt, dass g_1 und g_2 Stützgeraden von $ch(S)$ sind. Angenommen, g_1 würde $ch(S)$ schneiden und den $\mathbb{R}^2$ in zwei Halbebenen h_1 und h_2 teilen, so dass ein $x \in h_1$ mit $x \in \partial(ch(S))$ existiert und $p_2 \in h_2$. Dann gilt $|xp_2| \geq |xg_2| > |g_1g_2| = |p_1p_2|$. Widerspruch. Also existieren zwei parallele Stützgeraden mit Abstand d zueinander. □

Eine Alternative für die Lage des Mittelpunktes des $\mathcal{SEC}(\mathcal{S})$ stellt der Mittelpunkt eines *diametralen* Kreises dar. Unter einem diametralen Kreis $C_{dia}(m)$ verstehen wir einen Kreis, dessen Durchmesser gleich dem Durchmesser der Punktmenge S ist. $C_{dia}(m)$ wird durch zwei Elemente p_i und p_j aus S definiert, die maximalen

[1]Der Abstand paralleler Geraden entspricht der minimalen Distanz zwischen ihren Punkten, also der Länge des Lots zwischen den beiden Geraden.

Abstand in S zueinander haben und der Mittelpunkt m ist in dem Liniensegment p_ip_j enthalten. Der Radius eines diametralen Kreises entspricht dann $\frac{|p_ip_j|}{2}$.

Lemma 3.6 (Zusammenhang zwischen diametralen Kreis und $\mathcal{SEC}(\mathcal{S})$)
Sei S eine endliche Menge von Punkten im $\mathbb{R}^2$. Gibt es einen diametralen Kreis, der alle Punkte aus S enthält, entspricht $\mathcal{SEC}(\mathcal{S})$ diesem diametralen Kreis.



(a) $C_{dia}(m)$ definiert durch p_i und p_j

(b) p_j auf dem Rand von $C(x)$, p_i im Innern von $C(x)$

(c) p_i und p_j auf dem Rand von $C(x)$

Abbildung 3.5: Kreise um eine Punktmenge

Beweis:
Seien p_i und p_j zwei Punkte, die maximalen Abstand in der Punktmenge zueinander haben und den diametralen Kreis $C_{dia}(m)$ mit Radius $\frac{|p_ip_j|}{2}$ definieren. Falls in $C_{dia}(m)$ alle Punkte aus S liegen, ist zu zeigen, dass es keinen anderen Kreis gibt, der alle Punkte aus S enthält und einen kleineren Radius als $C_{dia}(m)$ hat. Sei $C(x)$ ein Kreis um die Punktmenge S. Dann sind auch p_i und p_j in $C(x)$ enthalten. Liegt einer der beiden Punkten nicht auf $\partial C(x)$, ist der Durchmesser von $C(x)$ größer als der Abstand $|p_ip_j|$ und dementsprechend der Radius größer als $\frac{|p_ip_j|}{2}$. Sind beide Punkte in $\partial C(x)$ enthalten, dann ist $r = |xp_i| = |xp_j|$. Wenn $C(x)$ nicht einem diametralen Kreis entspricht, kann ein gleichschenkliges Dreieck durch x, p_i, p_j gebildet werden. Sei h die Höhe des Dreiecks, so gilt

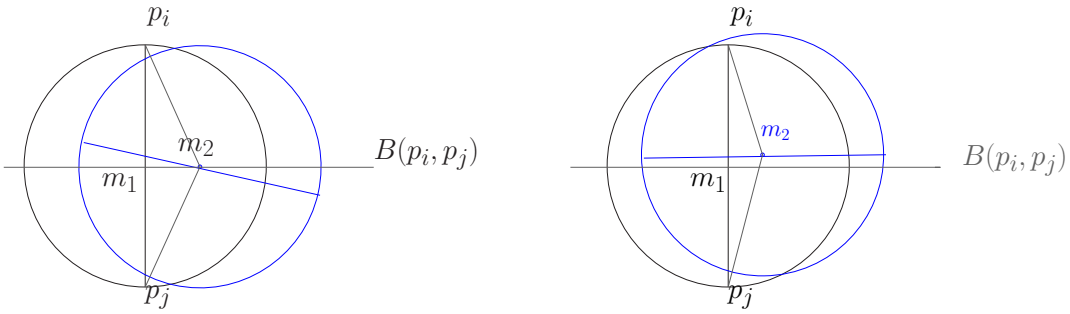
$$r = \sqrt{h^2 + \left(\frac{|p_ip_j|}{2}\right)^2} > \sqrt{\left(\frac{|p_ip_j|}{2}\right)^2} = \frac{|p_ip_j|}{2}.$$

Der Radius von $C(x)$ ist also größer als der von $C_{dia}(m)$. □

Da es bei einer n-elementigen Menge S höchstens $\frac{n}{2}$ Punktepaare für gerade n und n Punktepaare bei ungeraden n mit maximalen Abstand zueinander geben kann, können gegebenenfalls ebenso viele diametrale Kreise konstruiert werden. Eine Ausnahme besteht in dem Fall, wenn die Punktepaare kozirkular angeordnet sind und damit einen identischen diametralen Kreis definieren. Dann liegen nicht nur zwei Punkte auf dem Rand des Kreises, sondern mindestens drei. Treten mehrere

diametrale Kreise auf, können wir folgende nützliche Eigenschaft im Bezug auf S herleiten.

Lemma 3.7 (Diametrale Kreise und umschlossene Punkte)
Sei S eine endliche Menge von Punkten im $\mathbb{R}^2$. Existieren zwei verschiedene diametrale Kreise, enthält keiner von ihnen S vollständig.



(a) m_2 auf $B(p_i, p_j)$ (b) m_2 in $D(p_i, p_j)$

Abbildung 3.6: Diametrale Kreise

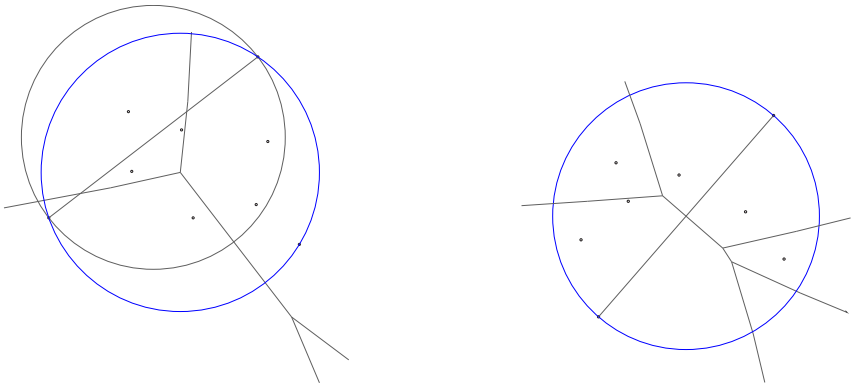
Beweis:
Angenommen, es existieren $C_{dia}(m_1)$ und $C_{dia}(m_2)$ mit $m_1 \neq m_2$. Der Durchmesser d der beiden Kreise ist per definitionem gleich und damit auch der Radius r. Seien p_i und p_j die definierenden Punkte von $C_{dia}(m_1)$ mit Abstand r zu m_1. m_1 entspricht dem Lotpunkt[2] auf den Bisektor $B(p_i, p_j)$ von p_i bzw. p_j. Jeder Punkt auf $B(p_i, p_j)$ hat einen größeren Abstand als r zu p_i bzw. p_j, falls er nicht m_1 entspricht. Liegt m_2 also auf dem Bisektor $B(p_i, p_j)$, dann ist die Distanz $|m_2 p_i|$ bzw. $|m_2 p_j|$ größer als r. Anderenfalls liegt m_2 in $D(p_i, p_j)$ bzw. $D(p_j, p_i)$. Angenommen, m_2 befindet sich in $D(p_i, p_j)$, dann ist m_2 weiter entfernt von p_j als von p_i und der Abstand $|m_2 p_j|$ ist größer als r. Analog gilt dies für $|m_2 p_i|$, falls m_2 in $D(p_j, p_i)$ liegt. □

Implizit wird in dem Beweis zu Lemma 3.6 gezeigt, dass für alle Kreise um S, auf deren Rand weniger als zwei Orte liegen, jeweils ein Kreis mit kleinerem Radius existiert, der ebenfalls S umschließt. Dadurch wird offensichtlich, dass der Rand von $\mathcal{SEC}(\mathcal{S})$ mindestens zwei Elemente aus S beinhaltet. Diese Eigenschaft ist sehr nützlich für die Bestimmung des Mittelpunktes m von $\mathcal{SEC}(\mathcal{S})$. Bei der Vorstellung des Voronoi-Diagramms $\mathcal{FVD}(\mathcal{S})$ wurden die Voronoi-Knoten und -Kanten unter anderem dadurch charakterisiert, dass sich jeweils ein Kreis um beliebige $x \in \mathcal{FVD}(\mathcal{S})$ finden lässt, der genau diese Eigenschaft erfüllt. Entspricht x einem Voronoi-Knoten, liegen mindestens drei Orte aus S auf dem Rand des Kreises, ist x in einer Voronoi-Kante enthalten, liegen genau zwei Orte aus S auf dem Rand des Kreises. Bei dem Sonderfall, dass $\mathcal{FVD}(\mathcal{S})$ nur aus einer Voronoi-Kante besteht, sind alle Punkte aus

[2]Der Abstand zwischen einem Lotpunkt auf einer Geraden und einem Punkt ist der geringste Abstand zwischen allen Elementen der Geraden und diesem Punkt.

S kollinear angeordnet oder S besitzt nur zwei Elemente. In diesem Fall führt der diametrale Kreis um alle Orte und es gibt keinen umschließenden Kreis mit kleinerem Radius.

Lemma 3.8 (Lage des Mittelpunktes des $\mathcal{SEC}(\mathcal{S})$)
Sei S eine endliche Menge von Punkten und $\mathcal{FVD}(\mathcal{S})$ gegeben im $\mathbb{R}^2$. Dann ist der $\mathcal{SEC}(\mathcal{S})$ eindeutig und dessen Mittelpunkt $m \in \mathbb{R}^2$ liegt entweder auf einem Voronoi-Knoten oder einer Voronoi-Kante. Im letzten Fall entspricht m dem Mittelpunkt eines diametralen Kreises.



(a) Mittelpunkt auf einem Voronoi-Knoten

(b) Mittelpunkt auf einer Voronoi-Kante

Abbildung 3.7: Der kleinste umfassende Kreis einer Punktmenge

Beweis:
Für den Beweis der Eindeutigkeit siehe [dBvKOS97]. Dass m dem Mittelpunkt des diametralen Kreises entsprechen kann, haben wir in Lemma3.6 bewiesen. Die Lage auf der Voronoi-Kante in diesem Fall ergibt sich direkt aus den Eigenschaften des $\mathcal{FVD}(\mathcal{S})$. Es liegen mindestens zwei Orte auf dem diametralen Kreis und da er alle restlichen Orte aus S in seinem Inneren enthält, ist der Mittelpunkt Element des $\mathcal{FVD}(\mathcal{S})$. Sind genau zwei Orte auf dem Kreis, gehört der Mittelpunkt zu einer Voronoi-Kante, ansonsten entspricht er einem Voronoi-Knoten.

Existiert kein diametraler Kreis $C_{dia}(x)$, der alle Punkte aus S enthält, ist zu zeigen, dass der Mittelpunkt m von $\mathcal{SEC}(\mathcal{S})$ auf einem Voronoi-Knoten liegt. Seien p_i und p_j ein Punktepaar aus S mit maximalen Abstand, das $C_{dia}(x)$ definiert. Dann müssen p_i und p_j im Rand von $\mathcal{SEC}(\mathcal{S})$ enthalten sein, da ansonsten ein Kreis mit kleinerem Radius um S gefunden werden kann (siehe Beweis zu Lemma 3.6). Sei $p_l \in S$ ein Punkt, der außerhalb von $C_{dia}(x)$ liegt und maximalen Abstand von allen Punkten aus S zu x hat. Dann ist der Radius eines umschließenden Kreises minimal, der p_i, p_j und p_l in seinem Rand enthält. Dementsprechend liegt der Mittelpunkt m von $\mathcal{SEC}(\mathcal{S})$ auf einem Voronoi-Knoten. □

Analog zum $\mathcal{LEC}(\mathcal{S})$ sind auch hier unter Annahme der allgemeinen Lage entweder zwei oder drei Punkte in dem $\mathcal{SEC}(\mathcal{S})$ enthalten.

Preparata und Shamos verweisen in [PS85] auf einige Anwendungen für den $\mathcal{SEC}(\mathcal{S})$ aus der Standortoptimierung und beschreiben dort auch das in der Einleitung erwähnte Beispiel des zu positionierenden Senders für ein Wohngebiet.

In [dBvKOS97] wird der Einsatz des $\mathcal{SEC}(\mathcal{S})$ bei festmontierten Roboterarmen, die sich um ihre eigene Achse drehen können, vorgestellt. Diese sollen innerhalb ihrer Reichweite Werkstücke aufheben und an anderer Stelle ablegen. Der ideale Platz für die Montierung ist dabei der Mittelpunkt des $\mathcal{SEC}(\mathcal{S})$ um die möglichen Abstellorte für die Werkstücke.

3.3 Ring mit minimaler Breite

Der Ring mit minimaler Breite oder auch *minimum-width annulus* $\mathcal{MWA}(\mathcal{S})$ wird in Veröffentlichungen unter verschiedenen Vorgaben betrachtet. *Rivlin* untersucht in [Riv79] den allgemeinen Fall ohne Einschränkung, während [dBBB+98] Bedingungen für die Radien der Kreise vorgeben. Vereinfachende Annahmen über die Lage der Punktmenge sind die Grundlagen in [SLW95] und [Ram99]. Veröffentlichungen wiederum wie die von Chan und Agarwal et al. ([Cha00], [AAHPS99]) haben nicht eine exakte Bestimmung des $\mathcal{MWA}(\mathcal{S})$ zum Ziel, sondern approximative Lösungen.

Wir werden den exakten Fall im $\mathbb{R}^2$ betrachten und benötigen für die Definition des $\mathcal{MWA}(\mathcal{S})$ den Raum $\mathbb{R}^2_\infty$, der entsteht, wenn wir den Raum $\mathbb{R}^2$ jeweils um eine Gerade im „negativen“ als auch „positiven“ unendlichen Bereich ergänzen.

Definition 3.3 (Ring mit minimaler Breite $\mathcal{MWA}(\mathcal{S})$)
Sei S eine n-elementige Menge von Punkten im $\mathbb{R}^2$. Dann verstehen wir unter $\mathcal{MWA}(\mathcal{S})$ den Ring $\mathcal{A}(m, r_1, r_0)$ um $m \in \mathbb{R}^2_\infty$ mit minimalem $r_0 - r_1$, der alle Punkte aus S enthält. Für $n = 2$ oder alle Punkte kollinear in S entspricht $\mathcal{MWA}(\mathcal{S})$ einer Geraden und für $n = 3$ einem Kreis durch die Punkte aus S.

Liegt der Mittelpunkt m im unendlichen Bereich, entspricht MWA nicht mehr einem Ring, sondern zwei parallelen Geraden, zwischen denen alle Punkte enthalten sind. Die abgeschlossene Menge zwischen diesen Geraden wird als *Streifen* bezeichnet. Für die praktische Umsetzung ist die theoretische Festlegung des Mittelpunktes im Unendlichen wenig hilfreich. Deswegen werden wir an dieser Stelle auf den Zusammenhang mit der konvexen Hülle eingehen.

Die *Breite* einer Punktmenge ist definiert als der minimale Abstand zwischen zwei parallelen Stützgeraden ihrer konvexen Hülle. Diese Breite entspricht der minimalen Breite eines Streifens, der die gesamte Menge enthält. Analog zum Durchmesser einer Punktmenge sind nicht alle Paare von Punkten auf dem Rand der konvexen Hülle zu analysieren, sondern die, die entgegengesetzt liegen. Hierbei ist jedoch nicht der Abstand entgegengesetzter Punkte zueinander von Interesse, sondern der Abstand der Kanten der konvexen Hülle zu jeweils einem entgegengesetzten Punkt.

Lemma 3.9 (Lage des Streifens mit minimaler Breite)
Sei S eine endliche Menge von Punkten im $\mathbb{R}^2$ und das konvexe Polygon $ch(S)$ gegeben. Dann sind die parallelen Geraden, die einen Streifen mit minimaler Breite festlegen, Stützgeraden von $ch(S)$. In mindestens einer der beiden Stützgeraden ist eine Kante der konvexen Hülle enthalten.

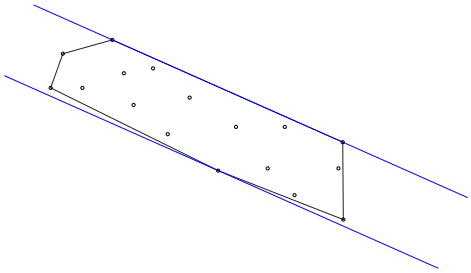


Abbildung 3.8: Streifen minimaler Breite

Beweis:
Zu zeigen ist, dass die Breite eines Streifens dann minimal ist, wenn mindestens eine der beiden definierenden Stützgeraden eine Kante von $ch(S)$ enthält. Dass zwei parallele Geraden g_1, g_2 eines solchen Streifens Stützgeraden von $ch(S)$ sein müssen, lässt sich folgendermaßen veranschaulichen. Falls g_1 und g_2 keine Stützgeraden wären, kann der Abstand zwischen ihnen solange verringert werden, bis beide jeweils einen Punkt von $\partial(ch(S))$ tangieren.

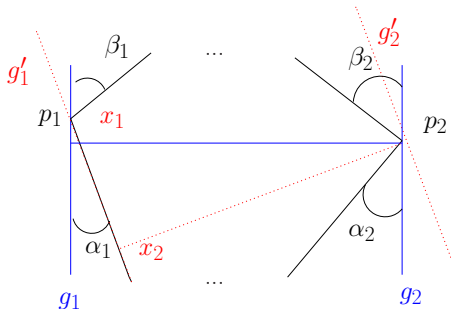


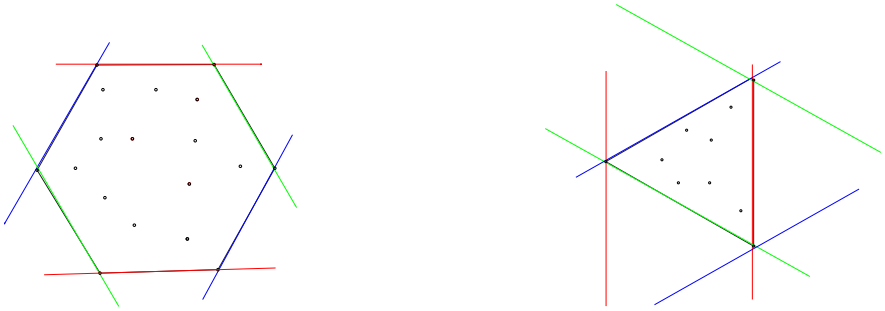
Abbildung 3.9: Parallele Stützgeraden

Seien p_1, p_2 entgegengesetzte Punkte aus S mit $p_1 \in g_1$, $p_2 \in g_2$ und keine Kante der konvexen Hülle in g_i mit $i \in \{1, 2\}$ enthalten. Dann definieren g_1 und g_2 jeweils zwei Winkel α_i, β_i zu den angrenzenden Kanten des Punktes p_i. Durch Drehung beider Geraden um $min\{\alpha_1, \beta_1, \alpha_2, \beta_2\}$ bleiben diese parallel und behalten die Eigenschaft der Stützgeraden, denn die Drehung belässt alle Punkte aus S auf der selben Seite von g_i oder verlegt sie auf g_i. In Abbildung 3.9 ist beispielhaft zu sehen, dass die Drehung um α_1 die Gerade g_1' auf eine Kante der konvexen Hülle verlegt und die Gerade g_2' weiterhin eine Stützgerade im Punkt p_2 bleibt. Jeweils über das Lot von

g_1, g_2 und das Lot von g_1', g_2' lässt sich ein rechtwinkliges Dreieck konstruieren mit den Eckpunkten x_1, x_2, p_2. Für die Seiten des Dreiecks gilt $|x_1p_2| \leq |g_1g_2|$, $|x_1x_2| > 0$ und $|g_1'g_2'| = \sqrt{|x_1p_2|^2 - |x_1x_2|^2}$. Daraus folgt

$$|g_1'g_2'| \leq \sqrt{|g_1g_2|^2 - |x_1x_2|^2} < \sqrt{|g_1g_2|^2} = |g_1g_2|.$$

Damit ist der Abstand zwischen zwei parallelen Stützgeraden, von denen mindestens eine durch eine Kante der konvexen Hülle induziert ist, minimal. Sind die Kanten der konvexen Hülle an entgegengesetzten Punkten parallel, liegt der Sonderfall vor, dass beide Stützgeraden eine Kante der konvexen Hülle enthalten. □



(a) Regelmäßiges Polygon mit 6 Ecken

(b) Regelmäßiges Polygon mit 3 Ecken

Abbildung 3.10: Streifen minimaler Breite

Liegt die konvexe Hülle als regelmäßiges Polygon vor, das heißt, die Seiten des Polygons sind alle gleich lang, gibt es mehrere Streifen minimaler Breite. Besitzt die konvexe Hülle n Ecken und n ist gerade, trifft dies genau für $\frac{n}{2}$ Streifen zu. Ist n ungerade, trifft dies genau für n Streifen zu.

Der Fall, dass $\mathcal{MWA}(\mathcal{S})$ einem Streifen minimaler Breite entspricht, tritt immer dann auf, wenn die Breite einer Punktmenge kleiner als die Breite eines umfassenden Rings ist. Für die Lage des Mittelpunktes dieses Rings können die strukturellen Eigenschaften der Voronoi-Diagramme $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ genutzt werden. Sind alle Punkte aus S kozirkular angeordnet, stimmt der innere und äußere Kreis überein. Es handelt sich also um einen Ring mit der Breite 0.

Lemma 3.10 (Lage des Rings mit minimaler Breite)
Sei S eine endliche Menge von Punkten, die nicht alle kollinear liegen und $\mathcal{VD}(\mathcal{S})$ bzw. $\mathcal{FVD}(\mathcal{S})$ gegeben. Dann liegt der Mittelpunkt des Rings mit minimaler Breite auf einem Schnittpunkt von $\mathcal{VD}(\mathcal{S})$ mit $\mathcal{FVD}(\mathcal{S})$. Insbesondere sind die Orte auf den Kreisen sind unter Annahme der allgemeinen Lage alternierend angeordnet, d.h. vom Mittelpunkt aus gesehen liegen im Winkelbogen zweier Orte des einen Kreises genau ein Ort des anderen Kreises.

Beweis:
Für den Beweis der alternierenden Anordnung siehe [Riv79]. Dort wird ebenfalls

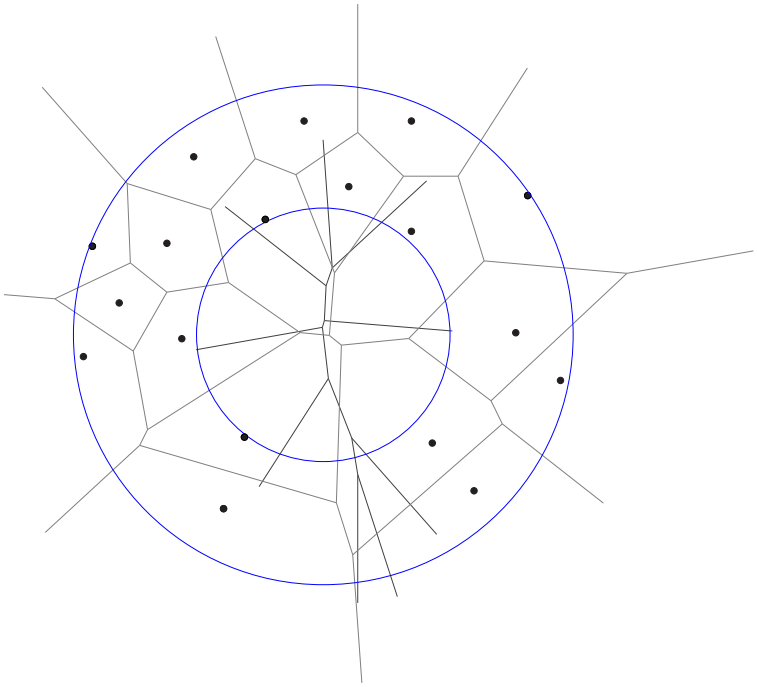


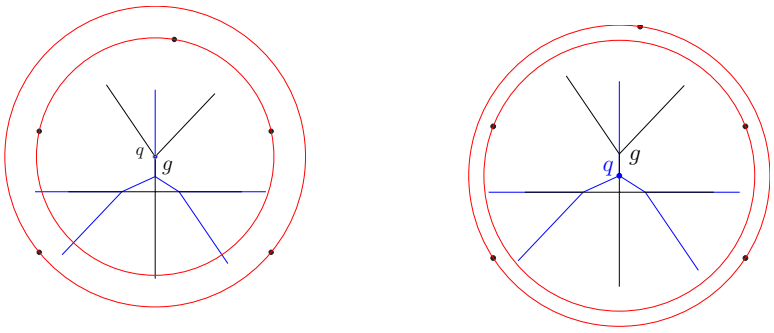
Abbildung 3.11: Ring minimaler Breite

gezeigt, dass mindestens je zwei Orte, falls es S mindestens vier Elemente beinhaltet, auf dem inneren und äußeren Kreis des Rings liegen. Unter der Voraussetzung der allgemeinen Lage für die Punktmenge handelt es sich jeweils um genau zwei Orte. Mit dieser Aussage ist die Lage des Mittelpunktes auf einem Schnittpunkt von $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ direkt nachvollziehbar. Der innere Kreis stellt einen leeren Kreis dar, der äußere einen umschließenden. Da jeweils mindestens zwei Orte auf den Kreisrändern liegen, muss der Mittelpunkt sowohl Element von $\mathcal{VD}(\mathcal{S})$ als auch Element von $\mathcal{FVD}(\mathcal{S})$ sein. □

Sollten Orte aus S in der Art angeordnet sein, dass sich Voronoi-Kanten aus $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ in mehr als einem Punkt schneiden, so kommt nicht jeder solche Punkt der Schnittmenge als Mittelpunkt für $\mathcal{MWA}(\mathcal{S})$ in Frage.

Lemma 3.11 (Mögliche Mittelpunkte von $\mathcal{MWA}(\mathcal{S})$)
Sei S eine n-elementige Menge von Punkten mit $n > 3$ und nicht alle Punkte befinden sich in kollinearer Lage. Seien $\mathcal{VD}(\mathcal{S})$ bzw. $\mathcal{FVD}(\mathcal{S})$ gegeben. Besteht der Schnitt einer Voronoi-Kante von $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ in der Punktmenge g, so kann ein Element $q \in g$ nur der Mittelpunkt eines Rings mit minimaler Breite sein, wenn q ein Voronoi-Knoten ist.

Beweis:
Seien p_{v1} und p_{v2} die Punkte aus S, deren Voronoi-Regionen eine Voronoi-Kante e_v von $\mathcal{VD}(\mathcal{S})$ bilden. Seien p_{f1} und p_{f2} die Punkte aus S, deren Voronoi-Regionen eine



(a) (b)

Abbildung 3.12: $\mathcal{VD}(\mathcal{S})$ (schwarz) und $\mathcal{FVD}(\mathcal{S})$ (blau) zu einer Punktmenge S, (a) Ring um $q \in g$, (b) Ring um $q \in g$ mit alternierenden Punkten

Voronoi-Kante e_f von $\mathcal{FVD}(\mathcal{S})$ bilden. Sei $g = e_v \cap e_f$ und $g \neq \emptyset$. Für den Beweis werden zwei Fälle unterschieden.

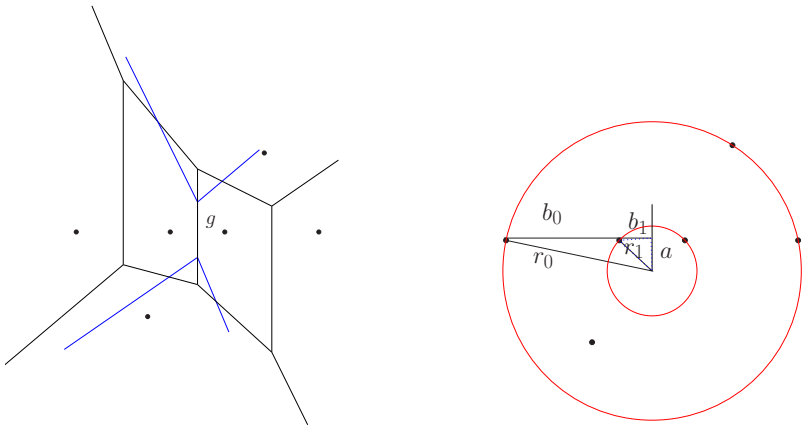
1. Seien p_{v1}, p_{v2}, p_{f1} und p_{f2} nicht kollinear angeordnet. Sei $q \in g$ ein beliebiger Mittelpunkt für einen Ring durch die 4 Punkte und q kein Voronoi-Knoten. Dann liegen p_{v1} und p_{v2} auf dem inneren Kreis im Winkel α bzw. $-\alpha$ zu g. p_{f1} und p_{f2} liegen auf dem äußeren Kreis im Winkel β bzw. $-\beta$ zu g. Somit können sich die 4 Punkte niemals in alternierender Lage auf dem Ring befinden und q kann nicht dem Mittelpunkt von $\mathcal{MWA}(\mathcal{S})$ entsprechen. Ist q ein Voronoi-Knoten, sind 5 Punkte aus S auf dem Ring und der zusätzliche Punkt kann so angeordnet sein, dass die Punkte im Winkelbogen wiederum alternierend liegen.

2. Seien p_{v1}, p_{v2}, p_{f1} und p_{f2} kollinear angeordnet. Dann ist zu zeigen, dass für $q \in g$ ein Ring um q breiter ist, falls q keinem Voronoi-Knoten entspricht. Die Breite des Rings lässt sich aus der Differenz der Radien der beiden Kreise berechnen und die Radien bestimmen wir aus ${r_i}^2 = a^2 + {b_i}^2$ mit $i \in \{0, 1\}$ (siehe Abbildung 3.13 auf der nächsten Seite).

$$
\begin{aligned}
f(a) &:= r_0 - r_1 \\
&= \sqrt{a^2 + {b_0}^2} - \sqrt{a^2 + {b_1}^2} \\
f'(a) &= \frac{a}{\sqrt{a^2 + {b_0}^2}} - \frac{a}{\sqrt{a^2 + {b_1}^2}} \leq 0, \text{ da } a \geq 0 \text{ und } b_1 < b_0
\end{aligned}
$$

Da offenbar $f'(a) < 0$ für $a > 0$, wird die Breite eines Rings mit wachsendem a kleiner. Damit ist die Breite eines Rings um einen Voronoi-Knoten kleiner oder gleich der Breite um einen beliebigen Punkt q aus g.

□

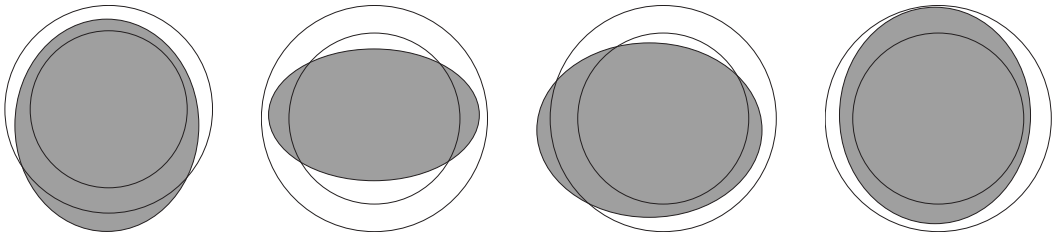


(a) $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$

(b) Ring um einen Voronoi-Knoten

Abbildung 3.13: Vier kollineare Punkte in S

Eine der Hauptanwendungen des $\mathcal{MWA}(\mathcal{S})$ liegt in der Metrologie und aus der umfangreichen Literatur sei an dieser Stelle nur [Yap95] genannt. Unter den Begriffen *roundness* (Rundung) und *flatness* (Flachheit) werden in der Metrologie Eigenschaften von Punktmengen aufgeführt. Die Analyse der *roundness* einer Punktmenge beinhaltet nach einem Vorschlag des *American National Standard Institute* die Untersuchung, ob die Punktmenge in einem Ring mit gegebener Breite enthalten ist. Die Ermittlung der Breite einer Punktmenge wird unter dem Problem der *flatness* einer Punktmenge aufgeführt. Sind maschinell gefertigte Werkstücke kreisförmig, kann ihre *roundness* anhand von Ringen minimaler Breite beurteilt werden, die eine vorgegebene Toleranz darstellen. Bei maschinellen Fertigungen entsprechen die Werkstücke im Idealfall exakt dem vorgegebenen Modell, können aber trotzdem eingesetzt werden, falls die Abweichungen von der Vorlage innerhalb der vorgegebenen Toleranz liegen.



(a) ungültig (b) ungültig (c) ungültig (d) gültig

Abbildung 3.14: Werkstücke bei der Qualitätssicherung

Bei der Qualitätssicherung kommen beispielsweise *coordinate measure machines (CMM)* oder *Elektronenmikroskope* zum Einsatz, die stichprobenartig die Oberfläche des Werkstücks überprüfen. Ragen die Koordinaten der Stichproben aus dem Ring heraus, aus dem äußeren und/oder dem inneren Kreis, hat es die Qualitätsprüfung

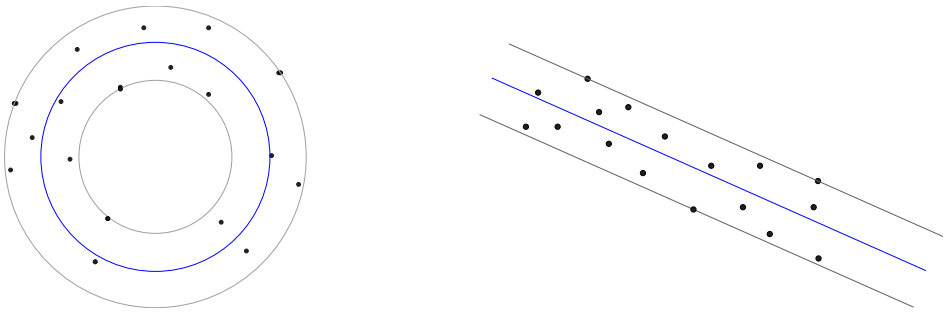
nicht bestanden. Stellt man sich zum Beispiel die Räder eines Schnellzuges vor, darf die Abweichung von dem idealen Muster aus Sicherheitsaspekten nicht zu groß sein. Die Ringe in der Abbildung 3.14 auf der vorherigen Seite veranschaulichen dies stark vergröbert. Weitere Anwendungen treten in Verbindung mit Delaunay-Triangulationen auf, deren Stabilität untersucht ([Tin98]) und unter anderem für die Analyse von Protein-Strukturen ([BS04]) genutzt wird.

3.4 Am besten angepasster Kreis

Analog zum $\mathcal{MWA}(\mathcal{S})$ betrachten wir den am besten angepassten Kreis $\mathcal{BFC}(\mathcal{S})$ (*best fitting circle*) einer Punktmenge unter dem Aspekt der Minimierung des maximalen Abstands eines Punktes zu dem Kreisrand. Der am besten angepasste Kreis wird in Veröffentlichungen unterschiedlich definiert. Aufgrund seiner analytischen Eigenschaften wird oftmals die Minimierung der quadratischen Abstände aller Punkte zu dem Kreisrand, das sogenannte *least squares fitting*, eingesetzt. Außerdem kann die Berechnung mit geometrischen oder algebraischen Distanzen vorgenommen werden, wobei wir den geometrischen Ansatz gewählt haben. Eine umfangreiche Übersicht geben [UJ00] und [CL03].

Definition 3.4 (Der am besten angepasste Kreis $\mathcal{BFC}(\mathcal{S})$)
Sei S eine n-elementige Menge von Punkten p_i für $i \in \{1, ..., n\}$ im $\mathbb{R}^2$. Dann verstehen wir unter $\mathcal{BFC}(\mathcal{S})$ den Kreis $C(m)$ um $m \in \mathbb{R}^2_\infty$, der den maximalen Abstand $|p_iC(m)|$ unter allen Kreisen minimiert.

Für den Fall, dass der Mittelpunkt im Unendlichen liegt, entspricht $\mathcal{BFC}(\mathcal{S})$ einer Geraden g und der maximale Abstand $|p_ig|$ wird minimiert.



(a) Der am besten angepasste Kreis

(b) Die am besten angepasste Gerade

Abbildung 3.15: $\mathcal{BFC}(\mathcal{S})$ einer Punktmenge

Auf die Verwandtschaft des $\mathcal{BFC}(\mathcal{S})$ zum $\mathcal{MWA}(\mathcal{S})$ gehen wir in dem nächsten Satz ein. Da $\mathcal{MWA}(\mathcal{S})$ nicht für jede Punktmenge eindeutig ist, kann es auch mehrere Möglichkeiten für die Lage des $\mathcal{BFC}(\mathcal{S})$ geben. Für uns ist jedoch nicht von Interesse, alle möglichen $\mathcal{BFC}(\mathcal{S})$ zu finden, sondern einen anzugeben, der die Bedingung der Minimierung des maximalen Abstands erfüllt.

Lemma 3.12 (Eigenschaften des $\mathcal{BFC}(\mathcal{S})$)
Sei S eine n-elementige Menge von Punkten im $\mathbb{R}^2$.

1. *Sei $\mathcal{MWA}(\mathcal{S})$ durch einen Ring minimaler Breite mit $\mathcal{A}(m, r_1, r_0)$ und $m \in R^2$ gegeben. Dann besitzt $\mathcal{BFC}(\mathcal{S})$ den gleichen Mittelpunkt wie $\mathcal{MWA}(\mathcal{S})$ und hat den Radius $\frac{r_1+r_0}{2}$.*

2. *Sei $\mathcal{MWA}(\mathcal{S})$ durch einen Streifen der Breite w gegeben. Dann stellt $\mathcal{BFC}(\mathcal{S})$ eine Gerade dar, die parallel innerhalb des Streifens verläuft im Abstand $\frac{w}{2}$ zu den begrenzenden Geraden.*

Beweis:

1. Sei $\mathcal{MWA}(\mathcal{S})$ durch einen Ring $\mathcal{A}(m, r_1, r_0)$ mit Mittelpunkt $m \in \mathbb{R}^2$ gegeben. Es gibt keinen Ring, der alle Punkte aus S umschließt und dessen Breite kleiner als $r_0 - r_1$ ist. Damit gibt es wiederum auch keinen Kreis, dessen minimaler Abstand zu allen Punkten aus S kleiner als $\frac{r_0-r_1}{2}$ sein kann. Also entspricht der Kreis $C(m)$ mit Radius $\frac{r_1+r_0}{2}$ einem $\mathcal{BFC}(\mathcal{S})$.

2. Sei $\mathcal{MWA}(\mathcal{S})$ durch einen Streifen in der Breite w der Punktmenge S gegeben. $\mathcal{MWA}(\mathcal{S})$ ist per definitionem der Streifen, der S enthält und die minimale Breite w besitzt. Dadurch gibt es keine Gerade, deren minimaler Abstand zu allen Punkten aus S kleiner als $\frac{w}{2}$ sein kann. Wird eine Gerade so parallel durch den Streifen geführt, dass der Abstand zu den begrenzenden Geraden des Streifens jeweils $\frac{w}{2}$ beträgt, ist ein $\mathcal{BFC}(\mathcal{S})$ in Geradenform gefunden.

□

Wenn wir das Anwendungsbeispiel des Rings mit minimaler Breite aus der Metrologie betrachten (siehe Kapitel 3.3), dann entspricht das vorgegebene Modell einem am besten angepassten Kreis. Die Toleranz für die Abweichung von diesem Modell wird durch einen Ring gegebener Breite festgelegt. Der Zusammenhang zwischen $\mathcal{BFC}(\mathcal{S})$ und $\mathcal{MWA}(\mathcal{S})$ ist also nicht nur theoretisch konstruiert, sondern auch relevant in der Praxis.

Die größte Schwierigkeit bei einer Entdeckung liegt nicht darin, die notwendigen Beobachtungen zu machen, als darin, sich bei ihrer Interpretation von traditionellen Vorstellungen zu lösen.
John Desmond Bernal (1901 - 1971)

4

Algorithmen

Bei der Entwicklung von Algorithmen haben wir unter anderem auch das Einsatzgebiet ihrer Implementation zu beachten. Das *Geometrie-Labor* der FernUniversität Hagen [urld] enthält einige Java-Applets, die Algorithmen aus dem Fachgebiet Algorithmische Geometrie veranschaulichen. Es steht unter anderem Studenten zur Verfügung, die die Lerninhalte des Kurses [Kle97] und weiterführende Themen interaktiv vertiefen möchten. Da die aus dieser Arbeit resultierende Implementation *FitCircle* ebenfalls im *Geometrie-Labor* eingebunden wird, soll auch *FitCircle* die Möglichkeit der Interaktion bieten, also einen sogenannten *Online*-Algorithmus realisieren. Für einen *Offline*-Algorithmus ist die vollständige Punktmenge schon vor dem Start bekannt. Ein *Online*-Algorithmus hingegen erhält die Punktmenge sukzessive. Hierbei werden Einfüge-, Verschiebe- und Löschereignisse unterschieden, wobei ein Verschiebeereignis eine Mischung aus Einfüge- und Löschereignis darstellt.

Für die Konzeption der Algorithmen werden wir zuerst die zu lösenden Problemstellungen genauer formulieren. Dabei benutzen wir den strukturellen Zusammenhang zwischen den vorgestellten Optimierungs- bzw. Approximationsaufgaben und den Voronoi-Diagrammen $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$. Die konvexe Hülle nimmt ebenfalls eine wichtige Rolle ein und dient unter anderem zur Berechnung des Durchmessers und der Breite eine Punktmenge.

Ein Kreis ist eindeutig durch seinen Mittelpunkt und seinen Radius festgelegt. Dementsprechend geben wir für die Konstruktion die zu überprüfenden Möglichkeiten der Lage des Mittelpunktes und die Bedingung für den Radius an. Analog zu der Reihenfolge, in denen die verschiedenen Optimierungen und Approximationen im vorherigen Kapitel vorgestellt wurden, sind die folgenden Probleme sortiert. Die Kreise und der Ring sind durch die vorgestellten Definitionen festgelegt und werden

im Weiteren nicht explizit erörtert.

Ein größter leerer Kreis wird unter der Voraussetzung, dass der Mittelpunkt innerhalb der konvexen Hülle liegt, ermittelt.

Problem 4.1 (Konstruktion des $\mathcal{LEC}(\mathcal{S})$)
Gegeben: *Eine Punktmenge S im $\mathbb{R}^2$*
Gesucht: *Ein größter leerer Kreis $\mathcal{LEC}(\mathcal{S})$*

Es ist das Voronoi-Diagramm $\mathcal{VD}(\mathcal{S})$ und das konvexe Polygon $ch(S)$, sowie deren Schnittpunkte zu bestimmen. Der größte leere Kreis mit einem Voronoi-Knoten oder einem ermittelten Schnittpunkt als Mittelpunkt entspricht $\mathcal{LEC}(\mathcal{S})$.

Aus den beiden Möglichkeiten für die Lage des Mittelpunktes des kleinsten umfassenden Kreises ergibt sich für dessen Konstruktion eine Fallunterscheidung. Entspricht der diametrale Kreis dem kleinsten umfassenden Kreis, wird die Ermittlung des furthest-point Voronoi-Diagramms $\mathcal{FVD}(\mathcal{S})$ nicht mehr benötigt.

Problem 4.2 (Konstruktion des $\mathcal{SEC}(\mathcal{S})$)
Gegeben: *Eine Punktmenge S im $\mathbb{R}^2$*
Gesucht: *Der kleinste umfassende Kreis $\mathcal{SEC}(\mathcal{S})$*

Es ist ein diametraler Kreis von S anhand des konvexen Polygons $ch(S)$ zu bestimmen. Liegen alle Orte aus S in diesem Kreis, ist $\mathcal{SEC}(\mathcal{S})$ gefunden. Anderenfalls wird das Voronoi-Diagramm $\mathcal{FVD}(\mathcal{S})$ bestimmt und der kleinste umfassende Kreis mit einem Voronoi-Knoten als Mittelpunkt entspricht $\mathcal{SEC}(\mathcal{S})$.

Für die Konstruktion des Rings mit minimaler Breite ist zu überprüfen, ob die Punktmenge eher kreisförmig oder linienförmig angeordnet ist. Sollte die Punktmenge eher linienförmig angeordnet sein, ist sie in einem Streifen enthalten, dessen Breite geringer als die Breite eines umfassenden Rings ist. Eine Kante der konvexen Hülle und einem von dieser Kante entgegengesetzten Punkt legt die Lage und Steigung des Streifens eindeutig fest. Bei der Ermittlung des am besten angepassten Kreis kann dessen Verwandtschaft mit dem Ring minimaler Breite genutzt werden. Die Konstruktion des Kreises baut direkt auf der Konstruktion des Ringes auf.

Die zu untersuchenden Schnittpunkte der Voronoi-Diagramme $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ könnten nach Lemma 3.10 unter Annahme der allgemeinen Lage dadurch eingeschränkt werden, dass nur solche als Mittelpunkt für einen Ring minimaler Breite dienen können, auf denen Orte aus S alternierend angeordnet sind. Die Schnittstellen sowie ein Punkt mit minimalen als auch ein Punkt mit maximalen Abstand zu diesen Schnittstellen sind auch unter dieser Einschränkung zu bestimmen. Zusätzlich ist die Lage mindestens zwei weiterer Orte zu analysieren. Die Breite eines ermittelten Rings hingegen kann in einer Operation berechnet und in einer weiteren mit dem bisherigen Minimum verglichen werden. Dementsprechend wird bei der Problembeschreibung für einen Algorithmus diese Einschränkung nicht weiter beachtet.

Problem 4.3 (Konstruktion des $\mathcal{MWA}(\mathcal{S})$ und des $\mathcal{BFC}(\mathcal{S})$)
Gegeben: *Eine Punktmenge S im $\mathbb{R}^2$*
Gesucht: *$\mathcal{MWA}(\mathcal{S})$ als Ring oder Streifen minimaler Breite*
Der zugehörige $\mathcal{BFC}(\mathcal{S})$ als Kreis oder Gerade

Es ist die Breite von S und der zugehörige Streifen anhand des konvexen Polygons $ch(S)$ zu bestimmen. Zudem sind die Voronoi-Diagramme $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ und deren Schnittpunkte zu ermitteln. Aus allen Ringen um diese Schnittpunkte als Mittelpunkte ist derjenige zu finden, der die minimale Breite besitzt. Ist dieser kleiner als die Breite von S, entspricht $\mathcal{MWA}(\mathcal{S})$ dem Ring minimaler Breite und $\mathcal{BFC}(\mathcal{S})$ dem am besten angepassten Kreis. Anderenfalls liegt $\mathcal{MWA}(\mathcal{S})$ in Form des ermittelten Streifen vor und $\mathcal{BFC}(\mathcal{S})$ als am besten angepasste Gerade.

Die genauere Formulierung der Approximations- und Optimierungsaufgaben besteht in einer Strukturierung in Teilprobleme. Diese Strukturierung wird bei der Angabe der Algorithmen genutzt und spiegelt sich in der Gliederung dieses Kapitels wider. Wir beginnen mit einem Unterkapitel über die Konstruktion der konvexen Hülle, das unter anderem grundlegende Eigenschaften der Lage eines Punktes relativ zu einem Liniensegment vorstellt. Da die konvexe Hülle zur Ermittlung des Durchmessers und der Breite einer Punktmenge dient, werden die zugehörigen Algorithmen ebenfalls an jener Stelle eingeführt. Das nächste Unterkapitel beinhaltet die Konstruktion der Voronoi-Diagramme $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$. Damit sind die Grundlagen für die zu ermittelnden Kreise und des Rings geschaffen, die in der Reihenfolge ihrer Problembeschreibung unterkapitelweise abgehandelt werden.

Bei den folgenden Algorithmen kommt unter anderem die algorithmische Technik des *Sweep* zum Einsatz. Das Sweep-Paradigma[1] kann zur Lösung einer Vielzahl von geometrischen Problemen eingesetzt werden. Eine abstrakte Definition von Sweep-Verfahren, die manchmal auch *Scan*-Verfahren genannt werden, ist unter [urlc] zu finden:
„Ein Sweep-Algorithmus verwandelt ein d-dimensionales geometrisches Problem in ein $(d-1)$-dimensionales, in dem es die d-te Dimension in eine zeitliche verwandelt. Dabei wird der d-dimensionale Raum, der die zu bearbeitenden Daten enthält, mit einer $(d-1)$-dimensionalen Geometrie"...„ , die quasi als „Scanner" fungiert, in einer zuvor festgelegten Richtung durchlaufen."

Allgemein können Sweep-Verfahren nicht nur in der Ebene (*Plane* sweep), sondern auch in höherdimensionalen Räumen (*Space* sweep) verwendet werden. In der vorliegenden Arbeit sind zweidimensionale Probleme von Interesse, die durch die Plane-Sweep-Technik in eine Folge von eindimensionalen Problemen transformiert werden. Wird für ein eindimensionales Problem eine Lösung gefunden, liegen die zugehörigen Problemanordnungen in der „Vergangenheit" und werden kein zweites Mal analysiert. Aus der Folge von Problemen wird eine Folge von Lösungen ge-

[1] Ein Paradigma ist in der Informatik eine algorithmische Technik.

neriert und so miteinander verknüpft, dass sich die Lösung des zweidimensionalen Problems ergibt. Eine praktische Umsetzung eines Plane-Sweep-Verfahrens beschreiben wir detailliert bei der Schnittpunktbestimmung der Voronoi-Diagramme $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$.

4.1 Konstruktion der konvexen Hülle

Die Konstruktion der konvexen Hülle einer ebenen Punktmenge beinhaltet die Bestimmung des konvexen Polygons $ch(S)$ durch Angabe seiner Ecken oder Kanten in einer Reihenfolge gegen den Uhrzeigersinn. Wir werden im Weiteren von der Eckendarstellung ausgehen. Um eine untere Schranke für die Laufzeit eines solchen Algorithmus zu erhalten, können wir das Sortierproblem auf das Problem der konvexen Hülle reduzieren.

Lemma 4.1 (Untere Schranke für die Zeitkomplexität im $\mathbb{R}^2$)
Die Konstruktion der konvexen Hülle einer n-elementigen Punktmenge im $\mathbb{R}^2$ erfordert $\Omega(n \log n)$ viel Rechenzeit.

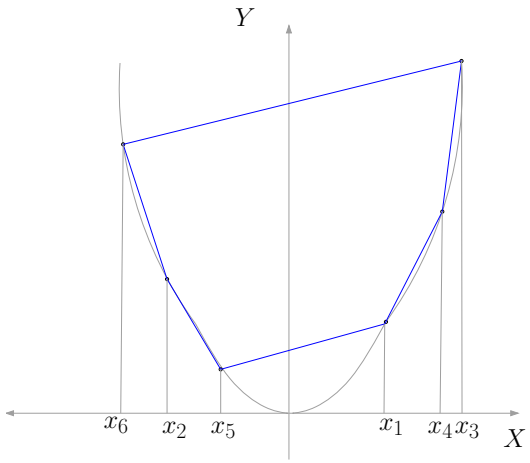


Abbildung 4.1: Die konvexe Hülle von sechs Punkten auf einer Parabel

Beweis:
Den Beweis führen wir analog zu [Kle97]. Es ist zu zeigen, dass für das Sortierproblem Π_{sort} im Bezug auf das Problem der konvexen Hülle Π_{conv} gilt: $\Pi_{sort} \propto \Pi_{conv}$.
Sei A ein beliebiger Algorithmus zur Konstruktion der konvexen Hülle einer n-elementigen Punktmenge im $\mathbb{R}^2$. Für n unsortierte reelle Zahlen $x_1, ..., x_n$ betrachten wir die Punktmenge $S := \{(x_1, x_1^2), ..., (x_n, x_n^2)\}$, die wir in linearer Zeit bestimmen können. S liegt auf einer Parabel und jeder Punkt (x_i, x_i^2) ist ein Eckpunkt der konvexen Hülle.
Nachdem wir über A das konvexe Polygon von S konstruiert haben, können wir in linearer Zeit seine Ecken entgegen dem Uhrzeigersinn ausgeben. Dabei beginnen

wir mit dem am weitesten links gelegenen Punkt und erhalten die Punktfolge mit aufsteigenden X-Koordinaten. Das Durchlaufen liefert uns in linearer Zeit eine Sortierung der Zahlen $x_1, ..., x_n$. Somit ist $\Pi_{sort} \propto \Pi_{conv}$. □

Für die Konstruktion der konvexen Hülle sind einige effiziente Verfahren mit der Laufzeit $O(n \log n)$ bekannt wie beispielsweise *Graham's Scan* und *inkrementelle* Algorithmen, die unter anderem in [Kle97] vorgestellt werden. Wird für eine sortierte Punktmenge die konvexe Hülle gesucht, ist der Laufzeitbedarf für die Sortierung $O(n \log n)$ und die konvexe Hülle kann dann in $O(n)$ gefunden werden. Wir werden an dieser Stelle keinen Algorithmus explizit beschreiben, sondern auf die Konstruktion des Voronoi-Diagramms $\mathcal{VD}(\mathcal{S})$ verweisen. Die Orte der Menge S, deren zugehörige Voronoi-Region unbeschränkt ist, liegen auf dem Rand der konvexen Hülle. Diese Eigenschaft nutzen wir aus, um mit Hilfe des $\mathcal{VD}(\mathcal{S})$ die konvexe Hülle in Linearzeit zu bestimmen.

Da wir Online-Algorithmen anstreben, ist eine inkrementelle bzw. dekrementelle Arbeitsweise von Interesse, wenn die konvexe Hülle einer Ausgangsmenge bereits konstruiert ist. Ist die konvexe Hülle $ch(S)$ ermittelt und durch das Polygon $P = (p_1, ..., p_n)$ gegeben, muss beim Einfügen eines Punktes q das Polygon dann neu ermittelt werden, wenn q außerhalb von P liegt. Der Sonderfall, dass q genau auf einer Kante der konvexen Hülle liegt, verändert zwar nicht die Form des Polygons, aber die Eckenfolge. Wird ein Punkt q aus S gelöscht, verändert sich P nur, falls $q \in \{p_1, ..., p_n\}$. Während das Einfügeereignis einen sogenannten *Point-in-Polygon-Test* erfordert, genügt für das Löschereignis ein *Elementtest*.

Für den Point-in-Polygon-Test betrachten wir zuerst die Lage eines Punktes q bezüglich eines Strahls s, der die Punkte p und r enthält. Wir wollen wissen, ob q rechts, links oder auf dem Strahl liegt. q liegt genau dann links von s, wenn die drei Punkte p, r, q gegen den Uhrzeigersinn orientiert sind. Analog gilt, dass q rechts von s liegt, wenn die drei Punkte p, r, q gegen den Uhrzeigersinn orientiert sind.

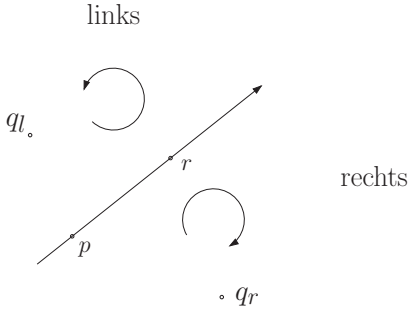


Abbildung 4.2: Orientierung von Punkten bezüglich eines Strahls

Falls q nicht auf dem Strahl s oder einer Verlängerung von s liegt, bilden die drei Punkte p, r, q ein Dreieck. Für die Bestimmung der Lage von q kann eine vorzeichenbehaftete Formel für eine Dreiecksfläche zur Hilfe genommen werden, die aus

der Determinante einer 3×3-Matrix resultiert. Da nicht der korrekte Wert, sondern das Vorzeichen und die Vergleichbarkeit der Werte ausschlaggebend für uns ist, berechnen wir das Doppelte der Dreiecksfläche, um die Division durch 2 zu sparen[2].

$$area(p,r,q) := \begin{vmatrix} p_1 & p_2 & 1 \\ r_1 & r_2 & 1 \\ q_1 & q_2 & 1 \end{vmatrix} = (r_1 - p_1)(q_2 - p_2) - (q_1 - p_1)(r_2 - p_2)$$

Dann gilt

$$area(p,r,q) \begin{cases} < 0 & \text{falls } q \text{ rechts von } pr \text{ liegt} \\ = 0 & \text{falls } q \text{ auf } pr \text{ liegt} \\ > 0 & \text{falls } q \text{ links von } pr \text{ liegt} \end{cases}$$

Die Orientierung eines Punktes bezüglich eines Strahls kann also in konstanter Zeit gefunden werden.

Mit Hilfe der *area*-Funktion kann nun überprüft werden, ob ein Punkt links eines Strahls liegt, also eine sogenannte *LeftOn*-Bedingung erfüllt. Für ein konvexes Polygon $P = (p_1, ..., p_n)$ gilt, dass ein Punkt q, der innerhalb von P liegt, links von jeder Kante des konvexen Polygons liegen muss. Sobald q von einer Kante rechts liegt, befindet q sich außerhalb von P. Für einen Point-in-Polygon-Test werden die Ecken p_i abgelaufen und die LeftOn-Bedingung solange überprüft, bis entweder alle p_i abgearbeitet oder die Bedingung unwahr ist.

Das Polygon sei in einer doppelt verketteten Liste gegeben, auf deren erstes Element mit $head(polygon)$ und auf das letzte Element mit $tail(polygon)$ zugegriffen wird. Außerdem kann der Vorgänger von p mit $before(p)$ und der Nachfolger mit $next(p)$ verarbeitet werden.

Algorithmus 1 PointInPolygon($polygon, q$)

```
if Anzahl der Ecken von polygon ≤ 2 then
  return false
p := head(polygon)
repeat
  führe LeftOn-Test für p, next(p) und q durch
  p := next(p)
until LeftOn-Test negativ or p = head(polygon)
return Ergebnis vom letzten LeftOn-Test
```

Der Zeitbedarf des vorgestellten Point-in-Polygon-Test ist aus $O(n)$. Wird ein konvexes Polygon P in einem *balancierten Suchbaum* verwaltet, kann die Laufzeit des Tests

[2]Der Flächeninhalt des Dreiecks p, r, q ist also $\frac{1}{2}|area(p,r,q)|$.

zu $O(\log n)$ verbessert werden. Dafür wird ausgenutzt, dass die p_i einer *angularen Ordnung* bezüglich eines Punktes im Innern von P unterliegen (siehe beispielsweise [Kar00]). Der Suchbaum hätte für einen Elementtest ebenfalls den Vorteil, dass dieser in $O(\log n)$ durchgeführt werden kann. Da wir jedoch für die Algorithmen zur Ermittlung des Durchmessers und der Breite eine *doppelt verkettete Liste* benötigen, wird dieser Performanceverlust in Kauf genommen. Bei den Tests könnte der Performanceverlust durch die Verwaltung zweier Datenstrukturen (der Liste und des Baums) für die konvexe Hülle umgangen werden. Sobald jedoch die konvexe Hülle sich verändert, wird die Anpassung beider Strukturen erforderlich. Der Performancegewinn bei den Tests steht also dem Performanceverlust bei Änderungen gegenüber.

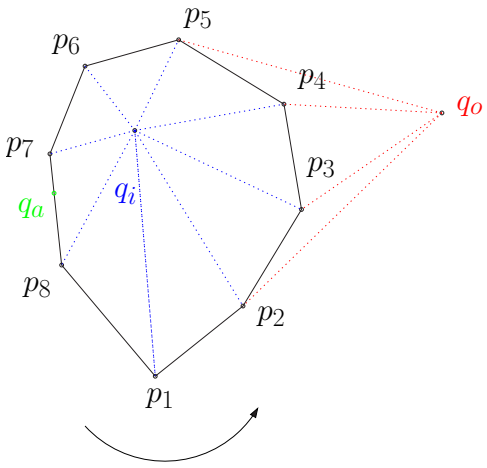


Abbildung 4.3: q_i innerhalb, q_a auf einer Kante und q_o außerhalb eines konvexen Polygons

Wir erweitern den Algorithmus *PointInPolygon*, der für alle konvexen Polygone einsetzbar ist, um eine Funktionalität zu dem Algorithmus *PointInConvHull*. Dieser setzt voraus, dass die konvexe Hülle einer Punktmenge als Polygon betrachtet wird.

Auf die Behandlung der Sonderfälle gehen wir später genauer ein. Liegt kein Sonderfall vor, prüfen wir, ob q sich außerhalb von $convHull$ oder auf einer Kante von $convHull$ befindet. Trifft dies zu, wird $convHull$ direkt angepasst. Betrachtet man die Abbildung 4.3, ist offensichtlich, dass durch das Einfügen von q gegebenenfalls einige benachbarte p_i nicht mehr auf dem Rand der konvexen Hülle liegen.

Sobald also eine LeftOn-Bedingung nicht mehr erfüllt ist, werden die folgenden p_i solange überprüft, bis die Bedingung sich wieder als wahr herausstellt. Damit sind die Punkte gefunden, die jeweils mit q zu der neuen konvexen Hülle verbunden werden. Sollte q auf einer Kante liegen, kann P ebenfalls direkt angepasst werden. Der Zeitbedarf des Algorithmus *PointInConvHull* ist also ebenfalls aus $O(n)$, da jede Ecke höchstens einmal besucht wird und die anderen Befehle in konstanter Zeit

durchgeführt werden können.

Algorithmus 2 PointInConvHull($convHull$, q)

```
Behandlung der Sonderfälle
qInside := true
r := head(convHull)
while LeftOn-Test für before(r), r und q nicht positiv and before(r) ≠
head(convHull) do
  r := before(r)
  qInside = false
end
p := head(convHull)
repeat
  führe LeftOn-Test für p, next(p) und q durch
  if LeftOn-Test positiv and qInside = false then
    setze q zwischen r und p
    return false
  else if LeftOn-Test nicht positiv and qInside then
    qInside := false
    r := p
  end
  p := next(p)
until p = head(convHull)
return true
```

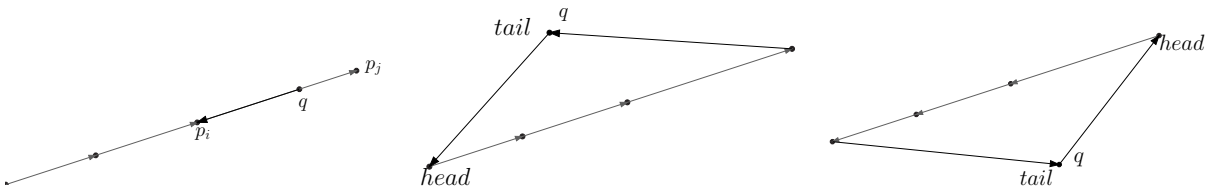
Wird der erste oder zweite Punkt in die Punktmenge eingefügt, kann der Aufbau von *convHull* ohne weitere Analyse erfolgen. Sollte es sich um eine kollineare Punktmenge handeln, wird *convHull* in dem Algorithmus *PointsCollinear* gefüllt. Die Variable *collinear* gibt entsprechend ihrer Bezeichnung wieder, ob eine Punktmenge in kollinearer Lage vorliegt.

Behandlung der Sonderfälle:

```
if head(convHull) = NULL then
    setze q als Kopf/Ende in convHull ein
    return true
end
if next(head(convHull)) = Null then
    setze q als zweiten Punkt geeignet in convHull ein
    collinear := true
    return false
end
if collinear and PointsCollinear(convHull, q) then
    return false
```

PointsCollinear wird nur dann aufgerufen, wenn alle Punkte bisher kollinear angeordnet sind, also ebenfalls, wenn die konvexe Hülle bisher nur aus einer Kante besteht.

Fällt der Test auf Kollinearität mit q positiv aus, wird den Punkten entlang gelaufen, an welcher Stelle q auf dem Liniensegment liegt. Dies können wir anhand einer Sortierung der Punkte nach ihren $X-$ und $Y-$Werten entscheiden. Sollte beim Durchlauf q nicht eingefügt worden sein, stellt dieser Punkt $tail(convHull)$ dar. Liegt q nicht kollinear, wird q in $convHull$ ergänzt und das Dreieck gegen den Uhrzeigersinn orientiert abgespeichert.



(a) q kollinear (b) q links (c) q rechts

Abbildung 4.4: Alternativen für die Lage von q bei kollinearer Ausgangsmenge

In Abbildung 4.4 wird als erstes ein Beispiel für den Test nach der passenden Einfügestelle gezeigt, falls q auf dem Liniensegment liegt. Das zweite Bild stellt die Situation dar, wenn q links liegt. Die dritte Situation zeigt $convHull$ nach dem Vertauschen der Orientierung in $convHull$, was aus der Lage von q resultiert.

Algorithmus 3 PointsCollinear($convHull$, q)

```
collinear := true
setze p auf next(head(convHull))
if area(before(p), p, q) = 0 then
  (* q ist kollinear *)
  repeat
    if q < p then
      setze q geeignet in convHull ein
    setze p auf next(p)
  until p = head(convHull) or q eingefügt
  if not q eingefügt then
    setze q als tail(convHull)
else
  (* q ist nicht kollinear *)
  if not LeftOn(head(convHull), tail(convHull), q) then
    vertausche Kopf/Ende und next/before von allen Punkten in convHull
  setze q als tail(convHull)
  collinear := false
end
return collinear
```

Wir betrachten nun die drei verschiedenen Ereignisse, die bezüglich einer Punktmenge auftreten können und die dann notwendigen Arbeitsschritte.

Das Einfügen eines Punktes q wird durch den Aufruf von *PointInConvHull* verarbeitet. Erfolgt das Löschen eines Punktes q aus der konvexen Hülle, wird überprüft, ob die Anzahl der Punkte in S größer als 3 ist oder die Punktmenge in kollinearer Lage vorliegt. In diesem Fall kann q aus $convHull$ ohne weitere Analyse gelöscht werden.

Anderenfalls gestaltet sich das Löschen eines Punktes von dem Rand des Polygons etwas komplizierter für die Ermittlung des neuen Polygons. Es bilden gegebenenfalls innere Punkte von $convHull$ neue Kanten. Stellt sich der Elementtest also als positiv heraus, wird das Polygon in einem getrennten Algorithmus neu ermittelt.

Reaktion auf **Ereignis 1** (Punkt q wird eingefügt)
  $inside$:=PointInConvHull($convHull, q$)

Reaktion auf **Ereignis 2** (Punkt q wird gelöscht)
  if Anzahl der Punkte in $S \leq 3$ **or** $collinear$ **then**
    löschen von q aus $convHull$
  else if $q \in convHullVertices$ **then**
    ConstructConvHull(S)

Reaktion auf **Ereignis 3** (Punkt q wird nach q' verschoben)
  Reaktion auf **Ereignis 2** für q
  Reaktion auf **Ereignis 1** für q'

Das Verschiebe-Ereignis werden wir bei weiteren Betrachtungen nur dann explizit aufführen, wenn die Vorgehensweise nicht der Reaktion auf ein Löschereignis von q und Einfügeereignis von q' entspricht.

Betrachtet man die Laufzeiten der Reaktionen auf die Ereignisse, liegen sie jeweils in $O(n)$. Werden die Punkte in einer ungünstigen Reihenfolge eingefügt und gelöscht, ergibt das ein Zeitverhalten aus $O(n^2)$ über die komplette Punktmenge.

4.1.1 Durchmesser und Breite einer Punktmenge

Wie bereits vorgestellt bilden entgegengesetzte Punkte auf dem Rand der konvexen Hülle die Grundlage für die Berechnung des Durchmessers und der Breite einer Punktmenge.

Wir nehmen an, dass für eine Punktmenge S die konvexe Hülle $ch(S)$ in Form eines konvexen Polygons $P = (p_1, ..., p_n)$ gegeben ist. Betrachtet man nun eine Ecke dieses konvexen Polygons, liegen deren entgegengesetzten Ecken in einer Folge. Das bedeutet, überprüft man die Ecken durch Abfragen gegen den Uhrzeigersinn und hat eine erste entsprechende Ecke gefunden, braucht man die Überprüfung der nächsten Ecken nur so lange fortsetzen, bis eine nicht mehr entgegengesetzt ist. Um diese

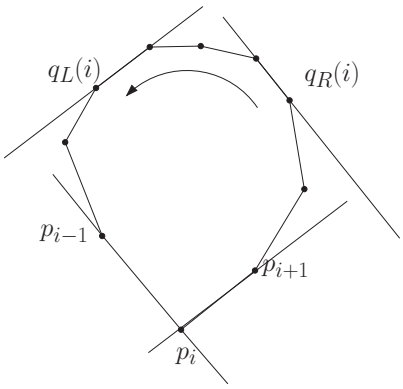


Abbildung 4.5: Folge von entgegengesetzten Punkten

Folge jeweils zu einem p_i zu finden, nutzen wir eine Beobachtung aus [PS85] und passen den dort vorgestellten Algorithmus auf unsere Belange an.

$q_R(i)$ sei der Startpunkt und $q_L(i)$ der Endpunkt der gesuchten Folge. P wird gegen den Uhrzeigersinn beginnend bei p_i durchlaufen und der Punkt, der am weitesten von der Kante $p_{i-1}p_i$ entfernt liegt, stellt $q_R(i)$ dar. Hat P parallele Kanten, wird der erste Punkt, der den größten Abstand hat, als Startpunkt übernommen. $q_L(i)$ wiederum ist analog ein Punkt mit größtem Abstand zu der Kante p_ip_{i+1}. Sollte in diesem Fall eine parallele Kante auftreten, entspricht $q_L(i)$ dem zweiten Punkt mit maximaler Distanz.

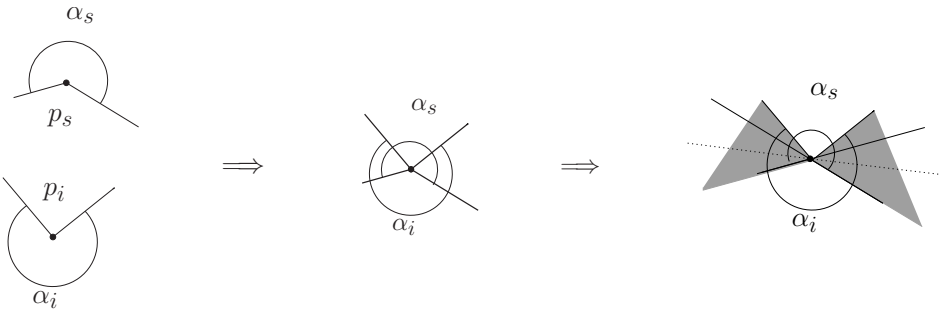


Abbildung 4.6: Bereiche für Geraden an entgegengesetzten Punkten

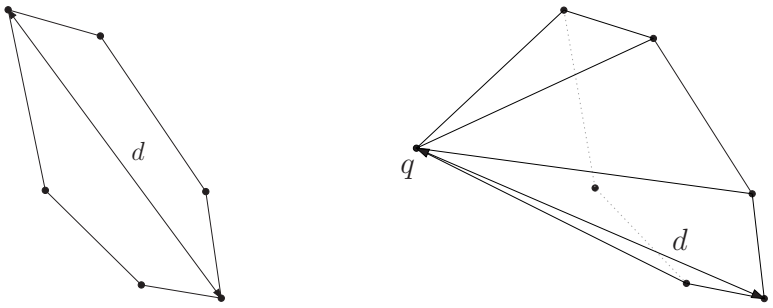
Jeder außen liegende Winkel eines konvexen Polygons ist größer π. Sei α_i der außen liegende Winkel von $p_{i-1}p_i$ und p_ip_{i+1}. In Abbildung 4.6 ist beispielhaft ein p_i mit einem entgegengesetzten Punkt p_s und dem zugehörigen Außenwinkel α_s angegeben. Verschiebt man nun p_i auf p_s, resultieren zwei Bereiche, durch die jeweils Stützgeraden an p_i bzw. p_s gelegt werden können. p_s ist nämlich genau dann entgegengesetzt von p_i, wenn es eine Gerade gibt, die durch den Durchschnitt der Bereiche von α_s und α_i führt. Aufgrund der Konvexität existiert eine solche Gerade nur für jedes Element der Folge ($q_R(i)$,...,$q_L(i)$), die gegen den Uhrzeigersinn orientiert ist (siehe [PS85]).

Ein Punkt mit maximalen Abstand zu einer Kante kann mit Hilfe der *area*-Funktion ermittelt werden[3]. Wandert man nun durch das Polygon von p_i und merkt sich die aktuelle Position in q_j, können die Flächeninhalte der Dreiecke p_{i-1}, p_i, q_j und p_{i-1}, p_i, q_{j+1} verglichen werden. Sobald das Dreieck mit einer Ecke q_{j+1} nicht größer ist als das mit einer Ecke qj, ist $q_R(i)$ gefunden. Analog wird die Ermittlung von $q_L(i)$ durchgeführt.

Damit sind die Grundlagen für einen Algorithmus zur Verwaltung entgegengesetzter Punkte geschaffen. Der nächste Schritt ist, den Durchmesser bzw. die Breite einer Punktmenge S anhand der entgegengesetzten Punkte zu bestimmen.

Zuerst unterscheiden wir nach der Kardinalität von S. Enthält S nur einen Punkt, entspricht dieser seiner konvexen Hülle und es kann trivialerweise kein entgegengesetzter Punkt gefunden werden. Sowohl der Durchmesser als auch die Breite haben den Wert 0. Besteht S aus zwei oder drei Elementen, sind diese jeweils entgegengesetzt. Im Fall von zwei Elementen existiert ein entgegengesetztes Punktepaar, deren Distanz gleich dem Durchmesser ist. Die Breite beträgt immer noch 0.

Im Fall von drei Elementen, die nicht kollinear angeordnet sind, existieren drei entgegengesetzte Punktepaare, deren Distanzen jeweils für den Durchmesser zu berechnen sind. Auch für die Breite sind drei Abstände zwischen Kanten und Punkten zu überprüfen. Eine Menge, deren Punkte alle kollinear angeordnet sind, hat die Breite 0 und den Abstand der Endpunkte des Liniensegments, das die konvexe Hülle darstellt, als Durchmesser. Trifft dieser Fall nicht zu und enthält S mehr als drei Elemente, können Aussagen über die beteiligten Punkte nicht mehr allgemein getroffen werden.



(a) Menge S (b) Menge $S \cup q$

Abbildung 4.7: Durchmesser einer Punktmenge

Für den Durchmesser von S wird allgemein das Maximum der Distanzen zwischen entgegengesetzten Punkten bestimmt. Der Durchmesser kann sich also immer dann ändern, wenn die konvexe Hülle angepasst wird. Sei für S der Durchmesser

[3]Der Flächeninhalt eines Dreiecks ist von einer Kante und der zugehörigen Höhe abhängig.

bekannt. Fügen wir nun einen Punkt q zu S hinzu, kann dies den Durchmesser nicht beeinflussen, falls q innerhalb der konvexen Hülle liegt. Anderenfalls sind für q die entgegengesetzten Punkte und die entsprechenden Distanzen zu q zu ermitteln. Es sind nicht alle Distanzen entgegengesetzter Punkte zu analysieren, da sich diese zwischen den Punkten aus $S \setminus q$ nicht verändern.

Der Algorithmus *Diameter* ermittelt in linearer Zeit die Veränderung des Durchmessers beim Einfügen eines Punktes q. In der Datenstruktur d wird der Durchmesser, ein zugehöriges Punktepaar und das Feld $inDiameterCircle$ abgelegt.

Algorithmus 4 Diameter($polygon, d, q$)

```
if Anzahl der Ecken von polygon = 1 then
  init(d)
  return
else if Anzahl der Ecken von polygon = 2 or alle Punkte kollinear then
  SetDiameter(d, head(polygon), tail(polygon))
  return
end
r := next(head(polygon))
p := before(q)
while area(p, next(p), next(r)) > area(p, next(p), r) do
  r := next(r)
p := q
SetDiameter(d, p, r)
while area(p, next(p), next(r)) > area(p, next(p), r) do
  r := next(r)
  SetDiameter(d, p, r)
end
(* Kanten sind parallel *)
if area(p, next(p), next(r)) = area(p, next(p), r) then
  SetDiameter(d, p, r)
end
```

Der Algorithmus *SetDiameter* verwaltet d in konstanter Zeit. Dass d während des sukzessiven Aufbaus von S nicht nur zu Beginn gleich der leeren Menge sein kann, wird deutlich, wenn die einzelnen Ereignisse für S vorgestellt werden. Sollte ein weiteres Paar mit dem Abstand des Durchmessers zueinander auftreten, wird direkt überprüft, ob dieses in dem diametralen Kreis liegt, der durch das bereits gefundene Punktepaar definiert wird. Sollte dies nicht zutreffen, benötigen wir für den kleinsten umfassenden Kreis keine Analyse, ob dieser einem diametralen Kreis entspricht und $inDiameterCircle$ wird auf $false$ gesetzt.

Algorithmus 5 SetDiameter(d, p, r)

```
if distance(p, r) > diameter or d = ∅ then
  insertD(distance(p, r), (p, r))
  inDiameterCircle := true
else if distance(p, r) = diameter then
  couple := getCouple(d)
  (* Mittelpunkt des diametralen Kreises des schon gefundenen Punktepaares *)
  m := center(couple(p1), couple(p2))
  if distance(m, p) * 2 > diameter or distance(m, r) * 2 > diameter then
    inDiameterCircle := false
  end
end
```

Für die Breite ist das Minimum der Distanzen zwischen einer Kante der konvexen Hülle und einem entgegengesetzten Punkt ausschlaggebend. Dies impliziert, dass beide Ecken der Kante entgegengesetzt dem Punkt liegen müssen. Die Breite der Punktmenge braucht beim Einfügen von q nur angepasst zu werden, falls q nicht in einem Streifen minimaler Breite liegt. Betrachten wir die Abbildung 4.8, ist für q_3 keine Anpassung notwendig, allerdings für q_1 und q_2.

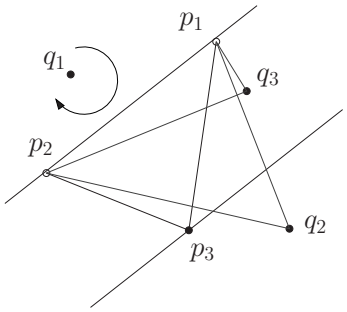


Abbildung 4.8: Streifen minimaler Breite mit einzufügenden Punkten

Wir benutzen einen *Point-In-Slab-Test*, um die Lage der Punkte zu analysieren. Seien p_1 und p_2 Punkte auf der beteiligten Kante der konvexen Hülle und p_3 der entsprechende entgegengesetzte Punkt. Dann liegt ein Punkt q innerhalb des definierten Streifens, wenn er folgende Bedingungen erfüllt:

$$\mathit{LeftOn}(p_1, p_2, q) \textbf{ and } \mathit{area}(p_1, p_2, p_3) > \mathit{area}(p_1, p_2, q))$$

In unserem Beispiel treffen beide Bedingungen für q_3 zu. q_1 wiederum erfüllt nicht die erste Bedingung und q_2 nicht die zweite Bedingung.

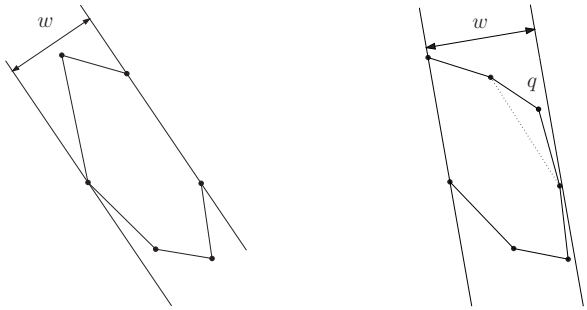
Die Datenstruktur w enthält sowohl die Breite als auch eine Liste der Punktetripel, die jeweils einen Streifen minimaler Breite definieren. In den Algorithmus *PointInSlab* prüfen wir nicht nur die Liste der Punktetripel in w, sondern aktualisieren sie auch. Sollten mehrere Streifen minimaler Breite vorliegen, sind diejenigen

nicht mehr zu verwalten, in denen q nicht liegt. Der Algorithmus benötigt in dem Spezialfall regelmäßiger Polygone lineare Zeit, ansonsten konstante Zeit.

Algorithmus 6 PointInSlab($polygon, w, q$)

```
qInside := false
if Anzahl der Ecken von polygon ≤ 3 or w = ∅ then
  return false
p := head(w)
repeat
  if Point-In-Slab-Test zwischen aktuellen tripel und q positiv then
    qInside := true
  else
    delete tripel from w
  p := next(p)
until alle Tripel in w durchlaufen
return qInside
```

Im Gegensatz zum Durchmesser erfordert jede Veränderung eines Punktes q auf dem Rand der konvexen Hülle, die nicht das Einfügen von q innerhalb eines Streifens minimaler Breite betrifft, die Analyse des kompletten Polygons für die Angabe der Breite.



(a) Menge S (b) Menge $S \cup q$

Abbildung 4.9: Breite einer Punktmenge

In dem Algorithmus *DiameterWidth* werden sowohl der Durchmesser als auch die Breite des kompletten Polygons berechnet. Die entgegengesetzten Punkte lassen sich in einer Folge ermitteln, da immer der entfernteste Punkt zu einer Kante $\overline{pnext(p)}$ gesucht wird. Ist der entfernteste Punkt gefunden, hat man den Endpunkt für die Folge der entgegengesetzten Punkte von p und den Startpunkt für die Folge des Nachfolgers von p. Der Sonderfall der parallelen Kanten ist dann jeweils noch explizit zu prüfen. Das Polygon wird mit r einmal durchlaufen und mit p dann wiederum nur bis zum ersten gefundenen r.

Dadurch werden entgegengesetzte Punktepaare nicht doppelt analysiert und die

Laufzeit von dem Algorithmus *DiameterWidth* liegt in $O(n)$.

Es fallen drei Stellen in *DiameterWidth* auf, bei denen für die Ermittlung von Durchmesser und Breite verschieden vorgegangen wird. Die Breite wird in der zweiten *while*-Schleife nur dann berechnet, wenn r eine entgegengesetzte Kante von $\overline{pnext(p)}$ ist. Hingegen wird die Breite in der dritten *while*-Schleife immer ermittelt, da beim Betreten dieser Schleife jeweils eine entgegengesetzte Kante zu p gefunden ist. Die Verarbeitung des Durchmessers ist dann notwendig, wenn p und r nicht mehr dem ersten gefundenen Punktepaar, das bereits verarbeitet wurde, entsprechen. Bei der Abfrage der parallelen Kanten wird nur der Durchmesser angepasst. Es ist nicht notwendig, dass die Breite anhand des nächsten Dreiecks untersucht wird, da die Parallelität der Kanten gewährleistet, dass dieser Abstand schon bei der Analyse der vorherigen Distanz Kante zu Punkt beachtet wurde. Die Distanz Punkt zu Punkt wiederum kann für parallele Kanten unterschiedlich sein.

Algorithmus 7 DiameterWidth($polygon, d, w$)

```
p := tail(polygon)
r := next(p)
Behandlung der Sonderfälle
while area(p, next(p), next(r)) > area(p, next(p), r) do
  r := next(r)
r_0 := r
while r ≠ head(polygon) do
  p := next(p)
  if area(p, next(p), r) > area(p, next(p), before(r)) then
    SetWidth(w, before(p), p, r)
  SetDiameter(d, p, r)
  while area(p, next(p), next(r)) > area(p, next(p), r) do
    r := next(r)
    SetWidth(w, before(r), r, p)
    if (p, r) ≠ (r_0, head(polygon)) then
      SetDiameter(d, p, r)
  end
  if area(p, next(p), next(r)) = area(p, next(p), r) then
    if (p, r) ≠ (r_0, tail(polygon)) then
      SetDiameter(d, p, r)
end
```

Die Behandlung der Sonderfälle in *DiameterWidth*, falls das Polygon weniger als drei Punkte hat oder alle Punkte kollinear liegen, haben wir wegen der Übersichtlichkeit im Folgenden getrennt aufgeführt.

Behandlung der Sonderfälle
  if Anzahl der Ecken von *polygon* ≤ 2
  or alle Punkte kollinear **then**
    init(w)
    if Anzahl der Ecken von *polygon* ≤ 1 **then**
      init(d)
    else
      SetDiameter($d, head(polygon), tail(polygon)$)
    end
  end

Der Algorithmus *SetWidth* beansprucht konstante Zeit. Analog zu *SetDiameter* können Fälle auftreten, in denen w gleich der leeren Menge ist. Dies läßt sich anhand der Ereignisse für einen Punkt q nachvollziehen. Sollte die Breite der Punktmenge bisher 0 betragen haben, ist die Breite der übergebenen Punkte immer aktuell.

Algorithmus 8 SetWidth(w, p, r, q)
  if $distance((p,r),q) < width$ **or** $width = 0$ **or** $w = \emptyset$ **then**
    init(w)
    insertWidth($w, distance((p,r),q)$
    insert($w, (p,r,q)$)
  else if $distance((p,r),q) = width$ **then**
    insert($w, (p,r,q)$)
  end

Zur Verwaltung des Durchmessers und der Breite benutzen wir die Datenstrukturen d und w. d führt jeweils den aktuellen Durchmesser und wird angepasst, falls sich die Liste der Punkte der konvexen Hülle durch ein Ereignis verändert. Die Breite einer Punktmenge kann sich beim Einfügen eines Punktes q verändern, falls q nicht in dem Streifen von $S \setminus q$ liegt. Beim Löschen von q ist seine Position ebenfalls zu prüfen. Sollte q auf dem Rand der konvexen Hülle liegen, sind d und w für das komplette Polygon zu ermitteln.

Reaktion auf **Ereignis 1** (Punkt q wird eingefügt)
  if not PointInConvHull($ConvHull, q$) **then**
    if PointInSlab($convHull, w, q$) **then**
      Diameter($convHull, d, q$)
    else
      DiameterWidth($convHull, d, w$)
  end

Reaktion auf **Ereignis 2** (Punkt q wird gelöscht)
if Anzahl der Punkte in $S \leq 3$
or *collinear*
or $q \in convHullVertices$ **then**
init(d), init(w)
DiameterWidth($convHull, d, w$)
end

4.2 Konstruktion der Voronoi-Diagramme $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$

Für die Konstruktion der Voronoi-Diagramme $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ gehen wir zuerst auf den Zusammenhang zu den Delaunay-Triangulationen $\mathcal{DT}(\mathcal{S})$ und $\mathcal{FDT}(\mathcal{S})$ ein. Seien $\mathcal{DT}(\mathcal{S})$ und $\mathcal{FDT}(\mathcal{S})$ durch eine Liste von Dreiecken gegeben. Die jeweils dualen Voronoi-Knoten sind anhand von Kreisen, die durch die Dreiecke definiert werden, zu finden. Sei (p, q, r) ein definierendes Punktetripel eines Delaunay-Dreiecks. Der Schnittpunkt zweier Bisektoren $B(p, q)$ und $B(q, r)$, der durch Lösen eines zweidimensionalen linearen Gleichungssystems berechnet werden kann, legt den Mittelpunkt eines umschließenden Kreises fest.

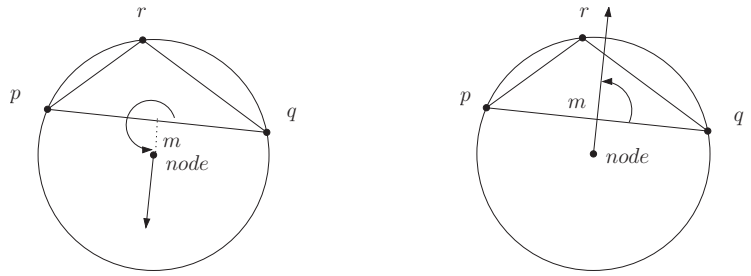
Allgemein lässt sich der Schnittpunkt von zwei Liniensegmenten $\overline{pq} = \{p + a(q - p); a \in \mathbb{R} \text{ mit } 0 \leq a \leq 1\}$ und $\overline{uv} = \{u + b(v - u); b \in \mathbb{R} \text{ mit } 0 \leq b \leq 1\}$ über die Gleichung $p + a(q - p) = u + b(v - u)$ ermitteln. Setzt man jeweils die Koordinaten von p, q, u und v ein, erhält man zwei Gleichungen mit den Unbekannten a und b. Diese können so aufgelöst werden, dass die Berechnungen unter *IntersectSegments* die Koordinaten für den Schnittpunkt s bestimmen. Für Liniensegmente liegen die Werte von a und b zwischen 0 und 1. Sollten die Segmente parallel zueinander liegen, hat D den Wert 0. Dieser Fall muss natürlich bei der Implementation abgefangen werden, damit keine Division durch 0 vorgenommen wird.

IntersectSegments($\overline{pq}, \overline{uv}$):
$D := (p_1 - q_1)(v_2 - u_2) - (u_1 - v_1)(p_2 - q_2)$
$a := (p_1(v_2 - u_2) + u_1(p_2 - v_2) + v_1(u_2 - p_2))/D$
$b := (p_1(u_2 - q_2) + q_1(p_2 - u_2) + u_1(q_2 - p_2))/D$
$s_1 := p_1 + a(q_1 - p_1)$
$s_2 := p_2 + a(q_2 - p_2)$

Die Kante $\overline{pq}$ induziert eine Gerade g in der Richtung p nach q. Sei m der Mittelpunkt auf der Kante zwischen p und q. Der Bisektor $B(p, q)$ führt durch m und stellt die um $\frac{\pi}{2}$ bzw. $\frac{3\pi}{2}$ gedrehte Gerade g um m dar. Darüber läßt sich der Schnittpunkt s zweier Bisektoren eines Dreiecks p, q, r folgendermaßen berechnen.

CircleCenter(p, q, r):
$D := (p_1 - q_1)(q_2 - r_2) - (q_1 - r_1)(p_2 - q_2)$
$a := ((p_1 - q_1)(p_1 + q_1) + (p_2 - q_2)(p_2 + q_2))/2D$
$b := ((q_1 - r_1)(q_1 + r_1) + (q_2 - r_2)(q_2 + r_2))/2D$
$s_1 := a(q_2 - r_2) - b(p_2 - q_2)$
$s_2 := b(p_1 - q_1) - a(q_1 - r_1)$

Die Koordinaten von s entsprechen den Koordinaten des Voronoi-Knoten *node*, der zu den Punkten p, q, r gehört.



(a) $\mathcal{VD}(\mathcal{S})$ (b) $\mathcal{FVD}(\mathcal{S})$

Abbildung 4.10: Orientierung der Strahlen bei Voronoi-Diagrammen

Für die Ermittlung der Voronoi-Kanten wiederum werden die Kanten jedes Dreiecks durchlaufen. Besitzt eine Kante des Dreiecks ein Nachbardreieck, werden die beiden zugehörigen Voronoi-Knoten durch eine Voronoi-Kante miteinander verbunden. Besitzt eine Kante des Dreiecks kein Nachbardreieck, liegt eine Kante der konvexen Hülle vor und die duale Voronoi-Kante ist ein Strahl. Dieser hat als Startpunkt den Voronoi-Knoten *node* und stellt eine Teilmenge von $B(p, q)$ dar. In welcher Richtung der Strahl orientiert ist, hängt davon ab, welches Voronoi-Diagramm ermittelt wird. Wird g um $\frac{\pi}{2}$ gegen den Uhrzeigersinn um den Mittelpunkt m der Kante $\overline{pq}$ gedreht, ist dies die Orientierung für einen Strahl des $\mathcal{FVD}(\mathcal{S})$. Eine Drehung um $\frac{3\pi}{2}$ legt den Strahl für $\mathcal{VD}(\mathcal{S})$ fest. Abbildung 4.10 zeigt dies beispielhaft für eine Kante.

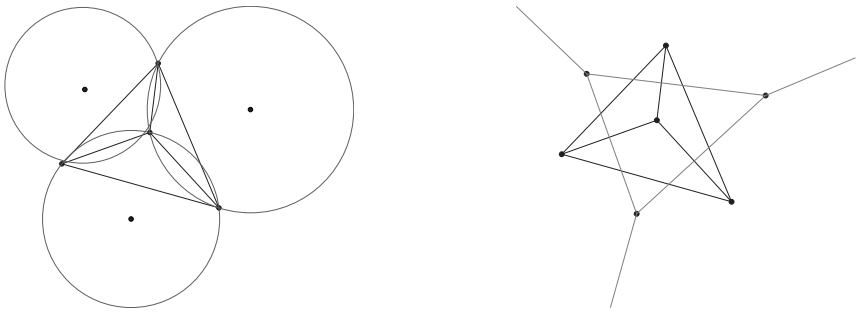
Damit sind die Voraussetzungen geschaffen, aus einer Delaunay-Triangulation das duale Voronoi-Diagramm zu konstruieren. Eine Voronoi-Kante wird entweder über zwei Punkte oder über einen Startpunkt und einen Richtungsvektor einer Geraden festgelegt. Als Parameter wird jeweils die Delaunay-Triangulation und der Winkel, um den eine Kante ohne Nachbardreieck gedreht werden muss, übergeben. Falls zwei Dreiecke den gleichen Umkreis besitzen, müssen sie benachbart sein und initiieren keine Voronoi-Kante.

Zwei Voronoi-Knoten werden nur dann zu einer Voronoi-Kante verbunden, falls das Nachbardreieck noch nicht verarbeitet wurde. Ansonsten würde die entsprechende Kante doppelt ermittelt. Außerdem ist abzuprüfen, ob ein degenerierter Fall vorliegt, bei dem mehr als drei Elemente der Punktmenge auf einem Umkreis um ein Dreieck liegen. Auch dann entsteht keine neue Voronoi-Kante. Beispiele für die

Konstruktion der Voronoi-Diagramme zeigen die Abbildungen 4.11 und 4.12.

Algorithmus 9 ConstructVD($dt, angle$)

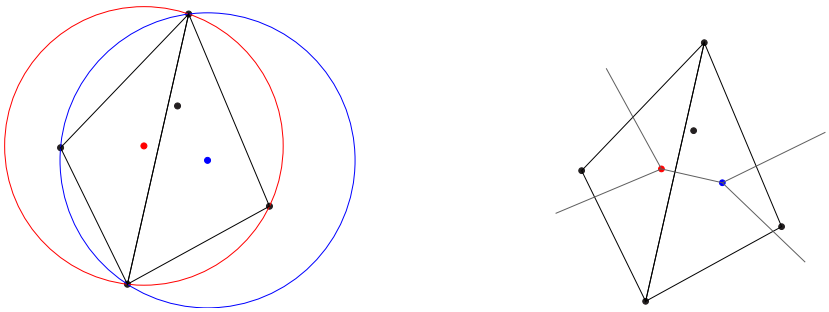
```
tria := first(dt)
while noch nicht alle Dreiecke von dt besucht do
  node := CircleCenter(tria(p1), tria(p2), tria(p3))
  for alle drei Kanten des aktuellen Dreiecks do
    tria_n := Nachbardreieck der aktuellen Kante e
    if tria_n ≠ ∅ and tria_n noch nicht besucht then
      (* die aktuelle Kante e liegt innerhalb der konvexen Hülle *)
      if node ≠ CircleCenter(tria_n(p1), tria_n(p2), tria_n(p3)) then
        (* tria und tria_n haben nicht den gleichen Umkreis *)
        edge := (node, CircleCenter(tria_n(p1), tria_n(p2), tria_n(p3)))
    else if tria_n = ∅ then
      (* die aktuelle Kante e ist eine Kante der konvexen Hülle *)
      edge := (node, rotate(e, angle))
    end
  end
  tria := next(tria)
end
```



(a) (b)

Abbildung 4.11: Konstruktion des $\mathcal{VD}(\mathcal{S})$ aus der $\mathcal{DT}(\mathcal{S})$

Somit können aus der Liste der Delaunay-Dreiecke jeweils die dualen Voronoi-Diagramme in Linearzeit konstruiert werden. Die untere Schranke für die Zeitkomplexität der Konstruktion dieser beiden geometrischen Datenstrukturen ist also gleich, falls sie größer als $\Omega(n)$ ist. Als Problem mit bekannter Komplexität wird an dieser Stelle die Konstruktion der konvexen Hülle in $O(n \log n)$ genutzt. Dieses Problem läßt sich in Linearzeit auf die Konstruktion von Delaunay-Triangulationen reduzieren.



(a) (b)

Abbildung 4.12: Konstruktion des $\mathcal{FVD}(\mathcal{S})$ aus der $\mathcal{FDT}(\mathcal{S})$

Lemma 4.2 (Untere Schranke für die Zeitkomplexität im $\mathbb{R}^2$)
Sei S eine n-elementige Menge im $\mathbb{R}^2$. Dann ist $\Omega(n \log n)$ die untere Schranke für die Zeitkomplexität der Konstruktion von $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ bzw. $\mathcal{DT}(\mathcal{S})$ und $\mathcal{FDT}(\mathcal{S})$.

Beweis:
Es ist zu zeigen, dass für das Problem der konvexen Hülle Π_{conv} im Bezug auf das Problem der Delaunay-Triangulationen Π_{del} gilt: $\Pi_{conv} \propto \Pi_{del}$.
Wir gehen davon aus, dass das Voronoi-Diagramm $\mathcal{VD}(\mathcal{S})$ implizit durch die Delaunay-Triangulation $\mathcal{DT}(\mathcal{S})$ gegeben ist. Die Anzahl der Dreiecke liegt in $O(n)$. Diese werden in der Art in einer Liste verwaltet, dass zu jeder Kante das Nachbardreieck bekannt ist. Existiert kein Nachbardreieck, ist diese Kante entsprechend gekennzeichnet und stellt zudem eine Kante der konvexen Hülle dar. Das Anfangselement der Liste ist ein Dreieck, das eine Kante der konvexen Hülle enthält. Wird diese Liste gegen den Uhrzeigersinn durchlaufen, wird jeweils die nächste Kante, die nur ein Dreieck berandet, berichtet. Durch diese Vorgehensweise werden die Kanten der konvexen Hülle gegen den Uhrzeigersinn in Linearzeit ausgegeben. Somit ist $\Pi_{conv} \propto \Pi_{del}$.
Bei einem beliebigen Algorithmus B zur Bestimmung der Delaunay-Triangulation $\mathcal{FDT}(\mathcal{S})$ benötigen wir keine Unterscheidung der Kanten der Dreiecke. Jeder Punkt eines Dreiecks der $\mathcal{FDT}(\mathcal{S})$ liegt auf dem Rand der konvexen Hülle. Werden also alle definierenden Punkte der Dreiecke gegen den Uhrzeigersinn berichtet, ohne Punkte doppelt anzugeben, enthält die Ausgabe das konvexe Polygon $ch(S)$. Somit ist auch für $\mathcal{FDT}(\mathcal{S})$ die Aussage $\Pi_{conv} \propto \Pi_{del}$ bewiesen. □

Die Vorgehensweise aus dem oben angeführten Beweis realisiert der Algorithmus *ConstructConvHull* mit S und $\mathcal{DT}(\mathcal{S})$ als Parameter. Wir haben $\mathcal{DT}(\mathcal{S})$ als Grundlage für den Algorithmus gewählt, da wir beide Delaunay-Triangulationen zu bestimmen haben. Für die Konstruktion von $\mathcal{FDT}(\mathcal{S})$ kann dann direkt nur der Rand der konvexen Hülle anstatt die komplette Punktmenge übergeben werden. In *ConstructConvHull* wird davon ausgegangen, dass die Dreiecke gegen den Uhrzei-

gersinn orientiert sind. Außerdem werden jeweils diejenigen gegen den Uhrzeigersinn durchlaufen, die mindestens einen Punkt mit dem Rand der konvexen Hülle gemeinsam haben. In Abbildung 4.13 sind beispielhaft die Dreiecke grau markiert, die mit dem Algorithmus durchlaufen werden.

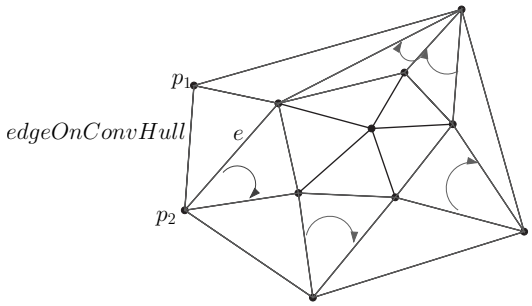


Abbildung 4.13: Konstruktion der konvexen Hülle

Sind die Dreiecke in der Orientierung nicht gegeben, kann dies durch die *area*-Funktion gewährleistet werden. Sollte die Datenstruktur von $\mathcal{DT}(\mathcal{S})$ nicht als erstes Element ein Dreieck auf dem Rand der konvexen Hülle enthalten, wird diese solange durchsucht, bis das erste Dreieck gefunden ist. Diese ergänzenden Schritte belassen die Laufzeit von *ConstructConvHull* in $O(n)$.

Algorithmus 10 ConstructConvHull(S)

```
ConstructDT(S)
convHull := ()
(* erstes Dreieck auf dem Rand der konvexen Hülle *)
tria := first(dt)
repeat
  for alle 3 Kanten von tria do
    tria_n := Nachbardreieck der aktuellen Kante e
    if tria_n = ∅ then
      edgeOnConvHull := e
      (* einfügen eines Punktes von edgeOnConvHull in die konvexe Hülle *)
      InsertSequence(convHull, edgeOnConvHull(p_1))
    end
  end
  suche Kante e in tria mit edgeOnConvHull(p_2) als Startpunkt
  tria := Nachbardreieck der Kante e
until alle Dreiecke auf dem Rand der konvexen Hülle besucht
```

Die untere Schranke für die Konstruktion von $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ beträgt nach Lemma 4.2 also $\Omega(n \log n)$. Es sind einige effiziente Verfahren bekannt, deren Lauf-

zeit in $O(n \log n)$ liegt. Eine Auswahl davon für $\mathcal{VD}(\mathcal{S})$ bzw. $\mathcal{DT}(\mathcal{S})$ wird in [Kle97] vorgestellt. Diese umfasst unter anderem einen *Divide-and-Conquer*-Algorithmus von Shamos und Hoey [SH75][4], einen *Sweep*-Algorithmus nach Fortune mit einer Sweep-Line als Scanner-Geometrie [For86] und *inkrementelle* Verfahren, die durch Randomisierung eine erwartete Laufzeit von $O(n \log n)$ aufweisen.

In den Artikeln [AKSS] und [AGSS87] zeigen die Autoren Möglichkeiten zum Ermitteln des $\mathcal{FVD}(\mathcal{S})$ in optimaler Laufzeit auf. In dem erstgenannten wird ein *Sweep*-Algorithmus mit einem sich zusammenziehenden Sweep-Circle eingesetzt. Der zweite kann für ein $\mathcal{FVD}(\mathcal{S})$ erweitert werden und arbeitet nach dem *Divide-and-Conquer*-Prinzip.

In dem *Geometrie-Labor* der FernUniversität Hagen [urld] sind in den Applets *VoroGlide* und *VoroFarthestPoint* Algorithmen zur Konstruktion von $\mathcal{DT}(\mathcal{S})$ bzw. $\mathcal{FDT}(\mathcal{S})$ realisiert. Da die aus dieser Arbeit resultierende Implementation *FitCircle* ebenfalls dort eingebunden wird und dessen Ressourcen nutzen kann, beschreiben wir nur kurz die Vorgehensweise der zugrundeliegenden Verfahren und verweisen auf entsprechende Literatur. *ConstructVD* und *ConstructConvHull* haben wir aus dem Grund so detailliert vorgestellt, da anhand dieser Algorithmen die von uns gewählten Datenstrukturen gefüllt werden und in der Art noch nicht in dem *Geometrie-Labor* vorliegen.

Der in dem Applet *VoroGlide* realisierte Algorithmus erstellt das $\mathcal{DT}(\mathcal{S})$ inkrementell. Wird ein Punkt p in der Art hinzugefügt, dass er außerhalb der durch die Delaunay-Dreiecke definierten Kreise liegt, ist ein neues Delaunay-Dreieck hinzuzufügen. Dies kann nur dann auftreten, wenn p sich außerhalb der konvexen Hülle befindet. Liegt p innerhalb eines Kreises, steht p also mit einem Dreieck in Konflikt, sind zwei Aufgaben zu erledigen. Es sind die neuen Delaunay-Kanten zu p zu bestimmen und zusätzlich die Dreiecke umzubauen, die mit p in Konflikt stehen. Dies geschieht anhand von zwei Regeln, deren Richtigkeit in [Kle97] bewiesen wird.

1. Sei $tria(q, s, r)$ ein Delaunay-Dreieck, in dessen Umkreis der Punkt p enthalten ist. Dann ist jedes Liniensegment $\overline{pq}$, $\overline{ps}$ und $\overline{pr}$ eine Delaunay-Kante.

2. Sei $tria(q, t, r)$ ein Delaunay-Dreieck, dessen Umkreis den Punkt p zwischen seinem Rand und der Kante $\overline{qr}$ enthält. Dann ist $\overline{qr}$ keine Delaunay-Kante mehr.

Aufgrund dieser Regeln werden die Delaunay-Kanten über Kreuz getauscht und ersetzt. Diese Aktion wird als *edge flip* bezeichnet.

In dem Algorithmus von *VoroGlide* wird ausgenutzt, dass die Anzahl der Konfliktdreiecke durch den Grad g von p eingeschränkt ist. Es gibt keine Konfliktdreiecke mehr, wenn alle Nachbardreiecke von den Dreiecken, die p als gemeinsamen

[4] In [Kle97] wird die Entdeckung dieses Algorithmus als Auslöser für die Entwicklung des Fachgebiets Algorithmische Geometrie genannt.

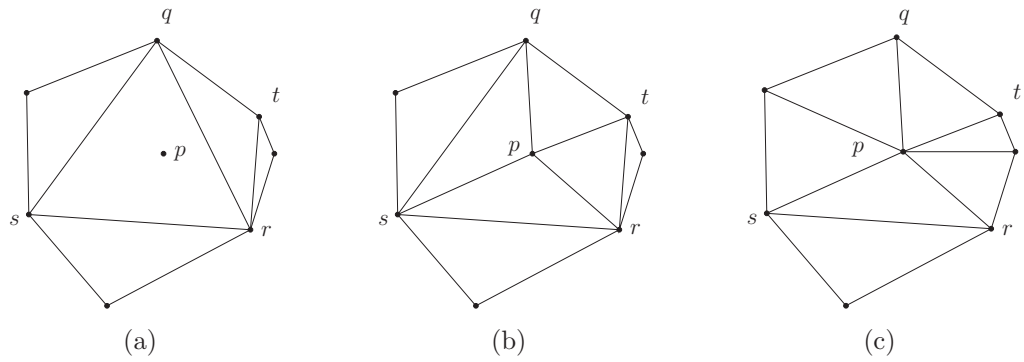


Abbildung 4.14: Umbauen der Delaunay-Dreiecke nach Einfügen von p

Eckpunkt haben, nicht mit p in Konflikt stehen. Damit läßt sich die neue Triangulation in der Zeit $O(g)$ konstruieren, falls das Dreieck bekannt ist, in dem p eingefügt wurde. In *VoroGlide* werden nur diese Dreiecke neu konstruiert und die Animation arbeitet mit fünf verschiedenen Layern in der Anzeige. Diese Vorgehensweise ist gerade beim Verschieben eines Punktes für die Laufzeit vorteilhaft. Eine genaue Beschreibung ist unter [IKKM03] nachzuvollziehen. Insgesamt liegt die Laufzeit im worst-case in $O(n^2)$.

Für die Konstruktion des $\mathcal{FDT}(\mathcal{S})$ dient der Algorithmus, der die Basis für das Applet *VoroFarthestPoint* darstellt und die Delaunay-Triangulation inkrementell ermittelt. Ein eingefügter Punkt p nimmt nur Einfluss auf $\mathcal{FDT}(\mathcal{S})$, wenn p auf dem Rand konvexen Hülle liegt. Die bisher bestehenden Dreiecke sind mit p dann in Konflikt, wenn p in keinem Umkreis enthalten ist. Das Umbauen der Dreiecke wiederum erfolgt ebenfalls analog zu $\mathcal{DT}(\mathcal{S})$ über *edge flips*. Die genaue Vorgehensweise wird in [For03] beschrieben. Über Randomisierung erreicht der Algorithmus eine erwartete Laufzeit von $O(n \log n)$.

4.3 Konstruktion des größten leeren Kreises $\mathcal{LEC}(\mathcal{S})$

Die Lage des Mittelpunktes von $\mathcal{LEC}(\mathcal{S})$ entspricht entweder einem Voronoi-Knoten von $\mathcal{DT}(\mathcal{S})$ oder einem Schnittpunkt von $\mathcal{DT}(\mathcal{S})$ mit dem Rand der konvexen Hülle von S. Durch *LargestEmptyCircle* werden diese beiden Möglichkeiten der Lage überprüft und anschließend der resultierende Kreis ausgegeben.

Abgesehen von der Unterscheidung nach der Anzahl der Elemente und nach der kollinearen Lage aller Elemente in S, besteht *LargestEmptyCircle* im Wesentlichen aus drei Algorithmen. In *ConstructDT* wird die Delaunay-Triangulation $\mathcal{DT}(\mathcal{S})$ konstruiert und ein Kennzeichen zurückgegeben, falls alle Punkte kozirkular liegen. Tritt die letztgenannte Bedingung für eine Punktmenge auf, gibt es nur einen Voronoi-Knoten und dieser stellt den Mittelpunkt von $\mathcal{LEC}(\mathcal{S})$ dar. Anderenfalls durchläuft *IntersectConvHullVoronoi* alle Schnittpunkte der konvexen Hülle mit $\mathcal{VD}(\mathcal{S})$ bei gleichzeitiger Überprüfung, welcher leere Kreis am größten ist. Die Dreiecke, die mindestens eine Kante mit dem Rand der konvexen Hülle gemeinsam haben oder

deren Voronoi-Knoten außerhalb der konvexen Hülle liegen, werden dabei als verarbeitet markiert. Dadurch sind bei *InnerVoronoiNodes* nur noch die Dreiecke von Interesse, die noch nicht markiert sind und deren Voronoi-Knoten damit weitere Kandidaten für den Mittelpunkt des $\mathcal{LEC}(\mathcal{S})$ darstellen.

Die Laufzeit von *LargestEmptyCircle* liegt in $O(n \log n)$, da *ConstructDT* diese Laufzeit aufweist. *IntersectConvHullVoronoi* und *InnerVoronoiNodes* benötigen lineare Laufzeit. Für die beiden letztgenannten Algorithmen werden wir dies bei der jeweiligen Beschreibung begründen. Bei einer ungünstigen Reihenfolge von Einfüge- und Löschereignissen ergibt dies eine Gesamtlaufzeit über die Punktmenge S von $O(n^2)$.

Algorithmus 11 LargestEmptyCircle(S)

```
maxRadius := 0
n := Anzahl der Elemente in S
if n = 1 then
  return
else if n = 2 or alle Punkte in S kollinear then
  suche benachbarte Punkte p und r in S mit maximalen Abstand
  mLEC := center(p, r)
  maxRadius := distance(p, r)/2
else
  ConstructDT(S)
  if n > 3 and alle Punkte in S kozirkular then
    weise p, r, q drei Punkte aus S zu
    mLEC :=CircleCenter(p, r, q)
    maxRadius := distance(mLEC, p)
  else
    (* Schnittpunkte von VD(S) mit der konvexen Hülle ermitteln *)
    IntersectConvHullVoronoi(dt)
    (* Voronoi-Knoten von VD(S) innerhalb der konvexen Hülle verarbeiten *)
    InnerVoronoiNodes(dt)
  end
end
drawCircle(mLEC, maxRadius)
```

Der Vorteil, den $\mathcal{LEC}(\mathcal{S})$ mit einem Mittelpunkt innerhalb der konvexen Hülle zu definieren anstatt wie in [Kle97] innerhalb eines unabhängigen konvexen Polygons, zeigt sich an mehreren Aspekten. Während bei einem unabhängigen Polygon jeder Voronoi-Knoten überprüft werden muss, ob er innerhalb des Polygons liegt, kann nach dem Aufruf von *IntersectConvHull* davon ausgegangen werden, dass alle noch nicht verarbeiteten Voronoi-Knoten innerhalb der konvexen Hülle liegen. Dies erspart einen *PointInPolygon*-Test in $O(n \log m)$ Zeit bei einem Polygon mit m Ecken

gegenüber *InnerVoronoiNodes* in linearer Zeit. Es sind zwar für beide Aufgabenstellungen die Schnittpunkte von $\mathcal{VD}(\mathcal{S})$ mit einem konvexen Polygon zu bestimmen, allerdings ist bei einem unabhängigen Polygon erst festzustellen, in welcher Voronoi-Region ein definierender Punkt der Polygonkanten liegt. Durch einen Lauf entlang der Voronoi-Regionen werden dann alle Schnittpunkte gefunden. Betrachten wir das konvexe Polygon, das die konvexe Hülle darstellt, ersparen wir uns diesen Schritt, da die Eckpunkte des Polygons in ihren zugehörigen Voronoi-Regionen liegen. Insgesamt nutzen wir bei der Ermittlung der Schnittpunkte die Tatsache aus, dass $\mathcal{DT}(\mathcal{S})$ vollständig in dem konvexen Polygon enthalten ist und gemeinsame Kanten mit dem Polygon besitzt.

Der Anwender kann auswählen, welche Kreisoptimierungen oder -approximationen angezeigt werden sollen. Dementsprechend geben wir im Folgenden bei den Reaktionen auf die Ereignisse nur jeweils die Schritte an, die für den jeweiligen Kreis oder Ring von Interesse sind.

Wird ein Punkt q eingefügt, kann sich der $\mathcal{LEC}(\mathcal{S})$ ändern, wenn q außerhalb der konvexen Hülle liegt oder q genau in das Innere von $\mathcal{LEC}(\mathcal{S})$ gelegt wird. Beim Löschen von q wiederum kann sich der $\mathcal{LEC}(\mathcal{S})$ immer ändern, da wir nicht wissen, ob an dieser Stelle ein leerer Kreis gefunden werden kann, der größer als der bisherige $\mathcal{LEC}(\mathcal{S})$ ist. Das Verschiebeereignis ist an dieser Stelle ergänzt, da wir nicht einfach die Reaktion auf das Löschereignis von q und das Einfügeereignis von q' aneinander hängen, sondern $\mathcal{DT}(\mathcal{S})$ und $\mathcal{LEC}(\mathcal{S})$ direkt für die neue Menge S ermitteln. Dies spart gegebenenfalls einen zweiten Durchlauf von *LargestEmptyCircle*.

Reaktion auf **Ereignis 1** (Punkt q wird eingefügt)
  if PointInConvHull($convHull, q$) **then**
    if PointInCircle($mLEC, maxRadius, q$) **then**
      LargestEmptyCircle(S)
  else
    LargestEmptyCircle(S)
  end

Reaktion auf **Ereignis 2** (Punkt q wird gelöscht)
  LargestEmptyCircle(S)

Reaktion auf **Ereignis 3** (Punkt q wird nach q' verschoben)
  $S := S \setminus q \cup q'$
  LargestEmptyCircle(S)

4.3.1 Schnittpunkte von $\mathcal{VD}(\mathcal{S})$ mit der konvexen Hülle

Die Dreiecke auf dem Rand der konvexen Hülle werden in der gleichen Art und Weise durchlaufen wie in dem Algorithmus *ConstructConvHull*. Sollte ein Dreieck noch nicht verarbeitet sein, werden die drei Kanten überprüft, ob sie auf dem Rand der konvexen Hülle liegen. Nur für diese Dreiecke ist eine Verarbeitung anzustoßen und die Lage des zugehörigen Voronoi-Knoten wird überprüft.

Sollte der Voronoi-Knoten links von der Kante der konvexen Hülle liegen, ist noch keine weitere Aussage über die Lage bezüglich der konvexen Hülle möglich. Wie in Abbildung 4.15 skizziert, kann sich der Voronoi-Knoten in diesem Fall immer noch außerhalb der konvexen Hülle befinden. Dadurch kommt der Schnittpunkt der unbeschränkten Voronoi-Kante mit dem Rand der konvexen Hülle als Mittelpunkt für $\mathcal{LEC}(\mathcal{S})$ in Frage und wird dementsprechend überprüft. Liegt der Voronoi-Knoten auf dem Rand der konvexen Hülle, kann der Voronoi-Knoten den Mittelpunkt für $\mathcal{LEC}(\mathcal{S})$ darstellen und der Radius wird analysiert. In diesem Fall wird das Dreieck als verarbeitet markiert, da der Voronoi-Knoten verarbeitet ist und nicht noch einmal in *InnerVoronoiNodes* berechnet werden muss.

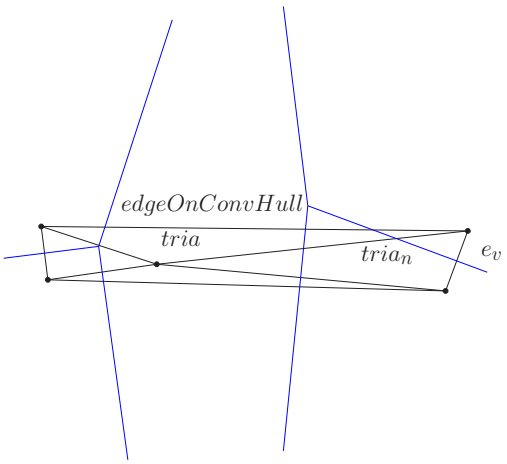


Abbildung 4.15: $\mathcal{VD}(\mathcal{S})$ (nicht vollständig) und $\mathcal{DT}(\mathcal{S})$

Liegt der Voronoi-Knoten außerhalb der konvexen Hülle, kommt er als Mittelpunkt für $\mathcal{LEC}(\mathcal{S})$ nicht in Frage. Die zugehörige unbeschränkte Voronoi-Kante zu der aktuellen Delaunay-Kante auf dem Rand der konvexen Hülle befindet sich ebenfalls außerhalb. Das Dreieck kann als verarbeitet markiert werden. In *CheckTria* erfolgt eine Analyse über die benachbarten Dreiecke, ob die zugehörigen Voronoi-Kanten Schnittpunkte mit dem Rand der konvexen Hülle bilden. Außerdem erfolgt eine Überprüfung, ob eine unbeschränkte Voronoi-Kante einen Schnittpunkt mit der aktuellen Kante auf der konvexen Hülle initiiert. In Abbildung 4.15 ist ein solcher

Fall exemplarisch dargestellt. e_v schneidet *edgeOnConvHull* und gehört zu $tria_n$.

Algorithmus 12 IntersectConvHullVoronoi(*dt*)

```
(* erstes Dreieck auf dem Rand der konvexen Hülle *)
tria := first(dt)
while noch nicht alle Dreiecke auf dem Rand der konvexen Hülle besucht do
  if not tria als verarbeitet markiert then
    for alle 3 Kanten von tria do
      e := aktuelle Kante von tria
      tria_n := Nachbardreieck der aktuellen Kante e
      if tria_n = ∅ then
        (* Kante auf dem Rand der konvexen Hülle *)
        edgeOnConvHull := e
        node := CircleCenter(tria(p_1), tria(p_2), tria(p_3))
        if LeftOn(edgeOnConvHull, node) or Collinear(edgeOnConvHull, node)
        then
          if length(edgeOnConvHull)/2) > maxRadius then
            mLEC :=center(edgeOnConvHull)
            maxRadius := length(edgeOnConvHull)/2
          end
          if Collinear(edgeOnConvHull, node) then
            markiere tria als verarbeitet
        else
          markiere tria als verarbeitet
          CheckTria(edgeOnConvHull, node, tria)
        end
      end
    end
    suche Kante e in tria mit edgeOnConvHull(p_2) als Startpunkt
    tria := Nachbardreieck der Kante e
  end
```

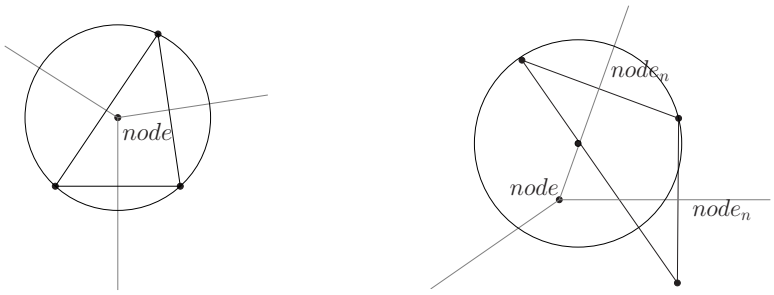
In *CheckTria* werden alle Schnittpunkte rekursiv zu einer Kante der konvexen Hülle *edgeOnConvHull* ermittelt.

Da beschränkte Voronoi-Kanten durch die Verbindung von entsprechenden Voronoi-Knoten entstehen, können die Kanten als relevant ermittelt werden, bei denen sich ein Knoten rechts und der andere links von *edgeOnConvHull* befindet. Dafür werden alle Nachbardreiecke des übergebenen Dreiecks untersucht, ob der zugehörige Voronoi-Knoten rechts von *edgeOnConvHull* liegt. Trifft dies zu und das Nachbardreieck wurde noch nicht verarbeitet, wird *CheckTria* wiederum aufgerufen und das Dreieck als verarbeitet markiert. Liegt der Voronoi-Knoten links von oder auf

edgeOnConvHull, können wir den entsprechenden Schnittpunkt der Voronoi-Kante mit *edgeOnConvHull* bestimmen. Für dieses Dreieck benötigen wir keine Untersuchung der Nachbardreiecke, da deren Voronoi-Knoten ebenfalls links von der zu untersuchenden Kante *edgeOnConvHull* liegen.

Um Schnittpunkte einer unbeschränkten Voronoi-Kante mit *edgeOnConvHull* zu bestimmen, wird die Eigenschaft ausgenutzt, dass ein Punkt dieser unbeschränkten Kante im Mittelpunkt der aktuellen Delaunay-Kante e liegt. Anstatt einen Schnittpunkt zwischen einem Strahl und einem Segment zu berechnen, wird dieser Mittelpunkt als „künstlicher“ Voronoi-Knoten $node_n$ angenommen. Der Strahl wird auf ein Liniensegment beschränkt, dass von dem real existierenden Voronoi-Knoten *node* zu $node_n$ führt. Dadurch ist gewährleistet, dass der Schnittpunkt von *edgeOnConvHull* mit einer unbeschränkte Voronoi-Kante gefunden wird, die nicht außerhalb der konvexen Hülle verläuft.

Diese Vorgehensweise lässt sich an dem Sonderfall veranschaulichen (siehe Abbildung 4.16), bei dem S aus drei Punkten und $\mathcal{DT}(\mathcal{S})$ damit aus einem Dreieck besteht, das gleichzeitig der konvexen Hülle entspricht. Enthält $\mathcal{DT}(\mathcal{S})$ nur ein Dreieck, kann der Voronoi-Knoten *node* wie im allgemeinen Fall sowohl innerhalb als auch außerhalb der konvexen Hülle liegen. Liegt *node* innerhalb, entspricht er dem Mittelpunkt von $\mathcal{LEC}(\mathcal{S})$ und wird durch *InnerVoronoiNodes* verarbeitet. Liegt er allerdings außerhalb, existieren vier Schnittpunkte von zwei unbeschränkten Voronoi-Kanten mit dem Rand der konvexen Hülle. Zwei Schnittpunkte werden darüber gefunden, dass der Voronoi-Knoten auf der linken Seite von den Kanten der konvexen Hülle liegt und den Mittelpunkten dieser Kanten entsprechen. Die anderen beiden Schnittpunkte liegen auf der Kante der konvexen Hülle, bei dem sich der Voronoi-Knoten auf der rechten Seite befindet. Dadurch erfolgt ein Aufruf von *CheckTria* und die Festlegung von den „künstlichen“ Voronoi-Knoten $node_n$.



(a) *node* innerhalb (b) *node* außerhalb

Abbildung 4.16: $\mathcal{LEC}(\mathcal{S})$ eines Dreiecks

Sind alle Kanten der konvexen Hülle abgelaufen, sind alle Dreiecke, deren Voronoi-Knoten außerhalb liegen, verarbeitet.

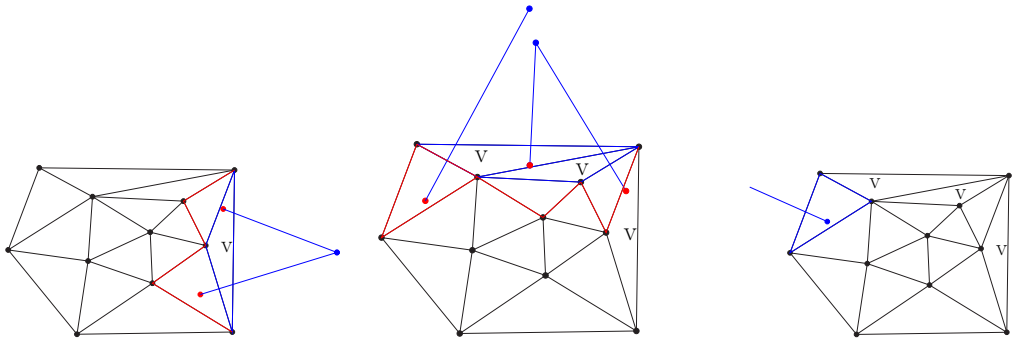
Die Anzahl der Voronoi-Knoten und -Kanten liegt wie bereits vorgestellt in $O(n)$. Da jede Voronoi-Kante höchstens zwei Kanten der konvexen Hülle und jede Kante der konvexen Hülle wiederum mindestens eine Voronoi-Kante schneidet, können nur $O(n)$ Schnittpunkte auftreten. Die Dreiecke, die während der Ermittlung der Schnittpunkte analysiert werden, können pro zu untersuchende Voronoi-Kante höchstens zweimal besucht werden, also insgesamt 6-mal. Die Laufzeit von *CheckTria* ist damit linear.

Algorithmus 13 CheckTria($edgeOnConvHull, node, tria$)

```
for alle 3 Kanten von tria do
  e := aktuelle Kante von tria
  tria_n := Nachbardreieck der aktuellen Kante e
  if tria_n ≠ ∅ then
    node_n := CircleCenter(tria_n(p_1), tria_n(p_2), tria_n(p_3))
  else if e ≠ edgeOnConvHull then
    (* e liegt auf der konvexen Hülle und entspricht nicht edgeOnConvHull *)
    node_n := center(e)
  end
  if e ≠ edgeOnConvHull and (LeftOn(edgeOnConvHull, node_n) or
  Collinear(edgeOnConvHull, node_n)) then
    (* die Voronoi-Kante zwischen node and node_n schneidet den Rand der kon-
    vexen Hülle *)
    voronoiEdge := (node, node_n)
    s :=IntersectSegments(edgeOnConvHull, voronoiEdge)
    if dist(s, voronoiEdge(p_1)) > maxRadius then
      mLEC := s
      maxRadius := dist(s, voronoiEdge(p_1))
    end
  else
    (* node_n liegt rechts von edgeOnConvHull *)
    if not tria_n ist als verarbeitet markiert then
      markiere tria_n als verarbeitet
      CheckTria( edgeOnConvHull, node_n, tria_n )
    end
  end
end
```

In Abbildung 4.17 wird exemplarisch eine Abarbeitung einiger Dreiecke auf dem Rand der konvexen Hülle dargestellt. In der ersten Teilabbildung wird eine Situation gezeigt, in der *CheckTria* einmal aufgerufen wird. Beide Schnittpunkte werden durch die direkten Nachbardreiecke des Dreiecks auf dem Rand der konvexen Hülle gefunden. Der Ablauf für die zweite Abbildung impliziert den zweimaligen Aufruf

und die dritte keinen Aufruf von *CheckTria*. Nach dem kompletten Durchlauf von *IntersectConvHullVoronoi* ist unser Beispiel, wie in Abbildung 4.18 dargestellt, verarbeitet.



(a) (b) (c)

Abbildung 4.17: Schrittweise Verarbeitung der Schnittpunkte von $\mathcal{DT}(\mathcal{S})$ mit der konvexen Hülle

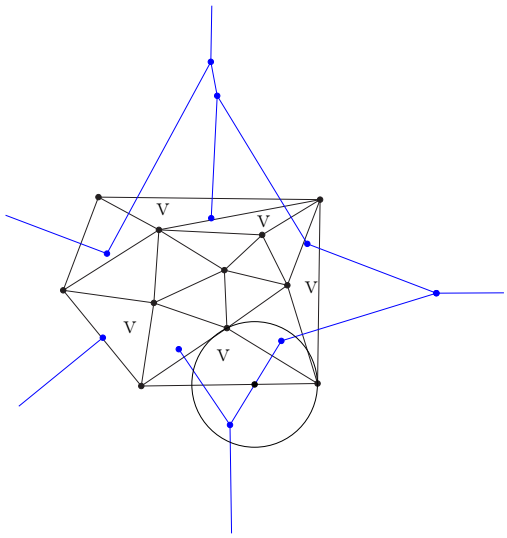


Abbildung 4.18: Schnittpunkte von $\mathcal{DT}(\mathcal{S})$ mit der konvexen Hülle

4.3.2 Voronoi-Knoten von $\mathcal{VD}(\mathcal{S})$ innerhalb der konvexen Hülle

In *InnerVoronoiNodes* werden alle Dreiecke von $\mathcal{DT}(\mathcal{S})$ durchlaufen und nur die, die durch *IntersectConvHull* nicht verarbeitet wurden, müssen für den $\mathcal{LEC}(\mathcal{S})$ analysiert werden. Für das aktuelle noch nicht verarbeitete Dreieck wird der zugehörige

Voronoi-Knoten *node* berechnet. Ist der Abstand *distance* zu einem Punkt des Dreiecks größer als der bisher ermittelte größte Radius, wird *node* als Mittelpunkt und *distance* als Radius für $\mathcal{LEC}(\mathcal{S})$ gewählt. Die Laufzeit des Algorithmus *InnerVoronoiNodes* liegt dementsprechend in $O(n)$.

Algorithmus 14 InnerVoronoiNodes(*dt*)

```
tria := first(dt)
(* Überprüfung aller Dreiecke von dt *)
while noch nicht alle Dreiecke von dt besucht do
  if not tria als verarbeitet markiert then
    (* Überprüfung der Voronoi-Knoten, die noch nicht verarbeitet wurden *)
    node := CircleCenter(tria(p1), tria(p2), tria(p3))
    if dist(node, tria(p1)) > maxRadius then
      mLEC := node
      maxRadius := dist(node, tria(p1))
    end
  end
  tria := next(tria)
end
```

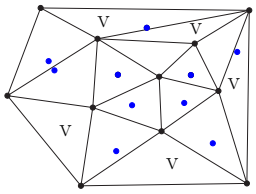


Abbildung 4.19: Innere Voronoi-Knoten mit $\mathcal{DT}(\mathcal{S})$

In unserem Beispiel wird durch den Aufruf von *InnerVoronoiNodes* kein größerer leerer Kreis ermittelt als der, der bereits in *IntersectConvHullVoronoi* gefunden wurde. Die folgende Abbildung zeigt also den resultierenden $\mathcal{LEC}(\mathcal{S})$.

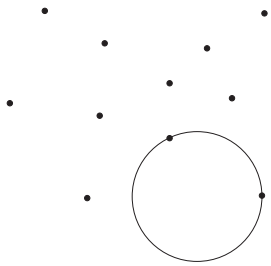


Abbildung 4.20: $\mathcal{LEC}(\mathcal{S})$ mit der Punktmenge S

4.4 Konstruktion des kleinsten umfassenden Kreises $\mathcal{SEC}(\mathcal{S})$

Für die Lage des Mittelpunktes von $\mathcal{SEC}(\mathcal{S})$ kommt entweder der Mittelpunkt eines diametralen Kreises oder ein Voronoi-Knoten des furthest-point Voronoi-Diagramm $\mathcal{FVD}(\mathcal{S})$ in Frage. Während der Ermittlung des Durchmessers ist schon geprüft worden, ob mehrere Punktepaare mit der Distanz des Durchmessers existieren und ob diese innerhalb des als erstes gefundenen diametralen Kreis liegen. Trifft dies nicht zu, kann $\mathcal{SEC}(\mathcal{S})$ keinem diametralen Kreis entsprechen und der Algorithmus *PointsInDiam* wird nicht aufgerufen. Sind alle Punkte aus S kozirkular oder kollinear angeordnet, entspricht $\mathcal{SEC}(\mathcal{S})$ einem diametralen Kreis und wird in *PointsInDiam* verarbeitet. Gibt *PointsInDiam* einen negativen Wert zurück, werden alle Voronoi-Knoten bezüglich des minimalen Radius eines umschließenden Kreises analysiert. Als abschließende Aktion erfolgt das Ausgeben von $\mathcal{SEC}(\mathcal{S})$.

Die Laufzeit von *PointsInDiam* liegt in $O(n)$ (siehe Kapitel 4.4.1) und die Anzahl der Voronoi-Knoten liegt ebenfalls in $O(n)$. Da *ConstructFDT* eine Zeitkomplexität von $O(n \log n)$ aufweist, resultiert für *SmallestEnclosingCircle* eine Laufzeit von $O(n \log n)$. Analog zur Konstruktion des $\mathcal{LEC}(\mathcal{S})$ beträgt die Gesamtlaufzeit über die vollständige Punktmenge S bei einer ungünstigen Einfüge- oder Löschreihenfolge $O(n^2)$.

Algorithmus 15 SmallestEnclosingCircle($convHull, d$)

```
if Anzahl der Elemente in convHull = 1 then
  minRadius := 0 return
end
minRadius := diameter/2
if inDiameterCircle and PointsInDiam(convHull, d) then
  mSEC := mDiam
else
  minRadius := maxInt
  ConstructFDT(convHull)
  (* durchlaufen der Delaunay-Triangulation fdt *)
  tria := head(fdt)
  while noch nicht alle Dreiecke von fdt besucht do
    m :=CircleCenter(tria(p1), tria(p2), tria(p3))
    if dist(tria(p1), m) < minRadius then
      mSEC := m
      minRadius := dist(tria(p1), m)
    end
    tria := next(tria)
  end
end
drawCircle(mSEC, minRadius)
```

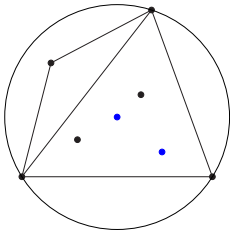


Abbildung 4.21: $\mathcal{FDT}(\mathcal{S})$ mit zugehörigen Voronoi-Knoten und umfassenden Kreis

Ein Beispiel für einen kleinsten umschließenden Kreis um einen Voronoi-Knoten wird in Abbildung 4.21 gezeigt.

Beim Einfügen eines Punktes q ist die Neukonstruktion von $\mathcal{SEC}(\mathcal{S})$ nur notwendig, wenn q nicht in dem ursprünglichen $\mathcal{SEC}(\mathcal{S})$ enthalten ist. Die Bedingung *PointInCircle* trifft auch zu, wenn q auf dem Rand von $\mathcal{SEC}(\mathcal{S})$ liegt. Das Löschereignis erfordert den Aufruf von *SmallestEnclosingCircle*, falls q auf dem Rand von $\mathcal{SEC}(\mathcal{S})$ liegt. Die Reaktion auf das Verschiebeereignis kombiniert die beiden **if**-Bedingungen aus Einfüge- und Löschereignis, um der doppelten Ausführung von *SmallestEnclosingCircle* vorzubeugen.

Reaktion auf **Ereignis 1** (Punkt q wird eingefügt)
  if not PointInCircle($mSEC, minRadius, q$) **then**
      SmallestEnclosingCircle($convHull, d$)
  end

Reaktion auf **Ereignis 2** (Punkt q wird gelöscht)
  if $dist(q, msec) = minRadius$ **then**
    SmallestEnclosingCircle($convHull, d$)
  end

Reaktion auf **Ereignis 3** (Punkt q wird nach q' verschoben)
  if not PointInCircle($mSEC, minRadius, q'$)
  or $dist(q, msec) = minRadius$ **then**
    SmallestEnclosingCircle($convHull, d$)
  end

4.4.1 Diametrale Kreise einer Punktmenge

Für die Konstruktion eines diametralen Kreises einer Punktmenge ist der Durchmesser einer Punktmenge ausschlaggebend. In der Datenstruktur d ist der Durchmesser und ein Punktepaar enthalten, das im Abstand des Durchmessers zueinander liegt. Es reicht aus, alle Punkte auf dem Rand der konvexen Hülle zu analysieren, ob sie in dem durch d definierten diametralen Kreis enthalten sind. Sollte ein Punkt außerhalb liegen, ist der *PointsInDiam*-Test negativ. Da der kompletten konvexen Hülle

entlang gewandert wird, liegt die Laufzeit von *PointsInDiam* in $O(n)$.

Algorithmus 16 PointsInDiam($polygon, d$)

```
couple :=getCouple(d)
m :=center(couple(p1), couple(p2))
r := head(polygon)
(* Lauf durch die konvexe Hülle *)
repeat
  if dist(m, r)) > minRadius then
    (* Ecke liegt außerhalb des Kreises *)
    return false
  r := next(r)
until r ≠ head(polygon)
return true
```

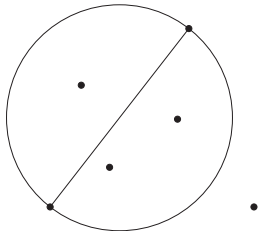


Abbildung 4.22: Diametraler Kreis mit der Punktmenge S

In unserem gewählten Beispiel enthält der diametrale Kreis nicht alle Punkte aus S und kann so nicht dem $\mathcal{SEC}(\mathcal{S})$ entsprechen. Der $\mathcal{SEC}(\mathcal{S})$ wird also durch den Vergleich der Radien um die Voronoi-Knoten konstruiert und stellt sich folgendermaßen dar.

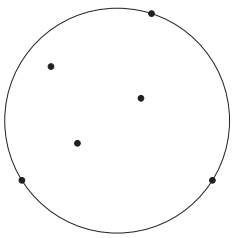


Abbildung 4.23: $\mathcal{SEC}(\mathcal{S})$ mit der Punktmenge S

4.5 Konstruktion des Rings mit minimaler Breite $\mathcal{MWA}(\mathcal{S})$

Für die Konstruktion von $\mathcal{MWA}(\mathcal{S})$ ist zu entscheiden, ob $\mathcal{MWA}(\mathcal{S})$ als Ring minimaler Breite oder als Streifen minimaler Breite vorliegt. Die Breite ist in der Datenstruktur w bereits abgelegt und steht dem Algorithmus *MinimumWidthAnnulus* zur Verfügung. Sollte sie 0 betragen, liegt entweder eine ein-elementige, eine zwei-elementige oder eine Punktmenge vor, deren Elemente alle in kollinearer Lage angeordnet sind. In diesen Fällen ist die Überprüfung auf einen Ring minimaler Breite nicht notwendig und es wird eine entsprechende Gerade ausgegeben. Trifft dies nicht zu, wird die Delaunay-Triangulation $\mathcal{DT}(\mathcal{S})$ bestimmt. Falls sich alle Punkte aus S in kozirkularer Lage befinden, liegt der Sonderfall vor, dass ein Ring minimaler Breite aus einem Kreis besteht. Der Ring hat also die Breite 0. Dann kann der Mittelpunkt von $\mathcal{MWA}(\mathcal{S})$ direkt durch den Mittelpunkt eines Kreises um ein Dreieck aus beliebigen, paarweise verschiedenen Elementen aus S bestimmt werden.

Liegen nicht alle Punkte kozirkular oder kollinear, ist die furthest-point Delaunay-Triangulation $\mathcal{FDT}(\mathcal{S})$ zu konstruieren. Die beiden Triangulationen sind die Datenbasis für *SweepIntersectVoronoi*.

Der Sweep-Algorithmus bestimmt die Schnittpunkte von den Voronoi-Diagrammen $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ und berechnet die minimale Distanz zwischen einem leeren und einem umschließenden Kreis. Ist die minimale Distanz kleiner als die Breite der Punktmenge, stellt $\mathcal{MWA}(\mathcal{S})$ einen Ring dar. Die Ausgabe wird durch das Zeichnen der beiden involvierten Kreise oder der beiden Geraden realisiert.

Algorithmus 17 MinimumWidthAnnulus($S, convHull, width$)

```
(* Konstruktion von MWA nur notwendig, wenn die Breite der Punktmenge
größer als 0 ist *)
if width > 0 then
  (* die Punktmenge S ist nicht kollinear angeordnet *)
  ConstructDT(S)
  if alle Punkte in S sind kozirkular angeordnet then
    weise p, r, q drei Punkte aus S zu
    mMWA :=CircleCenter(p, r, q)
    innerRadius, outerRadius := distance(mMWA, p)
  else
    ConstructFDT(convHull)
    SweepIntersectVoronoi(dt, fdt)
  end
end
```

Die Laufzeit von *MinimumWidthAnnulus* liegt in $O(n^2 \log n)$. Dies resultiert aus der Anzahl der Schnittpunkte zwischen $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$, die die Größenordnung

$O(n^2)$ annehmen können und durch *SweepIntersectVoronoi* ermittelt werden. Bei der Beschreibung des Sweep wird die Laufzeitabschätzung begründet. In [AAHPS99] wird gezeigt, dass im Durchschnitt die Anzahl der Schnittpunkte in $O(n)$ liegt. Setzen wir die lineare Anzahl der Schnittpunkte voraus, benötigt der Algorithmus $O(n \log n)$ Laufzeit.

Die Ausgabe von $\mathcal{MWA}(\mathcal{S})$ führen wir in einer getrennten Funktion durch, damit diese explizit bei den Reaktionen auf die Ereignisse aufgerufen werden kann. Liegt $\mathcal{MWA}(\mathcal{S})$ als Streifen vor, wird die Funktion *parallel* für die Ausgabe verwendet. $parallel(p, r, q, width)$ legt eine Gerade fest, die parallel zu dem Liniensegment $\overline{pr}$ in Richtung q im Abstand $width$ liegt.

DrawMWA:

```
if wMWA < width then
    (* MWA(S) ist ein Ring *)
    drawCircle(mMWA, innerRadius)
    if innerRadius ≠ outerRadius then
        drawCircle(mMWA, outerRadius)
    end
else
    (* MWA(S) ist ein Streifen *)
    (p, r, q) :=getTripel(w)
    if r ≠ 0 then
        (* S besteht aus mindestens zwei Punkten *)
        drawLine(p, r)
    end
    if width > 0 then
        drawLine(parallel(p, r, q, width))
end
```

Bei den Reaktionen auf die Ereignisse werden wir die Ermittlung der Breite der Punktmenge nicht mit einbeziehen, da wir diese schon angegeben haben. Die Breite der Punktmenge kann sich unabhängig von den folgenden Kriterien für den Ring ändern und die Ausgabe von $\mathcal{MWA}(\mathcal{S})$ ist dementsprechend zu veranlassen.

Beim Einfügen eines Punktes wird ein Ring minimaler Breite nur modifiziert, wenn dieser außerhalb des äußeren oder innerhalb des inneren Kreises liegt. Das Löschen eines Punktes q löst eine Neuermittlung des Rings nur aus, wenn q auf einem der Kreisränder liegt. Die Bedingungen des Einfüge- und Löschereignisses werden beim Verschieben von q gleichzeitig geprüft, um eine zweifache Ausführung von *MinimumWidthAnnulus* zu vermeiden.

Reaktion auf **Ereignis 1** (Punkt q wird eingefügt)
  if not PointInCircle($mMWA, outerRadius, q$)
  or PointInCircle($mMWA, innerRadius, q$) **then**
    MinimumWidthAnnulus($S, convHull, w$)
    DrawMWA($wMWA, mMWA, innerRadius, outerRadius, w$)
  end

Reaktion auf **Ereignis 2** (Punkt q wird gelöscht)
  if $dist(q, mMWA) = outerRadius$
  or $dist(q, mMWA) = innerRadius$ **then**
    MinimumWidthAnnulus($S, convHull, w$)
    DrawMWA($wMWA, mMWA, innerRadius, outerRadius, w$)
  end

Reaktion auf **Ereignis 3** (Punkt q wird nach q' verschoben)
  if $dist(q, mMWA) = outerRadius$
  or $dist(q, mMWA) = innerRadius$
  or not PointInCircle($mMWA, outerRadius, q'$)
  or PointInCircle($mMWA, innerRadius, q'$) **then**
    MinimumWidthAnnulus($S, convHull, w$)
    DrawMWA($wMWA, mMWA, innerRadius, outerRadius, w$)
  end

4.5.1 Schnittpunkte von $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$

Die Schnittstellenbestimmung von $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ setzen wir mit Hilfe eines Plane-Sweep-Algorithmus um. Als Scanner-Geometrie eignet sich eine vertikale Gerade, die sogenannte *Sweep line*, die über die Problemanordnung wandert. In unserem Fall entspricht die Problemanordnung den Liniensegmenten und den Strahlen von $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ und stellt damit einen Spezialfall für einen Plane-Sweep-Algorithmus über unabhängige Liniensegmente dar. Letzterer wird in einer Reihe von Veröffentlichungen beschrieben und als Beispiel sei hier nur [Kle97] genannt. Der Algorithmus, den wir im Folgenden beschreiben werden, ist von dem allgemeinen Fall entlehnt und auf die speziellen Bedürfnisse der vorliegenden Problemanordnung angepasst.

SweepIntersectVoronoi bekommt als Parameter die Delaunay-Triangulationen übergeben, aus denen die zugehörigen Voronoi-Diagramme ermittelt werden. Bevor der eigentlich Sweep, also das Scannen der Problemanordnung, beginnt, sind passende Datenstrukturen aufzubauen. Zu diesem Zweck wird der bereits vorgestellte Algorithmus *ConstructVD* in der Art modifiziert, dass die Datenstrukturen vd und $vd_{-\infty}$ die Voronoi-Kanten aus $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ sortiert verwalten.

In vd werden alle Liniensegmente und Strahlen in Richtung $+\infty$ (bezogen auf die X-Koordinaten) abgelegt. In $vd_{-\infty}$ werden alle Strahlen in Richtung $-\infty$ ab-

gelegt. Diese Datenstrukturen sind doppelt verkettete Listen, deren Elemente nach der Ausführung von *ConstructVD* alle Voronoi-Kanten von $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ mit den initiierenden zwei Punkten aus S enthalten. Für die Liniensegmente werden ihre Endpunkte und für die Strahlen ein Endpunkt und die Steigung abgelegt. Zusätzlich ist jeweils ein boolescher Wert zur Kennzeichnung gespeichert, zu welchem Voronoi-Diagramm die aktuelle Kante gehört.

Das Sortierkriterium in vd ist für die Liniensegmente und die Strahlen, die in Richtung $+\infty$ verlaufen, der linke Endpunkt. Da es sich hierbei um Voronoi-Knoten handelt, benötigen wir ein zweites Sortierkriterium, wenn der X-Wert des linken Endpunktes von mehreren Kanten identisch ist. Das zweite Kriterium stellt die Steigung dar. Für Liniensegmente ist die Steigung anhand ihrer Endpunkte in konstanter Zeit zu ermitteln. Die folgende Funktion ist für alle nicht vertikalen Liniensegmente anwendbar.

$$gradient(\overline{pq}) = \frac{q_2 - p_2}{q_1 - p_1}$$

Die Strahlen, die in Richtung $-\infty$ orientiert sind, werden nach Steigung sortiert in $vd_{-\infty}$ eingefügt.

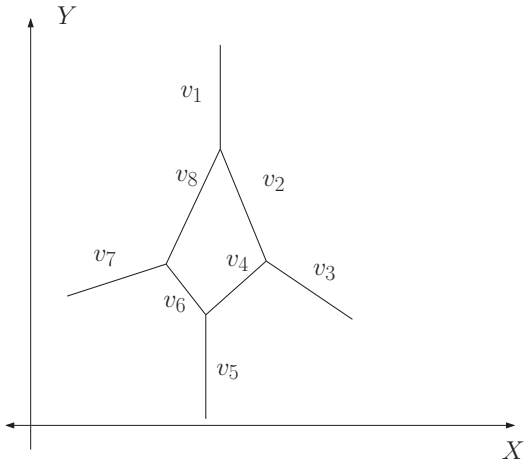


Abbildung 4.24: $\mathcal{VD}(\mathcal{S})$ mit vertikalen Voronoi-Kanten

Ein Sonderfall liegt vor, wenn eine Voronoi-Kante vertikal im Koordinatensystem verläuft. Diese werden sortiert nach dem kleineren Endpunkt in vd eingefügt. Um das vorgestellte Sortierkriterium über die Steigung anwenden zu können, wird die Steigung auf $+maxInt$ oder $-maxInt$ gesetzt. Das Vorzeichen ist abhängig davon, ob die Y-Koordinaten der Kante von dem analysierten Voronoi-Knoten aus größer oder kleiner werden. In Abbildung 4.24 werden zwei vertikale Voronoi-Kanten v_1 und v_5 dargestellt. Die Steigung von v_1 erhält den Wert $+maxInt$ und die Steigung von v_5 den Wert $-maxInt$. Diese Maßnahme gewährleistet, dass die Definition der Ordnung $<$, $>$ und $=$ für die Voronoi-Kanten über die Sortierkriterien eindeutig ist. Die Kanten aus dem Beispiel in Abbildung 4.24 sind nach der Ausführung von

ConstructVD folgendermaßen einsortiert.
$vd : v_6 < v_8 < v_5 < v_4 < v_2 < v_1 < v_3$
$vd_{-\infty}$: v_7

Allgemein ist der Aufbau einer Sortierung von Elementen mit anderen Datenstrukturen, wie beispielsweise einem balancierten Baum, performanter als mit einer doppelt verketteten Liste. Da jedoch eine sortierte Liste von Delaunay-Dreiecken an *ConstructVD* übergeben wird, sorgt diese Vorsortierung dafür, dass der Aufbau der Listen ebenfalls mit $O(n \log n)$ Zeitbedarf durchgeführt werden kann.

Algorithmus 18 ConstructVD($dt, angle, vd, vd_{-infty}, isVD$)

```
p_-∞ := null, p_seg := null
tria := first(dt)
while noch nicht alle Dreiecke von dt besucht do
  node := CircleCenter(tria(p_1), tria(p_2), tria(p_3))
  for alle drei Kanten des aktuellen Dreiecks do
    tria_n := Nachbardreieck der aktuellen Kante e
    if tria_n ≠ ∅ and tria_n noch nicht besucht then
      (* die aktuelle Kante e liegt innerhalb der konvexen Hülle *)
      node_n := CircleCenter(tria_n(p_1), tria_n(p_2), tria_n(p_3))
      if node ≠ node_n then
        (* tria und tria_n haben nicht den gleichen Umkreis *)
        if node < node_n then
          edge := (node, node_n)
        else
          edge := (node_n, node)
        InsertSort(vd, p_seg, edge, e, isVD)
    else if tria_n = ∅ then
      (* die aktuelle Kante e ist eine Kante der konvexen Hülle *)
      edge := (node, rotate(e, angle))
      if edge eine Kante in Richtung −∞ then
        InsertSort(vd_-∞, p_-∞, edge, e, isVD)
      else
        InsertSort(vd, p_seg, edge, e, isVD)
    end
  end
  tria := next(tria)
end
```

Die Vorsortierung der Voronoi-Kanten wird beim Einfügen eines neuen Elements durch *InsertSort* ausgenutzt. vd bzw. $vd_{-\infty}$ wird nicht für jedes Element vom Kopf aus durchlaufen, sondern beginnend von $p_{-\infty}$ bzw. p_{seg}, deren Wert in dem Para-

meter p abgelegt ist. Insbesondere für die Voronoi-Kanten, die einen gemeinsamen Voronoi-Knoten besitzen und direkt hintereinander verarbeitet werden, ist diese Vorgehensweise vorteilhaft.

Algorithmus 19 InsertSort($vd, p, voronoiEdge, delaunayEdge, isVD$)

```
if vd = null then
  neues Element (voronoiEdge, delaunayEdge, isVD) als Kopf von vd erstellen
  p := head(vd)
else
  (* suche nächst größeres Element *)
  while not p = null and p.voronoiEdge < voronoiEdge do
    p := next(p)
  if p = null then
    neues (voronoiEdge, delaunayEdge, isVD) Element an vd anhängen
    p := tail(vd)
  else
    neues Element (voronoiEdge, delaunayEdge, isVD) in vd hinter before(p)
    p := before(p)
  end
  (* suche nächst kleineres Element *)
  while not p = null and p.voronoiEdge > voronoiEdge do
    p := before(p)
  if p = null then
    neues Element (voronoiEdge, delaunayEdge, isVD) als Kopf von vd erstellen
    p := head(vd)
  else
    neues Element (voronoiEdge, delaunayEdge, isVD) in vd hinter p
    p := next(p)
end
```

In vd und $vd_{-\infty}$ sind nach der Ausführung von *ConstructVD* alle Liniensegmente und Strahlen enthalten, die für den Sweep benötigt werden. Die Aufteilung der Voronoi-Kanten auf zwei Datenstrukturen begründet sich in der Festlegung des zu scannenden Bereiches.

Um alle Schnittpunkte zu finden, ist es notwendig, einen geeigneten X-Wert als Startpunkt für den Sweep festzulegen. Bei dem allgemeinen Sweep-Algorithmus für unabhängige Liniensegmente ist der Startpunkt die kleinste X-Koordinate, die bei einem Endpunkt auftritt. Diese Festlegung können wir bei diesem Sweep nicht übernehmen, da ein Schnittpunkt zwischen zwei Strahlen in Richtung $-\infty$ einen kleineren X-Wert als ein Endpunkt der Voronoi-Kanten besitzen kann (siehe Abbildung 4.25 auf der nächsten Seite).

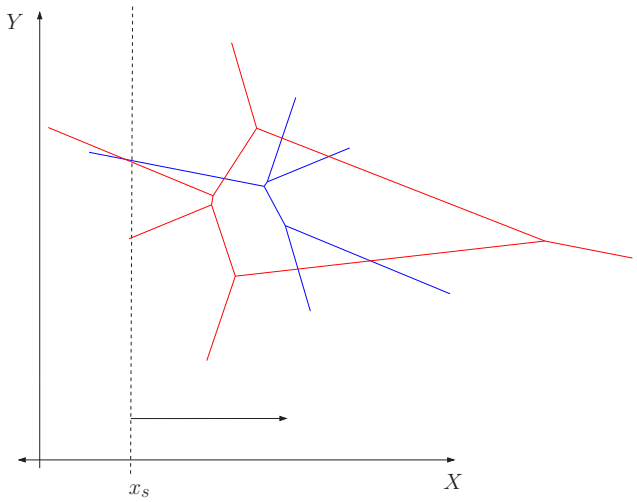


Abbildung 4.25: Festlegung des zu scannenden Bereiches

Der Startpunkt wird in *FindStartX* festgelegt, indem alle Strahlen, die in Richtung $-\infty$ verlaufen, nach ihrer Steigung sortiert und auf Schnittpunkt untereinander geprüft werden. Dabei müssen nicht alle Strahlen paarweise verglichen werden, sondern nur die, die zu jeweils dem anderen Voronoi-Diagramm gehören und in der Sortierung benachbart liegen. Bei dieser Vorgehensweise wird ausgenutzt, dass die Voronoi-Regionen konvex sind. Wird kein Schnittpunkt gefunden, ist die kleinste X-Koordinate eines Liniensegments bzw. eines Strahls Richtung $+\infty$ der Startpunkt.

Die Überprüfung auf Schnittpunkt erfordert nur eine kleine Erweiterung der Funktion *IntersectSegments* (s.u.) zu *IntersectRays*. Zuerst wird anhand des gegebenen Punktes auf einem Strahl und der Steigung ein zweiter Punkt auf einem Strahl berechnet. Mit den Punkten kann dann *IntersectSegments* aufgerufen werden und wenn a und b größer oder gleich 0 sind, existiert ein Schnittpunkt zwischen den Strahlen.

IntersectSegments($\overline{pq}, \overline{uv}$):
$$
\begin{aligned}
D &:= (p_1 - q_1)(v_2 - u_2) - (u_1 - v_1)(p_2 - q_2) \\
a &:= (p_1(v_2 - u_2) + u_1(p_2 - v_2) + v_1(u_2 - p_2))/D \\
b &:= (p_1(u_2 - q_2) + q_1(p_2 - u_2) + u_1(q_2 - p_2))/D \\
s_1 &:= p_1 + a(q_1 - p_1) \\
s_2 &:= p_2 + a(q_2 - p_2)
\end{aligned}
$$

Ist der Startpunkt x_s gefunden, werden für die nach $-\infty$ orientierten Strahlen die Y-Koordinaten zum Zeitpunkt x_s berechnet. Durch die Festlegung von x_s ist sichergestellt, dass diese Berechnung für jeden der ausgewählten Strahlen möglich ist.

Es werden nur Strahlen betrachtet, deren Endpunkt eine X-Koordinate größer als x_s besitzt. Denn ein Strahl, der in x_s beginnt und Richtung $-\infty$ orientiert ist, kann nur einen echten Schnittpunkt in dem Bereich von $-\infty$ bis x_s initiieren. Dieser wäre durch die Untersuchung der Schnittpunkte gefunden worden. Es kann noch ein „unechter“ Schnittpunkt genau in x_s auftreten. Dann handelt es sich an dieser Stelle um einen Voronoi-Knoten und der Schnittpunkt wird durch eine andere Voronoi-Kante des entsprechenden Voronoi-Diagramms gefunden.

Die berechnete Y-Koordinate wird zusammen mit x_s als Endpunkt für die ursprünglich unbeschränkte Voronoi-Kante angenommen. Darüber modifizieren wir künstlich die Strahlen zu Liniensegmente und fügen sie in vd ein. Sie können wie die ursprünglichen Liniensegmente in dem Sweep verarbeitet werden. vd bildet dabei die Datengrundlage.

Algorithmus 20 FindStartX($vd, vd_{-\infty}$)

```
(* startX wird zuerst der kleinste X-Wert eines Liniensegments zugewiesen *)
p := head(vd)
startX := p.voronoiEdge(point1(x))
(* Lauf durch die nach −∞ orientierten Strahlen *)
p := head(vd_{-∞})
while not next(p) = null do
  if not p.isVD = next(p).isVD then
    (* Voronoi-Kanten aus verschiedenen Voronoi-Diagrammen *)
    s := IntersectRays(p.voronoiEdge, next(p).voronoiEdge)
    (* Analyse, ob Schnittpunkt existiert *)
    if s and s.x < startX then
      startX = s.x
  end
end
p_seg := head(vd)
p := head(vd_{-∞})
while not p = null do
  (* nur die Strahlen betrachten, die nach startX noch existieren *)
  if p.voronoiEdge(point(x)) > startX then
    (* Verändern der Strahlen in Liniensegmente und einfügen in vd *)
    startY := calcY(p.voronoiEdge, startX)
    voronoiEdge := ((startX, startY), p.voronoiEdge(point1))
    InsertSort(vd, p_seg, voronoiEdge, p.delaunayEdge, p.isVD)
  end
end
```

Nach den Vorarbeiten kann an dieser Stelle auf den eigentlichen Sweep eingegangen werden. Die Sweep line definiert auf den Liniensegmenten oder Strahlen, die sie schneidet, eine Ordnung anhand der Y-Koordinaten. Betrachten wir Abbildung

4.26 gilt zum Zeitpunkt x_1 die Ordnung $f_1 < v_1$ und zum Zeitpunkt x_2 die Ordnung $f_2 < f_3 = v_3 < v_2$.

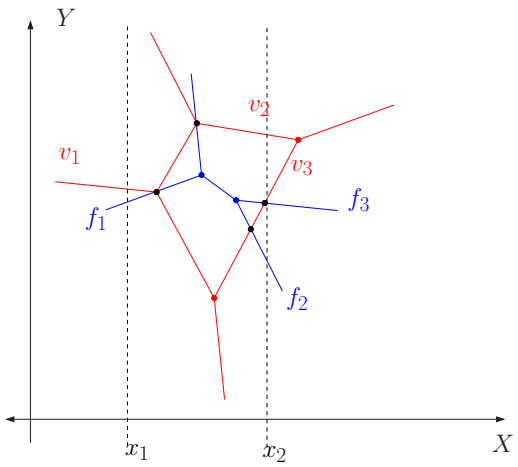


Abbildung 4.26: Die Sweep line zu zwei Zeitpunkten

Während die Ordnung auf f_1 und v_1 bzw. f_3 und v_3 für uns von Interesse für einen Schnittpunkt ist, ist $v_3 < v_2$ nur zu dem Zweck relevant, um festzustellen, welche Kanten zu dem jeweils anderen Voronoi-Diagramm benachbart sind. Zu einem Zeitpunkt vor x_2 mit hinreichend kleinem ε ist sowohl v_2 als auch v_3 benachbart mit f_3 und beide kommen für einen Schnittpunkt mit f_3 theoretisch in Betracht. Erst durch eine Analyse stellt sich heraus, dass f_3 nur v_3 schneidet. Es sind jedoch jeweils nur direkte Nachbarn zu überprüfen. Dies liegt an folgender Beobachtung, die in [Kar00] formuliert ist:

„Besitzen (genau) zwei Segmente in einem Punkt (x, y) einen (echten) Schnittpunkt, dann gibt es immer eine Stelle $x - \varepsilon$, an der sie in der durch die Sweep line definierte Ordnung direkte Nachbarn sind. An der Stelle x kehrt sich diese Ordnung dann um."

Diese Aussage läßt sich sowohl auf einen echten Schnittpunkt zwischen Liniensegmenten und Strahlen als auch zwischen zwei Strahlen übertragen. Ein „unechter" Schnittpunkt besteht zum Beispiel zwischen f_1 und v_1, da dieser genau auf dem Endpunkt von v_1 liegt. Es gilt zwar auch in diesem Fall, dass die beiden Strahlen für $x < x_1$ in einer hinreichend kleinen ε-Umgebung um den Schnitt herum benachbart sind, allerdings kehrt sich die Ordnung nicht um, da v_1 rechts von x_1 nicht existiert.

Die Ordnung der Liniensegmente und Strahlen führen wir in einer sogenannten *Sweep-Status-Struktur SSS* oder auch *Vertikalstruktur*. Entgegen der intuitiven Vorstellung, dass die Sweep line kontinuierlich über die Problemanordnung wandert, sind nur bestimmte Ereignisse als Haltepunkte zu analysieren. Für Liniensegmente können drei Ereignisse unterschieden werden, wobei davon ausgegangen wird, dass die Sweep line von links nach rechts wandert.

1. Die Sweep line stößt auf den linken Endpunkt eines Liniensegments und dieses wird in SSS aufgenommen.
2. Die Sweep line stößt auf den rechten Endpunkt eines Liniensegments und dieses wird aus SSS entfernt.
3. Die Sweep line stößt auf einen „echten“ Schnittpunkt von zwei Liniensegmenten und die Ordnung der beiden betroffenen Liniensegmente wird vertauscht.

Für Strahlen, die in Richtung $+\infty$ bezogen auf die X-Koordinaten (wie beispielsweise f_2 und f_3) orientiert sind, ist ebenfalls das erste Ereignis zutreffend und das dritte möglich für die von links nach rechts wandernde Sweep line. Ein rechter Endpunkt existiert in diesen Fällen nicht und braucht dementsprechend auch nicht verarbeitet werden. Die Strahlen, die in Richtung $-\infty$ bezogen auf die X-Koordinaten orientiert sind (wie beispielsweise v_1 und f_1), können das zweite und dritte Ereignis auslösen. Im Gegensatz zu Liniensegmenten besteht bei diesen Strahlen kein definierter linker Endpunkt. Durch die Modifikation der Strahlen in Richtung $-\infty$ in Liniensegmente treten in vd nur noch Liniensegmente und Strahlen in Richtung $+\infty$ auf.

vd wird in *SweepIntersectVoronoi* zum Aufbau einer Ereignisstruktur ES, der sogenannten *Horizontalstruktur*, benötigt. Diese speichert für den Sweep die Haltepunkte und ist eine doppelt verkettete Liste.

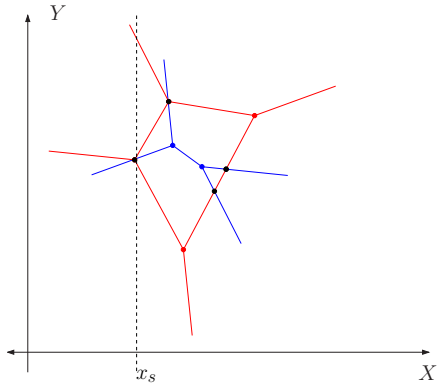


Abbildung 4.27: Startpunkt x_s an einem Schnittpunkt

Zusätzlich zu den Ereignissen linker Endpunkt, rechter Endpunkt und Schnittpunkt führen wir in ES das Ereignis echter Schnittpunkt. Als Schnittpunkt wird ein Ereignis bezeichnet, bei dem ein Voronoi-Knoten auf einer Voronoi-Kante des jeweils anderen Voronoi-Diagramms liegt, wie in Abbildung 4.27 am Startpunkt. Überlappende Voronoi-Kanten werden ebenfalls durch Schnittpunkt als Ereignis verarbeitet. Da nach Lemma 3.11 für den Mittelpunkt von $\mathcal{MWA}(\mathcal{S})$ nur die Voronoi-Knoten von überlappenden Voronoi-Kanten in Frage kommen, werden auch nur diese Punkte verarbeitet. Ein echter Schnittpunkt tritt dann als Ereignis auf, wenn sich die beteiligten Voronoi-Kanten nicht in ihren Endpunkten schneiden.

Der Typ des Haltepunktes wird dementsprechend in linker Endpunkt, rechter Endpunkt, Schnittpunkt und echter Schnittpunkt unterschieden. Bei den Endpunkten werden in dem Ereignis die Voronoi-Kante, die zugehörige Delaunay-Kante und ein Kennzeichen, zu welchem Voronoi-Diagramm die Voronoi-Kante gehört, verwaltet. Diese Variablen entsprechen dem Aufbau eines Elementes in *vd* und wir werden diese im Folgenden unter dem Datentyp *sweepSegment* zusammenfassen und die Variable vom Typ *sweepSegment* in den Ereignissen linker Endpunkt und rechter Endpunkt als *sSegment* bezeichnen.

Zur Charakterisierung der Schnittpunkte dienen zwei Variablen *underSegment* und *overSegment* vom Datentyp *sweepSegment*. Schnittpunkte werden durch zwei Voronoi-Kanten initiiert. In dem Gültigkeitsbereich der beiden Kanten können sie im Allgemeinen bis auf den Schnittpunkt in obere und untere Voronoi-Kante unterschieden werden. Der Sonderfall einer Überlappung von Voronoi-Kanten ermöglicht diese Unterscheidung nicht. Da die Ordnung der Voronoi-Kanten die Vorgehensweise bei diesen „unechten“ Schnittpunkten jedoch nicht beeinflusst, ist nicht relevant, welche Voronoi-Kante als *underSegment* oder *overSegment* abgelegt ist.

In ES wird zusätzlich zu dem Typ des Haltepunktes, die X- und Y-Koordinate des Haltepunktes abgelegt. Die Struktur ist nach folgenden Kriterien sortiert:

1. aufsteigend nach X-Koordinate
2. Typ des Haltepunktes in der Reihenfolge linker Endpunkt, Schnittpunkt, echter Schnittpunkt, rechter Endpunkt
3. Steigung der Voronoi-Kante
4. aufsteigend nach Y-Koordinate

Diese Sortierung gewährleistet, dass alle Ereignisse in der richtigen Reihenfolge abgearbeitet werden. Bei dem Sweep-Algorithmus für unabhängige Liniensegmente wird im Allgemeinen davon ausgegangen, dass an einer X-Koordinate nur ein Ereignis auftritt und damit die Sweep line stetig über die Problemanordnung wandert. Dies schränkt die Menge der Liniensegmente in der Art ein, dass keine „unechten“ Schnittpunkte auftreten und keine Haltepunkte gleiche X-Koordinaten besitzen dürfen. In [Kle97] werden Vorgehensweisen vorgestellt, um diese Eindeutigkeit zu sichern. Dadurch ist die Sortierung über das Kriterium der aufsteigenden X-Koordinate ausreichend für die richtige Reihenfolge der Bearbeitung der Haltepunkte.

Dies ist für Voronoi-Kanten nicht möglich, da Voronoi-Diagramme zusammenhängend sind und ein Voronoi-Knoten für mindestens 3 Kanten den Endpunkt darstellt. Dadurch ist implizit gegeben, dass die Sweep line an einer X-Koordinate solange stehen bleibt, bis alle Ereignisse in der vorgegebenen Sortierung verarbeitet sind.

Bevor der Sweep über das Bearbeiten der Ereignisse begonnen wird, initialisieren wir ES mit allen Endpunkten aus *vd* entsprechend der Sortierung. Die Sweep-Status-Struktur SSS enthält vor dem Sweep keine Elemente.

Erreichen wir mit der Sweep line einen linken Endpunkt, fügen wir ihn in SSS ein und ermitteln damit die direkten Nachbarn der Voronoi-Kante an diesem X-Wert. Für beide wird überprüft, ob sie aus dem anderen Voronoi-Diagramm stammen und einen Schnittpunkt mit der aktuellen Kante haben. Sollte ein echter oder „unechter“ Schnittpunkt existieren, wird das Ereignis in ES aufgenommen. Diese Aufgabe übernimmt der Algorithmus *TestIntersectionCreateEvent*.

Tritt das Ereignis Schnittpunkt auf, analysieren wir durch den Algorithmus *CheckWidth*, ob an dieser Stelle der Mittelpunkt von $\mathcal{MWA}(\mathcal{S})$ liegt. Weitere Aktionen sind hier nicht durchzuführen, da der „unechte“ Schnittpunkt gleichzeitig entweder einen linken oder rechten Endpunkt darstellt. Entsprechende Maßnahmen werden bei den beiden anderen Ereignissen vorgenommen. Im Gegensatz dazu wird bei einem echten Schnittpunkt nicht nur *CheckWidth* aufgerufen, sondern auch die Sortierung von *underSegment* und *overSegment* in SSS vertauscht. Da jetzt beide jeweils einen neuen Nachbarn besitzen, wird für die neue Konstellation jeweils *TestIntersectionCreateEvent* ausgeführt.

Das Erreichen eines rechten Endpunktes mit der Sweep line erfordert die Ermittlung der Nachbarn der aktuellen Voronoi-Kante. Diese wird aus SSS entfernt und die Nachbarn auf Schnitt miteinander geprüft.

SSS und ES werden als doppelt verkettete Liste realisiert. Sowohl ES als auch SSS können trotz der auszuführenden Sortierung beim Einfügen und Löschen eines Elementes effizient genutzt werden. Da jeweils ein Voronoi-Diagramm ein zusammenhängender Graph ist, der konvexe Regionen voneinander trennt, treten die Ereignisse zu einer Kante in der Sortierung dicht aufeinander auf. Zum Einfügen der Elemente in SSS und ES kann eine ähnliche Mimik wie bei *InsertSort* für den Aufbau von vd verwendet werden. Damit ist die Laufzeit des Einfügens in $O(n \log n)$. Aufgrund der Ähnlichkeit wird der Algorithmus *InsertSort* für ES und SSS nicht explizit aufgeführt. Dies gilt ebenso für die Algorithmen *Delete*, *Swap*, *Before*, *Next* für SSS und *NextEs* für ES.

Jeder oben aufgeführte Algorithmus für SSS erfordert eine Suche eines Elementes. Auch hier wird ausgenutzt, dass die gesuchte Voronoi-Kante effizient über die als letzte verarbeitete Voronoi-Kante gefunden werden kann. Die Aktion selbst wie beispielsweise löschen eines Elementes, vertauschen zweier Elemente kann in konstanter Zeit durchgeführt werden. In ES wird über *NextES* jeweils das nächste aktuelle Ereignis aus der Liste geliefert und danach gelöscht. Diese Aktion benötigt konstante Laufzeit.

Der Zeitbedarf von *SweepIntersectVoronoi* liegt in $O(n^2 \log n)$. Diese Aussage lässt sich anhand der einzelnen Arbeitsschritte nachvollziehen. Die Vorarbeiten, die mit *ConstructVD* geleistet werden, benötigen eine Laufzeit von $O(n \log n)$. Die Voronoi-Diagramme können zwar in linearer Zeit aus den entsprechenden Delaunay-Triangulationen ermittelt werden, die Sortierung von vd und $vd_{-\infty}$ liegt allerdings in $O(n \log n)$. *FindStartX* benötigt lineare Laufzeit. Der Zeitbedarf des Sweep richtet sich nach der Anzahl der Ereignisse und die pro Ereignis auszuführenden Aktionen. Es können $O(n^2)$ Haltepunkte auftreten. Die Anzahl der Endpunkte beträgt zwar

höchstens $2n$, da jedes Voronoi-Diagramm $O(n)$ Kanten besitzt, aber die Kanten können sich jeweils schneiden. Dadurch liegt die Anzahl der möglichen Schnittpunkte in $O(n^2)$. Da SSS und ES sortiert geführt werden, ist dies ein Zeitaufwand von $O(\log n)$ pro Haltepunkt. Daraus ergibt sich die Laufzeit von $O(n^2 \log n)$.

Algorithmus 21 SweepIntersectVoronoi(dt, fdt)

```
vd := null, vd_-∞ := null
ConstructVD(dt, 3π/2, vd, vd_-∞, true)
ConstructVD(fdt, π/2, vd, vd_-∞, false)
FindStartX(vd, vd_-∞)
Init(mMWA, wMWA, innerRadius, outerRadius)
SSS := null
InitES(vd)
while ES ≠ ∅ do
  event :=NextES()
  if event.type =LeftPoint then
    InsertSort(SSS, event)
    beforeSegment := Before(SSS, event, event.sSegment)
    TestIntersectionCreateEvent(ES, beforeSegment, event.sSegment)
    nextSegment := Next(SSS, event, event.sSegment)
    TestIntersectionCreateEvent(ES, event, nextSegment)
  else if event.type =Intersection then
    CheckWidth(event)
  else if event.type =RealIntersection then
    CheckWidth(event)
    Swap(SSS, event)
    beforeSegment := Before(SSS, event, event.overSegment)
    TestIntersectionCreateEvent(ES, beforeSegment, event.overSegment)
    nextSegment := Next(SSS, event, event.underSegment)
    TestIntersectionCreateEvent(ES, event.underSegment, nextSegment)
  else if event.type =RightPoint then
    beforeSegment := Before(SSS, event, event.sSegment)
    nextSegment := Next(SSS, event, event.sSegment)
    Delete(SSS, event.sSegment, event.x)
    TestIntersectionCreateEvent(ES, beforeSegment, nextSegment)
end
```

Der Algorithmus *TestIntersectionCreateEvent* prüft zwei Voronoi-Kanten auf Schnittpunkt und erzeugt gegebenenfalls ein Ereignis. Diese Analyse erfolgt nur in dem Fall, wenn die beiden als Parameter übergebenen Kanten aus verschiedenen Voronoi-Diagrammen stammen. *IntersectSegRay* ist eine Erweiterung von *IntersectSegments* und so angepasst, dass der Schnittpunkt zwischen zwei Liniensegmenten, zwei Strahlen und einem Liniensegment mit einem Strahl verarbeitet wird. Existiert ein Schnitt-

punkt, werden nicht nur die Koordinaten berechnet, sondern auch der Typ des Schnittpunktes unterschieden. Diese Unterscheidung in echten und „unechten“ Schnittpunkt wird für das Erzeugen eines Ereignisses benötigt, das mit den entsprechenden Daten gefüllt wird.

Algorithmus 22 TestIntersectionCreateEvent($ES, segment1, segment2$)

```
if not segment1.isVD = segment2.isVD then
  (* die Voronoi-Kanten sind aus verschiedenen Voronoi-Diagrammen *)
  s := IntersectSegRay(segment1.voronoiEdge, segment2.voronoiEdge)
  if s.type = RealIntersection or s.type = Intersection then
    (* ein Schnittpunkt existiert *)
    InsertSort(ES, s.type, s.x, s.y, segment1, segment2)
  end
end
```

Um die Breite eines Rings um einen Schnittpunkt in *CheckWidth* zu berechnen, ist zu analysieren, welche Voronoi-Kante zu $\mathcal{VD}(\mathcal{S})$ bzw. $\mathcal{FVD}(\mathcal{S})$ gehört. Ist dies überprüft, kann über einen Endpunkt der entsprechenden Delaunay-Kante der Abstand zum Schnittpunkt und damit der Radius eines Kreises ermittelt werden. Diese Vorgehensweise kann so gewählt werden, da die Endpunkte der Delaunay-Kante den Punkten aus S entsprechen, die die Voronoi-Kante initiieren. Sollte die Differenz zwischen den Radien kleiner als die bisher bestimmte Breite $wMWA$ sein, wird der Schnittpunkt als Mittelpunkt für den Ring mit minimaler Breite festgelegt.

Algorithmus 23 CheckWidth(*event*)

```
(* Berechnung der Radien *)
if event.undersegment(isVD) then
  iRadius :=distance(event.underSegment.delaunayEdge(point1), event.x)
  oRadius :=distance(event.overSegment.delaunayEdge(point1), event.x)
else
  iRadius :=distance(event.overSegment.delaunayEdge(point1), event.x)
  oRadius :=distance(event.underSegment.delaunayEdge(point1), event.x)
end
width := oRadius − iRadius
if width < wMWA then
  (* die Breite des aktuellen Rings ist kleiner als des bisher gefundenen *)
  wMWA := width
  mMWA := (event.x, event.y)
  innerRadius := iRadius
  outerRadius := oRadius
end
```

Nach der Ausführung von *SweepIntersectVoronoi* sind alle Schnittpunkte der Voronoi-Kanten aus $\mathcal{VD}(\mathcal{S})$ und $\mathcal{FVD}(\mathcal{S})$ daraufhin untersucht worden, welcher den Mittel-

punkt für einen Ring minimaler Breite darstellt. Dies bedeutet nicht, dass $\mathcal{MWA}(\mathcal{S})$ ein Ring und kein Streifen ist. Der Vergleich mit der Breite der Punktmenge erfolgt erst bei der Ausgabe von $\mathcal{MWA}(\mathcal{S})$.

4.5.2 Konstruktion des am besten angepassten Kreis $\mathcal{BFC}(\mathcal{S})$

Für die Konstruktion von $\mathcal{BFC}(\mathcal{S})$ ist kein zusätzlicher Algorithmus notwendig, sondern eine Ausgabefunktion *DrawBFC* ausreichend. Die Berechnungen werden in *MinimumWidthAnnulus* durchgeführt. Liegt $\mathcal{MWA}(\mathcal{S})$ als Ring vor, wird $\mathcal{BFC}(\mathcal{S})$ durch den am besten angepassten Kreis dargestellt. Anderenfalls ist $\mathcal{BFC}(\mathcal{S})$ eine Gerade. Wir unterscheiden die beiden Fälle, ob $\mathcal{MWA}(\mathcal{S})$ als Streifen oder als Gerade, die als spezieller Streifen der Breite 0 aufgefasst werden kann, ermittelt wurde. Im Fall des Streifens verläuft die Gerade $\mathcal{BFC}(\mathcal{S})$ parallel innerhalb des Streifens im Abstand $\frac{width}{2}$ zu den begrenzenden Geraden. Ist $\mathcal{MWA}(\mathcal{S})$ eine Gerade, entspricht diese Gerade $\mathcal{BFC}(\mathcal{S})$.

DrawBFC:

```
if mMWA < width then
    (* MWA(S) ist ein Ring *)
    bestRadius := (innerRadius + outerRadius)/2
    drawCircle(mMWA, bestRadius)
else
    (* MWA(S) ist ein Streifen *)
    (p, r, q) :=getTripel(w)
    ifr ≠ 0 then
        (* S besteht aus mindestens zwei Punkten *)
        if width = 0 then
            drawLine(p, r)
        else
            drawLine(parallel(p, r, q, width/2))
    end
end
```

Aufgrund dieser Vorgehensweise entsprechen die Reaktionen auf Einfüge-, Lösch- und Verschiebeereignisse den Reaktionen, die bereits zu $\mathcal{MWA}(\mathcal{S})$ beschrieben wurden. Dementsprechend werden sie an dieser Stelle nicht wiederholt.

Sobald entschieden ist, dass etwas gemacht werden kann und soll,
werden wir auch einen Weg dazu finden.
Abraham Lincoln (1809 - 1865)

5

Implementierung

Für die Implementierung der Algorithmen haben wir die objektorientierte Programmiersprache Java gewählt. Durch die Verbreitung des Internets und der immer größer werdenden praktischen Bedeutung des *World Wide Web (WWW)* hat sich diese universelle Programmiersprache in vielen Bereichen etabliert. Ihre Stärke liegt in der plattformübergreifenden Einsetzbarkeit. Dieser Vorteil führt allerdings neben Eigenschaften wie eingebauten Sicherheitsmechanismen zu einem Performanceverlust gegenüber anderen objektorientierten Sprachen, die direkt in Maschinencode kompiliert werden. In *Java Applets for the Dynamic Visualization of Voronoi-Diagrams* [IKKM03] wird Java analysiert und trotz eingeschränkter Performance für Forschung und Lehre empfohlen. Java-Applets, die sich in die Seiten des *WWW* integrieren lassen und somit über das Internet geladen und lokal ausgeführt werden können, bieten die Möglichkeit, jederzeit von beliebigen Rechnern mit einer installierten *Java Virtual Machine* ausführbar zu sein.

Das aus dieser Arbeit resultierende Java-Applet *FitCircle* gibt zu einer Punktmenge S den größten leeren Kreis $\mathcal{LEC}(\mathcal{S})$, den kleinsten umfassenden Kreis $\mathcal{SEC}(\mathcal{S})$, den Ring mit minimaler Breite $\mathcal{MWA}(\mathcal{S})$ und den am besten angepassten Kreis $\mathcal{BFC}(\mathcal{S})$ aus. *FitCircle* ist so aufgebaut worden, dass es die bereits existierenden Java-Klassen des *Geometrie-Labors* der FernUniversität Hagen [urld] nutzt und problemlos integriert werden kann. Die erstellten Java-Klassen sind sowohl als Sourcecode als auch in kompilierter Form auf dem CVS-Server *cvs.pi6.fernuni-hagen.de* unter */data/cvs/GeomLab/FitCirc* gespeichert.

Die Methoden in *FitCircle* stellen die Implementation der vorgestellten Algorithmen im vorherigen Kapitel dar. Dementsprechend werden degenerierte Eingabedaten jeweils gesondert verarbeitet. Dies erfordert die Untersuchung der Punktmengen

auf Kollinearität und Kozirkularität. Um Rundungsfehler zu umgehen, die zu einem verfälschten Ergebnis führen können, wird dies über Determinanten-Berechnung von Matrizen gelöst. Bei der Berechnung werden dadurch nur die drei Operationen Addition, Subtraktion und Multiplikation benötigt. Diese Vorgehensweise gewährleistet, dass bei der Verwendung von Fließkommazahlen nur sehr kleine Rundungsfehler entstehen. Diese können für die Größenordnung, in der sich die Koordinaten der Punktmenge befinden, vernachlässigt werden. Die gleiche Strategie wird unter anderem bei Vergleichen von Abständen angewandt.

An einigen Stellen, wie beispielsweise bei dem Vergleich der Breite einer Punktmenge und der Breite eines Rings, arbeiten wir mit quadratischen Abständen. Erst für die Ausgabe einer geometrischen Struktur ist der genaue Wert eines Abstands zu berechnen. Die Ausführung der Wurzelfunktion zum letztmöglichen Zeitpunkt hat den Vorteil, dass die Berechnung im Vorfeld nicht ausgeführt werden muss. Dies hat positiven Einfluss auf die Genauigkeit bei Vergleichen von Abstandswerten und ebenfalls auf die Laufzeit der Methoden.

Beginnen können ist Stärke, vollenden können ist Kraft.
Laotse (4. - 3. Jh. v. Chr)

6

Abschließende Bemerkungen

In der vorliegenden Arbeit werden Punktmengen und Kreise in der euklidischen Ebene betrachtet. Der strukturelle Zusammenhang zwischen den Voronoi-Diagrammen und den analysierten Kreisen resultiert unter anderem aus der Festlegung der Voronoi-Diagramme durch Bisektoren. Es wird die Eigenschaft des klassischen Voronoi-Diagramms ausgenutzt, dass ein Kreis mit Mittelpunkt auf einer Voronoi-Kante bzw. einem Voronoi-Knoten, deren zugehörigen Elemente aus der Punktmenge auf dem Kreisrand liegen, leer ist. Kreise mit Mittelpunkt auf einer Voronoi-Kante bzw. einem Voronoi-Knoten enthalten die gesamte Punktmenge, wenn der Kreis durch die Punkte führt, die die Voronoi-Regionen des furthest-point Voronoi-Diagramm initiieren.

Diese beiden Charakteristika lassen sich auch auf höherdimensionale Räume übertragen. Analog zu den Kreisen können im $\mathbb{R}^d$ anhand der Voronoi-Diagramme d-dimensionale Kugeln bestimmt werden, die entweder keinen oder alle Punkte enthalten. Der Ring in der Ebene ist äquivalent zu einer d-Kugelschale in höherdimensionalen Räumen, dessen Mittelpunkt ebenfalls auf einem Schnittpunkt der beiden Voronoi-Diagramme liegt.

Die vorgestellten Algorithmen lassen sich allerdings nicht alle auf höhere Dimensionen übertragen. Für den dreidimensionalen Raum können *Space-Sweep*-Algorithmen mit einer Ebene als Scanner-Geometrie eingesetzt werden. Darüber sind sowohl die beiden Voronoi-Diagramme als auch deren Schnittstellen bestimmbar. Der Algorithmus zur Ermittlung des kleinsten umfassenden Kreises ist im dreidimensionalen Raum mit Anpassungen ebenfalls anwendbar. Allerdings benutzt der Algorithmus zur Bestimmung des größten leeren Kreises die Eigenschaft, dass die Lage der Voronoi-Knoten nach einem Durchlauf durch die Dreiecke der Delaunay-Triangulation auf dem Rand der konvexen Hülle bekannt ist. Dies trifft nicht für den dreidimensionalen Fall zu.

Die effiziente Konstruktion der geometrischen Strukturen in höheren Dimensionen als der dritten erfordert andere Vorgehensweisen. *Sweep*-Algorithmen sind in diesem Fall nicht mehr einsetzbar. Brown hat in [Bro79] einen optimalen Algorithmus zur Erstellung des klassischen Voronoi-Diagramms im $\mathbb{R}^d$ veröffentlicht. Dabei wird das Problem der Konstruktion des Voronoi-Diagramms von n Punkten auf die Konstruktion der konvexen Hülle von n Punkten im $\mathbb{R}^{d+1}$ reduziert. Ein Ansatz zum Lösen der geometrischen Probleme in höherdimensionalen Räumen ist die Generalisierte Lineare Programmierung (siehe zum Beispiel [Fuh98]).

Literaturverzeichnis

[AAHPS99] Agarwal, Pankaj K., Boris Aronov, Sariel Har-Peled und Micha Sharir: *Approximation and Exact Algorithms for Minimum-Width Annuli and Shells.* In: *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, Seiten 380–389, 1999.

[AGSS87] Aggarwal, A., L. J. Guibas, J. Saxe und P. W. Shor: *A linear time algorithm for computing the Voronoi diagram of a convex polygon.* In: *Proc. 19th Annu. ACM Sympos. Theory Comput.*, Seiten 39–45, 1987.

[AK96] Aurenhammer, Franz und Rolf Klein: *Voronoi Diagrams.* Technischer Bericht 198, Department of Computer Science, FernUniversität Hagen, Germany, 1996.

[AKSS] Adam, B., P. Kauffmann, D. Schmitt und J.-C. Spehner: *A shrinking-circle sweep-algorithm to construct the farthest site Delaunay diagram in the plane.*

[AM91] Abramowski, S. und H. Müller: *Geometrisches Modellieren.* BI-Wissenschaftsverlag, Mannheim, 1991.

[Aur91] Aurenhammer, F.: *Voronoi diagrams: A survey of a fundamental geometric data structure.* ACM Comput. Surv., 23(3):345–405, September 1991.

[Bes01] Bespamyathnikh, S. N.: *An efficient algorithm for the three-dimensional diameter problem.* Discrete Comput. Geom., 25:235–255, 2001.

[BFS02] Byers, Simon, Juliana Freire und Cláudio T. Silva: *Efficient Acquisition of Web Data through Restricted Query Interfaces.* AT&T Labs-Research, Bell Laboratories, 2002.

[Bro79] Brown, K. Q.: *Voronoi diagrams from convex hulls.* Inform. Process. Lett., 9(5):223–228, 1979.

[BS04] Bandyopadhyay, D. und J. Snoeyink: *Almost-Delaunay Simplices: Nearest Neighbor Relations for Imprecise Points.* 2004.

[BSMM93] Bronstein, I. N., K. A. Semendjajew, G. Musiol und H. Mühlig: *Taschenbuch der Mathematik.* Harri Deutsch, Frankfurt am Main, 1993.

[Cha00] Chan, Timothy M.: *Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus.* In: *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, Seiten 300–309, 2000.

[CL03] Chernov, N. und C. Lesort: *Least squares fitting of circles and lines.* 2003.

[dBBB+98] Berg, M. de, P. Bose, D. Bremner, S. Ramaswami und G. Wilfong: *Computing Constrained Minimum-Width Annuli of Point Sets.* Comput. Aided Design, 30(4):267–275, April 1998.

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

[Del34] Delaunay, B.: *Sur la sphère vide. A la memoire de Georges Voronoi.* Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk, 7:793–800, 1934.

[Des44] Descartes, R.: *Principia Philosophiae.* Ludovicus Elzevirius, Amsterdam, 1644.

[Dir50] Dirichlet, G. L.: *Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen.* J. Reine Angew. Math., 40:209–227, 1850.

[Dum99] Dumke, R.: *Einführung, Algorithmen und Datenstrukturen.* Technischer Bericht, Department of Computer Science, Otto-von-Guericke-Universität Magdeburg, Germany, 1999.

[EM90] Edelsbrunner, H. und E. P. Mücke: *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms.* ACM Trans. Graph., 9(1):66–104, 1990.

[For86] Fortune, S.: *A sweepline algorithm for Voronoi diagrams.* In: *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, Seiten 313–322, 1986.

[For95] Fortune, S.: *Voronoi diagrams and Delaunay triangulations.* In: Du, D.-Z. und F. K. Hwang (Herausgeber): *Computing in Euclidean Geometry*, Band 4 der Reihe *Lecture Notes Series on Computing*, Seiten 225–265. World Scientific, Singapore, 2nd Auflage, 1995.

[For03] Forst: *Farthest-Point-Voronoi-Diagramme.* FernUniversität Hagen, Praktische Informatik VI, 2003. Seminararbeit.

[Fuh98] Fuhrmann, Artur: *Approximation konvexer Polyeder in der Hausdorff-Metrik.* Diplomarbeit, Universität des Saarlandes, Saarbrücken, Saarbrücken, Germany, 1998.

[Gra72] Graham, R. L.: *An efficient algorithm for determining the convex hull of a finite planar set.* Inform. Process. Lett., 1:132–133, 1972.

[hL03] Liao, Qing hu: *On Removing Extrinsic Degeneracies in Computational Geometry.* Project für CS507 - Computational Geometry, 2003.

[HSB02] Hase, Hayo, Wolfgang Schlüter und Armin Böer: *TIGO and its Future Application for the ITRF.* Bundesamt für Kartographie und Geodäsie, Fundamentalstation Wettzell, Kötzting, Germany, 2002.

[IKKM03] Icking, Christian, Rolf Klein, Peter Köllner und Lihong Ma: *Java Applets for the Dynamic Visualization of Voronoi Diagrams.* In: *Computer Science in Perspective*, Band 2598 der Reihe *Lecture Notes Comput. Sci.*, Seiten 191–205. Springer-Verlag, 2003.

[Kar00] Karch, Oliver: *Algorithmische Geometrie.* Universität Würzburg, 2000.

[Klö97] Klötzer, Frank: *Integration von triangulierten digitalen Geländemodellen und Landkarten.* Diplomarbeit, Rheinische Friedrich-Wilhelmsuniversität Bonn, Bonn, Germany, 1997.

[Kle89] Klein, Rolf: *Concrete and Abstract Voronoi Diagrams*, Band 400 der Reihe *Lecture Notes Comput. Sci.* Springer-Verlag, 1989.

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

[LS87] Leven, D. und Micha Sharir: *Intersection and proximity problems and Voronoi diagrams.* In: Schwartz, J. T. und C.-K. Yap (Herausgeber): *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, Seiten 187–228. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[Nee88] Nees, G.: *Regentengraphik und das ästhetische Laboratorium.* Erlangen, 1988.

[OBS92] Okabe, Atsuyuki, Barry Boots und Kokichi Sugihara: *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, Chichester, UK, 1992.

[PS85] Preparata, F. P. und M. I. Shamos: *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[Ram99] Ramos, Pedro A.: *Computing Roundness is Easy if the Set is Almost Round.* In: *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, Seiten 307–315, 1999.

[Riv79] Rivlin, T. J.: *Approximating by circles.* Computing, 21:93–104, 1979.

[Sau95] Sauer, Jörg: *Allgemeine Kollisionserkennung und Formrekonstruktion basierend auf Zellkomplexen.* Diplomarbeit, Universität des Saarlandes, Saarbrücken, Germany, 1995.

[SH75] Shamos, M. I. und D. Hoey: *Closest-Point Problems.* In: *Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci.*, Seiten 151–162, 1975.

[Sha78] Shamos, M. I.: *Computational Geometry.* Ph.D. Thesis, Dept. Comput. Sci., Yale Univ., New Haven, CT, 1978.

[She96] Shewchuk, Jonathan R.: *Robust Adaptive Floating-Point Geometric Predicates.* In: *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, Seiten 141–150, 1996.

[She97] Shewchuk, Jonathan Richard: *Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates.* Discrete Comput. Geom., 18(3):305–363, 1997.

[SLW95] Swanson, K., D. T. Lee und Vanban L. Wu: *An optimal algorithm for roundness determination on convex polygons.* Comput. Geom. Theory Appl., 5:225–235, 1995.

[Smi97] Smid, M.: *Algorithmische Geometrie: Reine Theorie?* 1997. Antrittsvorlesung.

[Ste99] Stewart, Ian: *Krater, Kunst und Kegelschnitte.* Spektrum der Wissenschaft, Seiten 144–146, Mai 1999. Mathematische Unterhaltungen.

[Syl57] Sylvester, J. J.: *A question on the geometry of situation.* Quart. J. Math., 1:79, 1857.

[Tin98] Tinnefeld, Karsten: *Stabilität in Delaunaytriangulierungen und Voronoidiagrammen unter Störungen der Eingabepunkte.* Diplomarbeit, Universität Dortmund, Dortmund, Germany, 1998.

[UJ00] Umbach, Dale und Kerry N. Jones: *A Few Methods for Fitting Circles to Data.* IEEE Trans. on Instrumentation and Measurement, 20, 2000.

[urla] *http://cs.valberta.ca/ graphics/software/ledacgal.html.*

[urlb] *http://www.cs.cmu.edu/∼quake/robust.html.*

[urlc] *http://www.kopfmensch.privat.t-online.de/german/geometrie/compgeom/sweep.htm.*

[urld] *http://wwwpi6.fernuni-hagen.de/GeomLab/.*

[Ver99] Verbeek, R.: *Einführung in die Theoretische Informatik B.* FernUniversität Hagen, 1999.

[Vor07] Voronoi, G. M.: *Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier Mémoire: Sur quelques propriétés des formes quadratiques positives parfaites.* J. Reine Angew. Math., 133:97–178, 1907.

[Yap95] Yap, C. K.: *Exact computational geometry and tolerancing metrology.* In: Avis, D. und J. Bose (Herausgeber): *Snapshots of Computational and Discrete Geometry, Vol. 3, Tech. Rep. SOCS-94.50.* McGill School of Comp. Sci., 1995.

[YB61] Yaglom, I. M. und V. G. Bolyanski: *Convex Figures.* English Translation, Holt, Rinehart and Winston, New York, NY, 1961.

Abbildungsverzeichnis

Algorithmenverzeichnis

A

Anhang

A.1 Benutzeranleitung

Das FitCircle-Applet animiert zu einer Punktmenge den größten leeren Kreis, den kleinsten umfassenden Kreis, den Ring minimaler Breite und den am besten angepassten Kreis zu einer Punktmenge in der euklidischen Ebene. Die Berechnung erfolgt anhand der in Kapitel 4 angegebenen Algorithmen.

Der Anwender kann selbst eine Punktmenge kreieren. Punkte können durch Drücken der linken Maustaste hinzugefügt, der rechten Maustaste gelöscht oder bei gedrückter linker Maustaste verschoben werden.

Weiterführende Funktionalitäten stehen in der Menüleiste zur Auswahl.

- **File**
  Exit: Das Applet wird beendet.
- **Show**
  Folgende verschiedene geometrische Strukturen zu einer Punktmenge können in der euklidischen Ebene verschiedenfarbig angezeigt werden:
  Largest Empty Circle
  Smallest Enclosing Circle
  Minimum Width Annulus
  Best Fitting Circle
  furthest-point Voronoi-Diagram
  nearest-point Voronoi-Diagram
  Convex Hull
- **Edit**
  Clear: Das Panel wird initialisiert, also alle Punkte und geometrischen Objekte aus der Oberfläche entfernt.

RunDemo: Es werden automatisch 20 Punkte hinzugefügt, wobei nur die geometrischen Strukturen angezeigt werden, die der Anwender ausgewählt hat.

- **Help**
  About: Der Anwender erhält eine Auskunft über die Version des Applets.
  Quick Help: Entsprechend des Titels wird eine kurze Hilfe zu dem Applet aufgerufen.

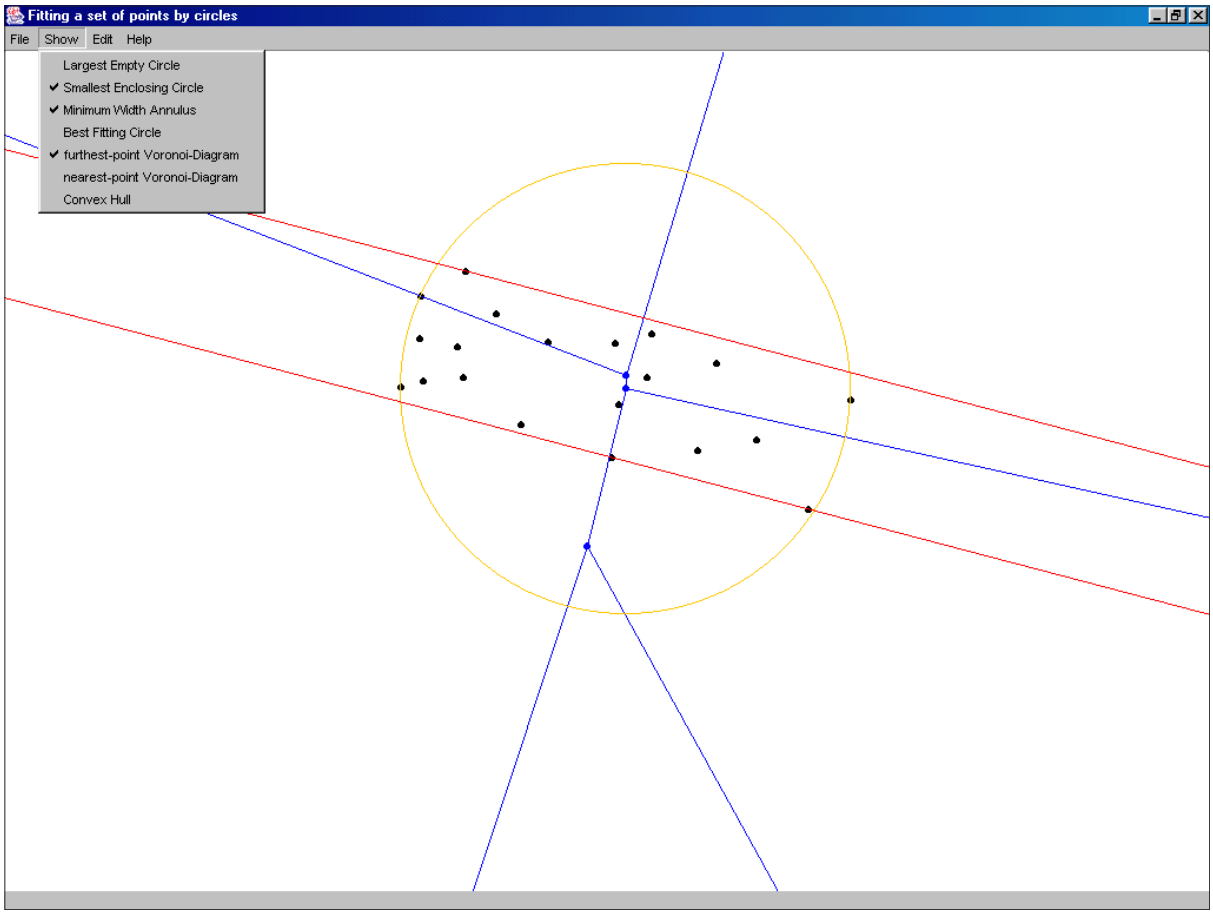


Abbildung A.1: Das FitCircle-Applet

A.2 Erklärung

Coesfeld, den 11. März 2005

Hiermit versichere ich, dass ich diese Diplomarbeit selbständig verfasst, alle Zitate kenntlich gemacht und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Sandra Gesing