Kapitel 3

Das Triangulierungsverfahren

Im letzten Kapitel haben wir ein einfaches Verfahren zur Approximation mit DMS-Splines kennengelernt. Dazu wird eine Triangulierung benötigt, die besonderen Anforderungen genügen muß. Im ersten Abschnitt dieses Kapitels sollen die Grundlagen, die für das im dritten Abschnitt vorgestellte Triangulierungsverfahren, welches das bereits beschriebene Problem löst, geschaffen werden. Das Verfahren, das im dritten Abschnitt vorgestellt wird, greift teilweise auf bereits bekannte Algorithmen zurück. Diese werden in Abschnitt zwei noch einmal vorgestellt. Zunächst kommen wir also zu den Grundlagen.

3.1 Theoretische Grundlagen

Es sollen zu Beginn einige Bezeichnungen festgelegt werden.

Definition 3.1 (ungerichtete, gerichtete und offene Kante)
Eine Kante $\mathbf{t}_1 \Leftrightarrow \mathbf{t}_2$ *ist die konvexe Hülle* $[\{\mathbf{t}_1, \mathbf{t}_2\}]$ *der beiden Punkte* $\mathbf{t}_1, \mathbf{t}_2 \in \mathbb{R}^2$. *Eine gerichtete Kante wird mit* $\mathbf{t}_1 \rightarrow \mathbf{t}_2$ *bezeichnet. Eine offene Kante* $\mathbf{t}_1 \overset{\circ}{\Leftrightarrow} \mathbf{t}_2$ *ist gegeben durch* $[\{\mathbf{t}_1, \mathbf{t}_2\}] \setminus \{\mathbf{t}_1, \mathbf{t}_2\}$.

Definition 3.2 (Lage von Punkten, Innen- und Außenkante einer Triangulierung, Nachbarschaft)

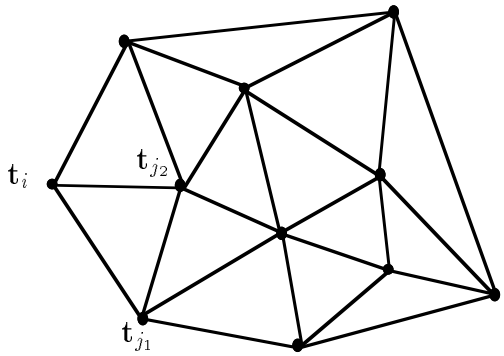
1. *Gegeben seien drei Punkte* $\mathbf{t}_i = (x_i, y_i) \in \mathbb{R}^2$ $(i = 1, 2, 3)$ *mit* $\mathbf{t}_2 \neq \mathbf{t}_3$. *Dann sagen wir* $\mathbf{t}_1$ *liegt links von* $\mathbf{t}_2 \rightarrow \mathbf{t}_3$ *, falls gilt*

$$(x_3 \Leftrightarrow x_2)(y_1 \Leftrightarrow y_2) \geq (x_1 \Leftrightarrow x_2)(y_3 \Leftrightarrow y_2).$$

   Wir sagen echt links statt links, falls in der Ungleichung sogar $>$ *gilt. Weiter heißt es echt rechts, falls* $<$ *in der Ungleichung gilt.*

2. *Gegeben sei eine Triangulierung* $\mathcal{T} = \mathcal{T}(\mathcal{S})$. *Existiert zu zwei Punkten* $\mathbf{t}_i, \mathbf{t}_j \in \mathcal{S}$ *ein Dreieck* $[\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k]$ *oder* $[\mathbf{t}_j, \mathbf{t}_i, \mathbf{t}_l]$ *aus* $\mathcal{T}$, *so nennen wir* $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j$ *eine* Kante von $\mathcal{T}$. *Existiert nur ein Dreieck, so ist* $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j$ *eine* Außenkante, *existieren beide, so ist* $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j$ *eine* Innenkante.

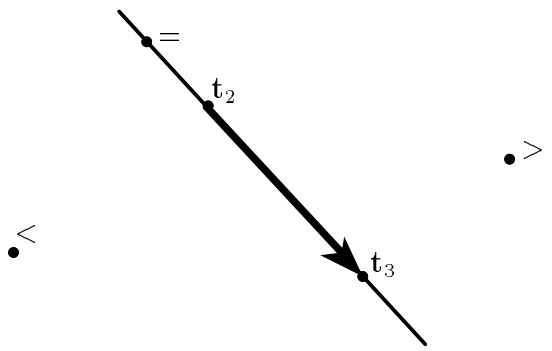
3. *Existiert eine Kante* $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j$, *so sind* $\mathbf{t}_i$ *und* $\mathbf{t}_j$ Nachbarn. *Die* Nachbarschaft $(\mathbf{t}_{j_1}, ..., \mathbf{t}_{j_g})$ *eines Punktes* $\mathbf{t}_i$ *nennen wir die entgegen dem Uhrzeigersinn geordnete Liste aller seiner Nachbarn. Der erste Nachbar ist frei wählbar, sofern* $\mathbf{t}_i$ *nicht auf dem Rand der Triangulierung liegt. Ansonsten ist der erste Nachbar* $\mathbf{t}_{j_1}$ *eindeutig bestimmt durch die Tatsache, daß auch er auf dem Rand der Triangulierung liegt und mit* $\mathbf{t}_{j_2}$ *und* $\mathbf{t}_i$ *ein Dreieck* $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_i]$ *der Triangulierung bildet.*



Bemerkung 3.3

1. Definition 3.2 *1.* hat folgende geometrische Bedeutung: Gilt in der Ungleichung

   - $>$, so liegt $\mathbf{t}_1$ links der *gerichteten* Geraden durch $\mathbf{t}_2$ *und* $\mathbf{t}_3$
   - $=$, so liegt $\mathbf{t}_1$ auf der Geraden
   - $<$, so liegt $\mathbf{t}_1$ rechts der Geraden.



2. Gegeben sei ein Dreieck $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$. Da wir eine Ordnung entgegen des Uhrzeigersinns vereinbart haben, gilt

   - $\mathbf{t}_1$ liegt echt links von $\mathbf{t}_2 \rightarrow \mathbf{t}_3$,
   - $\mathbf{t}_2$ liegt echt links von $\mathbf{t}_3 \rightarrow \mathbf{t}_1$ und
   - $\mathbf{t}_3$ liegt echt links von $\mathbf{t}_1 \rightarrow \mathbf{t}_2$.

3. Diese Ungleichung können wir vielfältig für die Implementierung verwenden, insbesondere um die Lage der Daten zu untersuchen. Z.B. liegt ein Datum $\mathbf{d}_i$ genau dann in einem Dreieck $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$, falls gilt:

   - $\mathbf{d}_i$ liegt links von $\mathbf{t}_1 \rightarrow \mathbf{t}_2$,
   - $\mathbf{d}_i$ liegt links von $\mathbf{t}_2 \rightarrow \mathbf{t}_3$ und
   - $\mathbf{d}_i$ liegt links von $\mathbf{t}_3 \rightarrow \mathbf{t}_1$.

Folgendes Lemma findet sich bei Schumaker [Shu87].

Lemma 3.4 (Anzahl an Dreiecken einer Triangulierung)
Die Anzahl m an Dreiecken einer Triangulierung $\mathcal{T} = \mathcal{T}(\mathcal{S})$ einer Punktmenge $\mathcal{S}$ ist durch $\mathcal{S}$ eindeutig festgelegt. Über Induktion kann gezeigt werden, daß gilt:

$$m = 2n \Leftrightarrow n_a \Leftrightarrow 2.$$

Hierbei ist $n = |\mathcal{S}|$ die Gesamtanzahl an Punkten und $n_a = |\mathcal{S} \cap \partial[\mathcal{S}]|$ die Anzahl an Punkten auf dem Rand der konvexen Hülle von $\mathcal{S}$ und somit auf dem Rand der Triangulierung.

Zu einer Menge $\mathcal{S}$ von Punkten existieren i.allg. mehrere Triangulierungen $\mathcal{T}_i(\mathcal{S})$, $i \in I$. Die Gestalt der Dreiecke einer Triangulierung ist aus numerischen Gründen von erheblicher Bedeutung. Um aus den möglichen Triangulierungen einer Punktmenge $\mathcal{S}$ eine möglichst *gute* auszuwählen, versucht man eine gegebene Triangulierung zu optimieren. Solche Optimierungsalgorithmen (einer wird in Abschnitt 3.2.2.3 vorgestellt) testen in konvexen Vierecken, die sich aus zwei Dreiecken der Triangulierung ergeben, die eine gemeinsame Kante besitzen, ob ein Diagonalentausch die Triangulierung bzgl. eines bestimmten Kriteriums verbessern würde. Ein paar wichtige Begriffe zur Optimierung von Triangulierungen sollen nun eingeführt werden. Solche Grundlagen finden sich z.B. in [Shu87], [ClR84] oder [HoL93].

Definition 3.5 (Diagonalentausch und Tauschtest)

1. *Gegeben seien zwei benachbarte Dreiecke $[\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k]$ und $[\mathbf{t}_i, \mathbf{t}_k, \mathbf{t}_l]$ einer Triangulierung $\mathcal{T}$, die ein echt konvexes Viereck bilden, d.h.*

(a) $[\{\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k, \mathbf{t}_l\}] = [\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k] \cup [\mathbf{t}_i, \mathbf{t}_k, \mathbf{t}_l]$ *und*

(b) keine drei Punkte sind kollinear.

Einen Diagonalentausch nennen wir das Ersetzen von

$$\{\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k\} \text{ und } \{\mathbf{t}_i, \mathbf{t}_k, \mathbf{t}_l\}$$

in $\mathcal{T}$ *durch*

$$\{\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_l\} \text{ und } \{\mathbf{t}_j, \mathbf{t}_k, \mathbf{t}_l\}.$$

2. *Gegeben sei die Kante* $\mathbf{t}_i \Leftrightarrow \mathbf{t}_k$ *einer Triangulierung* $\mathcal{T}(\mathcal{S})$ *von* $\mathcal{S}$. *Ein Tauschtest angewandt auf* $\mathbf{t}_i \Leftrightarrow \mathbf{t}_k$ *ist positiv bzgl. eines Kriteriums K, falls die folgenden Punkte erfüllt sind:*

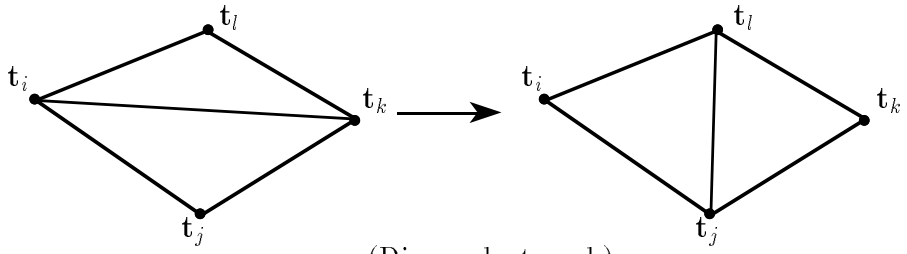
   - $\mathbf{t}_i \Leftrightarrow \mathbf{t}_k$ *ist eine Innenkante.*
   - *Die beiden Dreiecke* $[\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k]$ *und* $[\mathbf{t}_i, \mathbf{t}_k, \mathbf{t}_l]$ *von* $\mathcal{T}$ *bilden ein echt konvexes Viereck* $[\{\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k, \mathbf{t}_l\}]$.
   - *Die Triangulierung*

     $$\mathcal{T}_1(\tilde{S}) = \{\{\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_l\}, \{\mathbf{t}_j, \mathbf{t}_k, \mathbf{t}_l\}\}$$

     von $\tilde{S} = \{\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k, \mathbf{t}_l\}$ *ist bzgl. des Kriteriums K besser als die Triangulierung*

     $$T_2(\tilde{S}) = \{\{\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k\}, \{\mathbf{t}_i, \mathbf{t}_k, \mathbf{t}_l\}\}.$$

   Sonst ist der Tauschtest negativ.



(Diagonalentausch)

Ein wichtiges Kriterium zur Optimierung von Triangulierungen ist das folgende:

Definition 3.6 (Max-Min-Winkelkriterium)
$\tilde{\mathcal{T}}$ ist eine bessere Triangulierung als $\mathcal{T}$ im Sinne des Max-Min-Winkelkriteriums, falls gilt

$$\min_{i \in \{1,\ldots,m\}} \{a(T_i) : T_i \in \tilde{\mathcal{T}}\} > \min_{i \in \{1,\ldots,m\}} \{a(T_i) : T_i \in \mathcal{T}\}.$$

Dabei bezeichnet $a(T_i)$ *den kleinsten Innenwinkel im Dreieck* $[T_i]$.

Da bei den in dieser Arbeit behandelten Triangulierungen nicht nur auf die Gestalt der Dreiecke sondern auch auf die Lage der Daten geachtet werden muß, wird nun ein neues Kriterium eingeführt.

Definition 3.7 (erweitertes Max-Min-Winkelkriterium)
Gegeben sei eine Triangulierung $\mathcal{T}(\mathcal{S})$ und eine Datenmenge $\mathcal{D}$. Jedes Dreieck $[T_i]$ der Triangulierung $\mathcal{T}$ beinhalte mindestens d Daten aus $\mathcal{D}$:

$$|[\mathcal{T}_i] \cap \mathcal{D}| \geq d \quad \forall i = 1, ..., m.$$

$\tilde{\mathcal{T}}(\mathcal{S})$ ist eine bessere Triangulierung als $\mathcal{T}(\mathcal{S})$ bzgl. des erweiterten Max-Min-Winkelkriteriums, falls

- *$\tilde{\mathcal{T}}$ eine bessere Triangulierung bzgl. des Max-Min-Winkelkriteriums ist, und*
- *in jedem Dreieck $[T_i]$ der Triangulierung $\tilde{\mathcal{T}}$ auch mindestens d Daten enthalten sind.*

Durch Optimierungsalgorithmen werden i.allg. lokal optimale Triangulierungen angestrebt und erreicht:

Definition 3.8 (lokal optimale Triangulierungen)

1. *Eine Kante $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j$ einer Triangulierung ist lokal optimal (bzgl. eines Kriteriums K), falls ein Tauschtest angewandt auf $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j$ negativ ist.*
2. *Eine Triangulierung ist lokal optimal, falls alle Kanten lokal optimal sind.*

Bemerkung 3.9 (Bezeichnungen optimaler Triangulierungen)
Eine Triangulierung, die bzgl. des Max-Min-Winkelkriteriums lokal optimal ist, heißt Delaunay-, Dirichlet-, Thiessen- oder Voronoi-Triangulierung.

Noch schärfer sind die Anforderungen an globale Optimalität:

Definition 3.10 (global optimale Triangulierugen)
Sei a(T) eine numerische Berechnung der Güte eines Dreiecks bzgl. eines Kriteriums K (z.B. wäre $a(T)$:=kleinster Innenwinkel im Dreieck $[T]$ die Definition von a(T) beim Max-Min-Winkelkriterium). Desweiteren seien im Vektor

$$a(\mathcal{T}) := (a(T_1), ..., a(T_m))$$

die Güte der Dreiecke T_i der Triangulierung $\mathcal{T}$ in der Form eingetragen, daß T_{i+1} bzgl. K besser (bzw. gleich gut) ist als (bzw. wie) T_i für alle $i \in \{1, ..., m \Leftrightarrow 1\}$. Dann ist $\mathcal{T}(\mathcal{S})$ eine global optimale Triangulierung der Menge $\mathcal{S}$ bzgl. K, falls für jede Triangulierung $\tilde{\mathcal{T}}(\mathcal{S})$ gilt:

Zu jedem Element $T_i \in \mathcal{T}$ $(i \in \{1, ..., m\})$ *mit*

$$\tilde{T}_i \in \tilde{\mathcal{T}} \text{ ist (echt) besser bzgl. } K \text{ als } T_i$$

existiert ein Element $T_j \in \mathcal{T}$ $(j \in \{1, ..., i \Leftrightarrow 1\})$ *für welches gilt:*

$$\tilde{T}_j \in \tilde{\mathcal{T}} \text{ ist (echt) schlechter bzgl. } K \text{ als } T_j.$$

Eine lokal optimale Triangulierung ist nicht notwendig global optimal. Bei Delaunay-Triangulierungen ist das anders.

Satz 3.11 (globale Optimalität der Delaunay-Triangulierung)
Eine Delaunay-Triangulierung ist bzgl. des Max-Min-Winkelkriteriums global optimal.

Beweis: (siehe Lawson [Law77]). ■

Hat man eine Punktmenge $\mathcal{S} = \{\mathbf{t}_i \in \mathbb{R}^2 : i = 1, ..., n\}$ gegeben, so kann man die Ebene $\mathbb{R}^2$ in Voronoi-Regionen $D(\mathbf{t}_i, \mathcal{S})$ unterteilen, deren abgeschlossene Vereinigung erneut die gesamte Ebene $\mathbb{R}^2$ ergibt. Zum einen benötigen wir diese Voronoi-Parkettierung bei einem später beschriebenen Clustering-Algorithmus (siehe Abschnitt 3.2.3), zum anderen wird durch Satz 3.15 klar, woher der Begriff Voronoi-Triangulierung kommt.

Definition 3.12 (Voronoi-Region, Voronoi-Diagramm)
Sei $\mathcal{S} = \{\mathbf{t}_1, ..., \mathbf{t}_n\} \subset \mathbb{R}^2$ *gegeben. Desweiteren sei*

$$D(\mathbf{t}_i, \mathbf{t}_j) := \{\mathbf{z} \in \mathbb{R}^2 : |\mathbf{t}_i \Leftrightarrow \mathbf{z}| < |\mathbf{t}_j \Leftrightarrow \mathbf{z}|\} \quad \forall \mathbf{t}_i, \mathbf{t}_j \in \mathcal{S} \text{ mit } \mathbf{t}_i \neq \mathbf{t}_j.$$

Dann nennen wir

$$D(\mathbf{t}_i, \mathcal{S}) := \bigcap_{\substack{j \in \{1,...,n\} \\ i \neq j}} D(\mathbf{t}_i, \mathbf{t}_j)$$

die (offene) Voronoi-Region *von* $\mathbf{t}_i \in \mathcal{S}$ *(bzgl.* $\mathcal{S}$*).* $\mathbf{t}_i$ *heißt* Voronoi-Center-Punkt*. Weiter nennen wir*

$$V(\mathcal{S}) := \bigcup_{i \in \{1,...,n\}} \partial D(\mathbf{t}_i, \mathcal{S})$$

das Voronoi-Diagramm *von* $\mathcal{S}$*.*

Bemerkung 3.13
Offensichtlich bildet der Rand einer Voronoi-Region einen Polygonzug.

Definition 3.14 (Voronoi-Nachbarn)
Sei $\mathcal{S} = \{\mathbf{t}_1, ..., \mathbf{t}_n\} \subset \mathbb{R}^2$ *gegeben. Zwei Punkte* $\mathbf{t}_i, \mathbf{t}_j$ *mit* $i \neq j$ *heißen* Voronoi-Nachbarn, *falls gilt:*

$$\partial D(\mathbf{t}_i, \mathcal{S}) \cap \partial D(\mathbf{t}_j, \mathcal{S}) \neq \emptyset.$$

Man bezeichnet $\mathbf{t}_i, \mathbf{t}_j$ *als* schwache Voronoi-Nachbarn, *falls der Durchschnitt der beiden Voronoi-Regionen exakt ein Punkt ist. Ist der Durchschnitt eine ganze Kante, so spricht man von* starken Voronoi-Nachbarn.

Der folgende Satz wird oft als Definition für Delaunay-Triangulierungen verwendet. Es wird auch klar, weshalb diese Triangulierung auch Voronoi-Triangulierung heißt.

Satz 3.15 (Voronoi-Triangulierung)
Eine Triangulierung $\mathcal{T} = \mathcal{T}(\mathcal{S})$ *der Menge* $\mathcal{S} = \{\mathbf{t}_1, ..., \mathbf{t}_n\} \subset \mathbb{R}^2$ *ist genau dann eine Voronoi- bzw. Delaunay-Triangulierung, wenn alle starken Voronoi-Nachbarn* $\mathbf{t}_i, \mathbf{t}_j$ *der Menge* $\mathcal{S}$ *eine Kante* $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j$ *von T bilden.*

Beweis: (siehe [Law77]).

■

Bemerkung 3.16
Inhaltlich findet man die Aussage des obigen Satzes auch bei Schumaker [Shu87]. Dort ist lediglich die Delaunay-Triangulierung über die Voronoi-Nachbar-Beziehung definiert, und dann wird erwähnt, daß diese bzgl. des Max-Min-Winkelkriteriums lokal optimal ist. Zum Beweis wird auch dort auf Lawson [Law77] verwiesen. In diesem Artikel wird die Äquivalenz dreier Kriterien gezeigt: das Max-Min-Winkelkriterium, das sogenannte Voronoi-Regionen-Kriterium und das Umkreiskriterium. Beide neuen Kriterien sind von der Anschauung etwas schwieriger, und es soll an dieser Stelle nicht mehr genauer auf sie eingegangen werden, da sich aufgrund der Äquivalenz zum Max-Min-Winkelkriterium keine weiteren Erkenntnisse ergeben. Es sei abschließend lediglich noch erwähnt, daß man den Beweis von Satz 3.11 mit Hilfe der Voronoi-Regionen führen kann.

3.2 Bekannte Algorithmen

In dem folgenden Abschnitt werden Algorithmen zu den Problemen konvexe Hülle, Delaunay-Triangulierung und Clustering vorgestellt. Zu solchen geometrischen Problemstellungen gibt es eine große Menge an Literatur. Z.B. beschäftigen sich F. P. Preparata und M. I. Shamos [PSh85], G. Aumann und K. Spitzmüller [AuS93] oder E. Quaisser [Qua94] mit deren Lösungen. In Software-Paketen wie

z.B. *Mathematica* sind Implementierungen solcher Algorithmen bereits enthalten. Auch kann man sich Implementierungen in verschiedenen Programmiersprachen aus dem Internet ziehen. Da Lösungen zu den behandelten Problemen in dem in Abschnitt 3.3 vorgestellten Algorithmus benötigt werden, sei der Vollständigkeit halber jeweils ein Beispiel zur Lösung hier aufgeführt.

3.2.1 Konvexe Hülle

Der im folgenden vorgestellte Algorithmus bestimmt aus einer endlichen Menge $\mathcal{S}$ von Punkten im $\mathbb{R}^2$ zunächst einen geschlossenen Polygonzug, der sich nicht überschneidet und alle Punkte beinhaltet oder umschließt (vgl. auch das darauffolgende Beispiel). Im Anschluß wird aus diesem Polygonzug die konvexe Hülle generiert. Man findet ihn in [AuS93].

Algorithmus 3.17 (konvexe Hülle)
Gegeben sei die Menge $\mathcal{S} = \{\mathbf{t}_1, ..., \mathbf{t}_n\} \subset \mathbb{R}^2$. *Gesucht ist eine Liste* $\pi = (\mathbf{t}_{i_1}, ..., \mathbf{t}_{i_l})$, *die einen Polygonzug (entgegen dem Uhrzeigersinn) beschreibt, der genau die Punkte auf dem Rand der konvexen Hülle beinhaltet.*

Schritt 1: *Bestimme die Punkte* $\mathbf{t}_l$ *und* $\mathbf{t}_r$ *aus* $\mathcal{S}$ *mit minimaler bzw. maximaler x-Koordinate. (Existieren mehrere solcher Punkte, wähle für* $\mathbf{t}_l$ *den mit kleinster und für* $\mathbf{t}_r$ *den mit größter y-Koordinate.)*

Schritt 2: *Bestimme zwei Mengen* $\mathcal{S}_o$ *und* $\mathcal{S}_u$ *der Punkte aus* $\mathcal{S}$, *die oberhalb bzw. unterhalb der durch* $\mathbf{t}_l$ *und* $\mathbf{t}_r$ *bestimmten Gerade liegen. (*$\mathbf{t}_l$ *und* $\mathbf{t}_r$ *sollen zu beiden Mengen gehören, sonstige Punkte auf der Geraden werden nicht hinzugenommen, da sie nicht zur konvexen Hülle gehören können.)*

Schritt 3: *Bilde zwei Folgen* π_o *bzw.* π_u *der Punkte aus* $\mathcal{S}_o$ *bzw.* $\mathcal{S}_u$, *so daß die x-Werte bei* π_o *ab- und bei* π_u *zunehmen. (Existieren mehrere Punkte mit gleichen x-Werten, so nehme in* π_o *nur den mit dem größten y-Wert auf und bei* π_u *nur den mit dem kleinsten, wobei Punkte mit gleichen x-Werten wie* $\mathbf{t}_l$ *bzw.* $\mathbf{t}_r$ *von dieser Regel ausgenommen sind.)*

Schritt 4: *Verbinde* π_o *und* π_u *zu einem geschlossenen Polygonzug* π.

Schritt 5: *Für je drei aufeinanderfolgende Punkte* $\mathbf{t}_{i_1}, \mathbf{t}_{i_2}, \mathbf{t}_{i_3}$ *aus* π *muß gelten:*

$$\mathbf{t}_{i_3} \text{ liegt links von } \mathbf{t}_{i_1} \rightarrow \mathbf{t}_{i_2}.$$

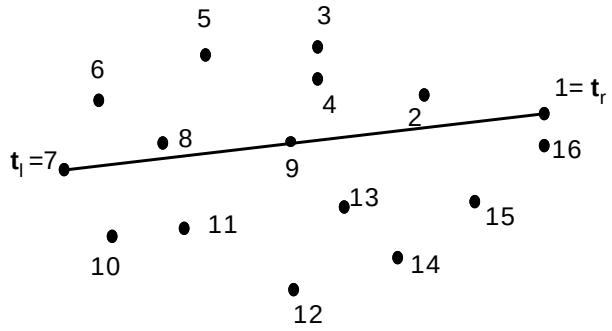
Gilt dies nicht, dann lösche $\mathbf{t}_{i_2}$ *aus* π. *Ist die Bedingung erfüllt, so hat man einen Polygonzug* π *gefunden, der die konvexe Hülle von* $\mathcal{S}$ *beschreibt und entgegen dem Uhrzeigersinn angeordnet ist.*

Beispiel 3.18
Gegeben sei die Menge $\mathcal{S} = \{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, ..., \mathbf{t}_{16}\}$, wie im Bild durch die Indices angegeben. Es ergeben sich in Schritt 1 und 2

$$t_l = t_7 \text{ und } t_r = t_1$$

sowie

$$\mathcal{S}_o = \{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{t}_4, \mathbf{t}_5, \mathbf{t}_6, \mathbf{t}_7, \mathbf{t}_8\} \text{ und } \mathcal{S}_u = \{\mathbf{t}_1, \mathbf{t}_7, \mathbf{t}_{10}, \mathbf{t}_{11}, \mathbf{t}_{12}, \mathbf{t}_{13}, \mathbf{t}_{14}, \mathbf{t}_{15}, \mathbf{t}_{16}\}.$$



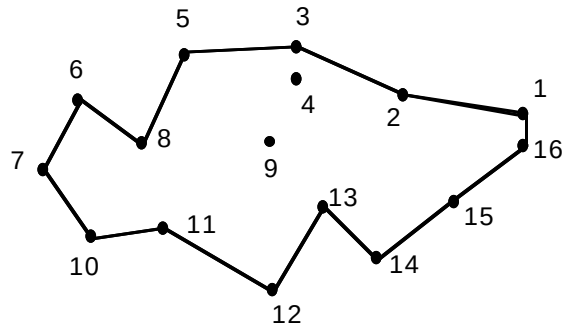
Schritt 3 ermittelt zunächst

$$\pi_o = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{t}_5, \mathbf{t}_8, \mathbf{t}_6, \mathbf{t}_7) \text{ sowie } \pi_u = (\mathbf{t}_7, \mathbf{t}_{10}, \mathbf{t}_{11}, \mathbf{t}_{12}, \mathbf{t}_{13}, \mathbf{t}_{14}, \mathbf{t}_{15}, \mathbf{t}_{16}, \mathbf{t}_1)$$

bevor Schritt 4 zu

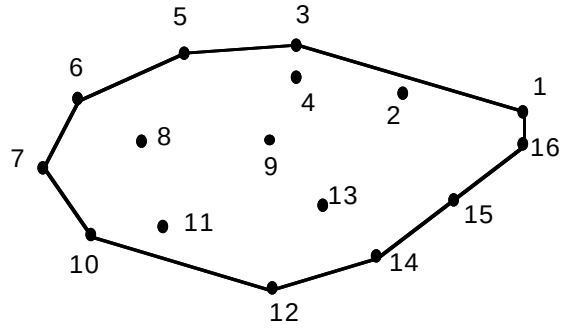
$$\pi = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{t}_5, \mathbf{t}_8, \mathbf{t}_6, \mathbf{t}_7, \mathbf{t}_{10}, \mathbf{t}_{11}, \mathbf{t}_{12}, \mathbf{t}_{13}, \mathbf{t}_{14}, \mathbf{t}_{15}, \mathbf{t}_{16}, \mathbf{t}_1)$$

führt.



Schritt 5 bringt dann letztendlich das Ergebnis:

$$\pi = (\mathbf{t}_1, \mathbf{t}_3, \mathbf{t}_5, \mathbf{t}_6, \mathbf{t}_7, \mathbf{t}_{10}, \mathbf{t}_{12}, \mathbf{t}_{14}, \mathbf{t}_{15}, \mathbf{t}_{16}, \mathbf{t}_1).$$



3.2.2 Delaunay-Triangulierung

Gegeben ist eine Menge $\mathcal{S} = \{\mathbf{t}_1, ..., \mathbf{t}_n\} \subset \mathbb{R}^2$, und gesucht ist eine Delaunay-Triangulierung $\mathcal{T}(\mathcal{S})$. Eine umfangreiche Liste solcher Konstruktionsalgorithmen findet sich in [Shu87]. Hier soll nun ein Algorithmus von Cline und Renka [ClR84] vorgestellt werden, der in mehrere Teilalgorithmen unterteilt ist. In dem genannten Artikel finden sich auch Aussagen zu möglichen Datenstrukturen, um möglichst speichereffizient zu arbeiten, was uns hier aber weniger interessiert.

3.2.2.1 Initialtriangulierung

Normalerweise erhält man die Initialtriangulierung, indem man die ersten drei Punkte als erstes Element der Triangulierung verwendet. Das ist aber nur möglich, falls diese nicht kollinear sind.

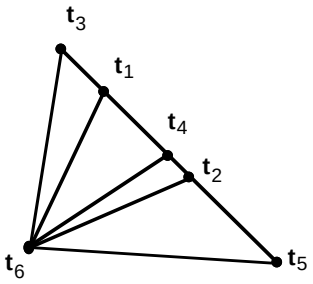
Algorithmus 3.19 (Initialtriangulierung)
Gegeben ist eine Menge $\mathcal{S} = \{\mathbf{t}_1, ..., \mathbf{t}_n\} \subset \mathbb{R}^2$ paarweise verschiedener Punkte. Gesucht ist ein minimales j ($j \leq n$) und eine Triangulierung $\mathcal{T}(\mathcal{S}_j)$, so daß eine Triangulierung $\mathcal{T}(\mathcal{S}_j)$ mit $\mathcal{S}_j = \{\mathbf{t}_1, ..., \mathbf{t}_j\}$ existiert.

Schritt 1: *Bestimme $j \geq 3$, so daß $\{\mathbf{t}_1, ..., \mathbf{t}_{j-1}\}$ kollinear liegen, die Punkte der Menge $\{\mathbf{t}_1, ..., \mathbf{t}_j\}$ jedoch nicht.*

Schritt 2: *Dann liegen $\{\mathbf{t}_1, ..., \mathbf{t}_{j-1}\}$ auf einer Geraden. Je zwei benachbarte Punkte davon bilden mit $\mathbf{t}_j$ ein Element der Initialtriangulierung. Somit erhalten wir*

$$\mathcal{T}(\{\mathbf{t}_1, ..., \mathbf{t}_j\}) = \{T_1, ..., T_{j-2}\}.$$

Eine Initialtriangulierung könnte beispielsweise wie folgt aussehen.

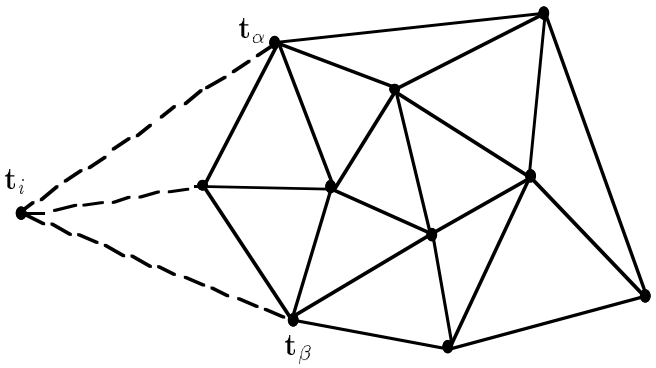


Es liegt mit der Initialtriangulierung offensichtlich immer eine Delaunay-Triangulierung vor. Im folgenden wird beschrieben, wie die Punkte $\mathbf{t}_{j+1}, ..., \mathbf{t}_n$ sukzessive hinzugefügt werden können. Dazu wird zunächst der einzufügende Punkt $\mathbf{t}_i$ lokalisiert und die Triangulierung entsprechend erweitert. Danach wird sie optimiert, so daß erneut eine Delaunay-Triangulierung entsteht.

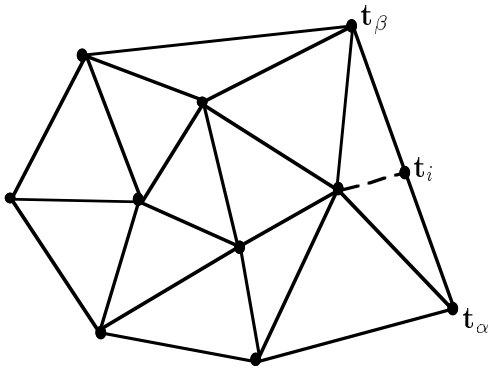
3.2.2.2 Punktlokalisierung und Erweiterung

Seien eine (Delaunay-) Triangulierung $\mathcal{T}(\mathcal{S}_{i-1})$ von $\mathcal{S}_{i-1} = \{\mathbf{t}_1, ..., \mathbf{t}_{i-1}\} \subset \mathbb{R}^2$ und ein Punkt $\mathbf{t}_i \in \mathbb{R}^2$ gegeben. Der folgende Algorithmus 3.20 liefert dann drei Indizes $\alpha, \beta \in \{1, ..., i \Leftrightarrow 1\}$ und $\gamma \in \{0, ..., i \Leftrightarrow 1\}$, wobei $\gamma = 0$ ist, wenn der zu lokalisierende Punkt $\mathbf{t}_i$ auf dem Rand oder außerhalb des bereits triangulierten Bereichs liegt. Anhand dieser drei Indizes kann man leicht die Lage des zu lokalisierenden Punktes bestimmen. Es können vier Fälle eintreten, die nun erläutert werden:

1. $\mathbf{t}_i \notin [\{\mathbf{t}_1, ..., \mathbf{t}_{i-1}\}]$: Der Algorithmus liefert dann mit α und β die Indizes zweier Punkte auf dem Rand des triangulierten Bereichs. Neue Elemente der Triangulierung entstehen dann, indem wir genau die Punkte aus $\mathcal{S}_{i-1}$ zwischen $\mathbf{t}_\alpha$ und $\mathbf{t}_\beta$ auf dem Rand (entgegen dem Uhrzeigersinn durchlaufen) mit $\mathbf{t}_i$ verbinden. Es werden also keine Elemente der alten Triangulierung gelöscht, sondern nur die entsprechenden neuen hinzugefügt.

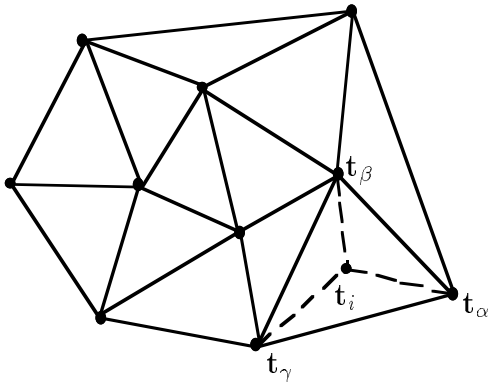


2. $\mathbf{t}_i \in \partial[\{\mathbf{t}_1, ..., \mathbf{t}_{i-1}\}]$: Der Algorithmus liefert dann mit α und β die Indizes der Punkte, die die Kante der Triangulierung definieren, auf der $\mathbf{t}_i$ liegt. Das (eindeutig bestimmte) Element der Triangulierung, das beide Punkte enthält, wird gelöscht, und die beiden neuentstandenen werden hinzugefügt (vgl. Skizze).

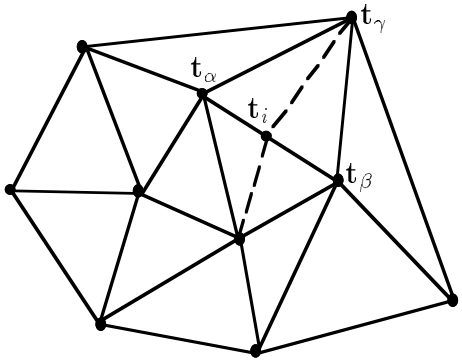


3. $\mathbf{t}_i \in [\{\mathbf{t}_1, ..., \mathbf{t}_{i-1}\}] \Leftrightarrow \partial[\{\mathbf{t}_1, ..., \mathbf{t}_{i-1}\}]$: Der Algorithmus liefert dann mit α, β und γ die Indizes dreier Punkte, die ein Element der Triangulierung definieren in dem $\mathbf{t}_i$ liegt. Hierbei können zwei Fälle auftreten:

   (a) $\mathbf{t}_i \in [\{\mathbf{t}_\alpha, \mathbf{t}_\beta, \mathbf{t}_\gamma\}] \Leftrightarrow \partial[\{\mathbf{t}_\alpha, \mathbf{t}_\beta, \mathbf{t}_\gamma\}]$: Dann wird das Element $\{\mathbf{t}_\alpha, \mathbf{t}_\beta, \mathbf{t}_\gamma\}$ gelöscht, und die drei neuentstandenen werden eingefügt.



   (b) $\mathbf{t}_i \in \partial[\{\mathbf{t}_\alpha, \mathbf{t}_\beta, \mathbf{t}_\gamma\}]$: Dann werden die beiden Elemente aus der Triangulierung gelöscht, deren konvexe Hüllen $\mathbf{t}_i$ enthalten, und die vier neuentstandenen werden eingefügt.



Kommen wir nun zur Beschreibung des Algorithmus zur Bestimmung der Indizes α, β und γ. In den Schritten 1 bis 5 wird zunächst initialisiert. In den Schritten 6 und 7 wird ein Kegel mit Scheitelpunkt $\mathbf{t}_\delta$ bestimmt, der $\mathbf{t}_i$ enthält. Als nächstes wird durch die Schritte 8 bis 13 die Position des Punktes $\mathbf{t}_i$ bestimmt und durch die Indizes – wie zuvor erklärt – beschrieben. Als letztes werden, falls $\mathbf{t}_i \notin [\mathcal{S}_{i-1}]$ gilt, in den Schritten 14 bis 18 die Punkte $\mathbf{t}_\alpha$ und $\mathbf{t}_\beta$ so bestimmt, daß auf dem Rand der Triangulierung zwischen ihnen genau die Punkte liegen, die mit $\mathbf{t}_i$ verbunden werden müssen (vgl. auch nochmal die Skizze zuvor unter 1.).

Algorithmus 3.20 (Punktlokalisierung)
Gegeben ist eine (Delaunay-) Triangulierung $\mathcal{T}(\mathcal{S}_{i-1})$ von $\mathcal{S}_{i-1} = \{\mathbf{t}_1, ..., \mathbf{t}_{i-1}\} \subset \mathbb{R}^2$ und ein Punkt $\mathbf{t}_i \in \mathbb{R}^2$. Gesucht sind drei Indices α, β und γ, wie zuvor beschrieben.

Schritt 1: *Sei $\delta \in \{1, ..., i \Leftrightarrow 1\}$ der Index eines beliebigen Punktes, z.B.* $\delta := 1$.

Schritt 2: *Bestimme die Indizes ε und λ des ersten und des letzten Nachbarn von $\mathbf{t}_\delta$ (vgl. Definition 3.2).*

Schritt 3: *Ist $\mathbf{t}_\delta$ ein Punkt auf dem Rand der Triangulierung und gilt*

$$\mathbf{t}_i \text{ liegt echt rechts von } \mathbf{t}_\delta \rightarrow \mathbf{t}_\varepsilon,$$

so setze

$$\alpha := \delta \text{ und } \beta := \varepsilon$$

und gehe zu **Schritt 14**.

Ist $\mathbf{t}_\delta$ ein Punkt auf dem Rand der Triangulierung und gilt

$$\mathbf{t}_i \text{ liegt echt rechts von } \mathbf{t}_\lambda \rightarrow \mathbf{t}_\delta,$$

so setze

$$\alpha := \lambda \text{ und } \beta := \delta$$

und gehe zu **Schritt 16**.

Schritt 4: *Suche unter den Nachbarn von* $\mathbf{t}_\delta$ *nach einem Punkt* $\mathbf{t}_k$*, für den gilt:*

$$\mathbf{t}_i \text{ liegt echt rechts von } \mathbf{t}_\delta \rightarrow \mathbf{t}_k.$$

Existiert ein solcher Punkt, so setze

$$\beta := k$$

und gehe zu **Schritt 6**.

Schritt 5: *Setze*

$$\delta := k,$$

wobei k *der Index des Punktes* $\mathbf{t}_k$ *ist, der mit* $\mathbf{t}_\lambda$ *und* $\mathbf{t}_\delta$ *ein Dreieck* $[\mathbf{t}_k, \mathbf{t}_\lambda, \mathbf{t}_\delta]$ *der Triangulierung bildet. Gehe zu* **Schritt 2**.

Schritt 6: *Setze*

$$\alpha := k,$$

wobei k *der Index des Punktes* $\mathbf{t}_k$ *ist, der mit* $\mathbf{t}_\beta$ *und* $\mathbf{t}_\delta$ *ein Dreieck* $[\mathbf{t}_k, \mathbf{t}_\beta, \mathbf{t}_\delta]$ *der Triangulierung bildet.*

Schritt 7: *Gilt*

$$\mathbf{t}_i \text{ liegt links von } \mathbf{t}_\delta \rightarrow \mathbf{t}_\alpha,$$

so setze

$$\gamma := \delta$$

und gehe zu **Schritt 8**. *Andernfalls setze*

$$\beta := \alpha$$

und gehe zu **Schritt 6**.

Schritt 8: *Gilt*

$$\mathbf{t}_i \text{ liegt links von } \mathbf{t}_\alpha \rightarrow \mathbf{t}_\beta,$$

so gehe zu **Schritt 12**.

Schritt 9: *Ist* $\mathbf{t}_\alpha \Leftrightarrow \mathbf{t}_\beta$ *eine Außenkante der Triangulierung, so gehe zu* **Schritt 14**.

Schritt 10: *Setze*

$$\kappa := k,$$

wobei k *der Index des Punktes* $\mathbf{t}_k$ *ist, der mit* $\mathbf{t}_\beta$ *und* $\mathbf{t}_\alpha$ *ein Dreieck* $[\mathbf{t}_k, \mathbf{t}_\beta, \mathbf{t}_\alpha]$ *der Triangulierung bildet.*

Schritt 11: *Gilt*

$$\mathbf{t}_i \textit{ liegt links von } \mathbf{t}_\delta \rightarrow \mathbf{t}_\kappa,$$

so setze

$$\gamma := \alpha \textit{ und } \alpha := \kappa.$$

Andernfalls setze

$$\gamma := \beta \textit{ und } \beta := \kappa.$$

Gehe zu **Schritt 8**.

Schritt 12: *Ist* $\mathbf{t}_\gamma \Leftrightarrow \mathbf{t}_\alpha$ *eine Außenkante der Triangulierung und gilt weiter*

$$\mathbf{t}_i \textit{ liegt links von } \mathbf{t}_\alpha \rightarrow \mathbf{t}_\gamma,$$

so setze

$$\beta := \alpha, \alpha := \gamma, \gamma := 0$$

und beende den Algorithmus.

Schritt 13: *Ist* $\mathbf{t}_\alpha \Leftrightarrow \mathbf{t}_\beta$ *eine Außenkante der Triangulierung und gilt weiter*

$$\mathbf{t}_i \textit{ liegt links von } \mathbf{t}_\beta \rightarrow \mathbf{t}_\alpha,$$

so setze

$$\gamma := 0.$$

Beende den Algorithmus.

Schritt 14: *Setze*

$$\varepsilon := k,$$

wobei k *der Index des ersten Nachbarn von* $\mathbf{t}_\beta$ *ist.*

Schritt 15: *Gilt*

$$\mathbf{t}_i \textit{ liegt echt rechts von } \mathbf{t}_\beta \rightarrow \mathbf{t}_\varepsilon,$$

so setze

$$\beta := \varepsilon$$

und gehe zu **Schritt 14**.

Schritt 16: *Setze*

$$\lambda := k,$$

wobei k *der Index des letzten Nachbarn von* $\mathbf{t}_\alpha$ *ist.*

Schritt 17: *Gilt*

$$\mathbf{t}_i \textit{ liegt echt rechts von } \mathbf{t}_\lambda \rightarrow \mathbf{t}_\alpha,$$

so setze

$$\alpha := \lambda$$

und gehe zu **Schritt 16**.

Schritt 18: *Setze*

$$\gamma := 0$$

und beende den Algorithmus.

Mit Hilfe des Ergebnisses, das der Algorithmus liefert, führt man dann nur noch die Ersetzungen in der Triangulierung durch, wie es zuvor ausführlich beschrieben wurde.

3.2.2.3 Optimierung der Triangulierung

Die Triangulierung, die nun vorliegt, ist i.allg. keine Delaunay-Triangulierung mehr. Um dies wieder zu erreichen, führen wir noch einen Optimierungsalgorithmus durch, der um den neu eingefügten Punkt herum prüft, ob Diagonalentausche durchgeführt werden können. Dazu werden Vierecke bestimmt, an denen $\mathbf{t}_i$ als Ecke beteiligt ist. Entstehen durch einen positiven Tauschtest und den anschließenden Diagonalentausch neue solcher Vierecke, so werden auch diese auf einen Diagonalentausch hin getestet. Man vergleiche auch das Beispiel im Anschluß an den Algorithmus.

Algorithmus 3.21 (Optimierungsalgorithmus)
Gegeben ist eine Triangulierung $\mathcal{T}'(\mathcal{S}_i)$, die entstanden ist, indem wir mit dem zuvor beschriebenen Algorithmus einen Punkt $\mathbf{t}_i \in \mathbb{R}^2$ in die Delaunay-Triangulierung $\mathcal{T}(\mathcal{S}_{i-1})$ eingefügt haben. Gesucht ist eine Delaunay-Triangulierung $\mathcal{T}(\mathcal{S}_i)$.

Schritt 1: *Setze*

$$\alpha := k \text{ und } \varepsilon := k,$$

wobei k der Index des ersten Nachbarn von $\mathbf{t}_i$ ist. ($\mathbf{t}_\beta$ aus dem Ergebnis des vorangegangenen Algorithmus kann immer als erster Nachbar verwendet werden.)

Schritt 2: *Setze*

$$\beta := k,$$

wobei k der Index des Punktes $\mathbf{t}_k$ ist, der mit $\mathbf{t}_i$ und $\mathbf{t}_\alpha$ ein Dreieck $[\mathbf{t}_k, \mathbf{t}_i, \mathbf{t}_\alpha]$ der Triangulierung bildet.

Schritt 3: *Ist $\mathbf{t}_\alpha \Leftrightarrow \mathbf{t}_\beta$ eine Außenkante der Triangulierung, so gehe zu* **Schritt 6**.

Schritt 4: *Setze*

$$\gamma := k,$$

wobei k der Index des Punktes $\mathbf{t}_k$ ist, der mit $\mathbf{t}_\beta$ und $\mathbf{t}_\alpha$ ein Dreieck $[\mathbf{t}_k, \mathbf{t}_\beta, \mathbf{t}_\alpha]$ der Triangulierung bildet.

Schritt 5: *Wende einen Tauschtest auf* $\mathbf{t}_\alpha \Leftrightarrow \mathbf{t}_\beta$ *bzgl. des Max-Min-Winkelkriteriums an. Ist der Test positiv, so führe den Diagonalentausch durch, setze*

$$\beta := \gamma$$

und gehe zu **Schritt 3**.

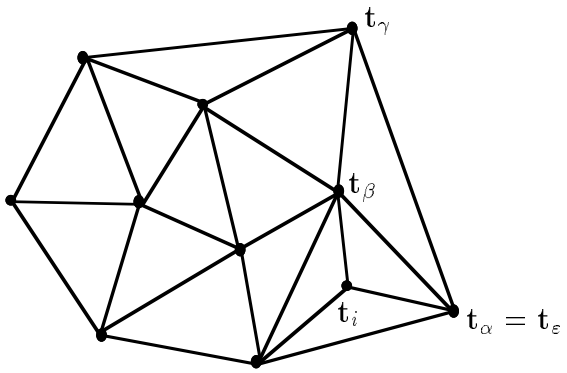
Schritt 6: *Setze*

$$\alpha := \beta.$$

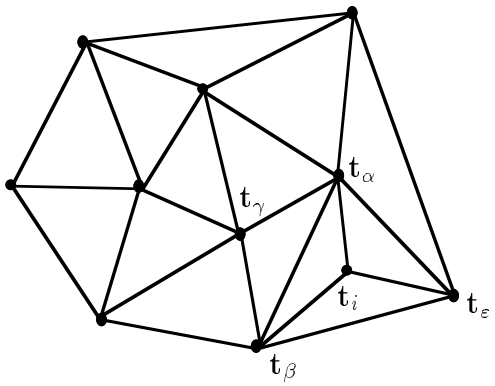
Schritt 7: *Gilt* $\alpha = \varepsilon$ *oder ist* $\mathbf{t}_\alpha \rightarrow \mathbf{t}_i$ *eine Außenkante, so beende den Algorithmus. Andernfalls gehe zu* **Schritt 2**.

An folgendem Beispiel soll die Funktionsweise des Optimierungsalgorithmus illustriert werden.

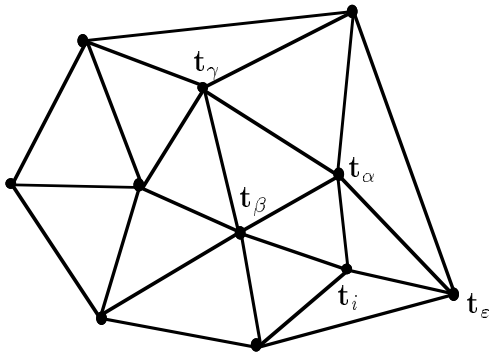
Beispiel 3.22 (Optimierungsalgorithmus)
Der Punkt $\mathbf{t}_i$ wurde eingefügt, und nun steht die Optimierung an.



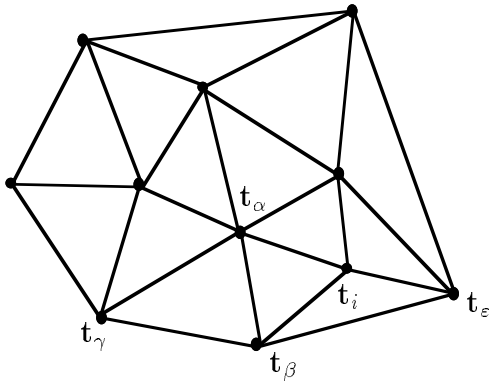
(Nach den Schritten 1, 2, 3 und 4.)



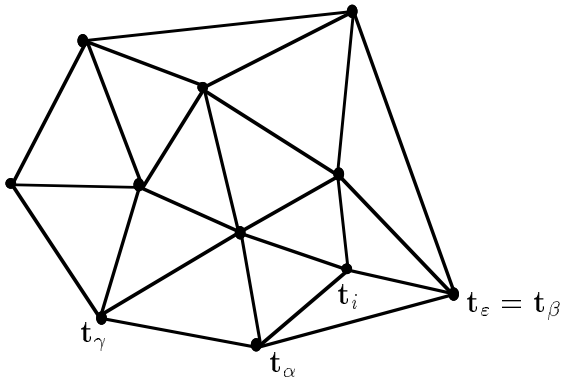
(Nach den Schritten 5, 6, 7, 2, 3 und 4.)



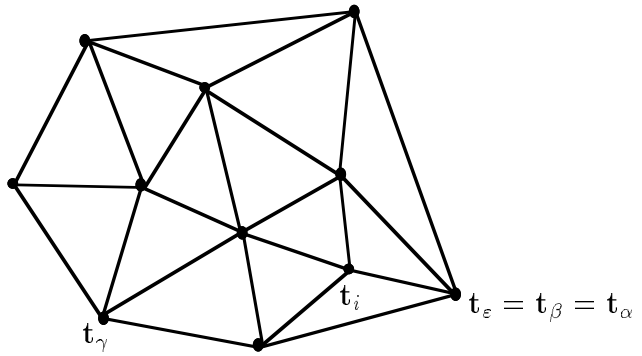
(Nach den Schritten 5, 3 und 4.)



(Nach den Schritten 5, 6, 7, 2, 3 und 4.)



(Nach den Schritten 5, 6, 7 und 2.)



(Nach den Schritten 3, 6 und 7 wird gestoppt.)

3.2.3 Clustering-Algorithmus

In Algorithmus A aus dem folgenden Abschnitt 3.3 wird als Hilfsmittel zur Bestimmung eines Teils der Punkte, aus denen die Triangulierungen erzeugt werden, ein Clustering-Algorithmus benutzt.

Der im folgenden beschriebene Algorithmus stammt von T. Schreiber [Shr91]. Es ist ein einfacher, schneller Clustering-Algorithmus. Clustering erklärt Schreiber als *Gruppieren gleichartiger Objekte unter Optimierung einer Kriteriums-Funktion.* In der ursprünglichen Form *Clustering is the grouping of similar objects* stammt die Beschreibung von Hartigan [Har75]. Der Algorithmus ist allgemein für den $\mathbb{R}^d$ gehalten und bedient sich multidimensionaler Voronoi-Regionen. Bei gegebenen Voronoi-Center-Punkten $\mathcal{S} = \{\mathbf{t}_1, ..., \mathbf{t}_n\} \subset \mathbb{R}^d$ ergeben sich die zugehörigen Voronoi-Regionen $D(\mathbf{t}_j, \mathcal{S})$ durch

$$D_j := D(\mathbf{t}_j, \mathcal{S}) = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x} \Leftrightarrow \mathbf{t}_j\|_2 < \|\mathbf{x} \Leftrightarrow \mathbf{t}_k\|_2, j \neq k\} \quad \forall \mathbf{t}_j \in \mathcal{S}.$$

Algorithmus 3.23 (Clustering-Algorithmus)
Gegeben seien die Punkte $\mathbf{d}_i \in \mathbb{R}^d$, $i = 1, ..., l$ *und zugehörige Gewichte* $w_i \in \mathbb{R}$ *mit* $w_i > 0$. *Gesucht sind Cluster-Punkte* $\mathbf{t}_i \in \mathbb{R}^d$, $i = 1, ..., n \leq l$.

Schritt 1: *Der erste Cluster-Punkt* $\mathbf{t}_1$ *ergibt sich als gewichtetes Mittel*

$$\left(\sum_{i=1}^{l} w_i\right)^{-1} \cdot \sum_{i=1}^{l} w_i \mathbf{d}_i$$

aller Daten. Die zugehörige Voronoi-Region D_1 *ist der gesamte Raum* $\mathbb{R}^d$.

Schritt 2: *Unterteile die Menge der Daten der Region D_e mit dem größten Fehler F_e in zwei Mengen. Die Fehlerfunktion sei gegeben durch:*

$$F(D_j) = F_j := \left(\sum_{i \in I_j} w_i\right)^{-1} \cdot \sum_{i \in I_j} w_i \| \mathbf{t}_j \Leftrightarrow \mathbf{d}_i \|^2,$$

wobei I_j eine Indexmenge ist, für die gilt $i \in I_j \Leftrightarrow \mathbf{d}_i \in D_j$.

Zur Unterteilung bestimme die neuen Indexmengen und ihre Mittel-Punkte:

1. *Berechne zunächst die Koordinaten-Achse $k \in \{1, ..., d\}$ mit der größten projizierten Abweichung:*

$$\sum_{i \in I_e} w_i (\mathbf{d}_i^k \Leftrightarrow \mathbf{t}_e^k) = \max_{m=1,...,d} \left\{ \sum_{i \in I_e} w_i (\mathbf{d}_i^m \Leftrightarrow \mathbf{t}_e^m) \right\},$$

   wobei $\mathbf{x}^m$ die m-te Komponente des Vektors $\mathbf{x}$ bezeichnet.

2. *Teile alle Punkte $\mathbf{d}_i$ ($i \in I_e$) mit einer Hyperebene durch den in D_e liegenden Cluster-Punkt $\mathbf{t}_e$ senkrecht zur k-ten Koordinatenachse. Bestimme die neuen Indexmengen und ihre zugehörigen Cluster-Punkte wie folgt:*

$$I_{e_1} = \{i : \mathbf{d}_i^k \leq \mathbf{t}_e^k, i \in I_e\} \quad \text{und} \quad I_{e_2} = \{i : \mathbf{d}_i^k > \mathbf{t}_e^k, i \in I_e\}$$

$$\mathbf{t}_\alpha = \left(\sum_{i \in I_{e_1}} w_i\right)^{-1} \cdot \sum_{i \in I_{e_1}} w_i \mathbf{d}_i \quad \text{und} \quad \mathbf{t}_\beta = \left(\sum_{i \in I_{e_2}} w_i\right)^{-1} \cdot \sum_{i \in I_{e_2}} w_i \mathbf{d}_i$$

Schritt 3: *Aktualisiere das Voronoi-Diagramm:*

Ersetze den Cluster-Punkt $\mathbf{t}_e$ durch die beiden neuen Cluster-Punkte $\mathbf{t}_\alpha$ und $\mathbf{t}_\beta$ und aktualisiere die Indexmengen I_j derart, daß gilt:

$$i \in I_j \Leftrightarrow \mathbf{d}_i \in D_j.$$

Schritt 4: *Für alle veränderten Regionen D_j:*

Ersetze $\mathbf{t}_j$ durch das gewichtete Mittel $\left(\sum_{i \in I_j} w_i\right)^{-1} \cdot \sum_{i \in I_j} w_i \mathbf{d}_i$.

Schritt 5: *Wiederhole Schritt 2 - Schritt 4 sooft wie gewünscht (z.B. bei n Cluster-Punkten noch n⇔2 mal oder in Abhängigkeit von der Fehlerfunktion).*

Bemerkung 3.24 (zu Algorithmus 3.23)

1. Die Veränderungen, die in Schritt 4 angesprochen werden, ergeben sich aus veränderten Voronoi-Regionen und den daraus eventuell resultierenden neuen Zuordnungen der Punkte unter Schritt 3.

2. T. A. Foley, S. Dayanand und D. Zeckzer verwenden in [FDZ95] ebenfalls diesen Algorithmus zur Ermittlung von Punkten, aus denen eine Triangulierung generiert wird, die eine Menge $\mathcal{D}$ an Daten überdeckt. Wie in der Einleitung bereits erwähnt, erfüllt diese Triangulierung allerdings nicht die geforderten Bedingungen (A2) und (A3).

3.3 Ein neuer Triangulierungsalgorithmus

In diesem Abschnitt wird ein neuer Triangulierungsalgorithmus vorgestellt, der das Problem 1.1 (bis auf zwei Sonderfälle, vgl. hierzu Algorithmus C) löst. Er besteht aus drei Teilalgorithmen: Algorithmus A, B und C. Algorithmus A bestimmt eine Punktmenge $\mathcal{S} = \{\mathbf{t}_1, ..., \mathbf{t}_l\} \subset \mathbb{R}^2$ und eine zugehörige Triangulierung $\mathcal{T}(\mathcal{S})$, bei der jedes Dreieck mindestens d Daten beinhaltet. Dazu werden zunächst Außenpunkte, die den Rand der Triangulierung festlegen, bestimmt. Anschließend ergänzen wir die Menge der Außenpunkte $\mathcal{S}_A$ durch Innenpunkte, bis wir mit dem dafür angewendeten Verfahren keine Punkte mehr hinzufügen können, ohne die Minimal-Anzahl d an Daten in den Dreiecken zu verletzen. Algorithmus B fügt dann mit einem anderen Verfahren sukzessive der Menge $\mathcal{S}$ Punkte $\mathbf{t}$ hinzu, so daß die Triangulierung $\mathcal{T}(\mathcal{S} \cup \{\mathbf{t}\})$ immer noch (wie zuvor $\mathcal{T}(\mathcal{S})$) in jedem Dreieck genügend Daten hat. Auf diese Weise wird die Triangulierung verfeinert, und wir nähern uns dem Ziel, eine Maximal-Anzahl von Daten in keinem Dreieck zu überschreiten. Kann Algorithmus B keine Punkte mehr hinzufügen, so beginnt Algorithmus C (wiederum mit einem anderen Verfahren), neue Punkte zu berechnen, und die Triangulierung somit zu verfeinern. Algorithmus C setzt in den Dreiecken mit $3d$ (oder mehr) Daten einen Punkt $\mathbf{t}$ so ein, daß die Dreiecke in je drei Dreiecke unterteilt werden, die ihrerseits je d Daten beinhalten. Auf diese Weise garantiert Algorithmus C eine maximale Anzahl von $3d \Leftrightarrow 1$ Daten in jedem Dreieck der Triangulierung. Eine Staffelung in drei Algorithmen, die auf verschiedene Art und Weise Punkte bestimmen, ist insofern sinnvoll, da jeder Algorithmus zu dem bestimmten Zeitpunkt Vorteile gegenüber den anderen hat. Zum Beispiel könnte Algorithmus B ganz entfallen. Eine Benutzung liefert aber in Bezug auf die Gestalt der Triangulierung viel bessere Ergebnisse, da Algorithmus C z.B. keine Punkte auf dem Rand der Triangulierung zufügen kann (vgl. auch die Beispiele in Abschnitt 3.5). Andererseits liefert Algorithmus B so gute Ergebnisse, daß die Triangulierung sehr oft gar

keine Dreiecke mehr besitzt, die $3d$ Daten beinhalten. Algorithmus C kommt also oft gar nicht zum Einsatz. Wollen wir allerdings eine maximale Anzahl an Daten in den Dreiecken garantieren, so können wir auf ihn nicht verzichten. Zur Veranschaulichung von Algorithmus A bzw. Algorithmus B finden sich Beispiele in Abschnitt 3.5.

3.3.1 Algorithmus A

Algorithmus A hat den Sinn, eine Triangulierung zu finden, die eine minimale Anzahl an Daten in den Dreiecken garantiert. Zusätzlich soll

- der triangulierte Bereich eine *kleine* Obermenge der konvexen Hülle der Daten bilden, und
- eine *schöne* Triangulierung berechnet werden.

Mit einer kleinen Obermenge ist gemeint, daß der triangulierte Bereich nur eine etwas größere Fläche einnimmt als die konvexe Hülle der Daten $\mathcal{D}$, in denen die Werte zu approximieren sind. Eine schöne Triangulierung ist bezogen auf die Tatsache, daß lange spitze Dreiecke vermieden werden sollen. Deshalb werden ausschließlich Delaunay-Triangulierungen verwandt.

Zu Beginn von Algorithmus A wird zunächst eine Punktmenge $\mathcal{S}_A$ bestimmt, die ein konvexes Polygon beschreibt. Dieses Polygon soll alle Daten enthalten. Daraufhin wird die Delaunay-Triangulierung $\mathcal{T}(\mathcal{S}_A)$ berechnet und geprüft, ob kein Dreieck die Minimal-Anzahl d an Daten, die es beinhaltet, unterschreitet. In einer Schleife werden sukzessive Cluster-Punkte berechnet. Diese Cluster-Punkte ergeben zusammen mit $\mathcal{S}_A$ die Menge $\mathcal{S}$. Wiederum wird eine Delaunay-Triangulierung $\mathcal{T}(\mathcal{S})$ berechnet und die Anzahl an Daten in den Dreiecken überprüft. Ergibt sich dreimal hintereinander, daß eine überprüfte Triangulierung in einem Dreieck zu wenig Daten beinhaltet, wird die Schleife abgebrochen, da wir davon ausgehen können, daß wir durch Hinzufügen weiterer Cluster-Punkte nicht mehr zu einer passenden Triangulierung gelangen werden. Die letzte Triangulierung, die in jedem Dreieck genügend Daten hatte, stellt das Ergebnis des Algorithmus A dar. Hatte noch keine Triangulierung in jedem Dreieck genügend Daten, wird die Anzahl der Außenpunkte reduziert, und wir springen erneut in die Schleife, um Cluster-Punkte zu berechnen.

Algorithmus 3.25 (Algorithmus A)
Gegeben ist eine Menge von Daten $\mathcal{D} = \{\mathbf{d}_1, ..., \mathbf{d}_l\} \subset \mathbb{R}^2$*, wobei nicht alle Daten aus* $\mathcal{D}$ *kollinear liegen. Gesucht ist eine Menge* $\mathcal{S} = \{\mathbf{t}_1, ..., \mathbf{t}_n\} \subset \mathbb{R}^2$ *und eine Triangulierung* $\mathcal{T}(\mathcal{S})$*, so daß die Triangulierung mit jedem Dreieck mindestens* $d \in \mathbb{N}$ *Daten* $(d \leq l)$ *aus* $\mathcal{D}$ *überdeckt:*

$$|[T_i] \cap \mathcal{D}| \geq d \quad \forall T_i \in \mathcal{T}.$$

Schritt 1: *Bestimme eine Punktmenge* $\mathcal{S}_A = \{\mathbf{t}_1, ..., \mathbf{t}_{n_1}\}$ *von Außenpunkten mit*

$$[\mathcal{S}_A] \cap \mathcal{D} = \mathcal{D},$$

die ein konvexes Polygon beschreibt (vgl. Algorithmus 3.27).

Schritt 2: *Bestimme die Delaunay-Triangulierung* $\mathcal{T}(\mathcal{S}_A)$ *von* $\mathcal{S}_A$ *und ordne jedem Element der Triangulierung* $T_i \in \mathcal{T}(\mathcal{S}_A) = \{T_1, ..., T_m\}$ *die Daten* $\mathbf{d}_i \in \mathcal{D} \cap [T_i]$ *zu, die es beinhaltet. Wir erhalten den Vektor* $\{\mathcal{D}_1, ..., \mathcal{D}_m\}$, *der die Datenmengen der einzelnen Dreiecke von* $\mathcal{T}(\mathcal{S}_A)$ *beinhaltet.*

Schritt 3: *Gilt*

$$|\mathcal{D}_i| \geq d \quad \forall i = 1, ..., m,$$

gehe zu **Schritt** *4. Ansonsten gehe zu* **Schritt 5**.

Schritt 4: *Setze*

$$tria := \mathcal{T}(\mathcal{S}_A), \mathcal{S}_I := \emptyset \text{ und } durchlauf := 1.$$

gehe zu **Schritt 6**. *($\mathcal{S}_I$ sind die Innenpunkte der Triangulierung.)*

Schritt 5: *Setze*

$$tria := \emptyset, \mathcal{S}_I := \emptyset \text{ und } durchlauf := 1.$$

Schritt 6: *Gilt*

$$durchlauf > 3,$$

gehe zu **Schritt 11**.

Schritt 7: *Erhöhe durch einen Clustering-Algorithmus (vgl. Algorithmus 3.30) die Anzahl der Innenpunkte der Triangulierung* $|\mathcal{S}_I|$ *um eins. Berechne die Delaunay-Triangulierung* $\mathcal{T}(\mathcal{S}) = \{T_1, ..., T_m\}$ *mit* $\mathcal{S} := \mathcal{S}_A \cup \mathcal{S}_I$ *und ordne jedem Element der Triangulierung die Daten zu, die es beinhaltet. (wie in* **Schritt 2**)

Schritt 8: *Gilt*

$$|\mathcal{D}_i| \geq d \quad \forall i = 1, ..., m,$$

gehe zu **Schritt 9**. *Ansonsten gehe zu* **Schritt 10**.

Schritt 9: *Setze*

$$tria := \mathcal{T}(\mathcal{S}) \text{ und } durchlauf := 1.$$

Gehe zu **Schritt 6**.

Schritt 10: *Setze*

$$durchlauf := durchlauf + 1.$$

Gehe zu **Schritt 6**.

Schritt 11: *Gilt*

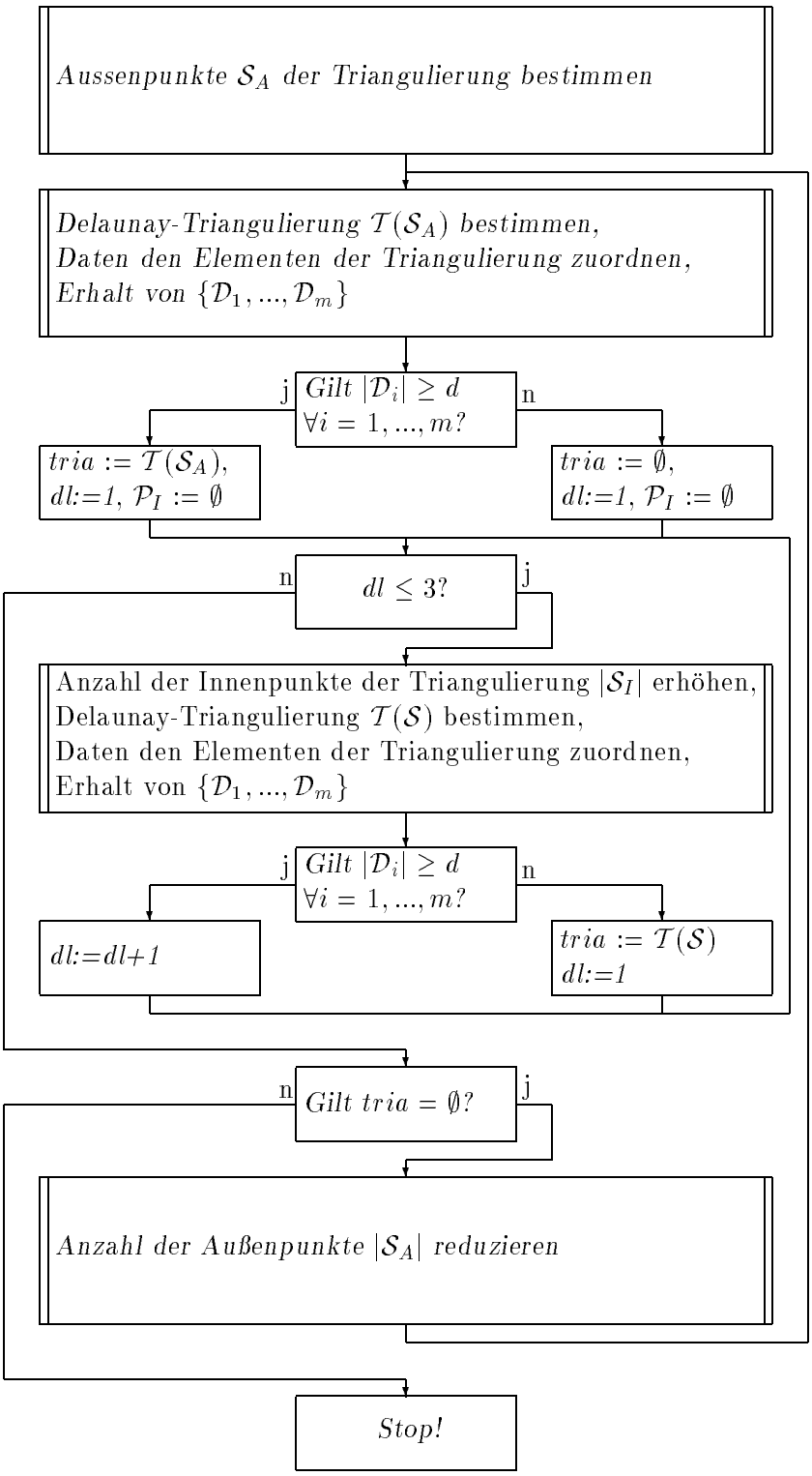
$$tria = \emptyset,$$

gehe zu **Schritt 12**. *Ansonsten beende den Algorithmus.*

Schritt 12: *Reduziere die Anzahl der Außenpunkte um eins, indem Algorithmus 3.31 angewandt wird. Wir erhalten die neue Punktmenge* $\mathcal{S}_A$. *Gehe zu* **Schritt 2**.

Bemerkung 3.26
Die Variable *durchlauf* steht für die Anzahl der aufeinanderfolgenden Fehldurchläufe der Schleife. Ein Fehldurchlauf liegt vor, wenn die neu erhaltene Triangulierung nicht die Anforderung (A2) aus Kapitel 1 erfüllt. Bei $durchlauf \leq 3$ ist die 3 willkürlich gewählt. Man könnte eine größere Konstante wählen, doch sinkt die Wahrscheinlichkeit, noch einmal eine *passende* Triangulierung zu finden, mit jedem neuen Cluster-Punkt. Es hat sich in Testdurchläufen gezeigt, daß ein Abbruch nach dem ersten Fehldurchlauf sehr oft zum gleichen Ergebnis geführt hätte. Ist ein späterer Durchlauf allerdings erfolgreich, so verbessert sich die Gestalt der Triangulierung meist beträchtlich. Der Wert 3 stellt also einen hier gewählten Kompromiß aus der Wahrscheinlichkeit einer Verbesserung der Triangulierung und der Laufzeit dar.

Den Ablauf des Algorithmus soll folgendes Flußdiagramm illustrieren.



Im folgenden Algorithmus wird dargestellt, wie man an eine Menge von Außenpunkten der Triangulierung kommen kann. In der konvexen Hülle dieser Punkte müssen natürlich alle Daten aus $\mathcal{D}$ enthalten sein. Es sind vielfältige Lösungen vorstellbar. Welche Vorteile mit dem hier beschriebenen Weg verbunden sind, wird anschließend kurz dargestellt.

Oftmals möchte man den zu triangulierenden Bereich jedoch gar nicht berechnen, sondern einfach vorgeben. Z.B. operiert man oft nur auf dem Einheitsquadrat $[0,1]^2$. Schritt 1 in Algorithmus 3.25 kann dann durch eine Definition des (meist einfachen) Bereichs ersetzt werden. Beispielsweise

$$\mathcal{S}_A := \{(0,0),(1,0),(1,1),(0,1)\}.$$

Nun kommen wir aber zur Beschreibung des angesprochenen Algorithmus.

Algorithmus 3.27 (Schritt 1 aus Algorithmus 3.25)
Gegeben ist eine Menge $\mathcal{D} = \{\mathbf{d}_1, ..., \mathbf{d}_l\} \subset \mathbb{R}^2$. *Gesucht ist eine Menge* $\mathcal{S}_A = \{\mathbf{t}_1, ..., \mathbf{t}_{n_1}\} \subset \mathbb{R}^2$ *mit*

$$[\mathcal{S}_A] \cap \mathcal{D} = \mathcal{D}.$$

Schritt 1: *Bestimme die Daten* $\mathbf{d_i} \in \mathcal{D}$, *die auf dem Rand der konvexen Hülle* $[\mathcal{D}]$ *von* $\mathcal{D}$ *liegen. Wir erhalten die Menge*

$$\mathcal{D}' := \{\mathbf{d}_{\nu_1}, ..., \mathbf{d}_{\nu_{l_1}}\} = \mathcal{D} \cap \partial[\mathcal{D}] \subseteq \mathcal{D}.$$

Schritt 2: *Bestimme die Menge* $\mathcal{S}_A := \{\mathbf{t}_1, ..., \mathbf{t}_{l_1}\}$ *wie folgt:*

$$\mathbf{t}_i := 1.1 \left(\mathbf{d}_{\nu_i} \Leftrightarrow \frac{1}{l_1} \sum_{j=1}^{l_1} \mathbf{d}_{\nu_j} \right) + \frac{1}{l_1} \sum_{j=1}^{l_1} \mathbf{d}_{\nu_j} \quad \forall i = 1, ..., l_1.$$

Schritt 3: *Streiche (soweit möglich) Punkte aus* $\mathcal{S}_A$, *die überflüssig in dem Sinne sind, daß auch ohne diese gilt:*

$$[\mathcal{S}_A] \supset \mathcal{D}.$$

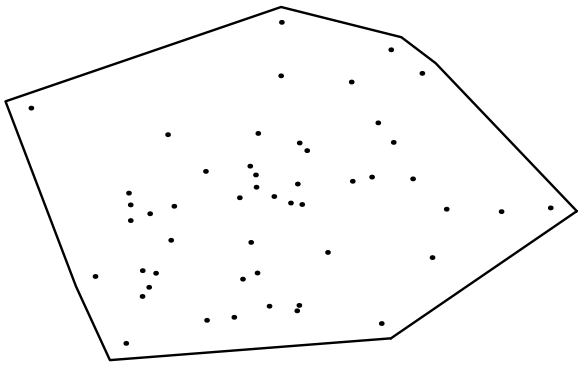
Sei dazu die Menge $\mathcal{S}_A$ *entgegen dem Uhrzeigersinn geordnet und* $\mathcal{S}_A^e = (\mathbf{t}_1, ..., \mathbf{t}_{l_1}, \mathbf{t}_1, \mathbf{t}_2)$ *eine um die ersten beiden Elemente noch einmal erweiterte Folge. Gilt für drei aufeinanderfolgende Punkte* $\mathbf{t}_{i_1}, \mathbf{t}_{i_2}, \mathbf{t}_{i_3}$ *aus* $\mathcal{S}_A^e$

$$\mathbf{d}_j \text{ liegt echt links von } \mathbf{t}_{i_1} \rightarrow \mathbf{t}_{i_3} \qquad \forall j = 1, ..., l_1 \ (\mathbf{d}_j \in \mathcal{D}'),$$

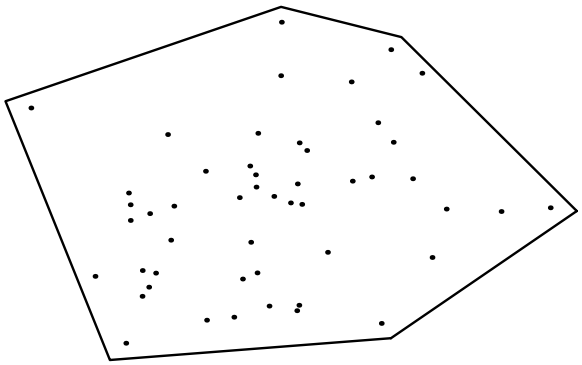
so kann $\mathbf{t}_{i_2}$ *aus* $\mathcal{S}_A^e$ *gelöscht werden.*

Bemerkung 3.28

1. Am folgenden Bild kann man gut erkennen, wie aus den Daten auf dem Rand der konvexen Hülle von $\mathcal{D}$ durch *Aufblasen* die Menge $\mathcal{S}_A$ entsteht. Dabei ist der Faktor 1.1 unter Schritt 2 (der bestimmt, wie weit aufgeblasen wird) willkürlich gewählt.



2. Das Streichen von Punkten im dritten Schritt hat den Zweck, daß insbesondere sehr kurze Kanten auf dem Rand des triangulierten Bereichs vermieden werden. Diese führen zu flächenmäßig kleinen Dreiecken in der Triangulierung, was ein rasches Finden einer ersten Triangulierung mit mindestens d Daten pro Dreieck verhindern könnte. Man erkennt am Bild unten, daß einer der beiden gestrichenen Punkte eine sehr kurze Kante verursacht hätte.



3. Es sei noch erwähnt, daß die Menge zu streichender Punkte nicht eindeutig bestimmt ist.

4. Durch die mit diesem Algorithmus vorgenommene Wahl an Punkten wird sichergestellt, daß der triangulierte Bereich nur wenig größer ist als die konvexe Hülle der Daten $\mathcal{D}$.

Bemerkung 3.29 (zu Schritt 2 aus Algorithmus 3.25)
Zur Bestimmung der Delaunay-Triangulierung sei auf 3.2.2 verwiesen. Für die Zuordnung der Daten können wir Algorithmus 3.20 verwenden.

Als nächstes ist der in Algorithmus 3.25 Schritt 7 angesprochene Clustering-Algorithmus zu beschreiben. Durch diesen werden Punkte im Inneren der Triangulierung hinzugefügt. Das hat die Vorteile, daß zum einen der im folgenden noch behandelte Algorithmus B bei größeren Start-Triangulierungen $T(\mathcal{S})$ (d.h. mit einer mächtigeren Menge $\mathcal{S}$) bessere Ergebnisse liefert als bei welchen aus sehr wenig Punkten. Zum anderen erfüllt die in Algorithmus 3.25 Schritt 2 berechnete Triangulierung oft die Anforderung (A2) aus Kapitel 1 nicht, da häufig schmale Dreiecke auftreten. Eine Triangulierung mit einem zugefügten Cluster-Punkt schafft oft Abhilfe bei diesem Problem (vgl. Beispiel 3.53).

Der Clustering-Algorithmus beruht auf dem von Schreiber [Shr91], der in Algorithmus 3.23 ausführlich beschrieben wurde. Geringfügige Änderungen sowie eine genaue Anpassung an das gegebene Problem (z.B. durch Angabe der Fehlerfunktion) sind vorgenommen worden.

Algorithmus 3.30 (Clustering gemäß Schritt 7 in Algorithmus 3.25)
Gegeben seien Daten $\mathcal{D} = \{\mathbf{d}_1, ..., \mathbf{d}_l\} \subset \mathbb{R}^2$. *Gesucht ist eine Menge von Cluster-Punkten* $\mathcal{S}_I = \{\mathbf{t}_1, ..., \mathbf{t}_{n_2}\} \subset \mathbb{R}^2$.

Schritt 1: *Der erste Cluster-Punkt* $\mathbf{t}_1$ *ergibt sich wie folgt:*

$$\mathbf{t}_1 := \frac{1}{n} \sum_{j=1}^{n} \mathbf{d}_j.$$

Dem Cluster-Punkt $\mathbf{t}_1$ *werden alle Daten* $\mathbf{d}_i \in \mathcal{D}$ *zugeordnet:* $\mathcal{M}_1 := \mathcal{D}$. *Setze* $\mathcal{S}_I := \{\mathbf{t}_1\}$.

Schritt 2: *Um die Menge der Cluster-Punkte* $\mathcal{S}_I$ *um eins zu erhöhen, gehe wie folgt vor:*

1. *Bestimme den Cluster-Punkt* $\mathbf{t}_i \in \mathcal{S}_I$, *dem die größte Menge* $\mathcal{M}_i$ *an Daten zugeordnet ist.*
2. *Streiche den so ausgewählten Cluster-Punkt, und ersetze ihn in* $\mathcal{S}_I$ *durch zwei neue* $\mathbf{t}_\alpha, \mathbf{t}_\beta$ *nach folgendem Schema (also* $\mathcal{S}_I := (\mathcal{S}_I \setminus \{\mathbf{t}_i\}) \cup \{\mathbf{t}_\alpha, \mathbf{t}_\beta\}$*):*
   (a) *Bestimme maximale x- und y-Werte der zugeordneten Daten* $\mathbf{d}_j \in \mathcal{M}_i$.
   (b) *Teile die Datenmenge durch eine Gerade in der Mitte des Rechtecks (das durch die Punkte* (x_{min}, y_{min}), (x_{max}, y_{min}),

(x_{max}, y_{max}), (x_{min}, y_{max}) bestimmt wird) parallel zur Achse mit der geringeren Ausdehnung der Datenpunkte (bei Gleichheit ist eine auszuwählen). So entstehen zwei Datenmengen $\mathcal{M}_i^\alpha$ und $\mathcal{M}_i^\beta$ ober- und unterhalb bzw. links und rechts der Geraden.

(c) *Die beiden neuen Cluster-Punkte $\mathbf{t}_\alpha$ und $\mathbf{t}_\beta$ entstehen nun als Mittel der beiden Datenmengen:*

$$\mathbf{t}_\alpha := \frac{1}{|\mathcal{M}_i^\alpha|} \sum_{\mathbf{d}_j \in \mathcal{M}_i^\alpha} \mathbf{d}_j$$

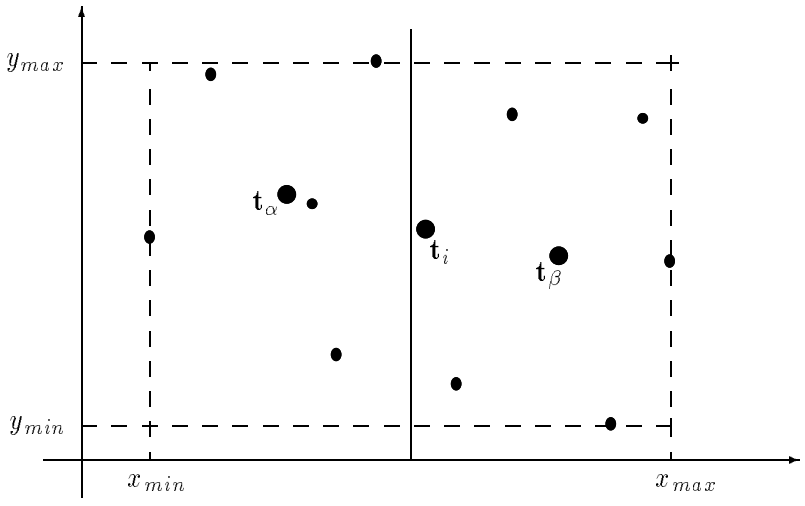
und

$$\mathbf{t}_\beta := \frac{1}{|\mathcal{M}_i^\beta|} \sum_{\mathbf{d}_j \in \mathcal{M}_i^\beta} \mathbf{d}_j.$$

3. *Ordne den Cluster-Punkten $\mathbf{t}_i$ genau die Daten $\mathbf{d}_j$ zu, die im Abschluß ihrer Voronoi-Region liegen, wenn die Cluster-Punkte die Menge der Voronoi-Center-Punkte darstellen. Dann gilt:*

$$\mathbf{d}_j \in \mathcal{M}_i \iff \|\mathbf{d}_j \Leftrightarrow \mathbf{t}_i\| = \min_k \|\mathbf{d}_j \Leftrightarrow \mathbf{t}_k\| \quad \textit{und}$$

$$\bigcup_i \mathcal{M}_i = \mathcal{D}$$



($\mathbf{t}_i$ mit zugeordneter Datenmenge $\mathcal{M}_i$ wird ersetzt durch $\mathbf{t}_\alpha$ und $\mathbf{t}_\beta$)

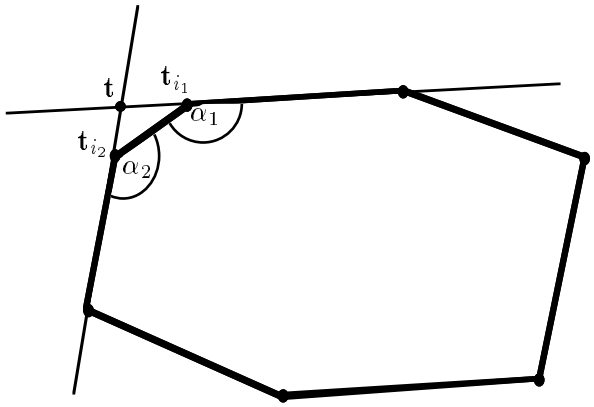
Die wesentlichen Änderungen in Algorithmus 3.30 im Vergleich zum originalen Algorithmus 3.23 von Schreiber bestehen zum einen darin, daß er auf diese spezielle Berechnung abgestimmt wurde. D.h. es sind die Gewichte w_i weggefallen, und eine andere Fehlerfunktion ist speziell angegeben worden. Die Region mit dem größten Fehler stellt die dar, die die meisten Daten enthält. Die zweite grundlegende Änderung ist, daß der zunächst angegebene Schritt 4 entfallen ist. Der Algorithmus wurde mit und ohne diesen Schritt getestet, und die Unterschiede in den Ergebnissen waren so gering, daß er letztendlich entfallen ist. Dies ist auf die besondere Fehlerfunktion zurückzuführen.

Wurde bis Schritt 11 in Algorithmus 3.25 keine passende Triangulierung gefunden, so können wir die Suche durch Reduzierung der Punkte $\mathcal{S}_A$ vereinfachen oder erst ermöglichen. Dies wird in Schritt 12 verlangt und im folgenden näher erläutert. Die Beschreibung des Algorithmus erfolgt zum optimalen Verständnis in Prosa, mit einer anschließenden Abbildung.

Algorithmus 3.31 (Schritt 12 aus Algorithmus 3.25)
Um die Menge der Außenpunkte $\mathcal{S}_A$ zu verkleinern, werden die zwei Punkte einer Kante $\mathbf{t}_{i_1} \Leftrightarrow \mathbf{t}_{i_2}$ $(\mathbf{t}_{i_1}, \mathbf{t}_{i_2} \in \mathcal{S}_A)$, auf die verzichtet werden soll, ersetzt durch den Schnittpunkt $\mathbf{t}$ der Verlängerung der beiden angrenzenden Kanten. Es sollen die Punkte der kürzesten Kante ersetzt werden, deren angrenzende Kanten so liegen, daß die beiden Geraden durch diese Kanten folgende Bedingung erfüllen:

> *Eine Gerade durch die Kante $\mathbf{t}_{i_1} \Leftrightarrow \mathbf{t}_{i_2}$, die ersetzt werden soll, teilt den Raum in zwei Halbebenen. Die beiden Geraden durch die angrenzenden Kanten haben einen Schnittpunkt in der Halbebene, die keine Daten enthält.*

Existiert keine solche Kante (vgl. Bemerkung 3.32 3.), so soll die Menge der Daten durch ein Dreieck eingeschlossen werden, dessen Eckpunkte dann $\mathcal{S}_A$ bilden.



Bemerkung 3.32 (zu Algorithmus 3.31)

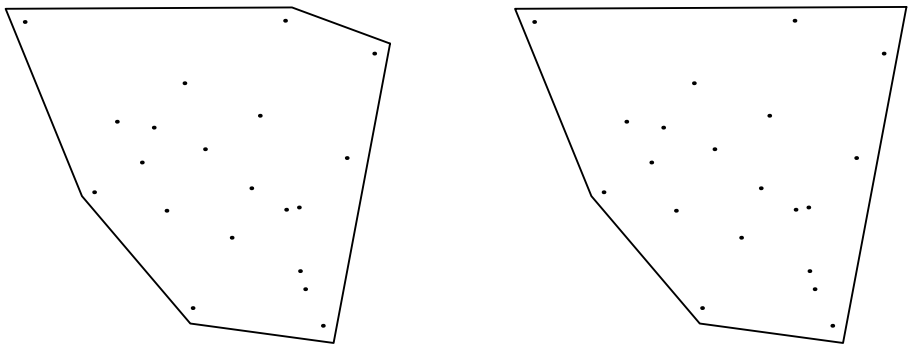
1. Die kürzeste Kante wurde gewählt, da kurze Kanten kleine Dreiecke verursachen und die Vermutung nahe liegt, daß daher keine Triangulierung gefunden wurde, die in jedem Dreieck genügend Daten hat.

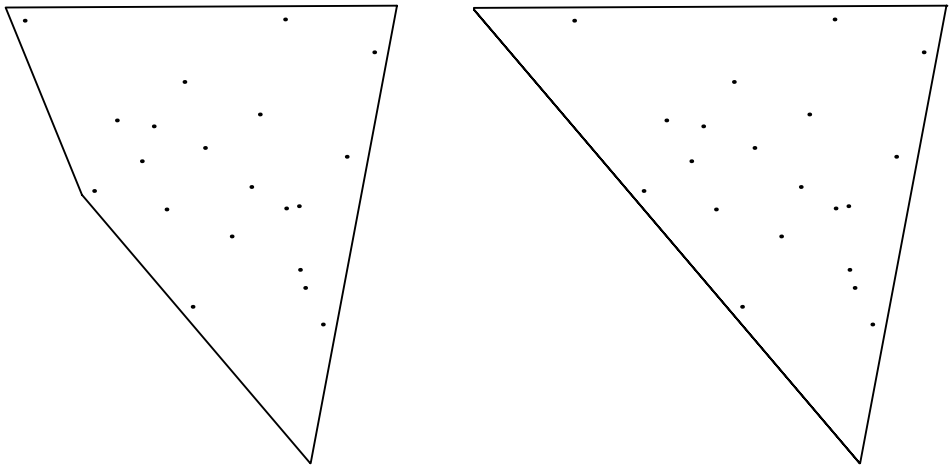
2. Leicht einzusehen ist, daß die Bedingung an die Lage eines Schnittpunktes genau dann gegeben ist, wenn die Summe der Innenwinkel α_1 und α_2 größer als π ist. (Bei $\alpha_1 + \alpha_2 = \pi$ existiert kein Schnittpunkt, und gilt $\alpha_1 + \alpha_2 < \pi$, so liegt der Schnittpunkt auf der falschen Seite.)

3. Desweiteren ist leicht zu erkennen, daß bei jedem konvexen Polygon mit (mindestens) fünf Ecken eine Kante existiert, die auf diese Weise ersetzt werden kann. Ein Viereck besitzt genau dann keine solche Kante, wenn es ein Parallelogramm ist. (Beweise ergeben sich sofort, wenn man berücksichtigt, daß die Summe der Innenwinkel in einem n-Eck $(n \Leftrightarrow 2)\pi$ ist, sowie daß bei den gegebenen Polygonen alle Innenwinkel kleiner als π sind.)

4. Aufgrund von Algorithmus 3.31 kann garantiert werden, daß eine Triangulierung gefunden wird, die in jedem Dreieck mindestens d Daten enthält (vgl. dazu Satz 3.46).

Beispiel 3.33 (zu Algorithmus 3.31)
An dem vorliegenden Beispiel soll demonstriert werden, wie Algorithmus 3.31 arbeitet. Dazu wurde eine Menge $\mathcal{D}$ von 20 Daten generiert und eine Mindestanzahl von $d = 20$ Daten pro Dreieck der Triangulierung verlangt. Folglich kann nur eine triviale Triangulierung aus genau einem Element akzeptiert werden. Dreimal müssen je zwei Punkte gegen einen dritten ausgetauscht werden.



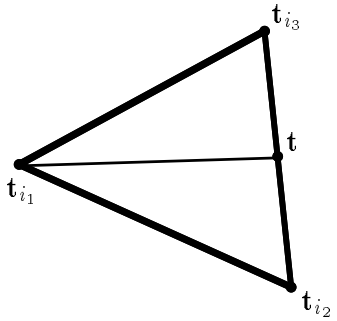


3.3.2 Algorithmus B

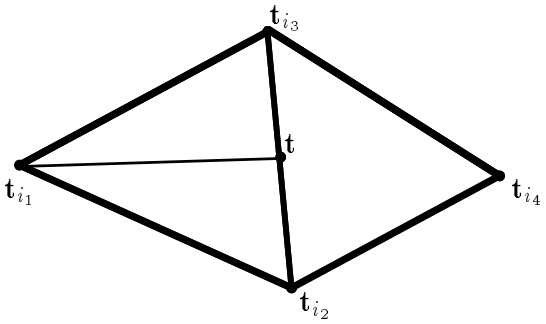
Algorithmus B untersucht eine gegebene Triangulierung, die in jedem Dreieck mindestens d Daten hat, ob sie verfeinert werden kann derart, daß Punkte hinzugefügt werden und die Triangulierung somit aus mehr Dreiecken besteht. Dabei soll jedes Dreieck weiterhin mindestens d Daten beinhalten.

Wir gehen von einer geordneten Triangulierung $\mathcal{T} = \{T_1, ..., T_m\}$ aus, der ein Vektor $\{\mathcal{D}_1, ..., \mathcal{D}_m\}$ zugeordnet ist, wobei $\mathcal{D}_i \subseteq \mathcal{D}$ genau die Menge der Daten ist, die T_i beinhaltet. Es soll gelten: $|\mathcal{D}_i| \geq |\mathcal{D}_{i+1}|$ $(i = 1, ..., m \Leftrightarrow 1)$.

Beim Einfügen von Punkten in die Triangulierung muß man meistens zwei Dreiecke zugleich betrachten. Es wäre leicht, ein Dreieck mit $2d$ Daten in zwei Dreiecke zu unterteilen, so daß jedes Dreieck d Daten beinhaltet.

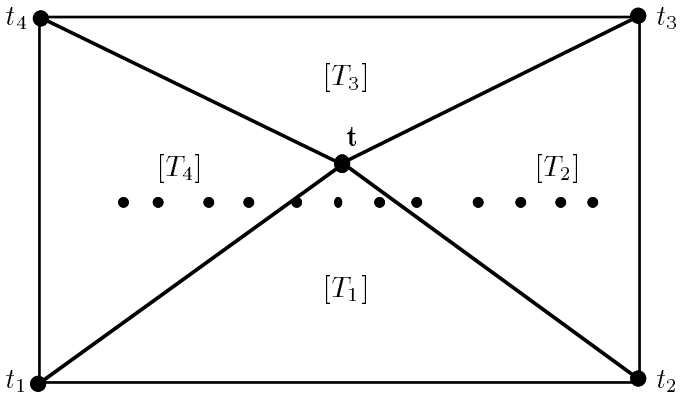


Doch dazu müßte $\mathbf{t}_{i_2} \Leftrightarrow \mathbf{t}_{i_3}$ eine Außenkante sein, denn sonst hätte man keine Triangulierung mehr vorliegen, da 3. und 4. in Definition 2.10 verletzt wären.



Die einzige Möglichkeit sich generell beim Einfügen von Punkten auf ein einziges Dreieck zu beschränken ist, im Inneren eines Dreiecks der Triangulierung einen Punkt hinzuzufügen. Die neuen Kanten erhält man durch Verbindungen des neuen Punktes zu den drei Eckpunkten des Dreiecks, in das dieser eingesetzt wurde. Dieses Verfahren wird in Algorithmus C angewandt und bringt diverse Nachteile mit sich, die später noch aufgezeigt werden.

Wir sind also darauf angewiesen, mehrere Dreiecke zu betrachten. Die Suche nach einem Punkt $\mathbf{t}$, der die Triangulierung weiter verfeinert, findet bei Algorithmus B meistens in einem Viereck statt (oder manchmal auf einer Dreiecksaußenseite). Leider existiert nicht in jedem Viereck, das (mindestens) $4d$ Daten beinhaltet, ein Punkt $\mathbf{t}$ derart, daß vier Dreiecke entstehen, die jeweils d Daten enthalten. Die folgende Abbildung verdeutlicht das:



- Liegt $\mathbf{t}$ unter der Linie, auf der die Daten liegen, so beinhaltet $[T_1]$ keine Daten.
- Liegt $\mathbf{t}$ darüber, so beinhaltet $[T_3]$ keine.
- Liegt $\mathbf{t}$ darauf, so beinhalten beide maximal ein Datum.

Aus diesem Grund muß die Suche eventuell abgebrochen werden, und dazu benötigen wir ein passendes Kriterium, auf das später noch eingegangen wird.

Im folgenden wird der Ablauf von Algorithmus B zunächst in groben Zügen beschrieben. Wie bereits in Algorithmus A gehen wir danach noch einmal genauer auf einzelne Schritte ein.

Algorithmus 3.34 (Algorithmus B)
Gegeben sei eine Triangulierung $\mathcal{T}(\mathcal{S}) = \{T_1, ..., T_m\}$ *der Menge* $\mathcal{S} \subset \mathbb{R}^2$, *sowie zugehörige Datenmengen* $\mathcal{D}_1, ..., \mathcal{D}_m \subset \mathcal{D} \subset \mathbb{R}^2$. *Dabei gelte*

$$|\mathcal{D}_1| \geq ... \geq |\mathcal{D}_m| \geq d \qquad \textit{sowie} \qquad [T_i] \cap \mathcal{D} = \mathcal{D}_i \ (i = 1, ..., m).$$

Gesucht ist eine feinere Triangulierung $\mathcal{T}(\mathcal{S}') = \{T_1, ..., T_{m'}\}$ *der Menge* $\mathcal{S}' \supset \mathcal{S}$, *so daß immer noch*

$$|\mathcal{D}_i| = |[T_i] \cap \mathcal{D}| \geq d \ (i = 1, ..., m')$$

gilt.

Schritt 1: *Setze* $j := 1$.

Schritt 2: *Gilt* $|\mathcal{D}_j| < 2d$ *oder hat die Triangulierung weniger als* j *Elemente, so beende den Algorithmus.*

Schritt 3: *Untersuche mit Hilfe des* j*-ten Dreiecks, ob ein Punkt* $\mathbf{t}$ *hinzugefügt werden kann. Falls ja, füge ihn hinzu, indem* $\mathcal{T}(\mathcal{S})$ *mit* $\mathcal{S} := \mathcal{S} \cup \{\mathbf{t}\}$ *und der neue Vektor* $(\mathcal{D}_1, ..., \mathcal{D}_m)$ *bestimmt werden. Findet man keinen Punkt* $\mathbf{t}$, *setze* $\{\mathbf{t}\} := \emptyset$ *(vgl. Algorithmus 3.35).*

Schritt 4: *Gilt* $\{\mathbf{t}\} = \emptyset$, *setze* $j := j+1$ *und gehe zu* **Schritt 2**. *Ansonsten gehe zu* **Schritt 5**.

Schritt 5: *Optimiere die neu erhaltene Triangulierung, und gehe dann zu* **Schritt 2**. *(Dabei soll die optimierte Triangulierung wieder geordnet sein, d.h. es gelte weiter* $|\mathcal{D}_i| \geq |\mathcal{D}_{i+1}|$.*)*

Zunächst wird der wichtige Schritt 3 genauer beschrieben. Die Skizzen in der anschließenden Bemerkung 3.36 dienen der besseren Veranschaulichung.

Algorithmus 3.35 (Schritt 3 aus Algorithmus 3.34)
Gegeben sei ein Element $T_j = \{\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}\}$ *einer Triangulierung* $\mathcal{T}(\mathcal{S}) = \{T_1, ..., T_m\}$ *sowie die Datenmengen* $(\mathcal{D}_1, ..., \mathcal{D}_m)$, *die die einzelnen Dreiecke der Triangulierung enthalten. Gesucht ist ein Punkt* $\mathbf{t}$ *in dem Dreieck* $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}]$ *oder einem angrenzenden (d.h. die beiden Dreiecke haben eine gemeinsame Kante), so daß in den neu entstehenden Dreiecken jeweils mindestens* d *Daten aus* $\mathcal{D}$ *liegen.*

Schritt 1: *Untersuche die Kante gegenüber dem größten Innenwinkel des Dreiecks* $[T_j] = [\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}]$ *(O.B.d.A. sei dies* $\mathbf{t}_{j_2} \Leftrightarrow \mathbf{t}_{j_3}$*) der Triangulierung* $\mathcal{T}$ *darauf, ob sie eine Außen- oder eine Innenkante von* $\mathcal{T}$ *ist.*

Schritt 2: *Ist sie eine Außenkante, so suche auf ihr nach einem Punkt* $\mathbf{t}$*, so daß die Dreiecke* $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}]$ *und* $[\mathbf{t}_{j_1}, \mathbf{t}, \mathbf{t}_{j_3}]$ *je mindestens* d *Daten beinhalten.*

Schritt 3: *Ist sie eine Innenkante gehe wie folgt vor:*

(a) *Bestimme das an* $[T_j]$ *angrenzende Dreieck* $[T_i] = [\mathbf{t}_{i_1}, \mathbf{t}_{i_2}, \mathbf{t}_{i_3}] = [\mathbf{t}_{i_1}, \mathbf{t}_{j_3}, \mathbf{t}_{j_2}]$*, das ebenfalls* $\mathbf{t}_{j_2} \Leftrightarrow \mathbf{t}_{j_3}$ *als Kante enthält (O.B.d.A. sei dies* $\mathbf{t}_{i_2} \Leftrightarrow \mathbf{t}_{i_3}$*).*

(b) *Gilt* $|\mathcal{D}_i \cup \mathcal{D}_j| \geq 4k$*, so suche, wie unter (i) und (ii) beschrieben, im Inneren des Vierecks* $[\{\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{i_1}, \mathbf{t}_{j_3}\}]$ *nach einem Punkt* $\mathbf{t}$*, so daß jedes der Dreiecke* $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}], [\mathbf{t}_{j_2}, \mathbf{t}_{i_1}, \mathbf{t}]$*,* $[\mathbf{t}_{i_1}, \mathbf{t}_{j_3}, \mathbf{t}]$ *und* $[\mathbf{t}_{j_3}, \mathbf{t}_{j_1}, \mathbf{t}]$ *mindestens* d *Daten beinhaltet.*

(i) *Ist das Viereck echt konvex, so suche zunächst auf der Strecke* $\mathbf{t}_{j_1} \Leftrightarrow \mathbf{t}_{i_1}$ *(vgl. Skizze in Bemerkung 3.36).*

(ii) *War die Suche nicht erfolgreich, oder ist das Viereck nicht echt konvex, so suche auf der Strecke* $\mathbf{t}_{j_2} \Leftrightarrow \mathbf{t}_{j_3}$ *nach einem geeigneten Punkt* $\mathbf{t}$*.*

Schritt 4: *War die Suche nicht erfolgreich, so setze sie zunächst mit Hilfe der Kante gegenüber des zweitgrößten Innenwinkels von* $[T_j]$ *fort (durchlaufe also erneut die Schritte 1 bis 3). War auch dies vergebens, so versuche es mit der Kante gegenüber des kleinsten Winkels, und falls auch dies erfolglos verlief, setze* $\{\mathbf{t}\} := \emptyset$*.*

Schritt 5: *Hat man hingegen einen Punkt* $\mathbf{t}$ *gefunden, so streiche man* T_j *bzw.* T_i *und* T_j *und ersetze sie durch die zwei bzw. vier neu erhaltenen Dreiecke. (Das Einordnen in die Triangulierung soll so erfolgen, daß sie weiterhin geordnet ist, d.h. es gilt weiter* $|\mathcal{D}_i| \geq |\mathcal{D}_{i+1}|$*.)*

Bemerkung 3.36 (zu Algorithmus 3.35)

1. Die Suche eines Punktes $\mathbf{t}$ gemäß Algorithmus 3.35 auf einer Außenkante $\mathbf{t}_{j_2} \Leftrightarrow \mathbf{t}_{j_3}$ eines Dreiecks $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}]$ wurde wie folgt realisiert:

   Schritt 1: *Setze* $a := 0.5, b := 0.5$ *und* $i := 1$*.*

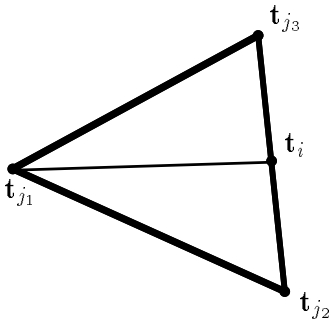
   Schritt 2: *Setze* $\mathbf{t}_i := a\mathbf{t}_{j_2} + b\mathbf{t}_{j_3}$*.*

   Schritt 3: *Gilt* $i \geq N$*, dann breche die Suche ab (*N *ist frei wählbar und bestimmt die Genauigkeit der Berechnung).*

   Schritt 4: *Prüfe, ob die Dreiecke* $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_i]$ *und* $[\mathbf{t}_{j_3}, \mathbf{t}_{j_1}, \mathbf{t}_i]$ *je mindestens* d *Daten beinhalten.*

   Schritt 5: *Ist dies der Fall, setze* $\mathbf{t} := \mathbf{t}_i$ *und breche ab.*

Schritt 6: *Beinhaltet* $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_i]$ *weniger als* d *Daten, so setze* $a := a \Leftrightarrow 0.5^{i+1}$ *und* $b := b + 0.5^{i+1}$, *sonst setze* $a := a + 0.5^{i+1}$ *und* $b := b \Leftrightarrow 0.5^{i+1}$. *Setze* $i := i + 1$ *und gehe zu* **Schritt 2**.



2. Die Punktsuche auf einer Strecke im Inneren eines Vierecks erfolgt analog. Es ergeben sich lediglich vier Dreiecke. Daher kommen zusätzlich zum N-ten Schleifendurchlauf zwei weitere Abbruchkriterien mit negativem Ausgang hinzu. Wir ersetzen Schritt 4 und Schritt 5 durch

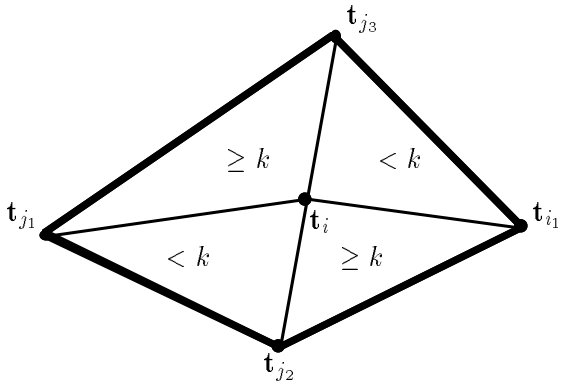
Schritt 4 : *Prüfe, ob die Dreiecke* $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_i], [\mathbf{t}_{j_2}, \mathbf{t}_{i_1}, \mathbf{t}_i], [\mathbf{t}_{i_1}, \mathbf{t}_{j_3}, \mathbf{t}_i]$, *und* $[\mathbf{t}_{j_3}, \mathbf{t}_{j_1}, \mathbf{t}_i]$ *je mindestens* d *Daten beinhalten.*

Schritt 5 : *Ist dies der Fall, setze* $\mathbf{t} := \mathbf{t}_i$ *und breche ab.*

Schritt 5 a: *Beinhalten die Dreiecke* $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_i]$ *und* $[\mathbf{t}_{i_1}, \mathbf{t}_{j_3}, \mathbf{t}_i]$ *je weniger als* d *Daten, so breche ab.*

Schritt 5 b: *Beinhalten die Dreiecke* $[\mathbf{t}_{j_3}, \mathbf{t}_{j_1}, \mathbf{t}_i]$ *und* $[\mathbf{t}_{j_2}, \mathbf{t}_{i_1}, \mathbf{t}_i]$ *je weniger als* d *Daten, so breche ab.*

Schritt 6 : *Beinhaltet* $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_i]$ *oder* $[\mathbf{t}_{j_2}, \mathbf{t}_{i_1}, \mathbf{t}_i]$ *weniger als* d *Daten, so setze* $a := a \Leftrightarrow 0.5^{i+1}$ *und* $b := b + 0.5^{i+1}$, *sonst setze* $a := a + 0.5^{i+1}$ *und* $b := b \Leftrightarrow 0.5^{i+1}$. *Setze* $i := i + 1$ *und gehe zu* **Schritt 2**.

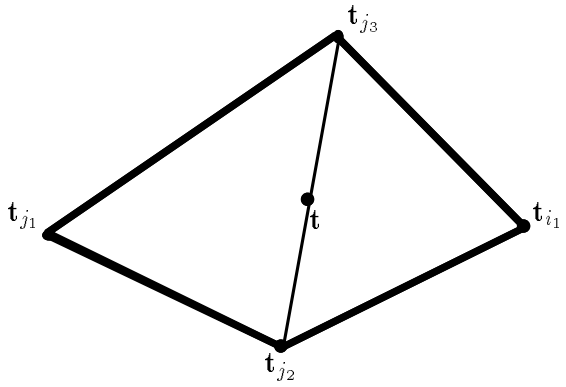


(Versuch der Punktsuche im Viereck gemäß Algorithmus 3.35, zusätzliches Abbruchkriterium (Schritt 5 a) wird angewandt)

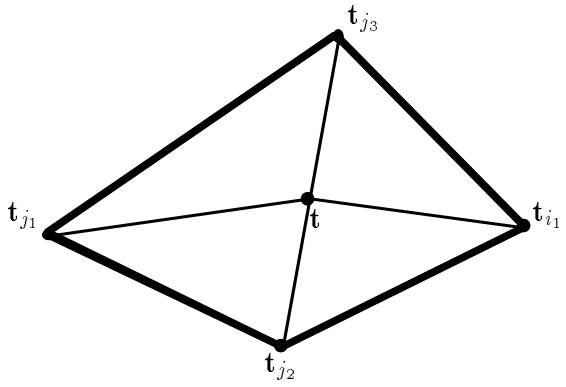
3. Die neu erhaltenen Dreiecke beinhalten fast immer weniger Daten als das untersuchte Dreieck $[T_j]$, so daß sie zu einem späteren Zeitpunkt nochmals untersucht werden können, da sie weiter hinten in der geordneten Triangulierung stehen.

Es hat sich gezeigt, daß die Verwendung eines Optimierungsalgorithmus für den Erhalt einer guten Triangulierung unabdingbar ist. In Schumaker [Shu87] wird auf einige Algorithmen verwiesen, die eine Triangulierung nach Einsetzen eines Punktes optimieren. Einer von diesen wurde bereits vorgestellt (vgl. Algorithmus 3.21). Das Einfügen eines Punktes durch Algorithmus 3.35 erfolgt jedoch eventuell anders als bei Cline und Renka (vgl. Abschnitt 3.2.2.2).

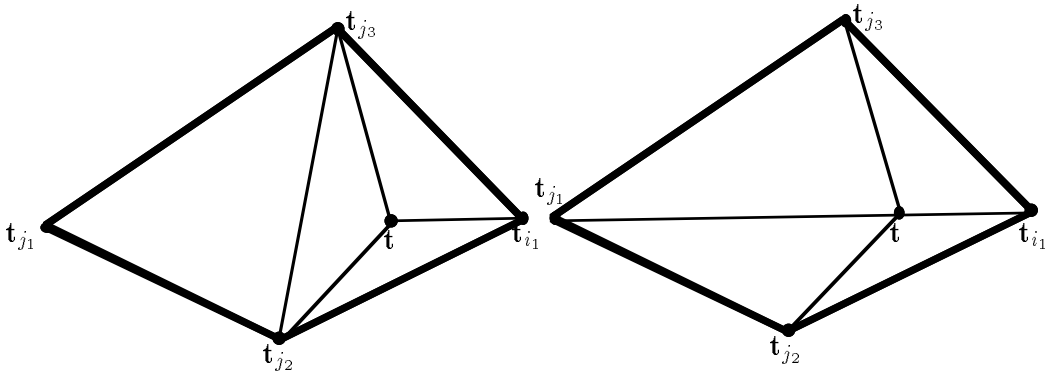
Angenommen es wird ein Punkt $\mathbf{t}$ in das Viereck $[\{\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{i_1}, \mathbf{t}_{j_3}\}]$, das sich aus den zwei Dreiecken $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}]$ und $[\mathbf{t}_{i_1}, \mathbf{t}_{j_3}, \mathbf{t}_{j_2}]$ ergibt, eingesetzt. Dann liegt der neue Punkt entweder auf der Kante $\mathbf{t}_{j_2} \Leftrightarrow \mathbf{t}_{j_3}$ oder auf $\mathbf{t}_{j_1} \Leftrightarrow \mathbf{t}_{i_1}$.



Liegt der neue Punkt $\mathbf{t}$ also auf $\mathbf{t}_{j_2} \Leftrightarrow \mathbf{t}_{j_3}$, so stimmt die neu erhaltene (Teil-) Triangulierung bei Cline und Renka mit der aus Algorithmus 3.35 überein:



Liegt $\mathbf{t}$ hingegen auf $\mathbf{t}_{j_1} \Leftrightarrow \mathbf{t}_{i_1}$, so ergeben sich unterschiedliche Ergebnisse:



(gemäß Cline, Renka) (gemäß Algorithmus 3.35)

In diesem Fall muß vorab ein Tauschtest bzgl. des erweiterten Max-Min-Winkelkriteriums auf die Kante $\mathbf{t} \Leftrightarrow \mathbf{t}_{i_1}$ bzw. $\mathbf{t} \Leftrightarrow \mathbf{t}_{j_1}$ angewandt werden, was davon abhängt, ob $\mathbf{t}$ in $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}]$ liegt oder in $[\mathbf{t}_{j_2}, \mathbf{t}_{i_1}, \mathbf{t}_{j_3}]$. Ist der Tauschtest positiv, so wird ein Diagonalentausch durchgeführt. Dieser Tauschtest würde im Verlauf des Optimierungslgorithmus von Cline und Renka (und der Wahl der Kanten gemäß Cline und Renka) irgendwann durchgeführt, je nach Wahl des ersten Nachbarn in Schritt 1 von Algorithmus 3.21 auch als erstes. Beim der Wahl der Kanten gemäß Algorithmus 3.35 ist allerdings nicht garantiert, daß dieser Tauschtest durchgeführt wird. Somit wird er vorangestellt und prüft lediglich, welche der beiden Möglichkeiten die bessere ist. Im Anschluß können wir dann allerdings auf den bereits vorgestellten Optimierungsalgorithmus 3.21 von Cline und Renka zurückgreifen.

Algorithmus 3.37 (Optimierung gemäß Algorithmus 3.34 Schritt 5) *Gegeben ist eine Triangulierung $\mathcal{T}'(\mathcal{S})$, in die ein Punkt $\mathbf{t}$ – wie in Algorithmus 3.35 zuvor beschrieben – eingefügt wurde. Gesucht ist eine optimierte Triangulierung $\mathcal{T}(\mathcal{S}_i)$.*

Schritt 0: *Wurde $\mathbf{t}$ auf einer Kante der ursprünglichen Triangulierung eingefügt, so gehe zu* **Schritt 1**. *Wurde $\mathbf{t}$ im Inneren eines Dreiecks $[\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}]$ der alten Triangulierung eingefügt, das mit $\mathbf{t}_{i_1}$ ein echt konvexes Viereck $[\{\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{i_1}, \mathbf{t}_{j_3}\}]$ bildet, gehe wie folgt vor:*

1. *Wende einen Tauschtest bzgl. des erweiterten Max-Min-Winkelkriteriums auf die Kante $\mathbf{t} \Leftrightarrow \mathbf{t}_{i_1}$ an.*
2. *Ist der Tauschtest positiv, führe den Diagonalentausch durch.*

Schritt 1: *Setze*

$$\alpha := k \text{ und } \varepsilon := k,$$

wobei k der Index des ersten Nachbarn von $\mathbf{t}$ ist.

Schritt 2: *Setze*

$$\beta := k,$$

wobei k der Index des Punktes $\mathbf{t}_k$ ist, der mit $\mathbf{t}$ und $\mathbf{t}_\alpha$ ein Dreieck $[\mathbf{t}_k, \mathbf{t}, \mathbf{t}_\alpha]$ der Triangulierung bildet.

Schritt 3: *Ist $\mathbf{t}_\alpha \Leftrightarrow \mathbf{t}_\beta$ eine Außenkante der Triangulierung, so gehe zu* **Schritt 6**.

Schritt 4: *Setze*

$$\gamma := k,$$

wobei k der Index des Punktes $\mathbf{t}_k$ ist, der mit $\mathbf{t}_\beta$ und $\mathbf{t}_\alpha$ ein Dreieck $[\mathbf{t}_k, \mathbf{t}_\beta, \mathbf{t}_\alpha]$ der Triangulierung bildet.

Schritt 5: *Wende einen Tauschtest auf $\mathbf{t}_\alpha \Leftrightarrow \mathbf{t}_\beta$ bzgl. des erweiterten Max-Min-Winkelkriteriums an. Ist der Test positiv, so führe den Diagonalentausch durch, setze*

$$\beta := \gamma$$

und gehe zu **Schritt 3**.

Schritt 6: *Setze*

$$\alpha := \beta.$$

Schritt 7: *Gilt $\alpha = \varepsilon$ oder ist $\mathbf{t}_\alpha \rightarrow \mathbf{t}$ eine Außenkante, so beende den Algorithmus. Andernfalls gehe zu* **Schritt 2**.

Bemerkung 3.38 (zur Optimierung)
Wird ein neuer Punkt $\mathbf{t}$ in eine Delaunay-Triangulierung eingesetzt und diese mit dem beschriebenen Algorithmus bzgl. des Max-Min-Winkelkriteriums optimiert, so entsteht erneut eine Delaunay-Triangulierung. Bei dem erweiterten Max-Min-Winkelkriterium können wir das nicht garantieren, da die Tauschtests an einer weiteren Bedingung scheitern können, nämlich an einer zu kleinen Menge an Daten in den Dreiecken. Wir erhalten allerdings eine bzgl. des erweiterten Max-Min-Winkelkriteriums lokal optimale Triangulierung. (Daraus folgt beim Max-Min-Winkelkriterium die globale Optimalität.)

3.3.3 Algorithmus C

Bislang haben wir erreicht, eine Triangulierung zu finden, die in jedem Dreieck eine Mindestanzahl an Daten hat. Durch die beiden vorangegangenen Algorithmen ist diese oft sogar schon sehr fein, d.h. daß in den Dreiecken auch nicht übermäßig viele Daten liegen. Eine Garantie dafür, sprich eine obere Schranke, konnten wir dafür allerdings bislang nicht geben. Das soll nun durch Algorithmus C erreicht werden. Wir beginnen zunächst mit dem dafür wichtigen Existenzsatz, aus dessen Beweis der Algorithmus abgeleitet wird.

Satz 3.39 (Grundlage für Algorithmus C)
Gegeben sei ein Dreieck $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$, *das eine Datenmenge* $\mathcal{D} = \{\mathbf{d}_1, ..., \mathbf{d}_l\}$ *mit* $l \geq 3d$ *beinhaltet*

$$\mathcal{D} \cap [\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3] = \mathcal{D}.$$

Für die Lage der Daten gilt:

- *Auf keiner offenen Dreieckskante* $\mathbf{t}_i \overset{\circ}{-} \mathbf{t}_j$ $((i,j) \in \{(1,2),(2,3),(1,3)\})$ *liegen soviele Daten, daß im Dreieck ohne jene Kante weniger als* $2d$ *Daten liegen:*

$$|\,([\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3] \setminus \mathbf{t}_i \overset{\circ}{-} \mathbf{t}_j) \cap \mathcal{D}| \leq 2d.$$

- *Auf keinen zwei (halboffenen) Dreieckskanten* $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j \cup \mathbf{t}_j \Leftrightarrow \mathbf{t}_k \setminus \{\mathbf{t}_i, \mathbf{t}_k\}$ $((i,j,k) \in \{(1,2,3),(2,3,1),(3,1,2)\})$ *liegen soviele Daten, daß im Dreieck ohne diese zwei Kanten weniger als* d *Daten liegen:*

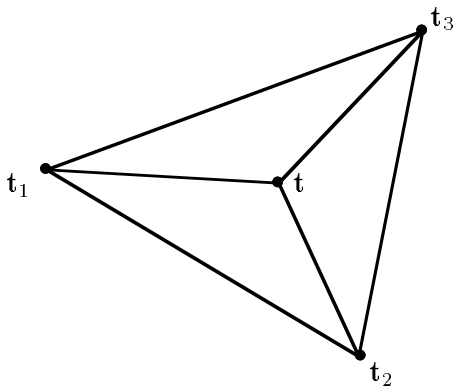
$$|\,([\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3] \setminus ((\mathbf{t}_i \Leftrightarrow \mathbf{t}_j \cup \mathbf{t}_j \Leftrightarrow \mathbf{t}_k) \setminus \{\mathbf{t}_i, \mathbf{t}_k\})) \cap \mathcal{D}| \leq d.$$

Dann existiert ein Punkt $\mathbf{t}$ *im Inneren des Dreiecks, so daß in den drei entstandenen Dreiecken je (mindestens)* d *Daten liegen. Diesen Punkt kann man durch einen Algorithmus bestimmen.*

Der folgende Beweis des Satzes ist konstruktiv. Wir gehen immer von einem gegebenen Punkt $\mathbf{t}$ aus. Daraufhin können verschiedene Fälle der Verteilung der Daten $\mathcal{D}$ auf die drei Teildreiecke $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3], [\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$ und $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]$ auftreten. Zu jedem einzelnen Fall wird ein Verfahren angegeben, wie wir einen neuen Punkt $\mathbf{t}'$ bestimmen. Mit diesem neuen Punkt tritt eventuell ein anderer Fall ein, und es wird gezeigt, daß wir nach endlich vielen neu bestimmten Punkten den Fall erreichen, daß in jedem Teildreieck mindestens d Daten liegen. Mit jedem neu bestimmten Punkt, tritt eine Verbesserung ein insofern, daß die Anzahl der Daten in den Dreiecken mit zu wenigen Daten vergrößert wird. Sind am Anfang (als Initialisierung wird der Schwerpunkt des Dreiecks empfohlen) alle Daten in einem Teildreieck, so benötigen wir maximal $2d$ Iterationen, um zum Ziel zu gelangen.

Beweis: Zunächst werden einige Bezeichnungen vereinbart.
Wir beginnen mit dem Inneren der drei Teildreiecke, die durch den Punkt $\mathbf{t}$ im Inneren von $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$ entstehen:

$$\begin{aligned} I_1 &= [\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3] \setminus \partial[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3] \\ I_2 &= [\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3] \setminus \partial[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3] \\ I_3 &= [\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}] \setminus \partial[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]. \end{aligned}$$



Als nächstes definieren wir zwei Teildreiecke, denen eine halboffene Kante entnommen wurde, was an dieser Stelle noch nicht einleuchtet, aber im Verlauf des Beweises noch von Bedeutung sein wird:

$$\begin{aligned} TD_2 &= [\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3] \setminus (\mathbf{t} \stackrel{o}{-} \mathbf{t}_3 \cup \{\mathbf{t}\}) \\ TD_3 &= [\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}] \setminus (\mathbf{t} \stackrel{o}{-} \mathbf{t}_2 \cup \{\mathbf{t}\}). \end{aligned}$$

Mengen $\mathcal{M}$, die sich bzgl. eines neuen Punktes $\mathbf{t}'$ ergeben, seien analog definiert und mit $\mathcal{M}'$ bezeichnet. Als letztes wollen wir noch die Teilmenge der Daten $\mathcal{D}$ in einer Teilmenge $\mathcal{M}$ des Dreiecks mit einer Bezeichnung versehen:

$$\mathcal{D}_{\mathcal{M}} = \mathcal{D} \cap \mathcal{M} \quad (\mathcal{M} \subset [\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]).$$

Nun sollen die verschiedenen Fälle der Verteilung der Daten auf drei Teildreiecke, die durch das Einfügen eines Punktes $\mathbf{t}$ in ein Dreieck entstehen können, zunächst einmal aufgelistet werden: von **Fall 1** *Kein Teildreieck enthält genügend Daten* bis **Fall 4** *Alle drei Teildreiecke enthalten genügend Daten.*

O.B.d.A. gelte:

$$\underbrace{|\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]}|}_{=:a} \leq \underbrace{|\mathcal{D}_{[\mathbf{t}_1,\mathbf{t},\mathbf{t}_3]}|}_{=:b} \leq \underbrace{|\mathcal{D}_{[\mathbf{t}_1,\mathbf{t}_2,\mathbf{t}]}|}_{=:c}.$$

Fall 1: $a, b, c < d$: Kann nicht eintreten. Ein Widerspruch wird hergeleitet.

Fall 2: $a, b < d \wedge c \geq d$: Konstruktion eines Punktes $\mathbf{t}'$, so daß

$$\begin{aligned} & (a' > a \ \wedge \ b' \geq b \ \wedge \ c' \geq d) \\ \vee \ & (a' \geq a \ \wedge \ b' > b \ \wedge \ c' \geq d) \end{aligned}$$

gilt. Daraus folgt sofort, daß mit dem neuen Punkt Fall 2 ($a', b' < d$), Fall 3 ($a' < d \ \oplus \ b' < d$) oder Fall 4 ($a', b' \geq d$) eintrifft (evtl. ist eine Umindizierung erforderlich, damit wieder $a' \leq b' \leq c'$ gilt). Trifft Fall 2 ein, ist auch klar, daß nach endlich vielen Schritten Fall 3 oder Fall 4 eintreffen wird.

Fall 3: $a < d \quad \wedge \quad b, c \geq d$: Diesen Fall können wir noch in 4 Teilfälle unterteilen. In zwei Teildreiecken liegen mindestens d Daten. Es kann aber sein, daß diese teilweise auf einer bestimmten Kante liegen, was für den Beweis besondere Bedeutung hat, falls im Dreieck ohne diese Kante nicht mehr genügend Daten liegen.

(a) $|\mathcal{D}_{TD_2}| < d \ \wedge \ |\mathcal{D}_{TD_3}| < d$: Kann nicht eintreten. Ein Widerspruch wird hergeleitet.

(b) $|\mathcal{D}_{TD_2}| \geq d \quad \wedge \quad |\mathcal{D}_{TD_3}| \geq d$: Konstruktion eines Punktes $\mathbf{t}'$, so daß

$$a' > a \ \wedge \ b', c' \geq d$$

gilt. Es tritt also entweder Fall 3 ($a' < d$) mit größerem $a := a'$ oder Fall 4 ($a' \geq d$) ein.

(c) $|\mathcal{D}_{TD_2}| \geq d \quad \wedge \quad |\mathcal{D}_{TD_3}| < d$: Konstruktion eines Punktes $\mathbf{t}'$, so daß

$$\begin{array}{ll} & \text{(i)} \quad (a' > a \ \wedge \ b', c' \geq d) \\ \vee & \text{(ii)} \quad (|\mathcal{D}_{TD'_3}| > |\mathcal{D}_{TD_3}| \ \wedge \ \underbrace{|\mathcal{D}_{TD'_2}|}_{\Rightarrow b' \geq d}, c' \geq d) \end{array}$$

gilt.

Zu (i): Es tritt also entweder Fall 3 ($a' < d$) mit größerem $a := a'$ oder Fall 4 ($a' \geq d$) ein.

Zu (ii): Es tritt also entweder Fall 3 (b) ($|\mathcal{D}_{TD'_3}| \geq d$) oder erneut Fall 3 (c) ($|\mathcal{D}_{TD'_3}| < d$) ein. Ist es Fall 3 (c), so ist klar, daß sich dies nicht endlos wiederholen kann, und wir somit nach endlich vielen neu berechneten Punkten $\mathbf{t}'$ bei Fall 3 (b) landen, oder im Fall 3 (c) trifft Punkt (i) zu.

(d) $|\mathcal{D}_{TD_2}| < d \quad \wedge \quad |\mathcal{D}_{TD_3}| \geq d$: Konstruktion eines Punktes $\mathbf{t}'$, so daß

$$\begin{array}{ll} & \text{(i)} \quad (a' > a \ \wedge \ b', c' \geq d) \\ \vee & \text{(ii)} \quad (|\mathcal{D}_{TD'_2}| > |\mathcal{D}_{TD_2}| \ \wedge \ |\mathcal{D}_{TD'_3}|, c' \geq d) \end{array}$$

gilt.

Die Argumentation ist analog in Fall 3 (c) geführt.

Da wir nach Fall 3 entweder erneut in Fall 3 oder in Fall 4 landen, und wir nach endlich vielen Durchläufen von Fall 3 die Anzahl der Daten in dem Teildreieck mit weniger als d Daten erhöht haben (ohne in den beiden anderen Teildreiecken unter d Daten zu sinken), ist klar, daß wir in endlich vielen Durchläufen Fall 4 erreichen.

Fall 4: $a, b, c \geq d$: Ziel ist es, hierher zu gelangen!

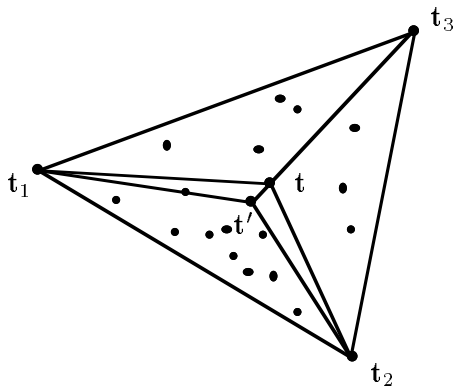
Zu Fall 1: Angenommen, es gelte also $a, b, c < d$. Dann führt

$$\begin{aligned} 3d &> |\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]}| + |\mathcal{D}_{[\mathbf{t}_1,\mathbf{t},\mathbf{t}_3]}| + |\mathcal{D}_{[\mathbf{t}_1,\mathbf{t}_2,\mathbf{t}]}| \\ &\geq |\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]} \cup \mathcal{D}_{[\mathbf{t}_1,\mathbf{t},\mathbf{t}_3]} \cup \mathcal{D}_{[\mathbf{t}_1,\mathbf{t}_2,\mathbf{t}]}| = |\mathcal{D}| \geq 3d \end{aligned}$$

sofort zum Widerspruch.

Zu Fall 2: Da in den Dreiecken $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3]$ und $[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$ jeweils zu wenig Daten liegen, wird der neue Punkt $\mathbf{t}'$ so gewählt, daß sich diese beiden Teildreiecke vergrößern. Der neue Punkt $\mathbf{t}'$ wird auf der Verlängerung der Kante $\mathbf{t}_3 \Leftrightarrow \mathbf{t}$ im Inneren des Dreiecks $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]$ liegen.

Das Teildreieck soll aber nur um ein Minimum verkleinert werden, d.h. daß alle Daten, die sich im Inneren von $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]$ befinden, wieder in $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}']$ liegen sollen. Somit muß ein Datum auf $\mathbf{t}_1 \Leftrightarrow \mathbf{t}'$ bzw. auf $\mathbf{t}_2 \Leftrightarrow \mathbf{t}'$ liegen und zu zwei Dreiecken zählen; und dadurch erhöht sich eben in einem der Teildreiecke mit zuwenig Daten deren Anzahl.



Die Bestimmung dieses Punktes $\mathbf{t}'$ wollen wir noch einmal genauer beschreiben.

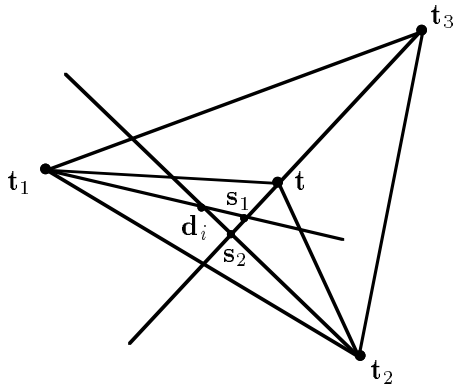
Zunächst gilt:

$$\mathcal{D}_{I_3} \neq \emptyset,$$

da sonst

$$|\mathcal{D}_{I_3} \cup \mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]} \cup \mathcal{D}_{[\mathbf{t}_1,\mathbf{t},\mathbf{t}_3]}| \leq |\mathcal{D}_{I_3}| + |\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]}| + |\mathcal{D}_{[\mathbf{t}_1,\mathbf{t},\mathbf{t}_3]}| < 0 + d + d = 2d$$

einen Widerspruch zur im Satz gemachten Voraussetzung der Anzahl an Daten im Dreieck ohne eine Außenkante darstellen würde. Zu jedem Datum $\mathbf{d}_i \in \mathcal{D}_{I_3}$ existieren nun zwei Schnittpunkte auf der Verlängerung der Kante $\mathbf{t}_3 \Leftrightarrow \mathbf{t}$ im Inneren des Dreiecks $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]$: zum einen als Schnitt mit der Geraden durch $\mathbf{d}_i$ und $\mathbf{t}_1$ und zum anderen als Schnitt mit der Geraden durch $\mathbf{d}_i$ und $\mathbf{t}_2$.



Von allen Schnittpunkten, die sich zu den jeweiligen Daten ergeben, wählen wir den als neuen Punkt $\mathbf{t}'$, der am nächsten an $\mathbf{t}$ liegt.

Aufgrund der Wahl von $\mathbf{t}'$ ergibt sich $c' = \mathcal{D}_{[\mathbf{t}_1,\mathbf{t}_2,\mathbf{t}']} = \mathcal{D}_{I_3} \cup \mathcal{D}_{\mathbf{t}_1-\mathbf{t}_2}$ und somit $c' \geq d$, denn sonst hätten wir mit

$$\begin{aligned} |\mathcal{D}| &= |\mathcal{D}_{I_3} \cup \mathcal{D}_{\mathbf{t}_1-\mathbf{t}_2} \cup \mathcal{D}_{[\mathbf{t}_1,\mathbf{t},\mathbf{t}_3]} \cup \mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]}| \\ &\leq |\mathcal{D}_{I_3} \cup \mathcal{D}_{\mathbf{t}_1-\mathbf{t}_2}| + |\mathcal{D}_{[\mathbf{t}_1,\mathbf{t},\mathbf{t}_3]}| + |\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]}| < d + d + d = 3d \end{aligned}$$

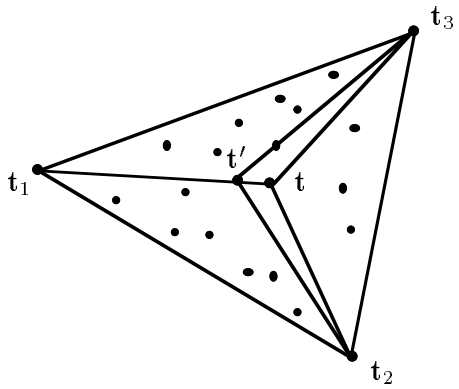
einen Widerspruch zur Gesamtmenge der Daten.

Zu Fall 3 (a): Angenommen es gelte $a, |\mathcal{D}_{TD_2}|, |\mathcal{D}_{TD_3}| < d$. Dann führt

$$\begin{aligned} 3d &> |\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]}| + |\mathcal{D}_{TD_2}| + |\mathcal{D}_{TD_3}| \\ &\geq |\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]} \cup \mathcal{D}_{TD_2} \cup \mathcal{D}_{TD_3}| = |\mathcal{D}| \geq 3d \end{aligned}$$

zum Widerspruch.

Zu Fall 3 (b): Da im Dreieck $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3]$ zu wenig Daten liegen, wird der neue Punkt $\mathbf{t}'$ so gewählt, daß sich dieses Teildreieck vergrößert. Der neue Punkt $\mathbf{t}'$ wird auf der Kante $\mathbf{t}_1 \Leftrightarrow \mathbf{t}$ liegen. Die beiden Teildreiecke $[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$ und $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3]$ sollen erneut nur um ein Minimum verkleinert werden, d.h. daß alle Daten, die sich im Inneren dieser Teildreiecke befinden (sowie auf der offenen Kante $\mathbf{t}_1 \stackrel{\circ}{-} \mathbf{t}$), wieder in $[\mathbf{t}_1, \mathbf{t}', \mathbf{t}_3]$ bzw. in $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}']$ liegen sollen. Somit muß ein Datum auf $\mathbf{t}_2 \Leftrightarrow \mathbf{t}'$ oder auf $\mathbf{t}_3 \Leftrightarrow \mathbf{t}'$ liegen und zu zwei Dreiecken zählen; und dadurch liegen nun in $[\mathbf{t}', \mathbf{t}_2, \mathbf{t}_3]$ mehr Daten als zuvor in $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3]$.



Nun zur genaueren Beschreibung der Bestimmung von $\mathbf{t}'$.

Es gilt $|\mathcal{D}_{I_2}| + |\mathcal{D}_{\mathbf{t}_1 \underline{\circ} \mathbf{t}}| > 0$ oder $|\mathcal{D}_{I_3}| + |\mathcal{D}_{\mathbf{t}_1 \underline{\circ} \mathbf{t}}| > 0$, denn gelte

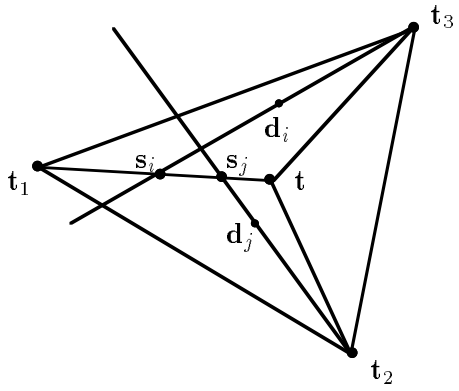
$$|\mathcal{D}_{I_2}| + |\mathcal{D}_{\mathbf{t}_1 \underline{\circ} \mathbf{t}}| + |\mathcal{D}_{I_3}| = 0,$$

so folgt mit

$$|\mathcal{D}_{I_2}| + |\mathcal{D}_{\mathbf{t}_1 \underline{\circ} \mathbf{t}}| + |\mathcal{D}_{I_3}| + |\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]}| < 0 + 0 + 0 + d = d$$

ein Widerspruch zur im Satz gemachten Voraussetzung zur Datenmenge im Dreieck ohne zwei Kanten.

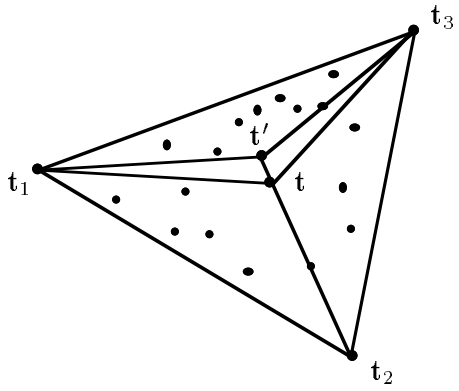
Zu jedem Datum $\mathbf{d}_i \in \mathcal{D}_{I_2}$ existiert ein Punkt $\mathbf{s}_i$ auf $\mathbf{t}_1 \underline{\circ} \mathbf{t}$ als Schnittpunkt mit der Geraden durch $\mathbf{t}_3$ und $\mathbf{d}_i$. Ebenso existiert zu jedem Datum $\mathbf{d}_j \in \mathcal{D}_{I_3}$ ein Punkt $\mathbf{s}_j$ auf $\mathbf{t}_1 \underline{\circ} \mathbf{t}$ als Schnittpunkt mit der Geraden durch $\mathbf{t}_2$ und $\mathbf{d}_j$.



Von allen Schnittpunkten und den Punkten aus $\mathcal{D}_{\mathbf{t}_1 \underline{\circ} \mathbf{t}}$ wählen wir den als neuen Punkt $\mathbf{t}'$, der am nächsten an $\mathbf{t}$ liegt.

Da durch die Wahl von $\mathbf{t}'$ nun $b' = |\mathcal{D}_{TD_2}|$ und $c' = |\mathcal{D}_{TD_3}|$ gilt, folgt sofort $b', c' \geq d$.

Zu Fall 3 (c): Da im Dreieck $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3]$ zu wenig Daten liegen, wird der neue Punkt $\mathbf{t}'$ so gewählt, daß sich dieses Teildreieck vergrößert. Im Gegensatz zum Fall 3 (b) können wir den neuen Punkt aber nicht mehr auf $\mathbf{t}_1 \underline{\circ} \mathbf{t}$ bestimmen, da $|\mathcal{D}_{TD_3}| < d$ gilt, und somit auf $\mathbf{t}_2 \Leftrightarrow \mathbf{t}$ so viele Daten liegen, daß durch einen neu bestimmten Punkt $\mathbf{t}'$ diese Kante im Dreieck $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}']$ zunächst nicht fehlen darf. Wir suchen uns den neuen Punkt also im Inneren des Dreiecks $[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$ und zwar auf der Verlängerung der Kante $\mathbf{t}_2 \Leftrightarrow \mathbf{t}$. Dabei soll $[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$ wieder nur um ein Minimum verkleinert werden, d.h. daß alle Daten, die sich im Inneren dieses Teildreiecks befinden, wieder in $[\mathbf{t}_1, \mathbf{t}', \mathbf{t}_3]$ liegen sollen. Somit muß ein Datum auf $\mathbf{t}_3 \Leftrightarrow \mathbf{t}'$ oder auf $\mathbf{t}_1 \Leftrightarrow \mathbf{t}'$ liegen und zu zwei Dreiecken zählen. Liegt dieses Datum auf $\mathbf{t}_3 \Leftrightarrow \mathbf{t}'$, so gilt $a' > a$. Liegt es hingegen auf $\mathbf{t}_1 \Leftrightarrow \mathbf{t}'$, so gilt $c' > c$ und $|\mathcal{D}_{TD'_3}| > |\mathcal{D}_{TD_3}|$.



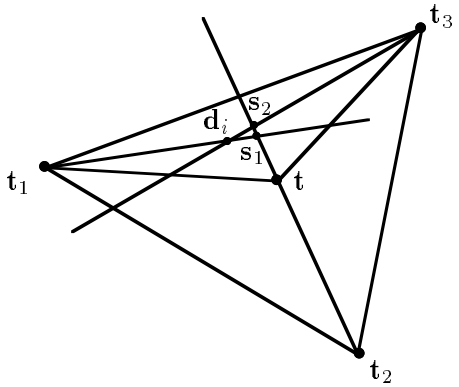
Auf die Bestimmung des Punktes $\mathbf{t}'$ wollen wir wieder genauer eingehen. Zunächst gilt:

$$\mathcal{D}_{I_2} \neq \emptyset,$$

da sonst

$$|\mathcal{D}_{I_2} \cup \mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]} \cup \mathcal{D}_{TD_3}| \leq |\mathcal{D}_{I_2}| + |\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]}| + |\mathcal{D}_{TD_3}| < 0 + d + d = 2d$$

einen Widerspruch zur im Satz gemachten Voraussetzung der Anzahl an Daten im Dreieck ohne eine Außenkante darstellen würde. Zu jedem Datum $\mathbf{d}_i \in \mathcal{D}_{I_2}$ existieren nun zwei Schnittpunkte auf der Verlängerung der Kante $\mathbf{t}_2 \Leftrightarrow \mathbf{t}$ im Inneren des Dreiecks $[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$: zum einen als Schnitt mit der Geraden durch $\mathbf{d}_i$ und $\mathbf{t}_1$ und zum anderen als Schnitt mit der Geraden durch $\mathbf{d}_i$ und $\mathbf{t}_3$ (vgl. auch Fall 2).



Von allen Schnittpunkten, die sich zu den jeweiligen Daten ergeben, wählen wir den als neuen Punkt $\mathbf{t}'$, der am nächsten an $\mathbf{t}$ liegt.

Aufgrund der Wahl von $\mathbf{t}'$ ergibt sich $b' = \mathcal{D}_{[\mathbf{t}_1,\mathbf{t}',\mathbf{t}_3]} = \mathcal{D}_{I_2} \cup \mathcal{D}_{\mathbf{t}_1-\mathbf{t}_3}$ und somit $b' \geq d$, denn sonst hätten wir mit

$$\begin{aligned} |\mathcal{D}| &= |\mathcal{D}_{I_2} \cup \mathcal{D}_{\mathbf{t}_1-\mathbf{t}_3} \cup \mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]} \cup \mathcal{D}_{TD_3}| \\ &\leq |\mathcal{D}_{I_2} \cup \mathcal{D}_{\mathbf{t}_1-\mathbf{t}_3}| + |\mathcal{D}_{[\mathbf{t},\mathbf{t}_2,\mathbf{t}_3]}| + |\mathcal{D}_{TD_3}| < d + d + d = 3d \end{aligned}$$

einen Widerspruch zur Gesamtmenge der Daten.

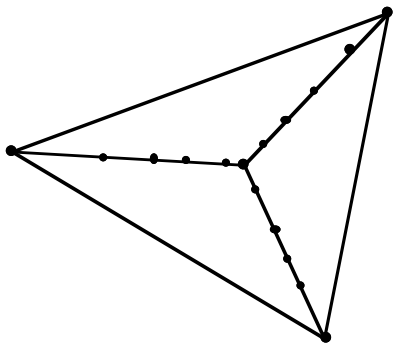
Liegt nun ein Datum $\mathbf{d}$ auf $\mathbf{t}_3 \Leftrightarrow \mathbf{t}'$ so gilt $a' > a$. Liegt kein Datum auf dieser Kante, so liegt eins auf $\mathbf{t}_1 \stackrel{o}{-} \mathbf{t}'$. Damit gilt $|\mathcal{D}_{TD'_3}| > |\mathcal{D}_{TD_3}|$. Desweiteren gilt wegen $\mathcal{D}_{\mathbf{t}_3 - \mathbf{t}'} = \emptyset$ immer noch $|\mathcal{D}_{TD'_2}| = |\mathcal{D}_{[\mathbf{t}_1, \mathbf{t}', \mathbf{t}_3]}| \geq d$.

Zu Fall 3 (d): Die Argumentation und die Konstruktion des Punktes verlaufen analog. Der neue Punkt $\mathbf{t}'$ wird hier allerdings im Inneren des Dreiecks $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]$ auf der Verlängerung der Kante $\mathbf{t}_3 \Leftrightarrow \mathbf{t}$ gesucht.

■

Bemerkung 3.40 (zu Satz 3.39)

1. Es ist im obigen Satz nicht möglich zu fordern, daß im Inneren jedes Teildreiecks d Daten liegen, auch nicht, wenn im Ausgangsdreieck keine Daten auf dem Rand liegen.



An obiger Abbildung sieht man, daß bei dem gewählten Punkt kein Teildreieck im Inneren Daten hat, aber auf dem Rand jeweils $2d = 8$. Würde man einen anderen Punkt im Inneren eines Teildreiecks wählen, so hätte man in einem Dreieck gar keine Daten. Ein Verschieben des Punkts auf dem Rand eines Teildreiecks würde das Problem offensichtlich auch nicht lösen.

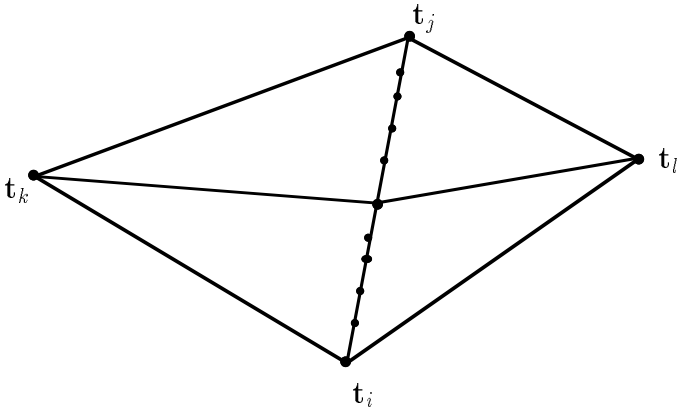
2. Im obigen (konstruktiven) Beweis wurden Daten auf dem Rand der Teildreiecke beiden selbigen zugeordnet. Darauf kann man verzichten und stattdessen eine geschickte eindeutige Zuordnung vornehmen, was aus Gründen des Erhalts der Übersichtlichkeit des Beweises nicht gemacht wurde. Bei der Implementierung des unten angegebenen zugehörigen Algorithmus C sei allerdings dazu geraten. Das kann zu weiteren Iterationen führen, da durch die eindeutige Zuordnung die Anzahl der Daten in manchen Teildreiecken sinken kann.

3. Die Voraussetzungen an die Lage der Daten im obigen Satz haben bei Scattered-Data-Problemen lediglich theoretischen Charakter, denn Kanten sind im $\mathbb{R}^2$ Nullmengen, und die Wahrscheinlichkeit, daß mehrere Daten kollinear und zudem noch auf einer Kante liegen, darf als „verschwindend gering“ angenommen werden.

4. Auch bei regelmäßig verteilten Daten (z.B. Spurdaten) sind diese Sonderfälle (bei einer hohen Anzahl von Testdurchläufen) nie aufgetreten.

Falls die Sonderfälle dennoch auftreten, so wäre, wie wir gleich sehen werden, die Menge der Daten in einem Dreieck trotzdem durch eine von d abhängige Konstante beschränkt.

Beispiel 3.41 (Sonderfälle)
Liegen auf einer Kante $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j$ eines Dreiecks $[\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k]$ $2d$ Daten, so ist es leicht einen Punkt $\mathbf{t}$ zu finden, der die Kante so unterteilt, daß zu beiden Seiten von $\mathbf{t}$ je d Daten liegen. (Dies erledigt *Algorithmus B*, falls die Genauigkeit bei der Punktsuche, die durch N festgelegt wurde, genügend groß ist – vgl. Schritt 3 in Bemerkung 3.36.) Diesen Punkt könnten wir in die Triangulierung einfügen (mit den zugehörigen Verbindungen), und sie würde immer noch alle Anforderungen erfüllen.



Unter vorheriger Verwendung von Algorithmus B können wir also davon ausgehen, daß auf zwei Kanten je maximal $2d \Leftrightarrow 1$ Daten liegen; im Rest des Dreiecks können nach den Voraussetzungen im Satz noch maximal $d \Leftrightarrow 1$ Daten liegen, damit kein Punkt $\mathbf{t}$ im Inneren gefunden wird. Im schlechtesten Fall liegen also $5d \Leftrightarrow 3$ Daten in einem Dreieck.

Entfallen die Voraussetzungen an die Lage der Daten in Satz 3.39, und läßt man im Gegenzug den zu berechnenden Punkt $\mathbf{t}$ auch auf dem Rand des Dreiecks zu, so gilt der Satz auch, und es könnten entartete Dreiecke auftreten.

Korollar 3.42 (Satz 3.39 mit entarteten Dreiecken)
Gegeben sei ein Dreieck $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$, das eine Datenmenge $\mathcal{D} = \{\mathbf{d}_1, ..., \mathbf{d}_l\}$ mit $l \geq 3d$ beinhaltet. Dann existiert ein Punkt $\mathbf{t} \in [\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$, so daß in den drei entstandenen (eventuell entarteten) Dreiecken je (mind.) d Daten liegen.

Beweis:
Der Beweis ist analog zu führen wie bei Satz 3.39. An vier Stellen in dem Beweis wurde ein Punkt $\mathbf{t}'$ bestimmt. Als Hilfsmittel dazu verwendeten wir Teilmengen von $\mathcal{D}$, die im Inneren eines Teildreiecks $[\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}]$ bzw. im Inneren der Vereinigung zweier Teildreiecke $[\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}] \cup [\mathbf{t}_k, \mathbf{t}_i, \mathbf{t}]$ lagen. Die gemachten Voraussetzungen benötigten wir an diesen Stellen, um zu zeigen, daß im Inneren Daten liegen. Zum Beweis unseres Korollars nimmt man nun an diesen Stellen noch die Menge $\mathcal{D}_{\mathbf{t}_i - \mathbf{t}_j}$ bzw. $\mathcal{D}_{\mathbf{t}_i - \mathbf{t}_j} \cup \mathcal{D}_{\mathbf{t}_k - \mathbf{t}_i}$ hinzu, und somit benötigt man die beiden zusätzlichen Voraussetzungen nicht mehr. Schnittpunkte können dann auch auf $\mathbf{t}_i \Leftrightarrow \mathbf{t}_j$ oder $\mathbf{t}_k \Leftrightarrow \mathbf{t}_i$ liegen, und man erhält entartete Dreiecke.

■

Kommen wir nun zur Beschreibung von Algorithmus C. Wir gehen zur Vereinfachung wieder von einer geordneten Triangulierung aus (vgl. Algorithmus B), d.h. mit steigenden Indizes der Elemente der Triangulierung sinkt die Anzahl der enthaltenen Daten. Konnten wir einen Punkt hinzunehmen, d.h. ein Dreieck der Triangulierung wurde durch die entstandenen drei Teildreiecke ersetzt, so folgt wieder eine Optimierung der Triangulierung. Diesmal ist das Ergebnis des Einfügens (im Gegensatz zu Algorithmus B) immer das gleiche wie bei Cline und Renka. Somit kann Algorithmus 3.37 ohne Schritt 0 übernommen werden.

Algorithmus 3.43 (Algorithmus C)
Gegeben sei eine Triangulierung $\mathcal{T}(\mathcal{S}) = \{T_1, ..., T_m\}$ der Menge $\mathcal{S} \subset \mathbb{R}^2$, sowie zugehörige Datenmengen $\mathcal{D}_1, ..., \mathcal{D}_m \subset \mathcal{D} \subset \mathbb{R}^2$. Dabei gelte

$$|\mathcal{D}_1| \geq ... \geq |\mathcal{D}_m| \geq d \qquad \textit{sowie} \qquad [T_i] \cap \mathcal{D} = \mathcal{D}_i \ (i = 1, ..., m).$$

Gesucht ist eine feinere Triangulierung $\mathcal{T}(\mathcal{S}') = \{T_1, ..., T_{m'}\}$ der Menge $\mathcal{S}' \supset \mathcal{S}$, so daß immer noch

$$|\mathcal{D}_i| = |[T_i] \cap \mathcal{D}| \geq d \ (i = 1, ..., m')$$

gilt. Desweiteren soll die Verfeinerung genau so stark sein, daß kein Dreieck der Triangulierung mehr als $3d \Leftrightarrow 1$ Daten enthält, es sei denn, es handelt sich um einen der beiden Sonderfälle in Satz 3.39.

Schritt 1: *Setze $j := 1$.*

Schritt 2: *Gilt $|\mathcal{D}_j| < 3d$ oder hat die Triangulierung weniger als j Elemente, beende den Algorithmus.*

Schritt 3: *Gelten die beiden Anforderungen aus Satz 3.39 an die Lage der Daten nicht, so setze* $j := j + 1$ *und gehe zu* **Schritt 2**.

Ansonsten bestimme einen Punkt $\mathbf{t}$ *im Inneren von* $[T_j]$*, so daß in jedem der drei entstehenden Teildreiecke mindestens* d *Daten liegen (vgl. Algorithmus 3.44). Ersetze in* $\mathcal{T}(\mathcal{S})$ *das* j*-te Element durch die drei neu entstandenen. Achte dabei auf die Einhaltung der Ordnung*

$$|\mathcal{D}_1| \geq ... \geq |\mathcal{D}_m| \geq d \qquad mit \qquad \mathcal{D}_i = [T_i] \cap \mathcal{D} \ (i = 1, ..., m).$$

Schritt 4: *Optimiere die neu erhaltene Triangulierung (durch Algorithmus 3.37 ohne Schritt 0), setze* $j := 1$ *und gehe dann zu* **Schritt 2**. *(Dabei soll die optimierte Triangulierung wieder geordnet sein, d.h. es gelte weiter* $|\mathcal{D}_i| \geq |\mathcal{D}_{i+1}|$*.)*

Die Bestimmung des Punktes $\mathbf{t}$ gemäß Schritt 3 kann einfach aus dem Beweis zu Satz 3.39 abgeleitet werden.

Algorithmus 3.44 (zu Schritt 3 in Algorithmus 3.43)
Gegeben sei ein Dreieck $[T_j] = [\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}]$ *einer Triangulierung* $\mathcal{T}(\mathcal{S})$*, das eine Datenmenge* $\mathcal{D}_j = \{\mathbf{d}_1, ..., \mathbf{d}_l\}$ *mit* $l \geq 3d$ *beinhaltet*

$$\mathcal{D}_j \cap [\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}] = \mathcal{D}_j.$$

Bestimme einen Punkt $\mathbf{t}$ *im Inneren des Dreiecks, so daß in den drei entstandenen Dreiecken je (mindestens)* d *Daten liegen. Setze zur Vereinfachung zunächst:*

$$\begin{array}{l} [T_{j_1}] := [\mathbf{t}, \mathbf{t}_{j_2}, \mathbf{t}_{j_3}], \\ [T_{j_2}] := [\mathbf{t}_{j_1}, \mathbf{t}, \mathbf{t}_{j_3}] \ \textit{und} \\ [T_{j_3}] := [\mathbf{t}_{j_1}, \mathbf{t}_{j_2}, \mathbf{t}]. \end{array}$$

Schritt 1: *Setze* $\mathbf{t} := 1/3 \cdot (\mathbf{t}_{j_1} + \mathbf{t}_{j_2} + \mathbf{t}_{j_3})$.

Schritt 2: *Gilt* $|\mathcal{D}_j \cap [T_{j_i}]| \geq d \ \forall i \in \{1, 2, 3\}$*, so breche ab.*

Schritt 3: *Gilt* $|\mathcal{D}_j \cap [T_{j_i}]| \geq d$ *für genau zwei Indices* $i \in \{1, 2, 3\}$*, so setze diese zu* i_1 *und* i_2 *(also* $(i_1, i_2) \in \{(1, 2), (2, 3), (3, 1)\}$*), und gehe zu* **Schritt 5**.

Schritt 4: *Gilt* $|\mathcal{D}_j \cap [T_{j_i}]| \geq d$ *für genau einen Index* $i \in \{1, 2, 3\}$*, so setze diesen zu* μ*, und gehe zu* **Schritt 7**.

Schritt 5: *Gilt* $|\mathcal{D}_j \cap ([T_{j_{i_1}}] \setminus (\mathbf{t} \stackrel{o}{-} \mathbf{t}_{j_{i_2}} \cup \{\mathbf{t}\}))| < d$*, so setze* $\mu := i_1$*, und gehe zu* **Schritt 7**.

Schritt 6: *Gilt* $|\mathcal{D}_j \cap ([T_{j_{i_2}}] \setminus (\mathbf{t} \stackrel{o}{-} \mathbf{t}_{j_{i_1}} \cup \{\mathbf{t}\}))| < d$*, so setze* $\mu := i_2$*, und gehe zu* **Schritt 7**.

Ansonsten setze $\mu := i_1$ *und* $\nu := i_2$*, und gehe zu* **Schritt 8**.

Schritt 7: *Berechne zu jedem Datum*

$$\mathbf{d}_i \in \mathcal{D}_j \cap ([T_{j_\mu}] \setminus \partial[T_{j_\mu}])$$

zwei Punkte, auf der Verlängerung der Kante $\mathbf{t}_{i_\mu} \Leftrightarrow \mathbf{t}$ *im Inneren des Dreiecks* $[T_{j_\mu}]$. *Diese ergeben sich als Schnittpunkte mit den beiden Geraden durch* $\mathbf{d}_i$ *und* $\mathbf{t}_{j_k} (k \in \{1,2,3\} \setminus \{\mu\})$.

Von allen so berechneten Punkten wähle den als neuen Punkt $\mathbf{t}$, *der den kürzesten Abstand zum bisherigen Punkt* $\mathbf{t}$ *hat. (Vergleiche z.B. Fall 2 im Beweis zu Satz 3.39.) Gehe zu* **Schritt 2**.

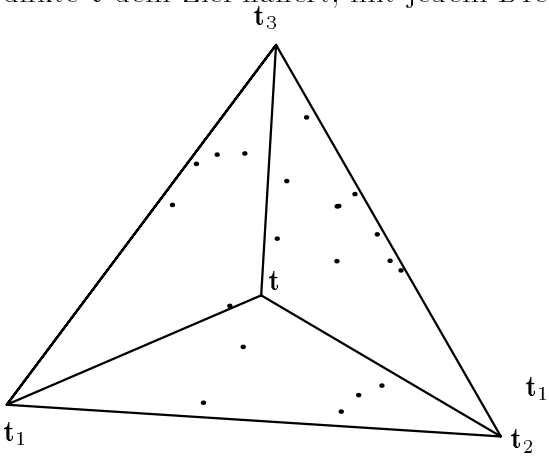
Schritt 8: *Berechne zu jedem Datum*

$$\mathbf{d}_i \in \mathcal{D}_j \cap (([T_{j_\mu}] \cup [T_{j_\nu}]) \setminus \partial([T_{j_\mu}] \cup [T_{j_\nu}]))$$

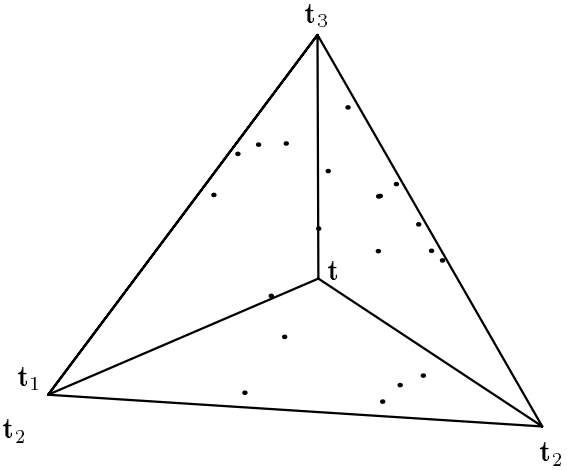
einen Punkt auf der Kante $\mathbf{t}_{j_k} \Leftrightarrow \mathbf{t}$ *mit eindeutigem* $k \in \{1,2,3\} \setminus \{\mu,\nu\}$. *Diese ergeben sich im Fall* $\mathbf{d}_i \in [T_{j_\mu}] \setminus \partial[T_{j_\mu}]$ *als Schnittpunkt mit der Gerade durch* $\mathbf{d}_i$ *und* $\mathbf{t}_{j_\nu}$, *im Fall* $\mathbf{d}_i \in [T_{j_\nu}] \setminus \partial[T_{j_\nu}]$ *als Schnittpunkt mit der Gerade durch* $\mathbf{d}_i$ *und* $\mathbf{t}_{j_\mu}$ *und im Fall* $\mathbf{d}_i \in \mathbf{t}_{i_k} \stackrel{o}{-} \mathbf{t}$ *(*k *wie oben) als* $\mathbf{d}_i$ *selbst.*

Von allen so berechneten Punkten wähle den als neuen Punkt $\mathbf{t}$, *der den kürzesten Abstand zum bisherigen Punkt* $\mathbf{t}$ *hat. (Vergleiche Fall 3 (c) im Beweis zu Satz 3.39.) Gehe zu* **Schritt 2**.

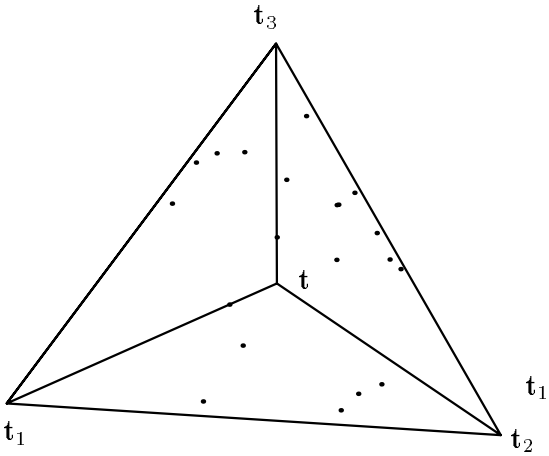
Beispiel 3.45 (zu Algorithmus 3.44)
An diesem Beispiel wird deutlich, wie man sich durch die jeweils neu bestimmten Punkte $\mathbf{t}$ dem Ziel nähert, mit jedem Dreieck $d = 6$ Daten zu überdecken.



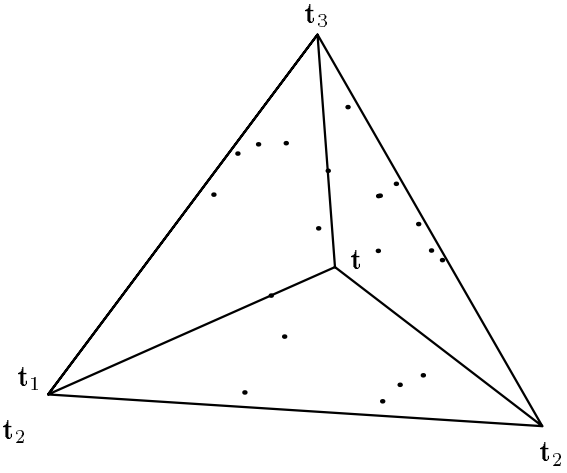
Der erste Punkt $\mathbf{t}$ wurde initialisiert. $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]$ und $[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$ beinhalten je 5 Daten und $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3]$ 10. $\mathbf{t}$ wird auf der Verlängerung der Kante $\mathbf{t}_1 - \mathbf{t}$ verschoben.



Ein Datum liegt auf der Kante $\mathbf{t}_3 - \mathbf{t}$. Somit enthält $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]$ 5, $[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$ 6 und $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3]$ 10 Daten. $\mathbf{t}$ wird auf der Kante $\mathbf{t}_3 - \mathbf{t}$ verschoben.



Auf den Kanten $\mathbf{t}_1 - \mathbf{t}$ und $\mathbf{t}_3 - \mathbf{t}$ liegt je ein Datum. Somit enthält $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]$ 6, $[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$ 6 und $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3]$ 10 Daten. Ordnet man die Daten eindeutig zu, kann man aber nicht jedem Teildreieck 6 zuordnen. $\mathbf{t}$ wird auf der Verlängerung der Kante $\mathbf{t}_1 - \mathbf{t}$ verschoben.



Auf den Kanten $\mathbf{t}_1 - \mathbf{t}$ und $\mathbf{t}_3 - \mathbf{t}$ liegt je ein Datum. Somit enthält $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}]$ 6, $[\mathbf{t}_1, \mathbf{t}, \mathbf{t}_3]$ 7 und $[\mathbf{t}, \mathbf{t}_2, \mathbf{t}_3]$ 9 Daten. Eine eindeutige Zuordnung ist möglich.

3.4 Korrektheit

In diesem Kapitel wollen wir zeigen, daß das in der Einleitung vorgestellte Problem 1.1 durch die Algorithmen A, B und C gelöst wird. Dabei seien die in Abschnitt 3.3.3 hinlänglich besprochenen Sonderfälle ausgenommen. Es soll nach Ablauf von Algorithmus C also eine Triangulierung bestimmt worden sein, die folgende Anforderungen erfüllt:

- **(A1)** Das triangulierte Gebiet beinhaltet alle Daten.
- **(A2)** Jedes Dreieck der Triangulierung enthält mindestens $d \in \mathbb{N}$ Daten.
- **(A3)** Jedes Dreieck enthält höchstens $3d \Leftrightarrow 1$ Daten.

Die vierte Forderung nach Vermeidung langer spitzer Dreiecke ist durch keine numerische Vorgabe – wie z.B. einem Minimalwert bzgl. der Innenwinkel, der nicht unterschritten werden darf – weiter spezifiziert und ist somit nicht beweisbar. Durch die ausschließliche Verwendung von Triangulierungen, die bzgl. des erweiterten Max-Min-Winkelkriteriums (vgl. Definition 3.7) lokal optimal sind, wird diese Forderung aber genügend berücksichtigt. Die lokale Optimalität erreichen wir in Algorithmus A durch die ausschließliche Verwendung von Delaunay-Triangulierungen und in den Algorithmen B und C durch die Verwendung eines Optimierungsalgorithmus.

Satz 3.46 (Korrektheit von Algorithmus A)
Algorithmus A endet nach endlich vielen Schritten und liefert bei Angabe einer Datenmenge $\mathcal{D} = \{\mathbf{d}_1, ..., \mathbf{d}_l\} \subset \mathbb{R}^2$*, wobei nicht alle Daten aus* $\mathcal{D}$ *kollinear liegen, und bei Angabe einer unteren Schranke* d*, mit* $d \leq l$*, eine Menge* $\mathcal{S} \subset \mathbb{R}^2$ *und eine Triangulierung* $\mathcal{T}(\mathcal{S})$*, so daß die Triangulierung mit jedem Dreieck mindestens* d *Daten* ($d \in \mathbb{N}$) *aus* $\mathcal{D}$ *überdeckt.*

Beweis: Die Menge $\mathcal{S}_A$ wird in Schritt 1 als konvexes Polygon bestimmt, in dem alle Daten $\mathcal{D}$ liegen. Diese Menge kann nur in Schritt 12 verändert werden. Dort bleiben die genannten Eigenschaften weiter erhalten. Da alle Punkte aus $\mathcal{S}_I$ Konvexkombinationen von Daten aus $\mathcal{D}$ sind, liegen auch sie in der konvexen Hülle von $\mathcal{S}_A$. Die Menge $\mathcal{S}$ wird somit aus Außen- und Innenpunkten gebildet.

Aufgrund dieser Geometrie von $\mathcal{S}$ und $\mathcal{D}$ liegen in jeder berechneten Triangulierung $\mathcal{T}(\mathcal{S})$ alle Daten $\mathcal{D}$, und Anforderung (A1) ist erfüllt.

Die Variable tria beinhaltet immer die letzte berechnete Triangulierung, die in jedem Dreieck genügend Daten beinhaltete. Somit ist zu zeigen, daß

- tria irgendwann mit einer Triangulierung (außer der leeren Menge) belegt wird
- der Algorithmus endet.

Angenommen es würde nie eine Triangulierung gefunden, die in jedem Dreieck d Daten hat, so würden zunächst die Schritte 1, 2, 3 und 5 durchlaufen. Nach den Schritten 6, 7 und 8 würde in Schritt 10 die Zählvariable *durchlauf* erhöht. Es kämen noch zweimal die Schritte 6, 7, 8 und 10, bis wir über Schritt 6 zu Schritt 11 gelangen würden, da dreimal hintereinander in Schritt 7 eine Triangulierung bestimmt wurde, die die Anforderung an die Minimalanzahl an Daten nicht in jedem Dreieck erfüllen konnte. In Schritt 12 würde dann die Anzahl an Punkten in $\mathcal{S}_A$ um eins reduziert derart, daß weiterhin

$$[\mathcal{S}_A] \cap \mathcal{D} = \mathcal{D}$$

gilt, und wir kämen erneut zu Schritt 2. Auf dem gleichen Weg (durch die Schritte des Algorithmus) kämen wir immer wieder zu Schritt 11, bis die Anzahl an Punkten in $\mathcal{S}_A$ von 4 auf 3 reduziert würde. Die nächste berechnete Triangulierung bestünde nur aus einem Dreieck und würde alle l Daten beinhalten und aufgrund von $l \geq d$ somit angenommen. Damit ergibt sich der gewollte Widerspruch.

Es bleibt lediglich zu zeigen, daß der Algorithmus nach endlich vielen Schritten endet. Haben wir einmal eine passende Triangulierung gefunden, so werden nur noch Punkte der Menge $\mathcal{S}_I$ zugefügt und geprüft, ob weiterhin $\mathcal{T}(\mathcal{S}) = \mathcal{T}(\mathcal{S}_A \cup \mathcal{S}_I)$

in jedem Dreieck genügend Daten hat. Ist dies nicht der Fall, wird die Zählvariable *durchlauf* um eins erhöht. Gilt

$$durchlauf = 4,$$

so wird in Schritt 11 abgebrochen. Ein Abbruch des Algorithmus wäre also nur zu verhindern, falls immer wieder neue Triangulierungen mit folglich immer mehr Dreiecken gefunden würden. Da die Anzahl der Daten aber beschränkt ist, und jedes Dreieck ein Minimum derer enthalten muß, ist ein Abbruch des Algorithmus garantiert.

■

Satz 3.47 (Korrektheit von Algorithmus B)
Algorithmus B endet nach endlich vielen Schritten und liefert bei Eingabe einer Triangulierung $\mathcal{T}(\mathcal{S}) = \{T_1, ..., T_m\}$, die in jedem Dreieck mindestens d Daten aus $\mathcal{D}$ beinhaltet eine Triangulierung $\mathcal{T}(\mathcal{S}') = \{T_1, ..., T_{m'}\}$ $(m' \geq m)$, die weiterhin in jedem Dreieck minimal d Daten enthält.

Beweis: Eine neue Triangulierung kann nur in Schritt 3 durch Einfügen eines neuen Punktes $\mathbf{t}$ entstehen, oder in Schritt 5 durch eine anschließende Optimierung der neuen Triangulierung. Wir werden noch sehen, daß Algorithmus 3.35, der Schritt 3 näher beschreibt, korrekt ist, d.h. daß er terminiert und eine Triangulierung liefert, die weiterhin die Anforderungen (A1) und (A2) erfüllt. Die Korrektheit der Optimierung ergibt sich aus der Verwendung des erweiterten Max-Min-Winkelkriteriums (vgl. Definition 3.7).

Zunächst wollen wir noch zeigen, daß Algorithmus B terminiert. Die Variable j bezeichnet die Position des zu untersuchenden Dreiecks in der geordneten Liste aller Dreiecke der Triangulierung. Mit steigender Position j fällt die Anzahl an Daten $|\mathcal{D}_j|$ im Dreieck $[T_j]$.

In Schritt 3 steigt entweder die Anzahl der Dreiecke der Triangulierung, falls ein passender Punkt $\mathbf{t}$ gefunden wurde, oder j wird erhöht. Da die Anzahl der Daten beschränkt ist, können nicht unendlich viele passende Punkte gefunden werden, denn ein Punkt $\mathbf{t}$ ist nur *passend*, falls die neue Triangulierung $\mathcal{T}(\mathcal{S} \cup \{\mathbf{t}\})$ wieder die Anforderung (A2) erfüllt. Somit endet der Algorithmus, falls er nicht irgendwann wegen $|\mathcal{D}_j| < 2d$ beendet wird, aufgrund der Tatsache, daß j über die Anzahl der Dreiecke der Triangulierung steigt.

Es bleibt also noch die Korrektheit von Algorithmus 3.35 zu zeigen. Die Schritte 1 bis 4 werden maximal dreimal durchlaufen und Schritt 5 genau einmal. Danach kehren wir zurück in den eigentlichen Algorithmus B. Wurde kein Punkt $\mathbf{t}$ gefunden, so wird, wie erforderlich, $\{\mathbf{t}\} := \emptyset$ gesetzt. Wurde ein Punkt $\mathbf{t}$ gefunden, so enthalten die neu entstandenen Dreiecke wieder mindestens d Daten, was Schritt 2 bzw. Schritt 3 in Algorithmus 3.35 garantieren. Somit erfüllt die neu erhaltene

Triangulierung Anforderung (A2). Anforderung (A1) ist erfüllt, da der triangulierte Bereich nicht verkleinert wird.

■

Satz 3.48 (Korrektheit von Algorithmus C)
Algorithmus C endet nach endlich vielen Schritten und liefert bei Eingabe einer Triangulierung, die die Anforderungen (A1) *und* (A2) *erfüllt, eine Triangulierung, die die Anforderungen* (A1), (A2) *und* (A3) *erfüllt. (Ausgenommen sind hierbei die Sonderfälle; vgl. Beispiel 3.41.)*

Beweis: Die Variable j bezeichnet erneut die Position des zu untersuchenden Dreiecks in der geordneten Liste aller Dreiecke der Triangulierung. Mit steigender Position j fällt die Anzahl an Daten $|\mathcal{D}_j|$ im Dreieck $[T_j]$.

Durch die Abfrage in Schritt 3 bzgl. der Anforderungen aus Satz 3.39 schließen wir das Vorkommen der Sonderfälle aus, und wir gelangen zu dem Dreieck der Triangulierung mit den meisten Daten, das keinen Sonderfall darstellt und mindestens $3d$ Daten beinhaltet. Existiert kein solches, wird bereits in Schritt 2 abgebrochen.

Die Korrektheit der Bestimmung des Punktes $\mathbf{t}$ unter Schritt 3 folgt aus dem Beweis zu Satz 3.39.

Da nur endlich viele Daten existieren, können nur endlich viele neue Punkte bestimmt werden, bis in der Triangulierung kein Dreieck mehr vorkommt, das mehr als d Daten beinhaltet und keinen Sonderfall darstellt. Somit ist ein Abbruch des Algorithmus und die Erfüllung von Anforderung (A3) garantiert.

Daß Anforderung (A1) weiterhin gilt, ist gesichert dadurch, daß sich der triangulierte Bereich nicht verkleinert.

Anforderung (A2) ergibt sich aus Satz 3.39 sowie der Verwendung des erweiterten Max-Min-Winkelkriteriums bei der Optimierung.

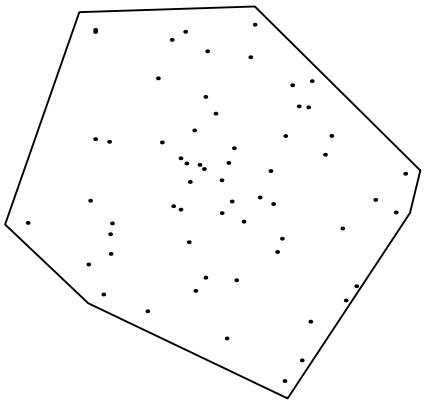
■

3.5 Beispiele zur Triangulierung

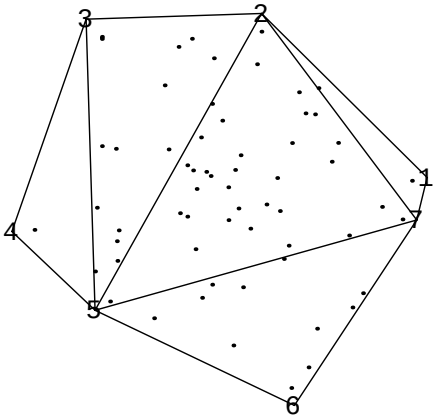
In diesem letzten Kapitel werden einige Versuche mit dem Triangulierungsverfahren illustriert. Sie sollen zum einen die Arbeitsweise verdeutlichen, zum anderen erklären, warum eine Splittung in drei Teilalgorithmen sinnvoll – für gute Ergebnisse gar notwendig ist. Desweiteren sollen schlicht Ergebnisse demonstriert werden.

Beispiel 3.49 (Ablauf von Algorithmus A)
Das Beispiel zeigt detailliert den Ablauf von Algorithmus A. Man wird sehen, daß zum einen die konvexe Hülle des triangulierten Bereichs verändert werden mußte,

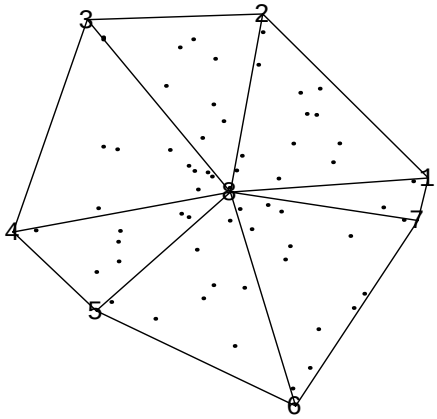
um eine passende Triangulierung zu finden, zum anderen durch veränderte und neue Punkte im Inneren (durch den Clustering-Algorithmus) die Triangulierung mal den Anforderungen (an eine Minimalanzahl an Daten pro Dreieck) entsprach und mal nicht. Schön zu erkennen ist auch, wie durch den Clustering-Algorithmus jeweils ein Punkt durch zwei neue ersetzt wird. Es wurde durch Zufallszahlen ein Datensatz mit $n = 60$ Daten bestimmt, und $d = 3$ Daten pro Dreieck wurden verlangt.



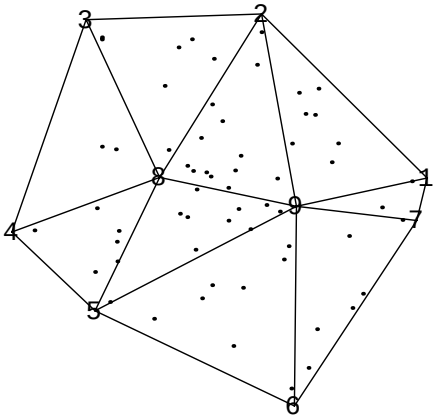
(die konvexe Hülle wurde in Schritt 1 bestimmt)



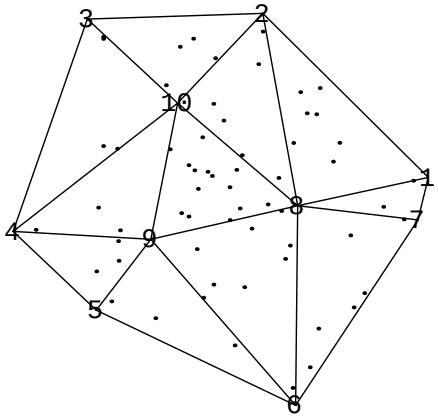
(nach Schritt 2, 3, 5: die Triangulierung wurde z.B. wegen Dreieck [1,2,7] abgelehnt, tria:=∅, dl:=durchlauf:=1)



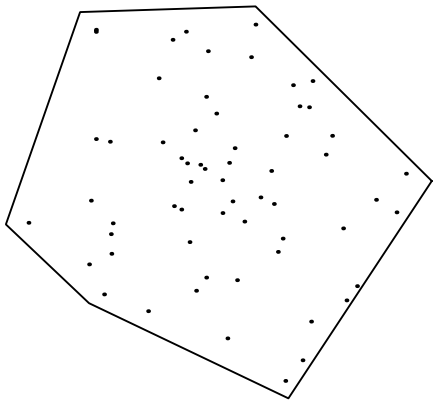
(nach Schritt 6, 7, 8, 10: die Triangulierung wurde wegen Dreieck [1,8,7] abgelehnt, tria=∅, dl:=2)



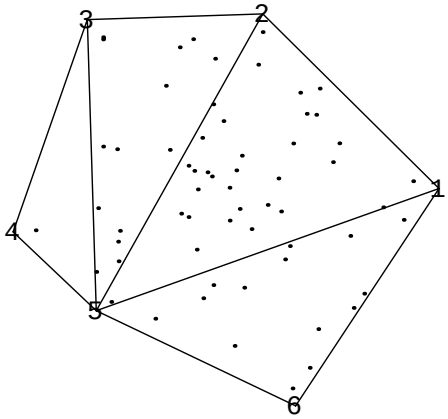
(nach Schritt 6, 7, 8, 10: die Triangulierung wurde wegen Dreieck [1,9,7] abgelehnt, tria=∅, dl:=3)



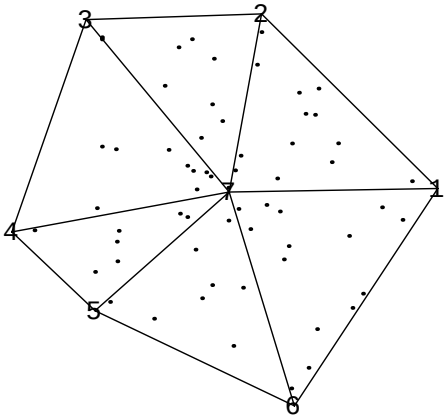
(nach Schritt 6, 7, 8, 10: die Triangulierung wurde wegen Dreieck [1,8,7] abgelehnt, tria=$\emptyset$, dl:=4)



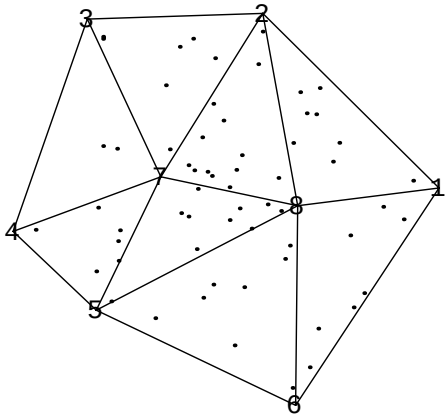
(nach Schritt 6, 11, 12: die Anzahl der Außenpunkte wurde reduziert)



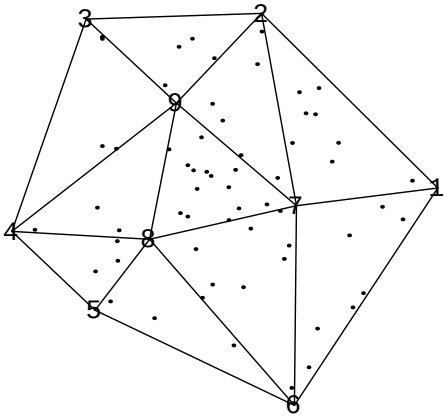
(nach Schritt 2, 3, 5: die Triangulierung wurde wegen Dreieck [3,4,5] abgelehnt, tria:=$\emptyset$, dl:=1)



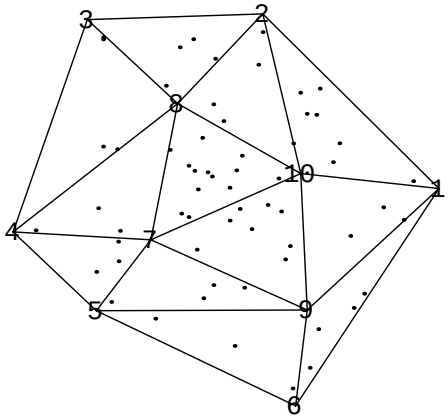
(nach Schritt 6, 7, 8, 9: die Triangulierung wurde angenommen, dl=1)



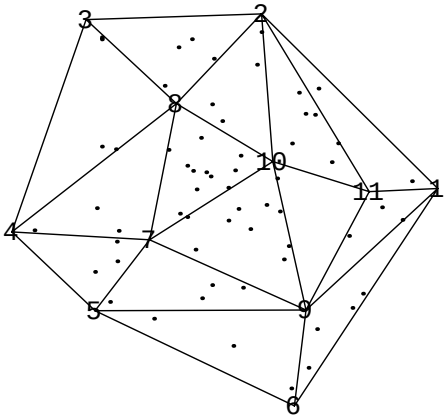
(nach Schritt 6, 7, 8, 10: die Triangulierung wurde wegen Dreieck [3,4,7] abgelehnt, dl:=2)



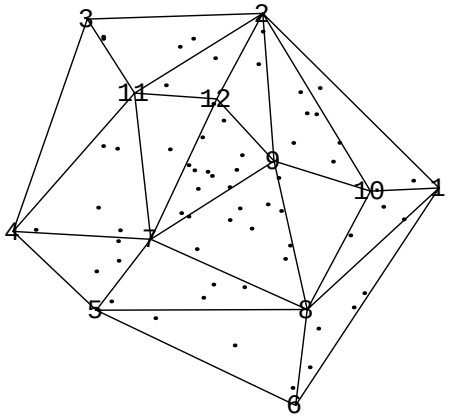
(nach Schritt 6, 7, 8, 9: die Triangulierung wurde angenommen, dl:=1)



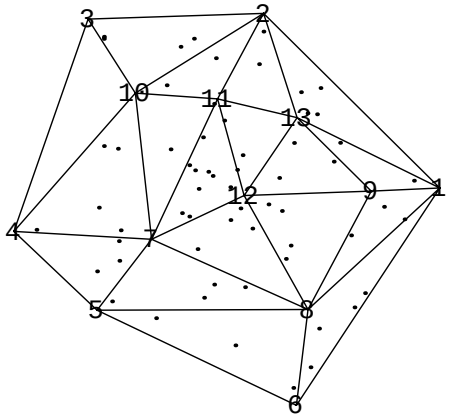
(nach Schritt 6, 7, 8, 9: die Triangulierung wurde angenommen, dl=1)



(nach Schritt 6, 7, 8, 10: die Triangulierung wurde z.B. wegen Dreieck [9,11,10] abgelehnt, dl:=2)

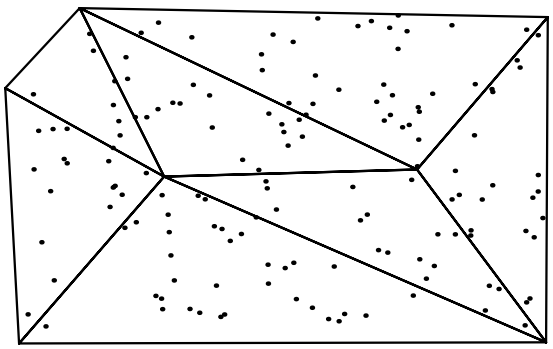


(nach Schritt 6, 7, 8, 10: die Triangulierung wurde z.B. wegen Dreieck [8,10,9] abgelehnt, dl:=3)

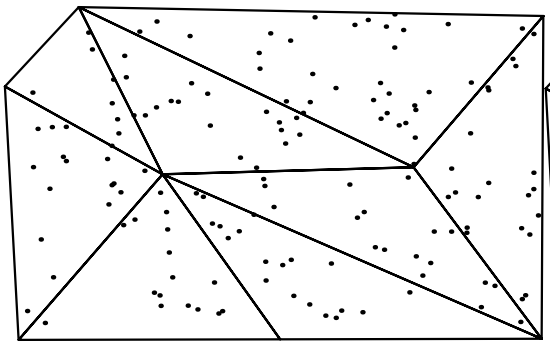


(nach Schritt 6, 7, 8, 10, 6, 11: die Triangulierung wurde z.B. wegen Dreieck [4,10,3] abgelehnt, dl:=4, Stop des Algorithmus)

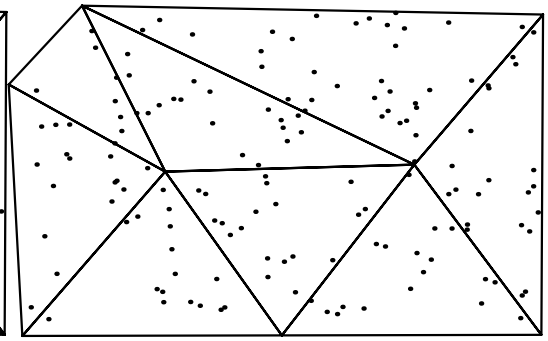
Beispiel 3.50 (Ablauf von Algorithmus B)
Das Verfahren der weiteren Punktbestimmung zur Verfeinerung der Triangulierung aus Algorithmus B wird an diesem Beispiel illustriert. Erneut wurden die 150 Daten mit Hilfe von Zufallszahlen erzeugt, und es wurden minimal $d = 6$ Daten pro Dreieck gefordert. Algorithmus B hat schon eine Maximalanzahl von 12 Daten in einem Dreieck erreicht. Zunächst wird das Ergebnis von Algorithmus A präsentiert und im Anschluß die Triangulierungen mit den jeweils neu eingefügten Punkten vor und nach der Optimierung.



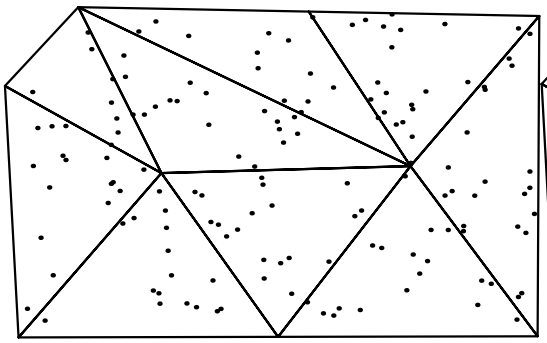
(die Triangulierung nach Algorithmus A)



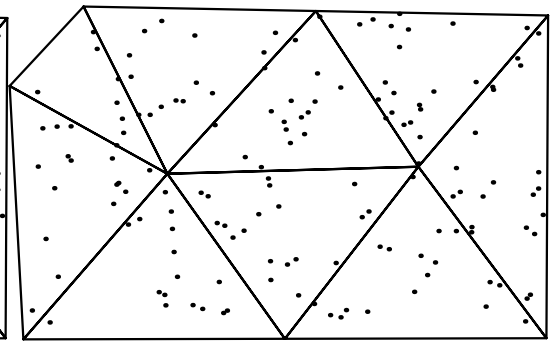
(ein neuer Punkt wurde eingefügt)



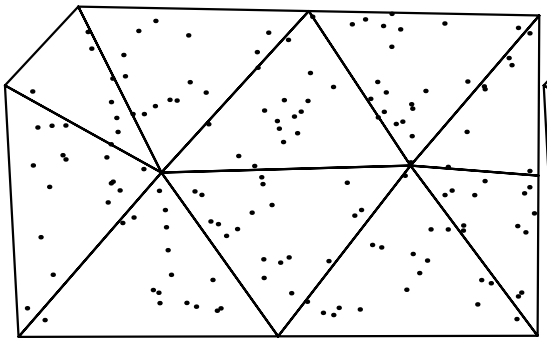
(nach der Optimierung)



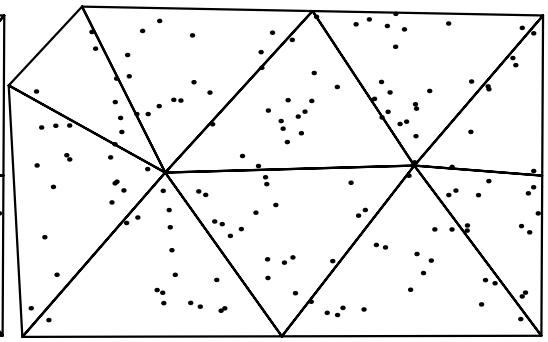
(ein neuer Punkt wurde eingefügt)



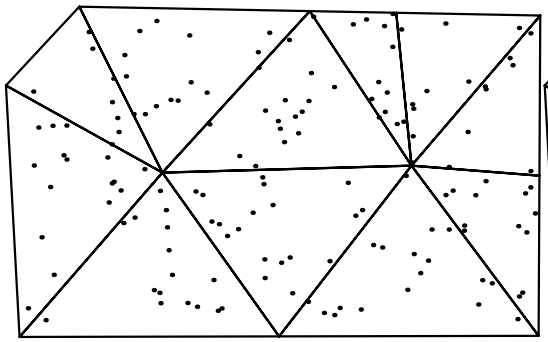
(nach der Optimierung)



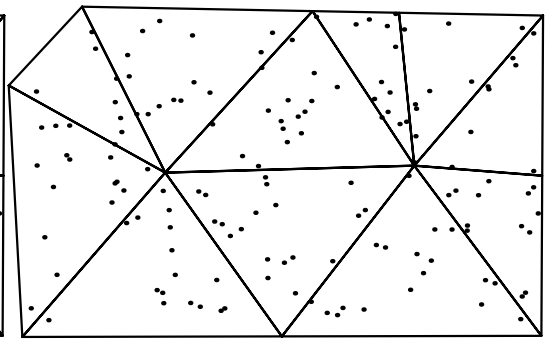
(ein neuer Punkt wurde eingefügt)



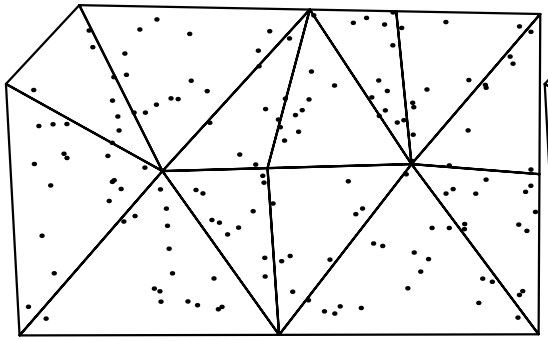
(kein Optimierungserfolg)



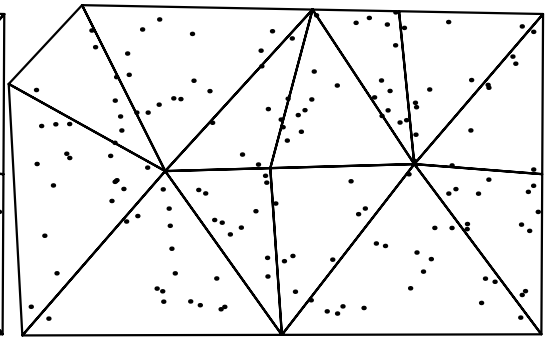
(ein neuer Punkt wurde eingefügt)



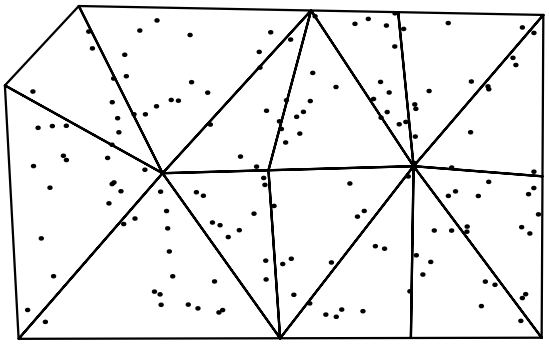
(kein Optimierungserfolg)



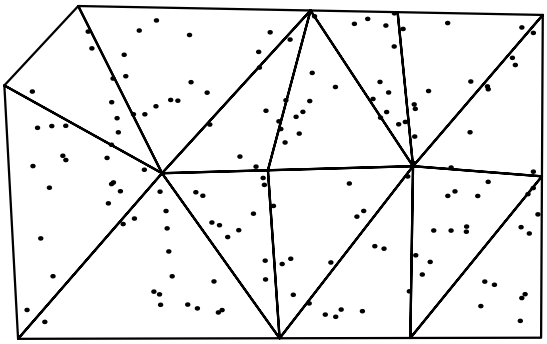
(ein neuer Punkt wurde eingefügt)



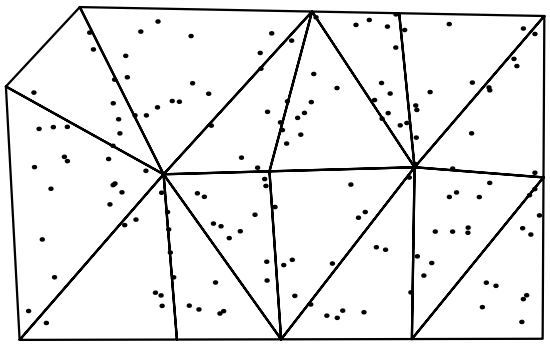
(kein Optimierungserfolg)



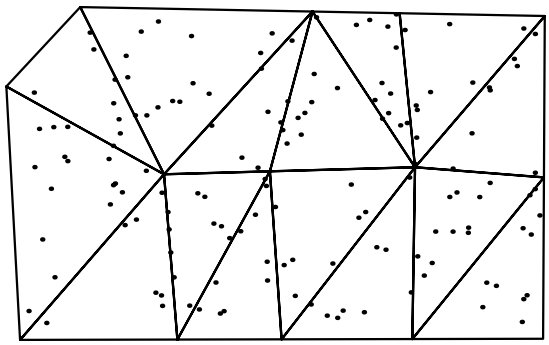
(ein neuer Punkt wurde eingefügt)



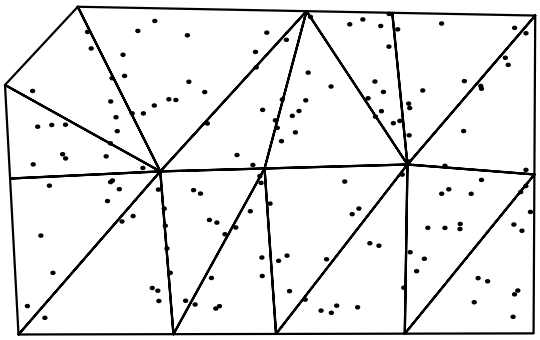
(nach der Optimierung)



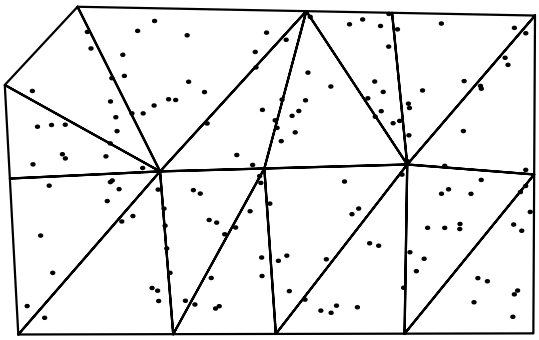
(ein neuer Punkt wurde eingefügt)



(nach der Optimierung)

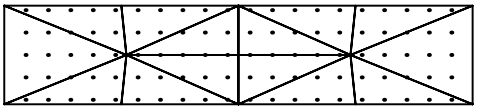


(ein neuer Punkt wurde eingefügt)

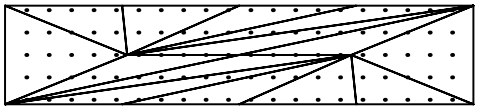


(kein Optimierungserfolg)

Beispiel 3.51 (Notwendigkeit der Optimierung)
Daß die Verwendung von Optimierungsalgorithmen Vorteile bringt in Bezug auf die Vermeidung langer spitzer Dreiecke, zeigt dieses Beispiel. Mit der Optimierung ergibt sich bzgl. des erweiterten Max-Min-Winkelkriteriums eine deutlich bessere Triangulierung. Weiter sieht man, daß auch bei regelmäßigen Daten gute Ergebnisse erzielt werden.



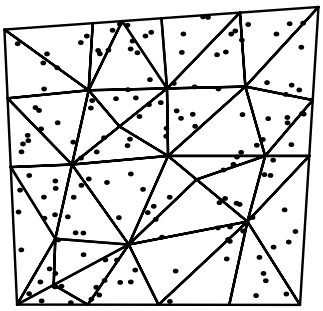
(mit Optimierungsalgorithmus)



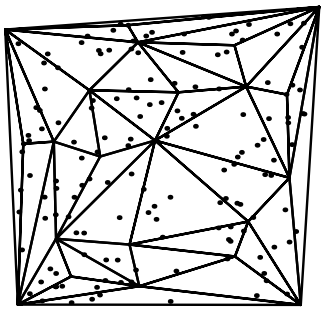
(ohne Optimierungsalgorithmus)

Beispiel 3.52 (Notwendigkeit von Algorithmus B)
Daß Algorithmus B für die Lösung der Problemstellung gar nicht notwendig ist, ist bereits mehrfach erwähnt worden. Ein Verwendung von Algorithmus B (ebenso wie die Einbindung von Optimierungsalgorithmen) liefert in Bezug auf die Vermeidung langer spitzer Dreiecke allerdings bessere Ergebnisse, wie dieses Beispiel

zeigt. Beim rechten Bild sind einige Dreiecke am Rand so spitz, daß man diese kaum noch erkennen kann. (150 Daten wurden zufällig erzeugt; ein Minimum von $d = 3$ wurde verlangt.)

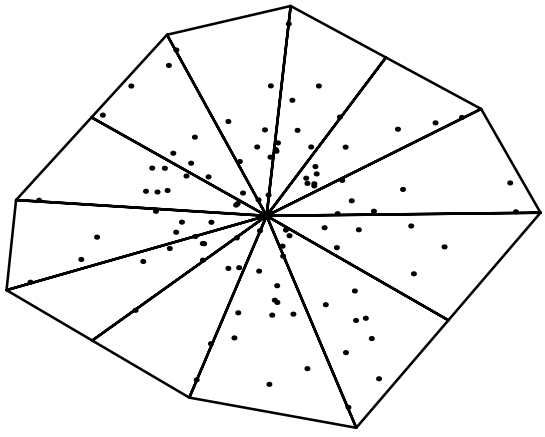


(Verwendung von Algorithmus B im Anschluß an Algorithmus A, Algorithmus C wurde dann nicht mehr benötigt)

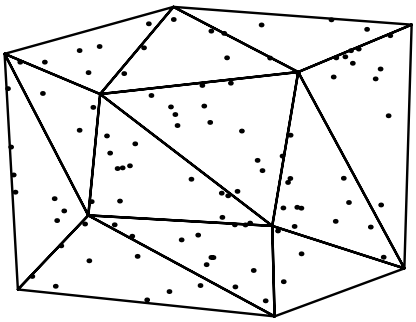


(Verwendung von Algorithmus C im Anschluß an Algorithmus A)

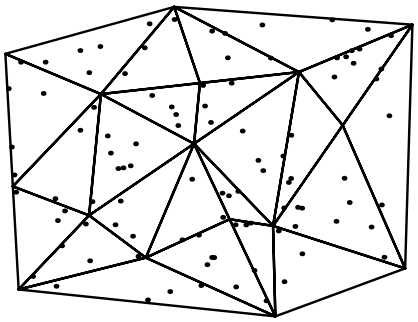
Beispiel 3.53 (Notwendigkeit der inneren Punkte in Algorithmus A) In Algorithmus A werden neben den Außenpunkten auch innere durch einen Clustering-Algorithmus berechnet. Dieses Verfahren wurde entwickelt, da wir vorher teilweise sehr viele Punkte abschneiden mußten (vgl. Schritt 12 in Algorithmus 3.25), bis wir überhaupt eine Triangulierung erhielten, die in jedem Dreieck genügend Daten hatte. Oft genügt ein einziger innerer Punkt, um ohne Veränderung der Außenpunkte sofort zu einem Ergebnis zu gelangen.



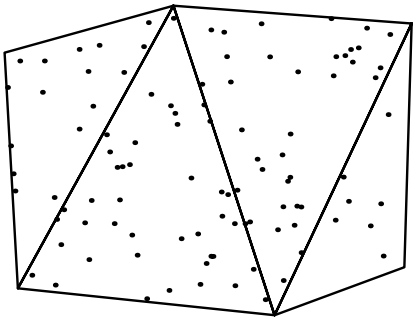
Zusätzlich stellte sich heraus, daß die Ergebnisse mit dem zusätzlichen Verfahren oft besser ausfielen.



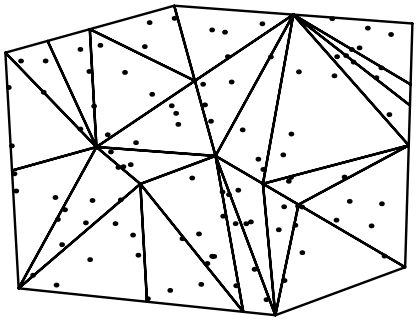
(nach Algorithmus A mit inneren Punkten)



(nach Algorithmus B mit inneren Punkten)



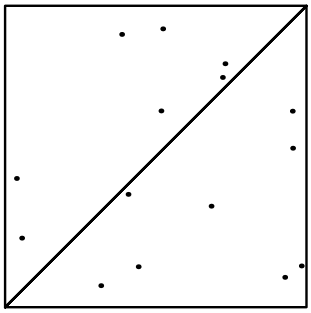
(nach Algorithmus A ohne innere Punkte)



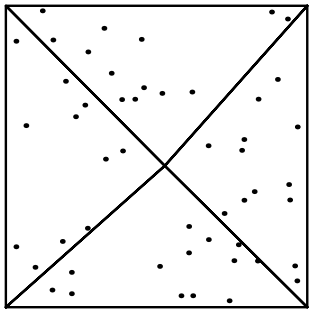
(nach Algorithmus B ohne innere Punkte)

Beispiel 3.54 (Triangulierungen zu den Beispielen aus Abschnitt 2.3) Es sollen der Vollständigkeit halber die Triangulierungen, die sich zu den Beispielen zur Approximation in Abschnitt 2.3 ergeben haben, dargestellt werden.

1. Bei den Beispielen zur Reproduktion wurden bei $k = 2$ minimal $d = 6$ Daten pro Dreieck gefordert; im Fall $k = 3$ waren es $d = 10$. Die Außenpunkte der Triangulierung wurden nicht berechnet, sondern als Einheitsquadrat fest vorgegeben.

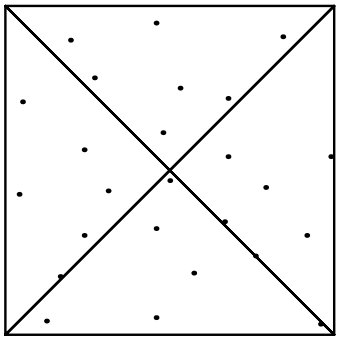


(Reproduktion $k = 2$)

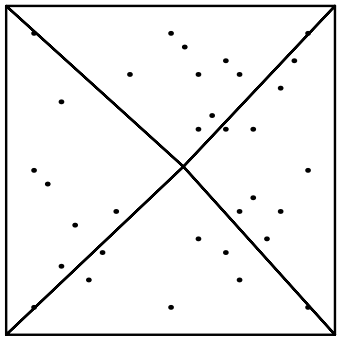


(Reproduktion $k = 3$)

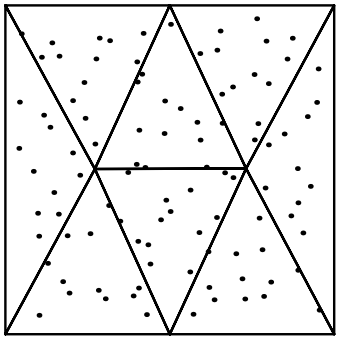
2. Auch für die Beispiele zur Franke-Funktion wurden im Falle $k = 2$ bzw. $k = 3$ in jedem Dreieck minimal 6 bzw. 10 Daten verlangt. Die Außenpunkte der Triangulierung wurden diesmal durch { (-0.1,-0.1), (1.1,-0.1), (1.1,1.1.), (-0.1,1.1) } vorgegeben.



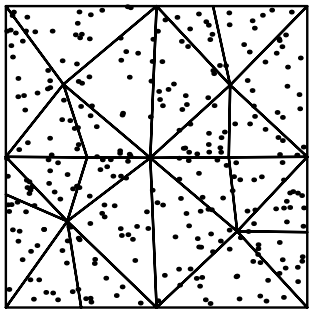
(25 Daten, $k = 2$)



(33 Daten, $k = 3$)



(100 Daten sowohl für $k = 2$ als auch für $k = 3$)



(300 Daten, $k = 2$)

Literaturverzeichnis

[AuS93] G. Aumann und K. Spitzmüller: *Computerorientierte Geometrie,* BI Wissenschaftsverlag, Mannheim, (1993).

[BSh70] I. S. Beresin und N. P. Shidkow: *Numerische Methoden 1,* VEB Deutscher Verlag der Wissenschaften, Berlin, (1970).

[ClR84] A. K. Cline, R. J. Renka: *A storage-efficient method for construction of a Thiessen triangulation,* Rocky Mountain J. of Math. 14, S. 119-139, (1984).

[ClR84a] A. K. Cline, R. J. Renka: *A triangle-based C^1 interpolation method,* Rocky Mountain J. of Math. 14, S. 223-237, (1984).

[CuS66] H. B. Curry und I. J. Schoenberg: *On Pòlya frequency functions IV: the fundamental spline functions and their limits,* in: J. Analyse Math. 17, S. 71-107, (1966).

[DaM83] W. Dahmen und C. A. Micchelli: *Recent progress in multivariate splines,* in: C. K. Chui, L. L. Schumaker und J. D. Ward (eds.), Approximation Theory IV, S. 27-121, Academic Press, New York, (1983).

[deB76] C. de Boor: *Splines as linear combinations of B-splines,* in: G. G. Lorentz, C. K. Chui und L. L. Schumaker (eds.), Approximation Theory II, S. 1-47, Academic Press, New York, (1976).

[deB78] C. de Boor: *A practical guide to Splines,* Springer, New York, (1978).

[DMS92] W. Dahmen, C. A. Micchelli und H. P. Seidel: *Blossoming begets B-spline bases built better by B-patches,* Mathematics of Computation 59, S. 97-115, (1992).

[FDZ95] T. A. Foley, S. Dayanand und D. Zeckzer: *Localized radial basis methods using rational triangle patches,* in: H. Hagen, G. Farin, H. Noltemeier (eds.), Computing Suppl. 10: Geometric Modeling, S. 163-176, Springer, Berlin, (1995).

[FrN91] R. Franke und G. M. Nielson: *Scattered data interpolation and application: a tutorial and survey,* in: H. Hagen und D. Roller (eds.), Geometric Modelling – Methods and Applications, S. 131-159, Springer, Berlin, (1991).

[HaD72] R. L. Harder und R. N. Desmarais: *Interpolating using surface splines,* J. Aircraft 9, S. 189-197, (1972).

[Har75] J. A. Hartigan: *Clustering Algorithms,* Wiley, New York, (1975).

[HoL93] J. Hoschek und D. Lasser: *Fundamentals of Computer Aided Geometric Design,* A. K. Peters, Wellesley (Massachusetts), (1993).

[Hus98] M. Hußmann: *Auswertungsverfahren für Simplex-Splines und Dahmen-Micchelli-Seidel-Splines,* Diplomarbeit im Fachbereich 11 an der Gerhard-Mercator-Universität Duisburg, (1998).

[Law77] C. L. Lawson: *Software for C^1 surface interpolation,* in: J. R. Rice (ed.), Mathematical Software III , S. 161-194, Academic Press, New York, (1977).

[Lor91] R. A. H. Lorentz: *Multivariate Birkhoff Interpolation,* Habilitationsschrift im Fachbereich Mathematik der Universität – Gesamthochschule – Duisburg (1991).

[Mic80] C. A. Micchelli: *A constructive approach to Kergin-interpolation in $\mathbb{R}^k$: multivariate B-splines and Lagrange-interpolation,* Rocky Mountain J. of Math. 10, S. 485-497, (1980).

[PSh85] F. P. Preparata und M. I. Shamos: *Computational Geometry,* Springer, New York, (1985).

[Qua94] E. Quaisser: *Diskrete Geometrie,* Spektrum Akademischer Verlag, Heidelberg, (1994).

[ScW92] R. Schaback und H. Werner: *Numerische Mathematik,* Springer, Berlin, (1992).

[Shu87] L. L. Schumaker: *Triangulation methods,* in: C. K. Chui, L. L. Schumaker, F. Utreras (eds.), Topics in Multivariate Approximation, S. 219-232, Academic Press, New York, (1987).

[Shr91] T. Schreiber: *A Voronoi diagram based adaptive k-means-type clustering algorithm for multidimensional weighted data,* in: H. Bieri, H. Noltemeier (eds.), Computational Geometry - Methods, Algorithms and Applications, S. 265-275, Springer, Berlin, (1991).

[Shw93] H. R. Schwarz: *Numerische Mathematik,* Teubner, Stuttgart, (1993).

[Sei89] H. P. Seidel: *A new multiaffine approach to B-splines,* CAGD 6, S. 23-32, (1989).

[WaM85] R. Wait, A. R. Mitchell: *Finite Element Analysis and Applications,* John Wiley, Chichester, (1985).

[Wen93] H. J. Wenz: *Ein neuer Zugang zur polynomialen Spline-Quasiinterpolation,* Dissertation Fachbereich Mathematik FernUniversität Hagen (1993).