Geometrische Algorithmen in der Flächenrückführung

Dissertation

zur Erlangung des Grades des
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik
(D290)

von
Dipl.-Inform. Frank Weller

Dortmund
2000

Tag der mündlichen Prüfung: 11. September 2000

Dekan:

Prof. Dr. Bernd Reusch

Gutachter:

Prof. Dr. Heinrich Müller
Universität Dortmund

Prof. Dr. Hans Hagen
Universität Kaiserslautern

Priv.-Doz. Dr. Paul Fischer
Universität Dortmund

Ich danke allen Freunden und Kollegen, ohne deren hilfreiche Unterstützung diese Dissertation nicht zustandegekommen wäre. Mein besonderer Dank gilt meinen Eltern, Paul Fischer, Christa Gnass, Mady Gruys, Hans Hagen, Eva Kreutzberger, Heinrich Müller und Thomas Schreiber.

Inhalt

Zusammenfassung 3

Kapitel 1. Einleitung 5
1.1. Flächenrückführung 5
1.2. Gegenstand der Arbeit 9
1.3. Definitionen und Notationen 11

Kapitel 2. Korrektheit des Oriented-Walk-Algorithmus 17
2.1. Der Oriented-Walk-Algorithmus 18
2.2. Voronoi– und Potenzdiagramme 24
2.3. Delaunaydiagramme sind im wesentlichen Potenzdiagramme 26
2.4. Reguläre Diagramme 30

Kapitel 3. Nicht-konvexe Triangulierungen 35
3.1. Triangulierung bestimmter Polygone durch Graham's Scan 36
3.2. Triangulieren mit Löchern 42

Kapitel 4. Konvexe Hüllen auf der Sphäre 53
4.1. Konvexe Mengen und einfache Polygonzüge auf der Sphäre 53
4.2. Die konvexe Hülle eines sphärischen Polygonzugs 60
4.3. Anwendung: Totale Absolutkrümmung von Dreiecksnetzen 68

Kapitel 5. Stabilität in Delaunaydiagrammen 73
5.1. Stabilität 73
5.2. Eine neue Charakterisierung von Stabilität 75
5.3. Das Entscheidungsproblem 79
5.4. Das Optimierungsproblem 86

Kapitel 6. Polyedrische Rekonstruktion durch Delaunay-Stabilität 95
6.1. Ausgangspunkt 95
6.2. Fehlermodelle für Kurven– und Flächenrekonstruktion 97
6.3. Flächenrekonstruktion durch Stabilität 103

Kapitel 7. Effiziente Aufzählung polygonaler Hüllen 111
7.1. Das Problem 111
7.2. Der Aufzählalgorithmus 113

Literaturverzeichnis 133

Zusammenfassung

Gegenstand der Flächenrückführung ist, aus einer gegebenen Menge von Abtastpunkten einer Fläche eine Näherung zu rekonstruieren, die die Fläche möglichst gut repräsentiert. Ein weit verbreiteter Ansatz ist, die rekonstruierte Fläche durch ein Netz aus Polygonen, meist Dreiecken, zu beschreiben. Die Schwierigkeit besteht darin, unter den kombinatorisch vielen Möglichkeiten eine „gute” Rekonstruktion zu erhalten, insbesondere für den Fall, daß die ursprünglich gegebene Fläche nicht bekannt ist. Im Zusammenhang mit Verfahren zur Flächenrückführung treten vielfältige geometrische Teilprobleme auf, die für sich gesehen interessant sind. In dieser Arbeit werden für eine Reihe solcher Probleme effiziente Algorithmen entwickelt. Zu nennen sind Korrektheitsbetrachtungen für den gebräuchlichen Oriented-Walk-Algorithmus, Triangulierung innerhalb frei wählbarer, nicht konvexer Gebiete in der Ebene, Berechnung konvexer Hüllen von Polygonen auf Sphären mit linearem Zeitaufwand, Stabilität von Delaunay-Facetten mit Anwendung auf die Rekonstruktion geschlossener Flächen sowie effiziente Aufzählung polygonaler Hüllen.

KAPITEL 1

Einleitung

1.1. Flächenrückführung

Flächenrückführung bedeutet, aus einer gegebenen Menge von Abtastpunkten einer Fläche eine Näherung zu rekonstruieren, die die Fläche möglichst gut repräsentiert. Diese Rekonstruktion kann als Approximations– oder als Interpolationsproblem verstanden werden. In dieser Arbeit wird von einer Interpolationsaufgabe ausgegangen. Ein weit verbreiteter Ansatz in diesem Fall ist, die rekonstruierte Fläche mit einem Netz aus Polygonen, meist Dreiecken, zu überziehen. Die Interpolationsbedingung erfüllt man dadurch, daß die Eckpunkte der Dreiecke genau die Abtastpunkte sind. Die Kanten und Dreiecke des Netzes sollen nur solche Abtastpunkte miteinander verbinden, die innerhalb der abgetasteten Fläche „benachbart" sind. Je nach Anforderung kann ein so konstruiertes Dreiecksnetz bereits das Endergebnis der Flächenrückführung darstellen, oder die Grundlage für Freiform-Interpolationsverfahren bilden, siehe z. B. [**AM91, MLLM$^+$92, Baj92, Nie97**].

Bei gestreut abgetasteten Flächen erweist sich die Flächenrückführung als ein schwieriges Problem. Dies liegt daran, daß nicht ohne weiteres klar ist, wie die Nachbarschaften der Abtastpunkte auf der gesuchten Fläche zu wählen sind. Für eine gegebene Punktmenge sind daher üblicherweise zahlreiche Lösungen möglich, unter denen die gewünschte passend zu charakterisieren ist. Eine Einführung in diese Problematik und eine recht umfassende Übersicht von Rekonstruktionsverfahren findet sich in [**MM97, MM98**].

Das Problem der Flächenrückführung tritt insbesondere im Zusammenhang mit dem Entwurf und der Fertigung von Werkstücken im Maschinenbau auf, siehe z. B. [**Fri97**]. Massenproduzierte Teile werden dabei oft so konstruiert, daß ihre Oberfläche sich aus einer von Hinterschnitten freien, d. h. injektiv projizierbaren „Oberseite" und einer ebensolchen „Unterseite" zusammensetzt. Eine Reihe von Fertigungsverfahren erfordert diese Art der Konstruktion, z. B. Gesenkschmieden, Tiefziehen und Gießen. Bei injektiv projizierbaren Flächen kann die Berechnung eines Dreiecksnetzes durch Triangulieren einer planaren Punktmenge geschehen. Dazu werden die Abtastpunkte in eine Ebene projiziert, über der die Fläche injektiv liegt, und dort trianguliert. Das Dreiecksnetz ergibt sich dann durch direktes Übertragen der planaren Triangulierung auf die ursprünglichen Punkte. Bei einer injektiv projizierbaren abgetasteten Fläche ist damit die Konstruktion eines Dreiecksnetzes durch planare Triangulierung recht einfach möglich.

Auf den ersten Blick scheint damit das Flächenrückführungsproblem für den injektiv projizierbaren Fall gelöst. Es zeigt sich aber, daß die Wahl der Triangulierung Einfluß auf die Qualität der Approximation hat und es im allgemeinen nicht genügt, eine beliebige Triangulierung zu nehmen. Die Abbildungsserie 1.1–1.4 illustriert dies anhand unterschiedlicher Triangulierungen einer gegebenen Fläche. Abbildung 1.1 zeigt die Frankesche Klippenfunktion, eine extrudierte Arcustangenskurve [**Fra82**]. Auf dieser Fläche wurden 100 Abtastpunkte verteilt. Neben der Delaunaytriangulierung der Abtastpunkte sehen wir zwei sogenannte *datenabhängige* Triangulierungen, ABN und TAC.

Die Qualität einer Triangulierung kann über eine Zielfunktion beschrieben werden. Die Delaunaytriangulierung z. B. erfüllt das *Max-Min-Angle-Kriterium,* d. h. sie maximiert den kleinsten in einem Dreieck auftretenden Innenwinkel. Diese Winkel werden in der (Projektions-)Ebene gemessen, das Kriterium beachtet also nicht die räumliche Lage der Abtastpunkte vor der Projektion. Die Triangulierung in der Projektionsebene wird dadurch sehr gleichförmig, aber ihr Dreiecksnetz zeigt im Bereich der Klippe eine starke Aufrauhung (Abbildung 1.2). Datenabhängige Zielfunktionen hingegen bewerten direkt das Dreiecksnetz. Das *ABN-Kriterium* maximiert die Keilwinkel zwischen benachbarten Dreiecken des

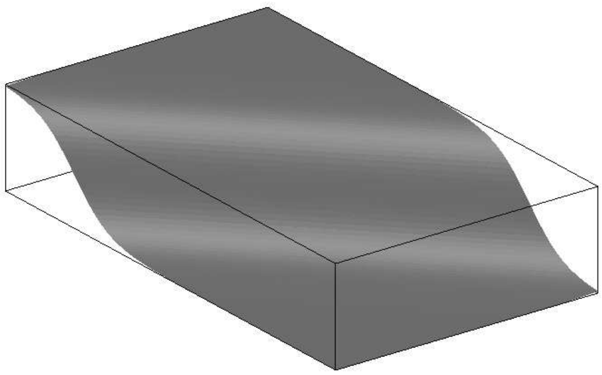


ABBILDUNG 1.1. Frankesche Klippenfunktion.

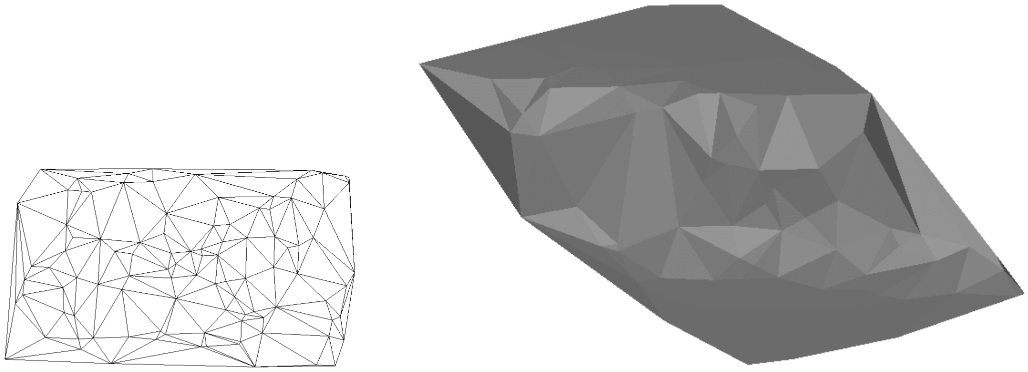


ABBILDUNG 1.2. Delaunay-Triangulierung (links) und korrespondierendes Dreiecksnetz (rechts) für die Klippenfunktion.

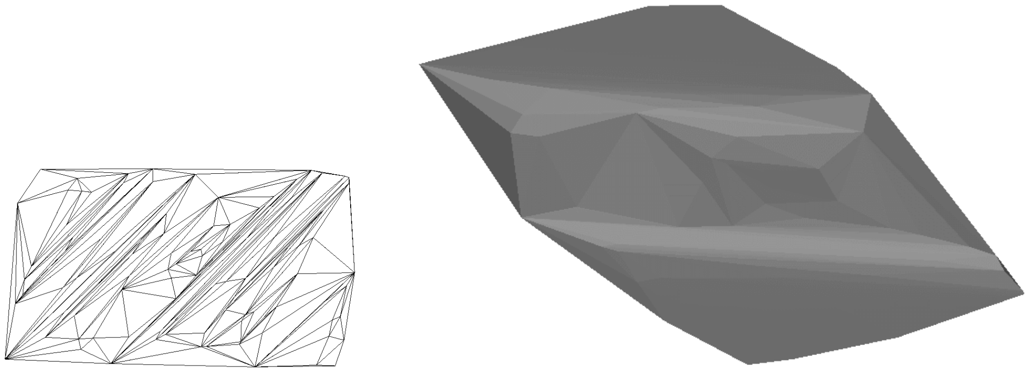


ABBILDUNG 1.3. ABN-Triangulierung der Klippenfunktion.

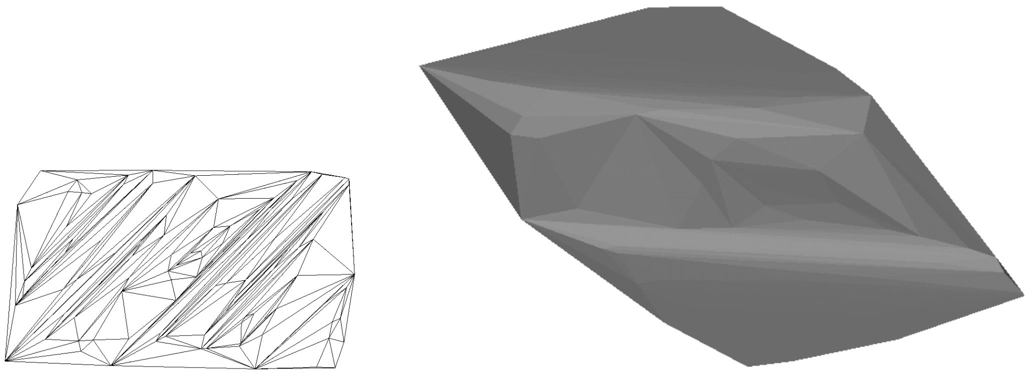


ABBILDUNG 1.4. TAC-Triangulierung der Klippenfunktion.

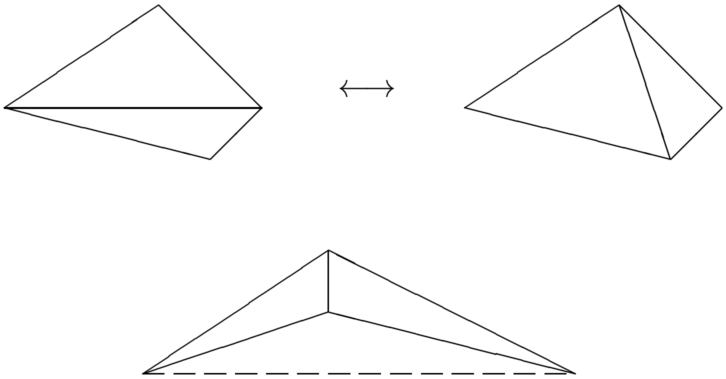


ABBILDUNG 1.5. Oben: Diagonalentausch einer inneren Triangulierungskante. Unten: Bei einem konkaven Viereck liegt die andere Diagonale (gestrichelt) außerhalb, ein Tausch ist nicht möglich.

Netzes [**CSYL88, DLR89, DLR90**], während das *TAC-Kriterium* die totale Absolutkrümmung des Netzes minimiert [**AvD95, vDA95**]. Im Unterschied zur Delaunaytriangulierung erzeugen ABN und TAC im Bereich der Klippe sehr lange, schmale Dreiecke (Abbildungen 1.3 und 1.4). Diese Dreiecke liegen quer zur maximalen Krümmung der Fläche, wodurch sie sich sehr eng an diese anschmiegen können. Das Resultat ist eine wesentlich bessere Approximation der abgetasteten Fläche.

Direkte Konstruktionsverfahren für optimale Triangulierungen sind für die Delaunaytriangulierung und für das sogenannte Min-Max-Angle-Kriterium [**ETW90**] bekannt. Bei datenabhängigen Zielfunktionen hingegen begnügt man sich in der Regel mit lokalen Optima, die man durch iterative Optimierungsverfahren berechnet. (Eine erschöpfende Suche verbietet sich aus Effizienzgründen, da die Anzahl möglicher Triangulierungen mit der Punktanzahl kombinatorisch anwächst.)

Grundoperation der Optimierungsverfahren ist der sogenannte *Diagonalentausch*. Er überführt eine planare Triangulierung T_1 in eine andere Triangulierung T_2 mit identischem Gebiet. Jede innere Kante k einer Triangulierung ist mit zwei Dreiecken inzident. Diese Dreiecke zusammen bilden ein Viereck, dessen eine Diagonale die Kante k ist. Ein Diagonalentausch ersetzt k durch die andere Diagonale (vgl. Abbildung 1.5). Der Tausch ist nur in streng konvexen Vierecken möglich, da sonst degenerierte oder einander überlappende Dreiecke entstehen. Zwei beliebige Triangulierungen T_1 und T_2 mit gleichem Gebiet lassen sich durch mehrfachen Diagonalentausch ineinander überführen [**HNU96**].

Eine Kante k wird als *suboptimal* bezeichnet, wenn ihr Tausch eine Verbesserung der Triangulierung im Hinblick auf die Zielfunktion bewirkt. Am häufigsten wird der Diagonalentausch im Rahmen der Optimierungsstrategie *lokale Suche* (siehe z. B. [**PS82**]) eingesetzt. Hier tauscht man solange wie möglich suboptimale Kanten. Komplexere Verfahren wie *Simulated Annealing* und *evolutionäre Algorithmen* führen zu besseren Ergebnissen, die jedoch mit einem höheren Zeitaufwand erkauft werden [**Sch93, PS97, WMAD98**].

1.2. Gegenstand der Arbeit

Gegenstand dieser Dissertation sind algorithmische Probleme, die im Zusammenhang mit triangulierungsbasierten Ansätzen zur Flächenrückführung, insbesondere für injektiv projizierbare Flächen, auftreten. Diese Probleme ergeben sich zum Teil aus Aufgaben, die in bekannten Ansätzen zur Flächenrückführung auftreten, zum Teil durch zusätzliche Anforderungen an die Flächenrückführungsaufgabe und schließlich auch durch einen neuen Ansatz für die Flächenrückführung im nicht projizierbaren Fall.

Kapitel 2 ist dem sogenannten *Oriented Walk* gewidmet. Dieser Algorithmus wird dazu verwendet, für einen beliebigen Punkt festzustellen, in welches Dreieck einer gegebenen Triangulierung er fällt. Diese Operation kann etwa von Interesse sein, wenn für eine als Dreiecksnetz rekonstruierte Fläche ein Resampling über einem quadratischen Raster durchgeführt wird. Das Prinzip des Oriented Walk besteht darin, sich von Dreieck zu Dreieck weiterzuhangeln, bis dasjenige Dreieck erreicht ist, in dem der gegebene Punkt liegt. Der Algorithmus läßt sich direkt auf allgemeine konvexzellige Diagramme und in beliebige Dimension verallgemeinern. Ein Problem dieses Algorithmus ist seine Terminierung. In dieser Arbeit wird an Beispielen gezeigt, daß die Terminierung des Oriented-Walk-Algorithmus nicht für alle Triangulierungen garantiert ist. Die garantierte Terminierung des Oriented-Walk-Algorithmus ist äquivalent zur garantierten Existenz einer Sichtbarkeitsordnung der Triangulierung bzw. des Diagramms. Für planare Delaunaytriangulierungen und für die große Klasse der strikt regulären Diagramme ist bekannt, daß stets Sichtbarkeitsordnungen existieren [**Ede89, Ede90, DFFN$^+$91**]. Aus dem Blickwinkel des Oriented Walk heraus ergeben sich für diese und für weitere Klassen von Diagrammen einfache und anschauliche Beweise.

Kapitel 3 befaßt sich mit *nicht-konvexen Triangulierungen* endlicher planarer Punktmengen. Im Unterschied zu den üblichen Triangulierungsverfahren, bei denen die Triangulierung die konvexe Hülle der gegebenen Punkte überdeckt, sollen hier Triangulierungen berechnet werden, die in diesem Sinne nicht konvex sind, also etwa auch Löcher enthalten können. Dies ist nötig, weil die abgetasteten Flächen selbst häufig Löcher enthalten. Die Rückführung solcher Flächen im injektiv projizierbaren Fall kann dadurch geschehen, daß die projizierten Abtastpunkte zunächst in der üblichen Weise konvex trianguliert werden. Aus dieser Triangulierung werden dann die notwendigen Löcher herausgeschnitten. Es wird ein Verfahren angegeben, bei dem die gewünschten Löcher durch geschlossene Polygonzüge festgelegt werden. Die Polygonzüge müssen nicht notwendigerweise aus Ecken und Kanten der gegebenen Triangulierung bestehen, wodurch die Spezifikation vereinfacht wird. Es wird eine Triangulierung der projizierten Abtastpunkte berechnet, die einen möglichst großen Teil des zulässigen Gebiets überdeckt.

Eine weitere Aufgabenstellung in Kapitel 3 besteht darin, ein nicht-konvexes Polygon zu triangulieren. Dieses Problem tritt auf, wenn man Punkte aus einer Triangulierung entfernt, etwa zur Ausdünnung der Datenmenge oder zur Eliminierung von Ausreißern. Durch Löschen eines Punktes und der mit ihm inzidenten Kanten und Dreiecke entsteht ein Loch, das es durch Einfügen neuer Kanten und Dreiecke zu schließen gilt. Aufgrund seiner Entstehung ist das Loch ein sternförmiges, aber im allgemeinen nicht konvexes Polygon. Es wird gezeigt, daß sternförmige und schwach kantensichtbare Polygone sich durch einen „umgekehrten Graham's Scan" in linearer Zeit triangulieren lassen. Bisher war dies nur für die Klasse der stark kantensichtbaren Polygone bekannt [**WS85**].

Inhalt von Kapitel 4 sind *konvexe Hüllen von Polygonen auf der Sphäre* und ihre effiziente Berechnung. In der Flächenrückführung tritt das Problem der Berechnung sphärischer konvexer Hüllen beispielsweise bei der Berechnung der totalen Absolutkrümmung eines Dreiecksnetzes im Raum auf. Wie in Abschnitt 1.1 gezeigt, ist die totale Absolutkrümmung Grundlage eines guten Maßes für die Qualität einer rekonstruierten Fläche. Es wird ein Algorithmus angegeben, der die konvexe Hülle eines sphärischen Polygons in linearer Zeit berechnet, wobei er sich unter anderem eine Reduktion auf den planaren Fall zunutze macht. Alboul und van Damme [**AvD95**] verwenden zur Berechnung der totalen Absolutkrümmung ein Verfahren mit $O(n^3)$ Zeitaufwand.

In Kapitel 5 wird die Stabilität von Delaunaykanten untersucht. Ausgangspunkt der Betrachtungen ist die Tatsache, daß eine Delaunaytriangulierung bereits durch ihre Ecken eindeutig bestimmt ist, von Spezialfällen abgesehen. Die Stabilität beschreibt nun, wie stark die Ecken verschoben werden dürfen, so daß ihre Delaunaytriangulierung aus kombinatorischer Sicht identisch mit der ursprünglichen Delaunaytriangulierung ist. Dies kann beispielsweise als Maß für den Grad der

Verrauschung der Daten gesehen werden, der erlaubt ist, ohne daß Auswirkungen auf die Triangulierung feststellbar sind. Neben der Triangulierung als Ganzes besitzt auch jede einzelne ihrer Kanten eine individuelle Stabilität. Für das Stabilitätsproblem einer Delaunaykante k werden ein Entscheidungs– und ein Optimierungsalgorithmus angegeben, die in Zeit $O(m)$ und $O(m \log m)$ laufen. Dabei ist m die Anzahl der zu k adjazenten Kanten.

Beim Experimentieren mit einer Implementierung der Stabilitätsalgorithmen hat es sich herausgestellt, daß Stabilität als Lösungsheuristik für das allgemeine Flächenrückführungsproblem eingesetzt werden kann. Bei hinreichend dichter Abtastung einer planaren Kurve sind diejenigen Delaunaykanten, die nahe an der Kurve verlaufen, häufig stabiler als Kanten, die von der Kurve weg verlaufen. Abbildung 6.1 (auf Seite 95) zeigt eine hierauf basierende Kurvenrekonstruktion. Diese Eigenschaft überträgt sich ins Dreidimensionale auf Dreiecke der Delaunaytetraedrisierung einer Menge von Abtastpunkten. Allerdings ist hier die „Trennschärfe" der Stabilität etwas geringer als im planaren Fall. In Kapitel 6 wird eine Modifikation des ursprünglichen Stabilitätsbegriffes entwickelt, die eine automatische Anpassung an die lokale Abtastdichte bewirkt.

Kapitel 7 behandelt die Aufzählung aller möglichen *polygonalen Hüllen* einer endlichen Punktmenge in der Ebene. Eine polygonale Hülle ist ein einfaches Polygon, das die gegebene Punktmenge enthält und dessen Eckpunkte zu der gegebenen Menge gehören. Es wird ein Verfahren vorgestellt, das alle polygonalen Hüllen von n gegebenen Punkten in $O(n^2)$ Zeit pro aufgezählter Hülle bei einem Speicherbedarf von $O(n^2)$ konstruiert. Das Verfahren beginnt mit der maximalen polygonalen Hülle, der konvexen Hülle der Punktmenge. Diese verkleinert es sukzessive, wobei es rekursiv verzweigt. Durch Merken gewisser Verzweigungsentscheidungen schließt das Verfahren aus, daß polygonale Hüllen mehrfach aufgezählt werden. Modifiziert man das Verfahren so, daß es unter den möglichen Verzweigungen genau eine zufällig auswählt, dann erzeugt es zufällige Polygone. Läßt man eine Heuristik Verzweigungen auswählen, dann kann man nach Polygonen mit bestimmten Eigenschaften suchen. Ein Beispiel hierfür sind geeignete Gebiete für nicht-konvexe Triangulierungen der gegebenen Punktmenge.

1.3. Definitionen und Notationen

In diesem Abschnitt werden einige grundlegende Begriffe und Notationen eingeführt. Weitere Definitionen finden sich in den jeweiligen Kapiteln, in denen sie gebraucht werden.

Die in dieser Arbeit betrachtete Geometrie bewegt sich in der Regel in Euklidischen Räumen, häufig der Dimension $d = 2$ oder $d = 3$. Mit diesen Räumen verbunden sind euklidische Metrik $\mathrm{dist}(\cdot,\cdot)$, Volumenmaß $\mathrm{Vol}(\cdot)$ und Winkelmaß. Punkte werden sowohl als Elemente des Raums aufgefaßt, als auch als einelementige Teilmengen (z. B. der Schnitt zweier Geraden). Da Punkte im Rechner als Koordinatentupel dargestellt werden, setzen wir in üblicher Weise den Euklidischen Raum der Dimension d mit dem reellen Vektorraum $\mathbb{R}^d$ gleich. Es steht uns jedoch von Fall zu Fall frei, ein kartesisches Koordinatensystem samt Ursprung unseren Bedürfnissen entsprechend zu wählen.

NOTATION 1.1. *Seien $M_1, M_2 \subseteq \mathbb{R}^d$. Dann bezeichnet $M_1 \vee M_2$ die* ***affine Hülle*** *und $M_1 \wedge M_2$ die* ***konvexe Hülle*** *von $M_1 \cup M_2$. Weiter werden durch*

$$\bigvee_{i=1}^{n} M_i \qquad \bigvee_{i\in I} M_i \qquad \bigvee M \qquad \bigwedge_{i=1}^{n} M_i \qquad \bigwedge_{i\in I} M_i \qquad \bigwedge M$$

die entsprechenden affinen bzw. konvexen Hüllen bezeichnet.

Für nicht kollineare Punkte p_1, p_2, p_3 ist also z. B. $p_1 \wedge p_2$ eine Strecke auf der Geraden $p_1 \vee p_2$ und $p_1 \wedge p_2 \wedge p_3$ ein Dreieck in der Ebene $p_1 \vee p_2 \vee p_3$.

NOTATION 1.2. *Zu $M \subseteq \mathbb{R}^d$ ist ∂M der* ***Rand****, $\overline{M}$ der* ***Abschluss*** *und $\mathrm{int} M$ das* ***Innere*** *von M.*

DEFINITION 1.3. *Sei M eine konvexe Teilmenge des $\mathbb{R}^d$. Wir definieren ihre* ***Dimension*** *$\dim(M)$ als die Dimension ihrer affinen Hülle $\bigvee M$. Für die leere Menge setzen wir $\dim(\emptyset) := -1$. Sei M die endliche Vereinigung konvexer Mengen $M_1, \ldots, M_n$, dann ist ihre Dimension*

$$\dim(M) := \max_{i=1}^{n} \dim(M_i)$$

DEFINITION 1.4. *Ein* ***konvexes Polyeder*** *im $\mathbb{R}^d$ ist der Durchschnitt endlich vieler abgeschlossener Halbräume. Ein* ***Polyeder*** *ist die Vereinigung endlich vieler konvexer Polyeder von gleicher Dimension. Ein* ***Polygon*** *ist ein planares Polyeder. Ein d-dimensionales Polyeder heißt* ***Simplex****, wenn es die konvexe Hülle von $d+1$ Punkten ist.*

Den Durchschnitt von null Halbräumen definieren wir als den gesamten Raum. Damit ist $\mathbb{R}^d$ selbst auch ein konvexes Polyeder. Simplizes in niedrigen Dimensionen sind Punkte ($d = 0$), Strecken ($d = 1$), Dreiecke ($d = 2$) und Tetraeder ($d = 3$).

DEFINITION 1.5. *Sei $M \subset \mathbb{R}^d$ und $p \in \partial M$. Sei H eine Hyperebene mit $p \in H$ und seien H^+ und H^- die beiden abgeschlossenen Halbräume, die H berandet. H heißt (globale)* ***Stützhyperebene*** *von M in p, wenn $M \subseteq H^+$ oder $M \subseteq H^-$ gilt. Existiert eine offene Umgebung U von p, so daß*

$$M \cap U \subseteq H^+ \quad \text{oder} \quad M \cap U \subseteq H^-$$

gilt, dann heißt H ***lokale Stützhyperebene*** *von M in p.*

Für $d = 3$ und $d = 2$ spricht man auch von **Stützebenen** und **Stützgeraden**. Mit Hilfe von Stützhyperebenen kann man die Oberfläche eines Polyeders strukturieren:

DEFINITION 1.6. *Sei Q ein konvexes Polyeder und H eine Stützhyperebene von Q. Dann ist $F := Q \cap H$ ein* ***Seitenpolyeder*** *von Q. Ein Seitenpolyeder heißt* ***Facette****, falls $\dim(F) = \dim(Q) - 1$,* ***Kante****, falls $\dim(F) = 1$, und* ***Ecke****, falls $\dim(F) = 0$ ist. Die Menge aller Ecken von Q bezeichnen wir mit $E(Q)$. Ist F eine Facette, und liegt Q im Halbraum H^+ (in H^-), dann bezeichnen wir H^+ (H^-) als* ***inneren Halbraum*** *und H^- (H^+) als* ***äußeren Halbraum*** *bezüglich F.*

Auch für nicht-konvexe Polyeder ist intuitiv klar, was unter inneren und äußeren Halbräumen, Facetten und Seitenpolyedern zu verstehen ist. Eine exakte Definition ist jedoch ziemlich umständlich, daher wollen wir darauf verzichten. Die Seitenpolyeder eines Simplex sind wiederum Simplizes. Sie werden daher auch als **Seitensimplizes** bezeichnet.

In bestimmten Kontexten will man abweichend von Definition 1.6 die Seitenpolyeder nicht allein aufgrund der Geometrie von Q definieren. So werden z. B. Polygone häufig als zyklische Folge ihrer Eckpunkte angegeben. Sind nun drei aufeinanderfolgende Punkte p_{i-1}, p_i und p_{i+1} der Folge kollinear, dann ist nach der Definition p_i keine Ecke des Polygons, und die gesamte Strecke $p_{i-1} \wedge p_{i+1}$ ist eine Kante. Im Gegensatz dazu sieht man in den meisten Fällen $p_{i-1} \wedge p_i$ und $p_i \wedge p_{i+1}$ als Kanten und p_i als **degenerierte Ecke** an. Auch in höheren Dimensionen kann man in ähnlicher Weise feiner unterteilte Seitenpolyeder definieren. Dabei gilt ein Seitenpolyeder eines Seitenpolyeders immer auch als Seitenpolyeder von Q selbst. Ein Seitenpolyeder F' wird als **degeneriert** bezeichnet, wenn kein Seitenpolyeder F gemäß Definition 1.6 existiert mit $F' \subseteq F$ und $\dim(F) = \dim(F')$. Ein Polyeder heißt **degeneriert**, wenn es degenerierte Seitenpolyeder hat.

DEFINITION 1.7. *Sei $M \subset \mathbb{R}^d$ gegeben. Zwei Punkte $p, q \in M$ heißen gegenseitig* ***sichtbar*** *in M, wenn die Strecke $p \wedge q$ in M liegt. Die Punkte heißen gegenseitig* ***strikt sichtbar*** *in M, wenn das Innere der Strecke $p \wedge q$ im Innern von M liegt.*

DEFINITION 1.8. *Ein Polyeder Q heißt **sternförmig**, wenn ein Punkt p existiert, so daß jeder Punkt $q \in Q$ von p aus in Q sichtbar ist. Die Menge aller Punkte p mit dieser Eigenschaft wird als **Kern** von Q bezeichnet.*

Der Kern eines Polyeders ist ein konvexes Polyeder. Man erhält ihn als Schnitt der abgeschlossenen inneren Halbräume bezüglich aller Facetten von Q (vgl. [**O'R87**]). Der Kern kann eine niedrigere Dimension haben als das Polyeder selbst. Nicht sternförmige Polyeder haben leere Kerne.

DEFINITION 1.9. *Sei $M \subseteq \mathbb{R}^d$ mit $\mathrm{Vol}(M) \neq 0$. Eine Menge von Teilmengen $Z_1, \ldots, Z_n \subseteq M$ heißt **Zerlegung** von M, wenn gilt*

1. $\displaystyle\bigcup_{i=1}^{n} Z_i = M$,
2. $Z_i \cap Z_j \subseteq \partial Z_i \cap \partial Z_j$ *für* $i \neq j$ *und*
3. $\mathrm{Vol}(Z_i) \neq 0$ *für alle* i.

*Die Z_i heißen **Zellen** der Zerlegung. Die Menge M ist das **Gebiet** der Zerlegung.*

DEFINITION 1.10. *Eine Zerlegung heißt **Zellkomplex**, wenn jede ihrer Zellen ein Polyeder ist und der Schnitt von je zwei Zellen entweder leer oder die Vereinigung gemeinsamer Seitenpolyeder ist. Ein Zellkomplex mit ausschließlich simplizialen Zellen heißt **Triangulierung**.*

In einem Komplex aus konvexen Zellen ist der Schnitt zweier Zellen entweder leer oder genau ein gemeinsames Seitenpolyeder. Zellkomplexe werden oft auch als **Diagramme** bezeichnet. Der Begriff Triangulierung rührt daher, daß im planaren Fall alle Zellen Dreiecke sind. Speziell in drei Dimensionen spricht man auch von **Tetraedrisierungen**.

Eine besondere Klasse von Triangulierungen geht auf den russischen Mathematiker Delone (in französischer Transskription Delaunay) zurück [**Del32, Del34**].

DEFINITION 1.11. *Sei eine endliche Punktmenge P mit $\bigvee P = \mathbb{R}^d$ gegeben. Eine **Delaunaykugel** bezüglich P ist eine Kugel K mit*

$$\operatorname{int} K \cap P = \emptyset \quad \textit{und} \quad \bigvee (\partial K \cap P) = \mathbb{R}^d \ .$$

*Ist K Delaunaykugel, dann heißt ∂K **Delaunaysphäre** und das konvexe Polyeder $\bigwedge(\partial K \cap P)$ **Delaunayzelle** bezüglich P. Der aus allen Delaunayzellen gebildete Zellkomplex ist das **Delaunaydiagramm** von P. Eine Triangulierung T ist eine **Delaunaytriangulierung** von P, wenn ihr Gebiet $\bigwedge P$ ist und jeder Simplex von T in einer Delaunayzelle enthalten ist. Ein Simplex von T heißt dann **Delaunaysimplex** bezüglich P.*

Eine Delaunaykugel ist durch die auf ihrem Rand liegenden Punkte von P eindeutig bestimmt, da diese Punkte nicht kohyperplanar sind. Liegen auf jeder Delaunaykugel genau $d+1$ Punkte von P, dann sind alle Delaunayzellen Simplizes, und das Delaunaydiagramm ist die eindeutige Delaunaytriangulierung von P. Hat das Delaunaydiagramm nicht-simpliziale Zellen, dann kann man diese Zellen auf verschiedene Arten in Simplizes zerlegen. Dabei erhält man verschiedene, und somit nicht-eindeutige, Delaunaytriangulierungen von P. Da die Ecken einer Delaunayzelle auf dem Rand der zugehörigen Delaunaykugel liegen, wird diese oft auch als **Umkugel** der Zelle bezeichnet.

DEFINITION 1.12. *Sei $P \subset \mathbb{R}^d$ endliche Punktmenge. Das **Voronoigebiet** von $p \in P$ bezüglich P ist die Menge aller Punkte, die zu keinem anderen Punkt von P näher liegen als zu p. Die Voronoigebiete bezüglich P bilden eine Zerlegung des $\mathbb{R}^d$, das **Voronoidiagramm** von P. Zwei Punkte $p_i, p_j \in P$ heißen **Voronoinachbarn**, wenn ihre Voronoigebiete sich schneiden. Haben die Voronoigebiete eine gemeinsame Facette, dann sind p_i und p_j **starke Voronoinachbarn**, ansonsten **schwache Voronoinachbarn**.*

Da das Voronoidiagramm ein Zellkomplex ist, bezeichnen wir die Voronoigebiete auch als **Voronoizellen**. Im Zusammenhang mit dem Voronoidiagramm werden die Punkte von P häufig als **Standorte** bezeichnet. Die Voronoizelle eines Standorts ist gewissermaßen sein Einzugsgebiet.

Ein Voronoidiagramm höherer Ordnung betrachtet gleichgroße Teilmengen einer Punktmenge P und weist ihnen Voronoigebiete zu.

DEFINITION 1.13. *Sei $P \subset \mathbb{R}^d$ endliche Punktmenge. Seien weiter $o \geq 1$ und $S \subset P$ mit $|S| = o$. Das **Voronoigebiet o-ter Ordnung** von S bezüglich P besteht aus denjenigen Punkten des Raumes, die die Punkte von S als o nächste Nachbarn haben, also*

$$\left\{ p \in \mathbb{R}^d : \max_{p_i \in S} \operatorname{dist}(p, p_i) \leq \min_{p_i \in P \setminus S} \operatorname{dist}(p, p_i) \right\} \quad .$$

*Die Voronoigebiete o-ter Ordnung bezüglich P bilden das **Voronoidiagramm o-ter Ordnung** von P.*

Ist die Ordnung $o = 1$, dann erhält man das herkömmliche Voronoidiagramm von M. Für $o = |M| - 1$ enthält $M \setminus S$ genau einen Punkt p_S. Liegt p im Voronoigebiet von S, dann ist

$$\operatorname{dist}(p, p_S) = \max_{p_i \in P} \operatorname{dist}(p, p_i) \ .$$

Daher wird dieses Diagramm auch **Farthest-Point-Voronoidiagramm** genannt.

KAPITEL 2

Korrektheit des Oriented-Walk-Algorithmus

Punktlokalisierung ist eine häufig benötigte Operation in Triangulierungen und anderen Zellkomplexen. Ein Anfragepunkt ist in Form seiner Koordinaten gegeben. Im folgenden soll q stets den Anfragepunkt bezeichnen. Lokalisierung bezeichnet die Bestimmung der Lage von q relativ zur Triangulierung T. Ihr Ergebnis ist entweder ein Dreieck von T, in dem q liegt, oder die Tatsache, daß q außerhalb der Triangulierung liegt.

Der Anfragepunkt q kann z. B. die Position des Mauszeigers sein, mit dem der Benutzer ein Dreieck der Triangulierung anwählen möchte. Das Programm erhält in diesem Fall die Koordinaten von q und muß das Dreieck bestimmen. In der Flächenrückführung wird Punktlokalisierung auch beim gerasterten Resampling von Dreiecksnetzen benötigt. Die Ausgangslage ist, daß man auf gestreuten Abtastpunkten durch Projektion und planare Triangulierung etwa in der xy-Ebene ein Dreiecksnetz konstruiert hat. Für ein gegebenes Punktraster in der xy-Ebene ist nun zu jedem Rasterpunkt der darüberliegende Punkt des Dreiecksnetzes gesucht. Im allgemeinen ist dieser Punkt keine Ecke des Netzes, sondern liegt auf einem der Dreiecke. Um das relevante Dreieck zu finden, lokalisiert man den Rasterpunkt in der planaren Triangulierung. Der gesuchte Punkt im Dreiecksnetz läßt sich dann einfach berechnen.

Dieses Kapitel behandelt die Korrektheit des Oriented Walk, eines weitverbreiteten Algorithmus zur Punktlokalisierung in Triangulierungen. Es zeigt sich, daß der Oriented-Walk-Algorithmus in planaren Delaunaytriangulierungen korrekt ist und insbesondere terminiert. Dasselbe gilt für Delaunaydiagramme und verallgemeinerte Voronoi– und Potenzdiagramme in d Dimensionen. In allgemeinen Triangulierungen hingegen, und damit auch in allgemeinen konvexzelligen Diagrammen, terminiert der Oriented Walk nicht notwendigerweise. Bei Dimensionen $d \geq 3$ können auch in Delaunaytriangulierungen Endlosschleifen auftreten. Der Korrektheitsbeweis für Delaunaydiagramme offenbart eine interessante Beziehung zwischen Delaunaydiagrammen und Potenzdiagrammen: Im Innern der konvexen Hülle $\bigwedge P$ ist das Delaunaydiagramm von P identisch mit dem Potenzdiagramm seiner eigenen Umkugeln.

Die garantierte Terminierung des Oriented Walk in einem Zellkomplex ist äquivalent zur Existenz einer Sichtbarkeitsordnung der Zellen. Sichtbarkeitsordnungen werden z. B. in Algorithmen zur Visualisierung von Volumendaten ausgenutzt. Unter diesem Aspekt untersuchen De Floriani et al. [**DFFN$^+$91**] planare Delaunaytriangulierungen und Edelsbrunner [**Ede89, Ede90**] die sehr allgemeine Klasse der *strikt regulären Diagramme* in beliebiger Dimension. Für beide Klassen werden in diesem Kapitel einfachere Beweise angegeben.

2.1. Der Oriented-Walk-Algorithmus

Ein einfaches iteratives Lokalisierungsverfahren in konvexen Triangulierungen ist Lawsons Oriented Walk [**Law77**]. Seine Grundidee besteht darin, sich von Dreieck zu Dreieck bis zum Anfragepunkt vorzuarbeiten. Der Algorithmus startet in einem beliebigen Dreieck der Triangulierung. Für die Kanten des Dreiecks wird durch je einen Halbebenentest bestimmt, ob q auf ihrer Außenseite liegt. Sei k eine Kante des Dreiecks, dann teilt die Gerade $\bigvee k$ die Ebene in zwei Halbebenen. Als Außenseite bezüglich k gilt die offene Halbebene, die zu dem Dreieck disjunkt ist. Ihr Komplement, also die abgeschlossene Halbebene, die das Dreieck enthält, wird als Innenseite angesehen (vgl. auch Definition 1.6). Liegt nun q auf der Außenseite, dann überschreitet der Oriented Walk die betreffende Kante und fährt im benachbarten Dreieck fort. Liegt q bezüglich aller Kanten auf der Innenseite, dann liegt q im aktuellen Dreieck. Überschreitet der Oriented Walk eine Randkante, dann liegt q außerhalb der Triangulierung.

Treten Anfragepunkte in zufälliger Reihenfolge an beliebigen Positionen auf, dann sind für die Lokalisierung solche Algorithmen effizienter, die auf hierarchischen Datenstrukturen beruhen [**GKS90, ES92, ES96, BT86**]. Beim gerasterten Resampling hingegen ist der Einsatz eines Oriented Walk auch aus Effizienzsicht sinnvoll. Wenn das Raster keine sehr grobe Auflösung besitzt (im Vergleich zu den gestreuten Abtastpunkten), dann liegen benachbarte Rasterpunkte in den meisten Fällen im selben oder in benachbarten Dreiecken der Triangulierung. Startet man jetzt den Oriented Walk im bereits bekannten Dreieck eines benachbarten Rasterpunktes, dann wird er den nächsten Rasterpunkt in der Regel mit wenigen Schritten lokalisieren.

Der Oriented Walk ist leicht in höhere Dimensionen zu übertragen. Aus dem Halbebenentest für Kanten wird ein Halbraumtest für Facetten. Der Algorithmus ist auch nicht auf Triangulierungen beschränkt. Vielmehr kann er in allen Zellkomplexen angewandt werden, die konvexe Zellen haben und ein konvexes

Gebiet überdecken. Der Halbraumtest für alle Facetten einer konvexen Zelle liefert entweder das Ergebnis „Punkt in der Zelle", oder er findet eine Facette, die zu überschreiten ist. Diese Form des Oriented Walk ist in Algorithmus 2.1 dargestellt.

Algorithmus 2.1 Oriented Walk

Gegeben: Ein Zellkomplex und ein Anfragepunkt q.
Gesucht: Eine Zelle, in der q liegt.

Z := beliebige Zelle des Komplexes.
for jede Facette F von Z **do**
    if Hyperebene $\bigvee F$ trennt Z und q (insbesondere $q \notin \bigvee F$) **then**
        if F ist Randfacette **then**
            q liegt außerhalb der Unterteilung.
            stop.
        else
            Z := Zelle auf der anderen Seite von F.
            goto 2.
        endif
    endif
endfor
q liegt in Z.

In der Literatur wird der Oriented Walk oft zur Punktlokalisierung in Delaunaytriangulierungen verwendet. Einige Autoren berichten dabei von Problemen durch Rundungsfehler, die zu falschen oder inkonsistenten Ergebnissen beim Halbebenen– bzw. Halbraumtest führen. Lawson schlägt in [**Law84**] vor, zum Triangulieren auf der Einheitskugel sämtliche Koordinaten auf eine bestimmte Anzahl signifikanter Stellen zu runden. Danach kann der Halbkugeltest auch in Gleitkomma-Arithmetik exakt ausgewertet werden. Palacios-Velez und Cuevas Renaud [**PVCR90**] beobachten, daß der Oriented Walk zwischen zwei benachbarten Dreiecken hin– und herspringt, und dadurch eine Endlosschleife erzeugt. Wir wollen jedoch in diesem Kapitel nicht auf numerische Fehlerursachen eingehen. Vielmehr wollen wir den Oriented Walk unter der Annahme betrachten, daß alle Berechnungen exakt ausgeführt werden.

Die partielle Korrektheit des Oriented-Walk-Algorithmus ist leicht einzusehen.

SATZ 2.1. *Wenn ein Oriented Walk in einem konvexzelligen Diagramm mit konvexem Gebiet terminiert, dann hat er den Anfragepunkt q korrekt lokalisiert.*

Beweis: Überschreitet der Oriented Walk eine Randfacette F des Diagramms, dann werden q und das Diagramm durch die Hyperebene $\bigvee F$ getrennt. Das Ergebnis „außerhalb" ist daher korrekt. Endet der Walk in einer Zelle Z des Diagramms, dann liegt q bezüglich jeder Facette von Z auf der Innenseite, also im selben abgeschlossenen Halbraum wie Z. Da die Zelle konvex ist, folgt $q \in Z$. ∎

In einem Diagramm mit nicht-konvexen Zellen würde mit Sicherheit niemand auf die Idee kommen, einen Oriented Walk anzuwenden. Aus der Tatsache, daß q bezüglich einer Facette auf der Außenseite liegt, folgt hier nämlich nicht, daß q nicht in der Zelle liegen kann. Dennoch wollen wir dieses Szenario kurz beleuchten. Satz 2.1 bleibt auch bei nicht-konvexen Zellen gültig, sofern das Gebiet des Diagramms konvex ist. Wenn der Oriented Walk in einer Zelle Z terminiert, dann liegt q im Durchschnitt der Innenseiten bezüglich aller Facetten von Z. Dieser Durchschnitt ist genau der Kern der Zelle, und damit in ihr enthalten. Insbesondere muß Z sternförmig sein. Im Umkehrschluß folgt, daß der Walk nicht terminieren kann, wenn q zwar im Gebiet des Diagramms, aber nicht im Kern einer Zelle liegt. Man könnte also den Oriented Walk durch eine geeignete Anfrage leicht in eine Endlosschleife zwingen.

Das weiter unten folgende Beispiel 2.2 zeigt, daß der Oriented Walk schon in planaren Triangulierungen in Endlosschleifen geraten kann. Wir machen in dem Beispiel eine Annahme über ein Implementierungsdetail, nämlich die Reihenfolge, in der die **for**-Schleife (Zeilen 2–12 in Algorithmus 2.1) die Kanten des aktuellen Dreiecks durchläuft. Hat der Oriented Walk eine Kante k gerade überschritten, dann braucht er den Halbebenentest für k nicht mehr durchzuführen. Es bleiben also noch die beiden anderen Kanten des neuen Dreiecks zu testen. (Eine derartige Modifikation schließt nebenbei auch die in [**PVCR90**] beschriebenen Zyklen aus zwei Dreiecken aus.) Für das Beispiel wollen wir annehmen, daß zuerst diejenige Kante getestet wird, die in mathematischem Umlaufsinn auf k folgt. (D. h. die Kante, die man nach Überschreiten von k rechts vor bzw. neben sich sieht.) Diese Annahme ist durchaus realistisch, wenn z. B. Triangulierungen in einer Winged-Edge-Datenstruktur dargestellt werden. Die Winged-Edge-Datenstruktur (siehe z. B. Abramowski und Müller [**AM91**]) dient zur Darstellung planarer Diagramme. In ihr wird jede Kante durch einen Datensatz repräsentiert, der unter anderem Verweise auf die jeweils nächste Kante der beiden inzidenten Zellen enthält. Die **for**-Schleife umläuft sinnvollerweise die aktuelle Zelle mit Hilfe der Verweise von Kante zu Kante.

BEISPIEL 2.2. Sei ein Oriented Walk für planare Triangulierungen derart implementiert, daß er in einem neuen Dreieck zuerst die rechts liegende Kante testet.

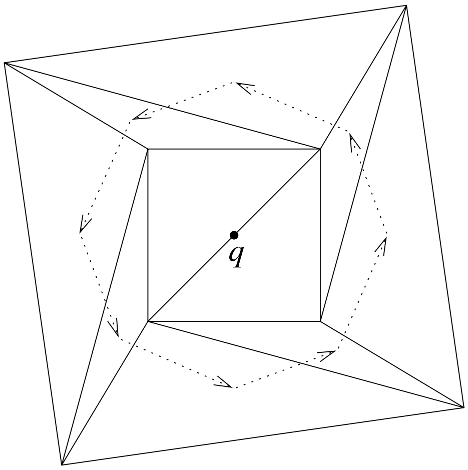


ABBILDUNG 2.1. Endlosschleife eines Oriented Walk.

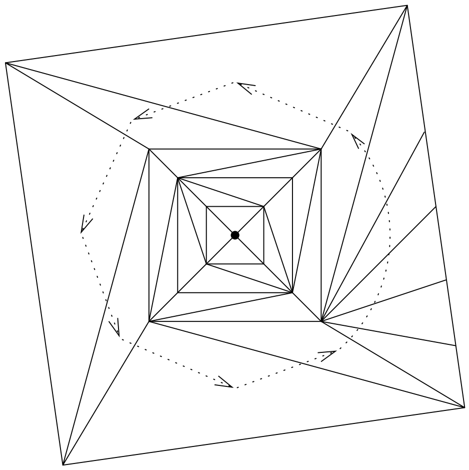


ABBILDUNG 2.2. Verlängerung des Zyklus und Vergrößerung seines Abstands zum Anfragepunkt.

Abbildung 2.1 zeigt diesen Oriented Walk in einer Endlosschleife. Der Zyklus hat eine Länge von 8 Dreiecken. Durch geeignetes Unterteilen dieser Dreiecke läßt sich der Zyklus beliebig verlängern. Nehmen wir z. B. das rechts am Rand gelegene Dreieck. In diesem Dreieck zeichnen wir Strecken von der links gelegenen Ecke zu je einem Punkt der rechts gelegenen Kante. Das Dreieck wird hierdurch unterteilt, und der Oriented Walk durchläuft die neuen Dreiecke, siehe Abbildung 2.2. Weiterhin können wir innerhalb des inneren Quadrats beliebig viele weitere Punkte in die Triangulierung einfügen, und dabei den Abstand von q zu dem Zyklus (gemessen in der Anzahl Kanten/Dreiecke, die q von dem Zyklus trennen) beliebig erhöhen. Ein Oriented Walk kann also Zyklen aus beliebig vielen Dreiecken durchlaufen, und diese Zyklen können beliebig viele Dreiecke umschließen und beliebig großen Abstand vom Anfragepunkt haben. ■

Triangulierung und Anfragepunkt in Abbildung 2.1 führen nicht zwangsläufig zu einer Endlosschleife. Im dritten, fünften, siebten usw. Dreieck könnte der Oriented Walk ebenso gut die linke Kante überschreiten, wenn er sie vor der rechten testen würde. Die Triangulierung bietet hier also nur die Möglichkeit zu einer Endlosschleife. Ob diese Möglichkeit „genutzt” wird, hängt von der Implementierung und ggf. auch von der Reihenfolge der Eingabe ab.

Die Möglichkeit einer Endlosschleife in einem bestimmten Diagramm ist eng mit einem Problem aus der Computergraphik verwandt, den sogenannten *Sichtbarkeitsordnungen*. Stellen wir uns einen Beobachter vor, der sich im Punkt q befindet. In der Regel sind nur wenige Facetten des Diagramms mit q ko-hyperplanar. Die übrigen haben von q aus gesehen eine Vorder– und Rückseite. Die inzidente Zelle Z' auf der Vorderseite verdeckt für q die inzidente Zelle Z auf der Rückseite der Facette. (Hierbei zählt bereits eine teilweise Verdeckung.) Für einen Oriented Walk bedeutet diese Lage der Zellen, daß er auf der Suche nach q von Z nach Z' gehen darf. Wir können die Verdeckungen durch einen gerichteten Graph darstellen, den Verdeckungsgraph. Der Verdeckunsgraph enthält eine gerichtete Kante von jeder Zelle Z zu jeder adjazenten Zelle Z', die Z (teilweise) verdeckt. Ein Oriented Walk entspricht nun genau einem gerichteten Pfad im Verdeckungsgraph. Endlosschleifen sind genau dann möglich, wenn der Verdeckungsgraph Zyklen enthält. Williams [**Wil92**] erfindet auf diesem Weg den Oriented-Walk-Algorithmus von neuem, indem er ihn direkt auf dem Verdeckungsgraph ausführt.

Ist der Verdeckungsgraph azyklisch, dann stellt seine transitive Hülle eine Halbordnung der Zellen dar. Die Halbordnung kann zu einer totalen Ordnung erweitert werden, die dann Sichtbarkeitsordnung heißt. Die Sichtbarkeitsordnung gibt eine Reihenfolge der Zellen derart an, daß keine Zelle eine später kommende Zelle verdeckt. Max, Hanrahan und Crawfis [**MHC90, Max93**] und Williams [**Wil92**] berechnen eine Sichtbarkeitsordnung aus dem Verdeckungsgraph. Sie verwenden die Ordnung zur Visualisierung von Volumendaten, die in einem Zellkomplex vorliegen. Eine andere Anwendung von Sichtbarkeitsordnungen ist der bekannte Painter's Algorithm (siehe z. B. [**Rog85**, S. 272]).

Edelsbrunner [**Ede89, Ede90**] zeigt, daß in *strikt regulären Diagrammen* (vgl. Definition 2.10 auf Seite 30) der Verdeckungsgraph stets azyklisch ist. Damit beweist er indirekt auch die totale Korrektheit des Oriented-Walk-Algorithmus in strikt regulären Diagrammen. In Satz 2.11 wird ein einfacherer Beweis für strikt reguläre Diagramme angegeben. Während Delaunaydiagramme strikt regulär sind, sind Delaunaytriangulierungen im allgemeinen nur regulär. In Beispiel 2.9 werden wir sehen, daß in höheren Dimensionen die einfache Regularität nicht ausreicht, um Zyklen auszuschließen. De Floriani et al. [**DFFN+91**] beweisen die Azyklizität in planaren Delaunaytriangulierungen. Satz 2.8 gibt hierfür einen einfachen und anschaulichen Beweis, der allerdings die Azyklizität von Delaunaydiagrammen voraussetzt.

Ein stets korrekter Algorithmus zur Punktlokalisierung in konvexzelligen Diagrammen mit konvexem Gebiet ist die Strahlverfolgung. Sie bewegt sich von einer Startecke p^* aus geradlinig auf den gesuchten Punkt q zu, folgt also dem

Algorithmus 2.2 Strahlverfolgung

```
Eingabe: Triangulierung T, Anfragepunkt q ∈ ℝ²
 1: Wähle eine Ecke p* der Triangulierung.
 2: Bestimme den Strahl s von p* durch q.
 3: δ := das mit p* inzidente Dreieck, durch das s läuft.
 4: repeat
 5:    Bestimme die Kante k von δ, durch die s das Dreieck verläßt.
 6:    p := s ∩ k.
 7:    if q ∈ p* ∧ p (∗ q liegt vor der Kante k ∗) then
 8:       q liegt in δ.
 9:    elseif k ist Randkante then
10:       q liegt außerhalb der Triangulierung.
11:    else
12:       δ := das Dreieck, in das s über k gelangt.
13:    endif
14: until q ist lokalisiert
```

Strahl s, der von p^* aus durch q geht. Die signifikanten Ereignisse der Strahlverfolgung sind die Übergänge des Strahls von einer Zelle zur nächsten. Der Übergang erfolgt im Schnittpunkt von s mit einem gemeinsamen Seitenpolyeder der beiden Zellen.

Cline und Renka [**CR84**] geben eine Strahlverfolgung für planare Triangulierungen in Pseudocode-Notation an, ohne jedoch die geometrische Bedeutung zu erläutern. Algorithmus 2.2 beschreibt diese Strahlverfolgung in geometrischen Begriffen. Er macht die vereinfachende Annahme, daß außer p^* keine Ecke der Triangulierung auf s liegt. Der Übergang in ein neues Dreieck erfolgt dann immer durch das Innere einer Kante. Ein einfaches Störungsmodell[1] macht diese Annahme gültig: Liegt eine Ecke p_i auf s, dann verhält sich der Algorithmus so, als läge p_i „knapp links" von s. Konkret führt dies dazu, daß in Schritt 5 als Austrittskante eine Kante von δ gewählt wird, die mit p_i inzident ist. Ist $p_i \in s$ eine Randecke der Triangulierung, dann wird sie so behandelt, als läge sie auf der „Außenseite" von s. Dadurch wird verhindert, daß die Strahlverfolgung vorzeitig abbricht, wenn der Suchstrahl auf mehreren kollinearen Randkanten entlangläuft. Cline und Renka behandeln den Fall $p_i \in s$ explizit, indem sie eine neue Strahlverfolgung mit p_i als Startpunkt durchführen. In höheren Dimensionen führen sowohl

[1]Zur Umgehung geometrischer Spezialfälle durch Störungsmodelle siehe etwa [**EM90, Sei98**].

die explizite Behandlung von Spezialfällen als auch der Störungsansatz zu erheblich komplizierteren Algorithmen. Dies ist der Grund, warum Mücke, Saias und Zhu [**MSZ96**] einen Oriented Walk für Delaunay-Tetraedrisierungen verwenden.

SATZ 2.3. *Algorithmus 2.2 ist korrekt.*

Beweis: Es ist leicht zu sehen, daß Algorithmus 2.2 tatsächlich den Strahl von p^* durch q verfolgt, und daß er q korrekt lokalisiert, sofern er terminiert.

Der Algorithmus führt in jedem Schleifendurchlauf Schritt 6 aus. Betrachten wir die Schnittpunkte $p = s \cap k$, die in diesem Schritt berechnet werden. Die Entfernung dieser Schnittpunkte von der in Schritt 1 gewählten Startecke wächst streng monoton. Der Strahl s besitzt nur endlich viele Schnitte mit den Kanten und Ecken der Triangulierung. Daher muß die Strahlverfolgung terminieren. Das Argument streng monoton wachsender Entfernungen können wir auch auf die Behandlung des Spezialfalles $p_i \in s$ durch das erwähnte Störungsmodell ausweiten. Statt Schnittpunkten mit s betrachten wir Schnittpunkte mit einem gestörten, d. h. genügend wenig nach rechts verschobenen Strahl. (Am Rand ist der Strahl eventuell nach links verschoben.) Der gestörte Strahl schneidet dieselben Kanten wie s in derselben Reihenfolge, aber mit streng monoton steigender Entfernung von seinem Ausgangspunkt. ■

2.2. Voronoi– und Potenzdiagramme

In diesem Abschnitt befassen wir uns mit einigen Klassen von Diagrammen, deren Definition auf Abstandsfunktionen beruht. Die erste dieser Klassen ist das Voronoidiagramm in beliebiger Dimension.

SATZ 2.4. *Der Oriented-Walk-Algorithmus terminiert in Voronoidiagrammen beliebiger Dimension.*

Beweis: Überschreite der Oriented Walk eine Facette F, etwa aus dem Voronoigebiet von p_i in das Voronoigebiet von p_j. Dann liegt q strikt auf derselben Seite der Hyperebene $\bigvee F$ wie p_j. Da $\bigvee F$ die Mittelsenkrechte von p_i und p_j ist, gilt $\operatorname{dist}(q, p_i) > \operatorname{dist}\big(q, p_j\big)$. Beim Übergang in eine neue Zelle verringert sich also der Abstand von q zum erzeugenden Punkt (Standort) der Zelle. Dadurch ist ein Zyklus ausgeschlossen. ■

Im Prinzip läßt sich das Beweisargument von Satz 2.4 auf Verallgemeinerungen des Voronoidiagramms übertragen. Viele dieser Verallgemeinerungen vereiteln jedoch den Oriented-Walk-Algorithmus allein aus dem Grund, daß ihre Zellen

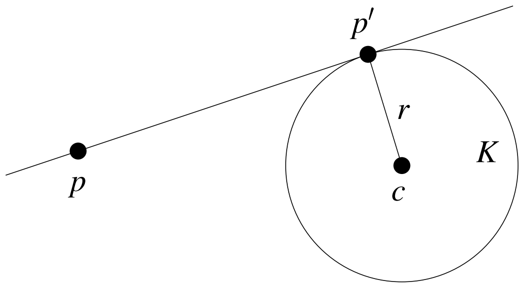


ABBILDUNG 2.3. Die Potenz von p bezüglich K ist das Quadrat der tangentialen Strecke $p \wedge p'$.

nicht notwendigerweise polyedrisch sind. Zwei Verallgemeinerungen mit polyedrischen Zellen sind Voronoidiagramme höherer Ordnung und Potenzdiagramme.

Beim Voronoidiagramm o-ter Ordnung sind die „Standorte" Teilmengen $S \subseteq P$ mit $|S| = o$. Das Diagramm basiert auf der Abstandsfunktion

$$\mathrm{dist}^{(o)}(p,S) := \max_{p_i \in S} \mathrm{dist}(p,p_i) \quad .$$

Seien $S, S' \subset P$ derart, daß ihre Zellen Z, Z' eine gemeinsame Facette F haben. Die Hyperebene $\bigvee F$ ist dann genau die äquidistante Punktmenge

$$\{p \in \mathbb{R}^d \mid \mathrm{dist}^{(o)}(p,S) = \mathrm{dist}^{(o)}\big(p,S'\big)\} \quad .$$

Durchschreitet ein Oriented Walk F von Z nach Z', dann gilt $\mathrm{dist}^{(o)}(p,S') < \mathrm{dist}^{(o)}(p,S)$. Der Abstand zum „Standort" der aktuellen Zelle wird also in jedem Schritt verringert.

Potenzdiagramme sind ähnlich wie Voronoidiagramme definiert, jedoch sind hier die Standorte Kugeln. Als Abstandsfunktion dient die Potenz eines Punktes bezüglich einer Kugel, auch unter dem Namen Laguerre-Metrik bekannt. Sei p ein Punkt und K eine Kugel mit Mittelpunkt c und Radius r, dann ist

$$\mathrm{pot}(p,K) := \mathrm{dist}(p,c)^2 - r^2 \quad .$$

Diese Funktion ist außerhalb der Kugel positiv, auf dem Rand Null und im Inneren negativ. Außerhalb von K mißt sie den quadrierten tangentialen Abstand zu K, siehe Abbildung 2.3. Die Zellen des Potenzdiagramms werden Potenzgebiete genannt. Eine ausführliche Diskussion von Potenzdiagrammen findet sich in [**Aur87**]. Imai, Iri und Murota [**IIM85**] behandeln den planaren Fall.

Die äquipotenten Punkte zweier Kugeln K und K' bilden eine Hyperebene, die sogenannte Chordale von K und K'. Falls die Ränder der Kugeln sich schneiden, dann enthält die Chordale diesen Schnitt, sie ist also durch $\bigvee(\partial K \cap \partial K')$ gegeben.

Haben die Potenzgebiete von K und K' eine gemeinsame Facette F, dann liegt F auf der Chordalen. Wenn nun ein Oriented Walk F von K zu K' hin durchschreitet, dann ist $\mathrm{pot}(q,K') < \mathrm{pot}(q,K)$.

Haben alle Kugeln den gleichen Radius, dann ist die Potenz als Abstandsfunktion äquivalent zum euklidischen Abstand vom Mittelpunkt einer Kugel. Insbesondere ist die Chordale zweier Kugeln die Mittelsenkrechte ihrer Mittelpunkte. Das Potenzdiagramm ist in diesem Fall das Voronoidiagramm der Kugelmittelpunkte.

Analog zu Voronoidiagrammen definiert man auch **Potenzdiagramme höherer Ordnung**. Hierbei sind die „Standorte" Teilmengen einer Menge von Kugeln. Beim korrekten Überschreiten einer Facette gilt wieder, daß die Abstandsfunktion streng monoton fällt.

SATZ 2.5. *In Voronoi– und Potenzdiagrammen beliebiger Ordnung terminiert der Oriented-Walk-Algorithmus.*
Der Beweis ergibt sich aus den vorangegangenen Ausführungen. ∎

2.3. Delaunaydiagramme sind im wesentlichen Potenzdiagramme

Mit den Potenzdiagrammen wollen wir uns noch etwas weiter befassen. Es zeigt sich, daß man Delaunaydiagramme (vgl. Definition 1.11) in einem gewissen Sinn als Potenzdiagramme ansehen kann. Wir betrachten das Delaunaydiagramm einer Punktmenge P. Sei F die gemeinsame Facette zweier Delaunayzellen Z und Z'. Die Ecken von F liegen auf den Delaunaysphären ∂K und $\partial K'$ der beiden Zellen. Da F die konvexe Hülle ihrer eigenen Ecken ist, muß sie auf der chordalen Hyperebene $\bigvee(\partial K \cap \partial K')$ der Delaunaysphären liegen. Die Vermutung liegt nahe, daß F auch eine Facette im Potenzdiagramm aller Delaunaysphären von P ist. Für Randfacetten des Delaunaydiagramms gilt diese Überlegung nicht, da sie jeweils zu genau einer Delaunayzelle gehören. Die Randfacetten überdecken den Rand der konvexen Hülle $\bigwedge P$. Im Gegensatz dazu stellt das Potenzdiagramm eine Unterteilung des ganzen Raumes dar. Seine „Randgebiete" sind unbeschränkt, und im allgemeinen besitzt es unbeschränkte Facetten, die über die konvexe Hülle $\bigwedge P$ hinausragen. Auf dem Rand und außerhalb der konvexen Hülle sind die beiden Diagramme also voneinander verschieden.

SATZ 2.6. *Sei $P \subset \mathbb{R}^d$ endlich. Im Inneren der konvexen Hülle $\bigwedge P$ ist das Delaunaydiagramm von P identisch mit dem Potenzdiagramm seiner eigenen Umkugeln.*

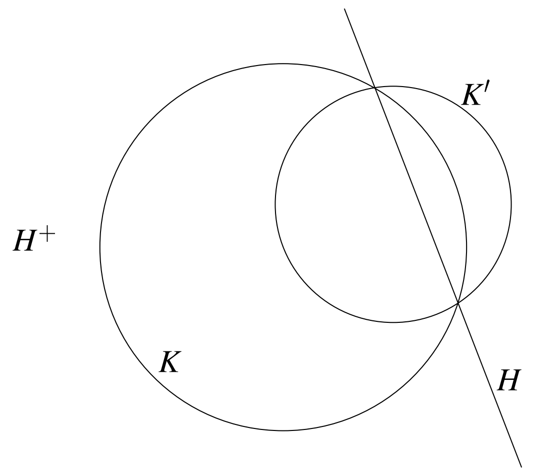


ABBILDUNG 2.4. Auf einer Seite der chordalen Hyperebene H umfaßt K K', auf der anderen Seite ist es umgekehrt.

Beweis: Sei Z eine Delaunayzelle mit Umkugel K, und sei K' Umkugel einer anderen Delaunayzelle. (Wenn es nur eine Delaunayzelle gibt, dann ist sie gleich $\bigwedge P$, und das Potenzgebiet der einzigen Umkugel ist der gesamte Raum.)

Ist $K \cap K'$ leer oder nur ein einziger Punkt, dann ist offensichtlich $\operatorname{pot}(p,K) \leq \operatorname{pot}(p,K')$ für alle $p \in Z \subset K$.

Ist $|K \cap K'| > 1$, dann ist $H := \bigvee(\partial K \cap \partial K')$ die chordale Hyperebene der beiden Umkugeln. Sei H^+ derjenige von H berandete, abgeschlossene Halbraum, für den $K \cap H^+ \supset K' \cap H^+$ gilt, vergleiche Abbildung 2.4. Für $p \in K \setminus K'$ ist $\operatorname{pot}(p,K) \leq 0 < \operatorname{pot}(p,K')$. Wegen $H^+ \cap K \setminus K' \neq \emptyset$ muß nun $\operatorname{pot}(p,K) \leq \operatorname{pot}(p,K')$ für alle $p \in H^+$ gelten. Betrachten wir die Eckenmenge $E(Z) \subset P$ der Zelle Z. Da K' eine Delaunaykugel ist, muß $E(Z) \cap \operatorname{int} K'$ leer sein. Andererseits ist $E(Z) \subset K$ und $K \setminus H^+ \subset \operatorname{int} K'$. Es folgt $E(Z) \subset H^+$, und damit $Z = \bigwedge E(Z) \subset H^+$.

Bezüglich jeder Delaunaykugel $K' \neq K$ gilt somit $\operatorname{pot}(p,K) \leq \operatorname{pot}(p,K')$ für alle $p \in Z$. Daher ist Z Teilmenge des Potenzgebiets von K.

Die Delaunayzellen sind also jeweils im Potenzgebiet ihrer eigenen Umkugel enthalten. Der umgekehrte Einschluß, sofern man die Betrachtung auf die konvexe Hülle $\bigwedge P$ einschränkt, folgt daraus, daß einerseits die Delaunayzellen $\bigwedge P$ überdecken und andererseits die Potenzgebiete der Umkugeln bis auf ihre Ränder paarweise disjunkt sind. ∎

Eine innere Facette F des Delaunaydiagramms ist Teilmenge einer Facette F' des Potenzdiagramms. Schneidet F den Rand der konvexen Hülle nicht, dann ist

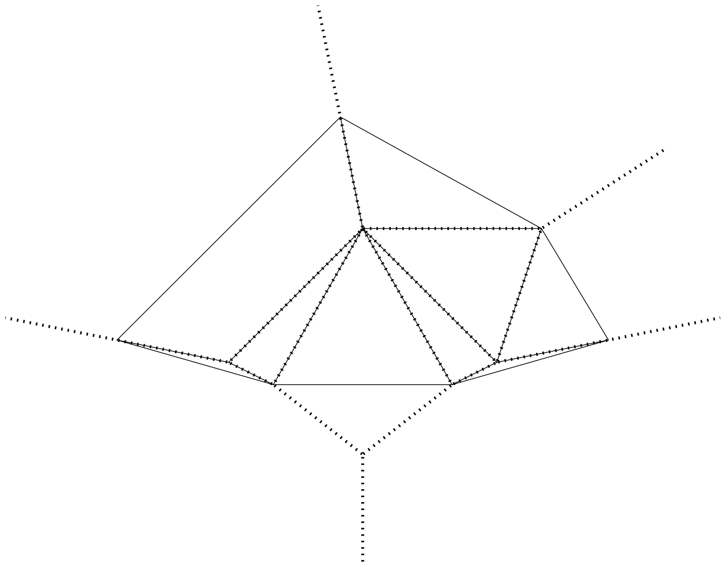


ABBILDUNG 2.5. Ein Delaunaydiagramm (durchgezogene Linien) und das Potenzdiagramm der Delaunaykreise (gepunktete Linien).

$F = F'$. Randfacetten des Delaunaydiagramms haben im Potenzdiagramm keine Entsprechung. Andernfalls gäbe es auf beiden Seiten von $\bigvee F$ Potenzgebiete mit darin enthaltenen Delaunayzellen. Dies stände im Widerspruch dazu, daß $\bigvee F$ Stützhyperebene von $\bigwedge M$ ist. Eine innere Zelle des Delaunaydiagramms ist mit dem korrespondierenden Potenzgebiet identisch, eine Randzelle ist echte Teilmenge des umgebenden Potenzgebiets. Wie man in Abbildung 2.5 sieht, kann dieses Potenzgebiet beschränkt oder unbeschränkt sein.

KOROLLAR 2.7. *In Delaunaydiagrammen terminiert der Oriented-Walk-Algorithmus.*

Beweis: Solange der Oriented Walk nur innere Facetten des Delaunaydiagramms überschreitet, verhält er sich exakt wie im Potenzdiagramm der Delaunaysphären. Nach Satz 2.5 kann hier keine Endlosschleife auftreten. Überschreitet der Walk eine Randfacette des Delaunaydiagramms, dann ist er beendet. ∎

Eine Delaunaytriangulierung geht aus einem Delaunaydiagramm hervor, indem man alle Delaunayzellen auf beliebige Weise in Simplizes unterteilt. Dabei entstehen neue Facetten innerhalb derjenigen Delaunayzellen, die nicht selbst schon Simplizes sind. Die Umkugel eines Delaunaysimplex ist identisch mit der Umkugel der Delaunayzelle, in der der Simplex liegt. Überschreitet ein Oriented

Walk eine Facette innerhalb einer Delaunayzelle, dann bleibt die aktuelle Umkugel gleich, und mit ihr die Potenz von q bezüglich der aktuellen Umkugel. Daher kann der Beweis von Satz 2.7 nicht auf Delaunaytriangulierungen übertragen werden. Es gilt jedoch weiterhin, daß sich die Potenz strikt verringert, wenn der Oriented Walk in eine andere Delaunayzelle wechselt. Ein Zyklus von Delaunaysimplizes muß also innerhalb einer einzigen Delaunayzelle liegen.

Im planaren Fall ist die Delaunayzelle Z ein Polygon, die Facetten innerhalb von Z sind Diagonalen von Z. Anders als im allgemeinen höherdimensionalen Fall wird Z von einer diagonalen Facette F in zwei Zusammenhangskomponenten unterteilt. Um innerhalb eines Delaunaypolygons einen Zyklus zu erzeugen, müßte der Oriented Walk mindestens eine der Diagonalen in beide Richtungen überschreiten (zu verschiedenen Zeitpunkten). Dies steht im Widerspruch zum Halbebenentest.

SATZ 2.8. *Der Oriented-Walk-Algorithmus terminiert in planaren Delaunaytriangulierungen.*

In Dimensionen $d \geq 3$ lassen sich Zyklen in Delaunaytriangulierungen konstruieren. Eine Möglichkeit besteht darin, einen $(d-1)$-dimensionalen Zyklus gewissermaßen in eine d-dimensionale Triangulierung einzubetten.

BEISPIEL 2.9. Sei H eine Hyperebene im $\mathbb{R}^d$. Sei T' eine $(d-1)$-dimensionale Triangulierung in H derart, daß ein Oriented Walk in T' mit Anfragepunkt $q \in H$ einen Zyklus durchlaufen kann. Dabei muß T' keine Delaunaytriangulierung sein. Wir legen eine Kugel $K \subset \mathbb{R}^d$ mit Mittelpunkt in H so, daß T' im Inneren von K liegt. Die Ecken unserer d-dimensionalen Triangulierung T werden auf der Sphäre ∂K liegen. Als erste Ecke wählen wir einen Punkt $z \in \partial K \setminus H$, siehe Abbildung 2.6. Seien nun $p'_1, \dots, p'_n$ die Ecken von T'. Durch Zentralprojektion mit Zentrum z projizieren wir die p'_i auf die Sphäre: Für $i = 1, \dots n$ hat die Gerade $z \vee p'_i$ zwei Schnittpunkte mit ∂K, wovon einer z selbst ist. Bezeichne p_i den Schnittpunkt ungleich z. Die Ecken von T sind $p_1, \dots, p_n$ und z. Die Hyperebene H trennt z von den übrigen Ecken. Für jeden Simplex $S' = \bigwedge_{i \in I} p'_i$ von T' ist $S := z \vee \bigwedge_{i \in I} p_i$ ein Simplex von T. Aufgrund der kosphärischen Lage aller Ecken von T ist S ein Delaunaysimplex, d. h. T ist (Teil einer) Delaunaytriangulierung. Nach Konstruktion ist T' quasi der Schnitt von T mit H. Zu einer Facette F von T ist $F' := F \cap H$ die analoge Facette von T'. Wegen $q \in H$ liefern der d-dimensionale Halbraumtest bezüglich F und der $(d-1)$-dimensionale Halbraumtest innerhalb H bezüglich F' analoge Ergebnisse. Der Zyklus des Oriented Walk in T' ist also auch ein Zyklus in T. Eine Delaunaytetraedrisierung

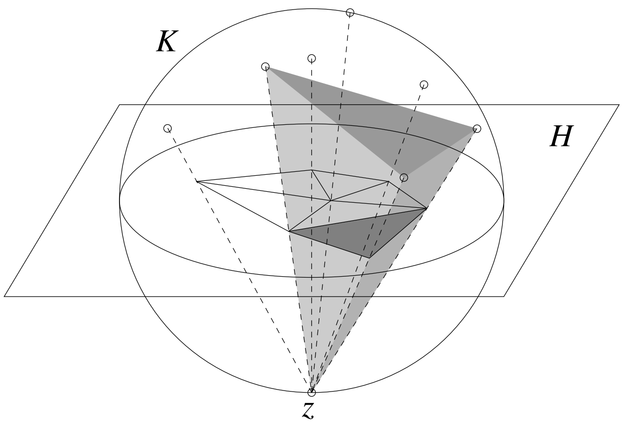


ABBILDUNG 2.6. Einbettung einer $(d-1)$-dimensionalen Triangulierung in eine d-dimensionale Delaunaytriangulierung.

mit Zyklus erhalten wir z. B. aus der Triangulierung in Beispiel 2.2. Für höhere Dimensionen können wir Zyklen durch Induktion über d erzeugen.

2.4. Reguläre Diagramme

Ein weiterer Beweis für Korollar 2.7 erschließt sich über reguläre Diagramme. Ein Diagramm heißt regulär, wenn man es so „anheben" kann, daß es die Unterseite eines konvexen Polyeders bildet. In der Literatur wird dieser Begriff meist auf Triangulierungen angewandt, z. B. in [**ES92, Fac95, MII96, ES96**]. Wir verwenden die Definition für allgemeinere Zellkomplexe.

DEFINITION 2.10. *Sei D ein Zellkomplex im $\mathbb{R}^d$. Wir betten $\mathbb{R}^d$ als Hyperebene in $\mathbb{R}^{d+1}$ ein und wählen ein Koordinatensystem derart, daß die z-Achse orthogonal auf $\mathbb{R}^d$ steht. Sei nun $D_\uparrow$ ein Polyeder im $\mathbb{R}^{d+1}$. Eine Facette von $D_\uparrow$ heißt* untere Facette*, wenn ihr auswärtiger Normalenvektor eine negative z-Koordinate hat. $D_\uparrow$ ist eine **Anhebung** von D, wenn die orthogonalen Projektionen seiner unteren Facetten auf $\mathbb{R}^d$ genau die Zellen von D sind. Gibt es ein konvexes Polyeder $D_\uparrow \subset \mathbb{R}^{d+1}$, das eine Anhebung von D ist, dann nennen wir D **regulär**. Wenn darüberhinaus $D_\uparrow$ nicht degeneriert ist, heißt D **strikt regulär**. Ist eine Anhebung $D_\uparrow$ gegeben und ist Z eine Zelle (F eine Facette) von D, dann bezeichnet $Z_\uparrow$ ($F_\uparrow$) ihr Urbild auf der Unterseite von $D_\uparrow$.*

In Verbindung mit Anhebungen bezeichnen wir die z-Richtung, die orthogonal zum eingebetteten $\mathbb{R}^d$ steht, auch als vertikale Richtung. Für eine Delaunaytriangulierung T läßt sich eine konvexe Anhebung leicht konstruieren, vergleiche etwa

[**Ede87**, S. 304]. Hierzu benutzt man das kanonische einschalige Rotationsparaboloid, dessen Rotationsachse die z-Achse ist und das in seinem Scheitelpunkt $\mathbb{R}^d$ berührt. (Genausogut eignet sich jedes einschalige Paraboloid, dessen Rotationsachse parallel zur z-Achse ist.) Die Ecken der Triangulierung werden auf das Paraboloid angehoben und die konvexe Hülle $T_\uparrow$ der angehobenen Ecken gebildet. Bei eindeutigen Delaunaytriangulierungen ist $T_\uparrow$ eine nicht degenerierte Anhebung von T. Liegen hingegen mehrere Simplizes in einer gemeinsamen Delaunayzelle, dann entspricht eine Facette von $T_\uparrow$ der gesamten Delaunayzelle. Sie muß noch geeignet in simpliziale Facetten unterteilt werden. Da diese simplizialen Facetten ko-hyperplanar sind, ist die Anhebung ein degeneriertes Polyeder. Nicht eindeutige Delaunaytriangulierungen sind also nur schwach regulär. Führt man die Paraboloid-Anhebung für ein Delaunaydiagramm durch, dann erhält man ein nicht degeneriertes Polyeder. Delaunaydiagramme sind also strikt regulär.

Auch Voronoidiagramme sind strikt regulär, siehe etwa [**dBvK$^+$97**, S. 159] oder [**Ede87**, S. 296]. Man hebt die Standorte auf das oben beschriebene Rotationsparaboloid an und legt in jedem angehobenen Punkt die tangentiale Hyperebene an das Paraboloid. Der Abschluß der maximalen über diesen Hyperebenen liegenden Punktmenge ist eine Anhebung des Voronoidiagramms. Mit fast derselben Konstruktion lassen sich auch Potenzdiagramme anheben [**Ede87**, S. 328]. Dazu hebt man den Mittelpunkt einer erzeugenden Kugel auf das Paraboloid, bildet hier die Tangentialhyperebene, und hebt diese nochmals um den quadrierten Radius der Kugel an (in z-Richtung). Die Anhebung des Potenzdiagramms ist nun der Abschluß der maximalen Punktmenge über den verschobenen Hyperebenen. Aurenhammer [**Aur87**, Theorem 4] zeigt weiter, daß jedes strikt reguläre Diagramm, dessen Zellen den gesamten $\mathbb{R}^d$ überdecken, ein Potenzdiagramm geeignet gewählter Kugeln ist.

SATZ 2.11. *Der Oriented-Walk-Algorithmus terminiert in strikt regulären Diagrammen.*

Beweis: Sei D ein strikt reguläres Diagramm mit nicht degenerierter Anhebung $D_\uparrow$. Wir wollen den Weg des Oriented Walk nicht in D, sondern auf der Unterseite von $D_\uparrow$ verfolgen. Sei daher $q_\uparrow$ die vertikale (Rück-)Projektion des Anfragepunktes q auf diese Unterseite.

Überschreite nun der Oriented Walk Facette F von Zelle Z nach Z'. Wir betrachten die Schnittpunkte der Hyperebenen $\bigvee Z_\uparrow$ und $\bigvee Z'_\uparrow$ mit der vertikalen Geraden $g := q \vee q_\uparrow$. Abbildung 2.7 zeigt einen zweidimensionalen Schnitt durch diese Situation. Die Schnittebene enthalte die Gerade g und einen beliebigen inneren Punkt von F. Aus der Konvexität von $D_\uparrow$ folgt, daß der Schnittpunkt von g mit

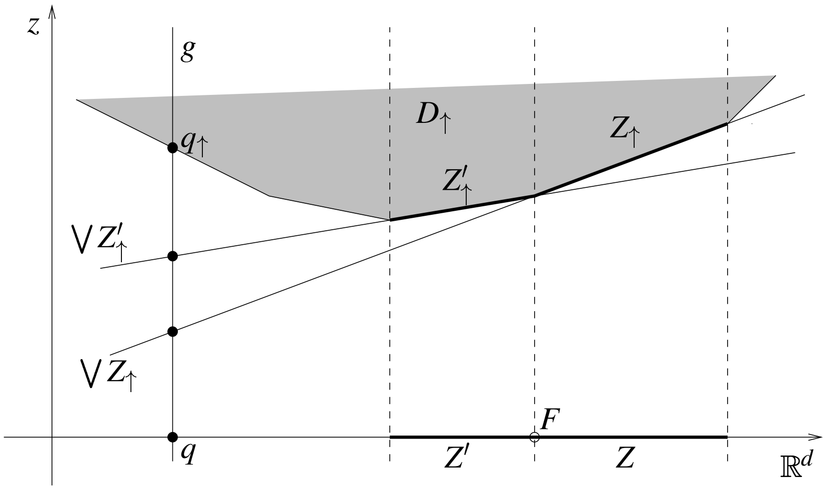


ABBILDUNG 2.7. Der Schnittpunkt von $g := q \vee q_\uparrow$ mit $\bigvee Z'_\uparrow$ liegt höher als derjenige mit $\bigvee Z_\uparrow$.

$\bigvee Z'$ echt höher liegt als derjenige mit $\bigvee Z$, vergleiche Abbildung 2.7. Das „Ansteigen" des Schnittpunktes beweist, daß der Oriented Walk keinen Zyklus durchlaufen kann. ∎

Die Nicht-Degeneriertheit der Anhebung ist für den Beweis nötig. Liegen mehrere untere Facetten von $D_\uparrow$ ko-hyperplanar, dann kann ohne zusätzliche Voraussetzungen ein Zyklus innerhalb dieser Facetten (bzw. ihrer Projektionen) nicht ausgeschlossen werden. Genau dies ist bei den höherdimensionalen Delaunaytriangulierungen in Beispiel 2.9 der Fall. Die Paraboloidanhebung ist hier degeneriert. Simplizes der Triangulierung, die in einer gemeinsamen Delaunayzelle liegen, werden in eine gemeinsame Hyperebene angehoben.

Zum Abschluß des Kapitels zeigt Abbildung 2.8 einen Überblick über die verschiedenen Klassen von Diagrammen, die wir betrachtet haben. Die Mehrzahl der angegebenen Korrektheitsbeweise folgt dem klassischen Muster zum beweisen von Schleifenterminierung. Für eine bestimmte Kenngröße wird gezeigt, daß sie sich erstens mit jedem Schleifendurchlauf streng monoton ändert und zweitens nur endlich viele Werte annehmen kann[2]. Die Kenngröße ist in unseren Fällen jeweils eine Art Abstandsmaß zwischen der aktuellen Zelle und dem Anfragepunkt. In Satz 2.11 handelt es sich um den vertikalen Abstand des angehobenen Anfragepunktes $q_\uparrow$ von der Hyperebene der angehobenen Zelle, sonst um den

[2]Der endliche Wertebereich wird oft dadurch bewiesen, daß eine ganzzahlige, etwa streng monoton fallende Kenngröße nach unten beschränkt ist.

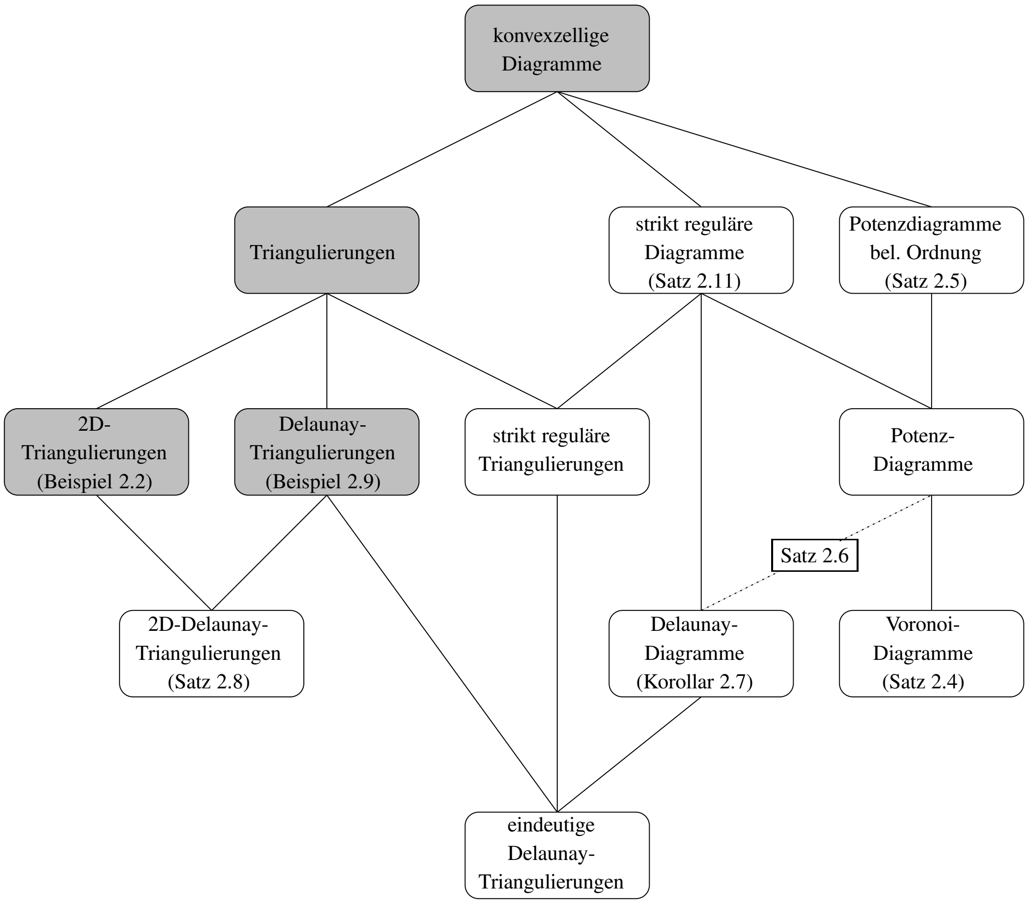


ABBILDUNG 2.8. Hierarchie der betrachteten Diagrammklassen. In den grau unterlegten Klassen können Endlosschleifen auftreten. Die durchgezogenen Verbindungslinien zeigen Teilklassenbeziehungen an. Die gepunktete Linie repräsentiert die besondere Beziehung zwischen Delaunaydiagrammen und Potenzdiagrammen.

Abstand von q zu den erzeugenden Elementen der Zelle. Der endliche Wertebereich folgt daraus, daß q fest ist und das Diagramm nur endlich viele Zellen besitzt. Bei den nicht eindeutigen Delaunaytriangulierungen fällt die Kenngröße nur schwach monoton. Im planaren Fall konnten wir die Beweislücke durch ein zusätzliches, topologisches Argument schließen. In höheren Dimensionen treten hingegen Endlosschleifen bei stationärer Kenngröße auf.

KAPITEL 3

Nicht-konvexe Triangulierungen

Dieses Kapitel bewegt sich im Kontext der Flächenrückführung durch planare Triangulierung. Diese Vorgehensweise ist immer dann anwendbar, wenn eine injektiv projizierbare Fläche abgetastet wurde. Die Abtastpunktmenge wird zunächst in eine geeignete Ebene projiziert, die projizierten Punkte werden trianguliert. Die planare Triangulierung wird auf die Abtastpunkte „rückübertragen", wodurch ein Dreiecksnetz entsteht. Bei dieser Anwendung der planaren Triangulierung begegnen uns verschiedene Versionen der Aufgabe, ein nicht-konvexes Gebiet zu triangulieren. Der Rand des Gebiets wird dabei durch einen oder mehrere Polygonzüge beschrieben.

Eine einfache Version der nicht-konvexen Triangulierungsaufgabe ist die, daß das Gebiet ein einfaches Polygon ist. Seine Eckpunkte sind so durch Diagonalen zu verbinden, daß das Polygon in Dreiecke zerlegt wird. Dabei dürfen sich die Diagonalen gegenseitig nicht überkreuzen. Für dieses Problem existiert ein Linearzeitalgorithmus von Chazelle [**Cha90, Cha91**], der jedoch sehr kompliziert und mit einem hohen konstanten Faktor behaftet ist. Woo und Shin [**WS85**] triangulieren stark kantensichtbare Polygone mit einem umgekehrten Graham's Scan in Linearzeit. In Abschnitt 3.1 sehen wir, daß dies auch für sternförmige und schwach kantensichtbare Polygone möglich ist.

Eine weitere Version ist die, daß ein Gebiet G_{zul} mit Löchern gegeben ist, dessen Rand durch mehrere einfache Polygonzüge beschrieben ist. In G_{zul} liegt die Menge P der projizierten Abtastpunkte. Gesucht ist eine im allgemeinen nicht konvexe Triangulierung von P, die eine maximale Teilmenge von G_{zul} überdeckt. Die Notwendigkeit einer nicht-konvexen Triangulierung ergibt sich häufig daraus, daß die Projektion der abgetasteten Fläche selbst nicht konvex ist. Abschnitt 3.2 entwickelt ein Verfahren für diese Aufgabe. Das Verfahren geht so vor, daß zuerst konvex trianguliert und dann aus der Triangulierung Löcher herausgeschnitten werden. Dies erleichtert die interaktive Festlegung des Gebiets G_{zul}.

Die Beschreibung der Algorithmen in diesem Kapitel geht von einer Datenstruktur aus, in der sowohl die Dreiecke als auch die Kanten einer Triangulierung dargestellt werden. In der Praxis ist es üblich, entweder nur Kanten oder nur Dreiecke explizit zu speichern. Die jeweils fehlenden Elemente sind dabei implizit gegeben und lassen sich bei Bedarf mit jeweils konstantem Zeitaufwand berechnen. Weiterhin nehmen wir an, daß in der Datenstruktur lokale Navigationsoperationen von der Art „Finde die inzidenten Eckpunkte / inzidenten Kanten / benachbarten Dreiecke eines gegebenen Dreiecks” oder „Finde eine beliebige inzidente Kante eines gegebenen Eckpunkts” nur konstante Zeit benötigen.

3.1. Triangulierung bestimmter Polygone durch Graham's Scan

Eine Grundaufgabe bei der Manipulation von Triangulierungen ist, Eckpunkte zu entfernen. Dies tritt etwa im Zusammenhang mit der Ausdünnung von Triangulierungen auf. Die Ausdünnung einer Triangulierung kann aus Effizienzgründen zweckmäßig sein, wenn sich bereits aus weniger Punkten ein Dreiecksnetz ergibt, das geometrisch nicht signifikant von dem ursprünglichen abweicht. Ein weiterer Grund, Abtastpunkte zu entfernen, sind sogenannte Ausreißer. Sowohl „entbehrliche” Punkte bei der Ausdünnung als auch Ausreißer lassen sich einfacher bestimmen, wenn man auf den Abtastpunkten bereits ein Dreiecksnetz konstruiert hat.

Ausreißer stellen typischerweise einen sehr geringen Teil der gesamten Abtastpunktmenge dar. Hier ist es effizienter, die Ausreißer aus dem bestehenden Dreiecksnetz (und der bestehenden Triangulierung) zu entfernen, als Triangulierung und Dreiecksnetz für die reduzierte Punktmenge von Grund auf neu zu konstruieren. Dasselbe gilt für Ausdünnungsverfahren, die Abtastpunkte sukzessive entfernen und dazwischen auf das modifizierte Dreiecksnetz zugreifen.

Um eine Ecke p aus der Triangulierung zu entfernen, löschen wir zunächst die Ecke selbst und alle mit ihr inzidenten Kanten und Dreiecke. Es entsteht ein Loch ω in der Triangulierung, welches wir wieder mit Kanten und Dreiecken füllen. In der Literatur ist dieses Problem unter dem Namen „Triangulierung einfacher Polygone” bekannt.

Speziell für das Entfernen von Ecken aus Delaunaytriangulierungen lösen Palacios-Velez und Cuevas Renaud [**PVCR90**] sowie, für beliebige Dimension, Schreiber [**Sch94**] dieses Problem. Die Verfahren nutzen bestimmte Eigenschaften von Delaunaytriangulierungen aus und sind daher nicht auf allgemeine Triangulierungen übertragbar.

Chazelle [**Cha90, Cha91**] beschreibt einen Linearzeit-Algorithmus zum triangulieren einfacher Polygone. Dieser Algorithmus ist jedoch sehr kompliziert, und die Laufzeit von $O(n)$ für n Ecken enthält einen hohen konstanten Faktor.

Kong, Everett und Toussaint [**KET90**] verwenden einen Graham's Scan, quasi mit umgekehrtem Vorzeichen, um ein einfaches Polygon ω zu triangulieren. Der ursprüngliche Graham's Scan [**Gra72**] füllt konkave Ecken eines sternförmigen Polygons auf, um die konvexe Hülle zu bilden (vergleiche auch Kapitel 4, Seite 66). Er benötigt dazu lineare Zeit. Der Algorithmus von Kong et al. schneidet konvexe Ecken ab, bis das Polygon nur noch ein Dreieck ist. Das Abschneiden ist eine Operation auf dem Polygonzug, der ω berandet. Es löscht eine Ecke p_i aus dem Polygonzug und ersetzt die Kanten $p_{i-1} \wedge p_i$ und $p_i \wedge p_{i+1}$ durch $p_{i-1} \wedge p_{i+1}$. Eine konvexe Ecke p_i darf genau dann abgeschnitten werden, wenn das abgeschlossene Dreieck δ_i, das sie mit ihrer Vorgänger– und Nachfolger-Ecke bildet, ein sogenanntes Ohr ist.

DEFINITION 3.1. *Seien p_{i-1}, p_i und p_{i+1} drei aufeinanderfolgende Ecken eines einfachen Polygons ω. Das Dreieck $\delta_i := p_{i-1} \wedge p_i \wedge p_{i+1}$ heißt **Ohr** von ω, wenn das Innere der Strecke $p_{i-1} \wedge p_{i+1}$ im Innern von ω liegt.*

Ist δ_i kein Ohr, dann hätte der Polygonzug nach Abschneiden von p_i Selbstüberschneidungen. Diese zusätzliche Bedingung steht im Gegensatz zum ursprünglichen Graham's Scan, der jede konkave Ecke auffüllen darf. Der Algorithmus von Kong et al. testet für jede konvexe Ecke p_i, ob δ_i ein Ohr ist. Dazu überprüft er, ob eine der konkaven Ecken des Polygons in δ_i liegt. Ist dies der Fall, dann ist δ_i kein Ohr und die Ecke p_i wird nicht abgeschnitten. Der Ohrtest benötigt für ein Dreieck $O(m)$ Zeit, wobei m die Anzahl der konkaven Ecken im Polygon ist. Das Durchlaufen der Ecken ist so organisiert wie beim Graham's Scan. Es benötigt $O(n)$ Zeit und kommt mit $O(n)$ Ohrtests aus. Der Zeitaufwand des gesamten Algorithmus ist daher $O(nm)$ für ein Polygon mit n Ecken. Die abgeschnittenen Ohren und das verbleibende Dreieck bilden schließlich eine Triangulierung des ursprünglichen Polygons.

Für diese Arbeit sind zwei spezielle Klassen von einfachen Polygonen relevant, nämlich sternförmige und schwach kantensichtbare Polygone. Sternförmige Polygone entstehen beispielsweise beim Entfernen einer Ecke aus einer Triangulierung. Schwach kantensichtbare Polygone begegnen uns in Algorithmus 3.3. Wir werden sehen, daß diese Polygone sich durch einen umgekehrten Graham's Scan in linearer Zeit triangulieren lassen.

Im folgenden sei ω stets ein einfaches Polygon mit den Ecken $p_1, \ldots, p_n$.

DEFINITION 3.2. *Eine* ***Ausleuchtmenge*** *in* ω *ist eine abgeschlossene, zusammenhängende Menge* $A \subseteq \omega$ *derart, daß jeder Punkt von* ω *von einem Punkt der Ausleuchtmenge aus sichtbar ist.*[1]

Ein Polygon ω *heißt*

- ***schwach kantensichtbar****, wenn eine seiner Kanten eine Ausleuchtmenge ist, und*
- ***stark kantensichtbar****, wenn ein Punkt auf einer seiner Kanten eine Ausleuchtmenge ist.*

BEMERKUNG 3.3. *Ein Polygon ist sternförmig genau dann, wenn es eine Ausleuchtmenge besitzt, die aus nur einem Punkt besteht.*

Das Loch ω, das beim Löschen einer Ecke p samt inzidenter Kanten und Dreiecke aus der Triangulierung entsteht, ist sternförmig: Die Ecke p war gemeinsame Ecke der Dreiecke, deren Vereinigung ω ist. In Abschnitt 3.2 werden uns auch schwach kantensichtbare Polygone begegnen.

Woo und Shin [**WS85**] triangulieren stark kantensichtbare Polygone durch einen umgekehrten Graham's Scan in linearer Zeit. Weiter verwenden sie den umgekehrten Graham's Scan als Subroutine, um sternförmige Polygone zu triangulieren. Dabei wird ein sternförmiges Polygon in bis zu drei stark kantensichtbare Polygone unterteilt.

Mit Hilfe unserer Definition einer Ausleuchtmenge können wir zeigen, daß der umgekehrte Graham's Scan direkt auf sternförmige und schwach kantensichtbare Polygone anwendbar ist.

LEMMA 3.4. *Sei* A *Ausleuchtmenge in* ω *und* p_i *konvexe Ecke von* ω*. Sei weiter* $\delta_i = p_{i-1} \wedge p_i \wedge p_{i+1}$*. Wenn* δ_i *die Ausleuchtmenge nicht oder nur in* p_{i-1} *und* p_{i+1} *schneidet, also* $A \cap \delta_i \setminus \{p_{i-1}, p_{i+1}\} = \emptyset$ *gilt, dann ist* δ_i *ein Ohr und* A *ist Ausleuchtmenge des einfachen Polygons* $\omega' := \overline{\omega \setminus \delta_i}$*.*

Beweis:
Sei $\delta_i^* = \delta_i \setminus \{p_{i-1}, p_{i+1}\}$. Wir betrachten je eine Sichtstrecke s_1 von A zu p_{i-1} und s_2 von A zu p_{i+1}, wobei die Sichtstrecken in ω liegen. Die Sichtstrecken haben keinen Punkt mit δ_i^* gemeinsam. Die in A liegenden Endpunkte der Sichtstrecken seien q_1 und q_2. Wir lassen ausdrücklich $q_1 = p_{i-1}$ und $q_2 = p_{i+1}$ zu. O. B. d. A. haben die Sichtstrecken mit A nur q_1 bzw. q_2 gemeinsam. (Andernfalls enthält

[1]Wäre die Ausleuchtmenge eine Lichtquelle, dann würde sie das gesamte Polygon ausleuchten.

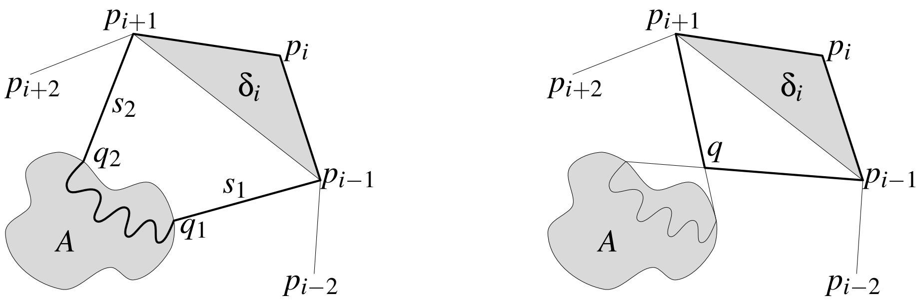


ABBILDUNG 3.1. Die Strecke $p_{i-1} \wedge p_{i+1}$ wird von einer geschlossenen Kurve $\mathcal{C}$ (fett gezeichnet) umgeschlossen.

die Menge $s_1 \cap A$ einen Punkt q_1' mit minimalem Abstand zu p_{i-1}, und wir ersetzen s_1 durch $q_1' \wedge p_{i-1}$. Mit s_2 verfahren wir analog.) Die Endpunkte q_1, q_2 der Sichtstrecken sind durch einen geeigneten, von Selbstüberschneidungen freien Pfad in A verbunden. Die anderen Endpunkte der Sichtkanten sind durch die Kanten $p_{i-1} \wedge p_i$ und $p_i \wedge p_{i+1}$ von δ_i verbunden. Die Vereinigung dieser fünf Objekte ist eine geschlossene Kurve $\mathcal{C}$ ohne Selbstüberschneidungen, sofern die beiden Sichtstrecken zueinander disjunkt sind. Dies ist in Abbildung 3.1 links dargestellt. Schneiden sich s_1 und s_2 in einem Punkt q wie in Abbildung 3.1 rechts, dann erhalten wir die Kurve $\mathcal{C}$ aus den beiden Polygonkanten und den Strecken $p_{i-1} \wedge q$ und $p_{i+1} \wedge q$. In beiden Fällen liegt die von $\mathcal{C}$ eingeschlossene Fläche in ω. Das Innere der Strecke $p_{i-1} \wedge p_{i+1}$ wird von $\mathcal{C}$ strikt eingeschlossen, und liegt daher im Innern von ω. Dies zeigt, daß δ_i ein Ohr von ω ist. Aus $\omega = \omega' \cup \delta_i^*$ und $A \cap \delta_i^* = \emptyset$ folgt schließlich, daß A in ω' enthalten und eine Ausleuchtmenge von ω' ist. ■

Wir ersetzten nun den Ohrtest im Algorithmus von Kong et al. durch den Test, ob die betrachtete Ecke p_i konvex ist und ob $\delta_i^* = \delta_i \setminus \{p_{i-1}, p_{i+1}\}$ die Ausleuchtmenge schneidet. Unter den Bedingungen

1. Der Test ist in konstanter Zeit möglich.
2. Der Test kann nicht alle Ohren eines Polygons übersehen.

erhalten wir einen Linearzeitalgorithmus.

Für sternförmige Polygone wählen wir als Ausleuchtmenge einen Punkt p aus dem Kern. Der Kern läßt sich nach Lee und Preparata [**LP79**] in linearer Zeit bestimmen. Bei unseren sternförmigen Triangulierungslöchern wissen wir bereits, daß die gelöschte Triangulierungsecke p zum Kern gehört. Mit dem Test aus Lemma 3.4 erhalten wir Algorithmus 3.1.

Algorithmus 3.1 Triangulieren eines sternförmigen Polygons

```
Eingabe: Ecken p_1, ..., p_n des Polygons als doppelt verkettete Liste (n ≥ 4),
    Punkt p aus seinem Kern.
1:  i := 1.
2:  while ω hat mehr als 4 Ecken do
3:      δ*_i := (p_{i−1} ∧ p_i ∧ p_{i+1}) \ {p_{i−1}, p_{i+1}}.
4:      if p_i ist konvexe Ecke des Lochs and p ∉ δ*_i then
5:          Schneide die Ecke p_i von ω ab.
6:          i := i − 1.
7:      else
8:          i := i + 1.
9:      endif
10: endwhile
11: Teile ω in zwei Dreiecke auf.
```

Die Schritte 6 und 8 sind so zu verstehen, daß der Algorithmus mit der Vorgänger– bzw. Nachfolger-Ecke von p_i fortfährt. Vorgänger und Nachfolger beziehen sich dabei auf das aktuelle Polygon (in Schritt 6 auf das Polygon, bevor p_i abgeschnitten wurde), nicht auf das ursprünglich eingegebene Polygon. Beim Abschneiden einer Ecke p_i fügen wir das Dreieck δ_i sowie dessen dritte Kante in die Triangulierung ein. (Da δ_i durch drei aufeinanderfolgende Punkte des Lochs gebildet wird, sind zwei seiner Kanten bereits in der Triangulierung enthalten.) Zu Schritt 11 ist anzumerken, daß sich jedes Viereck durch eine Diagonale in zwei Dreiecke zerlegen läßt (vergleiche Abbildung 1.5 auf Seite 8), und daß man dazu konstante Zeit benötigt. Die Diagonale und die beiden Dreiecke fügen wir in die Triangulierung ein. Entsteht beim Löschen einer Triangulierungsecke ein dreieckiges Loch, dann fügen wir das Dreieck direkt in die Triangulierung ein, ohne Algorithmus 3.1 aufzurufen.

SATZ 3.5. *Algorithmus 3.1 trianguliert ein sternförmiges Polygon ω mit $n \geq 4$ Ecken in Zeit $O(n)$.*

Beweis: Ist $n = 4$, dann trianguliert Schritt 11 das Polygon. Solange ω noch mehr als 4 Ecken hat, werden in der **while**-Schleife alle Ecken getestet, bis der Algorithmus ein Ohr findet. Die Bedingung in Schritt 4 ist hinreichend dafür, daß δ_i ein Ohr ist, aber nicht notwendig. Es könnte daher sein, daß unser umgekehrter Graham's Scan kein Ohr findet. Meisters [**Mei75**] zeigt, daß jedes einfache Polygon mit $n \geq 4$ Ecken mindestens zwei Ohren δ_i, δ_j besitzt, die sich gegenseitig nicht überlappen. Findet der Algorithmus beide Ohren nicht, dann muß p in $\delta_i^* \cap \delta_j^*$ liegen. Daraus folgt, daß sich die Strecken $p_{i-1} \wedge p_{i+1}$ und $p_{j-1} \wedge p_{j+1}$ überlappen

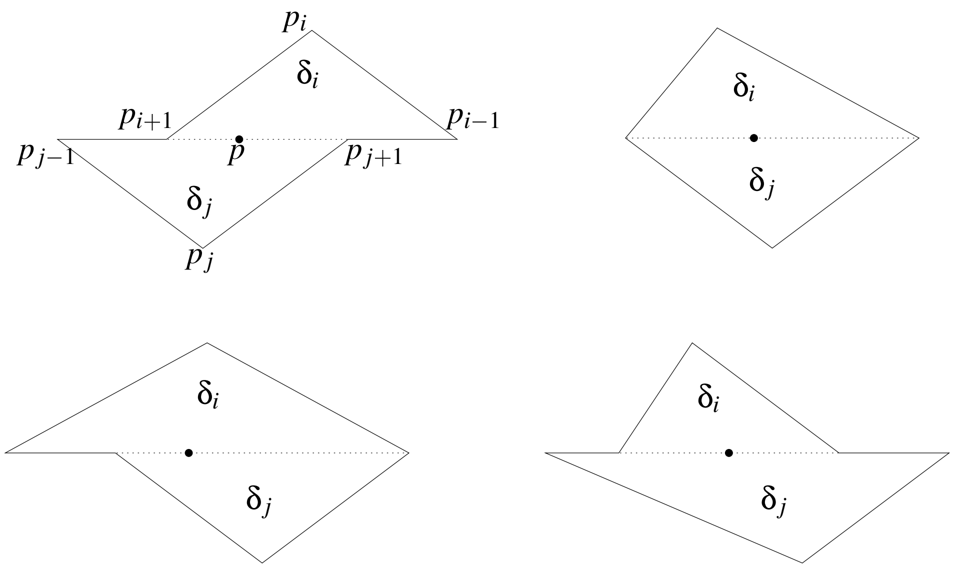


ABBILDUNG 3.2. Sternförmige Polygone mit aneinandergrenzenden Ohren δ_i und δ_j.

und daß p im Innern der Überlappung liegt. Abbildung 3.2 zeigt einige mögliche Formen von ω. Sind die Kanten identisch, dann ist ω wegen $p_{i-1} = p_{j+1}$ und $p_{j-1} = p_{i+1}$ ein Viereck und wird in Schritt 11 trianguliert. Andernfalls ist eines der Dreiecke $\delta_{i-1}, \delta_{i+1}, \delta_{j-1}, \delta_{j+1}$ ein Ohr, das p nicht enthält. Der Algorithmus muß dieses (oder ein anderes) Ohr finden.

Der Ohrtest für eine Ecke p_i benötigt nur konstante Zeit. Daraus ergibt sich eine lineare Laufzeit für den gesamten Scan. ∎

In ähnlicher Weise verfahren wir auch für schwach kantensichtbare Polygone.

SATZ 3.6. *Schwach kantensichtbare Polygone können durch einen umgekehrten Graham's Scan in linearer Zeit trianguliert werden.*

Beweis: Sei o. B. d. A. die Kante $p_n \wedge p_1 =: A$ Ausleuchtmenge des Polygons ω. Zur Triangulierung benutzen wir einen umgekehrten Graham's Scan analog zu Algorithmus 3.1. Wir ersetzen lediglich den Test $p \notin \delta_i^*$ in Schritt 4 durch $A \cap \delta_i^* = \emptyset$ oder, was äquivalent ist,

$$p_n \notin \delta_i^* \textbf{ and } p_1 \notin \delta_i^*$$

Wenn die Dreiecke δ_n und δ_1 Ohren sind, dann überlappen sie sich. Es gibt also ein Ohr $\delta_i \neq \delta_n, \delta_1$, sofern ω kein Dreieck ist. Würde δ_i^* die Polygonkante $p_n \wedge p_1$ schneiden, dann wäre δ_i kein Ohr. Es ist also sichergestellt, daß der Algorithmus δ_i als Ohr erkennt. ∎

Wie in Kapitel 1 (Seite 6 ff.) erwähnt, kommen bei der Flächenrückführung häufig Triangulierungen zum Einsatz, die eine vorgegebene Zielfunktion optimieren, um die Qualität des resultierenden Dreiecksnetzes zu verbessern. Durch das Entfernen von Ecken und Neutriangulierung des entstandenen Lochs wird die Optimalität im allgemeinen verloren gehen. Wir schließen daher eine Optimierung durch lokale Suche mit Diagonalentausch an. Da sich nur ein begrenzter Teil der Triangulierung geändert hat, sollten — eine entsprechende Größe der Triangulierung vorausgesetzt — die meisten Kanten nach wie vor optimal sein. Es ist zu erwarten, daß nur in einer bestimmten Umgebung der geänderten Stelle Kanten auf Suboptimalität überprüft und gegebenenfalls getauscht werden müssen.

3.2. Triangulieren mit Löchern

Wenn eine abgetastete Fläche Löcher aufweist, sollten diese auch in der Triangulierung und im Dreiecksnetz vorhanden sein. Das gilt gleichermaßen für Einbuchtungen am Rand der projizierten Fläche. Abbildung 3.3 zeigt eine solche Abtastung. Unter der Annahme gestreuter Abtastung mit möglicherweise stark unterschiedlichen Abtastdichten ist eine automatische Erkennung von Löchern und Einbuchtungen kaum möglich. Ein Mensch hingegen kann diese Stellen durch einen Vergleich mit der abgetasteten Fläche leicht identifizieren.

Lewis und Robinson [**LR78**] und De Floriani, Falcidieno und Pienovi [**DFFP85**] geben Algorithmen zur Triangulierung von nicht-konvexen Gebieten an. Für beide Algorithmen besteht die Eingabe aus der Menge P der projizierten Abtastpunkte und dem gewünschten Triangulierungsgebiet G. Der Rand des Gebiets muß durch Polygonzüge vorgegeben werden, deren Eckpunkte projizierte Abtastpunkte sind. Liegen um ein Loch herum viele Abtastpunkte, dann kann das interaktive Festlegen eines solchen Polygonzugs sehr aufwendig sein. Auch muß der Benutzer immer einen äußeren Rand angeben, selbst dann, wenn dieser konvex ist. Ein weiterer Nachteil dieser Algorithmen ist, daß sie keine dynamische Änderung des Triangulierungsgebiets erlauben. Dadurch ist eine Korrektur nach erfolgter Triangulierung nicht möglich, bzw. für jede noch so kleine Korrektur muß die gesamte Punktmenge neu trianguliert werden.

In diesem Abschnitt wird ein Verfahren entwickelt, das diese Nachteile vermeidet. Seine Grundidee ist ein zulässiges Gebiet, innerhalb dessen sich die Triangulierung „ausbreiten" kann, das sie aber nicht vollständig überdecken muß. Das zulässige Gebiet wird durch **Absperrungen** festgelegt. Eine Absperrung ist ein einfacher geschlossener Polygonzug, dessen Eckpunkte der Benutzer frei wählen kann. Er muß jedoch beachten, daß kein projizierter Abtastpunkt im abgesperrten

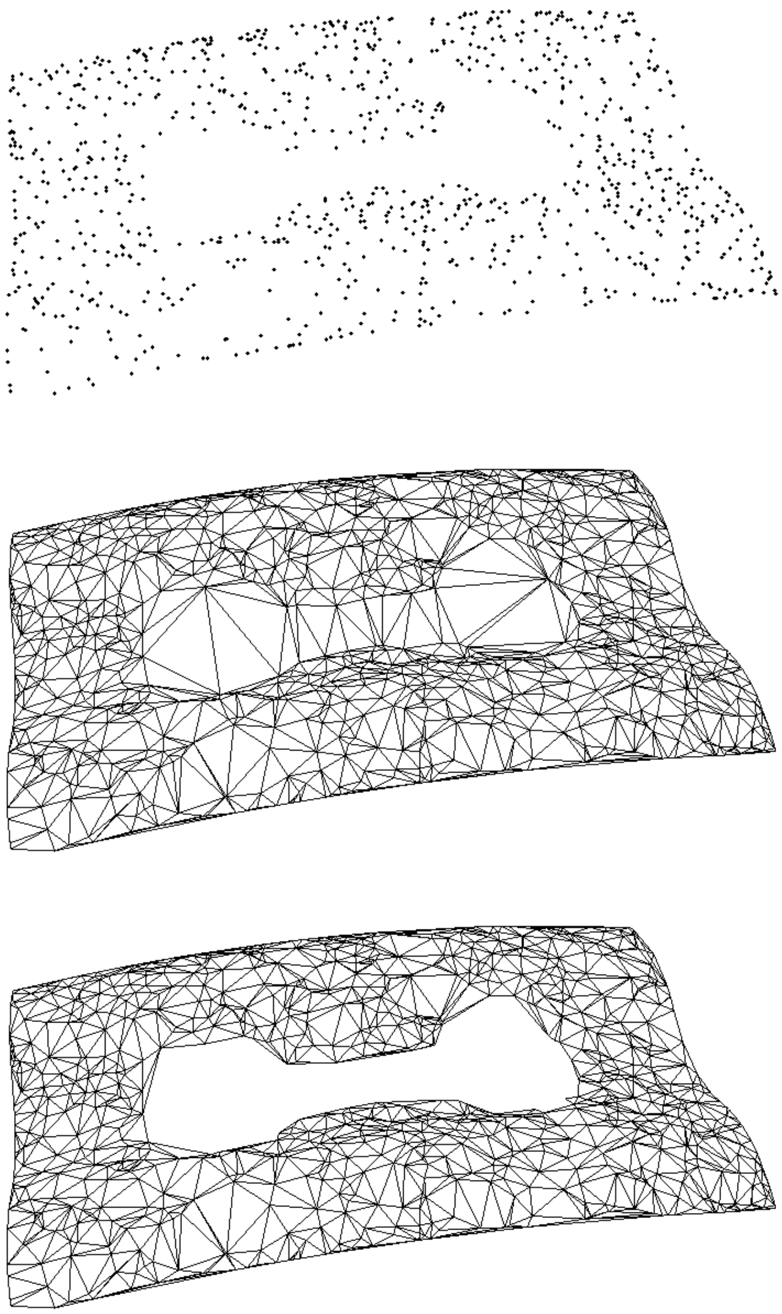


ABBILDUNG 3.3. Oben: Abtastpunkte von einer Fläche mit Loch (Bedienblende). Mitte: konvexe Triangulierung. Unten: Triangulierung mit Loch.

Bereich liegen darf. Der äußere Rand des zulässigen Gebiets kann durch eine Absperrung angegeben werden. Es ist aber auch möglich, einzelne Einbuchtungen durch entsprechend gewählte Absperrungen zu erzwingen. Ist beides nicht der Fall, dann ergibt sich als äußerer Rand der Triangulierung automatisch der Rand der konvexen Hülle. Das zulässige Gebiet läßt sich dynamisch ändern, so daß der Benutzer ganz gezielt unerwünschte Dreiecke und Kanten aus der Triangulierung herausschneiden kann.

Das zulässige Gebiet ist ein zusammenhängendes, möglicherweise unbeschränktes Polygon G_{zul}. Seinen Rand bilden die Absperrungen. Sie müssen paarweise disjunkt sein. Das Ziel unseres Verfahrens ist eine Triangulierung, die einen möglichst großen Teil von G_{zul} überdeckt. Je nach Wahl des zulässigen Gebiets kann dabei eine Triangulierung mit mehreren Zusammenhangskomponenten entstehen. In einigen Fällen kann man bestimmte Punkte zwar durch Kanten, aber nicht durch Dreiecke verbinden. Solche Kanten, die dann mit keinem Dreieck inzident sind, wollen wir ausdrücklich zulassen. Dazu müssen wir in diesem Abschnitt einen anderen Triangulierungsbegriff verwenden als in Definition 1.10.

DEFINITION 3.7. *Sei P eine endliche Punktmenge im $\mathbb{R}^2$. Eine **teilweise Triangulierung** T von P besteht aus einer Menge D von Dreiecken und einer Menge K von Kanten, für die gilt:*

1. *D ist eine Triangulierung gemäß Definition 1.10.*
2. *Die Randkanten aller $\delta \in D$ sind in K enthalten.*
3. *Die Ecken aller $\delta \in D$ und die Endpunkte aller $k \in K$ sind in P enthalten.*
4. *Der Schnitt zweier Kanten aus K ist leer oder ein gemeinsamer Endpunkt.*
5. *Ist die Kante $k \in K$ nicht Randkante von $\delta \in D$, dann ist ihr Schnitt mit δ leer oder ein gemeinsamer End–/Eckpunkt.*

*Das **Gebiet** der Triangulierung ist die Vereinigung ihrer Dreiecke, Kanten und Punkte.*

In diesem Abschnitt wollen wir den Begriff Triangulierung im Sinn von Definition 3.7 verstehen. Die projizierten Abtastpunkte bilden die Menge P. Wir suchen eine teilweise Triangulierung von P mit maximalem Gebiet innerhalb G_{zul}. Wenn G_{zul} die konvexe Hülle von P umfaßt, insbesondere im Fall $G_{\text{zul}} = \mathbb{R}^2$, dann ist das maximal mögliche Gebiet die konvexe Hülle von P. Abbildung 3.4 zeigt ein Beispiel mit einer Absperrung, die in die konvexe Hülle hineinragt. Hier gibt es zwei maximale Triangulierungen mit unterschiedlichen Gebieten.

In Abbildung 3.4 ist der rechte untere Eckpunkt der Absperrung gleichzeitig ein projizierter Abtastpunkt. Die zwei übrigen Eckpunkte gehören nicht zu P.

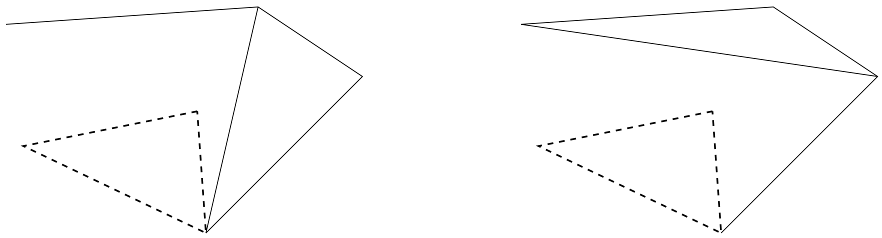


ABBILDUNG 3.4. Zwei Triangulierungen mit maximalen Gebieten. Die Absperrung ist gestrichelt dargestellt.

Zur Unterscheidung bezeichnen wir im folgenden alle projizierten Abtastpunkte als **Datenecken**, und alle Absperrungs-Eckpunkte, die keine Datenecken sind, als **Hilfsecken**. Unser Algorithmus konstruiert eine **Hilfstriangulierung** $\widetilde{T}$ der Daten– und Hilfsecken. $\widetilde{T}$ überdeckt die gemeinsame konvexe Hülle dieser Ecken. Alle Kanten der Absperrungen sind in der Kantenmenge $\widetilde{K}$ von $\widetilde{T}$ enthalten. Diejenigen Kanten und Dreiecke von $\widetilde{T}$, die in einem abgesperrten Bereich liegen oder mit einer Hilfsecke inzident sind, werden als **Hilfskanten und –dreiecke** bezeichnet. Die übrigen Kanten und Dreiecke heißen **eigentlich**. Sie bilden zusammen mit den Datenecken die **eigentliche Triangulierung** T.

Die Strahlverfolgung in Form von Algorithmus 2.2 kann Löcher und Einbuchtungen weder überspringen, noch feststellen, ob der gesuchte Punkt in ihnen liegt. Um Punkte in einer nicht-konvexen Triangulierung zu lokalisieren, müßte Algorithmus 2.2 erheblich erweitert werden. Die Hilfstriangulierung löst dieses Problem auf einfache Weise. Sie überdeckt Löcher und Einbuchtungen mit Hilfsdreiecken und macht sie so für die Strahlverfolgung zugänglich.

Der Benutzer erzeugt eine Absperrung, indem er nacheinander deren Eckpunkte $r_1, \dots, r_m$ eingibt. Die Eckpunkte können bereits vorhandene Datenecken sein, oder beliebige andere Punkte, die dann als Hilfsecken in die Hilfstriangulierung eingefügt werden. Das Programm stellt sicher, daß die Absperrungen paarweise disjunkt und nicht ineinander verschachtelt sind. Eckpunkte r_i, die in einem abgesperrten Bereich liegen, weist es zurück. Sobald der Benutzer r_{i+1} festgelegt hat, versucht das Programm eine (Hilfs-)Kante $k_i = r_i \wedge r_{i+1}$ zu erzeugen. Stellt sich hierbei heraus, daß $r_i \wedge r_{i+1}$ eine vorhandene Absperrung schneidet, dann wird die Eingabe r_{i+1} zurückgewiesen. Dies garantiert zum einen paarweise disjunkte Absperrungen, zum anderen verhindert es auch Selbstüberschneidungen der Absperrung, die gerade angelegt wird. Zum Abschluß der Absperrung werden r_m und r_1 durch eine Kante k_m verbunden. Dabei wird überprüft, daß im Innern des abgesperrten Bereichs weder Hilfs– noch Datenecken liegen.

Wenn r_i und r_{i+1} nicht bereits durch eine Kante k_i verbunden sind, muß der Algorithmus diese Kante erzeugen. Er bestimmt zunächst alle Kanten und Dreiecke, die das Innere der Strecke $r_i \wedge r_{i+1}$ schneiden. Hierzu dient eine Strahlverfolgung von r_i nach r_{i+1} (vgl. Algorithmus 2.2). Die gesuchten Kanten und Dreiecke sind genau diejenigen, die während der Strahlverfolgung überschritten bzw. durchlaufen werden. Während der Strahlverfolgung erfolgt die Überprüfung, daß $r_i \wedge r_{i+1}$ keine Absperrung schneidet. Die bei der Strahlverfolgung gefundenen Kanten und Dreiecke werden aus der Hilfstriangulierung gelöscht. Dabei entsteht ein echtes Loch, das von k_i in zwei Hälften geteilt wird.

Die beiden Lochhälften werden mit Kanten und Dreiecken gefüllt, um sofort wieder eine Hilfstriangulierung mit konvexem Gebiet herzustellen. Die Lochhälften sind schwach kantensichtbar mit Kante k_i als Ausleuchtmenge: Jeder Punkt einer Lochhälfte gehört zu einem der gelöschten Dreiecke, muß also vom Schnitt dieses Dreiecks mit k_i aus sichtbar sein. Zum Füllen einer Lochhälfte benutzen wir den gemäß Satz 3.6 modifizierten Algorithmus 3.1.

Beim Füllen der Lochhälften wird das Gebiet der eigentlichen Triangulierung nicht berücksichtigt. Die Maximierung von T erfolgt nach dem Abschluß der Absperrung, d. h. wenn die Randkanten $k_1, \ldots, k_m$ in der Hilfstriangulierung enthalten sind.

Wir führen die Maximierung auf ein Sichtbarkeitsproblem zurück. Dazu betrachten wir die Zwischenräume zwischen der eigentlichen Triangulierung und den Absperrungen. T ist genau dann maximal, wenn in keinen Zwischenraum mehr eine eigentliche Kante, also eine Kante zwischen zwei Datenecken, eingefügt werden kann. Dies ist wiederum genau dann der Fall, wenn es keinen Zwischenraum Z gibt, in dem zwei Datenecken gegenseitig strikt sichtbar sind.

Unser Verfahren führt die Maximierung von T aus, sobald alle Kanten einer neuen Absperrung in der Hilfstriangulierung $\widetilde{T}$ erzeugt sind. Um T zu maximieren, sucht das Programm nach gegenseitig strikt sichtbaren Datenecken am Rand eines Zwischenraums Z. Hierbei nutzt es aus, daß Z bereits durch $\widetilde{T}$ trianguliert ist. Die Suche startet von einer Datenecke p am Rand von Z aus, und bestimmt alle von p aus innerhalb Z strikt sichtbaren Datenecken. Jede dieser Ecken wird mit p durch eine Kante verbunden. Die Verbindungskanten werden auf die gleiche Art erzeugt, wie vorher schon die Absperrungskanten. Nachdem Sichtbarkeitssuche und Verbindungen für jede Datenecke am Rand von Z erfolgt sind, ist das Gebiet von T maximal.

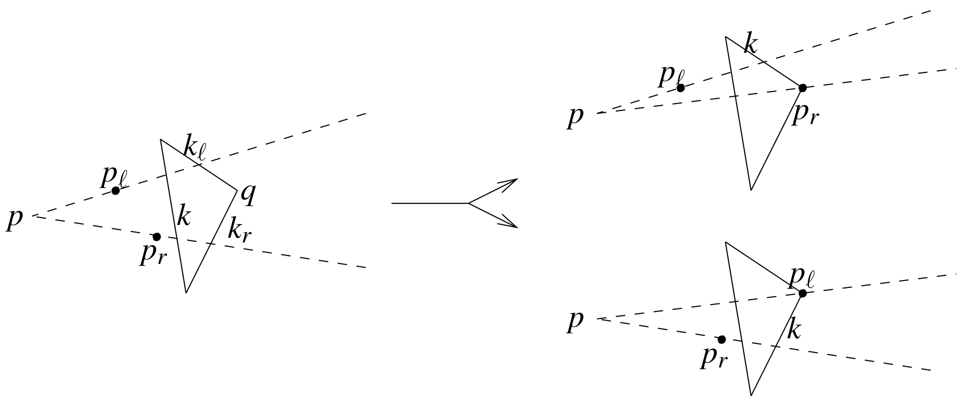
Algorithmus 3.2 führt die Sichtbarkeitssuche innerhalb des Zwischenraums Z durch. Er betrachtet zu jedem Zeitpunkt einen Winkel, d. h. eine von zwei Strahlen begrenzte Teilfläche der Ebene, dessen Scheitelpunkt in der Datenecke p liegt. Diesen Suchwinkel verfolgt der Algorithmus durch die Hilfstriangulierung $\widetilde{T}$. Wenn er auf eine Ecke q stößt, die innerhalb des Suchwinkels liegt, dann unterteilt er den Winkel an dem Strahl von p durch q. Die beiden Teilwinkel werden rekursiv verfolgt. Die Verfolgung selbst besteht im Überschreiten von Kanten von $\widetilde{T}$. Wird eine Randkante von Z überschritten, dann bricht die Verfolgung ab. Abbildung 3.5 zeigt die verschiedenen Fälle beim Überschreiten einer Kante.

Wenn p eine andere Datenecke q sieht, dann läuft die Verbindungskante durch ein Dreieck δ, das mit p inzident ist und in Z liegt. Am Anfang unserer rekursiven Winkelverfolgung stehen daher Suchwinkel, die solche Dreiecke δ genau umfassen. Algorithmus 3.2 repräsentiert den Suchwinkel durch zwei Ecken p_ℓ und p_r, die auf dem linken und rechten Begrenzungsstrahl liegen. Diese Ecken sind stets von p verschieden und beschreiben daher den Suchwinkel eindeutig. Algorithmus 3.2 führt die rekursive Sichtbarkeitssuche durch.

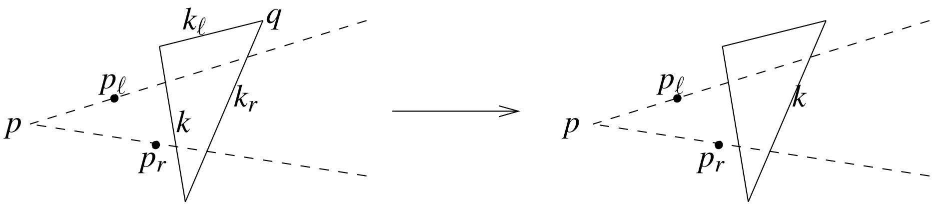
Die Bedingungen in den Schritten 12 und 14 sind auch dann erfüllt, wenn q auf dem jeweiligen Begrenzungsstrahl liegt. In diesem Fall wird q durch p_ℓ bzw. p_r verdeckt, ist also nicht strikt sichtbar.

Sei der Zwischenraum Z m-fach zusammenhängend und habe n Ecken. Die Laufzeit von Algorithmus 3.2 hängt von n und m ab. Bis auf die **for**-Schleife und die Rekursion benötigen alle Schritte des Algorithmus konstante Zeit. Da in jedem Durchlauf der **for**-Schleife mindestens ein rekursiver Aufruf erfolgt, ist die Laufzeit des Algorithmus proportional zur Gesamtanzahl der rekursiven Aufrufe. In jedem rekursiven Aufruf wird genau einmal eine Kante überschritten. Eine Kante k kann mehrmals überschritten werden, jeweils in verschiedenen rekursiven Aufrufen und mit verschiedenen Suchwinkeln. Dabei können sich die Suchwinkel, die k überschreiten, paarweise nicht überlappen. Vielmehr müssen sie oder zwei ihrer Vorfahren im Rekursionsbaum durch eine Ecke q von Z getrennt worden sein. Abbildung 3.6 veranschaulicht dies.

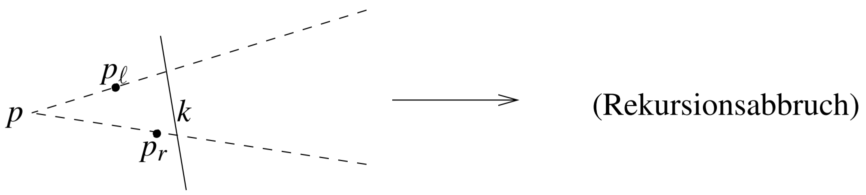
Die Ecke q gehört zum Rand von Z. Die Kante k und die Teile der beiden Suchwinkel zwischen p und k liegen in Z und umschließen q. Daher muß q Ecke einer Absperrung in Z sein, und die gesamte Absperrung wird von den beiden Suchwinkeln und k eingeschlossen. Z ist m-fach zusammenhängend, enthält also $m-1$ Absperrungen. Daher kann k von maximal m Suchwinkeln überschritten werden. Im Inneren von Z liegen $O(n)$ Kanten. Algorithmus 3.2 kann also maximal $O(nm)$ mal eine Kante überschreiten, und benötigt $O(nm)$ Zeit.



a) rekursive Unterteilung des Suchwinkels



b) keine Unterteilung



c) k ist Randkante von Z

ABBILDUNG 3.5. Die Sichtbarkeitssuche beim Überschreiten einer Kante k.

Algorithmus 3.2 Sichtbarkeitssuche

Eingabe: Datenecke p, Zwischenraum Z (als Teil der Hilfstriangulierung).
Ausgabe: Menge P_s der von p aus in Z strikt sichtbaren Datenecken.

$P_s := \emptyset$.
for jedes mit p inzidente Dreieck δ mit $\delta \subset Z$ **do**
  $k :=$ Kante von δ, die p gegenüberliegt.
  $p_\ell, p_r :=$ Endpunkte von k (in der entsprechenden Reihenfolge).
  ($*$ Anfang der rekursiven Verfolgung eines Suchwinkels $*$)
  ($*$ p_ℓ und p_r markieren die linke und rechte Begrenzung des Winkels, k ist die Kante, die als nächste überschritten wird. $*$)
  if k ist Randkante von Z **then**
    Rekursionsabbruch
  endif
  $\delta :=$ Dreieck hinter k.
  $q :=$ Ecke von δ, die k gegenüberliegt.
  $k_\ell, k_r :=$ Kanten von δ, die mit q inzident sind. ($*$ Vergleiche Abbildung 3.5. $*$)
  if q liegt nicht rechts des Strahls von p durch p_ℓ **then**
    Verfolge rekursiv mit $k := k_r$ (p_ℓ, p_r wie gehabt).
  elseif q liegt nicht links des Strahls von p durch p_r **then**
    Verfolge rekursiv mit $k := k_\ell$ (p_ℓ, p_r wie gehabt).
  else ($*$ q liegt im Innern des Suchwinkels. $*$)
    if q ist Datenecke **then** $P_s := P_s \cup \{q\}$ **endif**
    Verfolge rekursiv mit $k := k_\ell$ und $p_r := q$ (p_ℓ wie gehabt).
    Verfolge rekursiv mit $k := k_r$ und $p_\ell := q$ (p_r wie gehabt).
  endif
  ($*$ Ende der rekursiven Verfolgung eines Suchwinkels $*$)
endfor

Während der Sichtbarkeitssuche für eine einzelne Datenecke p ändert sich die Hilfstriangulierung nicht. Sobald P_s bestimmt ist, werden Kanten von p zu allen Ecken $q \in P_s$ erzeugt (sofern P_s nicht leer ist). Durch das Erzeugen dieser Verbindungskanten, die ja eigentliche Kanten sind, wird die eigentliche Triangulierung vergrößert und Z verkleinert. Danach erfolgt die Sichtbarkeitssuche für die nächste Datenecke am Rand von Z. Diese Suche berücksichtigt die zuvor erzeugten eigentlichen Kanten, d. h. sie findet innerhalb des verkleinerten Zwischenraums statt. Beim Erzeugen der Verbindungskanten können Sichtbarkeiten zwischen Randecken von Z unterbrochen werden, aber es können keine neuen Sichtbarkeiten entstehen. Daher muß jede Datenecke am Rand von Z nur einmal

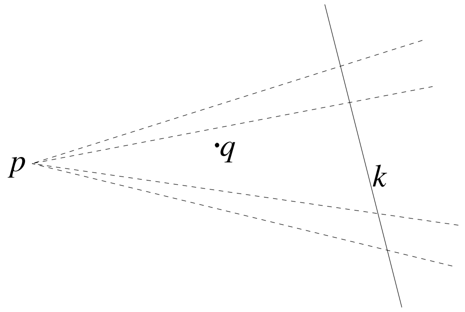


ABBILDUNG 3.6. Zwei Suchwinkel überkreuzen dieselbe Kante k.

Algorithmus 3.3 Erzeugen einer Kante $p \wedge q$

Eingabe: Ecken p und q der Hilfstriangulierung.
(Ausgabe: geänderte Hilfstriangulierung.)

```
Strahlverfolgung von p nach q (Algorithmus 2.2 mit Startecke p* := p und
Anfragepunkt q). Dabei durchlaufene Dreiecke und überschrittene Kanten
merken.
if Strahlverfolgung trifft auf eine Ecke p_i then
    Erzeuge rekursiv die Kanten p ∧ p_i und p_i ∧ q.
else
    Durchlaufene Dreiecke und überschrittene Kanten löschen.
    Kante p ∧ q einfügen.
    Lochhälften rechts und links von Kante p ∧ q füllen (Algorithmus 3.1).
endif
```

überprüft werden, auch wenn später noch andere Datenecken miteinander verbunden werden.

Das Anlegen eines Lochs im zulässigen Gebiet erfolgt in drei Schritten: Erzeugen der Absperrung in Form von Kanten und gegebenenfalls Hilfsecken, Maximierung des eigentlichen Gebiets und Optimierung der eigentlichen Triangulierung. Die Algorithmen 3.3 und 3.4 stellen diesen Vorgang detailliert dar.

Algorithmus 3.3 erzeugt eine gewünschte Kante $p \wedge q$ in der Hilfstriangulierung. Er löscht alle Kanten und Dreiecke, die der neuen Kante im Wege stehen. In das entstehende Loch fügt er die neue Kante ein. Es entstehen zwei Lochhälften, die beide von $p \wedge q$ aus schwach kantensichtbar sind. Sie werden gemäß Satz 3.6 wieder gefüllt. Liegen auf der Strecke $p \wedge q$ weitere Ecken, dann wird sie an diesen Ecken unterteilt, und für jedes Teilstück wird eine entsprechende Kante erzeugt.

Algorithmus 3.4 Erzeugen einer Absperrung

Eingabe: Eckpunkte $r_1, \ldots, r_m$ der Absperrung (Array mit Koordinaten).
(Ausgabe: geänderte Hilfstriangulierung.)

1: **for** jeden Eckpunkt r_i **do**
2:     **if** r_i ist nicht Ecke der Hilfstriangulierung **then**
3:         Füge r_i als Hilfsecke ein.
4:     **endif**
5:     Erzeuge Kante $r_{i-1} \wedge r_i$ (bzw. $r_m \wedge r_1$) in der Hilfstriangulierung (Algorithmus 3.3).
6: **endfor**
7: **for** alle Zwischenräume Z, die an die neue Absperrung angrenzen **do**
8:     **for** jede Datenecke p am Rand von Z **do**
9:         Sichtbarkeitssuche für p (Algorithmus 3.2).
10:         **for** jede von p aus in Z strikt sichtbare Datenecke q **do**
11:             Erzeuge Kante $p \wedge q$ (Algorithmus 3.3).
12:         **endfor**
13:     **endfor**
14: **endfor**
15: Optimiere T durch iterierten Diagonalentausch.

Die Formulierung von Algorithmus 3.4 geht der Einfachheit halber davon aus, daß die eingegebene Absperrung korrekt ist. Die nötigen Korrektheitstests sind weiter oben im Text beschrieben. In Schritt 5 können im Innern der Strecke $r_{i-1} \wedge r_i$ Datenecken liegen. Dann werden mehrere Kanten erzeugt, die alle zur Absperrung gehören. Die abschließende Optimierung ist wie nach dem Löschen von Datenecken notwendig, da beim Erzeugen der Kanten k_i und beim Maximieren der eigentlichen Triangulierung die Zielfunktion keine Rolle spielt. Auch hier muß die Optimierung nur in einer bestimmten Umgebung der neuen Absperrung durchgeführt werden.

KAPITEL 4

Konvexe Hüllen auf der Sphäre

In diesem Kapitel beschäftigen wir uns mit der Berechnung konvexer Hüllen auf der Sphäre $\mathcal{S}^2$. Dabei gehen wir von einem einfachen sphärischen Polygonzug aus, dessen konvexe Hülle wir bestimmen wollen. In der Ebene ist bekannt, daß die konvexe Hülle eines einfachen Polygonzugs mit n Ecken in $O(n)$ Zeit berechnet werden kann [**Lee83a**]. Die sphärische Variante dieses Problems ist in der Literatur bisher nicht untersucht worden. In diesem Kapitel wird ein Linearzeitalgorithmus für das sphärische Problem vorgestellt. Eine Anwendung dieses Algorithmus ergibt sich bei der Flächenrekonstruktion durch die datenabhängige TAC-Triangulierung.

4.1. Konvexe Mengen und einfache Polygonzüge auf der Sphäre

Wir betrachten den euklidischen Vektorraum $\mathbb{R}^3$ und in ihn eingebettet die Einheitssphäre $\mathcal{S}^2$ um den Ursprung O. Die Punkte auf der Sphäre sind also genau die Einheitsvektoren des Raumes. Die Antipode eines Punktes $p \in \mathcal{S}^2$ ist der Punkt $-p$.

Die Geodäten auf $\mathcal{S}^2$ sind die Großkreise, d. h. die Schnitte von $\mathcal{S}^2$ mit Ebenen durch den Ursprung. Sind p und q nicht antipodal, dann gibt es einen eindeutigen Großkreis durch die beiden Punkte. Der Großkreis wird von p und q in zwei Bögen unterteilt. Die kürzeste Verbindung zwischen p und q ist der kürzere der beiden Bögen.

DEFINITION 4.1. *Eine Punktmenge $M \subseteq \mathcal{S}^2$ heißt **konvex**, wenn zu jedem Paar von nicht antipodalen Punkten $p, q \in M$ auch die kürzeste Verbindung zwischen p und q in M liegt.*

Beispiele für konvexe Mengen auf $\mathcal{S}^2$:

1. Ein Großkreis.
2. Ein Großkreisbogen mit Bogenlänge $\leq \pi$.

3. Ein Schnitt von $\mathcal{S}^2$ mit einem Halbraum des $\mathbb{R}^3$, der den Ursprung O nicht im Innern enthält. Dies sind Hemisphären (O ist Randpunkt des Halbraums), Kappen, die von Kleinkreisen berandet sind und ganz in einer Hemisphäre liegen, und einzelne Punkte.
4. Ein antipodales Punktepaar.
5. Ganz $\mathcal{S}^2$.

Eine konvexe Menge ist entweder die ganze Sphäre, oder sie ist in einer geeignet gewählten Hemisphäre enthalten. Abgesehen von den Antipodenpaaren und den Großkreisen ist jede konvexe Menge einfach zusammenhängend. Diese Tatsachen können wir mit Hilfe des folgenden Lemmas zeigen.

LEMMA 4.2. *Sei $M \subset \mathcal{S}^2$ eine konvexe Menge und $p \in \mathcal{S}^2 \setminus M$. Dann existiert eine Hemisphäre H mit $p \in H$ und $H \cap M = \emptyset$.*

Beweis: Wenn M leer ist, gilt die Aussage trivialerweise. Andernfalls gibt es einen Großkreis G durch p, der M schneidet. Liege p etwa in der Zusammenhangskomponente B von $G \setminus M$. Die Bogenlänge von B ist mindestens π, da sonst p auf der kürzesten Verbindung zwischen zwei Punkten von M läge. Es gibt daher einen Halbkreis $G' \subseteq B$ mit $p \in G'$. Seien q und $-q$ die Endpunkte von G'. Lassen wir nun den Halbkreis im $\mathbb{R}^3$ um die Gerade durch q und $-q$ rotieren, dann überstreicht er die Sphäre (außer vielleicht den Punkten q und $-q$, die nicht notwendigerweise zu G' gehören). Wir führen diese Rotation nach beiden Seiten kontinuierlich aus, bis der rotierende Halbkreis M schneidet oder bis er mit G' einen Winkel von 180° bildet. Seien G_1 und G_2 die beiden offenen Halbkreise, die am Ende dieser kontinuierlichen Rotationen erreicht werden. Das Innere der Fläche, die bei den Rotationen überstrichen wurde, ist disjunkt zu M. Der Punkt p liegt in der überstrichenen Fläche. G_1 und G_2 haben dieselben Endpunkte wie G'. Sei α der Winkel, den G_1 und G_2 auf der überstrichenen Seite einschließen. Wenn $\alpha < 180^\circ$ ist, dann enthält G_1 einen Punkt $p_1 \in M$ und G_2 einen Punkt $p_2 \in M$. Die kürzeste Verbindung zwischen p_1 und p_2 läuft durch die überstrichene Fläche und kreuzt dabei G'. Dies führt zum Widerspruch, da einerseits die kürzeste Verbindung in M liegt und andererseits $G' \cap M$ leer ist. Somit ist $\alpha \geq 180^\circ$ und die überstrichene Fläche enthält eine Hemisphäre H mit $p \in H$. ∎

BEMERKUNG 4.3. Die Hemisphäre H in Lemma 4.2 braucht weder offen noch abgeschlossen sein. Man kann M und p so wählen, daß keine offene und keine abgeschlossene Hemisphäre die Aussage des Lemmas erfüllt, z. B.: Sei M eine Hemisphäre, die von ihrem Rand genau einen Halbkreis enthält, und liege p auf dem anderen Halbkreis des Randes.

KOROLLAR 4.4. *Ist $M \neq \mathcal{S}^2$ konvex, dann ist M in einer abgeschlossenen Hemisphäre enthalten.*

KOROLLAR 4.5. *Es gibt nur vier konvexe Mengen, die einen gegebenen Großkreis G enthalten: G selbst, ganz $\mathcal{S}^2$ und die zwei durch G berandeten abgeschlossenen Hemisphären.*

Beweis: Sei H eine der beiden von G berandeten offenen Hemisphären. Wenn M einen Punkt $p \in H$ nicht enthält, dann muß M zu ganz H disjunkt sein. Hieraus folgt die Behauptung. ∎

SATZ 4.6. *Sei $M \subseteq \mathcal{S}^2$ konvex, dann ist M ein Antipodenpaar, ein Großkreis oder eine einfach zusammenhängende Menge.*

Beweis: Die leere Menge ist per definitionem einfach zusammenhängend. Ist $M \neq \emptyset$, dann gibt es drei Fälle.
I. Sei M nicht zusammenhängend. Dann gibt es zwei Punkte $p \neq q$ in M, die durch keinen stetigen Pfad in M verbunden sind. Wäre $q \neq -p$, dann läge die kürzeste Verbindung zwischen p und q in M. Enthielte M außer p und $q = -p$ noch einen dritten Punkt, dann wären p und q über ihre kürzesten Verbindungen zu diesem dritten Punkt auch miteinander verbunden. Also ist $M = \{p, -p\}$ ein Antipodenpaar.
II. Sei M mehrfach zusammenhängend. Dann existiert in M eine geschlossene Jordankurve $\mathcal{C}$, so daß in jeder der beiden durch $\mathcal{C}$ berandeten offenen Flächen mindestens ein Punkt nicht zu M gehört. Nach Lemma 4.2 liegt jeder solche Punkt p in einer Hemisphäre $H \subset \mathcal{S}^2 \setminus M$. Offensichlich sind H und $\mathcal{C}$ disjunkt. Beide durch $\mathcal{C}$ berandeten Flächen umfassen also je eine Hemisphäre. Dies kann nur der Fall sein, wenn $\mathcal{C}$ ein Großkreis ist. Die beiden durch $\mathcal{C}$ berandeten offenen Hemisphären sind disjunkt zu M, also ist $M = \mathcal{C}$ ein Großkreis.
III. Als dritter Fall bleibt nur, daß M einfach zusammenhängend ist. ∎

DEFINITION 4.7. *Die **sphärische konvexe Hülle** einer Menge $M \subseteq \mathcal{S}^2$ definieren wir wie gewöhnlich als die kleinste konvexe Menge, die M enthält. Analog zu den Symbolen $\wedge$ und $\bigwedge$ bezeichnen wir mit $\wedge_\circ$ und $\bigwedge_\circ$ konvexe Hüllen auf $\mathcal{S}^2$.*

Betrachten wir nun Polygonzüge auf $\mathcal{S}^2$. Zunächst werden einige Grundbegriffe vorgestellt, vergleiche dazu z. B. [**Ham23**] oder [**Cra62**]. In der Ebene ist ein (geschlossener) Polygonzug eine (zyklische) Folge von Strecken, also kürzesten Verbindungen, zwischen den Eckpunkten. Analog dazu werden sphärische Polygonzüge definiert. Hierbei ist zu beachten, daß es zwischen zwei Punkten auf $\mathcal{S}^2$ nur dann eine eindeutige kürzeste Verbindung gibt, wenn sie sich nicht antipodal gegenüberliegen.

DEFINITION 4.8. *Seien $p_1, \ldots, p_n \in S^2$ derart, daß jeweils p_i und p_{i+1} sowie p_n und p_1 nicht antipodal zueinander liegen. Der geschlossene* ***sphärische Polygonzug*** *mit den* ***Ecken*** *$p_1, \ldots, p_n$ besteht aus allen Großkreisbögen, die kürzeste Verbindungen zwischen p_i und p_{i+1} bzw. zwischen p_n und p_1 sind. Ein Polygonzug heißt* ***einfach,*** *wenn er frei von Selbstüberschneidungen ist. Eine abgeschlossene sphärische Fläche ω heißt* ***einfaches sphärisches Polygon****, wenn ihr Rand Ω ein einfacher Polygonzug ist.*

Wie bereits in der Definition geschehen, werden wir auch im Rest dieses Kapitels den Zusatz sphärisch häufig weglassen, wenn er sich aus dem Zusammenhang ergibt. Die Darstellung eines Polygonzugs ist nicht eindeutig, da wir degenerierte Ecken zulassen. (Die Ecke p_i heißt degeneriert, wenn p_{i-1}, p_i und p_{i+1} auf einem gemeinsamen Großkreis liegen.) Ohne degenerierte Ecken lassen sich einige sphärische Polygonzüge nicht darstellen. Die minimal notwendige Anzahl an degenerierten Ecken ist leicht anzugeben. Ein Großkreisbogen mit Bogenlänge $< \pi$ ist die kürzeste Verbindung zwischen seinen Endpunkten und somit ohne degenerierte Ecke darstellbar. Ein Bogen mit Länge $\geq \pi$ aber $< 2\pi$ benötigt eine degenerierte Ecke, die von beiden Endpunkten weniger als π entfernt sein darf. Ein voller Großkreis ist nur mit drei degenerierten Ecken als (geschlossener) Polygonzug darstellbar.

Ein geschlossener einfacher Polygonzug zerlegt die Sphäre in zwei Polygone. Durch die Reihenfolge $p_1, \ldots, p_n$ der Ecken ist der Polygonzug orientiert. Wir können daher eines der beiden Polygone als links und das andere als rechts des Polygonzugs bezeichnen.

DEFINITION 4.9. *Sei Ω ein einfacher geschlossener Polygonzug mit den Ecken $p_1, \ldots, p_n$. Der* ***Drehwinkel*** *$\alpha(p_i)$ der Ecke p_i von Ω ist der rechts von Ω liegende Winkel zwischen den zwei Polygonbögen, die mit p_i inzident sind, minus $180°$. Der* ***totale Drehwinkel*** *$\alpha(\Omega)$ von Ω ist die Summe der Drehwinkel aller Ecken.*

Bei einfachen geschlossenen Polygonzügen in der Ebene beträgt der totale Drehwinkel $360°$ oder $-360°$, je nach Orientierung. Auf der Sphäre hingegen hängt der totale Drehwinkel linear vom Flächeninhalt des rechten Polygons ab. Für $\alpha(\Omega) = 360°$ würde das rechte Polygon die ganze Sphäre bedecken (bis auf Anteile mit Flächenmaß 0), bei $\alpha(\Omega) = 0°$ sind rechtes und linkes Polygon gleich groß, und für $\alpha(\Omega) = -360°$ hätte das rechte Polygon Flächenmaß 0. Da wir einfache Polygonzüge betrachten, können die Fälle $360°$ und $-360°$ nur als Grenzbetrachtung auftreten. Sie würden degenerierte Polygonzüge erfordern.

SATZ 4.10. *Ist Ω ein einfacher geschlossener Polygonzug mit mindestens einer nicht degenerierten Ecke, dann umfaßt die konvexe Hülle von Ω eines der durch Ω berandeten Polygone.*

Beweis: Die konvexe Hülle $\bigwedge_\circ \Omega \supseteq \Omega$ ist kein Antipodenpaar und kein Großkreis. Nach Lemma 4.6 ist $\bigwedge_\circ \Omega$ einfach zusammenhängend. Da Ω selbst in der konvexen Hülle liegt, muß sie mindestens eines der durch Ω berandeten Polygone umfassen. ∎

In der Ebene ist ein Polygon genau dann konvex, wenn es nur konvexe und eventuell degenerierte, aber keine konkaven Ecken besitzt. Dasselbe gilt auf $\mathcal{S}^2$. Zum Beweis benötigen wir zunächst zwei Lemmata.

LEMMA 4.11. *Der einfache geschlossene sphärische Polygonzug Ω sei mit einer minimalen Anzahl degenerierter Ecken dargestellt. Ferner habe Ω mindestens drei nicht degenerierte Ecken. Dann ist von drei aufeinanderfolgenden Ecken höchstens eine degeneriert.*

Beweis: Zwei aufeinanderfolgende degenerierte Ecken sind nur dann nötig, wenn ein Polygonzug einen vollen Großkreis enthält. Hätte nun der einfache Polygonzug Ω zwei aufeinanderfolgende degenerierte Ecken, dann läge er vollständig auf einem Großkreis und hätte nur degenerierte Ecken. Wir brauchen also nur noch zeigen, daß von drei aufeinanderfolgenden Ecken p_{i-1}, p_i und p_{i+1} nicht die erste und letzte gleichzeitig degeneriert sein können. Nehmen wir an, daß p_{i-1} und p_{i+1} degeneriert sind. Dann müssen die beiden Großkreisbögen $B_1 = (p_{i-2} \wedge_\circ p_{i-1}) \cup (p_{i-1} \wedge_\circ p_i)$ und $B_2 = (p_i \wedge_\circ p_{i+1}) \cup (p_{i+1} \wedge_\circ p_{i+2})$ Bogenlängen $\geq \pi$ haben. Aufgrund ihres gemeinsamen Endpunkts p_i müssen sich B_1 und B_2 auch in dessen Antipode $-p_i$ schneiden. Damit ist $p_{i-2} = p_{i+2} = -p_i$ (mit zyklischer Indexrechnung), und Ω hat nur zwei nicht degenerierte Ecken. ∎

LEMMA 4.12. *Sei Ω ein einfacher geschlossener sphärischer Polygonzug, dessen Ecken alle Drehwinkel $\geq 0°$ haben, und seien p_i, p_{i+1} zwei aufeinanderfolgende Ecken von Ω. Sei H die abgeschlossene Hemisphäre links des Großkreisbogens von p_i nach p_{i+1}. Dann ist $\Omega \subset H$.*

Beweis: Induktion über die Anzahl m nicht degenerierter Ecken von Ω.
$m = 2$: Der Polygonzug besteht aus zwei Halb-Großkreisen, die sich in einem Antipodenpaar (den nicht degenerierten Ecken) schneiden. Das Polygon links von Ω ist ein sphärisches Zweieck (entspricht einem Kugelsegment, vergleiche Abbildung 4.1) mit Öffnungswinkel $\leq 180°$. Damit ist es der Schnitt von zwei Hemisphären, auf deren Rändern je einer der Halb-Großkreise liegt. Jeder Großkreisbogen zwischen zwei aufeinanderfolgenden Ecken liegt damit ebenfalls auf dem Rand einer der beiden Hemisphären, und die entsprechende Hemisphäre liegt

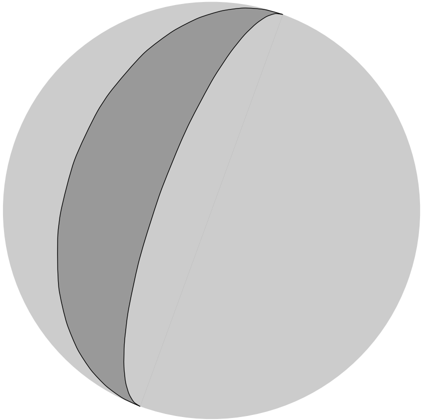


ABBILDUNG 4.1. Ein sphärisches Zweieck wird von zwei Halb-Großkreisen berandet.

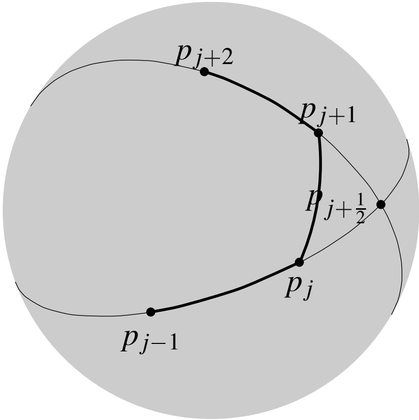


ABBILDUNG 4.2. Skizze zum Beweis von Lemma 4.12.

links des Bogens.
$m-1 \to m$: O. B. d. A. sei Ω minimal dargestellt. Aufgrund von Lemma 4.11 gibt es zwei aufeinanderfolgende, nicht degenerierte Ecken $\{p_j, p_{j+1}\} \neq \{p_i, p_{i+1}\}$. Wir erweitern das Polygon links von Ω derart durch ein sphärisches Dreieck, daß p_j und p_{j+1} degenerierte Ecken werden. Sei G_1 der Großkreis durch p_{j-1} und p_j, ebenso G_2 durch p_{j+1} und p_{j+2}. Von den zwei Schnittpunkten von G_1 mit G_2 sei $p_{j+\frac{1}{2}}$ derjenige, für den p_j auf dem kürzesten Großkreisbogen zwischen p_{j-1} und $p_{j+\frac{1}{2}}$ liegt. Es liegt dann auch p_{j+1} auf dem kürzesten Großkreisbogen zwischen p_{j+2} und $p_{j+\frac{1}{2}}$, siehe Abbildung 4.2. Wir bilden einen Polygonzug Ω', indem wir in Ω die Ecke $p_{j+\frac{1}{2}}$ zwischen p_j und p_{j+1} einfügen. Da der Bogen $p_j \wedge_\circ p_{j+1}$ kürzer als π ist, kann er die Großkreise G_1 und G_2 je nur einmal schneiden, nämlich in p_j bzw. p_{j+1}. Dies ist genau wie in Abbildung 4.2 dargestellt, und wir sehen, daß $p_{j+\frac{1}{2}}$ als Ecke von Ω' einen Drehwinkel $> 0°$ hat. In Ω' sind p_j und p_{j+1} degenerierte Ecken. Daher hat Ω' nur $m-1$ nicht degenerierte Ecken. Wegen $\{p_j, p_{j+1}\} \neq \{p_i, p_{i+1}\}$ sind p_i und p_{i+1} aufeinanderfolgende Ecken von Ω'. Nach Induktionsvoraussetzung ist Ω' in der Hemisphäre H enthalten. Aufgrund der Konstruktion ist auch Ω in H enthalten. ∎

Eine offene Hemisphäre H läßt sich durch Zentralprojektion bijektiv und geodätentreu auf eine geeignete Ebene E abbilden. Das Zentrum der Projektion ist der Ursprung O. Die Ebene E muß parallel zum Randkreis von H liegen und darf O nicht enthalten. Abbildung 4.3 zeigt, wie ein Großkreisbogen auf eine Strecke abgebildet wird. Aufgrund der Geodätentreue gehen sphärisch konvexe Teilmengen

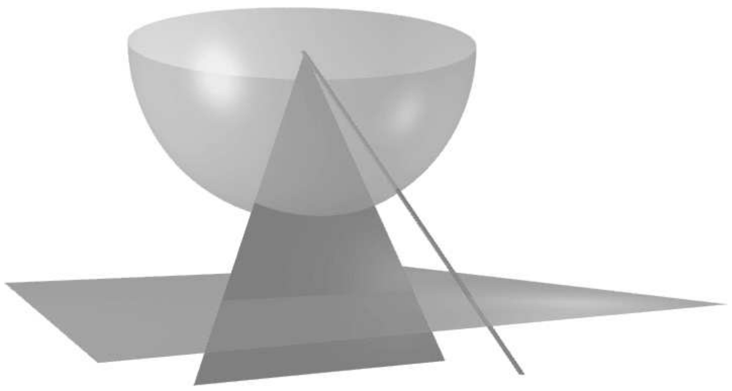


ABBILDUNG 4.3. Geodätentreue Projektion zwischen offener Hemisphäre und Ebene.

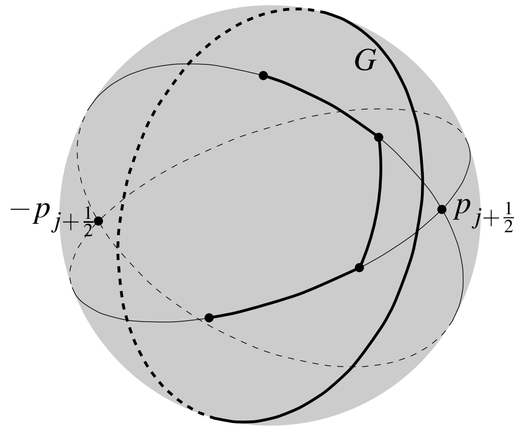


ABBILDUNG 4.4. Das Polygon aus Abbildung 4.2 liegt in einer offenen Hemisphäre.

von H in planar konvexe Teilmengen von E über und umgekehrt.

SATZ 4.13. *Sei $\Omega \subset \mathcal{S}^2$ ein einfacher geschlossener Polygonzug mit den Ecken $p_1, \ldots, p_n$ und $\alpha(p_i) \geq 0°$ für alle $i = 1, \ldots, n$. Dann ist das Polygon ω links von Ω konvex.*

Beweis: Wenn alle Ecken von Ω degeneriert sind, dann liegt Ω auf einem Großkreis und ω ist eine Hemisphäre. Wenn genau zwei Ecken nicht degeneriert sind, ist ω ein konvexes Zweieck wie in Abbildung 4.1. Sind mindestens drei Ecken nicht degeneriert, dann sehen wir aus dem Beweis von Lemma 4.12, daß ω in einem Zweieck echt enthalten ist. Insbesondere liegt die Ecke $p_{i+\frac{1}{2}}$ des Zweiecks außerhalb von ω. In Abbildung 4.4 sehen wir einen Großkreis G, der zwischen ω und $p_{i+\frac{1}{2}}$ verläuft und ω auch auf der „Rückseite" der dargestellten Sphäre nicht schneidet. Damit liegt ω in einer offenen Hemisphäre und kann geodätentreu in eine geeignete Ebene projiziert werden. Bei der Projektion ändert sich der Betrag der einzelnen Drehwinkel, aber nicht ihr Vorzeichen. Die Projektion ist, als planares Polygon ohne konkave Ecken, konvex. Daher muß auch ω sphärisch konvex sein. ∎

4.2. Die konvexe Hülle eines sphärischen Polygonzugs

Bevor wir konvexe Hüllen auf der Sphäre berechnen, betrachten wir dieses Problem zunächst auf dem Kreis. Als Modell nehmen wir einen beliebigen Großkreis $G \subset \mathcal{S}^2$. Auf ihm suchen wir die konvexe Hülle einer Punktmenge P. Den hierzu zu entwickelnden Algorithmus werden wir bei der Berechnung konvexer Hüllen auf der Sphäre mehrfach als Subroutine aufrufen.

Da eindeutige kürzeste Verbindungen zwischen Punkten von G ganz in G liegen, ist $\bigwedge_\circ P \subseteq G$. Auf der Zahlengeraden $\mathbb{R}$ ist die konvexe Hülle einer endlichen Punktmenge das kleinste Intervall, das die Punkte enthält. Auf dem Kreis sind drei Fälle zu unterscheiden:

1. Ein Antipodenpaar ist seine eigene konvexe Hülle.
2. Liegt P innerhalb eines Halbkreises von G, ohne ein Antipodenpaar zu sein, dann ist die konvexe Hülle der kürzeste Kreisbogen, der P enthält. Dies entspricht dem Intervall auf der Gerade.
3. Gibt es keinen Halbkreis, der P enthält, dann umfaßt die konvexe Hülle ganz G.

Wir formulieren den Algorithmus so, daß er nach dem Komplement der konvexen Hülle sucht. Das Komplement ist, von den trivialen Hüllen abgesehen, ein offener Kreisbogen $B \subset G$ der Länge $\geq \pi$ mit $B \cap P = \emptyset$. Die Endpunkte von B gehören zu P. Fügt man nun zu P einen weiteren Punkt p_{m+1} hinzu, dann liegt dieser Punkt entweder in der bisherigen konvexen Hülle oder in B. Im ersten Fall ändert sich die konvexe Hülle nicht. Im zweiten Fall teilt p_{m+1} den Bogen in zwei offene Teilbögen B_1 und B_2. Da die Länge von B weniger als 2π beträgt, kann nur einer der Teilbögen eine Länge $\geq \pi$ haben. Ist dies etwa B_1, dann ist sein Komplement die neue konvexe Hülle. B_2 ist Teil der konvexen Hülle, da er die kürzeste Verbindung zwischen seinen Endpunkten darstellt und diese in $P \cup \{p_{m+1}\}$ sind. Sind beide Teilbögen kürzer als π, dann ist G die neue konvexe Hülle.

Algorithmus 4.1 bestimmt das Komplement der konvexen Hülle genau auf diese inkrementelle Weise. Seine Eingabe ist nicht eine Menge, sondern eine Liste, in der Punkte mehrfach vorkommen dürfen. Um feststellen zu können, ob es

Algorithmus 4.1 Längster freier Bogen

Eingabe: Großkreis $G \subset \mathcal{S}^2$, m Punkte $p_1, \dots, p_m$ auf G (als doppelt verkettete Liste).

Ausgabe: Eine längste Zusammenhangskomponente B von $G \setminus \{p_1, \dots, p_m\}$, sofern sie nicht kürzer als π ist. Die Endpunkte q_1, q_2 von B.

1: (∗ Teilweise Dublettenentfernung: ∗)
Entferne aus der Liste $p_2, \dots, p_m$ alle Punkte, die gleich p_1 sind. Wenn Punkte $p_i = -p_1$ auftreten, lasse davon genau einen in der Liste. Die bereinigte Liste enthalte die Punkte $p'_1, \dots, p'_{m'}$.
2: **if** $m' = 2$ **and** $p'_1 = -p'_2$ **then**
3:     G hat zwei längste freie Bögen.
4:     $q_1 := p'_1$.
5:     $q_2 := p'_2$.
6: **else**
7:     **if** $p'_1 = -p'_2$ **then** vertausche p'_2 mit p'_3 **endif**
8:     $B := G \setminus (p'_1 \wedge_\circ p'_2)$
9:     **for** $i := 3, \dots, m'$ **do**
10:         **if** $p'_i \in B$ **then**
11:             Teile B an Punkt p'_i in zwei Teilbögen.
12:             B := der längere der beiden Teilbögen.
13:             **if** B ist kürzer als π **then**
14:                 Es gibt keinen freien Bogen mit Länge $\geq \pi$. **stop**
15:             **endif**
16:         **endif**
17:     **endfor**
18:     q_1, q_2 := Endpunkte von B.
19: **endif**

sich letztendlich nur um ein Antipodenpaar handelt, entfernt der Algorithmus in Schritt 1 alle Dubletten von p_1 und gegebenenfalls von $-p_1$. Bleiben dabei mehr als zwei Punkte übrig, dann sorgt Schritt 7 dafür, daß $p'_2 \neq -p'_1$ ist. Die konvexe Hülle der ersten beiden Punkte ist jetzt ein Kreisbogen. B wird als Komplement dieses Kreisbogens initialisiert. Danach führt die **for**-Schleife (Schritte 9–17) die inkrementelle Konstruktion durch.

In Schritt 1 werden nur Dubletten von p_1 und von $-p_1$ entfernt. Hierzu testet der Algorithmus $p_2, \dots, p_m$ auf Gleichheit mit p_1 oder $-p_1$. Der Zeitbedarf ist $O(m)$. Alle übrigen Einzelschritte benötigen konstante Zeit, und die **for**-Schleife wird $O(m)$ mal durchlaufen. Die Laufzeit von Algorithmus 4.1 ist daher $O(m)$.

Wenden wir uns nun den konvexen Hüllen auf der Sphäre zu. Die bereits erwähnte geodätentreue Projektion zwischen offener Hemisphäre und Ebene (siehe Abbildung 4.3) ist eine wichtige Grundlage unseres Algorithmus. Unter der geodätentreuen Projektion bleibt der Wahrheitswert von Aussagen der Form „Punkt p_i liegt rechts (links, auf) der Geodäte durch die Punkte p_j und p_k” erhalten. Ist also ein einfacher geschlossener sphärischer Polygonzug Ω in einer *offenen* Hemisphäre enthalten, und kennen wir diese Heimsphäre, dann können wir die konvexe Hülle von Ω auf dem Umweg über eine geeignete Ebene berechnen.

Zur Berechnung von konvexen Hüllen in der Ebene dient uns der Algorithmus von Lee [**Lee83a**]. Dieser Algorithmus benutzt die Koordinaten der Polygonecken lediglich für drei Teilprobleme:

1. Zur Bestimmung der Umlaufrichtung (im/gegen Uhrzeigersinn) von Ω, sofern diese nicht von vornherein feststeht.
2. Zur Bestimmung einer Polygonecke p, die garantiert eine Ecke der konvexen Hülle ist.
3. Zum Beantworten von Anfragen „Liegt Punkt p_i rechts, links oder auf der Gerade durch die Punkte p_j und p_k?”

Wie oben bereits angedeutet, läßt sich Teilproblem 3 auch ohne Projektion schon auf $\mathcal{S}^2$ lösen. Dazu brauchen wir nur das Vorzeichen der Determinante der drei Einheitsvektoren p_i, p_j und p_k betrachten. Wenn es uns gelingt, die anderen Teilprobleme ohne Projektion zu lösen, dann können wir den Algorithmus von Lee direkt auf $\mathcal{S}^2$ anwenden.

Wir lösen die beiden Teilprobleme nicht, sondern umgehen sie, indem wir die Sphäre in zwei Hemisphären zerschneiden. Dabei konstruieren wir aus Ω zwei „fast einfache” Polygonzüge Ω_1 und Ω_2, die jeweils in einer der beiden Hemisphären liegen. Ω_1 und Ω_2 sind nicht notwendigerweise eine Zerlegung von Ω, aber es muß $\bigwedge_\circ (\Omega_1 \cup \Omega_2) = \bigwedge_\circ \Omega$ gelten. Der Algorithmus von Lee liefert uns die konvexen Hüllen von Ω_1 und Ω_2. Aus diesen wiederum berechnen wir die konvexe Hülle von Ω. Abbildung 4.5 skizziert dieses Vorgehen.

Das Zerschneiden leistet Algorithmus 4.2. Er erkennt auch verschiedene Spezialfälle, in denen er direkt die konvexe Hülle ausgibt. Einige Schritte des Algorithmus werden im folgenden näher erläutert.
Die Annahme, daß das linke Polygon den kleineren Flächeninhalt hat, läßt sich nötigenfalls durch Umorientieren erfüllen. Der Zeitaufwand hierfür ist linear. Wenn die konvexe Hülle ein Polygon ist, enthält sie stets das kleinere der beiden durch Ω berandeten Polygone.
Schritt 9: Wenn es auf dem Schnittkreis G keinen (offenen) Bogen B mit Länge

Algorithmus 4.2 Zerschneiden eines sphärischen Polygonzugs

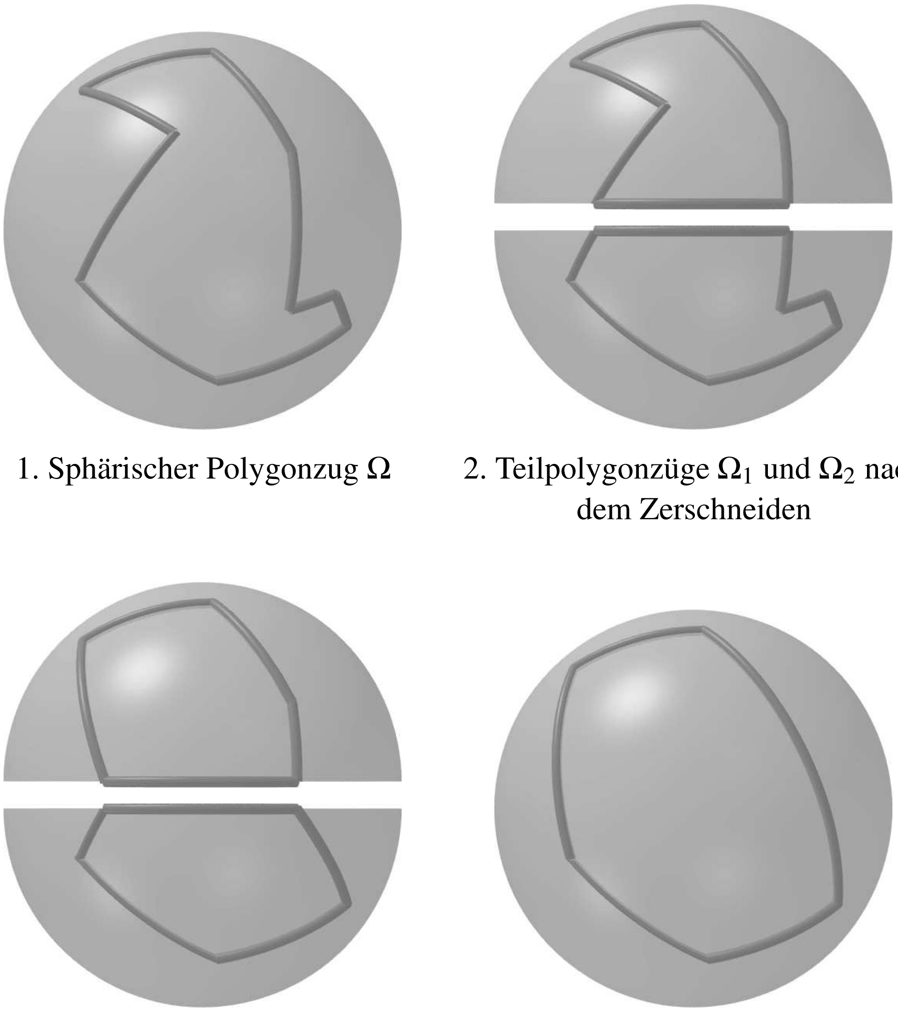
Eingabe: Ecken $p_1, \ldots, p_n$ eines einfachen geschlossenen Polygonzugs $\Omega \subset \mathcal{S}^2$ (als doppelt verkettete Liste).

Ausgabe im Normalfall: Geschlossene „Teil-"Polygonzüge Ω_1 und Ω_2, die jeweils in einer offenen Hemisphäre liegen. Es gilt $\bigwedge_\circ(\Omega_1 \cup \Omega_2) = \bigwedge_\circ \Omega$.

Ausgabe in Spezialfällen: Konvexe Hülle von Ω.

Annahme: O.B.d.A. sei Ω so orientiert, daß das linke Polygon keinen größeren Flächeninhalt hat als das rechte Polygon.

G := mittelsenkrechter Großkreis von p_1 und p_2.
for $i := 1, \ldots, n$ **do**
  if p_i und p_{i+1} liegen auf verschiedenen Seiten von G **then**
    Füge den Schnittpunkt $G \cap (p_i \wedge_\circ p_{i+1})$ als degenerierte Ecke in Ω ein.
  endif
endfor
Suche mit Algorithmus 4.1 einen längsten freien Bogen $B \subset G \setminus \Omega$.
if Algorithmus 4.1 findet keinen freien Bogen mit Länge $\geq \pi$ **then**
  $\bigwedge_\circ \Omega = \mathcal{S}^2$.
elseif B ist ein (offener) Halbkreis **then**
  Betrachte die Endpunkte q_1 und $q_2 = -q_1$ von B als Nord– und Südpol. Bestimme die Längengrade der übrigen Ecken von Ω. Suche analog zu Algorithmus 4.1 einen Winkelbereich von mindestens 180°, in dem keiner der Längengrade liegt.
  if alle Ecken liegen auf einem Paar gegenüberliegender Längengrade **then**
    die beiden Längengrade bilden einen Großkreis, und dieser ist $\bigwedge_\circ \Omega$.
  elseif ein freier Winkelbereich von $\geq 180°$ existiert **then**
    $\bigwedge_\circ \Omega$ ist das Komplement des freien Winkelbereichs.
  else
    $\bigwedge_\circ \Omega = \mathcal{S}^2$.
  endif
else
  Durchlaufe Ω von q_1 bis q_2. Bilde den geschlossenen Polygonzug Ω_1 aus denjenigen Ecken, die auf oder rechts von G liegen.
  Durchlaufe Ω von q_2 bis q_1. Bilde den geschlossenen Polygonzug Ω_2 aus denjenigen Ecken, die auf oder links von G liegen.
endif



1. Sphärischer Polygonzug Ω

2. Teilpolygonzüge Ω_1 und Ω_2 nach dem Zerschneiden

3. Konvexe Hüllen von Ω_1 und Ω_2

4. Konvexe Hülle von Ω

ABBILDUNG 4.5. Konstruktion der konvexen Hülle eines sphärischen Polygonzugs.

$\geq \pi$ gibt, der Ω nicht schneidet, dann gehört ganz G zur konvexen Hülle. Die Ecken p_1 und p_2 von Ω liegen strikt auf beiden Seiten von G. Nach Korollar 4.5 ist $\bigwedge_\circ \Omega = \mathcal{S}^2$.

Schritt 11: Wir bestimmen den freien Winkelbereich, indem wir die Längengrade mit dem Äquator schneiden und dann auf dem Äquator einen von Schnittpunkten freien Bogen suchen.

Schritt 15: Der freie Winkelbereich wird von zwei Längengraden berandet, also von zwei Halb-Großkreisbögen. Sein Komplement $\bigwedge_\circ \Omega$ ist daher entweder ein

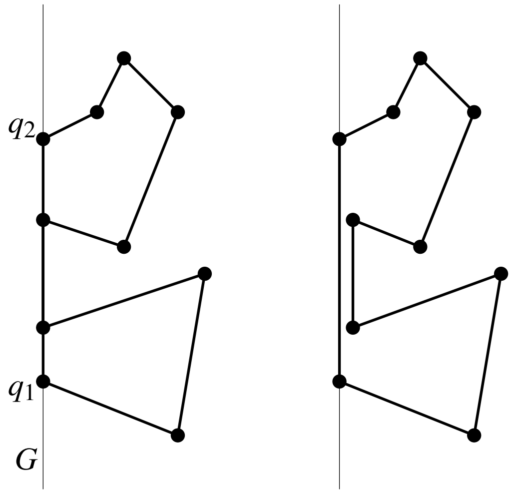


ABBILDUNG 4.6. Durch eine kleine Störung wird Ω_1 einfach, ohne die konvexe Hülle zu ändern.

sphärisches Zweieck mit q_1 und q_2 als nicht degenerierten Ecken, oder eine Hemisphäre.
Schritt 20 (und analog Schritt 21): Alle Bögen von Ω, die G kreuzen, wurden in Schritt 4 durch ihre Schnittpunkte mit G unterteilt. Daher ersetzt Schritt 20 genau diejenigen Teile von Ω, die links von G liegen, durch Bögen auf G. Dabei entstehen in Ω_1 Selbstberührungen, aber keine Selbstüberkreuzungen. Durch eine beliebig kleine Störung würde Ω_1 einfach, siehe Abbildung 4.6. Die Störung ändert die konvexe Hülle von Ω_1 nicht. Der Algorithmus von Lee berechnet die korrekte konvexe Hülle, ohne die Selbstberührung überhaupt zu erkennen.

Algorithmus 4.2 bestimmt entweder die konvexe Hülle von Ω, oder er liefert zwei Polygonzüge Ω_1 und Ω_2. Beide Polygonzüge sind „fast einfach": sie haben keine Selbstüberkreuzungen, höchstens Selbstberührungen. Durch geringfügige Rotationen von G um eine geeignete Achse im $\mathbb{R}^3$ sieht man, daß Ω_1 und Ω_2 jeweils in einer offenen Hemisphäre liegen. Die Ecken q_1 und q_2 sind Eckpunkte der jeweiligen konvexen Hülle. Darüberhinaus liegt der Großkreisbogen zwischen q_1 und q_2 auf dem jeweiligen Rand der konvexen Hüllen. Das Polygon links von Ω_1 (von Ω_2) ist in $\bigwedge_\circ \Omega_1$ (in $\bigwedge_\circ \Omega_2$) enthalten.

Wir können nun den Algorithmus von Lee, mit geringfügigen Modifikationen, auf jeden der beiden Polygonzüge Ω_1 und Ω_2 einzeln anwenden. Dieser Algorithmus durchläuft einen Polygonzug Ecke für Ecke. Er beginnt mit einer Ecke, die garantiert Eckpunkt der konvexen Hülle ist, und konstruiert in jedem Schritt die konvexe Hülle der bisher betrachteten Ecken. Statt einer einzelnen Ecke, die garantiert zum Rand der konvexen Hülle gehört, kennen wir bereits einen Großkreisbogen mit dieser Eigenschaft, nämlich $q_1 \wedge_\circ q_2 = G \setminus B$. Der Algorithmus von Lee benutzt die garantierte Ecke an zwei Stellen. Einmal als Startecke beim Durchlaufen des Polygonzugs, mit der Gewißheit, daß bestimmte „Rückwärtsschritte" nie über diese Ecke zurück laufen. Diese Rolle übernimmt in unserem Fall q_1 für Ω_1 und q_2 für Ω_2. Zum anderen blickt der Algorithmus von Lee auch vorwärts auf die garantierte Ecke, um festzustellen, ob eine neu betrachtete Ecke des Polygonzugs bereits in der bisher konstruierten Hülle liegt. Für diese „Vorausschau" verwenden wir nicht die Startecke, sondern den jeweils anderen Endpunkt des garantierten Bogens $G \setminus B$. Damit konstruiert der modifizierte Algorithmus in jedem Schritt die konvexe Hülle der betrachteten Ecken und des garantierten Bogens.

Wir erhalten zwei konvexe Polygone, deren Schnitt genau der Bogen $G \setminus B$ ist. (Genau genommen erhalten wir die Ränder dieser Polygone in Form von Polygonzügen.) Wir vereinigen diese beiden Polygone, indem wir aus beiden Polygonzügen den Bogen $G \setminus B$ entfernen und die nun offenen Polygonzüge in den Ecken q_1 und q_2 miteinander verketten. Der entstehende Polygonzug Ω^* schließt eine sternförmige Fläche ein: der Mittelpunkt c des Bogens $q_1 \wedge_\circ q_2 = G \setminus B$ gehört zu den beiden konvexen Polygonen, daher sind von ihm aus alle Punkte von Ω^* sichtbar.

DEFINITION 4.14. *Ein sphärisches Polygon ω heißt **sternförmig**, wenn es einen Punkt $q \in \omega$ gibt, so daß für jedes $p \in \omega$ alle kürzesten Verbindungen zwischen q und p in ω liegen. Die Menge aller q mit dieser Eigenschaft heißt **Kern** von ω.*

Das Polygon links von Ω^* ist nach den obigen Überlegungen sternförmig. Für sternförmige Polygone in der Ebene berechnet der sogenannte Graham's Scan (siehe [**Gra72**] oder auch [**PS85**, S. 106 ff.]) die konvexe Hülle. Er nutzt die Tatsache aus, daß man von einem planaren sternförmigen Polygon eine konkave Ecke „abschneiden" kann, ohne die Sternförmigkeit zu verlieren. Das Abschneiden einer konkaven Ecke p_i bedeutet, daß das Polygon um die Dreiecksfläche $p_{i-1} \wedge p_i \wedge p_{i+1}$ erweitert wird. Der Begriff Abschneiden bezieht sich genau genommen auf den Randpolygonzug Ω von ω. In ihm wird die Ecke p_i entfernt und die Kanten $p_{i-1} \wedge p_i$ und $p_i \wedge p_{i+1}$ werden durch $p_{i-1} \wedge p_{i+1}$ ersetzt. Der Graham's Scan durchläuft Ω und schneidet alle konkaven Ecken ab. Durch geeignetes Kombinieren von Vorwärts– und Rückwärtsschritten findet er auch die Ecken, die erst durch Abschneiden einer anderen Ecke konkav werden, in linearer Gesamtlaufzeit.

Auf der Sphäre läßt sich nicht jede konkave Ecke p_i eines sternförmigen Polygons ω ohne weiteres abschneiden. Nimmt man p_i aus dem Randpolygonzug heraus, dann können Selbstüberschneidungen auftreten. Dies wird plausibel, wenn wir das komplementäre Polygon

$$\omega^c := \overline{\mathcal{S}^2 \setminus \omega}$$

betrachten. Für ω^c ist p_i eine konvexe Ecke, deren Abschneiden auch in der Ebene zu Selbstüberschneidungen führen kann.

Wir werden sehen, daß sich Selbstüberschneidungen durch einen einfachen Test ausschließen lassen. Neben Resultaten aus Kapitel 3 benötigen wir dazu den folgenden Satz.

SATZ 4.15. *Sei $\Omega \subset \mathcal{S}^2$ ein einfacher geschlossener Polygonzug. Das Polygon ω links von Ω ist genau dann sternförmig, wenn das Polygon ω^c rechts von Ω sternförmig ist.*

Beweis: Sei ω sternförmig und liege p in seinem Kern. Läge die Antipode $-p$ auch in ω, dann wäre $\omega = \mathcal{S}^2$ aufgrund von Definition 4.14. Wir werden zeigen, daß $-p$ im Kern von ω^c liegt. Sei $q \in \omega^c$ beliebig. Wir betrachten den Halb-Großkreis $B := p \wedge_\circ q \wedge_\circ -p$. Da p im Kern von ω liegt, ist $\omega \cap B$ zusammenhängend. Damit ist auch $B' := B \cap \omega^c$ zusammenhängend. Wir sehen, daß $q \wedge_\circ -p \subset B'$ gelten muß, d. h. $-p$ liegt im Kern von ω^c. Insbesondere ist ω^c sternförmig. Der Umkehrschluß ergibt sich dadurch, daß wir den Polygonzug umorientieren und so links und rechts vertauschen. ∎

In Kapitel 3 haben wir gesehen, daß man von einem sternförmigen Polygon in der Ebene auch konvexe Ecken abschneiden kann, solange dabei der Kern des Polygons nicht vollständig abgeschnitten wird. Die Sternförmigkeit bleibt dabei erhalten. Bei näherer Betrachtung von Lemma 3.4, Algorithmus 3.1 und Satz 3.5 zeigt sich, daß diese auf $\mathcal{S}^2$ übertragbar sind. Wir können also von einem sphärischen sternförmigen Polygon eine konkave Ecke abschneiden, wenn danach der Kern des komplementären Polygons nicht leer ist.

Zu Beginn unseres Graham's Scan wissen wir, daß der Großkreisbogen $q_1 \wedge_\circ q_2$ zum Kern des Polygons links von Ω^* gehört. Nach Satz 4.15 gehört der antipodale Bogen $-q_1 \wedge_\circ -q_2$ zum Kern des Polygons rechts von Ω^*. Beide Bögen liegen auf dem Großkreis G, der in Schritt 1 von Algorithmus 4.2 gebildet wird. Während des Graham's Scan betrachten wir ständig die Schnittpunkte von G mit dem aktuellen Polygonzug. Wenn der Graham's Scan etwa die Ecke q_1 abschneidet, dann teilen wir den neu eingefügten Bogen an G auf. Wir fügen den Schnittpunkt des neuen Bogens mit G als zusätzliche, degenerierte Ecke q' in den Polygonzug ein.

Wird später q' abgeschnitten, dann verfahren wir analog, ebenso für q_2. Auf diese Weise kennen wir stets die beiden Schnittpunkte des aktuellen Polygonzugs mit G. Diese Schnittpunkte teilen G in zwei Bögen. Der Bogen links des aktuellen Polygonzugs überdeckt $q_1 \wedge_\circ q_2$. Solange er nicht länger als π ist, kann er nicht gleichzeitig $-q_1 \wedge_\circ -q_2$ überdecken. Wir wissen dann, daß der Kern des (komplementären) Polygons nicht leer ist und der Graham's Scan korrekt arbeitet. Wird der Graham's Scan beendet, ohne daß der linke Teil von G länger als π wird, dann gibt er den Randpolygonzug der konvexen Hülle aus. Wird der linke Teil von G länger als π, dann gehört ganz G zur konvexen Hülle. Wie in Schritt 9 von Algorithmus 4.2 können wir jetzt schließen, daß die konvexe Hülle gleich $\mathcal{S}^2$ ist (vergleiche die Erläuterung auf Seite 62). Der Zeitaufwand unseres modifizierten Graham's Scan bleibt $O(n)$.

Algorithmus 4.3 Konvexe Hülle eines sphärischen Polygonzugs durch Zerschneiden

Eingabe: Ecken $p_1, \dots, p_n$ eines einfachen geschlossenen Polygonzugs $\Omega \subset \mathcal{S}^2$ (als doppelt verkettete Liste).

1: Zerschneide Ω nach Algorithmus 4.2 in Ω_1 und Ω_2.
2: **if** Algorithmus 4.2 hat bereits $\bigwedge_\circ \Omega$ bestimmt **then stop endif**
3: Führe modifizierten Algorithmus von Lee auf Ω_1 aus, mit q_1 und q_2 als garantierten Ecken der konvexen Hülle.
4: Führe modifizierten Algorithmus von Lee auf Ω_2 aus, mit q_2 und q_1 als garantierten Ecken der konvexen Hülle.
5: Bilde aus den Randpolygonzügen von $\bigwedge_\circ \Omega_1$ und $\bigwedge_\circ \Omega_2$ den Randpolygonzug Ω^* von $\bigwedge_\circ \Omega_1 \cup \bigwedge_\circ \Omega_2$.
6: Führe auf Ω^* den modifizierten Graham's Scan aus.

Algorithmus 4.3 zeigt den gesamten Ablauf unseres Algorithmus. Da alle aufgerufenen Algorithmen im schlechtesten Fall lineare Laufzeit haben, hat auch Algorithmus 4.3 Laufzeit $O(n)$.

4.3. Anwendung: Totale Absolutkrümmung von Dreiecksnetzen

Wie wir bereits in Kapitel 1 gesehen haben, eignen sich verschiedene Triangulierungen unterschiedlich gut für die Konstruktion von Dreiecksnetzen. Üblicherweise mißt man die Güte einer Triangulierung durch eine Zielfunktion. Die Wahl dieser Zielfunktion hängt von dem jeweiligen Anwendungsgebiet ab. Für Finite-Elemente-Berechnungen etwa gilt es, schmale Dreiecke zu vermeiden, da diese die numerische Stabilität des Verfahrens verschlechtern. Hier erreicht man das Optimum durch die Delauneytriangulierung. Will man Vektorfelder rekonstruieren, dann ist eine möglichst geringe Anzahl kritischer Punkte wünschenswert [**SH98**]. Dies ist eine sogenannte *datenabhängige* Zielfunktion. Ihr Wert hängt nicht nur von der Triangulierung, sondern vielmehr von der zugehörigen Rekonstruktion — in diesem Fall ein stückweise lineares Vektorfeld — ab.

Zur Flächenrekonstruktion eignen sich in besonderem Maße die datenabhängigen Zielfunktionen ABN und TAC. Sie werden auf dem Dreiecksnetz ausgewertet, das sich aus der Triangulierung ergibt. ABN (Angles Between Normals, vgl. [**CSYL88, DLR89, DLR90**]) minimiert die Winkel zwischen den Normalen benachbarter Dreiecke. Der ABN-Idealfall ist ein planares Dreiecksnetz. TAC (Total Absolute Curvature, vgl. [**AvD95, vDA95**]) minimiert die totale Absolutkrümmung des Dreiecksnetzes. Auf glatten Flächen ist die totale Absolutkrümmung als Integral des Betrags der Gaußkrümmung über die gesamte Fläche definiert. Auf polygonalen Flächen konzentriert sie sich in den Eckpunkten. Dieses Kriterium hat die Eigenschaft, daß es in konvexen und konkaven Flächenbereichen wieder konvexe und konkave Dreiecksnetze erzeugt.

In Kapitel 1 haben wir Rekonstruktionen einer Fläche durch verschiedene Triangulierungen gesehen. Die Abbildungsserie 4.7–4.10 zeigt als weiteres Beispiel eine Fläche mit einem Knick. An dem Knick weisen die Triangulierungen deutliche Unterschiede auf. Die Delaunay-Triangulierung läßt den Knick kaum erkennen. Das ABN-Dreiecksnetz nähert sich dem Knick schon besser, aber noch nicht ideal. Die beste Rekonstruktion stellt das TAC-Netz dar. Hier ist ist der Knick vollständig herausgearbeitet.

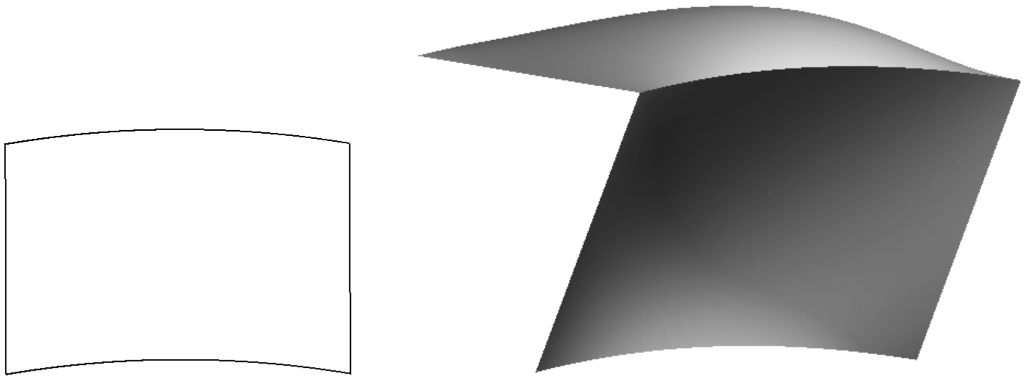


ABBILDUNG 4.7. Fläche mit Knick. Links der Umriß nach Projektion in die Triangulierungsebene.

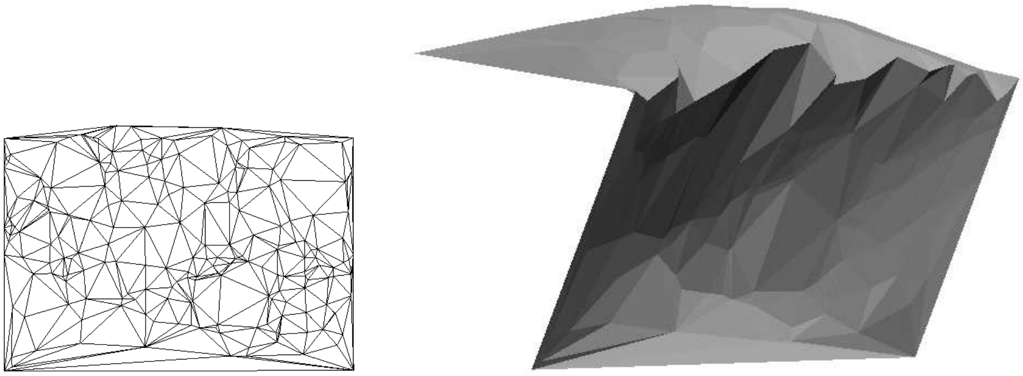


ABBILDUNG 4.8. Delaunay-Triangulierung des Knicks.

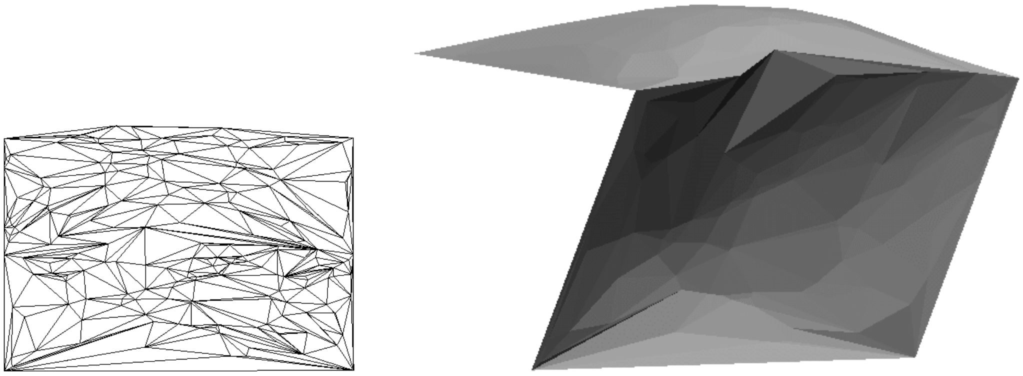


ABBILDUNG 4.9. ABN-Triangulierung des Knicks.

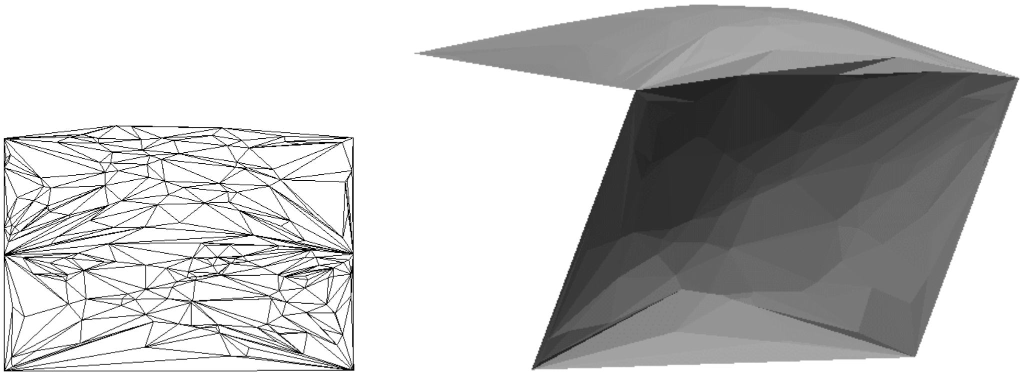


ABBILDUNG 4.10. TAC-Triangulierung des Knicks.

Für die Berechnung der totalen Absolutkrümmung eines Dreiecksnetzes benötigt man sphärische konvexe Hüllen. Die totale Absolutkrümmung wird in jedem Eckpunkt p_i des Dreiecksnetzes berechnet und hängt von den inzidenten Dreiecken ab. Im wesentlichen wird sie aus denjenigen Innenwinkeln der Dreiecke berechnet, in deren Scheitel p_i liegt. Dabei müssen jedoch verschiedene Fälle unterschieden werden. Die Fälle lassen sich erkennen, wenn man eine genügend kleine Sphäre um die betrachtete Ecke legt und mit den inzidenten Dreiecken schneidet. Die Sphäre muß so klein sein, daß alle anderen Ecken außerhalb liegen. Ihre Schnitte mit den inzidenten Dreiecken bilden dann einen einfachen geschlossenen sphärischen Polygonzug. Die Berechnung der totalen Absolutkrümmung bestimmt sich danach, ob der Polygonzug

(1) konvex,
(2) nicht konvex, aber in einer Hemisphäre enthalten, oder
(3) nicht in einer Hemisphäre enthalten

ist. Im Fall (2) wird zur Berechnung die konvexe Hülle benötigt. Die Unterscheidung der Fälle kann ebenfalls mit Hilfe der konvexen Hülle erfolgen. Dabei ist Fall (3) daran zu erkennen, daß die konvexe Hülle die gesamte Sphäre ist. Alboul und van Damme verwenden einen Algorithmus mit Laufzeit $O(n^3)$. Die Laufzeit beruht hauptsächlich darauf, daß für je zwei Ecken p_i, p_j des Polygonzugs getestet wird, ob alle übrigen Ecken auf derselben Seite des Großkreises durch p_i und p_j liegen [**Alb96**]. Einen Ansatz, der ebenfalls auf dem Seitentest basiert, realisiert Algorithmus 4.4. (Möglicherweise ist dies genau der Ansatz, den auch Alboul und van Damme verwenden.) Ist ein Paar von Ecken gefunden, so daß alle anderen Ecken auf einer Seite des Großkreises liegen, dann sind diese zwei Ecken garantiert auch Ecken der konvexen Hülle und können als Start– und Endpunkt für den Algorithmus von Lee dienen.

Algorithmus 4.3 und 4.4 wurden implementiert und die Laufzeiten miteinander verglichen. Grundlage des Vergleichs waren Daten aus der tatsächlichen Berechnung der totalen Absolutkrümmung im Zuge der Triangulierungsoptimierung. Es zeigte sich, daß Algorithmus 4.4 bis zu $n = 7$ Ecken schneller ist. Ab $n = 8$ Ecken kommt die schnellere asymptotische Laufzeit von Algorithmus 4.3 zum Tragen.

Algorithmus 4.4 Konvexe Hülle eines sphärischen Polygonzugs durch Seitentest

Eingabe: Ecken $p_1, \dots, p_n$ eines einfachen geschlossenen Polygonzugs $\Omega \subset \mathcal{S}^2$ (als Array).

Annahme: Keine drei Ecken liegen auf einem gemeinsamen Großkreis.

for $i = 1, \dots, n-1$ **do**
    for $j = i+1, \dots, n$ **do**
        G := Großkreis durch p_i und p_j
        if alle Ecken $p_k \in \{p_1, \dots p_n\} \setminus \{p_i, p_j\}$ liegen auf einer Seite von G **then**
            Führe modifizierten Algorithmus von Lee aus, mit p_i und p_j als garantierten Ecken der konvexen Hülle. **stop**
        endif
    endfor
endfor
$\bigwedge_\circ \Omega = \mathcal{S}^2$.

KAPITEL 5

Stabilität in Delaunaydiagrammen

Dieses Kapitel betrachtet das Verhalten von Delaunaydiagrammen im $\mathbb{R}^2$, wenn die zugrundeliegende Punktmenge z. B. durch Meß– oder Rundungsfehler gestört wird. Gegeben seien eine „exakte“ und eine „fehlerbehaftete“ Punktmenge $P = \{p_1, \dots, p_n\}$ und $P' = \{p'_1, \dots, p'_n\}$. Die Fehler, d. h. die Abstände von p'_i zu p_i, seien nach oben durch ε beschränkt. Es ist offensichtlich, daß die beiden Delaunaydiagramme D von P und D' von P' nicht notwendigerweise in ihrer Struktur übereinstimmen. Für bestimmte Kanten von D kann man jedoch garantieren, daß auch D' die entsprechende Kante enthalten muß. Eine solche Kante nennen wir ε-stabil. Neben diesem Entscheidungsproblem betrachten wir auch das Optimierungsproblem. Hierbei wollen wir zu einer gegebenen Kante wissen, wie groß der Fehler sein muß, um die Kante zu gefährden. Das Kapitel zeigt anhand einer neuen Charakterisierung von Stabilität notwendige und hinreichende Bedingungen für ε-Stabilität auf und stellt Algorithmen für das Entscheidungs– und das Optimierungsproblem vor.

5.1. Stabilität

Die folgenden beiden Definitionen führen den in diesem Kapitel verwendeten Stabilitätsbegriff ein:

DEFINITION 5.1. *Sei $P = \{p_1, \dots, p_n\}$ eine Punktmenge in der Ebene. Eine **ε-Störung** von P ist eine Punktmenge $P' \subset \mathbb{R}^2$ mit einer surjektiven Abbildung $' : P \to P' : p_i \mapsto p'_i$, so daß* $\mathrm{dist}(p_i, p'_i) \leq \varepsilon$ *für $1 \leq i \leq n$ gilt.*

DEFINITION 5.2. *Sei $k = p_i \wedge p_j$ eine Kante im Delaunaydiagramm von P. Wir bezeichnen k als **ε-stabil**, wenn im Delaunaydiagramm jeder ε-Störung P' von P die korrespondierende Kante $p'_i \wedge p'_j$ vorkommt. Andernfalls heißt k **ε-instabil**. Die **Stabilität** von k ist das Supremum über alle ε, für die k ε-stabil ist.*

Es kann passieren, daß ε größer oder gleich dem halben Abstand $1/2\,\mathrm{dist}(p_i, p_j)$ zweier Punkte ist. Dann gibt es ε-Störungen P' mit $p'_i = p'_j$. Das Delaunaydiagramm eines solchen P' enthält keine Kante $p'_i \wedge p'_j$. Die Kante $p_i \wedge p_j$ ist also ε-instabil. Nicht ganz so klar ist der Fall für Kanten der Form $p_i \wedge p_\ell$ und $p_j \wedge p_\ell$,

$\ell \neq i, j$. Soll man hier eine Kante zwischen p'_ℓ und $p'_i = p'_j$ als korrespondierend ansehen oder nicht? Wir wollen in einem solchen Fall alle Kanten, die mit p_i oder mit p_j inzident sind, als instabil betrachten.

Salesin [**Sal91**] untersucht geometrische Berechnungen unter dem Aspekt von Rundungsfehlern. Er verwendet ε-Störungen als mathematisches Werkzeug, um die numerischen Fehler abzuschätzen. Das Hauptaugenmerk liegt bei Salesin darauf, ob zu der tatsächlichen Eingabe eine ε-Störung existiert, für die das Berechnungsergebnis korrekt wäre. Eine typische Fragestellung ist z. B. „Sind drei gegebene Punkte bis auf eine ε-Störung kollinear?" Daneben betrachtet Salesin auch Stabilität im hier verwendeten Sinn, insbesondere für konvexe Polygone. Er führt aus [**Sal91**, S. 7–9], daß diese beiden Betrachtungsweisen einander diametral entgegengesetzt sind: Sei M die Menge aller Eingaben, die eine bestimmte Eigenschaft erfüllen. Weitet man, gemäß einer geeigneten Metrik auf dem Raum der sinnvollen Eingaben, die Menge M um ε aus, dann erhält man diejenigen Eingaben, die die Eigenschaft bis auf ε-Störungen erfüllen. Zieht man hingegen M um ε zusammen, dann besitzen die verbleibenden Eingaben die Eigenschaft mit ε-Stabilität.

Ramos [**Ram96**] zeigt folgende Kriterien für die Stabilität einer Delaunaykante $p_i \wedge p_j$ auf.

1. Existiert ein Punkt $p_k \in P \setminus \{p_i, p_j\}$ mit Abstand kleiner oder gleich 2ε von der Kante, dann ist die Kante ε-instabil.
2. Existieren zwei Punkte $p_k, p_\ell \in P \setminus \{p_i, p_j\}$, so daß der schmalste Ring, der p_i, p_j, p_k und p_ℓ enthält, nicht breiter als 2ε ist, dann ist die Kante ε-instabil.
3. Treffen weder 1 noch 2 zu, dann ist die Kante ε-stabil.
4. Es genügt, diejenigen Punkte p_k und p_ℓ zu betrachten, die zu p_i oder p_j benachbart sind.

Damit läßt sich die Stabilität einer einzelnen Kante als Minimum der Abstände nach 1 und der Ringbreiten nach 2 berechnen. Bei m betrachteten Nachbarpunkten ergeben sich $\theta(m^2)$ Paare für die Ringbetrachtung, so daß man $\theta(m^2)$ Zeit braucht. Die vorliegende Arbeit setzt an dieser Stelle an. In Abschnitt 5.2 wird eine neue Charakterisierung der Stabilität eingeführt, die in Abschnitt 5.3 zu einem Entscheidungsalgorithmus mit Laufzeit $O(m)$ und in Abschnitt 5.4 zu einem Optimierungsalgorithmus mit Laufzeit $O(m \log m)$ führt.

Im schlechtesten Fall kann $m = n - 2$ sein. Hier bietet Ramos eine Beschleunigungsmöglichkeit an, die das Voronoidiagramm dritter und vierter Ordnung ausnutzt. Die Punkte p_k und p_ℓ (bzw. der einzelne Punkt p_k), die für die Stabilität bestimmend sind, haben mit p_i und p_j zusammen ein nicht leeres Voronoigebiet vierter (dritter) Ordnung. Das Voronoidiagramm vierter (dritter) Ordnung kann in $O(n \log n)$ Zeit berechnet werden und besteht aus $O(n)$ Gebieten, die Stabilitätsbestimmung benötigt damit $O(n \log n)$ Zeit. Mit diesem Ansatz kann die Stabilität nicht nur einer, sondern aller Kanten in $O(n \log n)$ Zeit berechnet werden.

Abellanas, Hurtado und Ramos [**AHR95**], auch [**Ram96**] betrachten die Stabilität einer Delaunaytriangulierung als Ganzes. Offensichtlich ist diese durch das Minimum der einzelnen Kantenstabilitäten beschränkt. Jetzt genügt es, die beiden Punkte p_k und p_ℓ zu betrachten, die sowohl mit p_i als auch mit p_j benachbart sind. Dadurch verringert sich der Zeitaufwand auf $O(1)$ pro Kante, insgesamt also $O(n)$. Neben der Stabilität der vorhandenen Kanten muß man zusätzlich die Möglichkeit in Betracht ziehen, daß am Rand der Triangulierung neue Kanten entstehen. Das minimale ε, bei dem dieser Fall auftreten kann, läßt sich ebenfalls in $O(n)$ Zeit bestimmen.

5.2. Eine neue Charakterisierung von Stabilität

Bevor wir Bedingungen für ε-Stabilität betrachten, wollen wir zunächst untersuchen, wann überhaupt eine Delaunaykante zwischen zwei Punkten $p_i, p_j \in P$ existiert. Wir werden eine Existenzbedingung erhalten, die genau der Stabilitätsbedingung mit $\varepsilon = 0$ entspricht. Das Delaunaydiagramm D von P enthält eine Kante $p_i \wedge p_j$ genau dann, wenn p_i und p_j Voronoi-Nachbarn sind,

d. h. wenn im Voronoidiagramm von P die Zellen von p_i und p_j an einer Voronoikante zusammenstoßen. Für einen beliebigen inneren Punkt c der Voronoikante gilt:

$$\operatorname{dist}(c, p_i) = \operatorname{dist}\big(c, p_j\big) < \min_{k \neq i,j} \operatorname{dist}(c, p_k) \quad , \tag{5.1}$$

wobei $\operatorname{dist}(\cdot,\cdot)$ den Euklidischen Abstand bezeichnet. Existiere nun andererseits ein Punkt c, der (5.1) erfüllt. Aufgrund der Stetigkeit von $\operatorname{dist}(\cdot,\cdot)$ gilt die Ungleichung $\operatorname{dist}(c, p_i) < \min \operatorname{dist}(c, p_k)$ in einer Umgebung von c. Insbesondere gilt sie auf einem Teil der Mittelsenkrechten von p_i und p_j, welcher c enthält und Länge > 0 hat. Dieser Teil muß in der Voronoikante zwischen p_i und p_j enthalten sein, woraus die Existenz der Voronoikante folgt.

Betrachten wir nun einen Punkt c mit

$$\max\{\operatorname{dist}(c, p_i), \operatorname{dist}\big(c, p_j\big)\} < \min_{k \neq i,j} \operatorname{dist}(c, p_k) \quad . \tag{5.2}$$

Zunächst beobachten wir, daß Ungleichung (5.1) ein Spezialfall von Ungleichung (5.2) ist. Dieser Spezialfall ist dadurch charakterisiert, daß c auf der Mittelsenkrechten von p_i und p_j liegt. Gelte nun Ungleichung (5.2), etwa mit $\operatorname{dist}(c, p_i) \leq \operatorname{dist}\big(c, p_j\big)$. Wir verschieben den Punkt c geradlinig auf p_j zu, bis er auf der Mittelsenkrechten von p_i und p_j liegt. Jetzt gilt $\operatorname{dist}(c, p_i) = \operatorname{dist}\big(c, p_j\big)$. Durch das Verschieben hat sich $\operatorname{dist}\big(c, p_j\big)$ genau um die Länge der Verschiebung verringert. Die Abstände $\operatorname{dist}(c, p_k)$ von c zu anderen Punkten können sich um maximal diesen Betrag verringert haben. Der verschobene Punkt c erfüllt Ungleichung (5.1).

LEMMA 5.3. *Für eine Punktmenge $P = \{p_1, \dots, p_n\}$ und ihr Delaunaydiagramm sind äquivalent:*

1. *es existiert eine Delaunaykante $p_i \wedge p_j$,*
2. *es existiert ein Punkt c, der Ungleichung (5.1) erfüllt und*
3. *es existiert ein Punkt c, der Ungleichung (5.2) erfüllt.*

BEMERKUNG 5.4. *Der Punkt c ist Mittelpunkt einer (abgeschlossenen) Kreisscheibe K, die p_i und p_j enthält, aber keinen weiteren Punkt von P.*

LEMMA 5.5. *Existiere ein Kreis ∂K, der durch p_i und p_j sowie durch zwei weitere Punkte $p_k, p_\ell \in P$ geht. Seien weiter p_k und p_ℓ auf verschiedenen Seiten der Gerade $p_i \vee p_j$ gelegen. Dann ist $p_i \wedge p_j$ nicht Kante im Delaunaydiagramm von P.*

Beweis: Die Situation ist in Abbildung 5.1 links dargestellt. Sei c der Mittelpunkt

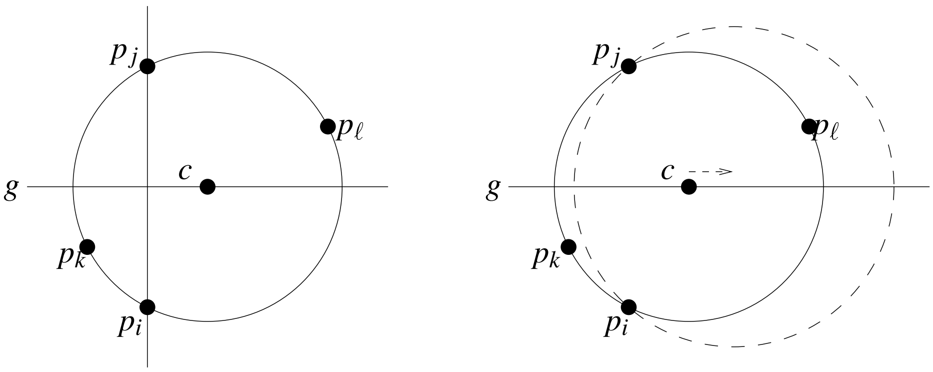


ABBILDUNG 5.1. Skizzen zum Beweis von Lemma 5.5.

von ∂K. c liegt auf der Mittelsenkrechten g von p_i und p_j. Wir verschieben nun c auf der Mittelsenkrechten nach rechts, und betrachten weiterhin die Kreise um c durch p_i und p_j, wie in Abbildung 5.1 rechts dargestellt. Jeder dieser Kreise enthält p_ℓ in seinem Inneren, d. h. p_ℓ liegt näher zu c als p_i und p_j. Verschieben wir c auf g nach links, dann gilt das gleiche mit p_k. Wir folgern, daß es keinen Punkt c gibt, der Ungleichung (5.1) erfüllt. Daher gibt es keine Delaunaykante $p_i \wedge p_j$. ■

Betrachten wir nun eine ε-Störung P' und die Kanten ihres Delaunaydiagramms. Ein verschobener Punkt p_i' liegt in einer abgeschlossenen Kreisscheibe um p_i mit Radius ε. Innerhalb dieser ε-Scheibe kann p_i' beliebig liegen. In Analogie zu Bemerkung 5.4 erscheint folgendes Kriterium für die garantierte Existenz einer Kante zwischen p_i' und p_j' plausibel: Es existiert eine Kreisscheibe, die die ε-Scheiben um p_i und p_j enthält, aber keine weitere ε-Scheibe schneidet. Dies ist in der Tat eine hinreichende und notwendige Bedingung. Wir wollen sie als Bedingung an den Mittelpunkt der Kreisscheibe formulieren.

SATZ 5.6. *Eine Kante $p_i \wedge p_j$ des Delaunaydiagramms D von P ist genau dann ε-stabil, wenn ein Punkt c existiert mit*

$$\max\{\operatorname{dist}(c,p_i),\operatorname{dist}(c,p_j)\} + \varepsilon < \min_{k\neq i,j} \operatorname{dist}(c,p_k) - \varepsilon \quad . \tag{ε-5.2}$$

Beweis:

„⇐“ :

Erfülle Punkt c die Ungleichung (ε-5.2). Für jede ε-Störung P' gilt

$$\operatorname{dist}(p_i,p_i') \le \varepsilon \quad , \quad \operatorname{dist}(p_j,p_j') \le \varepsilon \quad \text{und} \quad \operatorname{dist}(p_k,p_k') \le \varepsilon \quad \forall k \neq i,j \quad ,$$

und damit

$$\begin{aligned} \max\{\operatorname{dist}(c,p_i'),\operatorname{dist}(c,p_j')\} &\le \max\{\operatorname{dist}(c,p_i),\operatorname{dist}(c,p_j)\} + \varepsilon \\ &< \min_{k\neq i,j} \operatorname{dist}(c,p_k) - \varepsilon \\ &\le \min_{k\neq i,j} \operatorname{dist}(c,p_k') \quad . \end{aligned}$$

Nach Lemma 5.3 ist $p_i' \wedge p_j'$ Kante im Delaunaydiagramm von P'.

„⇒“ :

Existiere die Delaunaykante $p_i' \wedge p_j'$ für jede ε-Störung P'. Zunächst stellen wir fest, daß die konvexe Hülle der ε-Scheiben um p_i und p_j nicht von einer weiteren ε-Scheibe geschnitten wird. Gäbe es nämlich einen solchen Schnitt, wie in Abbildung 5.2 dargestellt, dann könnte man p_k' auf der Strecke zwischen p_i' und p_j' anordnen[1]. Bei dieser Wahl der verschobenen Punkte würde offensichtlich keine Delaunaykante $p_i' \wedge p_j'$ existieren. In Abbildung 5.2 sieht man auch die Äquivalenz

[1] Siehe auch [**Sal91**, Section 5.4]. Existiert der Schnitt, dann liegt p_k bis auf eine ε-Störung zwischen p_i und p_j.

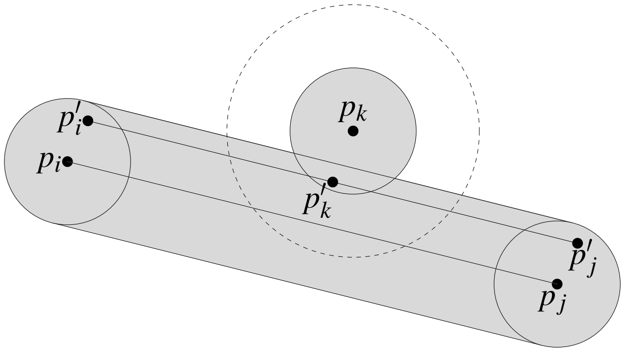


ABBILDUNG 5.2. Die konvexe Hülle der ε-Scheiben um p_i und p_j wird von einer weiteren ε-Scheibe geschnitten. Die 2ε-Scheibe um p_k ist als gestrichelter Kreis dargestellt.

dieser Aussage dazu, daß die Strecke $p_i \wedge p_j$ keine 2ε-Scheibe um einen Punkt p_k, $k \neq i, j$, schneidet.

Wir wählen ein Koordinatensystem so, daß die Mittelsenkrechte g von p_i und p_j identisch mit der x-Achse ist. Zu jedem $x \in \mathbb{R}$ definieren wir eine Kreisscheibe $K(x)$. Der Mittelpunkt dieser Kreisscheibe ist $(x, 0)$, und p_i und p_j liegen auf ihrem Rand. Verschiebt man den Mittelpunkt auf g nach rechts, dann vergrößert sich der Teil von $K(x)$, der rechts der Geraden $p_i \vee p_j$ liegt. Der Teil links der Geraden verkleinert sich, vergleiche Abbildung 5.1 rechts. Bei Verschiebung nach links verhält sich $K(x)$ genau symmetrisch. Um jeden Punkt p_k, $k \neq i, j$, legen wir nun eine Kreisscheibe mit Radius 2ε. Habe etwa eine Scheibe $K(x^*)$ mit der 2ε-Scheibe um p_k einen Schnittpunkt q rechts der Geraden $p_i \vee p_j$. Für jedes $x > x^*$ hat dann $K(x)$ ebenfalls q mit der 2ε-Scheibe gemeinsam. Läßt man x gegen $-\infty$ gehen, dann konvergiert $K(x)$ gegen die abgeschlossene Halbebene links von $p_i \vee p_j$. Der Teil des Kreises $\partial K(x)$ rechts von $p_i \vee p_j$ konvergiert gegen die offene Strecke $\operatorname{int}(p_i \wedge p_j)$. Da diese Strecke keine 2ε-Scheibe schneidet, muß es ein minimales x_r geben, für das $K(x_r)$ eine 2ε-Scheibe rechts von $p_i \vee p_j$ schneidet. Falls keine der 2ε-Scheiben wenigstens teilweise rechts von $p_i \vee p_j$ liegt, setzen wir $x_r := +\infty$. Analog läßt sich ein maximales x_ℓ bestimmen, für das $K(x_\ell)$ eine 2ε-Scheibe links von $p_i \vee p_j$ schneidet. Bei völliger Schnittfreiheit wählen wir $x_\ell := -\infty$. Wäre nun $x_\ell \geq x_r$, dann würde $K(x_\ell)$ sowohl rechts als auch links von $p_i \vee p_j$ mindestens eine 2ε-Scheibe schneiden. Seien p_k und p_ℓ die Mittelpunkte eines solchen Paares von 2ε-Scheiben, siehe Abbildung 5.3. Wir betrachten den Kreis $\partial K'$, der mit $K(x_\ell)$ konzentrisch ist und einen um ε größeren Radius hat. Die ε-Scheiben um p_i und p_j berühren $\partial K'$ von innen, die ε-Scheiben um p_k und p_ℓ schneiden ihn. Daher gibt es eine ε-Störung P', so daß p'_i, p'_j, p'_k und p'_ℓ auf $\partial K'$

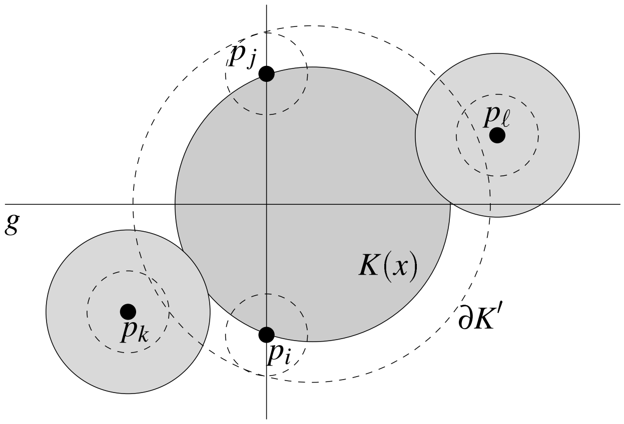


ABBILDUNG 5.3. Kreisscheibe $K(x)$ schneidet 2ε-Scheiben auf beiden Seiten der Gerade $p_i \vee p_j$. Gestrichelt: Kreis $\partial K'$ schneidet die ε-Scheiben der vier beteiligten Punkte.

liegen. Dabei liegen p'_k und p'_ℓ auf verschiedenen Seiten der Gerade $p'_i \vee p'_j$. Nach Lemma 5.5 wäre die Kante $p_i \wedge p_j$ ε-instabil, im Widerspruch zur Voraussetzung. Daher muß $x_\ell < x_r$ sein. Für $x_\ell < x < x_r$ schneidet die Kreisscheibe $K(x)$ keine der 2ε-Scheiben. Vergrößern wir den Radius dieser Kreisscheibe um ε, dann enthält sie ähnlich wie $\partial K'$ in Abbildung 5.3 die ε-Scheiben um p_i und p_j, schneidet aber keine weitere ε-Scheibe. Ihr Mittelpunkt $c = (x, 0)$ erfüllt Ungleichung (ε-5.2). ∎

Bei der Konstruktion der Kreisscheibe $K(x)$ im vorhergehenden Beweis liegt der Mittelpunkt $c = (x, 0)$ stets auf der Mittelsenkrechten von p_i und p_j. Es gilt daher

KOROLLAR 5.7. *Die Delaunaykante $p_i \wedge p_j$ ist genau dann ε-stabil, wenn ein Punkt c existiert mit*

$$(\varepsilon\text{-5.1}) \qquad \operatorname{dist}(c, p_i) = \operatorname{dist}\big(c, p_j\big) < \min_{k \neq i,j} \operatorname{dist}(c, p_k) - 2\varepsilon \quad .$$

5.3. Das Entscheidungsproblem

Wir betrachten das folgende Entscheidungsproblem:

Gegeben P und ε, ist die Delaunaykante $p_i \wedge p_j$ ε-stabil oder nicht?

Unser Test auf ε-Stabilität sucht Punkte c, die Ungleichung (ε-5.1) erfüllen. Wir betrachten zunächst nur den Punkt p_i und ein festes p_k, $k \neq i, j$. Die resultierende

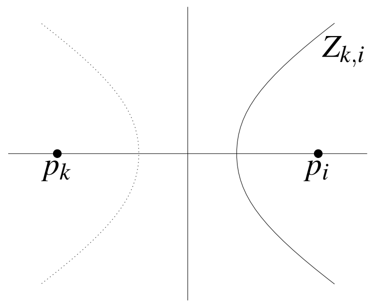


ABBILDUNG 5.4. Der rechte Hyperbelzweig $Z_{k,i}$ löst die Gleichung $\mathrm{dist}(c, p_i) = \mathrm{dist}(c, p_k) - 2\varepsilon$. Der andere Zweig (gepunktet) ist $Z_{i,k}$.

Ungleichung

$$\begin{aligned} &\mathrm{dist}(c, p_i) < \mathrm{dist}(c, p_k) - 2\varepsilon \\ \Leftrightarrow \quad &\mathrm{dist}(c, p_k) - \mathrm{dist}(c, p_i) > 2\varepsilon \end{aligned} \tag{5.3}$$

hat als Lösungsmenge ein im allgemeinen unbeschränktes Gebiet in der Ebene. Im Fall $\mathrm{dist}(p_i, p_k) \leq 2\varepsilon$ sehen wir mit Hilfe der Dreiecksungleichung, daß dieses Gebiet leer ist. Andernfalls ist der Rand des Gebiets ein durch

$$\mathrm{dist}(c, p_k) - \mathrm{dist}(c, p_i) = 2\varepsilon \tag{5.4}$$

definierter Zweig einer Hyperbel (siehe Abbildung 5.4). Wir nennen diesen Hyperbelzweig $Z_{k,i}$. Man beachte, daß das gesuchte Gebiet konvex ist. Die gesamte Hyperbel ist durch

$$\mathrm{dist}(c, p_k) - \mathrm{dist}(c, p_i) = \pm 2\varepsilon$$

definiert. Ihre Brennpunkte sind p_i und p_k. Die Hauptachsen sind die Gerade $p_i \vee p_k$ sowie die Mittelsenkrechte von p_i und p_k. Wegen $\mathrm{dist}(p_i, p_k) > 2\varepsilon$ ist die Hyperbel nicht entartet, insbesondere existiert sie immer.[2] Die Asymptoten der Hyperbel lassen sich geometrisch konstruieren. Dies ist in Abbildung 5.5 dargestellt. Sei K die Kreisscheibe um p_k mit Radius 2ε. K kann p_i nicht enthalten, wegen $\mathrm{dist}(p_i, p_k) > 2\varepsilon$. Wir legen zwei tangentiale Strecken von p_i bis an K. Die Mittelsenkrechten dieser Strecken sind die Asymptoten. Die Vereinigung der in Abbildung 5.5 fett gezeichneten Halb-Asymptoten bezeichnen wir als Asymptotenzweig $A_{k,i}$.

[2]Für $\mathrm{dist}(p_i, p_k) = 2\varepsilon$ entartet die Hyperbel zu zwei Halbgeraden, die in p_i bzw. p_k beginnen und entlang der ersten Hauptachse $p_i \vee p_k$ nach außen laufen. Für $\mathrm{dist}(p_i, p_k) < 2\varepsilon$ existiert die Hyperbel nicht.

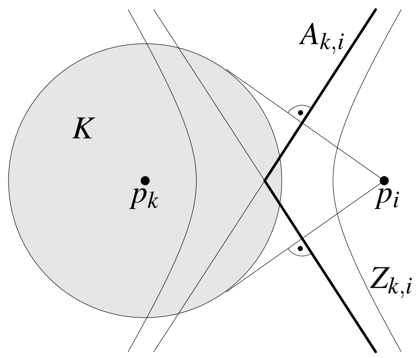


ABBILDUNG 5.5. Geometrische Konstruktion der Asymptoten.

Der Hyperbelzweig $Z_{k,i}$ liefert uns eine vergleichsweise große Fläche, in der die Lösungsmenge der Ungleichung (ε-5.1) enthalten sein muß. Wir wissen bereits, daß die Lösungsmenge Teilmenge der Mittelsenkrechten g von p_i und p_j ist. Für den Durchschnitt von $Z_{k,i}$ mit g gibt es im allgemeinen vier Möglichkeiten: kein Schnittpunkt, zwei Schnittpunkte, ein Schnittpunkt tangential und ein Schnittpunkt kreuzend. Wir halten p_i und p_j fest und variieren p_k. O. B. d. A. liege g horizontal, p_j ober– und p_i unterhalb von g, wie schon in Abbildung 5.1. Weiter liege p_k unterhalb oder auf g und auf oder links von $p_i \vee p_j$. Wir unterscheiden nun vier verschiedene Lagen von p_k samt seiner 2ε-Scheibe K. Sie sind in Abbildung 5.6 dargestellt.

1. Abbildung 5.6, links oben. K schneidet die Gerade $p_i \vee p_j$ nicht. Die Tangentenstrecken an K laufen von p_i aus nach links. Daher läuft eine Halbgerade von $A_{k,i}$ nach oben, die andere nach unten. Es folgt, daß $A_{k,i}$ die Gerade g in genau einem Punkt schneidet, und zwar kreuzend. Wie man leicht sieht, hat g dann auch genau einen kreuzenden Schnittpunkt mit $Z_{k,i}$.
2. Abbildung 5.6, rechts oben. K berührt die Gerade $p_i \vee p_j$ unterhalb von p_i. Eine Tangentenstrecke läuft nach links, die andere vertikal nach unten. Die Halbgeraden von $A_{k,i}$ laufen nach oben und horizontal, der Scheitelpunkt liegt strikt unterhalb von g. Wie im Fall 1 hat g mit $A_{k,i}$ und mit $Z_{k,i}$ jeweils genau einen kreuzenden Schnittpunkt.
3. Abbildung 5.6, links unten. K schneidet die Gerade $p_i \vee p_j$ in mehreren Punkten, der Schnitt liegt unterhalb von p_i. Eine Tangentenstrecke läuft nach links unten, die andere nach rechts unten. Beide Halbgeraden von $A_{k,i}$ laufen nach oben, und g hat mit $A_{k,i}$ zwei Schnittpunkte. Der Scheitelpunkt von $Z_{k,i}$ liegt strikt unterhalb g. Daher hat g auch mit $Z_{k,i}$ zwei Schnittpunkte.

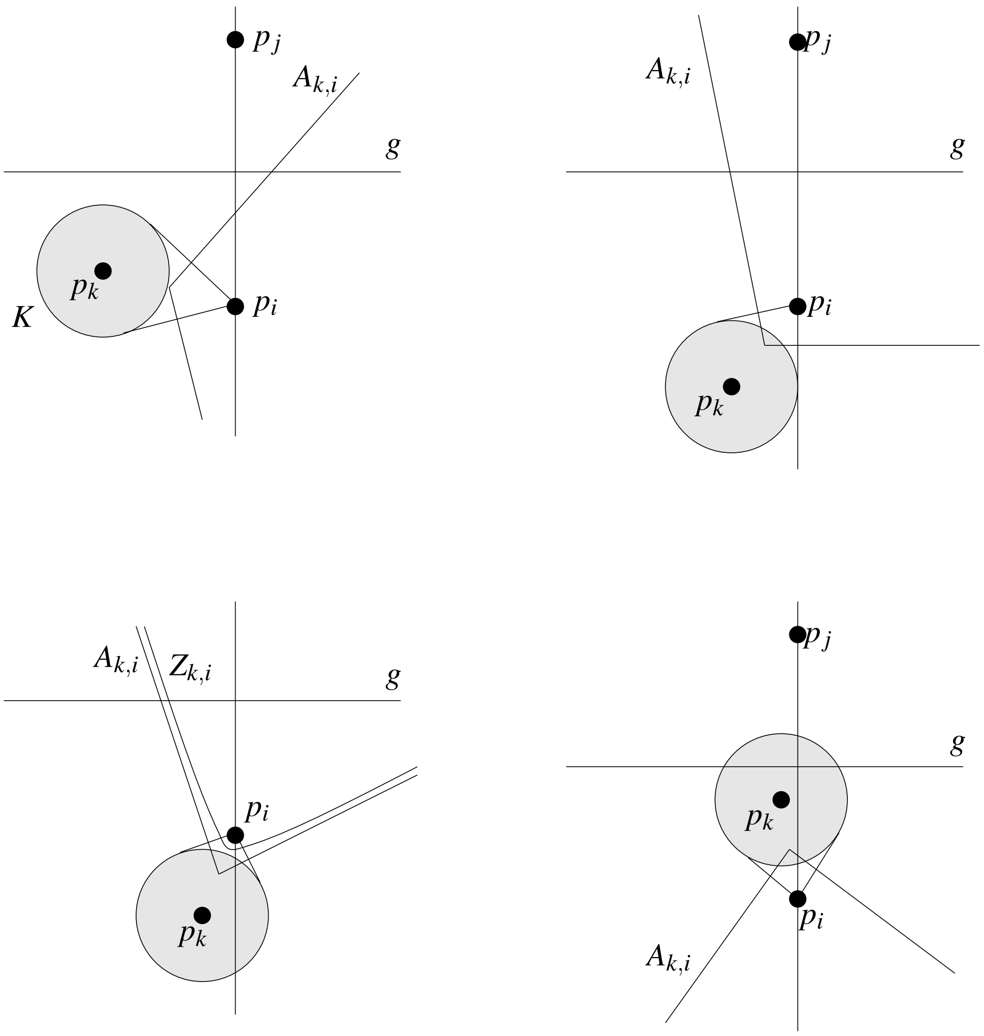


ABBILDUNG 5.6. Mögliche Lagen von p_k.

4. Abbildung 5.6, rechts unten. K schneidet die Strecke $p_i \wedge p_j$ in mindestens einem inneren Punkt der Strecke. Dies ist äquivalent zu der in Abbildung 5.2 gezeigten Situation: Die ε-Scheibe um p_k schneidet die konvexe Hülle der ε-Scheiben um p_i und p_j, aber nicht die Scheiben selbst. Eine Tangentenstrecke läuft nach links oben, die andere nach rechts oben. (Im Fall der Berührung läuft eine Tangentenstrecke vertikal nach oben.) Beide Halbgeraden von $A_{k,i}$ laufen nach unten (bei Berührung läuft eine horizontal). Der Scheitelpunkt von $A_{k,i}$ liegt unterhalb von g. Es existiert kein Schnittpunkt von g mit $A_{k,i}$, und daher auch nicht mit $Z_{k,i}$.

Zur Berechnung der Schnittpunkte von g mit $Z_{k,i}$ benötigen wir ein Koordinatensystem. Wir legen die x-Achse durch p_i und p_k und den Ursprung genau zwischen die beiden Punkte. Die Koordinatenachsen sind jetzt die Hauptachsen der Hyperbel (vgl. Abbildung 5.4). Bei dieser Wahl des Koordinatensystems wird die (gesamte) Hyperbel durch eine Gleichung der Form

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \tag{5.5}$$

beschrieben. Hierbei ist $a = \varepsilon$. Den Wert für b^2 erhalten wir aus der linearen Exzentrizität der Hyperbel. Sie ist zum einen der halbe Abstand zwischen den Brennpunkten, zum anderen gleich $\sqrt{a^2 + b^2}$.

$$\frac{\operatorname{dist}(p_i, p_k)}{2} = \sqrt{\varepsilon^2 + b^2} \qquad \Longrightarrow \qquad b^2 = \left(\frac{\operatorname{dist}(p_i, p_k)}{2}\right)^2 - \varepsilon^2$$

Um die Schnittpunkte der Hyperbel mit der Mittelsenkrechten g von p_i und p_j zu berechnen, beschreiben wir g in dem gewählten Koordinatensystem durch die parametrische Gleichung

$$\frac{1}{2}(p_i + p_j) + t \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} (p_i - p_j) \quad , \quad t \in \,]-\infty, \infty[\quad . \tag{5.6}$$

Offensichtlich hängt diese Gleichung von der Wahl des Koordinatensystems, und damit von p_k, ab. Die Parametrisierung von g, d. h. die Abbildung von Parametern t auf Punkte der Geraden, ist jedoch stets die gleiche. Sie wird lediglich in einem anderen Koordinatensystem dargestellt. Durch koordinatenweises Einsetzen in Gleichung (5.5) erhalten wir eine quadratische Gleichung in t. Ihre Lösungen beschreiben die Schnittpunkte der gesamten Hyperbel mit g. Durch Überprüfung von Bedingung (5.4) erkennen wir diejenigen Schnittpunkte, die auf dem Zweig $Z_{k,i}$ liegen. Die verschiedenen Möglichkeiten sind in Abbildung 5.7 von links nach rechts dargestellt. Bei zwei Schnittpunkten erfüllen alle strikt dazwischenliegenden Punkte Gleichung (5.3). Bei genau einem kreuzenden Schnittpunkt ist die Gleichung auf der von p_k abgewandten offenen Halbgerade erfüllt. Existiert kein Schnitt, dann erfüllt kein Punkt die Gleichung, und die Delaunaykante $p_i \wedge p_j$ ist ε-instabil. Dies sind alle Fälle, die uns bei der Diskussion der verschiedenen Lagen von p_k begegnet sind. Offenbar brauchen wir den Fall eines tangentialen Schnittpunkts nicht berücksichtigen. Die Schnittpunkte sind unabhängig davon, ob wir p_i und $Z_{k,i}$ oder p_j und $Z_{k,j}$ zugrundelegen: Sowohl $g \cap Z_{k,i}$ als auch $g \cap Z_{k,j}$ enthält genau die Endpunkte der Lösungsmenge von Ungleichung (ε-5.1) (mit nur

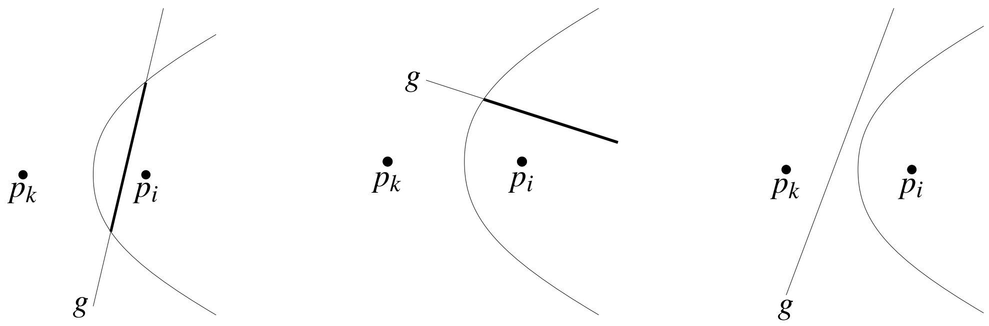


ABBILDUNG 5.7. Schnitte von g mit dem Hyperbelzweig.

einem Punkt $p_k \neq p_i, p_j$). Aus diesem Grund kann es auch mit $Z_{k,j}$ keine tangentialen Schnitte geben. Wir können also die Schnittberechnung nach Belieben mit p_i oder mit p_j durchführen.

Auf diese Weise erhalten wir für jedes $k \neq i, j$ ein offenes, möglicherweise leeres oder unendliches Intervall I_k auf der Mittelsenkrechten g. Der Durchschnitt aller dieser Intervalle ist genau dann nicht leer, wenn die Delaunaykante ε-stabil ist. Da die Parametrisierung nach Gleichung (5.6) unabhängig von p_k ist, bietet sie eine attraktive Möglichkeit zur Darstellung der Intervalle. Wir repräsentieren jedes I_k durch ein Intervall der reellen Zahlengerade, nämlich sein Urbild unter der Parametrisierung.

Der Durchschnitt von $n-2$ Zahlenintervallen läßt sich in Zeit $O(n)$ bestimmen. Bis hierher hat unser Stabilitätstest einen Aufwand von $O(n)$ für eine Delaunaykante $p_i \wedge p_j$. Wir können diesen Aufwand verringern, indem wir statt aller p_k nur die Voronoinachbarn von p_i und von p_j betrachten. Wie wir im folgenden sehen, bleibt der Stabilitätstest korrekt.

Wenn ein Punkt c Ungleichung (ε-5.1) erfüllt, dann auch Ungleichung (5.1), d. h. c liegt auf der Voronoikante von p_i und p_j. Sei p_ℓ weder ein Voronoinachbar von p_i noch von p_j. Die Strecke $c \wedge p_\ell$ schneidet das (abgeschlossene) Voronoigebiet eines Punktes p_v, der Voronoinachbar von p_i oder von p_j ist. Sei q ein Schnittpunkt der Strecke mit dem Voronoigebiet. Es ist

$$\mathrm{dist}(p_\mathrm{v}, q) \leq \mathrm{dist}(p_\ell, q)$$

und

$$\mathrm{dist}(p_\ell, c) = \mathrm{dist}(p_\ell, q) + \mathrm{dist}(q, c) \quad .$$

Für p_v, q und c gilt die Dreiecksungleichung, so daß

$$\begin{aligned}\mathrm{dist}(p_\mathrm{v},c) &\leq \mathrm{dist}(p_\mathrm{v},q)+\mathrm{dist}(q,c)\\ &\leq \mathrm{dist}(p_\ell,q)+\mathrm{dist}(q,c)\\ &= \mathrm{dist}(p_\ell,c) \quad .\end{aligned}$$

Wir sehen, daß p_ℓ für $\min \mathrm{dist}(c,p_k)$ unerheblich ist, solange c auf der Voronoikante von p_i und p_j liegt. Dies gilt selbst dann, wenn p_ℓ und p_i (oder p_j) schwache Voronoinachbarn sind, d. h. wenn ihre Voronoigebiete einen Voronoiknoten (aber keine Voronoikante) gemeinsam haben.

Gehört andererseits $c \in g$ nicht zur Voronoikante von p_i und p_j, dann liegt c außerhalb der Voronoigebiete von p_i und p_j. Die Strecke $p_i \wedge c$ schneidet das Voronoigebiet eines Voronoinachbarn p_v von p_i, und in diesem Fall gilt die Ungleichung

$$\mathrm{dist}(p_\mathrm{v},c) \leq \mathrm{dist}(p_i,c) \quad .$$

Punkt c kann Gleichung (5.3) selbst für $\varepsilon = 0$ nicht erfüllen, gehört also nicht zu I_v.

Wir brauchen also nur die Voronoinachbarn p_v von p_i und p_j betrachten, und die resultierenden Intervalle I_v miteinander schneiden. Da der Durchschnitt in der Voronoikante von p_i und p_j enthalten ist, kann die Betrachtung der übrigen p_ℓ ihn nicht mehr verändern. Das ergibt den folgenden Algorithmus:

Algorithmus 5.1 Stabilitätstest

Eingabe: Delaunaydiagramm von $P \subset \mathbb{R}^2$, Kante $p_i \wedge p_j$ des Delaunaydiagramms, ε.

1: Bestimme die starken Voronoinachbarn von p_i und von p_j.
2: Wenn p_i oder p_j zu seinem nächsten Nachbarn p_k einen Abstand $\leq 2\varepsilon$ hat, dann ist die Delaunaykante $p_i \wedge p_j$ ε-instabil.
3: Bestimme für jeden Voronoinachbarn p_v von p_i oder von p_j das offene Intervall $I_\mathrm{v} \subset g$.
4: Bilde den Durchschnitt der I_v.
5: Die Delaunaykante $p_i \wedge p_j$ ist genau dann ε-stabil, wenn der Durchschnitt nicht leer ist.

Im Delaunaydiagramm sind genau die starken Voronoinachbarn durch jeweils eine Kante verbunden. Schritt 1 muß also die Nachbarn von p_i und von p_j im Delaunaydiagramm bestimmen. Sei deren Anzahl m, dann benötigt jeder Schritt des Algorithmus $O(m)$ Zeit. Dabei gehen wir für Schritt 1 von einer ähnlichen Datenstruktur wie in Kapitel 3 aus (siehe Seite 35). Die Laufzeit von Algorithmus 5.1 ist daher $O(m)$. Im schlechtesten Fall ist dies immer noch $O(n)$, aber im Mittel besitzt jeder Punkt weniger als 6 Voronoinachbarn.

5.4. Das Optimierungsproblem

Nach dem Entscheidungsproblem wollen wir uns jetzt dem zugehörigen Optimierungsproblem zuwenden:

Gegeben P, bis zu welchem ε ist die Delaunaykante $p_i \wedge p_j$ stabil?

Wir suchen also

$$\varepsilon_{\sup}(p_i \wedge p_j) := \sup\{\varepsilon \mid p_i \wedge p_j \text{ ist } \varepsilon\text{-stabil}\} \quad .$$

Wie man leicht sieht, ist $p_i \wedge p_j$ für den Wert $\varepsilon = \varepsilon_{\sup}(p_i \wedge p_j)$ selbst bereits instabil. Entweder ist $\varepsilon_{\sup}(p_i \wedge p_j) = \frac{1}{2}\operatorname{dist}\big(p_i, p_j\big)$, oder es existiert ein Punkt c auf der Mittelsenkrechten g von $p_i \wedge p_j$, welcher (ε-5.1) als Gleichung erfüllt[3]. Für c gilt also

$$\operatorname{dist}(c, p_i) = \operatorname{dist}\big(c, p_j\big) = \min_{k \neq i,j} \operatorname{dist}(c, p_k) - 2\varepsilon$$

Andererseits darf kein Punkt der Mittelsenkrechten (ε-5.1) in seiner eigentlichen Form, nämlich als Ungleichung, erfüllen. Wir suchen jetzt für jeden Punkt von g dasjenige ε, für das er (ε-5.1) als Gleichung erfüllt. Dies beschreiben wir durch eine Funktion $f_{\min}^{i,j} : g \to \mathbb{R}$. Der Stabilitätsradius $\varepsilon_{\sup}$ ist dann das Supremum von $f_{\min}^{i,j}$ über g.

Betrachten wir zunächst nur einen weiteren Punkt p_k. Ein Punkt $c \in g$ erfüllt (ε-5.1) bezüglich p_k als Gleichung für den Wert

$$\varepsilon = \frac{1}{2} \underbrace{\Big(\operatorname{dist}(c, p_k) - \operatorname{dist}(c, p_i)\Big)}_{=: f_{k,i}(c)} \quad .$$

Offensichtlich ist $f_{\min}^{i,j}$ das punktweise Minimum aller $f_{k,i}$:

$$f_{\min}^{i,j}(c) = \min_{k \neq i,j} f_{k,i}(c) \quad \forall c \in g \quad .$$

[3]Es kann sein, daß dieser Punkt c einer der beiden „Endpunkte im Unendlichen“ von g ist. In diesem Fall berührt für $\varepsilon = \varepsilon_{\sup}(p_i \wedge p_j)$ eine weitere ε-Scheibe die konvexe Hülle der ε-Scheiben von p_i und p_j.

BEMERKUNG 5.8. *Genau wie beim Entscheidungsproblem brauchen wir nicht alle übrigen Punkte $p_k \in P \setminus \{p_i, p_j\}$ betrachten, sondern nur die Voronoinachbarn von p_i und von p_j. Dadurch wird evtl. der Wert von $f^{i,j}_{\min}$ außerhalb der Voronoikante von p_i und p_j falsch bestimmt. Da aber $f^{i,j}_{\min}$ in diesem Bereich negativ ist, und dies auch anhand der Nachbarpunkte korrekt erkannt wird, spielt dieser Fehler für die Berechnung von $\varepsilon_{\sup}$ keine Rolle.*

Die besondere Struktur der $f_{k,i}$ ermöglicht es uns, das punktweise Minimum mit kombinatorischen Methoden zu bestimmen. Diese Struktur wollen wir im folgenden näher ergründen. Wir wählen in der Ebene ein Koordinatensystem so, daß g die x-Achse ist und p_i auf der positiven y-Achse liegt. O. B. d. A. nehmen wir an, daß p_i und p_k nicht auf verschiedenen Seiten von g liegen (andernfalls verwenden wir p_j anstelle von p_i). Mit $c =: (x,0)$, $p_i =: (0,y_i)$ und $p_k =: (x_k,y_k)$ bekommt $f_{k,i}$ die Darstellung

$$f_{k,i}(c) = \sqrt{(x-x_k)^2 + y_k^2} - \sqrt{x^2 + y_i^2} \quad .$$

Statt $f_{k,i}(c)$ schreiben wir auch $f_{k,i}(x)$.

Da wir das Supremum von $f^{i,j}_{\min}$ suchen, kann das Verhalten von $f_{k,i}$ „im Unendlichen" von Interesse sein.

LEMMA 5.9.

$$\lim_{x\to\infty} f_{k,i}(x) = -x_k \qquad \textit{und} \qquad \lim_{x\to-\infty} f_{k,i}(x) = x_k \quad .$$

Beweis: Als Vorüberlegung betrachten wir zunächst den Ausdruck

$$\begin{aligned}
\sqrt{x^2+y_i^2} - \sqrt{x^2} &= \frac{\left(\sqrt{x^2+y_i^2} - \sqrt{x^2}\right)\left(\sqrt{x^2+y_i^2} + \sqrt{x^2}\right)}{\sqrt{x^2+y_i^2} + \sqrt{x^2}} \\
&= \frac{x^2+y_i^2-x^2}{\sqrt{x^2+y_i^2} + \sqrt{x^2}} \\
&= \frac{y_i^2}{\sqrt{x^2+y_i^2} + \sqrt{x^2}} \\
&\xrightarrow[x\to\pm\infty]{} 0
\end{aligned}$$

Analog dazu ist auch

$$\lim_{x\to\pm\infty} \sqrt{(x-x_k)^2 + y_k^2} - \sqrt{(x-x_k)^2} = 0 \quad .$$

Wenden wir uns nun dem Verhalten von $f_{k,i}$ für $x \to \infty$ zu. Für $x > \max\{x_k, 0\}$ gilt

$$\sqrt{(x-x_k)^2} = x - x_k \quad \text{und} \quad \sqrt{x^2} = x \quad .$$

Damit ist

$$\begin{aligned}
0 &= \lim_{x\to\infty}\left(\sqrt{(x-x_k)^2+y_k^2} - \sqrt{(x-x_k)^2}\right) - \lim_{x\to\infty}\left(\sqrt{x^2+y_i^2} - \sqrt{x^2}\right) \\
&= \lim_{x\to\infty}\left(\sqrt{(x-x_k)^2+y_k^2} - \sqrt{x^2+y_i^2} - (x-x_k) + x\right) \\
&= \lim_{x\to\infty} f_{k,i}(x) + x_k \quad .
\end{aligned}$$

Der Limes von $f_{k,i}$ für $x \to \infty$ existiert also und ist $-x_k$.
Für $x \to -\infty$ können wir $x < \min\{x_k, 0\}$ voraussetzen, so daß jetzt

$$\sqrt{(x-x_k)^2} = -(x - x_k) \quad \text{und} \quad \sqrt{x^2} = -x$$

gilt. Hieraus ergibt sich

$$\begin{aligned}
0 &= \lim_{x\to-\infty}\left(\sqrt{(x-x_k)^2+y_k^2} - \sqrt{(x-x_k)^2}\right) - \lim_{x\to-\infty}\left(\sqrt{x^2+y_i^2} - \sqrt{x^2}\right) \\
&= \lim_{x\to-\infty}\left(\sqrt{(x-x_k)^2+y_k^2} - \sqrt{x^2+y_i^2} + (x-x_k) - x\right) \\
&= \lim_{x\to-\infty} f_{k,i}(x) - x_k \quad ,
\end{aligned}$$

also ist $\lim_{x\to-\infty} f_{k,i}(x) = x_k$. ∎

Aus der Dreiecksungleichung ergibt sich

LEMMA 5.10. *Für alle $c \in g$ gilt*

$$-\mathrm{dist}(p_k, p_i) \leq f_{k,i}(c) \leq \mathrm{dist}(p_k, p_i) \tag{5.7}$$ ∎

In Gleichung (5.7) nimmt $f_{k,i}$ genau dann einen der Werte $\pm\mathrm{dist}(p_k, p_i)$ an, wenn c auf der Geraden $p_i \vee p_k$ und nicht zwischen p_i und p_k liegt. Aufgrund unserer Annahme, daß p_i und p_k nicht auf verschiedenen Seiten von g liegen, kann der Schnittpunkt $c := g \cap (p_i \vee p_k)$ nicht zwischen p_i und p_k liegen, sofern er überhaupt existiert. Wenn sich also die Geraden g und $p_i \vee p_k$ schneiden, dann nimmt $f_{k,i}$ im Schnittpunkt einen der Schrankenwerte an; $\mathrm{dist}(p_k, p_i)$, wenn p_i näher am Schnittpunkt liegt, und $-\mathrm{dist}(p_k, p_i)$, wenn p_k näher am Schnittpunkt liegt. Ist $p_i \vee p_k$ parallel zu g, dann ist $y_i = y_k$ und somit $\mathrm{dist}(p_k, p_i) = x_k$. In diesem Fall werden die beiden Schrankenwerte „im Unendlichen angenommen".

Die Gerade $p_i \vee p_k$ hat die Steigung $m = (y_k - y_i)/x_k$ und den y-Achsen-Abschnitt y_i. Ihr Schnittpunkt mit der x-Achse g liegt bei

$$x^* = \frac{-y_i}{m} = \frac{-y_i x_k}{y_k - y_i} \quad , \tag{5.8}$$

sofern $y_i \neq y_k$ gilt. Dieser Ausdruck für die x-Koordinate des Schnittpunkts ist auch dann korrekt, wenn $x_k = 0$ ist und die Gerade $p_i \vee p_k$ unendliche Steigung hat. Für $y_i = y_k$ sind die beiden Geraden parallel, es existiert also kein Schnittpunkt.

LEMMA 5.11. *Für $k \neq i, j$ hat die Funktion $f_{k,i}$ auf g höchstens eine lokale Extremstelle. Diese Stelle ist der Schnittpunkt von g mit der Geraden $p_i \vee p_k$.*

Beweis: Die Ableitung von $f_{k,i}(x)$ nach x ist

$$\begin{aligned}
f'_{k,i}(x) &= \frac{2(x-x_k)}{2\sqrt{(x-x_k)^2 + y_k^2}} - \frac{2x}{2\sqrt{x^2 + y_i^2}} \\
&= \frac{(x-x_k)\sqrt{x^2+y_i^2} - x\sqrt{(x-x_k)^2+y_k^2}}{\sqrt{(x-x_k)^2+y_k^2}\sqrt{x^2+y_i^2}} \quad .
\end{aligned}$$

Wegen $y_i \neq 0$ kann die zweite Wurzel im Nenner nicht verschwinden. Die erste Wurzel wird genau dann 0, wenn $y_k = 0$ und $x = x_k$ ist. Diesen Spezialfall werden wir später betrachten. Von ihm abgesehen ist $f_{k,i}(x)$ stetig differenzierbar über ganz $\mathbb{R}$. Als notwendige Voraussetzung für ein Extremum erhalten wir

$$\begin{aligned}
& \sqrt{x^2(x-x_k)^2 + y_i^2(x-x_k)^2} = \sqrt{x^2(x-x_k)^2 + x^2 y_k^2} \\
\Leftrightarrow \quad & y_i^2(x-x_k)^2 = x^2 y_k^2 \\
\Leftrightarrow \quad & 0 = (y_k^2 - y_i^2)x^2 + 2x_k y_i^2 x - y_i^2 x_k^2 \\
& = \big((y_k - y_i)x + y_i x_k\big)\big((y_k + y_i)x - y_i x_k\big) \\
& =: h(x) \quad .
\end{aligned}$$

Die vorletzte Zeile zeigt $h(x)$ in Linearfaktoren zerlegt. Wegen $y_i > 0$ und $y_k \geq 0$ kann der Leitkoeffizient $(y_k + y_i)$ des zweiten Linearfaktors nicht 0 sein. Wir unterscheiden zwei Fälle:

(I) $y_k \neq y_i$:
Die Funktion $h(x)$ hat zwei Nullstellen,

$$x_1^* = \frac{-y_i x_k}{y_k - y_i} \qquad \text{und} \qquad x_2^* = \frac{y_i x_k}{y_k + y_i} \quad .$$

(II) $y_k = y_i$:
In diesem Fall kann x_k nicht 0 sein, da sonst $p_k = p_i$ wäre. Der erste Linearfaktor von $h(x)$ ist also eine von 0 verschiedene Konstante, und wir erhalten lediglich die Nullstelle

$$x_2^* = \frac{y_i x_k}{y_k + y_i} \quad .$$

Die Gerade $p_i \vee p_k$ ist parallel zu g.

Die Nullstelle x_1^* kennen wir bereits aus Gleichung (5.8). Sie beschreibt den Schnittpunkt der Geraden $p_i \vee p_k$ mit g, in dem $f_{k,i}$ ein globales Extremum hat. Die Nullstelle x_2^* beschreibt den Schnittpunkt der Geraden $p_j \vee p_k$ mit g. Dies ist leicht zu sehen, wenn wir uns $y_j = -y_i$ vergegenwärtigen. Um das Verhalten von $f_{k,i}$ in diesem Schnittpunkt zu analysieren, betrachten wir den Hyperbelzweig $Z_{k,j}$ für den Wert $\varepsilon := f_{k,i}(x_2^*) = f_{k,j}(x_2^*)$, beziehungsweise $Z_{j,k}$ für den Wert $\varepsilon := -f_{k,i}(x_2^*)$, falls $f_{k,i}(x_2^*) < 0$ ist. Dieser Hyperbelzweig ist so gewählt, daß er durch den Punkt $c^* := (x_2^*, 0)$ läuft. Da c^* auf der ersten Hauptachse $p_j \vee p_k$ liegt, ist er der Scheitelpunkt des Hyperbelzweigs. Liegt in x_2^* nun eine Extremstelle von $f_{k,i}$ vor, dann muß g den Hyperbelzweig tangential berühren. Eine tangentiale Berührung im Scheitelpunkt bedeutet, daß g orthogonal zur Hauptachse $p_j \vee p_k$ ist. Daraus folgt $x_k = 0$ und $x_1^* = x_2^* = 0$. Wir sehen jetzt auch, daß es in Fall (II) kein lokales Extremum geben kann, denn $x_k = 0$ war dort ausgeschlossen. Als einzige Extremstelle bleibt also x_1^*.

Betrachten wir jetzt den Spezialfall $x = x_k$ und $y_k = 0$. An der Stelle x_k liegt ein globales Extremum vor, denn es handelt sich bei $(x_k, 0)$ um den Punkt p_k. Da $f_{k,i}(x)$ auf $\mathbb{R} \setminus \{x_k\}$ stetig differenzierbar ist, kann eine weitere Extremstelle nur bei x_1^* auftreten. Nun ist aber

$$x_1^* = \frac{-y_i x_k}{y_k - y_i} = \frac{-y_i x_k}{-y_i} = x_k \quad .$$

Die Funktion $f_{k,i}$ hat also höchstens ein Extremum auf g. ■

Das Lemma spielt für die Konstruktion des punktweisen Minimums keine Rolle, wohl aber für die anschließende Bestimmung des optimalen c und des zugehörigen $f_{\min}^{i,j}(c) = \varepsilon_{\sup}(p_i \wedge p_j)$. Für die Konstruktion von $f_{\min}^{i,j}$ ist folgende Tatsache wichtig.

LEMMA 5.12. *Sei $f_{k,i} \neq f_{\ell,i}$. Dann hat die Differenzfunktion $f_{k,i} - f_{\ell,i}$ maximal eine Nullstelle auf g.*

Beweis: Es ist

$$\begin{aligned} f_{k,i}(c) - f_{\ell,i}(c) &= \operatorname{dist}(c, p_k) - \operatorname{dist}(c, p_i) - (\operatorname{dist}(c, p_\ell) - \operatorname{dist}(c, p_i)) \\ &= \operatorname{dist}(c, p_k) - \operatorname{dist}(c, p_\ell) \quad . \end{aligned}$$

Die einzige Nullstelle dieser Funktion liegt im Schnittpunkt der Mittelsenkrechten von p_k und p_ℓ mit g, sofern der Schnitt nicht leer ist. Der Fall, daß die Mittelsenkrechte von p_k und p_ℓ mit g identisch ist, kann wegen $f_{k,i} \neq f_{\ell,i}$ nicht auftreten. ■

Die Funktionsgraphen von $f_{k,i}$ und $f_{\ell,i}$ können sich also höchstens einmal kreuzen. Man beachte, daß wir in einem Algorithmus den Fall $f_{k,i} = f_{\ell,i}$ explizit berücksichtigen müssen. Er kann auch für $p_k \neq p_\ell$ auftreten. Ist die Mittelsenkrechte parallel zu g, dann hat eine der beiden Funktionen in jedem Punkt von g einen kleineren Wert als die andere. Offensichtlich ist dies die Funktion, deren Punkt näher bei g liegt als der andere.

Wir wollen $f_{\min}^{i,j}$ durch einen Sweep entlang der Geraden g berechnen. Dazu verwenden wir unser Koordinatensystem mit g als x-Achse. Wie in Gleichung (5.6) stellen wir g durch einen Parameter dar, nur daß diesmal der Parameter x heißt und die exakte Bogenlänge wiedergibt. Auf diesen Parameter beziehen wir auch die $f_{k,i}$ und $f_{\min}^{i,j}$. Entsprechend unserer Annahme, daß kein Punkt außer p_j unterhalb der x-Achse liegt, ersetzen wir jede y-Koordinate durch ihren Betrag. Praktisch bedeutet das, daß wir jedem Punkt als x-Koordinate den Parameterwert seiner orthogonalen Projektion auf g zuordnen, und als y-Koordinate seinen (nicht negativen) Abstand von g.

Der Sweep bestimmt als erstes die asymptotischen Werte der $f_{k,i}$ für $x \to -\infty$ und sortiert die Funktionen aufsteigend nach diesen Werten. Die asymptotischen Werte sind nach Lemma 5.9 gerade die x-Koordinaten x_k der Punkte p_k. Haben zwei Punkte p_k und p_ℓ identische x-Koordinaten, dann sei o. B. d. A. $y_k \geq y_\ell \geq 0$. Wir werfen den Punkt p_k und die zugehörige Funktion $f_{k,i}$ weg, da $f_{k,i}(c) \geq f_{\ell,i}(c)$ für alle $c \in g$ gilt.

Während des Sweeps wird die Sortierung stets aktualisiert, wenn sich die Graphen zweier Funktionen überschneiden. Die jeweils aktuelle Sortierung bezeichnen wir durch die Permutation σ, also $f_{\sigma(1),i} < \cdots < f_{\sigma(m),i}$. Eine Überschneidung zweier Funktionsgraphen erkennen wir als Nullstelle der Differenzfunktion. Die jeweils nächste Überschneidung tritt immer zwischen Funktionen auf, die in der aktuellen Sortierung benachbart sind. Wir brauchen also immer nur die Nullstellen von Differenzfunktionen der Form $f_{\sigma(k),i} - f_{\sigma(k+1),i}$ zu kennen. Gemäß Lemma 5.12 berechnen wir eine solche Nullstelle als Schnittpunkt der Mittelsenkrechten von $p_{\sigma(k)}p_{\sigma(k+1)}$ mit der x-Achse.

Haben sich die Graphen von $f_{k,i}$ und $f_{\ell,i}$ einmal überkreuzt, dann können sie sich nach Lemma 5.12 nicht noch einmal überkreuzen. Das bedeutet, daß eine der beiden Funktionen, etwa $f_{k,i}$, ab diesem Punkt keinen Einfluß mehr auf $f_{\min}^{i,j}$ haben kann, da $f_{\ell,i}$ immer einen kleineren Wert liefert. Bei jeder Graphüberschneidung, also bei jedem Sweep-Ereignis, können wir eine der beteiligten Funktionen vollständig aus unserer sortierten Menge entfernen und brauchen sie nicht mehr zu betrachten. Im Programm implementieren wir die sortierte Menge der $f_{k,i}$ als eine doppelt verkettete Liste, aus der die betreffende Funktion einfach ausgekettet wird. An der Reihenfolge der übrigen Funktionen ändert sich nichts.

Pro Sweepereignis sortieren wir also eine der Funktionen aus und berechnen eine neue Überschneidung, und zwar zwischen den vorherigen Nachbarn der aussortierten Funktion, die jetzt zueinander benachbart sind. Wenn am Sweepereignis die bisher minimale Funktion $f_{\sigma(1),i}$ beteiligt ist, dann registrieren wir den Parameterwert sowie die neue minimale Funktion. Aus diesen registrierten Angaben erhalten wir zum Schluß die einzelnen Abschnitte von $f_{\min}^{i,j}$ sowie diejenigen $f_{k,i}$, mit denen $f_{\min}^{i,j}$ auf dem jeweiligen Abschnitt übereinstimmt.

Nachdem $f_{\min}^{i,j}$ konstruiert ist, bestimmen wir $\varepsilon_{\sup}(p_i \wedge p_j)$ als

$$\varepsilon_{\sup}(p_i \wedge p_j) = \min\left\{ \frac{1}{2}\mathrm{dist}\big(p_i, p_j\big) \, , \, \sup_{x\in\mathbb{R}} f_{\min}^{i,j}(x) \right\} \quad .$$

Sofern $p_i \wedge p_j$ wirklich Kante eines Delaunaydiagramms ist, hat $f_{\min}^{i,j}$ irgendwo einen Wert > 0. Um $\varepsilon_{\sup}$ zu bestimmen, betrachten wir in jedem Abschnitt von $f_{\min}^{i,j}$ das Supremum der minimalen Funktion $f_{k,i}$. Spätestens an dieser Stelle des Programms kann es notwendig werden, den asymptotischen Wert von $f_{k,i}$ für $x \to \pm\infty$ zu berechnen. Dieser Wert kann das gesuchte Abschnitts-Supremum sein, wenn der Abschnitt unbeschränkt ist. Aufgrund von Lemma 5.11 brauchen wir das Abschnittssupremum nur an den beiden Enden des Abschnitts zu suchen, sowie im globalen Maximum von $f_{k,i}$, wenn das globale Maximum von $f_{k,i}$ im Abschnitt liegt.

Sind für die Kante $p_i \wedge p_j$ m Nachbarpunkte p_k zu betrachten, dann benötigt der Sweep $O(m \log m)$ Zeit: $O(m \log m)$ für die anfängliche Sortierung und $O(m)$ Sweepereignisse mit jeweils $O(\log m)$ Zeit für die Verwaltung der noch ausstehenden Ereignisse in einer Prioritätsschlange.

Statt bei $x = -\infty$ kann der Sweep auch bei der Nullstelle einer geeignet ausgewählten Funktion $f_{k,i}$ beginnen. Wenn für p_k die x-Koordinate $x_k < 0$ ist, dann hat $f_{k,i}$ eine Nullstelle, die bei einem Parameterwert $x_0 \in \mathbb{R}$ liegt. Für $x < x_0$ gilt $f_{k,i}(x) < 0$, so daß dieser Teil von g für uns nicht interessant ist. Wir können daher den Sweep bei x_0 beginnend auf $+\infty$ zulaufen lassen.

Die Bestimmung von ε_{sup} können wir als lineare Optimierung unter nichtlinearen Nebenbedingungen formulieren. Die Größe ε muß maximiert werden, die Nebenbedingungen sind $\varepsilon \leq f_{k,i}$, $k \neq i, j$. Auch hierbei brauchen wir nur diejenigen k betrachten, für die p_k Voronoinachbar von p_i oder p_j ist. Das Optimierungsproblem läßt sich mit einer modifizierten Form des Algorithmus von Megiddo [**Meg83**] lösen. Dieser Algorithmus behandelt ursprünglich lineare Programme in der Ebene (mit linearen Nebenbedingungen). Die nötigen Modifikationen sind in [**Wel96**] beschrieben. Als Laufzeit ergibt sich $O(m)$, hierin ist jedoch ein hoher konstanter Faktor enthalten. In der Ebene ist die durchschnittliche Anzahl von Voronoinachbarn eines Punktes kleiner als 6, wobei der Durchschnitt über alle Punkte von P genommen wird. Daher ist nur selten mit einem m zu rechnen, bei dem der asymptotische Vorteil des Megiddo-Verfahrens zum Tragen kommt.

KAPITEL 6

Polyedrische Rekonstruktion durch Delaunay-Stabilität

Der Stabilitätsbegriff aus Kapitel 5 läßt sich zur Rekonstruktion gestreut abgetasteter Flächen einsetzen. Die Grundidee ist, daß eine Menge „besonders stabiler" Delaunaydreiecke eine stückweise lineare Rekonstruktion der Fläche bildet. Um akzeptable Rekonstruktionen zu erhalten, muß das Fehlermodell verändert werden. Die Modifikationen bewirken unter anderem, daß das resultierende Verfahren sich automatisch an die lokale Abtastdichte anpaßt. In der Ebene erlaubt das modifizierte Fehlermodell in Verbindung mit interaktiver Schwellwertbestimmung die Rekonstruktion abgetasteter Kurven. Zur Flächenrekonstruktion in drei Dimensionen genügt dieser einfache Ansatz nicht. Hier verwenden wir ein an Schreiber und Brunnett [**Sch96, SB97**] angelehntes, volumenorientiertes Verfahren.

6.1. Ausgangspunkt

Abbildung 6.1 zeigt eine planare Kurve mit darauf verteilten Abtastpunkten. Daneben sehen wir die Delaunaytriangulierung der Abtastpunkte. Es wurde ein geeignetes ε gewählt. Die ε-stabilen Kanten sind schwarz, die ε-instabilen grau gezeichnet. Die stabilen Kanten bilden einen Polygonzug, der bis auf eine überschüssige Kante der abgetasteten Kurve folgt. Dieses Verhalten stabiler Delaunaykanten wollen wir zur Rekonstruktion abgetasteter Kurven ausnutzen.

Stabile Kanten weisen eine gewisse Verwandtschaft mit den Kanten des Gabrielgraphen [**GS69**] auf. Zu einer gegebenen Punktmenge P enthält der Gabrielgraph

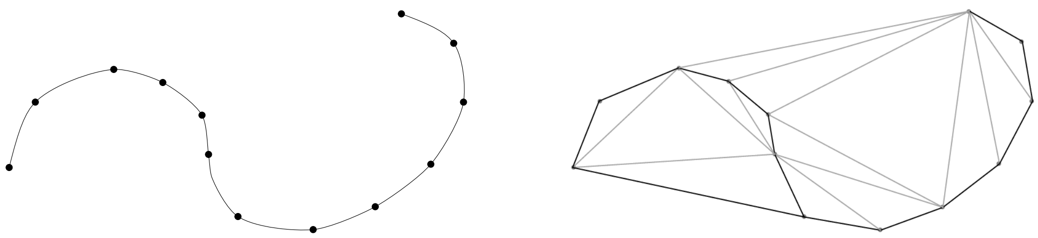


ABBILDUNG 6.1. Rekonstruktion einer Kurve durch stabile Delaunaykanten.

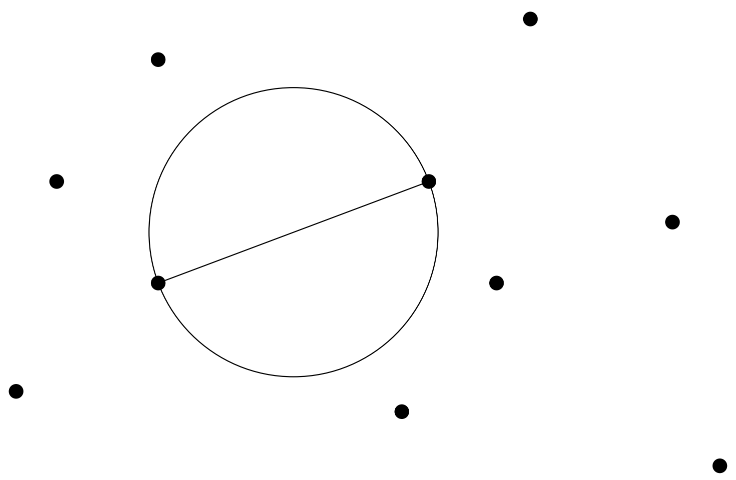


ABBILDUNG 6.2. Der minimale Umkreis einer Gabrielkante enthält keine weiteren Punkte.

diejenigen Kanten, deren minimaler Umkreis keine weiteren Punkte aus P enthält. Abbildung 6.2 zeigt eine Gabrielkante. Der minimale Umkreis einer Kante (bzw. einer Strecke) $p \wedge q$ ist definiert als derjenige Kreis, der $p \wedge q$ als Durchmesser hat. Er ist minimal in dem Sinn, daß er unter allen Kreisen durch p und q den kleinsten Radius hat. Da der Umkreis frei von weiteren Punkten ist, ist jede Gabrielkante nach Lemma 5.3 eine Delaunaykante. Hier zeigt sich die Ähnlichkeit von Gabrielkanten und stabilen Delaunaykanten. Für Stabilität fordern wir, daß ein beliebiger Punkt der Mittelsenkrechten Gleichung (ε-5.1) für ein gegebenes $\varepsilon > 0$ erfüllt. Bei einer Gabrielkante muß ein spezieller Punkt der Mittelsenkrechten, nämlich der Mittelpunkt der Kante, diese Gleichung für $\varepsilon = 0$ erfüllen.

Field [**Fie92**] verwendet eine räumliche Verallgemeinerung des Gabrielgraphen zur Flächenrekonstruktion. Er zieht solche Dreiecke $p_i \wedge p_j \wedge p_k$ in Erwägung, deren minimale Umkugel K keine weiteren Abtastpunkte enthält. Der Mittelpunkt der minimalen Umkugel liegt in der Ebene $p_i \vee p_j \vee p_k$ des Dreiecks, und die Punkte p_i, p_j und p_k liegen auf ihrem Rand. Der Umkreis des Dreiecks ist der Schnitt von ∂K mit der Ebene des Dreiecks, und somit ein Großkreis von ∂K.

Amenta und Bern [**AB98**] konstruieren zunächst das Voronoidiagramm von P und ordnen jedem Abtastpunkt ein oder zwei Voronoiknoten zu, die sie Pole nennen. Sie suchen dann Dreiecke, zu denen mindestens eine Umkugel weder Pole noch weitere Abtastpunkte enthält.

6.2. Fehlermodelle für Kurven– und Flächenrekonstruktion

Abbildung 6.3 zeigt verschiedene Rekonstruktionsversuche für eine Kurve, deren Abtastdichte stark variiert. Wählt man ε klein, dann bleiben zu viele unerwünschte Kanten stabil. Wählt man ε groß, dann werden in Bereichen mit dichter Abtastung erwünschte Kanten instabil. Wir benötigen einen Stabilitätsbegriff, der sich lokal an die Abtastdichte anpassen kann. Dies erreichen wir durch ein adaptives Fehlermodell.

Für jede Delaunaykante legen wir ein eigenes Fehlermodell fest. In diesem Fehlermodell hat jeder Punkt $p_i \in P$ eine individuelle Fehlerschranke ε_i. Sei $p_i \wedge p_j$ die betrachtete Kante. Wir setzen zunächst $\varepsilon_i = \varepsilon_j = 0$, d. h. die Kante selbst darf sich nicht bewegen. Punkte p_k, die weder Voronoinachbarn von p_i noch von p_j sind, erhalten ebenfalls $\varepsilon_k = 0$. Praktisch bedeutet das, daß wir diese Punkte ignorieren und nur die Voronoinachbarn von p_i und von p_j berücksichtigen. Für einen solchen Voronoinachbarn p_k messen wir zunächst seinen Abstand d_k von der Kante $p_i \wedge p_j$, vergleiche Abbildung 6.4. Dann setzen wir die Fehlerschranke fest als

$$\varepsilon_k := \begin{cases} \rho d_k & , d_k \leq \operatorname{dist}(p_i, p_j) \\ \rho \left(2\operatorname{dist}(p_i, p_j) - d_k\right) & , \operatorname{dist}(p_i, p_j) < d_k \leq 2\operatorname{dist}(p_i, p_j) \\ 0 & , \text{sonst} \end{cases} \tag{6.1}$$

Dies stellt eine Hutfunktion dar, die in Abbildung 6.5 zu sehen ist. Der Faktor ρ ist ein globaler Parameter, der für alle betrachteten Kanten und alle p_k gleich gewählt wird. Damit definieren wir

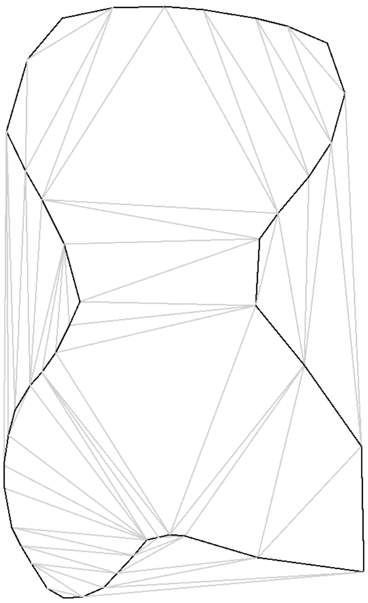
DEFINITION 6.1. *Eine Kante $p_i \wedge p_j$ heißt* ρ***-stabil****, wenn ein Punkt c ihrer Mittelsenkrechten die Gleichung*

$$\operatorname{dist}(c, p_i) = \operatorname{dist}(c, p_j) < \min_{k \neq i,j} \operatorname{dist}(c, p_k) - \varepsilon_k \tag{6.2}$$

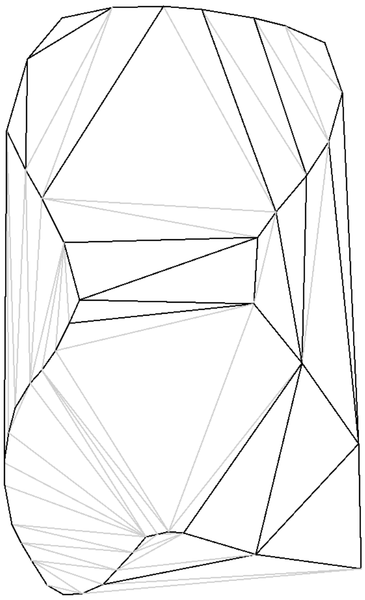
erfüllt.

Ist $p_i \wedge p_j$ ρ-stabil, dann existeirt eine Kreisscheibe um c, die (die 0-Scheiben von) p_i und p_j enthält und keine weitere ε_k-Scheibe schneidet.

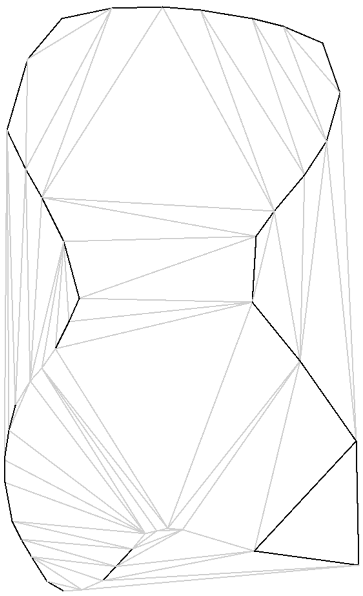
Wird für ein p_k die Schranke ε_k größer oder gleich seinem Abstand d_k zur aktuellen Kante $p_i \wedge p_j$, dann kann p_k auf die Kante verschoben werden. Damit ist $p_i \wedge p_j$ offensichtlich instabil. Der Fall $\varepsilon_k \geq d_k$ tritt immer dann auf, wenn $\rho \geq 1$ ist und ein Punkt p_k genügend nah an $p_i \wedge p_j$ liegt. Insbesondere muß eine Kante kürzer als alle zu ihr adjazenten Kanten sein, wenn sie für $\rho \geq 1$ stabil soll.



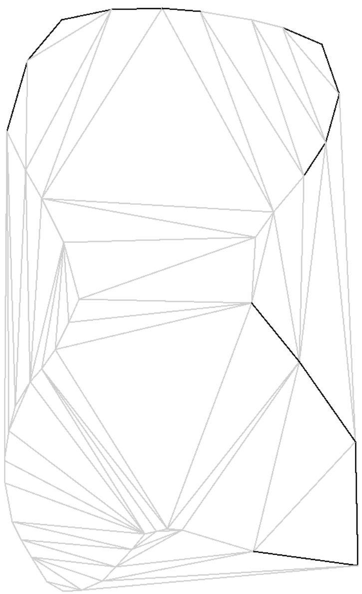
Gewünschte Rekonstruktion



$\varepsilon = 3$



$\varepsilon = 15$



$\varepsilon = 30$

ABBILDUNG 6.3. Rekonstruktion durch ε-Stabilität. Stabile Kanten sind schwarz, instabile grau gezeichnet. Aufgrund der unterschiedlich dichten Abtastung läßt sich kein geeigneter Schwellwert ε finden.

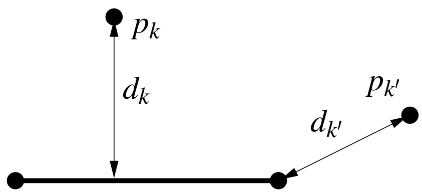


ABBILDUNG 6.4. Abstand von der aktuellen Kante $p_i \wedge p_j$.

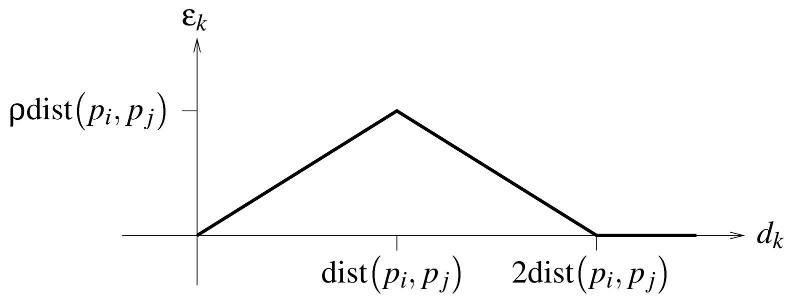


ABBILDUNG 6.5. Die Fehlerschranke ε_k in Abhängigkeit vom Abstand d_k zur betrachteten Kante.

Von zwei zueinander adjazenten Kanten kann dann höchstens eine stabil sein, so daß ein Polygonzug aus stabilen Kanten ausgeschlossen ist. Wir lassen daher nur $0 \leq \rho < 1$ zu, und geben ρ als Prozentzahl an.

Für die praktische Einsetzbarkeit eines Rekonstruktionsverfahrens ist es wichtig, daß das Ergebnis unabhängig von der — meist willkürlichen — Wahl des Koordinatensystems und des Maßstabs ist. Die Stabilität oder Instabilität einer Kante soll also invariant unter Translation, Rotation, Spiegelung[1] und Streckung sein.

SATZ 6.2. *ρ-Stabilität ist invariant unter Translation, Rotation, Spiegelung und Streckung.*

Beweis: Translation, Rotation und Spiegelung lassen die Länge $\mathrm{dist}(p_i, p_j)$ der Kante $p_i \wedge p_j$ sowie die Abstände d_k der anderen Punkte zu ihr unverändert. Da die Fehlerschranken ε_k nur von diesen Größen abhängen, bleiben sie ebenfalls gleich. Die Existenz eines Punktes c nach Gleichung (6.2) ist daher invariant unter Translation, Rotation und Spiegelung. Bei einer Streckung mit Streckfaktor $\lambda \neq 0$ werden die Kantenlänge und die Abstände zu $\lambda\mathrm{dist}(p_i, p_j)$ und λd_k. An Gleichung (6.1) sieht man, daß die Fehlerschranken zu $\lambda\varepsilon_k$ werden. Setzen wir

[1]Eine Spiegelung ergibt sich z. B. beim Wechsel von einem rechtsorientierten zu einem linksorientierten Koordinatensystem.

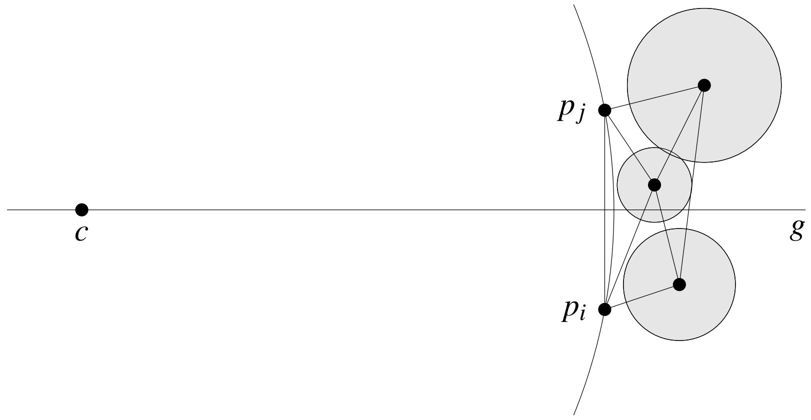


ABBILDUNG 6.6. Die Randkante $p_i \wedge p_j$ ist dadurch stabil, daß der Punkt c beliebig weit nach links ausweichen kann.

diese Größen in Gleichung (6.2) ein, dann kürzt sich der Streckfaktor heraus und wir erhalten die ursprüngliche Gleichung. Damit ist die Invarianz unter Streckung gezeigt. ∎

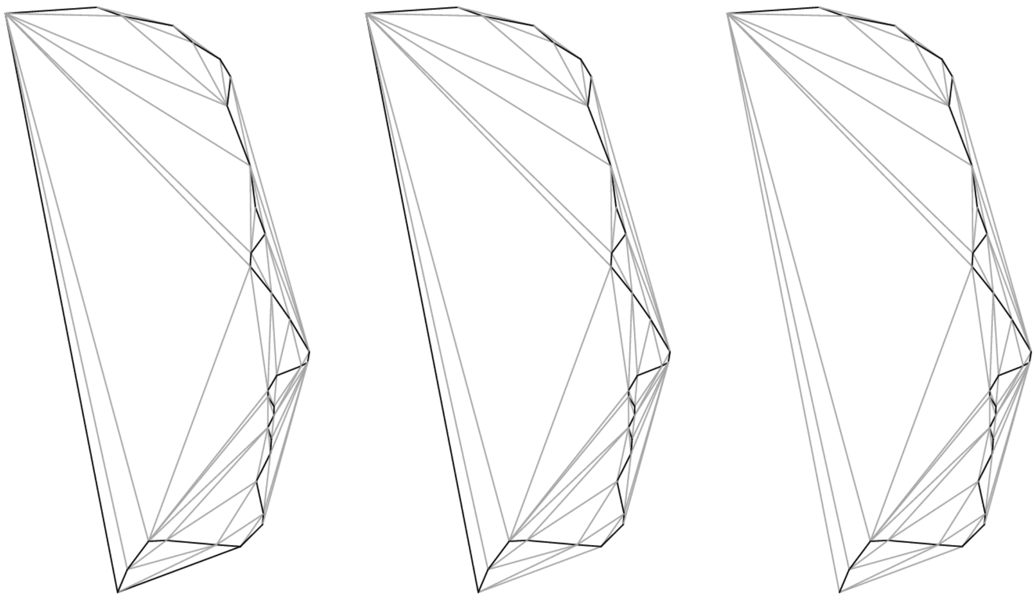
In Verbindung mit der Tatsache, daß wir nur die Voronoinachbarn von p_i und von p_j betrachten, bewirkt die Maßstabsunabhängigkeit eine lokale Adaption des Verfahrens an die Abtastdichte. Diesen Effekt kann man in Abbildung 6.3 oben links sehen. Die „gewünschte Rekonstruktion" der unterschiedlich dicht abgetasteten Kurve wurde durch ρ-Stabilität berechnet.

Bei Randkanten des Delaunaydiagramms kann der gesuchte Punkt c auf der Mittelsenkrechten sehr weit außerhalb des Diagramms liegen. Dadurch wird es der Kreisscheibe um c oft möglich, den auf der „Innenseite" liegenden ε_k-Scheiben auszuweichen. Dies ist in Abbildung 6.6 dargestellt. Wir bezeichnen eine solche Lage des Punktes c als *exzentrisch*. Bei inneren Kanten des Delaunaydiagramms tritt dieser Effekt nicht so extrem auf, da immer Voronoinachbarn von p_i und p_j auf beiden Seiten der Gerade $p_i \vee p_j$ liegen. Ein Beispiel für eine auf diese Weise stabile Randkante ist die überschüssige Kante in Abbildung 6.1. Ein weiteres Beispiel zeigt Abbildung 6.7 a). Um übermäßig stabile Randkanten zu vermeiden, begrenzen wir den zulässigen Abstand von c zur Kante durch eine Exzentrizitätsschranke χ. Das führt auf folgende Definition:

DEFINITION 6.3. *Eine Kante $p_i \wedge p_j$ heißt **ρ-χ-stabil**, wenn über Gleichung* (6.2) *hinaus zusätzlich*

$$\operatorname{dist}(c, p_i \vee p_j) \le \chi \operatorname{dist}(p_i, p_j) \tag{6.3}$$

gilt.



a) $\chi = \infty$, $\rho = 30\%$ b) $\chi = 2$, $\rho = 30\%$ c) $\chi = 1$, $\rho = 30\%$

ABBILDUNG 6.7. Rekonstruktionen mit verschiedenen Exzentrizitätsschranken χ.

Die eigentliche Schranke wird in dieser Definition relativ zur Länge der Kante festgelegt. Dadurch erreichen wir auch bei der Exzentrizitätsschranke ein maßstabsunabhängiges Verhalten und eine Adaption an die lokale Punktdichte. Die Invarianz der Exzentrizitätsschranke unter Translation, Rotation und Spiegelung ist offensichtlich.

BEMERKUNG 6.4. Für $\chi = 0$ muß c der Mittelpunkt der Kante $p_i \wedge p_j$ sein. Setzen wir zusätzlich $\rho = 0$, dann sind die stabilen Kanten genau die Gabrielkanten von P.

Zur Rekonstruktion von Polygonzügen in der Ebene erweist sich ein einfacher Schwellwertansatz als ausreichend. Die Rekonstruktion besteht hierbei aus denjenigen Delaunaykanten, die für ein bestimmtes ρ und χ stabil sind. Abbildungen 6.8 bis 6.11 zeigen weitere Beispiele.

Zu gegebenem ρ und χ testen wir die ρ-χ-Stabilität einer Kante $p_i \wedge p_j$ analog zu Algorithmus 5.1, indem wir bestimmte Hyperbelzweige mit der Mittelsenkrechten g von $p_i \wedge p_j$ schneiden. Im Unterschied zu Algorithmus 5.1 haben wir jetzt für jeden Hyperbelzweig $Z_{k,i}$ einen anderen Wert ε_k zu betrachten. Die Exzentrizität läßt sich direkt am Durchschnitt der Intervalle $I_k \subset \mathbb{R}$ ablesen. Bezeichne I

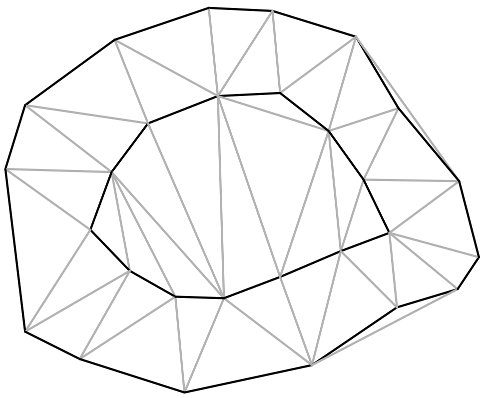


ABBILDUNG 6.8. $\rho = 30\%, \chi = 1$

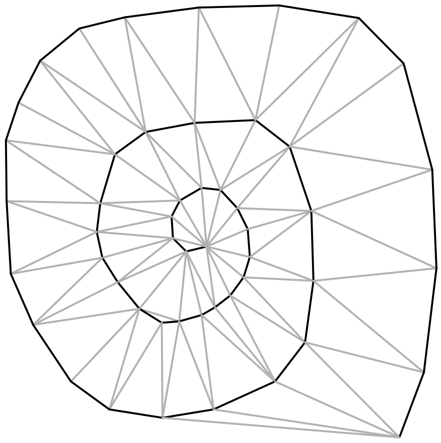


ABBILDUNG 6.9. $\rho = 30\%, \chi = 1$

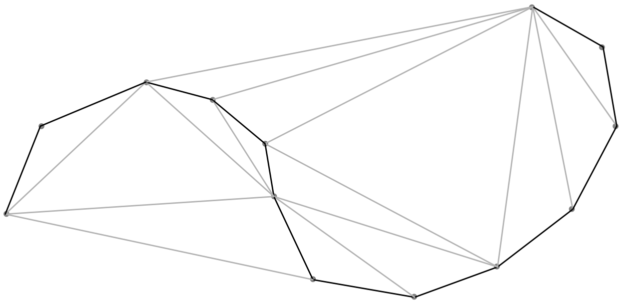


ABBILDUNG 6.10. $\rho = 30\%, \chi = 0{,}8$

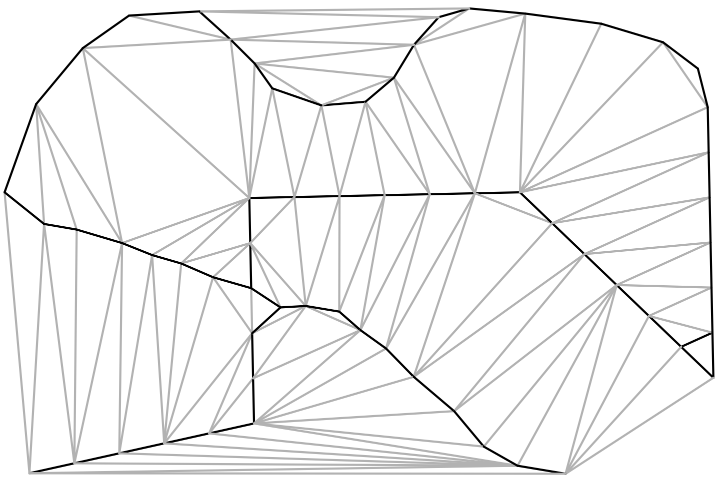


ABBILDUNG 6.11. $\rho = 30\%, \chi = 0{,}3$

Algorithmus 6.1 ρ-Stabilitätstest

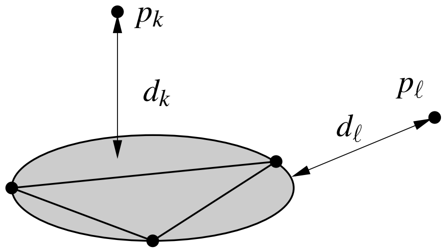
Eingabe: Delaunaydiagramm von $P \subset \mathbb{R}^2$, Kante $p_i \wedge p_j$ des Delaunaydiagramms, ρ und χ.
Ausgabe: „stabil" oder „instabil".

$I := [-\chi, \chi]$
for jedes p_k, das zu p_i oder zu p_j adjazent ist **do**
  Bestimme den Abstand d_k von p_k zur Kante.
  Berechne ε_k aus d_k nach Gleichung (6.1).
  Bestimme das zu p_k und ε_k gehörende Intervall I_k.
  $I := I \cap I_k$.
  if $I = \emptyset$ **then**
    $p_i \wedge p_j$ ist instabil. **stop**.
  endif
endfor
$p_i \wedge p_j$ ist stabil.

diesen Durchschnitt. Die I_k repräsentieren Abschnitte von g vermöge der Parametrisierung nach Gleichung (5.6). Diese Parametrisierung ist bereits relativ zur Länge von $p_i \wedge p_j$. Ein Zahlenwert von $\pm 1{,}7$ entspricht z. B. einem Punkt $c \in g$ mit Abstand $\operatorname{dist}(c, p_i \vee p_j) = 1{,}7 \operatorname{dist}(p_i, p_j)$ von der Kante. Eine vorgegebene Exzentrizitätsschranke χ ist also genau dann eingehalten, wenn der Durchschnitt von I mit dem Intervall $[-\chi, \chi]$ nicht leer ist. Algorithmus 6.1 führt diesen Test durch. Bei instabilen Kanten ist es oft nicht nötig, alle adjazenten Punkte zu betrachten. Im günstigsten Fall reicht ein Punkt aus, um die Instabilität festzustellen. Der Algorithmus initialisiert (die Variable) I zu $[-\chi, \chi]$ und schneidet fortlaufend I mit den Intervallen I_k. Wird dabei $I = \emptyset$, dann ist die Kante instabil und der Algorithmus bricht ab.

6.3. Flächenrekonstruktion durch Stabilität

Die in diesem Abschnitt vorgestellte Flächenrekonstruktion beginnt mit einer Delaunaytetraedrisierung der Abtastpunkte. Das Fehlermodell zur Betrachtung eines Dreiecks $\delta = p_i \wedge p_j \wedge p_\ell$ definieren wir analog zum planaren Fall. Die Eckpunkte p_i, p_j, p_ℓ erhalten als Fehlerschranke 0. Für ihre Voronoinachbarn gilt je ein individuelles ε_k, das wiederum von einem globalen Parameter ρ abhängt. Alle



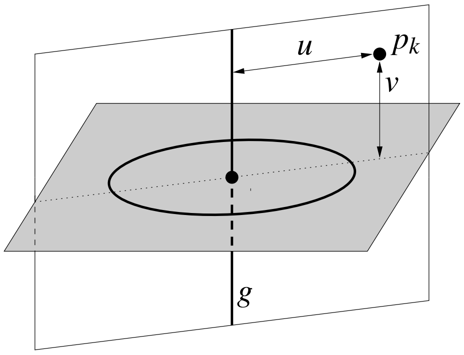


ABBILDUNG 6.12. Im Raum wird der Abstand zur Umkreisscheibe des betrachteten Dreiecks gemessen.

ABBILDUNG 6.13. Lokales Koordinatensystem in der Ebene $E = g \vee p_k$.

übrigen Punkte von P haben ebenfalls die Fehlerschranke 0. Stabilität unter diesem Fehlermodell bedeutet, daß ein Punkt $c \in \mathbb{R}^3$ existiert mit

$$\operatorname{dist}(c, p_i) = \operatorname{dist}\big(c, p_j\big) = \operatorname{dist}(c, p_\ell) < \min_{k \neq i,j,\ell} \operatorname{dist}(c, p_k) - \varepsilon_k \quad . \tag{6.4}$$

Diejenigen Punkte des $\mathbb{R}^3$, die zu p_i, p_j und p_ℓ äquidistant sind, bilden eine Gerade g, die Mittelsenkrechte des Dreiecks. Sie steht orthogonal zur Ebene $\bigvee \delta$ und schneidet diese im Umkreismittelpunkt des Dreiecks. Wenn ein Punkt c Gleichung (6.4) erfüllt, liegt er auf g. Sein Abstand zu p_i, p_j und p_ℓ ist derselbe wie zu jedem anderen Punkt des Umkreises. Daher ist δ nicht erst dann instabil, wenn die ε_k-Kugel um p_k das Dreieck selbst schneidet. Es genügt, daß die ε_k-Kugel die Umkreisscheibe von δ schneidet. Aus diesem Grund messen wir den Abstand d_k von p_k nicht zu δ, sondern zur Umkreisscheibe, siehe Abbildung 6.12. Aus d_k und ρ bilden wir ε_k wie im planaren Fall. An Stelle der Kantenlänge in Gleichung (6.1) tritt der Durchmesser des Umkreises. Die Exzentrizität von c ist sein Abstand zum Umkreismittelpunkt (gleich dem Abstand zur Dreiecksebene).

DEFINITION 6.5. *Ein Dreieck $\delta = p_i \wedge p_j \wedge p_\ell$ heißt* **ρ-χ-stabil**, *wenn ein Punkt c existiert mit*

$$\operatorname{dist}(c, p_i) = \operatorname{dist}\big(c, p_j\big) = \operatorname{dist}(c, p_\ell) < \min_{k \neq i,j,\ell} \operatorname{dist}(c, p_k) - \varepsilon_k$$

und

$$\operatorname{dist}\Big(c, \bigvee \delta\Big) \leq 2\chi r \quad .$$

Hierbei ist r der Umkreisradius von δ,

$$\varepsilon_k := \begin{cases} \rho d_k & , d_k \leq 2r \\ \rho\,(4r - d_k) & , 2r < d_k \leq 4r \\ 0 & , \textit{sonst} \end{cases}$$

und d_k der Abstand von p_k zur Umkreisscheibe von δ.

Wie in der Ebene gilt auch hier

SATZ 6.6. ρ-χ-*Stabilität ist invariant unter Translation, Rotation, Spiegelung und Streckung.*

Ein Punkt p_k spannt mit der Mittelsenkrechten g eine Ebene E_k auf, siehe Abbildung 6.13. Alle von p_k abhängenden Größen in Definition 6.5 lassen sich innerhalb dieser Ebene bestimmen. Diese Tatsache nutzen wir aus, um den Stabilitätstest auf den planaren Fall zu reduzieren. Wir übertragen jeden Voronoinachbarn p_k in ein lokales (u,v)-Koordinatensystem. Die u-Achse ist der Schnitt von E_k mit $\bigvee\delta$, die v-Achse ist die Mittelsenkrechte g. Wenn alle Voronoinachbarn übertragen sind, können wir die ρ-χ-Stabilität von δ in der (u,v)-Ebene berechnen. Sie ist nämlich äquivalent zur Stabilität der Kante $(-r,0)\wedge(r,0)$ — die übertragene Umkreisscheibe — gegenüber den übertragenen Punkten.

Ein einfaches Schwellwertverfahren, wie wir es im $\mathbb{R}^2$ verwendet haben, liefert im $\mathbb{R}^3$ oft unbefriedigende Ergebnisse. Abbildung 6.14 zeigt ein typisches Beispiel. Neben dem unerwünschten Dreieck zwischen Nase und Oberlippe gibt es noch weitere unerwünschte Dreiecke im Inneren des Kopfes, die in der Abbildung naturgemäß nicht sichtbar sind. Erhöht man hier den Schwellwert ρ, dann treten zuerst Löcher in der gewünschten Fläche auf, bevor die unerwünschten Dreiecke unterdrückt werden.

Abhilfe kann ein volumenorientiertes Verfahren schaffen, wie es Schreiber und Brunnett [**Sch96, SB97**] verwenden. Dieses sogenannte *Spannbaumverfahren* rekonstruiert das abgetastete Objekt in Form einer zusammenhängenden Teilmenge der Delaunaytetraeder. Die Tetraeder bilden ein Polyeder mit einem geschlossenen Dreiecksnetz als Oberfläche. Dieses Dreiecksnetz dient als Rekonstruktion der abgetasteten Fläche. Wir kombinieren die Vorgehensweise des Spannbaumverfahrens mit ρ-χ-Stabilität als Entscheidungskriterium für die Wahl der Tetraederteilmenge.

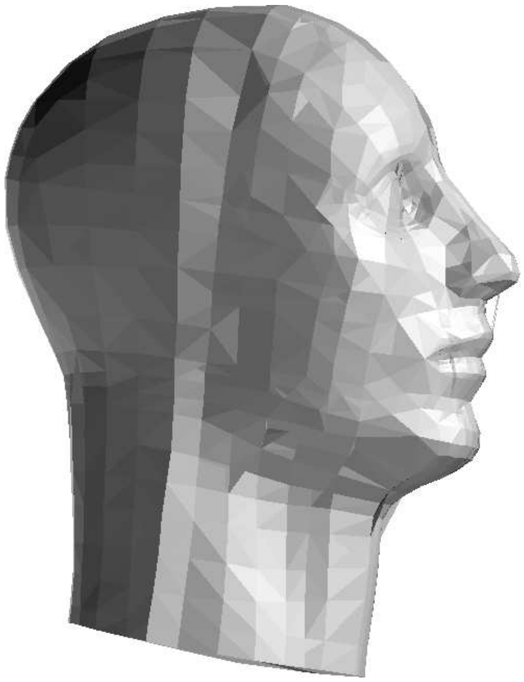


ABBILDUNG 6.14. Schwellwertrekonstruktion im $\mathbb{R}^3$.

Das Spannbaumverfahren basiert auf einem abstrakten Graph. Seine Kanten sind die Dreiecke der Delaunaytetraedrisierung, seine Knoten sind die Delaunaytetraeder sowie als Sonderknoten das Komplement der konvexen Hülle von P. Wir wählen ein festes χ und ordnen jedem Dreieck als Gewicht das supremale ρ zu, bis zu dem das Dreieck ρ-χ-stabil ist. Auf dem so gewichteten Graph wird ein minimaler Spannbaum berechnet. Da der Spannbaum alle Knoten des abstrakten Graphen enthält, überdeckt er gewissermaßen den gesamten $\mathbb{R}^3$. Aus dem Spannbaum wird seine schwerste Kante, also das stabilste Dreieck, entfernt. Dabei zerfällt der Baum in zwei Komponenten, die je einem Teilvolumen des $\mathbb{R}^3$ entsprechen. Eines dieser Teilvolumen ist das rekonstruierte Objekt, das andere sein Komplement. Das Komplement ist daran zu erkennen, daß es den Sonderknoten enthält.

Besonders bei rasterförmiger Abtastung einer Fläche ergeben sich häufig Delaunaytetraeder, deren vier Ecken fast auf einem Kreis liegen. Ein solches Tetraeder ist in der Regel sehr flach, und seine Dreiecke liegen nahe an der ursprünglichen Fläche. Das führt dazu, daß die Dreiecke sehr stabil sind. Dadurch kann das Tetraeder leicht als eine der beiden Spannbaumkomponenten identifiziert werden. Der Algorithmus liefert dann ein einziges Tetraeder als Lösung und den gesamten Rest des Raumes als Komplement. Um dies zu verhindern, kann der Benutzer

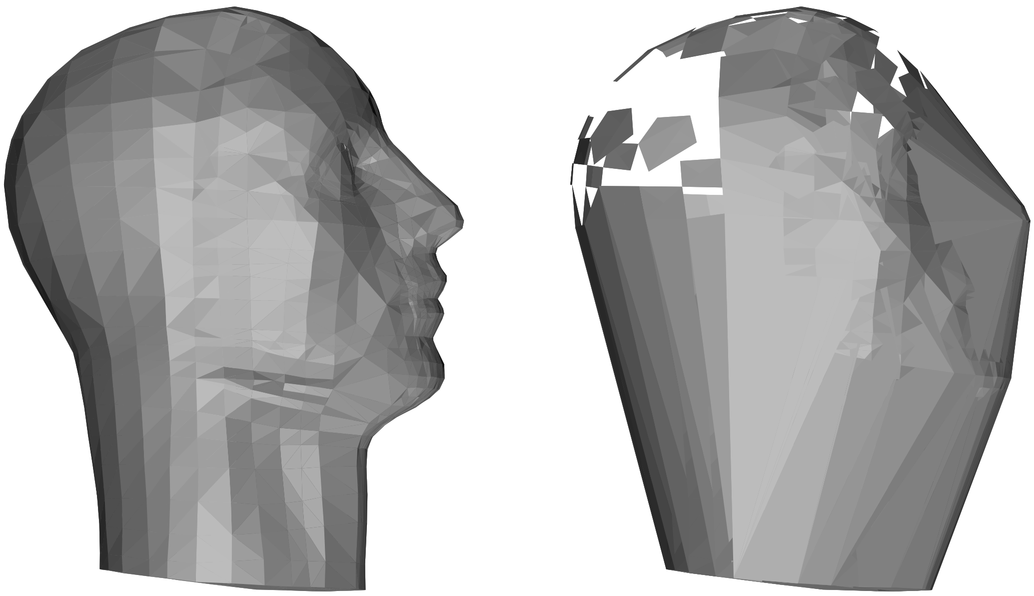


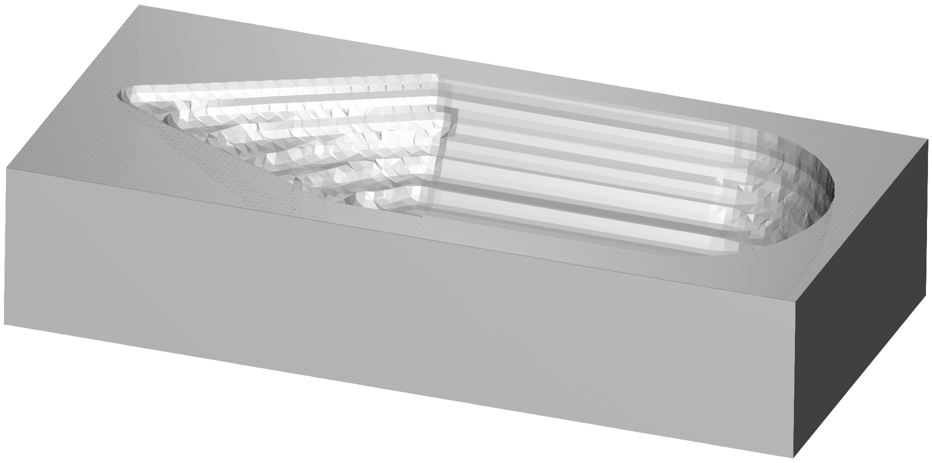
ABBILDUNG 6.15. 1487 Abtastpunkte, $\chi = 1$, Mindestkomponentengröße 100.

eine Mindestgröße für die beiden Komponenten festlegen. Wenn jetzt das Entfernen eines Dreiecks aus dem Spannbaum dazu führen würde, daß eine der beiden Komponenten zu wenige Tetraeder hätte, dann wird dieses Dreieck im Spannbaum gelassen und das nächst-stabile Dreieck entfernt.

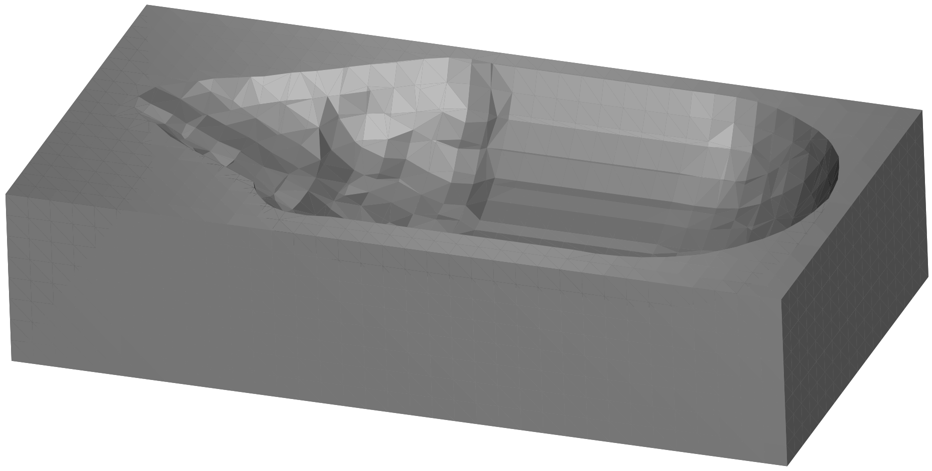
Abbildungen 6.15–6.17 zeigen Dreiecksnetze, die mit dem Spannbaumverfahren berechnet wurden. Um die Facettenstruktur erkennbar zu machen, sind die Netze mit Flat Shading dargestellt. Die gezeigten Komplemente sind jeweils auf die konvexe Hülle der Abtastpunkte bezogen.

Abbildung 6.15 liegt der Datensatz aus Abbildung 6.14 zugrunde. Das Spannbaumverfahren garantiert eine geschlossene Oberfläche. Das Komplement zerfällt ohne den äußeren Spannbaumknoten in mehrere Zusammenhangskomponenten.

Abbildung 6.16 zeigt eine Gußform, die aus einem Materialblock gefräst wird. Es ist ein Zwischenstadium aus dem Fräsprozeß dargestellt. Die Gußform wurde gerastert abgetastet. Mit 2800 Rasterpunkten ist die Abtastung viel zu grob, um die Fräsbahnen als solche zu erfassen. Daher ergibt sich die deutlich sichtbare Abweichung der Rekonstruktion von der abgetasteten Form. Das Komplement entspricht in diesem Fall einem hypothetischen Gußteil, das mit der Rekonstruktion gefertigt wurde.



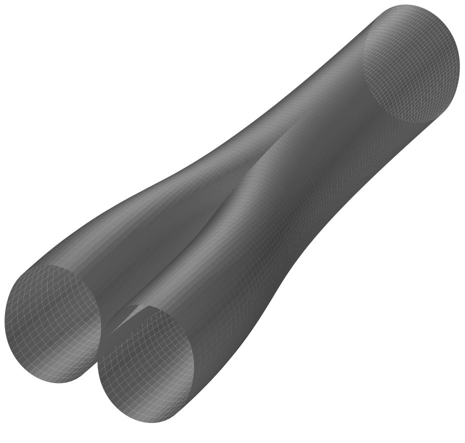
Gußform (ursprüngliches Objekt)



Rekonstrukion

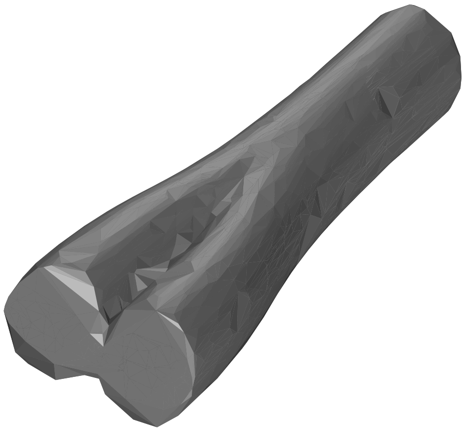
Komplement der Rekonstruktion (verschiedene Ansichten)

ABBILDUNG 6.16. 2800 Abtastpunkte, $\chi = 0{,}5$, Mindestkomponentengröße 10.

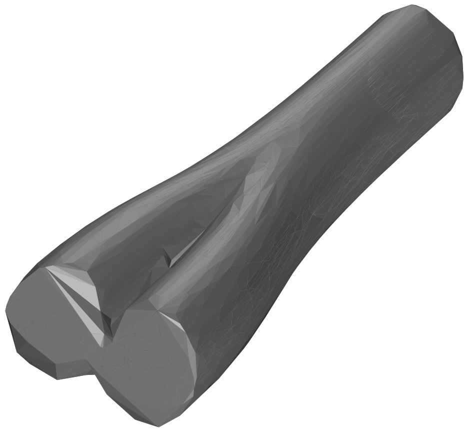


Ursprüngliche Fläche

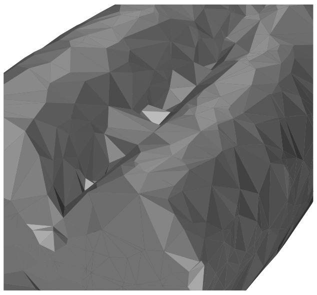
Rekonstruktion



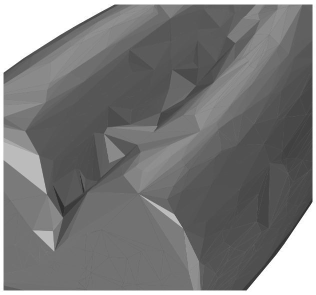
ABN-Optimierung
der Rekonstruktion



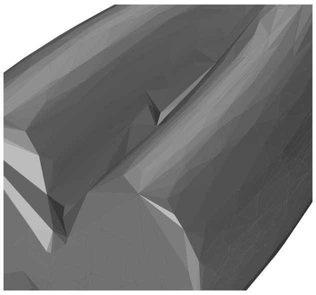
TAC-Optimierung
der Rekonstruktion



Rekonstruktion



ABN-Optimierung



TAC-Optimierung

ABBILDUNG 6.17. 2000 Abtastpunkte, $\chi = 1$, Mindestkomponentengröße 50.

In Abbildung 6.17 wird eine offene Fläche rekonstruiert. Die Rekonstruktion ist notwendigerweise ein geschlossenes Dreiecksnetz. Als weiteren Bearbeitungsschritt sehen wir eine Optimierung des Netzes, hier mit den Zielfunktionen ABN und TAC (vgl. Kapitel 1 und 4). Die Detailausschnitte zeigen eine deutliche Qualitätsverbesserung durch den Optimierungsschritt.

KAPITEL 7

Effiziente Aufzählung polygonaler Hüllen

Dieses Kapitel befaßt sich mit der Aufzählung polygonaler Hüllen einer gegebenen Punktmenge in der Ebene. Unter einer polygonalen Hülle verstehen wir ein einfaches Polygon, dessen Ecken zur gegebenen Punktmenge gehören und das die restlichen gegebenen Punkte umschließt. Es wird ein Algorithmus vorgestellt, der alle polygonalen Hüllen von n Punkten aufzählt. Dafür benötigt er im schlechtesten Fall pro Hülle $O(n^2)$ Zeit und insgesamt $O(n^2)$ Speicherplatz. Varianten des Algorithmus können zur zufälligen Erzeugung von Polygonen und zur heuristischen Suche nach nicht-konvexen Triangulierungsgebieten dienen.

7.1. Das Problem

Die Aufzählung diskreter Strukturen ist Grundvoraussetzung für erschöpfende Suche und kann als Basis für die Erzeugung (pseudo-)zufälliger Beispiele dienen. Ein Aufzählungsalgorithmus darf selbstverständlich kein Element der aufzuzählenden Menge „übersehen". Aus Effizienzgründen sollte er jedes Element nur einmal aufzählen. Ist die gewünschte Ausgabe etwa die aufzuzählende Menge (ohne Duplikate), dann erfordert die nötige Duplikateliminierung zusätzlichen Zeitaufwand. Noch gravierender kann der Mehraufwand sein, wenn die Aufzählung zur Lösung eines Optimierungsproblems durch erschöpfende Suche eingesetzt wird. Bei garantierter Einfachaufzählung wird man hier die Zielfunktion auswerten, sobald ein neues Element aufgezählt ist. Nur das bisher optimale Element muß gespeichert werden. Bei Mehrfachaufzählung steht man vor der Wahl, entweder die Zielfunktion für jede Aufzählung eines einzelnen Elements auszuwerten, oder zum Zweck der Duplikateliminierung alle aufgezählten Elemente zu speichern.

Die Erzeugung (pseudo-)zufälliger Polygone ist eng mit der Aufzählung verwandt. Arbeitet ein Aufzählungsalgorithmus, wie hier der Fall, nach einem verzweigend rekursiven Prinzip, dann kann er auch zur zufälligen Erzeugung nur eines Ergebnisses eingesetzt werden. In den einzelnen Rekursionsknoten werden nicht alle Verzweigungen verfolgt, sondern jeweils eine einzige zufällig ausgewählt. Zhu, Sundaram, Snoeyink und Mitchell [**ZSSM96**] erzeugen zufällige

x-monotone Polygone mit einer gegebenen Menge von Ecken. Ihr Algorithmus wählt in jedem Schritt eine von zwei möglichen Verzweigungen. In einem Vorverarbeitungsprozess werden die Anzahlen der Polygone berechnet, die nach Wahl des einen oder des anderen Zweigs noch möglich sind. Sind diese Anzahlen etwa h_1 und h_2, dann wählt der Algorithmus den ersten Zweig mit Wahrscheinlichkeit $h_1/(h_1+h_2)$ und den zweiten mit Wahrscheinlichkeit $h_2/(h_1+h_2)$. Hieraus resultiert eine Gleichverteilung der möglichen Polygone. Würde der Algorithmus jeweils beide Zweige rekursiv verfolgen, dann würde er alle x-monotonen Polygone aufzählen.

Im Gegensatz zu Zhu et al. fordern wir nur, daß ein Polygon eine Teilmenge der Eingabemenge P als Ecken verwendet. Der Rest von P muß im Polygon enthalten sein.

DEFINITION 7.1. *Sei P eine endliche Punktmenge im $\mathbb{R}^2$ mit $|P| \geq 4$. Eine* ***polygonale Hülle*** *von P ist ein einfaches Polygon ω mit $E(\omega) \subseteq P \subset \omega$.*

BEMERKUNG 7.2. *Der Begriff* Hülle *ist dadurch motiviert, daß die polygonalen Hüllen einerseits P enthalten und andererseits eine bestimmte Minimaleigenschaft erfüllen. Zu einem Polygon ω sei $E_P(\omega) := E(\omega) \cup (P \cap \partial\omega)$ die durch P induzierte Eckenmenge von ω. Wir betrachten also alle Punkte von P, die auf dem Rand von ω liegen, als Ecken von ω. Diese Ecken sind möglicherweise degeneriert. Durch*

$$\omega_1 \sqsubseteq \omega_2 \quad :\Longleftrightarrow \quad \omega_1 \subseteq \omega_2 \text{ und } |E_P(\omega_1)| \leq |E_P(\omega_2)|$$

wird jetzt eine Halbordnung auf den einfachen Polygonen definiert. Innerhalb derjenigen einfachen Polygone, die P enthalten, sind die polygonalen Hüllen die Minima bezüglich dieser Halbordnung.

Beweis: Sei $P \subset \omega$ und seien $z_1, z_2, z_3 \in E_P(\omega)$ drei aufeinanderfolgende Ecken von ω. Angenommen z_2 ist nicht in P. Dann ist der minimale Abstand ε der Kanten $z_1 \wedge z_2$ zu $P \setminus \{z_1\}$ und $z_2 \wedge z_3$ zu $P \setminus \{z_3\}$ ungleich Null. Verschieben wir jetzt z_2 entlang der Winkelhalbierenden der beiden Kanten um weniger als ε in das Innere von ω, dann enthält das geänderte Polygon ω' immer noch P. (Die Verschiebung muß klein genug gewählt werden, daß das Polygon einfach bleibt.) Die Anzahl der Eckpunkte ändert sich nicht, es ist also $\omega' \sqsubset \omega$. Ein Minimum bezüglich der Halbordnung muß daher $E_P(\omega) \subseteq P$ erfüllen, d. h. es muß eine polygonale Hülle sein.

Sei ω polygonale Hülle von P und $P \subset \omega' \sqsubseteq \omega$. Dann ist $E_P(\omega) \subseteq P \subset \omega' \subseteq \omega$. Da die Punkte in $E_P(\omega)$ Randpunkte von $\omega \supseteq \omega'$ sind, können sie nicht im Innern von ω' liegen. Es ist also $E_P(\omega) \subseteq E_P(\omega')$, und wegen $|E_P(\omega)| \geq |E_P(\omega')|$ müssen beide Eckenmengen gleich sein. Wir sehen, daß aus $\omega' \sqsubseteq \omega$ stets $\omega' = \omega$ folgt. Daher ist ω ein Minimum bezüglich der Halbordnung. ∎

7.2. Der Aufzählalgorithmus

Wir wollen zu einer gegebenen Punktmenge P sämtliche polygonalen Hüllen finden. Dabei stellen wir Polygone als geschlossene Polygonzüge bzw. als zyklische Folgen ihrer Eckpunkte dar. Degenerierte Ecken lassen wir ausdrücklich zu.

Je nach Lage der Punkte von P kann die Anzahl polygonaler Hüllen stark variieren. Enthält P genau die Ecken eines konvexen Polygons ω, dann ist ω die einzige polygonale Hülle von P. Nehmen wir einen inneren Punkt $p \in \operatorname{int}\omega$ hinzu, dann gibt es $|P|+1$ polygonale Hüllen. Neben ω erhalten wir weitere Hüllen, indem wir je eine Kante $p_i \wedge p_{i+1}$ von ω durch die Kanten $p_i \wedge p$ und $p \wedge p_{i+1}$ ersetzen.

Unsere erste Beobachtung ist, daß die Extrempunkte von P Ecken aller polygonalen Hüllen sein müssen.

SATZ 7.3. *Sei $p \in P$ eine möglicherweise degenerierte Ecke der konvexen Hülle von P. Für jede polygonale Hülle ω von P gilt $p \in E_P(\omega)$.*

Beweis: Einerseits ist die konvexe Hülle Obermenge jeder polygonalen Hülle, andererseits ist $p \in P \subset \omega$. Eine Stützgerade g an $\bigwedge P$ durch p ist daher auch Stützgerade an ω. ∎

In diesem Kapitel stellen wir ein einfaches Polygon als zyklische Folge seiner Eckpunkte dar. Der Aufzählungsalgorithmus beginnt mit der minimal möglichen Folge. Diese Folge entspricht der konvexen Hülle von P. Sie wird rekursiv erweitert, um andere polygonale Hüllen zu erhalten.

DEFINITION 7.4. *Sei $\zeta = (z_1, \dots, z_m)$ eine zyklische Folge von Punkten aus P. Durch ζ wird in der üblichen Weise ein geschlossener Polygonzug definiert. Ist dieser Polygonzug einfach, d. h. frei von Selbstüberschneidungen, dann bildet er den Rand eines einfachen Polygons $\omega(\zeta)$. Mit $E(\zeta)$ bezeichnen wir die Menge $\{z_1, \dots, z_m\}$. Wir nennen ζ eine **Hüllfolge** von P, wenn ihr Polygonzug einfach ist und die Menge $P \setminus E(\zeta)$ im Innern des Polygons $\omega(\zeta)$ liegt.*
*Wir schreiben $\zeta \preceq \zeta'$, wenn ζ eine Teilfolge von ζ' ist. Dabei muß ζ nicht zusammenhängende Teilfolge sein. Man beachte auch, daß es sich um zyklische Folgen handelt. So ist z. B. $(3,5,1) \prec (1,2,3,4,5) = (3,4,5,1,2)$. Eine Teilfolge aus zwei aufeinanderfolgenden Punkten heißt **Kante** von ζ.*

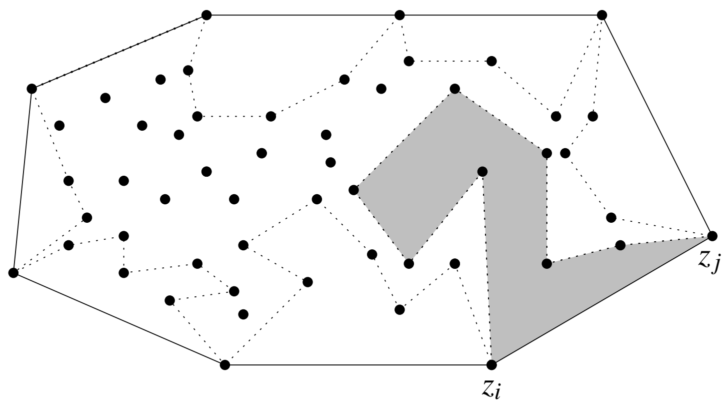


ABBILDUNG 7.1. Die Ecken der konvexen Hülle segmentieren jede beliebige Hüllfolge. Die Segmente entsprechen je einer Kante der konvexen Hülle.

Wir wählen die Reihenfolge der Punkte in einer Hüllfolge stets so, daß der Polygonzug im mathematischen Sinn durchlaufen wird. Durch diese Orientierung liegt das umschlossene Polygon links des Polygonzugs. Bei fester Wahl des Umlaufsinns gilt

LEMMA 7.5. *Sei ζ_0 die zyklische Folge, die der konvexen Hülle $\bigwedge P$ entspricht, wobei $E(\zeta_0) = E_P(\bigwedge P)$ gelte. Sei ζ eine Hüllfolge von P. Dann ist $\zeta_0 \preceq \zeta$.*

Beweis: Nach Satz 7.3 ist $E(\zeta_0) \subseteq E(\zeta)$. Es bleibt zu zeigen, daß die zyklische Reihenfolge der Punkte in ζ_0 dieselbe ist wie innerhalb von ζ. Sei $\xi = (z_i, \ldots, z_j)$ eine zusammenhängende Teilfolge von ζ mit $E(\xi) \cap E(\zeta_0) = \{z_i, z_j\}$. Der durch ξ definierte offene Polygonzug, also ohne die Kante (z_j, z_i), teilt $\bigwedge P$ in zwei Teile. Der offene Teil rechts des Polygonzugs ist in Abbildung 7.1 grau dargestellt. Er liegt vollständig außerhalb von $\omega(\zeta)$, da sonst der Rand von $\omega(\zeta)$ sich selbst überschneiden oder teilweise außerhalb der konvexen Hülle verlaufen müßte. Insbesondere kann der offene rechte Teil keine Punkte von $E(\zeta_0) \subseteq E(\zeta)$ enthalten. Daher muß (z_i, z_j) oder (z_j, z_i) eine Kante von ζ_0 sein. Aufgrund des Umlaufsinns folgt $\zeta_0 \preceq \zeta$. ∎

In diesem Kapitel wird ein rekursiver Algorithmus vorgestellt, der alle Hüllfolgen von P aufzählt. Zur Beschreibung dieses Algorithmus verwenden wir einen Baum, dessen Knoten die rekursiven Aufrufe repräsentieren. Das Argument eines rekursiven Aufrufs ist eine Hüllfolge ζ zusammen mit bestimmten Attributen. Die Folge ζ wird Knoten für Knoten erweitert um andere Folgen zu bilden. Die Attribute kontrollieren die weiteren Verzweigungen des Rekursionsbaums. Sie sorgen unter anderem dafür, daß keine Hüllfolge mehr als einmal aufgezählt wird. Das erste Attribut ist der Fixierungsindex f. Er gibt an, daß der anfängliche Teil $(z_1, \ldots, z_f)$ von ζ im aktuellen Rekursionsknoten und allen seinen Nachfahren unverändert bleibt. Sind alle Kanten inklusive (z_m, z_1) fixiert, dann setzen wir $f = m + 1$. Die übrigen Attribute sind Verbotslisten, und zwar eine für jede Kante (z_i, z_{i+1}) der aktuellen Hüllfolge. Eine Verbotsliste $VL(z_i, z_{i+1})$ enthält Punkte aus P. Diese Punkte dürfen in keinem Nachfahren des aktuellen Rekursionsknoten in der Hüllfolge zwischen z_i und z_{i+1} auftreten. Wir beschreiben einen Knoten des Rekursionsbaums durch ein Tripel (ζ, f, VL). Formal sehen wir hierbei VL als eine Funktion

$$VL : P \times P \to 2^P$$

an. Tatsächlich verwendet und speichert der Algorithmus jedoch nur die Verbotslisten der tatsächlich in ζ vorhandenen Kanten. In Korollar 7.8 werden wir sehen, daß bereits ζ und f einen Rekursionsknoten eindeutig identifizieren (bei fest gegebener Punktmenge P). Im Augenblick genügt es aber, das Tripel (ζ, f, VL) als Namen eines fest gewählten Knoten anzusehen. Den Rekursions-Teilbaum mit Wurzel (ζ, f, VL) bezeichnen wir mit $RTB(\zeta, f, VL)$.

Algorithmus 7.1 HüllRekursion(ζ, f, VL)

Eingabe: Hüllfolge ζ, Fixierungsindex f und Verbotslisten VL.
Globale Variablen: Punktmenge P.

if $f = m + 1$ **then**
  Gib ζ als Hüllfolge aus.
else
  $\widehat{VL} := VL$.
  for jedes $q \in P \setminus VL(z_f, z_{f+1})$, so daß $(z_1, \ldots, z_f, q, z_{f+1}, \ldots, z_m)$ eine Hüllfolge ist **do**
    $\widehat{\zeta} := (z_1, \ldots, z_f, q, z_{f+1}, \ldots, z_m)$.
    $\widehat{VL}(z_f, q) := \widehat{VL}(z_f, z_{f+1})$.
    $\widehat{VL}(q, z_{f+1}) := \widehat{VL}(z_f, z_{f+1})$.
    HüllRekursion($\widehat{\zeta}, f, \widehat{VL}$).
    $\widehat{VL}(z_f, z_{f+1}) := \widehat{VL}(z_f, z_{f+1}) \cup \{q\}$.
  endfor
  HüllRekursion($\zeta, f + 1, VL$).
endif

Der initiale Aufruf erfolgt mit der Folge ζ_0, die der konvexen Hülle von P entspricht, $f = 1$ und leeren Verbotslisten. Wir bezeichnen den Wurzelknoten des

Rekursionsbaums mit $(\zeta_0, 1, VL_\emptyset)$. In einem Rekursionsknoten wird versucht, die aktuelle Hüllfolge ζ zu erweitern. Dazu wird die erste noch nicht fixierte Kante (z_f, z_{f+1}) durch zwei Kanten (z_f, q) und (q, z_{f+1}) ersetzt. Für jede derart erweiterte Hüllfolge $\widehat{\zeta}$ erfolgt ein rekursiver Aufruf. Durch geeignete Veränderungen der Verbotslisten zwischen diesen Aufrufen stellt der Algorithmus sicher, daß keine Hüllfolge mehrmals aufgezählt wird. Der letzte rekursive Aufruf erfolgt mit der unveränderten Folge ζ, wobei zusätzlich die Kante (z_f, z_{f+1}) fixiert wird. Sind alle Kanten der aktuellen Hüllfolge fixiert, dann wird diese Folge ausgegeben und die Rekursion bricht ab. Algorithmus 7.1 stellt dieses Vorgehen in abstrakter Form dar. Die tatsächliche Implementierung weicht aus Effizienzgründen in mehreren Punkten von dieser Darstellung ab.

SATZ 7.6. *Algorithmus 7.1 zählt keine Hüllfolge mehrmals auf.*

Beweis: Wir betrachten einen Rekursionsknoten (ζ, f, VL) und die Unterbäume, die von ihm aus abzweigen. Ist $f = m+1$, dann endet die Rekursion in diesem Knoten. Der Teilbaum $RTB(\zeta, f, VL)$ gibt die Hüllfolge ζ genau einmal aus. Seien andernfalls $q_1, \ldots, q_\ell$ diejenigen Punkte, für die die **for**-Schleife ausgeführt wird, und zwar in dieser Reihenfolge. Es gibt dann $\ell + 1$ rekursive Aufrufe, aus denen $\ell + 1$ Unterbäume entstehen. Im j-ten Unterbaum, $1 \leq j \leq \ell$, gilt für jede Hüllfolge ζ'

$$(z_f, q_j, z_{f+1}) \preceq \zeta' \quad \text{und} \quad (z_f, q_i, z_{f+1}) \npreceq \zeta' \quad \text{für alle } i \in \{1, \ldots, j-1\}.$$

Ersteres gilt, weil Algorithmus 7.1 Hüllfolgen zwar erweitert, aber nie verkürzt. Letzteres folgt daraus, daß $q_1, \ldots, q_{j-1}$ in den Verbotslisten der Kanten (z_f, q_j) und (q_j, z_{f+1}) enthalten sind. Aus Verbotslisten wird kein Punkt gelöscht, und bei Erweiterung der Hüllfolge übernehmen die beiden neuen Kanten die Verbote der Kante, die sie ersetzen. So bleiben $q_1, \ldots, q_{j-1}$ im j-ten Teilbaum für alle Kanten zwischen z_f und z_{f+1} verboten. Im $(\ell+1)$-ten Unterbaum ist die Kante (z_j, z_{j+1}) fixiert, es können also keine Teilfolgen der Form (z_f, q_j, z_{f+1}) auftreten. Dies zeigt, daß keine Hüllfolge in mehr als einem der Unterbäume erscheint. Der Rekursionsknoten (ζ, f, VL) selbst gibt keine Hüllfolge aus. Durch Induktion über die Höhe des Teilbaums $RTB(\zeta, f, VL)$ sehen wir, daß in diesem Teilbaum jede Hüllfolge maximal einmal ausgegeben wird. Das gilt insbesondere für den gesamten Rekursionsbaum $RTB(\zeta_0, f, VL_\emptyset)$. ∎

Das folgende Lemma analysiert die Struktur des Rekursionsbaums. Daraus werden wir eine obere Schranke für die Größe des Baums herleiten. Mit Hilfe dieser Schranke können wir die Laufzeit des Algorithmus abschätzen.

LEMMA 7.7. *Eine Hüllfolge $\zeta = (z_1, \dots, z_m)$ erscheint in höchstens $m+1$ Knoten des Rekursionsbaums. Diese Knoten bilden eine ununterbrochene Kette von „letztgeborenen" Nachfahren.*

Beweis: Von allen Rekursionsknoten mit der Hüllfolge ζ habe (ζ, f, VL) minimale Pfadlänge zur Wurzel des Rekursionsbaums. Aus dem Beweis von Satz 7.6 folgt, daß ζ nicht außerhalb des Teilbaums $RTB(\zeta, f, VL)$ erscheint. Nur das „letztgeborene" Kind von (ζ, f, VL), das dem rekursiven Aufruf in Zeile 12 entspricht, übernimmt die Hüllfolge ζ. Die anderen rekursiven Aufrufe erfolgen mit einer erweiterten Folge. Die Knoten mit Folge ζ bilden daher eine ununterbrochene Kette von „letztgeborenen" Nachfahren. Bei einem Aufruf in Zeile 12 wird f erhöht, andererseits kann f nicht größer als $m+1$ werden. Daher kann ζ maximal in $m+1$ Rekursionsknoten erscheinen. ∎

Verschiedene Rekursionsknoten mit derselben Hüllfolge ζ unterscheiden sich durch ihren Fixierungsindex. Daraus folgt

KOROLLAR 7.8. *Ein Rekursionsknoten ist durch seine Hüllfolge ζ und seinen Fixierungsindex f eindeutig bestimmt.*

KOROLLAR 7.9. *Sei $|P| = n$ und h die Anzahl polygonaler Hüllen von P. Dann hat der Rekursionsbaum maximal $(n+1)h$ Knoten.*

Die Größe des Rekursionsbaums dient uns zur Abschätzung der Laufzeit des Algorithmus. Der Platzbedarf bestimmt sich unter anderem aus der Höhe des Rekursionsbaums.

SATZ 7.10. *Habe die konvexe Hülle $n_\text{b} := |E_P(\bigwedge P)|$ Ecken. Dann ist die Höhe des Rekursionsbaums nach oben durch $2n - n_\text{b} + 1$ beschränkt.*

Beweis: Wir betrachten den Pfad von der Wurzel zu einem beliebigen Knoten im Rekursionsbaum. Für jede Kante des Pfads hat der Algorithmus entweder eine Kante fixiert oder die Hüllfolge um eine Ecke erweitert. Die Wurzelfolge ζ_0, die der konvexen Hülle entspricht, kann nur um $n - n_\text{b}$ Ecken erweitert werden. Der Fixierungsindex kann höchstens von 1 auf $n+1$ erhöht werden. Daher kann der Pfad von der Wurzel maximal $2n - n_\text{b}$ Kanten lang sein. ∎

Eine rekursive Implementierung des Algorithmus hat höchstens $2n - n_\text{b} + 1$ Rekursionsknoten gleichzeitig auf dem Laufzeitstack. Das bedeutet insbesondere, daß keine Endlosrekursion möglich ist.

SATZ 7.11. *Algorithmus HüllRekursion zählt alle Hüllfolgen von P auf.*

Beweis: Sei $\zeta = (z_1, \ldots, z_m)$ eine Hüllfolge und (ζ, f, VL) ein Knoten im Rekursionsbaum. Wir bezeichnen eine Hüllfolge ζ' als abgeleitete Folge des Knotens, wenn gilt

1. $\zeta \preceq \zeta'$,
2. der fixierte Teil $(z_1, \ldots, z_f)$ von ζ ist eine zusammenhängende Teilfolge von ζ' und
3. ζ' verstößt gegen keine der Verbotslisten, d. h. für jede Kante (z_i, z_{i+1}) von ζ und alle $q \in VL(z_i, z_{i+1})$ gilt $(z_i, q, z_{i+1}) \not\preceq \zeta'$.

Behauptung: der Rekursions-Teilbaum $RTB(\zeta, f, VL)$ zählt alle abgeleiteten Folgen von (ζ, f, VL) auf. Wir beweisen die Behauptung durch Induktion über die Höhe des Teilbaums. Die Höhe bezeichnen wir mit $TBH(\zeta, f, VL)$. Nach Satz 7.10 wissen wir, daß die Höhe immer endlich ist. Wenn der Teilbaum Höhe 1 hat, besteht er aus einem einzigen Blattknoten. Das Blatt muß von der Form $(\zeta, m+1, VL)$ sein. Die einzige abgeleitete Folge ist ζ selbst, und sie wird von dem Blattknoten ausgegeben.

Sei ζ' eine abgeleitete Folge von (ζ, f, VL) mit $f \leq m$. Dann enthält ζ' eine zusammenhängende Teilfolge der Form $\xi = (z_f, \ldots, z_{f+1})$. Ist $\xi = (z_f, z_{f+1})$, dann ist ζ' auch abgeleitete Folge des Knotens $(\zeta, f+1, VL)$. Dieser ist ein Kind von (ζ, f, VL), also gilt $TBH(\zeta, f+1, VL) < TBH(\zeta, f, VL)$. Nach Induktionsvoraussetzung wird ζ' im Unterbaum $RTB(\zeta, f+1, VL)$ von $RTB(\zeta, f, VL)$ aufgezählt.

Ist $\xi \neq (z_f, z_{f+1})$, dann betrachten wir den durch ξ definierten geschlossenen Polygonzug, siehe Abbildung 7.2. Bis auf die Kante $z_{f+1} \wedge z_f$ liegt dieser Polygonzug im Innern der polygonalen Hülle ω, die durch ζ beschrieben wird. Er umschließt ein einfaches Polygon, das sich triangulieren läßt. Eines der Triangulierungsdreiecke, etwa δ, hat als Ecken z_f, z_{f+1} und einen weiteren Punkt q. Sei ω' die durch ζ' beschriebene polygonale Hülle. Das Dreieck δ liegt außerhalb von ω' (bis auf gemeinsame Randteile), es kann also keine Punkte von $P \setminus \{z_f, z_{f+1}, q\}$ enthalten. Außer $z_f \wedge z_{f+1}$ schneidet δ den Rand von ω nicht. Daher ist $(z_1, \ldots, z_f, q, z_{f+1}, \ldots, z_m)$ eine Hüllfolge. Da ζ' von (ζ, f, VL) abgeleitet ist, ist $q \notin VL(z_f, z_{f+1})$. Wir sehen, daß q einer der Punkte $q_1, \ldots, q_\ell$ ist, für die die **for**-Schleife im Knoten (ζ, f, VL) ausgeführt wird. Insbesondere wird die **for**-Schleife überhaupt ausgeführt. Sei q_j der erste Punkt, für den die **for**-Schleife ausgeführt wird und der in der Teilfolge ξ enthalten ist. Das j-te Kind

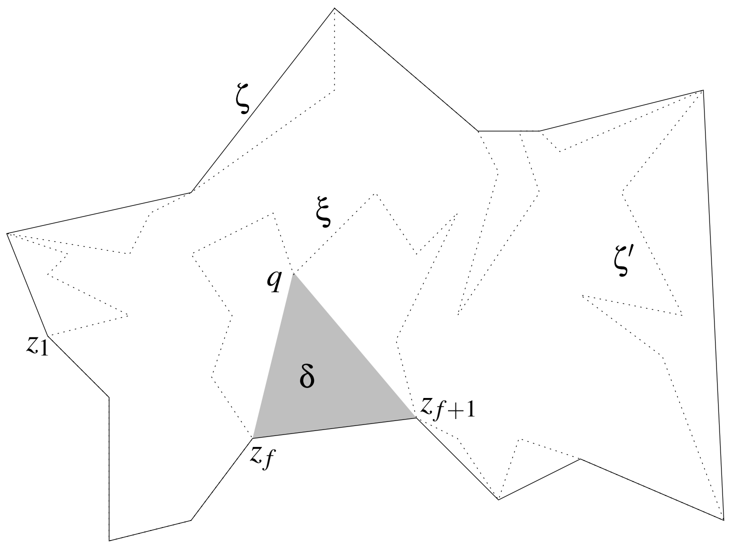


ABBILDUNG 7.2. Induktionsschritt im Beweis von Satz 7.11. Die Hüllfolgen ζ (durchgezogen) und ζ' (gepunktet) stimmen von z_1 bis z_f überein.

von (ζ, f, VL) ist $(\widehat{\zeta}, f, \widehat{VL})$ mit

$$\begin{aligned}
&\widehat{\zeta} = (z_1, \ldots, z_f, q_j, z_{f+1}, \ldots, z_m) \quad , \\
&\widehat{VL}(z_i, z_{i+1}) = VL(z_i, z_{i+1}) \quad \text{für alle } i \in \{1, \ldots, m\} \text{ und} \\
&\widehat{VL}(z_f, q_j) = \widehat{VL}(q_j, z_{f+1}) = VL(z_f, z_{f+1}) \cup \{q_1, \ldots, q_{j-1}\} \quad .
\end{aligned}$$

Da keiner der Punkte $q_1, \ldots, q_{j-1}$ in der Teilfolge ξ vorkommt, verstößt ζ' nicht gegen die Verbotslisten $\widehat{VL}$. Somit ist ζ' eine aus $(\widehat{\zeta}, f, \widehat{VL})$ abgeleitete Folge und wird nach Induktionsvoraussetzung im Unterbaum $RTB(\widehat{\zeta}, f, \widehat{VL})$ aufgezählt. Damit ist die Behauptung bewiesen.

Um den Satz zu beweisen, betrachten wir den Wurzelknoten $(\zeta_0, 1, VL_\emptyset)$ des Rekursionsbaums. Nach Lemma 7.5 ist jede Hüllfolge von diesem Knoten abgeleitet. ∎

BEMERKUNG 7.12. *Für eine Hüllfolge $\zeta \neq \zeta_0$ von P kann man auch HüllRekursion($\zeta, 1, VL_\emptyset$) aufrufen. Es werden dann alle Hüllfolgen ζ' aufgezählt, für die $\zeta \preceq \zeta'$ gilt. Die zugehörigen polygonalen Hüllen sind Teilmengen der durch ζ beschriebenen polygonalen Hülle.*

Algorithmus 7.1 ist eine sehr abstrakte Darstellung von HüllRekursion. Bevor wir seine Komplexität untersuchen können, müssen wir noch einige Details konkretisieren. Wir beginnen mit den Datenstrukturen. Die Punktmenge P speichern wir in einem Array $P[1..n]$ in Form ihrer Koordinaten. Im Programm wird jeder Punkt $p_i = P[i]$ durch seinen Index i repräsentiert. Die aktuelle Hüllfolge ζ wird als verkettete Liste HL der Punktindizes gespeichert. Als Fixierungsindex dient ein Pointer in die verkettete Liste, der Fall $f = m+1$ wird durch einen Nullpointer dargestellt. Die bisherige Formulierung von VL suggeriert, daß zu jeder Zeit für alle potentiellen Kanten Verbotslisten definiert sind. Der Algorithmus verwendet jedoch nur solche Verbotslisten, deren Kanten in der aktuellen Hüllfolge enthalten sind. Wird eine neue Kante eingefügt, dann wird ihre Verbotsliste explizit gesetzt (Zeilen 7 und 8 von Algorithmus 7.1). Daher ist es nicht nötig, Verbotslisten für „unbenutzte" Kanten zu speichern. Eine Verbotsliste wird durch ein Array $VA_j[1..n]$ von Bits dargestellt. Das Bit $VA_j[i]$ zeigt an, ob p_i in $VL(z_j, z_{i+1})$ enthalten ist. VA_j wird in der verketteten Liste HL gespeichert, und zwar im selben Listenelement wie der Anfangspunkt z_j der zugehörigen Kante.

Zeile 4 von Algorithmus 7.1 kopiert alle Verbotslisten. Im Rekursionsknoten werden jedoch nur zwei Arrays verändert, nämlich VA_f und das Array VA_q des neu eingefügten Listenelements. Vor Eintritt in die **for**-Schleife wird nur VA_f in eine lokale Variable VA_0 kopiert. Beim ersten Durchlaufen der **for**-Schleife wird zwischen z_f und z_{f+1} ein neues Listenelement in HL eingefügt. VA_f stellt jetzt bereits die Verbotsliste $\widehat{VL}(z_f, q)$ dar, Zeile 7 wird daher überhaupt nicht ausgeführt. In das Array VA_q des neuen Listenelements wird der Inhalt von VA_f kopiert, was Zeile 8 entspricht. Bei weiteren Schleifendurchläufen wird kein neues Listenelement eingefügt, sondern lediglich der Punktindex von q überschrieben. Die Änderungen der Verbotsliste in Zeile 10 werden direkt in VA_f und VA_q vorgenommen. Dadurch werden gleichzeitig Zeilen 7 und 8 des nächsten Schleifendurchlaufs ausgeführt. Nach dem letzten Schleifendurchlauf wird das Listenelement von q wieder aus HL entfernt. Mit Hilfe der Kopie VA_0 wird VA_f wieder in den vorherigen Zustand versetzt. Diese Wiederherstellung ist nicht so sehr für den rekursiven Aufruf in Zeile 12 nötig, denn der ließe sich auch vor Beginn der **for**-Schleife ausführen. Es muß aber sichergestellt sein, daß der Elternknoten im Rekursionsbaum die Verbotslisten unverändert vorfindet, wenn der aktuelle Rekursionsknoten beendet ist.

Die **for**-Schleife wird für alle Punkte q ausgeführt, für die die erweiterte Folge $\widehat{\zeta} := (z_1, \ldots, z_f, q, z_{f+1}, \ldots, z_m)$ eine Hüllfolge ist. Betrachten wir das Dreieck $\delta := q \wedge z_f \wedge z_{f+1}$ (vergleiche Abbildung 7.2). Läge ein weiterer Punkt $p \in P$ in δ, dann würde p entweder außerhalb der neuen Hülle oder als degenerierte Ecke auf ihrem Rand liegen. Nun sind degenerierte Ecken zwar erlaubt, sie müssen aber in der Hüllfolge aufgeführt sein, und das wäre für p nicht der Fall. Ferner darf der Schnitt von δ mit dem Rand $\partial\omega$ der aktuellen Hülle nur aus der Kante $z_f \wedge z_{f+1}$ bestehen. Ansonsten wäre der durch $\widehat{\zeta}$ definierte geschlossene Polygonzug selbstüberschneidend. Algorithmus 7.2 zählt diese Punkte q auf. Er benutzt zwei zyklische Listen $PWL(z_f)$ und $PWL(z_{f+1})$. In $PWL(p_i)$ sind alle Punkte von $P \setminus \{p_i\}$ nach ihrem Polarwinkel um p_i herum sortiert. Die Sortierung ist absteigend, so daß bei einem Durchlauf der Liste z_f im Uhrzeigersinn umlaufen wird. Mit $PWL(p_i)[j]$ wird der j-te Punkt in der polarwinkelsortierten Liste von p_i bezeichnet. Die Listen $PWL(p_1), \ldots, PWL(p_n)$ berechnen wir in einem Vorverarbeitungsschritt. Sie benötigen $O(n^2)$ Speicherplatz und können in optimaler Zeit $O(n^2)$ berechnet werden. Asano et al. [**AAGH$^+$86**] und Welzl [**Wel85**] verwenden hierzu Dualisierungstechniken nach Chazelle, Guibas und Lee [**CGL83, CGL85**] bzw. nach Edelsbrunner, O'Rourke und Seidel [**EOS83, EOS86**]. Overmars und Welzl [**OW88**] geben ein Verfahren ohne Dualisierung an. Algorithmus 7.2 durchläuft die Listen $PWL(z_f)$ und $PWL(z_{f+1})$ in synchronisierter Weise. Wenn in beiden Listen an der aktuellen Position derselbe Punkt q steht, enthält das Dreieck $\delta = z_f \wedge z_{f+1} \wedge q$ keinen weiteren Punkt aus P.

Um sicherzustellen, daß $\delta \cap \partial\omega = z_f \wedge z_{f+1}$ ist, werden nur solche Punkte in $PWL(z_f)$ betrachtet, die in ω von z_f aus strikt sichtbar sind und nicht auf dem Rand von ω liegen. Um diese Punkte zu identifizieren, verwenden wir das Sichtbarkeitspolygon von z_f in ω. Zusammen mit dem Umlauf um z_f verfolgt der Algorithmus einen umlaufenden Strahl s von z_f durch den Punkt $PWL(z_f)[i]$, und bestimmt den ersten Schnittpunkt des offenen Strahls $s \setminus \{z_f\}$ mit dem Rand des Sichtbarkeitspolygons. Da es sich um das Sichtbarkeitspolygon von z_f handelt, ist der Schnitt seines Randes mit dem offenen Strahl entweder leer, ein einzelner Punkt, oder eine Strecke. Liegt der aktuelle Listenpunkt $PWL(z_f)[i]$ hinter dem ersten Schnittpunkt, dann ist er von z_f aus unsichtbar und wird übersprungen. Ist $PWL(z_f)[i]$ selbst der erste Schnittpunkt, dann liegt er auf dem Rand von ω und wird ebenfalls übersprungen. Auf diese Weise werden alle Punkte auf dem Rand von ω übersprungen. Sie sind entweder unsichtbar oder (möglicherweise degenerierte) Ecken des Sichtbarkeitspolygons. Das Sichtbarkeitspolygon kann in Zeit $O(n)$ berechnet werden, siehe Lee [**Lee83b**] mit Korrekturen[1] von Joe und Simpson [**JS87**].

Da das Sichtbarkeitspolygon sternförmig ist und z_f in seinem Kern liegt, kann der erste Schnittpunkt des umlaufenden Strahls in $O(n)$ Gesamtzeit verfolgt werden.

[1]Ein früherer Linearzeit-Algorithmus von El Gindy und Avis [**EGA81**] ist laut [**JS87**] ebenfalls fehlerhaft, aber im Gegensatz zu [**Lee83b**] nicht ohne weiteres korrigierbar.

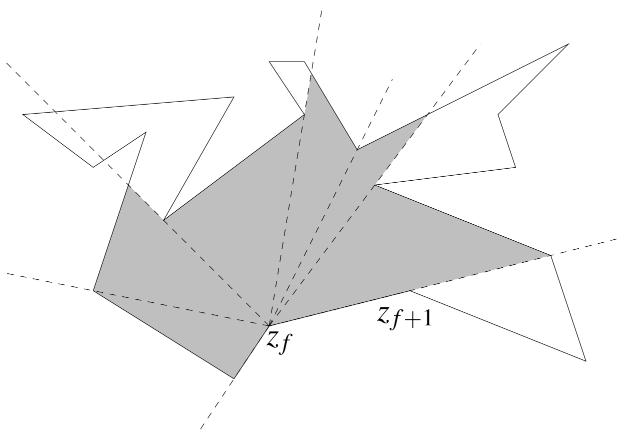


ABBILDUNG 7.3. Sichtbarkeitspolygon (grau) von z_f in ω. Die Positionen des umlaufenden Strahls, an denen die erste Schnittkante wechselt, sind gestrichelt eingezeichnet.

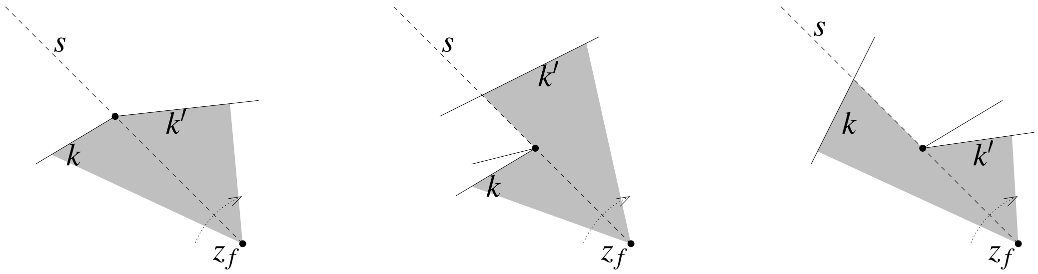


ABBILDUNG 7.4. Wechsel der ersten Schnittkante des Sichtbarkeitspolygons mit dem umlaufenden Strahl. Die eingezeichneten Kanten gehören zur polygonalen Hülle ω, ein Teil des Sichtbarkeitspolygons ist grau schattiert.

Der Algorithmus merkt sich jeweils die Kante k des Sichtbarkeitspolygons, die den Strahl s schneidet. In Abbildung 7.3 kann man sehen, daß sich hierbei keine zusätzlichen Ereignisse im Umlauf um z_f ergeben: Ein Wechsel der Schnittkante findet nur dann statt, wenn s eine Ecke des Polygons ω überstreicht, und diese Ecke ist in $PWL(z_f)$ enthalten. Abbildung 7.4 zeigt die drei möglichen Fälle, die beim Wechsel der Schnittkante auftreten können. Die Positionen des umlaufenden Strahls, in denen er mehr als eine Kante des Sichtbarkeitspolygons schneidet, sind genau die Wechselpositionen.

Für z_{f+1} und q brauchen wir keinen Sichtbarkeitstest durchzuführen. Wenn eine Kante von ω die Strecke $z_{f+1} \wedge q$ schneidet, dann schneidet sie entweder auch $z_f \wedge q$, oder sie hat einen Endpunkt im Dreieck $z_f \wedge z_{f+1} \wedge q$. Im ersten Fall ist

q auch von z_f aus unsichtbar, der zweite Fall wird durch den synchronisierten Umlauf um z_f und z_{f+1} behandelt.

Den synchronisierten Umlauf inklusive Sichtbarkeitstest führt Algorithmus 7.2 durch. Er berechnet eine Liste *EL* mit den Punkten, um die die aktuelle Hüllfolge erweitert werden kann. Der Algorithmus ist hier nur für den „allgemeinen Fall" formuliert, daß keine drei Punkte von *P* kollinear sind. Auf die Behandlung kollinearer Punkte geht Bemerkung 7.14 ein. Zunächst wollen wir einige Schritte des Algorithmus näher betrachten.

Zeilen 4, 6 und 28: Punkte, die von z_f und von z_{f+1} aus strikt sichtbar sind, liegen in der inneren Halbebene bezüglich der Gerade $z_f \vee z_{f+1}$. In Abbildung 7.3 liegt die innere Halbebene oberhalb der Gerade, und z_f ist konkave Ecke von ω. Hier überstreicht der Strahl s bei seinem Unlauf um z_f die gesamte innere Halbebene. Zu Beginn des Umlaufs bildet er mit der Strecke $z_f \wedge z_{f+1}$ einen Winkel von 180°, am Ende 0°. Diejenigen Punkte von *P*, die in der inneren Halbebene liegen, bilden einen (zyklisch) zusammenhängenden Teil von $PWL(z_f)$. Dieser Teil ist nicht leer, er kann aber die gesamte Liste umfassen, wenn $z_f \wedge z_{f+1}$ eine Kante der konvexen Hülle ist. Der „erste" Punkt in der inneren Halbebene ist derjenige, für den der Strahl s einen möglichst großen Winkel mit der Strecke $z_f \wedge z_{f+1}$ bildet (Zeile 6). Ist hingegen z_f eine konvexe Ecke, dann verdeckt die Kante $z_{f-1} \wedge z_f$ einen Teil der inneren Halbebene. Der Umlauf um z_f beginnt dann bei dem Punkt, der in $PWL(z_f)$ auf z_{f-1} folgt (Zeile 4). Der Umlauf ist beendet, wenn er z_{f+1} erreicht hat (Zeile 28).

Zeilen 11, 18 und 22: „Rechts von s" ist auf die Orientierung von s selbst bezogen, also letztendlich auf Polarwinkel, nicht auf x-Koordinaten.

Zeilen 11 und 18: In den Positionen, wo ein Wechsel der Schnittkante stattfindet, schneidet der umlaufende Strahl zwei oder drei Kanten des Sichtbarkeitspolygons, vergleiche Abbildung 7.4. Nur eine dieser Kanten, k' in Abbildung 7.4, wird im weiteren Umlauf noch geschnitten. Ihr Inneres liegt rechts von s. Der Algorithmus findet diese Kante, indem er auf dem Rand des Sichtbarkeitspolygons im Uhrzeigersinn weitergeht.

Zeilen 13–20: In dieser **while**-Schleife findet der Umlauf um z_f in einzelnen Etappen statt. Aufgrund der Schleifenbedingung werden solche Punkte übersprungen, die verboten, unsichtbar oder bereits in ζ enthalten sind. Das Ende des gesamten Umlaufs, $q = z_{f+1}$, muß explizit abgefragt werden.

Zeilen 14, 23 und 26: Die Indexerhöhungen sind modulo der Listenlänge $n-1$ zu verstehen.

Algorithmus 7.2 ErweiterungsPunkte(EL)

Ausgabe: Liste EL der Erweiterungspunkte.

Globale Variablen: Punktmenge P, Hüllfolge ζ, Fixierungsindex f, Verbotsliste VA_f, polarwinkelsortierte Listen $PWL(z_f)$ und $PWL(z_{f+1})$.

Berechne das Sichtbarkeitspolygon von z_f innerhalb der aktuellen Hülle $\omega(\zeta)$.
$EL := \emptyset$.
if z_f ist konvexe Ecke von ω **then**
    Bestimme in $PWL(z_f)$ den Index i so, daß $PWL(z_f)[i-1] = z_{f-1}$ ist.
else
    Bestimme in $PWL(z_f)$ den zyklisch ersten Index i so, daß $PWL(z_f)[i]$ in der inneren Halbebene bezüglich $z_f \vee z_{f+1}$ liegt.
endif
$q := PWL(z_f)[i]$.
s := Strahl von z_f durch q.
Bestimme j so, daß $PWL(z_{f+1})[j] = q$.
k := Schnittkante von $s \setminus \{z_f\}$ mit dem Rand des Sichtbarkeitspolygons, wobei int k strikt rechts von s liegt.
repeat
    while $q \neq z_{f+1}$ **and** (q ist in VA_f als verboten markiert **or** $k \cap (z_f \wedge q) \neq \emptyset$ **or** q ist Ecke des Sichtbarkeitspolygons) **do**
        $i := i+1$.
        $q := PWL(z_f)[i]$.
        s := Strahl von z_f durch q.
        if q ist Ecke des Sichtbarkeitspolygons **then**
            k := Schnittkante von $s \setminus \{z_f\}$ mit dem Rand des Sichtbarkeitspolygons, wobei int k strikt rechts von s liegt.
        endif
    endwhile
    if $q \neq z_{f+1}$ **then**
        while $PWL(z_{f+1})[j+1]$ liegt nicht strikt rechts von s **do**
            $j := j+1$.
        endwhile
        if $PWL(z_{f+1})[j] = q$ **then** $EL := EL \cup \{q\}$ **endif**
        $i := i+1$.
    endif
until $q = z_{f+1}$.

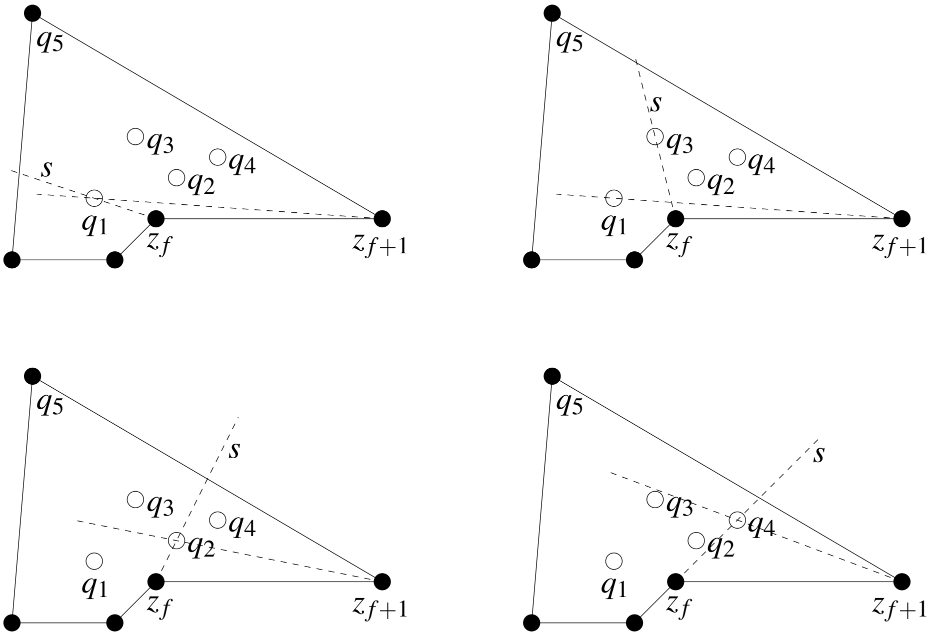


ABBILDUNG 7.5. Phasen des synchronisierten Umlaufs in Algorithmus 7.2. Die Reihenfolge $q_1, \ldots, q_5$ entspricht der polarwinkelsortierten Liste $PWL(z_{f+1})$.
Oben links: Der Umlauf beginnt mit q_1. Die beiden **while**-Schleifen werden nicht ausgeführt, so daß q_1 in *EL* aufgenommen wird. In Zeile 26 geht der Umlauf um z_f weiter zu q_5.
Oben rechts: Da q_5 zur aktuellen Hüllfolge gehört, ist s in der ersten **while**-Schleife weiter zu q_3 gelaufen. Der Umlauf um z_{f+1} geht nicht weiter, da q_2 nach wie vor rechts von s liegt. *EL* wird nicht verändert und s läuft in Zeile 26 weiter zu q_2.
Unten links: Der Umlauf um z_{f+1} erreicht in der zweiten **while**-Schleife q_2. Dieser Punkt wird in *EL* aufgenommen, und s läuft weiter zu q_4.
Unten rechts: Der Umlauf um z_{f+1} hat q_3 übersprungen und q_4 erreicht. Nach Aufnahme von q_4 in *EL* läuft s zu z_{f+1}, der synchronisierte Umlauf endet.

Abbildung 7.5 zeigt, wie der synchronisierte Umlauf von weiteren Punkten freie Dreiecke findet.

LEMMA 7.13. *Wenn keine drei Punkte von P kollinear sind, dann findet Algorithmus 7.2 genau die Punkte $q \in P$, für die $(z_1, \ldots, z_f, q, z_{f+1}, \ldots, z_m)$ eine Hüllfolge ist.*

Beweis: Sei $j_{\min}$ der Wert, den Index j in Zeile 10 erhält. Zuerst bemerken wir, daß $PWL(z_{f+1})[j]$ zu keinem Zeitpunkt rechts des Strahls s liegt: Der Strahl selbst bewegt sich nur im Uhrzeigersinn, und die Schleifenbedingung in Zeile 22 stellt sicher, daß $PWL(z_{f+1})[j]$ den Strahl nicht „überholt". Dasselbe gilt für die Punkte $PWL(z_{f+1})[j_{\min}], \ldots, PWL(z_{f+1})[j-1]$. Wenn also der Algorithmus einen Punkt $q = PWL(z_f)[i] = PWL(z_{f+1})[j]$ in EL aufnimmt, dann liegen $PWL(z_{f+1})[j_{\min}], \ldots, PWL(z_{f+1})[j-1]$ strikt links von s. Die Punkte, die in $PWL(z_{f+1})$ auf $q = PWL(z_{f+1})[j]$ folgen, liegen strikt rechts des Strahls von z_{f+1} durch q. Da die beiden Strahlen das Dreieck $\delta = z_f \wedge z_{f+1} \wedge q$ begrenzen, kann kein weiterer Punkt von P in δ liegen. Wir wissen weiterhin, daß q von z_f aus strikt sichtbar ist. Würde nun q gegenüber z_{f+1} durch eine Kante von ω verdeckt, dann müßte diese Kante entweder einen Endpunkt in $\delta \setminus (z_f \wedge z_{f+1})$ haben oder die Strecke $z_f \wedge q$ schneiden. (Dies gilt auch, wenn es sich bei der verdeckenden Kante um $z_{f+1} \wedge z_{f+2}$ handelt.) Beides ist nach dem bisher gezeigten unmöglich. Daher ist $(z_1, \ldots, z_f, q, z_{f+1}, \ldots, z_m)$ eine Hüllfolge.

Sei umgekehrt $(z_1, \ldots, z_f, q^*, z_{f+1}, \ldots, z_m)$ eine Hüllfolge. Dann liegt q^* im Inneren des Sichtbarkeitspolygons und ist in VA_f nicht als verboten markiert. Insbesondere liegt q^* strikt in der inneren Halbebene bezüglich $z_f \wedge z_{f+1}$ und rechts des Strahls von z_f durch z_{f-1}. Daher wird q^* in der ersten **while**-Schleife (Zeilen 13–20) aufgezählt, und die Schleife endet mit $q = q^*$. Auch zu diesem Zeitpunkt liegen die Punkte $PWL(z_{f+1})[j_{\min}], \ldots, PWL(z_{f+1})[j-1]$ strikt links von s. Daher kann q^* nicht unter diesen Punkten sein. Gäbe es in $PWL(z_{f+1})$ vor q^* einen Punkt, der rechts des Strahls s liegt, dann läge dieser Punkt im Dreieck δ, und $(z_1, \ldots, z_f, q^*, z_{f+1}, \ldots, z_m)$ wäre keine Hüllfolge. Die zweite **while**-Schleife (Zeilen 22–24) läuft jetzt so lange, bis $PWL(z_{f+1})[j] = q^*$ ist. In Zeile 25 wird schließlich q^* zur Liste EL hinzugefügt. ■

BEMERKUNG 7.14. (Algorithmus ErweiterungsPunkte mit kollinearen Punkten) Gibt es in P drei oder mehr kollineare Punkte, dann können beim Umlauf um z_f oder um z_{f+1} mehrere Punkte gleichzeitig getroffen werden. Durch eine geringfügige Störung von z_f und z_{f+1} können diese Ereignisse zeitlich entzerrt werden. Wie wir sehen werden, kann man den Effekt der Störung erreichen, ohne die Punkte tatsächlich zu verschieben. Die Störung besteht darin, daß wir die Kante $z_f \wedge z_{f+1}$ nach beiden Seiten verlängern. Abbildung 7.6 zeigt, wie hierdurch die Kollinearität von z_f oder z_{f+1} mit weiteren Punkten aufgehoben wird. Die Reihenfolge, in der kollineare Punkte nach der Entzerrung durchlaufen werden, entspricht ihrem Abstand von z_f bzw. z_{f+1}. Punkte, die mit z_f kollinear liegen, erscheinen im Umlauf um z_f nach fallendem Abstand sortiert. Mit z_{f+1} kollineare Punkte werden durch die Entzerrung nach steigendem Abstand sortiert. Die

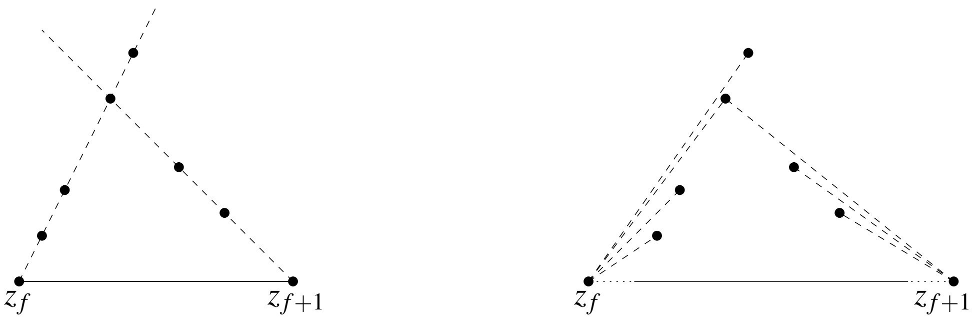


ABBILDUNG 7.6. Kollineare Punkte können beim Umlauf als gleichzeitige Ereignisse auftreten. Durch eine geringfügige Verlängerung der Kante $z_f \wedge z_{f+1}$ werden diese Ereignisse entzerrt.

Reihenfolge aller übrigen Punkte ändert sich nicht, sofern die Verlängerung der Kante $z_f \wedge z_{f+1}$ klein genug gewählt wird. Statt nun die Verlängerung tatsächlich durchzuführen, stellen wir lediglich sicher, daß kollineare Punkte in der entsprechenden Reihenfolge durchlaufen werden. Die hierzu nötigen Änderungen werden am Beispiel der polarwinkelsortierten Liste $PWL(p_1)$ erklärt. Liegen mehrere Punkte kollinear mit p_1, dann fügen wir diese Punkte nach fallendem Abstand zu p_1 sortiert in $PWL(p_1)$ ein. Seien z. B. $PWL(p_1)[r], \ldots, PWL(p_1)[s]$ derart sortierte kollineare Punkte. Ist jetzt in Algorithmus 7.2 $z_f = p_1$, dann liegt $PWL(p_1)$ in entzerrter Reihenfolge vor und wird normal durchlaufen. Ist in einem anderen Aufruf des Algorithmus $z_{f+1} = p_1$, dann müssen $PWL(p_1)[r], \ldots, PWL(p_1)[s]$ beim Umlauf genau umgekehrt durchlaufen werden. Wir erweitern die polarwinkelsortierten Listen um einen Kollinearitätszähler für jeden Punkt. Der Kollinearitätszähler von $PWL(z_{f+1})[i]$ gibt an, mit wie vielen weiteren Punkten z_{f+1} und $PWL(z_{f+1})[i]$ kollinear sind. Für „normale" Punkte ist er also 0, für $PWL(p_1)[r]$ wäre er $s - r$. Trifft der Umlauf um z_{f+1} auf den Listeneintrag $PWL(p_1)[r]$ mit Kollinearitätszähler $\neq 0$, dann springt er in der Liste zu $PWL(p_1)[s]$ und läuft anschließend bis zu $PWL(p_1)[r]$ zurück. Danach setzt er den Listendurchlauf mit $PWL(p_1)[s+1]$ fort.

Mit z_f kollineare Punkte können zu Problemen führen, wenn mehrere von ihnen Ecken des Sichtbarkeitspolygons sind. Es kann in diesem Fall vorkommen, daß der Algorithmus in Zeile 18 auf eine Ecke trifft, die gar nicht mit der aktuellen Schnittkante inzident ist. Um dies zu verhindern, betrachten wir jeweils nur den Punkt mit dem kleinsten Abstand zu z_f als Ecke des Sichtbarkeitspolygons. In Abbildung 7.7 ist dieser Punkt ausgefüllt gezeichnet. Dies tun wir selbst dann,

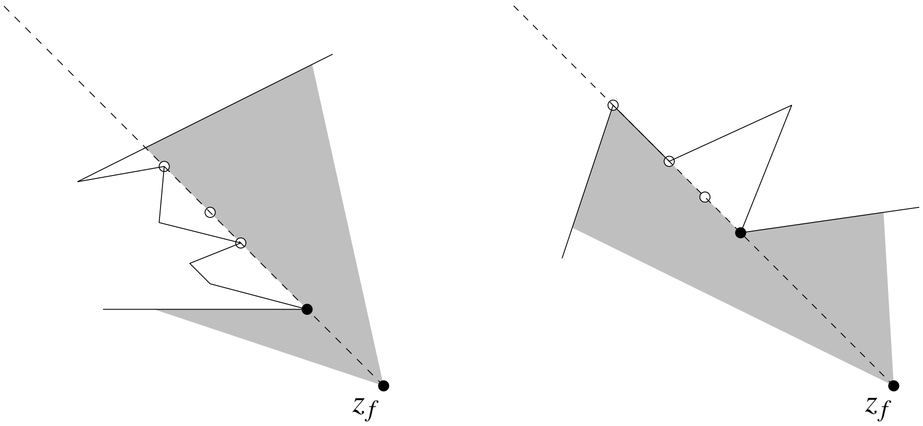


ABBILDUNG 7.7. Sind mehrere Punkte von P auf dem Rand des Sichtbarkeitspolygons kollinear mit z_f, dann zählt nur der nächste von ihnen als Ecke des Sichtbarkeitspolygons.

wenn einer der übrigen Punkte auf einer nicht degenerierten Ecke des Sichtbarkeitspolygons liegt, wie in Abbildung 7.7 rechts.

Schließlich müssen wir noch die Bedingung der **while**-Schleife in Zeile 22 an die hypothetische Verlängerung anpassen. Die Anpassung betrifft Punkte, die auf dem Strahl s liegen. Wie man in Abbildung 7.6 leicht sehen kann, kommen Punkte zwischen z_f und q durch die Verlängerung strikt rechts von s zu liegen. Punkte hingegen, die auf s hinter q liegen, sind nach der Verlängerung strikt links von s. Die **while**-Schleife soll abbrechen, wenn $PWL(z_{f+1})[j]$ nach der Velängerung strikt rechts von s läge. Wir müssen die Bedingung daher erweitern zu

22: **while** $PWL(z_{f+1})[j]$ liegt nicht strikt rechts von s
and $PWL(z_{f+1})[j] \notin \operatorname{int}(z_f \wedge q)$ **do**

■

SATZ 7.15. *Werden kollineare Punkt gemäß Bemerkung 7.14 behandelt, dann findet Algorithmus 7.2 genau die Punkte $q \in P$, für die $(z_1, \ldots, z_f, q, z_{f+1}, \ldots, z_m)$ eine Hüllfolge ist.*

Beweis: Die Behandlung kollinearer Punkte gemäß Bemerkung 7.14 entspricht einer genügend kleinen Verlängerung der Kante $z_f \wedge z_{f+1}$. Insbesondere ist die hypothetische Verlängerung so klein, daß kein zusätzlicher Punkt $p \in P$ in einem Dreieck $\delta = z_f \wedge z_{f+1} \wedge q$ zu liegen kommt. Andernfalls würde sich die Reihenfolge von p und q in einer der beiden polarwinkelsortierten Listen ändern. Durch die Verlängerung seiner Basis vergrößert sich das Dreieck δ, daher kann auch kein

Punkt „hinauswandern". Die einzig mögliche Veränderung in Bezug auf δ ist, daß ein Punkt vom Rand ins Innere „wandert". Dies ändert nichts an der Unzulässigkeit von q als Erweiterungspunkt.

Die Verlängerung ist weiterhin so klein, daß z_f keine Kante der aktuellen Hülle berührt oder überschreitet. Aus diesem Grund, und weil die Endpunkte von Hüllkanten ebenfalls in den polarwinkelsortierten Listen enthalten sind, können von z_f aus keine Punkte unsichtbar werden. Strikt sichtbar werden können nur solche Punkte wie die leer gezeichneten in Abbildung 7.7 rechts, die auf einer mit z_f kollinearen Kante des Sichtbarkeitspolygons liegen. (Die entsprechenden Punkte links in der Abbildung bleiben unsichtbar.) Diese Punkte kommen als Erweiterungspunkte nicht in Frage, da die ausgefüllt gezeichnete Ecke des Sichtbarkeitspolygons im Dreieck δ liegt. Der Algorithmus erkennt dies aufgrund der Änderung von Zeile 22. Die Menge der zulässigen Erweiterungspunkte ändert sich also nicht, und der modifizierte Algorithmus bestimmt sie korrekt. ∎

Algorithmus 7.3 zeigt HüllRekursion in konkretisierter Fassung. Zu den bereits besprochenen Änderungen gegenüber Algorithmus 7.1 kommt noch eine weitere hinzu. Ist der Fixierungsindex $f = m$, dann erfolgt in Algorithmus 7.1 ein rekursiver Aufruf mit $f = m+1$. Die einzige Aufgabe dieses Aufrufs ist, die Hüllfolge auszugeben. Algorithmus 7.3 gibt in Zeile 16 statt des rekursiven Aufrufs direkt die Hüllfolge aus. Die Initialisierungsschritte sowie der erste rekursive Aufruf erfolgen in Algorithmus 7.4.

Im folgenden analysieren wir die Komplexität von HüllAufzählung.

LEMMA 7.16. *Ein einzelner Rekursionsknoten von Algorithmus 7.3 benötigt $O(n)$ Zeit und Platz im schlechtesten Fall.*

Beweis: In einem Rekursionsknoten werden maximal drei Verbotsarrays kopiert, der Zeit– und Platzbedarf hierfür ist $O(n)$. Einfügen in und Entfernen aus der verketteten Liste *HL* erfolgt in konstanter Zeit (da f auf das Listenelement vor dem zu löschenden Element zeigt). Die **for**-Schleife wird $O(n)$ mal in je konstanter Zeit durchlaufen. Der Aufruf von ErweiterungsPunkte und die Ausgabe der Hüllfolge benötigen lineare Zeit und Platz. ∎

SATZ 7.17. *Sei h die Anzahl polygonaler Hüllen von P. HüllAufzählung benötigt im schlechtesten Fall $O(hn^2)$ Zeit und $O(n^2)$ Platz.*

Beweis: Die Schranken folgen aus Lemma 7.16 in Verbindung mit Korollar 7.9 und Satz 7.10. Der Rekursionsbaum hat $O(hn)$ Knoten, und jeder Knoten benötigt $O(n)$ Zeit. Die Initialisierung in HüllAufzählung benötigt $O(n^2)$ Zeit und Platz

Algorithmus 7.3 HüllRekursion(f)

Eingabe: Pointer f (Fixierungsindex).

Globale Variablen: verkettete Liste HL, Punktarray P, polarwinkelsortierte Listen $PWL(p_1), \ldots, PWL(p_n)$.

ErweiterungsPunkte(EL).
if $EL \neq \emptyset$ **then**
    Ecke und Verbotsarray des Listenelements in HL, auf das f zeigt, seien z_f und VA_f, die Ecke des nachfolgenden Elements z_{f+1}.
    Erzeuge neues Listenelement (z_q, VA_q) und füge es in HL hinter der Stelle f ein.
    $VA_q := VA_f$.
    $VA_0 := VA_f$.
    for jedes $q \in EL$ **do**
        $z_q := q$.
        HüllRekursion(f).
        Markiere q in VA_f und VA_q als verboten.
    endfor
    Lösche Element (z_q, VA_q) aus Liste HL.
    $VA_f := VA_0$.
endif
if f zeigt auf letztes Element der Liste HL **then**
    Gib die Punkte von HL als Hüllfolge ζ aus.
else
    Setze f auf das nachfolgende Element.
    HüllRekursion(f).
endif

Algorithmus 7.4 HüllAufzählung(P)

Eingabe: Punktmenge (Array) $P = \{p_1, \ldots, p_n\}$.

Berechne die polarwinkelsortierten Listen $PWL(p_1), \ldots, PWL(p_n)$.
Berechne die konvexe Hülle von P.
Erzeuge eine verkettete Liste HL, die die konvexe Hüllfolge ζ_0 enthält. Setze alle Bits in allen Verbotsarrays von HL auf „nicht verboten".
Setze Pointer f auf das erste Element in HL.
HüllRekursion(f).

zur Berechnung der polarwinkelsortierten Listen und um die leeren Verbotsarrays anzulegen. Eine eventuelle Sortierung kollinearer Punkte nach Abstand ist in dieser Zeit bereits enthalten. Die Berechnung der konvexen Hülle wird hiervon dominiert. Der Zeitbedarf des gesamten Algorithmus ist daher $O(hn^2)$.

Die polarwinkelsortierten Listen benötigen zusammen $O(n^2)$ Platz, ebenso die verkettete Liste *HL*, in der die Verbotsarrays enthalten sind. Nach Satz 7.10 ist die Höhe des Rekursionsbaums $O(n)$, es belegen also zu jedem Zeitpunkt nur $O(n)$ Aufrufe von HüllRekursion Platz auf dem Laufzeit-Stack. Wir erhalten einen Gesamtplatzbedarf von $O(n^2)$. ■

Wie bereits am Anfang dieses Kapitels erwähnt, läßt sich unser Algorithmus so modifizieren, daß er eine (pseudo-)zufällige polygonale Hülle erzeugt. Hierzu wird in HüllRekursion von den rekursiven Aufrufen nur ein einziger, zufällig gewählter, ausgeführt. Da wir nicht wissen, wie viele polygonale Hüllen in den jeweiligen Rekursions-Unterbäumen aufgezählt würden, können wir keine gleichverteilte Erzeugung garantieren. Der Platzbedarf zur zufälligen Erzeugung ist genau wie zur Aufzählung $O(n^2)$. Der Zeitbedarf ist $O(n^2)$, denn es erfolgen nur $O(n)$ rekursive Aufrufe auf dem zufälligen Pfad von der Wurzel des Rekursionsbaums zu einem Blatt.

Statt zufälliger Auswahl rekursiver Aufrufe ist auch eine heuristische Auswahl möglich. Dabei können auch mehrere rekursive Aufrufe in einem Rekursionsknoten vorkommen. Die Heuristik sollte so gestaltet sein, daß sie zum einen den Rekursionsbaum deutlich verkleinert und zum anderen polygonale Hüllen mit bestimmten Eigenschaften erzeugt. Mit einem solchen Ansatz kann man zum Beispiel geeignete Gebiete für nicht-konvexe planare Triangulierungen suchen. Das Gebiet einer Triangulierung von Punktmenge P ist eine polygonale Hülle von P, wenn tatsächlich alle Punkte trianguliert werden und die Triangulierung keine Löcher aufweist. Als heuristische Kriterien für diese Anwendung können etwa ein möglichst geringer Abstand der neuen Ecke q vom Rand der konvexen Hülle oder ein möglichst großer Außenwinkel der neuen polygonalen Hülle in q dienen. Ist eine polygonale Hülle gefunden, dann wird P in ihr trianguliert. Auf diese Weise lassen sich ohne Interaktion nicht-konvexe Triangulierungen berechnen, die in Abhängigkeit von der Heuristik bestimmte Qualitätsmerkmale erfüllen.

Literaturverzeichnis

[AAGH$^+$86] Takao Asano, Tetsuo Asano, Leonidas Guibas, John Hershberger und Hiroshi Imai: Visibility of Disjoint Polygons. *Algorithmica*, Bd. 1, Nr. 1, S. 49–63, 1986.

[AB98] Nina Amenta und Marshall Bern: Surface Reconstruction by Voronoi Filtering. In *Fourteenth Annual Symposium on Computational Geometry, Minneapolis, Minnesota, June 7–10, 1998*, S. 39–48. ACM Press, New York, 1998.

[AHR95] Manuel Abellanas, Ferran Hurtado und Pedro A. Ramos: Redrawing a Graph within a Geometric Tolerance. In Roberto Tamassia (Hrsg.): *Graph Drawing, DIMACS International Workshop GD '94. Princeton, NJ, October 10–12, 1994*, Lecture Notes in Computer Science 894, S. 246–253. Springer-Verlag, 1995.

[Alb96] Lyuba Alboul: personal communication, 1996.

[AM91] Stephan Abramowski und Heinrich Müller: *Geometrisches Modellieren*. Reihe Informatik; Band 75. BI Wissenschaftsverlag, 1991.

[Aur87] Franz Aurenhammer: Power Diagrams: Properties, Algorithms and Applications. *SIAM Journal on Computing*, Bd. 16, Nr. 1, S. 77–96, Februar 1987.

[AvD95] L. Alboul und R. M. J. van Damme: Polyhedral Metrics in Surface Reconstruction: Tight Triangulations. Memorandum No. 1275, Universiteit Twente, Juli 1995.

[Baj92] Chandrajit L. Bajaj: Surface Fitting Using Implicit Algebraic Surface Patches. In Hagen [**Hag92b**], S. 23–52, 1992.

[BT86] Jean-Daniel Boissonnat und Monique Tellaud: An Hierarchical Representation of Objects: The Delaunay Tree. In *Second Annual Symposium on Computational Geometry, Yorktown Heights, NY, June 2–4, 1986*, S. 260–268. ACM Press, New York, 1986.

[CGL83] B. Chazelle, L. Guibas und D. Lee: The Power of Geometric Duality. In *24th Annual Symposium on Foundations of Computer Science* [**FOCS83**], S. 217–225, 1983. Auch als [**CGL85**] veröffentlicht.

[CGL85] Bernard Chazelle, Leo J. Guibas und D. T. Lee: The Power of Geometric Duality. *BIT*, Bd. 25, S. 76–90, 1985. Auch als [**CGL83**] veröffentlicht.

[Cha90] Bernard Chazelle: Triangulating a Simple Polygon in Linear Time. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, October 22–24, 1990*, S. 220–230. IEEE Computer Science Press, 1990. Auch als [**Cha91**] veröffentlicht.

[Cha91] B. Chazelle: Triangulating a Simple Polygon in Linear Time. *Discrete & Computational Geometry*, Bd. 6, S. 485–524, 1991. Auch als [**Cha90**] veröffentlicht.

[CR84] A. K. Cline und R. J. Renka: A Storage-Efficient Method for Construction of a Thiessen Triangulation. *Rocky Mountain Journal of Mathematics*, Bd. 14, Nr. 1, S. 119–139, 1984.

[Cra62] Paul Crantz: *Sphärische Trigonometrie*. Teubner, Leipzig, 1962.

[CSYL88] B. K. Choi, H. Y. Shin, I. Y. Yoon und J. W. Lee: Triangulation of Scattered Data in 3D Space. *Computer Aided Design*, Bd. 20, S. 238–248, 1988.

[dBvK+97] Mark de Berg, Marc van Kreveld, Mark Overmars und Otfried Schwarzkopf: *Computational Geometry. Algorithms and Applications*. Springer-Verlag, 1997.

[Del32] B. Delaunay: Neue Darstellung der geometrischen Kristallographie. *Zeitschrift für Kristallographie*, Bd. 84, S. 109–149, 1932.

[Del34] B. Delaunay: Sur la Sphère Vide. *Izvestija Akademii Nauk SSSR, Otdelenie Matematicheskich i Estestvennych Nauk*, Bd. 7, S. 793–800, 1934.

[DFFN+91] Leila De Floriani, Bianca Falcidieno, George Nagy und Caterina Pienovi: On Sorting Triangles in a Delaunay Tessellation. *Algorithmica*, Bd. 6, Nr. 4, S. 522–532, 1991.

[DFFP85] L. De Floriani, Bianca Falcidieno und Caterina Pienovi: Delaunay-Based Representation of Surfaces Defined Over Arbitrarily Shaped Domains. *Computer Vision, Graphics, and Image Processing*, Bd. 32, S. 127–140, 1985.

[DLR89] Nira Dyn, David Levin und Shmuel Rippa: Algorithms for the Construction of Data Dependent Triangulations. In M. G. Cox und J. C. Mason (Hrsg.): *Algorithms for Approximation II*, S. 185–192. Clarendon Press, Oxford, 1989.

[DLR90] Nira Dyn, David Levin und Shmuel Rippa: Data Dependent Triangulation for Piecewise Linear Interpolation. *IMA Journal of Numerical Analysis*, Bd. 10, S. 137–154, 1990.

[Ede87] Herbert Edelsbrunner: *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science 10. Springer-Verlag, 1987.

[Ede89] H. Edelsbrunner: An Acyclicity Theorem for Cell Complexes in d Dimensions. In *Fifth Annual Symposium on Computational Geometry, Saarbrucken, West Germany, June 5 - 7, 1989*, S. 145–151. ACM Press, New York, 1989. Auch als [**Ede90**] veröffentlicht.

[Ede90] H. Edelsbrunner: An Acyclicity Theorem for Cell Complexes in d Dimension. *Combinatorica*, Bd. 10, Nr. 3, S. 251–260, 1990. Auch als [**Ede89**] veröffentlicht.

[EGA81] H. El Gindy und D. Avis: A Linear Algorithm for Computing the Visibility Polygon from a Point. *Journal of Algorithms*, Bd. 2, S. 186–197, 1981.

[EM90] H. Edelsbrunner und E. P. Mücke: Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics*, Bd. 9, Nr. 1, S. 66–104, 1990.

[EOS83] H. Edelsbrunner, J. O'Rourke und R. Seidel: Constructing Arrangements of Lines and Hyperplanes with Applications. In *24th Annual Symposium on Foundations of Computer Science* [**FOCS83**], S. 83–91, 1983. Auch als [**EOS86**] veröffentlicht.

[EOS86] H. Edelsbrunner, J. O'Rourke und R. Seidel: Constructing Arrangements of Lines and Hyperplanes with Applications. *SIAM Journal on Computing*, Bd. 15, S. 341–363, 1986. Auch als [**EOS83**] veröffentlicht.

[ES92] Herbert Edelsbrunner und Nimish R. Shah: Incremental Topological Flipping Works for Regular Triangulations. In David Avis (Hrsg.): *8th Annual Symposium on Computational Geometry, Berlin, Germany, June 10–12, 1992*, S. 43–52. ACM Press, New York, 1992.

[ES96] H. Edelsbrunner und N. R. Shah: Incremental Topological Flipping Works for Regular Triangulations. *Algorithmica*, Bd. 15, Nr. 3, S. 223–241, 1996.

[ETW90] Herbert Edelsbrunner, Tiow Seng Tan und Roman Waupotitsch: An $O(n^2 \log n)$ Time Algorithm for the MinMax Angle Triangulation. In *Sixth Annual Symposium on Computational Geometry*, S. 44–52. ACM Press, New York, 1990.

[Fac95] Michael A. Facello: Implementation of a Randomized Algorithm for Delaunay and Regular Triangulations in Three Dimensions. *Computer Aided Geometric Design*, Bd. 12, S. 349–370, 1995.

[Fie92] David A. Field: Delaunay Criteria for Triangulating Surfaces. In Joseph D. Warren (Hrsg.): *Curves and Surfaces in Computer Vision and Graphics III*, Proc. SPIE Vol. 1830, S. 237–246, 1992.

[FOCS83] IEEE: *24th Annual Symposium on Foundations of Computer Science, Tucson, AZ, November 7–9, 1983*. IEEE Computer Science Press, 1983.

[Fra82] Richard Franke: Scattered Data Interpolation: Test of Some Methods. *Mathematics of Computation*, Bd. 38, Nr. 157, S. 181–200, Januar 1982.

[Fri97] Joachim Friedhoff: *Aufbereitung von 3D-Digitalisierdaten für den Werkzeug–, Formen– und Modellbau*. Dissertation, Universität Dortmund, Fakultät Maschinenbau, September 1996. Vulkan Verlag, Essen, 1997.

[GKS90] Leonidas J. Guibas, Donald E. Knuth und Micha Sharir: Randomized Incremental Construction of Delaunay and Voronoi Diagrams. In *Automata, Languages and Programming. Proceedings of the* 17^{th} *international colloquium. Warwick University, England, July 16 - 20, 1990*, Lecture Notes in Computer Science 443, S. 414–431. Springer-Verlag, 1990.

[Gra72] R. L. Graham: An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, Bd. 1, S. 132–133, 1972.

[GS69] K. R. Gabriel und R. R. Sokal: A New Statistical Approach to Geographic Variation Analysis. *Systematic Zoology*, Bd. 18, S. 259–278, 1969.

[Hag92a] Hans Hagen (Hrsg.): *Curve and Surface Design*. SIAM, Philadelphia, PA, 1992.

[Hag92b] Hans Hagen (Hrsg.): *Topics in Surface Modeling*. SIAM, Philadelphia, PA, 1992.

[Ham23] E. Hammer: *Lehr– und Handbuch der ebenen und sphärischen Trigonometrie*. J. B. Metzlersche Verlagsbuchhandlung, Stuttgart, 1923.

[HMN93] Hans Hagen, Heinrich Müller und Gregory M. Nielson (Hrsg.): *Focus on Scientific Visualization*. Springer-Verlag, 1993.

[HNU96] F. Hurtado, M. Noy und J. Urrutia: Flipping Edges in Triangulations. In [**SCG96**], S. 214–223, 1996.

[IIM85] Hiroshi Imai, Masao Iri und Kazuo Murota: Voronoi Diagram in the Laguerre Geometry and its Applications. *SIAM Journal on Computing*, Bd. 14, Nr. 1, S. 93–105, Februar 1985.

[JS87] B. Joe und R. B. Simpson: Corrections to Lee's Visibility Polygon Algorithm. *BIT*, Bd. 27, S. 458–473, 1987.

[KET90] Xianshu Kong, Hazel Everett und Godfried Toussaint: The Graham Scan Triangulates Simple Polygons. *Pattern Recognition Letters*, Bd. 11, Nr. 11, S. 713–716, November 1990.

[Law77] C. L. Lawson: Software for C^1 Surface Interpolation. In J. R. Rice (Hrsg.): *Mathematical Software III*, S. 161–194. Academic Press, New York, 1977.

[Law84] Charles L. Lawson: C^1 Surface Interpolation for Scattered Data on a Sphere. *Rocky Mountain Journal of Mathematics*, Bd. 14, Nr. 1, S. 177–202, 1984.

[Lee83a] D. T. Lee: On Finding the Convex Hull of a Simple Polygon. *International Journal of Computer and Information Sciences*, Bd. 12, Nr. 2, S. 87–98, 1983.

[Lee83b] D. T. Lee: Visibility of a Simple Polygon. *Computer Vision, Graphics, and Image Processing*, Bd. 22, S. 207–221, 1983.

[LP79] D. T. Lee und F. P Preparata: An Optimal algorithm for Finding the Kernel of a Polygon. *Journal of the ACM*, Bd. 26, Nr. 3, S. 415–421, Juli 1979.

[LR78] B. A. Lewis und J. S. Robinson: Triangulation of Planar Regions With Applications. *The Computer Journal*, Bd. 21, S. 324–332, 1978.

[Max93] Nelson L. Max: Sorting for Polyhedron Compositing. In Hagen et al. [**HMN93**], S. 259–268, 1993.

[Meg83] Nimrod Megiddo: Linear-Time Algorithms for Linear Programming in R^3 and Related Problems. *SIAM Journal on Computing*, Bd. 12, Nr. 4, S. 759–776, November 1983.

[Mei75] G. H. Meisters: Polygons Have Ears. *American Mathematical Monthly*, Bd. 82, S. 684–651, 1975.

[MHC90] Nelson Max, Pat Hanrahan und Roger Crawfis: Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions. *Computer Graphics*, Bd. 24, Nr. 5, S. 27–33, November 1990.

[MII96] T. Masada, H. Imai und K. Imai: Enumeration of Regular Triangulations. In *Twelfth Annual Symposium on Computational Geometry* [**SCG96**], S. 224–233, 1996.

[MLLM$^+$92] Stephen Mann, Charles Loop, Michael Lounsbery, David Meyers, James Painter, Tony DeRose und Kenneth Sloan: A Survey of Parametric Scattered Data Fitting Using Triangular Interpolants. In Hagen [**Hag92a**], S. 145–172, 1992.

[MM97] Robert Mencl und Heinrich Müller: Interpolation and Approximation of Surfaces from Three-Dimensional Scattered Data Points. Forschungsbericht 662/1997, Universität Dortmund, FB Informatik, 44221 Dortmund, Germany, Dezember 1997. Auch als [**MM98**] veröffentlicht.

[MM98] Robert Mencl und Heinrich Müller: Interpolation and Approximation of Surfaces from Three-Dimensional Scattered Data Points. In *Eurographics '98 State-of-the-Art Reports*, S. 51–67, 1998. Auch als [**MM97**] veröffentlicht.

[MSZ96] Ernst P. Mücke, Isaac Saias und Binhai Zhu: Fast Randomized Point Location without Preprocessing in Two- and Three-Dimensional Delaunay Triangulations. In *Twelfth Annual Symposium on Computational Geometry* [**SCG96**], S. 274–283, 1996.

[NHM97] Gregory M. Nielson, Hans Hagen und Heinrich Müller (Hrsg.): *Scientific Visualization : Overviews, Methodologies, and Techniques*. IEEE Computer Society Press, 1997.

[Nie97] Gregory M. Nielson: Tools for Triangulations and Tetrahedrizations and Constructing Functions Defined over Them. In Nielson et al. [**NHM97**], S. 429–525, 1997.

[O'R87] Joseph O'Rourke: *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.

[OW88] Mark H. Overmars und Emo Welzl: New Methods for Computing Visibility Graphs. In *Fourth Annual Symposium on Computational Geometry, Urbana-Champaign, Illinois, June 6–8, 1988*, S. 164–171. ACM Press, New York, 1988.

[PS82] Christos H. Papadimitriou und Kenneth Steiglitz: *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[PS85] Franco P. Preparata und Michael Ian Shamos: *Computational Geometry*. Springer-Verlag, 1985.

[PS97] Giovanni Prestifilippo und Joachim Sprave: Optimal Triangulation by Means of Evolutionary Algorithms. In *Second Int'l Conf. Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'97), Glasgow, UK, September 1–4, 1997*, S. 492–497. IEE Press, London, 1997.

[PVCR90] O. Palacios-Velez und B. Cuevas Renaud: A Dynamic Hierarchical Subdivision Algorithm for Computing Delaunay Triangulations and Other Closest-Point Problems. *ACM Transactions on Mathematical Software*, Bd. 16, Nr. 3, S. 275–292, September 1990.

[Ram96] Pedro A. Ramos: *Tolerancia de Estructuras Geométricas y Combinatorias*. Dissertation, Universidad Politécnica de Madrid, Departamento de Matemática Aplicada, Facultad de Informática, 1996.

[Rog85] David F. Rogers: *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.

[Sal91] David Henry Salesin: *Epsilon Geometry : Building Robust Algorithms from Imprecise Computations*. PhD Dissertation, Stanford University, Department of Computer Science, 1991. Auch als Technical Report STAN-CS 91-1398 veröffentlicht.

[SB97] Th. Schreiber und G. Brunnett: Approximating 3D Objects from Measured Points. In *30th ISATA, Florence, Italy, June 16–19, 1997*. Automotive Automation Ltd., Croydon, UK, 1997.

[SCG96] *Twelfth Annual Symposium on Computational Geometry, Philadelphia, PA, May 24–26 1996*. ACM Press, New York, 1996.

[Sch93] Larry L. Schumaker: Computing Optimal Triangulations Using Simulated Annealing. *Computer Aided Geometric Design*, Bd. 10, S. 329–345, 1993.

[Sch94] Thomas Schreiber: *Analyse und Approximation von unstrukturierten Daten und Freiformflächen*. Dissertation, Universität Kaiserslautern, Fachbereich Informatik, Februar 1994.

[Sch96] Thomas Schreiber: Approximation of 3D Objects. In G. Farin, H. Bieri, G. Brunnett und T. DeRose (Hrsg.): *Geometric Modeling: Dagstuhl 1996*, Computing Supplementum 13. Springer-Verlag, 1996.

[Sei98] R. Seidel: The Nature and Meaning of Perturbations in Geometric Computing. *Discrete & Computational Geometry*, Bd. 19, Nr. 1, S. 1–17, Januar 1998.

[SH98] Gerik Scheuermann und Hans Hagen: A Data Dependent Triangulation for Vector Fields. In *Computer Graphics International ’98*, S. 96–102. IEEE, 1998.

[vDA95] R. van Damme und L. Alboul: Tight Triangulations. In Morten Dæhlen, Tom Lyche und Larry L. Schumaker (Hrsg.): *Mathematical Methods for Curves and Surfaces*, S. 517–526. Vanderbilt University Press, 1995.

[Wel85] Emo Welzl: Constructing the Visibility Graph for n Line Segments in $O(n^2)$ Time. *Information Processing Letters*, Bd. 20, S. 167–171, 1985.

[Wel96] Frank Weller: Stable Edges in Delaunay Triangulations. Forschungsbericht 637/1996, Universität Dortmund, FB Informatik, 44221 Dortmund, Germany, Dezember 1996.

[Wil92] Peter L. Williams: Visibility Ordering Meshed Polyhedra. *ACM Transactions on Graphics*, Bd. 11, Nr. 2, S. 103–126, April 1992.

[WMAD98] K. Weinert, J. Mehnen, F. Albersmann und P. Drerup: New Solutions for Surface Reconstruction from Discrete Point Data by Means of Computational Intelligence. In *CIRP International Seminar on Intelligent Computation in Manufacturing Engineering (ICME'98), Capri, Italy, July 1–3, 1998*, S. 431–438, 1998.

[WS85] T. C. Woo und S. Y. Shin: A Linear Time Algorithm for Triangulating a Point-Visible Polygon. *ACM Transactions on Graphics*, Bd. 4, Nr. 1, S. 60–70, Januar 1985.

[ZSSM96] Chong Zhu, Gopalakrishnan Sundaram, Jack Snoeyink und Joseph S. B. Mitchell: Generating Random Polygons with Given Vertices. *Computational Geometry*, Bd. 6, Nr. 5, S. 277–290, September 1996.